Міністерство освіти і науки України

Львівський національний університет імені Івана Франка

Факультет прикладної математики та інформатики

Кафедра інформаційних систем

МАГІСТЕРСЬКА РОБОТА

АВТОМАТИЗАЦІЯ ПРОЦЕСІВ НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ ТА НЕПЕРЕРВНОГО РОЗГОРТАННЯ ВЕБ-ЗАСТОСУНКУ

Виконала: студентка групи ПМіМ-22

Спеціальності 122 - комп'ютерні науки

Кузьменко Д.О.

Науковий керівник: доц. Козій І. Я.

Рецензент: доц. Огородник Н.П.

ДЕКАН
факультету прикладної
математики та інформатики
ЛНУ ім. Івана Франка

Львів 2022

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет *прикладної математики та інформатики*

Кафедра *інформаційних систем*

Освітньо-кваліфікаційний рівень *магістр*

Напрям підготовки

Спеціальність *122 – комп'ютерні науки*

«ЗАТВЕРДЖУЮ»

Зав. кафедрою *проф.Шинкаренко Г.А.*

« 5 » *вересня* 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

Кузьменко Діани Олександрівни

(прізвище, ім'я, по батькові)

1. Тема роботи

Автоматизація процесів неперервної інтеграції та неперервного розгортання веб-застосунку

керівник роботи *Козій Ірина Ярославівна, доцент*

затверджені Вченою радою факультету від « 13 » вересня 2022 р., № 15

2. Строк подання студентом роботи *12 грудня 2022 року*

3. Вихідні дані до роботи

Курси на платформі Udemy по хмарному провайдеру AWS, статті по сучасних проблемах для розробників, книжки по DevOps та CI/CD процесам, мова програмування Bash, python, утиліта командного рядка Terraform, документації до використаних інструментів

4. Зміст магістерської роботи(перелік питань, які потрібно розробити)

1. Огляд сучасного стану проблеми
2. Аналіз інструментів для побудови неперервної інтеграції та розгортання
3. Побудова автоматизованого процесу за допомогою сучасних інструментів
4. Розробка скриптів та налаштувань для автоматизованого процесу
5. Перевірка коректної роботи автоматизованого процесу
6. Робота над стійкістю та безпекою існуючої інфраструктури

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Презентація магістерської роботи

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *01 вересня 2022 року*

КАЛЕНДАРНИЙ ПЛАН

№	Найменування етапів дипломної (кваліфікаційної) роботи	Строк виконання етапів роботи	Примітка
1.	*Огляд актуальних проблем для розробників*	*10.09.2022*	
2.	*Постановка задачі магістерської роботи*	*18.09.2022*	
3.	*Вибір проекту для побудови CI/CD процесу*	*06.10.2022*	
4.	Аналіз та вибір інструментів для реалізації кроків роботи	*12.10.2022*	
5.	*Розробка інфраструктури для доставки проекту*	*06.11.2022*	
6.	*Розробка інфраструктури для високої доступності сайту*	*28.11.2022*	
7.	Кінцеве тестування існуючого процесу	03.12.2022	
8.	Підготовка презентації та доповіді	11.12.2022	

Студент _____
підпис

Керівник роботи _____
підпис

ЗМІСТ

1. ВСТУП……..3

2. ЗАГАЛЬНІ ВІДОМОСТІ……6

3. СУТЬ МАГІСТЕРСЬКОЇ РОБОТИ……9

3.1. Опис роботи……...9

3.2. Створення плану інфраструктури високої доступності……13

3.3. Реалізація за допомогою інструментів автоматизації……..…..15

4. ЗАСТОСУВАННЯ НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ ТА НЕПЕРЕРВНОГО РОЗГОРТАННЯ…….……..…….15

5. ІМПЛЕМЕНТАЦІЯ……..…….……...16

6. ВИСНОВОК……...…….…….....16

7. СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…….…….....17

ВСТУП

Тема та ідея цієї роботи виникла на основі сучасних тенденцій та актуальних проблем розгортання проєктів (а саме веб-застосунків) . Досить часто можна спостерігати, що в процесі доставки проєктів до кінцевого користувача виникають неочікувані недоробки, дефекти, або ж час доставки є досить довгим. Також одна з великих проблем в розробці програмного забезпечення це інтеграція коду від різних розробників, тобто процес, коли люди починають об'єднувати код із різних гілок в одну – власне ту гілку, яка й доставляється та розгортається на сервері. Це не є проблемою для інженерів, які працюють самостійно, але це може пригальмувати процес розробки, коли багато людей працюють над спільною кодовою базою, адже тоді при інтеграціях часто трапляються конфлікти чи помилки.

Власне для вирішення таких проблем розробників виникло таке поняття як DevOps методологія , а також DevOps інженер – це людина, яка є спеціалістом у даній методології , орієнтується у всіх циклах розробки , тестування та експлуатації продукту. На цьому етапі варто надати характеристику декільком термінам, які будуть використовуватись в даній роботі:

DevOps (development operations) – це є комбінація практик та інструментів з розробки проекту (Dev) а також використання програмного забезпечення (Ops). Основною ціллю такої методології є зменшення життєвого циклу розробки продукту та забезпечення безперервної доставки та розгортання програмного забезпечення високої якості за допомогою гнучких методів розробки. Можна сказати що DevOps інженер це переправа між розробниками і користувачами продукту, який забирає на себе завдання, щоб розвантажити розробників і гарантувати користувачеві безперебійно користуватись

програмним забезпеченням.

CI/CD – це комбінація неперервної інтеграції та неперервного розгортання (Continuous integration and continuous deployment) програмного забезпечення в процесі розробки. Також варто зазначити, що CD може використовуватись як continuous delivery – безперервна доставка продукту [9,10].

Переважно якраз DevOps інженери використовують CI/CD технологію для того, щоб автоматизувати проміжні цикли розробки для зручного користування зацікавленими сторонами (інженерами якості, розробниками, клієнтами і т.д). Якщо сказати іншими словами, то після того, як розробник пише код проекту, то потрібно цей код протестувати, побудувати, реалізувати на кінцевий веб-сервер. В цій роботі було використано методику неперервної інтеграції та неперервного розгортання для вирішення вищезгаданих проблем.

Мета цієї роботи полягала в тому, щоб створити працюючу інфраструктуру високої доступності з використанням інструментів автоматизації для неперервної інтеграції та неперервної поставки для веб-застосунку.

Так як метою роботи не є написання коду та розробка самого проекту, а це більше про побудову інфраструктури, тобто те, що є під гарною оболонкою сайту, то код проекту я використала з попередніх років навчання в університеті. За основу завдання я використала програму на мові програмування PHP – це є веб-сайт для бронювання готелів в подорожах. Саме для цього проекту в даній роботі потрібно було дослідити як саме автоматизувати побудову PHP проекту, які є найбільш поширені інструменти для тестування та доставки побудованого пакету на сервер , яка їхня вартість , переваги та недоліки . Також в основі цієї роботи було задумано використати хмарні технології

(я обрала платформу AWS), обрати цих імплементувати автоматизацію процесів неперервної інтеграції та неперервного розгортання , продемонструвати що при зміні коду в репозиторії проект збереться заново автоматично і кінцевий користувач побачить зміни при оновлені сторінки.

2. ЗАГАЛЬНІ ВІДОМОСТІ

Термін “Continuous integration(неперервна інтеграція)” вперше був озвучений Граді Бучем ще в 1991, та з часом його визначення змінилось під сучасні проблеми. [1]

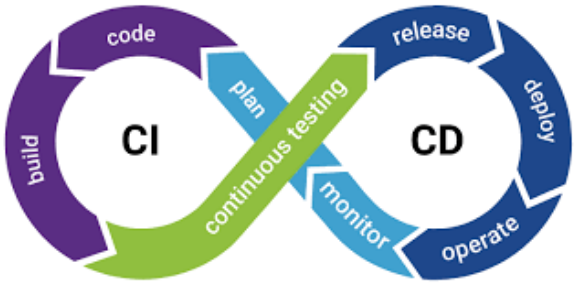


Рис. 1. Ознайомлення з CI/CD методологією

Ідея CI в тому, щоб зменшити кількість проблем під час нових змін в коді програмі, при кожній новій ітерації або ж версії коду. Цього можна досягти, якщо інтеграції будуть проводитися часто, тобто кожен розробник буде додавати свої зміни до спільного репозиторію як мінімум раз на день, але чим частіше - тим краще. За допомогою цієї практики загальний відсоток різниці в коді за одну інтеграцію буде незначним, а при виникненні конфлікту його буде простіше розв’язати. Патрік Колдвелл у книзі “"Code Leader: Using People, Tools, and Processes to Build Successful Software” описує процес Continuous integration як “інтегрувати зміни якомога раніше і частіше”[2].

Є набір практик та правил, яких слід дотримуватися при імплементації CI, оглянемо основні з них:

- Використовувати систему контролю версій
- Побудови мають бути автоматизовані
- Функціонал має бути покритий тестами
- Commits (додавання змін) до спільного репозиторію робляться мінімум раз на день
- При кожній зміні в репозиторії автоматично створюється побудова проєкту та запускається набір тестів

Важлива ідея в тому, що процес CI має бути автоматизованим.

Якщо побудова не може бути успішно виконаною, або тести не проходять, то розробникам має прийти про це сповіщення, і виправлення помилки має найвищий пріоритет.

Зазвичай процес CI кроки можна описати наступною послідовністю:

1. Розробник вносить зміни до коду репозиторію
2. CI-сервер бачить ці зміни (по тригеру, або сам періодично робить перевірку)
3. CI-сервер робить побудову проекту, запускає тести, записує всі результати.

Якщо сталася помилка - згідно з налаштуваннями відправляється повідомлення розробникам на пошту, в повідомлення , або в додаток Teams, Slack, Skype і тому подібне.

Так як сам процес CI/CD все більше набуває популярності, то важливо розуміти які ж його переваги в порівнянні зі звичайним процесом розгортання проекту, що стає причиною такої популярності :

- Зменшення ризиків , адже є перевірка коду і непрацюючий код не може потрапити у виробництво
- Завдяки частим інтеграціям, які робляться невеликими частинами, простіше знайти місце в коді, яке спричинило помилку
- Автоматизація рутинних процесів - таких як побудова, тестування та розгортання програмного забезпечення допомагає зберегти час і зменшує ризик людського фактору
- Зазвичай процес CI змушує розробників писати код, який є більш модульним та менш комплексним

Але тут варто зауважити, що CI також потребує певні вимоги,

необхідні для його налаштування:

- Тести мають бути написані для всього нового функціоналу та для знаходження помилок
- Треба налаштувати CI сервер, який буде відповідальним за побудову та запуск тестів для проекту при появі нових змін
- Розробникам треба частіше об’єднувати свої зміни з основним репозиторієм, принаймні раз на день, що не завжди може вдаватися
- На налаштування CI треба додаткові кошти - платити за обчислювальні ресурси, можливо за ліцензію для CI программного забезпечення, яка може дорого коштувати для великих компаній.

На Рис.1 можемо побачити гарну візуалізацію того, що процес CI є постійним. Далі буде детальніше описано кожен з цих етапів.

В цій роботі я буду користуватись хмарним середовищем AWS, тому що воно надає велику кількість безкоштовних ресурсів, а також за оплата за ресурси відбувається по мірі їх використання (“Pay as you go”) .

3. СУТЬ МАГІСТЕРСЬКОЇ РОБОТИ

3.1 ОПИС РОБОТИ

Опис теорії

Неперервна інтеграція (далі CI) складається з таких кроків як планування , розробка, побудова проекту (Plan, Code, Build) . [Рис. 2] :

- Планування (Plan) . На цьому етапі визначається який функціонал має бути доданий, це детально описується в завданні (або декількох завданнях) і до кожного завдання прикріплюється відповідальний розробник.
- Розробка (Code) . Інженер займається імплементацією завдання, тобто пише код який буде покривати вимоги, які поставлені в завданні та додає цей код в репозиторій .
- Збірка проекту (Build) – отримання інформаційного продукту з початкового коду. Цей крок переважно виконується автоматично після додавання нових змін до репозиторію

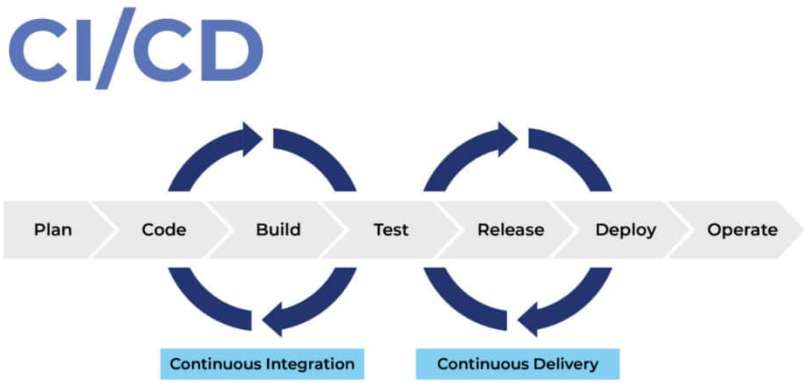


Рис. 2. Опис кроків CI/CD процесу

Далі на етапі тестування (коли за допомогою додаткових інструментів автоматично тестується якість коду) процес переходить в наступну фазу :

Неперервна доставка (далі CD - continuous delivery) , яка

складається з :

- Тестування (test) - як вже було згадано, на цьому етапі код автоматично тестується за допомогою обраних інструментів
- Демонстрування (release) – код з другорядних гілок поєднується з основною гілкою репозиторію
- Розгортання (deploy) – процес перенесення проекту на сервер, налаштування всіх залежностей
- та власне роботи проекту (Operate) : проект вже працює і ним можуть користуватись кінцеві користувачі

Основною задачею цієї роботи було створити CI/CD процес та високо доступну інфраструктуру для веб-застосунку.

Перед початком роботи я ознайомилась з інструментами, які можуть використовуватись для кожного з кроків CI/CD процесу. Першим кроком потрібно було обрати інструмент, який надалі використовувався для збереження і контролю версій коду . Було розглянуто наступні варіанти: GitHub, GitLab, BitBucket, AWS CodeCommit repository. Спочатку я хотіла використовувати AWS CodeCommit, адже було цікаво мати весь CI/CD на AWS, але почавши розбиратись з плагінами та додатковими інструментами я зіштовхнулась з низкою проблем (приєднання SonarQube, інтеграція з Jenkins) , вирішення таких проблем виглядало заплутаним, та не завжди інтеграція спрацьовувала тау, як це очікувалось. Тому я проаналізувала детальніше всі вищезгадані інструменти та мною було обрано іншу систему для контролю версій коду, а саме GitHub. Він зручний у використанні і має велику кількість можливих інтеграції з

інструментами для збірки проекту , також GitHub є одним з найпопулярніших рішень , відповідно і більшість проблем, з якими я могла зустрітись – вже були описані в документації.

GitHub допомагає розробникам орієнтуватись в коді, розуміти що відбувається і хто та які зміни вносив. Він зручний для роботи багатьох команд одночасно, для перегляду і підтвердження змін. Також плагін цього інструмента є у Visual Studio Code , що робить роботу значно комфортнішою .

Наступним та основним інструментом для збірки та CI/CD було обрано Jenkins, тому що це є безкоштовний інструмент і він чудово справляється в ролі сервера для автоматизації неперервної доставки.

Для зручності передачі і доступу до додатку було знайдено рішення, що додаток можна покласти в Docker контейнер – образ з запущеним кодом веб-застосунку.
Також було використано хмарні сервіси (AWS) для розташування веб-серверу (EC2 instance).
В результаті загальний список інструментів для цієї роботи отримав наступний вигляд:

- Docker - для зручного переміщення та використання образу з запущеним веб-застосунком
- Jenkins - для збірки проекту а також використовується як сервер автоматизації для моніторингу безперервної інтеграції та доставки.
- AWS - це хмарна платформа, буде використовуватись для того, щоб зберігати веб-застосунок на віддаленому сервері, замість того, щоб це робити локально на жорсткому диску.
- Ansible - програмне рішення для віддаленого управління конфігураціями. Воно дозволяє

налаштовувати віддалені машини.

- MySQL- база даних для роботи веб-застосунку, там зберігаються дані зареєстрованих користувачів для авторизації на веб-сайт
- PHP-мова програмування веб-застосунку на якому робиться приклад неперервної інтеграції та доставки
- NGINX- веб-сервер, на якому зберігається наша веб-аплікація
- Git - розподілена система керування версіями файлів та роботи в команді. У випадку моєї роботи використовуються для зберігання коду.

Виходить, що CI/CD процес матиме наступний вигляд (Рис. 3):

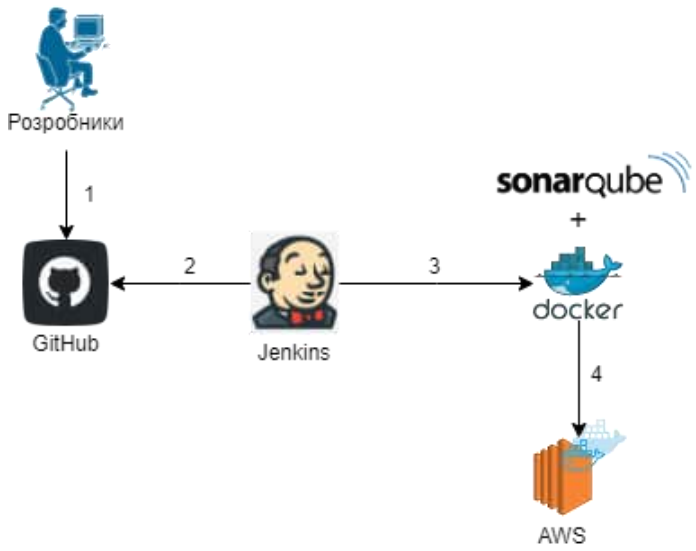


Рис. 3. CI/CD процес

Крок 1. Код веб-застосунку початково знаходиться на системі контролю версій GitHub. При змінах в коді відбувається наступний процес :

Крок 2. Збірка образу в Jenkins та публікація в DockerHub

1. Проект з Jenkins відправляється автоматично на Maven

2. В результаті отримуємо готовий побудований проект в Maven

3. Maven файл відправляється на NGINX веб-сервер, який знаходиться в Docker контейнері

Крок 3. Jenkins розгортає застосунок в Docker-контейнері.

Крок 4. Віддалений сервер працює з запущеним веб-застосунком

3.2 Створення плану інфраструктури високої доступності

Коли кінцевий користувач робить якісь операції на сайті , то він не бачить процесу який відбується під гарною оболонкою, тому й CI/CD для користувача може виглядати як просто зміна вигляду сторінки, але під цією веб-сторінкою знаходиться сервер, а точніше декілька серверів. Інфраструктура високої доступності (high-availability) дозволяє веб-застосунку бути більш стійким для зовнішніх навантажень . Гарним прикладом тут може бути сайт УКРПОШТИ (https://www.ukrposhta.ua/ua/vlasna-marka) , коли на нього вистачили у продаж марки і відбувся ажіотаж – сайт перестав працювати, тому що не витримав навантаження. Переважно високо доступну інфраструктуру варто робити для сайтів з продажами, або де може передбачатись висока активність.

На рис.4 можемо подивитись на імплементовану систему для веб-застосунку який використовується в цій роботі. При великому навантаженні один з серверів може відмовитись працювати.

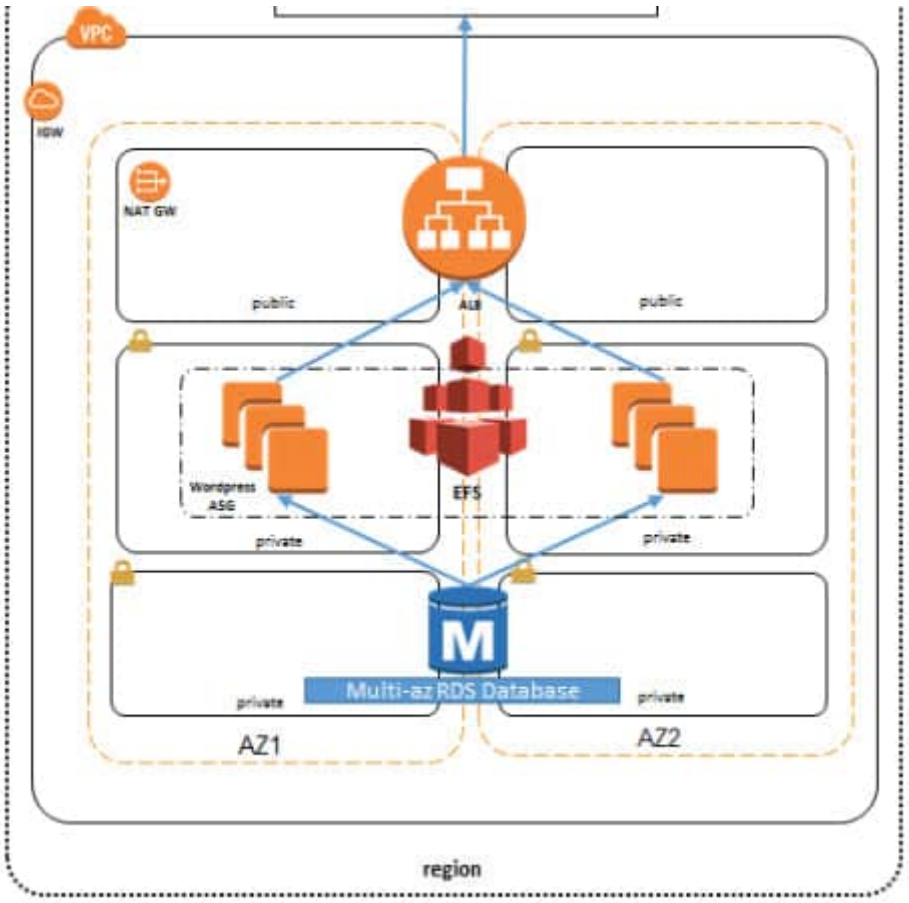


Рис. 4. Інфраструктура для налаштування веб-застосунку

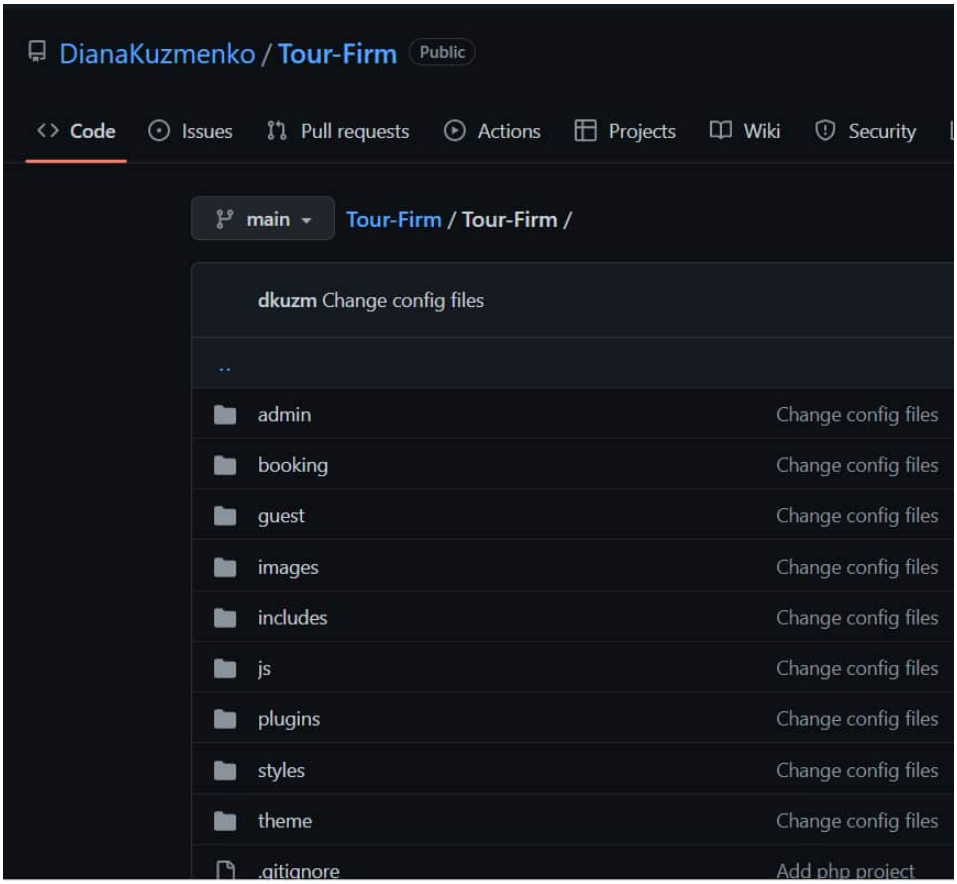
У випадку відмови одного з них – веб-сторінка далі працюватиме, адже налаштоване авто-підняння нового серверу. Також є балансер, який балансує навантаження між двома серверами пришвидшуючи процес виконання запитів клієнтом.

4. ЗАСТОСУВАННЯ НЕПЕРЕРВНОЇ ІНТЕГРАЦІЇ ТА НЕПЕРЕРВНОГО РОЗГОРТАННЯ

CI / CD або CICD, як правило, відноситься до поєднаних практик безперервної інтеграції та безперервної доставки, або безперервного розгортання. CI / CD усуває розриви між розробкою та експлуатацією та командами, забезпечуючи автоматизацію побудови, тестування та розгортання програм. Ця методологія все частіше використовується при доставці проектів до клієнта завдяки тому що легше виявляти дефекти, підвищити працездатність та забезпечити більш швидкі цикли запуску. Цей процес відрізняється від традиційних методів, коли набір оновлень програмного забезпечення інтегрувався в один великий пакет перед розгортанням новішої версії. Тож на сьогоднішній день CI/CD методологія є досить важливою у процесі доставки проекту до кінцевого користувача, якої намагаються дотримуватись багато інженерів.

5. ІМПЛЕМЕНТАЦІЯ

1. Створюємо проект на GitHub, на який завантажуємо веб-застосунок на PHP



2. Створюємо новий сервер на AWS :

Так як сенс роботи це автоматизувати повністю процес розгортання, то і створення серверів автоматизуємо за допомогою Terraform . Напишемо наступні модулі для створення серверу на AWS:

Спочатку налаштовуємо доступ до AWS акаунту. Визначаємо в Terraform тип провайдера – AWS, передаємо відкритий і секретний ключ, який дозволяє доступитись саме до мого AWS акаунта і проводити там подальше створення інстраструктури.

```
provider "aws" {
region     = "${var.region}"    #AWS region
```

```
access_key = "${var.ACK}"      #Access key
secret_key = "${var.SCK}"      #Secret key
}
```

Далі визначаємо налаштування для мережі, та діапазон публічних IP-адрес, які можуть бути присвоєні для серверу :

```
resource "aws_subnet" "pubSB" {    //налаштування публічних IP-адрес для доступу до веб-сайту
  vpc_id                = "${var.vpc-id}"
  cidr_block            = "10.130.9.0/24"
  availability_zone     = "${var.AZ1}"
  map_public_ip_on_launch = true

  tags = {
    "${var.tag-Name-name}"  = "${var.tag-Name-value}-pubSB"
    "${var.tag-Owner-name}" = "${var.tag-Owner-value}"
  }
}
```

Також прописуємо налаштування для приватної мережі. В цій мережі буде розташовуватись база даних (згідно з Рис.4), та веб-сервери. Налаштування приватної мережі потрібно для інтернет-безпеки даної моделі веб-застосунку, адже якщо всі IP-адреси будуть доступними звідусіль, то значно зростає ризик DDoS та інших атак.

```
resource "aws_subnet" "pvAZ1" {
  vpc_id                = "${var.vpc-id}"
  cidr_block            = "10.130.10.0/24"
  availability_zone     = "${var.AZ1}"
```

```
 map_public_ip_on_launch = false

 tags = {
  "${var.tag-Name-name}"  = "${var.tag-Name-value}-private-Z1"
  "${var.tag-Owner-name}" = "${var.tag-Owner-value}"
 }
```

Окремо опишемо ресурс таблиці маршрутизації для приватних та публічних доступів :
Публічний доступ буде відбуватись через інтернет-шлюз:

```
resource "aws_route_table" "RTpub" {
 vpc_id = "${var.vpc-id}"

 route {
  cidr_block = "0.0.0.0/0"    //доступ з будь-яких IP-адрес
  gateway_id = "${var.igw-id}" //ідентифікаційний номер шлюза
 }

 tags = {
  "${var.tag-Name-name}"  = "${var.tag-Name-value}-RT-pub"
  "${var.tag-Owner-name}" = "${var.tag-Owner-value}"
 }
```

Для того, щоб проконтролювати групи безпеки (security groups) для сервера (ec2_instance) , то я також описала які порти мають бути відкриті та для кого:

```
resource "aws_security_group" "sgec2" {
 name        = "sg_ec2t1"
```

```
  description = "Security Group for ec2"
  vpc_id      = "${var.vpc-id}"

  ingress {
    from_port   = 80  //http порт
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 443 //http порт
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 3306 // порт для спілкування з базою даних
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 22 //ssh порт, для доступу до серверу через ssh
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

Такий саме блок коду прописую для балансера навантаження, адже він має бути доступний з інтернету, а самі сервери недоступні. Коли ми заходимо

на сайт то ми переходимо безпосередньо на балансер навантаження а далі він нас перенаправляє на менш завантажений сервер .

```
resource "aws_security_group" "sgELB" {
  name        = "sg_elb"
  description = "Security Group for ELB"
  vpc_id      = "${var.vpc-id}"

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

Так як ми вже налаштували групи безпеки (security group) для серверів і дали їм знати через який порт потрібно обмінюватись інформацією з базою даних, то потрібно таке саме налаштування дати для бази даних :

```
resource "aws_security_group" "sgDB" {
  name        = "sg_rds"
  description = "Security Group for DB"
  vpc_id      = "${var.vpc-id}"

  ingress {
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

Для того, щоб сервери були ідентичними в середині, налаштовуємо для них єдину файлову систему. Тобто, з обидвох серверів можна буде на неї знайти і при зміні в файловій системі це можна буде побачити з обидвох серверів.

```
resource "aws_efs_file_system" "file_sys" {
  creation_token = "Tour-Firm-TF"

  tags = {
    "${var.tag-Name-name}"  = "${var.tag-Name-value}-EFS"
    "${var.tag-Owner-name}" = "${var.tag-Owner-value}"
  }
}
```

```
resource "aws_efs_mount_target" "efs-mt" {
  file_system_id  = "${aws_efs_file_system. file_sys.id}"
  subnet_id       = "${aws_subnet.pvZ1.id}"
  security_groups = ["${aws_security_group.sgEFS.id}"]
}
```

Наразі для коректної роботи веб-сайту буде достатньо мінімальних налаштувань бази даних. Я обрала базу даних MySQL, адже вона досить легка в конфігураціях та користуванні. Тип машини для бази даних t2.micro – це не найменшим, але є безкоштовним типом бази даних. Він дозволяє розміщувати інформацію до 1Гб , та використовувати 1 ядро. Так як в цій базі даних буде розміщена інформація тільки про логіни, паролі користувачів та бронювання готелей то наразі таких ресурсів цілком достатньо.

```
resource "aws_db_instance" "dbinst" {
  allocated_storage   = 20
  storage_type        = "gp2"
  engine              = "mysql"
  engine_version      = "5.7.19"
  instance_class      = "db.t2.micro"
  identifier          = "${var.tag-Name-value}-db-test"
  skip_final_snapshot = true
  name                = "wp_myblog"
  username            = "${var.dbuser}"
  password            = "${var.dbpass}"
  port                = "3306"

  db_subnet_group_name    = "${aws_db_subnet_group.dbsbg.name}"
  vpc_security_group_ids  = ["${aws_security_group.sgDB.id}"]
```

```
  maintenance_window = "Mon:00:00-Mon:03:00"
  backup_window      = "03:00-06:00"  // В цій базі даних робиться резервна копія кожного дня вночі. В такому випадку її легко можна буде відновити, якщо раптом вона зламається.

  multi_az               = true
  backup_retention_period = 0

  tags = {
    Owner   = "${var.tag-Owner-value}"
    Name    = "${var.tag-Name-value}"
```

Наступним кроком налаштовую групи авто-підняття нового сервера (auto-scaling groups) , щоб у випадку відмови одного з серверів це не повпливало на роботу сайті і кінцевий користувач все-одно міг би зайти на веб-сайт.

```
resource "aws_autoscaling_group" "name-asg" {
  name                  = "${var.tag-Name-value}-asg"
  max_size                 = 2 //прописую максильмальну кількість серверів яка мала би працювати одночасно
  min_size               = 0
  health_check_grace_period = 300
  default_cooldown         = 60
  health_check_type        = "EC2"
  desired_capacity         = 1
  force_delete             = true
  launch_configuration     = "${aws_launch_configuration.launch.name}"
  vpc_zone_identifier      = ["${aws_subnet.pvZ1.id}", "${aws_subnet.pvZ2.id}"]
  target_group_arns        = ["${aws_lb_target_group.name-tg.arn}"]
```

```
  timeouts {
    delete = "15m"
  }
```

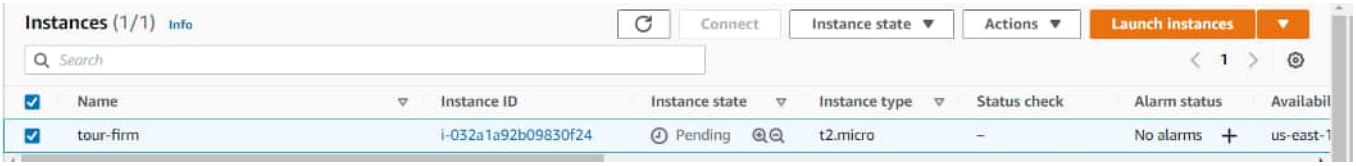
Коли інфраструктура буде створена і коректно працювати, вона буде самостійно створювати або видаляти старі сервери за потреби. Але при цьому, було би добре щоб ми знали коли це стається. Адже можливо таке, що серверу буде недостатньо ресурсів, він буде кожного дня перестворюватись. В такому випадку потрібно буде обрати інший тип машини. Щоб знати коли є велика завантаженість серверу я створила також метрики, які будуть сповіщати мене, якщо використовується критична кількість ресурсів серверу.

```
resource "aws_cloudwatch_metric_alarm" "CPU-high" {
  alarm_name          = "terraform-test-tour-firm"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "120"
  statistic           = "Average"
  threshold           = "70"
  alarm_description   = "This metric monitors ec2 cpu utilization"
  alarm_actions       = ["${aws_autoscaling_policy.agents-scale-up.arn}"]
  dimensions          = {
    AutoScalingGroupName = "${aws_autoscaling_group.name-asg.name}"
  }
}
```

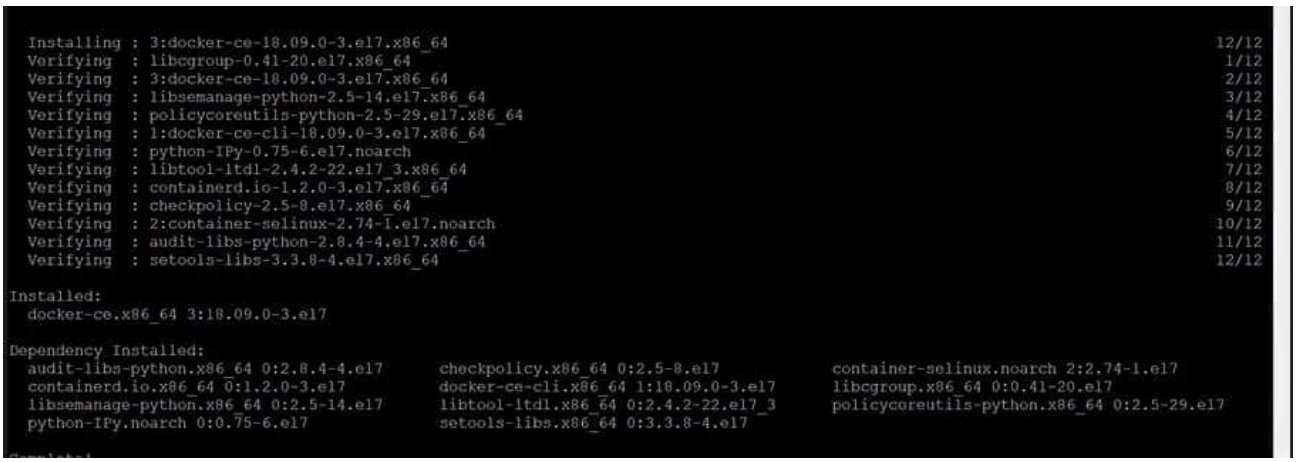
В даному випадку якщо використання CPU буде більше або дорівнювати 70

відсоткам, то мені прийде сповіщення на пошту з описом серверу, метрики і використання CPU даний момент часу. Ця метрика буде перевірятись кожні 2 хвилини (120 секунд) і надсилатись мені тільки якщо впродовж двох таких періодів середнє використання CPU буде підходити під описане налаштування оповіщення (alert).

Після того як було описано інфраструктуру за допомогою тераформу, то був запущений код, який створив наступний сервер:



3. Встановлюємо на створений сервер Docker як сервіс



4. Редагуємо конфігураційний файл Docker ,відповідно до вимог Jenkins. Щооб на докер-контейнері був такий користувач як Jenkins, які має обов’язкові права для виконання скриптів

```
ENV JENKINS_PASS redhat

# skip initial setup
ENV JAVA_OPTS -Djenkins.install.runSetupWizard=false

# installing required plugins
COPY plugins.txt /usr/share/jenkins/plugins.txt
RUN /usr/local/bin/install-plugins.sh < /usr/share/jenkins/plugins.txt


USER root

# install docker
RUN yum install -y sudo  yum-utils device-mapper-persistent-data lvm2
RUN yum-config-manager \
    --add-repo \
    https://download.docker.com/linux/centos/docker-ce.repo
RUN yum install -y  docker-ce-cli  --nobest
# RUN systemctl start docker

# RUN groupadd docker
RUN usermod -aG docker jenkins

# give jenkins user sudoer permission
RUN echo -e "jenkins  ALL=(ALL)  NOPASSWD: ALL" >> /etc/sudoers

# give jenkins user sudoer permission
RUN echo -e "jenkins  ALL=(ALL)  NOPASSWD: ALL" >> /etc/sudoers.d/jenkins

RUN chmod u+wx /etc/sudoers
```

5. Налаштовуємо Jenkins сервер за допомогою наступних команд на сервері:

sudo yum install java-1.8.0-openjdk-devel – встановлюємо OpenJDK

curl --silent --location http://pkg.jenkins-ci.org/redhat-stable/jenkins.repo | sudo tee /etc/yum.repos.d/jenkins.repo – додаємо шлях для Jenkins репозиторія на сервері

sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key - додаємо Jenkins ключ

Перевіряємо чи Jenkins репозиторій коректно додався:

[root@centos ~]# cat /etc/yum.repos.d/jenkins.repo

[jenkins]

name=Jenkins-stable

baseurl=http://pkg.jenkins.io/redhat-stable

gpgcheck=1

Можемо побачити що такий файл існує, значить можемо встановлювати Jenkins:

sudo yum install jenkins

```
Installed size: 68 M

Downloading packages:

Running transaction check

Running transaction test

Transaction test succeeded

Running transaction

Installing : jenkins-2.277.1-1.1.noarch 1/1

Verifying : jenkins-2.277.1-1.1.noarch 1/1

Installed:

jenkins.noarch 0:2.277.1-1.1

Complete!
[root@centos yum.repos.d]#
```

З логів бачимо що сервіс успішно встановлений, тому ми його запускаємо :

sudo systemctl start jenkins

sudo systemctl enable jenkins

Перевіряємо роботу Jenksins сервера, для початку виведемо статус сервісу на консоль:

systemctl status jenkins

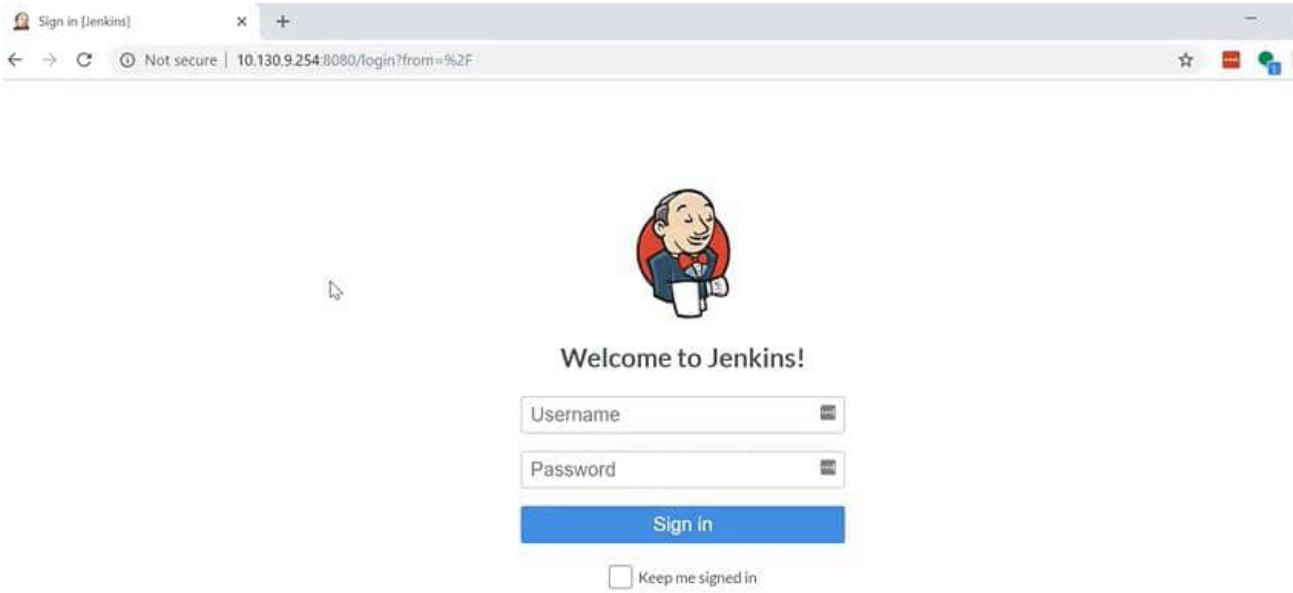
```
[root@centos ~]# systemctl status jenkins
● jenkins.service - LSB: Jenkins Automation Server
   Loaded: loaded (/etc/rc.d/init.d/jenkins; bad; vendor preset: disabled)
   Active: active (exited) since Sun 2021-03-14 11:22:13 EDT; 1min 2s ago
     Docs: man:systemd-sysv-generator(8)

Mar 14 11:22:11 centos systemd[1]: Starting LSB: Jenkins Automation Server...
Mar 14 11:22:11 centos runuser[3051]: pam_unix(runuser:session): session opened for u
Mar 14 11:22:13 centos jenkins[3045]: Starting Jenkins [  OK  ]
Mar 14 11:22:13 centos systemd[1]: Started LSB: Jenkins Automation Server.
[root@centos ~]#
```

Налаштовуємо Firewall таким чином, щоб дозоляти цьому серверу працювати по 8080 порту:

sudo firewall-cmd --permanent --zone=public --add-port=8080/tcp

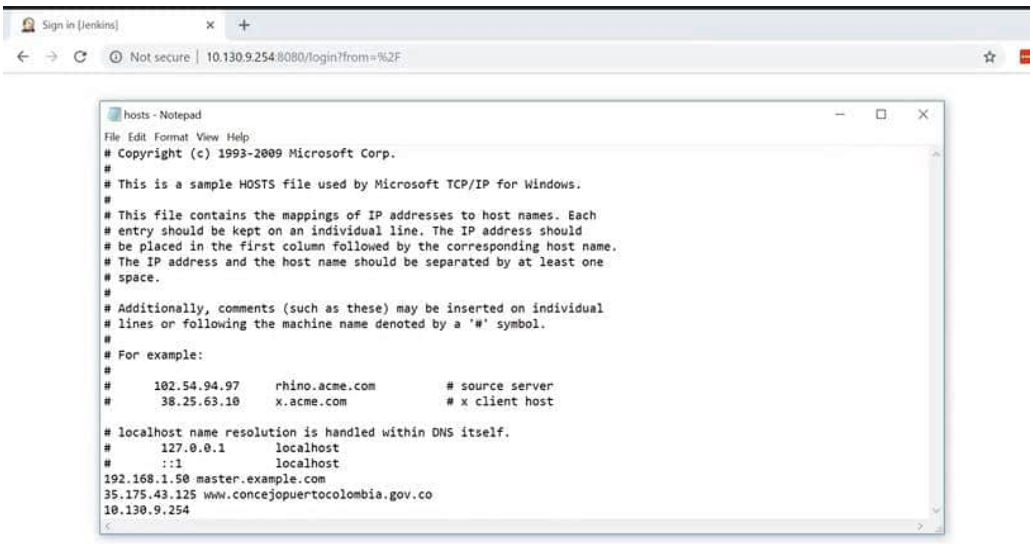
sudo firewall-cmd –reload

На цьому етапі всі необхідні кроки для налаштування Jenkins закінчені, тому можна спробувати зайти на ip-адресу сервера де цей сервіс розташований:

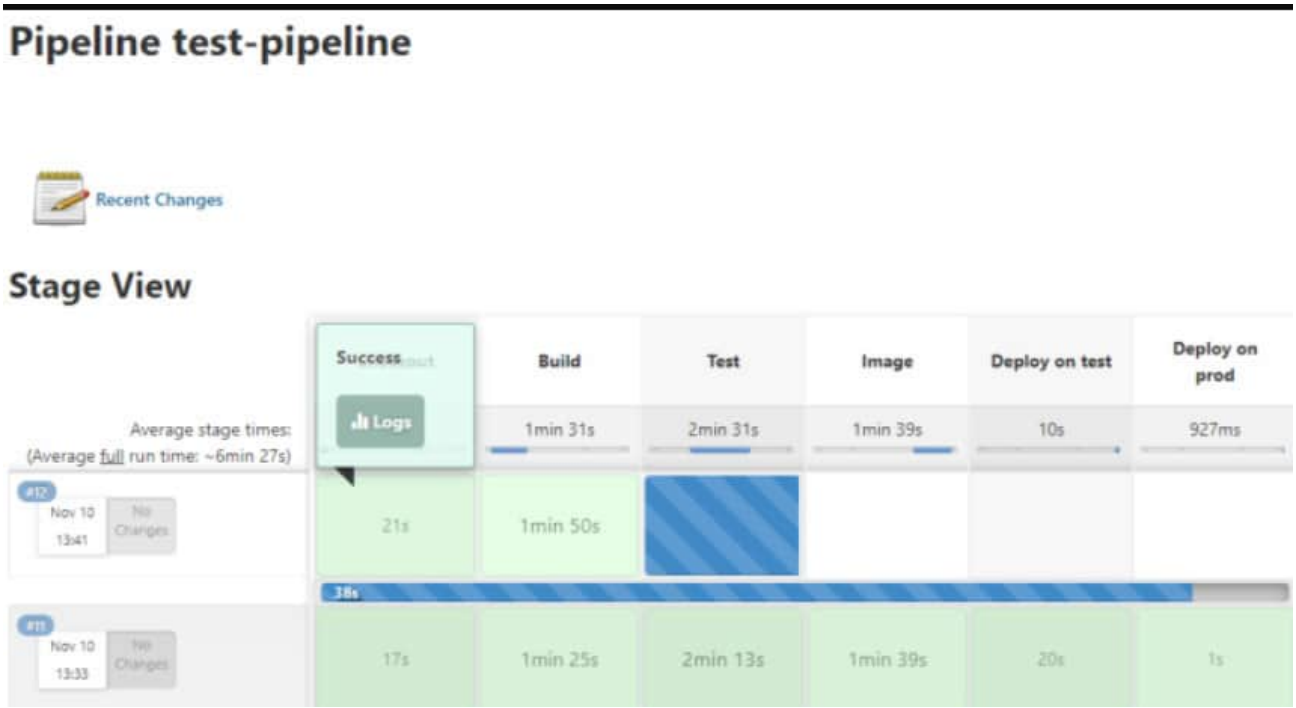


Можемо побачити що він успішно запустився. Дані для входу адміністратора прописані в конфігураційному файлі за замовчуванням і їх можна змінити.

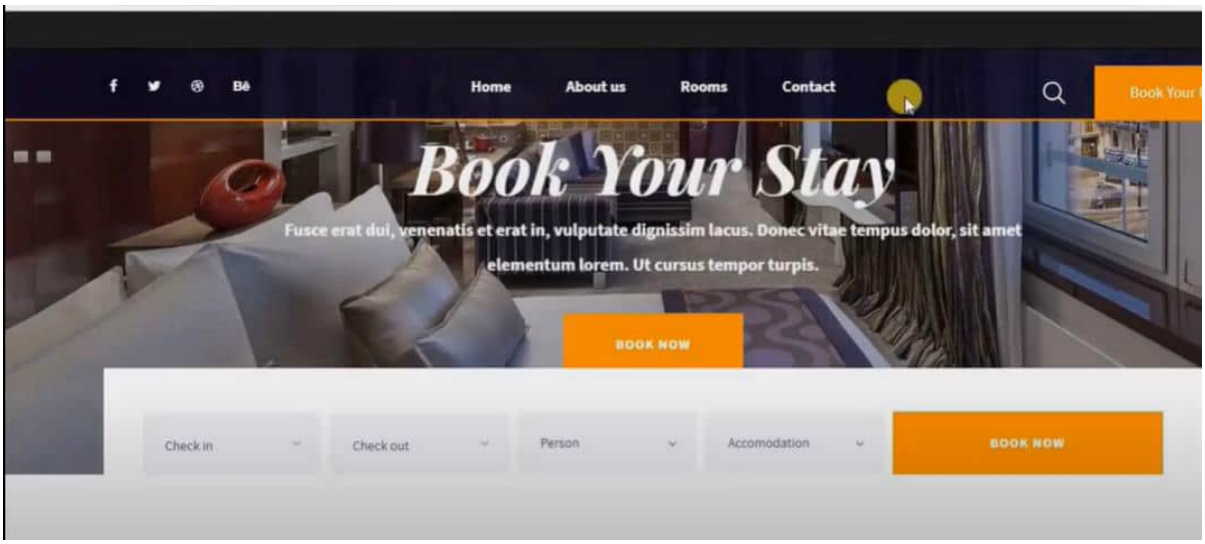
6. Змінимо IP-адресу на сервері на DNS ім’я для простішого доступу

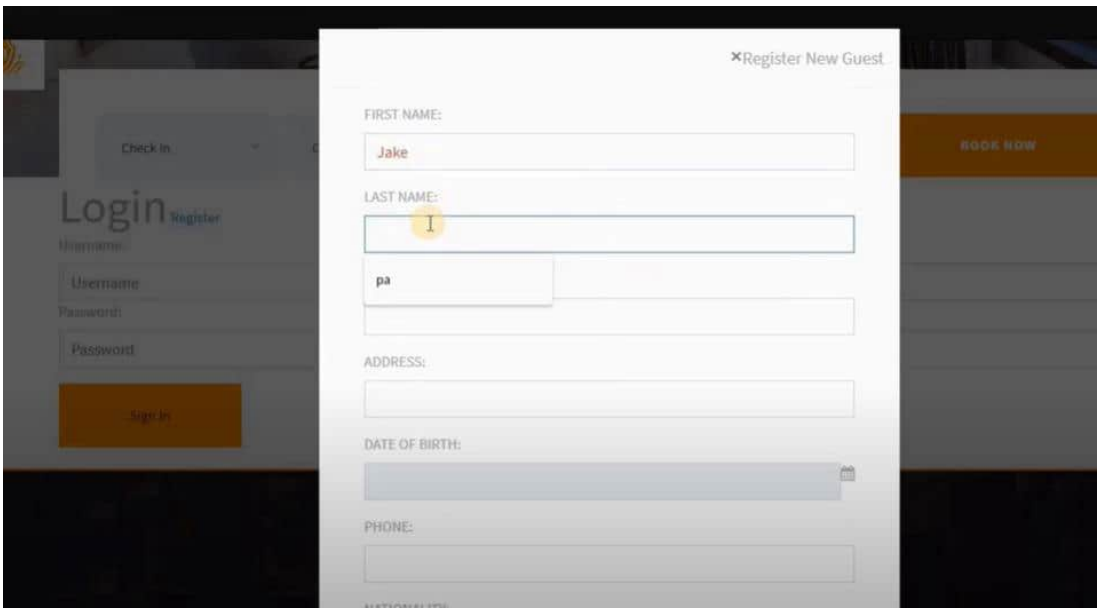


7. Створюємо відповідні скрипти (pipelines) на Jenkins , робимо зміну коду в GitHub та слідкуємо за тим чи виконався процес



8. В результаті переходимо за IP адресою нашого сервера та бачимо останні внесені зміни GitHub (в моєму випадку було додано текст “Book your stay”.





Для перевірки бази даних реєструємось на сайті, вводимо свої данні , та резервуємо житло. Наразі ці кроки перевірки також пройшли успішно.

ВИСНОВОК

Використання комбінації неперервної інтеграції та неперервної доставки є набором технік і передових практик для роботи над IT-проєктами. При використанні комбінації команда розробників може оперативно і частіше вносити зміни в код. В результаті цієї роботи було розроблено повний цикл CI/CD процесу для веб-застосунку на мові програмування PHP, також було створено інфраструктуру з декількома серверами та балансером навантаження. Для надійності та безпеки веб-застосунку інфраструктура проекту була зроблена більш високо доступною (High-availability) та менш вразливою до зовнішніх факторів (таких як DDos-атаки, відкриті порти) а також використовувалась зашифрована база даних для надійності даних користувачів. Поставлена задача досягнута та план виконано. Дане рішення може використовуватись також для державних сайтів, адже зараз часто на них можуть відбуватись атаки, або ж у зв’язку з новинами – функціонал сайтів може дещо змінюватись і моє рішення дозволить завжди мати працюючий веб-застосунок у кінцевого користувача.

ВИКОРИСТАНІ ДЖЕРЕЛА

1. Booch, Grady (1991). Object Oriented Design: With Applications. Benjamin Cummings. p. 209. ISBN 9780805300918. 18 Серпня 2014.

2. Патрік Колдвелл у книзі “"Code Leader: Using People, Tools, and Processes to Build Successful Software (Квітень 30, 2008)

3. Висока доступність. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/High_availability

4. SLA, SLO, SLI. [Електронний ресурс]. Режим доступу: https://www.atlassian.com/incident-management/kpis/sla-vs-slo-vs-sli

5. Аварійне відновлення. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Disaster_recovery

6. Капустян М. В. Оптимізація організації та побудови архітектури захищених корпоративних мереж / Автореферат дисертації на здобуття наукового ступеня доктора технічних наук, 2009 (ДСК).

7. Структура якісної архітектури. [Електронний ресурс]. Режим доступу: https://aws.amazon.com/well-architected

8. Інженерія надійності. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Reliability_engineering

9. Розгортання програмного забезпечення. [Електронний ресурс]. Режим доступу: https://en.wikipedia.org/wiki/Software_deployment

https://docs.microsoft.com/en-us/azure/architecture/patterns/compensating-transaction

10. Інфраструктура як код. [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Інфраструктура_як_код

11. Хмарний провайдер Amazon Web Services. [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/Amazon_Web_Services

12. Система керування вмістом WordPress. [Електронний ресурс]. Режим доступу: https://uk.wikipedia.org/wiki/WordPress

13. *El Khalyly, B.; Belangour, A.; Banane, M.; Erraissi, A. (2020).* "A new metamodel approach of CI/CD applied to Internet of Things Ecosystem". *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS): 1–6.* doi:10.1109/ICECOCS50124.2020.9314485.

14. *Sane, P. (2021).* "A Brief Survey of Current Software Engineering Practices in Continuous Integration and Automated Accessibility Testing". *2021 Sixth International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET): 130–134.* doi:10.1109/WiSPNET51692.2021.9419464.

15. https://docs.aws.amazon.com/