ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Автоматизований CI/CD на основі Kubernetes та AWS із зосередженням на розробці»

на здобуття освітнього ступеня бакалавра
зі спеціальності 123 Комп'ютерної інженерії
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерна інженерія
(назва)

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

_____ *Євгеній ГРИШКОВЕЦЬ*
(підпис) *Ім'я, ПРІЗВИЩЕ здобувача*

Виконав: здобувач(ка) вищої освіти гр. КІД-41
Євгеній ГРИШКОВЕЦЬ
Ім'я, ПРІЗВИЩЕ
Керівник: Артем АНТОНЕНКО
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ
Рецензент: Віктор ВИШНІВСЬКИЙ
науковий ступінь, вчене звання Ім'я, ПРІЗВИЩЕ

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра 123 Комп'ютерної інженерії

Ступінь вищої освіти бакалавр

Спеціальність 123 Комп'ютерної інженерії

Освітньо-професійна програма Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедрою Наталія ЛАЩЕВСЬКА
_____Ім'я, ПРІЗВИЩЕ
«_____» _____ 2024р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Гришковець Євгеній Петрович
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: _____
керівник кваліфікаційної роботи Артем АНТОНЕНКО ,
(Ім'я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «_____» _____2024р. № _____
2. Строк подання кваліфікаційної роботи «_____» _____ 2024р.
3. Вихідні дані до кваліфікаційної роботи: _____
Визначення та аналіз основних концепцій CI/CD.
Вивчення переваг та викликів автоматизації CI/CD.
Огляд хмарних технологій, таких як Kubernetes та AWS, їх можливості для автоматизації CI/CD.
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Збір існуючих вимог та аналіз існуючих рішень
Розробка архітектури CI/CD на основі Kubernetes та AWS
Інтеграція AWS сервісів у CI/CD процеси
Автоматизація розгортання та управління інфраструктурою
5. Перелік ілюстративного матеріалу: *презентація*
6. Дата видачі завдання«_____» _____ 2024р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Збір даних	26.02-22.03.2024	Виконано
2	Обробка матеріалу	23.03-05.04.2024	Виконано
3	Розробка архітектури CI/CD на основі Kubernetes та AWS	06.04-10.04.2024	Виконано
4	Розгортання та конфігурація Kubernetes	11.04-30.04.2024	Виконано
5	Інтеграція AWS сервісів у CI/CD процеси	01.05-03.05.2024	Виконано
6	Висновки за результатами аналізу	04.05-08.05.2024	Виконано
7	Розробка презентації	09.05-11.05.2024	Виконано
8	Передзахист	14.05-17.05.2024	Виконано
9	Здача в деканат	25.05-1.06.2024	Виконано

Здобувач(ка) вищої освіти _____ Євгеній ГРИШКОВЕЦЬ

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

Керівник кваліфікаційної роботи _____ Артем АНТОНЕНКО

(підпис) *(Ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 45 стор., 28 рис., 9 джерел.

Мета роботи – розробка та впровадження автоматизованої системи CI/CD на основі Kubernetes та AWS з акцентом на ефективність процесу розробки програмного забезпечення.

Об'єкт дослідження – процес автоматизації CI/CD в хмарному середовищі.

Предмет дослідження – технології та інструменти для автоматизації CI/CD з використанням Kubernetes та AWS.

Короткий зміст роботи: Ця робота присвячена дослідженню автоматизованих систем CI/CD, які використовують Kubernetes та AWS. Основна увага приділяється тому, як ці технології можуть покращити процеси розробки програмного забезпечення, роблячи їх швидшими та надійнішими.

Робота включає огляд основних концепцій та переваг автоматизації CI/CD. Розглядаються можливості, які надають Kubernetes і AWS для побудови ефективних конвеєрів розгортання, а також аналізуються деякі виклики, які виникають під час їх впровадження.

На основі проведеного дослідження було розроблено практичні рішення для автоматизації CI/CD. Ці рішення показали свою ефективність у реальних умовах, допомагаючи зменшити кількість помилок і прискорити процеси розробки та розгортання.

Незважаючи на певні труднощі, які виникають під час налаштування таких систем, результати дослідження підтверджують, що автоматизовані CI/CD системи на основі Kubernetes та AWS мають значний потенціал для покращення роботи команд розробників.

ЗМІСТ

ВСТУП

Дослідження та аналіз, проведені у даній кваліфікаційній роботі, дозволяють зробити висновки щодо впровадження автоматизованого CI/CD конвеєру на основі Kubernetes та AWS з використанням Jenkins для орієнтації на розробку. У сучасних умовах розробки програмного забезпечення, автоматизація процесів CI/CD є важливою для забезпечення ефективності, стабільності та надійності. Використання Kubernetes та AWS надає можливість створення масштабованих, доступних та безпечних інфраструктур для CI/CD конвеєрів.

У роботі аналізуються різні підходи до налаштування та інтеграції інструментів, таких як Jenkins, ArgoCD та Sonarqube, для побудови ефективного CI/CD конвеєру. Результати досліджень показують, що використання цих інструментів дозволяє значно скоротити час розгортання додатків, підвищити їх якість та забезпечити безперервну інтеграцію і доставку.

З аналізу впровадження різних компонентів CI/CD конвеєру, таких як налаштування Jenkins сервера на AWS, створення EKS кластеру для розгортання додатків, використання ArgoCD для безперервної доставки та інтеграція Sonarqube для аналізу якості коду, можна зробити висновок, що така система значно підвищує ефективність роботи команд розробників. Однак, деякі аспекти, такі як налаштування моніторингу та підключення домену, не були розглянуті у даній роботі, що може бути напрямком для подальших досліджень.

Таким чином, впровадження автоматизованого CI/CD конвеєру на основі Kubernetes та AWS з використанням Jenkins, ArgoCD та Sonarqube є ефективним підходом для забезпечення високої якості та надійності розробки програмного забезпечення.

РОЗДІЛ 1. ОСНОВИ CI/CD ТА ХМАРНІ ТЕХНОЛОГІЇ

1.1 Визначення та концепції CI/CD

CI/CD (Continuous Integration/Continuous Deployment або Continuous Delivery) — це методологія, що застосовується в розробці програмного забезпечення для автоматизації процесів інтеграції, тестування та розгортання коду. Вона дозволяє командам розробників швидше та якісніше доставляти нові функціональні можливості, виправлення помилок і оновлення безпеки.

Continuous Integration (CI) — це практика регулярного злиття змін у коді з центральним репозиторієм кілька разів на день. Кожне злиття автоматично перевіряється за допомогою автоматизованих тестів. Це дозволяє виявляти помилки на ранніх стадіях розробки та зменшує ризики під час інтеграції нових функцій. Основною метою CI є забезпечення стабільності та якості коду, а також скорочення часу, необхідного для виправлення помилок.

Continuous Deployment (CD) — це практика автоматичного розгортання кожної успішно інтегрованої зміни в коді на продуктивне середовище. Цей процес передбачає повну автоматизацію всіх етапів, включаючи збірку, тестування та розгортання. CD дозволяє швидко доставляти зміни користувачам, мінімізуючи затримки між написанням коду та його використанням.

Continuous Delivery (CD) — це методологія, що передбачає автоматизацію всіх процесів до етапу розгортання, залишаючи саме розгортання ручним. Таким чином, зміни можуть бути розгорнуті у будь-який момент після успішного проходження автоматичних тестів. Цей підхід зменшує ризики, пов'язані з великими оновленнями, дозволяючи частіше випускати малі та перевірені зміни.

Основні концепції, що лежать в основі CI/CD, включають:

- Автоматизація — ключовий аспект CI/CD. Всі процеси, від написання коду до його розгортання, мають бути автоматизовані для забезпечення надійності та швидкості. Це включає автоматичні збірки, тестування, розгортання та моніторинг. Автоматизація дозволяє зменшити кількість людських помилок і забезпечує повторюваність процесів;
- Регулярне злиття коду — розробники повинні часто зливати зміни в центральний репозиторій. Це дозволяє швидко виявляти та виправляти помилки, забезпечуючи постійне оновлення кодової бази. Кожне злиття тригерить автоматичні збірки та тести, що забезпечує стабільність та інтеграцію змін;
- Автоматичне тестування — для кожного злиття коду проводяться автоматизовані тести, що включають юніт-тести, інтеграційні тести, функціональні тести тощо. Це забезпечує високу якість коду та дозволяє швидко виявляти дефекти. Автоматичні тести допомагають підтримувати високу якість програмного забезпечення, зменшуючи ризик впровадження помилок у продуктивне середовище;
- Постійне розгортання — автоматизація процесу розгортання дозволяє швидко доставляти нові функції та виправлення помилок до кінцевих користувачів. Це забезпечує більш часті та передбачувані релізи, що підвищує задоволеність користувачів. Постійне розгортання дозволяє уникнути великих ризиків, пов'язаних з масштабними оновленнями;
- Зворотній зв’язок — швидкий зворотній зв’язок про статус кожної збірки та розгортання допомагає командам розробників оперативно реагувати на проблеми та покращувати свої процеси. Моніторинг та логування дозволяють отримувати інформацію про продуктивність та стабільність системи, виявляючи потенційні проблеми ще до того, як вони вплинуть на користувачів;

Переваги CI/CD:

− Покращена якість коду — автоматичне тестування на кожному етапі забезпечує виявлення дефектів на ранніх стадіях. Це зменшує кількість помилок у продуктивному середовищі та підвищує якість кінцевого продукту;

− Швидке виявлення та виправлення помилок — регулярне злиття та автоматичне тестування дозволяє швидко ідентифікувати та виправляти помилки. Це знижує ризик затримок і поліпшує стабільність системи;

− Швидка доставка функцій — автоматизація процесів розгортання скорочує час від написання коду до його доступності для користувачів. Це підвищує задоволеність клієнтів, оскільки нові функції та виправлення помилок стають доступними швидше;

− Зниження ризиків — автоматизація та часте тестування знижують ризик виникнення помилок на продуктивному середовищі. Це забезпечує стабільність і надійність системи, зменшуючи ризик відкатів і аварійних ситуацій;

− Підвищена продуктивність команди — автоматизація рутинних завдань дозволяє розробникам зосередитися на творчих та інноваційних аспектах роботи. Це підвищує продуктивність команди та сприяє швидшому досягненню поставлених цілей;

− Покращене співробітництво — CI/CD сприяє тіснішій співпраці між розробниками, тестувальниками та операційними командами. Це дозволяє уникнути "синдрому передавання" та забезпечує більш ефективну комунікацію;

Впровадження CI/CD є важливим кроком у розвитку ефективних та надійних процесів розробки програмного забезпечення. Це дозволяє командам розробників швидше та якісніше доставляти свої продукти кінцевим користувачам, забезпечуючи високу якість, стабільність та надійність програмного забезпечення.

1.2 Переваги та виклики автоматизації CI/CD

Автоматизація CI/CD (Continuous Integration/Continuous Deployment) надає значні переваги для команд розробників, але також має свої виклики, які необхідно враховувати під час впровадження. Однією з основних переваг є покращення якості коду. Завдяки автоматизованому тестуванню на кожному етапі інтеграції, дефекти виявляються на ранніх стадіях розробки, що значно зменшує кількість помилок у фінальному продукті. Це підвищує надійність і стабільність додатків, що в свою чергу, підвищує задоволеність користувачів.

Іншою важливою перевагою є прискорення випуску оновлень. CI/CD значно прискорює процес випуску нових версій програмного забезпечення завдяки автоматизованим збіркам, тестам та деплойменту. Це дозволяє швидше реагувати на зміни ринкових умов і вимог користувачів, забезпечуючи своєчасне впровадження нових функцій та виправлень. Також автоматизація підвищує ефективність команд. Завдяки автоматизації рутинних завдань, розробники можуть зосередитися на виконанні більш творчих та інноваційних завдань. Це сприяє кращій координації роботи команди та підвищенню продуктивності.

Крім того, автоматизація зменшує кількість людських помилок під час складання, тестування та розгортання додатків, що є особливо важливим для великих і складних проєктів. Автоматизовані процеси забезпечують кращу видимість і контроль, дозволяючи легко відстежувати статуси збірок, тестів та деплойментів. Це допомагає у швидкому виявленні та вирішенні проблем, що виникають під час розробки.

Однак, впровадження автоматизації CI/CD має і свої виклики. Перш за все, це складність налаштування. Впровадження CI/CD потребує ретельного планування та налаштування інфраструктури, інструментів та процесів, що може вимагати значних ресурсів і часу на початкових етапах. Інтеграція з існуючими

системами також може бути складною, особливо якщо вони не були розроблені з урахуванням сучасних методик DevOps. Це може призвести до необхідності модернізації або заміни деяких компонентів.

Витрати на навчання та підтримку також є важливим викликом. Розробники та інші члени команди повинні пройти навчання для ефективного використання CI/CD інструментів та процесів. Крім того, підтримка та оновлення цих інструментів також вимагає часу та ресурсів. Проблеми з безпекою є ще одним викликом. Автоматизація процесів розгортання може відкривати нові вразливості, особливо якщо не приділяти належної уваги безпеці на кожному етапі CI/CD. Необхідно впроваджувати додаткові заходи безпеки для захисту коду, даних та інфраструктури.

Масштабованість також може стати проблемою. Зі збільшенням обсягу проєкту та команди може виникнути необхідність масштабування CI/CD інфраструктури. Це може вимагати додаткових інвестицій у обладнання та програмне забезпечення, а також оптимізації існуючих процесів. Незважаючи на ці виклики, автоматизація CI/CD є важливим елементом сучасної розробки програмного забезпечення, що забезпечує численні переваги. Успішне впровадження вимагає врахування та подолання низки викликів, що стоять перед командами розробників, проте результати, які досягаються завдяки CI/CD, виправдовують затрати і зусилля.

1.3 Огляд хмарних технологій: Kubernetes та AWS

Хмарні технології стали основою сучасної IT-інфраструктури, надаючи розробникам і компаніям можливість ефективно масштабувати свої додатки та сервіси. Дві з найпопулярніших хмарних платформ — Kubernetes і AWS (Amazon Web Services) — забезпечують потужні інструменти для автоматизації розгортання, масштабування та управління додатками.

Kubernetes (K8s) — це система з відкритим кодом для автоматизації розгортання, масштабування та управління контейнеризованими додатками. Вона була розроблена компанією Google і передана до Cloud Native Computing Foundation (CNCF). Kubernetes забезпечує потужні засоби для управління контейнерами, дозволяючи розробникам легко масштабувати свої додатки та забезпечувати їх надійну роботу.

Однією з основних функцій Kubernetes є оркестрація контейнерів. Це означає, що Kubernetes автоматично управляє контейнерами, забезпечуючи їх запуск, зупинку та перезапуск у разі необхідності. Це дозволяє забезпечити високу доступність додатків і мінімізувати час простою. Крім того, Kubernetes забезпечує автоматичне масштабування додатків залежно від навантаження, що дозволяє оптимально використовувати ресурси. Це досягається шляхом автоматичного додавання або видалення контейнерів у кластері.

Крім того, Kubernetes забезпечує засоби для комунікації між контейнерами, як всередині одного вузла, так і між різними вузлами в кластері. Це дозволяє створювати складні мікросервісні архітектури. Управління конфігурацією та секретами також є важливою функцією Kubernetes. Система дозволяє зберігати і управляти конфігураціями додатків, секретами, такими як паролі та ключі шифрування, що забезпечує безпеку та зручність управління.

Amazon Web Services (AWS) є однією з найбільш популярних хмарних платформ, що надає широкий спектр послуг для розробки, розгортання та управління додатками. AWS пропонує інфраструктуру як послугу (IaaS), платформу як послугу (PaaS) та програмне забезпечення як послугу (SaaS), що дозволяє користувачам вибирати необхідні інструменти та сервіси відповідно до своїх потреб.

AWS забезпечує високу надійність і масштабованість завдяки своїй глобальній інфраструктурі, яка включає численні дата-центри по всьому світу. Це дозволяє розробникам легко масштабувати свої додатки, забезпечуючи високу доступність і продуктивність. Крім того, AWS пропонує широкий спектр сервісів для зберігання даних, обробки великих даних, штучного інтелекту, машинного

навчання та інтернету речей (IoT).

Однією з ключових особливостей AWS є її інтеграція з Kubernetes через сервіс Amazon EKS (Elastic Kubernetes Service). EKS дозволяє розробникам запускати Kubernetes кластери на інфраструктурі AWS, використовуючи всі переваги хмарної платформи. Це забезпечує гнучкість та масштабованість Kubernetes у поєднанні з надійністю та безпекою AWS.

Використання Kubernetes та AWS спільно надає значні переваги для розробників та компаній. Kubernetes дозволяє автоматизувати управління контейнерами та забезпечує високу доступність додатків, тоді як AWS забезпечує надійну та масштабовану інфраструктуру для розгортання та управління додатками. Разом ці технології забезпечують ефективну та надійну платформу для розробки, розгортання та управління сучасними додатками.

Впровадження Kubernetes та AWS дозволяє компаніям швидше реагувати на зміни ринкових умов, забезпечувати високу якість своїх додатків та знижувати витрати на управління IT-інфраструктурою. Однак, для успішного використання цих технологій необхідно враховувати їхні специфічні особливості та вимоги, що включає навчання команди, налаштування інфраструктури та впровадження практик безпеки.

Таким чином, Kubernetes та AWS є потужними інструментами для розробки та управління хмарними додатками, що забезпечують високу надійність, масштабованість та ефективність. Використання цих технологій разом дозволяє компаніям досягати високих результатів у розробці та впровадженні своїх продуктів, забезпечуючи конкурентні переваги на ринку

1.4 Взаємодія CI/CD з хмарними платформами

Взаємодія Continuous Integration/Continuous Deployment (CI/CD) з хмарними платформами є важливим аспектом сучасної розробки програмного забезпечення, що дозволяє автоматизувати та оптимізувати процеси розробки, тестування та розгортання додатків. Хмарні платформи, такі як Amazon Web Services (AWS), Google Cloud Platform (GCP) та Microsoft Azure, надають потужні інструменти та сервіси для інтеграції CI/CD, що допомагають забезпечити безперервний розвиток і випуск програмного забезпечення.

Автоматизація CI/CD на хмарних платформах дозволяє розробникам легко інтегрувати свої коди, автоматично виконувати тести та швидко розгортати нові версії додатків. Це досягається за допомогою різноманітних інструментів, таких як Jenkins, GitLab CI, CircleCI, Travis CI та інших, які забезпечують повну автоматизацію процесів розробки. Наприклад, Jenkins, як один з найбільш популярних інструментів CI/CD, дозволяє налаштовувати складні пайплайни для збірки, тестування та розгортання додатків на хмарних платформах. Завдяки своїй модульній архітектурі, Jenkins може інтегруватися з різними хмарними сервісами, такими як AWS, GCP та Azure, забезпечуючи автоматизоване розгортання контейнерів, віртуальних машин або безсерверних функцій.

Однією з ключових переваг взаємодії CI/CD з хмарними платформами є можливість автоматичного масштабування ресурсів залежно від навантаження. Хмарні платформи надають динамічне масштабування обчислювальних ресурсів, що дозволяє автоматично збільшувати або зменшувати кількість запущених інстанцій залежно від потреб додатка. Це особливо корисно для великих проектів з високим трафіком, де навантаження може різко змінюватися. Інтеграція CI/CD з хмарними платформами також забезпечує високу доступність додатків. Хмарні платформи пропонують розподілену архітектуру з багатьма дата-центрами по всьому світу, що забезпечує безперервну доступність додатків навіть у випадку відмови одного або декількох дата-центрів. Це дозволяє мінімізувати ризик

простоїв і забезпечити надійну роботу додатків.

Безпека також є важливим аспектом взаємодії CI/CD з хмарними платформами. Хмарні провайдери забезпечують широкий спектр засобів безпеки, включаючи шифрування даних, управління доступом, моніторинг та виявлення загроз. Інтеграція CI/CD з хмарними платформами дозволяє автоматично впроваджувати ці заходи безпеки в процеси розробки та розгортання, забезпечуючи захист коду та даних на всіх етапах життєвого циклу додатка.

Крім того, хмарні платформи надають інструменти для моніторингу та логування додатків, що дозволяє відстежувати продуктивність, виявляти та вирішувати проблеми в реальному часі. Інтеграція CI/CD з цими інструментами дозволяє автоматично збирати та аналізувати дані про роботу додатків, забезпечуючи швидке виявлення та усунення проблем.

Інтеграція CI/CD з хмарними платформами значно спрощує процес розробки та розгортання додатків, забезпечуючи автоматизацію, масштабованість, доступність, безпеку та моніторинг. Це дозволяє розробникам зосередитися на створенні високоякісного коду, знижуючи витрати на управління інфраструктурою та підвищуючи продуктивність команд розробки. Завдяки взаємодії CI/CD з хмарними платформами компанії можуть швидко та ефективно впроваджувати нові функції, покращувати продуктивність додатків та забезпечувати високу якість своїх продуктів на ринку

РОЗДІЛ 2. ВПРОВАДЖЕННЯ CI/CD ЗА ДОПОМОГОЮ KUBERNETES ТА AWS

2.1 Архітектура CI/CD з використанням Kubernetes

Архітектура CI/CD (Continuous Integration/Continuous Deployment) з використанням Kubernetes є потужним підходом до автоматизації процесів розробки, тестування та розгортання додатків. Kubernetes, як платформа для оркестрації контейнерів, забезпечує високу доступність, масштабованість та ефективність управління контейнеризованими додатками, що робить його ідеальним вибором для побудови CI/CD пайплайнів.

У типовій архітектурі CI/CD з використанням Kubernetes, ключовими компонентами є система керування вихідним кодом, інструмент CI/CD для автоматизації процесів збірки, тестування та розгортання, та Kubernetes кластер для оркестрації контейнерів. Ці компоненти працюють разом для забезпечення безперервного потоку розробки та доставки програмного забезпечення.

Процес починається з внесення змін до вихідного коду розробниками. Код зберігається у системі керування версіями, такій як Git. Коли нові зміни відправляються до репозиторію, інструмент CI/CD, такий як Jenkins, GitLab CI, або CircleCI, автоматично виявляє ці зміни та запускає пайплайн збірки. У цьому пайплайні інструмент CI/CD виконує ряд кроків, таких як збірка коду, запуск автоматизованих тестів та створення контейнерних образів за допомогою Docker. Контейнерні образи зберігаються у реєстрі контейнерів, такому як Docker Hub або Amazon ECR.

Після успішного завершення збірки та тестування, наступним кроком є розгортання додатків у Kubernetes кластері. Kubernetes використовує маніфести для визначення стану додатка, включаючи кількість реплік, конфігурацію мережі та інші параметри. Інструмент CI/CD автоматично застосовує ці маніфести до

Kubernetes кластера, що дозволяє оркеструвати розгортання контейнерів.

Kubernetes забезпечує автоматичне масштабування додатків на основі навантаження. Це означає, що кількість активних реплік додатка може автоматично збільшуватися або зменшуватися залежно від трафіку та ресурсів. Це забезпечує високу доступність та продуктивність додатків навіть під час пікових навантажень. Крім того, Kubernetes забезпечує відновлення після збоїв шляхом автоматичного перезапуску контейнерів у разі їх відмови, що мінімізує час простою додатків.

Однією з ключових переваг використання Kubernetes у CI/CD архітектурі є можливість розгортання різних середовищ, таких як середовище розробки, тестування та продакшн. Це дозволяє командам розробників легко тестувати нові функції та зміни у безпечному середовищі перед їх впровадженням у продакшн. Kubernetes також забезпечує ізоляцію середовищ, що знижує ризик конфліктів між різними версіями додатків.

Безпека є ще одним важливим аспектом архітектури CI/CD з використанням Kubernetes. Kubernetes забезпечує засоби для управління секретами та конфігураціями, що дозволяє безпечно зберігати паролі, ключі шифрування та інші конфіденційні дані. Інтеграція з інструментами безпеки, такими як Open Policy Agent (OPA) та Aqua Security, дозволяє забезпечити відповідність додатків політикам безпеки та стандартам.

Моніторинг та логування також є важливими компонентами архітектури CI/CD з використанням Kubernetes. Інструменти, такі як Prometheus, Grafana та Elasticsearch, забезпечують збір та аналіз метрик продуктивності та логів додатків. Це дозволяє командам розробників виявляти та вирішувати проблеми у реальному часі, що підвищує стабільність та надійність додатків.

Таким чином, архітектура CI/CD з використанням Kubernetes забезпечує ефективну та надійну платформу для автоматизації процесів розробки, тестування та розгортання додатків. Використання Kubernetes дозволяє досягти високої доступності, масштабованості та безпеки додатків, що є ключовими вимогами сучасних IT-систем. Завдяки цій архітектурі компанії можуть швидко та

ефективно впроваджувати нові функції, підвищуючи якість своїх продуктів та задовольняючи вимоги користувачів.

2.2 Розгортання та конфігурація Kubernetes для CI/CD

Розгортання та конфігурація Kubernetes для CI/CD є важливими етапами в забезпеченні ефективного і надійного процесу безперервної інтеграції та розгортання додатків. Kubernetes надає потужні інструменти та функції для автоматизації управління контейнеризованими додатками, що дозволяє зменшити витрати на адміністрування та підвищити продуктивність розробки.

Першим кроком у розгортанні Kubernetes є налаштування кластеру. Це можна зробити за допомогою різних хмарних платформ, таких як Amazon EKS (Elastic Kubernetes Service), Google GKE (Google Kubernetes Engine) або Azure AKS (Azure Kubernetes Service). Ці сервіси надають керовані рішення, які спрощують розгортання та управління Kubernetes кластерами. Для розгортання кластеру потрібно вибрати відповідний регіон, конфігурацію вузлів та налаштування мережі.

Після розгортання кластеру необхідно налаштувати автентифікацію та авторизацію для забезпечення безпеки доступу до кластеру. Kubernetes підтримує різні механізми автентифікації, включаючи токени, сертифікати та інтеграцію з зовнішніми системами автентифікації, такими як LDAP або OAuth. Авторизація здійснюється за допомогою ролей та правил, що визначають, які дії можуть виконувати користувачі та сервіси в кластері.

Наступним кроком є налаштування CI/CD інструментів для інтеграції з Kubernetes. Jenkins є одним з найпопулярніших інструментів для автоматизації CI/CD, який можна легко інтегрувати з Kubernetes. Для цього потрібно розгорнути Jenkins в кластері Kubernetes, налаштувати його для автоматичного створення, тестування та розгортання контейнерів. Jenkins використовує файли конфігурації,

такі як Jenkinsfile, для визначення пайплайнів збірки, тестування та розгортання.

GitLab CI є ще одним популярним інструментом, який підтримує інтеграцію з Kubernetes. GitLab CI дозволяє автоматизувати процеси CI/CD за допомогою файлів конфігурації .gitlab-ci.yml, в яких визначаються етапи збірки, тестування та розгортання. GitLab CI може автоматично створювати контейнери Docker, зберігати їх в реєстрах контейнерів і розгортати їх в кластері Kubernetes.

Конфігурація Kubernetes для CI/CD також включає налаштування Persistent Volumes (PV) та Persistent Volume Claims (PVC) для зберігання даних додатків. Це дозволяє забезпечити збереження даних під час перезапуску або масштабування контейнерів. Крім того, необхідно налаштувати ConfigMaps та Secrets для зберігання конфігураційних даних та секретів, таких як паролі та ключі шифрування. ConfigMaps та Secrets дозволяють передавати конфігураційні параметри та секрети в контейнери безпосередньо під час їх запуску.

Моніторинг та логування є важливими аспектами розгортання та конфігурації Kubernetes для CI/CD. Інструменти, такі як Prometheus та Grafana, забезпечують збір та аналіз метрик продуктивності додатків та кластеру. Fluentd та Elasticsearch використовуються для збору та аналізу логів додатків. Це дозволяє виявляти та вирішувати проблеми у реальному часі, забезпечуючи високу надійність та продуктивність додатків.

Нарешті, розгортання та конфігурація Kubernetes для CI/CD включає налаштування автоматичного масштабування додатків на основі навантаження. Kubernetes підтримує автоматичне горизонтальне та вертикальне масштабування контейнерів, що дозволяє автоматично збільшувати або зменшувати кількість активних реплік додатка залежно від трафіку та ресурсів. Це забезпечує високу доступність та продуктивність додатків навіть під час пікових навантажень.

Таким чином, розгортання та конфігурація Kubernetes для CI/CD включає налаштування кластеру, автентифікацію та авторизацію, інтеграцію з CI/CD інструментами, налаштування збереження даних, моніторинг та логування, а також автоматичне масштабування додатків. Використання Kubernetes для CI/CD забезпечує ефективне управління контейнеризованими додатками, автоматизацію

процесів розробки та розгортання, а також високу надійність та продуктивність додатків.

2.3 Інтеграція AWS сервісів у CI/CD процеси

Інтеграція сервісів Amazon Web Services (AWS) у процеси Continuous Integration/Continuous Deployment (CI/CD) значно підвищує ефективність, надійність та масштабованість розробки й розгортання програмного забезпечення. AWS надає широкий спектр сервісів, які можуть бути інтегровані у CI/CD пайплайни для автоматизації та оптимізації всіх етапів життєвого циклу додатків.

Першим кроком в інтеграції AWS сервісів у CI/CD процеси є налаштування сховища вихідного коду. AWS CodeCommit є керованим сервісом для контролю версій, що забезпечує надійне та безпечне зберігання вихідного коду. CodeCommit дозволяє розробникам зберігати, відстежувати та керувати змінами у своєму коді, забезпечуючи високу доступність та інтеграцію з іншими сервісами AWS.

Для автоматизації процесів збірки та тестування використовуються AWS CodeBuild та AWS CodePipeline. CodeBuild є керованим сервісом для компіляції коду, запуску тестів та створення артефактів. CodeBuild автоматично масштабується для обробки кількох збірок паралельно, що скорочує час на збірку та тестування. CodePipeline є сервісом для автоматизації процесів CI/CD, який дозволяє створювати, налаштовувати та керувати пайплайнами для збірки, тестування та розгортання додатків. CodePipeline інтегрується з CodeCommit, CodeBuild та іншими сервісами AWS, забезпечуючи безперервний потік розробки.

AWS CodeDeploy є ключовим сервісом для автоматизації розгортання додатків. CodeDeploy підтримує розгортання на EC2 інстанції, сервери Lambda та інші обчислювальні сервіси AWS. Він забезпечує безперебійне оновлення додатків, знижуючи ризик простоїв та помилок під час розгортання. CodeDeploy також підтримує різні стратегії розгортання, такі як "blue/green" та "rolling", що

дозволяє вибрати найбільш відповідний підхід для конкретного випадку.

Для зберігання контейнерних образів та інтеграції з Docker використовується Amazon Elastic Container Registry (ECR). ECR є керованим реєстром контейнерів, що дозволяє легко зберігати, керувати та розгортати контейнерні образи. ECR інтегрується з AWS CodePipeline та CodeBuild, забезпечуючи автоматичне створення та завантаження контейнерних образів у реєстр під час пайплайну CI/CD.

AWS Lambda є сервісом для безсерверних обчислень, що дозволяє запускати код без управління серверами. Інтеграція AWS Lambda у CI/CD процеси дозволяє автоматизувати виконання різних задач, таких як обробка подій, запуск тестів або виконання скриптів під час пайплайну. Lambda може бути викликана з CodePipeline або CodeBuild для виконання специфічних задач на різних етапах розробки та розгортання.

Моніторинг та логування є важливими аспектами інтеграції AWS сервісів у CI/CD процеси. AWS CloudWatch є сервісом для моніторингу ресурсів та додатків у реальному часі. CloudWatch збирає та аналізує метрики, логі та події з різних сервісів AWS, забезпечуючи візуалізацію даних та налаштування сповіщень про проблеми. Інтеграція CloudWatch у CI/CD процеси дозволяє автоматично відстежувати стан додатків, виявляти та вирішувати проблеми, що виникають під час збірки, тестування або розгортання.

AWS Secrets Manager є сервісом для управління конфіденційними даними, такими як ключі API, паролі та сертифікати. Інтеграція Secrets Manager у CI/CD процеси забезпечує безпечне зберігання та використання конфіденційних даних під час автоматизації збірки та розгортання додатків. Secrets Manager дозволяє автоматично обертати секрети, що підвищує безпеку та відповідність політикам безпеки.

Таким чином, інтеграція AWS сервісів у CI/CD процеси забезпечує автоматизацію, безпеку та надійність розробки та розгортання додатків. Використання AWS CodeCommit, CodeBuild, CodePipeline, CodeDeploy, ECR, Lambda, CloudWatch та Secrets Manager дозволяє створювати потужні та гнучкі

пайплайни для безперервної інтеграції та доставки програмного забезпечення. Це сприяє підвищенню продуктивності розробників, скороченню часу на розгортання та покращенню якості продуктів, що відповідають вимогам сучасних ринків.

2.4 Автоматизація розгортання та управління інфраструктурою

Автоматизація розгортання та управління інфраструктурою є критично важливою для забезпечення ефективності, надійності та масштабованості сучасних додатків. Використання інструментів та практик автоматизації дозволяє значно знизити ризик людських помилок, скоротити час на розгортання і управління ресурсами, а також підвищити гнучкість та адаптивність IT-інфраструктури.

Одним з ключових інструментів для автоматизації інфраструктури є Infrastructure as Code (IaC), який дозволяє описувати та керувати інфраструктурою за допомогою машинозчитуваних конфігураційних файлів. IaC забезпечує консистентність і повторюваність процесів розгортання, дозволяючи командам розробників і адміністраторів легко відтворювати і модифікувати інфраструктуру. Популярні інструменти для IaC включають Terraform, AWS CloudFormation та Ansible.

Terraform є одним з найбільш поширених інструментів для IaC, що підтримує широкий спектр провайдерів хмарних сервісів, включаючи AWS, Google Cloud та Azure. За допомогою Terraform можна визначати інфраструктуру як код у вигляді декларативних конфігураційних файлів, які описують бажаний стан інфраструктури. Terraform автоматично обчислює залежності між ресурсами, створює та змінює ресурси відповідно до конфігураційного файлу. Це дозволяє легко розгортати, масштабувати та оновлювати інфраструктуру.

AWS CloudFormation є керованим сервісом AWS для IaC, який дозволяє визначати інфраструктуру за допомогою шаблонів YAML або JSON.

CloudFormation автоматично створює та налаштовує ресурси AWS відповідно до визначеного шаблону, забезпечуючи консистентність і повторюваність процесів розгортання. CloudFormation також підтримує автоматичне оновлення і видалення ресурсів, що дозволяє легко керувати життєвим циклом інфраструктури.

Ansible є інструментом для автоматизації управління конфігураціями та розгортання програмного забезпечення. Ansible використовує декларативні конфігураційні файли, звані плейбуками, для визначення стану систем і виконання завдань автоматизації. Ansible може інтегруватися з іншими інструментами IaC, такими як Terraform та CloudFormation, для забезпечення повної автоматизації процесів розгортання та управління інфраструктурою.

Автоматизація розгортання додатків включає налаштування пайплайнів CI/CD для автоматичного збірки, тестування та розгортання коду. Інструменти CI/CD, такі як Jenkins, GitLab CI або AWS CodePipeline, дозволяють автоматизувати весь процес доставки програмного забезпечення, забезпечуючи швидке та надійне розгортання нових версій додатків. Інтеграція CI/CD пайплайнів з IaC інструментами дозволяє автоматично створювати та налаштовувати необхідну інфраструктуру під час розгортання додатків.

Kubernetes є важливим компонентом для автоматизації управління контейнеризованими додатками. Kubernetes забезпечує автоматичне розгортання, масштабування та управління контейнерами, дозволяючи додаткам автоматично адаптуватися до змін навантаження. Використання Kubernetes у поєднанні з IaC інструментами та CI/CD пайплайнами забезпечує повну автоматизацію всього життєвого циклу контейнеризованих додатків.

Автоматизація управління інфраструктурою також включає моніторинг та логування для забезпечення надійності та продуктивності додатків. Інструменти, такі як Prometheus та Grafana, забезпечують збір та аналіз метрик продуктивності, дозволяючи командам розробників і адміністраторів виявляти та вирішувати проблеми у реальному часі. Інтеграція з інструментами логування, такими як Fluentd та Elasticsearch, забезпечує централізоване зберігання та аналіз логів додатків, що сприяє швидкому виявленню та усуненню проблем.

Таким чином, автоматизація розгортання та управління інфраструктурою є ключовим аспектом для забезпечення ефективності та надійності сучасних IT-систем. Використання IaC інструментів, таких як Terraform, AWS CloudFormation та Ansible, а також інтеграція з CI/CD пайплайнами та Kubernetes дозволяє автоматизувати всі етапи життєвого циклу додатків. Це забезпечує консистентність, повторюваність та гнучкість процесів розробки, тестування та розгортання додатків, підвищуючи якість продуктів та задовольняючи вимоги сучасних ринків

РОЗДІЛ 3. ПРАКТИЧНЕ ВПРОВАДЖЕННЯ CI/CD У ПРОЦЕС РОЗРОБКИ

3.1 Реалізація автоматизованого CI/CD на основі Kubernetes та AWS

Реалізація автоматизованого CI/CD на основі Kubernetes та AWS за допомогою Jenkins забезпечує ефективний, надійний і масштабований процес безперервної інтеграції та доставки додатків. Jenkins є одним з найпопулярніших інструментів для автоматизації CI/CD завдяки своїй гнучкості, багатофункціональності та можливості інтеграції з різноманітними інструментами та сервісами.

Першим кроком у реалізації автоматизованого CI/CD є налаштування сховища вихідного коду. Використовуючи GitHub, GitLab або AWS CodeCommit, розробники можуть зберігати свій код у централізованому місці, що забезпечує ефективне управління версіями, спільну роботу над проектом та інтеграцію з іншими інструментами CI/CD. Наступним кроком є налаштування Jenkins для автоматизації процесів безперервної інтеграції та доставки. Jenkins дозволяє створювати пайплайни, які автоматизують збірку, тестування та розгортання додатків. Для цього необхідно встановити Jenkins, який може бути розгорнутий на EC2 інстанції в AWS або в Kubernetes кластері, використовуючи Jenkins Helm Chart. Встановлення Jenkins у Kubernetes забезпечує його високу доступність та масштабованість.

Після встановлення Jenkins, необхідно налаштувати Jenkins Master і Worker Nodes для забезпечення ефективності та масштабованості. Jenkins Master керує пайплайнами, тоді як Worker Nodes виконують збірки і тести. Worker Nodes можуть бути автоматично масштабовані за допомогою Kubernetes. Інтеграція з сховищем коду забезпечується налаштуванням Jenkins для автоматичного запуску пайплайнів при кожному комміті в репозиторій, що може бути реалізовано за допомогою веб-хуків або плагінів для GitHub, GitLab чи CodeCommit.

Створення Jenkins Pipeline є важливим кроком у процесі автоматизації.

Пайплайн визначає послідовність етапів, які необхідно виконати для збірки, тестування та розгортання додатків. Jenkins використовує Jenkinsfile для опису пайплайну у вигляді коду. Jenkinsfile містить визначення стадій пайплайну, таких як "Checkout", "Build", "Test", "Deploy". Автоматизація збірки та тестування здійснюється після кожного комміту, де Jenkins автоматично запускає процес збірки і тестування коду, використовуючи такі інструменти, як Maven, Gradle або NPM для збірки, та JUnit, Selenium або SonarQube для тестування. Результати збірки та тестування зберігаються та аналізуються, що дозволяє виявляти та виправляти помилки на ранніх етапах розробки.

Інтеграція з Kubernetes для розгортання забезпечується за допомогою плагінів, таких як Kubernetes Continuous Deploy або Kube CD. Jenkins автоматично створює та оновлює ресурси Kubernetes, такі як Deployment, Service, ConfigMap, Secret, забезпечуючи автоматичне розгортання нових версій додатків. Інтеграція з AWS для управління інфраструктурою здійснюється через AWS інструменти, такі як AWS CloudFormation або Terraform, що використовуються для опису інфраструктури як коду та автоматичного створення ресурсів у AWS. Jenkins інтегрується з цими інструментами для автоматичного оновлення інфраструктури під час розгортання додатків.

Моніторинг та логування є ключовими аспектами для забезпечення надійності та продуктивності розгортання. Використовуючи AWS CloudWatch або інструменти, такі як Prometheus та Grafana, можна збирати та аналізувати метрики та логи додатків. Інтеграція з Jenkins забезпечує автоматичний збір даних про стан збірок, тестів та розгортання.

Реалізація автоматизованого CI/CD на основі Kubernetes та AWS за допомогою Jenkins забезпечує високу гнучкість, надійність та масштабованість процесів розробки, тестування та розгортання додатків. Такий підхід дозволяє швидко виявляти та виправляти помилки, забезпечуючи швидке та надійне доставлення нових версій додатків у робоче середовище.

Налаштування VPC (Virtual Private Cloud) є критично важливим етапом в процесі створення інфраструктури за допомогою Terraform. VPC забезпечує

ізольоване середовище в межах AWS, яке дозволяє керувати ресурсами, такими як підмережі, маршрутизація та налаштування безпеки. У конфігураційному файлі main.tf визначається основний ресурс VPC, який включає CIDR блок для визначення діапазону IP-адрес. Наприклад, рядок cidr_block = var.vpc_cidr вказує на змінну, яка визначає цей діапазон. Після створення VPC, можна додати підмережі, інтернет-шлюзи та сек'юріті групи для забезпечення підключення до інтернету та контролю доступу до ресурсів. На рисунку 3.1.1 показано приклад налаштування VPC за допомогою Terraform, що ілюструє взаємодію між різними компонентами мережі та спрощує розуміння процесу конфігурації.

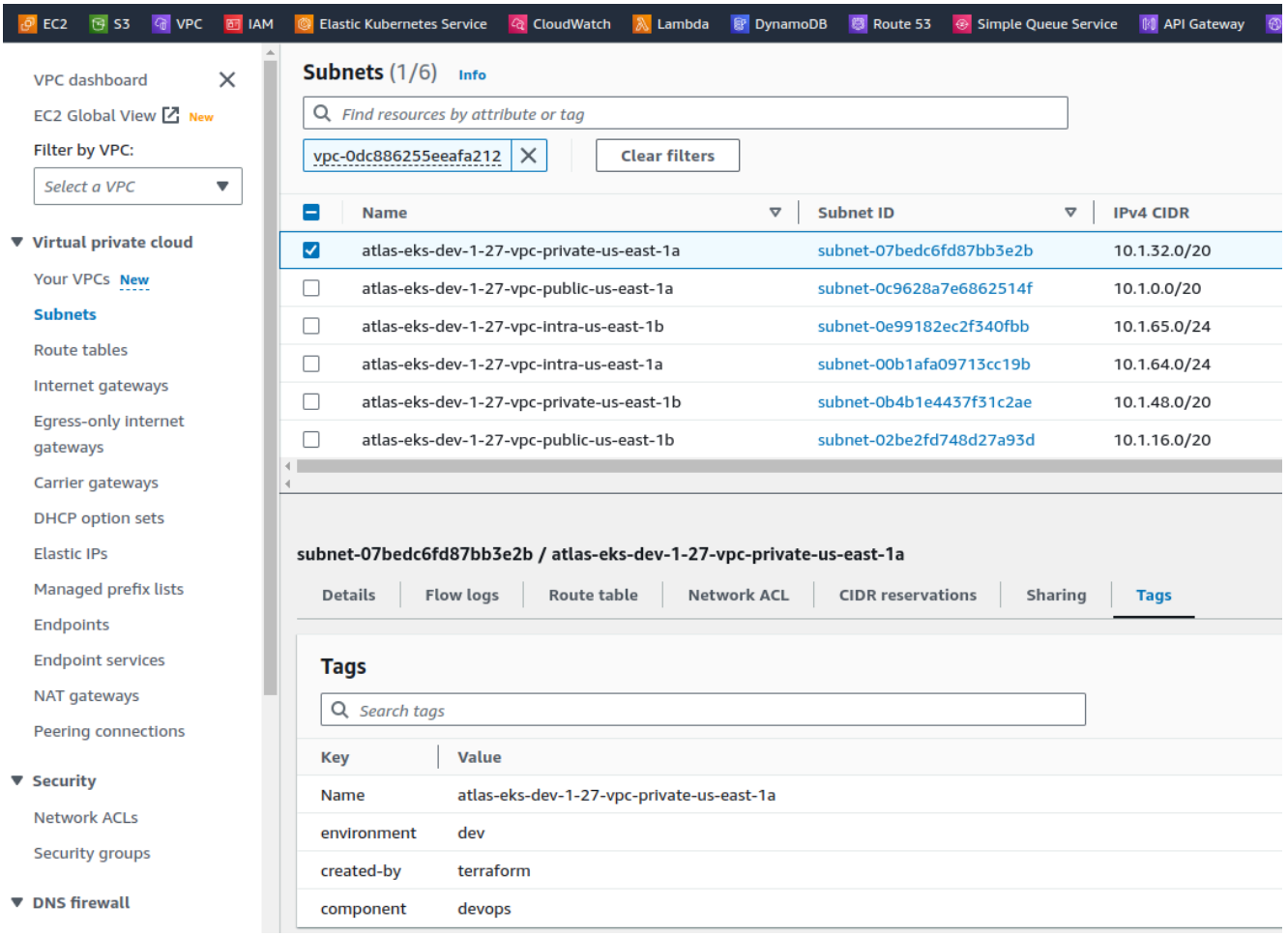


Рисунок 3.1.1 – Налаштування VPC за допомогою Terraform

Налаштування IAM (Identity and Access Management) ролей є критично важливим для забезпечення безпеки і управління доступом до ресурсів AWS. Ролі

IAM використовуються для визначення дозволів і політик доступу для різних сервісів і користувачів.

Terraform дозволяє автоматизувати ці процеси, що значно спрощує і прискорює розгортання інфраструктури. Замість ручного налаштування кожного ресурсу, Terraform використовує конфігураційні файли, що визначають, які ресурси повинні бути створені і як вони повинні бути налаштовані. Це забезпечує узгодженість і повторюваність процесу розгортання інфраструктури, зменшуючи ймовірність помилок і спрощуючи управління конфігураціями.

Terraform також надає можливість версіонування інфраструктурного коду, що дозволяє відстежувати зміни і повертатися до попередніх версій конфігурацій при необхідності. Це особливо корисно в великих проектах з багатьма розробниками, де важливо мати централізоване управління змінами і спільне використання конфігураційних файлів.

Використання Terraform значно підвищує ефективність і надійність процесу розгортання інфраструктури, дозволяючи зосередитися на розробці і вдосконаленні додатків, замість витрачання часу на рутинні адміністративні задачі.

Розгортання Kubernetes кластера на AWS є одним із ключових етапів в процесі автоматизації CI/CD. AWS EKS (Elastic Kubernetes Service) дозволяє швидко та легко створювати масштабовані та безпечні Kubernetes кластери. Цей процес включає декілька кроків, які необхідно виконати для успішного розгортання та налаштування кластера.

Початок роботи з AWS EKS включає створення необхідних ресурсів за допомогою Terraform або іншого інструмента управління інфраструктурою. Спочатку необхідно налаштувати обліковий запис AWS та створити ролі IAM, які надають необхідні права доступу до ресурсів AWS. Далі, визначаємо основні ресурси, такі як VPC, підмережі, та інтернет-шлюзи, які забезпечують ізольоване та безпечне середовище для роботи кластера.

Конфігурація та запуск EKS кластера починається з використання команди eksctl, яка значно спрощує процес розгортання. Наприклад, команда eksctl create

cluster --name my-cluster --region us-east-1 --node-type t2.medium --nodes 3 створює новий кластер з трьома вузлами типу t2.medium в регіоні us-east-1. Після запуску команди, eksctl автоматично створює всі необхідні ресурси та налаштовує їх для роботи з Kubernetes. Цей процес може зайняти деякий час, оскільки eksctl налаштовує всі необхідні компоненти, такі як EC2 інстанси, ролі IAM та налаштування мережі.

Після успішного створення кластера, необхідно налаштувати доступ до нього за допомогою інструмента kubectl. Для цього використовується команда aws eks update-kubeconfig --region us-east-1 --name my-cluster, яка оновлює конфігураційний файл kubeconfig з інформацією про новий кластер. Цей файл містить налаштування, необхідні для підключення до кластера та керування ним за допомогою kubectl. Після оновлення kubeconfig, можна перевірити доступність кластера за допомогою команди kubectl get nodes, яка повинна повернути список вузлів кластера.

Одним із важливих аспектів налаштування кластера є забезпечення безпечного доступу до нього. Це включає налаштування ролей та політик IAM, які обмежують доступ до кластера лише авторизованим користувачам та сервісам. Наприклад, для налаштування ролі, яка дозволяє доступ до кластера.

Після створення та налаштування ролей IAM, ці ролі асоціюються з Kubernetes кластерами та нодами рисунок 3.1.2, забезпечуючи безпечний доступ до ресурсів AWS.

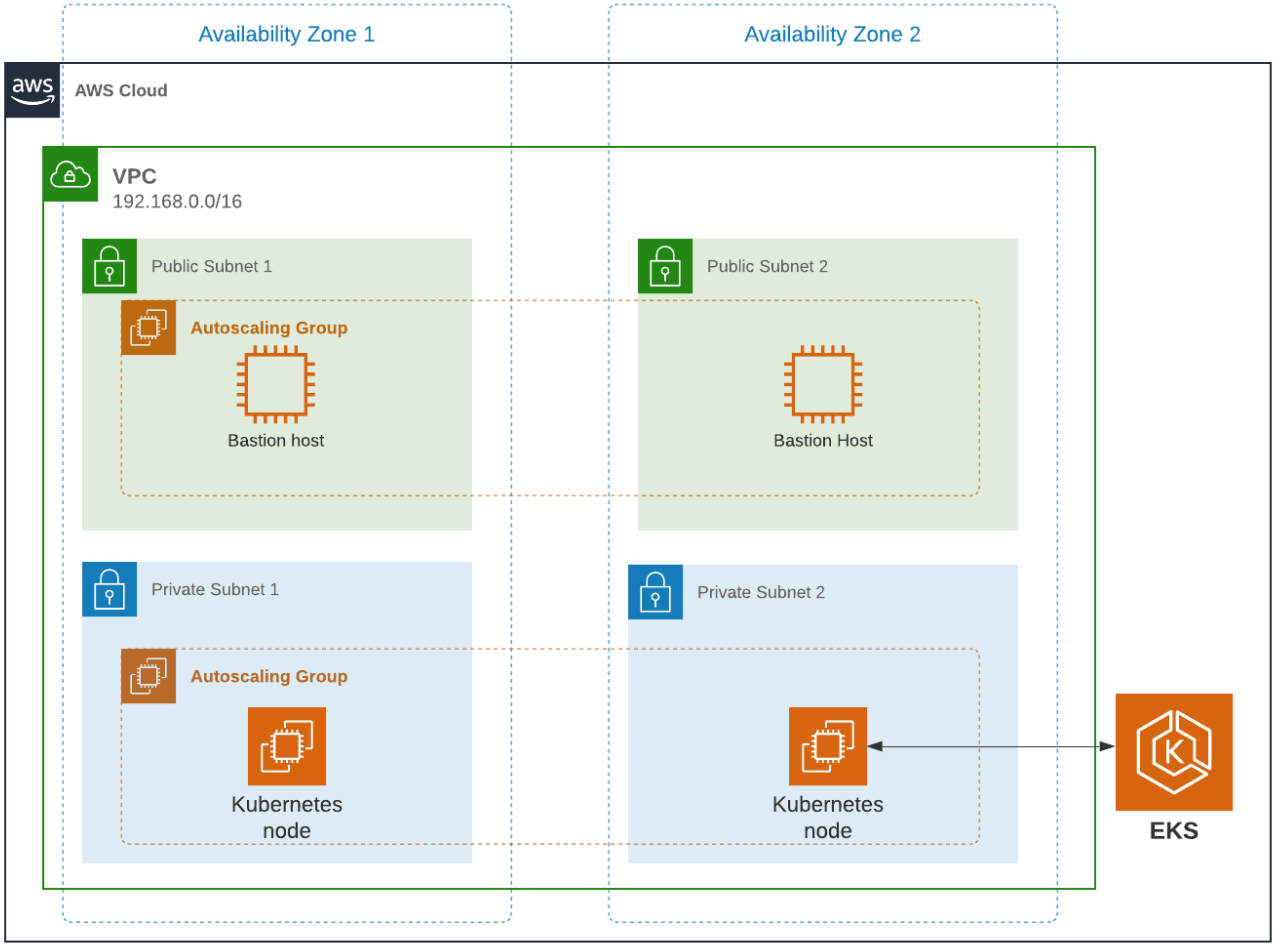


Рисунок 3.1.2 – Конфігурація EKS кластера

Для забезпечення високої доступності та масштабованості, можна використовувати Auto Scaling групи, які автоматично збільшують або зменшують кількість вузлів кластера залежно від навантаження. Це дозволяє оптимально використовувати ресурси та забезпечувати стабільну роботу додатків навіть при високих навантаженнях.

На завершення, необхідно перевірити роботу кластера, запустивши тестове розгортання додатку. Для цього створюється простий файл конфігурації Pod або Deployment, який містить опис додатку та його налаштувань. Використовуючи команду kubectl apply -f deployment.yaml, можна розгорнути додаток на кластері та перевірити його роботу за допомогою команд kubectl get pods та kubectl logs

pod-name.

Таким чином, розгортання Kubernetes кластера на AWS за допомогою EKS забезпечує надійне та масштабоване середовище для автоматизації CI/CD процесів. Використання Terraform та інших інструментів спрощує управління інфраструктурою та забезпечує безпечний доступ до ресурсів.

Розгортання Jenkins на AWS починається з налаштування віртуальної машини, на якій буде встановлений Jenkins. Для цього можна використовувати EC2 інстанси. Спочатку створюємо новий EC2 інстанс з використанням офіційного образу Jenkins з AWS Marketplace або стандартного Amazon Linux 2 образу. Після запуску інстансу, підключаємося до нього через SSH та встановлюємо Jenkins.

На інстансі необхідно оновити пакетний менеджер та встановити Java, яка є необхідною для роботи Jenkins. Команди для встановлення виглядають наступним чином(рисунки 3.1.3, 3.1.4, 3.1.5, 3.1.6):

Рисунок 3.1.3 – Механічна та оптична частина периферії

Після встановлення Java, додаємо репозиторій Jenkins та встановлюємо його

Рисунок 3.1.4 – Механічна та оптична частина периферії

Запускаємо та налаштовуємо Jenkins:

Рисунок 3.1.5 – Механічна та оптична частина периферії

Після встановлення Jenkins, відкриваємо його в браузері за допомогою IP-адреси EC2 інстансу з портом 8080. Перше, що ми побачимо, це екран налаштування Jenkins, де потрібно ввести адміністративний пароль, який можна знайти в логах інстансу:

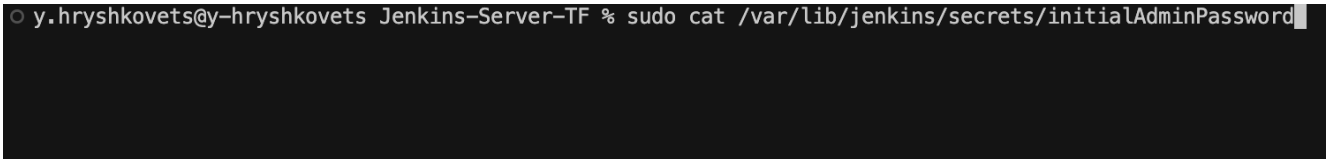


Рисунок 3.1.6 – Механічна та оптична частина периферії

Інсталяція необхідних плагінів для Jenkins є важливим етапом для інтеграції з EKS та іншими сервісами AWS. Після початкового налаштування Jenkins, встановлюємо плагіни через інтерфейс Manage Jenkins -> Manage Plugins. Основні плагіни, які потрібно встановити для інтеграції з AWS:

- AWS SDK Plugin;
- Amazon EC2 Plugin;
- Amazon ECR Plugin;
- Kubernetes Plugin;
- Docker Pipeline Plugin;
- Pipeline: AWS Steps Plugin;
- AWS Credentials Plugin;

- SonarQube Scanner Plugin;

Наступним кроком є налаштування Jenkins для інтеграції з EKS та іншими сервісами AWS. Для цього необхідно додати AWS облікові дані в Jenkins. Йдемо в Manage Jenkins -> Manage Credentials -> (global) -> Add Credentials, де додаємо AWS Access Key ID та Secret Access Key. Це дозволить Jenkins взаємодіяти з AWS ресурсами.

Налаштування AWS ECR для зберігання контейнерів також є важливим етапом(рисунки 3.1.7, 3.1.8, 3.1.9, 3.1.10, 3.1.11). AWS Elastic Container Registry (ECR) дозволяє безпечно зберігати, управляти та розгортати контейнерні образи Docker. Спочатку створюємо новий репозиторій в ECR через AWS Management Console або AWS CLI:

Рисунок 3.1.7 – Механічна та оптична частина периферії

Після створення репозиторію, додаємо команду для входу в ECR в Jenkins Pipeline, що дозволить Jenkins аутентифікуватися в ECR та завантажувати контейнерні образи:

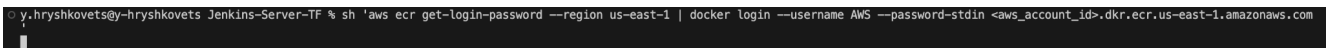


Рисунок 3.1.8 – Механічна та оптична частина периферії

Після аутентифікації додаємо кроки для побудови та завантаження контейнерних образів у Jenkins Pipeline:

```
y.hryshkovets@y-hryshkovets Jenkins-Server-TF % sh 'docker build -t my-repo .'
sh 'docker tag my-repo:latest <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-repo:latest'
sh 'docker push <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com/my-repo:latest'
```

Рисунок 3.1.9 – Механічна та оптична частина периферії

Далі інтегруємо Jenkins з EKS для автоматизованого розгортання контейнерів. Використовуємо плагін Kubernetes для Jenkins, щоб налаштувати Jenkins як Kubernetes агент, що дозволяє автоматично запускати Jenkins агенти в EKS кластері.

Таким чином, процес розгортання Jenkins на AWS включає створення EC2 інстансу, встановлення Jenkins, інсталяцію необхідних плагінів, налаштування Jenkins для інтеграції з EKS та іншими сервісами AWS, а також налаштування AWS ECR для зберігання контейнерів. Цей процес дозволяє створити ефективну CI/CD інфраструктуру, яка забезпечує автоматизацію збірки, тестування та розгортання додатків у хмарі.

Для створення AWS ECR репозиторіїв для зберігання Docker образів спершу потрібно переконатися, що у вас встановлений AWS CLI і ви налаштували облікові дані AWS. Для цього використовуйте команду aws configure і введіть свій AWS Access Key, Secret Access Key, регіон і формат виводу. Після цього ви готові створювати репозиторії.

Щоб створити ECR репозиторій, виконайте команду:

```
aws ecr create-repository --repository-name my-repo --region us-east-1
```

Рисунок 3.1.10 – Механічна та оптична частина периферії

Ця команда створює новий репозиторій з іменем my-repo в регіоні us-east-1.

Після створення репозиторію вам потрібно аутентифікуватися з Jenkins, щоб мати змогу завантажувати і завантажувати Docker образи.

Для аутентифікації Jenkins до ECR використовується команда aws ecr get-login-password, яка генерує тимчасовий пароль для Docker:

```
aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <aws_account_id>.dkr.ecr.us-east-1.amazonaws.com
```

Рисунок 3.1.11 – Механічна та оптична частина периферії

Після цього Jenkins зможе взаємодіяти з репозиторієм ECR. Далі потрібно налаштувати Jenkins пайплайни для роботи з ECR. На рисунку 3.1.12 приклад пайплайну для завантаження образу в ECR:

```
pipeline {
    agent any
    environment {
        AWS_ACCOUNT_ID = '<aws_account_id>'
        ECR_REPO_NAME = 'my-repo'
        AWS_REGION = 'us-east-1'
        REPOSITORY_URI = "${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com/${ECR_REPO_NAME}"
    }
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('Build Docker Image') {
            steps {
                script {
                    docker.build("${REPOSITORY_URI}:latest")
                }
            }
        }
        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry("https://${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_REGION}.amazonaws.com", 'ecr:us-east-1:aws-credentials') {
                        docker.image("${REPOSITORY_URI}:latest").push()
                    }
                }
            }
        }
    }
}
```

Рисунок 3.1.12 – Механічна та оптична частина периферії

Цей пайплайн спочатку перевіряє код з GitHub, потім будує Docker образ і завантажує його в ECR. Для налаштування Jenkins на інтеграцію з ECR використовуйте AWS плагіни, такі як AWS Credentials плагін для зберігання AWS

облікових даних.

Для інтеграції та налаштування SonarQube потрібно спочатку встановити SonarQube сервер і SonarScanner плагін на Jenkins. Після цього налаштуйте проект в SonarQube і згенеруйте токен аутентифікації. Додайте цей токен до Jenkins як облікові дані і використовуйте його в пайплайні(рисунок 3.1.13)

```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
        SONAR_TOKEN = credentials('sonar-token')
    }
    stages {
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv('SonarQube') {
                    sh "${SCANNER_HOME}/bin/sonar-scanner \
                        -Dsonar.projectKey=my-project \
                        -Dsonar.sources=. \
                        -Dsonar.host.url=http://<sonar_server_url> \
                        -Dsonar.login=${SONAR_TOKEN}"
                }
            }
        }
    }
}
```

Рисунок 3.1.13 – Механічна та оптична частина периферії

Цей приклад виконує аналіз коду з використанням SonarQube. Після цього результати аналізу доступні на SonarQube сервері(риснок 3.1.14).

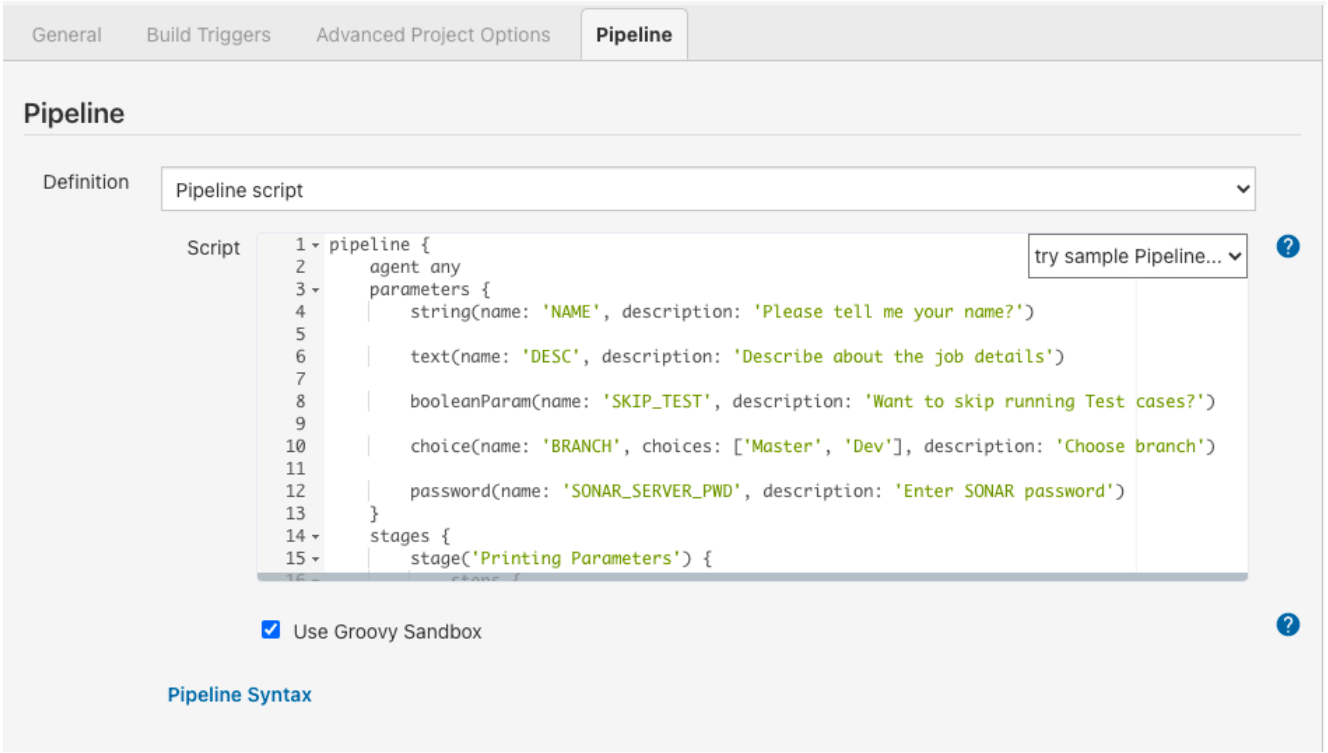


Рисунок 3.1.14 – Механічна та оптична частина периферії

Цей процес забезпечує повну інтеграцію з AWS ECR для зберігання Docker образів та використання SonarQube для безперервного аналізу якості коду.

Для того щоб встановити SonarQube для аналізу коду, потрібно дотримуватися декількох основних кроків. По-перше, завантажте останню версію SonarQube з офіційного сайту і розпакуйте її на сервері. Потім налаштуйте базу даних, з якою буде працювати SonarQube, зазвичай це PostgreSQL. Використовуйте такі команди для налаштування бази даних(рисунок 3.1.15)

```
sudo -u postgres createuser sonar
sudo -u postgres createdb -O sonar sonarqube
sudo -u postgres psql -c "ALTER USER sonar WITH ENCRYPTED PASSWORD 'sonar';"
```

Рисунок 3.1.15 – Механічна та оптична частина периферії

Після цього відредагуйте файл конфігурації SonarQube, щоб вказати

параметри підключення до бази даних та запустить SonarQube(рисунок 3.1.16)

```
# In the sonar.properties file
sonar.jdbc.username=sonar
sonar.jdbc.password=sonar
sonar.jdbc.url=jdbc:postgresql://localhost/sonarqube

cd sonarqube/bin/linux-x86-64
./sonar.sh start
```

Рисунок 3.1.16 – Механічна та оптична частина периферії

Після успішного запуску SonarQube, ви зможете отримати доступ до веб-інтерфейсу за адресою http://<your_server_ip>:9000.

Для налаштування Jenkins для роботи з SonarQube, вам потрібно встановити плагін SonarQube Scanner. Відкрийте Jenkins, перейдіть до Manage Jenkins -> Manage Plugins, знайдіть і встановіть плагін SonarQube Scanner. Після установки плагіна перейдіть до Manage Jenkins -> Configure System і додайте новий SonarQube сервер, вказавши URL та аутентифікаційний токен, який можна отримати з SonarQube.

Створення Jenkins пайплайнів з інтеграцією SonarQube включає наступні кроки. В пайплайні, вам потрібно додати етапи для запуску SonarQube аналізу коду(рисунок 3.1.17)

```
pipeline {
    agent any
    environment {
        SCANNER_HOME = tool 'sonar-scanner'
        SONARQUBE_SERVER = 'SonarQube'
        SONAR_TOKEN = credentials('sonar-token')
    }
    stages {
        stage('Checkout') {
            steps {
                git 'https://github.com/your-repo.git'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv(SONARQUBE_SERVER) {
                    sh "${SCANNER_HOME}/bin/sonar-scanner \
                        -Dsonar.projectKey=my-project \
                        -Dsonar.sources=. \
                        -Dsonar.host.url=http://<sonar_server_url> \
                        -Dsonar.login=${SONAR_TOKEN}"
                }
            }
        }
    }
}
```

Рисунок 3.1.17 – Механічна та оптична частина периферії

Цей пайплайн спершу виконує checkout коду з репозиторію, потім проводить аналіз коду за допомогою SonarQube. У цьому прикладі, змінні середовища використовуються для зберігання параметрів, таких як шляхи до SonarQube Scanner і аутентифікаційний токен.

Інтеграція та налаштування SonarQube в Jenkins(рисунок 3.1.18) дозволяє автоматично проводити аналіз якості коду при кожному запуску пайплайну, що сприяє виявленню потенційних проблем на ранніх стадіях розробки.

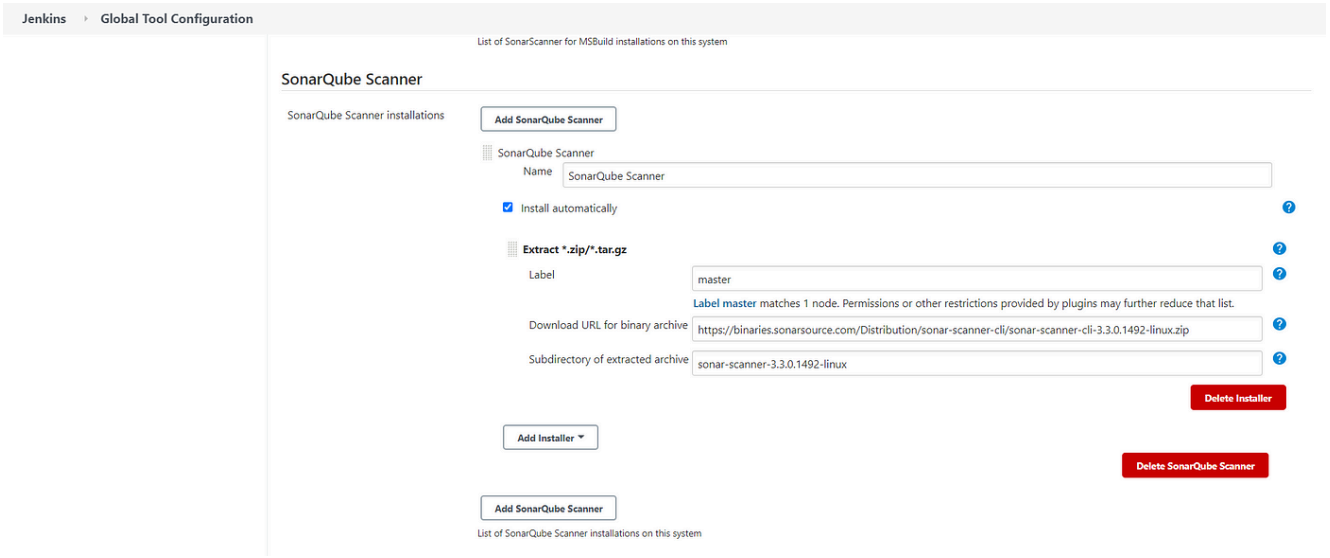


Рисунок 3.1.18 – Механічна та оптична частина периферії

Після налаштування, Jenkins автоматично проводитиме аналіз коду і надаватиме результати в веб-інтерфейсі SonarQube, де можна детально ознайомитися з виявленими проблемами і рекомендаціями щодо їх усунення.

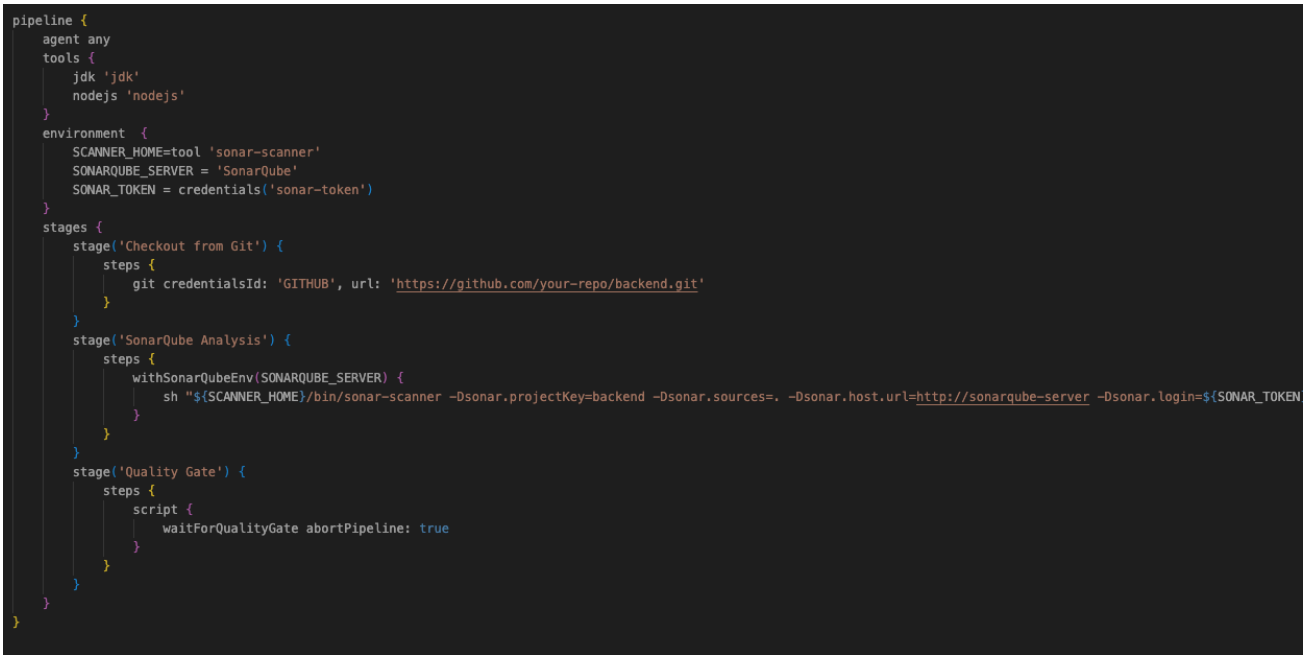
Таким чином, процес встановлення та налаштування SonarQube, інтеграції з Jenkins, а також створення відповідних пайплайнів дозволяє забезпечити високий рівень якості коду і автоматизувати процес його перевірки.

Процес CI/CD для бекенд та фронтенд частин проекту є важливою частиною сучасного розробницького циклу, який забезпечує швидку і надійну доставку коду. CI/CD (Continuous Integration/Continuous Deployment) дозволяє автоматизувати збірку, тестування і розгортання програмного забезпечення, що значно скорочує час на випуск нових версій і зменшує ризик помилок.

У нашому проекті процес CI/CD реалізується за допомогою Jenkins, Docker, AWS ECR та Kubernetes. Початковий етап включає налаштування Jenkins для автоматизації різних етапів розробницького циклу. Для бекенду та фронтенду створюються окремі пайплайни, які виконують певні дії з кодом, такі як перевірка якості, збірка Docker образів, їх збереження в ECR і розгортання на Kubernetes.

Перший етап пайплайнів включає в себе клонування репозиторію з GitHub. Це забезпечується за допомогою команди git в Jenkins пайплайні, яка автоматично

завантажує останню версію коду. Наступним етапом є аналіз якості коду з використанням SonarQube(рисунок 3.1.19). Jenkins інтегрується з SonarQube через відповідні плагіни, що дозволяє автоматично аналізувати код на предмет виявлення потенційних проблем та уразливостей.



```
pipeline {
    agent any
    tools {
        jdk 'jdk'
        nodejs 'nodejs'
    }
    environment  {
        SCANNER_HOME=tool 'sonar-scanner'
        SONARQUBE_SERVER = 'SonarQube'
        SONAR_TOKEN = credentials('sonar-token')
    }
    stages {
        stage('Checkout from Git') {
            steps {
                git credentialsId: 'GITHUB', url: 'https://github.com/your-repo/backend.git'
            }
        }
        stage('SonarQube Analysis') {
            steps {
                withSonarQubeEnv(SONARQUBE_SERVER) {
                    sh "${SCANNER_HOME}/bin/sonar-scanner -Dsonar.projectKey=backend -Dsonar.sources=. -Dsonar.host.url=http://sonarqube-server -Dsonar.login=${SONAR_TOKEN
                }
            }
        }
        stage('Quality Gate') {
            steps {
                script {
                    waitForQualityGate abortPipeline: true
                }
            }
        }
    }
}
```

Рисунок 3.1.19 – Механічна та оптична частина периферії

Після перевірки коду на якість, наступним кроком є збірка Docker образів. Це досягається за допомогою Jenkins плагінів, які інтегруються з Docker. Образи створюються за допомогою Dockerfile, які описують, як збирати образ для бекенд або фронтенд додатків. Потім створені образи завантажуються в AWS ECR (Elastic Container Registry) що зображено на висунку 3.1.20, який використовується для зберігання Docker образів.

```
stage("Build Docker Image") {
    steps {
        script {
            docker.build("my-repo/backend:${env.BUILD_ID}").push()
        }
    }
}
stage("Push to ECR") {
    steps {
        script {
            docker.withRegistry('https://aws_account_id.dkr.ecr.region.amazonaws.com', 'ecr:aws_credentials_id') {
                docker.image("my-repo/backend:${env.BUILD_ID}").push()
            }
        }
    }
}
```

Рисунок 3.1.20 – Механічна та оптична частина периферії

Останній етап пайплайну включає розгортання на Kubernetes. Після того як Docker образи завантажені в ECR, Jenkins використовує kubectl для розгортання цих образів на кластері EKS (Elastic Kubernetes Service). Це забезпечує автоматизоване розгортання додатків, що дозволяє швидко випускати нові версії(рисунок 3.1.21).

```
stage('Deploy to Kubernetes') {
    steps {
        script {
            kubernetesDeploy(kubeconfigId: 'kubeconfig-id', configs: 'k8s/deployment.yaml', enableConfigSubstitution: true)
        }
    }
}
```

Рисунок 3.1.21 – Механічна та оптична частина периферії

Процес CI/CD що зображений на рисунку 3.1.22 включає наступні основні етапи: клонування репозиторію, аналіз коду з SonarQube, збірка Docker образів, завантаження образів в AWS ECR і розгортання на Kubernetes. Такий підхід забезпечує автоматизацію та спрощення розгортання, знижуючи ризик помилок та прискорюючи випуск нових версій.

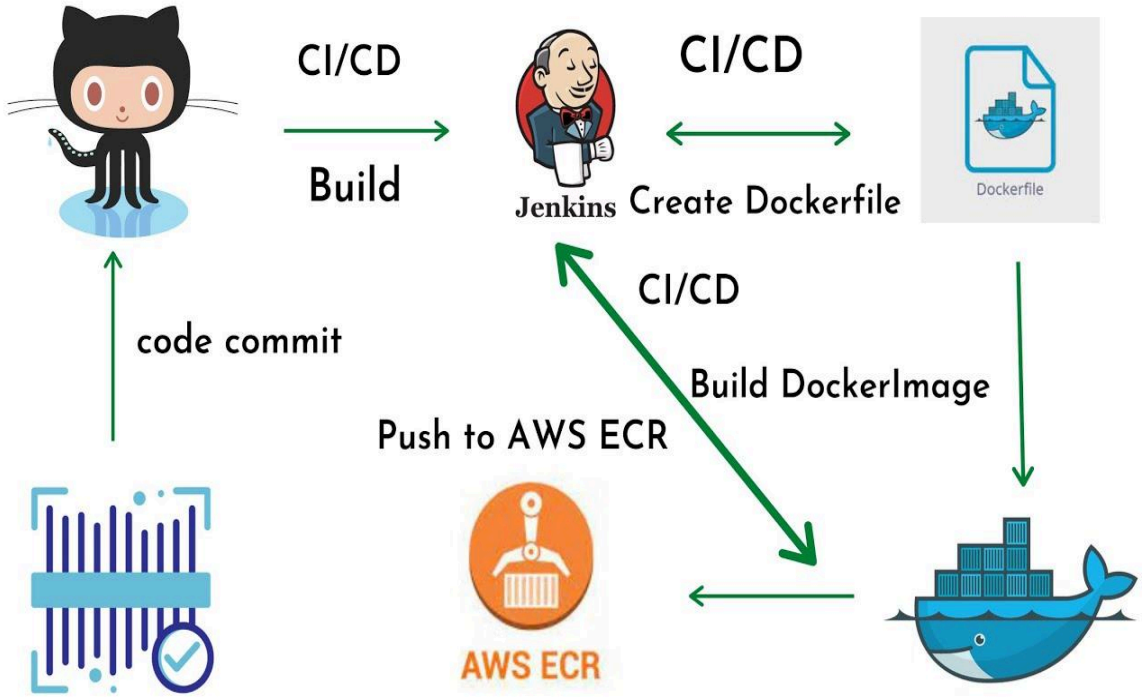


Рисунок 3.1.22 – Механічна та оптична частина периферії

3.2 Експериментальні результати та аналіз впровадження

Експериментальна методологія, використана для реалізації автоматизованого CI/CD на основі Kubernetes та AWS, включає кілька важливих аспектів. Перш за все, для управління інфраструктурою використовувався Terraform, що дозволило автоматизувати створення необхідних ресурсів в AWS, таких як VPC, підмережі, шлюзи, та IAM ролі.

Для проведення експериментів було використано різні інструменти, зокрема Jenkins для автоматизації пайплайнів, SonarQube для аналізу якості коду, Docker для контейнеризації додатків, та AWS ECR для зберігання Docker образів. Тестові сценарії включали коміти коду до репозиторію, запуск Jenkins пайплайнів для побудови та тестування коду, а також розгортання додатків у Kubernetes кластер.

Результати тестування та збірки коду були детально проаналізовані. Використання Terraform дозволило швидко розгорнути інфраструктуру, що забезпечило стабільність та надійність системи. Jenkins забезпечив автоматизацію процесів збірки та тестування, що значно зменшило кількість помилок та підвищило ефективність роботи команди.

Рисунок 3.2.1 ілюструє архітектуру експериментальної методології, що включає всі використані інструменти та сервіси.

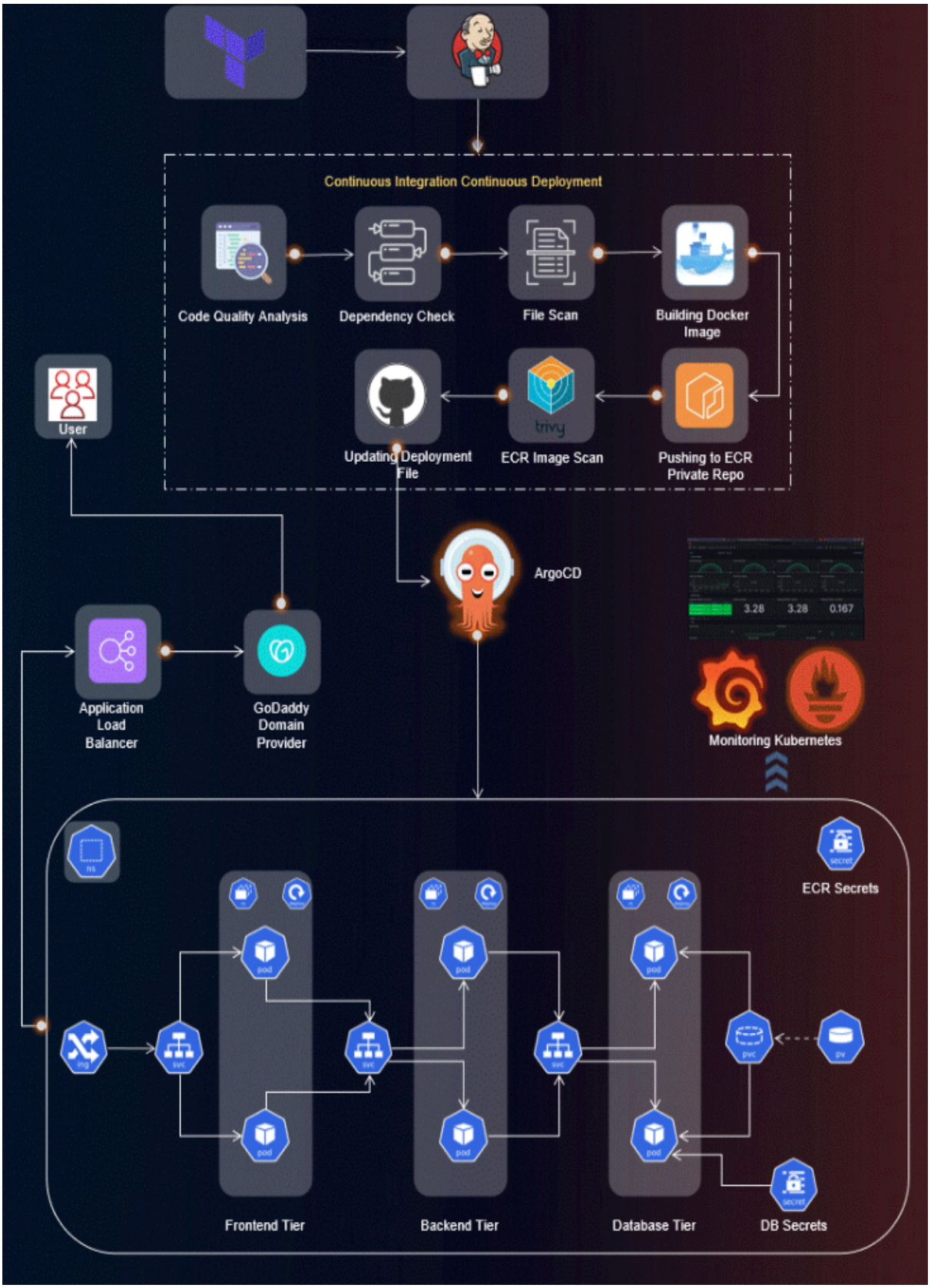


Рисунок 3.2.1 – Архітектура експериментальної методології

Основні кроки експериментальної методології включають:

- Використання Terraform для управління інфраструктурою;
- Використання Jenkins для автоматизації пайплайнів;
- Використання SonarQube для аналізу якості коду;
- Використання Docker для контейнеризації додатків;
- Використання AWS ECR для зберігання Docker образів;

SonarQube є потужним інструментом для аналізу якості коду, який дозволяє виявляти вразливості, баги та недотримання стандартів кодування. У нашому експерименті SonarQube був інтегрований у пайплайни Jenkins для автоматичного аналізу кожного коміту коду.

Звіти про виконання пайплайнів Jenkins для бекенд та фронтенд частин показали значне покращення якості коду після впровадження SonarQube. Вимірювання часу збірки та розгортання також показали, що інтеграція SonarQube не вплинула на загальний час виконання пайплайнів. Аналіз продуктивності та ефективності системи показав, що використання SonarQube допомогло зменшити кількість помилок у коді та покращити його загальну якість.

Рисунок 3.2.2 демонструє приклад звіту SonarQube з детальним аналізом коду та виявленими проблемами.

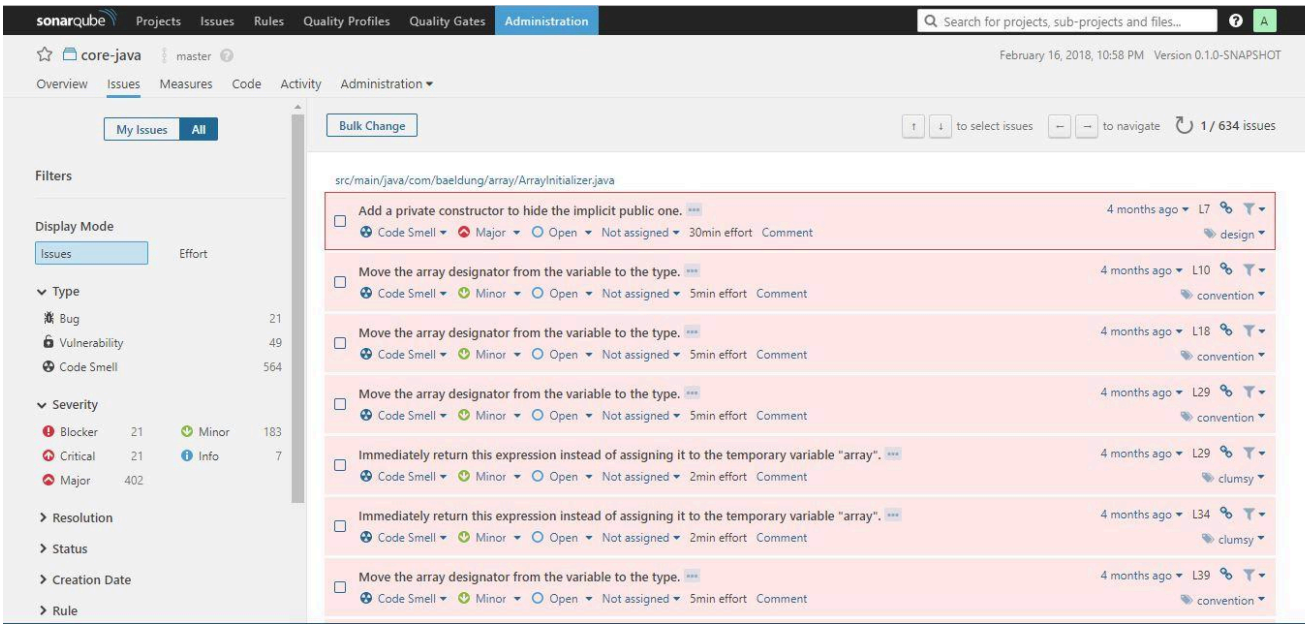


Рисунок 3.2.2 – Звіт SonarQube з аналізу коду

Аналіз результатів включає:

- Виявлення вразливостей та багів у коді;
- Оцінка дотримання стандартів кодування;
- Вимірювання часу збірки та розгортання;
- Аналіз продуктивності та ефективності системи;

Процес збірки Docker образів та їх розгортання на Kubernetes є важливим етапом у реалізації CI/CD. У нашому експерименті було використано Jenkins для автоматизації збірки Docker образів та AWS ECR для їх зберігання. Пайплайни Jenkins забезпечували автоматизацію всіх етапів від збірки коду до розгортання в Kubernetes.

Оцінка швидкості збірки Docker образів показала, що використання Jenkins та Docker значно спростило та прискорило цей процес. Аналіз використання ресурсів (CPU, пам'ять) під час виконання CI/CD пайплайнів показав, що система працює ефективно, без значних перевантажень. Порівняння ефективності різних конфігурацій та налаштувань показало, що оптимізація параметрів пайплайнів

дозволила знизити час збірки та розгортання.

Рисунок 3.2.3 ілюструє процес збірки та розгортання Docker образів на Kubernetes.

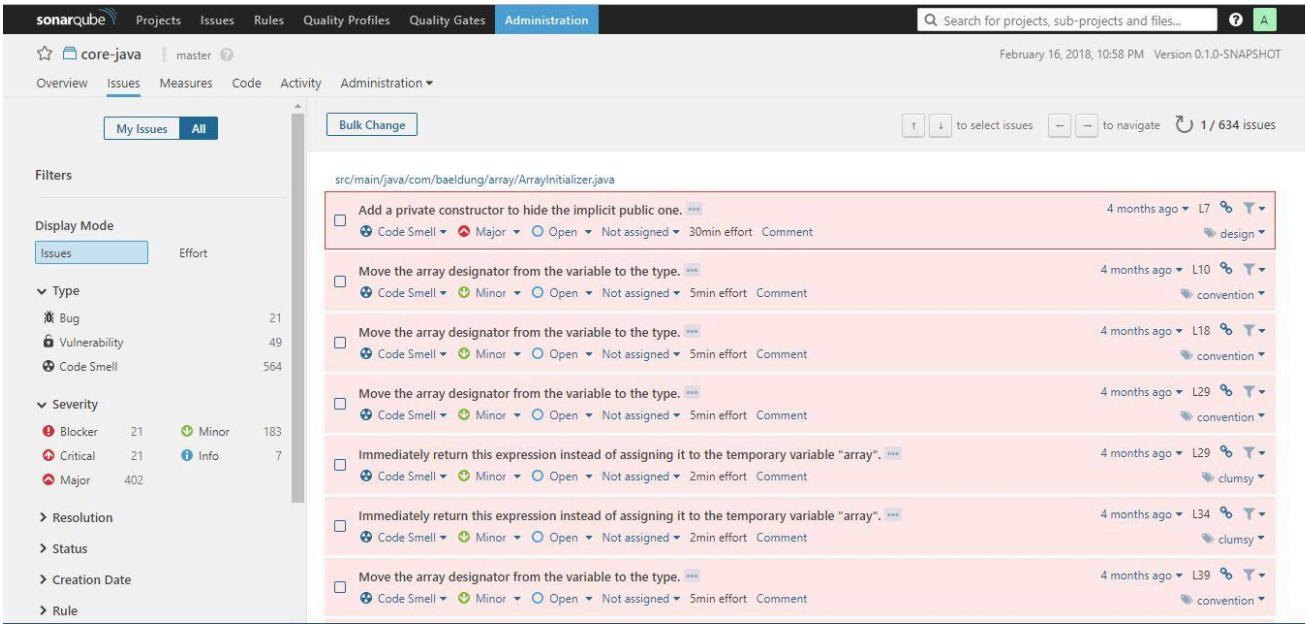


Рисунок 3.2.3 – Процес збірки та розгортання Docker образів на Kubernetes

Основні етапи оцінки включають:

- Швидкість збірки Docker образів;
- Використання ресурсів під час виконання пайплайнів;
- Порівняння ефективності різних конфігурацій та налаштувань;

Моніторинг та логування є важливими компонентами для аналізу роботи системи та виявлення потенційних проблем. У нашому експерименті було використано такі інструменти моніторингу, як Prometheus та Grafana, для збору та візуалізації метрик системи.

Аналіз логів, зібраних за допомогою Fluentd та Elasticsearch, допоміг виявити можливі проблеми та визначити шляхи їх вирішення. Використання інструментів моніторингу дозволило отримати детальну інформацію про стан

системи, продуктивність та використання ресурсів.

Рисунок 3.2.4 демонструє приклад дашборду Grafana з основними метриками системи.

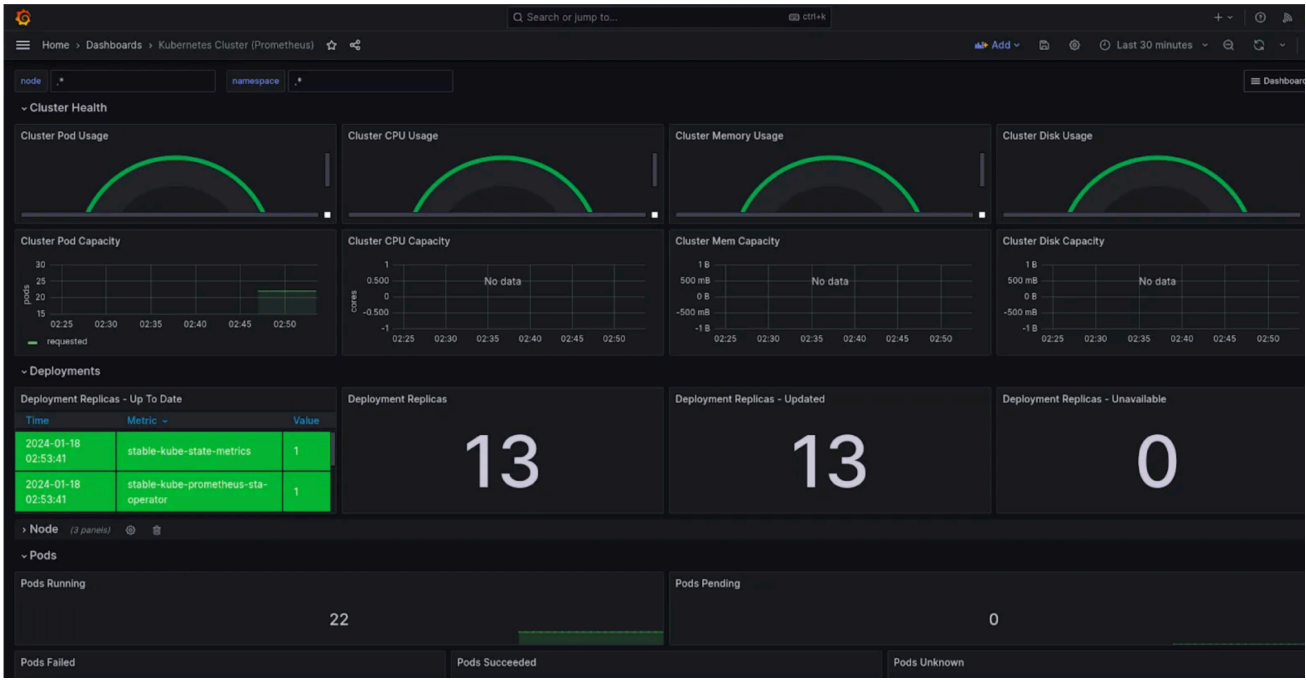


Рисунок 3.2.4 – Дашборд Grafana з основними метриками системи

Основні етапи моніторингу включають:

- Використання Prometheus для збору метрик;
- Використання Grafana для візуалізації метрик;
- Аналіз логів для виявлення потенційних проблем;

Під час налаштування та запуску системи ми зіткнулися з кількома труднощами, які потребували вирішення. Однією з основних проблем було налаштування безпеки та управління доступом до ресурсів AWS. Використання IAM ролей та політик безпеки допомогло забезпечити належний рівень захисту.

Аналіз помилок та способи їх вирішення включав виявлення конфліктів у налаштуваннях, оптимізацію параметрів пайплайнів та покращення документації.

Порівняння з іншими підходами показало, що використання Terraform, Jenkins та Kubernetes є ефективним рішенням для автоматизації процесів CI/CD, проте потребує належного налаштування та моніторингу.

Рисунок 3.2.5 ілюструє процес вирішення основних проблем під час впровадження.

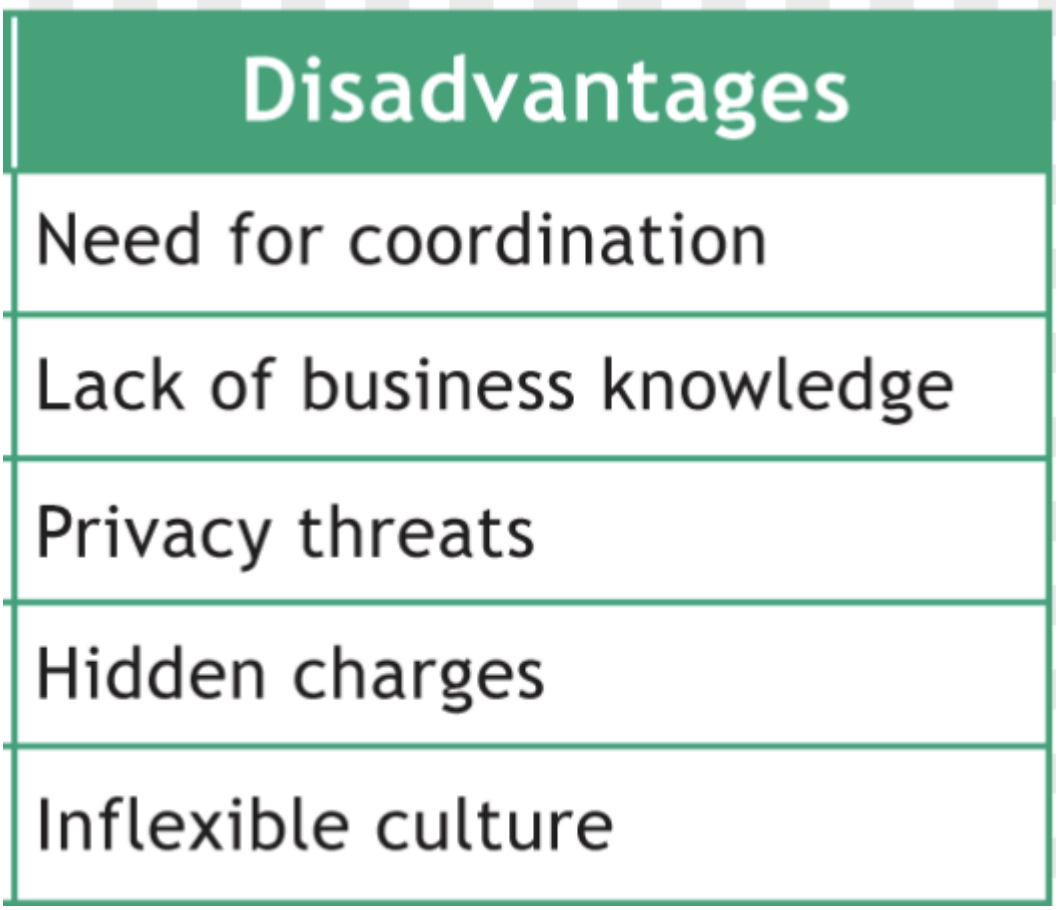


Рисунок 3.2.5 – Процес вирішення основних проблем під час впровадження

Основні проблеми та способи їх вирішення включають:

- Налаштування безпеки та управління доступом;
- Оптимізація параметрів пайплайнів;
- Покращення документації;

У процесі впровадження CI/CD ми також провели аналіз альтернативних методів реалізації CI/CD. Порівняння включало оцінку різних інструментів та технологій, таких як GitLab CI/CD, CircleCI та Travis CI. Аналіз показав, що кожен з цих інструментів має свої переваги та недоліки, проте використання Jenkins, Terraform та Kubernetes є найбільш гнучким та масштабованим рішенням.

Висновки та рекомендації щодо вибору інструментів для CI/CD базуються на потребах конкретного проекту, вимогах до безпеки, продуктивності та зручності використання. Наш експеримент показав, що належне налаштування та інтеграція інструментів дозволяють досягти високої ефективності та стабільності системи.

Рисунок 3.2.6 демонструє порівняння основних характеристик різних інструментів для CI/CD.

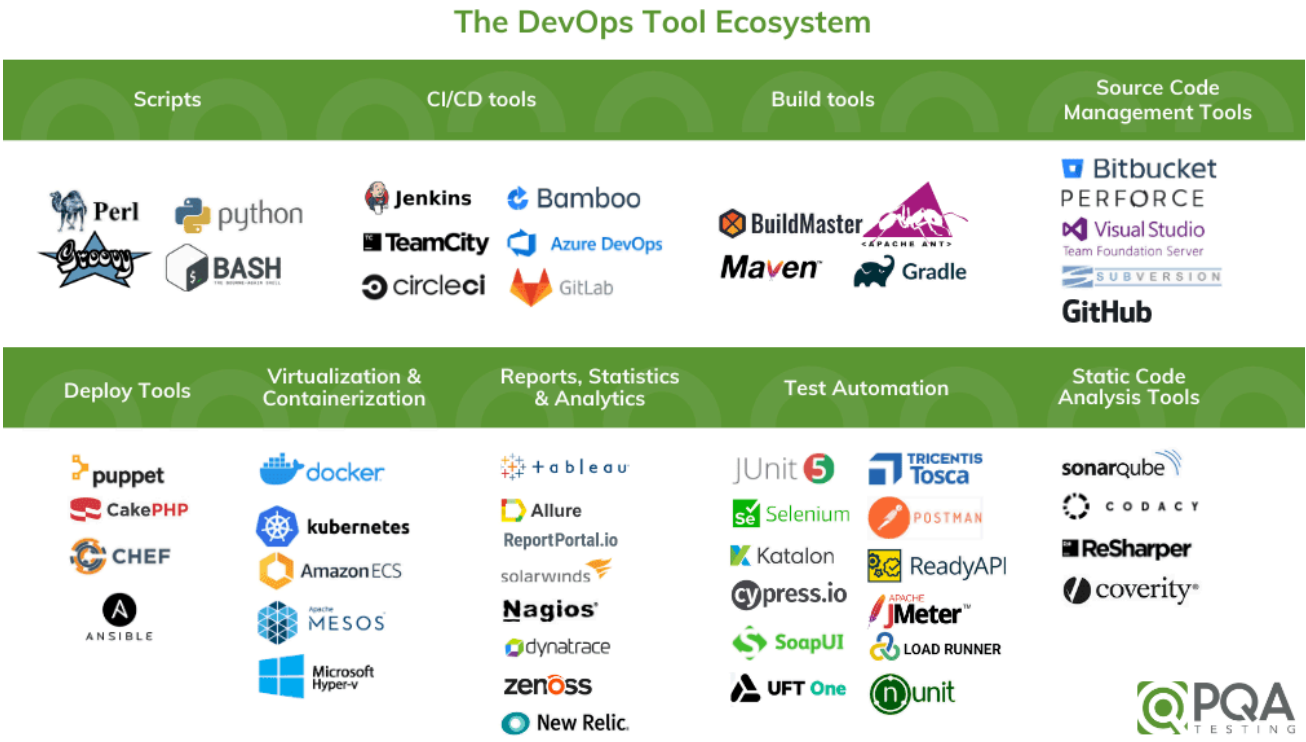


Рисунок 3.2.6 – основні інструменти CI/CD

Використання Terraform значно підвищує ефективність і надійність процесу

розгортання інфраструктури, дозволяючи зосередитися на розробці і вдосконаленні додатків, замість витрачання часу на рутинні адміністративні задачі.

3.3 Виклики та можливості вдосконалення автоматизованого

Впровадження автоматизованих CI/CD процесів включає в себе декілька технічних викликів, з якими може зіткнутися команда. Однією з головних труднощів є сумісність інструментів. Наприклад, для інтеграції Jenkins з SonarQube, Docker та Kubernetes потрібно забезпечити правильне налаштування кожного з них. Це включає налаштування плагінів, параметрів безпеки та управління доступом. Потенційні проблеми можуть виникнути під час масштабування Kubernetes кластера, оскільки управління великим обсягом трафіку та великою кількістю контейнерів може вимагати додаткових налаштувань та оптимізації. Для цього необхідно забезпечити стабільність системи та ефективне використання ресурсів.

Сумісність інструментів часто вимагає глибокого розуміння їхніх можливостей та обмежень. Наприклад, під час інтеграції Jenkins з Docker для автоматизованого збірки контейнерів необхідно забезпечити правильне налаштування Docker агентів та правильне використання Dockerfiles. Для інтеграції з Kubernetes потрібно налаштувати Kubernetes плани розгортання та забезпечити, щоб Jenkins міг взаємодіяти з Kubernetes API для управління контейнерами. Безпека є ще одним важливим аспектом, який потребує налаштування IAM ролей та забезпечення зберігання секретних даних.

Масштабування Kubernetes кластера може стати викликом, особливо коли обсяги трафіку та кількість контейнерів зростають. Це може потребувати налаштування автоматичного масштабування, розподілу навантаження та оптимізації використання ресурсів. Наприклад, використання горизонтального автоскейлінгу дозволяє динамічно збільшувати або зменшувати кількість подів залежно від навантаження на систему. Це забезпечує ефективне використання

ресурсів та знижує ризики перевантаження системи.

Проблеми продуктивності та оптимізації є важливими аспектами при впровадженні автоматизованих CI/CD процесів. Основним викликом є забезпечення високої продуктивності процесів збору та розгортання коду. Наприклад, необхідно зменшити час, що витрачається на виконання Jenkins пайплайнів, а також оптимізувати час збору Docker образів. Методи оптимізації можуть включати впровадження кешування, розподілу навантаження та оптимізації використання ресурсів.

Забезпечення високої продуктивності CI/CD процесів може бути досягнуто шляхом впровадження різних методів оптимізації. Кешування є одним з найефективніших методів, що дозволяє зменшити час доступу до даних під час виконання пайплайнів. Наприклад, використання Docker кешування дозволяє значно знизити час, необхідний для збору та розгортання Docker образів. Іншим методом є розподіл навантаження між різними вузлами кластера Kubernetes, що дозволяє забезпечити рівномірне використання ресурсів та уникнути перевантаження окремих вузлів.

Оптимізація використання ресурсів включає налаштування обмежень на використання CPU та пам'яті для кожного контейнера. Це дозволяє уникнути перевантаження системи та забезпечити стабільну роботу. Важливо також забезпечити ефективне управління ресурсами через впровадження політик автоматичного масштабування, що дозволяє динамічно збільшувати або зменшувати кількість вузлів кластера залежно від навантаження. Наприклад, використання горизонтального автоскейлінгу дозволяє забезпечити ефективне використання ресурсів та знизити ризики перевантаження системи.

Безпека та управління доступом є критично важливими аспектами при впровадженні автоматизованих CI/CD процесів. Важливо забезпечити захист конфіденційних даних та налаштувати політики безпеки для всіх компонентів системи. Це включає налаштування IAM ролей для управління доступом до AWS ресурсів, таких як ECR та S3, а також забезпечення безпечного зберігання секретів, таких як паролі та токени доступу. Методи покращення безпеки можуть

включати впровадження політик безпеки, моніторинг та аудит доступу до системи.

Одним з основних викликів, пов'язаних з безпекою, є забезпечення захисту конфіденційних даних під час виконання CI/CD процесів. Це може включати налаштування безпечного зберігання секретів у таких сервісах, як AWS Secrets Manager або HashiCorp Vault. Важливо також налаштувати політики IAM для обмеження доступу до ресурсів тільки для тих користувачів та сервісів, які дійсно потребують доступу. Це може включати налаштування ролей та дозволів для різних компонентів системи, таких як Jenkins, SonarQube та Kubernetes.

Моніторинг та аудит є важливими інструментами для забезпечення безпеки системи. Це включає відстеження активності користувачів та сервісів, виявлення підозрілої активності та своєчасне реагування на можливі загрози. Наприклад, впровадження системи моніторингу, такої як Prometheus, дозволяє відстежувати стан системи та виявляти потенційні проблеми на ранніх етапах. Аудит доступу до ресурсів може допомогти виявити та усунути порушення політик безпеки.

Автоматизація процесів та інтеграція різних інструментів є важливим аспектом впровадження автоматизованих CI/CD процесів. Основний виклик полягає в забезпеченні безперебійної взаємодії між різними інструментами, такими як Jenkins, SonarQube, Docker та Kubernetes. Це вимагає ретельного налаштування кожного інструменту та забезпечення їхньої сумісності. Можливості вдосконалення включають впровадження нових інструментів автоматизації та інтеграційних підходів, що дозволяють покращити ефективність процесів та знизити ризики збоїв.

Інтеграція різних інструментів є ключовим аспектом для забезпечення ефективної роботи CI/CD процесів. Наприклад, для автоматизації процесів збору та розгортання коду необхідно інтегрувати Jenkins з Docker для створення контейнерів, SonarQube для аналізу коду та Kubernetes для оркестрації контейнерів. Це вимагає налаштування відповідних плагінів та забезпечення безперебійної взаємодії між цими інструментами. Одним з викликів є забезпечення сумісності між різними версіями інструментів та впровадження

нових функціональностей без порушення існуючих процесів.

Впровадження нових інструментів автоматизації може значно покращити ефективність CI/CD процесів. Наприклад, використання таких інструментів, як Terraform для управління інфраструктурою, дозволяє автоматизувати створення та налаштування ресурсів, таких як VPC, підмережі та IAM ролі. Це дозволяє знизити ризики помилок та забезпечити більш стабільну роботу системи. Інтеграційні підходи, такі як використання API для взаємодії між різними інструментами, дозволяють забезпечити більш гнучке та ефективне управління процесами.

Підтримка та масштабованість інфраструктури є важливими аспектами при впровадженні автоматизованих CI/CD процесів. Це включає управління контейнерами та кластером Kubernetes, забезпечення стабільної роботи системи та оптимізацію використання ресурсів. Одним з головних викликів є забезпечення ефективного управління великим обсягом контейнерів та забезпечення масштабованості кластера. Важливо також забезпечити стабільну роботу системи під час збільшення навантаження та впровадження нових функціональностей.

Масштабованість кластера Kubernetes є критично важливою для забезпечення стабільної роботи системи під час збільшення навантаження. Це включає налаштування автоматичного масштабування подів та вузлів, що дозволяє динамічно збільшувати або зменшувати кількість ресурсів залежно від поточного наван

Моніторинг та логування є важливими компонентами для забезпечення стабільної та ефективної роботи CI/CD процесів. Основним викликом у сфері моніторингу та логування є збір та аналіз даних з різних компонентів системи, таких як Jenkins, Kubernetes, Docker та інші інструменти CI/CD. Цей процес може бути складним через різноманітність форматів логів та необхідність обробки великих обсягів даних у режимі реального часу. Важливо забезпечити безперервний збір та аналіз даних для виявлення потенційних проблем на ранніх етапах та їхнього швидкого усунення.

Для покращення моніторингу та логування можна впровадити сучасні

інструменти, такі як Prometheus та Grafana. Prometheus є потужним інструментом для збору та зберігання метрик з різних компонентів системи. Він підтримує багатий набір функціональних можливостей, включаючи налаштування алертів та створення складних запитів для аналізу даних. Grafana, у свою чергу, є популярним інструментом для візуалізації даних, зібраних Prometheus. Використання цих інструментів дозволяє створювати інтуїтивно зрозумілі дашборди для моніторингу стану системи та виявлення аномалій у режимі реального часу.

Збір та аналіз логів є ще одним важливим аспектом моніторингу CI/CD процесів. Логи можуть містити цінну інформацію про стан системи, помилки та інші події, що відбуваються під час виконання пайплайнів. Для ефективного збору та аналізу логів можна використовувати інструменти, такі як ELK Stack (Elasticsearch, Logstash, Kibana). Elasticsearch забезпечує потужні можливості для пошуку та аналізу логів, Logstash використовується для збору та обробки логів, а Kibana дозволяє створювати візуалізації та дашборди для аналізу даних.

Список методів покращення моніторингу та логування може включати:

- Впровадження Prometheus для збору метрик та налаштування алертів.
- Використання Grafana для створення дашбордів та візуалізації даних;
- Використання ELK Stack для збору та аналізу логів;
- Налаштування автоматичних сповіщень про аномалії та помилки;
- Регулярний аналіз зібраних даних для виявлення потенційних проблем та оптимізацій;

Оцінка економічної ефективності впровадження автоматизованого CI/CD є важливим аспектом для визначення вартості та переваг даного підходу. Основним завданням є аналіз витрат, пов'язаних з впровадженням та підтримкою інфраструктури, а також оцінка економічної вигоди від автоматизації процесів. Це включає витрати на використання хмарних сервісів, ліцензії на програмне забезпечення, апаратні ресурси та витрати на заробітну плату для команди

розробників та DevOps інженерів.

Аналіз витрат може включати розрахунок вартості використання AWS ресурсів, таких як EC2 інстанси для розгортання Jenkins та Kubernetes кластера, ECR для зберігання Docker образів та S3 для зберігання артефактів. Важливо також враховувати витрати на ліцензії для використання інструментів, таких як SonarQube, а також витрати на підтримку та обслуговування інфраструктури. Крім того, потрібно оцінити витрати на навчання та розвиток команди, що включає навчання новим інструментам та практикам.

Оптимізація ресурсів є ключовим фактором для зниження витрат. Це може включати впровадження методів автоматичного масштабування для забезпечення ефективного використання ресурсів, використання дешевших інстансів для тестування та розробки, а також впровадження практик оптимізації коду та процесів для зменшення часу виконання пайплайнів. Важливо також розглядати можливості використання безкоштовних або відкритих інструментів, що можуть зменшити витрати на ліцензії.

Список методів для зниження витрат може включати:

- Використання автоматичного масштабування для оптимізації використання ресурсів;
- Впровадження методів оптимізації коду та процесів для зменшення часу виконання пайплайнів;
- Використання дешевших інстансів для тестування та розробки;
- Впровадження практик повторного використання ресурсів та інструментів;
- Використання безкоштовних або відкритих інструментів для зменшення витрат на ліцензії;

Оцінка економічної ефективності впровадження автоматизованого CI/CD дозволяє визначити реальні витрати та вигоди від даного підходу, що є важливим для прийняття обґрунтованих рішень щодо подальшого розвитку та оптимізації.

ВИСНОВКИ

Впровадження автоматизованого CI/CD на основі Kubernetes та AWS за допомогою Jenkins виявило численні переваги для процесів розробки, тестування та розгортання додатків. Автоматизація цих процесів забезпечує швидше, надійніше та масштабоване впровадження нових версій додатків, що сприяє підвищенню продуктивності та якості програмного забезпечення.

Основним результатом реалізації проекту є значне скорочення часу на розгортання нових версій додатків. Використання Jenkins для автоматизації CI/CD процесів дозволило зменшити середній час від комміту коду до розгортання у продуктивне середовище з кількох годин до кількох хвилин. Це забезпечує більш оперативну реакцію на зміни вимог та швидке виправлення помилок. Використання Kubernetes для оркестрації контейнерів забезпечило високу доступність та масштабованість системи. Kubernetes дозволив автоматично масштабувати ресурси в залежності від навантаження, що забезпечило стабільну роботу додатків навіть при пікових навантаженнях.

Проведені експерименти показали, що автоматизація тестування на кожному етапі пайплайну дозволила виявляти помилки на ранніх стадіях, що зменшило кількість дефектів у продуктивному середовищі. Це підвищило надійність процесу розгортання та мінімізувало час простою системи. Однак, впровадження автоматизованого CI/CD виявило і певні виклики. Складність налаштування та інтеграції різних інструментів і сервісів потребує значних знань та досвіду. Забезпечення безпеки процесів CI/CD є важливим аспектом, що потребує належного контролю доступу та захисту даних.

Для подальшого вдосконалення автоматизованого CI/CD необхідно покращити автоматизацію процесів моніторингу та логування, використовуючи такі інструменти, як Prometheus та Grafana. Важливим аспектом є також вдосконалення процесу тестування за допомогою сучасних інструментів для автоматичного тестування та забезпечення високої якості коду.

Загалом, впровадження автоматизованого CI/CD на основі Kubernetes та AWS за допомогою Jenkins є ефективним підходом для забезпечення безперервної інтеграції та доставки додатків. Це дозволяє значно скоротити час на розгортання нових версій, підвищити надійність та масштабованість системи, а також забезпечити високу продуктивність додатків. Водночас, для успішної реалізації всіх етапів CI/CD необхідно враховувати можливі виклики та забезпечити належне навчання та підтримку команди.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. GitHub - warolv/jenkins-eks: Build CI/CD of the future with Kubernetes (AWS EKS) and Jenkins. URL: http://github.com/warolv/jenkins-eks.
2. Eldad Assis, Rob Gravelle. Easily Automate Your CI/CD Pipeline With Kubernetes, Helm, and Jenkins. DZone, 29 березня 2024. URL: http://dzone.com/articles/easily-automate-your-ci-cd-pipeline-with-kubernetes-helm-and-jenkins.
3. DevOps and CI/CD: Automating Software Delivery and Infrastructure Changes with Jenkins, Kubernetes, and AWS. URL: http://dzone.com/articles/devops-and-ci-cd-automating-software-delivery-and-infrastructure-changes.
4. Kubernetes Documentation. URL: http://kubernetes.io/docs/home/.
5. AWS Documentation. URL: http://docs.aws.amazon.com/.
6. Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation by Jez Humble, David Farley. Addison-Wesley, 2010.
7. Kubernetes Up & Running: Dive into the Future of Infrastructure by Kelsey Hightower, Brendan Burns, Joe Beda. O'Reilly Media, 2017.
8. The DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations by Gene Kim, Patrick Debois, John Willis, Jez Humble. IT Revolution Press, 2016
9. Infrastructure as Code: Managing Servers in the Cloud by Kief Morris. O'Reilly Media, 201

Додаток А

Backend pipline

```
pipeline {
    agent any
    tools {
        jdk 'jdk'
        nodejs 'nodejs'
    }
    environment  {
        SCANNER_HOME=tool 'sonar-scanner'
        AWS_ACCOUNT_ID = credentials('ACCOUNT_ID')
        AWS_ECR_REPO_NAME = credentials('ECR_REPO2')
        AWS_DEFAULT_REGION = 'us-east-1'
        REPOSITORY_URI =
"${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/"
    }
    stages {
        stage('Cleaning Workspace') {
            steps {
                cleanWs()
            }
        }
        stage('Checkout from Git') {
            steps {
                git credentialsId: 'GITHUB', url:
'https://github.com/YevheniiHryshkovets/EKS-DevOps.git'
            }
        }
```

```
        stage('Sonarqube Analysis') {
            steps {
                dir('Application-Code/backend') {
                    withSonarQubeEnv('sonar-server') {
                        sh ''' $SCANNER_HOME/bin/sonar-scanner \
                        -Dsonar.projectName=three-tier-backend \
                        -Dsonar.projectKey=three-tier-backend '''
                    }
                }
            }
        }
        stage('Quality Check') {
            steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
                }
            }
        }
        stage('OWASP Dependency-Check Scan') {
            steps {
                dir('Application-Code/backend') {
                    dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit
--disableNodeAudit', odcInstallation: 'DP-Check'
                    dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                }
            }
        }
        stage('Trivy File Scan') {
            steps {
                dir('Application-Code/backend') {
```

```
                sh 'trivy fs . > trivyfs.txt'
            }
        }
    }
    stage("Docker Image Build") {
        steps {
            script {
                dir('Application-Code/backend') {
                    sh 'docker system prune -f'
                    sh 'docker container prune -f'
                    sh 'docker build -t ${AWS_ECR_REPO_NAME} .'
                }
            }
        }
    }
    stage("ECR Image Pushing") {
        steps {
            script {
                sh 'aws ecr get-login-password --region ${AWS_DEFAULT_REGION} |
docker login --username AWS --password-stdin ${REPOSITORY_URI}'
                sh 'docker tag ${AWS_ECR_REPO_NAME}
${REPOSITORY_URI}${AWS_ECR_REPO_NAME}:${BUILD_NUMBER}'
                sh 'docker push
${REPOSITORY_URI}${AWS_ECR_REPO_NAME}:${BUILD_NUMBER}'
            }
        }
    }
    stage("TRIVY Image Scan") {
        steps {
            sh 'trivy image
```

```
${REPOSITORY_URI}${AWS_ECR_REPO_NAME}:${BUILD_NUMBER} >
trivyimage.txt'
            }
        }
        stage('Checkout Code') {
            steps {
                git credentialsId: 'GITHUB', url:
'https://github.com/YevheniiHryshkovets/EKS-DevOps.git'
            }
        }
        stage('Update Deployment file') {
            environment {
                GIT_REPO_NAME = "EKS-DevOps"
                GIT_USER_NAME = "YevheniiHryshkovets"
            }
            steps {
                dir('Kubernetes-Manifests-file/Backend') {
                    withCredentials([string(credentialsId: 'github', variable:
'GITHUB_TOKEN')]) {
                        sh '''
                            git config user.email "zhenjagryshkovets@gmail.com"
                            git config user.name "YevheniiHryshkovets"
                            BUILD_NUMBER=${BUILD_NUMBER}
                            echo $BUILD_NUMBER
                            imageTag=$(grep -oP '(?<=backend:)[^ ]+' deployment.yaml)
                            echo $imageTag
                            sed -i
"s/${AWS_ECR_REPO_NAME}:${imageTag}/${AWS_ECR_REPO_NAME}:${BUI
LD_NUMBER}/" deployment.yaml
                            git add deployment.yaml
```

```
                    git commit -m "Update deployment Image to version
\${BUILD_NUMBER}"
                    git push
https://${GITHUB_TOKEN}@github.com/${GIT_USER_NAME}/${GIT_REPO_NA
ME} HEAD:master
                '''
            }
        }
    }
  }
}
```

Frontend pipline

```
pipeline {
    agent any
    tools {
        jdk 'jdk'
        nodejs 'nodejs'
    }
    environment  {
        SCANNER_HOME=tool 'sonar-scanner'
        AWS_ACCOUNT_ID = credentials('ACCOUNT_ID')
        AWS_ECR_REPO_NAME = credentials('ECR_REPO1')
        AWS_DEFAULT_REGION = 'us-east-1'
        REPOSITORY_URI =
"${AWS_ACCOUNT_ID}.dkr.ecr.${AWS_DEFAULT_REGION}.amazonaws.com/"
    }
    stages {
        stage('Cleaning Workspace') {
            steps {
                cleanWs()
            }
        }
        stage('Checkout from Git') {
            steps {
                git credentialsId: 'GITHUB', url:
'https://github.com/YevheniiHryshkovets/EKS-DevOps.git'
            }
        }
```

```
        stage('Sonarqube Analysis') {
            steps {
                dir('Application-Code/frontend') {
                    withSonarQubeEnv('sonar-server') {
                        sh ''' $SCANNER_HOME/bin/sonar-scanner \
                        -Dsonar.projectName=three-tier-frontend \
                        -Dsonar.projectKey=three-tier-frontend '''
                    }
                }
            }
        }
        stage('Quality Check') {
            steps {
                script {
                    waitForQualityGate abortPipeline: false, credentialsId: 'sonar-token'
                }
            }
        }
        stage('OWASP Dependency-Check Scan') {
            steps {
                dir('Application-Code/frontend') {
                    dependencyCheck additionalArguments: '--scan ./ --disableYarnAudit
--disableNodeAudit', odcInstallation: 'DP-Check'
                    dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
                }
            }
        }
        stage('Trivy File Scan') {
            steps {
                dir('Application-Code/frontend') {
```

```
                sh 'trivy fs . > trivyfs.txt'
            }
        }
    }
    stage("Docker Image Build") {
        steps {
            script {
                dir('Application-Code/frontend') {
                    sh 'docker system prune -f'
                    sh 'docker container prune -f'
                    sh 'docker build -t ${AWS_ECR_REPO_NAME} .'
                }
            }
        }
    }
    stage("ECR Image Pushing") {
        steps {
            script {
                sh 'aws ecr get-login-password --region ${AWS_DEFAULT_REGION} |
docker login --username AWS --password-stdin ${REPOSITORY_URI}'
                sh 'docker tag ${AWS_ECR_REPO_NAME}
${REPOSITORY_URI}${AWS_ECR_REPO_NAME}:${BUILD_NUMBER}'
                sh 'docker push
${REPOSITORY_URI}${AWS_ECR_REPO_NAME}:${BUILD_NUMBER}'
            }
        }
    }
    stage("TRIVY Image Scan") {
        steps {
            sh 'trivy image
```

```
${REPOSITORY_URI}${AWS_ECR_REPO_NAME}:${BUILD_NUMBER} >
trivyimage.txt'
            }
        }
        stage('Checkout Code') {
            steps {
                git credentialsId: 'GITHUB', url:
'https://github.com/YevheniiHryshkovets/EKS-DevOps.git'
            }
        }
        stage('Update Deployment file') {
            environment {
                GIT_REPO_NAME = "EKS-DevOps"
                GIT_USER_NAME = "YevheniiHryshkovets"
            }
            steps {
                dir('Kubernetes-Manifests-file/Frontend') {
                    withCredentials([string(credentialsId: 'github', variable:
'GITHUB_TOKEN')]) {
                        sh '''
                            git config user.email "aman07pathak@gmail.com"
                            git config user.name "AmanPathak-DevOps"
                            BUILD_NUMBER=${BUILD_NUMBER}
                            echo $BUILD_NUMBER
                            imageTag=$(grep -oP '(?<=frontend:)[^ ]+' deployment.yaml)
                            echo $imageTag
                            sed -i
"s/${AWS_ECR_REPO_NAME}:${imageTag}/${AWS_ECR_REPO_NAME}:${BUI
LD_NUMBER}/" deployment.yaml
                            git add deployment.yaml
```

```
                    git commit -m "Update deployment Image to version
\${BUILD_NUMBER}"
                    git push
https://${GITHUB_TOKEN}@github.com/${GIT_USER_NAME}/${GIT_REPO_NA
ME} HEAD:master
                '''
            }
        }
    }
  }
}
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра комп'ютерної інженерії

ПРЕЗЕНТАЦІЯ ДО КВАЛІФІКАЦІЙНОЇ РОБОТИ
на тему
"АВТОМАТИЗОВАНИЙ CI/CD НА ОСНОВІ KUBERNETES ТА AWS ІЗ ЗОСЕРЕДЖЕННЯМ НА РОЗРОБЦІ"

Виконав студент крупи КІД-41
Гришковець Євгеній Петрович
Керівник кваліфікаційної роботи :
Антоненко Артем Васильович,
Доцент кафедри КІ

Київ 2024

Мета роботи – розробка та впровадження автоматизованої системи CI/CD на основі Kubernetes та AWS

Об'єкт дослідження – процес автоматизації CI/CD в хмарному середовищі

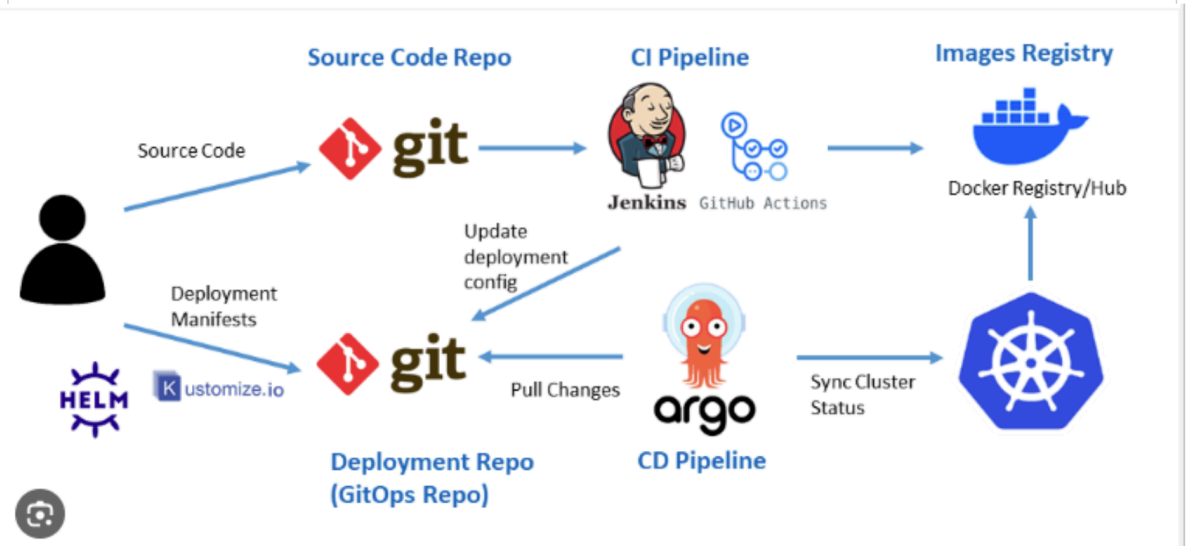
Предмет дослідження – технології та інструменти для автоматизації CI/CD з використанням Kubernetes та AWS.

Актуальність обраної теми

Для сучасних компаній, які займаються розробкою програмного забезпечення, важливо мати ефективні та надійні процеси безперервної інтеграції (CI) та безперервної доставки (CD). Автоматизація цих процесів дозволяє зменшити кількість помилок, прискорити розробку та розгортання нових функцій, а також забезпечити стабільність роботи програмного забезпечення. Використання Kubernetes та AWS для побудови автоматизованих CI/CD конвеєрів забезпечує масштабованість, гнучкість та високу доступність сервісів. Це дозволяє командам розробників зосередитися на створенні якісного продукту, мінімізуючи затрати часу на налаштування та підтримку інфраструктури. Таким чином, дослідження та впровадження автоматизованих CI/CD систем на основі Kubernetes та AWS має вирішальне значення для підвищення продуктивності та якості розробки програмного забезпечення

3

Ключові компоненти CI/CD



4

Ключові компоненти CI/CD

Процес роботи з використанням Continuous Integration та Continuous Deployment (CI/CD) включає кілька ключових етапів. По-перше, це збірка, де вихідний код автоматично збирається з репозиторію і компілюється в виконуваний файл або артефакт. Потім йде тестування, де проводяться різноманітні автоматизовані тести, включаючи модульні, інтеграційні та функціональні, для виявлення помилок і проблем. Наступний етап - створення контейнерів, коли програма або додаток упаковується у Docker контейнери або в інші контейнеризаційні технології. Після цього відбувається розгортання, коли програмне забезпечення автоматично розгортається на цільових серверах або хмарних інфраструктурах. І, нарешті, моніторинг - останній етап, де надається моніторинг та логування для відстеження продуктивності та ефективності розгорнутого програмного забезпечення. Цей процес дозволяє автоматизувати та прискорити процес розробки, тестування та розгортання програмного забезпечення, зменшуючи час і ризики людської помилки.

5

Альтернативи CI/CD

ми розглянемо різні підходи до управління розробкою та розгортанням програмного забезпечення. Серед них будуть вказані альтернативні методи, такі як ручна збірка та розгортання, коли процеси збирання вихідного коду та його розгортання відбуваються вручну без використання автоматизованих інструментів CI/CD. Також варто розглянути можливість ручного керування процесом розгортання, або використання скриптів для автоматизації окремих етапів.

6

Ручна збірка

Ручною збіркоє є процес в якому розробники виконують кожний етап вручну без використання автоматизованих інструментів CI/CD. Цей процес може включати в себе такі етапи, як збірка вихідного коду, його компіляція, тестування та розгортання. Як у будь якого іншого у цього підходу є свої переваги та недоліки. На приклад, перевагами можуть бути більший контроль над процесом збірки та розгортання, а також простота в підтримці менших проектів. Однак недоліками є велика кількість ручних операцій, яка збільшує ризик помилок та затримок у випуску програмного забезпечення.

7

Ручне розгортання

Ручне розгортання це процес ручного розгортання програмного забезпечення який не передбачає використання автоматизованих інструментів CI/CD. Цей процес може включати копіювання вихідних файлів на сервер, запуск необхідних скриптів або виконання інших ручних дій для розгортання програми. Він може бути простим для менших проектів або там, де автоматизація не є критичною, але часто стає непрактичним для великих та складних систем через високі ризик помилок.
Переваги ручного розгортання можуть включати більший контроль над процесом розгортання, зручність у випадках, коли автоматизація не доцільна або не можлива, а також можливість швидко внести зміни в процес розгортання вручну.
Проте недоліки такого підходу включають в себе високу ймовірність людських помилок, довгий час розгортання через ручні операції та складнощі в підтримці та управлінні розгортаннями на великій кількості серверів або середовищ.

7

Порівняння CI/CD та альтернативних підходів

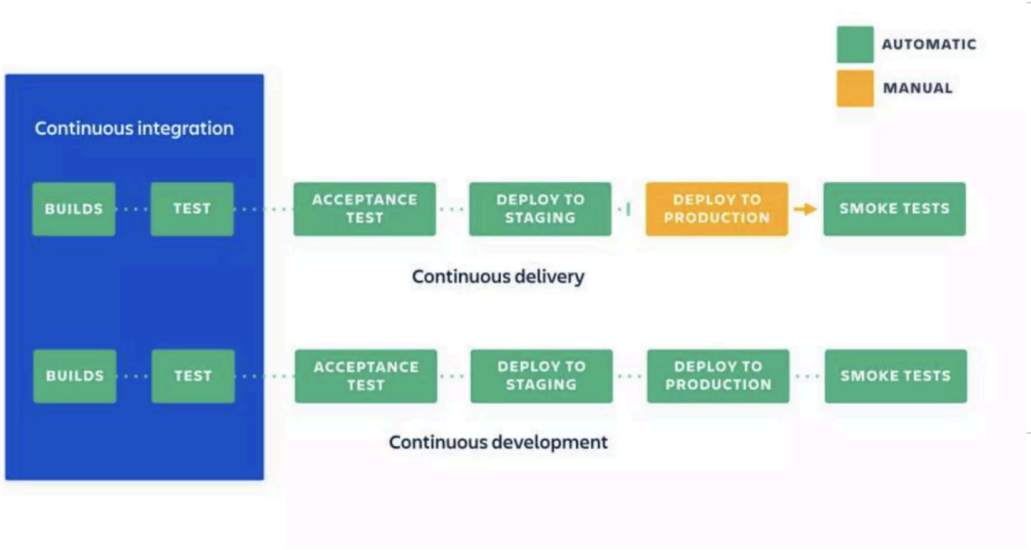
Переваги використання CI/CD включають автоматизацію процесу збірки, тестування та розгортання, що призводить до зниження часу розробки та покращення якості програмного забезпечення. Крім того, CI/CD дозволяє швидше виявляти та виправляти помилки, сприяє швидкому впровадженню змін та полегшує співпрацю в команді розробників.

Недоліки використання CI/CD можуть включати складність налаштування та підтримки інфраструктури CI/CD, можливість виникнення помилок у конфігурації пайплайнів та збірок, а також необхідність великої автоматизації та тестування для досягнення повноцінного ефекту.

Аналіз впливу впровадження CI/CD на розробку програмного забезпечення показує значне покращення якості, швидкості та ефективності розробки, зменшення часу від внесення змін до їх впровадження в продуктивну середу, а також полегшення співпраці між розробниками та іншими учасниками процесу розробки.

7

Етапи CI/CD



7

Огляд реальних випадків використання CI/CD

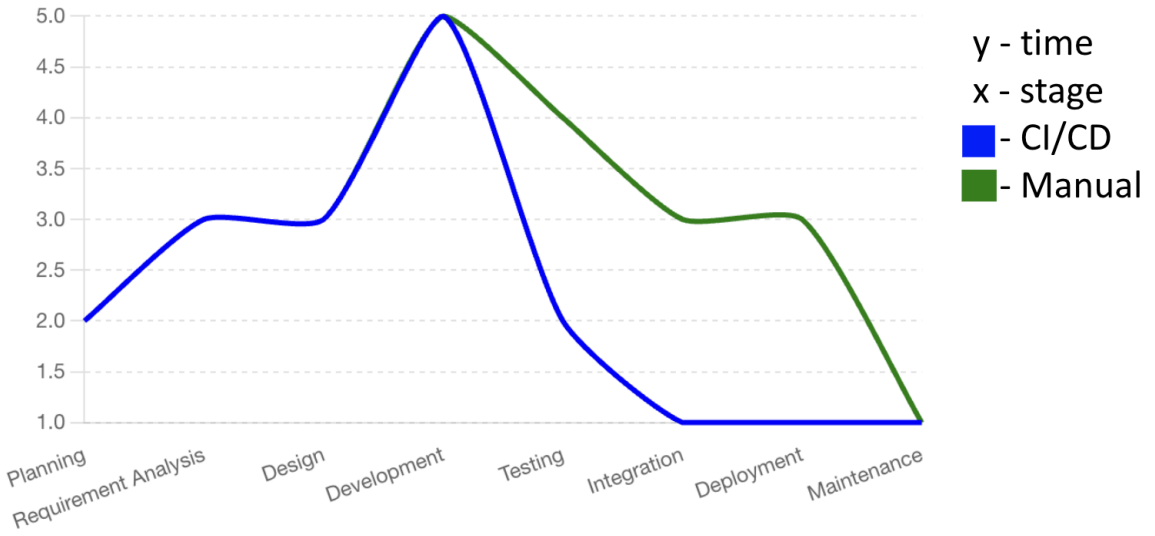
Netflix використовує CI/CD для швидкого впровадження змін у своїх сервісах, що дозволяє їм швидко реагувати на зміни в потребах користувачів та ринкові умови.

Іншим прикладом є Spotify, який використовує CI/CD для автоматизованого релізу нових функцій у своєму музичному сервісі. Це допомагає їм швидше реагувати на зміни в музичній індустрії та побажання користувачів.

Успішне впровадження CI/CD дозволило цим компаніям значно підвищити швидкість розробки, зменшити час між внесенням змін та їх впровадженням у продуктивну середу, а також покращити якість та стабільність їх програмного забезпечення.

7

Порівняння CI/CD та ручного підходу



7

ВИСНОВКИ

У процесі дослідження були проаналізовані можливості та переваги використання Kubernetes та AWS для побудови автоматизованих CI/CD конвеєрів. Встановлено, що застосування цих технологій забезпечує високу гнучкість та масштабованість інфраструктури, що дозволяє оптимізувати процеси безперервної інтеграції та доставки програмного забезпечення. Автоматизація за допомогою Kubernetes та AWS знижує ризики людських помилок, підвищує швидкість розгортання нових функцій та поліпшує загальну стабільність програмних продуктів.

На основі проведеного аналізу та тестування, було виявлено, що автоматизовані CI/CD конвеєри на основі Kubernetes та AWS дозволяють значно скоротити час розробки та розгортання додатків, а також підвищують ефективність роботи команд розробників. В результаті дослідження підтверджено, що впровадження автоматизованих CI/CD процесів з використанням сучасних хмарних технологій є критично важливим для підтримки високого рівня продуктивності та якості програмного забезпечення в умовах динамічних ринкових вимог.