

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіoeлектроніки
Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ
БЕЗПЕРЕРВНОЇ ІНТЕГРАЦІЇ І ДОСТАВКИ ДОДАТКІВ
У ХМАРНУ ПЛАТФОРМУ AWS

Виконав: студент 4 курсу, групи КНТ-527

Спеціальності 123 «Комп'ютерна інженерія»
(код і найменування спеціальності)

Освітня програма (спеціалізація)
«Комп'ютерна інженерія»

Фінько Владислав Сергійович
(прізвище та ініціали)

Керівник Скрупський С.Ю.
(прізвище та ініціали)

Рецензент Морщавка С.В.
(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет інформатики та радіоелектроніки, комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри Кудерметов Р.К.

« » 2021 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА

Фінька Владислава Сергійовича

(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Розробка комп'ютерної системи для безперервної інтеграції і доставки додатків у хмарну платформу AWS

керівник проекту (роботи) Скрупський Степан Юрійович, к. т. н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від « 17 » березня 2021 року № 81

2. Строк подання студентом проекту (роботи) 04 травня 2021 року

3. Вихідні дані до проекту (роботи) предметна область, загальні відомості про технології, комп'ютерна система, працездатна інфраструктура

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Опис предметної області. Аналіз технологій програмного забезпечення;

2) Аналіз апаратної та інфраструктурної частини комп'ютерної системи;

3) Розробка програмної частини комп'ютерної системи;

4) Випробування системи;

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

Креслення 1 - Блок-схема алгоритму роботи системи;

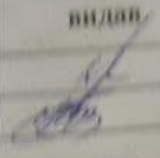
Плакат 1 - Макет проектування мережі;

Плакат 2 - Макет проектування сервісу для розгортання докер образу;

Плакат 3 - Макет проектування сервера Jenkins;

Плакат 4 - Макет результатів роботи комп'ютерної системи;

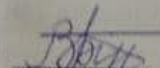
6. Консультанти розділів проекту (роботи)

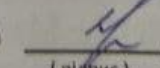
Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	Скрупський С.Ю., к. т. н., доцент		
Нормоконтроль	Щербак Н.В., ст. викладач		

7. Дата видачі завдання « 01 » березня 2021 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Срок виконання етапів проекту (роботи)	Примітка
1	Опис предметної області. Аналіз технологій програмного забезпечення	01.03.2021 р.	
2	Аналіз апаратної та інфраструктурної частини комп'ютерної системи	25.03.2021 р.	
3	Розробка програмної частини комп'ютерної системи	07.04.2021 р.	
4	Випробування системи	20.04.2021 р.	
5	Оформлення пояснювальної записки та графічної частини. Підготовка до захисту	30.04.2021 р.	

Студент  В. С. Фінко
(підпис) (ініціали та прізвище)

Керівник проекту (роботи)  С. Ю. Скрупський
(підпис) (ініціали та прізвище)

РЕФЕРАТ

ПЗ: 63 с., 24 рис., 4 таблиці, 24 лістинга, 36 джерел.

AWS, CI/CD, CPU, DOCKER, EC2, GITHUB, JENKINS, ПОБУДОВА ІНФРАСТРУКТУРИ, РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ

Об'єкт розробки – комп'ютерна система для безперервної інтеграції і доставки додатків у хмарну платформу AWS.

Мета роботи – розробка комп'ютерної системи для безперервної інтеграції і доставки додатків у хмарну платформу AWS.

Проект складається з чотирьох розділів.

Перший розділ присвячено опису предметної області безперервної інтеграції і доставки. Проводиться детальний огляд об'єкта. Визначається послідовність дій при розробці. Розглядаються технології з метою обрання найкращого кандидата для поставленої задачі.

У другому розділі виконується аналіз апаратної та інфраструктурної частини. Визначається тип екземпляра та тип диску, які надалі будуть використані при розробці.

У третьому розділі на підставі проведеного аналізу та опису предметної області проводиться розробка комп'ютерної системи, виконується побудова інфраструктури, встановлюється взаємодія з пайплайном безперервної інтеграції.

Четвертий розділ присвячено випробуванню готової комп'ютерної системи. За допомогою цього розділу з'являється можливість яка дозволяє визначити оптимальний варіант роботи системи вцілому та знайти можливості для покращення вже готових елементів.

ЗМІСТ

Скорочення та умовні позначки	6
Вступ.....	7
1 Опис предметної області	9
1.1 Опис об'єкту розробки	9
1.2 Створення організаційної моделі системи	10
1.3 Аналіз технологій безперервної інтеграції для реалізації комп'ютерної системи	12
1.4 Аналіз репозиторіїв для реалізації комп'ютерної системи.....	15
1.5 Аналіз систем контейнеризації та віртуалізації для розробки комп'ютерної системи	17
1.6 Аналіз хмарних технологій для реалізації комп'ютерної системи.....	20
2 Аналіз апаратного та інфраструктурного забезпечення комп'ютерної системи	23
3 Розробка комп'ютерної системи.....	30
3.1 Розробка інфраструктури у хмарі AWS	31
3.2 Побудова GitHub організації.....	45
3.3 Розробка пайплайну безперервної інтеграції	47
4 Випробування комп'ютерної системи	54
Висновки	59
Перелік джерел посилання	60

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

ПЗ	-	програма або безліч програм, що використовуються для управління комп'ютером
OCI	-	проект відкритих стандартів віртуалізації, Open Container Initiative
EBS	-	сховище на рівні блоків, Elastic Block Store
DNS	-	система доменних імен, Domain Name System
IAM	-	сервіс надає можливості безпечного управління доступом до ресурсів, Identity and Access Management
EC2	-	сервіс для оренди віртуальних серверів, Elastic Compute Cloud
IP	-	унікальний числовий ідентифікатор, Internet Protocol
CIDR	-	метод IP-адресації, Classless Inter-Domain Routing
CPU	-	інтегральна схема, Central Processing Unit
SSD	-	пристрій на основі мікросхем пам'яті, Solid-State Drive
ECS	-	сервіс оркестрації контейнерів, Elastic Container Service
URL	-	система уніфікованих адрес, Uniform Resource Locator

ВСТУП

Традиційно випуск нових версій ПЗ був пов'язаний з чималими труднощами, проблеми починалися на ранніх етапах розробки. Окремі групи працювали над своїми модулями незалежно один від одного і перед випуском нових версій повинні були спочатку інтегрувати їх разом. У процесі інтеграції, коли модулі вперше об'єднувалися один з одним, системи часто навіть не компілювалися. Часом доводилося витратити дні і навіть тижні, доки не вдавалося інтегрувати всі зміни і досягнути успішної компіляції. Тільки після цього можна було почати етап розгортання.

На наступному етапі розробникам доводилося витратити час, займаючись розгортанням нових версій ПЗ, створюючи тестові оточення. Тестувати програмні забезпечення в цих середовищах доводилося вручну. Найчастіше в ході тестування виникали помилки. Але на той час розробники вже були зайняті роботою над іншими проектами, і, щоб згадати деталі старих і виправити помилки, їм потрібен був час. Нарешті виправлений код необхідно було перенести в діюче оточення. Процес перенесення був дуже трудомістким, тому що робочі оточення компаній часто мали дуже складну організацію. Вигоди від впровадження якої-небудь однієї функції не виправдовували зусиль, що витрачалися на інтегрування і розгортання. Такі події вкрай рідко закінчувалися встановленням нової версії програмного забезпечення.

За допомогою революційних технологій безперервної інтеграції і безперервного розгортання можна запобігти подібним проблемам. Знижується трудомісткість інтеграції, система стає більш передбачуваною, неполадки виявляються і виправляються швидко, розгортання здійснюється частіше – аж до декількох разів на день. Завжди відбувається використання модульних тестів для останніх редагувань. Постійна наявність поточної стабільної версії разом з продуктами збірок – для тестування, демонстрації. Це скорочує час введення у дію

нових можливостей, що дає суттєву перевагу для бізнесу: він отримує можливість швидше реагувати на зміни ринку.

Тому, для розробки комп'ютерної системи безперервної інтеграції і доставки в хмарну платформу AWS було проведено:

- опис предметної області;
- аналіз технологій для реалізації комп'ютерної системи;
- аналіз апаратного та інфраструктурного забезпечення;
- розробка інфраструктури;
- розробка безперервної інтеграції;
- випробування готової системи.

1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис об'єкту розробки

CI/CD (Continuous Integration, Continuous Delivery – безперервна інтеграція і доставка) – це технологія автоматизації тестування та доставки нових модулів розроблюваного проекту зацікавленим сторонам (розробники, аналітики, інженери якості, кінцеві користувачі і ін.) [1];

Процес безперервної інтеграції програмного забезпечення необхідний. У проекті, що виконується однією людиною з небагатьма зовнішніми залежностями, інтеграція програмного забезпечення – не дуже істотна проблема, але при збільшенні складності проекту виникає нагальна потреба в інтеграції і перевірці злагодженої роботи компонентів програмного забезпечення, причому заздалегідь і часто. Чекати кінця проекту для проведення інтеграції та виявлення всього спектру можливих помилок – нерозумно. До того ж не сприяє якості програмного забезпечення, а найчастіше призводить до затримки передачі проекту на наступний етап розгортання [2];

У той же час без безперервного розгортання не обійтися. Більшість сучасних додатків складно розгортати внаслідок того, що в них багато взаємозалежних змінюваних частин. Багато організацій розгортають ПЗ вручну. Це означає, що процес розгортання розбивається на ряд атомарних стадій, кожна з яких виконується відповідальним виконавцем або командою. На кожній стадії потрібно приймати складні рішення, в результаті чого підвищується ймовірність помилки. І навіть, якщо не відбудеться помилок, відмінності в принципах управління різними стадіями призводять до нестабільних результатів. Ці відмінності не можуть вплинути на процес розгортання позитивним чином.

Для організації процесу CI/CD на виділеному сервері запускається служба, в завдання якої входять:

- отримання вихідного коду зі сховища;

- визначення гілки проекту;
- збірка проекту;
- виконання тестів;
- відправка образів у віддалений репозиторій;
- видалення використаних образів;
- розгортання готового проекту.

Якщо дотримуватися такої інструкції, можна забезпечити побудову стійкої та налагодженої роботи будь-якої системи, що збільшує прибуток компанії та зменшує час простою обладнання та робочої сили.

1.2 Створення організаційної моделі системи

CI/CD-конвеєр – складається з наступних етапів:

- написання коду. Закрита побудова виконується на окремих обчислювальних машинах перед передачею коду у сховище з контролем версій, для гарантії того, що внесені зміни не призведуть до помилки при інтеграційній побудові. Для контролю версій використовується система Git, або аналогічні рішення. Після того як учасники команди опублікують код своїх модулів в основній гілці, почнеться наступний етап [1];

- збірка. Система контролю версій запускає автоматичну збірку і тестування проекту. Тригери для початку збірки налаштовуються командою індивідуально – фіксація змін в основній гілці проекту, складання за розкладом, за запитом та ін. Для автоматизації збирання використовується Jenkins [1];

- ручне тестування. Коли CI система успішно перевірила працездатність тестової версії, то код відправляється тестувальникам для ручного обстеження. При цьому тестова збірка отримує номер кандидата для подальшого релізу продукту (наприклад, v.1.0.0-1) [1];

- реліз. За підсумками ручного тестування збірка отримує виправлення, а підсумковий номер версії кандидата підвищується (наприклад, після першого виправлення версія v.1.0.0-1 стає v.1.0.0-2). Після цього випускається версія коду для клієнта (наприклад, v.1.0.0) і починається наступний етап циклу [1];
- розгортання. На цьому етапі робоча версія продукту для клієнтів автоматично публікується на production серверах. Після цього клієнт може взаємодіяти з програмою і ознайомитися з її функціоналом як безпосередньо через готовий інтерфейс, так і через хмарні сервіси [1];
- підтримка і моніторинг. Кінцеві користувачі починають працювати з продуктом. Команда розробки підтримує його і аналізує досвід користувачів.
- планування. На основі призначеного для користувача досвіду формуються запити на новий функціонал для продукту, готується план доробок. Після цього цикл замикається і переходить у початкову стадію – написання коду. Далі починається нова ітерація CI/CD розробки [1];

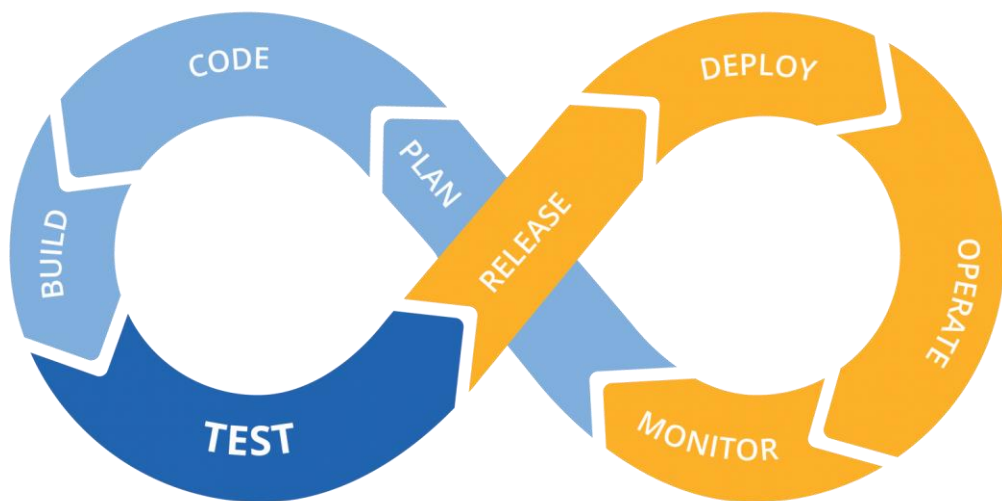


Рисунок 1.1 – CI/CD-конвеєр

В результаті ми отримали готовий CI/CD-конвеєр (рис. 1.1). За допомогою якого можна часто вносити зміни у додаток з надійним процесом поставки. Крім

стандартизації збірки, розробки тестів і автоматизації розгортання ми отримуємо цілісний виробничий процес по розгортанню змін коду. Впровадження CI/CD дозволяє розробникам зосередитися на поліпшенні додатків і не витратити сили на його розгортання [3];

1.3 Аналіз технологій безперервної інтеграції для реалізації комп'ютерної системи

Безперервна інтеграція можлива за допомогою декількох технологій. Найпопулярніші з яких Gitlab і Jenkins. Такі технології як CircleCI та Travis CI мають істотні недоліки, серед яких складне налаштування системи в деяких випадках, велика кількість помилок, а також низька гнучкість. Тим не менш, для наших цілей ці варіанти не підходять, тому порівнювати необхідно основні – Jenkins і GitLab, це дуже хороші інструменти, кожен з яких здатний забезпечити нормальну роботу CI/CD-конвеєра. Але, якщо їх порівняти, виявиться, що вони, хоча і багато в чому схожі, чимось один від одного відрізняються.

Jenkins програмна система з відкритим вихідним кодом, призначена для забезпечення процесу безперервної інтеграції програмного забезпечення. Відгалужена в 2008 році від проекту Hudson, що належить компанії Oracle, основним його автором – Косуке Кавагуті. Поширюється під ліцензією MIT [4];

До основних переваг Jenkins над іншими технологіями безперервної інтеграції відносять:

- підтримка безліч платформ. Jenkins може працювати на платформах macOS, Windows і Linux. Він може функціонувати і в середовищі Docker, що дозволяє організовувати однакове і швидке виконання автоматизованих завдань [5];

- просте встановлення і налаштування. Jenkins досить просто встановлювати і налаштовувати. При цьому і процес оновлення системи теж влаштований дуже

зручно. Варто згадати про якість документації, так як в ній можна знайти відповіді на різноманітні питання, пов'язані з встановленням і налаштуванням Jenkins [6];

- розвинена екосистема. Екосистема плагінів Jenkins виглядає більш розвинутою в порівнянні з екосистемами модулів інших CI/CD-інструментів. В даний час існує більше 1500 плагінів для Jenkins. Ці плагіни спрямовані на вирішення широкого спектра завдань, з їх допомогою можна автоматизувати найрізноманітніші проекти. Багатство вибору безкоштовних модулів означає, що у того, хто використовує Jenkins, немає гострої необхідності в покупці дорогих платних плагінів [5];

- підтримка роботи в розподілених середовищах. Jenkins дозволяє організовувати розподілені збірки з використанням декількох комп'ютерів. Ця можливість може бути застосована у великих проектах і використовує схему роботи, відповідно до якої існує один головний сервер Jenkins і кілька підлеглих машин. Підлеглі машини можуть використовуватися і в ситуаціях, коли потрібно організувати тестування проекту в різних середовищах. Ці можливості вигідно відрізняють Jenkins від інших подібних проектів [6];

Однак, як і будь-яка програмна система, Jenkins має свої недоліки:

- вартість. Потрібно виділений сервер, що тягне за собою додаткові витрати.
- зручність. Інтерфейс застарів і не зручний для користувача у порівнянні з поточними тенденціями призначеного для користувача інтерфейсу [7];

GitLab – веб-інструмент з відкритим вихідним кодом, який представляє систему управління репозиторіями коду для Git з власної вікі, системою стеження за помилками, CI/CD пайплайном і іншими функціями [8];

Спочатку GitLab був рішенням для управління вихідним кодом і спільної розробки. Пізніше GitLab перетворився в інтегроване рішення, яке допомагає розробникам вести безперервний процес розгортання для тестування, створення і розгортання коду. Стежити за ходом тестів, підвищувати контроль над якістю, фокусування на побудові продукту замість налаштування інструментів [8];

Основні переваги Gitlab:

- вартість. Завдяки автоматичному масштабуванню CI-раннер, відповідальних за виконання конкретних завдань, можна серйозно заощадити на вартості оренди серверних потужностей. Це дуже важливо, особливо, якщо мова йде про середовища, де тестування проектів виконується паралельно. Крім того, це важливо для великих проектів, що складаються з декількох репозиторіїв [5];

- зручність. Результати випробувань зручно аналізувати у інтерфейсі системи.

- підтримка системи контролю версій. GitLab CI/CD – це система, яка здатна працювати не тільки з кодом, розміщеним у репозиторіях GitLab. Наприклад, код можна зберігати у GitHub-репозиторії, а CI/CD-конвеєр можна організувати на базі GitLab з використанням GitLab CI/CD [5];

- наявність засобів для відстеження проблем. GitLab CI/CD дозволяє виконувати паралельне тестування різних гілок коду [5];

Проте в Gitlab є свої недоліки. Серед основних:

- тестування. Не можна протестувати результати об'єднання гілок до їх фактичного об'єднання.

- при описі стадій CI/CD-конвеєра в них поки не можна виділяти окремі етапи.

Незважаючи на значні плюси Gitlab, в питаннях безперервної інтеграції Jenkins виграє. З іншого боку, все більше організацій впроваджують GitLab у свої проекти.

І Jenkins, і Gitlab мають свої переваги та недоліки. Дискусія, яка технологія краще, тільки почалася, і напевно чи вона закінчиться найближчим часом.

Для реалізації безперервної інтеграції, я обрав технологію Jenkins. Важливим аргументом при прийнятті рішення для мене стали особисті навички та досвід роботи с Jenkins, а також різноманітний набір плагінів.

1.4 Аналіз репозиторіїв для реалізації комп'ютерної системи

Для розробки комп'ютерної системи необхідний репозиторій для збереження файлів проекту. До найпопулярніших відносять GitHub, Bitbucket, RhodeCode, Gogs, Visual Studio Team Service.

Для аналізу особливостей та визначення переваг та недоліків було обрано два веб-сервіси.

GitHub – найбільший в світі хостинг для зберігання і роботи з ІТ-проектами. GitHub виконує свої функції, дозволяючи легко обмінюватися кодом між декількома комп'ютерами і розробниками. Він став інструментом централізованої організації спільноти open source і, в свою чергу, використовується тисячами компаній і команд [9];

Серед головних особливостей GitHub, які визначають його одним із найпопулярніших репозиторіїв для збереження коду наступні:

- pull request. Функція дає можливість відкрити нову гілку з обговоренням відправленого коду. За допомогою якої, автор оригінального проекту, а також інші його учасники, можуть брати участь в обговоренні запропонованих змін до тих пір, поки автор проекту не буде ними задоволений. Після чого автор проекту може додати запропоновані зміни у проект [10];

- форкінг сховища. Функція дозволяє створювати свої власні розгалуження, вносити туди зміни, а потім відправляти свої внесені зміни в оригінальний репозиторій проекту, шляхом створення запиту на прийняття змін. Таким чином, проекти не стурбовані тим, щоб користувачі, які хотіли б виступати у ролі співавторів, мали право на внесення змін шляхом їх відправки [10];

- корпоративні рішення. GitHub пропонує можливості рівня підприємства для груп розробників і організацій.

- висока сумісність. Більшість послуг хмарного хостингу ідеально підходять для GitHub. Будь-який проект, розташований на цьому хостингу вихідного коду, може бути налаштований для цих сервісів протягом декількох секунд [12];

- відстеження помилок. GitHub інтегрує цю функцію, щоб спростити пошук і виправлення помилок. Щоб зрозуміти, що потрібно зробити, розробник просто відкриває панель кожного проекту і фільтрує інформацію [11];

- розумний пошук. GitHub дозволяє легко знаходити проекти з відкритим вихідним кодом. Також можна знайти інформацію, що стосується вивчення і пошуку проектів з відкритим вихідним кодом і їх об'єднання в групи. Крім того, GitHub показує сховища трендів і сортує їх за днями, тижнями або місяцями [12];

Bitbucket – веб-сервіс для хостингу проектів і їх спільної розробки. За призначенням і основним пропонованим функціям аналогічний GitHub, від якого відрізняється з одного боку, меншою користувальницькою базою. З іншого боку, має певні переваги у плані розміщення непублічних репозиторіїв – можливістю їх безкоштовного хостингу з обмеженням на розмір команди не більше п'яти осіб і меншою орендною платою при великому розмірі команди. А також управління правами доступу на рівні окремих гілок проекту [13];

Серед головних особливостей Bitbucket:

- безпека. У Bitbucket Premium включені розширені можливості забезпечення безпеки, що передбачають необхідність двофакторної аутентифікації [14];

- спеціальний пошук по коду. Швидкий аналіз пошукових результатів завдяки семантичному пошуку, який має в своєму розпорядженні визначення на початку списку [14];

- права доступу до гілок. Можливість точного контролю доступу в рамках команди. Дозволяє встановлювати права доступу до певної гілки для групи людей.

- зручність. Bitbucket має простий веб-інтерфейс, і його досить легко знайти або перейти до відповідної гілки, запитам на витяг, або будь-яким іншим функціям інструменту [15];

- вбудовані обговорення. Bitbucket пропонує відмінну рису для взаємодії з розробниками, відому як вбудовані обговорення. Вбудовані коментарі і бесіда можуть бути зроблені всередині фрагмента коду [14];

Одне з основних відмінностей між GitHub і BitBucket полягає у тому, що GitHub в основному фокусується на зростанні своєї величезної спільноти з відкритим вихідним кодом, тоді як BitBucket прагне до реєстрації більшої кількості корпоративних клієнтів. Проте, обидві платформи в останні роки досягли величезного успіху: BitBucket може похвалитися більш ніж 10 мільйонами користувачів, а GitHub – лідер з більш ніж 50 мільйонами користувачів.

Обидва інструменти відмінно підходять для розробки комп'ютерної системи. Кожний варіант має свої переваги та недоліки над іншими, однак базові можливості мають усі.

Тому, на основі аналізу та особистого рішення інструментом для збереження файлів проекту було обрано GitHub. Головні фактори такого вибору популярність веб-сервісу та велика підтримка спільноти.

1.5 Аналіз систем контейнеризації та віртуалізації для розробки комп'ютерної системи

Безперервне розгортання засноване на віртуальних машинах, тому що практично неможливо використовувати фізичні машини для підтримки різних етапів CI/CD-конвеєра. Віртуальні машини повністю відокремлені один від одного. Кожна має власні віртуальні жорсткі диски і власну операційну систему. Однак, щоб звести до мінімуму зусилля по підготовці оточень для додатка, зазвичай на всіх машинах встановлюється одна і та ж операційна система з однаковим набором інших компонентів оточень. Додаток у більшості випадків вимагає наявності кількох машин, наприклад сервера баз даних і сервера додатків [2];

Такий підхід тягне розпорошення ресурсів: віртуальні машини в значній мірі ідентичні, але налаштовуються абсолютно незалежно один від одного. Необхідно, щоб була можливість швидко підготувати і запустити нову віртуальну машину, стару віртуальну машину зі старим програмним забезпеченням можна було б просто видалити і користуватися новою. Це значно спростило б встановлення програмного забезпечення. Тому метод контейнеризації найкраще підходить у даному випадку [2];

Провівши детальний аналіз, для порівняння можливостей були обрані дві технології Docker та Podman. Кожна з них має свої особливості, тому необхідно визначити найкращого кандидата для вирішення поставленої задачі.

Docker – це відкрита платформа для розробки, доставки і експлуатації додатків, розроблена для більш швидкого викладання додатків. За допомогою Docker можна відокремити додаток від інфраструктури і звертатися з інфраструктурою як з керованим додатком. Docker допомагає викладати код швидше, швидше тестувати, швидше викладати додатки і зменшити час між написанням коду і запуском коду. Docker робить це за допомогою легковагої платформи контейнерної віртуалізації, використовуючи процеси і утиліти, які допомагають керувати і викладати додатки [17];

Docker, мабуть одна з найпопулярніших платформ для автоматизації розгортання. Вона має декілька особливостей, які надають їй перевагу над іншими схожими платформами. А саме:

- ізоляція. Додатки та ресурси ізольовані і розділені. Кожен контейнер має свої власні ресурси, які ізольовані від інших контейнерів. Можна мати різні контейнери для окремих додатків, які запускають абсолютно різні стеки. Docker забезпечує чисте видалення додатків, оскільки кожен додаток запускається у своєму власному контейнері [16];

- стандартизація. Docker надає відтворювані середовища розробки, складання, тестування і виробництва. Стандартизація інфраструктури обслуговування по всьому конвеєру дозволяє кожному члену команди працювати

в середовищі виробничого контролю. Завдяки цьому, інженери отримують більше можливостей для ефективного аналізу та виправлення помилок у додатку [16];

- версіонування. Контейнери Docker дозволяють фіксувати зміни у зображеннях і контролювати їх версію. Наприклад, якщо виконується оновлення компонента, яке порушує усе середовище, дуже легко виконати відкат до попередньої версії образу. Весь цей процес може бути перевірений за декілька хвилин [17];

- просте викладання і розгортання. Docker контейнери мають змогу працювати на локальній машині, як реальній так і віртуальній, як у дата-центрі, так і у хмарі [17];

Podman – це контейнерний тягач з відкритим вихідним кодом. Він керується Red Hat на відмінно від Docker не потребує якого-небудь сервісу для запуску контейнерів. Уся робота проводиться через бібліотеку libpod. Потребує OCI-сумісне runtime для запуску контейнерів. Обидва інструменти роблять одне і те ж: керують зображеннями і контейнерами [18];

Головні особливості, які привертають увагу інженерів до цієї платформи наступні:

- безпека. Одним з основних переваг використання Podman є підвищена безпека. Контейнери під керуванням Podman можуть запускатися як root, так і звичайним користувачем, зберігаючи майже всі функціональні можливості. Це велика перевага, оскільки можна запускати контейнери з різними користувачами і з різними привілеями [19];

- demonless. Podman працює з процесом виконання контейнера runC, який запускається безпосередньо в ядрі Linux, без використання демона. Замість демона Podman запускає контейнери і модулі як дочірні процеси, Podman взаємодіє з libpod, бібліотекою, яка дозволяє іншим інструментам керувати модулями і контейнерами [19];

- управління зображеннями. Образи, створені Podman, сумісні з іншими інструментами управління контейнерами, які відповідають стандарту OCI, отже, їх можна відправити в інші реєстри контейнерів [19];

На основі проведеного аналізу та особистих переваг, для розробки комп'ютерної системи я обрав Docker. Головними чинниками у прийнятті рішення стали популярність та особисті навички з цією технологією.

1.6 Аналіз хмарних технологій для реалізації комп'ютерної системи

Доставка додатків буде здійснюватися на одну з хмарних платформ. Серед найпопулярніших хмарних платформ розрізняють: AWS, Microsoft Azure, GCP, IBM Cloud, VMWare vCloud Suite, Sharepoint Online тощо. На основі особистого досвіду, спектр вибору було зменшено до двох кандидатів AWS та Microsoft Azure.

Amazon Web Services – комерційна публічна хмара, яка підтримується і розвивається компанією Amazon з 2006 року. Надає передплатникам послуги як по інфраструктурній моделі, так і платформного рівня. В значній мірі вплинуло на формування концепції хмарних обчислень у цілому, і визначило основні напрямки розвитку публічної моделі розгортання [20];

Сервіси AWS володіють неперевершеними характеристиками, включають ретельно продумані рішення, гарантують надійність, безпеку і продуктивність для додатків. Понад 14 років AWS поставляє хмарні сервіси мільйонам клієнтів по всьому світу для використання у різноманітних цілях. AWS пропонує найнадійніші на ринку рішення для роботи у будь-якому масштабі, що робить її безсумнівним лідером серед постачальників хмарних послуг [21];

До основних переваг AWS відносяться:

- загальне сховище даних. Amazon Elastic File System надає просту, масштабовану, повністю керовану і еластичну файлову систему NFS, яку можна

використовувати з сервісами AWS Cloud і локальними ресурсами. Файлова система масштабується на вимогу, не перериваючи роботу додатків. Вона автоматично розширюється і стискається в залежності від додавання або видалення файлів, тому не потрібно виділяти додаткові ресурси і управляти ними в міру зростання [22];

- масштабованість і висока продуктивність. Такі засоби AWS як Auto Scaling і Elastic Load Balancing забезпечують масштабування додатків в бік збільшення або зменшення з урахуванням поточних вимог. Завдяки великій інфраструктурі Amazon можна отримати доступ до обчислювальних ресурсів і ресурсів сховищ саме тоді, коли вони будуть потрібні [23];

- безпека. Сервіс AWS Identity and Access Management надає можливості безпечного управління доступом до сервісів та ресурсів. Використовуючи IAM, можна створювати користувачів і групи, керувати ними, а також використовувати дозвіл, щоб надавати або забороняти доступ до ресурсів AWS [24];

- інфраструктура як код. AWS CloudFormation пропонує простий спосіб моделювання колекції супутніх ресурсів AWS і сторонніх ресурсів, їх швидкого і однакового виділення, а також управління ними на протязі всього життєвого циклу за рахунок роботи з інфраструктурою як з кодом. Шаблон CloudFormation описує необхідні ресурси і їх залежності, що дозволяє запускати і налаштовувати їх разом у вигляді стека [25];

Серед недоліків AWS наступні:

- несумісна і слабка гібридна стратегія.
- незважаючи на те, що AWS робить акцент на економічну ефективність, він досить дорогий в порівнянні з Azure [26];

Microsoft Azure – хмарна платформа компанії Microsoft. Надає можливість розробки, виконання додатків і зберігання даних на серверах, розташованих в розподілених дата-центрах [27];

Основні переваги Microsoft Azure:

- зниження витрат. Microsoft Azure – це оптимальне рішення для періодичних робочих навантажень. Деякі програми потрібно виконувати лише час від часу. За

допомогою Windows Azure можна легко створити і розгорнути такий додаток, а також зробити його загальнодоступним. Після того, як мета досягнута, додаток можна безперешкодно видалити і оплатити тільки той час, протягом якого воно було розгорнуто [28];

- резервне копіювання. Azure пропонує комплексне рішення резервного копіювання та відновлення, яке відрізняється простотою, надійністю, масштабованістю і економічністю і може бути інтегровано з локальними рішеннями для захисту даних. У разі збоїв у роботі служб або ненавмисного видалення або пошкодження даних, можна швидко і злагоджено відновити необхідний проект [29];

- обчислення. Сервіс Virtual Machines дозволяє користувачам запускати віртуальні машини загального призначення під управлінням Microsoft Windows, Linux, а також передналаштовані образи популярних сервісних пакетів [27];

- можливість формування гібридного середовища. Azure володіє неперевершеними гібридними можливостями, які роблять його унікальним і незвичайним для багатьох популярних постачальників хмарних послуг. Це забезпечує легку мобільність і створює надійну платформу між публічною хмарою і локальними користувачами [30];

- практично необмежена потужність. Будь-який проект у потрібний йому час може додати собі стільки ресурсів, скільки потрібно для виконання завдання. При цьому хмара буде також стабільно і чітко працювати з усіма іншими важливими проектами [31];

Серед недоліків Microsoft Azure:

- досить часті перебої у роботі сервісів.
- багато організацій незадоволені якістю технічної підтримки Microsoft.

Отже, на основі проведеного аналізу можна зробити висновок, що обидві хмарні платформи мають свої сильні і слабкі сторони: AWS і Microsoft Azure зараз найбільш затребувані хмарні платформи. Пропоновані обчислювальні потужності в компаній перебувають практично на рівних, список пропонованих сервісів також

постійно зростає. Все ж вирішальним фактором вибору для мене є відмовостійкість та наявність сервісу CloudFormation за допомогою якого легко моделювати колекції ресурсів AWS. Порівняння властивостей хмарних технологій та технологій безперервної інтеграції наведені у таблицях 1.1-1.2.

Таблиця 1.1 – Порівняння технологій Jenkins та Gitlab CI/CD

Властивість	Jenkins	Gitlab CI/CD
Просте встановлення та налаштування	+	-
Підтримка багатьох платформ	+	-
Розвинена екосистема	+	-
Підтримка роботи в розподілених середовищах	+	-
Вартість	-	+
Зручність	-	+
Підтримка системи контролю версій	-	+

Таблиця 1.2 – Порівняння технологій AWS та Microsoft Azure

Властивість	AWS	Microsoft Azure
Розвинена гібридна стратегія	-	+
Безпечне управління доступом до сервісів	+	-
Технічна підтримка	+	-
Вартість	-	+
Вбудований сервіс для моделювання ресурсів	+	-

2 АНАЛІЗ АПАРАТНОГО ТА ІНФРАСТРУКТУРНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Для реалізації комп'ютерної системи необхідно визначити тип екземпляра та тип диску, які надалі будуть використані при розробці.

Екземпляр T2 – це екземпляр з підвищеною продуктивністю, що забезпечує базовий рівень продуктивності CPU з можливістю його підвищення.

Безлімітні екземпляри T2 можуть підтримувати високу продуктивність CPU до тих пір, поки це вимагає робоче навантаження. Для більшості робочих навантажень, загального призначення, безлімітні екземпляри T2 забезпечують достатню продуктивність без додаткової плати. Якщо потрібно видати більше ресурсів CPU на екземплярі, протягом тривалого часу, це також легко можна зробити [32];

Перевагою використання цього типу значень є:

- високочастотні процесори Intel Xeon;
- постійний базовий рівень продуктивності та CPU з можливістю підвищення продуктивності (регулюється кредитами CPU);
- економічний тип екземплярів спільного призначення. Доступний на рівні безкоштовного користування;
- збалансоване співвідношення обчислювальних, мережевих ресурсів та ресурсів пам'яті [32];

Базовий рівень продуктивності та можливість його підвищення визначають кредити CPU. Екземпляри T2 регулярно отримують певне число кредитів, які залежать від розміру екземпляру. Вони накопичують кредити CPU за час простою і споживають їх у активному стані. Екземпляри T2 добре підходять для різноманітних робочих навантажень загального призначення, включаючи мікросервіси, інтерактивні додатки з низькою зарядкою, малі та середні бази даних, віртуальні робочі столи, середовища розробки, збірки та тестування, репозиторії кодів та прототипи продуктів [32];

Кожен екземпляр з підвищеною продуктивністю безперервно заробляє (з дозволом на рівні мілісекунд) встановлену ставку кредитів CPU на годину, в залежності від розміру примірника. Процес обліку нарахування чи витрачання кредитів також відбувається з роздільною здатністю у мілісекундах. Тому не потрібно турбуватися про перевитрату кредитів процесора. Короткий сплеск процесора використовує невелику частку кредиту процесора [32];

Якщо екземпляр з підвищеною продуктивністю використовує менше ресурсів CPU, ніж потрібно для базового використання (наприклад, коли він перебуває у режимі очікування), невитрачені кредити накопичуються на кредитному балансі CPU. Якщо екземпляр повинен перевищити базовий рівень використання, він витрачає нараховані кредити. Чим більше кредитів накопичено екземпляром підвищеної продуктивності, тим більше часу він зможе перебувати за межами базового рівня, коли потрібне більше використання процесора [36];

Коли екземпляр з підвищеною продуктивністю, налаштований за допомогою стандартних конфігурацій, знаходячись у робочому стані, він безперервно заробляє встановлену норму зароблених кредитів на годину. Для T2 Standard, при зупинці екземпляра, він втрачає усі накопичені кредити. Його кредитний баланс скидається до нуля. При перезавантаженні він отримує новий набір стартових кредитів і починає знову накопичувати зароблені кредити [36];

Екземпляри T2 Standard отримують два типи кредитів CPU: зароблені кредити і кредити на запуск. Коли екземпляр T2 Standard знаходиться у робочому стані, він безперервно заробляє (з точністю до мілісекунди) встановлену норму зароблених кредитів на годину. Спочатку він немає кредитів для нормального функціонування. Тому, щоб забезпечити хороший поштовх, екземпляр отримує початкові кредити при старті, які витрачаються у першу чергу, поки він накопичує зароблені кредити [33];

Надалі описані метрики для екземплярів з підвищеною продуктивністю:

- CPUCreditUsage. Кількість кредитів, витрачених екземпляром на використання процесором. Один кредит дорівнює одному віртуальному CPU, що працює зі 100% завантаженням протягом однієї хвилини, або еквівалентної комбінації віртуальних CPU, використання і часу. Показники кредиту доступні тільки з п'ятихвилинної періодичністю. Для того, щоб вказати період більше п'яти хвилин, використовується статистика «Сума» замість статистики «Середнє» [33];

- CPUCreditBalance. Кількість зароблених кредитів, накопичених примірником з моменту його запуску. Для T2 Standard CPUCreditBalance також

включає кількість нарахованих кредитів. Кредити нараховуються на кредитний баланс після того, як вони зароблені. Знімаються з кредитного балансу, коли вони витрачені. Кредитний баланс має максимальну межу, що визначається розміром примірника. Після досягнення ліміту усі нові зароблені кредити скидаються. Для T2 Standard стартові кредити не зараховуються у ліміт. Кредити CPUCreditBalance доступні для екземпляра, які можна витратити, щоб перевищити базове завантаження CPU [33];

- CPUSurplusCreditBalance. Кількість надлишкових кредитів, які були витрачені екземпляром, коли його значення CPUCreditBalance дорівнює нулю. Значення CPUSurplusCreditBalance оплачується за рахунок зароблених кредитів CPU. Якщо надлишкові кредити перевищують максимальну кількість, що екземпляр може заробити за двадцяти чотирьох годинний період, то стягується додаткова плата [33];

- CPUSurplusCreditsCharge. Кількість витрачених надлишкових кредитів, які не виплачуються заробітними кредитами CPU, і які вимагають додаткової оплати. Витрачені надлишкові кредити нараховуються, у випадку, якщо витрачені надлишки кредитів перевищують максимальну кількість, яку екземпляр може заробити за 24 години [33];

Після того як було визначено тип екземпляру необхідно визначитися з типом EBS.

Amazon Elastic Block Store (Amazon EBS) пропонує томи сховищ на рівні блоків для використання з екземпляром EC2. Томи EBS поведуть себе як неформатизовані блочні пристрої. Можливо підключити їх як пристрої до екземпляра. Томи EBS, підключені до екземпляра, відображаються як томи сховища, які зберігаються, незалежно від терміну дії екземпляра. Також є можливість створити файлову систему поверх цих томів або використовувати їх так, як використовується блоковий пристрій [34];

Томи EBS, приєднані до запущеного екземпляра, можуть автоматично від'єднуватися від екземпляра залишаючи недоторканими дані. Для цього

необхідно зняти прапорець Delete on Termination, при налаштуванні томів EBS для екземпляра у консолі EC2. Том можна потім приєднати до нового екземпляра, що забезпечує швидке відновлення. Якщо встановлено прапорець Delete on Termination, томи видалятимуться після видалення екземпляра EC2. Якщо використано екземпляр з підтримкою EBS, можна зупинити та перезапустити цей екземпляр, не впливаючи на дані, що зберігаються у вкладеному томі. Том залишається прикріпленим протягом циклу зупинки-запуску. Це дозволяє необмежено обробляти та зберігати дані на томі, лише використовуючи ресурси обробки та зберігання, коли це потрібно. Дані зберігаються на томі, доки обсяг не буде видалено явно. Зберігання фізичного блоку, що використовується видаленими томами EBS, перезаписується нулями перед тим, як його розподілити на інший обліковий запис [34];

Є варіант створення тому EBS у певній зоні доступності, а потім приєднання його до екземпляру в тій же зоні доступності. Щоб зробити цей том досяжним за межами зони доступності, можливо створити миттєвий знімок і відновити цей миттєвий знімок на новому томі у будь-якому місці цього регіону. Можливо скопіювати миттєві знімки у інші регіони, а потім відновити їх на нових томах, щоб використати декілька регіонів AWS для географічного розширення, міграції центрів обробки даних та аварійного відновлення [34];

Можливо створювати EBS-томи як зашифровані. При створенні зашифрованого тому EBS і приєднанні його до підтримуваного типу примірника, дані, що зберігаються у неактивному стані на томі, дискові операції введення-виведення і моментальні знімки, створені з тому, зашифровуються. Шифрування відбувається на серверах, на яких розміщені екземпляри EC2, забезпечуючи шифрування даних при передачі від примірників EC2 у сховище EBS [34];

Amazon EBS пропонує наступні типи томів: SSD загального призначення, SSD з виділеним IOPS, HDD з оптимізованою пропускнуою здатністю та холодний HDD [32];

Далі представлена інформація про продуктивність та варіанти використання для кожного типу тома.

Томи SSD загального призначення (gp2 і gp3) врівноважують ціну і продуктивність для широкого спектра транзакційних робочих навантажень. Ці томи ідеально підходять для таких випадків використання, як завантажувальні томи, однокземплярні бази даних середнього розміру, а також засоби розробки та тестування [32];

Томи SSD з виділеним IOPS (io1 і io2) підтримують до 64 000 операцій введення-виведення за секунду та пропускну здатність 1000 МБ/с. Це дозволяє попередньо масштабувати до десятків тисяч операцій введення-виведення за секунду на екземплярі EC2 [32];

Томи жорстких дисків з оптимізованої пропускнуою спроможністю (st1) надають недорогі магнітні сховища, які визначають продуктивність з точки зору пропускнуої здатності, а не IOPS. Ці томи ідеально підходять для великих послідовних робочих навантажень, сховищ даних і обробки журналів [32];

Томи холодних жорстких дисків (sc1) представляють собою недорогі магнітні сховища, які визначають продуктивність з точки зору пропускнуої здатності, а не операцій введення-виведення за секунду. Ці томи ідеально підходять для великих послідовних робочих навантажень з холодними даними. Якщо потрібно нечастий доступ до даних і потрібно заощадити, ці томи забезпечують недороге блокове сховище [32];

Продуктивність томів gp2 пов'язана з розміром тому, який визначає базовий рівень продуктивності тому і швидкість накопичення кредитів введення-виведення (рис. 2.1). Великі томи мають більш високий базовий рівень продуктивності і швидше накопичують кредити введення-виведення. Кредити введення-виведення представляють доступну смугу пропускання, яку том gp2 може використовувати для великої кількості операцій, коли потрібно продуктивність, що перевищує базову. Чим більше кредитів том має для введення-виведення, тим більше часу він

може знаходитися за межами базового рівня продуктивності і тим краще він буде працювати, коли потрібна велика продуктивність [32];

Початковий кредитний баланс призначений для забезпечення швидкого початкового циклу для завантажувальних томів і для надання зручного початкового завантаження для інших додатків [32];

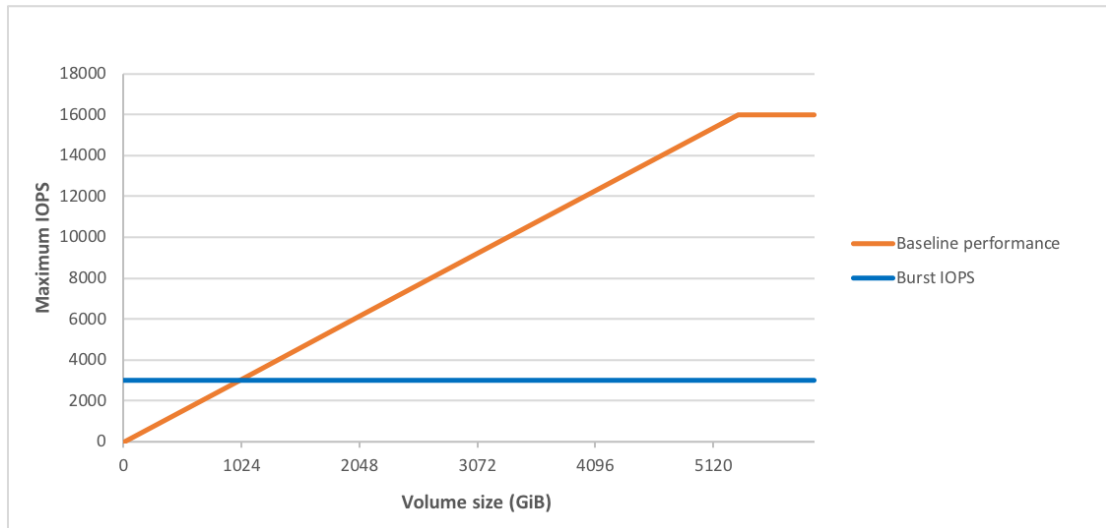


Рисунок 2.1 – Залежність продуктивності тому від його розміру

Перевірки станів у томах дозволяють краще розуміти, відстежувати і керувати потенційними невідповідностями даних на томах Amazon EBS. Вони призначені для надання інформації, необхідної для визначення того, чи пошкоджені томи Amazon EBS, і для допомоги в управлінні обробкою потенційно неузгоджених томів [35];

Перевірки стану тому – це автоматичні тести, які запускаються кожні 5 хвилин і повертають статус «пройшов» або «не пройшов». Якщо всі перевірки пройдені, стан тому у нормі. Якщо перевірка не вдалася, стан тому погіршується. Якщо статус – «недостатньо даних», то перевірка тому може тривати [35];

Коли Amazon EBS визначає, що дані томи потенційно несумісні, за замовчуванням він відключає введення-виведення для цього тому з будь-яких підключених екземплярів EC2, що допомагає запобігти пошкодженню даних [35];

Для побудови сервера необхідний мінімальний набір конфігурацій, що зображено в таблицях 2.1-2.2.

Таблиця 2.1 – Характеристика типу екземпляра

Name	vCPU	RAM	Arch
t2.micro	1	1GB	64-bit

Таблиця 2.2 – Характеристика типу диску

Name	Volume size	Max.IOPS	Max throughput
gp2	1 GB - 16 TB	16,000	250 MB/s

В результаті було проаналізовано апаратне та інфраструктурне забезпечення комп'ютерної системи.

3 РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ

Після опису предметної області та аналізу технологій настає головний етап розробки комп'ютерної системи. Запропонована комп'ютерна система складається з наступних трьох етапів:

- розробка інфраструктури у хмарі AWS;
- створення GitHub організації;
- розробка пайплайну безперервної інтеграції;

Саме у такому порядку розроблялася комп'ютерна система у даній дипломній роботі. Важливим пріоритетом стало визначення основних сервісів AWS для побудови інфраструктури, та взаємодія з пайплайном безперервної інтеграції, який напряду пов'язаний з GitHub організацією.

3.1 Розробка інфраструктури у хмарі AWS

На першому етапі нам необхідно підготувати AWS інфраструктуру для Jenkins instance. Для цього в CloudFormation нам необхідно створити свою мережу. Створювати ми будемо її за допомогою сервісу VPC, який надасть нам можливість запускати ресурси в AWS, у повністю ізольованому віртуальному середовищі. В результаті ми отримаємо можливість повністю контролювати середовище віртуальної мережі, в тому числі вибирати власний діапазон IP-адрес, створювати підмережі, а також налаштовувати таблиці маршрутів і мережеві шлюзи (рис. 3.1). Код для налаштування мережі наведений у лістингу 3.1.

Лістинг 3.1 – Налаштування мережі

```
AWSTemplateFormatVersion: 2010-09-09
Description: 'Dedicated VPC for Jenkins build environment'
Resources:
  myVPC:
    Type: 'AWS::EC2::VPC'
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: true
  myInternetGateway:
    Type: 'AWS::EC2::InternetGateway'
  AttachGateway:
    Type: 'AWS::EC2::VPCGatewayAttachment'
    Properties:
      VpcId:
        Ref: myVPC
      InternetGatewayId:
        Ref: myInternetGateway
```

Перші два рядки лістингу – стандартні рядки налаштувань шаблону. В наступних рядках ми вказуємо назву ресурса та тип, який ми збираємося використовувати.

Наступний крок властивості. Тут ми перелічуємо всі необхідні нам елементи, які ми б хотіли додати до нашої мережі. Варто зазначити, що ресурси поділяються на обов'язкові та необов'язкові. Ми використовуємо обов'язкові ресурси завжди, так

як без них шаблон просто не буде працювати. На відмінно від необов'язкових, які ми додаємо за власним бажанням. CIDR блок нашої мережі буде 10.0.0.0/16.

Для наших цілей нам необхідно, щоб була можливість робити DNS-запис. Тому ми додали рядок `EnableDnsSupport`, та виставили його в дієздатне положення. Після цього, слід створити шлюз, та провести його до нашої мережі, для того щоб сервера в нашій мережі мали змогу виходити до інтернету.

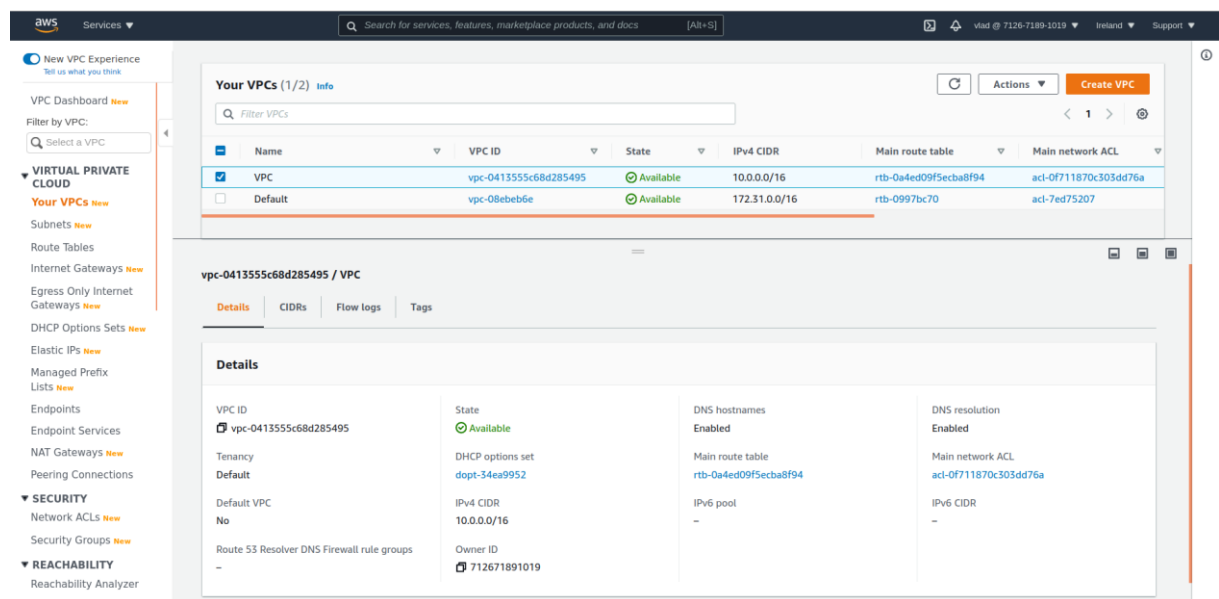


Рисунок 3.1 – Створення мережі

Після створення мережі необхідно сформувати підмережі. Сформуємо одразу дві, щоб у разі збою в одній підмережі instance одразу запустився в іншій (рис. 3.2). Перший крок для налаштування – це посилання до нашої головної мережі `myVPC` до двох підмереж за допомогою функції `Ref`. Внутрішня функція `Ref` повертає ідентифікатор зазначеного ресурсу. Далі ми вказуємо CIDR блоки наших підмереж. Кожній підмережі повинна відповідати одна `AvailabilityZone`. За допомогою функції `Select` та `GetAZs` отримуємо дві послідовні `AvailabilityZone`. Код для налаштування підмереж наведений у лістингу 3.2.

Лістинг 3.2 – Налаштування підмереж
`PublicSubnet1:`

```

Type: 'AWS::EC2::Subnet'
Properties:
  VpcId:
    Ref: myVPC
  CidrBlock: 10.0.11.0/24
  AvailabilityZone: !Select
    - 0
    - !GetAZs ' '
PublicSubnet2:
Type: 'AWS::EC2::Subnet'
Properties:
  VpcId:
    Ref: myVPC
  CidrBlock: 10.0.21.0/24
  AvailabilityZone: !Select
    - 1
    - !GetAZs ' '

```

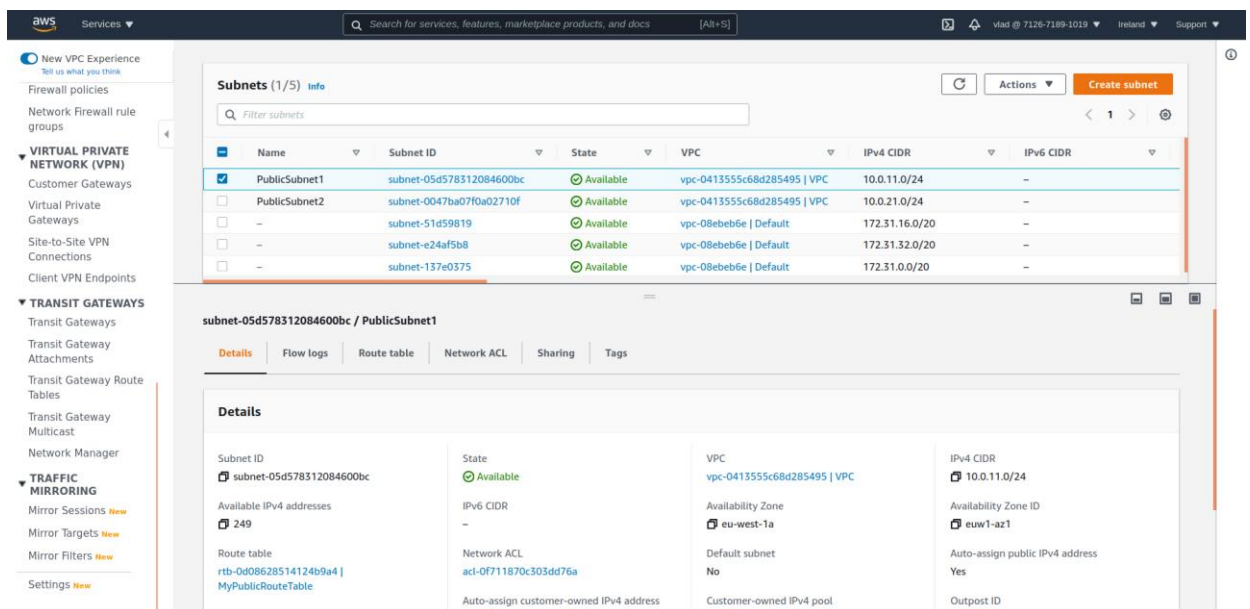


Рисунок 3.2 – Створення відкритих підмереж

Кожна підмережа повинна бути пов'язана з таблицею маршрутів. Таблиця маршрутів містить набір правил, які використовуються для визначення куди спрямовується мережевий трафік. Підмережа одночасно може бути пов'язана лише з однією таблицею маршрутів. Проте можна пов'язати кілька підмереж з однією таблицею маршрутів. Створимо маршрут та з'єднаємо його з інтернет-шлюзом. Код для налаштування таблиці маршрутів наведений у лістингу 3.3.

Лістинг 3.3 – Налаштування таблиці маршрутів

```
myPublicRouteTable:
  Type: 'AWS::EC2::RouteTable'
  Properties:
    VpcId:
      Ref: myVPC
myRoute:
  Type: 'AWS::EC2::Route'
  Properties:
    RouteTableId:
      Ref: myPublicRouteTable
    DestinationCidrBlock: 0.0.0.0/0
    GatewayId:
      Ref: myInternetGateway
```

Після створення таблиці маршрутів, та заповнення її конкретним маршрутом треба приєднати її до підмереж. Для цього ми використовуємо ресурс `SubnetRouteTableAssociation`, та прописуємо у цьому ресурсі властивості `SubnetId` та `RouteTableId` за допомогою яких ми посилаємося на необхідні нам елементи. Код для з'єднання підмереж з таблицею маршрутів наведений у лістингу 3.4.

Лістинг 3.4 – З'єднання підмереж з таблицею маршрутів

```
myPublicSubnetRouteTableAssociation1:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId:
      Ref: PublicSubnet1
    RouteTableId:
      Ref: myPublicRouteTable
myPublicSubnetRouteTableAssociation2:
  Type: 'AWS::EC2::SubnetRouteTableAssociation'
  Properties:
    SubnetId:
      Ref: PublicSubnet2
    RouteTableId:
      Ref: myPublicRouteTable
```

В результаті мережа повністю сформована. Тепер можна запустити шаблон. Отримаємо готовий стек с назвою `vpc`.

Наступний етап розробки інфраструктури – це створення IAM ролі, яка буде підключатись к Jenkins EC2 instance. Для цього пропишемо назву ресурса, а також властивості, серед яких `Statement` є основним елементом політики. Елемент

Statement може містити одиночний оператор або масив окремих операторів. Елемент Effect є обов'язковим і вказує, чи отримає оператор дозвіл, чи явне заперечення. Дійсні значення для ефекту – дозволити та заборонити.

Елемент Principal використовується у політиці, щоб вказати принцип, який дозволяє або забороняє доступ до певного ресурсу.

Елемент Action описує конкретну дію. У данному випадку дія Assume Role, яка повертає набір тимчасових облікових даних безпеки, які використовуються для доступу до ресурсів AWS. Для передачі ролі IAM екземпляру EC2 застосовуємо профіль екземпляра. Код для налаштування ролі доступу наведений у лістингу 3.5.

Лістинг 3.5 – Налаштування ролі доступу

```
JenkinsIAMRole:
  Type: 'AWS::IAM::Role'
  Properties:
    RoleName: JenkinsIAMRole
    Statement:
      - Effect: "Allow"
        Principal:
          Service:
            - "ec2.amazonaws.com"
        Action:
          - "sts:AssumeRole"
    ManagedPolicyArns:
      - arn:aws:iam::aws:policy/service-
role/AmazonEC2RoleforSSM
  JenkinsInstanceProfile:
    Type: "AWS::IAM::InstanceProfile"
    Properties:
      Roles:
        - !Ref 'JenkinsIAMRole'
```

На цьому створення ролі для instance завершено. Тепер можна запуснути шаблон, отримати готовий стек, та перейти до наступного етапу.

Після того, як ми створили мережу в якій буде розташовано наша віртуальна машина, та роль в якій було вказано повноваження які отримує екземпляр. Надалі необхідно сформуванати сам instance. Для цього спочатку треба передати параметри до шаблону де розробляється віртуальна машина. Параметри дозволяють вводити

власні значення у шаблон кожного разу, коли оновлюється стек. Код для налаштування параметрів наведений у лістингу 3.6.

Лістинг 3.6 – Налаштування параметрів

```
Parameters:
  myVPC:
    Type: String
    Default: vpc-0413555c68d285495
  MyPublicSubnet1:
    Type: String
    Default: subnet-05d578312084600bc
  MyPublicSubnet2:
    Type: String
    Default: subnet-0047ba07f0a02710f
  ParentIAMStack:
    Type: String
    Default: IAMRole
    Default: 1
  AmiId:
    Type: String
    Default: ami-06ce3edf0cff21f07
  CorporateIps1:
    Type: String
    Default: 195.211.150.4/32
```

Наступна дія – це опис ресурсів до шаблону, в якому розробляється instance. Спочатку, розробимо групу безпеки для балансувальника навантаження. При створенні нової групи безпеки вхідні правила відсутні. Отже, забороняється вхідний трафік, що надходить з іншого хосту доки не буде додано правила вхідного сигналу до групи безпеки. Тому додали ці правила. Після чого доступ відкритий на вісімдесятому порту для одної IP-адреси, яка вказана у параметрах шаблону. Трафік буде здійснюватися за протоколом TCP. Код для налаштування групи безпеки наведений у лістингу 3.7.

Лістинг 3.7 – Налаштування групи безпеки

```
JenkinsALBSG:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: ALB Security Group
    VpcId:
      Ref: myVPC
```

```
SecurityGroupIngress:
- IpProtocol: tcp
  CidrIp: !Ref CorporateIps1
  FromPort: 80
  ToPort: 80
```

Load balancer необхідний для автоматичного розподілу вхідного трафіку додатків по декількох цільових об'єктах в одній або декількох зонах доступності. Він стежить за станом своїх зареєстрованих цілей і направляє трафік тільки до здорових цілей (рис. 3.3). Код для налаштування балансувальника навантаження наведений у лістингу 3.8.

Лістинг 3.8 – Налаштування балансувальника навантаження

JenkinsALB:

```
Type: 'AWS::ElasticLoadBalancingV2::LoadBalancer'
Properties:
  SecurityGroups:
  - !Ref JenkinsALBSG
  Subnets:
  - !Ref MyPublicSubnet1
  - !Ref MyPublicSubnet2
  Type: application
```

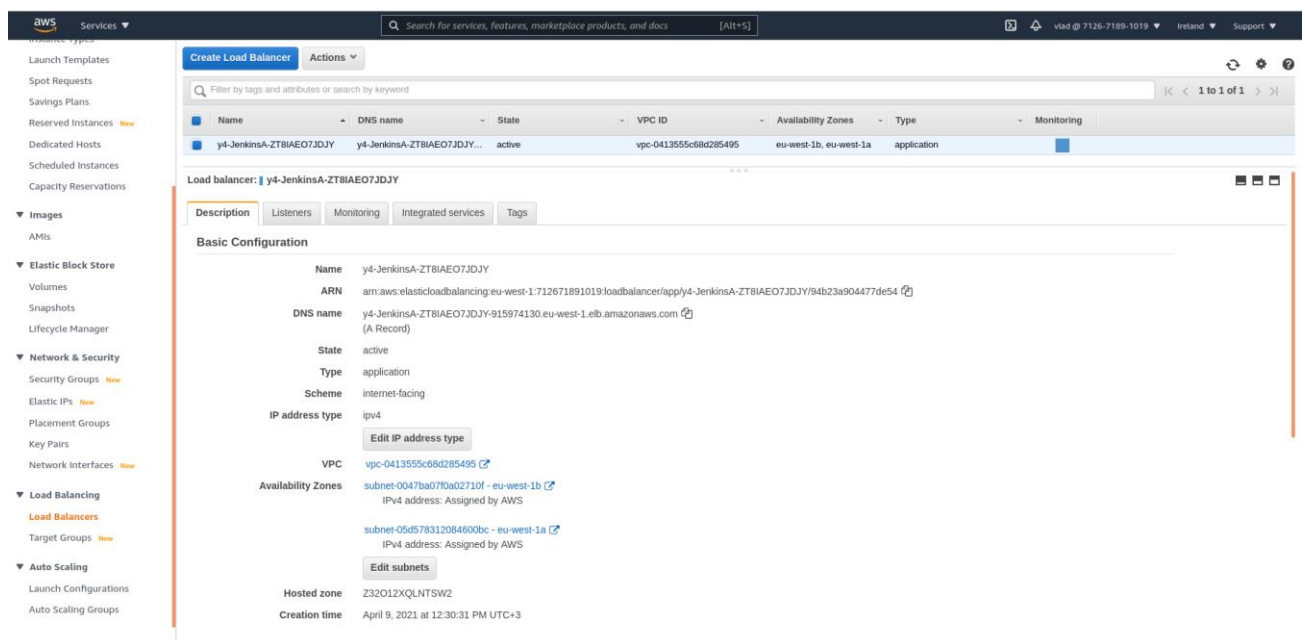


Рисунок 3.3 – Створення балансувальника навантаження

Для налаштування серверу, теж необхідна група безпеки розташована на порту 8080, та яка пов'язана з групою безпеки балансувальника навантаження. Тобто, щоб мати можливість увійти на сервер, треба обов'язково це зробити через доменне ім'я Load balancer. Код для налаштування групи безпеки для балансувальника навантаження наведений у лістингу 3.9.

Лістинг 3.9 – Налаштування групи безпеки для балансувальника навантаження

```
JenkinsSecurityGroup:
  Type: 'AWS::EC2::SecurityGroup'
  Properties:
    GroupDescription: Jenkins Security Group
    VpcId:
      Ref: myVPC
    SecurityGroupIngress:
      - IpProtocol: tcp
        FromPort: 8080
        ToPort: 8080
        SourceSecurityGroupId: !Ref JenkinsALBSG
```

Цільова група направляє запити до зареєстрованої цілі, такої як екземпляр EC2, використовуючи вказаний протокол та номер порту. Також налаштовано перевірку працездатності для цільової групи. Перевірка працездатності виконується для цілей, зареєстрованих у цільовій групі, яка вказана в правилі прослуховування для балансувальника навантаження. Код для налаштування цільової групи наведений у лістингу 3.10.

Лістинг 3.10 – Налаштування цільової групи

```
JenkinsALBTargetGroup:
  Type: 'AWS::ElasticLoadBalancingV2::TargetGroup'
  Properties:
    Port: 8080
    Protocol: HTTP
    VpcId: !Ref myVPC
    HealthCheckPath: /login
```

Слухач перевіряє запити на з'єднання від клієнтів, використовуючи налаштовані протокол і порт. Правила, які формулюються для слухача, зазначають,

як балансувальник навантаження направляє запити своїм зареєстрованим цілям. Кожне правило складається з пріоритету одного, або декількох дій і одного або декількох умов. Коли умови для правила виконані, його дії виконуються. Код для налаштування слухача наведений у лістингу 3.11.

Лістинг 3.11 – Налаштування слухача

```
JenkinsELBListener:
  Type: 'AWS::ElasticLoadBalancingV2::Listener'
  Properties:
    DefaultActions:
      - Type: forward
        TargetGroupArn: !Ref JenkinsALBTargetGroup
    LoadBalancerArn: !Ref JenkinsALB
    Port: 80
    Protocol: HTTP
```

Для розробки необхідне розподілене сховище даних. Тому використали сервіс EFS (рис. 3.4). За допомогою якого з'являється можливість копіювати інформацію між кількома центрами обробки даних. Це дозволяє залишатися доступним, навіть якщо цілий центр обробки даних страждає від перебоїв, що забезпечує високу доступність і надійність. Створили зашифровану файлову систему. Додали точку монтування, зробили посилання до підмережі та до самої файлової системи. Також прикріпили групу безпеки, яка надає доступ до точок монтування лише з сервера. Код для налаштування файлової системи наведений у лістингу 3.12.

Лістинг 3.12 – Налаштування файлової системи

```
SharedEFSVolume:
  Type: 'AWS::EFS::FileSystem'
  Properties:
    Encrypted: true
SharedMountTarget1:
  Type: 'AWS::EFS::MountTarget'
  Properties:
    FileSystemId:
      Ref: SharedEFSVolume
    SubnetId:
      Ref: MyPublicSubnet1
    SecurityGroups:
      - !Ref EFSSecurityGroup
```

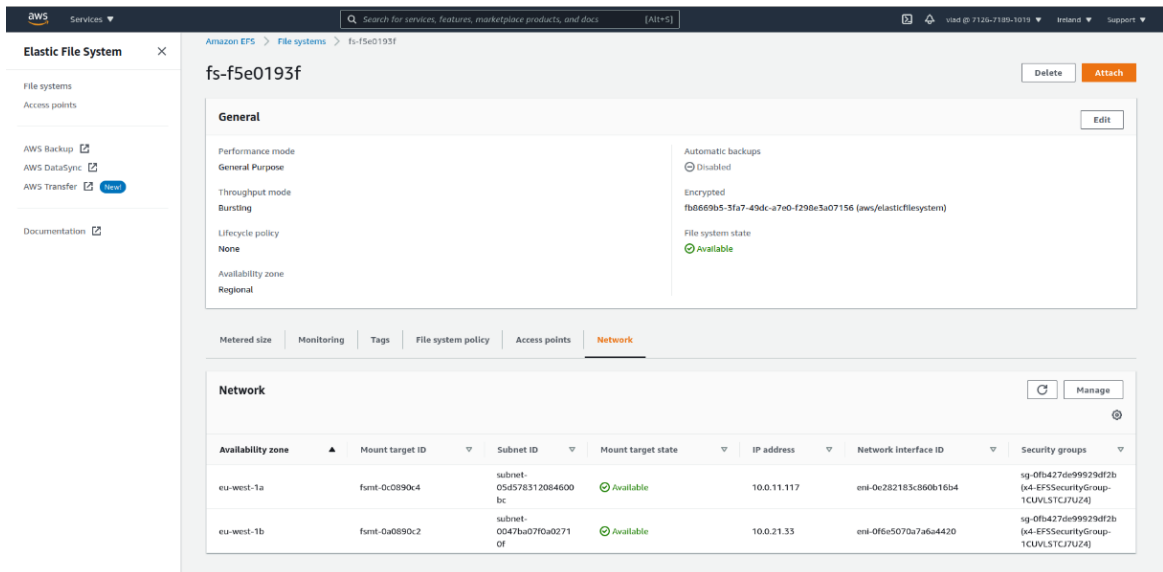


Рисунок 3.4 – Створення файлової системи

Створимо шаблон запуску, що містить інформацію про конфігурації для запуску екземпляра. Використовуючи тип `AWS :: CloudFormation :: Init`, вмикаються метаданні в екземпляр для допоміжного скрипта `cfn-init`. Шаблон викликає сценарій `cfn-init`, сценарій шукає метаданні ресурсу, що базуються на ключових словах метаданих `AWS :: CloudFormation :: Init`. Використовуючи набір конфігурацій, визначили порядок обробки ключів конфігурації. Шаблон не запускатиметься, доки точки монтування не будуть налаштовані як діючі. Код для визначення набору конфігурацій наведений у лістингу 3.13.

Лістинг 3.13 – Визначення набору конфігурацій

JenkinsLT:

```
Type: 'AWS::EC2::LaunchTemplate'
DependsOn:
- SharedMountTarget1
- SharedMountTarget2
Metadata:
  'AWS::CloudFormation::Init':
    configSets:
      default:
        - mount
        - install
        - setup
        - custom
```

- run

Перший крок монтування. На цьому кроці відбувається конфігурування директорії. У цю директорію будуть надходити усі зміни, які пов'язані з Multibranch Pipeline. Для цього спочатку треба завантажити утиліти, що дозволять примонтувати файлову систему до директорії. Далі створили користувача Jenkins, вибравши user id, та призначивши цьому користувачеві директорію /var/lib/jenkins. Остання дія – це монтування вже створених розділів. На наступному кроці встановлюємо необхідні нам пакети. На етапі setup прописана повна послідовність дій, яка необхідна для запуску Jenkins сервера. Останні два етапи необхідні для активації ще не запусчених сервісів. Код для конфігурування екземпляра наведений у лістингу 3.14.

Лістинг 3.14 – Конфігурування екземпляра

```
mount:
  packages:
    yum:
      amazon-efs-utils: []
  users:
    jenkins:
      uid: 500
      homeDir: /var/lib/jenkins
  commands:
    a_mountpoint_mkdir:
      command: 'mkdir /var/lib/jenkins && chown -R
500:500 /var/lib/jenkins'
      test: '[ ! -d /var/lib/jenkins ]'
    b_mountpoint_mount:
      command: !Sub
        - 'mount -t efs "${JenkinsStorage}:/"
/var/lib/jenkins'
        - JenkinsStorage: !Ref SharedEFSVolume
      test: >-
        if mount | grep -q /var/lib/jenkins; then exit
1; else exit 0;
      fi
```

CloudFormation містить набір допоміжних скриптів таких як cfn-init, cfn-signal, заснованих на cloud-init. Ці допоміжні сценарії можливо викликати з

шаблону CloudFormation для встановлення, налаштування і оновлення додатків на примірнику Amazon EC2, розташованому в одному шаблоні.

Допоміжний сценарій cfn-signal повідомляє CloudFormation, що екземпляри Amazon EC2 були успішно створені або оновлені. Код для налаштування допоміжних скриптів наведений у лістингу 3.15.

Лістинг 3.15 – Налаштування допоміжних скриптів

```
Properties:
  UserData: !Base64
    'Fn::Sub': >
      #!/bin/bash -x
      /opt/aws/bin/cfn-init -v --stack ${AWS::StackName}
--resource
      JenkinsLT --region ${AWS::Region}
      /opt/aws/bin/cfn-signal -e $? --stack
${AWS::StackName} --resource
      JenkinsASG --region ${AWS::Region}
```

Сервіс Route 53 перенаправляє запити користувачів до балансувальника навантаження за рахунок перетворення доменних імен у формат цифрових IP-адрес. При цьому Amazon Route 53 повністю сумісний з протоколом IPv6. Властивість HostedZoneName надає можливість створити запис завдяки вибраній зоні розміщення. Код для налаштування доменного імені наведений у лістингу 3.16.

Лістинг 3.16 – Налаштування доменного імені

```
myDNSRecord:
  Type: AWS::Route53::RecordSetGroup
  Properties:
    HostedZoneName: skyrockets.uk.
    RecordSets:
      - Name: jenkins.skyrockets.uk.
        Type: A
        AliasTarget:
          HostedZoneId: Z32O12XQLNTSW2
          DNSName: !GetAtt 'JenkinsALB.DNSName'
```

Auto Scaling group містить instance Amazon EC2, який розглядається як мета автоматичного масштабування і управління (рис. 3.5). Розмір Auto Scaling group залежить від кількості екземплярів, які встановлюються властивістю

DesiredCapacity. Група Auto Scaling продовжує підтримувати фіксовану кількість екземплярів, навіть якщо екземпляр стає непрацездатним. При виникненні неполадки, група масштабування завершує роботу непрацездатного екземпляра і запускає інший екземпляр для його заміни. Властивість HealthCheckGracePeriod дозволяє затримати наступне масштабування на певний проміжок часу, доки instance не буде в активному положенні. Код для налаштування групи автоматичного масштабування наведено у лістингу 3.17.

Лістинг 3.17 – Налаштування групи автоматичного масштабування JenkinsASG:

```
Type: 'AWS::AutoScaling::AutoScalingGroup'
Properties:
  AutoScalingGroupName: JenkinsASG
  MinSize: '1'
  MaxSize: '3'
  DesiredCapacity: '1'
  HealthCheckGracePeriod: 300
  LaunchTemplate:
    LaunchTemplateId: !Ref JenkinsLT
    Version: !Ref TemplateVersion
```

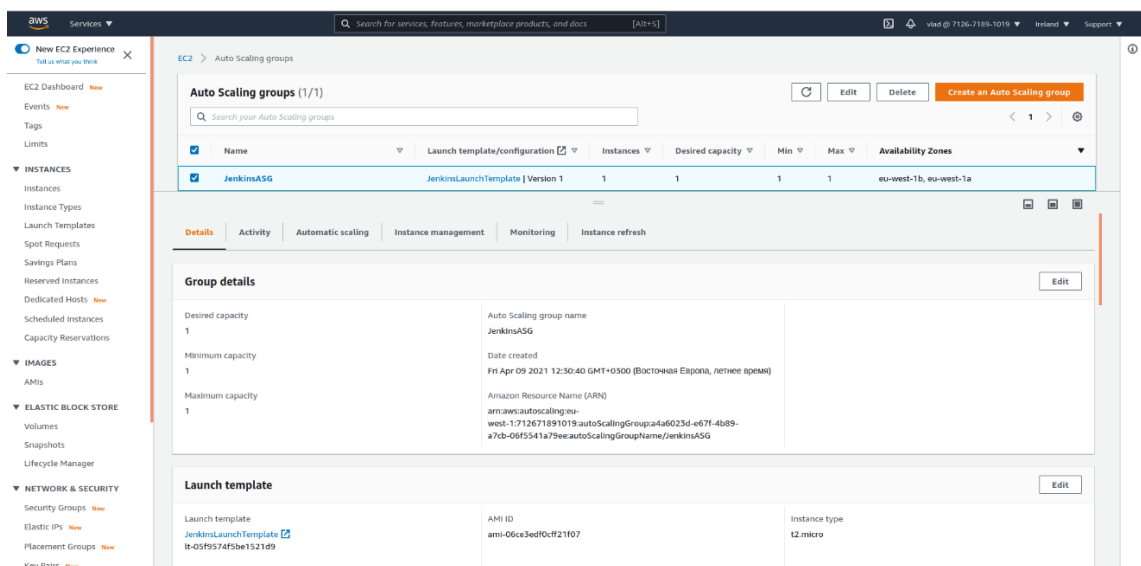


Рисунок 3.5 – Створення групи автоматичного масштабування

Для управління докер контейнерами буде використано сервіс ECS, який відмінно справляється зі своїми цілями завдяки тому, що кластер ECS можна

запускати за допомогою AWS Fargate для безсерверних обчислень на базі контейнерів. Так як було застосовано тип запуску Fargate, поле CPU є обов'язковим. У властивості ContainerDefinitions налаштовано основні параметри, які дозволяють сформулювати задачу, а саме:

- name. Ім'я контейнеру.
- portmappings. Список зіставлення портів для контейнера. Зіставлення портів дозволяють контейнерам отримувати доступ до портів на екземплярі контейнера хосту для надсилання або отримання трафіку.
- image. Зображення, що використовується для запуску контейнера. Коли починається нова задача, агент контейнера Amazon ECS витягує останню версію вказаного зображення, та тег для створення контейнера. Однак подальші оновлення образу у сховищі не поширюються на вже запущені задачі. Код для налаштування сервісу ECS наведений у лістингу 3.18.

Лістинг 3.18 – Налаштування сервісу ECS

```
Taskdefinition1:
  Type: 'AWS::ECS::TaskDefinition'
  Properties:
    RequiresCompatibilities:
      - FARGATE
    ExecutionRoleArn: !GetAtt[TaskDefinitionExecutionRole, Arn]
    NetworkMode: awsvpc
    Cpu: 0.25 vCPU
    Memory: 0.5GB
    ContainerDefinitions:
      - Name: my-app
        Image: !Ref ImageArn
        PortMappings:
          - ContainerPort: !Ref ContainerPort1
            Protocol: tcp
```

Перший етап розробки інфраструктури завершено. Така інфраструктура є надійною від несправностей, пов'язаних з неполадками у робочому середовищі. Наявність балансувальника навантаження та групи масштабування якнайкраще позначається на доступності додатків у будь-який момент, за що клієнти будуть безмежно вдячні.

3.2 Побудова GitHub організації

Для того, щоб створити організацію, необхідно перейти до профілю на GitHub, натиснути на іконку "+" у правому верхньому кутку сторінки та вибрати пункт "New organization" з меню.

Далі треба вказати назву організації. Вона буде мати назву "MyDevOpsWay", також необхідно ввести email адресу у якості основного методу зв'язку, у даному випадку vladfnnk@gmail.com. В результаті з'являється можливість запрошувати людей у якості співвласника аккаунта. Організації безкоштовні, якщо планується працювати над проектами з відкритим вихідним кодом. Також з'являється можливість автоматично починати відслідковувати всі створені репозиторії в організації. Після того, як всі кроки були зроблені, отримали сформовану організацію (рис. 3.6).

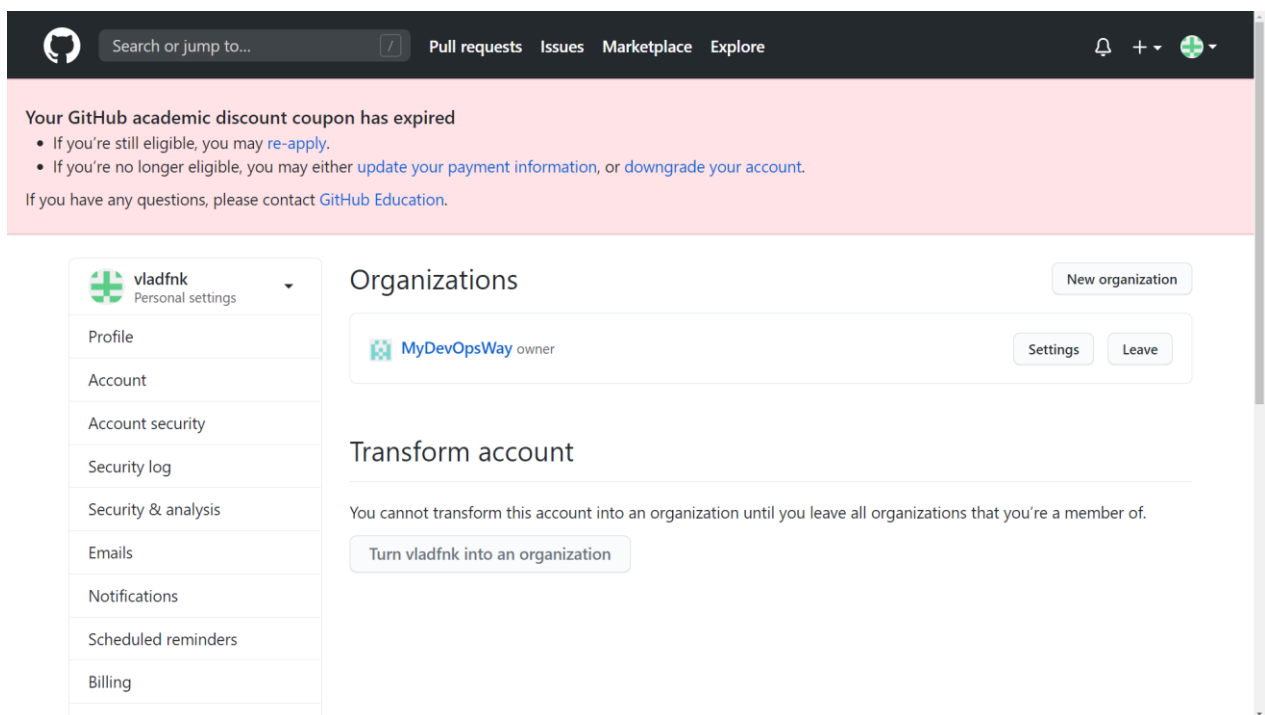


Рисунок 3.6 – GitHub організація

На наступному кроці треба пов'язати GitHub організацію з сервером. Для цього зареєструємо додаток OAuth на GitHub (рис. 3.7). Щоб додати OAuth, необхідно мати обліковий запис GitHub і права доступу до сховища. Приєднуємо додаток до головного сховища, щоб користувач у репозиторії міг отримати доступ до pipeline відповідно до правил політики.

Register a new OAuth application

Application name *

 Something users will recognize and trust.

Homepage URL *

 The full URL to your application homepage.

Application description

 This is displayed to all users of your application.

Authorization callback URL *

 Your application's callback URL. Read our [OAuth documentation](#) for more information.

[Register application](#) [Cancel](#)

© 2020 GitHub, Inc. [Terms](#) [Privacy](#) [Security](#) [Status](#) [Help](#) [Contact GitHub](#) [Pricing](#) [API](#) [Training](#) [Blog](#) [About](#)

Рисунок 3.7 – OAuth додаток

Далі треба зробити налаштування на сервері. Тому перейшли до Jenkins як адміністратор, та у розділі Manage Jenkins обрали плагін GitHub Authorization. Потім натиснули значок “встановити без перезавантаження” або, якщо на екземплярі не запуснені збірки, можливо завантажити і перезавантажити Jenkins.

Тепер необхідно зазначити параметри клієнта Jenkins для OAuth. Перейшли до глобальних налаштувань безпеки в Jenkins у розділі “Управління” та заповнили необхідні дані (рис. 3.8).

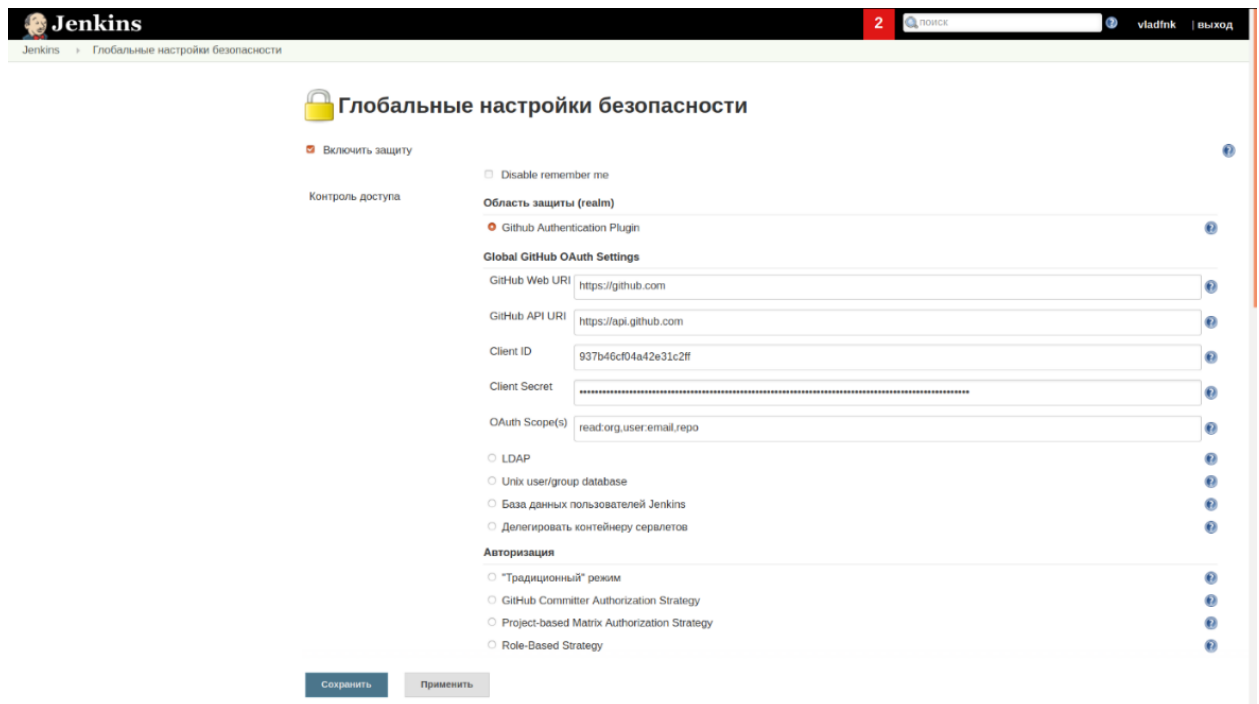


Рисунок 3.8 – Глобальні налаштування безпеки

Після усіх цих налаштувань можна увійти в систему з обліковими даними GitHub. Таким чином, будь-який користувач, який запрошений до GitHub організації буде долучений до проекту.

3.3 Розробка пайплайну безперервної інтеграції

Jenkins Pipeline записується у текстовий файл, котрий має назву Jenkinsfile. Він у свою чергу, дозволяє створювати пайплайн у вигляді коду.

Код та усі додаткові параметри, які необхідні для створення пайплайну будуть зберігатися на GitHub. Для того, щоб Jenkins мав доступ до репозиторію де зберігається пайплайн, необхідно в організації на GitHub створити користувача. Тому було створено користувача з назвою serviceuserforjenkins. Після цього перейшли до вкладки Personal access tokens та сгенерували новий токен (рис. 3.9).

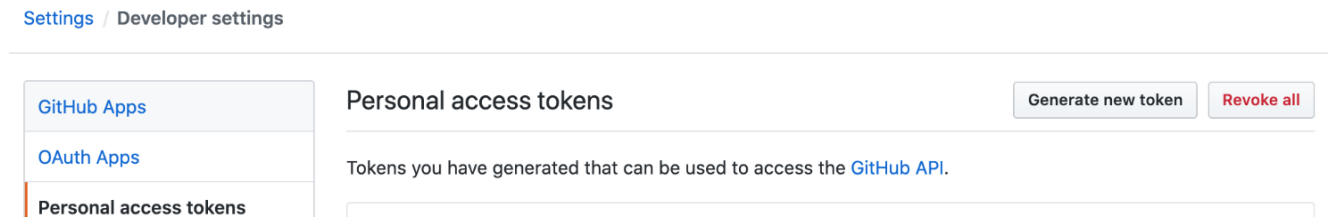


Рисунок 3.9 – Personal access tokens

Наступна дія передбачає створення облікових даних для спеціального використання Jenkins (рис. 3.10). Для цього перейшли до глобальних облікових даних та додали новий елемент, що містить назву та токен, який був сгенерований попередньою дією.



Рисунок 3.10 – Облікові данні

Для розробки нам необхідний тип проекту Multibranch Pipeline, який дозволить реалізовувати Jenkinsfile для різних гілок одного проекту. Після створення проекту необхідно перейти до його налаштувань. У розділі джерела гілок важливо додати облікові дані користувача та адресу репозиторія (рис. 3.11), в якому знаходиться Jenkinsfile.

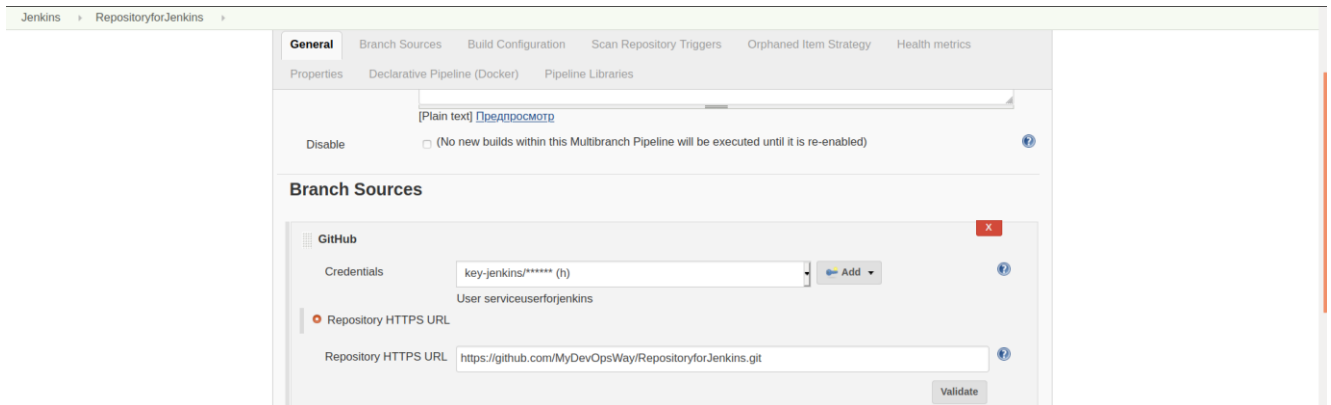


Рисунок 3.11 – Branch sources

Jenkins Pipeline формує у собі певну ієрархію у декларативному стилі. На найвищому рівні розташований – пайплайн (Pipeline), який містить у собі певні етапи (Stages). Етапи мають складатися з кроків (Steps). У свою чергу кроками мають бути скрипти (Script).

Спочатку оголосимо змінні, на рівні всього скрипта, за межами блоку pipeline з використанням ключового слова `def`. Після цього вони стають доступні для використання у всіх блоках скрипта. Структура оголошення змінних за межами блоку pipeline наведена у лістингу 3.19.

Лістинг 3.19 – Оголошення змінних за межами блоку pipeline

```
def buildSiteApp() {
    sh 'docker build -t ${APP_NAME}:${DEPLOY_TO} .'
}
def performTests() {
    sh 'docker run --rm -d -p 3473:80 ${APP_NAME}:${DEPLOY_TO}'
    echo "Test has been started"
}
def PushToECR() {
    sh 'docker tag ${APP_NAME}:${DEPLOY_TO}
    712671891019.dkr.ecr.eu-west-
    1.amazonaws.com/${Repository}:latest'
    docker.withRegistry (ECRURL, ECRURED) {
        docker.image('712671891019.dkr.ecr.eu-west-
        1.amazonaws.com/${Repository}').push('latest')
    }
}
def DockerCleanup() {
```

```

    sh 'docker rmi 712671891019.dkr.ecr.eu-west-
1.amazonaws.com/${Repository}:latest'
}

```

Застосування змінних середовища що містять у собі послідовне співвідношення ключ-значення дуже зручне. Такі дії потрібні на випадок якщо важливо вносити будь-які рідко змінювані властивості.

Jenkins Pipeline виставляє змінні середовища через глобальну змінну env, яка доступна з будь-якого місця у Jenkinsfile. Структура оголошення змінних наведена у лістингу 3.20.

Лістинг 3.20 – Оголошення змінних

```

pipeline {
  environment {
    APP_NAME = 'amazonecsdeepdive'
    DEPLOY_TO = "${BRANCH_NAME}"
    ECRURL = 'https://712671891019.dkr.ecr.eu-west-
1.amazonaws.com/app1'
    ECRURED = 'ecr:eu-west-1:aws-d'
    Repository = 'app1'
  }
}

```

Далі опишемо самі етапи створення Jenkins Pipeline. Перший етап – це ініціалізація, яка запускається у випадку збірки будь-якої гілки. Тут визначається до якої саме гілки належить проект. Якщо гілка починається з назви Pull request, то буде виконано лише декілька етапів. Серед яких побудова докер образу та подальше тестування з метою виявлення несправностей. У випадку, якщо проект належить до гілки, яка не містить Pull request, то збудований та протестований образ відправляється до ECR репозиторію. На останньому кроці розгортається за допомогою сервісу ECS. Код для етапу ініціалізації наведено у лістингу 3.21.

Лістинг 3.21 – Етап ініціалізації

```

stages {
  stage ("Init checks") {
    steps {
      echo "Init checks process has been started"
      script {

```

```

        if (env.BRANCH_NAME.startsWith('PR')) {
            echo 'This is a pull request!'
        } else {
            echo 'Set environment variables'
            DEPLOY_TO_ECS = 'true'
        }
    }
}
}

```

Побудова образу виконується за допомогою викликаної функції `buildSiteApp()`, в якій використовується команда `docker build` з прапором `-t`. Цей прапор надає можливість задати ім'я. Команда будує образ, звертаючись до файлу `Dockerfile`, який знаходиться у поточній директорії, на що вказує точка у кінці.

Команда `docker run` запускає контейнер на основі побудованого образу. Опції `--rm` вказують команді на автоматичне вилучення контейнера при виході з нього. Для того, щоб контейнер продовжував працювати при виході з термінальної сесії, запускаємо його в автономному режимі застосовуючи опцію `-d`. Побудова образу та подальше тестування наведено у лістингу 3.22.

Лістинг 3.22 – Побудова образу та подальше тестування

```

stage ("Docker build") {
    steps {
        echo "Docker build process has been started"
        buildSiteApp()
    }
}
stage ("Run tests") {
    steps {
        echo "Run tests process has been started"
        performTests()
    }
}

```

У випадку, якщо повернено параметр `DEPLOY_TO_ECS`, тоді виконується етап відправки докер образу до віддаленого репозиторію. У функції `PushToECR()`, скориставшись методом `withRegistry()`, передаємо облікові данні та користувацьку URL-адресу реєстру. Після того, як образ успішно доставлено його необхідно

видалити. Структура відправки докер образу до віддаленого репозиторію та подальше видалення наведено у лістингу 3.23.

Лістинг 3.23 – Відправка докер образу до віддаленого репозиторію та подальше видалення

```
stage ("Push to ECR") {
    when {
        expression {
            return DEPLOY_TO_ECS
        }
    }
    steps {
        PushToECR()
    }
}
stage ("Docker cleanup") {
    steps {
        DockerCleanup()
    }
}
```

Останній етап – це розгортання образу у сервісі ECS. Створюємо набір змін для оновлення стеку CloudFormation за допомогою заданого шаблону. Вказуємо `paramsFile`, що містить параметри. Щоб уникнути обмеження швидкості на API AWS, змінюємо інтервал опитування за замовчуванням до 1000 мс, використовуючи параметр `pollInterval`. Виконуємо попередньо створений набір змін для оновлення стеку CloudFormation. Код для розгортання образу наведений у лістингу 3.24.

Лістинг 3.24 – Розгортання образу

```
stage ("Deploy ECS service") {
    steps {
        cfnCreateChangeSet(stack:'u7', changeSet:"u7-change-set",
            file:'cf-ecs-service.yml',paramsFile:'cf-params.yaml',
            params:["ImageArn=712671891019.dkr.ecr.eu-west-1.amazonaws.com/${Repository}:latest"],create:'true',
            timeoutInMinutes: 10, pollInterval: 1000)
        cfnExecuteChangeSet(stack:"u7",changeSet:"u7-change-set",
            timeoutInMinutes: 10, pollInterval: 1000)
    }
}
```

На рисунку 3.12 продемонстрована готова блок-схема.

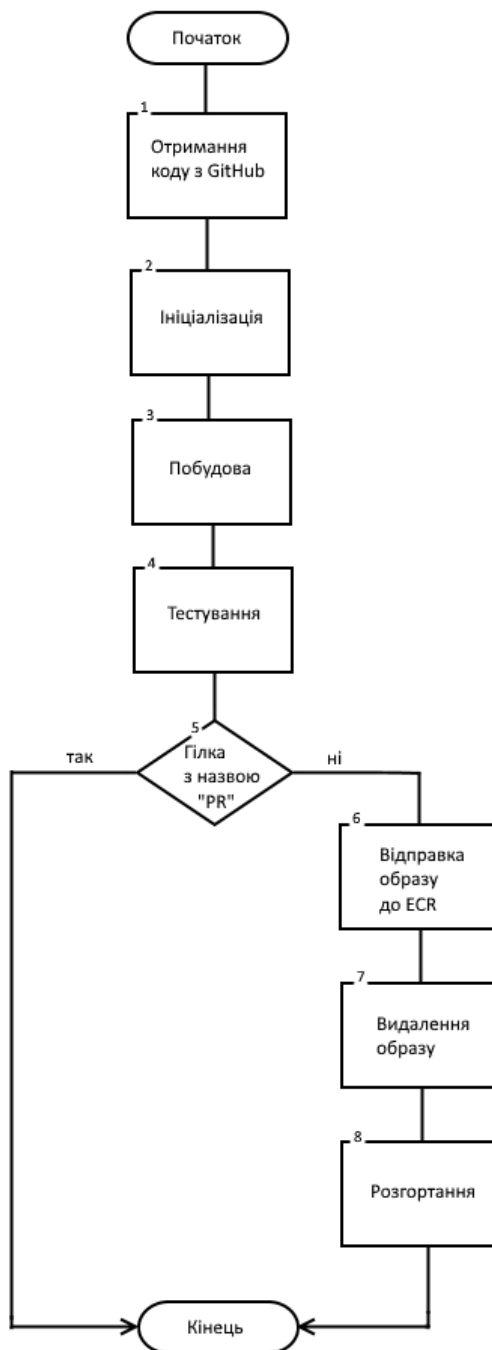


Рисунок 3.12 – Блок-схема

В результаті повністю налаштований пайплайн. Остання складова розробки завершена і наступає етап випробування готової комп'ютерної системи.

4 ВИПРОБУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Етап випробування комп'ютерної системи є вкрай важливим. Він дозволяє визначити оптимальний варіант роботи системи в цілому та знайти можливості для покращення вже готових елементів.

Для реалізації комп'ютерної системи необхідно випробувати існуюче обладнання. Від правильного вибору частини залежить швидкість та надійність роботи всієї системи.

Крім цього, необхідно забезпечити сервер безперебійним джерелом енергії та організувати резервне копіювання важливих файлів для запобігання аварійних ситуацій.

Перш за все важливо перевірити JenkinsServer на здатність невідмовної роботи. У разі передчасного завершення роботи сервера одразу буде запущено інший. Для цього у полі Instance state вибираємо дію Terminate instance. Після чого екземпляр буде негайно видалено (рис. 4.1).

Status ▾	Description ▾
Successful	Launching a new EC2 instance: i-0f817a033472efa9a
WaitingForELBConnection Draining	Terminating EC2 instance: i-07c9a942a504c5e5c - Waiting For ELB Connection Draining.
Successful	Launching a new EC2 instance: i-07c9a942a504c5e5c

Рисунок 4.1 – Перевірка на здатність невідмовної роботи сервера

Після того як група масштабування дізнається про те, що сервер завершив свою роботу то спочатку відслідкує від'єднання з балансувальником навантаженням. Після чого, через декілька хвилин, розпочне роботу іншого сервера з точно такими ж характеристиками. Таке масштабування можливе завдяки тому, що в параметрах налаштована мінімальна кількість робочих екземплярів (один).

Наступне випробування перевіряє наявність авторизації користувача через GitHub організацію, яка виконана з подальшою можливістю доступу до цільового сервера (рис. 4.2).

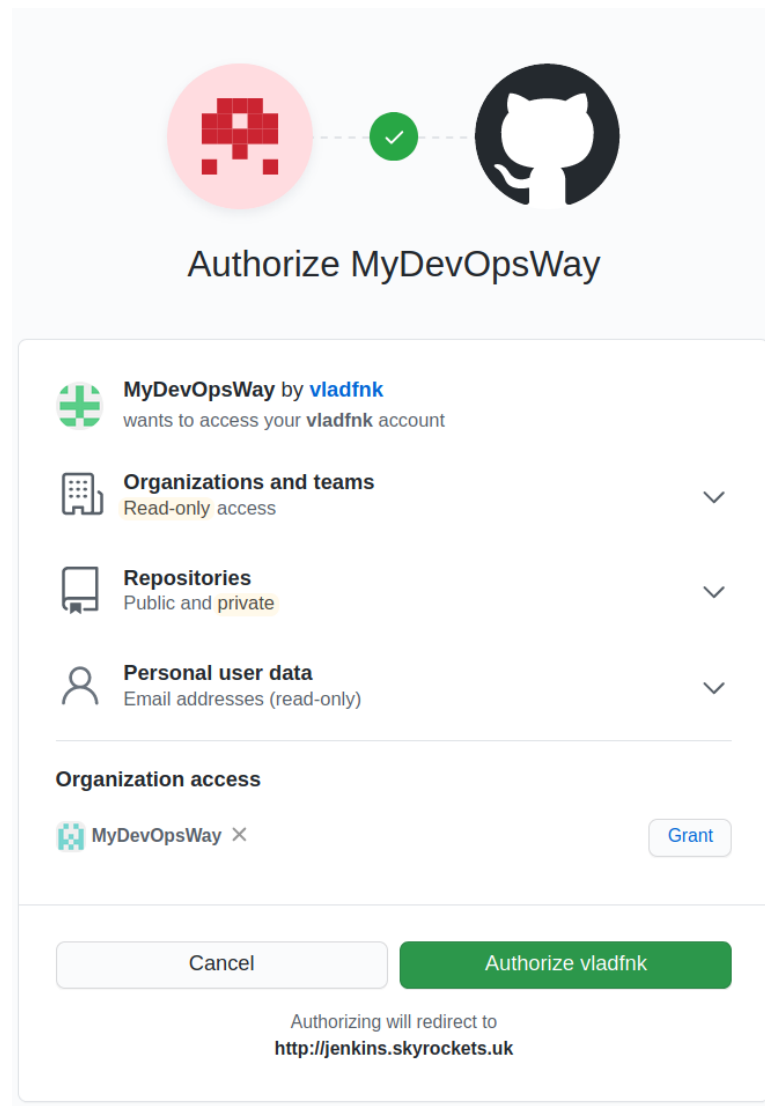


Рисунок 4.2 – Створення GitHub організації

Далі перевіряється на дійсність Jenkins pipeline з подальшим розгортанням у хмарі AWS. Для цього спочатку випробуємо гілку, яка починається з назви Pull request. Після етапу тестування pipeline завершує свою роботу (рис. 4.3).

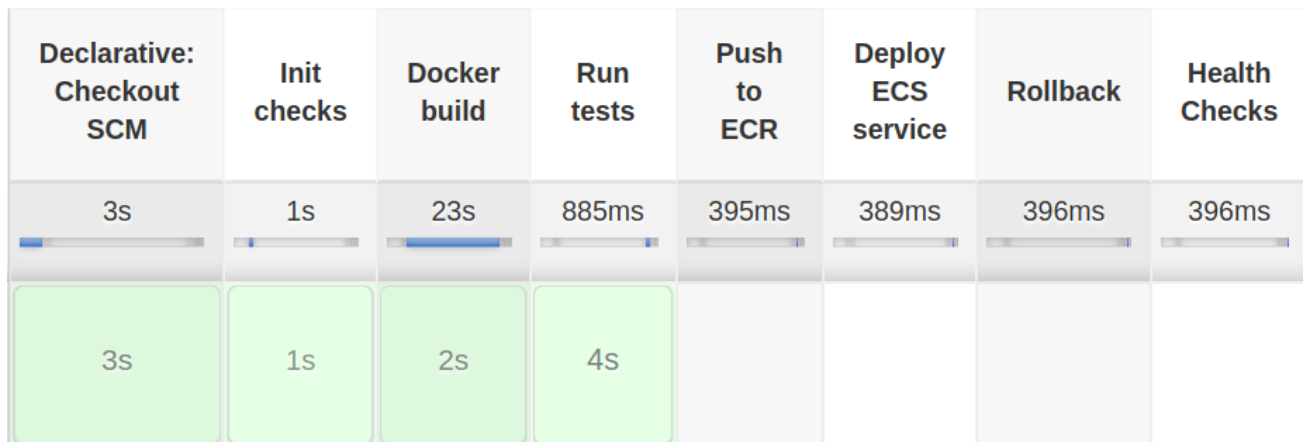


Рисунок 4.3 – Випробування гілки яка починається з назви Pull request

Після цього необхідно випробувати Jenkins pipeline для гілки в якій відсутній заголовок Pull request. В результаті випробування повинні виконуватися усі етапи без винятку (рис. 4.4).

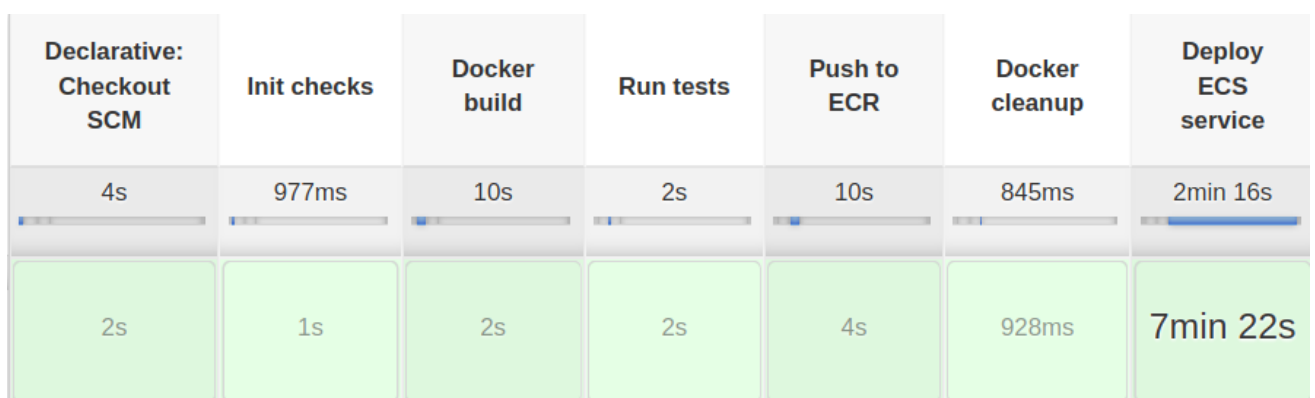


Рисунок 4.4 – Випробування гілки яка починається без назви Pull request

Продемонстровано наявність віддаленого репозиторію для зберігання образів, які були збудовані та випробуванні на сервері (рис. 4.5).

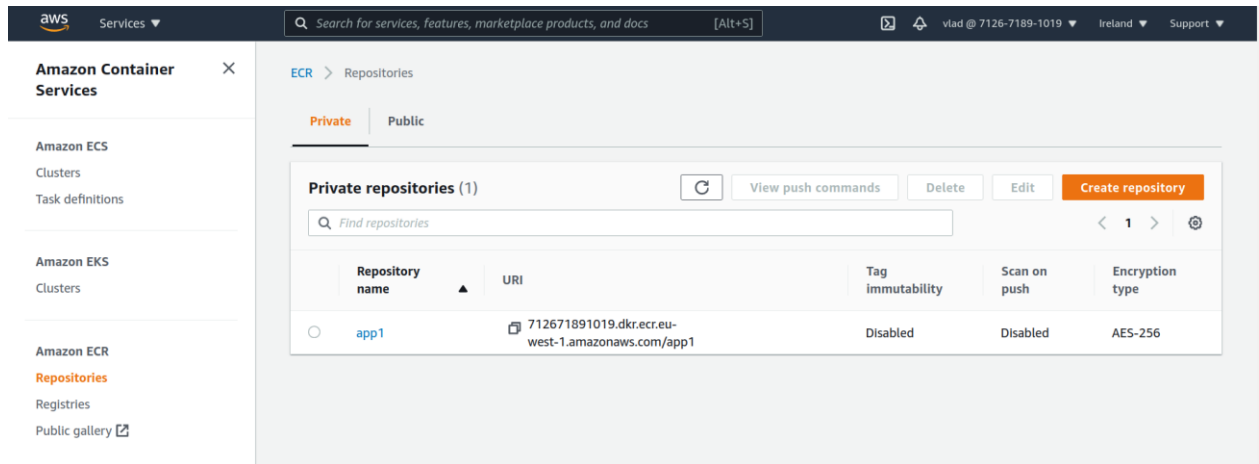


Рисунок 4.5 – Віддалений репозиторій для зберігання образів

Образ з тегом v1 відображає перший запуск Jenkins Job (рис. 4.6).

☐	Image tag	Size (MB)	Image URI	Digest	Scan status
☐	v1	97.23	Copy URI	sha256:53fb3c40c2b1c84...	Complete

Рисунок 4.6 – Образ з тегом v1

Надалі з використанням цього образу було побудовано Task (рис. 4.7).

	Name	Contain...	Status	Image	Image Digest
▶	my-app	106afe39...	RUNNING	712671891019.dkr.ecr.eu-west-1.amazonaws.com/app1:v1	sha256:53fb3c40c2b1c8478a6cc7...

Рисунок 4.7 – Готовий Task

В результаті успішної доставки образу та створення Task, у якому розташований необхідний образ, було розгорнуто сервіс, який містить у собі певну інформацію. У данному випадку повинен відображатися надпис «Hello World! v1», що і було отримано після першого випробування системи (рис. 4.8).

Hello World! v1

Рисунок 4.8 – Перша версія додатка

Образ з тегом `lates` відображає наступний запуск Jenkins Job у якому знаходиться інша версія додатка (рис. 4.9).

☐	Image tag	Size (MB) ▾	Image URI	Digest	Scan status
☐	latest	100.01	 Copy URI	 sha256:438dbac5abdbc5...	Complete

Рисунок 4.9 – Образ з тегом `lates`

В результаті виконання замість першої версії додатка повинна відображатися наступна версія (рис. 4.10).

Hello World! v2

Рисунок 4.10 – Друга версія додатка

ВИСНОВКИ

В результаті виконання дипломної роботи було проведено опис предметної області безперервної інтеграції і доставки. Зроблено детальний огляд об'єкта. Було розглянуто технології з метою обрання найкращого кандидата для поставленої задачі.

Визначено послідовність дій при розробці, а саме:

- розробка інфраструктури у хмарі AWS;
- створення GitHub організації;
- розробка пайплайну безперервної інтеграції.

Також було виконано аналіз апаратної та інфраструктурної частини. Визначено тип екземпляра та тип диску, які надалі будуть використані при розробці.

У ході розробки були описані основні дії кожної з частин комп'ютерної системи. Спроектowana інфраструктура є надійною від несправностей, пов'язаних з неполадками у робочому середовищі. При використанні GitHub організації дуже легко об'єднувати людей для вирішення завдання. Після того як були додані налаштування на сервері будь-який користувач, який запрошений до GitHub організації буде долучний до проекту.

Крім цього було випробувано готову комп'ютерну систему. За допомогою цього випробування з'являється можливість яка дозволяє визначити оптимальний варіант роботи системи в цілому та знайти можливості для покращення вже готових елементів.

Перенесення програмного забезпечення у робоче оточення – повільний і ризикований процес. Проводячі оптимізація цього процесу можливо зробити розробку програм більш корисною та ефективною. У цьому сенсі безперервна інтеграція та безперервне розгортання є одними з найкращих засобів прискорення розробки програмних проектів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Что такое методология разработки CI/CD. URL: <https://selectel.ru/blog/what-is-ci-cd> (дата зверення 11.03.2021)
2. Еберхард В. Continuous delivery. Практика безперервних апдейтів. – СПб.:Пітер, 2018. – 320с.
3. Что такое CI/CD. URL: <https://habr.com/ru/company/otus/blog/515078/> (дата зверення 11.03.2021)
4. Терминология Jenkins. URL: <https://ru.wikipedia.org/wiki/Jenkins> (дата зверення 11.03.2021)
5. Знакомство з Jenkins. URL: <https://novikov.com.ua/%D0%B1%D0%B8%D1%82%D0%B2%D0%B0-jenkins-vs-gitlab-ci-cd> (дата зверення 11.03.2021)
6. Битва Jenkins и GitLab CI/CD. URL: <https://habr.com/ru/company/ruvds/blog/522334/> (дата зверення 17.03.2021)
7. Дженкинс альтернативы. URL: <https://coderlessons.com/tutorials/devops/uchebnik-po-jenkins/9-dzhenkins-alternativu> (дата зверення 17.03.2021)
8. Терминология GitLab. URL: <https://ru.wikipedia.org/wiki/GitLab> (дата зверення 17.03.2021)
9. GitHub – легендарная платформа для совместного управления кодом. URL: <https://linuxvsem.ru/programs/github-obzor> (дата зверення 17.03.2021)
10. GitHub – Внесение собственного вклада в проекты. URL: <https://git-scm.com/book/ru/v2/GitHub> (дата зверення 23.03.2021)
11. GitLab против GitHub. URL: <https://senior.ua/articles/gitlab-protiv-github> (дата зверення 23.03.2021)

12. Каковы плюсы и минусы GitHub по сравнению с Bitbucket. URL: <https://otvetus.com/kakovi-plyusi-i-minusi-github-po-sravneniyu-s-bitbucket-298826> (дата зверения 23.03.2021)

13. Терминология Bitbucket. URL: <https://ru.wikipedia.org/wiki/Bitbucket> (дата зверения 23.03.2021)

14. Сравнение Bitbucket и GitHub. URL: <https://bitbucket.org/product/ru/comparison/bitbucket-vs-github> (дата зверения 23.03.2021)

15. 12 причин, почему разработчики должны использовать Bitbucket. URL: <https://ru.rockycode.com/12-reasons-why-developers-should-use-bitbucket> (дата зверения 23.03.2021)

16. Топ 10 преимуществ, которые вы получите, используя Docker. URL: <https://cloud.kiev.ua/top-10-preimushhestv-kotorye-vy-poluchite/> (дата зверения 23.03.2021)

17. Понимая Docker. URL: <https://habr.com/ru/post/253877> (дата зверения 03.04.2021)

18. Docker и все, все, все. URL: <https://prohoster.info/blog/administrirovanie/docker-i-vse-vse-vse> (дата зверения 03.04.2021)

19. Подман против Докера. URL: <https://xperimentalhamid.com/ru/devops/podman-vs-docker/> (дата зверения 03.04.2021)

20. Терминология AWS. URL: https://ru.wikipedia.org/wiki/Amazon_Web_Services (дата зверения 03.04.2021)

21. Облачные вычисления с помощью AWS. URL: <https://aws.amazon.com/ru/what-is-aws/> (дата зверения 03.04.2021)

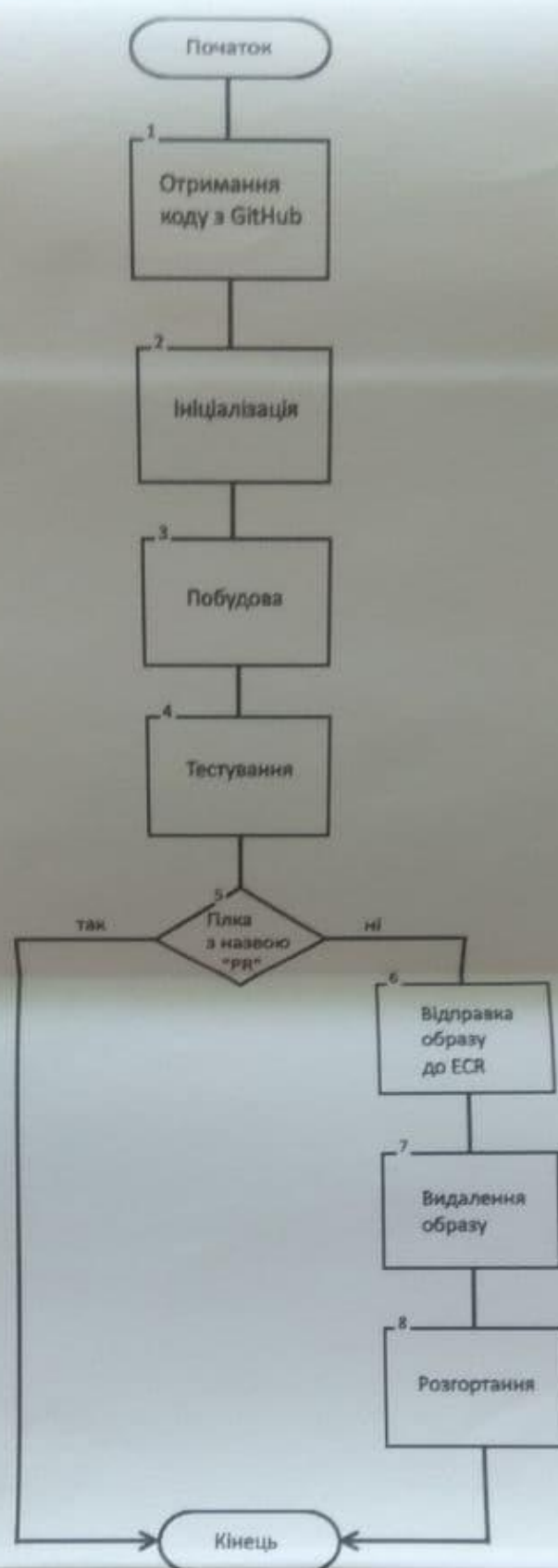
22. Возможности Amazon EFS. URL: <https://aws.amazon.com/ru/efs/features/> (дата зверения 03.04.2021)

23. Кратко о преимуществах. URL:
<https://aws.amazon.com/ru/application-hosting/benefits/> (дата зверения 03.04.2021)
24. AWS Identity and Access Management. URL:
<https://aws.amazon.com/ru/iam/> (дата зверения 09.04.2021)
25. AWS CloudFormation. URL:
<https://aws.amazon.com/ru/cloudformation/> (дата зверения 09.04.2021)
26. Каковы плюсы и минусы использования Amazon Web Services.
 URL: <https://otvetus.com/kakovi-plyusi-i-minusi-ispolzovaniya-amazon-web-services-aws-57306> (дата зверения 09.04.2021)
27. Терминология Microsoft Azure. URL:
https://ru.wikipedia.org/wiki/Microsoft_Azure (дата зверения 09.04.2021)
28. Выбираем Windows Azure: подходит ли платформа для вашего приложения. URL: <https://habr.com/ru/company/microsoft/blog/155617/> (дата зверения 09.04.2021)
29. Резервное копирование и аварийное восстановление. URL:
<https://azure.microsoft.com/ru-ru/solutions/backup-and-disaster-recovery/> (дата зверения 12.04.2021)
30. Преимущества Azure. URL: <https://ru.photo-555.com/9755097-advantages-of-azure> (дата зверения 12.04.2021)
31. Облако Microsoft Windows Azure: что это, и как устроено. URL:
<https://it-rating.in.ua/oblako-microsoft-windows-azure-cto-eto-i-kak-ustroeno> (дата зверения 12.04.2021)
32. Мониторинг CPU. URL:
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances-monitoring-cpu-credits.html> (дата зверения 12.04.2021)
33. Типы томов. URL:
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-volume-types.html> (дата зверения 12.04.2021)

34. Amazon Elastic Block Store URL:
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AmazonEBS.html> (дата
зверення 12.04.2021)

35. Мониторинг EBS. URL:
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-volume-status.html> (дата зверення 12.04.2021)

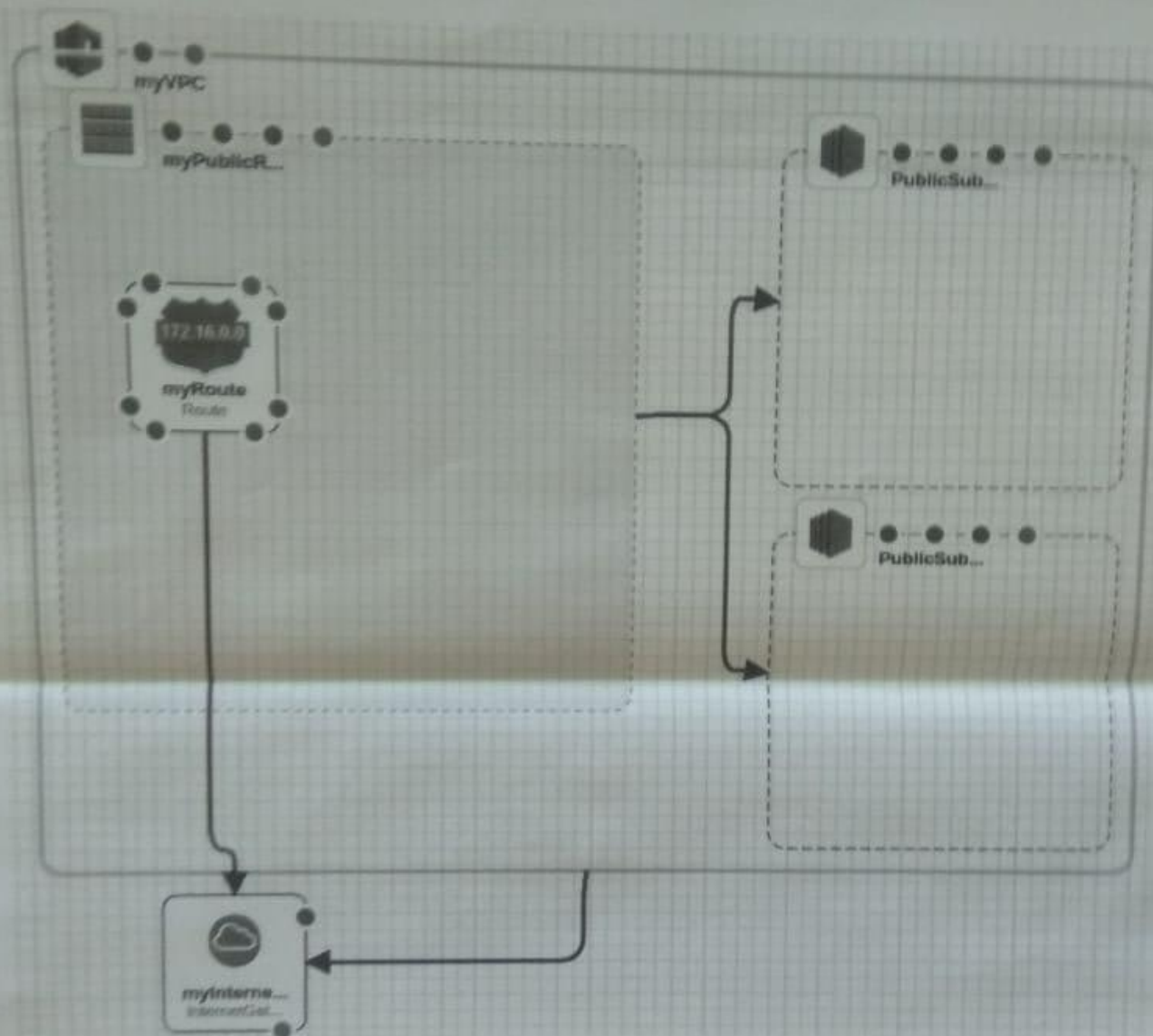
36. CPU кредиты. URL:
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-credits-baseline-concepts.html> (дата зверення 12.04.2021)



13.02070849.00043 А1					
Зм.	Лист	№ докум.	Підп.	Дата	Розробка клієнтської системи для безперервної інтеграції з доставкою документів у хмарну платформу AWS
Розроб.	Філюк В.С.				Блок-схема алгоритму роботи системи
Перев.	Скряпський С.Ю.				
Т. контр.					
Н. Контр.	Шербиш Н.В.				
Затверд.	Кудерметов Р.К.				

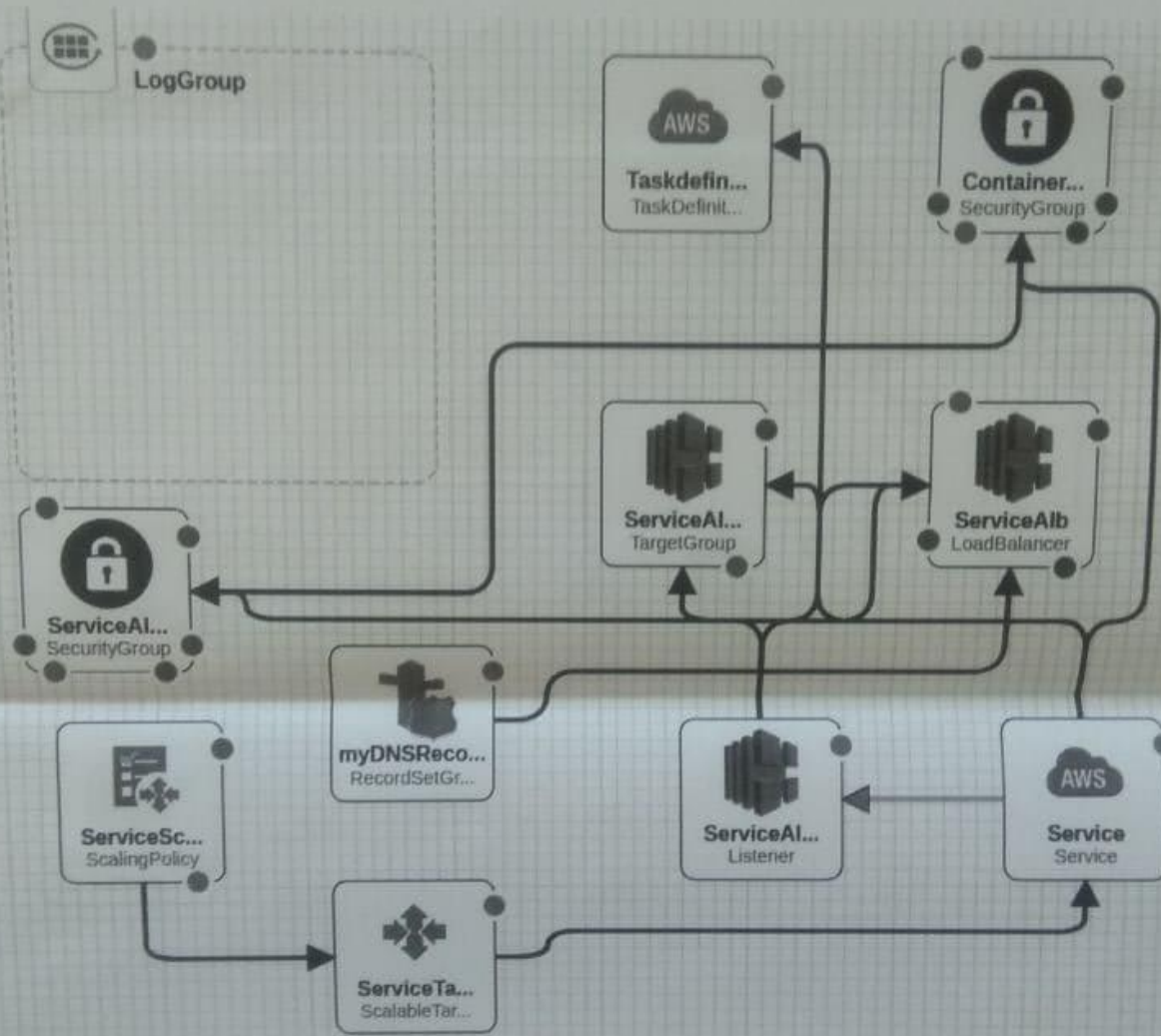
Лист	Місяц	Місяць
Лист 1	Листів 1	
НУ "Запорізька політехніка" КНТ-327		

Проектування мережі



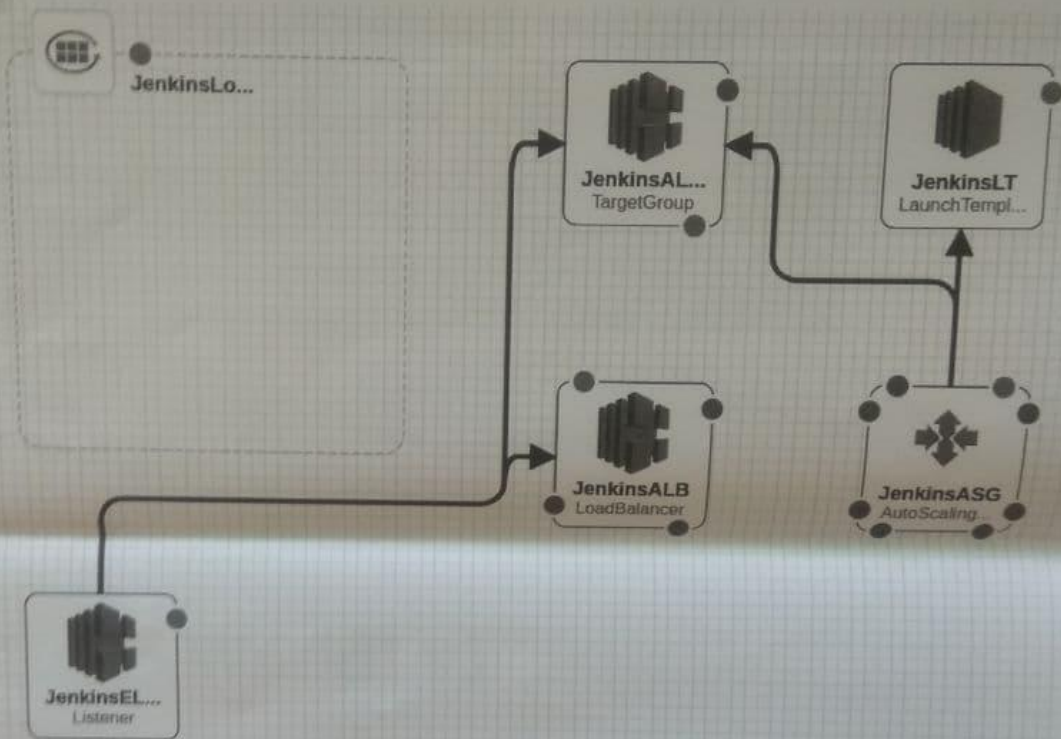
13.02070849.00043 ПЛ1					Розробка комп'ютерної системи для безперервної роботи і доставки даних у хмару платформи AWS		
За	Лист	№ докум.	Шлях	Дата	Проектування мережі		
Розроб.	Фініш В.С.				Лист 1 / Листів 1		
Перев.	Сергійчук С.Ю.				ІНУ "Інформаційна політика" КНТ-325		
Т. контр.							
Н. контр.	Шербак Н.В.						
Затверд.	Кудерметов Р.К.						

Проектування сервісу для розгортання докер образу



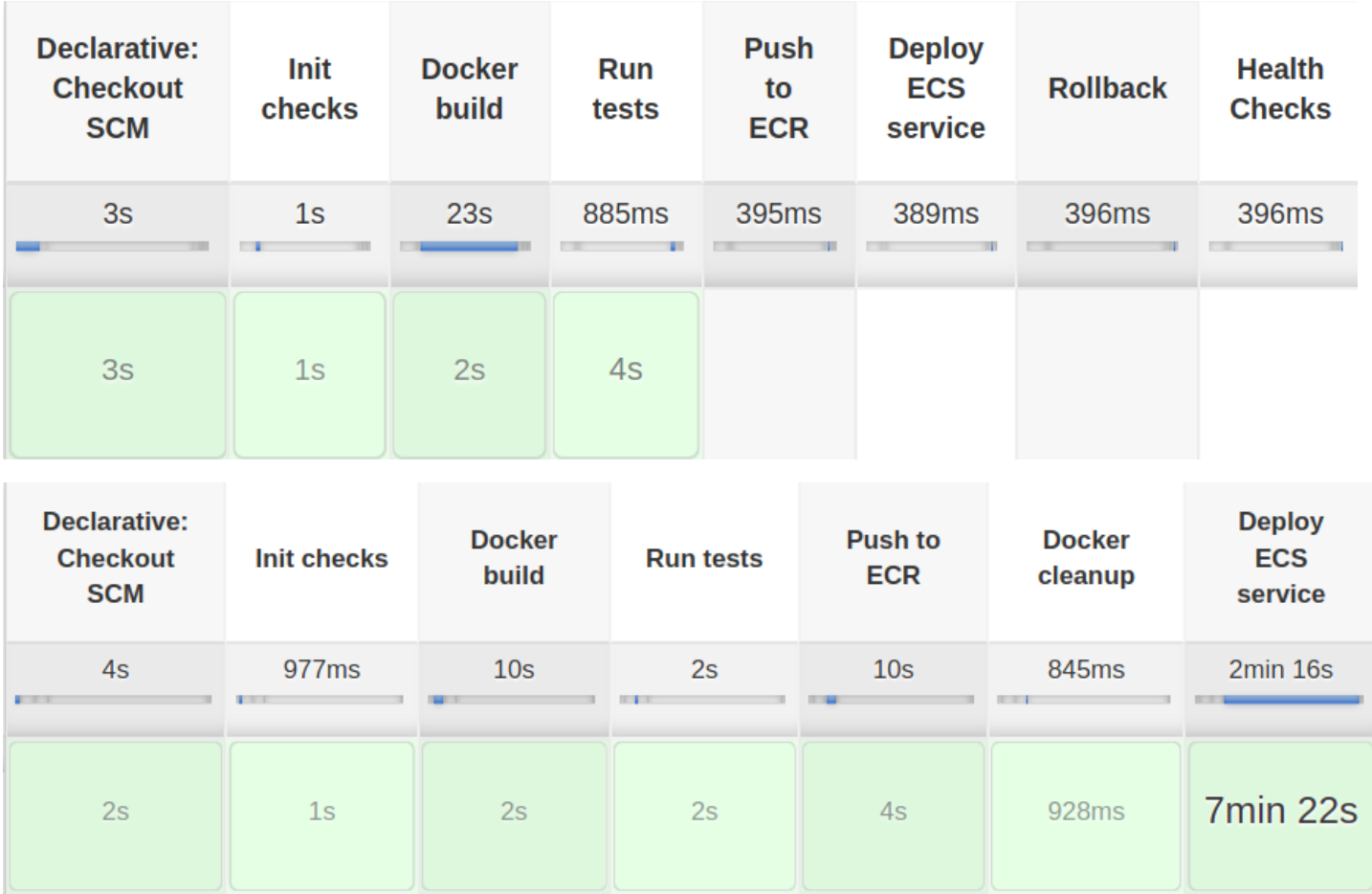
				13.02070849.00043 ПЛІ			
				Розробка комп'ютерної системи для біометричної ідентифікації і доставки послуг до з'ясування інформації АВС			
				Проектування сервісу для розгортання шлюзу об'єкта			
Зм.		Лист		№ докум.		Підп.	
Розроб.		Філіпп В.С.		/		/	
Перев.		Скороходський С.Ю.		/		/	
І. випуск.						Лист 1 / Листів 1	
Н. Кошар.		Шербан Н.В.		/		ІНУ "Інформаційна політехніка"	
Замовит.		Кудерницький Р.Б.		/		КНТ-327	

Проектування сервера Jenkins



					13.02070849.00043 ПЛ/ТЗ		
№	Лист	№ докум.	Подп.	Дата	Результат компьютерной системы для безаварийной интеграции с системой доставки и хранения платформы AWS		
Розроб.		Філімо В.С.			Проектування сервера Jenkins		
Перев.		Скрябинский С.Ю.					
Утвер.							
И. Эксперт		Шербина Н.В.			Лист 1 Листов 1		
Заказчик		Кузнецов Р.Б.			ИТУ "Защитная технология" КНТ-327		

Результати роботи комп'ютерної системи



☐	Image tag	Size (MB)	▼	Image URI	Digest	Scan status
☐	v1	97.23		Copy URI	sha256:53fb3c40c2b1c84...	Complete

☐	Image tag	Size (MB)	▼	Image URI	Digest	Scan status
☐	latest	100.01		Copy URI	sha256:438dbac5abdbc5...	Complete

Hello World! v1

Hello World! v2

					13.02070849.00043 ПЛ14						
					Розробка комп'ютерної системи для безперервної інтеграції і доставки додатків у хмарну платформу AWS Результати роботи комп'ютерної системи	Лист.			Маса	Масштаб	
Зм.	Лист.	№ докум.	Підп.	Дата							
Розроб.		Фінько В.С.									
Перев.		Скрупський С.Ю.									
Т. контр.						Лист 1		Листів 1			
						НУ "Запорізька політехніка" КНТ-527					
Н. Контр.		Щербак Н.В.									
Затверд.		Кудерметов Р.К.									