МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет Іфокомунікацій
Кафедра Інформаційно-мережної інженерії

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

другий (магістерський)
(рівень вищої освіти)

Будування системи освіти студентів на основі сучасних технологій автоматизації

Виконав:
студент 2 курсу, групи ІМІм-21-2
Степанов О. О.
(прізвище, ініціали)

Спеціальність	172 – Телекомунікації та радіотехніка
Тип програми	Освітньо-наукова
Керівник	доц. каф.ІМІ Скорик Ю. В.

(посада, прізвище, ініціали)

Допускається до захисту

Зав. Кафедри _____ В.М. Безрук

2023 р.

Не містить відомостей, заборонених до відкритого публікування

Студент_____/ *О. О. Степанов*/

Керівник _____/ *Ю. В. Скорик*/

Харківський національний університет радіоелектроніки

Факультет _____Інфокомунікацій_____

Кафедра _____Інформаційно-мережної інженерії_____

Рівень вищої освіти _____другий (магістерський)_____ _____

Спеціальність _____172 – Телекомунікації та радіотехніка_____

(код і повна назва)

Тип програми _____освітньо-наукова_____

Освітня програма _____Інформаційно-мережної інженерії_____

(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

«___»_____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові _____Степанову Олександру Олександровичу_____

(прізвище, ім'я, по батькові)

1. Тема роботи ____«Будування системи освіти студентів на основі сучасних____ технологій автоматизації»_____

затверджена указом університету від « 27 » __04_____ 20_23_р. №398Ст

2. Термін подання студентом роботи до екзамінаційної комісії « 18 » ___травня____2023р.

3. Вихідні дані до роботи *проектування рішення удосконалення навчальної платформи навчання, Bash, Jenkins, GitHub, AWS.*_____

4. Перелік питань, що потрібно опрацювати в роботі

*Вступ*_____

*1.Концепція*_____

*2. Програмна реалізація рішення*_____

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускної кафедри) *Слайди у форматі Power (назва, мета і задачі роботи, концепція рішення, реалізація роботи студента, огляд системи автоматизованої перевірки, демонстрація роботи, перевірка системи виставлення оцінок, висновки)*

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз завдання та літературних джерел	06.04.2023	*виконано*
2	Огляд існуючих рішень	11.04.2023	*виконано*
3	Проектування рішення удосконалення навчальної платформи	14.04.2023	*виконано*
4	Оформлення пояснювальної записки	26.04.2023	*виконано*
7	Рецензування	21.05.2023	*виконано*
8	Занесення диплома в електронний архів	22.05.2023	*виконано*
9	Допуск до захисту у зав. кафедри	18.05.2023	*виконано*

Дата видачі завдання 1.04 20 23 р.

Студент _____
(підпис)

Керівник роботи _____ доцент каф. ІМІ Скорик Ю.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка 59 с., 15 джерел, 34 рис., 2 додатки.

Об’єкт роботи – навчальна платформа, яка автоматично приймає та перевіряє студентські роботи.

Метою цієї роботи є аналіз кількох важливих аспектів, що стосуються оцифрування та автоматизації процесів навчання, зокрема розглядаючи використання інструментів автоматизації в таких завданнях, як оцінювання студентів. Крім того, ми досліджуватимемо переважаючі тенденції та практики в IT-індустрії та досліджуватимемо їх застосування у сфері освіти. Нарешті, ми оцінимо ефективність цифрових систем навчання порівняно з традиційними методами, щоб оцінити потенційні переваги та недоліки, пов’язані з цим підходом.

Методи розробки базуються на таких технологіях як Jenkins, GitHub, AWS.

У результаті роботи пропонується удосконалення навчальної платформи Cloud Mentor, шляхом розробки рішення для перевірки студентських робіт. Удосконалення навчальної платформи Cloud Mentor дозволить вирішити проблеми, пов'язані з недостатньою продуктивністю викладача для перевірки робіт, що збільшить ефективність навчання та розширить спектр навчання.

AWS, JENKINS, BASH, GITHUB, НАВЧАЛЬНА ПЛАТФОРМА, PIPELINE, LINUX.

THE ABSRACT

Explanatory slip 59 p., 15 sources, 34 fig., 2 app..

The object of the work is an educational platform that provides opportunities for training and practical implementation of cloud services.

The purpose of this paper is to analyze several crucial aspects concerning the digitalization and automation of learning procedures, specifically addressing the utilization of automation tools in tasks like student assessment. Additionally, we will explore prevailing trends and practices in the IT industry and investigate their applicability in the field of education. Lastly, we will evaluate the effectiveness of digital learning systems in comparison to traditional methods, aiming to assess the potential benefits and drawbacks associated with this approach.

Development methods are based on such technologies as Jenkins, GitHub, AWS.

As a result of the work, it is proposed to improve the Cloud Mentor educational platform by developing a solution for checking student works. The improvement of the Cloud Mentor training platform will allow to solve the problems related to the insufficient productivity of the teacher to check the works, which will increase the effectiveness of training and expand the range of training.

AWS, JENKINS, BASH, GITHUB, НАВЧАЛЬНА ПЛАТФОРМА, PIPELINE, LINUX.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AWS (Amazon Web Services) – дочірня компанія Amazon.com, яка продає послуги платформи хмарних обчислень приватним особам, компаніям та урядам.

CI/CD – Continuous integration / Continuous deployment – це комбінація безперервної інтеграції та безперервного розгортання програмного забезпечення в процесі розробки.

ВСТУП

У сучасну цифрову епоху традиційні системи навчання стикаються з низкою проблем, особливо коли йдеться про роботу з великою кількістю студентів. обмеження вчителів-людей ускладнюють їм керування кількома завданнями одночасно, що може бути серйозною перешкодою для ефективного викладання та навчання. ця проблема стала ще більш гострою у зв’язку з глобальними подіями, які призвели до широкого впровадження дистанційного навчання.

Однак поширення цифрових методів дистанційного навчання відкрило нові можливості для освіти. ці методи пропонують багато переваг, включаючи можливість охопити більшу кількість студентів, незалежно від їхнього географічного розташування. вони також пропонують певну гнучкість і зручність, які просто неможливі при традиційному навчанні в класі.

У той же час прогрес в інформаційних технологіях призвів до розробки ряду засобів автоматизації та хмарних сервісів, які можуть допомогти подолати обмеження традиційних методів навчання. ці інструменти можуть допомогти підвищити надійність і ефективність систем навчання, а також полегшити вчителям керування великою кількістю учнів.

У даній роботі розглядатимуться деякі ключові проблеми, пов’язані з оцифруванням і автоматизацією процесів навчання, з особливим акцентом на використання інструментів автоматизації для таких завдань, як перевірка робіт студентів. ми також розглянемо деякі сучасні тенденції та практики в іт-індустрії та дослідимо, як їх можна застосувати до освіти. нарешті, ми порівняємо ефективність цифрових систем навчання з традиційними методами, щоб оцінити потенційні переваги та недоліки цього підходу.

КОНЦЕПЦІЯ

1.1 Проблематика

На теперішній час, в умовах дистанційного навчання зросла необхідність підвищення продуктивності роботи викладачів вищих навчальних закладів. Підвищення продуктивності здійснюється за допомогою прискорення процесу перевірки студентських робіт. Таке покращення дозволить викладачам перевіряти більше робіт за той же проміжок часу, або приділяти більше часу іншим активностям, таким як написання наукових статей тощо.

В силу специфіки рішення та загальної компетенції, перш за все необхідно допомогти викладачам, які працюють у сфері програмування та інших сфер інформаційних технологій, оскільки засоби, які будуть розглядатись у роботі спрямовані на перевірку програм.

Перед початком автоматизування процесів, необхідно визначити, що саме буде автоматизовано, для чого, та який результат планується отримати [1].

1.2 Пошук рішення та інструментів для реалізації

Засоби автоматичної перевірки програм вже існують, самим популярним серед них є SonarQube [2]. Цей продукт відповідає за огляд програмного коду та оцінює його якість, проте він не може оцінити якість роботи, яку робить програма.

На ринку програмного забезпечення взагалі дуже мало рішень, які реалізують перевірку чого-небудь програмного. І тим більше можливість знайти засіб, який дозволяє створювати задачі для студентів та перевірювати їх виконання являється близько-нульовою.

Проте, є багато засобів автоматизації процесів, які не залежні від того, який процес вони автоматизують. Одним з прикладів є Jenkins.

Jenkins - це дуже гнучкий і відомий інструмент CI/CD [3], створений для автоматизації різних завдань, пов'язаних із розробкою програмних проектів. Даний інструмент повністю написаний на Java, випущений під ліцензією Масачусетського технологічного інституту (MIT) та має потужний набір функцій, спрямованих на оптимізацію процесів, пов'язаних із збіркою, тестуванням, розгортанням, інтеграцією та випуском програмного забезпечення. Інструмент сумісний з різними

операційними системами, включаючи macOS, Windows та безліч дистрибутивів Linux, таких як OpenSUSE, Ubuntu та Red Hat. Крім того, існують пакети інсталяції Jenkins, призначені для різних платформ.

Команда Jenkins створила близько 1000 плагінів, які допомагають у безшовній інтеграції з різними технологіями. У скриптах можна використовувати системи аутентифікації, що дозволяє безпечно підключатися до захищених ресурсів та інших закритих систем.

Автоматизовані процеси у Jenkins можна виконувати моніторинг, дозволяючи користувачам стежити за прогресом на кожному етапі та оцінювати успішність кожного етапу без необхідності графічного інтерфейсу. Замість цього, термінальні можливості інструменту пропонують простий та ефективний спосіб відстеження прогресу.

Ще одним критичним плюсом є безкоштовність даного інструменту. Дженкінс має відкритий вихідний код та підтримку суспільства, що дозволяє йому стрімко нарощувати свій функціонал. Даний плюс не треба забувати, оскільки у випадку, якщо інструмент платний – то не факт що навчальний заклад зможе фінансувати використання даних інструментів, незалежно від тих переваг, яких він надає [4].

Беручи до уваги переваги інструменту Jenkins, у роботі надалі вирішено застосовувати його. Тепер потрібно зрозуміти, як його можемо застосувати до студентських робіт та які задачі він зможе автоматично перевіряти.

1.3 Побудування моделі

Наступний етап – розробка концепції [5]. Даний процес має найбільшу свободу дії, оскільки нема чітких рамок в одначас такий процес може не сподобатися через невизначеність того, що отримаємо в кінці.

Проте, оскільки є конкретна ціль та є інструмент, який допомогже досягти цю ціль – то можна почати створювати схематичний опис алгоритму роботи проекту. Перш за все необхідно уявити студента та його роботу. Потім потрібно зрозуміти, хто перевірятиме роботу та як. І наприкінці необхідно визначитися, що буде в результаті.

Усі ці міркування вилилися у концепт, представлений нижче (рис.1.1):

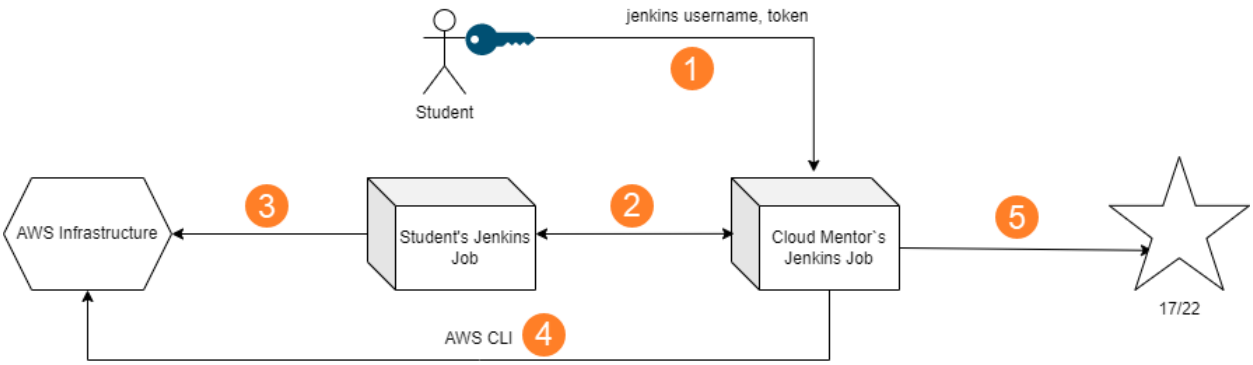

Рисунок 1.1 – Концепція роботи системи перевірки студентських завдань

На рис. 1.1 зображені наступні інтелектуальні об’єкти:

– студент із ключем, який символізує дані облікових записів, такі як логін та пароль до свого програмного середовища, включаючи Jenkins;
– Student’s Jenkins Job – це робота, яку виконуватиме студент, користуючись Дженкінсом;
– AWS Infrastructure – це варіант одного із завдань, які викладачі надають студентам у сфері інформаційних технологій. Таким завданням може бути побудування серверної мережі за допомогою хмарних провайдерів, таких як Amazon Web Services;
– Cloud Mentor’s Jenkins Job – це система, яка буде автоматично перевіряти роботу студента;
– Зірка – це представлення кінцевого результату, який ми отримаємо. Кожна робота повинна бути оцінена, тому кінцевим результатом буде оцінка роботи студента.

Значення оцінки «17/22» було обрано довільно, на даний момент ще не прийняте остаточне рішення стосовно того, якої розмірності повинна бути оцінка. Поки що оцінка формується абсолютною величиною, яка дорівнює кількості пройдених тестів (17 пройдених тестів), відносно до загальної кількості тестів, яка дорівнює 22.

Взаємодія між даними інтелектуальними об’єктами формується наступним чином:

– Студент отримує чітко поставлене завдання, яке може представляти собою будування сервреної мережі в інфраструктурі від провайдеру Amazon. Студент виконує завдання, створює на його основі автоматизований процес у Jenkins та здає його на перевірку до системи, яка умовно названа Cloud Mentor. Студент передає системі дані облікового запису Дженкінсу,

створеного спеціально для Клауд Ментору. До цих даних можуть відносяться IP-адреса, ім’я користувача, пароль або токен тощо;
- Далі Дженкінс викладача запускає на виконання автоматизований процес студента;
- Автоматизований процес виконує свою роботу, яка умовно є створенням інфраструктури;
- Дженкінс викладача робить перевірку інфраструктури, яку створив студент. Робиться це шляхом відправки запитів до AWS для перевірки наявності тих чи інших ресурсів та їх параметрів. Для виконання такої перевірки також потрібні будуть дані для аутентифікації системи.
- Після виконання перевірок формується підсумкова оцінка роботи студента.

Після визначення критеріїв вдалої реалізації, висунуті вимоги до засобів рішення та знайдені відповідні інструменти, які спроможні реалізувати та автоматизувати процеси виконання та перевірки завдань, далі можна приступати до побудови налагодженого процесу.

2 ПРОГРАМНА РЕАЛІЗАЦІЯ РІШЕННЯ

2.1 Робота студента

Перш за все потрібно розробити умови завдання, яке отримуватиме студент. Оскільки чітко поставлене завдання надає можливість поставити чіткі механізми перевірки, даний етап вважається першочерговим.

Кафедра інформаційно-мережної інженерії харківського національного університету радіоелектроніки йде вслід за новими технологіями, які розроблюють передові світові компанії. Останні декілька років викладачі кафедри вчать студентів користуватися засобами побудови серверних кластерів у так званих «хмарах» [6]. До таких рішень відноситься вже раніше згаданий Amazon Web Services [7]. На кафедрі вже мається досвід з навчання та перевірки завдань, які стосуються AWS, тому було прийняте рішення черпати ідеї у завданнях, які надають викладачі кафедри.

Так як ця робота зосереджена саме на автоматизації процесів виконання та перевірки завдань, то розроблятись складне завдання для студента не буде, а буде взято готове завдання та рішення цього завдання.

Після консультації з викладачами кафедри, було розроблена така ідея для завдання:

- Користуючись сервісом AWS S3 (Simple Storage Service), створити сутність, яка називається «S3 Bucket».
- S3 Bucket повинен бути створений у регіоні us-east-1, (Північна Вірджинія, США).
- S3 Bucket повинен бути публічний доступ, тобто щоб до нього могли дістатися з усього Інтернету.
- Зробити зі створеного бакету «S3 Static Web Site». Тобто створити зі сховища статичний веб сайт. Який повинен містити назву створеного бакету.
- Усі кроки повинні бути автоматизованими, виконуватися не власноруч, а через інструмент CI/CD, який називається Jenkins.

Таким чином, метою завдання є не тільки опановування сервісів від Amazon, а ще й отримання навичок автоматизації процесів завдяки інструменту Jenkins [7].

2.2 Механізм верифікації

Далі потрібно оглянути завдання з точки зору потенціальних моментів, які варто перевіряти, перш ніж стверджувати чи було виконано завдання чи ні.

Так наприклад в даний момент можна стверджувати що кінцевим результатом виконаного завдання є статичний веб сайт, тому доречно буде перевірити чи є працює сайт.

Далі, якщо в умовах говориться про виконання роботи посередньо через Jenkins, то тоді можна говорити про перевірки, які стосуються даного програмного середовища.

Перш за все приходить ідея перевірити чи запускається автоматизований процес, чи виконується він вдало, та чи працює він як треба.

Таким чином, у нас вже є певні місця де доречно зробити перевірку. При тому більшість із них поки що скоріше стосується середовища ніж самого завдання, але тим не менш, наразі не стоїть ціль зробити багато перевірок. Мета – розробити алгоритм, який передбачає запуск таких перевірок [7].

2.3 Перевірка працездатності

На даний момент вже нічого не заважає розпочати розробку автоматизації. Як і у випадку з розробкою концепції, першим ділом необхідно розпочати зі студентської роботи.

Хоча будемо працювати в основному із Дженкінсом, на практиці бажано побачити автоматично створений S3 Bucket, у якому виконані певні налаштування.

Візьмемо за приклад власний обліковий запис в AWS та подивимося на сервіс S3. Якщо перекласти назву сервісу з англійської на українську – то стає зрозуміло, що даний сервіс позиціонується як примітивне сховище даних, зокрема файлів. Простота даного сервісу стає ще більш зрозумілою, якщо ознайомитися з документацією даного сервісу, де говориться, що сервіс навіть не підтримує файлову ієрархію, тобто він не може реалізувати вкладеність.

Хоча сервіс показує, що у бакеті можна створити папки, насправді це усього лишь візуальне відображення. Усі папки в S3 – то лише префікси, які додаються до назви файлів. Тому сутності сервісу й називаються бакети, що в переводі з англійської означає відро. Таким чином S3 Bucket являється несортованим сховищем даних. Варто зазначити, що даний сервіс не має регіональних обмежень. Це означає, якщо бакет був створений в регіоні us-east-1 (Північна Вірджинія), а ми

раптово вирішили переключитися на регіон eu-central-1 (Франкфурт) – то надалі зможемо переглядати бакети з Північної Вірджинії у головному меню S3. Сервіс простого сховища являється одним з найперших та найпопулярніших сервісів, які були створені у Amazon Web Services. Дане сховище має доволі гнучку тарифікацію, яка залежить від об'єму даних, інтенсивності користування. Різні тарифи пропонують різні швидкості доступу до фалів а також різний рівень надійності, який можна досягти шляхом впровадження функції резервного копіювання файлів на серверах Amazon з різних регіонів.

Даний обліковий запис використовувався раніше, тому можемо побачити як виглядають бакети. На рис. 2.1 можна побачити їх регіон, налаштування доступу, дату створення. За бажанням, можна побачити більше даних, якщо виконати певне налаштування.

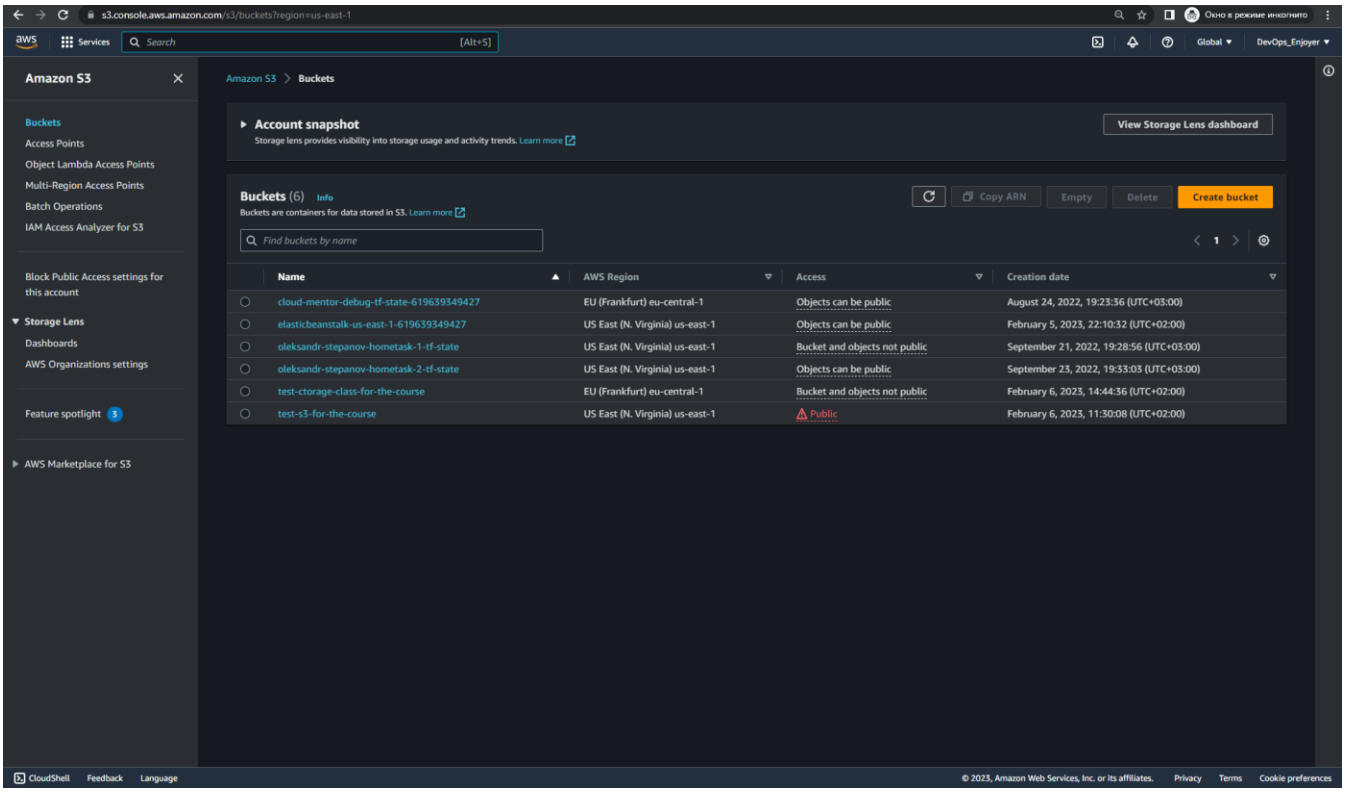

Рисунок 2.1 – Головне меню сервісу S3

На рис. 2.2 зображено, як файли зберігаються в бакеті, їх можна завантажувати в бакет, завантажувати з бакету та видаляти.

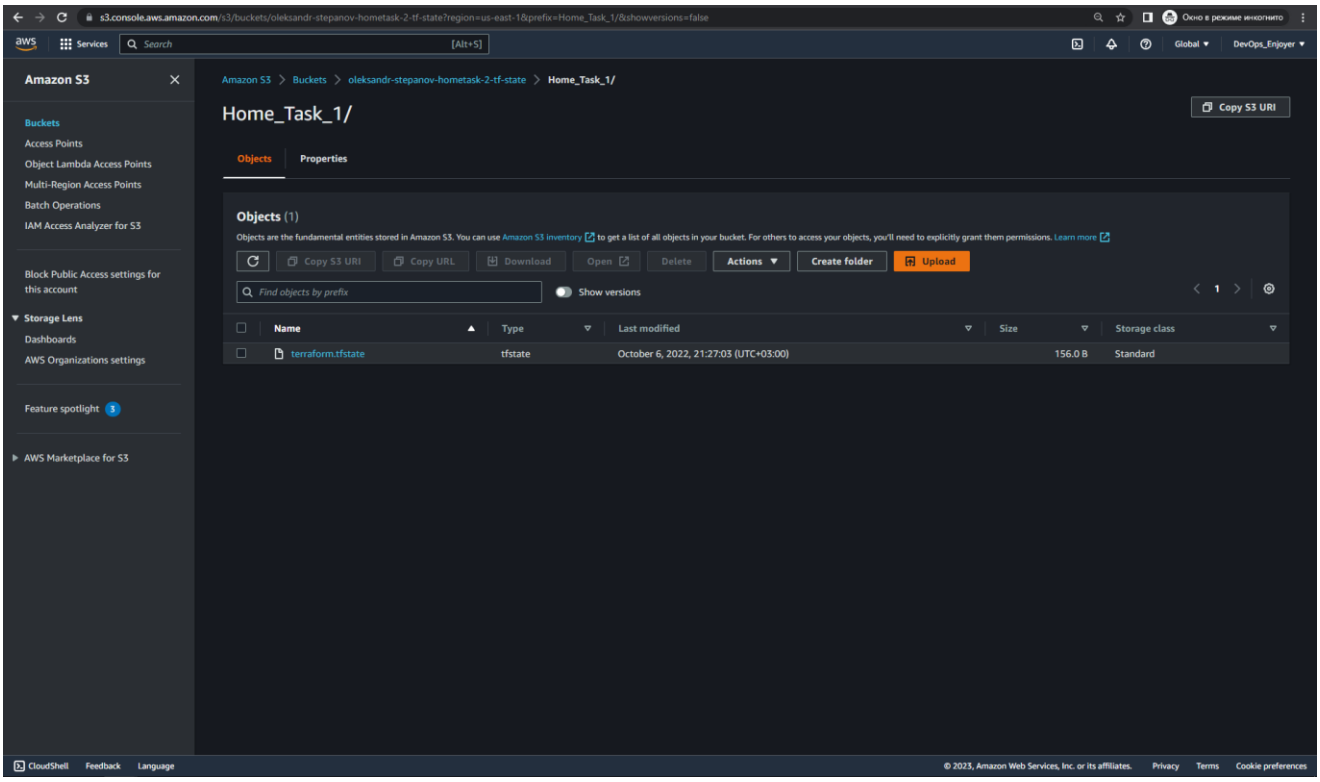

Рисунок 2.2 – Вигляд файлів всередині S3 бакету

І хоча вже перелічино основні особливості сервісу раніше, залишилася ще одна корисна функція, яку можна побачити в налаштуваннях бакету. Вона називається Static website hosting. Ця функція дозволяє зробити з бакету статичний вебсайт (рис. 2.3). Для цього потрібно завантажити файл у форматі «.html» та вказати бакету користуватися саме їм. Зазвичай цей файл називається «index.html». Також можна додати «error.html» файл, який буде показувати бажану інформацію якщо ресурс, на який було здійснено полсилання, не існує, проте це опціонально. Також потрібно зробити бакет публічним, щоб кожен зміг переглянути сторінку.

Червоною рамкою обведена опція «Static website hosting». Коли опція налаштована та активна, вона містить URL, по якому можна переглянути «index.html» файл. Зараз вона неактивна, проте в завданні студент повинен бути створити публічний бакет та налаштувати «Static website hosting».

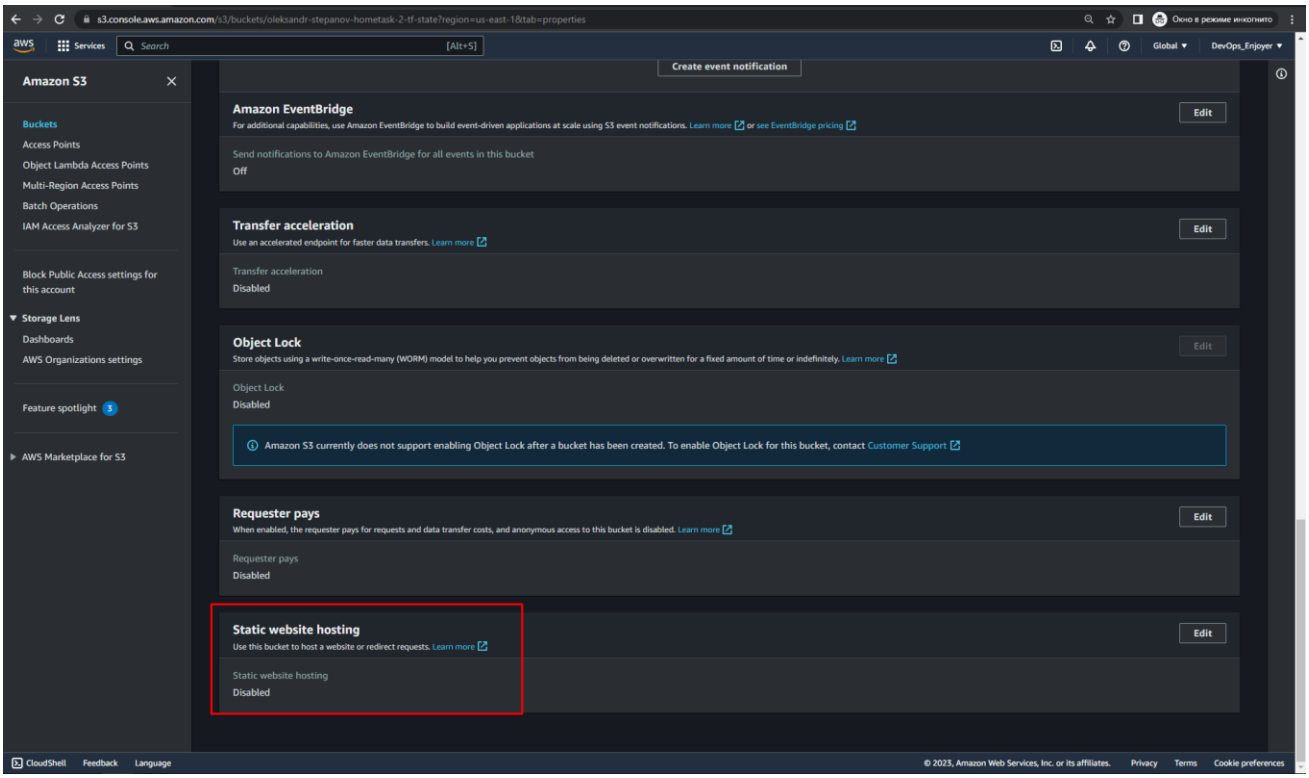

Рисунок 2.3 – Опція «Static website hosting»

Зробити це можна використовуючи команди, які виконуються у терміналі. AWS підтримує роботу в терміналі через утиліту «AWS CLI». Дана утиліта існує у версіях для Linux, MacOS Windows. Після встановлення утиліти, треба налаштувати її шляхом вказівки access key, secret access key, регіона за замовчуванням та формату виводу даних. Access key та secret access key можна зробити в сервісі AWS IAM (Identity and Access Management). Після налаштування можна виконувати AWS CLI команди. Майже будь-які дії, які робляться у AWS Management Console, можна зробити в терміналі, а то й більше. Так наприклад AWS S3 Glacier, один із підсервісів AWS, який відповідає за зберігання даних за найдешевшим тарифом, на даний момент керується лише через AWS S3 API, тому веб версія не є самодостатнім варіантом користування усіма послугами від Amazon [8].

Реалізацію виконання студентського завдання зі створення S3 Static website hosting можна побачити на рис. 2.4.

SecretiveRabbit test — Latest commit 53e77a0 last week — History

1 contributor

29 lines (28 sloc) | 1.29 KB — Raw | Blame

```
#!/bin/bash
aws s3api create-bucket --bucket alexander-stepanov123 --region us-east-1
echo '<!DOCTYPE html><html><head><title>My S3 Bucket</title></head><body><h1>Welcome to my S3 Bucket</h1><p>The name of this bucket is: <strong>alexander-stepanov123</strong><
echo '<!DOCTYPE html><html><head><title>404 - Not Found</title></head><body><h1>Oops! Something went wrong.</h1><p>The requested resource was not found on this server.</p></bo

cat > bucket-policy.json << EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": ["s3:GetObject"],
      "Resource": "arn:aws:s3:::alexander-stepanov123/*"
    }
  ]
}
EOF
sleep 7
aws s3 cp index.html s3://alexander-stepanov123
aws s3 cp error.html s3://alexander-stepanov123
aws s3api put-public-access-block --bucket alexander-stepanov123 --public-access-block-configuration BlockPublicAcls=false,IgnorePublicAcls=false,BlockPublicPolicy=false,Restr
aws s3 website s3://alexander-stepanov123/ --index-document index.html --error-document error.html
sleep 5
aws s3api put-bucket-policy --bucket alexander-stepanov123 --policy file://bucket-policy.json
rm index.html
rm error.html
rm bucket-policy.json
```

Рисунок 2.4 – Скрипт створення «Static website hosting» використовуючи команди AWS CLI

В силу специфіки розробленого алгоритму скрипт було покладено в публічний репозиторій на GitHub [8].

Даний скрипт спочатку сворює бакет з ім’ям «alexander-stepanov123» за допомогою команди «aws s3api create-bucket», потім створює «index.html» та «error.html» файли, створює файл, який містить налаштування політики користування бакетом, щоб можна було переглядати файли в бакеті, після цього скрипт завантажує файли «index.html» та «error.html» в бакет, потім він робиться публічним командою «aws s3api put-public-access-block». Після чого включається «Static website hosting», застосовується політика, яка дозволяє перегляд файлів з даного S3 бакету та видаляються файли, таким чином виконуючи чистку після роботи скрипта.

Перевірити роботу скрипта можна шляхом завантаження файлу та виконання його локально, попередньо не забувши налаштувати AWS CLI. Також треба надати скрипту права на виконання, інакше він не запуститься. Окрім цього, треба мати на увазі, що назва бакету повинна бути унікальна на весь світ, тому якщо бакет не створюється через те, що такий вже існує – тоді треба змінити його назву у скрипті на іншу.

Після виконання скрипта, що займає близько 20 секунд, можна оновити сторінку з бакетами та переконатися, що бакет з назвою «alexander-stepanov123» був створений публічним та у регіоні «Південна Вірджинія» (рис. 2.5)

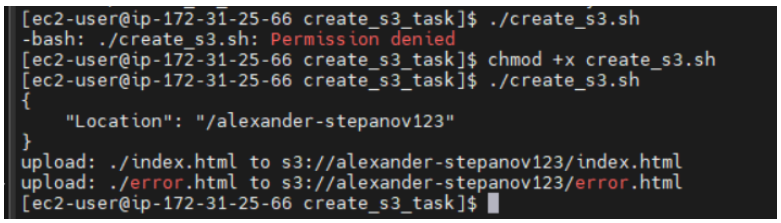

Рисунок 2.5 – Запуск скрипта на виконання

Далі, якщо зайти у бакет – то можна переконатися, що файли «index.html» та «error.html» були створені та завантажені до S3. Тепер давайте перевіримо чи була ввімкнена та налаштована опція «Static website hosting» (рис. 2.6 – 2.9):

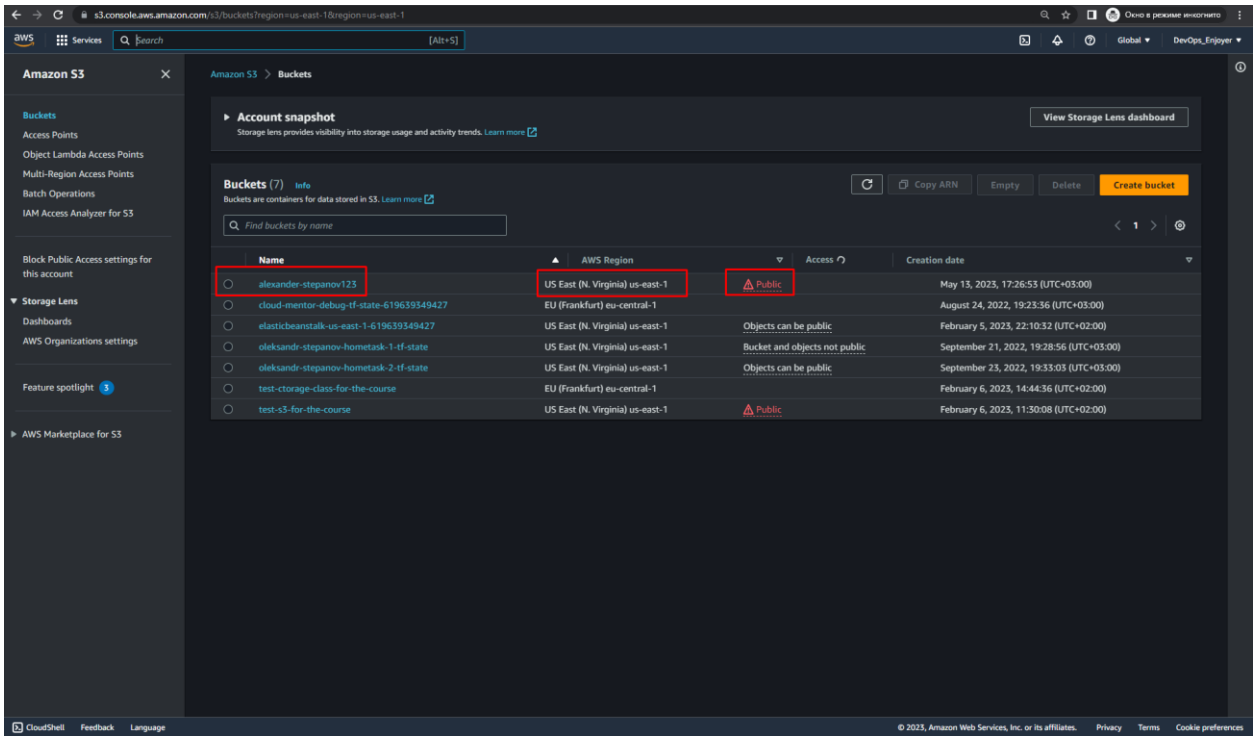

Рисунок 2.6 – Перевірка створення бакету

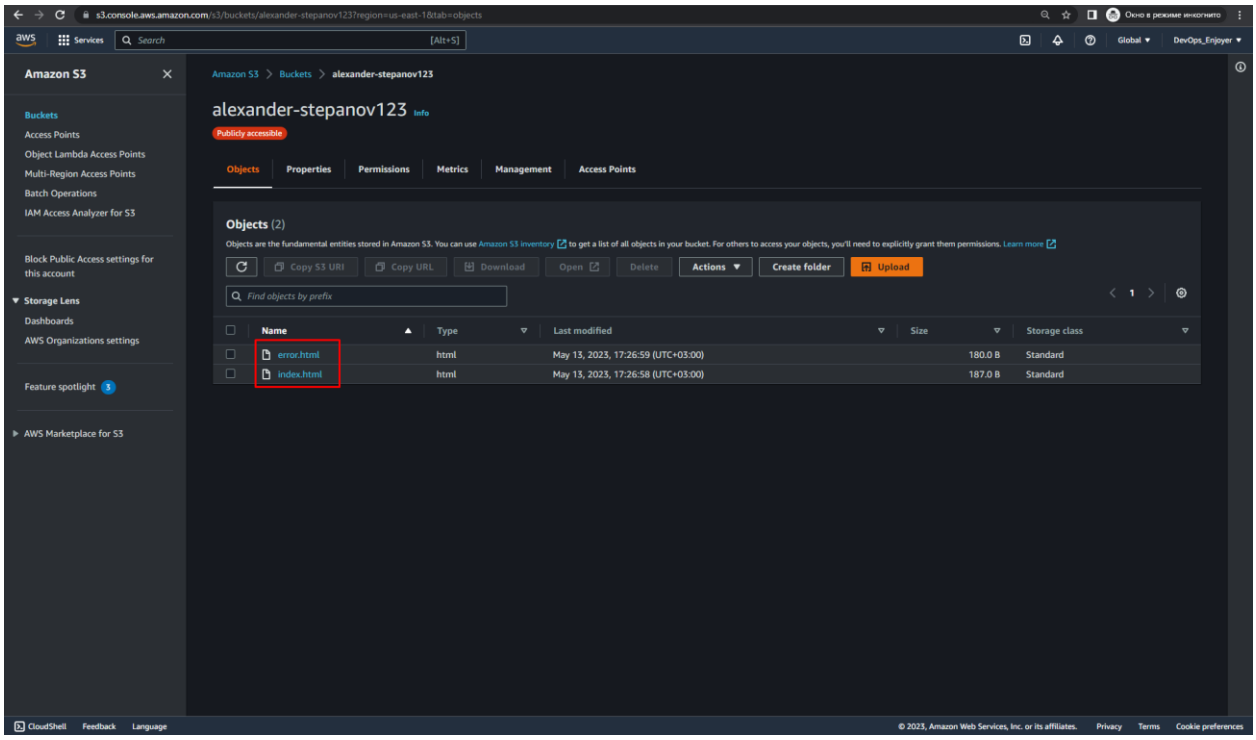

Рисунок 2.7 – Перевірка наявності файлів

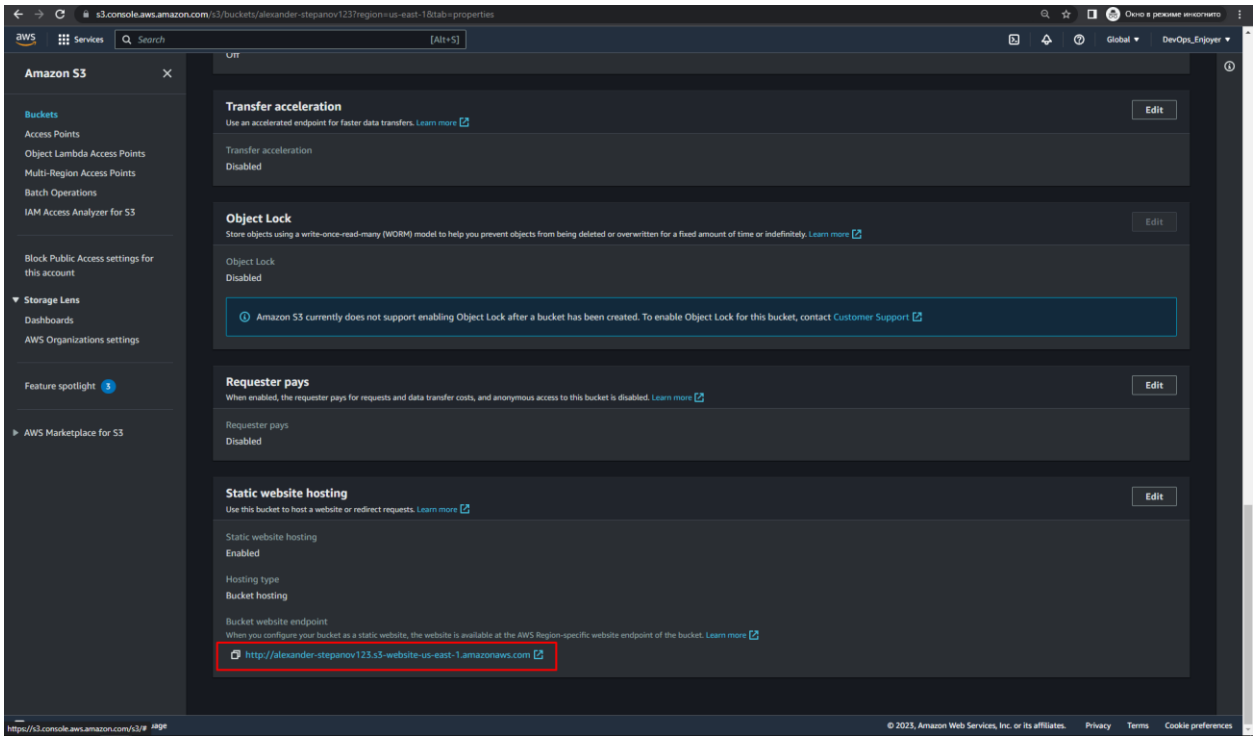

Рисунок 2.8 – Перевірка наявності опції «Static website hosting»

Для того, щоб переконатися, що статичний веб сайт працює, треба натиснути на URL [9]. Якщо все працюватиме – то ми побачимо текст із назвою нашого бакету.

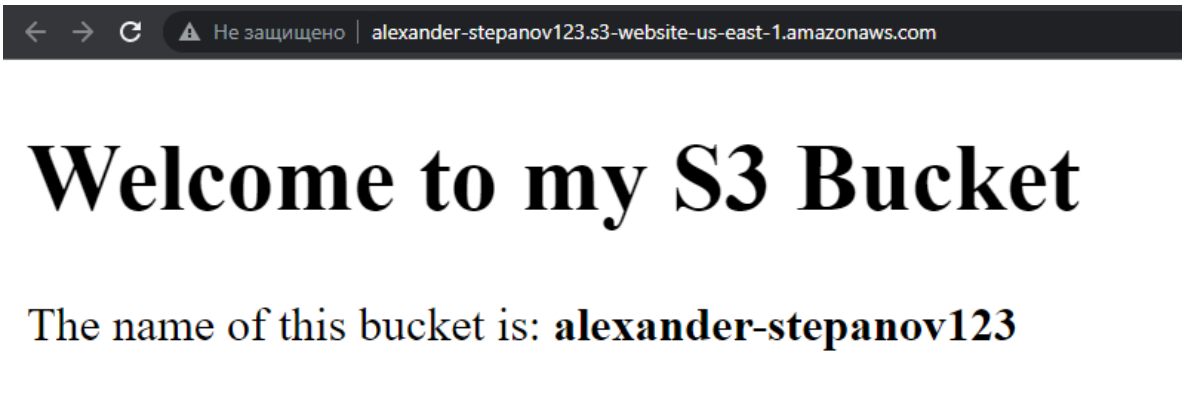

Рисунок 2.9 – Перевірка роботи статичного веб сайту

Таким чином, відтепер ми маємо умови для студентського завдання та готове рішення у виді працюючий скрипт. Це ще не повна автоматизація, бо воно запускається власноруч, а не через Jenkins, тому тепер доцільно перейти до етапу, де ми виконуємо це завдання у Дженкінсі.

Дженкінс – це програмний засіб, який потрібно встановити та налаштувати, детальна інструкція знаходиться за посиланням [10]. Як і AWS CLI, Jenkins є у версіях для Windows, MacOS та Linux [11]. У нашому випадку Дженкінс працює на сервері, який надається сервісом EC2 (Elastic Compute Cloud) від AWS. На сервері встановлена операційна система «Amazon Linux 2» (рис. 2.10).

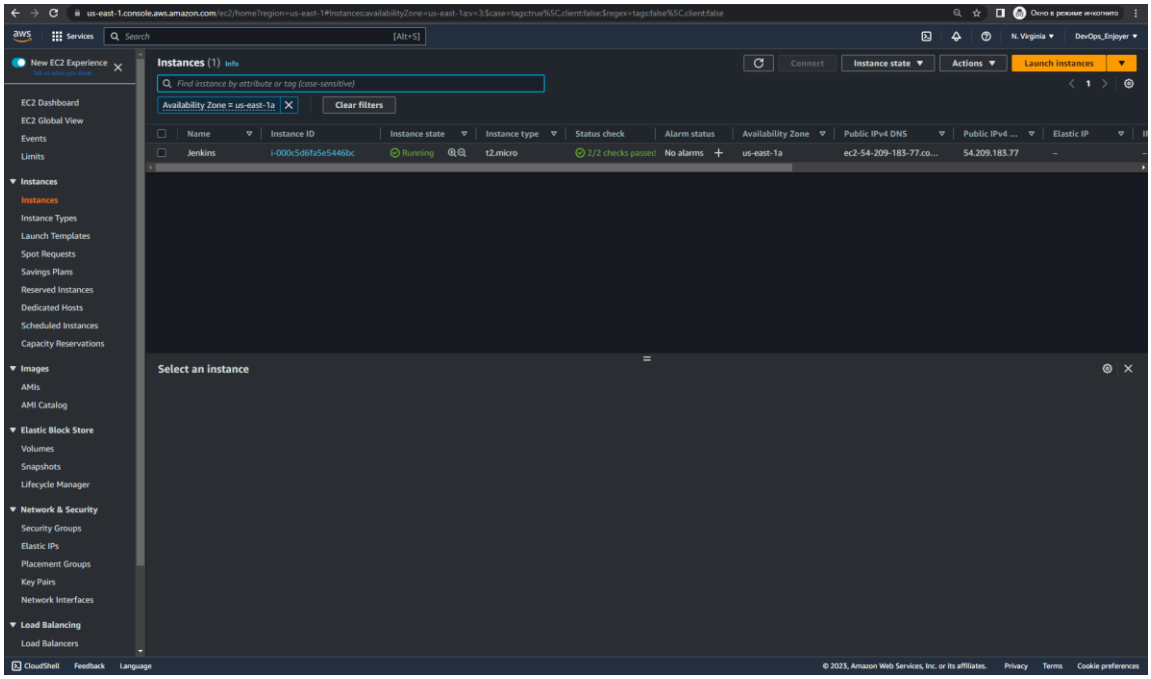

Рисунок 2.10 – EC2 сервер, на якому встановлений Jenkins

Для своєї роботи Дженкінс потребує, щоб у системі була встановлена Java [12] (рис. 2.11).

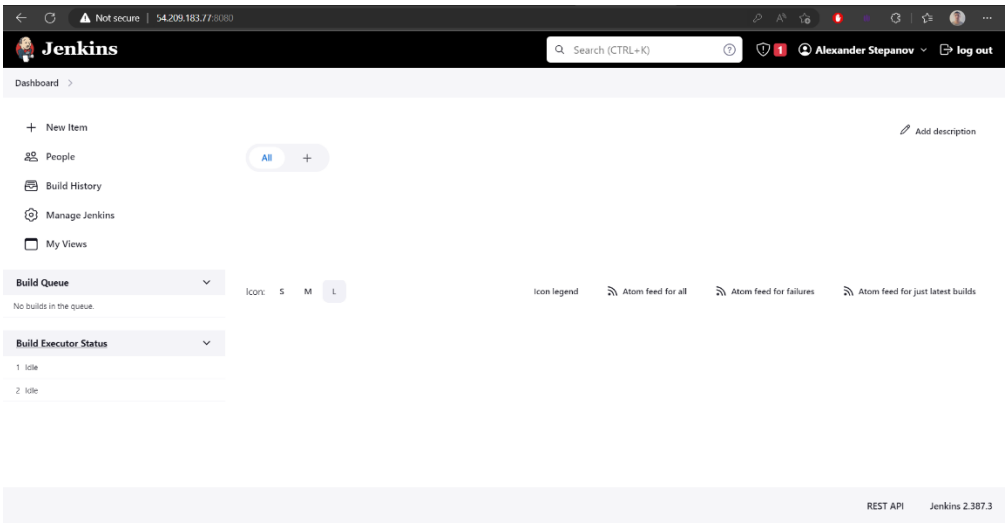

Рисунок 2.11 – Головне меню Jenkins

Якщо натиснути на кнопку «New Item» – з’явиться меню створення нової сутності в Jenkins. Нам потрібна сутність типу «Pipeline». Назвемо її «create_s3_bucket (рис. 2.12)

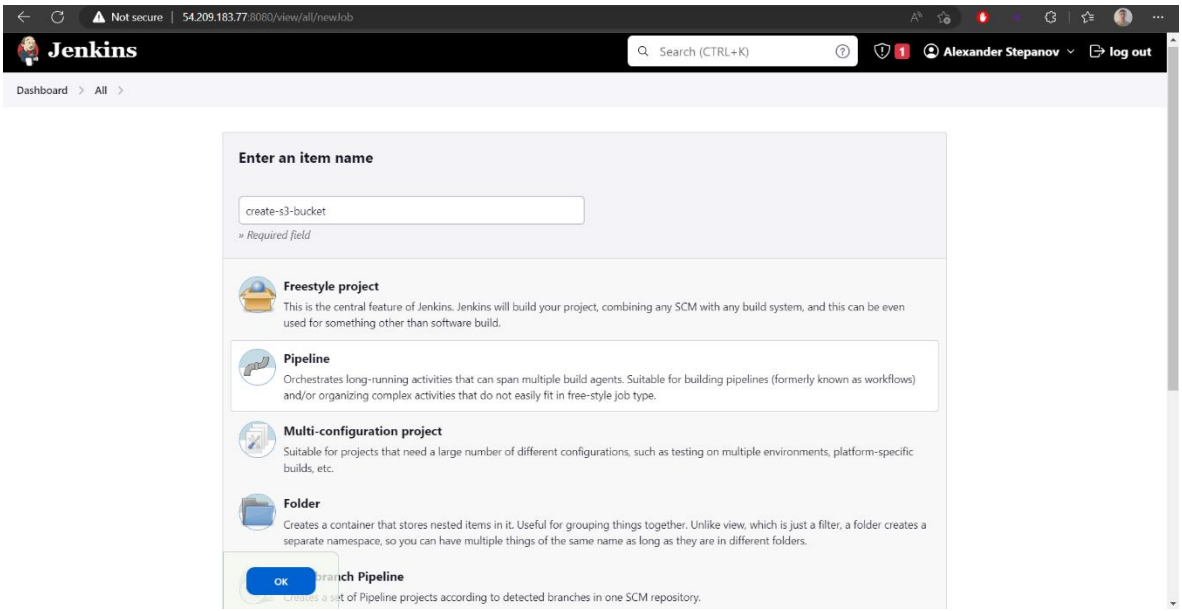

Рисунок 2.12 – Створення сутності типу «Pipeline» в Jenkins

Після натискання кнопки «ОК» ми перейдемо до детального налаштування (рис. 2.13).

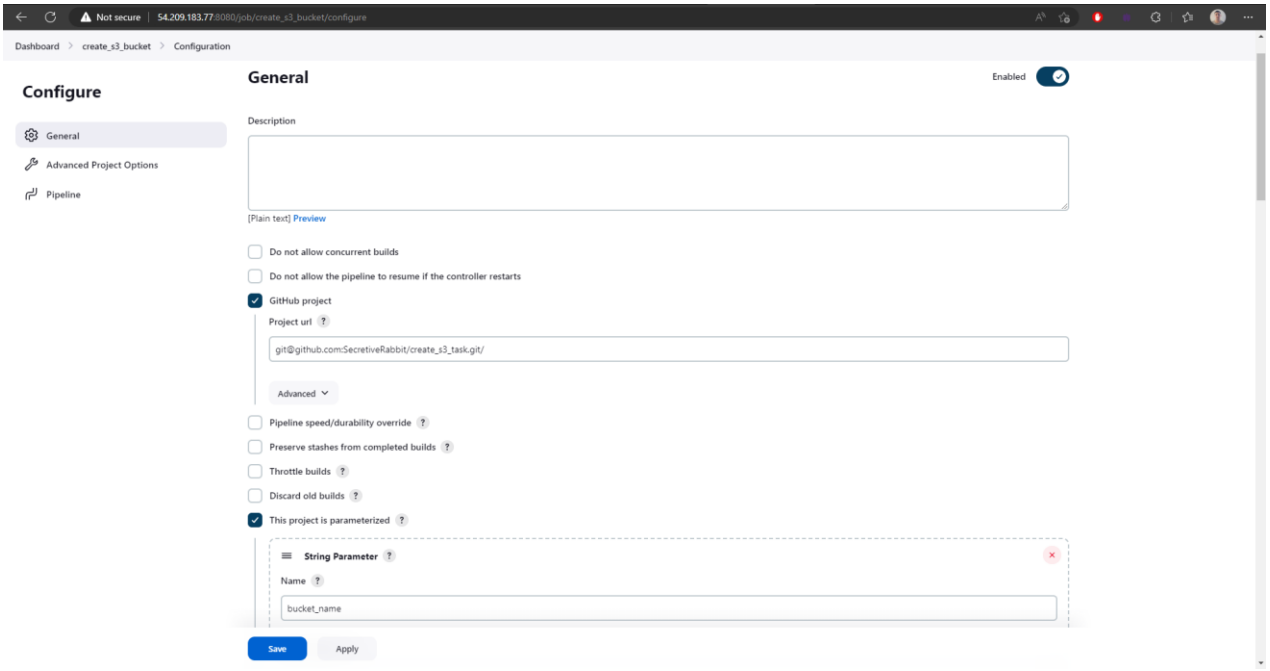

Рисунок 2.13 – Налаштування Jenkins Pipeline

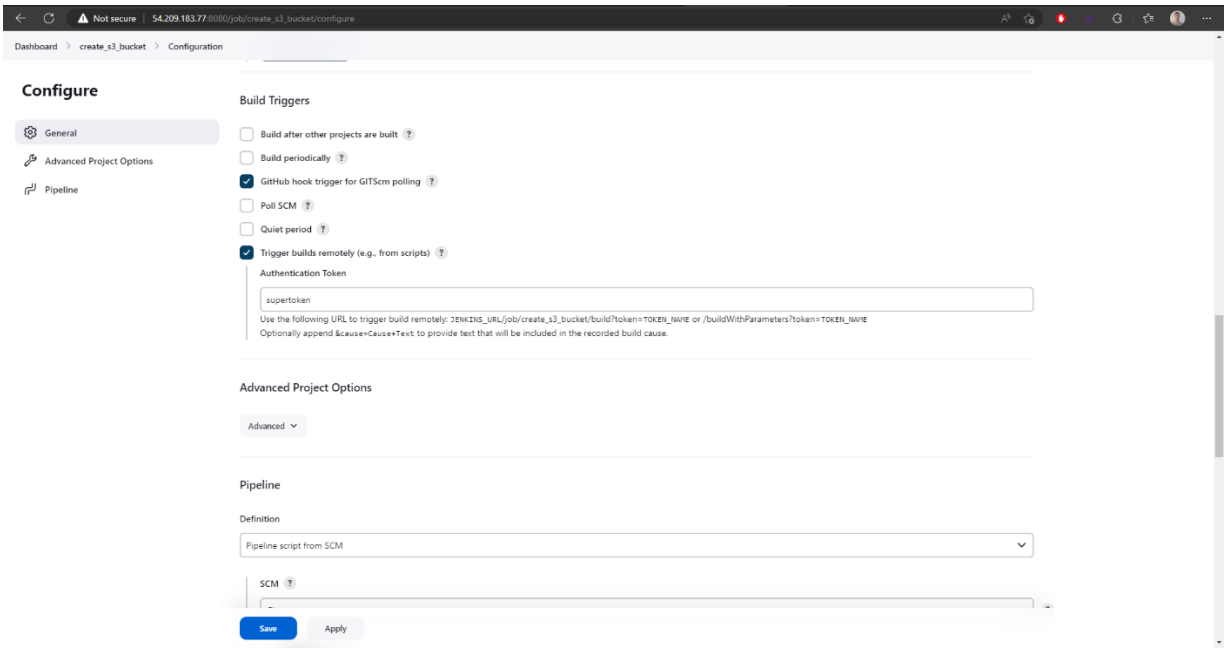

Рисунок 2.14 – Налаштування Build Triggers

Для коректної роботи пайплайну потрібно відмітити пункт «GitHub project», щоб пайплайн зміг виконувати Jenkinsfile, який завантажено у «GitHub». Також відмічено пункт «This project is parameterized», сам цей пункт на даний момент не

несе будь-якого впливу, проте скоріш за все студентські роботи потребуватимуть того, щоб можна було працювати з параметризованими процесами, тому обрано даний пункт (рис. 2.14).

Для того, щоб Cloud Mentor також міг запускати автоматизований процес, відмічено пункт «Trigger build remotely». «GitHub hook trigger for GITscm polling» для даної роботи являється необов'язковим. Він дозволяє запускати Jenkins Pipeline після того як робимо зміни у GitHub репозиторії, з яким пайплайн засинронізований. Варто відмітити що в термінології Jenkins пайплайн також можна називати «Job». Для роботи даного параметру потрібно, щоб у GitHub був налаштований вебхук, а також щоб у Jenkins був встановлений GitHub плагін, який зазвичай встановлюється із стандартним пакетом плагінів під час первинного налаштування Jenkins (рис. 2.15) [12].

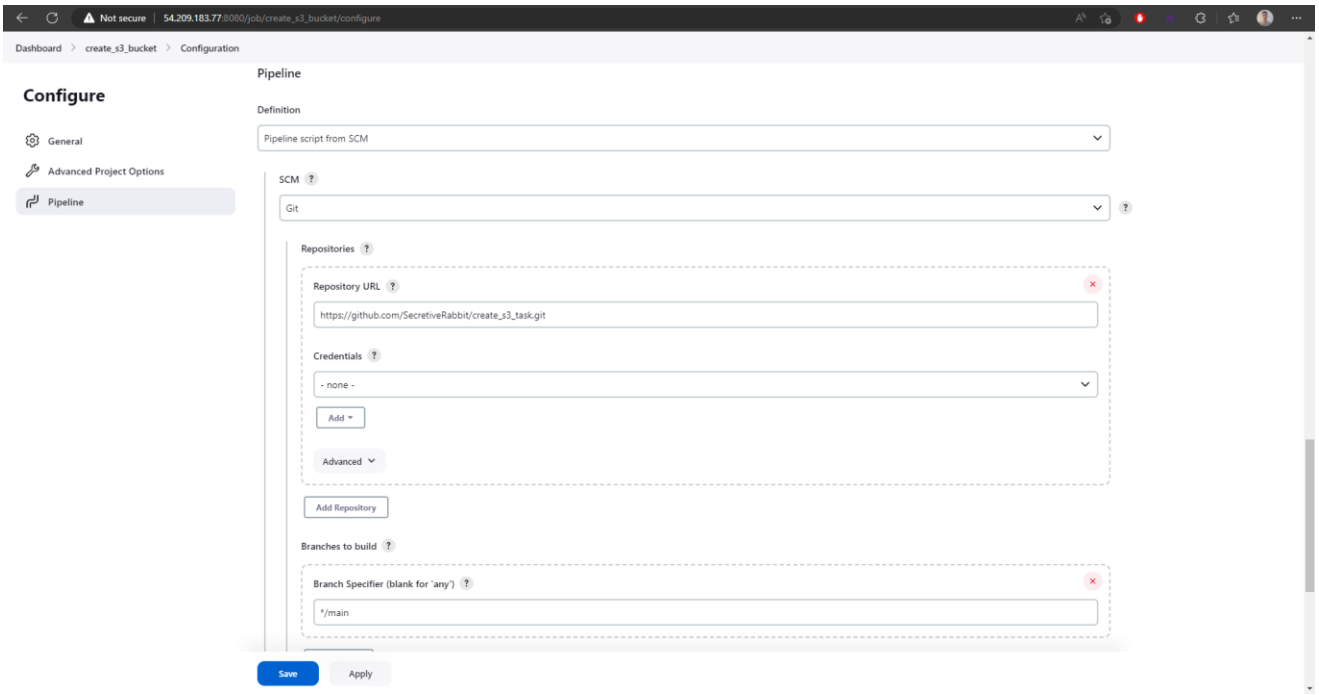

Рисунок 2.15 – Налаштування Pipeline definition

Бажано зробити так, щоб Jenkins брав сценарій з GitHub репозиторію, тому обрано «Pipeline script from SCM», у якості засобу SCM (Source Code Management) вказали Git, надали URL до репозиторії в GitHub та у якості гілки, з якої братиметься код, вказано «main».

На останок треба вказати назву файлу, який міститиме сценарій, який буде виконувати Jenkins. За замовчуванням такий файл має назву «Jenkinsfile», це треба вказати у полі «Script Path» (рис. 2.16).

Дані, які знаходяться у «Jenkinsfile» – це і є пайплайн сам по собі. У ньому містяться усі вказівки, які виконуватиме Jenkins.

Пайплайн буває декларативний та імперативний [13]. Декларативний пайплайн складається з етапів, на яких виконується робота. Кожен етап містить кроки, які можуть бути окремими операторами, викликами функцій або сценаріями, або нестандартними оболонками коду, що надаються різними плагінами.

Імперативний пайплайн є менш формальним і забезпечує більшу гнучкість. Однак часто потрібно більше ручного керування. Наприклад, під час використання сценарного конвеєра необхідний код не завантажується автоматично з репозиторію GitHub на підпорядкованих вузлах. Таким чином, у коді конвеєра важливо чітко вказати, що і звідки завантажувати, а також що саме виконувати.

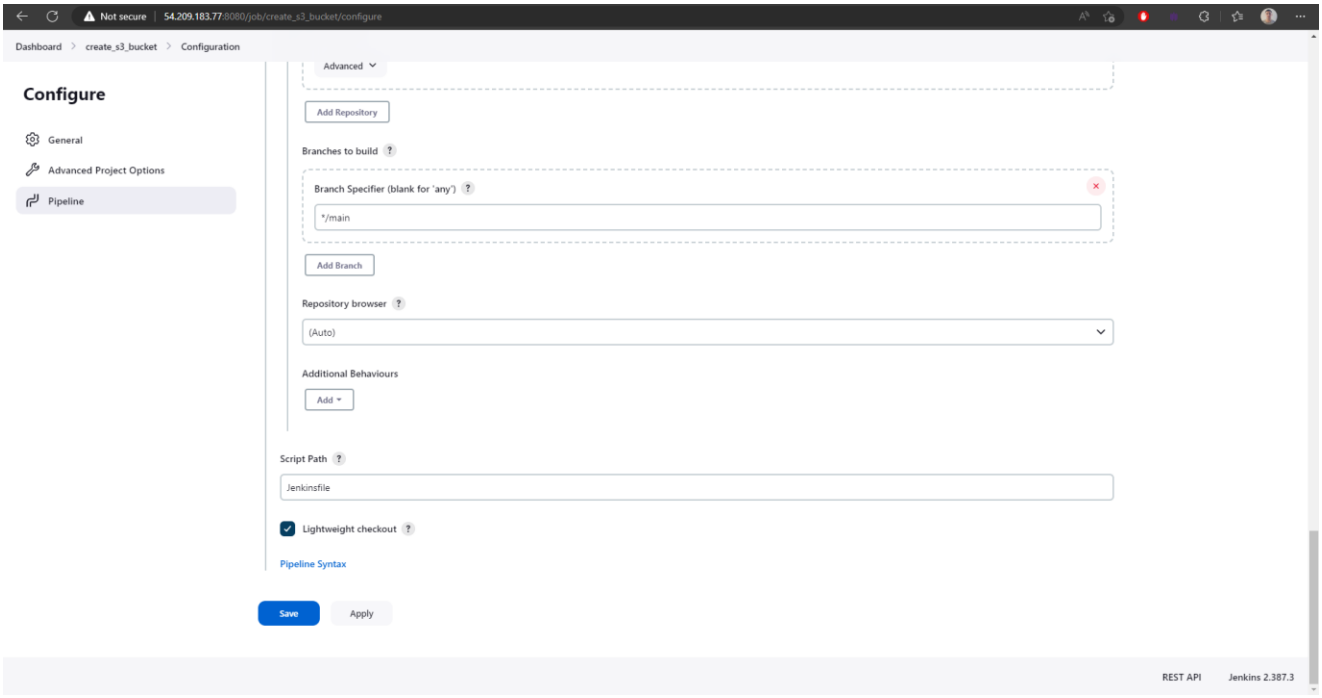

Рисунок 2.16 – Налаштування Jenkins Pipeline Script Path

Даний пайплайн являється декларативним, у ньому буде покрокове виконання системних команд і він матиме декілька логічно розділених етапів.

Сценарій, який лежить в Дженкінс файлі, можна переглянути на рис. 2.17.

SecretiveRabbit Update Jenkinsfile — Latest commit 8228ebb 2 weeks ago — History

1 contributor

26 lines (21 sloc) | 533 Bytes — Raw | Blame

```
pipeline {
    agent any

    stages {

        stage('Pull') {
            steps {
                sh 'rm -rf create_s3_task'
                sh 'git clone https://github.com/SecretiveRabbit/create_s3_task.git'
            }
        }


        stage('Build') {
            steps {
                sh 'chmod +x create_s3_task/create_s3.sh'
            }
        }

        stage('Run script') {
            steps {
                sh 'create_s3_task/create_s3.sh'
            }
        }
    }
}
```

Рисунок 2.17 – Jenkinsfile

Логічно пайплайн поділений на 3 етапи:

– Pull – стягування файлів з репозиторію GitHub, де лежить скрипт, який створюватиме S3 бакет;
– Build – для того, щоб цей скрипт можна було запустити, йому надаються права на виконання;
– Run script – виконується запуск скрипту, який створює статичний веб сайт.

Після налаштування пайплайну залишається натиснути «ОК», після чого перенесе до меню перегляду пайплайну, через нього можна побачити статус пайплайну, якщо він виконується або вже виконувався, можна його запустити, перейти до перегляду логів, видалити пайплайн. Також функціонал даного меню можна розширити сторонніми плагінами, але це зазвичай потрібно лише просунутим користувачам для побудови дійсно складних автоматизованих процесів (рис. 2.18).

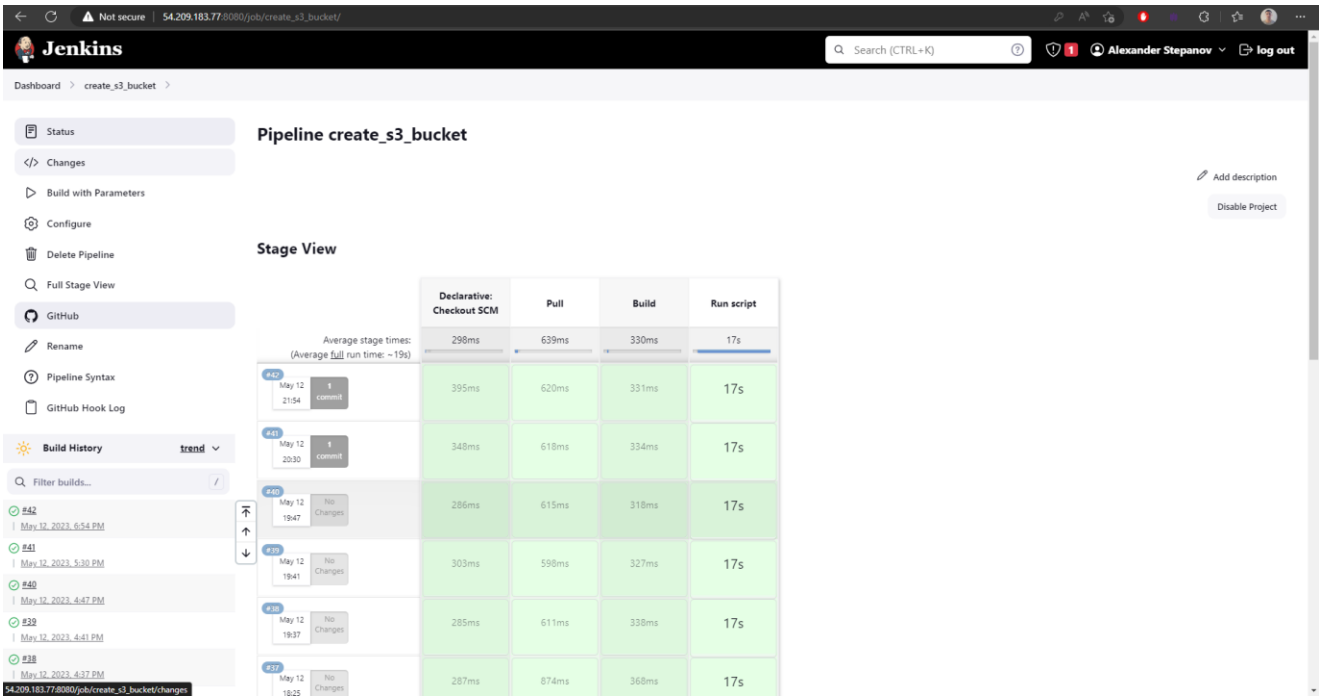

Рисунок 2.18 – Створений Jenkins Pipeline

Було здійснено багато спроб запустити створення бакету, тому є 42 спроби, на початку такого не буде.Якщо вийти назад з пайплайну, можна переглянути усі створені сутності (рис.2.19).

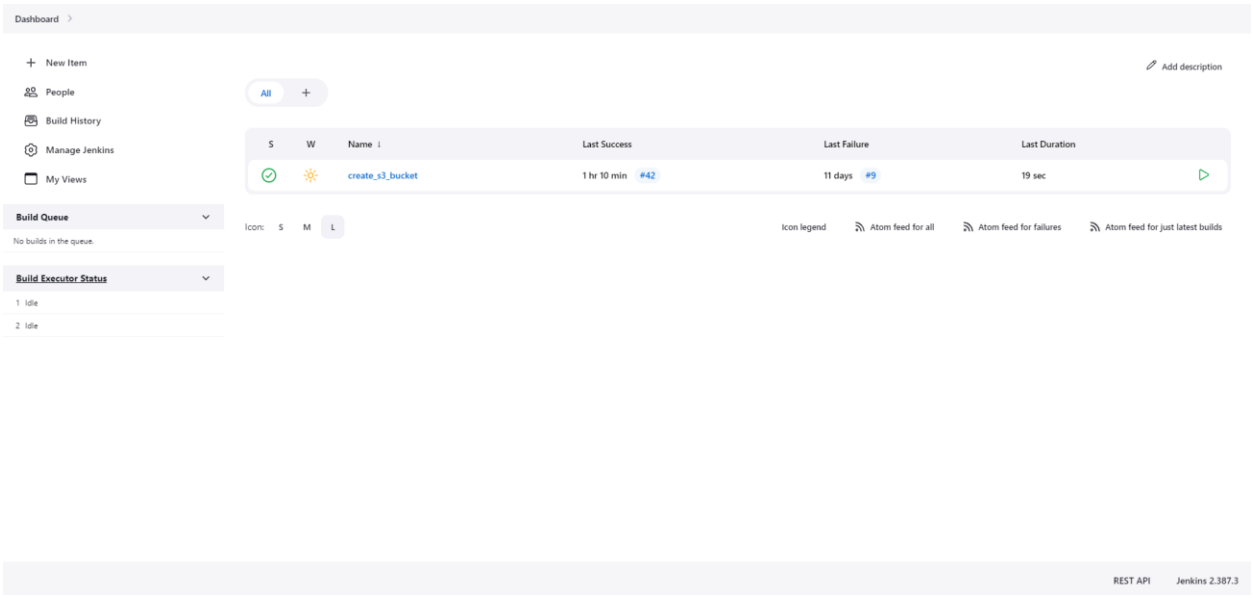

Рисунок 2.19 – Меню створених сутностей

Для запуску пайплайну треба натиснути кнопку «Build with parameter». Тоді Jenkins запропонує ввести параметри перед тим, як почати виконання процесу (рис. 2.20).

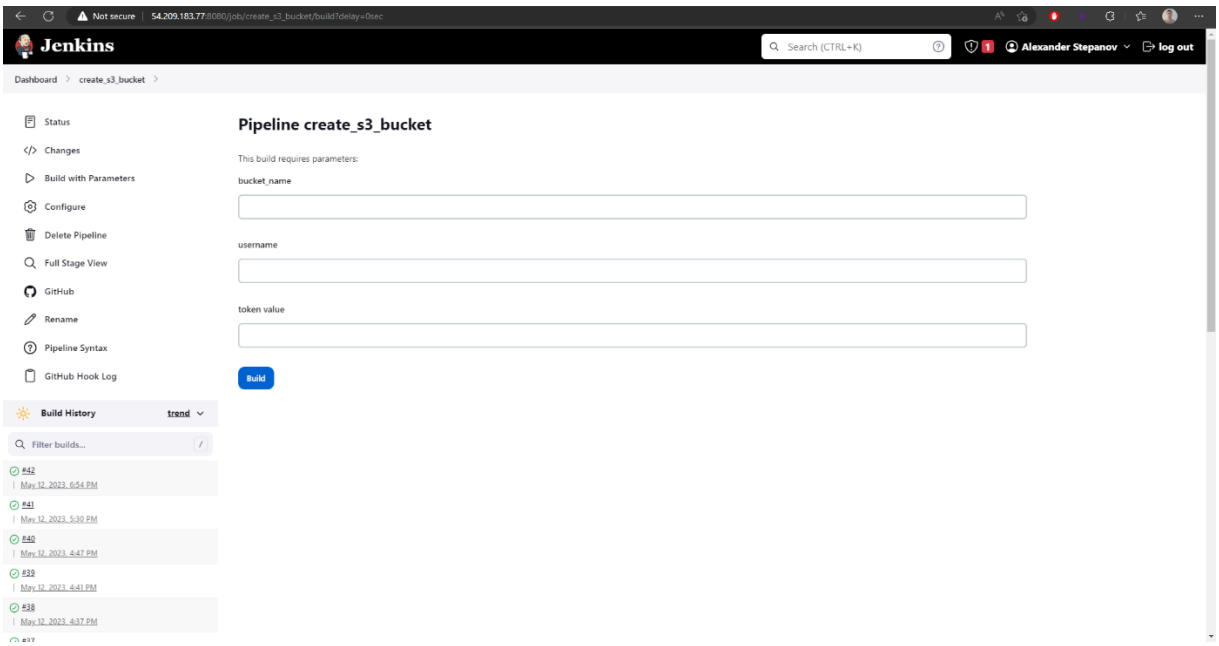

Рисунок 2.20 – Меню «Build with parameters»

Після запуску пайплайну можна подивитися на прогрес виконання (рис. 2.21).

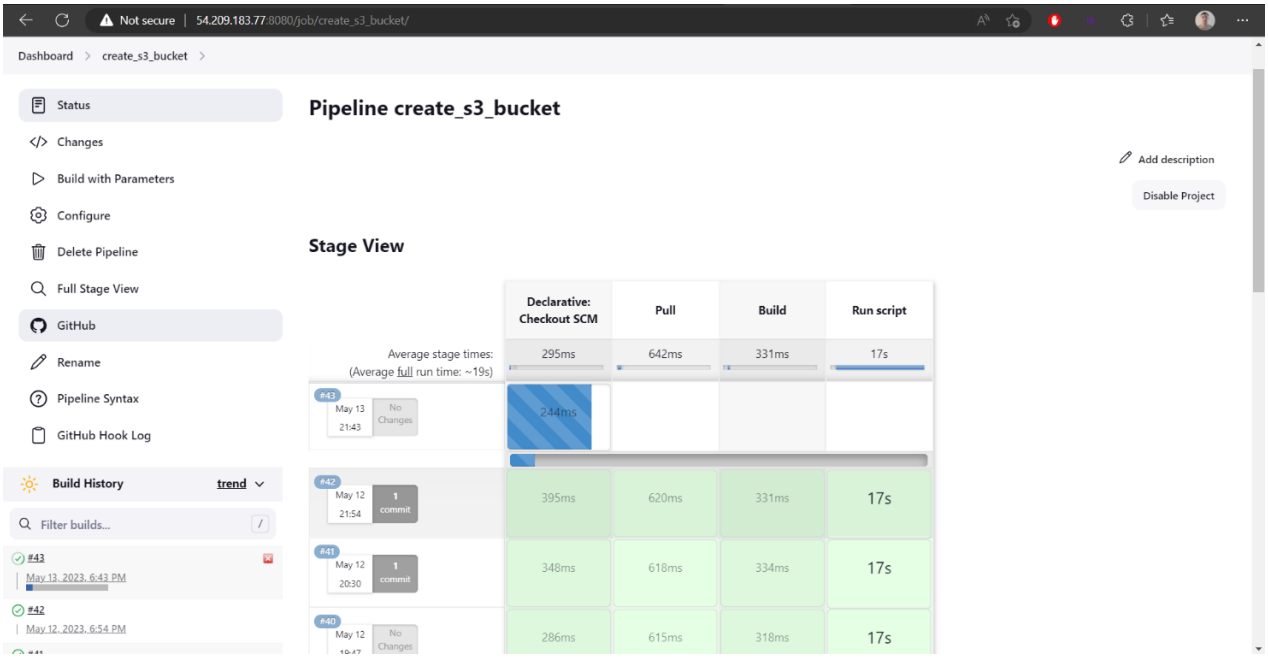

Рисунок 2.21 – Статус виконання пайплайну

Шкала прогресу, яка додається завдяки плагінам в Jenkins, дуже інформативно показує етапи, час, який було витрачено на кожен з етапів та місце, на якому знаходиться пайплайн. Визначається це завдяки командам, які відносяться до кожного з етапів. Якщо усі команди виконалися успішно – йде етап зафарбовується зеленим та пайплайн переходить до наступного.

Також можна переглядати прогрес виконання іншим способом – використовуючи розділ «Console Output». Даний розділ виводить усю інформацію, яку виводять команди, що виконуються Дженкінсом у даному пайплайні. Даний інструмент найбільш усього корисний коли у пайплайні стає помилка і треба дізнатися причину [14].

У меню можна потрапити двома шляхами:

– Натиснувши на будь-який з номерів спроб запуску пайплайну, що розташоване зліва знизу, та обрати вкладку «Console Output».
– Там же, біля номеру спроби запуску пайплайну, відвести курсор справа від номеру, поки не з’явиться стрілка вниз. Натиснути на стрілку, обрати «Console Output».

У даному випадку помилок не виникало, тому можна подивитись на вивід інформації з команд у терміналі, що зображено на рис. 2.22.

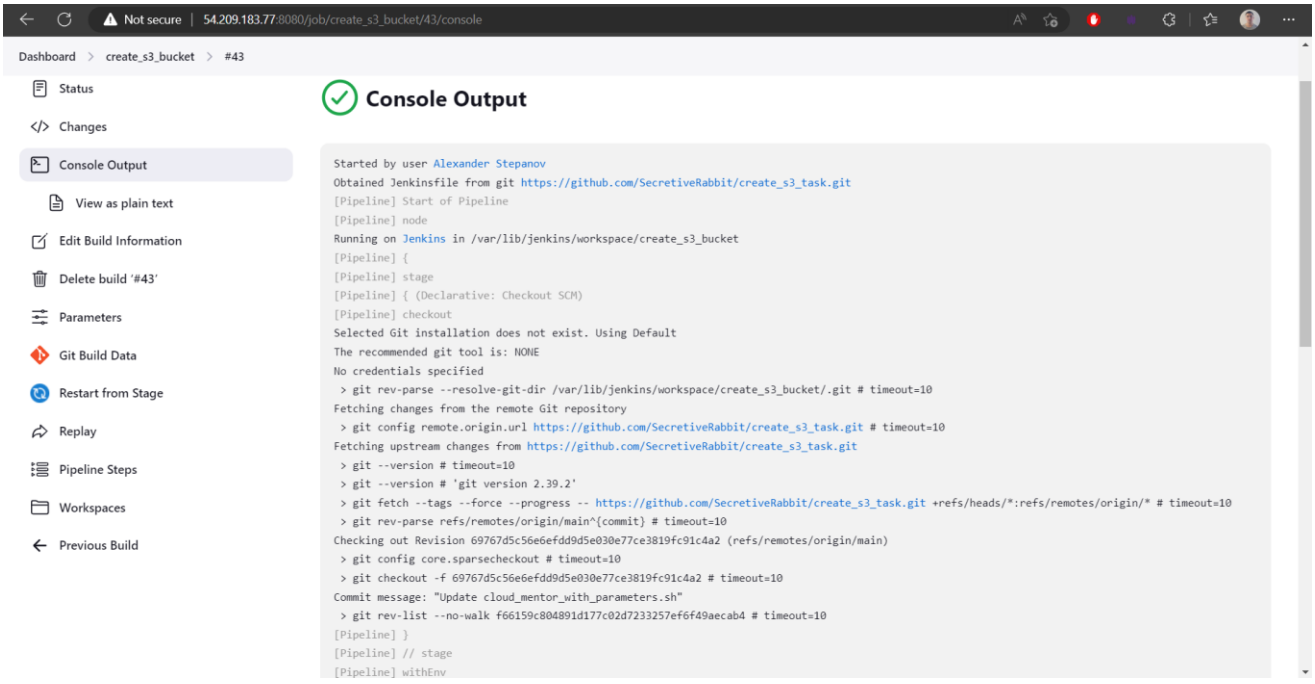

Рисунок 2.22 – Меню «Console Output»

В усіх без виключення випадках в кінці виконання пайплайну відображається

статус виконання. Це може бути або «SUCCESS» або «FAILURE» (рис. 2.23).

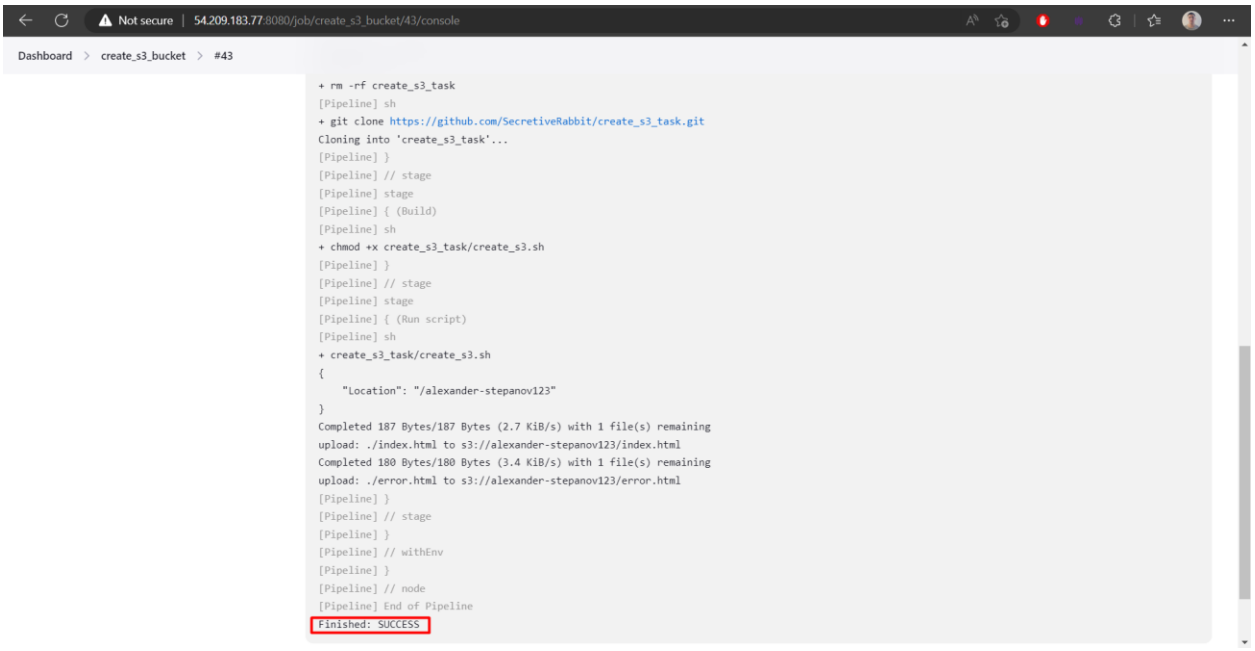

Рисунок 2.23 – Кінцевий статус виконання пайплайну

Тому можна стверджувати, що студентське завдання було виконане, так як автоматизований процес виконався успішно, створив S3 бакет із функцією хостінгу статичного веб сайту. Тепер можна приступати до розробки маханізмів верифікації.

Даний момент часу є найбільш підходящим для того, щоб почати міркувати над механізмами перевірки, оскільки конкретика стосовно будування умов завдання та процесу його виконання стала цілком зрозуміла. Етап перевірки є найбільш цікавим серед усіх, тому що це і є мета даної роботи.

Як вже згадувалося раніше, серед критеріїв перевірки можна виділити головні три:

– Робота була виконана Дженкінсом;
– Робота виконана без помилок у пайплайні;
– Робота виконана правильно.

Таким чином, можна думати над тим, як ці моменти можна задовільнити. Звичайних команд, які є в терміналі, тут буде недостатньо, тому що тут передбачається глибинне розуміння того, як працює Jenkins, як виконуються команди, виникає купа умовностей, оскільки студент може по-різному досягти однієї і тієї ж цілі. Використовуючи методи різного ступеня ефективності. Тому перш за все треба сфокусуватися на реалізації перевірки.

Також треба придумати, як Дженкінс Клауд Ментора буде взаємодіяти з Дженкінсом студента. Яким чином він буде отримувати доступ до нього і якої буде взаємодія [8].

І останній складний момент – це процес формування оцінки. Бо робота може бути виконана частково, покривати не всі вимоги, які були поставлені до задачі.

Ґрунтуючись на критеріях успішного виконання завдання, було прийняте рішення писати програму користуючись інтерпретатором bash [14]. Програмна реалізація представлена на рис. 2.24 – 2.25.

96 lines (84 sloc) | 2.72 KB — Raw | Blame

```bash
#!/bin/bash

# Set the initial values for the variables
score_max=4
score_current=0
ip_address="54.209.183.77"
username="alexander"
token_name="supertoken"
token_value="11e4336999ba2df92e4aed4d39e0badb74"

# Parse command line arguments
while [[ $# -gt 0 ]]; do
  key="$1"
  case $key in
    -a|--ip-address)
      ip_address="$2"
      shift 2
      ;;
    -u|--username)
      username="$2"
      shift 2
      ;;
    -n|--token-name)
      token_name="$2"
      shift 2
      ;;
    -v|--token-value)
      token_value="$2"
      shift 2
      ;;
    -s|--site-name)
      site_name="$2"
      shift 2
      ;;
    *)
      echo "Unknown option: $1"
      exit 1
      ;;
  esac
done

# Get the last build number before triggering the job
last_build_before_triggering=$(curl -s -u "${username}:${token_value}" http://${ip_address}:8080/job/create_s3_bucket/lastBuild/buildNumber)
```

Рисунок 2.24 – Програмна реалізація перевірки роботи студента, частина перша

```
last_build_before_triggering=$(curl -s -u "${username}:${token_value}" http://${ip_address}:8080/job/create_s3_bucket/lastBuild/buildNumber)

# Trigger the Jenkins job and capture the output
jenkins_output=$(curl -s -u "${username}:${token_value}" http://${ip_address}:8080/job/create_s3_bucket/buildWithParameters\?token\=${token_name}\&test1\=test2\&jenk\=dsa)

# Check if the output is empty
if [ -z "$jenkins_output" ]; then
    # If the output is empty, increment the score_current variable
    ((score_current++))
    echo "The Jenkins Job was triggered"
fi

# Sleep for 10 seconds
sleep 10

# Get the last build number after triggering the job
last_build_after_triggering=$(curl -s -u "${username}:${token_value}" http://${ip_address}:8080/job/create_s3_bucket/lastBuild/buildNumber)

# Compare the last build numbers and increment the score_current variable if they're different
if [ "$last_build_before_triggering" != "$last_build_after_triggering" ]; then
    ((score_current++))
    echo "The Jenkins Job was started"
fi

# Check if the Jenkins job completed successfully
for i in {1..10}; do
    echo "Checking if the Jenkins job is completed successfully (attempt $i)"
    result=$(curl -s -u "${username}:${token_value}" http://${ip_address}:8080/job/create_s3_bucket/lastBuild/api/json -H "Content-Type: application/json" -X GET | grep '"resu
    if [ $? -eq 0 ]; then
        echo "The Jenkins Job is completed successfully"
        ((score_current++))
        break
    else
        sleep 10
    fi
done

if [ $? -ne 0 ]; then
    echo "The Jenkins Job failed or didn't complete"
fi

# Check if the S3 Static Web Site is created
if [ -z "$site_name" ]; then
    site_name="alexander-stepanov123"
fi
s3_output=$(curl -s http://${site_name}.s3-website-us-east-1.amazonaws.com)
if [[ "$s3_output" == *"${site_name}"* ]]; then
    ((score_current++))
    echo "The S3 Static Site works"
else
    echo "It seems something is wrong with the Static S3 Web Site"
fi

echo "Current score: $score_current out of $score_max"
```

Рисунок 2.25 – Програмна реалізація перевірки роботи студента, частина друга

Самостійно переглянути програмний код можна у тому ж репозиторії, що ми згадували раніше, «create_s3_task».

Даний програмний код запускає пайплайн студента, чекає поки він виконається та виконує перевірки. Програма, отримує доступ до Дженкінса студента за допомогою виділеного користувача та створеного для нього токену.

Програма спочатку ініціалізує змінні, зокрема лічильник успішних перевірок. Значення цього лічильника у нашому випадку і є результуючою оцінку. На початку програми ця змінна дорівнює нулю, проте зростає на один із кожною успішною перевіркою.

Першою перевіркою є сам запуск пайплайну. Це робиться завдяки API запиту до Дженкінса студента, який реалізован командою «curl». Якщо запит пройде успішно – програма не поверне ніякого виводу на екран. Якщо вивід є – пайплайн не запустився. У разі успішного запуску лічильник оцінки збільшиться на один.

Наступна перевірка така: програма запитує порядковий номер останнього запуску пайплайну. Потім відбувається запуск пайплайну, після чого повторно запитується порядковий номер останнього запуску пайплайну і порівнює їх між собою. Якщо порядковий номер змінився – тоді повідомлення про успішний старт пайплайну виводиться у термінал, після чого оцінка зростає на один.

Далі програма впродовж певного часу опитує Дженкінс студента чи не виконалась джоба. Робиться перевірка максимум 10 з інтервалами в 10 секунд. В рузльтаті виходить що програма максимум чекає 100 секунд. Під час опитування програма отримує відповідь про статус виконання джоби та шукає у статусі «“result:” “SUCCESS”». Якщо вона знаходить це у відповіді, програма виходить з циклу перевірки та збільшує лічильник оцінки. Якщо після 10 перевірок програма так і не знаходить у відповіді повідомлення про успіх – програма посилає на екран терміналу повідомлення про те, що програма не була виконана успішно або ще не завершилася.

Останньою перевіркою є перевірка створеної статичної сторінки. URL, який створюється для статичного веб сайту, має відповідний шаблон і його легко визначити наперед. На даний момент часу шаблон має такий вигляд (рис. 2.26).

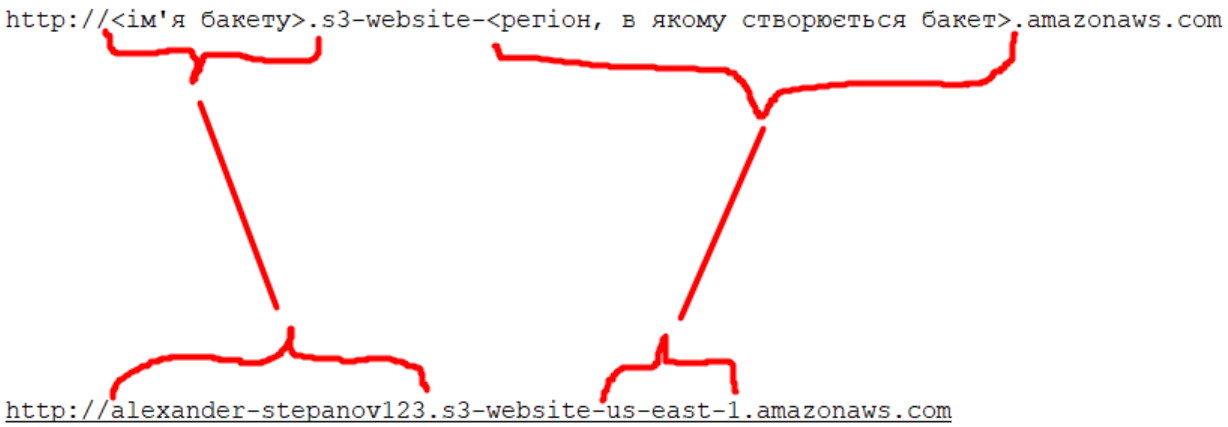

Рисунок 2.26 – Шаблон створення URL для статичного веб сайту

Користуючись цими знаннями, при наперед відомій назві бакету можна переконатися чи створений був бакет чи ні. Програма перевіряє URL та аналізує відповідь. Якщо у відповіді є назва бакета, яку студент повинен був покласти у

«index.html» файл – тоді завдання вважається виконаним успішно і відповідно лічильник збільшується на 1. Якщо ні – на екран терміналу виводиться повідомлення про те, що скоріш за все бакет не був налаштований належним чином.

Як вже раніше було згадано, для виконання перевірки програма потребує ім’я користувача та токен. Токен можна створити у налаштуваннях Дженкінса, якщо пройтись таким шляхом: вийти в «головне меню», потім перейти до «Manage Jenkins», далі в «Manage Users», потім обрати користувача або спочатку створити і обрати, якщо потрібно, далі натиснути «Configure» та пролистати до розділу «API Token» [15] (рис. 2.27).

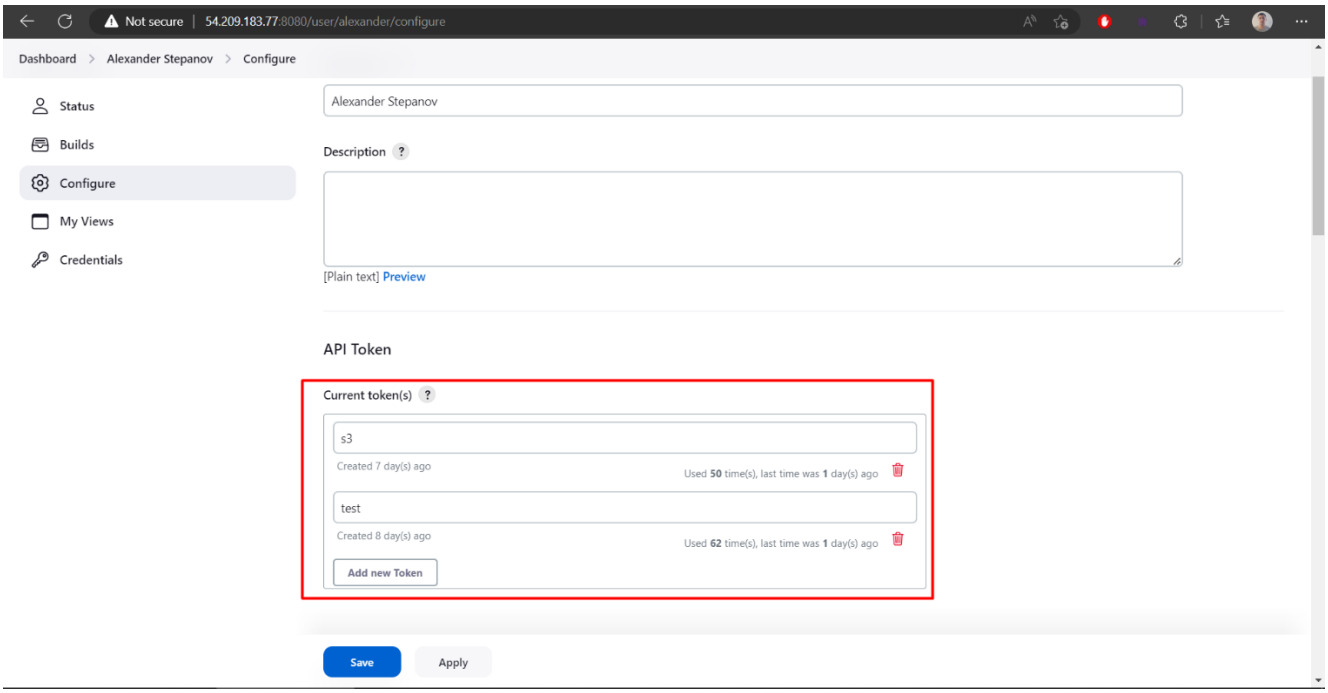

Рисунок 2.27 – створення токену

Також вчительский Дженкінс повинен знати IP адресу серверу, на якому встановлений Дженкінс студента, щоб мати змогу відправляти йому запиту про статус та запускати джобу.

Тепер, коли ми маємо програму, яка перевіряю роботу студента та виставляє оцінку, давайте створимо на її основі Jenkins Job у вчительскому середовищі. Для уникнення повторень ми не будемо заново показувати як створювати джоби, доцільно показати лише налаштування, які необхідно зробити (рис. 2.28 – 2.29).

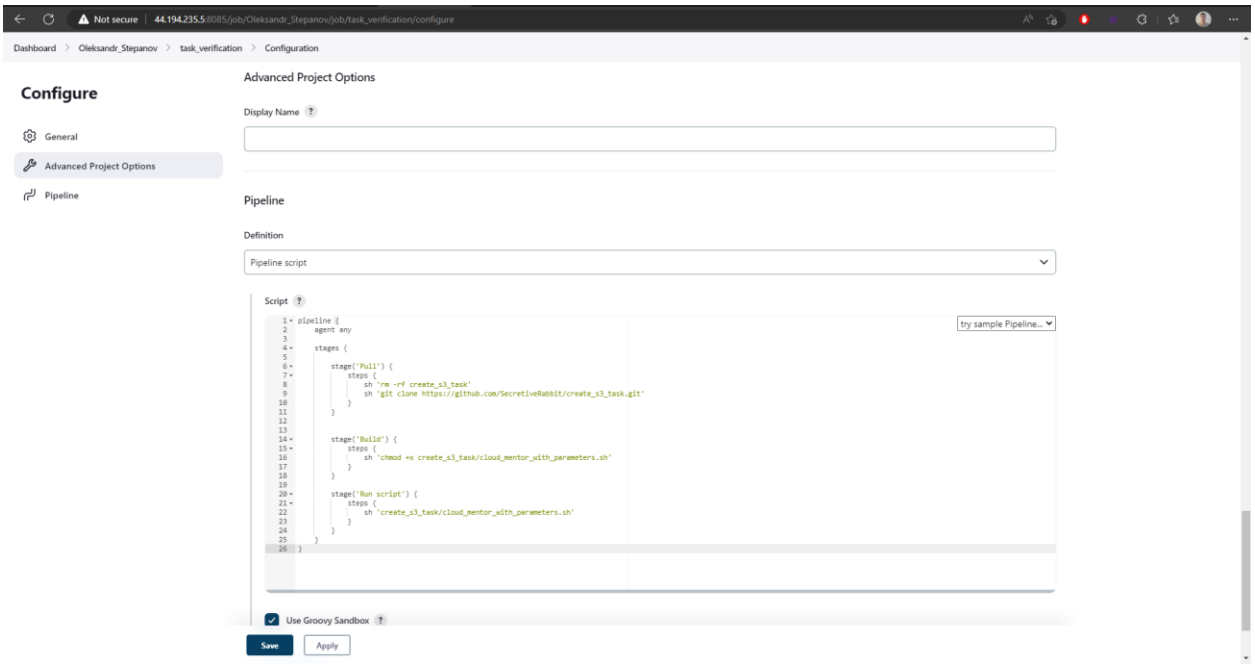

Рисунок 2.28 – Сценарій для виконання джоби з перевірки студентської роботи

Сценарій складається з трьох етапів, логічно він дуже схожий на пайплайн студентської роботи:

– Pull – стягування файлів з репозиторію GitHub, де лежить скрипт, який створюватиме S3 бакет;
– Build – для того, щоб програму можна було запустити, їй надаються права на виконання;
– Run script – виконується запуск скрипту, який виконує перевірку створеного «S3 static web site hosting».

Тепер цю джобу можна запустити. Вона приймає параметри, які необхідні для роботи програми:

– IP-адреса студентського Дженкінсу;
– Назва токену, який буде використовуватися;
– Значення токену.

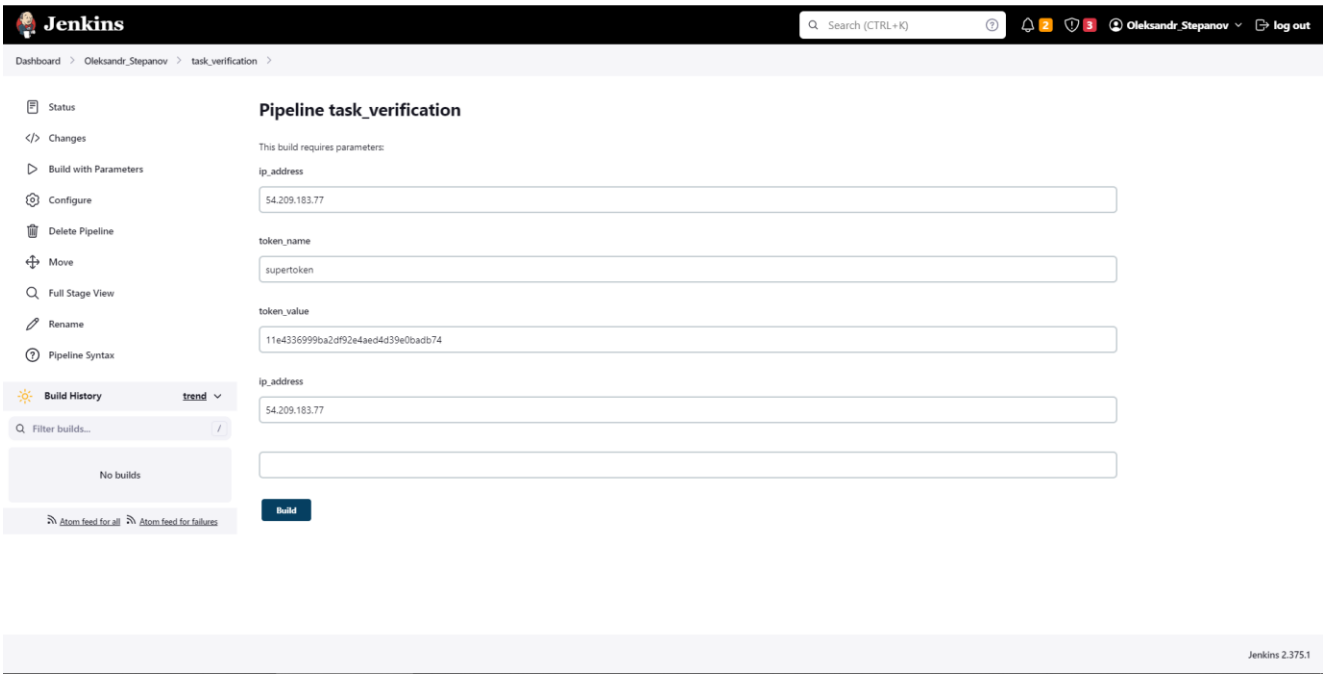

Рисунок 2.29 – Запуск «task_verification» з параметрами

Можна спостерігати процес виконання джоби (рис. 2.30).

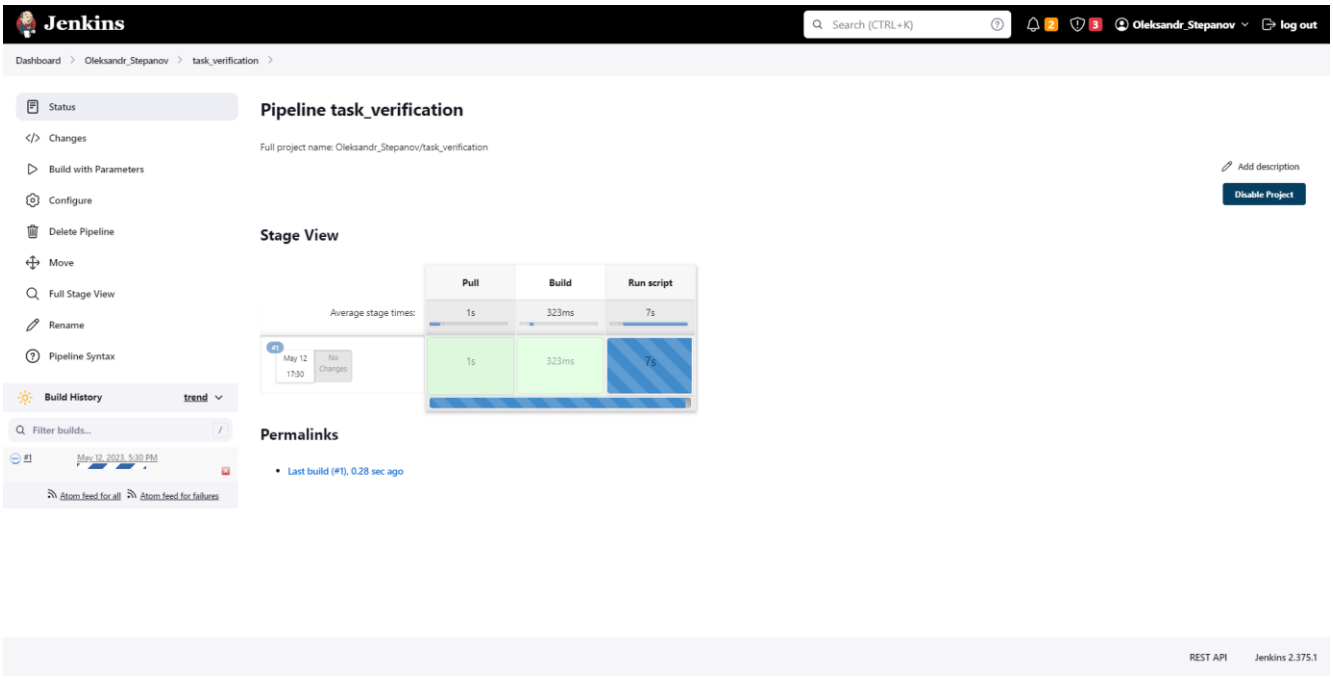

Рисунок 2.30 – Процес виконання джоби

Джоба завершилася успішно (рис. 2.31):

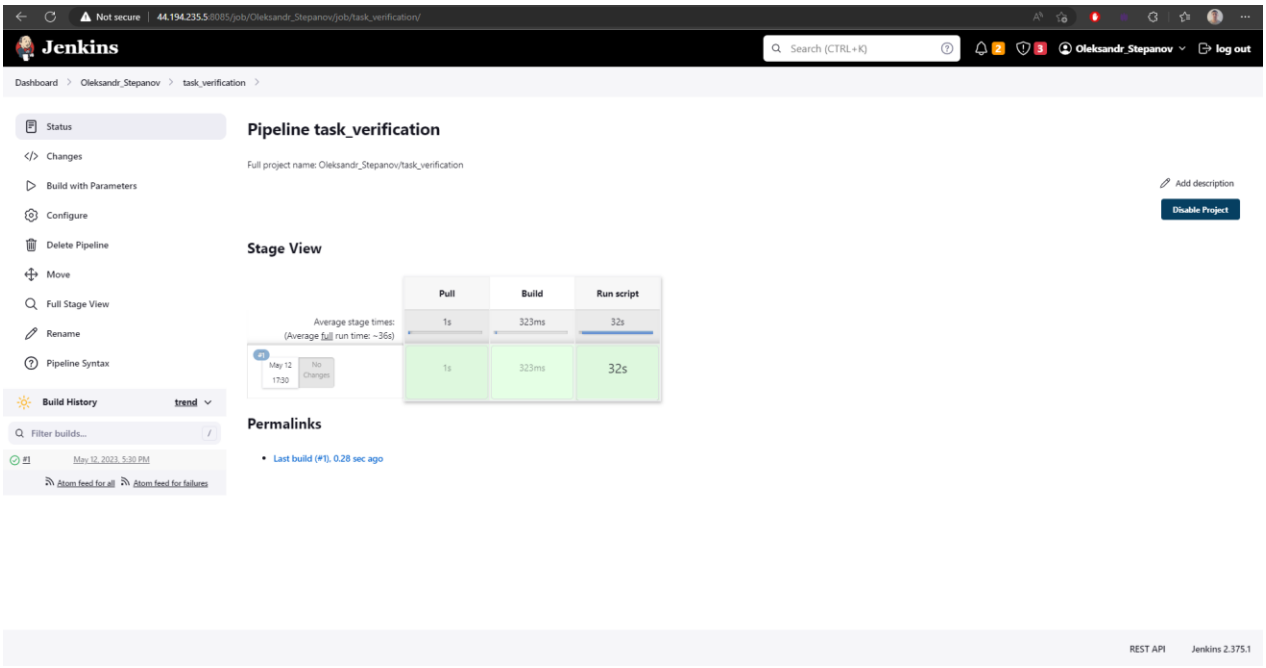

Рисунок 2.31 – Статус виконання джоби

Повідомлення, які програма вивела на термінал, можна побачити у «Console Output» (2.32):

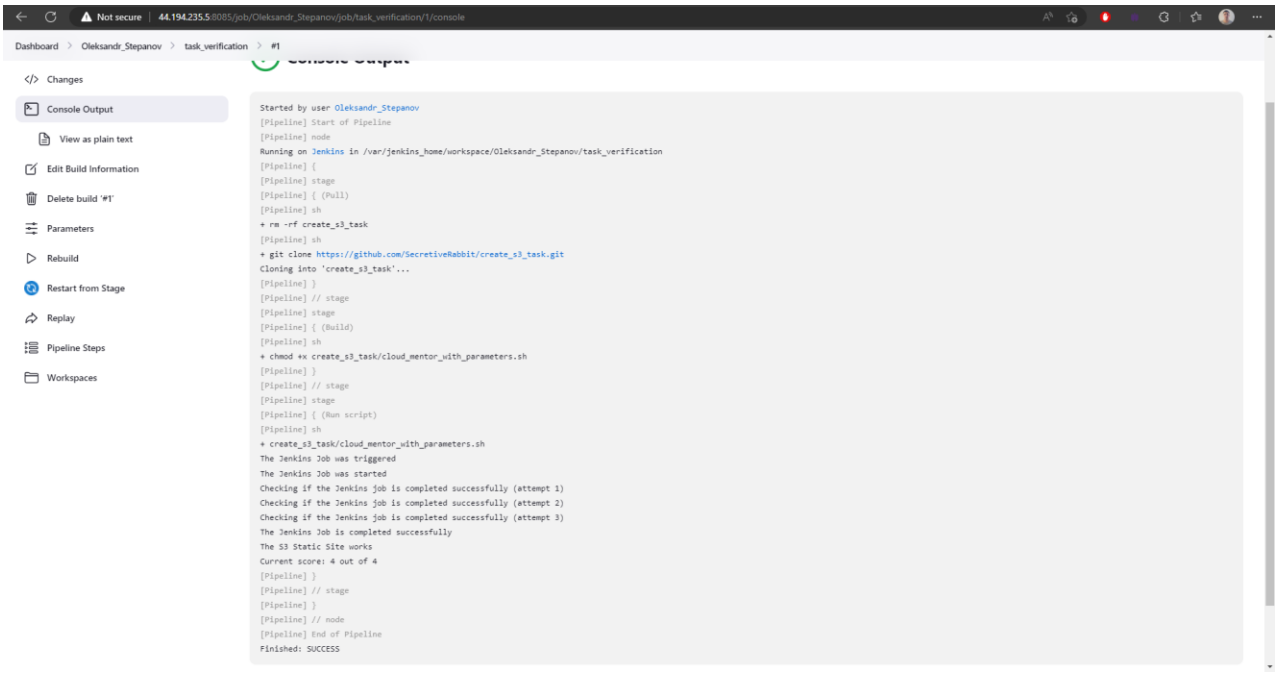

Рисунок 2.32 – Подробиці виконання пайплайну

Тепер необхідно перевірити, який буде отримано результат якщо свідомо провалити перевірку. Для цього змінемо URL для перевірки статичного веб сайту,

замість використання протоколу «http» будемо перевіряти «https»-версію веб-сторінки, якої, очевидно, нема, оскільки це не було в умовах завдання. Налаштування «https» потребує використання сертифікату, підписаного відповідальним лицем. Для зміни URL зайдемо на GitHub та поміняємо команду (2.33):

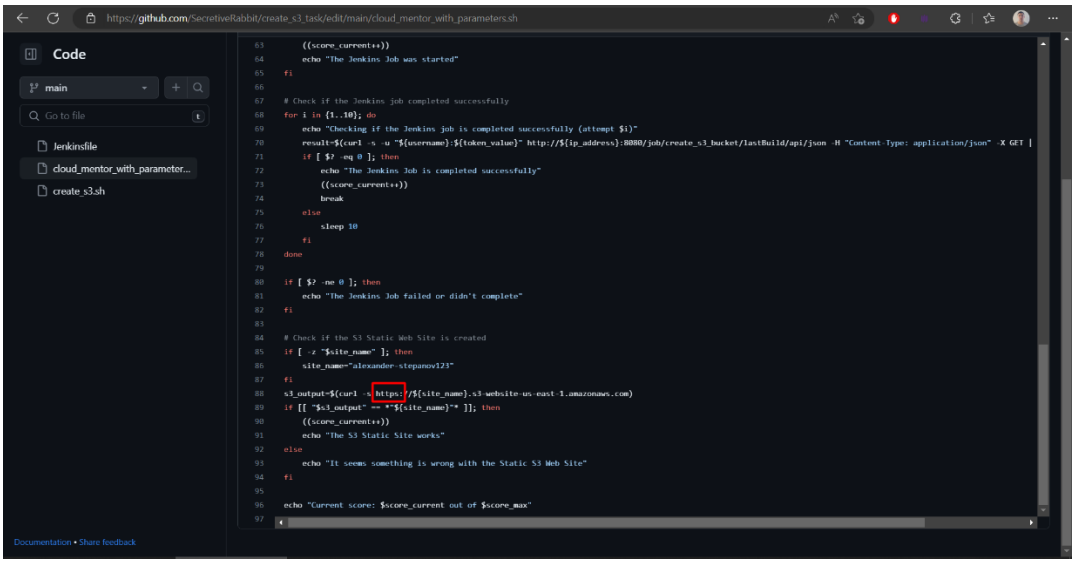

Рисунок 2.33 – Заміна «http» на «https»

Тепер подивимось який нам видадуть результат (2.34):

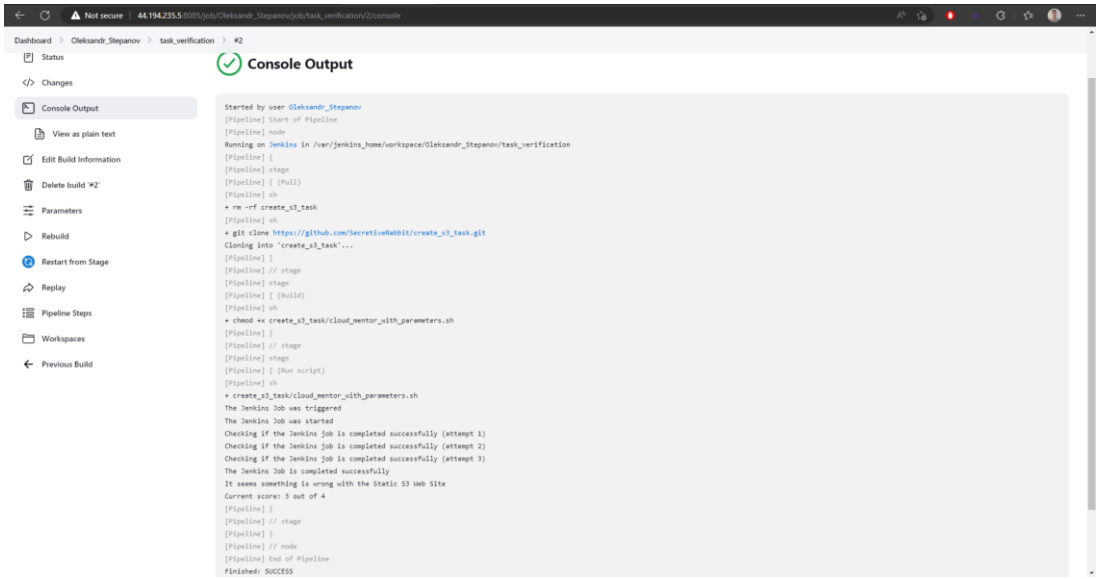

Рисунок 2.34 – Оцінка 3 з 4

Система видала нам оцінку 3 з 4, при цьому закінчившись успішно. Тому можна певною мірою стверджувати про надійність роботи розробленого рішення.

ВИСНОВКИ

У даній кваліфікаційній роботі було розглянуто питання недоліків сучасної системи освіти та як вони впливають на сучасну освіту, була запропонована концепція рішення актуальної проблеми у вигляді алгоритму, реалізованого за допомогою сучасних систем автоматизації. Також здійснен огляд системи CI/CD, яка була використана як основа для нашої роботи, підкреслені переваги. На основі даного інструменту була створена архітектура програмного рішення поставленої задачі.

Використовуючи Jenkins як платформу для автоматизації процесів розроблено завдання, яке використовує сучасні програмні засоби та які використовуються кваліфікованими спеціалістами на підприємствах. Завдання є схоже з тим, що наразі видають студентам вищих навчальних закладів. Розроблено для завдання працююче, еталонне рішення та поміщено у Jenkins, що знаходиться у хмарному середовищі.

Після цього було розроблено програмне рішення, яке здійснює перевірки студентських робіт. Виконуючи чотири базові перевірки, студенту виставляється оцінка. Цю програму також поміщено у систему автоматизації процесів.

Після реалізації концепції було здійснено тестування усього програмного комплексу. У якості перевірки був спочатку перший запуск Jenkins викладача, який запустив автоматизовний процес студента на виконання. Згодом, в цілях тестування системи виставлення оцінок, було здійснено навмисне псування роботи студента. Система пройшла перевірки, виставивши оцінку нижче максимальної.

Програму можна подалі вдосконалювати, але наразі вже може бути запропонована для подальшого впровадження у реальний процес навчання у сфері інформаційних технологій.

ПЕРЕЛІК ПОСИЛАНЬ

1. Джез Хамбл, Дейвід Фарлі, Безперервне розгортання ПЗ: автоматизація процесів складання, тестування та впровадження нових версій програм (Signature Series). – Сан Франциско: Print2print, 2020 – 432 с.

2. SonarQube [Електронний ресурс] - Режим доступу до ресурсу: https://docs.sonarqube.org/latest/.

3. Rafal L. Continuous Delivery with Docker and Jenkins / Leszko Rafal. – Livery Place, 2017. – 521 с. – (Packt Publishing Ltd.)

4. Jenkins [Електронний ресурс] – Режим доступу до ресурсу: https://www.jenkins.io/.

5. Програмування. Введення у професію. Том 1. Ази програмування. – Сан Франциско: Print2print, 2021 – 655 с.

6. Ількевич Н.С. Хмарні технології в освіті. Навчально-методичний посібник для студентів фізико-математичного факультету. – Житомир: вид-во ЖДУ, 2021. – 88 с.

7. AWS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/.

8. GitHub [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/features.

9. Лащевски Т., Арора К., Фарр Е. Хмарні Архітектури: розробка стійких хмарних додатків. – СПб .: Пітер. – 2021. - 320 с.

10. Jenkins installing [Електронний ресурс] – Режим доступу до ресурсу: https://www.jenkins.io/doc/book/installing/.

11. Колисниченко Д.Н. Linux. Повний посібник з роботи та адміністрування. – Москва: Наука і Техніка. – 2021. – 480 с.

12. Герберт Шилдт, Java. Повне керівництво. 12-е видання у 2-х томах. – Київ: Науковий світ. – 2022. – 1356 с.

13. Jenkins Pipeline. Що це і як використовувати. [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/companies/yoomoney/articles/538664/.

14. Арнольд Роббінс. Характеристики Bash. Кишеньковий довідник системного адміністратора. – Київ: Науковий світ. – 2023. – 152 с.

15. Створення персонального токену доступу [Електронний ресурс] – Режим доступу до ресурсу: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token.