МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

Навчально-науковий інститут електронних та інформаційних технологій
Кафедра інформаційних та комп'ютерних систем

Допущено до захисту
Завідувач кафедри
к.е.н., доцент Базилевич В.М.

«___» _____2020 р.

ВИПУСКНА КВАЛІФІКЦІЙНА РОБОТА
за освітньо-професійною програмою бакалавра

СИСТЕМА АВТОМАТИЗОВАНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ДЛЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРІ НА БАЗІ DOCKER

Спеціальність 123 – Комп'ютерна інженерія
Галузь знань 12 – Інформаційні технології

Виконавець:
студент гр. КІ–161
Сослуєв Олександр В'ячеславович
(підпис)

Керівник:
доцент _____ *к.т.н., доцент* _____
(посада) *(науковий ступінь, вчене звання)*
Риндич Євген Володимирович
(підпис)

Чернігів 2020

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

Навчально-науковий інститут електронних та інформаційних технологій
Кафедра інформаційних та комп'ютерних систем

ЗАТВЕРДЖУЮ:

Зав. кафедри

к.е.н., доцент Базилевич В.М.

"____" _____2020г.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ
ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ЗА ОСВІТНЬО-ПРОФЕСІЙНОЮ ПРОГРАМОЮ БАКАЛАВРА

Сослуєва Олександра В'ячеславовича

Тема роботи: СИСТЕМА АВТОМАТИЗОВАНОГО БАЛАНСУВАННЯ НАВАНТАЖЕННЯ ДЛЯ МІКРОСЕРВІСНОЇ АРХІТЕКТУРІ НА БАЗІ DOCKER

Тему затверджено наказом ректора
від "__"_____ 2020 р. №_____

1. **Вхідні дані до роботи:**
   Балансування навантаження, перенаправлення 50% користувачів до новою версії сайту та 50% - до старої, контейнеризація, використання Kubernetes.
2. **Зміст розрахунково-пояснювальної записки :**
   Кваліфікаційна робота складається зі вступу, основної частини та висновків. Основна частина складається із трьох розділів: «Аналіз задачі та інструменти її створення», «Розробка конфігурації» та «Реалізація системи».
3. **Демонстраційні матеріали:**
   14 слайдів презентації роботи

4. Календарний план

№	Назва етапів роботи	Термін виконання	Примітки
1	Визначення з керівником	27.01.2020	
2	Визначитися з темою роботи	20.01.2020	
3	Індивідуальне завдання	3.02.2020	
4	Знайомство з проблемою, інформаційними джерелами, аналіз існуючих рішень.	20.03.2020	
5	Прийняття рішень стосовно основних моментів розробки	10.04.2020	
6	Звіт з першої частини роботи (ІЗ, Вступ та аналіз задачі)	24.04.2020	Звіт
7	Збори з питань практики	08.05.2020	
8	Розробка системи	15.05.2020	
9	Звіт з практики, розділ розробка	17.05.2020	Звіт
10	Реалізація системи	24.05.2020	
11	Попередній захист	29.05.2020	
12	Підпис роботи керівником		
13	Перевірка на «плагіат»		
14	Рецензування роботи		
15	Підпис завідувача кафедрою		
16	Захист кваліфікаційної роботи	16.06.2020	

Завдання підготував:
керівник Риндич Євген Володимирович
(підпис)
«___» _____2020 р.

Завдання одержав:
студент Сослуєв Олександр В’ячеславович
(підпис)
«___» _____2020 р.

РЕФЕРАТ

Дипломна робота, 106 с., 22 рис., 1 табл., 21 джерел.

Дипломна робота має полягає в написанні конфігурації для веб-сайту. Конфігурація повинна працювати на всіх платформах як Linux, Windows, Mac OS. Docker - це дуже зручний інструмент для управління ізольованими Linux-контейнерами. За допомогою цього інструменту можна операційній системи запускати процеси в ізольованому оточенні на базі спеціально створених образів.

Метою роботи є :

-Проектування системи з використанням системи віртуалізації за допомогою інструменту Docker, що включає розробку и налаштування конфігурації.

-Розробка конфігурації для автоматизованого балансування навантаження.

-Можливість налаштування проценту переходу користувачів на сайт.

Наприклад в системі банкінгу, є дві версії, одна з яких бета-версія сайту, друга це стара стабільна версія. Це необхідно для того щоб тестувальники на великих проєктах, витрачали менше ресурсів та часу на виконання тестів. Середній час відвідування сайту банку займає близько 30 хвилин.

За останні роки розробники додатків та сайтів, хотіли щоб їх продукт стабільно працював на декількох платформах. Швидко запускався та розгортався за короткий час. Один раз витрачавши час на конфігурацію, а потім її використовуєш в декількох проектах де вона необхідна, значно спрощує розробку.

UBUNTU DOCKER KUBERNETES JMETER JS HTML CSS

THE ABSTRACT

Diplom work, 106 pp. 22 fig., 1 tab., 21 sources.

The subject of the thesis is to write the configuration for the website. The configuration must work on all platforms like Linux, Windows, Mac OS. Docker is a very handy tool for managing isolated Linux containers. With this tool, you can run the operating system processes in an isolated environment based on specially created images.

The purpose of the work is:

-Design the system using a virtualization system using the Docker tool, which includes the development and configuration of the configuration.

-Configuration development for automated load balancing.

-Ability to adjust the percentage of users coming to the site.

For example, in the banking system, there are two versions, one of which is a beta version of the site, the other is an old stable version. This is necessary so that testers on large projects spend less resources and time on tests. The average time to visit the bank's website is about 30 minutes.

In recent years, application and website developers have wanted their product to run stably on multiple platforms. It started up and deployed quickly in a short time. Once you spend time on the configuration, and then use it in several projects where it is needed, greatly simplifies development.

UBUNTU DOCKER KUBERNETES JMETER JS HTML CSS

ЗМІСТ

ВСТУП 9

1 АНАЛІЗ ЗАДАЧІ ТА ІНСТРУМЕНТИ ЇЇ СТВОРЕННЯ 10

1.1 Контейнеризація 10

1.2 Віртуальна машина 10

1.3 Контейнер 11

1.4 Аналіз предметної області 14

1.4.1 Системи редиркет 14

1.4.2 Розгортання 15

1.4.3 Балансування навантаження 16

1.5 Цілі та задачі системи 17

1.6 Призначення системи 18

1.7 Вимоги до системи 18

2 РОЗРОБКА КОФІГУРАЦІЇ 19

2.1 Вибір технологій та засобів для написання конфігурації 19

2.1.1 Вибір інструменту для контейнеризації 19

2.1.2 Існуючи інструменті для оркестрації 26

2.1.3 Інструменти для навантаження системи 32

2.1.4 Вибір скриптової мови програмування для автоматизації запуску 36

2.1.5 Вибір формату для конфігурації 37

2.1.6 Вибір веб-серверу 39

2.2 Архітектура системи 44
3 РЕАЛІЗАЦІЯ СИСТЕМИ 46
3.1 Встановлення потрібних інструментів 46
3.1.1 Встановлення Docker 46
3.1.2 Встановлення Minikube 50
3.1.3 Встановлення Jmeter 54
3.2 Розробка веб-серверу для сайту 55
3.3 Розробка веб-серверу для балансування навантаження 57
3.4 Використання Jmater 60
3.5 Використання платформи Kubernetes 64
ВИСНОВКИ 65
ПЕРЕЛІК ПОСИЛАНЬ 66
ДОДАТОКИ 68
ДОДАТОК А 69

ПЕРЕЛІК ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ - Програмне забезпечення

Docker - інструмент для управління ізольованими Linux-контейнерами.

ОС – операційна система

Гіпервізор - комп'ютерна програма або обладнання, що забезпечує одночасне, паралельне виконання декількох операційних систем на одному і тому ж комп'ютері.

ВМ (віртуальна машина) - модель обчислювальної машини, створеної шляхом віртуалізації обчислювальних ресурсів: процесора, оперативної пам'яті, пристроїв зберігання та вводу і виводу інформації

Daemon (демон) – клієнт Docker, що доступен по REST API

Nginx — вільний веб-сервер і проксі-сервер.

HTTP — протокол передачі даних, що використовується в комп'ютерних мережах. Назва скорочена від Hyper Text Transfer Protocol, протокол передачі гіпер-текстових документів.

ВСТУП

Зараз розробка додатків йде дуже активно, тому виникають проблеми як передати продукт клієнту, так щоб він запрацював на сервері у замовника.

Головна мета роботи – розробити автоматизовану систему балансування навантаження. При налаштуванні за замовчуванням система перенаправляє 50% користувачів до новою версії сайту, 50% - до старої, завдяки чому клієнт буде отримувати нову версію додатку на всіх пристроях і воно буду працювати коректно.

В залежності від навантаження системи балансувати навантаження, розгортати новий сервер або згортати вже не потрібний.

Додатки мають багато додаткових бібліотек та феймворків, які потрібні для роботи додатку. Кожного разу переносити це все до клієнта не дуже зручно. Тому постає питання як це все зробити автоматизовано. Всім же розробникам додатку буде краще коли їх додаток буду стабільно працювати на всіх платформах та у всіх клієнтів.

Є кілька варіантів вирішення цих проблем. Наприклад хмарні сервіси, віртуальні машини та контейнеризація.

В цій роботі буде використана контейнеризація.

Контейнеризація - це легка віртуалізація і ізоляція ресурсів на рівні операційної системи, яка дозволяє запускати додаток і необхідний йому мінімум системних бібліотек в повністю стандартизовано контейнері, що з'єднують з хостом або чим-небудь зовнішнім по відношенню до нього за допомогою певних інтерфейсів. Контейнер не залежить від ресурсів або архітектури хоста, на якому він працює.

1 АНАЛІЗ ЗАДАЧІ ТА ІНСТРУМЕНТИ ЇЇ СТВОРЕННЯ

1.1 Контейнеризація

При розробці веб-сайту, його необхідно передати користувачеві. Готується багато різних файлів, скриптів та інструкцій по установці. А потім виникають проблеми у клієнта на зразок: «у мене нічого не працює», «ваш скрипт впав на середині - що тепер робити».

Друга проблема – тиражованість. Якщо додаток тиражований і замість одного клієнта є сотні або тисячі покупців. Це стає ще складніше, якщо згадати про необхідність установки нових версій продукту.

Третя проблема – надмірне використання. При розробці декількох веб-сайтів. Всі вони використовують один і той же технологічний стек (наприклад –nginx, php, nodejs, mysql, phpmyadmin). Але при цьому, щоб запустити новий сайт потрібно заново готувати на новий сервер майже однакову конфігурацію. Це займає дуже багато часу. Не має можливості просто взяти та сказати - «хочу сервер, як на сайті інтернет магазину, але тільки з іншим веб архівом»

Є кілька варіантів вирішення цих проблем. Наприклад: хмарні серіси, віртуальні машини та контейнеризація.

1.2 Віртуальна машина

Для ізоляції процесів, запущених на одному хості, запуску додатків, призначених для різних платформ, можна використовувати віртуальні машини. Віртуальні машини ділять між собою фізичні ресурси хоста:

- процесор
- пам'ять
- дисковий простір

- мережеві інтерфейси

Приклад роботи віртуальної машини зображений на рисунку 1.1

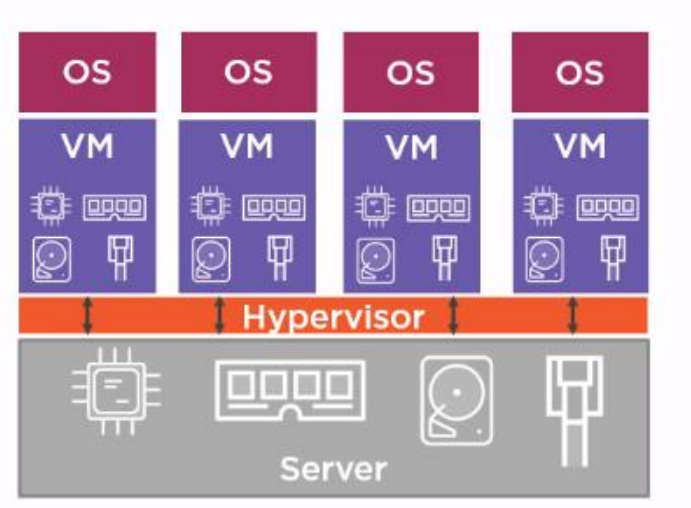


Рисунок 1.1- Віртуальна машина

На кожній віртуальній машині встановлюємо потрібну ОС і запускаємо додатки. Недоліком такого підходу є те, що значна частина ресурсів хоста витрачається не на корисне навантаження (робота додатків), а на роботу декількох ОС [1].

1.3 Контейнер

Альтернативним підходом до ізоляції додатків є контейнери. Саме поняття контейнерів не ново і давно відомо в Linux. Ідея полягає в тому, щоб в рамках однієї ОС виділити ізольовану область і запускати в ній додаток. В цьому випадку говоримо про віртуалізації на рівні ОС. На відміну від ВМ контейнери ізольовано використовують свій шматочок ОС:

- файлова система
- дерево процесів
- мережеві інтерфейси

Приклад роботи віртуальної машини зображений на рисунку 1.2

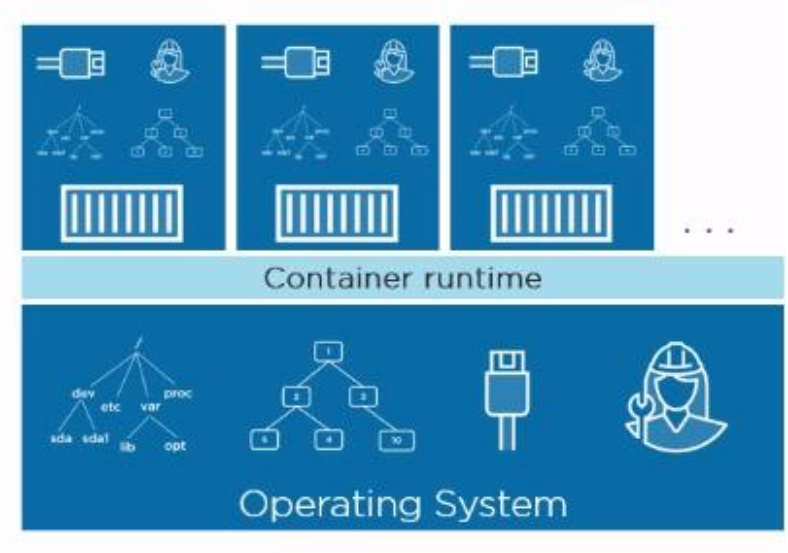


Рисунок 1.2- Контейнер

Тобто додаток, запущене в контейнері думає, що воно одне у всій ОС. Ізоляція досягається за рахунок використання таких Linux-механізмів, як namespaces і control groups. Якщо говорити просто, то namespaces забезпечують ізоляцію в рамках ОС, а control groups встановлюють ліміти на споживання контейнером ресурсів хоста, щоб збалансувати розподіл ресурсів між запущеними контейнерами [1].

Тобто контейнери самі по собі не є чимось новим, просто проект Docker. По-перше, приховав складні механізми :

- Простір імен (Namespaces)

Docker використовує технологію, названу namespaces щоб забезпечити ізольовану робочу область, яку називають контейнером. Коли запускається контейнер, Docker створює набір namespaces для цього контейнера.

Ці простори імен забезпечують шар ізоляції. Кожен аспект контейнера працює в окремому просторі імен, і його доступ обмежений цим простором імен.

- Docker Engine використовує простори імен, такі як у Linux:

- The pid namespace: ізоляція процесу (PID: ідентифікатор процесу).
- The net namespace: Управління мережевими інтерфейсами (NET: Мережа).
- The ipc namespace**:** управління доступом до ресурсів IPC (IPC: Інтерпроцесовий зв'язок).
- The mnt namespace**:** Управління точками монтування файлової системи (MNT: Монтування).
- The uts namespace**:** ізоляція ідентифікаторів ядра і версії. (UTS: Система Unix Timesharing).

- Контрольні групи (control groups)

Docker Engine в Linux також покладається на іншу технологію, яку називають control groups (групами). Група обмежує додаток певним набором ресурсів. Контрольні групи дозволяють Docker Engine ділитися доступними апаратними ресурсами на контейнери та, можливо, виконувати обмеження та обмеження. Наприклад є можливість обмежити доступну пам'ять певним контейнером.

- Файлові системи (Union file systems)

Файлові системи Union або UnionFS - це файлові системи, які працюють, створюючи шари, роблячи їх дуже легкими та швидкими. Docker Engine використовує UnionFS для надання будівельних блоків для контейнерів. Docker Engine може використовувати декілька варіантів UnionFS, включаючи AUFS, btrfs, vfs та DeviceMapper.

- Формат контейнера

Docker Engine об'єднує простори імен, групи керування та UnionFS в обгортку, що називається формат контейнера. Типовим форматом контейнера є libcontainer. Надалі Docker може підтримувати інші формати контейнерів, інтегруючись з такими технологіями, як BSD Jails або Solaris Zones.

А по-друге, він оточений екосистемою, що забезпечує зручне використання контейнерів на всіх стадіях розробки ПЗ.

1.4 Аналіз предметної області

Буде створена нова версія сайту, всього буде їх дві. При налаштуванні за замовчуванням система перенаправляє 50% користувачів до новою версії сайту, 50% - до старої, тобто першому користувачу відкриється стара версія а другому нова, третьому знов стара. Це потрібно щоб побачити є якісь проблеми на новій версії, тобто взаємодія з користувачем щоб доробити нову версію сайту за необхідністю. Також під час сильного навантаження сайтів автоматично масштабувати систему.

1.4.1 Системи редиркет

Редирект - це автоматична переадресація користувача на URL, який відрізняється від запитаного їм спочатку. Якщо редирект налаштований коректно і не суперечить правилам пошукових систем, то для користувача, як правило, сам процес перенаправлення залишається непоміченим.

Найпростіший приклад редиректу: користувач вводить одну адресу інтернет-магазину в адресному рядку, а в підсумку потрапляє на інший, більш актуальний ресурс цієї компанії. Сторінка, з якої перенаправили користувача, в цьому випадку називається донор (від лат. Dono - «дарую»). Сторінка, на яку його направили - акцептор (лат. Accipio - «я приймаю, отримую»).

Типи редиректов:

- 301 Moved Permanently

301 редирект - це постійний редирект, який демонструє, що документ перенесений на новий URL-адресу. 301 редирект кращий варіант для оптимізації

сайту під пошукові системи. Він дозволяє замінити адресу документа у видачі зі старого на новий без втрати позицій і трафіку.

- 302 Found, 302 Moved Temporarily

302 редирект демонструє, що запитаний ресурс тимчасово переміщений. Даний вид редиректу може бути використаний для сторінок з короткостроковими акціями і розпродажами конкретного товару. Не рекомендованно використовувати його для адрес, змінених назавжди. Роботи пошукових систем індексують тільки ту сторінку, на яку встановлено редирект, але у видачі залишиться старий URL. Однак, якщо робот пошукової системи вирішить, що помилково налаштований 302 редирект замість 301, він замінить адресу у видачі, як і в випадку з 301 перенаправленням. І повернути його назад буде проблематично.

- 307 Moved Temporarily

Тимчасовий редирект, який вказує, що документ тимчасово доступний за іншою URL. Відмінність від 302: збереження методу передачі запиту (GET, POST), який вказує пошуковикам, що кешувати документ не варто (якщо немає додаткових вказівок).

Можна встановити 307 Moved Temporarily, коли контент переміщається тільки тимчасово (наприклад, при перепроєктуванні). Це дозволяє пошуковим системам зрозуміти, що сервер сумісний з HTTP 1.1.

Крім того, клієнти не повинні автоматично переадресовувати запити POST / PUT / DELETE. Кешування має виконуватися тільки в тому випадку, якщо у відповіді сервера є заголовки Cache-Control або Expires [2].

1.4.2 Розгортання

Розгортання програмного забезпечення є всі заходи, які роблять систему програмного забезпечення для використання.

Загальний процес розгортання складається з декількох взаємопов'язаних заходів з можливими переходами між ними. Ці дії можуть відбуватися на виробник стороні або на споживчій стороні або обидва. Оскільки кожна система програмного забезпечення є унікальною, точні процеси або процедури, навряд чи можуть бути визначені в межах кожної активності. Таким чином, «розгортання» слід тлумачити як загальний процес, який повинен бути налаштований у відповідності з конкретними вимогами або характеристиками.

Приклади інструментів розгортання програмного забезпечення:

- Ansible

Ansible є відкритим вихідним кодом програмного забезпечення ініціалізації, управління конфігурацією і додатки розгортання інструменту. Вона працює на багатьох Unix-подібних систем, і може налаштувати і Unix-подібних систем, а також Microsoft Windows. Вона включає в свою власну мову декларативного опису конфігурації системи. Ansible була написана Майклом де Гаан і придбала Red Hat в 2015 році. Ansible є безагентним, тимчасово вилученим підключенням через SSH або віддалене управління Windows (дозволяє віддаленому PowerShell виконання), щоб зробити свої завдання.

- Salt

Salt (іноді згадується як SaltStack Platform) відкрите програмне забезпечення для керування конфігурацією та віддаленого виконання написане на Python. Підтримує підхід "Infrastructure as Code" до розгортання і керування хмарами [3].

1.4.3 Балансування навантаження

У комп'ютерах балансування навантаження розподіляє навантаження між декількома обчислювальними ресурсами, такими як комп'ютери, комп'ютерні кластери, мережі, центральні процесори або диски. Мета балансування навантаження - оптимізація використання ресурсів, максимізація пропускної

здатності, зменшення часу відгуку і запобігання перевантаження будь-якого одного ресурсу. Використання декількох компонентів балансування навантаження замість одного може підвищити надійність і доступність за рахунок резервування. Балансування навантаження передбачає зазвичай наявність спеціального програмного забезпечення або апаратних засобів, таких як багаторівневий комутатор або система доменних імен.

Приклади інструментів балансування навантаження:

- Round Robin

Round Robin, або алгоритм кругового обслуговування, являє собою перебір по круговому циклу: перший запит передається одного сервера, потім наступний запит передається іншому і так до досягнення останнього сервера, а потім все починається спочатку.

- Sticky Sessions

Sticky Sessions - алгоритм розподілу вхідних запитів, при якому з'єднання передаються на один і той же сервер групи. Він використовується, наприклад, в веб-сервері Nginx. Сесії користувача можуть бути закріплені за конкретним сервером за допомогою методу IP hash. За допомогою цього методу запити розподіляються по серверам на основі IP-Адреса клієнта. Метод гарантує, що запити одного і того ж клієнта буде передаватися на один і той же сервер. Якщо закріплений за конкретною адресую сервер недоступний, запит буде переправлений на інший сервер [4].

1.5 Цілі та задачі системи

Основною ціллю системи є організація системи автоматизованого балансування навантаження для мікро-сервісної архітектури на базі Docker та Kubernetes.

Автоматично перенаправляти користувачів за заданим алгоритмом до старої та нової версії сайту.

Автоматизація тестування системи розгорнути в контейнерах кілька мікросервіси (сайтів). При великому навантажені системи відкривати нову копію сайту, щоб зменшити навантаженя.

1.6 Призначення системи

Даний проект несе практичний характер який використовують в багатьох IT компаніях. Дозволяє автоматизувати доставку продукту на продакшен сервер и його масштабування при навантаженні, а також виявити дефекти які не були знайдені на етапі тестування за рахунок залучення великої кількості користувачів.

1.7 Вимоги до системи

Основною вимогою до системи це автоматичне масштабування при навантаженні сервера. Організувати за замовчуванням систему перенаправлення 50% користувачів до новою версії сайту, 50% - до старої

Автоматизувати запуск конфігурації на локальній машині за допомогою bash скрипта. Система має працювати з веб-серверами які на яких розміщенні сайти мати можливість налаштування переходу користувачів.

2 РОЗРОБКА КОФІГУРАЦІЇ

2.1 Вибір технологій та засобів для написання конфігурації

В даному пункті проводиться вибір системи управління контентом (масштабування та балансування), мови програмування скрипта для автоматизації запуску, інструменти для контейнеризації.

2.1.1 Вибір інструменту для контейнеризації

В дипломній роботі буде використовуватися Docker, так як порівняно з іншими схожими інструментами, він найкраще підходить для поставлених задач.

Існує декілька інструментів для контейнеризації вони схожі між собою але мають відмінності при встановлені та використанні та роботою з ними.

Серед самих розповсюджених це LXD lightervisor**,** rkt(відомий як Rocket), Docker.

Тепер розглянемо кожний з інструментів них більш детально.

- LXD lightervisor.

LXD - це демон, який надає REST API для управління LXC контейнерами. Головна мета: зробити щось схоже на віртуальні машини під апаратною віртуалізацією, але з використанням Linux контейнерів. LXD запущений на кожному хості де будуть обслуговуватися контейнери і з клієнтської машини є можливість переміщати контейнери з хоста на хост [5].

Є 2 способи:

- інструмент командного рядка на ім'я lxc.
- плагін OpenStack Nova по імені nova-compute-lxd.

На рисунку 2.1 зображено логотип LXD

Рисунок 2.1 – Логотип LXD

Основні компоненти LXD

- Контейнери

  Контейнери LXD містять:

  - Файлова система (rootfs).
  - Список опцій конфігурації, включаючи ліміти ресурсів. Опції безпеки, оточення і т.д.
  - Перерахування пристроїв типу дисків, мережевих інтерфейсів.
  - Набір профілів для контейнера звідки унаслідуються конфігурації.
  - Деякі властивості даного контейнера: архітектура, ім'я
  - Стан при використанні CRIU.

- Знімки

Знімки контейнера незмінні в тому сенсі, що їх не можна модифікувати, крім перейменування, знищення або відновлення. Варто зазначити, що оскільки зберігається весь стан контейнера, то фактично ви маєте концепцією 'stateful' знімків. Це дає можливість відмінити зміни в контейнері, включаючи стан його CPU і пам'яті.

- Образи

LXD заснований на використанні образів. Всі контейнери створені з образів. Образи зазвичай представляють собою чистий Linux дистрибутив, подібно до тих що використовується у віртуальній машині або в хмарних середовищах. Можливо публікувати (publish) контейнер, роблячи з нього образ,

який вже використовується локальними або віддаленими LXD хостами.

- CoreOS — rkt (Rocket)

Проект CoreOS, що розвиває, засноване на ідеях контейнерної ізоляції, серверне оточення, представив випуск інструментарію управління контейнерами rkt (раніше відомий як Rocket), який позиціонується як більш безпечна, переносима і адаптована для серверного застосування альтернатива інструментарію Docker. Код rkt написаний на мові Go[6].

Логотип rkt зображений на рисунку 2.2.

Рисунок 2.2 – Логотип rkt

Основні особливості rkt

- Розширені механізми забезпечення безпеки: ізоляція з використанням гіпервізора KVM, підтримка SELinux, SVirt і seccomp, інтеграція TPM (Trusted Platform Module), верифікація образів по цифровому підпису.
- Крім традиційних контейнерів на основі просторів імен (namespaces) і груп управління (cgroups) ізольовані оточення можуть створюватися і за допомогою інших технологій, включаючи звичайний chroot (контейнери fly) і віртуальне оточення Clear Linux (компактні і швидкі як звичайні контейнери, але надають більш високий рівень ізоляції за рахунок застосування гіпервізора KVM);
- Підтримка образів Docker і контейнерів, побудованих відповідно до специфікації App Container. Таким чином, rkt можна застосовувати в середовищах, які спочатку були розгорнуті за допомогою Docker. Також їх доцільно

використовувати для побудови нової інфраструктури на базі формату App Container;

- Docker

Docker - це відкрита платформа для розробки, доставки і експлуатації додатків. Docker розроблений для більш швидкого викладання додатків. За допомогою docker можна відокремити додаток від інфраструктури і звертатися з інфраструктурою як керованим додатком.

Docker допомагає викладати код швидше, швидше тестувати, швидше викладати додатки і зменшити час між написанням і запуском коду. Docker робить це за допомогою легкої платформи контейнерної віртуалізації, використовуючи процеси і утиліти, які допомагають керувати і викладати програми. Надалі це може дозволити виконати інтеграцію та легко викласти мікросервіс у продакшен (CI and CD). Наведемо приклад: девелопери розробляють продукт локально, а потім вони можуть розшарити свій локальніий набір Docker образів зі своїми колегами. Коли вони завершують цикл розробки, вони роблять білд образів, а потім заливають їх на тестовий майданчик, де можна зробити перевірку лінтерами, або запустити тести. З тестового плацдарму можна залити образи на продакшен сервери.

Docker не є заміною для LXC. "LXC" ставиться до можливостей ядра Linux (зокрема, імена та контрольні групи), які дозволяє пісочниця процесів один від одного, і контролює розподілу
їх ресурсів. На низькому рівні функцій ядра, Docker пропонує інструмент високого рівня з кількома потужними функціональними можливостями [7]:

• Портативний розгортання кількох машин

• Орієнтовані додатки

• Автоматична збірка

• Версії

• Компонент повторного використання

• Обмін

• Інструмент екосистеми

Логотип Docker зображений на рисунку 2.3

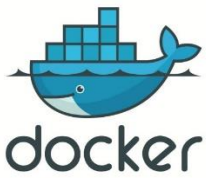


Рисунок 2.3 – Логотип Docker

Клієнт і сервер працюють або по протоколу REST або через сокет. Користувач може взаємодіяти з сервером тільки через клієнт-додаток. Клієнт взаємодіє з сервером отримуючи команди від користувача.

Архітектура Docker

- Docker-demon

На рисунку 2.4 показано як демон запускається на хості. Користувач використовує к лієнт, щоб взаємодіяти із сервером.

- Docker-client

Docker-client це головний компонент Docker, він може взаємодіяти з dockerdemon за домогою набору команд, які вводить користувач.

- Усередині Docker-a

Docker складається з трьох компонентів:

• Docker-образ - це read-only шаблон. Образ, наприклад, може містити Ubuntu с Nginx і розгорнутим додатком. Образи використовуються для створення контейнерів. Docker з легкістю дозволяє створювати нові образи, оновлювати існуючі, створювати на базі готових образів нові.

• Docker-реєстр зберігає образи. Реєстри розподіляють на публічні та приватні. У публічному Docker реєстрі знаходиться база даних образів дуже великого обсягу. Контейнери нагадують директорії.

У контейнерах міститься все необхідне для роботи програми. З контейнерами можна виконувати наступні дії: створювати, запускати, зупиняти, переносити та видаляти. Кожен контейнер може бути створена тільки з образу

• Контейнери. Вони нагадують нам звичайні директорії операційної системи. Контейнери**.** Будь-який контейнер має такі властивості як ізольованість і безпечність, заснована платформою додатка. В основі будь-якого образу лежить базовий образ. Мжна навести наступні приклади: debian це базовий образ Debian, mint – Mint, fedora – Dedora, ubuntu – Ubuntu. Використання образу для створення нового – загальна практика. Якщо ми маємо образflask, то його можна використати як базову для Python веб-додатків.

Docker образи можна створювати з базових образів. Будь-яка інструкція може створити образ. Говорячи про інструкції зазвичай розуміють наступне: робота з файлами та директоріями, створення environment оточення, підказки щодо запуску контейнерів та запуск і виконання Docker-команди**.**

На рисунку 2.4 зображено як демон запускається на хості.

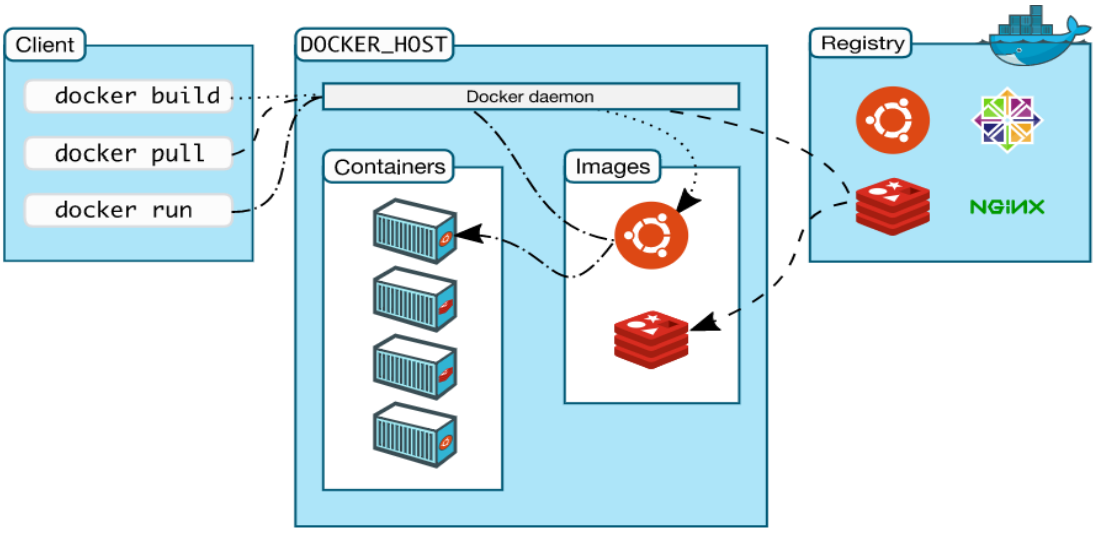


Рисунок 2.4 - Архітектура Docker

Робота зображення

Образ - це read-only паттерн, з якого може бути створений контейнер. Будь-який образ звичайно складається з різного набору рівнів. Docker використовує UFS

для того, щоб ці рівні можна було поєднати в один образ. UFS створює свою файлову систему за допомогою прозорого накладання файлів та директорій з різних гілок системи.

Саме завдяки використанню таких рівні Docker такий легкий у розмірах. Для того щоб змінити додаток вам треба всього додати новий рівень. У цьому перевага Docker перед VM – вам треба додати новий рівень і не треба перезбирати увесь додаток. Це дозволяє працювати з образами, і поширювати їх набагато простіше і швидше [8].

Робота Docker реєстра

Register - це сховище Docker образів. Після створення образу ви можете виконувати різні дії з ним, зокрема опублікувати його на Docker Hub, також ви можете створити свій локальний реєстр, який буде доступним лише усередині команди. За допомогою Docker ви легко можете знайти раніше опубліковані образи і завантажити їх на локальну машину, де можна їх використати для створення своїх нових контейнерів.

У Docker Hub існують публічні(public) та приватні(private) реєстри. Пошук і завантаження образів з публічних реєстрів доступне для всіх. Образи з приватних репозиторіїв не попадають до результатів пошуку, і їх можуть використовувати тільки обрані люди та створювати з них контейнери.

Робота контейнерів

До складу контейнеру входить операційна система, призначена для користування файлами та метаданими. Будь-який контейнер створюється з образу. Цей образ передає Docker, що він знаходиться усередені його контейнера, а також надає інформацію про конфігураційні дані. Docker образ використовує read-only паттерн. Docker під час запуску контейнеру створює рівень для читання та запису зверху образу у якому може бути запущено додаток [7].

- Порівняльна характиристика LXD та Docker

  LXD фокусується на системних контейнерах. Тобто LXD маніпулює

контейнерами з повноцінною Linux системою всередині так само, як ви б працювали з системою, на фізичному обладнанні або в віртуальною машиною. Основні системи управління конфігураціями і розгортання додатків можуть використовувати LXD так само, як і в звичайному використанні на фізичному обладнанні, віртуальній машині або хмарних сервісах.

Docker навпаки фокусується на мінімалістичний, легких контейнерах, які зазвичай не оновлюють або перелаштовують, а найімовірніше будуть замінені цілком. Це робить Docker і подібні платформи більше схожими до систем розгортання додатків, ніж до інструментів управління машинами. Ці дві моделі не є взаємовиключними, ви можете використовувати LXD для надання повноцінної Linux системи, а користувачі всередині контейнера можуть за допомогою Docker ставити потрібне їм ПЗ.

2.1.2 Існуючи інструменті для оркестрації

Використання контейнерів - один з найбільш активно розвиваються сегментів IT-ринку, орієнтований на корпоративне використання хмарних послуг.

Контейнери на ринку віртуалізації з'явилися відносно недавно. В основному завдяки технології Docker, що дозволяє запускати додатки в контейнері. Це схоже на звичайну віртуальну машину, але набагато краще, оскільки досягається практично повна незалежність від інфраструктури і знижене споживання ресурсами. Саме тому зараз можна помітити зсув від традиційних віртуальних машин в сторону контейнерів. Великі підприємства використовують їх при створенні хмарної IT-інфраструктури.

Коли говорять про контейнери, не можна не згадати про оркестрації. Оркестрації - це координація взаємодії декількох контейнерів. Звичайно, можна працювати і без оркестрації - ніхто не забороняє створити контейнер, в якому будуть запущені всі необхідні процеси. Однак в цьому випадку ви будете

позбавлені гнучкості, масштабованості, а також виникнуть питання безпеки, оскільки запущені в одному контейнері процеси не будуть ізольовані і зможуть впливати друг на друга.

Оркестрації дозволяє створювати інформаційні системи з безлічі контейнерів, кожен з яких відповідає тільки за одну певну задачу, а спілкування здійснюється через мережеві порти і загальні каталоги. При необхідності кожен такий контейнер можна замінити іншим, що дозволяє, наприклад, швидко перейти на іншу версію бази даних при необхідності.

Існують різні платформи для оркестрації контейнерів. Вони схожі між собою але мають відмінності при встановлені та використанні та роботою з ними Платформи дозволяють реалізувати зручні та ефективні засоби розгортання контейнерних систем, побудови єдиної централізованої консолі для застосування політик управління.

Найбільш відомі такі системи: Kubernetes, Docker Swarm і Apache Mesos. Це не єдині системи - є ще Nomad, Fleet, Aurora, Amazon EC2 Container Service, Microsoft Azure Container Service, проте вони менш популярні.

В даній дипломній роботі використана платформа Kubernetes. Порівняно з іншими платформами він краще всього взаємодіє с інструментом Docker, а також має необхідні можливості для реалізації проекту.

Приклади платформ для оркестрації

- Kubernetes

Kubernetes - OpenSource-система для управління контейнерними кластерами. З'явилася в результаті напрацювань Google при використанні механізму для ізоляції процесів у віртуальному середовищі (Borg). У 2014 р Google відкрила код Kubernetes і стала поширювати систему під ліцензією Apache 2.0.

Завдяки тому, що Google відкрила код і зробила свою систему оркестрації відкритою, вона стала найпопулярнішою. Сьогодні багато хто розглядає її як краще і універсальне рішення для роботи з додатками в хмарному середовищі. Платформу

оркестрації підтримують такі іменити вендори як Red Hat, IBM, CoreOS Yahoo, Microsoft, Apple, Amazon, Bosch, IBM, Docker та Samsung.

Логотип Kubernetes зображений на рисунку 2.5

Рисунок 2.5- Логотип Kubernetes

Kubernetes вибудовує ефективну систему розподілу контейнерів по вузлах кластера в залежності від поточного навантаження і наявних потреб в роботі сервісів. Ця платформа оркестрації здатна обслуговувати величезну кількість хостів, управляти та запускати одночасно велику кількість контейнерів Docker або Rocket, відстежувати їх стан, контролювати спільну роботу, проводити балансування навантаження, може забезпечувати спільне розміщення та реплікацію і багато іншого.

Kubernetes - дуже складна система і дозволяє робити практично все, що стосується оркестрації контейнерів. У неї відмінний dashboard, гнучкі політики безпеки, вона підтримує розподілені / мережеві файлові системи без танців з бубнами, володіє широкими можливостями по визначенню своїх метаданих для сервісів, що зручно у великих інфраструктурах. Все б добре, але у неї є один бич - досить мізерна документація, особливо для такої складної системи оркестрації.

Google користується контейнерної технологією вже більше десяти років. Вона має такий унікальний досвід як запуск близько 2 млрд контейнерів за один тиждень. Google вирішила поділитися своїм досвідом у ствоерні платформи, яка може масштабувати запуск контейнерів – це все увійшло у Kubernetes.

Проект переслідує на меті дві цілі. Kubernetes допомагає запускати велику кількість контейнерів на різних хостах, а також допомагає виконувати балансування навантаження. У проекті існує API, через яке можна виконувати майже усі операції віддалено

Сьогодні виділяють наступні концепції:

• Node: машина в кластері Kubernetes.

• Pod: група контейнерів які запускаються як єдине ціле.

• Replication Controllers: виконує реплікацію.

• Services: абстракція над подами та їх політикою доступу.

• Volumes: директорія яка доступна в контейнері.

• kubectl: інтерфейс командного рядка для управління Kubernetes.

Якщо казати про архітектуру, то можна сказати наступне. Працюючи Kubernetes кластер складається з агента, який запускаться на kubelet та master компоненти поверх розподіленого сховища.

Можна розбити архітектуру системи на сервіси, які працюють на кожній ноді і сервіси рівня управління кластера. На кожній ноді Kubernetes запускаються мікросервіси, які обовязкові для керування нодою через master та для старту додатків. Звичайно, на будь-якій ноді буде запущенно Docker.

Kubelet може керувати подами, образами, а також контейнерами та розділами. На буд-якій ноді буде запущен балансувальник. Цей сервіс працює на кожній ноді у кластері і може бути налаштован через Kubernetes API.

Система керування Kubernetes поділяється на компоненти. Сьогодні усі компоненти запускаються через master, але зовсім скоро це змінять, щоб була можливість опрацьовувати відмови кластеру. Усі компоненти працюють разом, щоб була єдина система (один екземпляр кластеру).

Стан майстра можна перевіриту в etcd. Завдяки цьому можна надійно зберігати файли конфігурації та своєчасне сповіщення інших компонентів системи про зміну стану.

Kubernetes API компонент відповідає за роботу api сервера. Він призначений для того, щоб обробляти CRUD операції та виконувати бізнес-логіку, яка була розроблена в інших компонентах. Зазвичай обробляє REST, виконуючи перевірку і оновлюючи стан у etcd.

- Docker Swarm

Docker Swarm - друга за популярністю система оркестрації і це не випадково. Адже компанія Docker була першою, хто запропонував ефективну і зручну для корпоративного використання систему в 2013 р Docker значно спростила розгортання повноцінних віртуальних систем і оркестрації в цілому.

Інструмент контейнерної кластеризації Docker Swarm з'явився трохи пізніше і став частиною платформи Docker. Він дозволяє об'єднувати Docker-хости в загальний віртуальний хост [9].

Логотип Docker Swarm зображений на рисунку 2.6

Рисунок 2.6-Логотип Docker Swarm

Docker Swarm цікава, перш за все, малим і середнім підприємствам, потреби яких не виходять за рамки запуску понад 50 тисяч контейнерів і 1000 нод.

Незважаючи на те, що Docker Swarm займає більш скромні позиції в порівнянні з Kubernetes, її підтримка з боку корпоративного ринку досить вагома. Чого вартий тільки активна підтримка з боку Microsoft Azure! Саме тому дана платформа зараз активно розвивається, і ніхто не знає, яку частку ринку вона займе через пару років.

- Apache Mesos - для кластера

На третьому місці за популярністю система Apache Mesos. Спочатку вона з'явилася як дослідницький проект в Університеті Берклі. Вперше вона була зібрана в повноцінний продукт і представлена публічно в 2009 р. [9].

Apache Mesos - це централізована відмовостійка система для управління кластером. Дозволяє об'єднувати в групи окремі вузли, згідно певним вимогам, а також забезпечувати їх ізоляцію від інших IT-ресурсів.

Логотип Apache Mesos зображений на рисунку 2.7

Рисунок 2.7- Логотип Apache Mesos

Суть роботи Apache Mesos відрізняється від звичної і вже стала традиційною моделі віртуалізації. Традиційний підхід передбачає дроблення обчислювального середовища, що складається з безлічі фізичних машин, на їх віртуальні аналоги. Apache Mesos працює інакше: вона об'єднує існуючі об'єкти в єдиний віртуальний ресурс, формуючи великі кластери і ефективну систему управління серверної інфраструктурою, в якій кожному кластеру виділяється свій індивідуальний пул ресурсів.

Система з підтримкою оркестрації отримала широку підтримку вендорів. В якості операційної системи для центрів обробки даних її використовують понад 60 великих компаній, в тому числі Microsoft, Cisco, Dell, HPE, Autodesk, Twitter, eBay і ін. Apache Mesos буде цікава великим компаніям, яким доводиться працювати з дуже великою кількістю контейнерних кластерів (на відміну від тієї ж Docker Swarm).

2.1.3 Інструменти для навантаження системи

Тестування навантаження (loadtesting) - підвид тестування продуктивності, збір показників та визначення продуктивності і часу відгуку програмно-технічної системи або пристрою у відповідь на зовнішній запит з метою встановлення відповідності вимогам, які висуваються до даної системи (пристрою) [10].

Для дослідження часу відгуку системи на високих або пікових навантаженнях виробляється стрес-тестування, при якому створювана на систему навантаження перевищує нормальні сценарії її використання. Не існує чіткої межі між навантажувальним і стрес-тестуванням, однак ці поняття не варто змішувати, так як ці види тестування відповідають на різні бізнес-питання і використовують різну методологію.

Слід зазначити, що для більшості видів тестування продуктивності використовується один і той же інструментарій, що вміє виконувати типові завдання.

Існує поширене помилкове розуміння того, що інструменти для навантажувального тестування системи - це інструменти такі ж за принципом запису і відтворення як і інструменти для автоматизації регресійного тестування . Інструменти для навантажувального тестування працюють на рівні протоколу, тоді як інструменти для автоматизації регресійного тестування працюють на рівні об'єктів графічного інтерфейсу користувача.

Існують різні інструменти для навантаження системи. Вони схожі між собою але мають відмінності при встановлені та використанні та роботою з ними. Найбільш відомі такі інструменти Jmeter, LoadRunner, Gatling. Виходячи з умов дипломної роботи найкраще підійде Jmeter, за йогу більш менш зручнійсть у використані та настройку, порівняно з іншими.

- JMeter

  Apache JMeter — інструмент для проведення навантажувального тестування,

що розробляється Apache Software Foundation, підпроекту Jakarta. Хоча спочатку JMeter розроблявся як засіб тестування веб-застосунків, натепер він здатний проводити навантажувальні тести для JDBC-з'єднань, FTP, LDAP, SOAP, JMS, POP3, IMAP, HTTP і TCP.

Логотип JMeter зображений рисунку 2.8

Рисунок 2.8- Логотип JMeter

Є частим вибором для багатьох варіантів і типів навантажувальноготестування: зручний GUI, незалежність від платформи. Підтримка багатопоточності, розширюваність, відмінні можливості по створенню звітів, підтримка багатьох протоколів для запитів. Завдяки модульній архітектурі JMeter можна розширити в потрібну користувачу сторону, реалізуючи навіть вельми екзотичні сценарії тестування - причому, якщо жоден з написаних співтовариством плагінів нас не влаштує, можна взяти API і написати власний. При необхідності з JMeter можна вибудувати, хоч і обмежену, але розподілене тестування, коли навантаження буде створюватися відразу декількома машинами [11].

Однією із зручних функцій JMeter є робота в режимі проксі: вказуємо в налаштуваннях браузера в якості проксі «127.0.0.1:8080» і відвідуємо браузером потрібні нам сторінки потрібного сайту, тим часом JMeter зберігає всі наші дії і всі супутні запити у вигляді скрипта, який пізніше можна буде відредагувати, як потрібно - це робить процес створення HTTP-тестів помітно простіше.

До речі, остання версія 3.2, що вийшла в квітні цього року, навчилася

віддавати результати тестування в InfluxDB за допомогою асинхронних HTTP-запитів. Правда, починаючи саме з версії 3.2, JMeter став вимагати тільки Java 8, але це, напевно, не найвища ціна за прогрес.

Зберігання тестових сценаріїв у JMeter реалізовано в XML-файлах, що, як виявилося, створює масу проблем: їх зовсім незручно писати руками, як незручна і робота з такими файлами в системах управління версіями . Конкуруючі на поле навантажувального тестування продукти, такі, як Яндекс.Танк або Taurus, навчилися самостійно і на льоту формувати файли з тестами і передавати їх в JMeter на виконання, таким чином користуючись силою і досвідом JMeter, але даючи можливість користувачам створювати тести у вигляді більш читаються і легше збережених в CVS тестових скриптів.

- LoadRunner

LoadRunner - це інструмент для тестування програмного забезпечення від Micro Focus. Він використовується для тестування програм, вимірювання поведінки та продуктивності системи при навантаженні. LoadRunner може імітувати тисячі користувачів одночасно за допомогою прикладного програмного забезпечення, записуючи та пізніше аналізуючи ефективність ключових компонентів програми [11].

Це один із давно існуючий на ринку і в певних колах дуже відомий продукт, більшому поширенню якого завадила прийнята компанією-виробником політика ліцензування. Після злиття підрозділу ПО компанії Hewlett Packard Enterprise з Micro Focus International, звичну назву HPE LoadRunner змінилося на Micro Focus LoadRunner.

Інтерес представляє логіку створення тесту, де багато віртуальних користувачів паралельно щось роблять з тестованим додатком. Це дає можливість не тільки оцінити здатність додатки обробити потік одночасних запитів, але і зрозуміти, як впливає робота одних користувачів, активно щось роблять з сервісом, на роботу

інших. При цьому мова йде про широкий вибір протоколів взаємодії з тестованим додатком.

Логотип LoadRunner зображений рисунку 2.9

Рисунок 2.9- Логотип LoadRunner

HP свого часу створила дуже хороший набір інструментів автоматизації функціонального і навантажувального тестування, які, при необхідності, інтегруються в процес розробки ПО, і LoadRunner вміє інтегруватися з ними (зокрема, з HP Quality Center, HP QuickTest Professional).

Деякий час тому виробник вирішив повернутися обличчям до тих, хто не готовий відразу платити за ліцензію, і поставляє LoadRunner з безкоштовною ліцензією (де вписана ліміт на 50 віртуальних користувачів, і заборонена невелика частина всього набору підтримуваних протоколів), а гроші беруться за подальше розширення можливостей . Складно сказати, наскільки це сприятиме підвищенню інтересу до цього, без сумніву, цікавого інструменту, при наявності у нього таких сильних конкурентів.

- Gatling

Gatling - це тестування з відкритим кодом навантаження та працездатності на основі Scala, Akka та Netty. Перший стабільний випуск був опублікований 13 січня 2012 року. У 2015 році засновник Gatling Stéphane Landelle створив компанію "Gatling Corp", присвячену розробці проекту з відкритим кодом [11].

Логотип Gatling зображений рисунку 2.10

Рисунок 2.10- Логотип Gatling

Дуже потужний і серйозний інструмент - в першу чергу, через продуктивності і широти підтримки протоколів «з коробки». Наприклад, там, де тестування навантаження з JMeter буде повільним і болісним (на жаль, плагін підтримки роботи з веб-сокетами не особливо швидкий, що ідейно конфліктує зі швидкістю роботи самих веб-сокетів), Galting майже напевно створить потрібну навантаження без особливих складнощів.

Слід врахувати, що, на відміну від JMeter, Gatling не використовує GUI і взагалі вважається засобом, орієнтованим на дослідну, «грамотну» аудиторію, здатну створити тестовий скрипт у вигляді текстового файлу.

Є у Gatling і мінуси, за які його критикують. По-перше, документація могла б бути і краще, по-друге, для роботи з ним непогано знати Scala: і сам Gatling, як інструмент тестування, і тестові сценарії пишуться саме на цій мові. По-третє, розробники «іноді» в минулому кардинально змінювали API, в результаті можна було виявити, що тести, написані на півроку раніше, «не йдуть» на новій версії, або вимагають доопрацювання / міграції.

У Gatling також відсутня можливість робити розподілене тестування, що обмежує можливі області застосування.

2.1.4 Вибір скриптової мови програмування для автоматизації запуску

Скриптова мова (англ. scripting language) — мова програмування, розроблена для запису «сценаріїв», послідовностей операцій, які користувач може виконувати на комп'ютері. Прості скриптові мови раніше часто називали мовами пакетної

обробки (batch languages або job control languages). Сценарії зазвичай інтерпретуються, а не компілюються [12].

В даній роботі для автоматизації запуску конфігурації буде використовуватись скриптова мова Bash. Так як запуск конфігурацію не великий піддійте звичайний Bash-скрипт.

Bash-скрипти- це сценарії командного рядка, написані для оболонки bash. Існують і інші оболонки, наприклад - zsh, tcsh, ksh, але ми зосередимося на bash. Цей матеріал призначений для всіх бажаючих, єдина умова - вміння працювати в командному рядку Linux [13].

Сценарії командного рядка - це набори тих же самих команд, які можна вводити з клавіатури, зібрані в файли і об'єднані якоюсь спільною метою. При цьому результати роботи команд можуть становити або самостійну цінність, або служити вхідними даними для інших команд. Сценарії - це потужний спосіб автоматизації часто виконуваних дій.

2.1.5 Вибір формату для конфігурації

Для конфігураційних файлів існує маса форматів: списки значень, пари «параметр-значення», INI-файли, YAML, JSON, XML і безліч інших [14].

В даній роботі буде використовуватися формат YAML, замість JSON, тому що в роботі потрібно описати конфігурацію декількох маніфестоф. А формат файлу YAML для цього предназначений.

Пиклади фалів для конфігурації

- YAML

YAML — зручний для читання людиною формат серіалізації даних, концептуально близький до мов розмітки, але орієнтований на зручність введення-виведення типових структур даних багатьох мов програмування.

Назва YAML це рекурсивний акронім YAML Ain't Markup Language («YAML

— не мова розмітки»). У назві відображена історія розвитку: на ранніх етапах мова називалася Yet Another Markup Language («Ще одна мова розмітки») і навіть розглядалася як конкурент XML, але пізніше була перейменована з метою акцентувати увагу на даних, а не на розбивці документів [15].

- JSON

  JSON - це текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією).

  Порівняння JSON & YAML [16]:

- YAML найкраще підходить для конфігурації, де JSON кращий як формат серіалізації або подання даних для ваших API.
- YAML аж ніяк не святий грааль або заміна JSON - необхідно використовувати формат даних, який має найбільш сенс для того, щоб досягти мети.
- Але в деяких випадках YAML має кілька великих переваг перед JSON, включаючи можливість самостійної посилання, підтримку складних типів даних, вбудовані літеральні блоки, коментарі тощо.
- Конфігураційні файли у форматі YAML розроблені для того, щоб читати і редагувати людину.
- JSON, навпаки, розроблений лише для того, щоб читати людину - навмисно бракує функцій для підтримки редагування. Почнемо з відсутності підтримки коментарів - це навмисно залишається поза специфікацією JSON, оскільки це не те, для чого призначений формат.
- Велика перемога YAML - це те, що він підтримує коментарі. Це дуже корисно, особливо коли використовують його для конфігурації. Для обміну даними багато функцій YAML втрачають свою привабливість.

- YAML парсери молодші і, як відомо, менш захищені.
- JSON виграє як формат серіалізації. Він більш чіткий і підходить для обміну даними між apis.
- JSON поставляється з набагато простішими характеристиками, ніж YAML. Ви можете навчитися JSON набагато швидше, ніж ви зможете навчитися YAML, оскільки він не є настільки надійним у своєму наборі функцій.

2.1.6 Вибір веб-серверу

Поняття «веб-сервер» може ставитися як до апаратної начинки, так і до програмного забезпечення. Або навіть до обох частин, які працюють спільно.

З точки зору апратної начинки - це комп'ютер, який зберігає файли сайту (HTML-документи, CSS-стилі, JavaScript-файли, картинки та інші) і доставляє їх на пристрій кінцевого користувача (веб-браузер). Він підключений до мережі Інтернет і може бути доступний через доменне ім'я, подібне mozilla.org.

З точки зору програмного забезпечення, веб-сервер включає в себе кілька компонентів, які контролюють доступ веб-користувачів до розміщених на сервері файлів, як мінімум - це HTTP-сервер. HTTP-сервер - це частина програмного забезпечення, яка розуміє URL'и (веб-адреси) і HTTP (протокол, який ваш браузер використовує для перегляду веб-сторінок).

В даній роботі буде використаний веб-сервер Nginx, виходячи з порівняльної таблиці 3.1.

Приклади веб-серверів:

- Apache

Apache HTTP Server був розроблений Робертом Маккула в 1995 році, а з 1999 року розробляється під управлінням Apache Software Foundation - фонду розвитку програмного забезпечення Apache. Так як HTTP сервер це перший і самий популярний проект фонду його зазвичай називають просто Apache [17].

Веб-сервер Apache був найпопулярнішим веб-сервером в інтернеті з 1996 року. Завдяки його популярності у Apache сильна документація і інтеграція зі стороннім софтом.

Адміністратори часто вибирають Apache через його гнучкості, потужності і широку поширеність. Він може бути розширений за допомогою системи динамічно завантажуваних модулів і виконувати програми на великій кількості різних мов програмування без використання зовнішнього програмного забезпечення.

Логотип Apache зображений на рисунку 2.11

Рисунок 2.11- Логотип Apache

- Короткий огляд Apache

Apache був розроблений для доставки веб-контенту, доступ до якого здійснюється через Інтернет. Він відомий тим, що грав ключову роль в початковому зростанні інтернету. Apache - це програмне забезпечення з відкритим вихідним кодом, розроблене і підтримуване відкритим співтовариством розробників і працює в самих різних операційних системах. Архітектура включає в себе ядро Apache і модулі. Основний компонент надає базову серверну функцію, тому він отримує підключення і управляє паралелізмом. Різні модулі відповідають різним функціям, які виконуються по кожному запиту. Конкретне розгортання

Apache може бути налаштоване для включення різних модулів, таких як функції безпеки, управління динамічним контентом або для базової обробки HTTP-запитів.

Модель «один сервер робить все» стала ключем до раннього успіху Apache. Однак у міру збільшення рівня трафіку і збільшення кількості веб-сторінок і обмеження продуктивності настройка Apache на роботу з реальним трафіком ускладнювала.

- Nginx

У 2002 році Ігор Сисоєв почав роботу над Nginx для того щоб вирішити проблему C10K - вимога до ПО працювати з 10 тисячами одночасних з'єднань. Перший публічний реліз був випущений в 2004 році, поставлена мета була досягнута завдяки асинхронної event-driven архітектурі.

Nginx почав набирати популярність з моменту релізу завдяки своїй легковажності (light-weight resource utilization) і можливості легко масштабуватись на мінімальному залізі. Nginx чудовий при віддачі статичного контенту і спроектований так, щоб передавати динамічні запити іншому ПО призначеному для їх обробки.

Адміністратори часто вибирають Nginx через його ефективного споживання ресурсів і чуйності під навантаженням, а також через можливість використовувати його і як веб-сервер, і як проксі.

- Короткий огляд Nginx

Nginx був розроблений спеціально для усунення обмежень продуктивності веб-серверів Apache. Продуктивність і масштабованість Nginx обумовлені архітектурою, керованої подіями. Він значно відрізняється від підходу Apache. У Nginx кожен робочий процес може одночасно обробляти тисячі HTTP-з'єднань.

Отже, Nginx - це легка, масштабована і високопродуктивна реалізація. Ця архітектура робить обробку великих і флуктуючих навантажень на дані набагато більш передбачуваною з точки зору використання ОЗУ, використання ЦП і затримки [18].

Nginx також має багатий набір функцій і може виконувати різні ролі сервера:

1. Зворотний проксі-сервер для протоколів HTTP, HTTPS, SMTP, POP3 та IMAP
2. Балансувальник навантаження і HTTP-кеш
3. Інтерфейсний проксі для Apache і інших веб-серверів, що поєднує гнучкість Apache з хорошою продуктивністю статичного контенту Nginx

Логотип Nginx зображений на рисунку 2.12

Рисунок 2.12-Логотип Nginx

- Порівняння Apache та Nginx
  - Простота

Розробляти і оновлювати додатки на Apache дуже просто. Модель «одне з'єднання на процес» дозволяє дуже легко вставляти модулі в будь-якій точці логіки веб-обслуговування.

Розробники можуть додавати код таким чином, що в разі збоїв буде порушено тільки робочий процес, що виконує код. Обробка всіх інших з'єднань триватиме без перешкод.

Nginx, з іншого боку, має складну архітектуру, тому розробка модулів не легка. Розробники модулів Nginx повинні бути дуже обережні, щоб створюватиефективний і точний код, без збоїв, і відповідним чином взаємодіяти зі складним ядром, керованим подіями, щоб уникнути блокування операцій.

- Продуктивність

Продуктивність вимірюється тим, як сервер доставляє великі обсяги

контенту в браузер клієнта, і це важливий фактор. Контент може бути статичним.

– Статичний контент

Nginx працює в 2,5 рази швидше, ніж Apache, згідно з тестом продуктивності, що виконується до 1000 одночасних підключень. Інший тест з 512 одночасними підключеннями показав, що Nginx приблизно в два рази швидше і споживає менше пам'яті. Безсумнівно, Nginx має перевагу перед Apache зі статичним контентом.

– Підтримка ОС

Apache працює у всіх операційних системах, таких як UNIX, Linux або BSD, і повністю підтримує Microsoft Windows. Nginx також працює на декількох сучасних Unix-подібних системах і підтримує Windows, але його продуктивність в Windows не так стабільна, як на платформах UNIX.

– Безпека

І Apache, і Nginx є безпечними веб-серверами. Apache Security Team існує, щоб надати допомогу і поради проектам Apache з питань безпеки і координувати обробку вразливостей безпеки. Важливо правильно налаштувати сервери і знати, що робить кожен параметр в налаштуваннях.

Порівняльну характеристику зображено в таблиці 3.1

«*Таблиця 3.1* - Порівняльна характеристика Веб-серверів [18]»

Особливість	Apache	Nginx
Простота	Легко розробляти і впроваджувати інновації завдяки своїй моделі «одне з'єднання на процес»	Складний в розробці, оскільки він має складну архітектуру для одночасної обробки декількох з'єднань.
Продуктивність - Статичний контент	Продуктивність - Статичний контент	В 2,5 рази швидше ніж Apache і споживає менше пам'яті

«*Продовження таблиці 1.2* - Порівняльна характеристика Веб-серверів»

Безпека	Це безпечний веб-сервер. Розуміння і настройка функцій безпеки важливі	Це безпечний веб-сервер. Розуміння і настройка функцій безпеки важливі
Гнучкість	Можна налаштувати, додавши модулі. Apache мав динамічне завантаження модулів найдовше.	Nginx версії 1.11.5 і Nginx Plus Release R11 представили сумісність для динамічних модулів.
Підтримка і документація	Відмінна підтримка та документація доступні, як це було на ринку протягом дуже довгого часу	Незважаючи на слабке початок підтримки і документації для Nginx, він швидко зростав, тому тепер у нього є відмінна підтримка ресурсів і доступна документація.

2.2 Архітектура системи

Архітектура система автоматизованого балансування навантаження для мікро-сервісної архітектури складається із трьох контейнерів Dcoker, в яких завернуті два веб-сервери на базі Nginx на яких знаходяться версії сайту. Третій контейнер потрібний для балансування навантаження системи на базі Nginx за допомогою Load Balancer.

Платформа Kubernetes потрібна для масштабування та роботи з контейнерами.

Робота ситсеми залежіть від навантаження користувачами через інструмент

Jmeter. Балансировка здійснюється за допомогою YAML файлу в якому прописані порти та налаштування, як саме розподілювати навантаження.

Автоматизація запуску системи відбувається завдяки написаному скрипту на мові bash.

На рисунку 2.13 показано балансування навантаження для мікро-сервісної архітектури.

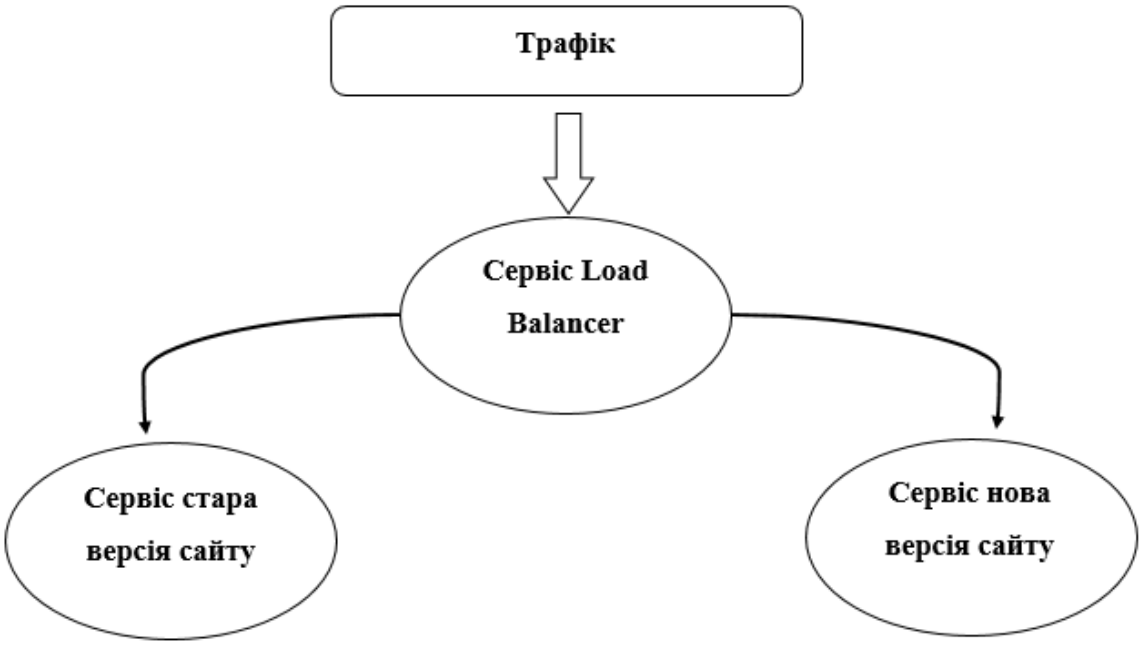


Рисунок 2.13- Балансування навантаження для мікро-сервісної архітектури

3 РЕАЛІЗАЦІЯ СИСТЕМИ

3.1 Встановлення потрібних інструментів

Потрібно встановити такі інструменти Docker-для контейнерів, Kubernetes – для масштабування та роботи контейнерів та Jmeter-для навантаження системи.

3.1.1 Встановлення Docker

У роботі будемо використовувати Ubuntu 18.04.

- Видалити старі версії [19]

Більш ранні версії Docker називалися docker, docker.io або docker-engine. Якщо вони встановлені, видаліть їх:

```
$ sudo apt-get remove docker docker-engine docker.io containerd runc
```

- Методи установки

Можна встановити Docker Engine по-різному, в залежності від потреб:

1. Більшість користувачів налаштовують репозиторії Docker і встановлюють з них, щоб спростити завдання встановлення та оновлення. Це рекомендований підхід.
2. Деякі користувачі завантажують пакет DEB і встановлюють його вручну, а також повністю керують оновленнями. Це корисно в таких ситуаціях, як установка Docker в системах з повітряним зазором без доступу до Інтернету.
3. У середовищах тестування і розробки деякі користувачі вибирають автоматичні допоміжні сценарії для установки Docker.

- Встановити за допомогою сховища

Перед першою установкою Docker Engine на нову хост-машину необхідно налаштувати репозиторій Docker. Після цього можна встановити і оновити Docker.

Налаштування сховища

1. Оновлення apt індекс пакета і встановлення пакетів, щоб дозволити apt використовувати сховища поверх HTTPS:

```
$ sudo apt-get update

$ sudo apt-get install \

    apt-transport-https \

    ca-certificates \

    curl \

    gnupg-agent \

    software-properties-common
```

2. Додаваня офіційний GPG-ключ Docker:

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
```

Перевірка, чи є ключ із відбитком 9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88, шукаючи останні 8 символів відбитка.

```
$ sudo apt-key fingerprint 0EBFCD88

pub   rsa4096 2017-02-22 [SCEA]

      9DC8 5822 9FC7 DD38 854A  E2D8 8D81 803C 0EBF CD88

uid   [ unknown] Docker Release (CE deb) <docker@docker.com>

sub   rsa4096 2017-02-22 [S]
```

3. Використати наступну команду для налаштування стабільного сховища. Щоб додати нічний або тестовий сховище, потрібно додати слово nightly або test (або те й інше) після слова стабільного в наведених нижче командах. Дізнайтеся про нічні та тестові канали.

```
 $ sudo add-apt-repository \

   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \

   $(lsb_release -cs) \ stable"
```

- Встановити Docker Engine

1. Оновлення індекса підходящого пакету та встановити останню версію Docker Engine та контейнер.

```
$ sudo apt-get update
```

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io
```

2. Щоб встановити конкретну версію Docker Engine, потрібно вибрати доступні версії в репозиторії:
   - Щоб побачити доступні версії:

```
apt-cache madison docker-ce
```

```
  docker-ce | 5:18.09.1~3-0~ubuntu-xenial |
https://download.docker.com/linux/ubuntu  xenial/stable amd64
Packages
```

```
  docker-ce | 5:18.09.0~3-0~ubuntu-xenial |
https://download.docker.com/linux/ubuntu  xenial/stable amd64
Packages
```

```
  docker-ce | 18.06.1~ce~3-0~ubuntu       |
https://download.docker.com/linux/ubuntu  xenial/stable amd64
Packages
```

```
  docker-ce | 18.06.0~ce~3-0~ubuntu       |
https://download.docker.com/linux/ubuntu  xenial/stable amd64
Packages
```

...

   - Встановити конкретну версію, використовуючи рядок версії з другого стовпця, наприклад, 5:18.09.1~3-0~ubuntu-xenial

```
$ sudo apt-get install docker-ce=<VERSION_STRING> docker-ce-
cli=<VERSION_STRING> containerd.io
```

3. Переконатись, що Docker Engine встановлено правильно, запустивши зображення hello-world.

```
$ sudo docker run hello-world
```

Ця команда завантажує тестове зображення і запускає його в контейнері. Коли контейнер запускається, він друкує інформаційне повідомлення і завершує роботу.

Docker Engine встановлений і працює. Docker Группа створена, але жоден користувач не додав до нього. Потрібно завжди використовувати sudo для запуску команд Docker [19].

- Після встановлення docker для Linux

Демон Docker прив'язується до сокета Unix замість порту TCP. За замовчуванням Unix-сокет належить користувачу root, а інші користувачі можуть отримати доступ до нього лише за допомогою sudo. Демон Docker завжди працює як користувач root.

Якщо не потрібно завжди попередньо виконувати команду docker на sudo, створити групу Unix під назвою docker та додайте до неї користувачів. Коли запускається демон Docker, він створює Unix-сокет, доступний членам Docker-групи.

Щоб створити docker групу і додати свого користувача:

1. Створити docker групу.

```
$ sudo groupadd docker
```

2. Додати свого користувача в docker групу.

```
$ sudo usermod -aG docker $USER
```

3. Вийти з системи і знову увійти в неї, щоб знову оцінити членство в групі.
   - При тестуванні на віртуальній машині може знадобитися перезапустити віртуальну машину, щоб зміни вступили в силу.
   - У Linux ви також можете запустити наступну команду, щоб активувати зміни в групах:

```
$ newgrp docker
```

4. Переконатися , що можна запускати docker команди без sudo

```
$ docker run hello-world
```

Ця команда завантажує тестове зображення і запускає його в контейнері. Коли контейнер запускається, він друкує інформаційне повідомлення і завершує роботу.

3.1.2 Встановлення Minikube

Minikube - це інструмент, який дозволяє легко запускати Kubernetes локально. Minikube запускає одноузловой кластер Kubernetes всередині віртуальної машини (ВМ) на вашому ноутбуці для користувачів, що бажають випробувати Kubernetes або щодня розробляти його.

Можливості Minikube [20]:

- Minikube підтримує такі можливості Kubernetes:
- DNS
- сервіси NodePort
- Словники конфігурації (ConfigMaps) і секрети (Secrets)
- Панель управління (Dashboard)
- Середовище виконання контейнера: Docker, CRI-O і containerd
- Підтримка CNI (Container Network Interface)
- Ingress
    - Методи встановлення minikube

1. Установка Minikube через пакет

Доступні експериментальні пакети для Minikube; можна завантажити пакети для Linux (AMD64) зі сторінки релізів Minikube на GitHub. Використовувати пакетний менеджер в вашому дистрибутиві Linux для установки потрібного пакету.

2. Установка Minikube за допомогою прямого посилання
3. Установка Minikube через Homebrew

- **Підготовка до роботи**

Щоб перевірити, чи підтримується віртуалізація в Linux, потрібно виконати наступну команду і перевірити, що висновок не порожній:

```
$ grep -E --color 'vmx|svm' /proc/cpuinfo
```

Використовувана мажорна версія kubectl не повинна відрізнятися від тієї, яка використовується в кластері. Наприклад, версія v1.2 може працювати з версіями v1.1, v1.2 і v1.3. Використання останньої версії kubectl допоможе уникнути непередбачених проблем.

- **Методи встановлення kubectl**

Інструмент командного рядка Kubernetes kubectl дозволяє запускати команди для кластерів Kubernetes. Ви можете використовувати kubectl для розгортання додатків, перевірки і управління ресурсів кластера, а також для перегляду логів.

Можна встановити kubectl по-різному, в залежності від потреб:

1. Встановлення двійкового файлу kubectl за допомогою curl в Linux.
2. Встановлення за допомогою вбудованого пакетного менеджера.
3. Встановлення за допомогою стороннього пакетного менеджера.

- **Встановлення двійкового файлу kubectl за допомогою curl в Linux**

1. Завантажити останню версію за допомогою команди:

```
$ curl -LO https://storage.googleapis.com/kubernetes-
release/release/`curl -s https://storage.googleapis.com/kubernetes-
release/release/stable.txt`/bin/linux/amd64/kubectl
```

2. Зробити двійковий файл kubectl виконуваним:

```
$ chmod +x ./kubectl
```

3. Перемістити двійковий файл в директорію з змінної оточення PATH:

```
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

4. Переконатися, що використовується остання версія:

```
$ kubectl version --client
```

- **Встановлення Hypervisor**

Для роботи minikube потрібен Hypervisor. Якщо ще не встановлено гипервизор, встановити один з них:

• KVM, який також використовує QEMU

• VirtualBox

Minikube також підтримує опцію --vm-driver = none, яка запускає компоненти Kubernetes на хості, а не на віртуальній машині. Для використання цього драйвера потрібно тільки Docker і Linux, але не гипервизор.

Якщо ви використовуєте драйвер none в Debian і його похідних, використовуйте пакети .deb для Docker, а не snap-пакет, який не працює з Minikube. Ви можете завантажити .deb-пакети з сайту Docker.

В даній роботі буде використан VirtualBox.

1. Установка VirtualBox Ubuntu 18.04 виконується з офіційного репозиторію розробників. Тільки так можна отримати найсвіжішу версію платформи. Для початку встановити пакети, які понадобяться:

```
$ sudo apt install gcc make linux-headers-$(uname -r) dkms
```

2. Далі необхідно додати репозиторій. Для цього потрібно виконати команду:

```
$ wget -q https://www.virtualbox.org/download/oracle_vbox_2016.asc -
O- | sudo apt-key add -
wget -q https://www.virtualbox.org/download/oracle_vbox.asc -O- |
sudo apt-key add-
sudo sh -c 'echo "deb
http://download.virtualbox.org/virtualbox/debian $(lsb_release -sc)
contrib" >> /etc/apt/sources.list.d/virtualbox.list'
```

3. Перед тим як встановити VirtualBox в Ubuntu 18.04 потрібно оновити репозиторії наступною командою:

```
 $ sudo apt update
```

4. Видалити стару версію або версію з репозиторіїв, якщо вона була встановлена:

```
$ sudo apt purge virtualbox*
```

5. Встановлення VirtualBox:

```
$ sudo apt install virtualbox-6.0
```

- Установка Minikube за допомогою прямого посилання

1. Також є можливість завантажити двійковий файл і використовувати його замість встановлення пакета:

```
$ curl -Lo minikube
https://storage.googleapis.com/minikube/releases/latest/minikube-
linux-amd64 \&& chmod +x minikube
```

2. Щоб виконуваний файл Minikube був доступний з будь-якої директорії потрібно виконати наступні команди:

```
$ sudo mkdir -p /usr/local/bin/
$ sudo install minikube /usr/local/bin/
```

- Перевірка установки

1. Щоб переконатися в тому, що гіпервизор і Minikube були встановлені коректно, запустіть таку команду, яка запускає локальний кластер Kubernetes:

```
$ minikube start --vm-driver=<driver_name>
```

Для викори опції --vm-driver з командою minikube start вкажіть ім'я встановленого вами гіпервизора в нижньому регістрі в заповнювачі <driver_name>. В даній роботі замість <driver_name>, буде <virtualbox>, так як використовуємо VirtualBox.

2. Запустити minikube за допомогою команди

```
$ minikube start
```

3. Після того, як команда minikube start відпрацювала успішно, виконати команду для перевірки стану кластера:

```
$ minikube status
```

4. Якщо кластер вже запущений, то у висновку команди minikube status має бути щось на зразок цього:

```
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

5. Після перевірки на успішність встановлення Minikube з обраним гіпервізором, можна продовжити використовувати Minikube або зупинити кластер. Щоб зупинити кластер виконати команду нижче:

```
$ minikube stop
```

6. Очищення локального стану

Якщо після команди minikube start повертає помилку machine does not exist, то потрібно виконати команду для очищення локального стану

```
$ minikube delete
```

3.1.3 Встановлення Jmeter

Щоб працювати с Jmeter, для початку потрібно встановити Java, а поті сам Jmeter.

Java - мова програмування і обчислювальна платформа, на якій ґрунтуються багато сучасних програми [21].

1. Зазначена нижче команда відобразить версію встановленої в системі java, якщо вона не встановлена, то буде показаний список можливих пакетів з java.

```
$ java -version
```

Команда встановлення Java в Linux Ubuntu 18.04:

```
$ sudo apt-get install openjdk-9-jre
```

Видалення Java

```
$ sudo apt-get purge openjdk*
```

2. Після того як Java встановлено, скопіювати посилання на архів з останньою версією Apache JMeter з офіційного сайту

http://jmeter.apache.org/download_jmeter.cgi і завантажити його:

```
$ wget http://apache.volia.net//jmeter/binaries/apache-jmeter-
4.0.tgz
```

3. Розпакувати архів:

```
$ tar -xf apache-jmeter-4.0.tgz
```

4. Запустити

```
$ cd apache-jmeter-4.0/bin/./jmeter
```

3.2 Розробка веб-серверу для сайту

Для старої та нової версії сайту розроблена конфігурація в YAML файлі, де описано відразу Service, Deployment та HorizontalPodAutoscaler.

Deployment (Розгортання) - це об'єкт у Kubernetes, який дозволяє керувати набором однакових pod. При розгортанні ви оголошуєте один об'єкт у файлі YAML. Цей об'єкт несе відповідальність за створення pod, за те, щоб вони були в курсі, і забезпечили їх достатню кількість. Ви також можете легко масштабувати свої програми за допомогою розгортання Kubernetes.

HorizontalPodAutoscaler (Горизонтальний Pod Autoscaler)- автоматично масштабує кількість pod у контролері реплікації, розгортанні, наборі реплік або наборі станів на основі спостереженого використання центрального процесора (або, за умови користувальницької підтримки, за деякими іншими показниками, що надаються додатком). Автоматичне масштабування горизонтальних pod не застосовується до об'єктів, які не можна масштабувати, наприклад, DaemonSets.

ReplicaSet- Метою ReplicaSet є підтримка стабільного набору реплік Pods, що працює в будь-який момент часу. Як такий, його часто використовують для гарантування наявності визначеної кількості однакових Pods.

Листинг 3.1 - Service файлу старої версії сайту

```
apiVersion: v1
kind: Service
metadata:
name: nginx-v1
  namespace: default
  labels:
    app: nginx-v1
spec:
name: nginx-v1
  namespace: default
```

Продовження листнгу 3.1

```
  labels:
    app: nginx-v1
spec:
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: nginx-v1
```

Листинг 3.2 - Service файлу нової версії сайту.

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-v2
  namespace: default
  labels:
    app: nginx-v2
spec:
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: nginx-v2
```

Service- Абстрактний спосіб розкрити додаток, що працює на наборі Pods як мережеву послугу. У Kubernetes вам не потрібно змінювати свою програму, щоб використовувати незнайомий механізм виявлення послуги. Kubernetes надає Pods власні IP-адреси та єдине DNS-ім’я для набору Pods, і може завантажувати баланс через них.

Листинг 3.3 - Отримаємо файли сайту з github.com та використовуемо в контейнері.

```
initContainers:
      - name: install
        image: busybox
        command:
        - wget
        - "-O"
        - "/work-
```

.Продовження листнгу 3.3

```
        dir/index.html"  # storage to file /usr/share/nginx/html/
        - https://raw.githubusercontent.com/asosluev/diplom/master/
k8s/html/hello-world-v1/index.html
        volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
      dnsPolicy: Default
      volumes:
      - name: workdir
```

3.3 Розробка веб-серверу для балансування навантаження

Він повинен запускатися в кластері перед запуском сайтів.

Він одразу буде слідкувати за навантаженням системи та за необхідністю додавати нові pod, або навпаки зменшувати їх.

Листинг 3.4 - конфигурауї Nginx для балансування навантаження.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  nginx.conf: |
    user nginx;
    worker_processes 1;
    error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;
    events {
      worker_connections  10240;
    }
    http {
        # include       /etc/nginx/mime.types;
        default_type  application/octet-stream;
        log_format  main  '$remote_addr - $remote_user [$time_local
] "$request" '
                          '$status $body_bytes_sent "$http_referer"
 '
                          '"$http_user_agent" "$http_x_forwarded_fo
r"';
        access_log  /var/log/nginx/access.log  main;
        sendfile        on;
        #tcp_nopush     on;
        keepalive_timeout  65;
```

Продовження листингу 3.4

```
        #gzip  on;
           include /etc/nginx/conf.d/*.conf;
    }
  default.conf: |
    upstream app {
      server nginx-v1:80;
      server nginx-v2:80;
      keepalive 1024;
    }
    server {
        listen       80;
        server_name  localhost;
        location / {
          proxy_pass http://app/;
          proxy_http_version 1.1;
            # root   /usr/share/nginx/html;
            # index  index.html index.htm;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
```

ConfigMap - це об’єкт API, який дозволяє зберігати конфігурацію для використання іншими об'єктами. На відміну від більшості об'єктів Kubernetes, які мають a spec, у ConfigMap є data розділ для зберігання елементів (ключів) та їх значень.

В даному випадку конфігурація вкючена в один файл .yaml щоб не перезапускати контейнер кожного разу після внесених змін.

3.4 Розробка автоматизації запуску системи

Для того щоб запустити систему автоматично потрібен bash скрипт. Задяки йому не потрібно вводити багато команд та прейматися тим, що якусь команду не ввели.

Листинг 3.5 - Скрипт myscript.sh

```
#!/bin/bash
clear
folder="diplom"
c_diplom_url="https://github.com/asosluev/diplom.git"
folder="diplom"
repo_url=$c_diplom_url
if [ -d ${folder} ]
then
  echo -e "\e[31m${folder} already exist!\e[39m"
  echo "try to update"
  cd ${folder}
  git remote update
```

Проврека чи існує файли конфігурації на локальній машині, якщо так то обновити їх з репозиторія на github.com, якщо файлыв не маэ то скачати їх з того самого репозіторія.

Листинг 3.6 - Команди які використані в myscript.sh

`echo `sudo apt-get update``	оновити репозиторії
`echo `minikube start --vm-driver=virtualbox``	Запуск кластера Kubernetes на minikube, з використанням virtualbox
`echo `kubectl apply f ./diplom/k8s/yaml/``	Встановіть файли конфігурації
`echo `kubectl get svc,pods -o wide``	Перевірка встановлення конфіураційних файлів
`echo `minikube addons enable metrics-server``	Включення в minikube модолю для работи автомаштабування
`echo `minikube service nginx-loadbalancer --url``	Отримати доступ локально в minikube
`echo `minikube dashboard``	Відкрити dashboard Kubernetes

3.5 Використання Jmater

1. Запуск програми здійснюється в папці де встановленно jmeter/bin за допомогою команди:

   $ ./jmeter.sh

   Після чого відкривається вікно програми з багатьма налаштуваннями в залежності від потреб користувача. В данному випадку буде використовуватись звичайний тест навантаженя користувачами сайту.

   На рисунку 3.1 зображено початковий інтерфейс

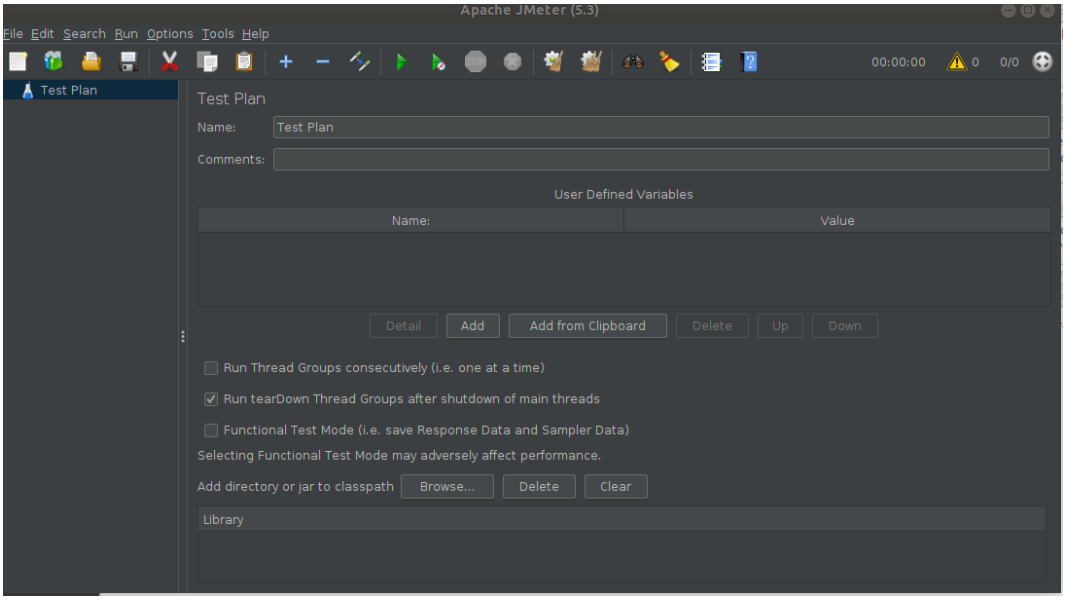


Рисунок 3.1-Інтерфес jmeter

2. Потрібно додати користувачів, які будуть ходити на сторінку. Для цього потрібно правим кліком натиснути на “Test Plan”і далі в / Add / Threads (Users) / Thread Group.

   На рисунку 3.2 зображено початковий інтерфейс

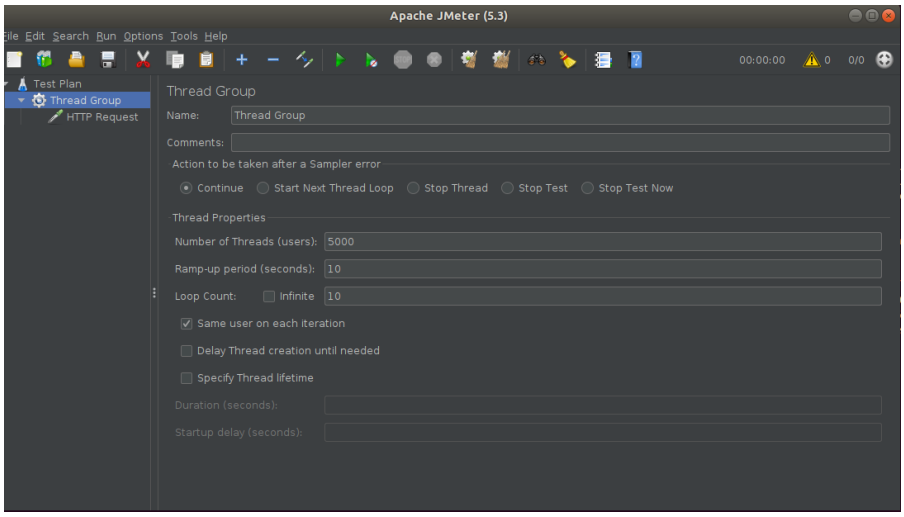


Рисунок 3.2-Інтерфес Test Plan

- Number of Threads - число користувачів, які будуть звертатися за запитом, вказаною в HTTP Request Defaults.

  В даному тесті обрано кількість 5000.
- Ramp-Up Period - час, протягом якого будуть додаватися користувачі. Дозволяє робити плавний і прогнозований старт.

  В даному тесті обрано кількість 10.
- Loop Count - кількість ітерацій. На скріншоті вище значення дорівнює 1, тобто кожен користувач зробить всього 1 запит.

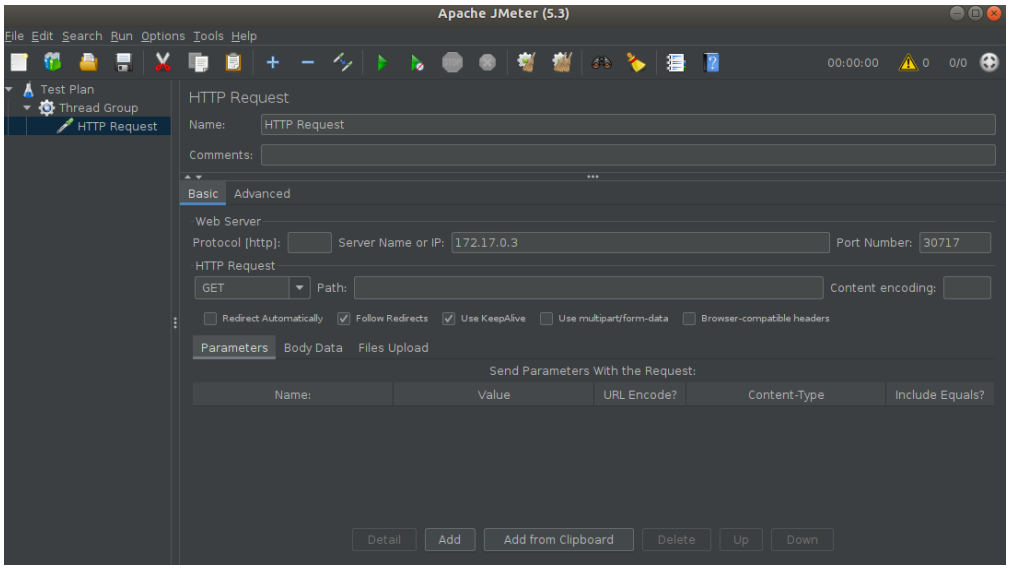
  В даному тесті обрано кількість 10.

3. Для запуску тесту потрібно вказати інструменту адресу сервера, який буде схильний до навантаженні з типом запиту. Це робиться через правий клік по вже доданому Thread Group і вибір Add / Sampler / HTTP Request

   Після додавання елемента можна вказувати адресу сервера.

   В даній роботі використовується minikube у якого адреса змінюється після перезапуску. Тому кожного разу потрібно дивитися її відповідною командую

   На рисунку 3.3 зображено інтерфейс для заповнення адреси сервер



Рисунку 3.3-Інтерфейс для заповнення адреси серверу

4. Тест потрібно зберегти в папку /bin далі, запустити термінал / командний рядок і перейти в папку /bin. Графічну оболонку краще закрити перед запуском тесту. Запуск тесту здійснюється командою:
   sh jmeter.sh -n -t **test_name.jmx** -l **log.jtl**
5. Запускаємо тест і чекаємо його завершення.
   На рисунку 3.4 зображено виконання тесту

```
asosluev@asosluev-Lenovo-B51-35:~/apache-jmeter-5.3/bin$ sh jmeter.sh -n -t test3.jmx -l test3.jtl
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by com.thoughtworks.xstream.core.util.Fields (file:/home/asosluev/ap
r) to field java.util.TreeMap.comparator
WARNING: Please consider reporting this to the maintainers of com.thoughtworks.xstream.core.util.Fields
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
Creating summariser <summary>
Created the tree successfully using test3.jmx
Starting standalone test @ Thu May 28 16:52:33 EEST 2020 (1590673953119)
Waiting for possible Shutdown/StopTestNow/HeapDump/ThreadDump message on port 4445
Warning: Nashorn engine is planned to be removed from a future JDK release
summary +  13295 in 00:00:25 =  539,3/s Avg:   751 Min:     1 Max: 16881 Err:     0 (0,00%) Active: 227
summary +   9214 in 00:01:30 =  102,4/s Avg: 11150 Min:     1 Max: 97487 Err:    36 (0,39%) Active: 399
summary =  22509 in 00:01:55 =  196,3/s Avg:  5008 Min:     1 Max: 97487 Err:    36 (0,16%)
summary +   9491 in 00:00:31 =  309,5/s Avg: 18419 Min:     0 Max: 101787 Err:   838 (8,83%) Active: 31
summary =  32000 in 00:02:27 =  217,1/s Avg:  8986 Min:     0 Max: 101787 Err:   874 (2,73%)
summary +  16683 in 00:00:31 =  537,8/s Avg:  3123 Min:     1 Max: 30619 Err:     0 (0,00%) Active: 323
summary =  48683 in 00:02:58 =  272,9/s Avg:  6977 Min:     0 Max: 101787 Err:   874 (1,80%)
summary +   1317 in 00:00:05 =  283,8/s Avg:  7804 Min:     0 Max: 32114 Err:     0 (0,00%) Active: 0 S
```

Рисунок 3.4 - Виконання тесту

6. Найпростіший звіт - Summary Report, який додається через правий клік по Test Plan і, далі, Add / Listener / Summary Report. Listener'и можна додавати і в ході налаштування тесту. Отже, Summary Report виглядає наступним чином на рисунку 3.5

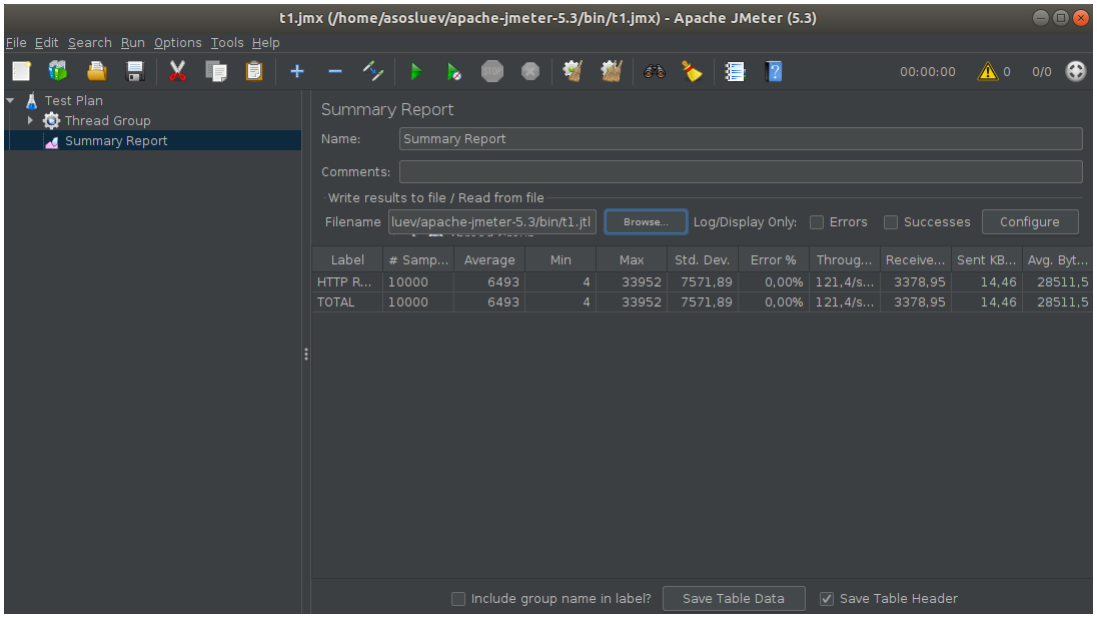


Рисунок 3.5 - Summary Report

7. Що можна зрозуміти з даного звіту?
    - Було зроблено 10000 запитів (Samples) за вказаною в конфіги адресою (1000 користувачів по 10 разів кожен).
    - Середній час відповіді склало трохи менше ніж 6,5 секунди (Average); мінімальний час відповіді трохи менше ніж 0,04 секунди (Min); максимальний час відповіді трохи менше ніж 33,9 секунди (Max).
    - У секунду проходило 121,4 запиту (Throughput).
    - Error%-кількість помилок у відсотках, які повернув сервер
    - Received і Sent KB / sec - кількість отриманих і відправлених даних
    - Avg.Bytes - середня кількість отриманих даних

3.5 Використання платформи Kubernetes

На рисунку 3.6 зображено кластер Kubernetes без тесту навантаження інструментом Jmeter.

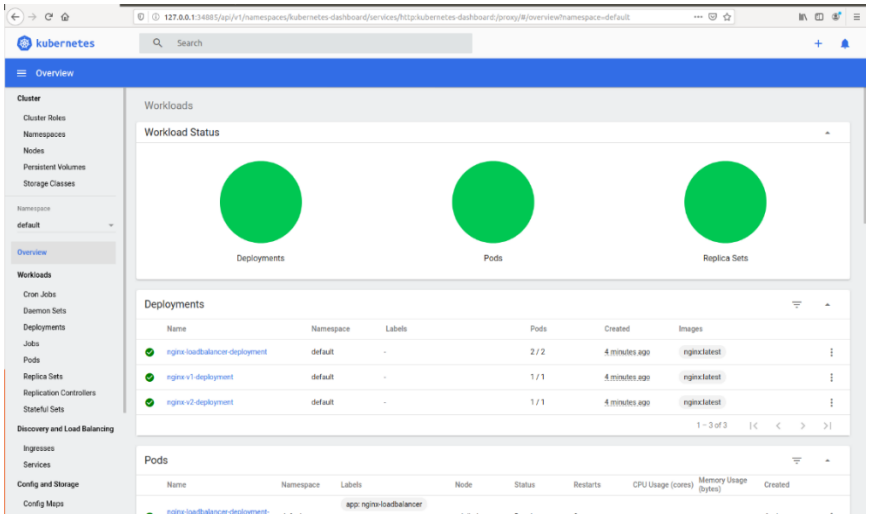


Рисунок 3.6 - кластер Kubernetes без навантаження

На рисунку 3.7 зображено кластер Kubernetes з навантаженням за допомогою інструменту Jmeter. На якому видно як додаються необходіми pod.

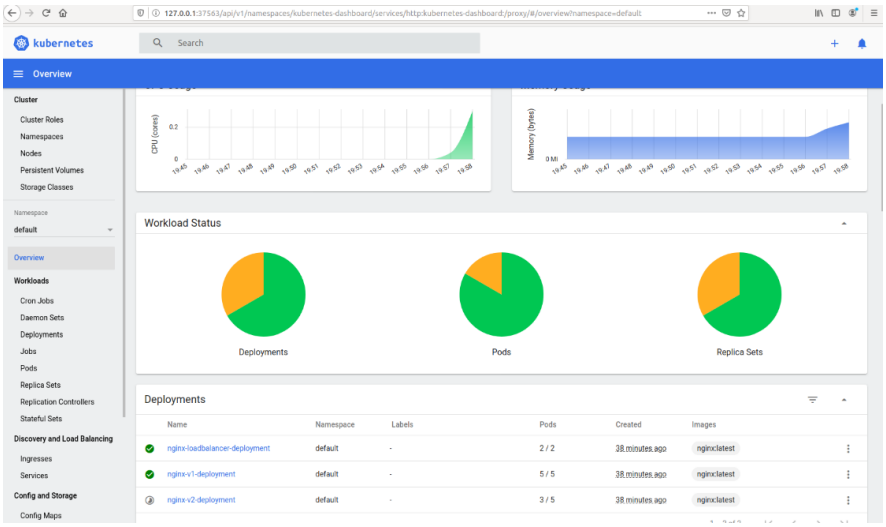


Рисунок 3.7- кластер Kubernetes з навантаженням

ВИСНОВКИ

В ході виконання дипломної роботи було стовренно систему автоматизованого балансування навантаження для мікро-сервісної архітектури на базі Docker. При налаштуванні за замовчуванням система перенаправляє 50% користувачів до новою версії сайту, 50% - до старої. Користувачів перенаправляє для тестування нової версії, щоб не втратити постійних користувачів для цього потрібна стара стабільна версія.

Система працює на платформах такі як Linux, Windows, Mac OS.

Під час виконання роботи були порівняні та використані інструменти контейнеризації, системи автоматизованого балансування, а також систем окестрації контейнерів.

На даний час найбільш популярним інструментом для контейнеризації являється Docker та платформа Kubernetes для налаштування контейнерів та їх масштабування. Балансування навантаження дуже корисний інструмент, щоб платити менше якщо не потрібно багато ресурсів та надавати більш стабільну та швидкодійну систему користувачу при навантаженні серверу. Система розподільного навантаження допомагає тестувальникам набагато швидше проводити тестування, за допомогою користувачів, перенаправляючи їх на бета-версію програмного забезпечення.

В даній дипломній роботі була розроблена конфігурація, завдяки якій можливо переносити сайти з одного серверу на інший за більш короткий час, автоматизували запуску серверів. Це допоможе в подальшому витрачати менше часу на їх запуск. Балансування навантаження працює за принципом є навантаження відкриваються нові контейнери, якщо навантаження спаде то контейнери згортаються автоматично.

ПЕРЕЛІК ПОСИЛАНЬ

1. Docker-начало [Електронний ресурс].-Режим доступу: URL: https://habr.com/ru/post/353238/

2. Руководство по редиректам: как их обнаружить и настроить [Електронний ресурс].-Режим доступу: URL: https://netpeaksoftware.com/ru/blog/rukovodstvo-po-redirektam-kak-ih-obnaruzhit-i-nastroit

3. Розгортання програмного забезпечення [Електронний ресурс]. - Режим доступу: URL: https://uk.wikipedia.org/wiki/Розгортання_програмного_забезпечення

4. Балансировка нагрузки: основные алгоритмы и методы [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/company/selectel/blog/250201/

5. Ubuntu LXD. [Електронний ресурс]. - Режим доступу: URL: https://www.ubuntu.com/cloud/lxd

6. Встановлення та налаштування платформи Docker для створення, розгортання і запуску додатків на прикладі Python Flask application : дип. спец.. комп. наук : 7.05010103 / . – Київ, 2017. – 25-26 с.

7. Docker's documentation. [Електронний ресурс]. - Режим доступу: URL: https://docs.docker.com/

8. Изучаем Docker [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/company/ruvds/blog/439978/

9. Что такое оркестрация контейнеров [Електронний ресурс]. - Режим доступу: URL: https://www.xelent.ru/blog/chto-takoe-orkestratsiya-konteynerov/

10. Нагрузочное тестирование [Електронний ресурс]. - Режим доступу: URL: https://ru.wikipedia.org/wiki/Нагрузочное_тестирование

11. Обзор инструментария для нагрузочного и перформанс-тестув [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/company/jugru/blog/337928/

12. Скриптова мова [Електронний ресурс]. - Режим доступу: URL: https://uk.wikipedia.org/wiki/Скриптова_мова

13. Bash-скрипты: начало [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/company/ruvds/blog/325522/

14. 10 шагов к YAML-дзену [Електронний ресурс]. - Режим доступу: URL: https://habr.com/ru/company/redhatrussia/blog/462125/

15. YAML [Електронний ресурс]. - Режим доступу: URL: https://uk.wikipedia.org/wiki/YAML

16. YAML vs JSON [Електронний ресурс]. - Режим доступу: URL: https://www.json2yaml.com/yaml-vs-json

17. Настройка веб-сервера Nginx [Електронний ресурс]. - Режим доступу: URL: https://school9.perm.ru/docs/nginx.html

18. APACHE VS NGINX – СРАВНЕНИЕ И ПРЕИМУЩЕСТВА [Електронний ресурс]. - Режим доступу: URL: https://wiki.merionet.ru/servernye-resheniya/34/apache-vs-nginx-sravnenie-i-preimushhestva/

19. Install Docker Engine on Ubuntu [Електронний ресурс]. - Режим доступу: URL: https://docs.docker.com/engine/install/ubuntu/

20. Установка Kubernetes с помощью Minikube [Електронний ресурс]. - [Електронний ресурс]. - Режим доступу: URL: https://kubernetes.io/ru/docs/setup/learning-environment/minikube/

21. ІНСТАЛЮВАННЯ JMETER НА WINDOWS, LINUX, MAC — ДЕТАЛЬНО [Електронний ресурс]. - Режим доступу: URL: https://www.quality-assurance-group.com/instalyuvannya-jmeter-na-windows-linux-mac/

ДОДАТКИ

ДОДАТОК А
КОНФІГУРАЦІЯ

Обозначение	Наименование	Примеч.
	Документація	
ЧНТУ	Система автоматизованого балансування навантаження для мікро-сервісної архітектури на базі Docker	
	Текст конфігурації	
	Текст сайту	
	Компоненти	
	Операційна система	
	Microsoft Windows, Linux	
	Web-сервер	
	Nginx 1.18.0	
	Контейнеризація	
	Docker 19.03.08	
	Окестрація	
	Kubernetes(K8s) 1.18	
	Навантаження	
	Jmeter 5.3	

Міністерство освіти та науки України
Чернігівський національний технологічний університет
Кафедра інформаційних та комп'ютерних систем

Система автоматизованого балансування навантаження для мікросервісної архітектури на базі Docker

Текст кофігурації

ЧНТУ

Листов - 35

Виконавець

студент гр. КІ-161 О.В. Сослуєв

Чернігів – 2020

АНОТАЦІЯ

Даний документ містить вихідні тексти веб-сайтів та конфігураційні файли для системи автоматизованого балансування навантаження для мікросервісної.

Вихідні тексти сайтів написані на мові JavaScript, мова розмітки HTML і CSS.Конфігураційні тексти написані в файлі YAML.

ЗМІСТ

1 Файли конфигурації 74
1.1 LoadBalancer 74
1.2 Nginix v1 76
1.3 Nginx v2 78
1.4 Конфігурація nginx 80
1.4.1 Default.conf 80
1.4.2 Nginx.conf 81
2 Файли сайту 83
2.1 Index.html 83
2.2 style.css 89
2.3 football.js 96
2.4 jQuery.js 106

1 ФАЙЛИ КОНФИГУРАЦІЇ

1.1 LoadBalancer

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-loadbalancer
  namespace: default
  labels:
    app: nginx-loadbalancer
spec:
  ports:
  - port: 80
    protocol: TCP
    name: http
    targetPort: 80
  type: LoadBalancer
  selector:
    app: nginx-loadbalancer
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: nginx-conf
data:
  nginx.conf: |
    user nginx;
    worker_processes 1;
    error_log  /var/log/nginx/error.log warn;
    pid        /var/run/nginx.pid;
    events {
      worker_connections  10240;
    }
    http {
        # include       /etc/nginx/mime.types;
        default_type  application/octet-stream;
        log_format  main  '$remote_addr - $remote_user [$time_local]
 "$request" '
                          '$status $body_bytes_sent "$http_referer"
'
                          '"$http_user_agent" "$http_x_forwarded_for
"';
        access_log  /var/log/nginx/access.log  main;
```

```
        sendfile        on;
        #tcp_nopush     on;
        keepalive_timeout  65;
        #gzip  on;
        include /etc/nginx/conf.d/*.conf;
    }
  default.conf: |
    upstream app {
      server nginx-v1:80;
      server nginx-v2:80;
      keepalive 1024;
    }
    server {
        listen       80;
        server_name  localhost;
        location / {
          proxy_pass http://app/;
          proxy_http_version 1.1;
            # root   /usr/share/nginx/html;
            # index  index.html index.htm;
        }
        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   /usr/share/nginx/html;
        }
    }
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-loadbalancer-deployment
  namespace: default
spec:
  selector:
    matchLabels:
      app: nginx-loadbalancer
  replicas: 2 # tells deployment to run 2 pods
  template:
    metadata:
      labels:
        app: nginx-loadbalancer
    spec:
      containers:
      - name: nginx-loadbalancer
        image: nginx:latest
        ports:
        - containerPort: 80
```

```
        volumeMounts:
        - mountPath: /etc/nginx # mount nginx-
conf volumn to /etc/nginx
          readOnly: true
          name: nginx-conf
        - mountPath: /var/log/nginx
          name: log
      volumes:
      - name: nginx-conf
        configMap:
          name: nginx-conf # place ConfigMap `nginx-
conf` on /etc/nginx
          items:
            - key: nginx.conf
              path: nginx.conf
            - key: default.conf
              path: conf.d/default.conf # dig directory
      - name: log
        emptyDir: {}
```

1.2 Nginix v1

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-v1
  namespace: default
  labels:
    app: nginx-v1
spec:
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: nginx-v1
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-v1-deployment
  namespace: default
spec:
  selector:
    matchLabels:
      app: nginx-v1
```

```
  replicas: 1 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx-v1
    spec:
      containers:
      - name: nginx-v1
        image: nginx:latest
        ports:
        - containerPort: 80
        resources:
          limits:
            cpu: 200m
          requests:
            cpu: 100m
        volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html  # storage to file /usr/s
hare/nginx/html/
      initContainers:
      - name: install
        image: busybox
        command:
        - wget
        - "-O"
        - "/work-
dir/index.html"  # storage to file /usr/share/nginx/html/
        - https://raw.githubusercontent.com/asosluev/diplom/master/k
8s/html/hello-world-v1/index.html
        volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
      dnsPolicy: Default
      volumes:
      - name: workdir
        emptyDir: {}
---
# Add HPA
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v1
metadata:
  name: nginx-v1-deployment
spec:
  maxReplicas:  5
  minReplicas: 1
  scaleTargetRef:
```

```
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-v1-deployment
  targetCPUUtilizationPercentage: 2
status:
  currentCPUUtilizationPercentage: 0
  currentReplicas: 1
  desiredReplicas: 1
```

1.3 Nginx v2

```
apiVersion: v1
kind: Service
metadata:
  name: nginx-v2
  namespace: default
  labels:
    app: nginx-v2
spec:
  ports:
  - port: 80
    protocol: TCP
    name: http
  selector:
    app: nginx-v2
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-v2-deployment
  namespace: default
spec:
  selector:
    matchLabels:
      app: nginx-v2
  replicas: 1 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx-v2
    spec:
      containers:
      - name: nginx-v2
        image: nginx:latest
        ports:
        - containerPort: 80
```

```
        resources:
          limits:
            cpu: 100m
          requests:
            cpu: 100m
        volumeMounts:
        - name: workdir
          mountPath: /usr/share/nginx/html  # storage to file /usr/s
hare/nginx/html/
      initContainers:
      - name: install
        image: busybox
        command:
        - wget
        - "-O"
        - "/work-
dir/index.html"  # storage to file /usr/share/nginx/html/
        - https://raw.githubusercontent.com/asosluev/diplom/master/k
8s/html/hello-world-v2/index.html
        volumeMounts:
        - name: workdir
          mountPath: "/work-dir"
      dnsPolicy: Default
      volumes:
      - name: workdir
        emptyDir: {}
---
# Add HPA
kind: HorizontalPodAutoscaler
apiVersion: autoscaling/v1
metadata:
  name: nginx-v2-deployment
spec:
  maxReplicas: 5
  minReplicas: 1
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: nginx-v2-deployment
  targetCPUUtilizationPercentage: 2
status:
  currentCPUUtilizationPercentage: 0
  currentReplicas: 1
  desiredReplicas: 1
```

1.4 Конфігурація nginx

1.4.1 Default.conf

```
server {
    listen       80;
    server_name  localhost;

    #charset koi8-r;
    #access_log  /var/log/nginx/host.access.log  main;

    location / {
        root   /usr/share/nginx/html;
        index  index.html index.htm;
    }

    #error_page  404              /404.html;

    # redirect server error pages to the static page /50x.html
    #
    error_page   500 502 503 504  /50x.html;
    location = /50x.html {
        root   /usr/share/nginx/html;
    }

    # proxy the PHP scripts to Apache listening on 127.0.0.1:80
    #
    #location ~ \.php$ {
    #    proxy_pass   http://127.0.0.1;
    #}

    # pass the PHP scripts to FastCGI server listening on 127.0.0.1:
9000
    #
    #location ~ \.php$ {
    #    root           html;
    #    fastcgi_pass   127.0.0.1:9000;
    #    fastcgi_index  index.php;
    #    fastcgi_param  SCRIPT_FILENAME  /scripts$fastcgi_script_nam
e;
    #    include        fastcgi_params;
    #}

    # deny access to .htaccess files, if Apache's document root
    # concurs with nginx's one
    #
```

```
    #location ~ /\.ht {
    #    deny  all;
    #}
}
```

1.4.2 Nginx.conf

```
user  nginx;
worker_processes  1;

error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;


events {
    worker_connections  1024;
}


http {
    include       /etc/nginx/mime.types;
    default_type  application/octet-stream;

    log_format  main  '$remote_addr - $remote_user [$time_local] "$r
equest" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for"';

    access_log  /var/log/nginx/access.log  main;

    sendfile        on;
    #tcp_nopush     on;

    keepalive_timeout  65;

    #gzip  on;

    include /etc/nginx/conf.d/*.conf;
}
```

1.5 myscript.sh

```
#!/bin/bash
clear

folder="diplom"
```

```
c_diplom_url="https://github.com/asosluev/diplom.git"

folder="diplom"
repo_url=$c_diplom_url
if [ -d ${folder} ]
then
  echo -e "\e[31m${folder} already exist!\e[39m"
  echo "try to update"
  cd ${folder}
  git remote update
  count=$(git rev-list HEAD...origin/master --count)
  if [ $count -gt "0" ]
  then
    echo -e "\e[31m"
    read -r -p  "Update [y/n] " answer
    echo -e "\e[39m"
    if [ "$answer" = "y" ]
    then
      git fetch --all
      git reset --hard origin/master
    fi
  fi
  cd -
else
  echo "${folder} not exist"
  git clone $repo_url
  if [ $? -eq "0" ]
  then
    echo -e "\e[32m${folder} repo clone is Ok!\e[39m"
  fi
fi



echo `minikube start`
#echo $kube



echo `kubectl apply -f ./diplom/k8s/yaml/`
#echo $start

echo `kubectl get svc,pods -o wide`
#echo $com
```

2 ФАЙЛИ САЙТУ

2.1 Index.html

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
                <link href="http://code.ionicframework.com/ionicons/
2.0.1/css/ionicons.min.css" rel="stylesheet" type="text/css">

        <link type="text/css" rel="stylesheet" href="JS/style.css">
        <title>Football-Results</title>
    </head>

    <body>

        <div class="header">
            <div class="last-match">
                <div class="last-match-title">
                    Last Match
                </div>

                <div class="last-match-score">2 : 1</div>
                <div class="team" id="team-1">Green FC</div>
                <div class="team" id="team-2">Black FC</div>

                <div class="btn" id="add-match-btn">
                    Add Match
                </div>
            </div>

        </div>

        <div class="add-match">
            <div class="match-row">
                <select class="team-select" id="team-select-1">
                    <option disabled selected value> -
- select a team -- </option>
                    <option value="Club-1" >AC Milan</option>
                    <option value="Club-2" >Juventus</option>
                    <option value="Club-3" >Real Madryt</option>
                    <option value="Club-
4" >Manchester United</option>
                </select>
```

```
                <input type="number" class="enter-score" id="score-
1" min="0" max="99">
                <input type="number" class="enter-score" id="score-
2" min="0" max="99">

                <select class="team-select" id="team-select-2">
                    <option disabled selected value> -
- select a team -- </option>
                    <option value="Club-1" >Club 1</option>
                    <option value="Club-2" >Club 2</option>
                    <option value="Club-4" >Club 2</option>
                </select>

                <input type="date" class="enter-date" id="match-
date">

            </div>
        </div>

        <div class="table">
            <h2 class="table-title">
                Ranking
            </h2>
            <div class="table-list">

                <div class="table-item" id="table-item-header">
                    <div class="table-item-pos-header">#</div>
                    <div class="table-item-name-header">Team</div>
                    <div class="table-item-mp-header">Matches</div>
                    <div class="table-item-win-header">Wins</div>
                    <div class="table-item-Draw-header">Draws</div>
                    <div class="table-item-Losses-
header">Losses</div>
                    <div class="table-item-pts-header">Points</div>
                    <div class="table-item-delete">
                        <button class="item-delete-btn">
                            <i class="ion-ios-close-outline"></i>
                        </button>
                    </div>
                </div>

                <div class="table-item">
                    <div class="item-pos">1</div>
                    <div id="item-name">Juventus</div>
                    <div class="item-mp">12</div>
                    <div class="item-win">5</div>
```

```
        <div class="item-draw">4</div>
        <div class="item-losses">3</div>
        <div class="item-pts">19</div>
        <div class="table-item-delete">
            <button class="item-delete-btn">
                <i class="ion-ios-close-outline"></i>
            </button>
        </div>
    </div>


    <div class="table-item">
        <div class="table-item-pos">2</div>
        <div class="table-item-name">Śląsk</div>
        <div class="table-item-mp">12</div>
        <div class="table-item-win">5</div>
        <div class="table-item-draw">4</div>
        <div class="table-item-losses">3</div>
        <div class="table-item-pts">19</div>
        <div class="table-item-delete-btn">
            <button class="item-delete">
                <i class="ion-ios-close-outline"></i>
            </button>
        </div>
    </div>


    <div class="table-item">
        <div class="table-item-pos">2</div>
        <div class="table-item-name">AC Milan</div>
        <div class="table-item-mp">12</div>
        <div class="table-item-win">5</div>
        <div class="table-item-draw">4</div>
        <div class="table-item-losses">3</div>
        <div class="table-item-pts">19</div>
        <div class="table-item-delete">
            <button class="item-delete-btn">
                <i class="ion-ios-close-outline"></i>
            </button>
        </div>
    </div>

    <div class="table-item">
        <div class="table-item-pos">2</div>
        <div class="table-item-name">Juventus</div>
        <div class="table-item-mp">12</div>
        <div class="table-item-win">5</div>
        <div class="table-item-draw">4</div>
```

```
                <div class="table-item-losses">3</div>
                <div class="table-item-pts">19</div>
                <div class="table-item-delete">
                    <button class="item-delete-btn">
                        <i class="ion-ios-close-outline"></i>
                    </button>
                </div>
            </div>

            <div class="table-item">
                <div class="table-item-pos">2</div>
                <div class="table-item-name">PSG</div>
                <div class="table-item-mp">12</div>
                <div class="table-item-win">5</div>
                <div class="table-item-draw">4</div>
                <div class="table-item-losses">3</div>
                <div class="table-item-pts">19</div>
                <div class="table-item-delete">
                    <button class="item-delete-btn">
                        <i class="ion-ios-close-outline"></i>
                    </button>
                </div>
            </div>

            <div class="table-item">
                <div class="table-item-pos">2</div>
                <div class="table-item-name">Juventus</div>
                <div class="table-item-mp">12</div>
                <div class="table-item-win">5</div>
                <div class="table-item-draw">4</div>
                <div class="table-item-losses">3</div>
                <div class="table-item-pts">19</div>
                <div class="table-item-delete">
                    <button class="item-delete-btn">
                        <i class="ion-ios-close-outline"></i>
                    </button>
                </div>
            </div>

        </div>
    </div>

    <div class="table">
        <h2 class="table-title">
            Matches
        </h2>
```

```
<div class="match-list">
    <div class="table-item" id="match-item-header">
        <!--<div>#</div>-->
        <div>Date</div>
        <div>Team 1</div>
        <div>Result</div>
        <div>Team 2</div>

        <div class="table-item-delete">
            <button class="item-delete-btn">
                <i class="ion-ios-close-outline"></i>
            </button>
        </div>
    </div>

    <div class="table-item">

        <div>23/03/2015</div>

        <div>Juventus</div>
        <div>2:1</div>
        <div>PSG</div>

        <div class="table-item-delete">
            <button class="item-delete-btn">
                <i class="ion-ios-close-outline"></i>
            </button>
        </div>
    </div>

    <div class="table-item">


        <div>23/03/2015</div>

        <div>Juventus</div>
        <div>2:1</div>
        <div>PSG</div>

        <div class="table-item-delete">
            <button class="item-delete-btn">
                <i class="ion-ios-close-outline"></i>
            </button>
        </div>
    </div>

    <div class="table-item">
```

```
                    <div>23/03/2015</div>

                    <div>Slask</div>
                    <div>2:1</div>
                    <div>Arka</div>

                    <div class="table-item-delete">
                        <button class="item-delete-btn">
                            <i class="ion-ios-close-outline"></i>
                        </button>
                    </div>
                </div>

                <div class="table-item">


                    <div>23/03/2015</div>

                    <div>Lechia</div>
                    <div>2:1</div>
                    <div>Bayern</div>

                    <div class="table-item-delete">
                        <button class="item-delete-btn">
                            <i class="ion-ios-close-outline"></i>
                        </button>
                    </div>
                </div>

            </div>

        </div>

        <div class="add-team">
            <div class="add-team-wrapper">
                <div class="add-team-row">
                    <div class="btn" id="add-team-btn">
                        Add Team
                    </div>

                <input placeholder="Team name... ('clear' for remove
all data.)" class="enter-name" id="add-team-field">
                </div>
            </div>

        </div>
```

```
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.
2.1/jquery.min.js"></script>
        <script src="JS/jQuery.js"></script>
        <script src="JS/football.js"></script>

</html>
```

2.2 style.css

```
*{
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-size: 16px;
    font-family: 'Cabin', sans-serif;
/*    padding i border wliczaja sie do width/height */
}

.clearfix::after {
    content: "";
    display: table;
    clear: both;
}

body {
    color: black;
    background-color: #fffffa;
    position: relative;
    height: 100vh;
}

.header {
    height: 40vh;
    width: 100%;
    background-image: linear-
gradient(rgba(0, 150, 50, 0.4), rgba(0, 0, 0, 0.4)), url(https://obo
i-printcolor.com/wp-
content/uploads/pc_photobase/sport/all_sport/pc_1610005_myah_stadion
.jpg);
    background-size: cover;
    background-position: center;
    position: relative;
}

.last-match {
    position: relative;
```

```
    width: 50%;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    color: #ffffff;
}

.last-match-title {
    font-size: 250%;
    text-align: center;
}

.last-match-score {
    margin-top: 5%;
    font-size: 220%;
    text-align: center;
}

.team {
    display: inline-block;
    font-size: 150%;
}

#team-2 {
    float: right;
}

.btn {
    margin-top: 10%;
    position: relative;
    left: 50%;
    transform: translateX(-50%);
    left: 50%;
    height: 35px;
    width: 200px;
    border: 2px solid white;
/*    border-radius: 20px;*/
    color: white;
    text-align: center;
    vertical-align: middle;
    line-height: 35px;
}

.btn:hover {
    background-color: rgba(0, 255, 108, 0.66);
    cursor: pointer;
}
```

```
.btn:active {
    background-color: rgba(0, 255, 108, 0.5);
}

.add-match {
    padding: 20px;
    width: 100%;
    background-color: #f0f0f0;
}

.match-row {
    width: 60%;
    height: 100%;
    margin-left: 20%;
    text-align: center;
}

.team-select {
    width: 35%;
    height: 44px;
    border: 1px solid black;
    color: inherit;
/*    background-color: #c8ffc1;*/
    background-color: inherit;
}

.team-select:active,
.team-select:focus {
    background-color: #ceffce;
}

#team-select-1 {
    float: left;
}

#team-select-2 {
    float: right;
}

.enter-score {
    border: 1px solid black;
    width: 50px;
    height: 44px;
    color: inherit;
    text-align: center;
    background-color: #ffffff;
```

```
}

.enter-date {
    border: 1px solid black;
    display: block;
    width: 20%;
    margin-left: 40%;
    margin-top: 10px;
    color: inherit;
    background-color: white;
    mar
}

.table {
    width: 60%;
    margin-left: 20%;
}

.table-title {
    text-align: center;
    font-size: 150%;
    font-weight: normal;
    padding-top: 50px;
    padding-bottom: 25px;
    color: #3d3d3d;
}

.table-title:after {
    content: "";
    display: block;
    height: 1px;
/*    width: %;*/
    background: #d6d6d6;
    margin: 10px 35%;
    margin-top: 20px;
}

#table-item-header {
    background-color: #fff;
    padding-bottom: 5px;
}

#table-item-header * {
    display: inline-block;
    font-weight: normal;
    color: #777777;
    font-size: 80%;
```

```
}

#table-item-header > :last-child {
    display: none;
}

.table-item {
    display: flex;
}

.table-item:nth-child(odd) {
    background-color: #f5f5f5;
}

.table-item:nth-child(2) {
    background-color: #ceffce;
    color: #005008;
}

.table-item:last-child {
    background-color: #ffcece;
    color: #500008;
}

.table-item > :nth-child(2),
.table-item > :nth-last-child(2) {
    font-weight: bold;
}

.table-item * {
    display: inline-block;
    flex-basis: 100%;
    text-align: center;
    padding: 10px;
    overflow: hidden;
}

.table-item-delete {
    background-color: white;
    margin: 0;
    padding: 0;
    max-width: 0px;
    overflow: visible;
}

.item-delete-btn {
    opacity: 0;
```

```
/*    transition: opacity 0.3s;*/
    background: none;
    border: none;
    cursor: pointer;
}

.item-delete-btn i {
    margin: 0;
    padding: 0;
}

.item-delete-btn:focus { outline: none; }
.item-delete-btn:active {
    transform: translateX(2px);
    color: red;
}

.table-item:hover .table-item-delete,
.match-item:hover .table-item-delete {
    display: inline-block;
}
.table-item:hover .item-delete-btn,
.match-item:hover .item-delete-btn {
    opacity: 1;
    transition: opacity 0.5s;
}




#match-item-header {
    background-color: #fff;
    padding-bottom: 5px;
}

#match-item-header * {
    display: inline-block;
    font-weight: normal;
    color: #777777;
    font-size: 80%;
}

#match-item-header > :last-child {
    display: none;
}

.match-list .table-item {
```

```
    background-color: white;
    color: black;
}

.match-list .table-item * {
    font-weight: normal;
}

.match-list .table-item:nth-child(odd) {
    background-color: #f5f5f5;
}



.add-team {
    height: 40vh;
    width: 100%;
    background-image: linear-
gradient(rgba(0, 150, 50, 0.4), rgba(0, 0, 0, 0.4)), url(https://upl
oad.wikimedia.org/wikipedia/commons/b/bb/UPC-Arena_Panorama.jpg);
    background-color: #f0f0f0;
    margin-top: 100px;
    background-position: center;
    position: relative;
}

.add-team-wrapper {
    width: 60%;
    margin-left: 20%;
    height: 100%;
    padding: 20px;
}

.add-team .btn {
    color: black;
    border-color: black;
    margin-top: 10px;
    padding: 0;
}

.enter-name {
    position: relative;
    left: 50%;
    transform: translateX(-50%);
    height: 44px;
    width: 300px;
    margin-top: 20px;
    margin-bottom: 50px;
```

```
    z-index: 9999999;
}
```

2.3 football.js

```
var footballController = (function() {

    var Team = function(name) {
        this.name = name;
        this.position = 0;
        this.matchesPlayed = 0;
        this.wins = 0;
        this.draws = 0;
        this.losses = 0;
        this.points = 0;
    };

    var Match = function(id, date, team1, team2, score1, score2) {
        this.id = id;
        this.date = date;
        this.team1 = team1;
        this.team2 = team2;
        this.score1 = score1;
        this.score2 = score2;
    };

    var data = {
        allTeams: [],
        allMatches: []
    };

    var sortTeams = function() {
        function compare(a,b) {
            if (a.points < b.points)
                return 1;
            if (a.points > b.points)
                return -1;
            return 0;
        }

        data.allTeams.sort(compare);
    };

    var updateTeamsPosition = function() {
        data.allTeams.forEach(function(item, index) {
            item.position = index + 1;
        });
```

```
};

var sortMatches = function() {
    data.allMatches.sort(function(a, b) {
        if (b.date > a.date)
            return 1;
        if (b.date < a.date)
            return -1;

        if (b.id > a.id)
            return 1;
        if (b.id < a.id)
            return -1;
    });
};

var setWin = function(team, points, set = 1) {
    team.matchesPlayed += 1 * set;
    team.wins += 1 * set;
    team.points += points * set;
};
var setDraw = function(team, points, set = 1) {
    team.matchesPlayed += 1 * set;
    team.draws += 1 * set;
    team.points += points * set;
};
var setLoss = function(team, set = 1) {
    team.matchesPlayed += 1 * set;
    team.losses += 1 * set;
};

return {

    tmpGetData: function() {
        return data;
    },

    getTeamAndIndexByName: function(name) {
        var team, index;

        data.allTeams.every(function(item, i) {
            if (name === item.name) {
                team = item;
                index = i;
                return false;
            }
            return true;
```

```
        });

        return {team, index};
    },

    getMatchAndIndexByID: function(id) {
        var match, index;

        data.allMatches.every(function(item, i) {
            if (parseInt(id) === item.id) {
                match = item;
                index = i;
                return false;
            }
            return true;
        });

        return {match, index};
    },

    createTeam: function(name) {
        return new Team(name);
    },

    addTeam: function(team) {
            data.allTeams.push(team);
    },

    removeTeam: function(teamIndex) {

        data.allTeams.splice(teamIndex, 1);
    },

    teamExists: function(name) {
        var teamsNames = data.allTeams.map(function(team) {
            return team.name;
        });

        return teamsNames.includes(name);
    },

    createMatch: function(matchInput) {
        var mi = matchInput;

        var id;
        if (data.allMatches.length > 0) {
            id = data.allMatches[0].id + 1;
```

```
            } else {
                id = 1;
            }

            return new Match(id, mi.matchDate, mi.team1, mi.team2, m
i.teamScore1, mi.teamScore2);
        },

        addMatch: function(match) {
            data.allMatches.push(match);
        },

        removeMatch: function(matchIndex) {
            data.allMatches.splice(matchIndex, 1);
        },

        getAllMatches: function() {
            return data.allMatches;
        },

        getAllTeams: function() {
            return data.allTeams;
        },

        updateTeamsStats: function(match, unSet) {
            var team1 = this.getTeamAndIndexByName(match.team1).team
;
            var team2 = this.getTeamAndIndexByName(match.team2).team
;
            if (!team1 || !team2)
                return;
            var score1 = match.score1;
            var score2 = match.score2;

            unSet = unSet ? -1 : 1;

            if (score1 > score2) {
                setWin(team1, 3, unSet);
                setLoss(team2, unSet);
            } else if ( score1 < score2) {
                setWin(team2, 3, unSet);
                setLoss(team1, unSet);
            } else {
                setDraw(team1, 1, unSet);
                setDraw(team2, 1, unSet);
            }
```

```
        },

        updateLists: function() {
            sortTeams();
            updateTeamsPosition();
            sortMatches();
        },

        saveDataToLocalStorage: function() {
            localStorage.footballAppData = JSON.stringify(data);
        },

        loadDataFromLocalStorage: function() {
            if (localStorage.footballAppData) {
                data = JSON.parse(localStorage.footballAppData);
            }
        },

        clearLocalStorage: function() {
            localStorage.removeItem("footballAppData");
            data = {
                allMatches: [],
                allTeams: []
            };
        }

    };
})();



var UIController = (function() {

    var DOMElements = {
        addTeamBtn: '#add-team-btn',
        addTeamInput: '#add-team-field',
        addMatchBtn: '#add-match-btn',
        table: '.table-list',
        teamSelect1: '#team-select-1',
        teamSelect2: '#team-select-2',
        teamScore1: '#score-1',
        teamScore2: '#score-2',
        matchDate: '#match-date',
        matchList: '.match-list',
        bannerTeam1: '#team-1',
        bannerTeam2: '#team-2',
        bannerResult: '.last-match-score',
        bannerTitle: '.last-match-title'
```

```
    };

    return {

        getTeamInput: function() {
            return document.querySelector(DOMElements.addTeamInput).
value;
        },

        getMatchInput: function() {
            return {
                team1: document.querySelector(DOMElements.teamSelect
1).value,
                team2: document.querySelector(DOMElements.teamSelect
2).value,
                teamScore1: document.querySelector(DOMElements.teamS
core1).value,
                teamScore2: document.querySelector(DOMElements.teamS
core2).value,
                matchDate: document.querySelector(DOMElements.matchD
ate).value
            }
        },

        getDOMElements: function() {
            return DOMElements;
        },

        addTeamsToOptions: function(teams) {

            teams.forEach(function(team) {
                var teamHtmlSelect = '<option value="Value" >Name</o
ption>';

                teamHtmlSelect = teamHtmlSelect.replace('Value', tea
m.name);
                teamHtmlSelect = teamHtmlSelect.replace('Name', team
.name);
                document.querySelector(DOMElements.teamSelect1).inse
rtAdjacentHTML('beforeend', teamHtmlSelect);
                document.querySelector(DOMElements.teamSelect2).inse
rtAdjacentHTML('beforeend', teamHtmlSelect);

            });

        },
```

```
        addTeam: function(team) {
            var teamHtmlDiv = '<div class="table-item" id="item-
name"><div class="item-
pos">Position</div><div>ClubName</div><div class="item-
mp">MatchesPlayed</div><div class="item-
win">Wins</div><div class="item-draw">Draws</div><div class="item-
losses">Losses</div><div class="item-
pts">Points</div><div class="table-item-delete"><button class="item-
delete-btn"><i class="ion-ios-close-
outline"></i></button></div></div>';

            teamHtmlDiv = teamHtmlDiv.replace('Position', team.posit
ion);
            teamHtmlDiv = teamHtmlDiv.replace('item-
name', team.name);
            teamHtmlDiv = teamHtmlDiv.replace('ClubName', team.name)
;
            teamHtmlDiv = teamHtmlDiv.replace('MatchesPlayed', team.
matchesPlayed);
            teamHtmlDiv = teamHtmlDiv.replace('Wins', team.wins);
            teamHtmlDiv = teamHtmlDiv.replace('Draws', team.draws);
            teamHtmlDiv = teamHtmlDiv.replace('Losses', team.losses)
;
            teamHtmlDiv = teamHtmlDiv.replace('Points', team.points)
;

            document.querySelector(DOMElements.table).insertAdjacent
HTML('beforeend', teamHtmlDiv);
        },

        addMatch: function(match) {
            var matchHtmlDiv = '<div class="table-item" id="Match-
ID"><div>Date</div><div>Team1</div><div>Score1 : Score2</div><div>Te
am2</div><div class="table-item-delete"><button class="item-delete-
btn"><i class="ion-ios-close-outline"></i></button></div></div>';

            matchHtmlDiv = matchHtmlDiv.replace('Match-ID', 'match-
' + match.id);
            matchHtmlDiv = matchHtmlDiv.replace('Date', match.date);
            matchHtmlDiv = matchHtmlDiv.replace('Team1', match.team1
);
            matchHtmlDiv = matchHtmlDiv.replace('Score1', match.scor
e1);
            matchHtmlDiv = matchHtmlDiv.replace('Score2', match.scor
e2);
            matchHtmlDiv = matchHtmlDiv.replace('Team2', match.team2
);
```

```
            document.querySelector(DOMElements.matchList).insertAdja
centHTML('beforeend', matchHtmlDiv);
        },

        updateBanner: function(match) {
            document.querySelector(DOMElements.bannerTeam1).innerHTM
L = match.team1;
            document.querySelector(DOMElements.bannerTeam2).innerHTM
L = match.team2;
            document.querySelector(DOMElements.bannerResult).innerHT
ML = match.score1 + " : " + match.score2;
        },

        clearList: function(htmlElement) {
            htmlElement = document.querySelector(htmlElement);
            while (htmlElement.childNodes.length > 2) {
                htmlElement.removeChild(htmlElement.lastChild);
            }
        },

        clearBanner: function() {
            document.querySelector(DOMElements.bannerTeam1).innerHTM
L = ". . .";
            document.querySelector(DOMElements.bannerTeam2).innerHTM
L = ". . .";
            document.querySelector(DOMElements.bannerResult).innerHT
ML = ". . .";
        }

    };
})();



var controller = (function(UICtrl, fbCtrl) {

    var setEvtListeners = function() {
        var doms = UICtrl.getDOMElements();

        document.querySelector(doms.addTeamBtn).addEventListener('cl
ick', addTeam);
        document.querySelector(doms.addMatchBtn).addEventListener('c
lick', addMatch);
        document.addEventListener('keypress', function(evt) {
            if (evt.keyCode === 13 && document.querySelector(doms.ad
dTeamInput) === document.activeElement) {
                addTeam();
```

```
            } else if (evt.keyCode === 13) {
                addMatch();
            }
        });

        document.querySelector(doms.matchList).addEventListener('cli
ck', deleteMatch);
        document.querySelector(doms.table).addEventListener('click',
 deleteTeam);

    };

    var deleteTeam = function(evt) {
        var teamName = evt.target.parentNode.parentNode.parentNode.i
d;
        if (teamName && confirm("Do you really want to delete the te
am?")) {

            var teamWrap = fbCtrl.getTeamAndIndexByName(teamName);
            fbCtrl.removeTeam(teamWrap.index);
            updateUI();
            fbCtrl.saveDataToLocalStorage();
        }
    };

    var deleteMatch = function(evt) {
        var matchID = evt.target.parentNode.parentNode.parentNode.id
;
        matchID = matchID.split("-")[1];

        if (matchID && confirm("Do you really want to delete the mat
ch?")) {
            var matchWrap = fbCtrl.getMatchAndIndexByID(matchID);
            fbCtrl.updateTeamsStats(matchWrap.match, true);
            fbCtrl.removeMatch(matchWrap.index);
            updateUI();
            fbCtrl.saveDataToLocalStorage();
        }

    };

    var addTeam = function() {

        var teamName = UICtrl.getTeamInput();

        if (teamName === "clear") {
```

```
            if (confirm("This operation will remove all data. Do you
want to continue?")) {
                fbCtrl.clearLocalStorage();
            }
        } else if (teamName === "") {
            alert("Name can't be blank.");
        } else if (fbCtrl.teamExists(teamName)) {
            alert("That team already exists.");
        } else {
            var newTeam = fbCtrl.createTeam(teamName);
            fbCtrl.addTeam(newTeam);
            fbCtrl.saveDataToLocalStorage();
        }

        updateUI();
    };

    var addMatch = function() {
        var mInput = UICtrl.getMatchInput();

        for (var key in mInput) {
            if (mInput[key] === "") {
                alert("Enter data.");
                return;
            }
        }
        if (mInput.team1 === mInput.team2) {
            alert("Choose another team.");
            return;
        }

        var newMatch = fbCtrl.createMatch(mInput);
        fbCtrl.addMatch(newMatch);
        fbCtrl.updateTeamsStats(newMatch);

        fbCtrl.saveDataToLocalStorage();
        updateUI();
    };

    var updateUI = function() {
        var doms = UICtrl.getDOMElements();

        fbCtrl.updateLists();
        UICtrl.clearList(doms.table);
        UICtrl.clearList(doms.matchList);

        var teams = fbCtrl.getAllTeams();
```

```
        var matches = fbCtrl.getAllMatches();

        if (matches.length > 0) {
            UICtrl.updateBanner(matches[0]);
        }

        teams.forEach(function(team) {
            UICtrl.addTeam(team);
        });
        UICtrl.clearList(UICtrl.getDOMElements().teamSelect1);
        UICtrl.clearList(UICtrl.getDOMElements().teamSelect2);
        UICtrl.addTeamsToOptions(fbCtrl.getAllTeams());

        matches.forEach(function(match) {
            UICtrl.addMatch(match);
        });

        document.querySelector(doms.addTeamInput).value = "";
    }

    return {

        initialize: function() {
            setEvtListeners();
            UICtrl.clearBanner();
            fbCtrl.loadDataFromLocalStorage();
            updateUI();
        }

    };
})(UIController, footballController);

controller.initialize();
```

2.4 jQuery.js

```
$(".add-match").hide();
 $("#add-match-btn").click(function() {
     $(".add-match").slideDown();
      });
```