Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)

Кафедра _____Інформаційних управляючих систем_____
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти _____другий (магістерський)_____

____Дослідження та використання методу розгортання Enterprise-додатків в k8s кластері_____ _____
(тема)

Виконав:
студент _2_ курсу, групи _ІУСТм-22-1____
_____Пироженко Вадим Сергійович_____
(прізвище, ім'я, по батькові)

Спеціальність ____122 Комп'ютерні_____
_____науки_____
(код і повна назва спеціальності)
Тип програми __ __освітньо-професійна_____
(освітньо-професійна або освітньо-наукова)
Освітня програма _Інформаційні управляючі системи та технології_____
(повна назва освітньої програми)

Керівник _____Марина КУДРЯВЦЕВА_____
(Власне ім'я ПРІЗВИЩЕ)

Допускається до захисту

Зав. кафедри _____ ___Костянтин ПЕТРОВ__
(підпис) (Власне ім'я ПРІЗВИЩЕ)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Інформаційних управляючих систем

Рівень вищої освіти другий (магістерський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Інформаційні управляючі системи та технології
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«_20_» листопада 20 23 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Пироженку Вадиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження та використання методу розгортання Enterprise-додатків в k8s кластері

затверджена наказом університету від 16 листопада 2023 р. № 1359Ст

2. Термін подання студентом роботи до екзаменаційної комісії 17 січня 2024 р.

3. Вихідні дані до роботи науково-технічні публікації та дані Інтернет-джерел з тематики кваліфікаційної роботи

4. Перелік питань, що потрібно опрацювати в роботі Аналіз предметної області. Огляд існуючих методів розгортання Enterprise-додатків в k8s кластері. Аналіз хмарних технологій. Вдосконалення методу розгортання Enterprise-додатків в k8s кластері. Проведення експериментальної перевірки розробленого методу. Аналіз отриманих результатів.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	20.11.2023	виконано
2	Аналіз літератури та Інтернет-джерел	21.11.2023–24.11.2023	виконано
3	Аналіз предметної області	27.11.2023–30.11.2023	виконано
4	Огляд існуючих методів розгортання Enterprise-додатків в k8s кластері	01.12.2023–06.12.2023	виконано
5	Аналіз CI/CD технолоній	07.12.2023–11.12.2023	виконано
6	Модифікація методу розгортання	13.12.2023–15.12.2023	виконано
7	Проведення експерементальної перевірки	18.12.2023–20.12.2023	виконано
8	Аналіз отриманих результатів.	21.12.2023–22.12.2023	виконано
9	Написання пояснювальної записки	22.12.2023–29.12.2023	виконано
10	Підготовка презентації	29.12.2023–03.01.2024	виконано
11	Надання роботи для перевірки на плагіат	03.01.2024	виконано
12	Надання роботи на підпис	05.01.2024	виконано
13	Надання роботи на рецензію	08.01.2024	виконано
14	Надання підписаної завідувачем кафедрою роботи в ЕК	17.01.2024	виконано
15	Захист кваліфікаційної роботи	19.01.2024	виконано

Дата видачі завдання __20___ __листопада___2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. каф. ІУС Марина Кудрявцева
(підпис) (посада, власне ім’я, прізвище)

РЕФЕРАТ

Пояснювальна записка до магістерської кваліфікаційної роботи містить: 97 с., 75 рис., 10 форм., 1 табл., 20 джерел, 1 додаток.

CI/CD, CI/CD ПАЙПЛАЙНИ, ENTERPRISE-ДОДАТКИ, HELM, KUBERNETES.

Об'єкт дослідження – Enterprise-додатки Kubernetes.

Предмет дослідження – імперативний метод, декларативний метод, менеджери пакетів, концепція GitOps, оператори Kubernetes та системи неперервної інтеграції та постачання (CI/CD).

Мета роботи – вдосконалення методу розгортання додатків у Kubernetes з метою ефективного впровадження підприємницьких програм в це середовище.

Дослідження, проведені у цій роботі, базуються на використанні методів CI/CD та пакетного менеджера.

На основі результатів виконаних досліджень був модифікований метод для розгортання Enterprise-додатку з використанням CI/CD. Модифікований метод може використовуватися більшістю організацій з типовою лінійно-функціональною організаційною структурою.

Кваліфікаційна робота виконана згідно з методичними вказівками [1], та вимогами ДСТУ 3008:2015 [2].

ABSTRACT

Thesis contains: 97 p., 75 fig., 10 form., 1 table, 20 sources., 1 appendix.

CI/CD, CI/CD PIPELINES, ENTERPRISE APPLICATIONS, HELM, KUBERNETES.

The object of Enterprise applications on Kubernetes.

The subject of investigation encompasses the imperative method, declarative method, package managers, the GitOps concept, Kubernetes operators, and continuous integration and delivery systems (CI/CD).

The aim of the work is to enhance the method of deploying applications in Kubernetes for the effective implementation of entrepreneurial programs in this environment.

The research conducted in this work is based on the application of CI/CD methods and a package manager.

The research conducted in the work is based on the use of CI/CD methods and a package manager.

Based on the results of the conducted research, a method for deploying an Enterprise application using CI/CD was modified. The modified method can be utilized by the majority of organizations with a typical linear-functional organizational structure.

The thesis was carried out in accordance with the guidelines [1] and the requirements of DSTU 3008:2015 [2].

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАКИ

API – Application programming interface

AWS − Amazon Web Services

CI/CD – Безперервна інтеграція (CI) та безперервне розгортання (CD)

Git – Розподілена система управління версіями

GitOps – Процес системних операцій, пов’язаний з інструментом Git

Helm – Менеджер пакетів

IaaS − Infrastructure as a Service

IAM − Identity and Access Management

IC – Інформаційна Система

JCasC – Jenkins Configuration as Code

JSON – javaScript Object Notation

Kubernetes (K8s) – відкрите програмне забезпечення для оркестрування контейнеризованих програм

S3 – Amazon Simple Storage Service

VSM – Value Stream Map (VSM)

YAML – Формат серіалізації даних пакетів

ВСТУП

У багатьох корпоративних структурах, навіть у тих, що не займаються інформаційними технологіями, визначено присутність власних цифрових сервісів, які підлягають систематичному впровадженню та оновленню. Цей процес включає в себе додавання нововведень, виправлення виявлених недоліків, ліквідацію вразливостей та активацію нових функцій. Важливою характеристикою цього циклу розробки є необхідність забезпечення неперервної функціональності під час оновлення, забезпечуючи безперебійну роботу. Необхідність у швидкості впровадження змін та гнучкості додатків викликала перетворення архітектурних підходів у сфері програмного забезпечення, призводячи до відмови від монолітних структур на користь мікросервісної архітектури.

З метою спрощення трансферу мікросервісів з тестового середовища на продуктивне та підвищення ефективності розробки, ці мікросервіси були впаковані в контейнери. З поступовим удосконаленням програм, кількість мікросервісів та контейнерів збільшувалася, викликаючи потребу в оптимізованому управлінні та конфігурації процесу розробки програмного забезпечення. З метою вирішення цих викликів виник та розвинувся Kubernetes, який забезпечує централізоване управління контейнерами, прискорюючи та спрощуючи процес введення нових продуктів на ринок і створення ефективного циклу розробки та тестування.

У кваліфікаційній роботі будуть досліджуватися найпопулярніші методи CI/CD для використання в проектах з мікросервісами на Kubernetes. Для аналізу було обрано бізнес-проект, для якого необхідно обрати рішення CI/CD, а також для тестування перенести існуючі компоненти на мікросервіси, що будуть розгорнуті на кластері Kubernetes.

Метою кваліфікаційної роботи є вдосконалення методу розгортання додатків у Kubernetes з метою ефективного впровадження підприємницьких програм в це середовище. Важливо знайти та дослідити методи, які б підходили для вирішення поставленої задачі найбільш ефективно. Поставлена мета вимагає вирішення наступних наукових задач:

− аналіз сучасного стану технологічного ландшафту, пов'язаного з розгортанням Enterprise-додатків у Kubernetes кластері;

− дослідження існуючих підходів і методів, які використовуються для розгортання додатків в середовищі Kubernetes, а також визначення переваг та обмежень кожного з існуючих методів;

− влосконалення та опис методу, спрямованого на оптимізацію процесу розгортання Enterprise-додатків в середовищі Kubernetes;

− реалізація експерименту з використанням вдосконаленого методу розгортання додатків в реальному або симульованому Kubernetes кластері;

− порівняння результатів з існуючими методами розгортання та висновки щодо вдосконаленого методу розгортання Enterprise-додатків в Kubernetes.

1 ОПИС ТА АНАЛІЗ МЕТОДІВ РОЗГОРТАННЯ ENTERPRISE-ДОДАТКІВ В K8S КЛАСТЕРІ

1.1 Загальна характеристика методів розгортання додатків в Kubernetes кластері

Методи розгортання додатків в Kubernetes кластері можуть включати різні підходи і інструменти залежно від потреб та вимог проекту.

Загалом, всі ці методи можна класифікувати за такими основними типами:

− імперативний;

− декларативний;

− розробка шаблонів та використання інструментів, таких як Helm та інші менеджери пакетів;

− GitOps;

− оператори Kubernetes;

− CI/CD пайплайни.

Додатково, в цьому контексті важливим елементом Agile-підходу є використання Continuous Integration та Continuous Deployment (CI/CD). Безперервна інтеграція (CI) та безперервне розгортання (CD) — це популярні практики розробки програмного забезпечення для автоматизації та скорочення часу зворотного зв'язку. Однак, неправильно встановлені конвеєри CI/CD можуть спричинити затримки у розробці. З цієї причини розглянемо найпопулярніші практики CI/CD та порівняємо їх.

1.2 Імперативний метод

Імперативний метод дозволяє використовувати API Kubernetes не вимагаючи файлів конфігурації чи маніфестів у форматі YAML. У такому підході все, що потрібно зробити, це сказати, що ми хочемо зробити, і Kubernetes візьме на себе відповідальність за визначення того, що потрібно зробити для досягнення очікуваного результату [3]. На рисунку 1.1 наведено кілька кроків, потрібно виконати для розгортання додатку в Kubernetes:

– створення деплойменту;

– експозиція служби;

– перевірка стану.

```
kubectl create deployment my-app --image=your-docker-image
kubectl expose deployment my-app --port=80 --type=LoadBalancer
kubectl get services

```

Рисунок 1.1 – Розгортання додатку в Kubernetes імперативним методом

1.3 Декларативний метод

Декларативний метод розгортання додатку в Kubernetes полягає в тому, що розробник описує бажаний стан системи у вигляді конфігураційних файлів, а не визначає конкретні кроки для досягнення цього стану [3]. Основна ідея полягає в тому, щоб вказати, як має виглядати система в результаті розгортання, і залишити на сам Kubernetes завдання

визначення оптимального шляху до цього стану. Приклад конфігураційного файлу YAML наведено на рисунку 1.2.

```yaml
# deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: your-docker-image
        ports:
        - containerPort: 80
```

Рисунок 1.2 – Конфігураційнний файл YAML

У цьому прикладі, розробник описує бажаний стан системи, вказуючи кількість реплік, контейнери, образи та інші параметри. Після застосування цього конфігураційного файлу до Kubernetes, сам Kubernetes вирішить, як запустити та утримувати три репліки додатку, як вказано в конфігурації.

1.4 Розробка шаблонів та використання інструментів, таких як Helm та інші менеджери пакетів

Helm - це інструмент управління пакетами для Kubernetes, який спрощує розгортання та керування додатками та їх залежностями в середовищі Kubernetes. Використання Helm та інших пакетних менеджерів

дозволяє описати, управляти та розгортати додатки на кластері Kubernetes [4]. Helm Chart - це пакет, який містить всі необхідні ресурси та конфігурацію для розгортання додатку в Kubernetes приклад якого наведено на рисунку 1.3. Helm Chart легко створювати, версіонувати, публікувати та ділитися ними.

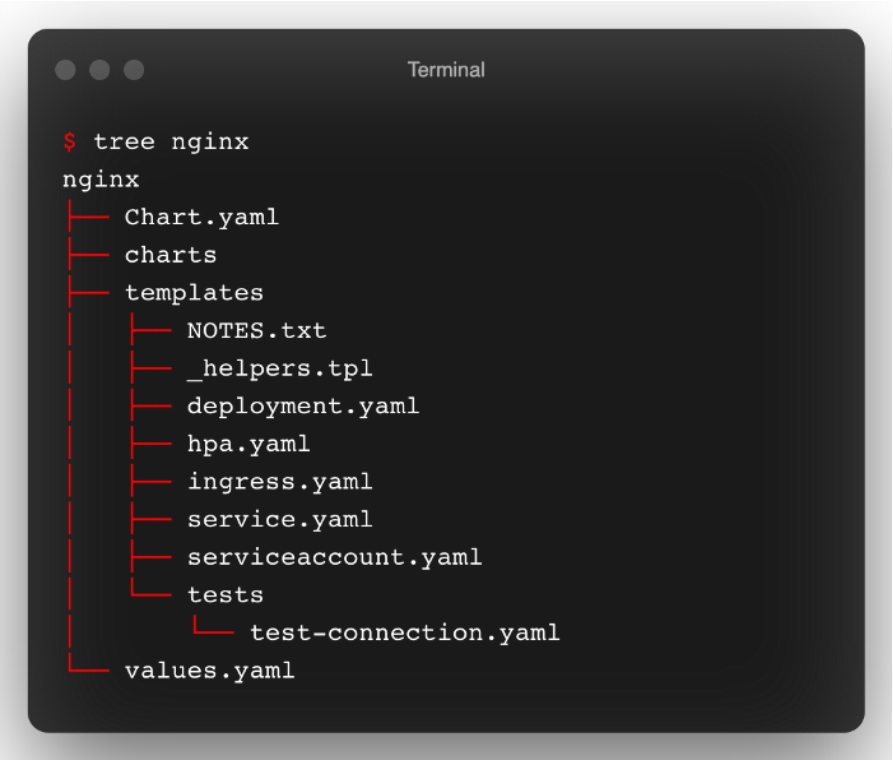

Рисунок 1.3 – Графічне представлення Helm Chart

Після детального розгляду Helm Chart тепер давайте звернемо погляд на іншу захоплюючу альтернативу - Kustomize. Як і Helm Chart, Kustomize дозволяє кастомізувати конфігурації Kubernetes, використовуючи файли та робочі каталоги. Проте для встановлення додатку Kustomize не використовує менеджер пакетів.

1.5 GitOps

GitOps це методологія управління інфраструктурою та розгортанням програмного забезпечення, в якій вся конфігурація та опис стану інфраструктури зберігаються у системі контролю версій, такій як Git. Основна ідея полягає в тому, що стан системи повинен відображати стан, визначений у Git-репозиторії. В основі GitOps лежать принципи декларативної конфігурації, керування версіями, автоматизоване розгортання та самовідновлення [5].

Підхід GitOps, придуманий у Weaveworks, ґрунтується на ідеї того, що є безперервний reconciliation loop - який продовжує стежити за джерелом правди (у нашому випадку Git-репозиторієм) і безперервно синхронізувати його стан у реальний світ (у нашому випадку Kubernetes-кластер).

Якщо подивитися на те, як працюють контролери в Kubernetes, ми побачимо таку ж логіку, за винятком того, що вони дивляться в Kubernetes API, а не в Git-репозиторій.

Є об'єкт Deployment, що генерує ресурс ReplicaSet. Кожен об'єкт ReplicaSet обслуговується контролером ReplicaSet, який генерує вже безпосередньо піди. Якщо ми видаляємо якусь підручну, ReplicaSet-контролер помітить цю зміну в кластері і спробує привести її до стану, описаного в об'єкті. Те саме станеться і з об'єктом ReplicaSet — якщо його видалити, то Deployment-контролер створить новий ReplicaSet, щоб відповідати стану, описаному в Deployment. Це працює і в інший бік – якщо ми оновлюємо спеку об'єкта Deployment, створюються новий ReplicaSet та новий Pod.

Контролери та безперервне приведення до бажаного стану – це основна логіка Kubernetes. Саме їй надихнулися творці підходу GitOps.

GitOps має на увазі, що є якийсь GitOps-оператор наведено на рисунку 1.4 або контролер, який робить те саме, що і контролери в Kubernetes, але дивиться не в Kubernetes API, а в конкретний Git-репозиторій. Тобто бере стан Git-репозиторія та перекладає його в Kubernetes. А також слідкує за тим, щоб воно завжди йому відповідало (рисунок 1.4).

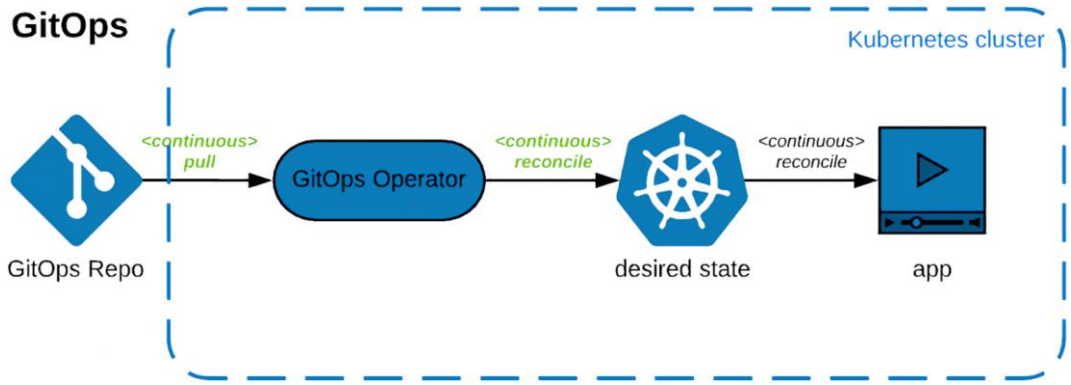

Рисунок 1.4 – Графічне представлення GitOps методології

Тут варто згадати, що GitOps не обов'язково зав'язаний на Git. Як джерело правди може виступати S3-бакет або навіть Helm-репозиорій. Найбіль популярними інструментами GitOps є ArgoCD та Flux.

Argo CD - один із найпопулярніших GitOps-інструментів. Він живе всередині Kubernetes і там же розгортає сутність [5]. Argo CD надає зручний RBAC, тобто управління правами та доступами. В інтерфейсі можна переглянути свої дії, керувати програмами та примусово синхронізувати їх. Argo CD входить до CNCF, що викликає до нього велику довіру. Argo CD може розгортати в кластер:

– звичайні маніфести;

– kustomize;

– jsonnet;

– helm-чарти.

Маніфест файли можна взяти з репозиторію Git або Helm. Argo CD розгортає маніфести тільки в кластери Kubernetes, але не тільки в кластер, в якому знаходиться сам: він вміє керувати великою кількістю кластерів одночасно. Приклад розгортання додатків в Kubernetes наведено на рисунку 1.5.

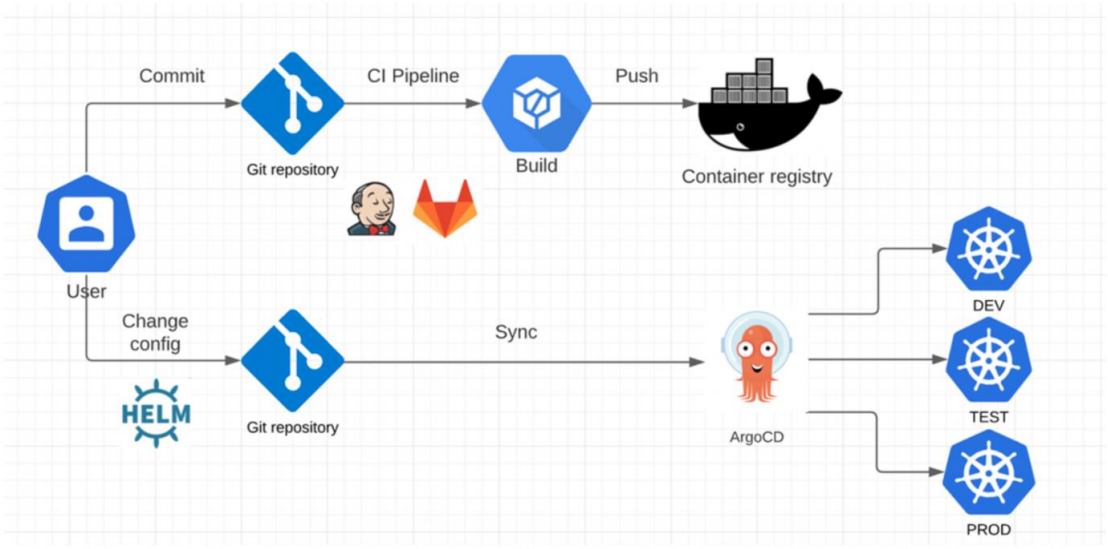

Рисунок 1.5 − Схема розгортання додатка в Kubernetes з Argo CD

Flux відрізняється в своєму підході. Він реалізує патерн Kubernetes із кастомними ресурсами та контролерами [5]. Схема розгортання додатку за допомогою Flux наведена на рисунку 1.6. У Flux є source контролер, який завантажує зміни із source repo.

Крім Git, це може бути S3 або Helm Chart Repository. А також окремі Kustomize-контролер та Helm-контролер, які займаються застосуванням маніфестів у кластер. Якби ми хотіли розширити Flux, то довелося б написати окремий контролер і CustomResource для нього.

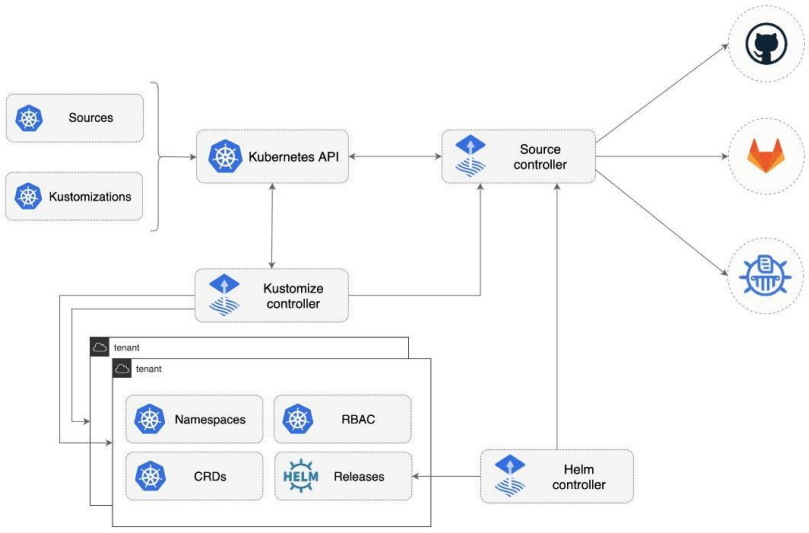

Рисунок 1.6 − Схема розгортання додатка в Kubernetes з Flux

1.6 Оператори Kubernetes

Оператори Kubernetes - це спеціальні контролери, які розширюють функціональність Kubernetes, дозволяючи автоматизувати рутинні завдання та управління додатками. Вони використовують природні можливості Kubernetes для забезпечення автоматизованого розгортання, масштабування та керування додатками [6]. Його сила в тому, що він може взаємодіяти як з кластером API, так і з зовнішніми ресурсами. А найголовніше, він може приводити систему в потрібний стан своїми власними силами. Від розробників потрібно лише декларативний опис потрібного стейту. Найважливіша частина кастомного ресурсу в Kubernetes-оператори – це контролер. Він здійснює всю логіку взаємодії та працює за принципом нескінченної петлі, тому позначений символом кругової стрілки. Контролер стежить за змінами в ресурсах і призводить систему до бажаного стану. Схема використання оператора в Kubernetes наведена на рисунку 1.7

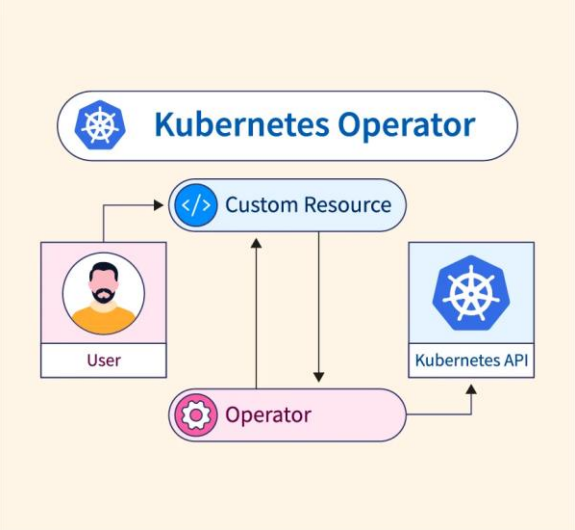

Рисунок 1.7 − Оператор Kubernetes

Контролер зручний тим, що в ньому один раз описується логіка і далі вся ручна робота лягає на плечі вбудованих засобів Kubernetes-контролера та самого кластера. Оператори призначені для конкретних програм.

1.7 CI/CD Пайплайни

Інтеграція з Continuous Integration та Continuous Deployment для автоматизації процесів збирання, тестування та розгортання коду в Kubernetes кластер.

Існує небагато варіантів класифікації CI/CD. За розміщенням — локальні та хмарні. За наявними інструментами — тільки CI, тільки CD, CI/CD одночасно. Нижче наведено приклади за наявними інструментами.

Локальні — GitLab CI/CD, TeamCity, Bamboo, GoCD, Jenkins, CircleCI.

Хмарні — BitBucket Pipelines, Heroku CI, Travis CI, Codeship, Buddy CI, AWS CodeBuild.

Тільки CI — CircleCI, Travis CI, Concourse CI тощо.

Тільки CD — GoCD, GoCD та інші.

У цьому підрозділі наведено знайдені популярні рішення CI/CD, їх коротка характеристика та особливості, сфери застосування та існуючі інструменти.

GitLab CI/CD — є досить новим учасником простору CI/CD, але він вже досяг першого місця в Forrester Wave for Continuous Integration Tools. Це величезне досягнення в такій переповненій та висококваліфікованій галузі. Що робить GitLab CI таким чудовим? Він використовує файл YAML для опису всього конвеєру. Він також має функціонал, який називається Auto DevOps, що дозволяє для більш простих проектів автоматично будувати конвеєр з вбудованими кількома тестами. Ця система використовує вбудовані пакети Herokuish (Проект консолідує та відокремлює логіку сумісності Heroku (запуск buildpacks, синтаксичний аналіз Procfile) та підтримку робочого процесу (імпорт/експорт кульок) із конкретних зображень платформи, таких як Dokku/Buildstep, Deis, Flynn тощо.) для визначення мови та способу побудови програми [7]. Нижче показано організацію процесу CI/CD (рисунок 1.8).

Деякі мови також можуть управляти базами даних, що є справжньою зміною гри для створення нових додатків та їх розгортання до виробництва з самого початку процесу розробки. Система має вбудовану інтеграцію в Kubernetes і автоматично розгорне вашу програму в кластер Kubernetes, використовуючи одну з декількох різних методологій розгортання, таких як розгортання на основі відсотків та синьо-зелені розгортання [7].

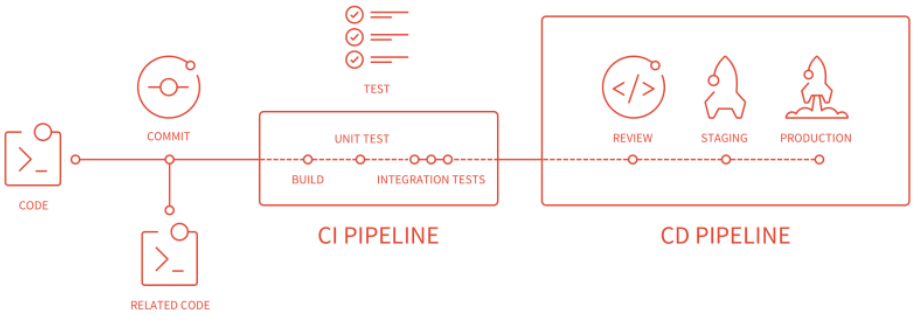

Рисунок 1.8 − Організація процесу CI/CD в GitLabCI

На додаток до своїх функціональних можливостей CI, GitLab пропонує безліч додаткових функцій, таких як операції та моніторинг за допомогою Prometheus, розгорнутого автоматично з вашим додатком; управління портфелем та проектами за допомогою GitLab Issues, Epics та Milestones; перевірки безпеки, вбудовані в конвеєр, результати яких надаються як сукупність для багатьох проектів; і можливість редагувати код прямо в GitLab за допомогою WebIDE, який навіть може забезпечити попередній перегляд або виконати частину конвеєру для швидшого зворотного зв'язку.

CircleCI — це безперервна інтеграція, інженерний процес, який розробники програмного забезпечення використовують для розробки, тестування та розгортання програм на декількох платформах з більшою легкістю та швидкістю. CircleCI пропонує продукти та можливості на рівні підприємств із універсальністю стартапів. Працюють з Linux, macOS, Ios та Windows-SaaS або за брандмауером.

Організації віддають перевагу CircleCI, оскільки завдання швидко виконуються, а збірки можна оптимізувати за швидкістю. Він може бути налаштований на ефективний запуск дуже складних конвеєрів за допомогою складного кешування, кешування шару докера, класів ресурсів для роботи

на більш швидких машинах та ціноутворення продуктивності. Як розробник, який використовує circleci.com, ви можете використовувати SSH для будь-якої роботи для налагодження проблем збірки, налаштувати паралелізм у файлі .circleci/config.yml для швидшого запуску завдань та налаштувати кешування двома простими клавішами для повторного використання даних із попередніх завдань у вашому робочому процесі. Він пропонує відстеження та розуміння ваших збірок як оператора CircleCI або адміністратора, встановленого на ваших власних серверах, і використовує кластер Nomad для планування.

CircleCI оголосила про розширення партнерських інтеграцій для кращої підтримки команд розробників управління, проектування, тестування та випуску програм для середовищ Kubernetes.

CircleCI використовує Kubernetes для випуску того, що компанія називає «кулями». Orbs — це пакети, що містять програми налаштування, завдання, команди та додатки для побудови та виконання операцій. На рисунку 1.9 показана організація процесу CI.

CircleCI допомагає командам автоматизувати свої збірки, тестувати та випускати дії для розгортання частіше та з більшою впевненістю, особливо в міру збільшення тиску на створення все більшої кількості додатків. Нові інтеграції партнерів включають сфери для AWS, Azure, VMware, Red Hat, Kublr та Helm в рамках Програми партнерських технологій у CircleCI, яка стартувала в листопаді 2018 року.

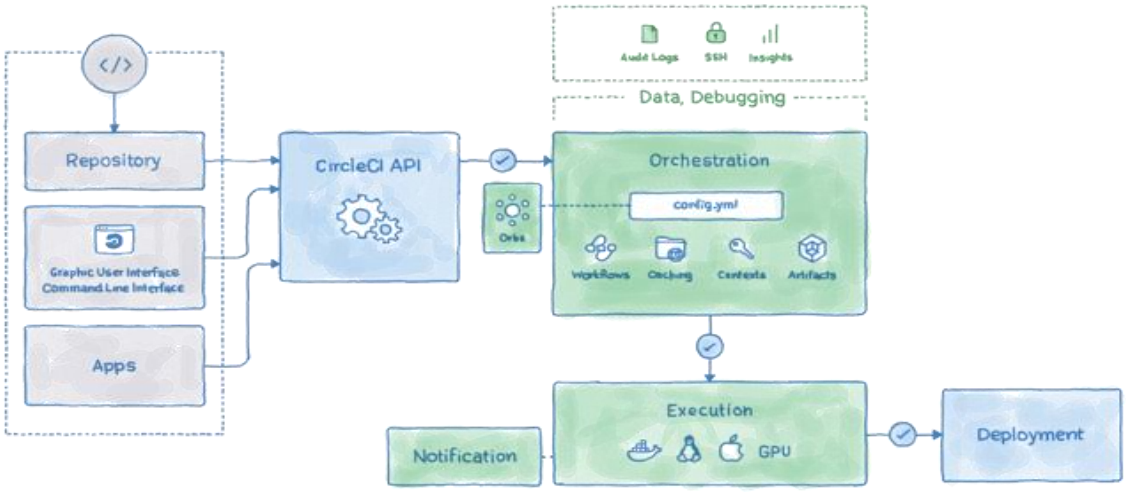

Рисунок 1.9 − Організація процесу CI з CircleCI

CircleCI та інтеграція шарів на базі Kubernetes наших партнерів дозволяють клієнтам легко інтегруватися з реєстрами контейнерів, ефективно управляти розгортанням та забезпечувати середовища Kubernetes за вимогою. Додавання більшої кількості партнерів за допомогою Kubernetes відкрило б більше хмарних середовищ, в яких можна обміняти кулі. Використання програмного забезпечення дає змогу публікувати послуги та одночасно оптимізувати та координувати код як у хмарах AWS, так і в Azure - або, можливо, щось незначне простіше, наприклад, підтримувати окремі регіональні центри обробки даних без зайвих клопотів [7].

GoCD — походить від великих Thoughtworks, що є достатнім свідченням для його можливостей та ефективності. Головною відміннісю GoCD від решти пакетів є його функція Value Stream Map (VSM). Фактично pipeline’и можуть бути з'єднані ланцюгами разом з одним pipeline, що забезпечує «матеріал» для наступного pipeline (рисунок 1.10). Це дозволяє збільшити незалежність різних команд з різними обов'язками в процесі розгортання. Це може бути корисною функцією при впровадженні цього

типу систем у старих організаціях, які мають намір тримати команди окремими, але якщо всі користуватимуться одним і тим же інструментом, це спростить пізніше пошук вузьких місць у VSM та реорганізацію команд або роботу з підвищення ефективності.

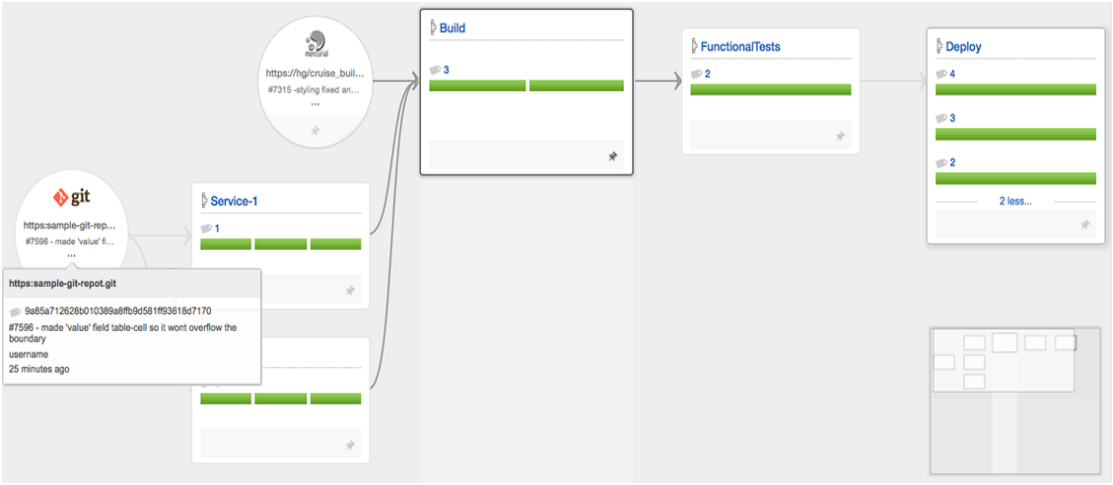

Рисунок 1.10 – З’єднання ланцюгами у GoCD

Неймовірно ціно мати VSM для кожного продукту в компанії; те, що GoCD дозволяє це описати в JSON або YAML у контролі версій і подати візуально з усіма даними, що залежать від часу очікування, робить цей інструмент ще більш цінним для організації, яка намагається краще зрозуміти себе. Почніть із встановлення GoCD і нанесення на карту вашого процесу лише із ручними схваленнями. Потім попросіть кожну команду скористатися схваленнями вручну, щоб ви могли розпочати збір даних про те, де можуть існувати вузькі місця [7].

Travis CI — CI/CD інструмент конвеєри якого зберігаються у форматі YAML з вашим вихідним кодом, і він безперешкодно інтегрується з такими інструментами, як GitHub. Він не тільки може використовуватися як SaaS, але він також має версію, яку можна розмістити. Набагато простіше розгорнути Тревісом CI на Kubernetes за допомогою діаграм Helm.

Якщо ви розробляєте відкритий вихідний код, ви можете безкоштовно використовувати версію SaaS Travis CI. Це зменшує витрати і дозволяє використовувати досить поширену платформу для розробки відкритого вихідного коду, не вимагаючи нічого запускати [8].

Travis Ci підтримує безліч мов програмування серед яких є і ruby (що не дивно, тому що спочатку він розроблявся для ruby-проектів). Почати користуватися сервісом дуже просто.

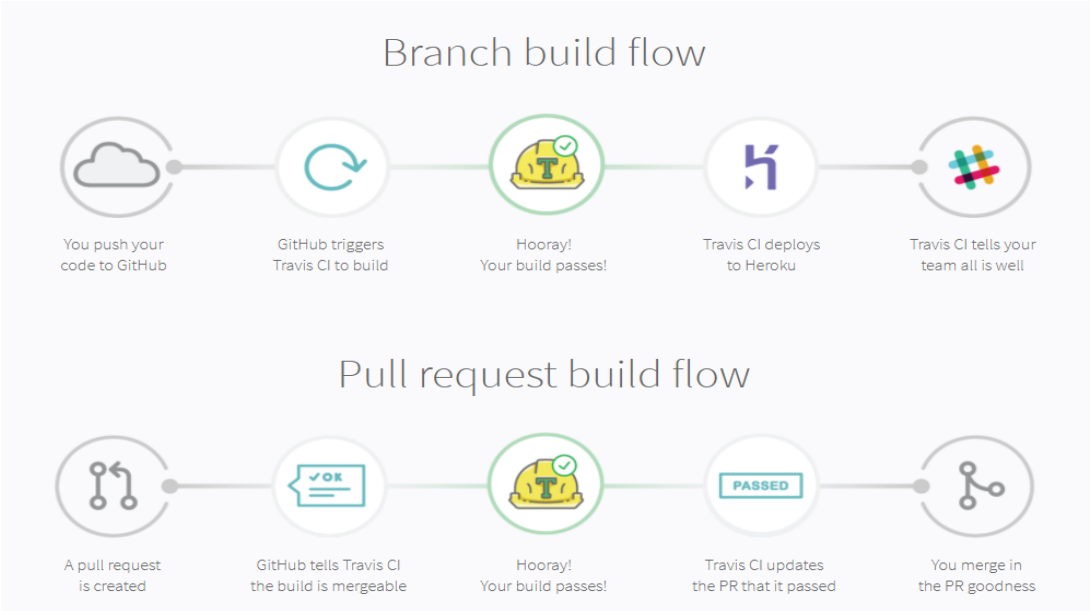

Рисунок 1.11 – Збірка та тестування у TravisCi

Якщо налаштування пройшло успішно, то Travis Ci починає безперервно тестувати проект, відображаючи при цьому поточний статус: червоний колір (виникли проблеми при тестуванні), жовтий (є попередження) і зелений (всі тести пройдені успішно). Крім статусу можна побачити: повідомлення про помилку або попередження, якщо щось пішло не так; останній коміт і його автора; історію збірок і т.д. В цілому інтерфейс досить інформативний і зрозумілий. Крім цього, Travis Ci буде сповіщати

про проблеми по електронній пошті. Процес збірки гілки та збірки з запиту показано на рисунку 1.11.

Jenkins — оригінал, поважний, фактичний стандарт у CI/CD. Конфігурація Jenkins як код (JCasC) повинна допомогти вирішити складні проблеми конфігурації, це дозволить виконати нульову конфігурацію майстрів через файл YAML. Подібно до інших систем CI/CD. Jenkins Evergreen прагне зробити цей процес ще простішим, надаючи заздалегідь визначені конфігурації Jenkins на основі різних випадків використання. Ці дистрибутиви повинні бути простішими в обслуговуванні та модернізації, ніж звичайні дистрибутиви Jenkins.

Jenkins 2 представив нативну конвеєрну функціональність із двома типами конвеєрів. Jenkins X є повною трансформацією Jenkins і, швидше за все, буде реалізацією Cloud Native Jenkins.

Pipeline надає розширюваний набір інструментів для моделювання простих до складних конвеєрів доставки «як код» за допомогою синтаксису специфічної мови домену (DSL) конвеєра. Процес показано на рисунку 1.12.

Визначення конвеєра Jenkins записується у текстовий файл (Jenkinsfile), який, у свою чергу, може бути переданий до сховища джерел керування проектом. Це основа «Pipeline як коду»; обробка конвеєра CD частини програми, яка підлягає версії та перегляду, як і будь-який інший код.

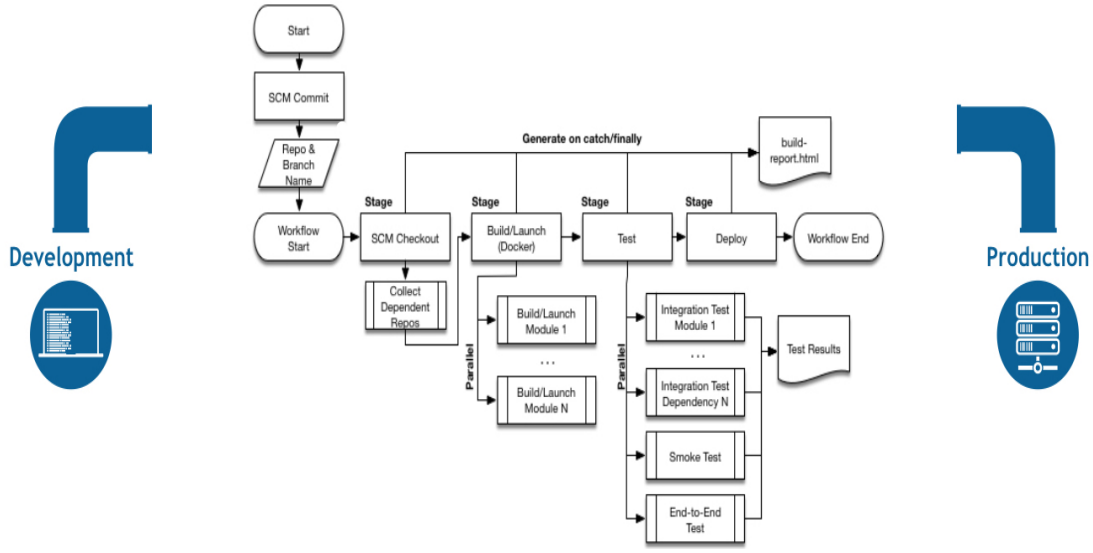

Рисунок 1.12 − Конвеєр Jenkins

Створення Jenkinsfile та передача його до контролю джерел забезпечує ряд переваг:

− автоматично створює процес побудови конвеєра для всіх гілок та запитів витягування;

− перегляд коду та ітерація на конвеєрі (разом із рештою вихідного коду);

− контрольний журнал для Pipeline;

− єдине джерело (single source of truth) для Pipeline, яке можуть переглядати та редагувати багато учасників проекту.

Хоча синтаксис визначення конвеєра, як у веб-інтерфейсі, так і за допомогою a Jenkinsfile, однаковий, як правило, вважається найкращою практикою визначати конвеєр у a Jenkinsfile та перевіряти його в контролі джерела [3].

Concourse CI — Система складається з мікросервісів, і кожне завдання виконується в контейнері. Однією з найкорисніших функцій, якої не мають інші інструменти, є можливість запуску завдання із вашої

локальної системи із вашими локальними змінами. Це означає, що ви можете розробляти локально (за умови, що у вас є підключення до сервера Concourse) і запускати свої збірки так само, як вони будуть виконуватися в реальному конвеєрі збірки. Крім того, ви можете повторно виконати невдалі збірки з вашої локальної системи та вносити певні зміни для перевірки виправлень [9].

Concourse також має просту систему розширення, яка спирається на фундаментальну концепцію ресурсів. По суті, кожна нова функція, яку ви хочете надати своєму конвеєру, може бути реалізована в образі Docker і включена як новий тип ресурсу у вашу конфігурацію. Це зберігає всю функціональність, інкапсульовану в єдиний незмінний артефакт, який можна модернізувати та модифікувати незалежно, і порушення змін не обов'язково повинно порушувати всі ваші збірки одночасно. Нижче показано ці функції та завдання (рисунок 1.13).

Кожне завдання має план побудови, де декларуються вхідні ресурси роботи та що з ними запускати, коли вони змінюються. Потім ваш конвеєр візуалізується у веб-інтерфейсі, в один клік на завдання можна подивитися чому воно не вдалося. Вся конфігурація та адміністрування здійснюється за допомогою flyCLI. Кожне завдання вказує свій власний образ, надаючи йому повний контроль над своїми залежностями, а не керуючи пакетами та станом ваших працівників [9].

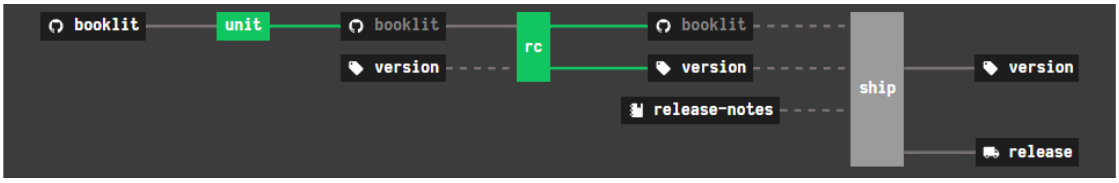

Рисунок 1.13 – Функції та завдання в Concourse

Підхід Concourse до простору CI/CD значно відрізняється від інших інструментів, тому що він намагається вивести себе з рівняння в максимально можливій мірі, мінімізуючи стан і абстрагуючись кожен зовнішній фактор в щось, що він називає «ресурсами». Мета цієї філософії — зробити сервер інтеграції повністю одноразовим, щоб одні й ті ж процеси можна було легко запускати на будь-якому сервері Concourse.

DroneCI — сучасна платформа CI/CD, побудована на основі архітектури, орієнтованої на контейнери. Робочий процес на основі контейнерів лежить в основі дизайну Drone. Drone написаний на Go і був вперше випущений в 2014 році під ліцензією Apache.

Drone виступає в якості середнього координуючого шару між Docker і провайдером сховища. Замість того, щоб запускати сервер CI/CD і потім підключатися до служби хостингу системи управління версіями, Drone вимагає попередньої інформації про обліковий запис сховища для початкового завантаження своїх власних моделей аутентифікації, користувачів і дозволів. Як і всі процеси CI, сам Drone запускається як контейнер. Він підтримує кілька баз даних і постачальників репозиторіїв і має вбудовану підтримку для настройки сертифікатів TLS/SSL з Let's Encrypt для шифрування транспорту [5].

Drone шукає спеціальні YAML-файли [15] в репозиторіях для визначення конвеєра. Синтаксис розроблений так, щоб його можна було легко прочитати і висловити, щоб будь-хто, хто використовує репозиторій, міг зрозуміти процес безперервної інтеграції. Drone надає систему плагінів, але вона використовується не так, як в Jenkins. У Drone плагіни - це спеціальні контейнери Docker [14], використовувані для скидання попередньо налаштованих завдань в звичайний робочий процес (рисунок 1.14). Це полегшує виконання спільних завдань, викликаючи плагін з

декількома параметрами, а не сценарії всього процесу вручну. У цьому сенсі плагіни Drone чимось схожі на службові команди Unix, які призначені для виконання одного вузькоспрямованого завдання. Розроблявся за принципом — «Configuration as a code» [7].

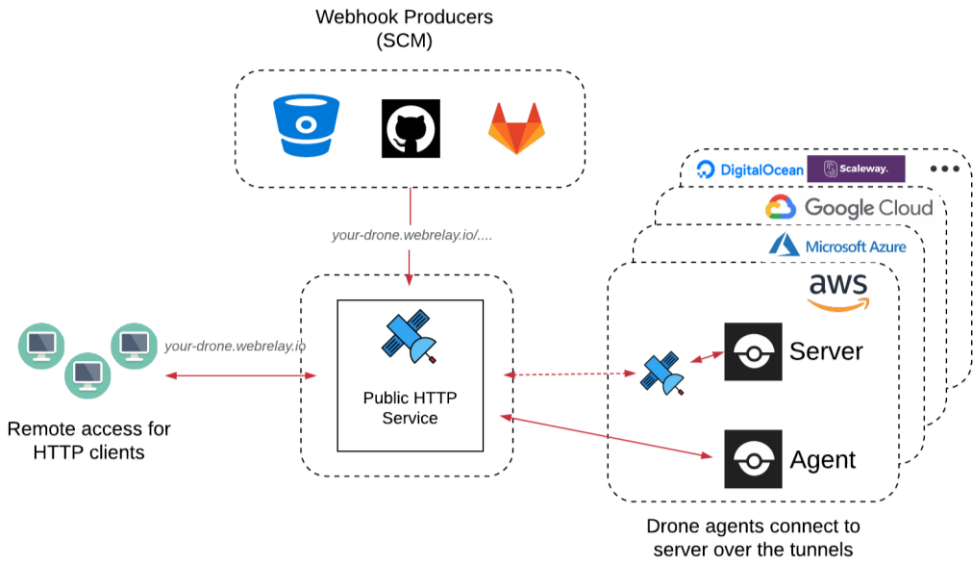

Рисунок 1.14 − Робочий процес DroneCI

Spinnaker — походить від Netflix і більше зосереджений на постійному розгортанні, ніж на постійній інтеграції. Він може інтегруватися з іншими інструментами, включаючи Travis та Jenkins, для запуску тестових конвеєрів та розгортання. Він також має інтеграцію з інструментами моніторингу, такими як Prometheus та Datadog, для прийняття рішень щодо розгортання на основі метрик, що надаються цими системами. Наприклад, розгортання канар Canary використовує концепцію судді та зібрані метрики, щоб визначити, чи не спричинило останнє розгортання якусь деградацію відповідних метрик, і чи слід її відкочувати назад, чи розгортання може продовжуватися (рисунок 1.15).

Кілька додаткових унікальних функцій, пов'язаних з розгортанням, охоплюють область, на яку часто не звертають уваги при обговоренні безперервного розгортання, і може здатися навіть протилежною, але критично важливою для успіху: Spinnaker допомагає зробити безперервне розгортання трохи менш безперервним. Це запобіжить запуску стадії протягом певного часу, щоб запобігти розгортанню під час критичного часу життєвого циклу програми. Він також може застосовувати затвердження вручну, щоб гарантувати, що випуск відбудеться тоді, коли бізнес отримає найбільшу користь від змін. Насправді, суть постійної інтеграції та постійного розгортання полягає в тому, щоб бути готовим до впровадження змін так швидко, як потрібно змінити бізнес [7].

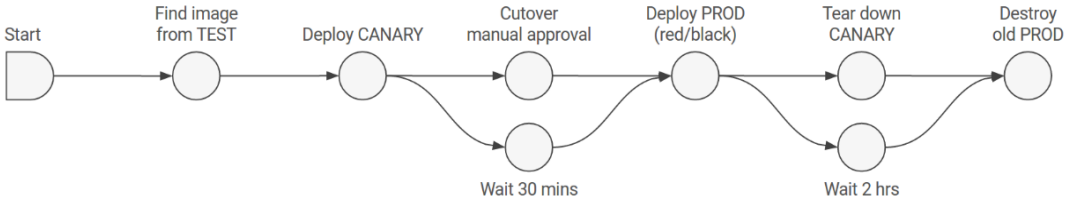

Рисунок 1.15 − Конвеєр Spinnaker

1.8 Попередній аналіз

В сучасному інформаційному середовищі ефективне розгортання додатків є критичною складовою для забезпечення стабільної та масштабованої роботи програмного забезпечення. У контексті оркестрації контейнерів Kubernetes став невід'ємною частиною інфраструктури. Однак, для оптимального використання потужностей Kubernetes, важливо вибрати

належні методи розгортання, включаючи використання менеджера пакетів Helm та CI/CD підходів.

Менеджер пакетів Helm стає важливим інструментом для стандартизації та автоматизації розгортання додатків. Використовуючи Helm charts, розробники можуть упаковувати та поширювати свої додатки, а адміністратори здійснювати конфігураційне управління.

Принципово важливим єтапом - є Continuous Integration, який включає автоматизовану інтеграцію коду в репозитарій. Тестування, збірка та перевірка коду допомагають забезпечити високу якість програмного забезпечення.

Continuous Delivery розширює CI, автоматизуючи процес розгортання в тестове та продуктивне середовище. Використання Helm у CI/CD пайплайнах сприяє автоматизації розгортання Helm charts та управлінню конфігурацією.

Helm може бути інтегрований в CI/CD пайплайни для автоматизації розгортання та конфігурації додатків в Kubernetes. Відбувається пакування Helm charts, їх розгортання та налаштування відповідних ресурсів.

Переваги використання CI/CD та Helm – є високий рівень автоматизації, полегшуючи завдання розробників та DevOps-інженерів.

Розгортання додатків в Kubernetes є складною задачею, яка вимагає вибору належних методів. Використання Helm та CI/CD підходів сприяє стандартизації, автоматизації та ефективному управлінню додатками в середовищі Kubernetes. Це не лише забезпечує надійність розгортання, але й забезпечує шлях до створення масштабованих та стійких до помилок систем.

Після проведення передварительного аналізу предметної області та розгляду сучасних методів та стратегій, які можуть бути використані у процесі розгортання корпоративних додатків в Kubernetes-кластері, можна

сформулювати відповідний висновок. На даному етапі еволюції вибір методів виконується на основі комплексного аналізу численних популярних рішень і врахування вимог до тестового проекту.

Отже, нинішнім викликом є розв'язання завдання удосконалення існуючих методів та розроблення нового, універсального методу, придатного для вирішення проблем при проектуванні CI/CD-інфраструктури з використанням найбільш популярних технологій, стратегій та підходів. Це буде результатом проведеного дослідження.

Метою даного дослідження є вивчення та аналіз можливостей використання методів у процесі проектування CI/CD-інфраструктури, а також модифікація на їх основі існуючого методу розгортання додатків у Kubernetes.

Для досягнення поставленої мети необхідно вирішити такі основні завдання:

− дослідження сучасного стану технологічного ландшафту, пов'язаного з розгортанням Enterprise-додатків у Kubernetes кластері;

− аналіз існуючих підходів і методів, які використовуються для розгортання додатків в середовищі Kubernetes, а також визначення переваг та обмежень кожного з існуючих методів;

− вдосконалення та опис методу, спрямованого на оптимізацію процесу розгортання Enterprise-додатків в середовищі Kubernetes;

− реалізація експерименту з використанням вдосконаленого методу розгортання додатків в реальному або симульованому Kubernetes кластері;

− порівняння результатів з існуючими методами розгортання та висновки щодо вдосконаленого методу розгортання Enterprise-додатків в Kubernetes.

2 ДОСЛІДЖЕННЯ ОБРАНИХ МЕТОДІВ

2.1 Дослідження компонентів проекту

Для ефективної імплементації вибраних рішень науково обгрунтовано рекомендується передбачити аналіз всіх компонентів системи проекту. Проект включає різноманітні сервіси та підсистеми, які можуть бути розподілені на кілька визначених груп. Перша група включає в себе всі моніторингові рішення з візуалізацією, такі як Prometheus, а також аналітична та візуалізаційна платформа Grafana [18]. На рисунку 2.1, зображено взаємодію між окремими компонентами системи моніторингу та візуалізації.

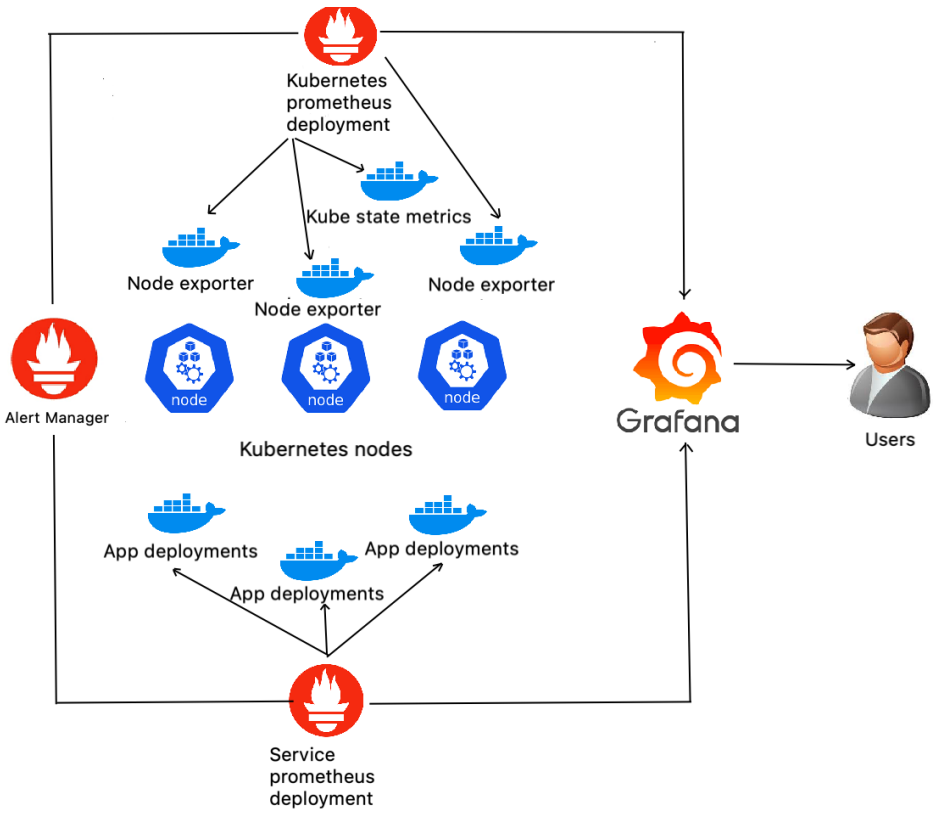

Рисунок 2.1 – Взаємодія користувача з системами моніторингу

До другої групи належать сервіси, що використовуються для розробки аналітичних моделей, які подальше використовуються у бізнес-аналітиці. Цей клас відзначається найбільшою динамікою збільшення

чисельності, оскільки при залученні нових клієнтів може виникати необхідність розробки нових моделей.

Наступною категорією є API [13], що безпосередньо взаємодіють з клієнтами, які постійно потребують доступу до сховища даних. Група інструментів для екстракції даних з первинних джерел у сховище подібна за рівнем взаємодії. Ці дані вимагають постійного завантаження та передварителі обробки.

Остання категорія включає існуючі реляційні та нереляційні сховища даних, такі як Oracle, MySQL, PostgreSQL, MongoDB, а також інструменти взаємодії з ними.

2.2 Підготовка до впровадження CI/CD

Наступним етапом є аналіз існуючих груп сервісів з метою визначення тих, які мають найвищий пріоритет для перенесення в нове середовище. З урахуванням специфіки та принципів використання контейнерів, групу, яка включає бази даних, необхідно залишити в незміненому стані, у той час як інші сервіси варто перенести. Усі ці сервіси можуть бути запущені в окремих контейнерах, забезпечуючи ефективне використання фізичних ресурсів компанії під управлінням Kubernetes. На рисунку 2.2 представлено загальний принцип конфігурації майбутньої системи.

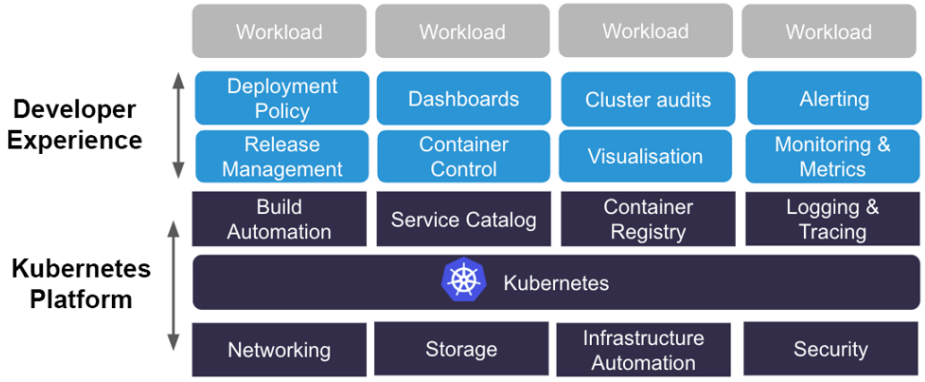

Рисунок 2.2 – Принцип побудови майбутньої системи

У той самий період команда прийняла рішення щодо системи контролю версій для групи сервісів, вибравши GitLab, і висловила пропозицію використовувати відповідні інструменти CI/CD після аналізу. Після уважного огляду всієї системи та ідентифікації використовуваних технологій, виникло питання про доцільність використання інструментів Jenkins та Spinnaker.

Подальше розгортання Spinnaker та Jenkins проведено для групи аналітичних моделей на тестовому середовищі.

Поки адміністратори налаштовували середовища команда розробила діаграму CI/CD (рисунок 2.3).

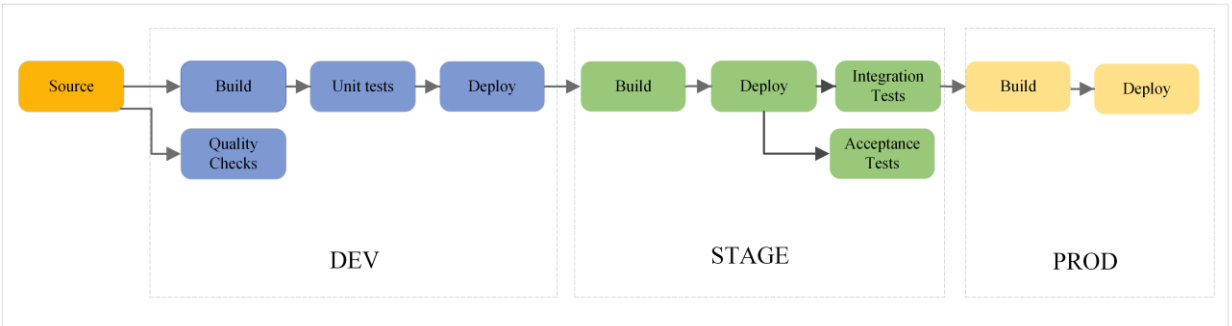

Рисунок 2.3 – Діаграма CI/CD

2.3 Дослідження програмної системи Jenkins

Розглянемо програмну систему CI/CD Jenkins, яка володіє рядом функціональних можливостей. Зокрема, в рамках Jenkins існує підтримка метрик Prometheus, що є однією з інтегрованих систем, а кількість розширень для взаємодії з системами сповіщень виявляється ще більшою.

Нижче подано кілька прикладів застосування Jenkins у тестовому середовищі. На першому графічному зображенні (рисунок 2.4) відображено загальний інтерфейс з побудованими конвеєрами, а також ілюстровано процеси, що виникають під час компонування (рисунок 2.5).

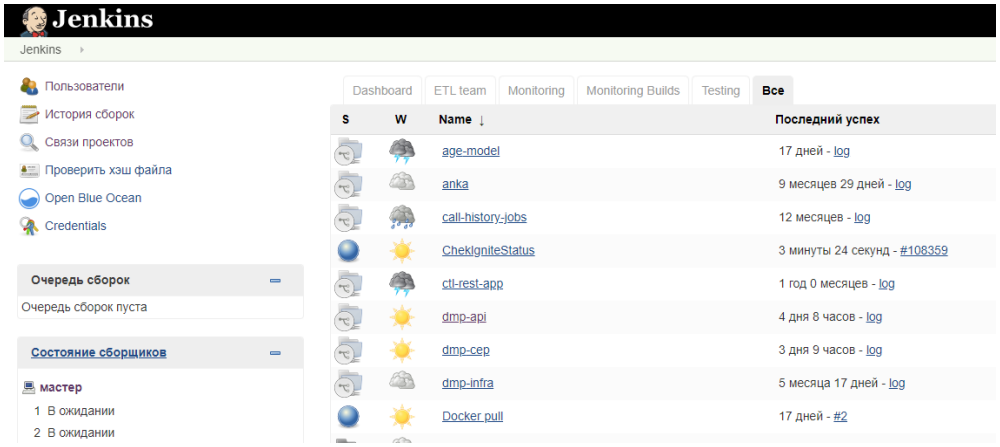

Рисунок 2.4 – Загальний інтерфейс

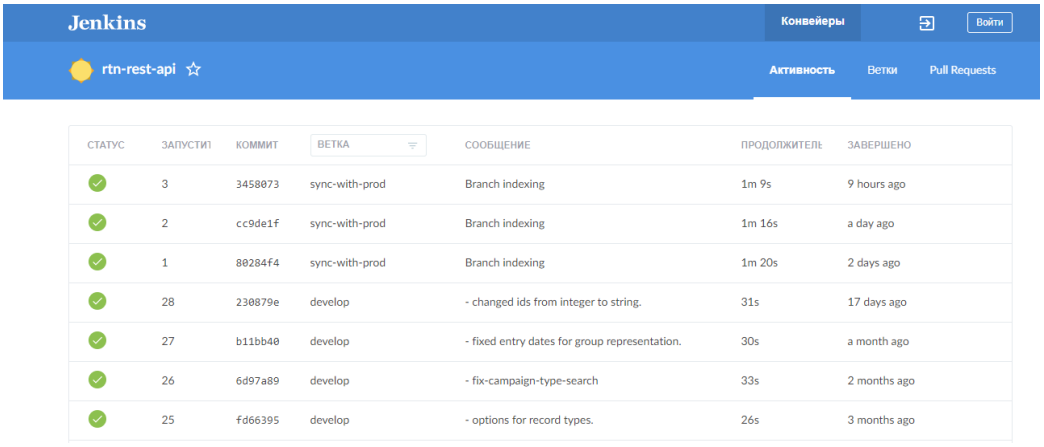

Рисунок 2.5 – Деталізація конвеєру, виконані компонування

Для детальної перевірки конвеєрів можемо зайти у процес та побачити на якому етапі виникла помилка (рисунок 2.6).

Рисунок 2.6 – Помилка у конвеєрі

У той же час, на прикладі відпрацьованого без помилок, можемо детально бачити таку інформацію у окремих меню: етапи конвеєру, зміни, тести та артефакти, зображені на наступних рисунках 2.7 та 2.8.

Рисунок 2.7 – Етапи конвеєру

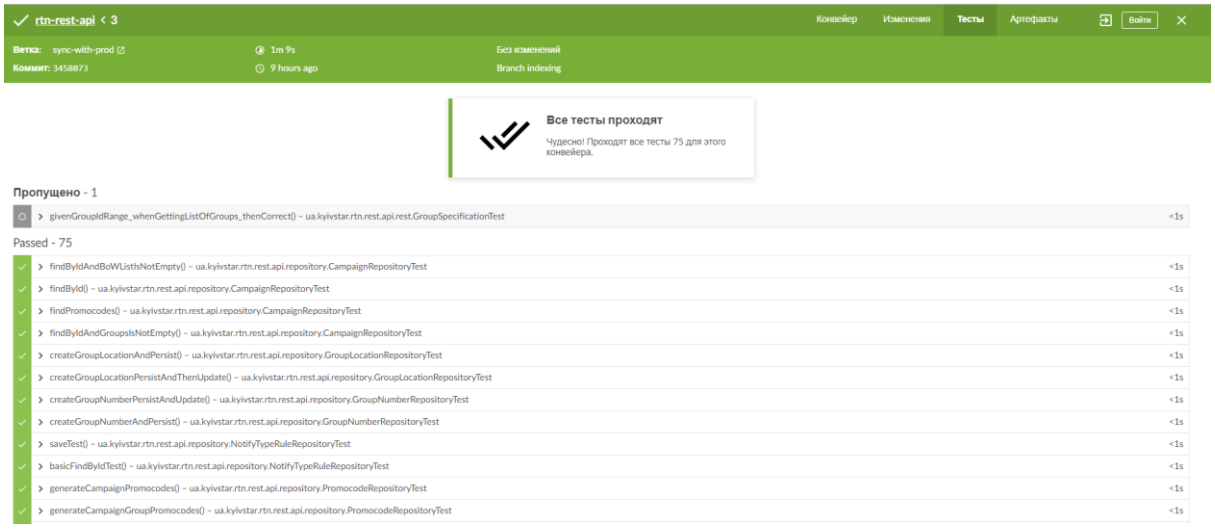

Рисунок 2.8 – Пройдені тести

2.4 Дослідження програмної системи Spinnaker

Розглянемо іншу платформа безперервної доставки з відкритим кодом для швидкого та впевненого випуску змін програмного забезпечення під назвою Spinnaker. Spinnaker був випробуваний у виробництві сотнями команд у мільйонах розгортань. Він поєднує в собі потужну та гнучку систему керування конвеєрами з інтеграцією до основних хмарних постачальників.

Spinnaker нативно підтримує розгортання програм у кількох хмарних провайдерах. Він підтримує такі хмарні платформи, як AWS, Azure, GCP, Oracle Cloud, Cloud Foundry і OpenStack. Також його можна легко інтегрувати з іншими інструментами, такими як:

- Jenkins;
- Travis CI;
- Git event;
- CRON jobs.

Нижче подано кілька прикладів застосування Spinnaker у тестовому середовищі. На першому графічному зображенні (рисунок 2.9) відображено конвеєр який успішно завершився та на другому зображенні (рисунок 2.10) відображено конвеєр що завершився з помилкою.

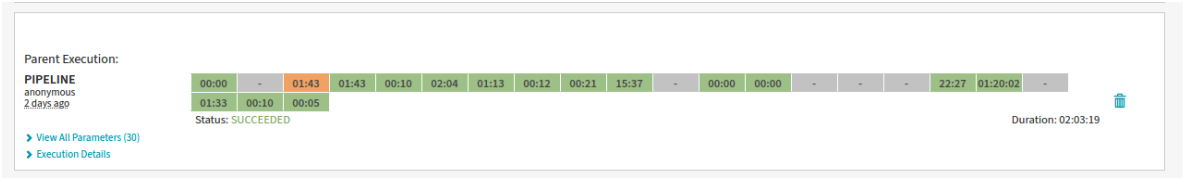

Рисунок 2.9 – Успішно пройдений конвеєр Spinnaker

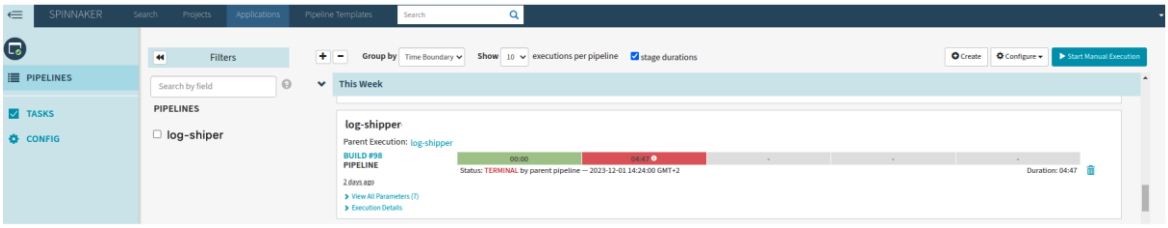

Рисунок 2.10 – Пройдений конвеєр Spinnaker з помилкою

Для детальної перевірки конфігурації конвеєра можемо зайти в налаштування і побачити конфігураційну мапу (рисунок 2.11).

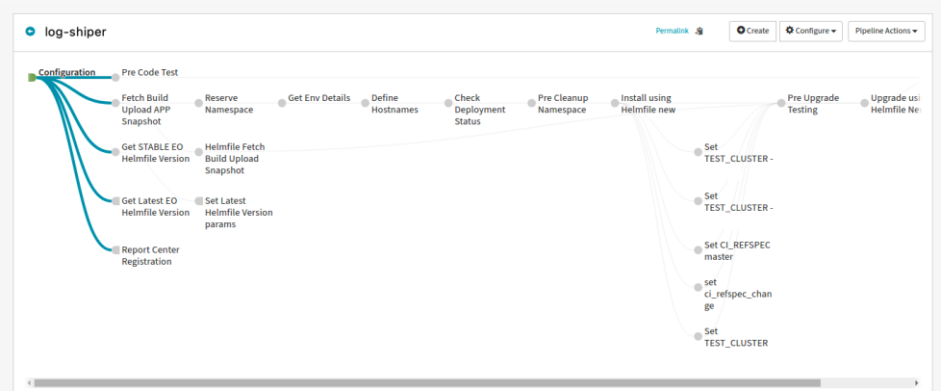

Рисунок 2.11 – Конфігураційна мапа конвеєра

На іншому прикладі відображено інграцію Jenkins та Spinnaker (рисунок 2.12)

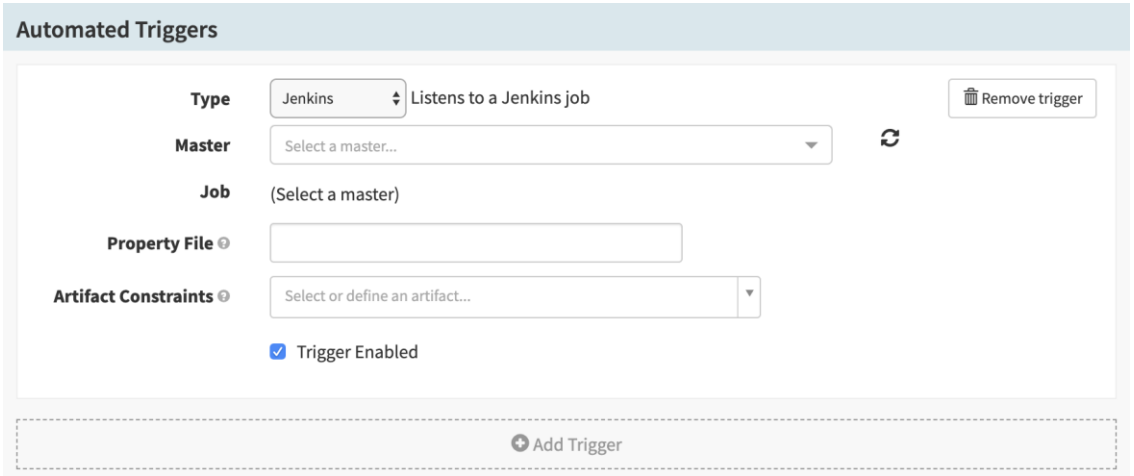

Рисунок 2.12 – Інтеграція Spinnaker та Jenkins

2.5 Дослідження менеджера пакетів Helm

Helm - це інструмент для керування додатками Kubernetes, який дозволяє легко встановлювати, оновлювати та управляти встановленими додатками на Kubernetes кластері. Говорячи Helm, ми насправді маємо на увазі дві програми: клієнтський helm і його серверного, що живе в Kubernetes компаньйона — tiller. На пару вони можуть шукати, встановлювати, апгрейдити та видаляти пакети-додатки, які в хельмівській термінології звані чартами (chart). Helm Chart - це пакетизований інсталяційний дескриптор для Kubernetes.

Helm Charts містять всі необхідні конфігурації та параметри для встановлення конкретного додатку або мікросервісу на Kubernetes (рисунок 2.13). Вони включають в себе опис ресурсів Kubernetes (таких як служби, деплойменти, службові об'єкти тощо), налаштування, значення за замовчуванням, шаблони та інші елементи, необхідні для належної роботи додатку в середовищі Kubernetes.

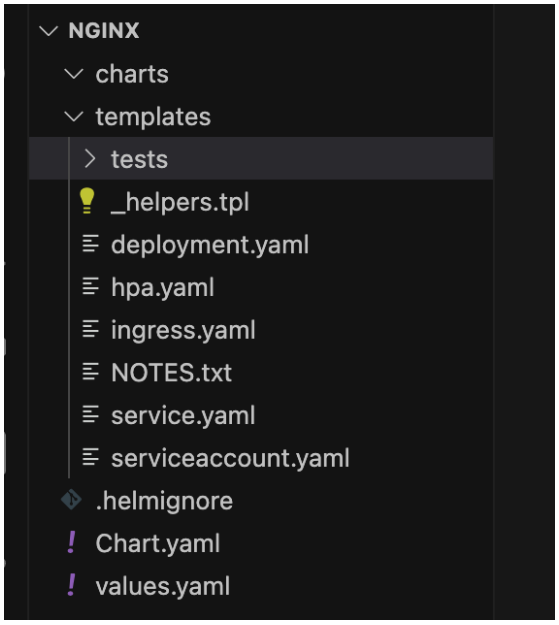

Рисунок 2.13 − Helm Chart

У контексті Helm та розгортання додатків в Kubernetes, важливим елементом є Umbrella Chart. Основна ідея Umbrella Chart полягає в об'єднанні кількох Helm Charts в один комплексний пакет для зручного управління та розгортання пов'язаних додатків (рисунок 2.13). Цей підхід полегшує організацію та управління складними системами, що складаються з різних компонентів чи мікросервісів.

```
my-umbrella-chart/
|-- Chart.yaml
|-- values.yaml
|-- charts/
|   |-- subchart1/
|       |-- Chart.yaml
|       |-- ...
|   |-- subchart2/
|       |-- Chart.yaml
|       |-- ...
|-- templates/
|   |-- _helpers.tpl
|   |-- deployment.yaml
|   |-- service.yaml
|   |-- ...
```

Рисунок 2.13 − Helm Chart

В кореневому файлі Chart.yaml Umbrella Chart визначаються залежності в розділі dependencies. Залежності вказують на інші Helm Charts (підчарти), які входять в склад Umbrella Chart і будуть автоматично встановлені та керовані разом із зазначеним Umbrella Chart (рисунок 2.14).

Chart.yaml > ...

Helm Chart.yaml - The Chart.yaml file is required for a chart (chart.json)

```yaml
apiVersion: v2
name: nginx
description: A Helm chart for Kubernetes
type: application
version: 0.1.0
appVersion: "1.16.0"

dependencies:
- name: secretstore-ingress-keda-apps-helm
  alias: helm-app
  version: 0.0.1
  repository: oci://ghcr.io/bart-jansen
```

Рисунок 2.14 − Helm Chart

2.6 Переваги обраних рішень

Jenkins, як один з піонерських серверів безперервної інтеграції з відкритим вихідним кодом, залишається визнаним лідером у цій галузі. Він проявляє значну гнучкість та володіє інтерфейсом, орієнтованим на користувача, що робить його надзвичайно привабливим для використання.

Природа гнучкості Jenkins може вважатися перевагою або недоліком, в залежності від вимог конкретного проекту. У контексті досліджуваного проекту важливою є наявність графічного інтерфейсу, який сприяє полегшенню розуміння інструменту. Використання різноманітних модулів у Jenkins дозволяє візуалізувати різні види звітів, таких як результати тестів, покриття коду та інші аспекти, що піддавалися статичному аналізу.

Зручність об'єднання та оркестрування завдань в графічному інтерфейсі порівняно спрощена у порівнянні з використанням YAML-файлів. Однак використання файлів YAML для визначення конвеєра збірки вигідно з точки зору чистоти і структурованості. Це дозволяє документувати зміни в конвеєрі прямо в репозиторії як частину вихідного коду. Такий підхід дозволяє використовувати різні файли YAML для різних гілок випуску, забезпечуючи таким чином деталізоване відслідковування конфігурацій для кожної гілки.

Щодо використання Spinnaker у проекті то він надає кілька важливих переваг.

1. Багатоплатформеність. Spinnaker є багатоплатформеним інструментом і підтримує різні хмарні платформи, такі як AWS, Google Cloud, Microsoft Azure, і Kubernetes. Це робить його ідеальним вибором для проектів, які використовують різні хмарні інфраструктури або комбінування хмарних та локальних обчислень.

2. Неперервна доставка та Розгортання (CD). Spinnaker спрощує процес неперервної доставки і розгортання, надаючи можливість автоматизувати весь

цей цикл від створення пакету до його розгортання в середовищі виробництва. Це призводить до швидших та більш безпечних циклів розробки.

3. Безпечні розгортання. Spinnaker дозволяє використовувати різноманітні стратегії розгортання, такі як розгортання поетапне (Canary), а / б тестування та стратегії відновлення після збоїв. Це допомагає знижувати ризики та забезпечує більш безпечні випуски програмного забезпечення.

4. Візуалізація процесу доставки. Spinnaker надає інтуїтивний графічний інтерфейс, який дозволяє візуалізувати весь процес доставки програмного забезпечення, починаючи від збірки та завершуючи розгортанням. Це полегшує відслідковування та аналіз кожного етапу процесу.

5. Інтеграція з іншими інструментами. Spinnaker легко інтегрується з іншими популярними інструментами для розробки, такими як Jenkins, Git, і Slack. Це дозволяє побудувати потужний стек інструментів для автоматизації розробки та доставки.

6. Розширюваність. Spinnaker є розширюваним інструментом, який можна легко адаптувати до конкретних потреб проекту. Він підтримує створення власних розширень та плагінів, що дозволяє забезпечити відповідність конкретним вимогам та інфраструктурі.

В цілому, використання Spinnaker дозволяє підняти ефективність розробки та надає інструменти для управління та автоматизації процесу доставки програмного забезпечення в будь-яких областях розгортання.

Використання Helm на проекті має кілька ключових переваг.

1. Автоматизація розгортання та керування конфігурацією. Helm дозволяє автоматизувати розгортання додатків та їхню конфігурацію за допомогою чітко визначених пакетів (charts). Helm Chart забезпечує упорядковану структуру для організації та управління конфігурацією додатків.

2. Стандартизація та повторне використання. Helm дозволяє стандартизувати конфігурацію та повторно використовувати її для різних середовищ (розробка, тестування, виробництво). Також Helm Chart має

стандартизовану структуру, що дозволяє легко обмінюватися та використовувати готові рішення.

3. Модульність та розширюваність. Helm дозволяє створювати та використовувати різні Helm Chart для різних компонентів додатку та легко розширювати чи модифікувати для відповіді специфічним потребам проекту.

4. Керування залежностями. Вбудована система керування залежностями дозволяє ефективного управляти залежностями між додатками. Helm Chart може містити залежності, що полегшує управління іншими Helm Chart або ресурсами.

5. Версіонування та історія релізів. Helm забезпечує можливість версіонування Helm Charts та простий спосіб управління історією релізів. Також Helm має підтримку версіонування, що дозволяє легко відслідковувати та повертатися до певних версій конфігурацій.

6. Umbrella Chart для управління багатьма Helm Charts. Umbrella Chart дозволяє об'єднувати кілька Helm Charts для керування їхнім розгортанням як єдиною системою. Вищезазначене дає можливість складний додаток організовувати у вигляді підпроектів, де кожен Helm Chart відповідає окремому компоненту додатку.

7. Спрощення мікросервісної архітектури. Використання Umbrella Chart може спрощувати керування та розгортання мікросервісної архітектури, де кожен Helm Chart відповідає окремому сервісу.

Загальною перевагою є те, що Helm, Helm Chart та Umbrella Chart сприяють створенню більш структурованих, автоматизованих та легко управляються конфігурацій для розгортання додатків на різних стадіях розробки та виробництва.

3 ВДОСКОНАЛЕННЯ МЕТОДУ ВПРОВАДЖЕННЯ КОРПОРАТИВНИХ ДОДАТКІВ У КЛАСТЕРІ KUBERNETES (K8S)

3.1 Підвищення Ефективності Jenkins, Spinnaker і Helm

Покращенням у проекті з обраними рішеннями є спроба впровадження спільного рішення Jenkins, Spinnaker та Helm. Спочатку це може здатися трохи надлишковим, але поєднання Jenkins, Spinnaker та Helm — це досить потужний інструмент.

Helm керує пакунками в середовищі Kubernetes, призначеним для темплейтування конфігурацій та організації структури саб чартів для ефективного розгортання мікросервісів. Його функціональність включає механізми темплейтінгу, які дозволяють динамічно генерувати конфігурації, структуру саб чартів для логічного організованого управління, версіювання для відстеження змін та забезпечення сумісності версій, а також збереження конфігурацій у вигляді значень, що полегшує гнучке конфігурування середовищ розгортання мікросервісів в середовищі Kubernetes.

Jenkins виконує завдання і полегшує діалог зі сторонніми інструментами. Ще одна перевага такої конструкції — слабке з'єднання між інструментами:

− можна замінити будь-який з компонентів фабрики збірки без необхідності переробляти весь процес CI/CD;

− можна мати гетерогенну середу збірки, що об'єднує (можливо, декілька) рішень, наприклад, Jenkins, TeamCity, тощо, і при цьому має один інструмент моніторингу.

Spinnaker можна розглядати як надстройку над Jenkins в тому сенсі, що вони можуть взаємодіяти та доповнювати один одного для створення повноцінного та потужного процесу CI/CD. Spinnaker спеціалізується на розгортанні з використанням різноманітних стратегій, таких як Canary, Blue/Green, і Automated Rollback та забезпечує оркестрацію і керування комплексними процесами розгортання. Комбінація Spinnaker і Jenkins дозволяє

отримати вигоди обох інструментів, використовуючи Jenkins для локальних CI/CD процесів та Spinnaker для розгортання та управління завданнями у різних хмарних середовищах.

Хоча поєднання Helm, Jenkins і Spinnaker може стати потужним інструментом для керування розгортанням та неперервною інтеграцією в Kubernetes, існують деякі потенційні недоліки та виклики при їхньому спільному використанні:

− складність конфігурації;

− потрібність великої експертизи;

− інтеграційні проблеми;

− великий обсяг конфігурації;

− швидкість та ефективність.

Було розроблено рішення інтеграції інструментів CI/CD — Spinnaker + Jenkins та Helm для конфігурацій середовищ і впровадження рішення на проекті.

3.2 Формалізований опис вдосконаленого методу впровадження корпоративного додатку в кластер Kubernetes

Розгортання нової версії програмного продукту, по суті, визначаються конвеєром CI/CD. Кожен крок у безперервній доставці - від об’єднання змін коду до доставки придатних для виробництва збірок - передбачає автоматизацію тестування та автоматизацію випуску коду.

Оскільки корпоративний додаток включає в себе велику кількість мікро сервісів то в методі пропонується використовувати наступний підхід автоматизації розгортання додатку після злиття розробником в головну вітку компонента додатка. Після того як код зберігається в центральному сховищі та

надішле фіксацію коду автоматично запускається конвеєр CI/CD який включає такі етапи.

1. Quality Checks. Коли розробник створює код і публікує його в репозиторії, системі негайно буде запропоновано розпочати подальший процес аналізу коду. Код повинен перевіритись на наявність статичних політик.

2. Build. Для компіляції потрібно налаштувати контейнер докерів як агенти збірки. Потім інструмент CI/CD витягує останні зміни коду.

Компіляція та створення виконуваних файлів або пакетів переноситься до сховища (наприклад AWS S3) для зберігання.

3. Тестування. Під час тестування виконуються кілька типів тестів CI:

– перевірка якості коду може бути або не бути, залежно від того, коли формальний процес CI починається;

– Unit Tests є фундаментальними тестами які виконується, коли додаються, розробляються або нові функції;

– Integration Tests - це міжмодульне тестування додатку буде основним акцентом інтеграційного тестування в контекст постійної інтеграції;

– Acceptance Tests - це тестування, щоб переконатися, що програмне забезпечення відповідає вимогам, встановленим на етапі його розробки.

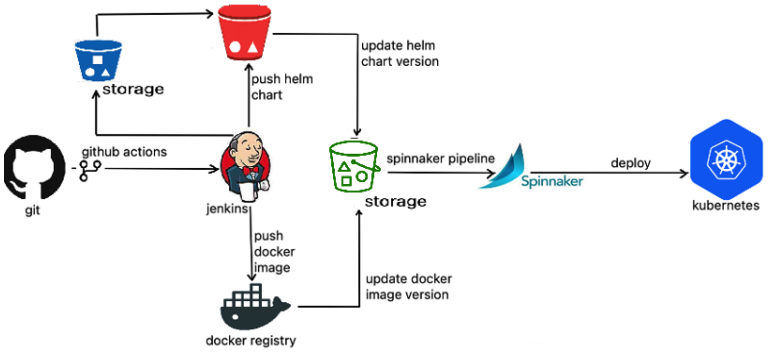

Рисунок 3.1 – Блок схема робочого процесу CI/CD

4. Завантаження Docker Image і Helm Chart до сховища (рисунок 3.1). Створенні на попередньому етапі виконувані файли та пакети збираються в

Docker Image та доставляється сховища. На основі нового Docker Image збирається нова діаграма компонента (Helm Chart) та завантажується в сховище (наприклад AWS S3) та збирається діаграма верхнього рівня (Umbrella Chart) та завантажується в сховище.

5. Розгортання. Розгортання додатку включає в себе створення ізольованих середовищ - Dev, Stage, Prod. Конфігурація середовища Dev повинна мати мінімальні значення ресурсів що необхідні тільки для запуска додатку. Мета Dev середовища дозволити розробникам програмного забезпечення пакетно змінювати код разом і розгортати їх за допомогою CD у віддаленому середовищі розробника. Коли розробник об'єднує гілку функції (Feature Branch) та головну вітку (Master), інженери переміщують код у Stage, використовуючи свій конвеєр розгортання. Перш ніж проблеми з кодом стануть нашими клієнтами, це останній шанс виявити помилки, зниження продуктивності та ризики безпеки. Середовища Prod та Stage повинні бути максимально схожими, так як в Stage середовищі проходить складне тестування що допомагає виявити та вирішити будь-які труднощі, які можуть виникнути під час зміни версії. Після успішного тестування створюється нова версія додатку, яка потім впроваджується в PROD середовищі.

Життєвий цикл розгортання системи визначено в контексті трьох основних середовищ: розробка (DEV), етап тестування (STAGE) та експлуатація (PROD), що наведений на рисунку 3.2.

Рисунок 3.2 – Життєвий цикл розгортання додатка

Вдосконалений метод автоматизації розгортання корпоративного додатку в кластер Kubernetes можна представити у вигляді схеми яка наведена на

рисунку 3.3, що включає контейнери Docker, Сховище, конвеєр CI/CD та ізольовані середовища.

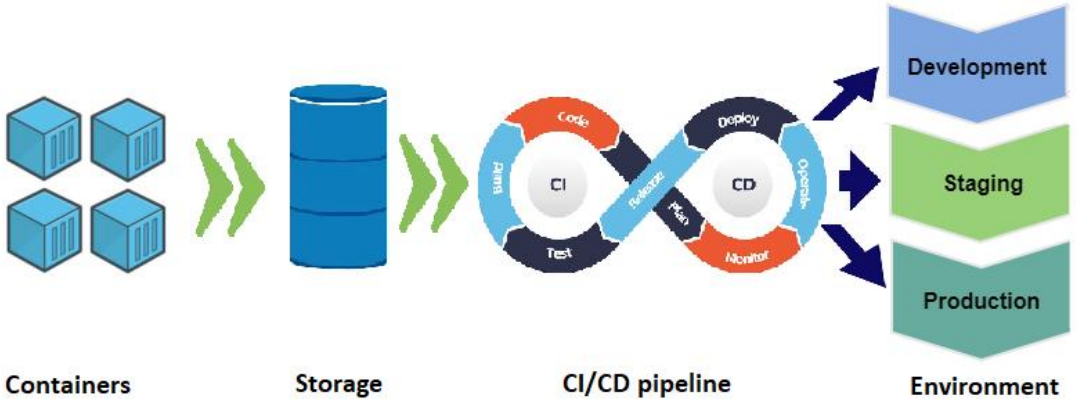

Рисунок 3.3 − Розгортання додатку в середовища

Контейнери Docker використовуються як агенти збірки для компіляції і створення виконуваних файлів або пакетів мікросервісів. Сховище AWS S3 використовується для зберігання збірок. Конвеєр CI/CD запускається після злиття розробником в головну вітку компонента додатка. Ізоловані середовища використовуються для розгортання додатку в різних етапах життєвого циклу розробки і розгортання. Ізоловані середовища дозволяють зменшити ризик впливу невдалого розгортання на продуктивне середовище.

Для створення математичної моделі методу розгортання Enterprise-додатку в кластер Kubernetes за допомогою автоматизованої поставки (CI/CD), можна сформулювати цей процес у вигляді математичних виразів та рівнянь.

Вхідні параметри:

G - Git-репозиторій з вихідним кодом додатку;

C - Кластер Kubernetes, розділений на окремі середовища: DEVELOPMENT (*Cdev*), STAGE (*Cstage*), PRODUCTION (*Cprod*);

J - Jenkins сервер;

D - Docker образ додатку;

U - Сховище (реєстр) Docker образів;

H - Helm чарти;

S - Spinnaker для розгортання в кластерах *Cdev*, *Cstage* і *Cprod*.

Етапи в Jenkins:

- з Git *G* тягнемо код *Csource=GitPull(G);*
- збираємо код в Docker образ *D*=Build(*Csource*);
- Юніт-тести *Tunit*=RunUnitTests(*Csource*);
- Acceptance тести *Tacceptance*=RunAcceptanceTests(*Csource*);
- Integration тести *Tintegration*=RunIntegrationTests(*Csource*);
- Quality тести *Tquality*=RunQualityTests(*Csource*).

Створення Helm чартів:

- створення Helm Chart для додатку *Happ*=CreateHelmChart(*Csource*);
- створення Umbrella Chart *Humbrella*=CreateUmbrellaChart(*Happ*).

Розгортання в Kubernetes за допомогою Spinnaker:

- оновлення версій чартів (*Happ,Humbrella*);
- завантаження Helm чартів в сховище Docker образів UploadCharts(*Happ,Humbrella,U*).

Вибір середовища:

DeployToK8s(*Happ,Cdev,S*) - для середовища DEVELOPMENT;

DeployToK8s(*Happ,Cstage,S*) - для середовища STAGE;

DeployToK8s(*Humbrella,Cprod,S*) - для середовища PRODUCTION.

Математичні Вирази:

Csource, D, Tunit, Tacceptance, Tintegration, Tquality, Happ, Humbrella, U, S - змінні, що представляють стан системи на кожному кроці;

GitPull(*G*), Build(*Csource*), RunUnitTests(*Csource*), RunAcceptanceTests(*Csource*), RunIntegrationTests(*Csource*), RunQualityTests(*Csource*), CreateHelmChart(*Csource*), CreateUmbrellaChart(*Happ*), UploadCharts(*Happ,Humbrella,U*), DeployToK8s(*H,C,S*) - функції, що визначають перехід в новий стан системи.

Умови:

- успішне виконання кожного кроку забезпечується виконанням

відповідних тестів і створенням необхідних артефактів (Docker образів, Helm чартів);

− розгортання в кластерах *Cdev, Cstage, Cprod* може бути здійснене після успішного завершення всіх попередніх кроків.

Вдосконалений метод має наступні переваги:

− автоматизоване тестування кожного мікросервісу дозволяє підвищити надійність і безпеку розгортання, а також зменшити час, необхідний для впровадження нових версій;

− використання контейнерів Docker як агентів збірки дозволяє спростити і автоматизувати процес збірки мікросервісів;

− використання сховища AWS S3 для зберігання збірок дозволяє забезпечити надійне і масштабоване зберігання збірок;

− розгортання додатку в ізольованих середовищах дозволяє зменшити ризик впливу невдалого розгортання на продуктивне середовище.

Час впровадження нових версій корпоративного додатку можна виміряти за допомогою наступної формули:

$$T = T1 + T2 + T3 + T4 , \quad (3.1)$$

де T - загальний час впровадження нових версій;

T1 - час, необхідний для виконання етапу Quality Checks;

T2 - час, необхідний для виконання етапу Build;

T3 - час, необхідний для виконання етапу Testing;

T4 - час, необхідний для виконання етапу Deployment.

Ефективність вдосконаленого методу можна визначити за допомогою наступних формул:

$$E1 = (T1 - T1') / T1 , \quad (3.2)$$

де E1 - коефіцієнт ефективності в частині зменшення часу впровадження нових версій;

T1 - час впровадження нових версій за допомогою традиційного методу;

T1' - час впровадження нових версій за допомогою вдосконаленого методу.

$$E2 = (N1 - N1') / N1 , \quad (3.3)$$

де E2 - коефіцієнт ефективності в частині зменшення кількості помилок, виявлених під час розгортання;

N1 - кількість помилок, виявлених під час розгортання за допомогою традиційного методу;

N1' - кількість помилок, виявлених під час розгортання за допомогою

$$E3 = (R' - R) / R , \quad (3.4)$$

де E3 - коефіцієнт ефективності в частині підвищення надійності розгортання;

R' - надійність розгортання за допомогою вдосконаленого методу;

R - надійність розгортання за допомогою традиційного методу.

$$E4 = (S' - S) / S , \quad (3.5)$$

де E4 - коефіцієнт ефективності в частині підвищення безпеки розгортання;

S' - безпека розгортання за допомогою вдосконаленого методу;

S - безпека розгортання за допомогою традиційного методу.

4 РОЗРОБКА ПРОГРАМНОЇ СИСТЕМИ

Для розробки ІС було обрано популярний провайдер хмарних обчислень – Amazon Web Services. Для розгортання кластеру буде використаний Elastic Kubernetes Service та Amazon Simple Storage Service (Amazon S3). Для збереження коду сервісів буде використовуватися система контролю версій GitHub. Інструменти CI/CD Jenkins та Spinnaker разом з SonarQube будуть встановлені на тойже кластер що і додаток.

4.1 Elastic Kubernetes Service від Amazon Web services

Сервіс Amazon Elastic Kubernetes (Amazon EKS) представляє собою керовану платформу, яка спрощує впровадження Kubernetes в середовищі AWS, без необхідності вручну налаштовувати або підтримувати власний панель управління Kubernetes. Amazon EKS ефективно керує екземплярами площини управління Kubernetes в різних зонах доступності для забезпечення високої доступності. Система автоматично виявляє та замінює несправні екземпляри площини управління, а також надає автоматичне оновлення версій та виправлення для них. Amazon EKS інтегрований з багатьма послугами AWS для забезпечення масштабованості та безпеки для програм, включаючи наступне (рисунок 4.1):

- Amazon ECR для зображень контейнерів;
- Elastic Load Balancing для розподілу навантаження;
- IAM для аутентифікації;
- Amazon VPC для ізоляції.

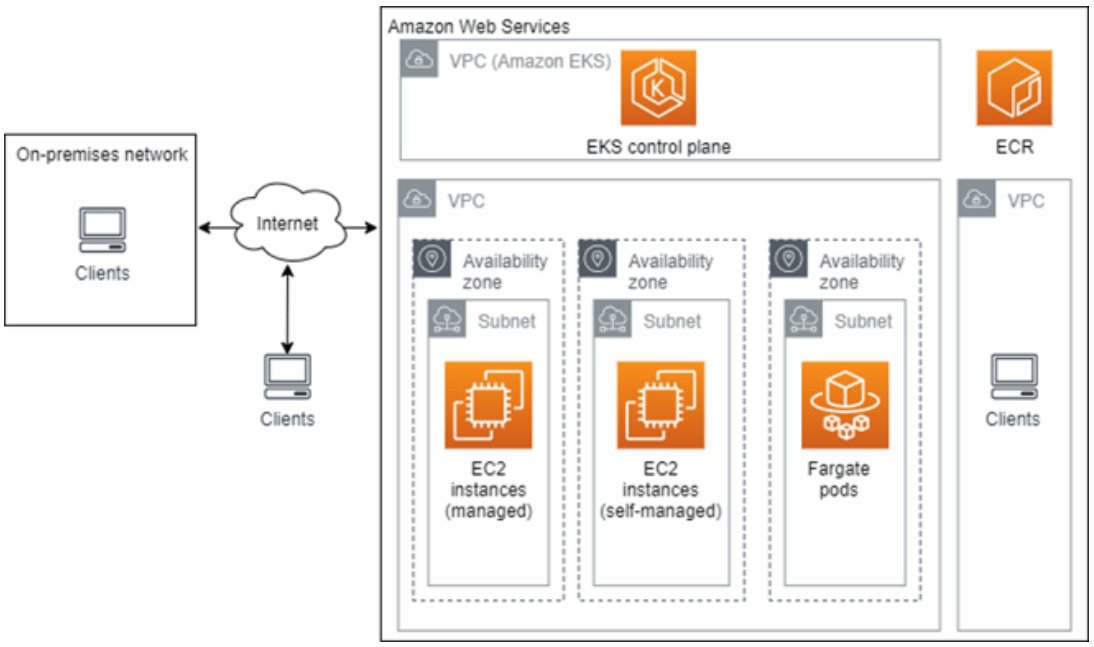

Рисунок 4.1 – Взаємодія EKS з іншими AWS сервісами

Amazon EKS розгортає актуальні версії відкритого програмного забезпечення Kubernetes, надаючи можливість використовувати всі існуючі плагіни та інструменти від громадськості Kubernetes. Прикладні програми, які працюють в середовищі Amazon EKS, повністю сумісні з програмами, що функціонують у будь-якому стандартному контексті Kubernetes, незалежно від їх розташування у локальних центрах обробки даних або в загальнодоступних хмарових середовищах. Це означає, що перенесення будь-якої стандартної додаткової програми Kubernetes до Amazon EKS можливе без будь-якої необхідності внесення змін у вихідний код.

Amazon EKS управляє єдиною панеллю управління Kubernetes для кожного кластера, і контрольна інфраструктура не розділяється між кластерами чи обліковими записами AWS. Ця контрольна панель складається принаймні з двох вузлів сервера API та трьох вузлів etcd, які операційно працюють через три зони доступності в межах регіону. Amazon EKS автоматично виявляє та замінює несправні екземпляри контрольної панелі, перезавантажуючи їх у зонах доступності за потребою.

Amazon EKS використовує архітектуру AWS Regions для забезпечення високої доступності. Мережеві політики Amazon VPC використовуються для

обмеження трафіку між компонентами контрольної панелі в межах одного кластера. Компоненти контрольної панелі для кластера не можуть переглядати або встановлювати зв'язок з іншими кластерами чи обліковими записами AWS, за винятком випадків, коли це дозволено політикою Kubernetes RBAC.

Для створення кластера EKS було використано CloudFormation. Конфігурація:

```
---
  AWSTemplateFormatVersion: '2010-09-09'

    EKSVersion:
      Type: String
      Default: 1.26
      AllowedValues:
        - 1.27
        - 1.28
      Description: The desired version of your AWS EKS Cluster.

    EKSNodeGroupName:
      Type: String
      Default: NodeGroup01
      Description: The desired name of your AWS EKS Node Group.

    EKSDesiredWorkerNode:
      Type: Number
      Default: 2
      Description: Number of desired Worker Node.
      MinValue: 1
      MaxValue: 5

    EKSWorkerNodeInstanceType:
      Type: String
      Default: t3.medium
      AllowedValues: [t3.medium, t3.large]
      ConstraintDescription: Must be a valid EC2 instance type
      Description: EC2 instance type for the node instances.

    VpcBlock:
      Type: String
      Default: 10.0.0.0/16
      Description: The CIDR range for the VPC. This should be a valid private (RFC
1918) CIDR range.
      AllowedPattern: (\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/(\d{1,2})
      ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

    PublicSubnet01Block:
      Type: String
```

```
    Default: 10.0.0.0/24
    Description: CidrBlock for public subnet 01 within the VPC.
    AllowedPattern: (\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/(\d{1,2})
    ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

  PublicSubnet02Block:
    Type: String
    Default: 10.0.1.0/24
    Description: CidrBlock for public subnet 02 within the VPC.
    AllowedPattern: (\d{1,3})\.(\d{1,3})\.(\d{1,3})\.(\d{1,3})/(\d{1,2})
    ConstraintDescription: must be a valid IP CIDR range of the form x.x.x.x/x.

  AvailabilityZonePublicSubnet01:
    Type: CommaDelimitedList<AWS::EC2::AvailabilityZone::Name>
    Default: eu-west-3a
    Description: Availability Zone for the Public Subnet 01.

  AvailabilityZonePublicSubnet02:
    Type: CommaDelimitedList<AWS::EC2::AvailabilityZone::Name>
    Default: eu-west-3b
    Description: Availability Zone for the Public Subnet 02.
```

Створений EKS кластеру наведено на рисунку 4.2.

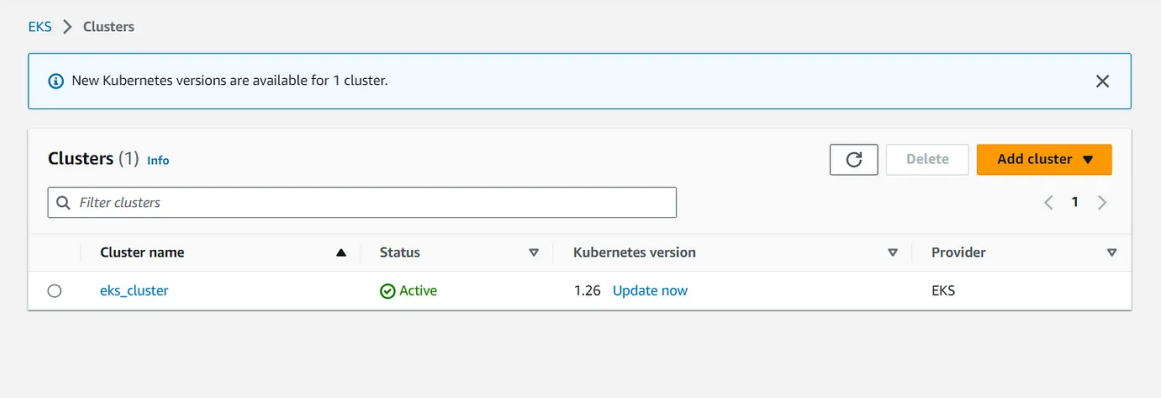

Рисунок 4.2 – Створений EKS кластер

4.2 Amazon Simple Storage Service від Amazon Web services

Amazon Simple Storage Service (Amazon S3) від Amazon Web Services (AWS) є ідеальним рішенням для збереження чартів. Цей сервіс облачного

сховища надає надійний, масштабований та доступний механізм для збереження даних, включаючи графічні чарти.

Завдяки Amazon S3 можна створити віртуальний сховище для чартів, де кожен чарт може бути представлений як об'єкт (файл) у відповідному буці. Amazon S3 забезпечує високий рівень доступності і долі, а також автоматичні засоби резервного копіювання. Крім того, можна налаштовувати правила доступу для забезпечення безпеки та контролю доступу до збережених чартів.

Використання Amazon S3 для збереження чартів надає зручність і масштабованість, що робить його ефективним рішенням для управління та забезпечення доступу до графічних чартів в середовищі хмарних технологій.

Для створення Amazon S3 було використано CloudFormation. Конфігурація:

```
AWSTemplateFormatVersion: "2010-09-09"
Description: "Creates S3 bucket"

Parameters:
    S3BucketName:
        Description: Enter S3 bucket name
        Type: String
        MinLength: 3
        MaxLength: 10

Resources:
    S3Bucket: # logical id/name
        Type: AWS::S3::Bucket # type of resource
        Properties:
            BucketName: !Ref S3BucketName
Outputs:
    BucketName:
        Value: !Ref 'S3Bucket'
```

Створений Amazon S3 bucket наведено на рисунку 4.3.

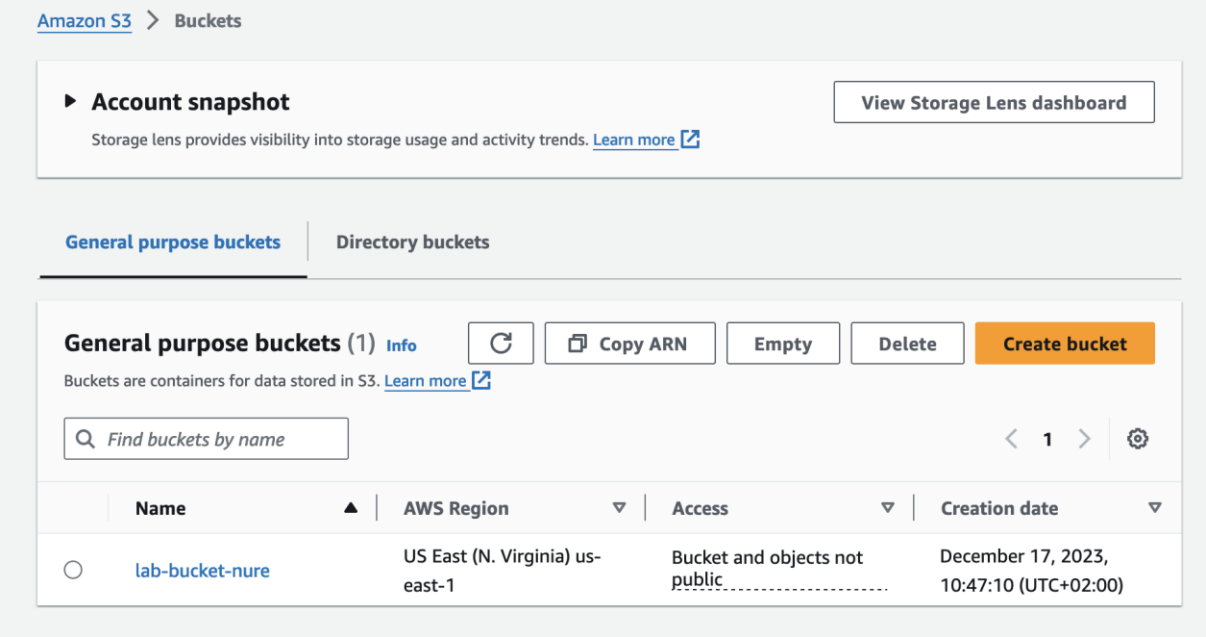

Рисунок 4.3 – Amazon S3 bucket

4.3 Розгортання Jenkins та Spinnaker в AWS

Розгортання систем автоматизації розробки та розгортання, таких як Jenkins та Spinnaker, в середовищі Amazon Web Services (AWS) на EC2 інстансах, надає ефективний інструментарій для створення та керування процесами Continuous Integration (CI) та Continuous Deployment (CD). Розглянемо кроки їхнього початкового розгортання та налаштування.

Обирання типу EC2 інстансів для розгортання Jenkins та Spinnaker в AWS залежить від ряду факторів, таких як розмір вашого проекту, обсяг обробки даних, пам'ять, обчислювальна продуктивність та інші вимоги. Деякі типи EC2 інстанси наведені на рисунку 4.4.

Instance	vCPU*	CPU Credits/hour	Mem (GiB)	Storage	Network Performance (Gbps)***
t3.nano	2	6	0.5	EBS-Only	Up to 5
t3.micro	2	12	1	EBS-Only	Up to 5
t3.small	2	24	2	EBS-Only	Up to 5
t3.medium	2	24	4	EBS-Only	Up to 5
t3.large	2	36	8	EBS-Only	Up to 5
t3.xlarge	4	96	16	EBS-Only	Up to 5
t3.2xlarge	8	192	32	EBS-Only	Up to 5

Рисунок 4.4 – Типи EC2 інстансів

Створення EC2 інстанса для Jenkins сервера включає такі кроки як: вибір AMI, тип інстансу, налаштування групи безпеки. Після цих кроків ви створите EC2 інстанс, який можна використовувати для розгортання та налаштування Jenkins (рисунок 4.5).

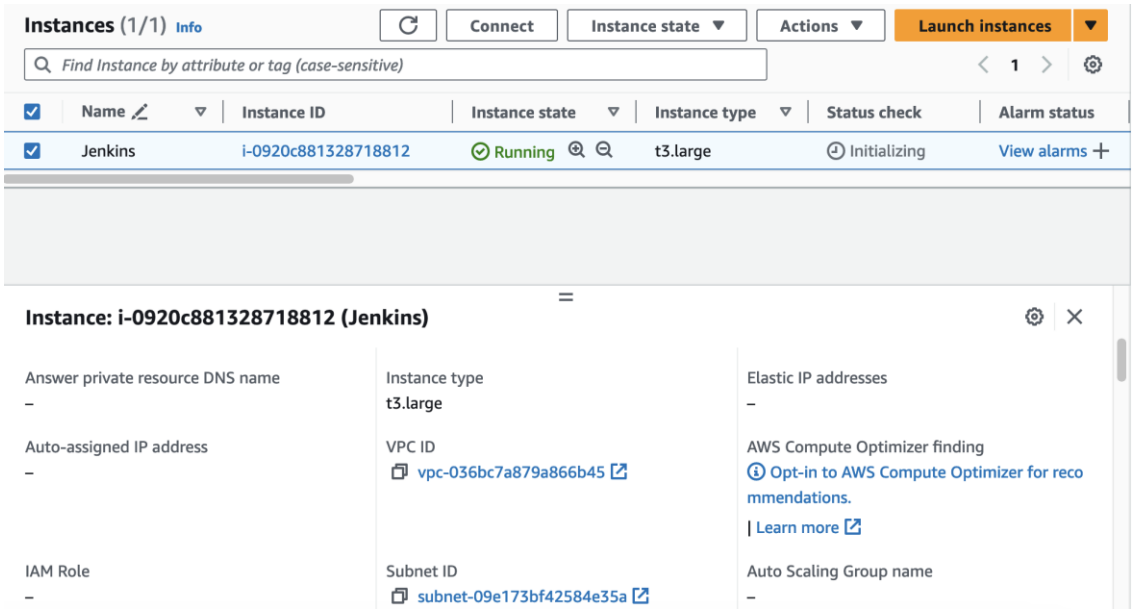

Рисунок 4.5 – Jenkins EC2 інстанс

У процесі інсталяції системи Jenkins на екземплярі EC2 використовувалися команди у середовищі командного рядка bash, що були виконані безпосередньо на зазначеному екземплярі (рисунок 4.6).

```
[ec2-user ~]$ sudo yum update -y
[ec2-user ~]$ sudo wget -O /etc/yum.repos.d/jenkins.repo \
    https://pkg.jenkins.io/redhat-stable/jenkins.repo
[ec2-user ~]$ sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key
[ec2-user ~]$ sudo yum upgrade
[ec2-user ~]$ sudo dnf install java-17-amazon-corretto -y
[ec2-user ~]$ sudo yum install jenkins -y
[ec2-user ~]$ sudo systemctl enable jenkins
[ec2-user ~]$ sudo systemctl start jenkins
[ec2-user ~]$ sudo systemctl status jenkins
```

Рисунок 4.6 – Встановлення Jenkins на EC2 інстанс

Після успішного виконання вище зазначених команд треба знайти початковий пароль, що знаходиться /var/log/jenkins/jenkins.log та можна перевірити роботу Jenkins server (рисунок 4.7).

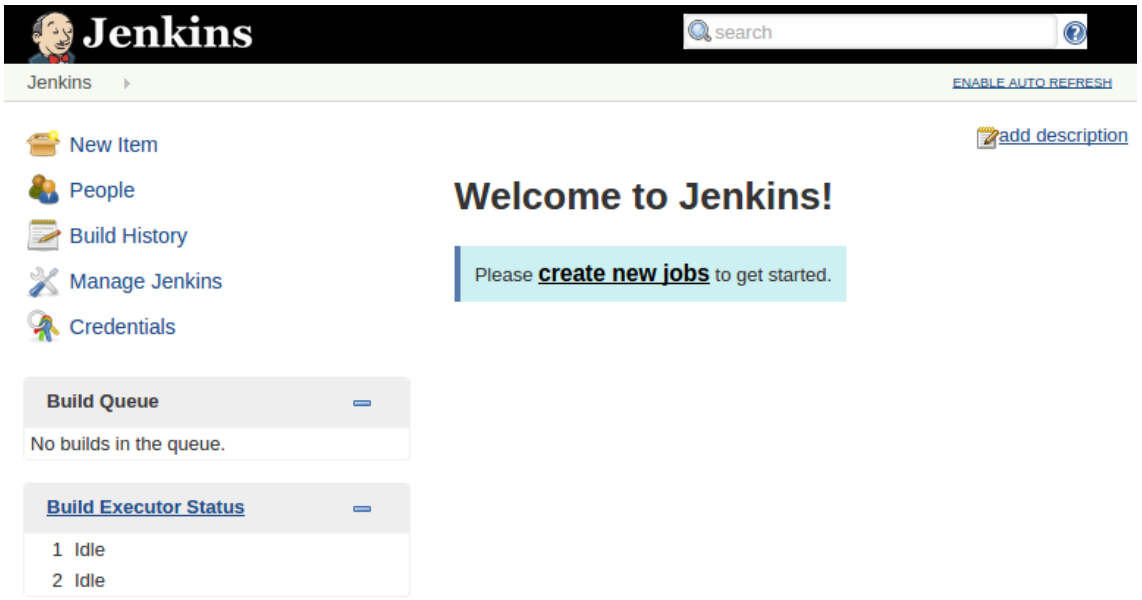

Рисунок 4.7 – Інстальований Jenkins на EC2 інстанс

Створнення AWS EC2 інстанс для Spinnaker ідентичне що й для Jenkins окрім типу інстанса. Для Spinnaker було обрано t3.xlarge оскільки потребує більше ресурсів (рисунок 4.8).

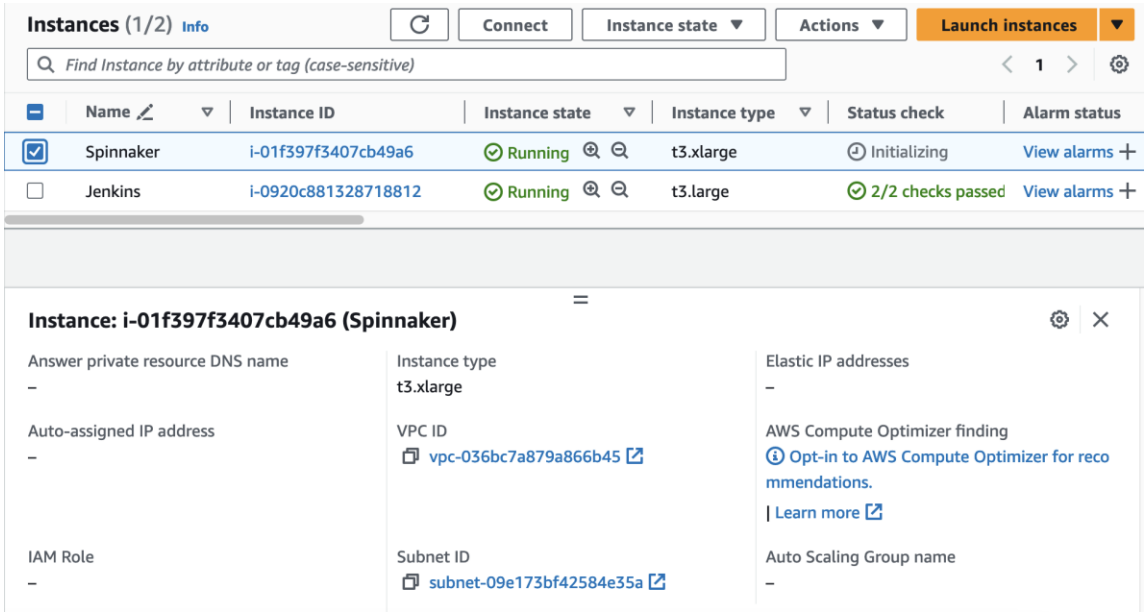

Рисунок 4.8 – Spinnaker EC2 інстанс

Інсталяція Spinnaker на AWS EC2 інстанс включає етапи як і для Jenkins. Встановлений Spinnaker наведено на рисунку 4.9.

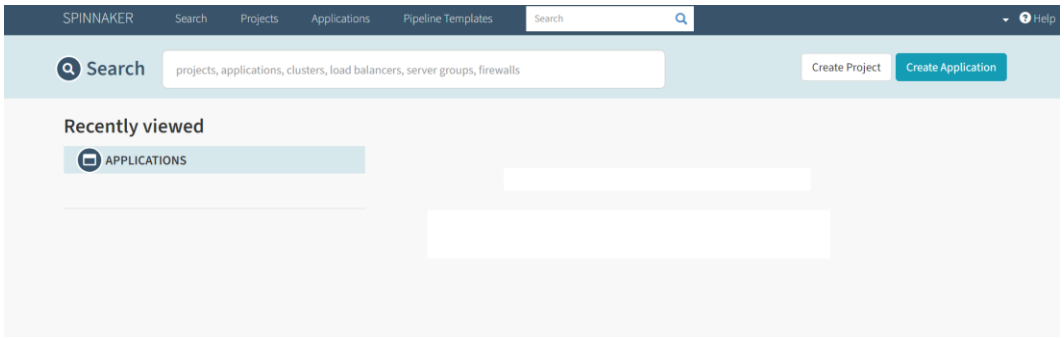

Рисунок 4.9 – Інстальований Spinnaker на EC2 інстанс

4.4 Налаштування CI/CD та створення Helm чартів

Для реалізації інформаційної системи був розроблений експериментальний програмний додаток, який отримав назву "Hello World" і був написаний мовою програмування Java (рисунок 4.10).

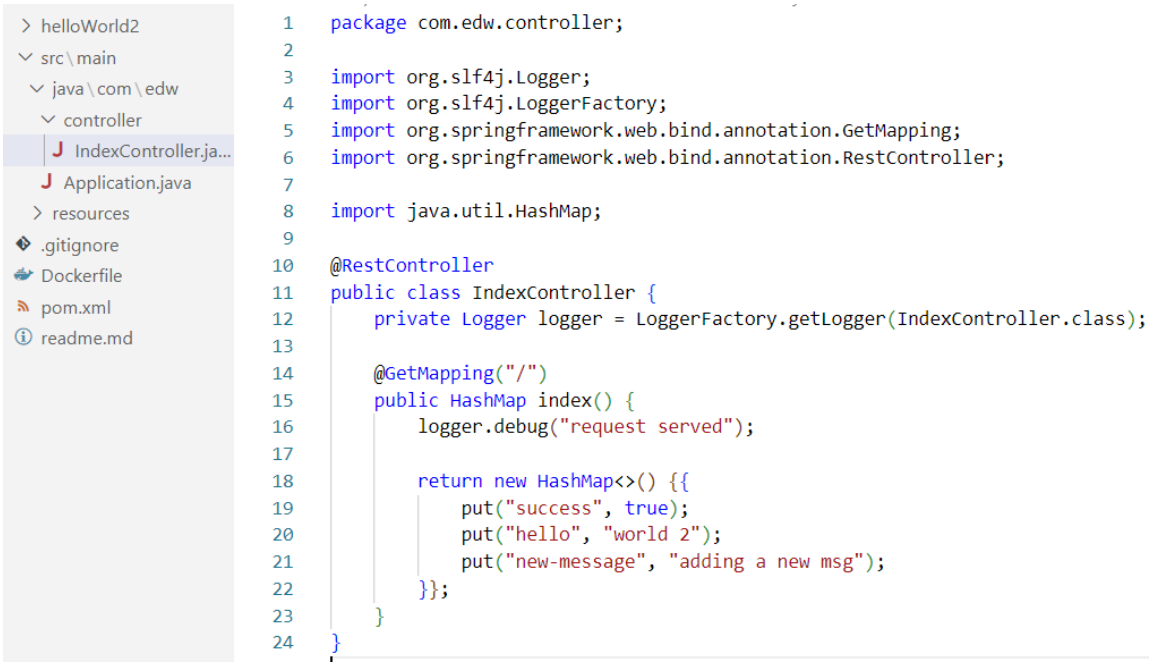

Рисунок 4.10 – Експериментальний додаток

Для даного додатку було створено два репозиторії з іменами "helloworld1" та "helloworld2" на веб-платформі GitHub. Окрім вище зазначених репозиторіїв створенно ще репозиторії під назвою worl для Umbrella чарта та репозиторій ci-world для коду CI/CD пайплайна (рисунок 4.11).

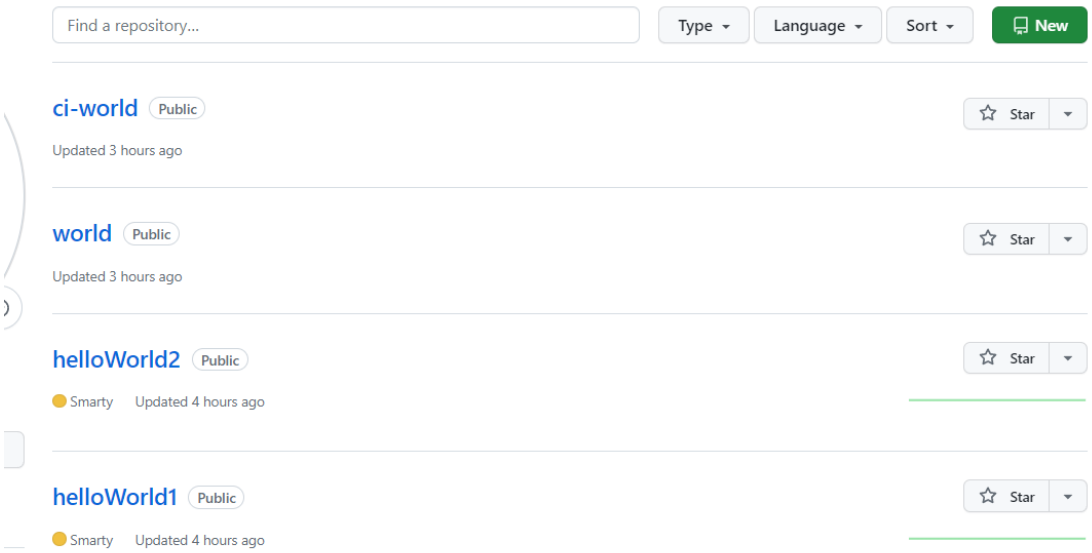

Рисунок 4.11 – Репозиторії для реалізації ІС

Код, призначений для реалізації проекту "ci-world", структурований утрьох основних директоріях: "env", "jenkins" та "spinnaker" (рисунок 4.12). У директорії "env" розташовані файли, що містять набір значень для чартів. Директорія "jenkins" включає конфігураційні файли, необхідні для налаштування процесів неперервної інтеграції та доставки (CI/CD) на сервері Jenkins. Конфігурації для сервера Jenkins включають Jenkinsfile для кожної задачі та набір етапів. У директорії "spinnaker" розміщені конфігураційні файли для налаштування сервера Spinnaker.

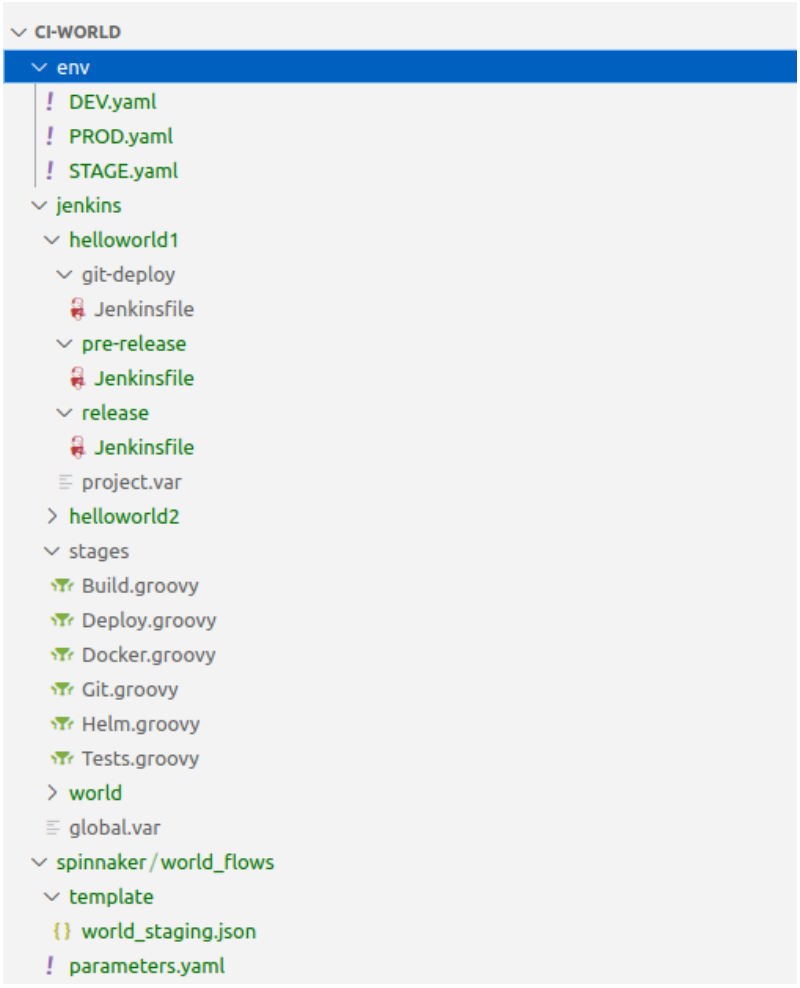

Рисунок 4.12 − Структура "ci-world" проекту

Надалі розглянемо створений Helm-чарт "world", що включає два підчарти: "helloworld1" та "helloworld2". У файлі "Chart.yaml" можна знайти

імена та версії підчартів, а також посилання на сховище, де зберігаються відповідні чарти, а саме, в хмарному сховищі S3 (рисунок 4.13).

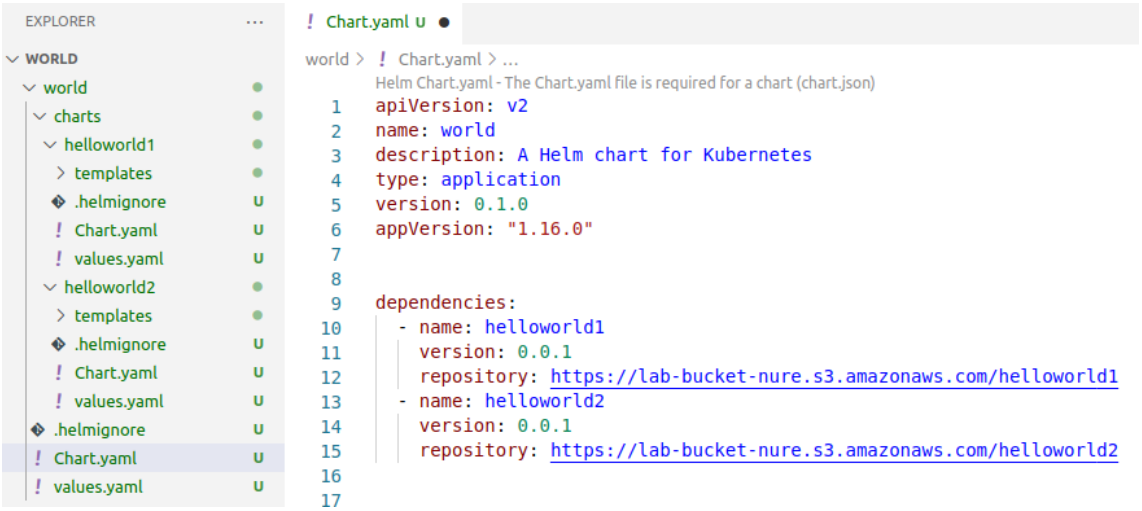

Рисунок 4.13 – Структура хелм чарту "world"

Кожен з пакованих чартів підлягає передачі в хмарне сховище S3, де кожен із них утримується в окремій директорії, як відображено на рисунку 4.14.

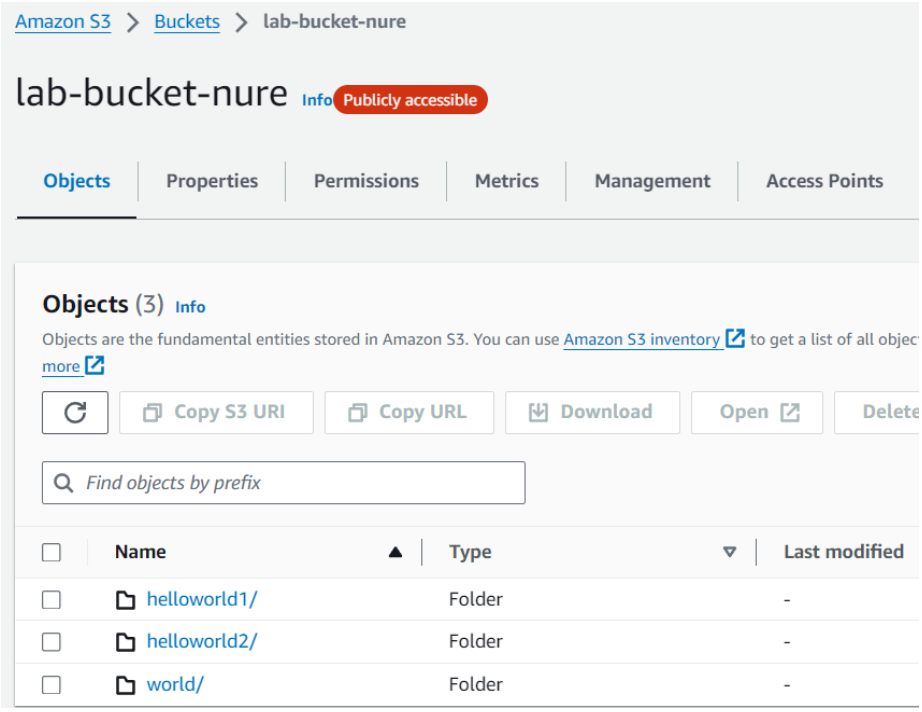

Рисунок 4.14 – Структура директорій в хмарному сховищі S3

З метою конфігурації пайплайнів на сервері Jenkins, було створено окремий Jenkinsfile для кожної завданої роботи. Усі Jenkinsfile є ідентичними, з винятком етапів, які імпортуються з директорії "stages" у репозиторії "ci-world" (рисунок 4.11). Конфігурація виглядає наступним чином:

```
#!/usr/bin/env groovy

node('agent') {
    try {

        // Set Job's ENVs
        env.GLOBAL_VAR_PATH = 'jenkins/global.var'
        env.PROJECT_VAR_PATH = 'jenkins/helloworld1/project.var'

        // Run Prepare stage
        stage('Prepare') {

            println('SCM...')
            checkout(scm)

            println('Load Global vars...')
            load(env.GLOBAL_VAR_PATH)

            println('Load Project vars...')
            load(env.PROJECT_VAR_PATH)

            println('Load Stages..')
            buildStage = load(env.Build)
            gitStage = load(env.Git)
            dockerStage = load(env.Docker)
            helmStage = load(env.Helm)
            testsStage = load(env.Tests)
            deployStage = load(env.Deploy)

        }

        //Run Git Stage
        gitStage.GitClone(project: env.PROJECT)

        // Run Build Stage
        buildStage.BuildProject(project: env.PROJECT)

        // Run Unittests Stage
        testsStage.UnitTests(project: env.PROJECT,
                             stageName: '')

        // Run Docker Stage
        dockerStage.BuildImage(project: env.PROJECT,
                               version: env.VERSION)
```

```
            // Run Helm Stage
            helmStage.HelmPackage(project: env.PROJECT,
                                  version: env.VERSION)

             // Run Deploy Stage
            deployStage.PrepareDeploy(project: env.PROJECT,
                                      version: env.VERSION)

            // Run Deploy Stage
            deployStage.Deploy(project: env.PROJECT)

            // Run Unittests Stage
            testsStage.Acceptence(project: env.PROJECT)

        } catch (Exception errJob) {
            println(errJob)
}
```

Стейдж може бути представлений у вигляді функціонального блоку, який здійснює послідовність індивідуальних етапів, як показано на рисунку 4.15.

```
/* Stage for package chart
 */
def HelmPackage(String project, String version, String stageName='') {
    stage(stageName ? stageName : 'Package chart') {
        dir(project) {
            String comm
            println('Package chart... ')

            comm = """helm package ${project} --version ${version}
                    |curl -X PUT -T "${project}/${project}-${version}" https://lab-bucket-nure.s3.amazonaws.com/${project}/""".stripMargin()
            sh(comm)
        }
    }
}
```

Рисунок 4.15 – Функціональний блок

Після початку виконання, створена завдання (джоба) на сервері Jenkins має структурований вигляд, що включає послідовні етапи: "Prepare", "Checkout project", "Build", "Unit tests", "Build Image", "Packege Helm Chart", "Prepare deploy", "Deploy", а також "Acceptance Tests" (рисунок 4.16).

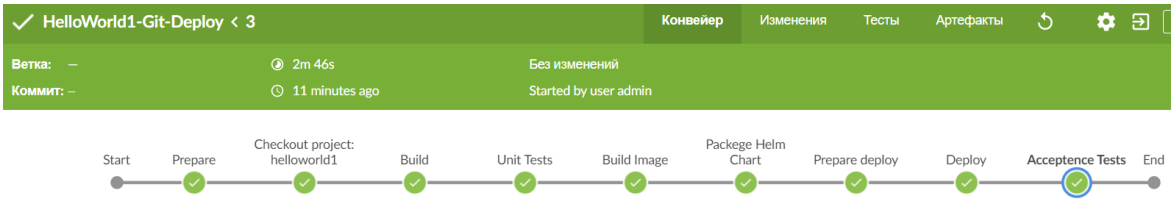

Рисунок 4.16 – Пройдений конвеєр на Jenkins server

Для проектів "helloworld1" та "helloworld2" були створені завдання, включаючи "Git-Deploy", "Git-Quality-Checks", "Pre-Release" та "Release". У випадку проекту "world" на сервері Jenkins були налаштовані завдання "Git-Deploy", "Pre-Release" та "Release" (рисунок 4.17).

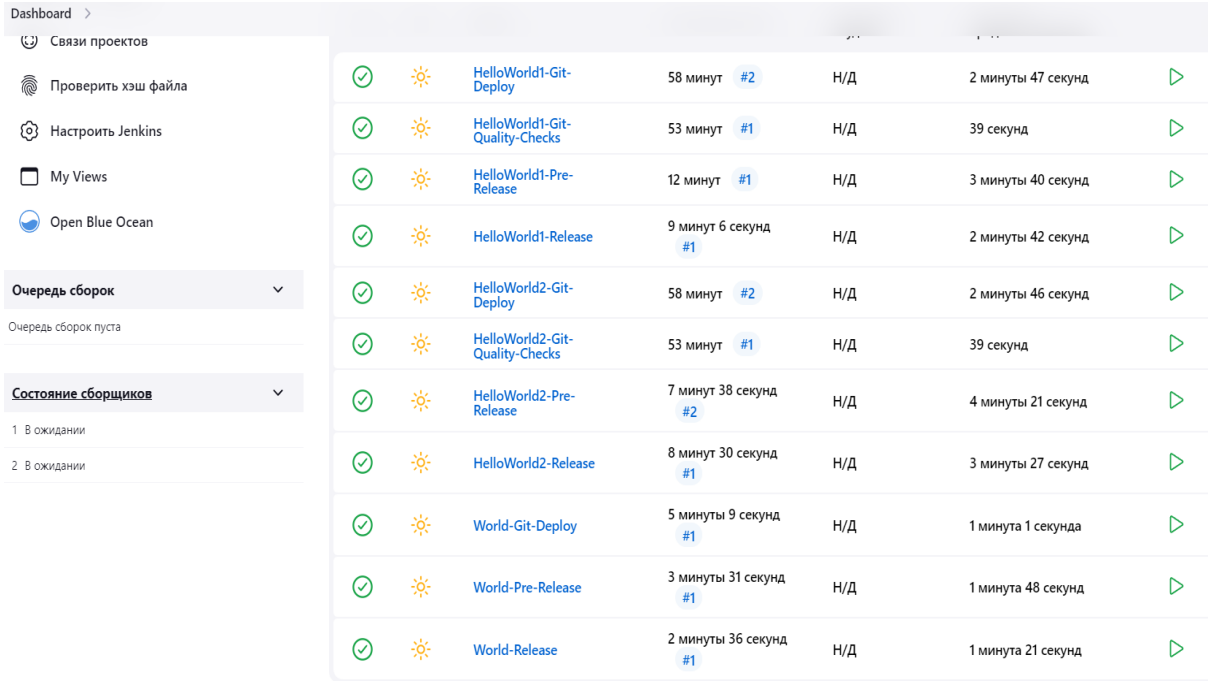

Рисунок 4.17 – Створенні завдання на Jenkins server

На наступному етапі проводиться конфігурація сервера Spinnaker. Файли конфігурації розміщені в репозиторії "ci-world" у директорії "spinnaker". У файлі "parameters.yaml" містяться параметри пайплайнів додатків "helloworld1", "helloworld2" та "world" для Spinnaker. Шаблон знаходиться у директорії "template" (рисунок 4.18).

Рисунок 4.18 − Конфігуруційні файли для spinnaker server

Важливо відзначити, що в залежності від використаного пайплайну для розгортання додатка використовується файл параметрів для Helm-чартів у середовищах розробки (DEV), тестування (STAGE) та продукції (PROD). Зазначені файли розташовані у репозиторії "ci-world" у директорії "env" та включають значення ресурсів, що визначають параметри додатка, який розгортається в кластері (рисунок 4.19).

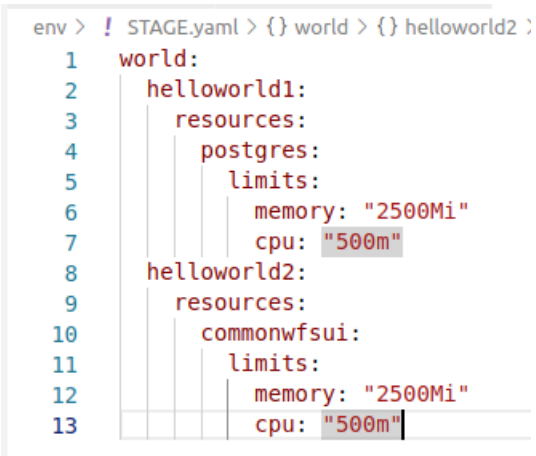

Рисунок 4.19 − Параметри конфігурації додатку

Після інтеграції внесених змін у додатку до гілки "мастер", ініціюється виконання пайплайну Spinnaker. У випадку успішного виконання цього пайплайну виникає релізна версія додатку (рисунок 4.20).

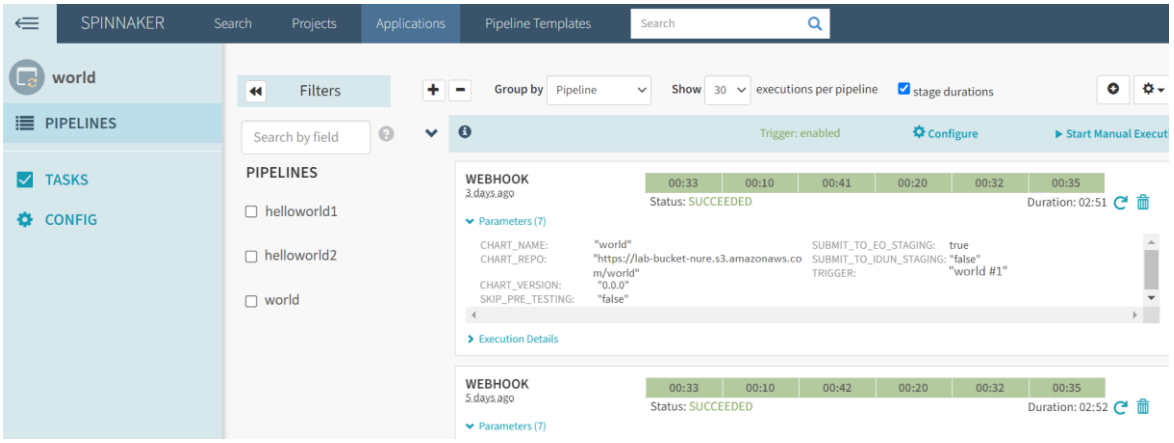

Рисунок 4.20 – Spinnaker пайплайни

Для оцінки успішності розгортання додатку в кластері Kubernetes можна виконати перевірку коректного стану усіх подів, що були запущені (рисунок 4.21).

```
NAME                                  READY   STATUS    RESTARTS   AGE
world-helloworld1-66c5549bbc-rhdmg    1/1     Running   0          4m39s
world-helloworld2-849bbb549c-gsmq9    1/1     Running   0          4m39s
```

Рисунок 4.21 – Поди розгорнутого додатку

Здійснення перевірки розгорнутої версії додатку в кластері Kubernetes можливо за допомогою вивчення відповідного стану у контексті кластера (рисунок 4.22).

```
NAME    NAMESPACE       REVISION        UPDATED                                 STATUS          CHART           APP VERSION
world   world           1               2023-12-18 22:03:14.668369164 +0300 MSK deployed        world-0.0.2     1.16.0
```

Рисунок 4.22 – Версія розгорнутого додатку в кластері

Такий підхід дозволяє ефективно впроваджувати зміни в багатосервісному корпоративному додатку паралельно та проводити випуск

нових версій продукту, використовуючи засоби неперервної інтеграції/неперервної доставки (CI/CD) та менеджера пакетів Helm для розгортання.

Час впровадження нових версій звичайним методом складає: T = 0.3 + 2 + 1.32 + 0.45 = 4.47 хвилин. За допомогою вдосконаленого методу впровадження нових версій складе: T' = 0,15 + 1.18 + 0.32 + 0.15 = 2.20 хвилин.

Коефіцієнт ефективності в частині зменшення часу впровадження нових версій складе: E1 = (4.47 – 2.20) / 4.47 = 50%. Тобто, вдосконалений метод дозволяє скоротити час впровадження нових версій на 50%.

Коефіцієнт ефективності в частині зменшення кількості помилок, виявлених під час розгортання, складає E2 = (3 - 2) / 3 = 30%. Тобто, вдосконалений метод дозволяє зменшити кількість помилок, виявлених під час розгортання, на 30%.

В традиційному методі впровадження нових версій корпоративного додатку ймовірність невдалого розгортання становить 10%. Вдосконалений метод дозволяє підвищити надійність розгортання до 99%. Тоді коефіцієнт ефективності в частині підвищення надійності розгортання складе E3 = (99 - 10) / 10 = 89%. Тобто, вдосконалений метод дозволяє підвищити надійність розгортання на 89%.

В традиційному методі впровадження нових версій корпоративного додатку ймовірність порушення безпеки під час розгортання становить 1%. Вдосконалений метод дозволяє підвищити безпеку розгортання до 99,99%. Тоді коефіцієнт ефективності в частині підвищення безпеки розгортання складе E4 = (99,99 - 1) / 1 = 99,99%. Тобто, вдосконалений метод дозволяє підвищити безпеку розгортання на 99,99%.

Експерименти показали, що вдосконалений метод дозволяє:

− знизити час, необхідний для впровадження нових версій, на 50%;

− знизити кількість помилок, виявлених під час розгортання, на 30%;

− збільшити надійність і безпеку розгортання.

Вдосконалений метод автоматизації розгортання корпоративного додатку в кластер Kubernetes є ефективним способом підвищення якості і надійності розгортання. Метод може бути використаний для будь-якого корпоративного додатку, що складається з великої кількості мікросервісів.

У подальших дослідженнях можна провести аналіз ефективності вдосконаленого методу в різних умовах. Наприклад, можна вивчити вплив розміру додатку, складності тестів і навантаження на продуктивне середовище на час і якість розгортання.

ВИСНОВКИ

У сфері розробки програмного забезпечення спостерігається зростаючий інтерес до вдосконалення процесів налагодження та забезпечення безперервної доставки і розгортання. Технології та методології неперервно розвиваються, що призводить до вдосконалення інструментів. Відзначається, що сучасні вимоги до неперервної інтеграції та постійного розгортання включають в себе обов'язковість проведення неперервних тестувань внесених змін, а також підготовку та налаштування тестового середовища для забезпечення якості.

Зазначається, що ринок пропонує різноманітні рішення, що може викликати труднощі у визначенні оптимальної технології, яка при впровадженні принесе максимальну користь.

У ході даного дослідження були аналізовані широко поширені методи для неперервної інтеграції та постійного розгортання (CI/CD) та менеджери пакетів. Результатом цього дослідження є порівняльний аналіз різних рішень, після чого було вибрано методи для подальшого детального аналізу. Задачею для тестування був визначений тестовий проект з відкритим вихідним кодом, який найбільш відповідав вихідній задачі.

Після визначення оптимальних методів у контексті неперервної інтеграції та постійного розгортання (CI/CD), виконано їх імплементацію на обраному досліджуваному проекті. До кожного з обраних рішень були розроблені конвеєри, спрямовані на тестування функціональності та налагодження системи з метою забезпечення їх ефективності та правильної функціональності.

Підчас тестування було запропоновано альтернативу обраним трьом рішенням, а саме їх поєднання (Spinnaker + Jenkins + Helm) через неможливість використання тільки одного рішення.

Дослідження успішно пройшло етап апробації під час участі в міжнародному молодіжному форумі "Радіоелектроніка та Молодь у XXI столітті".

Результати дослідження можуть бути використані при виборі рішень CI/CD для проектів рівня Enterprise для вирішення потреб та викликів великих корпорацій чи організацій. Ці проекти відзначаються розмаїттям технічних, фінансових, та бізнес-аспектів і можуть включати в себе розробку та впровадження комплексних систем, стратегічне планування, інтеграцію технологій, а також забезпечення високого рівня масштабованості та безпеки.

Отже, робота є актуальною, в подальшому її можна вдосконалити шляхом детального аналізу найпоширеніших рішень, їх тестування та дослідження можливих інтеграцій між ними.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Методичні вказівки щодо розробки та оформлення магістерської кваліфікаційної роботи за спеціальністю 122 Комп'ютерні науки (освітня програма «Управління проєктами в галузі інформаційних технологій» освітньокваліфікаційного рівня «магістр» / Упоряд.: Петров К.Е., Левикін В.М., Чалий С.Ф., Євланов М.В., Саєнко В.І., Міхнов Д.К., Міхнова А.В., Чала О.В. – Харків: ХНУРЕ, 2021. – 24 с.

2. ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура і правила оформлювання. – Чинний від 22.06.2015. – Київ: ДП «УкрНДНЦ», 2016. – 31 с.

3. Manage Kubernetes Objects [Електронний ресурс] // Kubernetes. – 2023. – Режим доступу до ресурсу: https://kubernetes.io/docs/tasks/manage-kubernetes-objects — Дата доступу: 01.12.2023

4. Helm documentation [Електронний ресурс] // Helm. – 2023. – Режим доступу до ресурсу: https://helm.sh/docs.

5. 5 GitOps Best Practices [Електронний ресурс] // Medium. – 2023. – Режим доступу до ресурсу: https://blog.argoproj.io/5-gitops-best-practices-d95cb0cbe9ff — Дата доступу: 01.12.2023

6. Operator pattern [Електронний ресурс] // Kubernetes. – 2023. – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/extend-kubernetes/operator/ — Дата доступу: 01.12.2023

7. «7 CI/CD tools for sysadmins» [Електронний ресурс] — Режим доступу: https://opensource.com/article/18/12/cicd-tools-sysadmins — Дата доступу: 01.12.2023

8. «Что такое travis-ci.org и с чем его едят?» [Електронний ресурс] — Режим доступу: https://habr.com/ru/post/140344/ — Дата доступу: 03.12.2023

9. «Concourse is an open-source continuous thing-doer.» [Електронний ресурс] — Режим доступу: https://concourse-ci.org/ — Дата доступу: 03.12.2023

10. [Електронний ресурс] — Режим доступу: https://github.com/mmorejon/microservices-docker-go-mongodb — Дата доступу: 03.12.2023

11. [Електронний ресурс] — Режим доступу: https://github.com/sqshq/PiggyMetrics — Дата доступу: 03.12.2023

12. «Best practices for operating containers» [Електронний ресурс] — Режим доступу: https://cloud.google.com/solutions/best-practices-for-operating-containers — Дата доступу: 03.12.2023

13. «REST» [Електронний ресурс] — Режим доступу: https://uk.wikipedia.org/wiki/REST — Дата доступу: 04.12.2023

14. [Електронний ресурс] — Режим доступу: https://docs.docker.com/get-started/overview/ — Дата доступу: 04.12.2023

15. [Електронний ресурс] — Режим доступу: https://yaml.org/spec/history/2001-12-10.html — Дата доступу: 04.12.2023

16. «Microservices» [Електронний ресурс] — Режим доступу:— https://martinfowler.com/articles/microservices.html Дата доступу: 06.12.2023

17. «Мікросервіси для початківців» [Електронний ресурс] — Режим доступу: — https://blog.ukrnames.com/veb-master/mikroservisi-dlya-pochatkivtsiv Дата доступу: 06.12.2023

18. Setup Prometheus monitoring on Kubernetes using Grafana [Електронний ресурс] // Medium. – 2023. – Режим доступу до ресурсу: https://medium.com/@sushantkapare1717/setup-prometheus-monitoring-on-kubernetes-using-grafana-fe09cb3656f7 — Дата доступу: 06.12.2023

19. Filatov V. O., Yerokhin A. L., Zolotukhin O. V., Kudryavtseva M. S. Hybrid simulation models for complex decision-making problems with partial

uncertainty // Information Extraction and Processing. 2022, 50(126), 78-86. DOI:https://doi.org/10.15407/vidbir2022.50.078

20. 27-й Міжнародний молодіжний форум «Радіоелектроніка і молодь у XXI столітті» [Електронний ресурс] // ХНУРЕ. — 2023. — Режим доступу: — https://nure.ua/konferencii-ta-workshops/mizhnarodnij-molodizhnij-forum-radioelektronika-i-molod-u-hhi-stolitti/xxiii-mizhnarodnij-molodizhnij-forum-radioelektronika-i-molod-u-hhi-stolitti — Дата доступу: 24.12.2023