МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня бакалавр

зі спеціальності 122 – Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційне алгоритмічне та програмне забезпечення хмарної ІТ-інфраструктури»
здобувача групи ІНпн-91-0 Піменова Олега Олеговича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Олег ПІМЕНОВ
(підпис)

Керівник,
асистент кафедри комп'ютерних наук,
кандидат фізико-математичних наук Олександр ВЛАСЕНКО
(підпис)

Суми – 2023

Сумський державний університет
Центр заочної, дистанційної та вечірньої форм навчання
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 – Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІНпн-91-0 Піменова Олега Олеговича

1. Тема роботи: «Інформаційне алгоритмічне та програмне забезпечення хмарної ІТ-інфраструктури» затверджую наказом по СумДУ від *«01» червня 2023 р. № 0475-VI*
2. Термін здачі здобувачем кваліфікаційної роботи *до 09 червня 2023 року*
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для забезпечення хмарної ІТ-інфраструктури бізнес-підприємства. 3) Розробка забезпечення хмарної ІТ-інфраструктури бізнес-підприємства. 4) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 2023 р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*		
2	*Огляд технологій, що використовуються для забезпечення хмарної ІТ-інфраструктури бізнес-підприємства*		
3	*Розробка забезпечення хмарної ІТ-інфраструктури бізнес-підприємства.*		
4	*Аналіз отриманих результатів*		
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*		

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

АНОТАЦІЯ

Записка: 70 стр., 46 рис., 4 додатки, 21 використаних джерел.

Обґрунтування актуальності теми роботи – тема випуску та оновлення продукту ІТ компанії за методологією DevOps є дуже актуальною, оскільки задовольняє практичну потребу в ефективних та спрощених процесах розробки та розгортання програмного забезпечення. Ця робота безпосередньо спрямована на задоволення зростаючого попиту на швидкі та надійні випуски програмного забезпечення, ефективне використання хмарних, контейнерних технологій, тощо.

Об'єкт дослідження – розгортання продукту ІТ-компанії з використанням практик DevOps.

Мета роботи – автоматизувати робочий процес розгортання продукту, забезпечити безперервну інтеграцію та розгортання, підвищити масштабованість та, за рахунок цього, покращити співпрацю між командами розробників та операторів.

Методи дослідження – аналіз літератури, практична реалізація проєкту з використанням інструментів Git, GitHub, Jenkins, SonarQube, Linux, AWS та Docker, аналіз результатів впровадження та оцінка їх відповідності меті дослідження.

Результати – розроблений проєкт, оснований на практиці DevOps та інструментах, що забезпечують автоматизований процес розгортання, безперервну інтеграцію і розгортання, покращену масштабованість, покращену співпрацю між командами розробників та операторів.

DEVOPS, МЕТОДОЛОГІЯ, АВТОМАТИЗАЦІЯ, ІНФРАСТРУКТУРА, ІТ-ПРОДУКТ

ЗМІСТ

ВСТУП

Актуальність. Розгортання продукту IT-компанії з використанням практик DevOps дуже актуальна з кількох причин.

По-перше, зростає потреба в ефективних і спрощених процесах розробки та розгортання програмного забезпечення. У сучасному конкурентному цифровому просторі компанії повинні мати можливість випускати нові функції програмного забезпечення швидко та надійно. Практики DevOps можуть допомогти автоматизувати та оптимізувати процес розробки та розгортання програмного забезпечення, завдяки чому виведення нових функцій на ринок стане швидшим і легшим [1].

По-друге, практики DevOps можуть допомогти покращити співпрацю між командами розробки та операцій. Традиційно команди розробки та експлуатації працювали окремо. Це може призвести до затримок і помилок у процесі розгортання програмного забезпечення. Практики DevOps можуть допомогти зруйнувати ці розриви та покращити зв’язок і співпрацю між командами розробки та операцій.

По-третє, практики DevOps можуть допомогти підвищити масштабованість і надійність. Завдяки автоматизації процесу розгортання програмного забезпечення та використанню хмарних і контейнерних технологій практики DevOps можуть допомогти зробити IT-системи більш масштабованими та надійними. Це важливо для підприємств, яким потрібно мати можливість обробляти великі обсяги трафіку або яким потрібно мати можливість швидко масштабувати свою IT-інфраструктуру, щоб задовольнити мінливий попит [1].

Об’єкт дослідження. Об'єктом дослідження є методологія DevOps яка забезпечує ефективний процес розробки та розгортання програмного забезпечення в IT-компанії. Він включає в себе набір процедур, методологій,

інструментів та практик, які використовуються для створення, тестування, розгортання та підтримки програмного продукту. В рамках цієї роботи, особливий акцент робиться на практики DevOps та їх вплив на ці процеси..

Предмет дослідження. Предметом дослідження є впровадження та використання практик DevOps в процесах розробки та розгортання програмного забезпечення. Це включає в себе аналіз інструментів, таких як Git, GitHub, Jenkins, SonarQube, Linux, AWS, Docker, та їх використання для автоматизації робочих процесів, забезпечення безперервної інтеграції та розгортання, підвищення масштабованості та покращення співпраці між командами розробників та операторів. Також до предмету дослідження входить вивчення впливу практик DevOps на ефективність, швидкість і надійність процесів розробки та розгортання програмного забезпечення.

Структура. Дана робота складається зі вступу, аналітичного огляду, аналізу аналогічних методологій, постановки задачі, вибору методу розв’язання задачі, опису інформаційного та програмного забезпечення системи, опису програмної реалізації, висновків, списку використаних джерел та додатків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Сучасний стан методології DevOps

Сучасний стан методології DevOps характеризується широким впровадженням і постійним розвитком у відповідь на мінливі потреби команд розробників і операторів програмного забезпечення.

DevOps набула значної популярності та широкого впровадження в організаціях усіх розмірів та галузей. Компанії визнають переваги оптимізації процесу доставки програмного забезпечення, сприяння співпраці між командами та підвищення загальної ефективності. Підхід DevOps вже не обмежується лише першими прихильниками, а став загальноприйнятою практикою в індустрії розробки програмного забезпечення [2].

DevOps-методологія підтримується багатою екосистемою інструментів і технологій, які полегшують різні аспекти життєвого циклу розробки програмного забезпечення. Ці інструменти охоплюють такі сфери, як управління версіями (наприклад, Git), безперервна інтеграція/безперервне розгортання (CI/CD) (наприклад, Jenkins, GitLab CI/CD, CircleCI), управління конфігурацією (наприклад, Ansible, Chef, Puppet), інфраструктура як код (наприклад, Terraform, AWS CloudFormation), контейнеризація (наприклад, Docker, Kubernetes) та моніторинг (наприклад, Prometheus, Grafana) [3]. Ця широка екосистема інструментів надає організаціям гнучкість у виборі та інтеграції інструментів, які найкраще відповідають їхнім конкретним потребам.

DevOps заохочує підхід "зсуву вліво" в розробці програмного забезпечення, акцентуючи увагу на ранньому і безперервному тестуванні, співпраці та автоматизації. Тестування стало невід'ємною частиною процесу розробки, з підвищеною увагою до модульного тестування, інтеграційного тестування та

автоматизованого регресійного тестування. Підхід "зсуву вліво" гарантує, що проблеми виявляються і вирішуються на ранніх стадіях циклу розробки, зменшуючи ризик того, що дефекти потраплять на більш пізні етапи розгортання.

Поява хмарних обчислень значно вплинула на методологію DevOps. Хмарні платформи, такі як Amazon Web Services (AWS), Microsoft Azure і Google Cloud Platform (GCP), надають інфраструктуру і сервіси, необхідні для масштабованого і гнучкого розгортання додатків. DevOps використовує можливості хмарних технологій для надання ресурсів на вимогу, автоматизації управління інфраструктурою та сприяння швидкому масштабуванню для задоволення мінливого попиту. Інструменти інфраструктури як коду (IaC) дозволяють визначати конфігурації інфраструктури та керувати ними, забезпечуючи узгодженість, повторюваність та контроль версій.

Безпека привертає все більше уваги в методології DevOps, що призвело до появи DevSecOps. DevSecOps має на меті інтегрувати практики безпеки у весь життєвий цикл розробки програмного забезпечення, гарантуючи, що міркування безпеки враховуються на ранніх стадіях розробки. Тестування безпеки, сканування вразливостей та інструменти аналізу коду включені в конвеєр CI/CD, що дозволяє проактивно виявляти та зменшувати ризики безпеки [4].

DevOps – це не просто впровадження нових інструментів і процесів; він також підкреслює культурні зміни в організаціях. Співпраця, комунікація та спільна відповідальність є ключовими аспектами культури DevOps. Ізоляція між командами розробників, операційних та інших підрозділів руйнується, що сприяє міжфункціональній співпраці та обміну знаннями. Гнучкі методології та практики, такі як Scrum і Kanban, часто йдуть пліч-о-пліч з DevOps, сприяючи ітеративному розвитку, швидкому зворотному зв'язку та постійному вдосконаленню [4].

Методологія DevOps охоплює культуру безперервного навчання та вдосконалення. Організації використовують метрики, моніторинг та цикли зворотного зв'язку, щоб отримати уявлення про ефективність своїх практик DevOps. Цей підхід, заснований на даних, дозволяє виявляти вузькі місця, оптимізувати процеси та експериментувати з новими інструментами чи практиками.

Підсумовуючи, поточний стан методології DevOps характеризується широким впровадженням, різноманітною екосистемою інструментів, підходом "зсуву вліво", хмарною та інфраструктурною гнучкістю, інтеграцією практик безпеки, культурною трансформацією та фокусом на безперервному навчанні та вдосконаленні [2]. DevOps продовжує розвиватися, що зумовлено потребою у швидшій та надійнішій доставці програмного забезпечення та постійно мінливим технологічним прогресом.

1.2 Аналіз аналогічних методологій

Порівнюючи DevOps з іншими методологіями розробки програмного забезпечення, важливо відзначити, що DevOps не є заміною цих методологій, а скоріше додатковим підходом, який підвищує їх ефективність. Давайте обговоримо переваги методології DevOps по відношенню до деяких інших методологій:

1.2.1 Методологія Waterfall

Методологія Waterfall (Рисунок. 1.1) дотримується лінійного та послідовного підходу, де кожна фаза розробки (вимоги, проєктування, реалізація, верифікація, супроводження) завершується перед тим, як перейти до наступної. DevOps пропонує значну перевагу над Waterfall, сприяючи ітеративній розробці

та безперервному зворотному зв'язку. DevOps дозволяє швидше виявляти і вирішувати проблеми, знижує ризик дорогого перероблення і забезпечує більш оперативний підхід до зміни вимог [4].

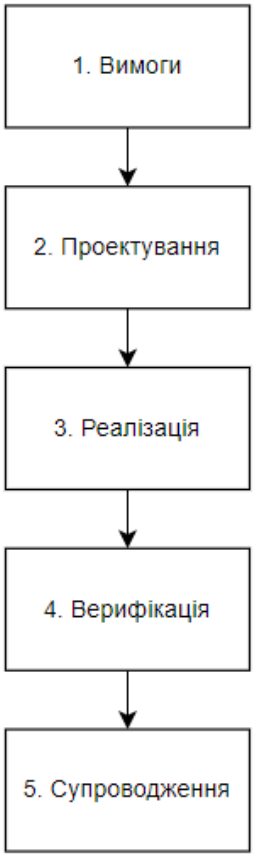

Рисунок 1.1 — Етапи фаз розробки програмного забезпечення за методологією Waterfall

Методологія Waterfall – це лінійний і послідовний підхід до розробки програмного забезпечення. Він складається з декількох етапів, які виконуються в певному порядку:

1) **Вимоги**. На цьому етапі команда проєкту збирає та документує вимоги до програмної системи. Це передбачає розуміння потреб та очікувань зацікавлених сторін, включаючи користувачів, клієнтів та власників бізнесу.

Вимоги детально визначаються, включаючи функціональні та нефункціональні аспекти, і, як правило, документуються в документі специфікації вимог [5].

2) **Проєктування**. Після того, як вимоги визначені, починається етап проєктування. Команда проєктувальників перетворює вимоги в детальний проєкт системи. Це включає архітектурний дизайн, дизайн бази даних, дизайн інтерфейсу користувача та інші відповідні компоненти дизайну. Результатом цього етапу є документ зі специфікацією проєкту, який слугує планом для етапу розробки [5].

3) **Реалізація**. На етапі реалізації команда розробників починає кодування програмного забезпечення на основі проєктних специфікацій. Це включає в себе написання, тестування та інтеграцію модулів коду. Для розробки програмної системи використовуються мови програмування, фреймворки та інструменти, визначені на етапі проєктування. Результатом цього етапу є виконуваний код, який представляє функціонуюче програмне забезпечення [5].

4) **Верифікація**. Після розробки коду починається етап верифікації. Цей етап спрямований на забезпечення того, щоб програмне забезпечення відповідало заданим вимогам і функціонувало правильно. Для виявлення та усунення будь-яких дефектів або проблем використовуються різні методи тестування, такі як модульне тестування, інтеграційне тестування, системне тестування та тестування прийнятності для користувача. Метою є перевірка програмного забезпечення на відповідність встановленим вимогам і забезпечення його якості та надійності [5].

5) **Обслуговування**. Після того, як програмне забезпечення було розроблено і перевірено, воно розгортається у виробничому середовищі на етапі супроводу. Цей етап передбачає постійну підтримку, виправлення помилок і вдосконалення на основі відгуків користувачів і мінливих вимог. Діяльність з супроводу може включати вирішення проблем, оновлення програмних

компонентів та включення нових функцій або вдосконалень. Метою є забезпечення безперервної функціональності та зручності використання програмного забезпечення протягом усього його життєвого циклу [5].

Ці етапи методології Waterfall, як правило, виконуються в суворій послідовності, причому кожен етап завершується до переходу до наступного. Важливо відзначити, що методологія Waterfall часто вважається менш гнучкою порівняно з більш ітеративними та гнучкими підходами, оскільки зміни або оновлення вимог або елементів дизайну можуть бути складними для адаптації після завершення етапу.

1.2.2 Гнучкі методології

Гнучкі методології, такі як Scrum або Kanban, зосереджені на створенні програмного забезпечення в інкрементних ітераціях. У той час як Agile-методології вже наголошують на співпраці та зворотному зв'язку з клієнтами, DevOps доповнює Agile, забезпечуючи безперервну інтеграцію, тестування та розгортання змін у коді. DevOps автоматизує процеси, впорядковує співпрацю між командами розробників і операторів та підвищує ефективність, тим самим ще більше прискорюючи створення цінних програмних інкрементів [3].

1.2.3 Lean методологія

Методологія Lean спрямована на усунення відходів та оптимізацію потоку створення цінності, зосереджуючись на ефективності та постійному вдосконаленні. DevOps узгоджується з принципами Lean, наголошуючи на автоматизації, зменшенні ручної передачі та оптимізації процесів. Автоматизуючи завдання, DevOps усуває марнотратство, скорочує час очікування і підвищує загальну ефективність розробки та доставки програмного забезпечення [4].

1.2.4 Спіральна методологія

Методологія Spiral поєднує в собі елементи як Waterfall, так і Agile, зосереджуючись на зменшенні ризиків та ітеративній розробці. DevOps доповнює спіральну методологію, забезпечуючи надійну основу для безперервної інтеграції, розгортання та моніторингу. Це дозволяє швидше виявляти та зменшувати ризики завдяки автоматизованому тестуванню та моніторингу, а також полегшує ітеративну розробку та зворотній зв'язок з клієнтами [4].

1.2.5 Швидка розробка додатків (RAD)

Методологія RAD наголошує на швидкому створенні прототипів та ітеративній розробці для швидкої доставки робочого програмного забезпечення. DevOps підтримує RAD, забезпечуючи ефективний та автоматизований спосіб безперервної інтеграції, тестування та розгортання змін. Завдяки DevOps команди RAD можуть оптимізувати процеси розробки та розгортання, скоротити час циклу та покращити співпрацю, що в кінцевому підсумку прискорює створення високоякісного програмного забезпечення [3].

Таким чином, методологія DevOps пропонує значні переваги над різними методологіями розробки програмного забезпечення, сприяючи ітеративній розробці, безперервній інтеграції та співпраці між командами розробників і операторів. Вона доповнює інші методології, підвищуючи ефективність, прискорюючи доставку та забезпечуючи більш гнучкий і орієнтований на клієнта підхід.

1.3 Постановка задачі

Метою кваліфікаційної роботи є показати приклад налагодження інфраструктури, створеної за допомогою програмних інструментів та практик, які лежать в основі методології DevOps.

Проєкт складається з наступних етапів:

1. Створення на локальній машині репозиторію з допомогою системи керування версій Git. Цей етап передбачає ініціалізацію репозиторію та збереження початкового проєкту.
2. Створення віддаленого репозиторію GitHub та відправка початкового проєкту (веб-сайту) з локальної машини на GitHub. Цей етап дозволяє зберігати та керувати версіями проєкту за допомогою віддаленого репозиторію.
3. Створення та налаштування на хмарному провайдері AWS інстансів на платформі Linux (віртуальних машин) для роботи Jenkins, SonarQube та Docker. Цей етап включає налаштування хмарних ресурсів для подальшої роботи з інструментами DevOps.
4. Захищене підключення по SSH до інстансів за допомогою додатку MobaXtern. Цей етап забезпечує безпечний доступ до віртуальних машин та можливість їх налаштування.
5. Завантаження та налаштування сервісів на віртуальних машинах та через веб-додатки для їх роботи між собою. Цей етап передбачає встановлення та налаштування необхідних сервісів, таких як Jenkins, SonarQube та Docker, для подальшої автоматизації розгортання та управління продуктом.
6. Запуск автоматизованої інфраструктури. Цей етап передбачає запуск налаштованої інфраструктури, що дозволяє проводити безперервну інтеграцію, розгортання та управління продуктом з використанням практик DevOps.

2 ВИБІР МЕТОДУ РОЗВ'ЯЗАННЯ ЗАДАЧІ

2.1 Інформаційна модель

Отже, після встановлення мети роботи було створено наступну структуру:

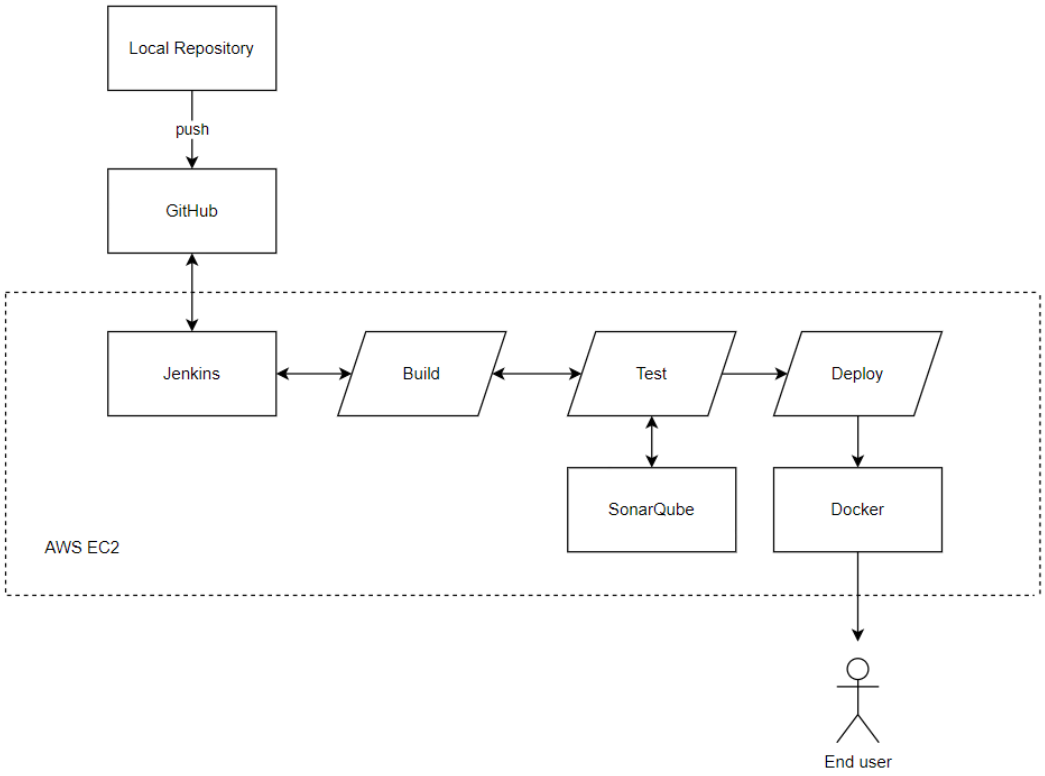

Рисунок 2.1 – Архітектура проєкту

де Local Repository – хостова машина;

GitHub – хмарне сховище файлів (у нашому випадку репозиторій проєкту);

Jenkins – інструмент зв'язку усіх програмних інструментів для спільної роботи;

SonarQube – платформа неперервного аналізу статичного коду.

Docker – служить прикладом ізольованого, незалежного від налаштувань операційної система, середовища.

Даний “конвеєр” працює наступним чином:

1) Розробник робить зміни в своєму локальному репозиторії (Local Repository), які після команди “git push”, яка надсилає зміни на віддалений репозиторій GitHub, оновлює репозиторій;

2) На Jenkins, який під’єднаний до GitHub, отримує триггер, що репозиторій був оновлений, і автоматично починає процес збірки коду та аналізує отриманий код на вразливість, синтаксичні помилки, тощо;

3) Після успішного завершення збірки та аналізу коду, репозиторій надсилається на Docker сервер, який формує Dicker Image, та на основі нього формує контейнер, доступний, при зміні налаштувань доступу в EC2, за портом 8085, доступ до якого має вже кінцевий користувач (у нашому випадку це замовник сайту, який може протестувати поведінку продукту (у нашому випадку сайт) на ізольованій системі.

2.2 Вибір програмних інструментів

2.2.1 Система контролю версій

Системи контролю версій (СКВ) – це програмні інструменти, які дозволяють відстежувати зміни у файловій системі та контролювати версії файлів та проєктів. Найпоширенішими системами контролю версій є Git, Mercurial, Subversion (SVN) та CVS.

У даній роботі було використано систему контролю версій Git для керування та відстеження змін у вихідному коді під час розробки програмного забезпечення.

Git – це розподілена система контролю версій (VCS), яка використовується для відстеження змін у вихідному коді під час розробки програмного забезпечення. Вона дозволяє декільком розробникам співпрацювати над

проєктом, керувати різними версіями файлів, а також ефективно об'єднувати та синхронізувати свою роботу [6]. Git розроблений як швидкий, гнучкий і масштабований, що робить його однією з найпоширеніших систем контролю версій в індустрії.

Основні можливості та способи використання Git:

1) **Контроль версій**: Git зберігає повну історію змін, внесених до файлів у сховищі, що дозволяє розробникам відстежувати, порівнювати та відкочувати зміни за потреби (Рисунок 3.2.1). Це дозволяє командам працювати одночасно над різними гілками, об'єднувати зміни та ефективно вирішувати конфлікти. Можливості контролю версій Git'a забезпечують надійний та структурований спосіб управління кодовими базами [7].

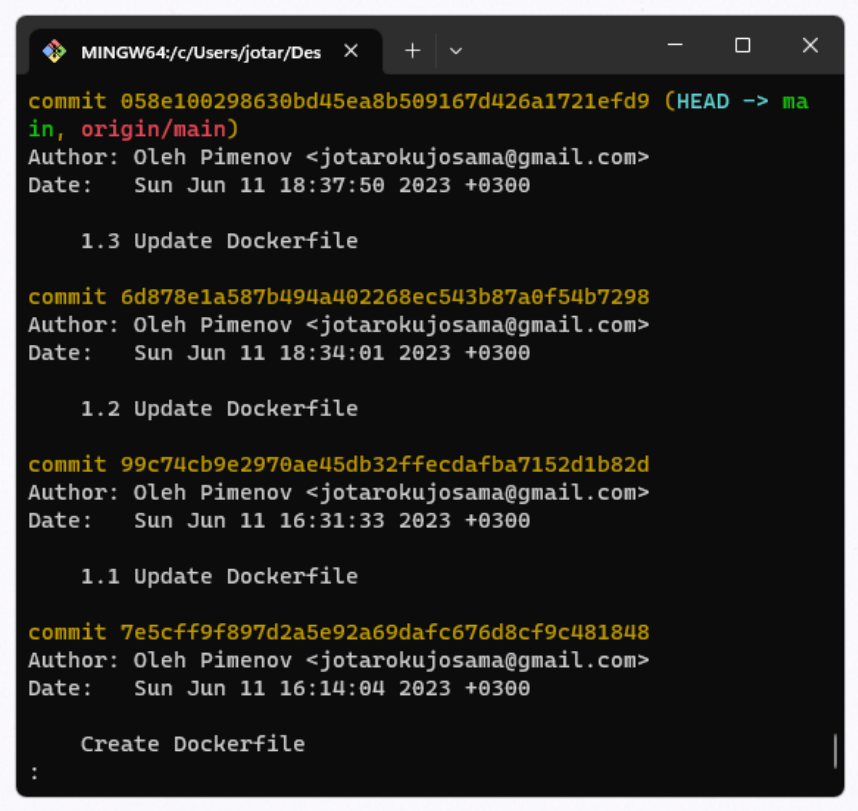

Рисунок 2.2 – Приклад виконання команди git log (для отримання списку комітів в репозиторій)

2) **Розподілена розробка**: Git – це розподілена система контролю версій, що означає, що кожен розробник має локальну копію всього репозиторію. Це дозволяє працювати в автономному режимі, пришвидшує роботу та полегшує співпрацю між розподіленими командами. Розробники можуть фіксувати зміни, створювати гілки та виконувати різні операції локально до синхронізації з центральним репозиторієм [7].

3) **Розгалуження та злиття**: Git надає надійні можливості розгалуження та злиття. Розробники можуть створювати легкі гілки для роботи над новими функціями, виправленнями помилок або експериментів незалежно, не впливаючи на основну кодову базу. Функція злиття Git'a дозволяє об'єднувати зміни з різних гілок та інтегрувати їх назад в основну гілку, забезпечуючи плавний та контрольований робочий процес розробки [6].

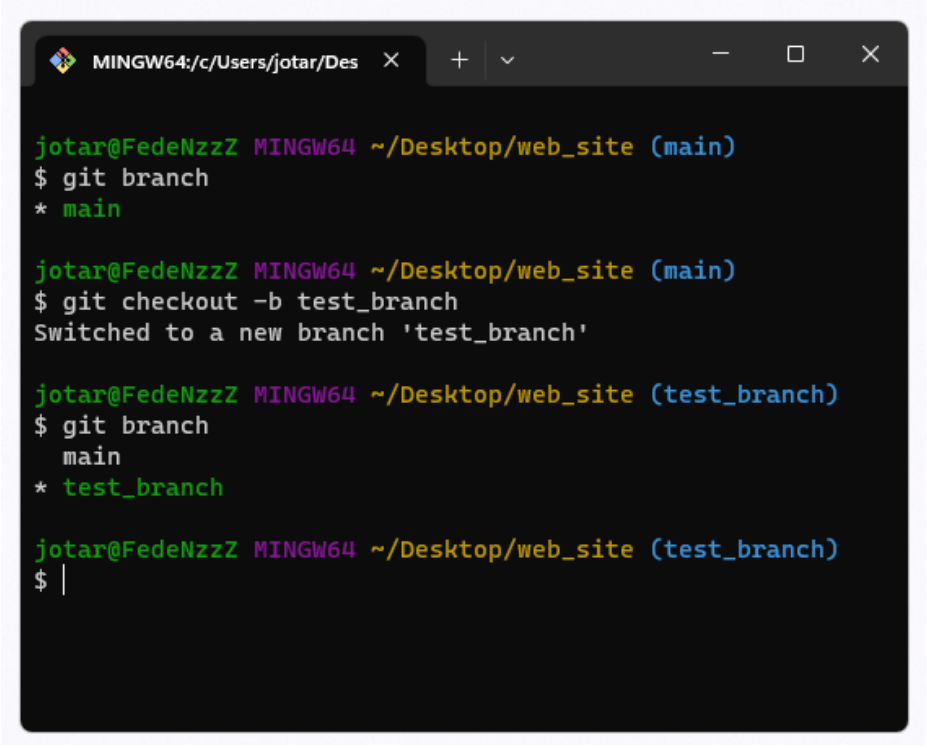

Рисунок 2.3 – Приклад створення нової гілки репозиторію під назвою «test_branch» з одночасним переходом на неї

4) **Співпраця та обмін кодом**: Git полегшує співпрацю між розробниками, надаючи механізми для спільного використання коду та внесення змін. Розробники можуть клонувати репозиторії, витягувати зміни з інших репозиторіїв та вносити власні зміни до віддалених репозиторіїв. Платформи Git, такі як GitHub, GitLab та Bitbucket, ще більше покращують співпрацю, надаючи централізоване місце для розміщення репозиторіїв, управління запитами на витягування та уможливлюючи перегляд коду [6].

5) **Гнучкість робочого процесу**: Git підтримує різні робочі процеси розробки, включаючи централізований, з розгалуженням функцій та Gitflow. Це дозволяє командам прийняти робочий процес, який відповідає їхньому процесу розробки, стратегії розгалуження та підходу до управління випусками. Гнучкість Git'у дозволяє командам пристосовувати свій робочий процес до конкретних потреб і адаптуватися по мірі розвитку проєкту [7].

6) **Швидкість та продуктивність**: Git розроблений для швидкої та ефективної роботи, навіть з великими кодовими базами та довгими історіями. Такі операції, як фіксація змін, розгалуження та злиття, оптимізовані для швидкості, що дозволяє розробникам працювати безперебійно і без значних затримок. Продуктивність Git'у сприяє безперебійній та чуйній розробці [7].

Порівняно з іншими системами контролю версій, Git має кілька переваг. Наприклад, в порівнянні зі старішими системами, такими як Subversion (SVN) та CVS, Git є розподіленою системою, що полегшує роботу в розподілених командах та забезпечує більш гнучкий та ефективний робочий процес. Крім того, Git є дуже популярним у сфері розробки програмного забезпечення, що забезпечує доступ до багатьох ресурсів та досвіду спільноти розробників.

2.2.2 Хмарний репозиторій контролю версій

Хмарний репозиторій контролю версій – це веб-платформа, яка дозволяє розробникам зберігати та керувати своїм кодом, забезпечуючи доступ до нього з будь-якої точки планети, за умови підключення до інтернет-мережі та доступності до репозиторія. Існує кілька популярних хмарних репозиторіїв контролю версій, таких як GitHub, GitLab та Bitbucket. У даній роботі був обраний GitHub.

GitHub – це веб-платформа хостингу для репозиторіїв контролю версій. Вона надає розробникам середовище для спільної роботи, де вони можуть зберігати, керувати та ділитися своїм кодом. GitHub використовує систему контролю версій Git і розширює її функціональність за допомогою додаткових функцій і сервісів. Нижче наведено обґрунтування цього вибору:

1) **Хостинг коду**: GitHub слугує центральним хабом для розміщення Git-репозиторіїв. Розробники можуть створювати репозиторії для своїх проєктів, незалежно від того, чи це особисті проєкти, чи спільні командні зусилля. Сховища можуть бути загальнодоступними, що дозволяє будь-кому отримати доступ до коду та зробити свій внесок у нього, або приватними, що надають обмежений доступ для авторизованих співробітників [8].

2) **Співпраця та внесок**: GitHub сприяє співпраці між розробниками. Він дозволяє декільком учасникам одночасно працювати над одним проєктом, керуючи змінами за допомогою гілок і запитів на витягування. Розробники можуть створювати гілки для самостійної роботи над новими функціями або виправленнями помилок, а потім об'єднувати свої зміни назад в основну кодову базу за допомогою pull-запитів. GitHub надає інструменти для перегляду коду, обговорення та коментування, що дозволяє ефективно співпрацювати та гарантує якість внесків [8].

3) **Відстеження проблем**: GitHub пропонує систему відстеження проблем, яка допомагає керувати завданнями, повідомленнями про помилки, запитами на функції та іншими питаннями, пов'язаними з проєктом. Розробники можуть створювати завдання, призначати їх окремим особам або командам, відстежувати їх прогрес і обговорювати рішення. Ця функція полегшує управління проєктами та покращує комунікацію всередині команд розробників [9].

4) **Безперервна інтеграція та розгортання**: GitHub інтегрується з популярними інструментами безперервної інтеграції та розгортання (CI/CD), такими як Jenkins, Travis CI та CircleCI. Ця інтеграція дозволяє розробникам автоматизувати процеси збірки, тестування та розгортання, викликані змінами в коді. Інтегруючи інструменти CI/CD з GitHub, команди можуть гарантувати, що їхній код постійно тестується, перевіряється та розгортається у виробничих середовищах [9].

5) **Спільнота та проєкти з відкритим кодом**: GitHub має активну спільноту розробників, що робить його центром для проєктів з відкритим кодом. Це платформа, на якій розробники можуть демонструвати свої роботи, співпрацювати з іншими та долучатися до проєктів з відкритим кодом, розміщених на платформі. Соціальні функції GitHub, такі як зірки, форки та дискусії, сприяють залученню спільноти та обміну знаннями [9].

6) **Управління проєктами та документація**: GitHub пропонує функції для управління проєктами та документування. Розробники можуть створювати дошки проєктів, відстежувати завдання та організовувати свою роботу за допомогою дошок у стилі Канбан. GitHub також підтримує створення вікі та сторінок документації, що дозволяє розробникам документувати свої проєкти, API або бібліотеки в самому репозиторії [8].

GitHub широко використовується окремими розробниками, командами розробників та спільнотами з відкритим кодом як центральна платформа для розміщення коду, співпраці та управління проєктами.

2.2.3 Система CI/CD

Система CI/CD (Continuous Integration/Continuous Delivery або Continuous Deployment) є практикою розробки програмного забезпечення, яка спрямована на автоматизацію та прискорення процесу розробки, тестування та випуску програмного забезпечення. CI/CD забезпечує безперервну інтеграцію нового коду в основний репозиторій, його автоматичну збірку, тестування та доставку у виробниче середовище.

У ролі застосунку, що використовується для автоматизації та зв'язку між усіма додатками використовується Jenkins.

Jenkins – це сервер автоматизації з відкритим вихідним кодом, який широко використовується для безперервної інтеграції та безперервної доставки (CI/CD) у розробці програмного забезпечення. Він надає платформу для автоматизації різних завдань, пов'язаних з життєвим циклом розробки програмного забезпечення, таких як створення, тестування та розгортання додатків. Jenkins дозволяє командам оптимізувати процеси розробки, підвищити продуктивність та забезпечити випуск високоякісного програмного забезпечення.

Ключові особливості та сфери використання Jenkins:

1) **Безперервна інтеграція (CI)**: Jenkins забезпечує безперервну інтеграцію, автоматично створюючи та тестуючи зміни програмного забезпечення, коли вони вносяться до системи контролю версій. Він може бути налаштований на запуск збірок, запуск тестів і створення звітів при виявленні змін. Це допомагає виявляти проблеми на ранніх стадіях циклу розробки, сприяє

співпраці та гарантує, що програмне забезпечення залишається у стані готовності до випуску [10].

2) **Безперервна доставка та розгортання (CD)**: Jenkins підтримує безперервну доставку та розгортання, автоматизуючи процес пакування, розгортання та випуску програмного забезпечення в різних середовищах. Це дозволяє створювати конвеєри або робочі процеси, які визначають кроки, пов'язані з побудовою, тестуванням і розгортанням додатків. Така автоматизація зменшує ручну працю, усуває людські помилки та забезпечує швидший і надійніший випуск програмного забезпечення [10].

3) **Розширюваність та інтеграція**: Jenkins надає широку екосистему плагінів, які розширюють його функціональність та забезпечують інтеграцію з різними інструментами та технологіями. Ці плагіни охоплюють широкий спектр областей, включаючи системи управління вихідним кодом (наприклад, Git, Subversion), інструменти збірки (наприклад, Maven, Gradle), фреймворки тестування, інструменти розгортання та служби сповіщень. Така розширюваність дозволяє командам налаштовувати Jenkins відповідно до своїх конкретних потреб та легко інтегрувати його в існуючий інструментарій розробки [11].

4) **Розподілений та масштабований**: Jenkins підтримує розподілені збірки, дозволяючи командам розподіляти робочі навантаження для збірки та тестування між декількома машинами або агентами. Це покращує продуктивність і масштабованість, особливо для великих проєктів або організацій з декількома командами розробників. Jenkins також надає вбудовану підтримку розподілених систем контролю версій, що дозволяє ефективно співпрацювати в розподілених середовищах розробки [11].

5) **Моніторинг та звітність**: Jenkins надає комплексні можливості моніторингу та звітності. Він генерує детальні звіти про результати збірки і тестування, покриття коду та інші метрики, дозволяючи командам відстежувати

стан і якість своїх програмних проєктів. Jenkins також може інтегруватися із зовнішніми інструментами звітності та аналізу, забезпечуючи видимість статусу проєкту та уможливлюючи прийняття рішень на основі даних [10].

Jenkins має кілька переваг порівняно з аналогами, що робить його привабливим вибором для автоматизації та зв'язку в системі CI/CD. Ось кілька прикладів:

1) Jenkins vs. Travis CI:

- Jenkins має ширшу підтримка мов програмування, платформ і технологій порівняно з Travis CI, що робить його універсальнішим і більш гнучким інструментом.
- Jenkins має розширену екосистему плагінів, що дозволяє легко інтегрувати його з різними інструментами та сервісами.
- Jenkins підтримує розподілену архітектуру, що дозволяє розподілити робоче навантаження між декількома машинами або агентами.

2) Jenkins vs. CircleCI:

- Jenkins має безкоштовну версію з відкритим вихідним кодом, що робить його доступним для широкого кола користувачів, в той час як CircleCI має платні плани, особливо для комерційного використання.
- Jenkins має більшу гнучкість у налаштуванні та розширенні завдяки своїй розширеної екосистемі плагінів.
- Jenkins підтримує велику кількість інтегрованих інструментів і технологій, що дозволяє легко інтегрувати його з існуючим стеком розробки.

3) Jenkins vs. GitLab CI/CD:

- Jenkins є незалежним від системи контролю версій, тоді як GitLab CI/CD є вбудованим у GitLab. Це означає, що Jenkins може бути використаний з будь-якою системою контролю версій.

- Jenkins має більшу кількість плагінів та інтеграцій, що дозволяє легко розширити його функціональність та підключити до різних інструментів.
- Jenkins надає більшу гнучкість у налаштуванні складних робочих процесів і конвеєрів.

Загалом, Jenkins є популярним і широко використовуваним інструментом для автоматизації CI/CD завдяки своїй гнучкості, розширюваності, широкій підтримці і сильній спільноті. Ці переваги роблять Jenkins привабливим вибором для розробників і команд, які шукають потужне та багатомодульне рішення для автоматизації своїх процесів розробки програмного забезпечення.

2.2.4 Система статичного аналізу коду

Системи статичного аналізу коду (Static Code Analysis Systems) є інструментами, які використовуються для автоматичної перевірки якості вихідного коду без його виконання. Вони аналізують структуру та синтаксис коду, виявляють помилки, вразливості, неправильні практики та надають рекомендації для його поліпшення. Деякі з таких систем аналізу коду включають Checkstyle, PMD, SonarQube, FindBugs, ESLint і PyLint.

SonarQube – це інструмент статичного аналізу коду з відкритим вихідним кодом, який використовується для безперервної перевірки якості коду. Він аналізує вихідний код, виявляє помилки, вразливості, "запахи" коду та надає дієві рекомендації для покращення загальної якості програмних проєктів [12]. SonarQube допомагає розробникам підтримувати стандарти коду, виявляти потенційні проблеми та забезпечувати надійність і ремонтопридатність своєї кодової бази.

Ключові функції та можливості SonarQube:

1) **Аналіз якості коду**: SonarQube виконує статичний аналіз коду, скануючи файли вихідного коду для виявлення різних типів проблем у коді. Він

аналізує код за набором попередньо визначених правил і надає докладні звіти про якість коду, виділяючи області, які потребують поліпшення. Цей аналіз допомагає розробникам виявляти та вирішувати проблеми на ранніх стадіях процесу розробки, зменшуючи технічний борг та покращуючи загальну якість коду [12,13].

2) **Виявлення "запахів" коду та вразливостей**: SonarQube виявляє "запахи" коду, які є індикаторами проблем з проєктуванням або реалізацією, що можуть призвести до проблем з ремонтопридатністю. Він також виявляє вразливості в коді, такі як потенційні порушення безпеки або небезпечні практики кодування. Висвітлюючи ці проблеми, SonarQube допомагає розробникам проактивно вирішувати їх, покращуючи загальну безпеку та надійність програмного забезпечення [12,13].

3) **Безперервна перевірка**: SonarQube інтегрується з інструментами збірки і системами контролю версій, що дозволяє здійснювати безперервну перевірку коду як частину робочого процесу розробки. Він може бути інтегрований в конвеєри CI/CD для автоматичного аналізу змін коду, забезпечуючи миттєвий зворотній зв'язок з розробниками. Це дозволяє командам виявляти проблеми на ранніх стадіях і гарантувати, що у виробництво розгортається лише високоякісний код [13].

4) **Налаштовувані профілі якості**: SonarQube дозволяє командам визначати власні профілі якості на основі їхніх конкретних вимог. Ці профілі можуть включати набори правил, пристосовані до стандартів кодування команди, найкращих практик і потреб конкретного проєкту. Налаштовуючи профілі якості, команди можуть гарантувати, що аналіз відповідає їхнім стандартам кодування і зосереджується на найбільш важливих для їхнього проєкту питаннях [13].

5) **Звітність і метрики**: SonarQube генерує комплексні звіти і метрики, які дають уявлення про тенденції якості коду, технічну заборгованість та інші

відповідні показники. Він візуалізує дані на зручній інформаційній панелі, що дозволяє легко відстежувати прогрес покращення якості коду з плином часу. Звіти допомагають командам відстежувати стан своєї кодової бази, визначати пріоритети рефакторингу та приймати рішення на основі даних для покращення якості програмного забезпечення [13].

SonarQube є потужним інструментом для управління якістю коду і широко використовується в різних середовищах розробки програмного забезпечення. Він забезпечує надійну та зручну підтримку коду, допомагаючи розробникам виявляти та вирішувати проблеми на ранніх стадіях розробки, поліпшувати безпеку та загальну якість програмного забезпечення.

2.2.5 Контейнеризація

Контейнеризація – це методологія, яка дозволяє розробникам упаковувати програми та їх залежності в ізольовані середовища, відомі як контейнери. Контейнери забезпечують портативність, ефективне використання ресурсів та спрощений процес розгортання додатків. Вони дозволяють працювати узгоджено в різних обчислювальних середовищах, усувають конфлікти залежностей та спрощують управління версіями. Контейнеризація також надає можливості для масштабування, безперервної інтеграції та розгортання додатків. Docker є однією з таких платформ.

Docker преставляє собою найпопулярнішу програмну платформу для контейнеризації, яка практично не має конкурентів. Тому обраний був саме він.

Docker – це платформа з відкритим вихідним кодом, яка дозволяє розробникам автоматизувати розгортання та управління додатками в програмних контейнерах.

Огляд Docker та його використання:

1) **Контейнеризація**: Docker дозволяє упаковувати програми у контейнери, які включають усі необхідні залежності та конфігурації. Контейнери є ізольованими та портативними, тобто вони можуть працювати на будь-якій системі з встановленим Docker, незалежно від базової інфраструктури. Контейнеризація спрощує процес розгортання та забезпечує узгоджену поведінку в різних середовищах [14].

2) **Ефективність використання ресурсів**: Контейнери Docker ефективно використовують системні ресурси хоста. Вони використовують ядро операційної системи хоста і потребують лише необхідних бібліотек та залежностей для запуску програми. Контейнери займають мінімум місця, що робить їх легкими та швидкими для запуску. Таке ефективне використання ресурсів дозволяє запускати декілька контейнерів на одному хості, оптимізуючи розподіл ресурсів та максимально використовуючи інфраструктуру [14].

3) **Управління залежностями**: Docker спрощує управління залежностями додатків. Пакуючи додатки та їх залежності в контейнери, він усуває конфлікти між різними версіями бібліотек або програмних компонентів. Додатки працюють у своїх ізольованих середовищах, забезпечуючи доступність необхідних залежностей без втручання в роботу інших додатків або хост-системи [15].

4) **Масштабованість та відтворюваність**: Docker забезпечує горизонтальне масштабування додатків шляхом запуску декількох контейнерів у розподілених системах. Контейнери Docker можна легко реплікувати, що забезпечує гнучке масштабування на основі попиту. Крім того, Docker надає механізм керування версіями та відстеження образів контейнерів, забезпечуючи відтворюваність розгортань. Це полегшує керування різними версіями додатків та відкат до попередніх версій за потреби [15].

5) **Безперервна інтеграція та розгортання**: Docker добре інтегрується з робочими процесами безперервної інтеграції та розгортання (CI/CD).

Контейнери Docker можна створювати та тестувати у послідовний та відтворюваний спосіб, гарантуючи, що додаток поводитиметься однаково у середовищах розробки, тестування та виробництва. Образи контейнерів Docker можуть бути розгорнуті на різних етапах конвеєра CI/CD, що сприяє автоматизації та стандартизації процесу доставки програмного забезпечення [14].

6) **Екосистема та оркестрування**: Docker має багату екосистему інструментів та сервісів, які розширюють його функціональність. Docker Compose дозволяє розробникам визначати багатоконтейнерні додатки за допомогою YAML-файлу, спрощуючи управління складними мультисервісними архітектурами. Docker Swarm та Kubernetes - це оркестрові платформи, які дозволяють керувати та масштабувати контейнерні додатки на кластерах машин. Ці інструменти надають розширені можливості для балансування навантаження, виявлення сервісів та забезпечення високої доступності [15].

Docker був використаний в даній роботі через його простоту використання, яка дозволяє швидко розпочати роботу з контейнерами без складних налаштувань. Крім того, його ефективне використання ресурсів дозволяє максимально оптимізувати використання обчислювальної потужності. Можливість масштабування додатків у Docker дозволяє легко збільшувати їх обсяг залежно від потреб. Нарешті, Docker ідеально інтегрується з процесами безперервної інтеграції та розгортання, що спрощує автоматизацію та стандартизацію доставки програмного забезпечення.

2.2.6 Хмарний провайдер

Хмарні провайдери – це компанії, які надають послуги обчислення, зберігання даних і різні інші послуги через Інтернет. Вони мають великі масиви серверів і інфраструктури, що розташовані у різних регіонах світу і пов'язані великою мережею.

Хмарні провайдери дозволяють організаціям та користувачам здійснювати обчислення, зберігання даних, розгортання веб-додатків, запуск інфраструктури, аналіз даних та багато іншого без необхідності володіти та підтримувати велику фізичну інфраструктуру власними силами. [16] Замість цього, вони можуть орендувати ресурси у хмарному провайдері, платити лише за фактично використані послуги та масштабувати їх відповідно до потреб.

Кожен хмарний провайдер має свій набір послуг та характеристик, але загальною метою є надання легкого доступу до потужних обчислювальних ресурсів і інструментів розробки. Зазвичай вони пропонують віртуальні машини, контейнери, зберігання об'єктів, бази даних, інструменти розгортання, аналітичні сервіси, штучний інтелект, інтернет речей та інші послуги.

Клієнти можуть використовувати хмарні послуги для розв'язання своїх завдань безпосередньо через веб-консоль або за допомогою програмних інтерфейсів (API). Хмарні провайдери також надають можливості масштабування ресурсів вгору або вниз, в залежності від потреб користувача [17].

На сьогоднішній день існує кілька провідних хмарних провайдерів, таких як Amazon Web Services (AWS), Microsoft Azure та Google Cloud Platform (GCP). Вони усі надають різноманітні хмарні сервіси, включаючи обчислювальні ресурси, сховища даних, мережі та інші послуги.

У даній роботі використовується саме AWS через його наступні переваги:

- **Лідер ринку та досвід**: AWS є лідером ринку в хмарному просторі протягом тривалого часу. Amazon почав пропонувати послуги хмарних обчислень у 2006 році, за кілька років до того, як Google і Microsoft вийшли на ринок. Ця перевага забезпечила AWS широку клієнтську базу, розгалужену інфраструктуру та широкий спектр послуг.

- **Пропозиції послуг**: AWS має більший портфель послуг у порівнянні з GCP та Azure. Станом на 2021 рік AWS пропонує понад 200 повнофункціональних послуг з центрів обробки даних по всьому світу. Ці послуги охоплюють різні сфери, такі як обчислення, зберігання, бази даних, аналітика, мережеві технології, мобільні технології, інструменти для розробників, інструменти управління, IoT, безпека та корпоративні додатки [20].
- **Глобальна присутність**: AWS має ширшу глобальну присутність у порівнянні з GCP та Azure. Він має більше зон доступності та центрів обробки даних по всьому світу, що може бути критично важливим фактором для підприємств, які прагнуть до мінімальної затримки та високої надмірності.
- **Масштабованість та продуктивність**: AWS пропонує надійні, масштабовані рішення, які можуть задовольнити потреби як стартапів, так і підприємств. Він також демонструє високу продуктивність в еталонних тестах.
- **Зрілі пропозиції послуг**: Завдяки ранньому запуску, багато сервісів AWS є більш зрілими та багатофункціональними порівняно з аналогічними сервісами в GCP та Azure. Наприклад, Amazon S3 (Simple Storage Service) та Amazon EC2 (Elastic Compute Cloud) часто розглядаються як більш надійні та міцні у порівнянні з їхніми аналогами в GCP та Azure.
- **Зручний для розробників**: AWS має широкі набори SDK (Software Development Kits) та добре задокументований API, що полегшує розробникам створення, розгортання та керування додатками.
- **Ціноутворення**: AWS пропонує підхід до ціноутворення за принципом "платити по мірі використання". Хоча GCP та Azure також пропонують схожі моделі ціноутворення, AWS часто виявляється більш економічно вигідним рішенням, особливо для великих робочих навантажень завдяки знижкам за обсяг та цінами на зарезервовані екземпляри.

- **Інтеграція та сумісність**: Сервіси AWS добре працюють разом, і AWS добре інтегрується з популярними сторонніми рішеннями. Це часто може полегшити підприємствам перенесення існуючої архітектури в хмару AWS.
- **Навчання та підтримка**: AWS пропонує широкий спектр навчальних та сертифікаційних програм, які користуються великою повагою в галузі. Вона також має потужну службу підтримки клієнтів та технічну підтримку з широким спектром планів, що відповідають різним потребам бізнесу.
- **Безпека**: Хоча всі три провайдери приділяють велику увагу безпеці, модель безпеки AWS є зрілою і перевіреною найдовшим перебуванням на ринку. Він також пропонує безліч інструментів і функцій, які допоможуть забезпечити безпеку хмарних операцій.

AWS надає широкий спектр хмарних сервісів, включаючи обчислювальні потужності, сховища, управління базами даних, мережу тощо. Однією з основних послуг, яка була використана в даній роботі, є Amazon Elastic Compute Cloud (EC2).

Amazon EC2 – це веб-сервіс, який пропонує обчислювальні потужності в хмарі з можливістю масштабування. Він надає віртуальні сервери, відомі як екземпляри, що дозволяють користувачам створювати та керувати віртуальними машинами в інфраструктурі AWS. Екземпляри EC2 можна налаштувати за допомогою різних конфігурацій, таких як вибір операційної системи, процесора, пам'яті, сховища та мережевих параметрів.

EC2 пропонує:

- **Масштабовані обчислення**: EC2 дозволяє користувачам масштабувати свої обчислювальні ресурси на основі попиту. Користувачі можуть легко запустити кілька екземплярів або налаштувати потужність існуючих екземплярів, щоб впоратися зі зміною робочих навантажень. EC2 забезпечує гнучкість щодо типів і розмірів екземплярів, дозволяючи користувачам обирати

найбільш підходящу конфігурацію відповідно до продуктивності та вимог до ресурсів їхніх додатків [21].

- **Гнучка конфігурація**: Користувачі мають контроль над різними аспектами своїх екземплярів EC2. Вони можуть вибрати потрібну операційну систему, налаштувати параметри безпеки, визначити мережеві параметри та вибрати варіанти зберігання даних відповідно до потреб свого додатку. EC2 пропонує широкий вибір типів екземплярів, оптимізованих для різних сценаріїв використання, таких як екземпляри загального призначення, оптимізовані для пам'яті, оптимізовані для обчислень та оптимізовані для зберігання [21].
- **Еластичність та висока доступність**: EC2 надає функції для забезпечення високої доступності та відмовостійкості. Користувачі можуть використовувати функцію автоматичного масштабування, яка автоматично регулює кількість екземплярів на основі заздалегідь визначених правил, щоб впоратися з різним навантаженням трафіку та підтримувати продуктивність. Екземпляри EC2 також можуть бути розгорнуті в декількох зонах доступності для розподілу робочого навантаження і захисту від збоїв інфраструктури [21].
- **Плата за фактом використання**: EC2 працює за моделлю "оплата по факту", що дозволяє користувачам платити лише за ресурси, які вони споживають. Користувачі мають можливість запускати або зупиняти екземпляри за потреби, а рахунки виставляються лише за фактичну тривалість використання. Така модель ціноутворення робить EC2 економічно ефективним і дозволяє користувачам оптимізувати свої витрати на інфраструктуру [21].
- **Інтеграція з іншими сервісами AWS**: EC2 легко інтегрується з іншими сервісами AWS, що дозволяє користувачам створювати масштабовані та надійні рішення. Користувачі можуть використовувати такі сервіси, як Amazon Elastic Block Store (EBS) для постійного зберігання даних, Amazon Simple Storage Service (S3) для зберігання об'єктів та Amazon Relational Database Service (RDS)

для керованих БД, щоб розширити можливості своїх екземплярів EC2 [21]. EC2 широко використовується підприємствами та розробниками для запуску різних типів додатків, від невеликих веб-серверів до великомасштабних корпоративних систем, у хмарній інфраструктурі AWS.

2.2.7 Серверна операційна система

У якості серверної операційної системи був використаний Linux.

Linux – це операційна система з відкритим вихідним кодом, яка широко використовується як серверна платформа завдяки своїй стабільності, безпеці, гнучкості та економічній ефективності. Вона забезпечує надійну основу для розміщення різних типів додатків і сервісів.

Порівнюючи Linux з Windows і macOS як серверні платформи, слід враховувати кілька ключових моментів:

1. **Стабільність і надійність**: Linux відома своєю стабільністю та надійністю, часто перевершуючи Windows та macOS за часом безвідмовної роботи та швидкістю відгуку системи. Дистрибутиви Linux, такі як Ubuntu, CentOS і Debian, розроблені для роботи з великими робочими навантаженнями і забезпечують стабільну продуктивність протягом тривалого часу [19].

2. **Безпека**: Linux має гарну репутацію в плані безпеки. Його відкритий вихідний код дозволяє швидко виявляти вразливості та виправляти їх, а сувора система дозволів забезпечує детальний контроль над доступом користувачів та їхніми привілеями. Крім того, дистрибутиви Linux часто мають вбудовані функції безпеки, що робить їх менш вразливими до шкідливих програм та кіберзагроз у порівнянні з Windows [19].

3. **Гнучкість та кастомізація**: Linux пропонує високий ступінь гнучкості та кастомізації. Він забезпечує широку підтримку різних мов програмування, бібліотек і фреймворків, що дозволяє розробникам адаптувати серверне

середовище до своїх конкретних потреб. Крім того, Linux надає потужні інструменти командного рядка та великий репозиторій програмного забезпечення, що дозволяє адміністраторам конфігурувати та оптимізувати сервер відповідно до своїх вимог.

4. **Економічна ефективність**: Linux, як правило, безкоштовний у використанні, що робить його економічно вигідним вибором для розгортання серверів. Операційна система не вимагає ліцензійних платежів, а відкритий характер Linux заохочує процвітаючу спільноту, яка надає підтримку, документацію та регулярні оновлення [19].

Але, Windows і macOS також мають переваги як серверні платформи:

1. **Windows Server**: Windows Server пропонує безперешкодну інтеграцію з іншими продуктами та сервісами Microsoft, що робить її кращим вибором для організацій, які вкладають значні кошти в екосистему Microsoft. Вона має зручний інтерфейс, широкі інструменти управління та відмінну підтримку Active Directory, що робить її ідеальною для корпоративних середовищ [18].

2. **macOS Server**: macOS Server, хоча і не так широко поширений, як Linux або Windows Server, надає зручний інтерфейс, потужні мережеві можливості та безперешкодну інтеграцію з пристроями та сервісами Apple. Йому часто надають перевагу для невеликих або спеціалізованих розгортань у середовищах, орієнтованих на macOS.

Отже, вибір між Linux, Windows та macOS в якості серверної платформи залежить від кількох факторів, таких як конкретні вимоги вашого проєкту, сумісність з існуючою інфраструктурою, рівень необхідної кастомізації та досвідченість адміністраторів. Linux пропонує стабільність, безпеку, гнучкість та економічність, що робить його популярним вибором для широкого спектру серверних додатків. Однак Windows і macOS мають свої сильні сторони і можуть краще підходити для певних сценаріїв або середовищ.

3 ІНФОРМАЦІЙНЕ ТА ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

3.1 Опис програмної реалізації

Перш за все, після розробки проєкту (Рисунок 3.1) потрібно завантажити систему контроля версій з офіційного сайту (Рисунок 3.2) та ініціалізувати репозиторій (див. Додаток 1).

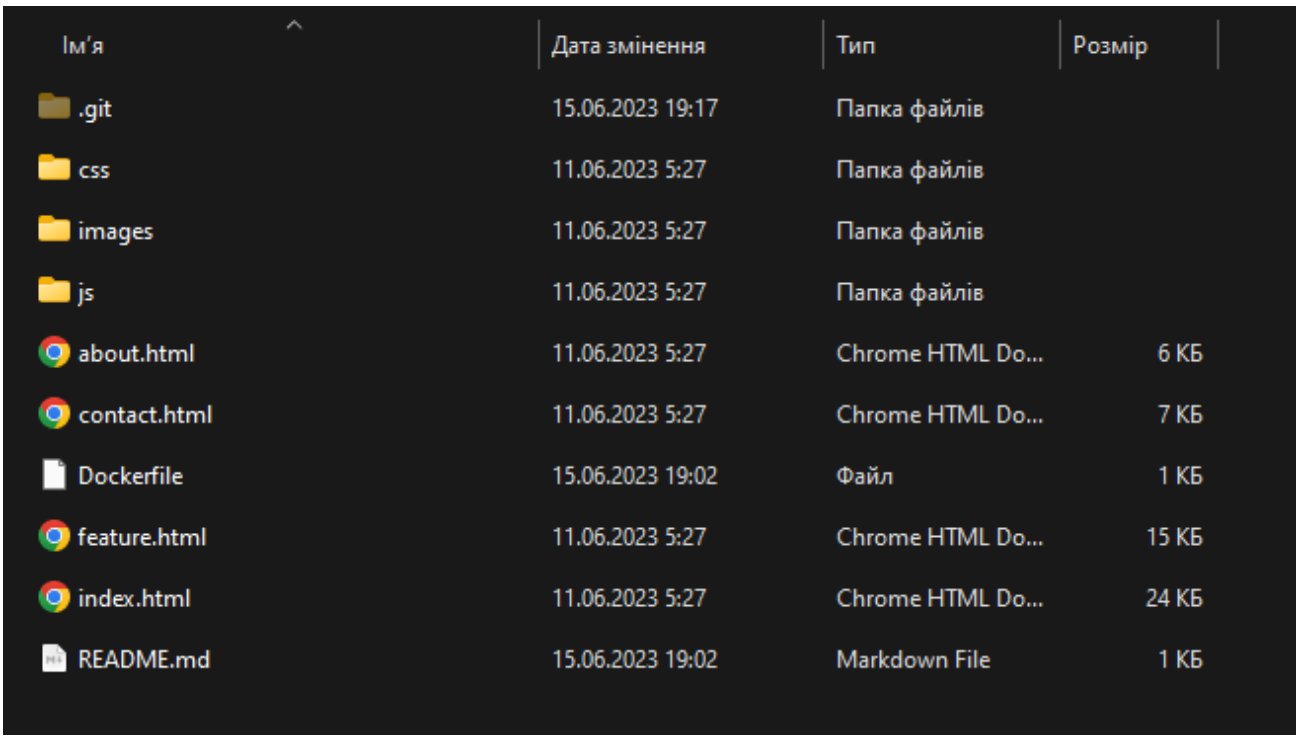

Рисунок 3.1 – Локальний репозиторій з проєктом

Після цього потрібно створити аккаунт на платформі GitHub або ,при його наявності, створити репозиторій (Рисунок 3.3), в який буде перенесено зміст локального репозиторію за допомогою GIT команд (див. Додаток 1, Рисунок 3.4, Рисунок 3.5).

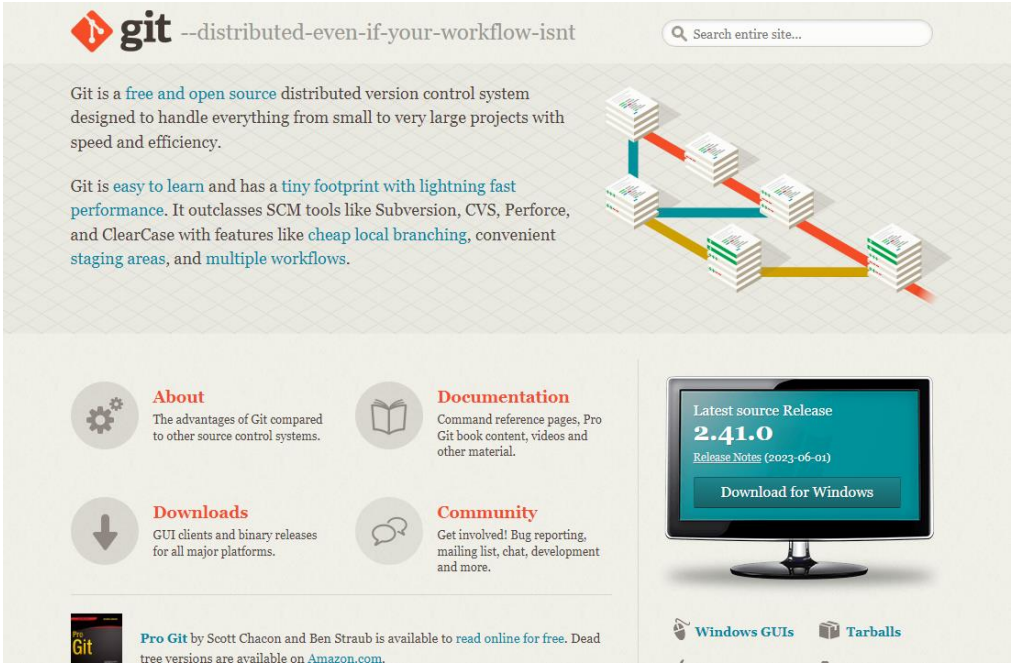

Рисунок 3.2 – офіційний сайт Git

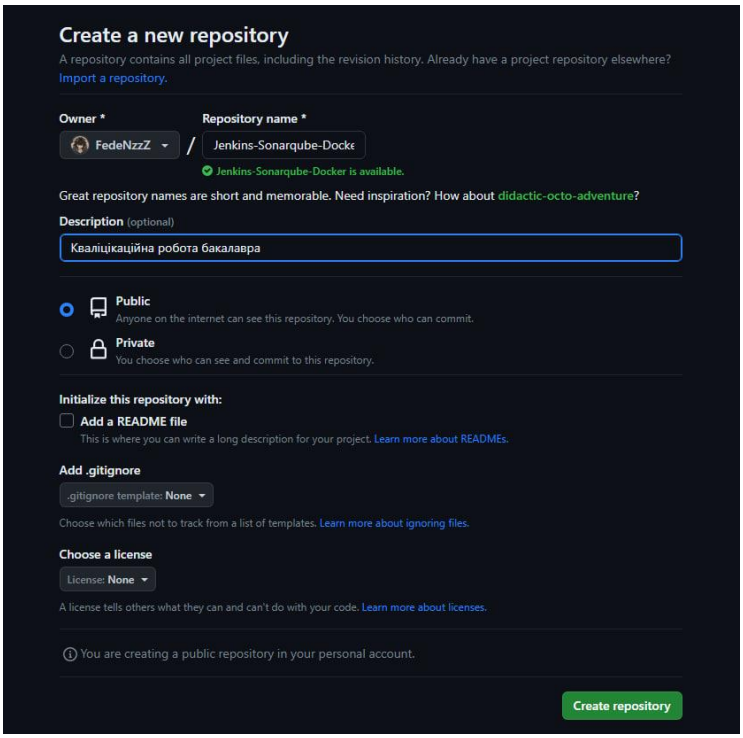

Рисунок 3.3 – Створення репозиторію GitHub

```
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        about.html
        contact.html
        css/
        feature.html
        images/
        index.html
        js/
```

Рисунок 3.4 – Вивід команди “git status”

```
jotar@FedeNzzZ MINGW64 ~/Desktop/web_site (main)
$ git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   about.html
        new file:   contact.html
        new file:   css/bootstrap.css
        new file:   css/responsive.css
        new file:   css/style.css
        new file:   css/style.css.map
        new file:   css/style.scss
        new file:   feature.html
        new file:   images/about-bg.jpg
        new file:   images/client.png
        new file:   images/cloud-download.png
        new file:   images/contact-img.png
        new file:   images/download-img.png
        new file:   images/fb.png
        new file:   images/heart.png
        new file:   images/instagram.png
        new file:   images/instagram1.png
        new file:   images/left-arrow.png
        new file:   images/linkedin.png
        new file:   images/linkedin1.png
        new file:   images/playstore.png
```

Рисунок 3.5 – Фіксування нових файлів в локальному репозиторії

Після цього ми бачимо наш перенесений локальний репозиторій у віддаленому репозиторії GitHub (Рисунок 3.6).

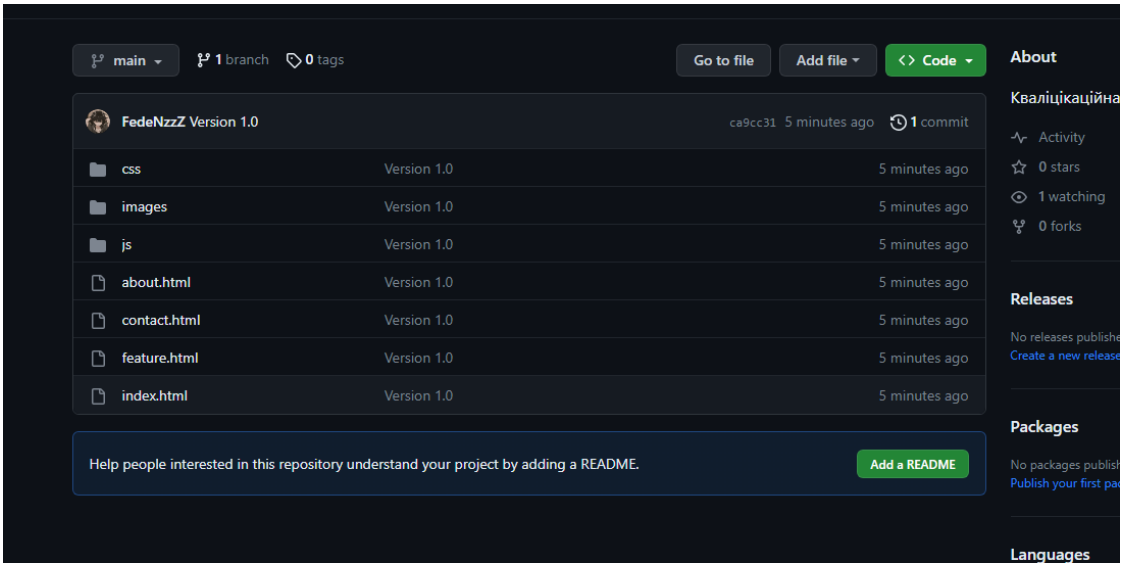

Рисунок 3.6 – Пененесений локальний репозиторій в GitHub

Тепер створемо віртуальні машини за допомогою інструмента EC2:

1. Зареєструватися на сайті AWS;
2. Відкрити інструмент EC2 (Рисунок 3.7);
3. Створити інстранс (віртуальна машина), зазначивши ім’я “Jenkins”, операційну систему Linux (дестрибутив Ubuntu) вибрати підходящий тип пам'яті (Рисунок 3.8);
4. Створити парний SSH ключ (Рисунок 3.8), після чого почнеться завантаження його (Рисунок 3.9).
5. Натиснути кнопку “Lounch Instance”, після чого віртуальна машина буде додана;
6. Виконати повторно ці дії, але змінюючи ім’я для SonarQube та Docker, та вибираючи вже раніше створений SSH ключ. У результаті буде створено та запущено три інстранси (Рисунок 3.10);

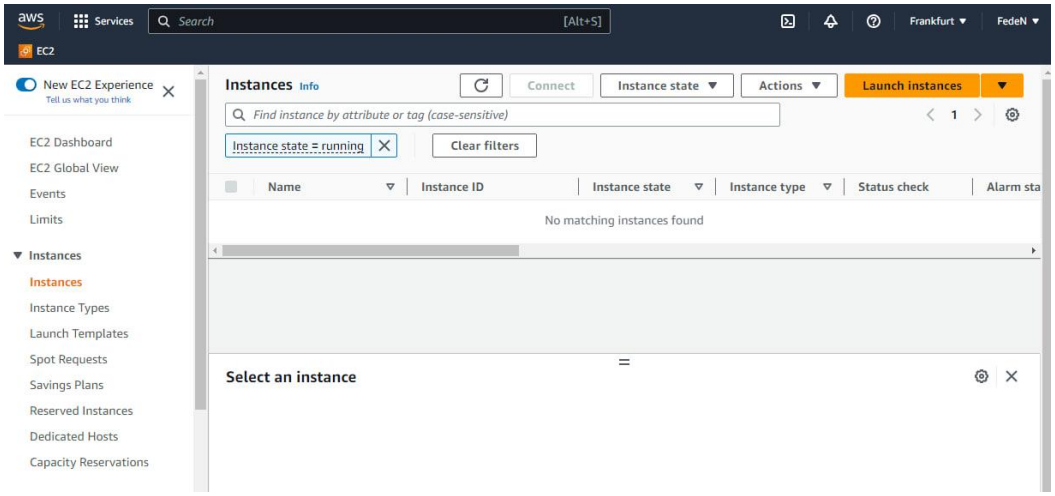

Рисунок 3.7 – Відкриття інструмента EC2

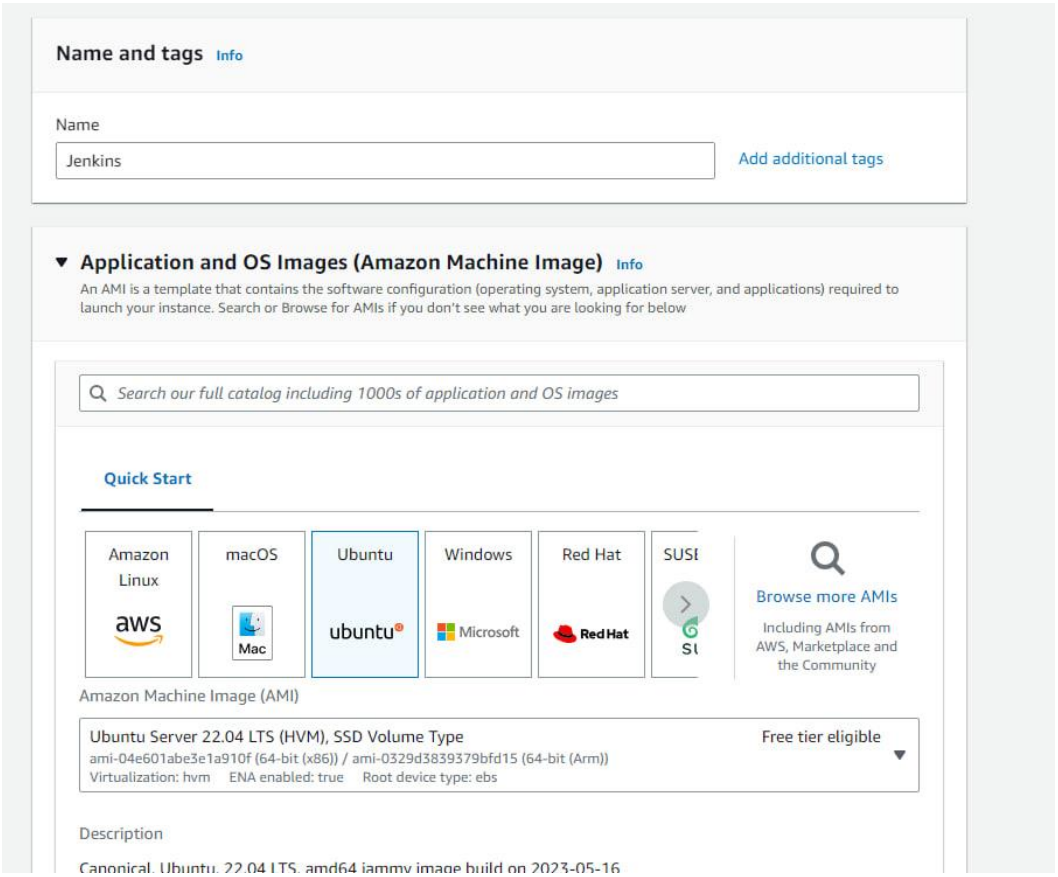

Рисунок 3.8 – Вибір ім’я та операційної системи

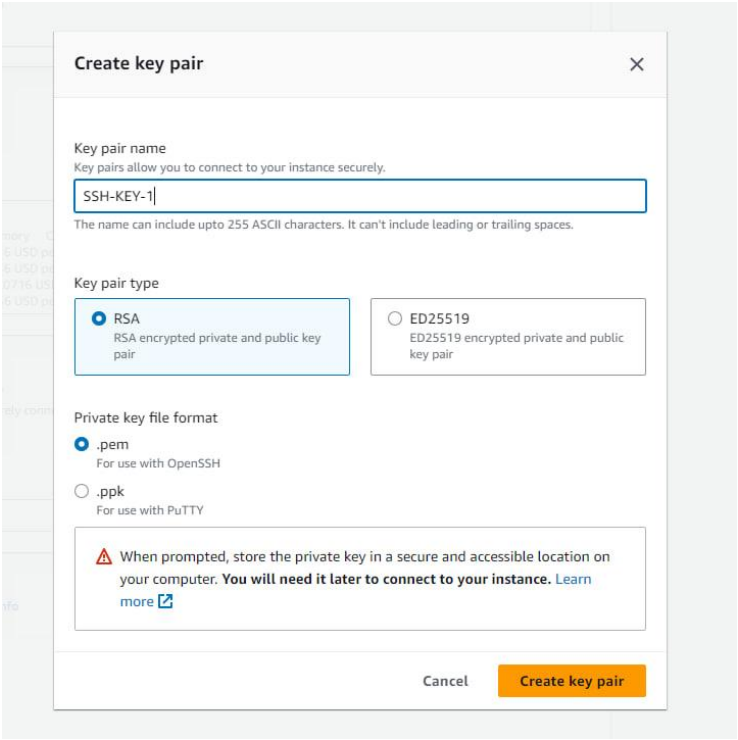

Рисунок 3.9 – Створення ключ-пари

Рисунок 3.10 – завантажений файл-ключ

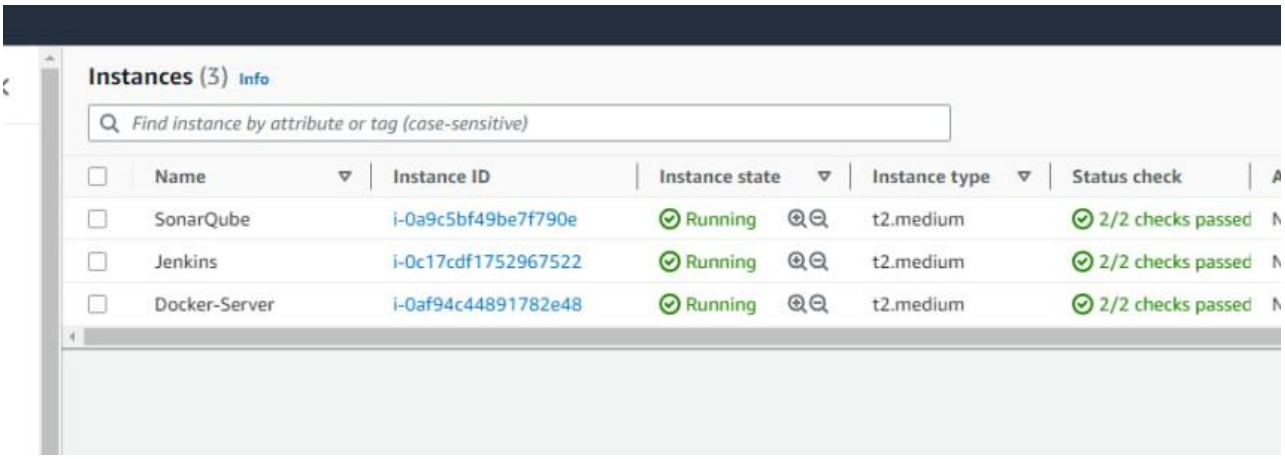

Рисунок 3.11 – Працюючі інстранси SonarQube, Jenkins, Docker

Виконавши ці дії ми можемо підключитись до кожного віртуального сервера за допомогою програми SonarQube (Рисунок 3.12) зовнішнього IP, котрий вказаний в інформації про інстанс, та вставивши ключ-пару, яку ми завантажили (Рисунок 3.13).

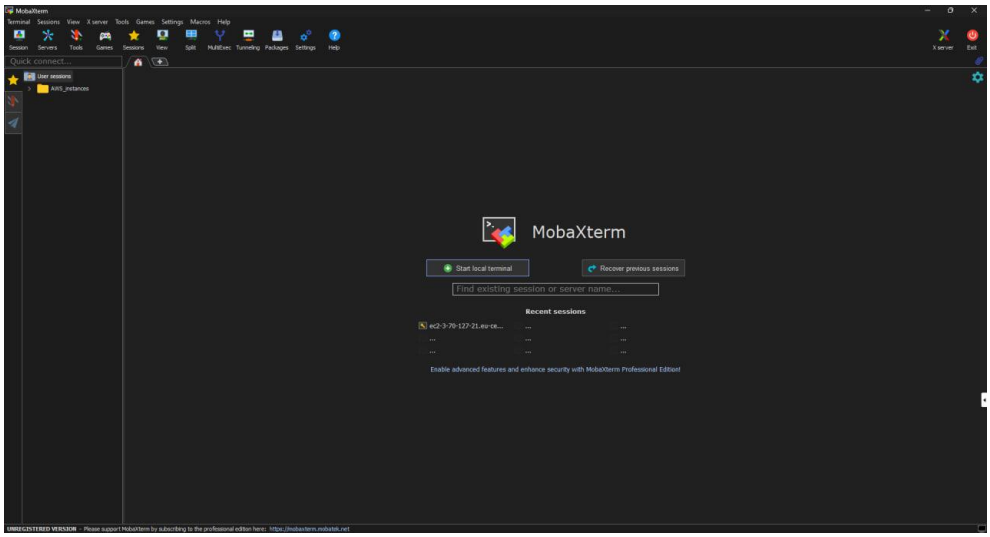

Рисунок 3.12 – Робочий інтерфейс програми SonarQube

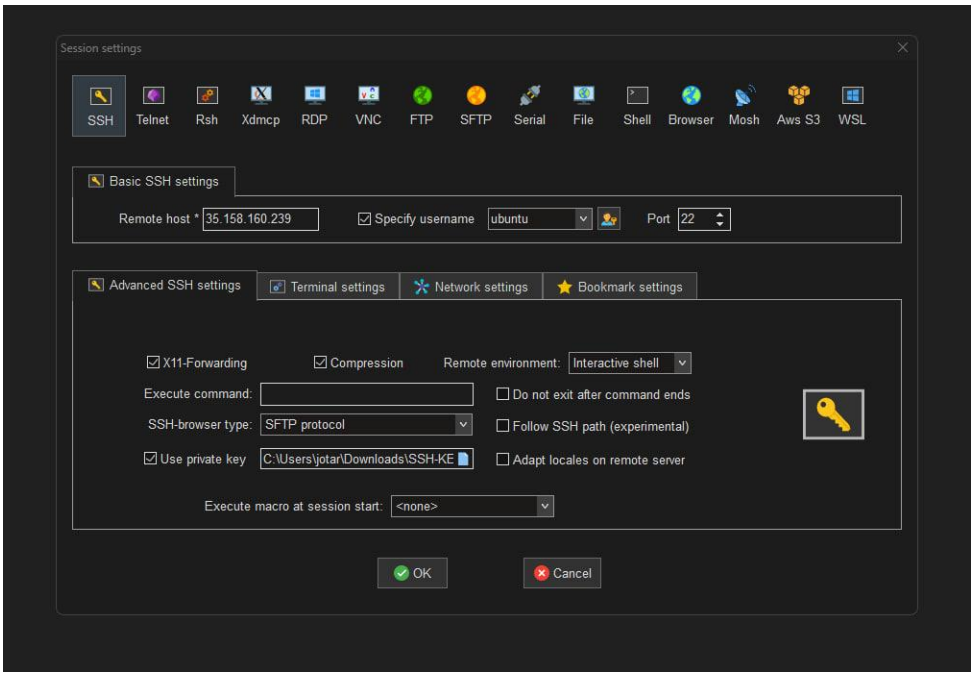

Рисунок 3.13 – Вставка зовнішнього IP, username та ключа-пари

Після цього ми потрапляємо в термінал сервера Jenkins (Рисунок 3.14), де ми виконуємо команду призначену для оновлення інформації про доступні пакети з офіційних репозиторіїв, завантажуємо образ Jenkins, за допомогою списка команд, вказаних в офіційній документації сайту Jenkins'а (Рисунок 3.15).

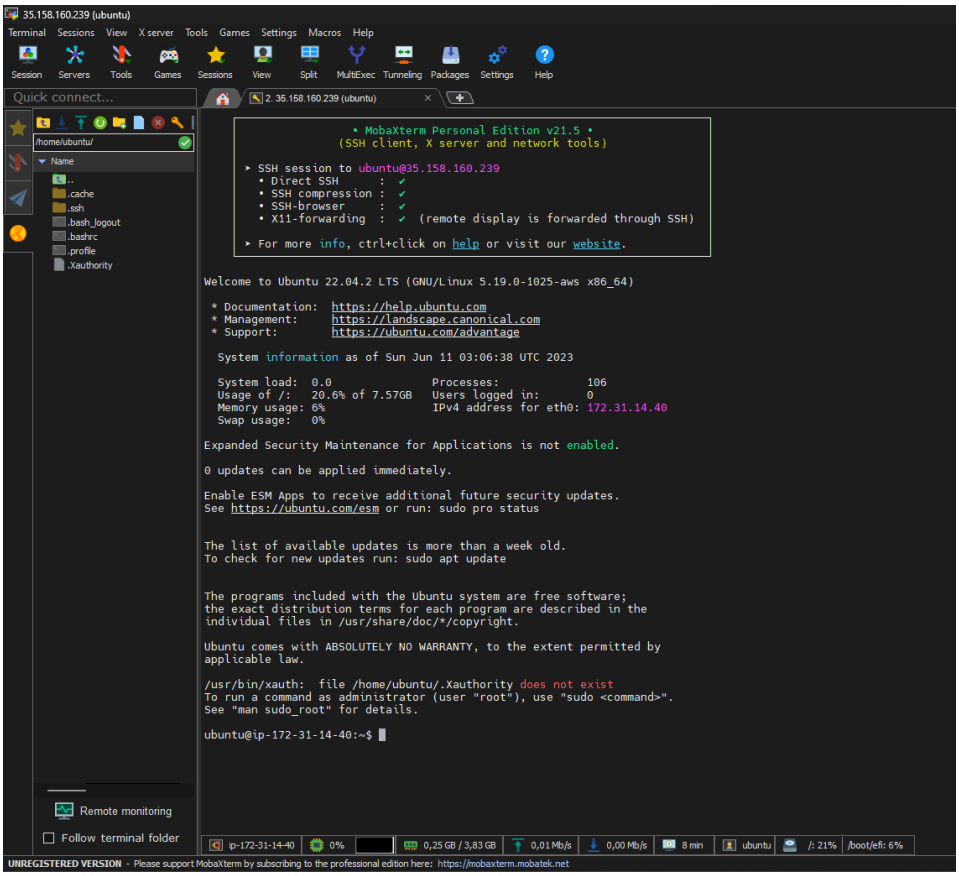

Рисунок 3.14 – Термінал сервера Jenkins

Debian/Ubuntu

On Debian and Debian-based distributions like Ubuntu you can install Jenkins through apt.

Long Term Support release

A LTS (Long-Term Support) release is chosen every 12 weeks from the stream of regular releases as the stable release for that time period. It can be installed from the debian-stable apt repository.

```
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

BASH

Weekly release

A new release is produced weekly to deliver bug fixes and features to users and plugin developers. It can be installed from the debian apt repository.

Рисунок 3.15 – Офіційна документація з покроковим встановленням Jenkins

Однією з умов роботи Jenkins є встановлення Java 11 (див. Додаток Б).

Після цього потрібно перевірити роботу Jenkins та відкрити порт 8080 через налаштування EC2, зазначивши порт 8080, та вибір класу IP, який матиме доступ до цього порту (Рисунок 3.16).

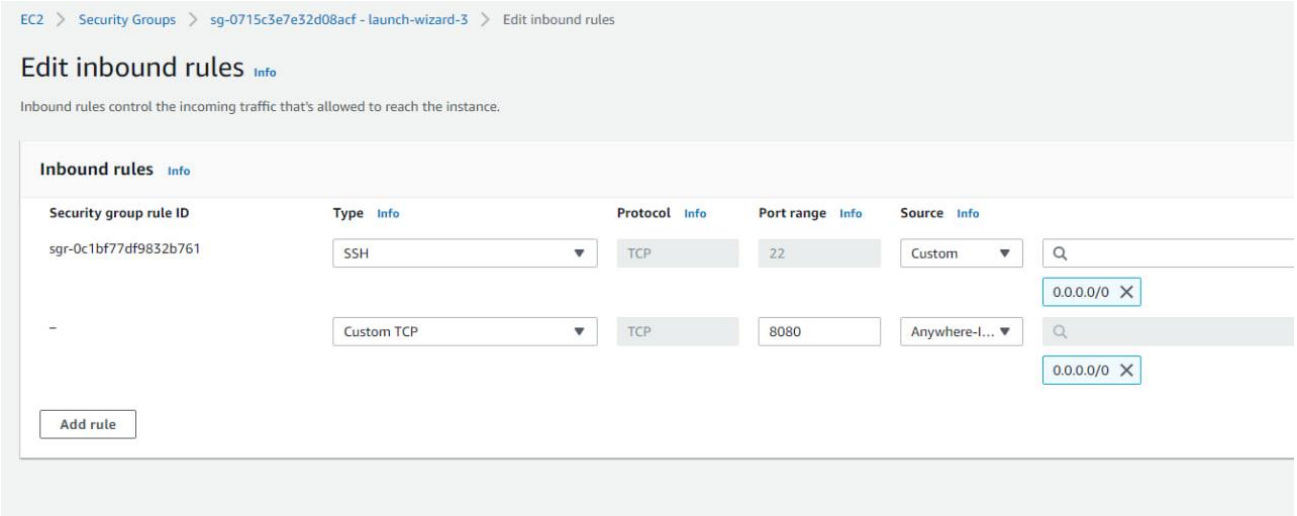

Рисунок 3.16 – Налаштування портів та доступності

Переходимо в браузер та в рядку URL вписуємо зовнішній IP та порт 8080 через «:» наступним чином: http://35.158.160.239:8080/

У рядку пароля вставляємо пароль, що знаходиться в файлах Jenkins на віртуальній машині по вказаному шляху (Рисунок3.17).

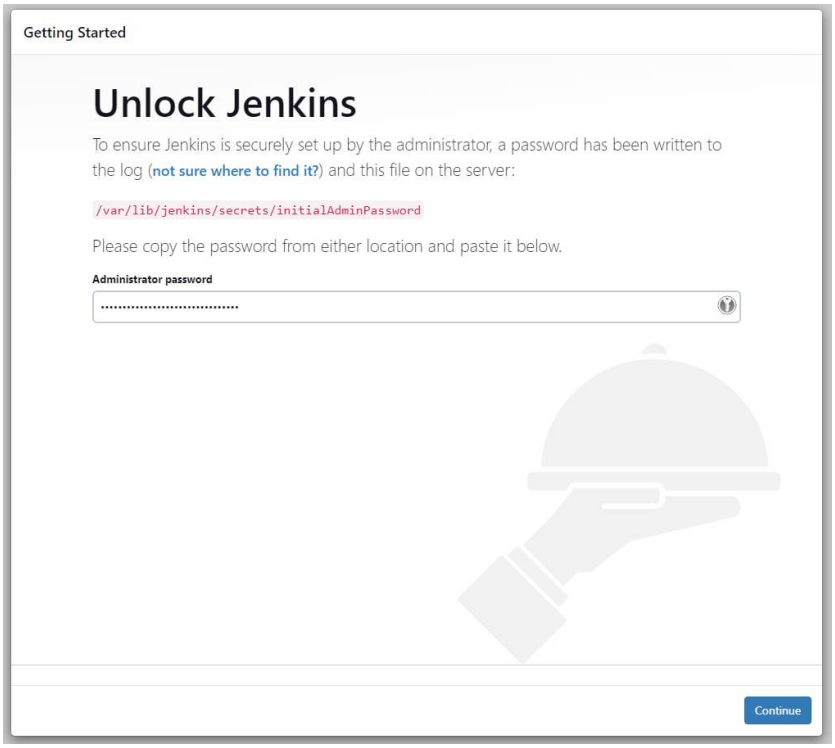

Рисунок 3.17 – Вставка пароля Jenkins

Після цього завантажуємо необхідні стандартні плагіни та потрапляємо на головну робочу сторінку Jenkins нашого сервера (Рисунок 3.18).

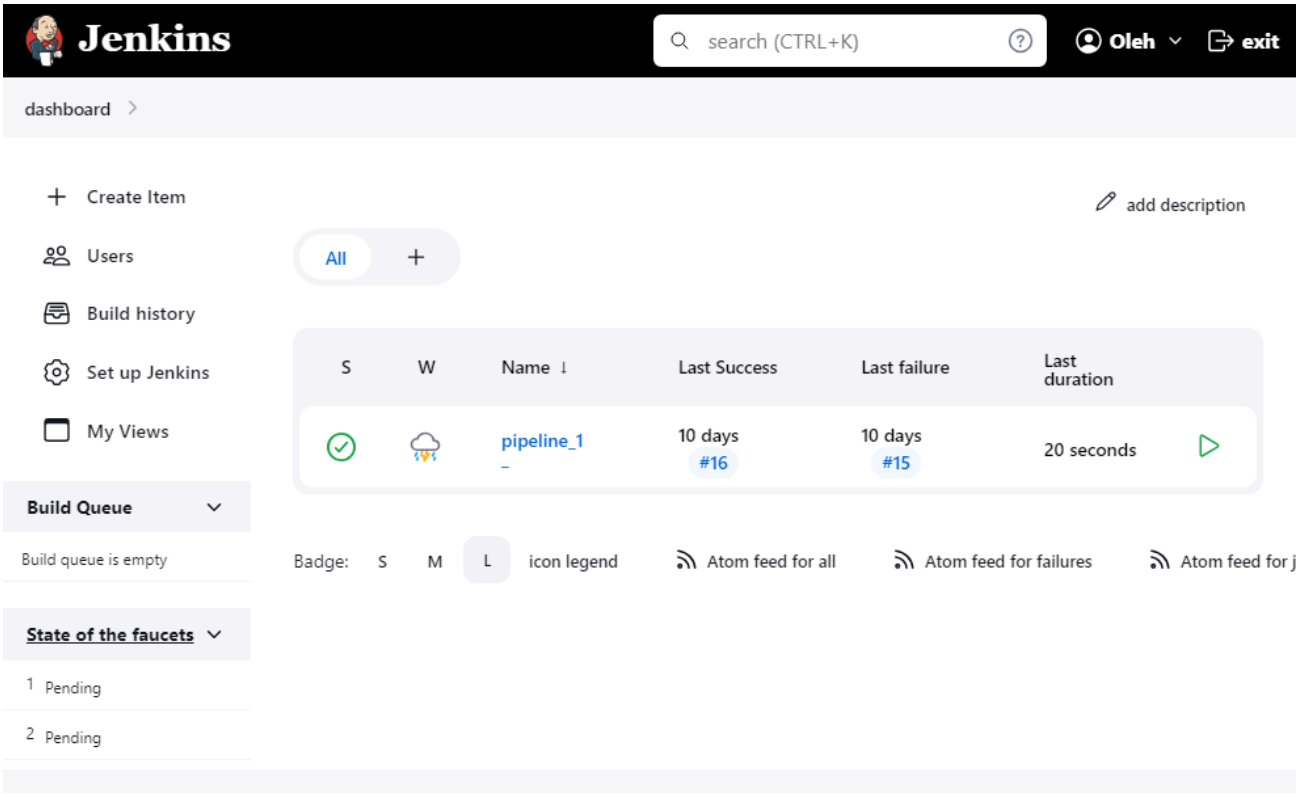

Рисунок 3.18 – Головна сторінка сервера Jenkins

Створюємо конвеєр вільної конфігурації для самостійного налаштування, переходимо в налаштування, обираємо систему початкового коду Git, та вказуємо URL нашого GitHub репозиторію, та зазначаємо усі файли головної гілки “main” (Рисунок 3.19)

У тригерах збірки налаштовуємо GitHub hook, для того, щоб зміни в репозиторії автоматично стягувалися в Jenkins та відбувалася збірка (Рисунок 3.20, Рисунок3.21).

Робимо тестовий білд (збірку), додавши новий файл README.md та бачимо успішний результат (Рисунок 3.22).

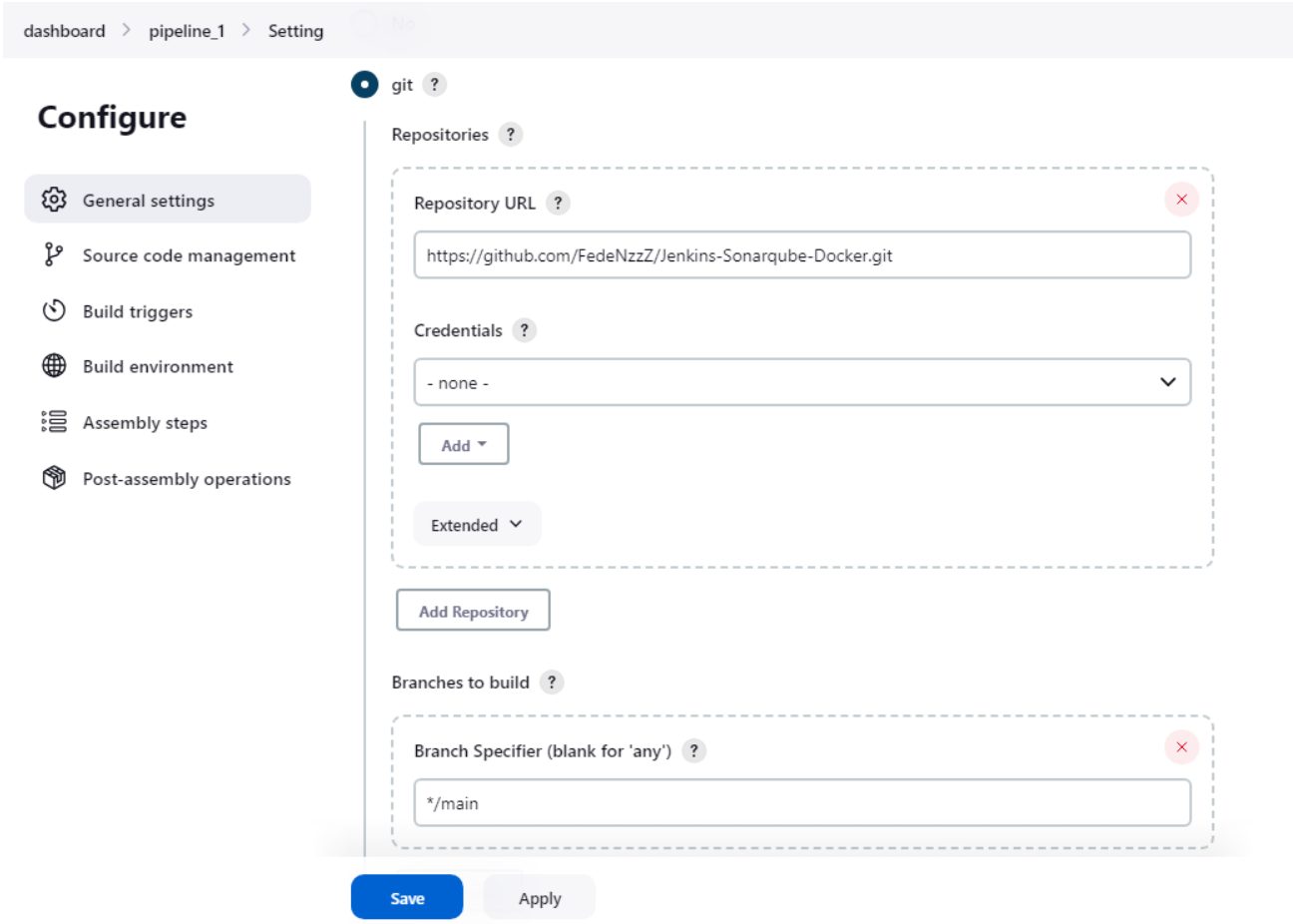

Рисунок 3.19 – Додавання URL та головної гілки репозиторія

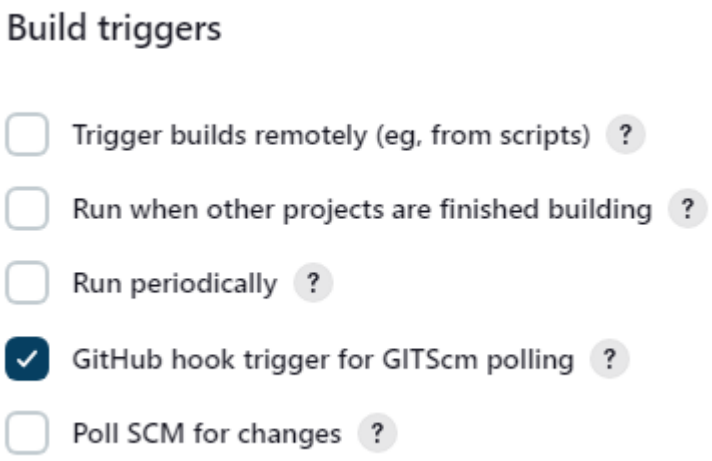

Рисунок 3.20– Тригер збірки

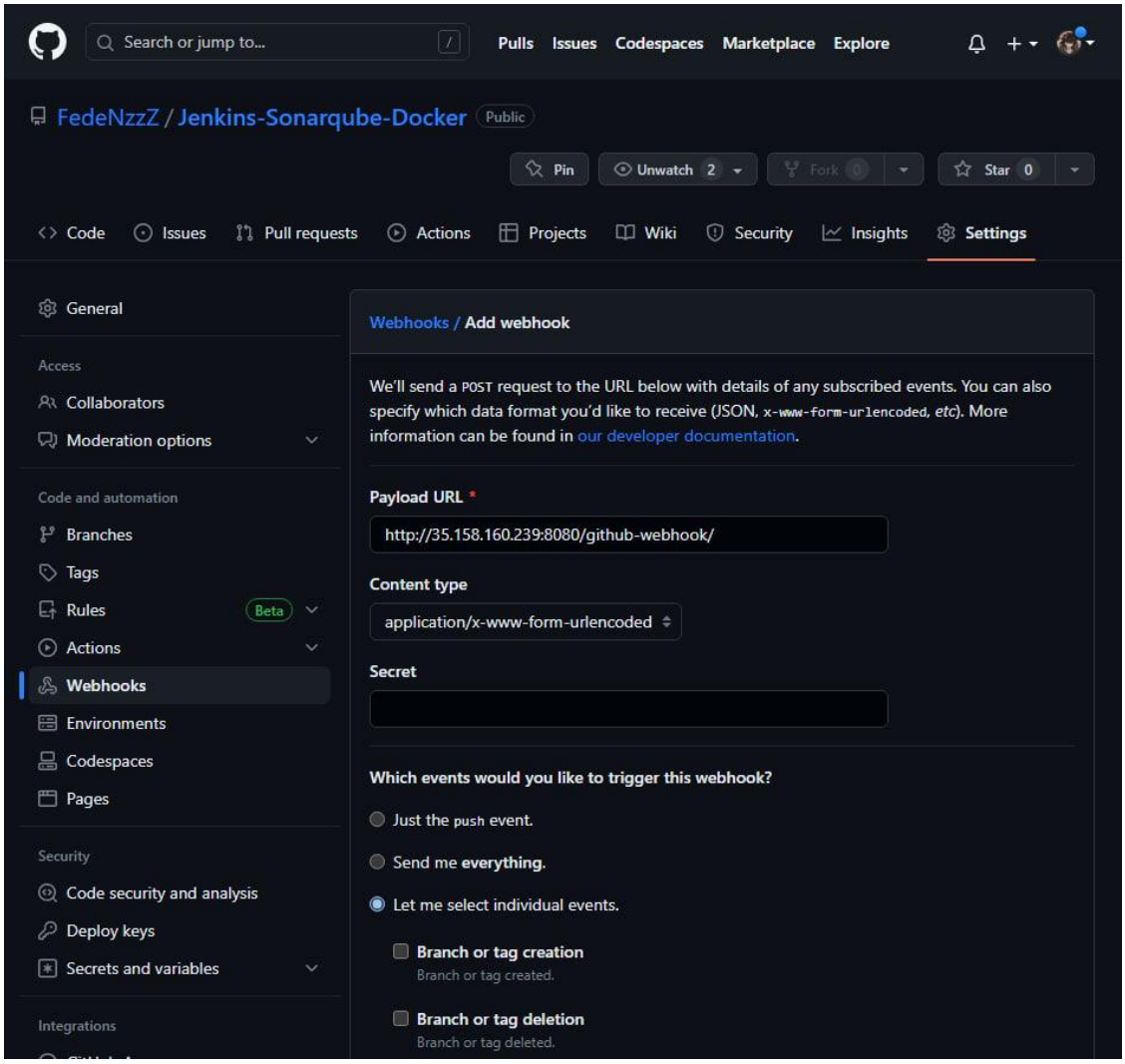

Рисунок 3.21 – Посилання на GitHub Webhooks

Далі таким же методом, як ми підключалися через SSH, підключаємося в SonarQube, перевіряємо наявність нових версій пакетів, встановлюємо Java 17, встановлюємо завантажувач з офіційного сайту за допомогою терміналу, розпаковуємо архів та запускаємо службу SonarQube (див. Додаток В).

Після перевірки на працездатність (Рисунок 3.23), переходимо через браузер по зовнішньому IP та порту 9000, доступ до якого ми відкрили необхідному класу IP користувачам, який ми встановили в налаштуваннях EC2: http://3.74.227.244:9000/

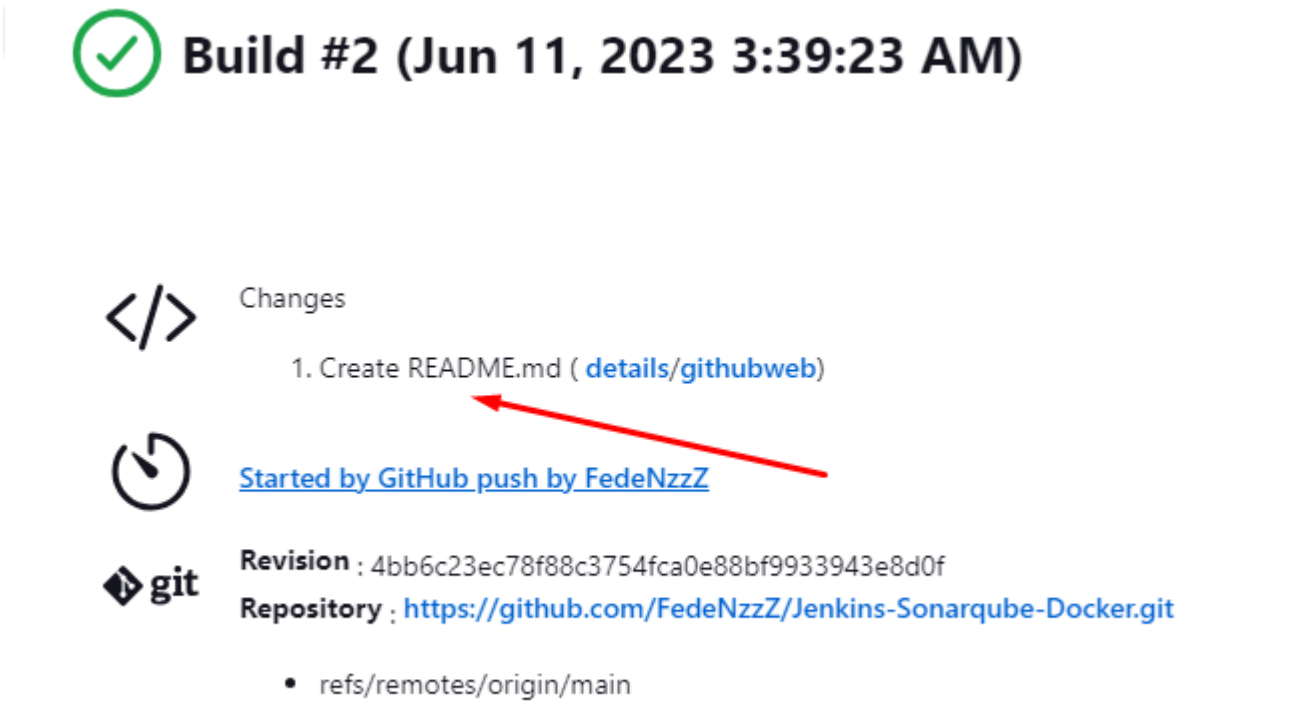

Рисунок 3.22 – Відбулася автоматична збірка коду, та другий білд (додавання файлу README.md)

Реєструємось (Рисунок 3.24), створюємо в Jekninsfile на сайті SonarQube ключ в налаштуваннях та зберігаємо його (Рисунок 3.25).

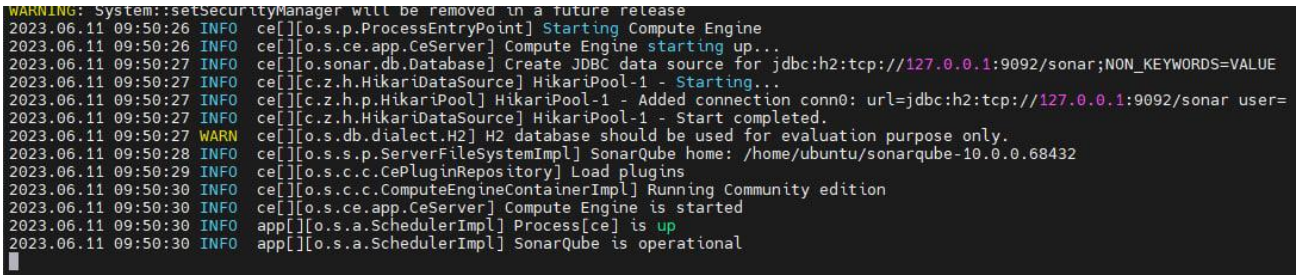

Рисунок 3.23– Превірка працездатності SonarQube

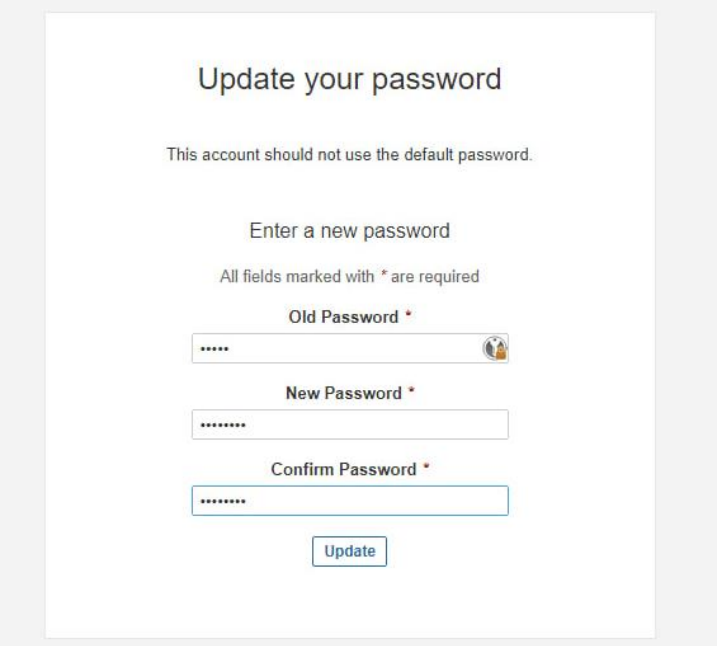

Рисунок 3.24 – SOME TEXT

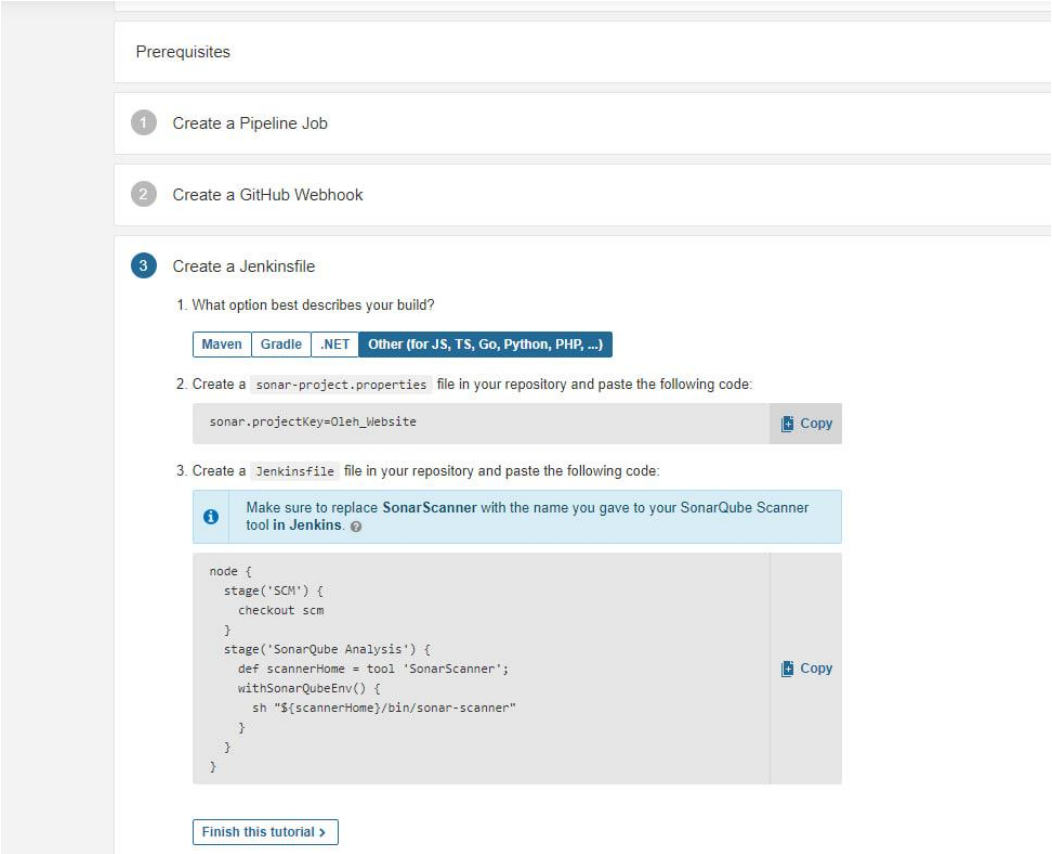

Рисунок 3.25– Ключ до Jekninsfile

Далі завантажуємо плагін SonarQube Scanner в Jeknins (Рисунок 3.26), завантажуємо допоміжні плагіни. Створюємо токен: обираємо потрібну версію, вставляємо IP адресу на порт SonarQube в Jeknins, вставляємо ключ, котрий ми запам’ятовували раніше, та даємо назву (Рисунок 3.27).

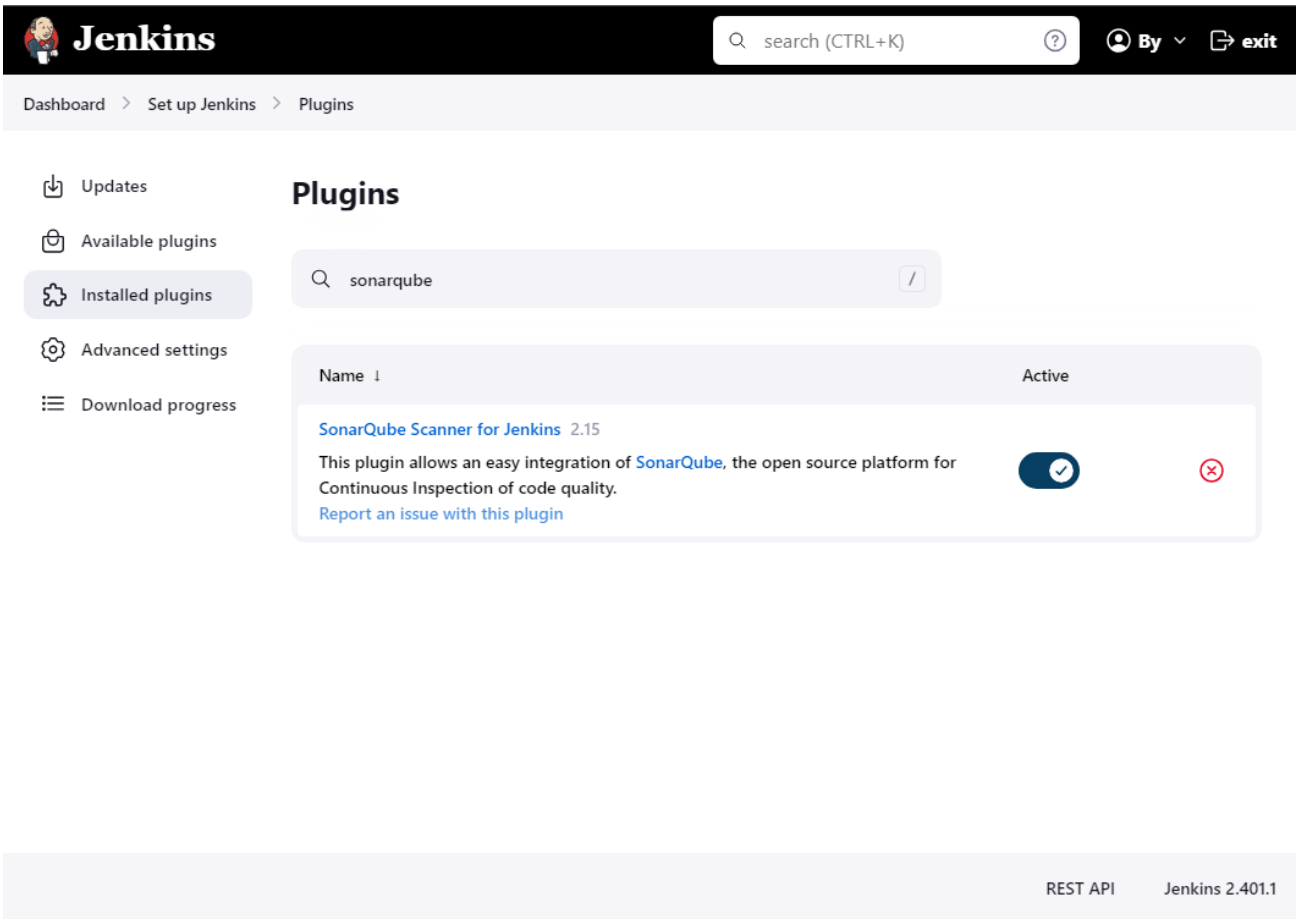

Рисунок 3.26– Завантаження плагіна

Виконуємо тестовий білд зі скануванням (Рисунок 3.28) та бачимо про успішне закінчення. Дані сканування ми можемо побачити на WEB версії нашого сервера SonarQube (Рисунок 3.29).

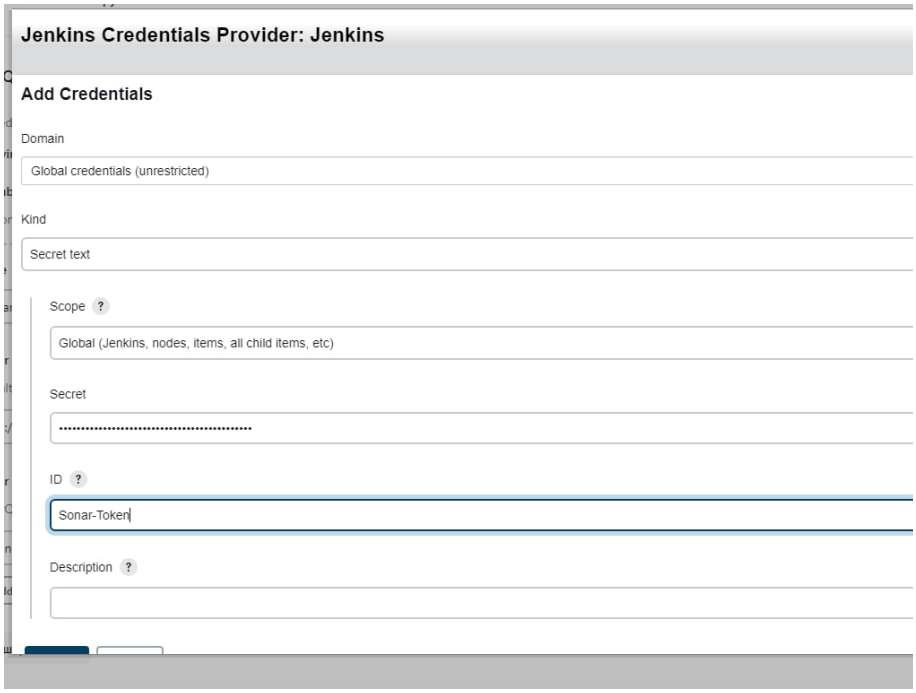

Рисунок 3.27– Ключ та назва токена

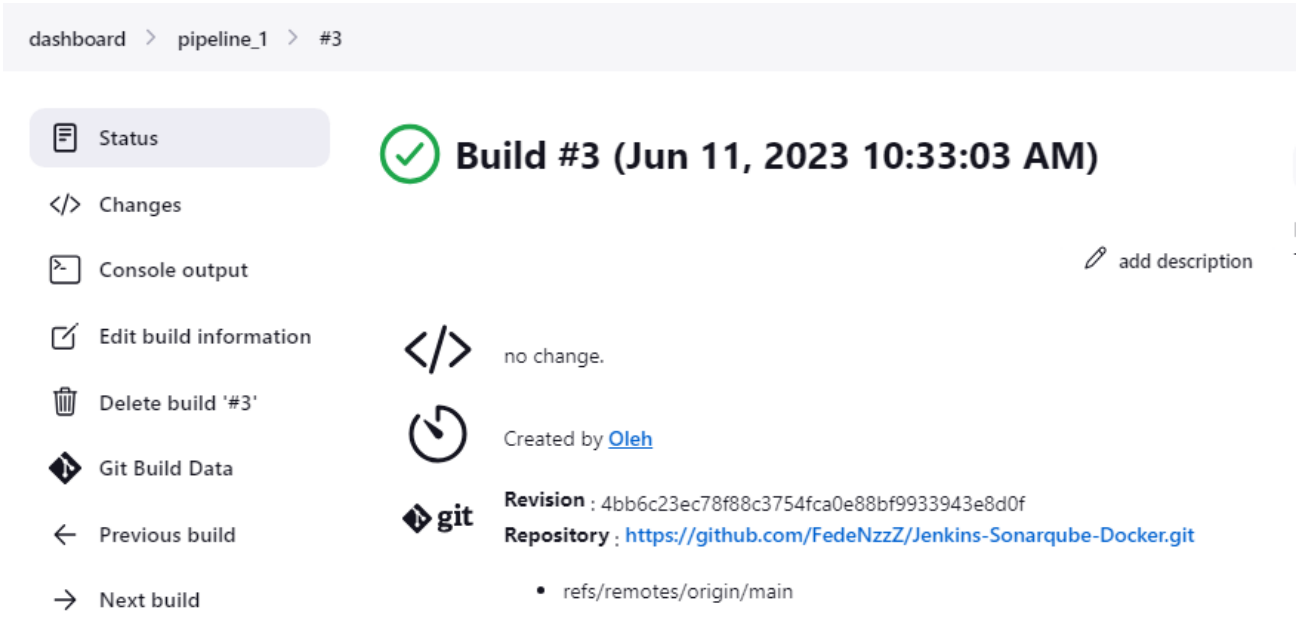

Рисунок 3.28– Тестовий білд зі скануванням

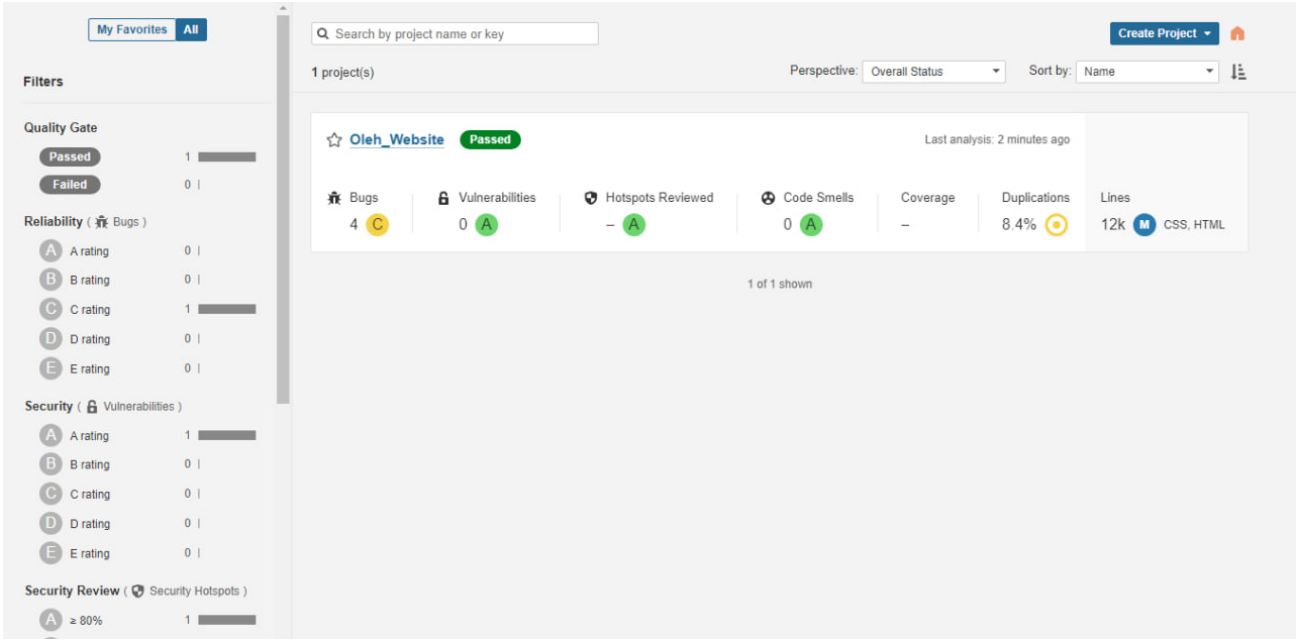

Рисунок 3.29 – Результат сканування коду

Після цього підключаємося на сервер Docker через MobaXtern, змінюємо назву для комфортності (Рисунок 3.30). Завантажуємо Docker за допомогою команд вказаних на офіційній документації Docker (Рисунок 3.31)

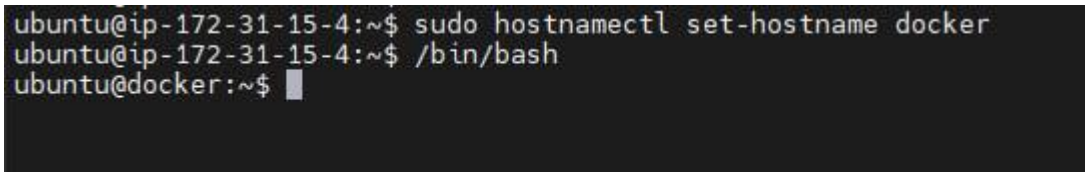

Рисунок 3.30– Зміна назви користувача Ubuntu

Далі потрібно змінити правила доступності Для цього редактуємо відповідний файл (див. Додаток Г), та змінюємо відповідні налаштування видаливши знак «#» перед потрібним атрибутом (Рисунок 3.32, Рисунок 3.33).

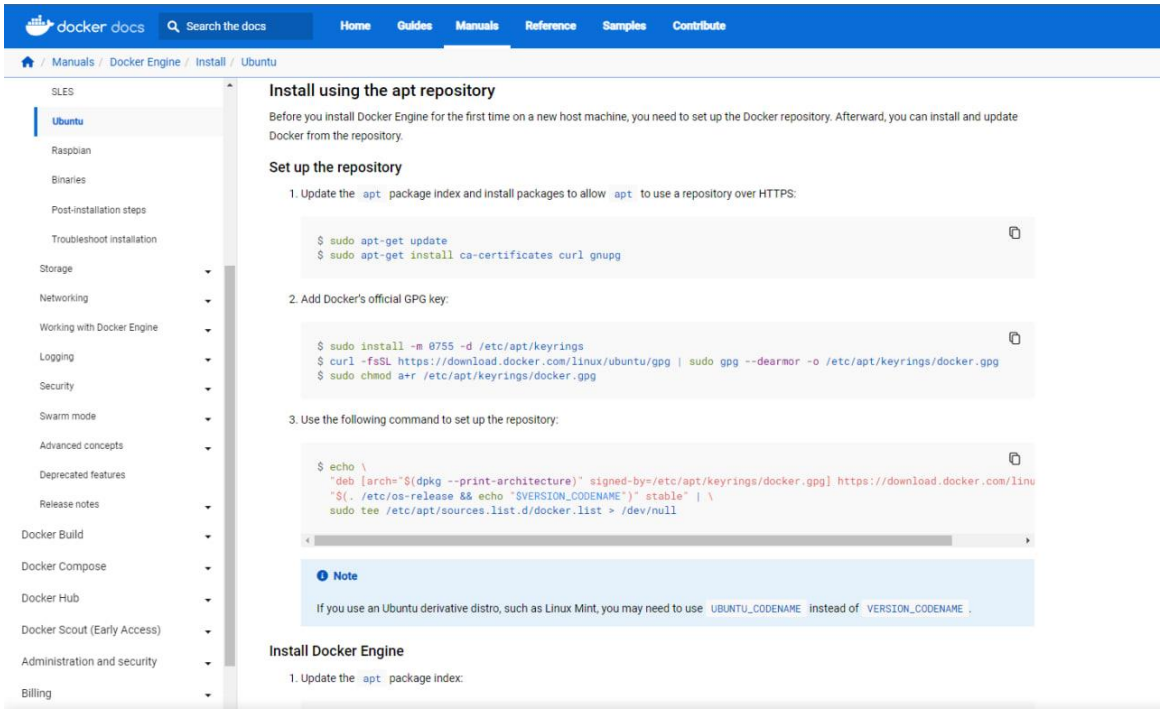

Рисунок 3.31 – Офіційна документація Docker

```
# Ciphers and keying
#RekeyLimit default none

# Logging
#SyslogFacility AUTH
#LogLevel INFO

# Authentication:

#LoginGraceTime 2m
#PermitRootLogin prohibit-password
#StrictModes yes
#MaxAuthTries 6
#MaxSessions 10

PubkeyAuthentication yes

# Expect .ssh/authorized_keys2 to be disregarded by default in future.
#AuthorizedKeysFile     .ssh/authorized_keys .ssh/authorized_keys2

#AuthorizedPrincipalsFile none

#AuthorizedKeysCommand none
#AuthorizedKeysCommandUser nobody

# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
```

Рисунок 3.32– Зміна доступності за публічним ключем

```
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
#HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
# HostbasedAuthentication
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
#IgnoreRhosts yes

# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no

# Change to yes to enable challenge-response passwords (beware issues with
# some PAM modules and threads)
KbdInteractiveAuthentication no

# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
```

Рисунок 3.33 – Зміна доступу на доступ за паролем

Тепер перезапустимо служби, та поставимо новий пароль (Рисунок 3.34), після чого додаємо на Jenkins WEB доступ за ключем, який ми налаштували (див. Додаток Г), для того щоб, можна було виконувати віддалені команди через Jenkins на сервері Docker, додавши групу Docker (Рисунок 3.35), та додавши наш сервер Docker (Рисунок 3.36).

```
root@docker:/home/ubuntu# passwd ubuntu
New password:
Retype new password:
passwd: password updated successfully
root@docker:/home/ubuntu#
```

Рисунок 3.34– Зміна паролю

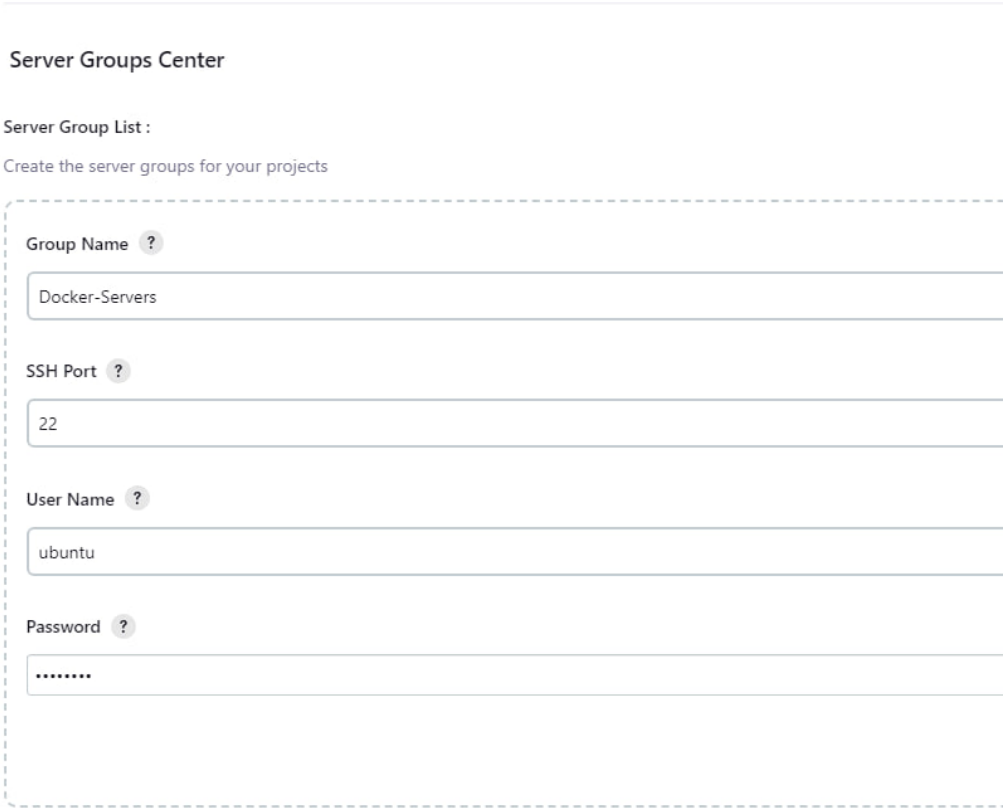

Рисунок 3.35 – Додавання сервер групи Docker

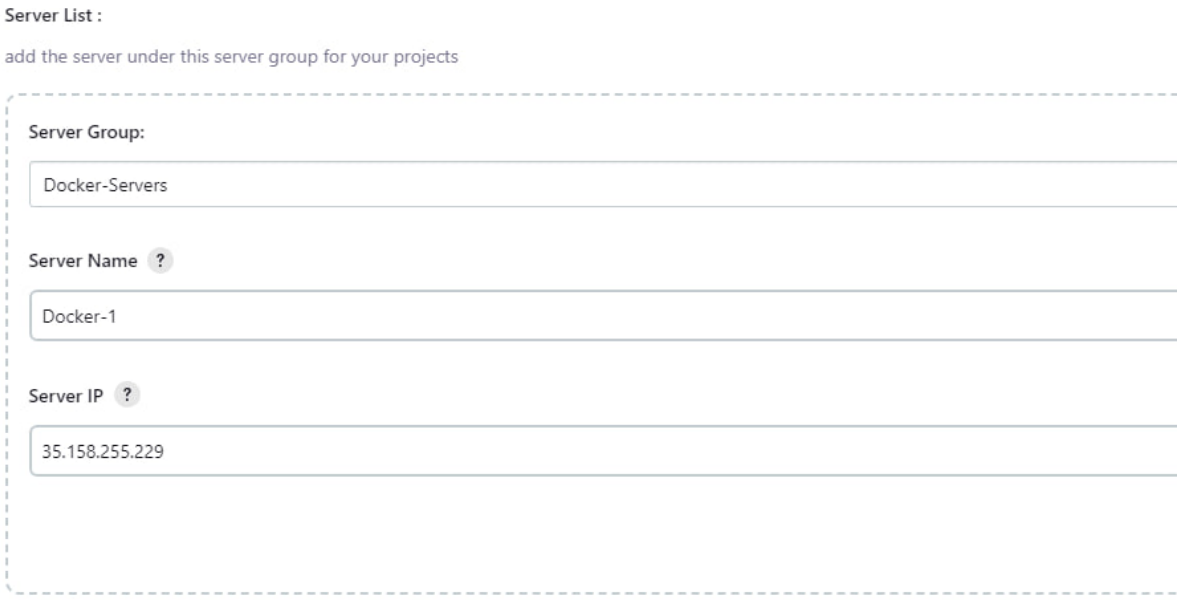

Рисунок 3.36– Додавання сервера Docker

Тепер для тесту сумісності системи додамо в репозиторій файл test.txt. Після чого починається автоматична збірка (Рисунок 3.27).

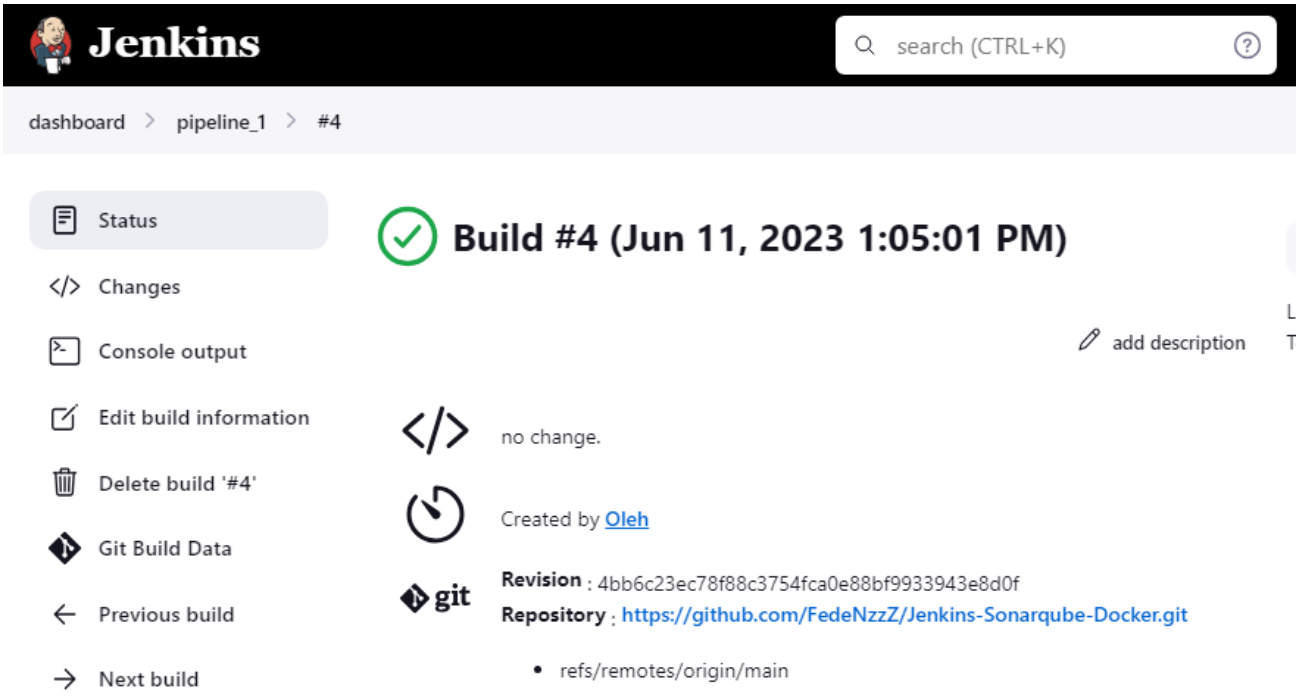

Рисунок 3.37 – Автоматична збірка з доданим файлом test.txt

Перевіряємо наявність нового файлу на Docker сервері (Рисунок 3.38).

Тепер у нашому репозиторії потрібно додати файл Dockerfile та додати туди код (Рисунок 3.39), для копіювання прийнятого репозиторію в папку (website), на основі якої буде створений Docker Image (образ контейнеру, котрий буде доступний за портом 8085). Для цього потрібно додати команду в Jenkins, яка буде створювати контейнер на основі змісту папки «website» (Рисунок 3.40).

```
root@docker:/home/ubuntu# ll
total 36
drwxr-x--- 4 ubuntu ubuntu 4096 Jun 11 13:05 ./
drwxr-xr-x 3 root   root   4096 Jun 11 03:01 ../
-rw------- 1 ubuntu ubuntu   60 Jun 11 10:38 .Xauthority
-rw------- 1 ubuntu ubuntu   28 Jun 11 13:05 .bash_history
-rw-r--r-- 1 ubuntu ubuntu  220 Jan  6  2022 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Jan  6  2022 .bashrc
drwx------ 2 ubuntu ubuntu 4096 Jun 11 10:38 .cache/
-rw-r--r-- 1 ubuntu ubuntu  807 Jan  6  2022 .profile
drwx------ 2 ubuntu ubuntu 4096 Jun 11 03:01 .ssh/
-rw-r--r-- 1 ubuntu ubuntu    0 Jun 11 10:40 .sudo_as_admin_successful
-rw-rw-r-- 1 ubuntu ubuntu    0 Jun 11 13:05 test.txt
root@docker:/home/ubuntu#
```

Рисунок 3.38 – Перевірка наявності доданого файлу

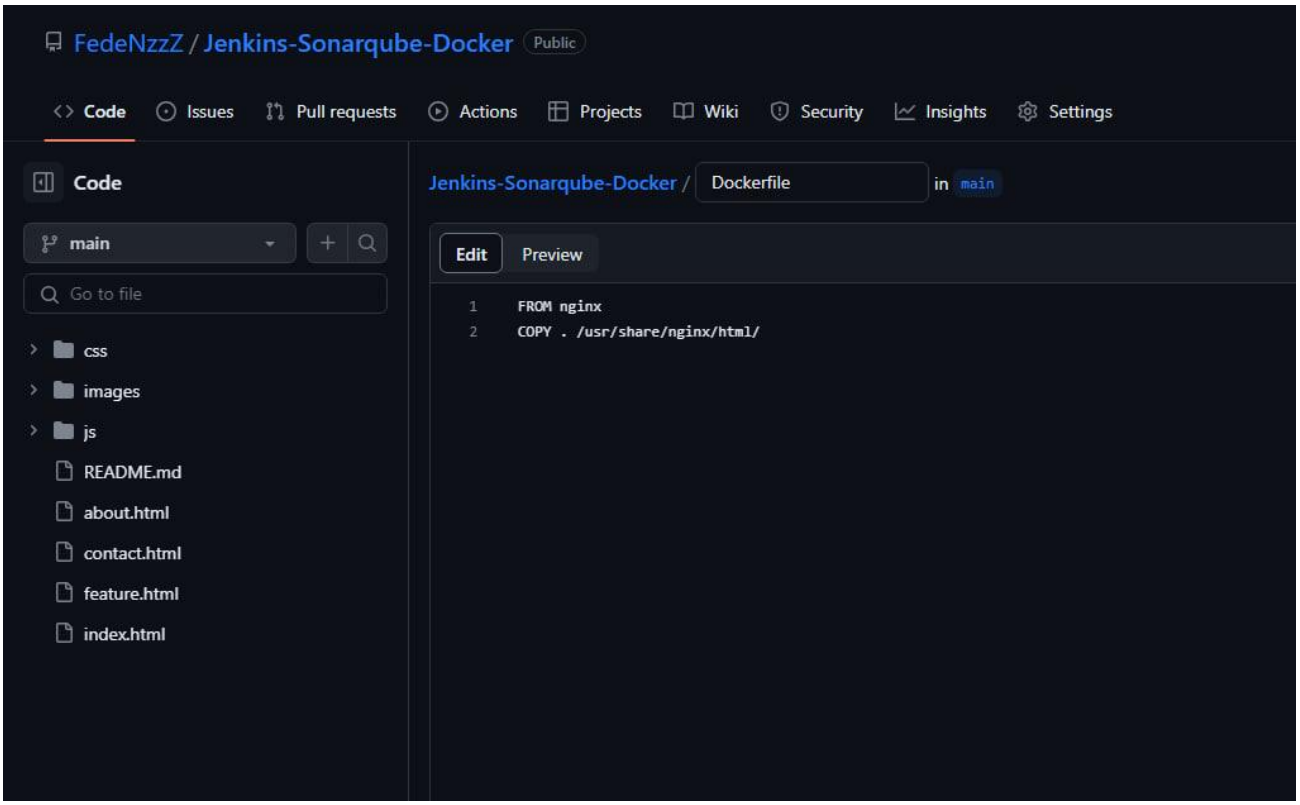

Рисунок 3.39 – Створення файлу Dockerfile

Тепер додаємо відповідну команду (див. Додаток Б) в Jenkins (Рисунок 3.40), для копіювання отриманого репозиторію в папку «website».

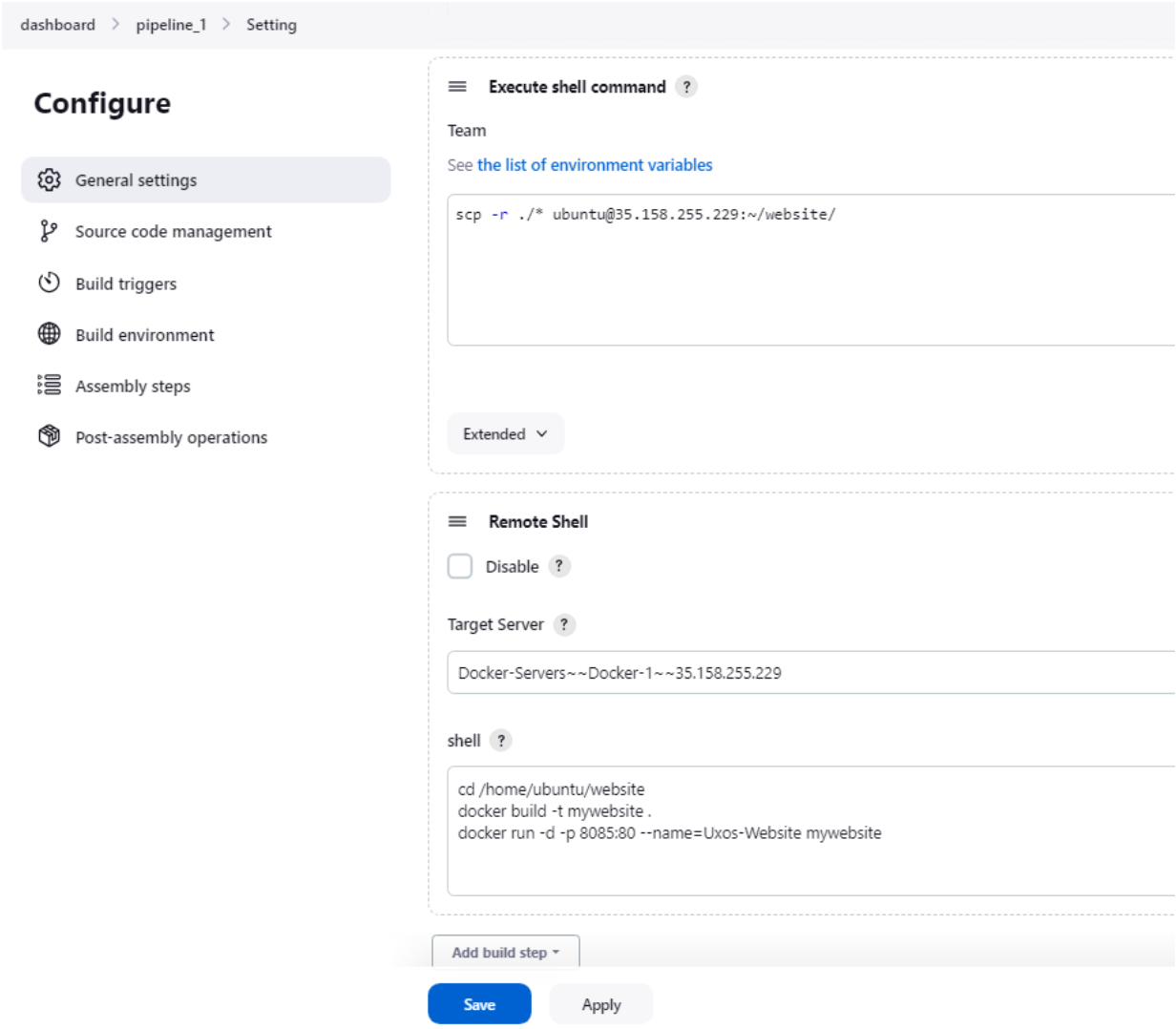

Рисунок 3.40 – додавання програми для копіювання репозиторію в папку “website”

Тепер створимо образ докера, і створимо контейнер на його основі вмісту папки «website», додаємо поточного користувача в групу docker та даємо доступ до виконання команд супер-користувача (див. Додаток Г).

У результаті виконання роботи, піля білду (Рисунок 3.41), фіксування доданих змін, контейнер повинен автоматично розгорнутися з Docker Image на вказаному порті, доступ до якого, у нашому випадку, доступний через зовнішній IP та порт:8085 (Рисунок 3.41, Рисунок 3.42).

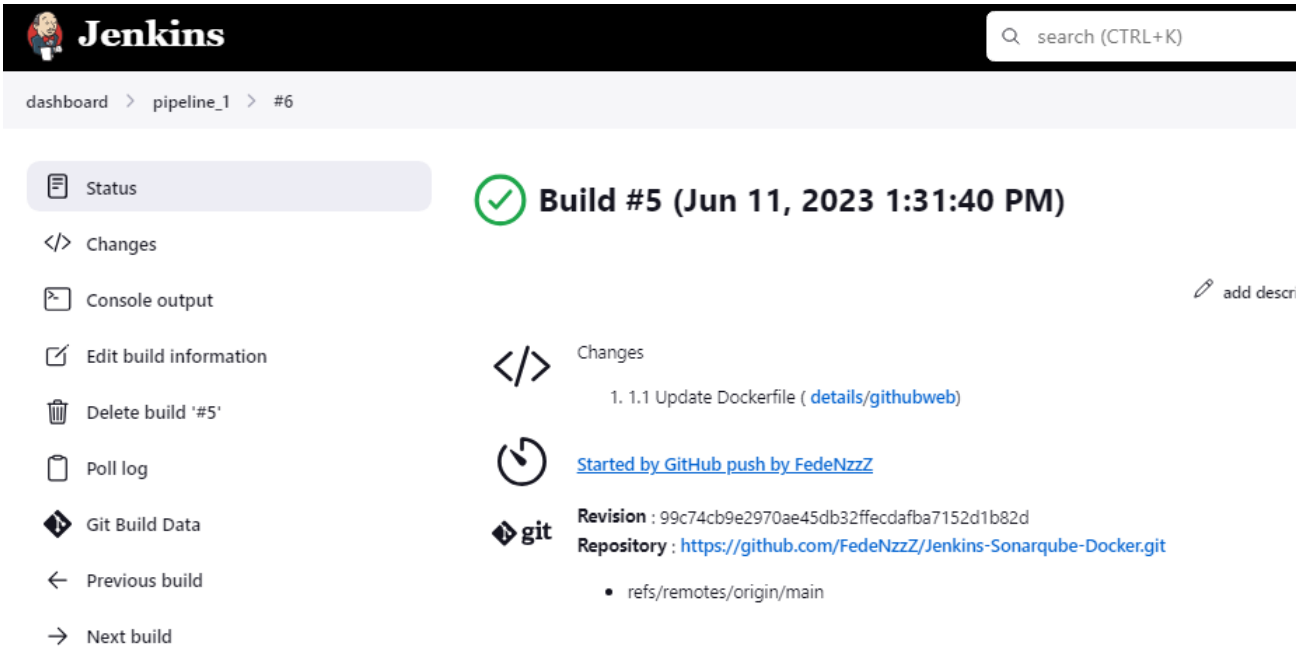

Рисунок 3.41 – Автоматичний bild

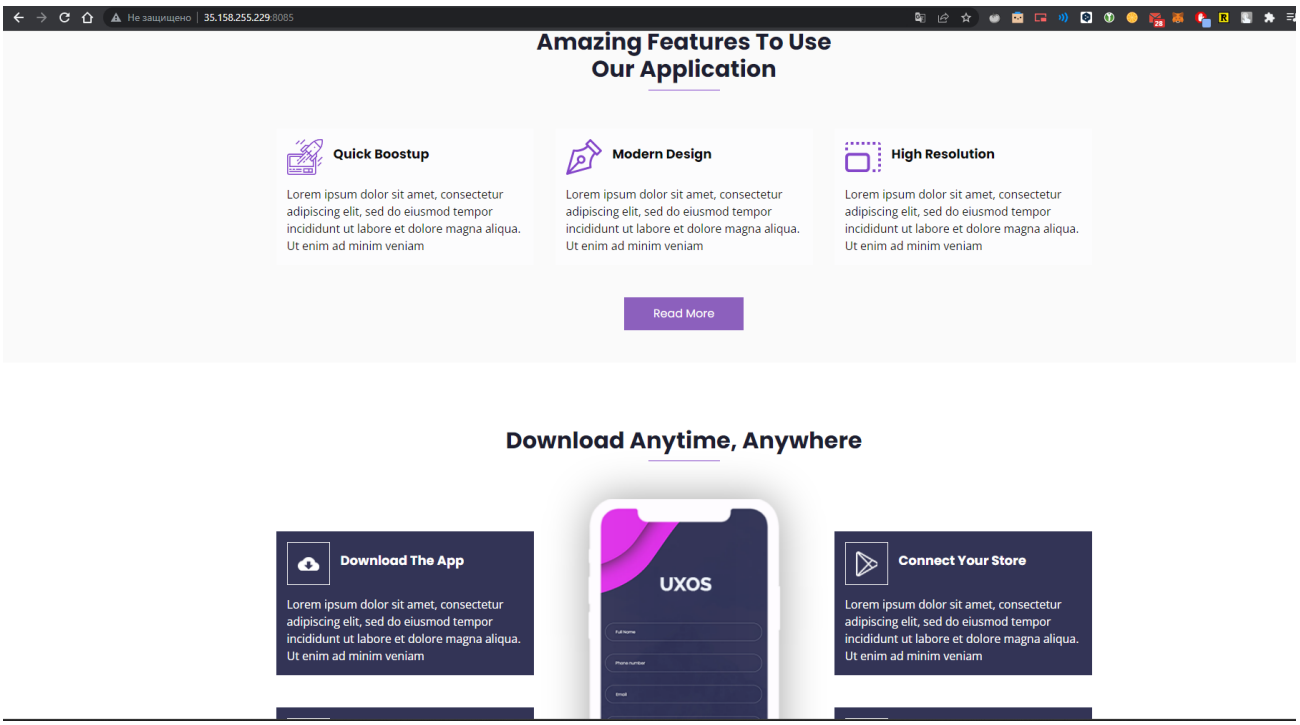

Рисунок 3.42 – Розгорнутий веб-сайт

ВИСНОВОК

Отже, це дослідження проводилося в різних аспектах розробки програмного забезпечення, зосередившись на ключових інструментах і практиках, що використовуються в сучасних середовищах розробки. Теоретичне дослідження забезпечило всебічне розуміння ключових концепцій та методологій, а практична реалізація показала практичне застосування та переваги використання саме методології DevOps.

Результат даної роботи підкреслив важливість систем контролю версій. Розподілена природа Git'у, можливості розгалуження та інтеграція з іншими інструментами роблять його важливим компонентом у сучасних робочих процесах розробки програмного забезпечення.

Практики безперервної інтеграції та безперервної доставки (CI/CD) були обговорені як життєво важливі для прискорення циклів розробки, покращення якості коду та полегшення безперебійного розгортання. Jenkins, сервер автоматизації з відкритим вихідним кодом, був визначений як потужний інструмент для автоматизації різних завдань у конвеєрі CI/CD, таких як створення, тестування та розгортання додатків. Його розширюваність, можливості інтеграції та підтримка розподілених збірок роблять його кращим вибором серед розробників.

Системи статичного аналізу коду, на прикладі SonarQube, були відзначені за їхню здатність виявляти проблеми в коді, вразливості та дотримання стандартів кодування. Комплексний аналіз SonarQube, дієві рекомендації та інтеграція з робочими процесами розробки сприяють підтримці якості та безпеки коду протягом усього процесу розробки програмного забезпечення.

Контейнеризація з'явилася як трансформаційна технологія для пакування та розгортання додатків в ізольованих та портативних середовищах. Переваги

Docker включають ефективне використання ресурсів, спрощене управління залежностями, масштабованість та безперешкодну інтеграцію з робочими процесами CI/CD. Розгалужена екосистема та інструменти оркестрування ще більше підвищують його корисність та застосовність у різноманітних сценаріях розробки.

Крім того, дослідження визнало важливість хмарних провайдерів у сучасній розробці програмного забезпечення. AWS з її сервісом Amazon EC2 була відзначена як провідна хмарна платформа, що пропонує масштабовані обчислювальні ресурси, гнучкість конфігурації, високу доступність та безперешкодну інтеграцію з іншими сервісами AWS. Цінова модель AWS, що передбачає оплату по мірі використання, та широкий спектр послуг роблять AWS найкращим вибором для розміщення та управління додатками в хмарі.

Подальші дослідження в цій галузі можуть дослідити нові тенденції та технології в розробці програмного забезпечення. Це включає в себе досягнення в контейнеризації, такі як Kubernetes для оркестрування контейнерів, і прийняття безсерверних обчислювальних моделей, які пропонують підвищену масштабованість і зниження витрат на управління інфраструктурою. Крім того, вивчення інтеграції штучного інтелекту і машинного навчання в робочі процеси розробки може дати цінну інформацію щодо автоматизації тестування, аналізу коду і прийняття рішень.

Насамкінець, дослідження надало вичерпний огляд основних інструментів та практик у розробці програмного забезпечення, продемонструвавши їхню важливість та практичне застосування. Інформація, отримана з цього дослідження, слугує основою для подальших досліджень і розвідок у цій динамічній галузі, що постійно розвивається.

У ході виконання кваліфікаційної роботи було виконано такі завдання:

1. Створення локального репозиторію

2. Клонування змісту локального репозиторія на віддалений репозиторій

3. Налагодження «конвеєра» автоматичної збірки та аналізу кода на помилки

4. Автоматичне створення ізольованого контейнера з проєктом (веб-сайтом), при успішній збірці.

Надалі планується дослідити такі технології як оркестрування контейнерами для автомичного масштабування, управління декількома контейнерами, а саме Kubernetis, інструменти «інфраструктури як коду», а саме Terraform, та запро

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Techslang. What is Containers-as-a-Service? – Definition by Techslang. Techslang – Tech Explained in Simple Terms. URL: https://www.techslang.com/definition/what-is-containers-as-a-service/ (дата звернення: 12.04.2023).

2. What is DevOps? - DevOps Models Explained - Amazon Web Services (AWS). Amazon Web Services, Inc. URL: https://aws.amazon.com/devops/what-is-devops/?nc1=h_ls (дата звернення: 13.04.2023).

3. What is DevOps? | Atlassian. Atlassian. URL: https://www.atlassian.com/devops (дата звернення: 13.04.2023).

4. Merkow M. S. Securing DevOps. Practical Security for Agile and DevOps. Boca Raton, 2021. С. 127–137. URL: https://doi.org/10.1201/9781003265566-10 (дата звернення: 14.04.2023).

5. Heriyanti F., Ishak A. Design of logistics information system in the finished product warehouse with the waterfall method: review literature. IOP Conference Series: Materials Science and Engineering. 2020. Т. 801. С. 012100. URL: https://doi.org/10.1088/1757-899x/801/1/012100 (дата звернення: 15.06.2023).

6. Chacon S., Straub B. Git and Other Systems. Pro Git. Berkeley, CA, 2014. С. 307–356. URL: https://doi.org/10.1007/978-1-4842-0076-6_9 (дата звернення: 21.04.2023).

7. Somasundaram R. Git: Version control for everyone. Packt Publishing, 2013. 180 с.

8. Code of Conduct Conversations in Open Source Software Projects on Github | Proceedings of the ACM on Human-Computer Interaction. Proceedings of the ACM on Human-Computer Interaction. URL: https://doi.org/10.1145/3449093 (дата звернення: 19.04.2023).

9. Haack P., Guthals S. GitHub for Dummies. Wiley & Sons, Incorporated, John, 2019. 368 с.

10. Leszko R. Continuous Delivery with Docker and Jenkins: Create secure applications by building complete CI/CD pipelines, 2nd Edition. Packt Publishing, 2019. 350 с.

11. McAllister J. Mastering Jenkins. Packt Publishing, 2015. 334 с.

12. Campbell G. A., Papapetrou P. P. SonarQube in Action. Manning Publications, 2013. 392 с.

13. SonarQube 10.1. SonarQube 10.1. URL: https://docs.sonarqube.org/latest/ (дата звернення: 11.05.2023).

14. Cook J. Docker. Docker for Data Science. Berkeley, CA, 2017. С. 29–47. URL: https://doi.org/10.1007/978-1-4842-3012-1_2 (дата звернення: 20.05.2023).

15. Docker Docs: How to build, share, and run applications. Docker Documentation. URL: https://docs.docker.com/ (дата звернення: 20.05.2023).

16. IIBA. A Guide to the Business Analysis Body of Knowledge. International Institute of Business Analysis, 2015. 512 с.

17. Lunn K. Software Development Life Cycle. Software Development with UML. London, 2003. С. 53–68. URL: https://doi.org/10.1007/978-0-230-80419-7_5 (дата звернення: 10.06.2023).

18. Blokdyk G. International Software Testing Qualifications Board Certified Tester a Complete Guide - 2020 Edition. Emereo Pty Limited, 2020. 324 с.

19. Chacon S., Straub B. Git and Other Systems. Pro Git. Berkeley, CA, 2014. С. 307–356. URL: https://doi.org/10.1007/978-1-4842-0076-6_9 (дата звернення: 21.04.2023).

19. Van Vugt S. Installing Linux. Beginning the Linux Command Line. Berkeley, CA, 2015. С. 361–382. URL: https://doi.org/10.1007/978-1-4302-6829-1_15 (дата звернення: 15.06.2023).

20. Henning S., Hasselbring W. The Titan Control Center for Industrial DevOps analytics research. Software Impacts. 2021. T. 7. C. 100050. URL: https://doi.org/10.1016/j.simpa.2020.100050 (дата звернення: 18.06.2023).

21. What is Amazon EC2? - Amazon Elastic Compute Cloud. URL: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html (дата звернення: 19.06.2023).

ДОДАТОК А

```
$ git init
$ git config --global user.email "jotarokujosama@gmail.com"
$ git config --global user.name "Oleh Pimenov"
$ git remote add origin https://github.com/FedeNzzZ/Jenkins-Sonarqube-Docker.git
$ git branch -M main
$ git status

$ git add .
$ git commit -m "Version 1.0"
$ git push -u origin main
```

ДОДАТОК Б

Linux (Jenkins):

```
sudo apt update

curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | sudo tee \
  /usr/share/keyrings/jenkins-keyring.asc > /dev/null
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
sudo apt install openjdk-11-jre
systemctl status jenkins
```

Jenkins WEB:

```
scp -r ./* ubuntu@35.158.255.229:~/website/
```

ДОДАТОК В

sudo apt update

sudo apt install openjdk-17-jre

wget https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-10.0.0.68432.zip?_gl=1*z0o5a6*_gcl_au*OTY1NTM1MTI1LjE2ODY0NzU0NjQ.*_ga*NTk3NjQ2NDkzLjE2ODY0NzU0NjQ.*_ga_9JZ0GZ5TC6*MTY4NjQ3NTQ2My4xLjEuMTY4NjQ3NjA3NC41OS4wLjA.

unzip sonarqube-10.0.0.68432.zip

cd sonarqube-10.0.0.68432/

cd bin

cd linux-x86-64/

./sonar.sh

ДОДАТОК Г

```
sudo hostnamectl set-hostname docker
sudo su
nano /etc/ssh/sshd_config
ssh-copy-id ubuntu@35.158.255.229
ll
sudo usermod -aG docker ubuntu
newgrp docker
```