Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет інфокомунікацій
(повна назва)

Кафедра інформаційно-мережної інженерії
(повна назва)

**КВАЛІФІКАЦІЙНА РОБОТА**
**Пояснювальна записка**

Рівень віщої освіти другий (магістерський)

Методи автоматизації процесів розробки веб-додатків у кластерному середовищі
(тема)

Виконав:
студент 2 курсу, групи ІМІм-19-2

Спеціальності 172 Телекомунікації та радіотехніка
(код і повна назва спеціальності)

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

Освітня програма Інформаційно-мережна інженерія
(повна назва освітньої програми)

Ходаківський М.А.
(прізвище та ініціали)

Керівник проф. Безрук В.М.
(посада, прізвище та ініціали)

Допускається до захисту
Зав. кафедри _____ Безрук В.М.
(підпис) (прізвище та ініціали)

2021 р.

Не містить відомостей, заборонених до відкритого публікування

Студент _____ Ходаківський М.А.

Керівник_____ Безрук В.М.

Харківський національний університет радіоелектроніки

Факультет інфокомунікацій

Кафедра інформаційно-мережної інженерії

Рівень вищої освіти другий (магістерський)

Спеціальність 172 Телекомунікації та радіотехніка
(код і повна назва)

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

Освітня програма Інформаційно- мережна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 ___ р.

# ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Ходаківському Миколі Анатолійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Методи автоматизації процесів розробки веб-додатка у кластерному середовищі

затверджені наказом по університету від " 12 " березня 2021 року № 350 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 21 травня 2021р.

3. Вихідні дані до роботи Розробити архітектуру на базі контейнерного оркестратору для підвищення продуктивності розробки додатку, впровадити методологію devops для автоматизації процесів розробки.

4. Перелік питань, які потрібно опрацювати в роботі

Вступ

1 Infrastructure as Code

2 Організація серверу

3 Оркестратори контейнерів

4 Організація kubernetes кластеру

5 Побудова конвеєру CICD

Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів)

Слайди у форматі Power Point (назва, мета роботи, віртуалізація та контейнеризація, вибір контейнерного оркестратору, архітектура kubernetes, вибір інструменту безперервної інтеграції та автоматизації, структура Jenkins пайплаїнів, демонстрація, висновки)

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів роботи | Терміни виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Ознайомлення із завданням. Уточнення ТЗ. | 12.03.2021 | |
| 2 | Підбір літератури за темою роботи. | 15.03-16.03.21 | |
| 3 | Виконання розділу 1 | 17.03-20.03.21 | |
| 4 | Виконання розділу 2 | 21.03-26.03.21 | |
| 5 | Виконання розділу 3 | 27.04-06.04.21 | |
| 6 | Виконання розділу 4 | 07.04-19.04.21 | |
| 7 | Виконання розділу 5 | 20.04-28.04.21 | |
| 8 | Оформлення пояснювальної записки | 08.04-10.05.21 | |
| 9 | Оформлення презентаційного матеріалу, | 11.05-13.05.21 | |
| | підготовка до захисту у ЕК | | |
| | | | |
| | | | |

Дата видачі завдання 18 березня 2021р.

Студент _____
(підпис)

Керівник роботи _____ проф. Безрук В.М.
(підпис) (посада, прізвище та ініціали)

# РЕФЕРАТ

Пояснювальна записка: 86 с., 27 рис., 20 джерел., 4 додатки

Мета роботи: Розробити автоматизовану систему та вибрати методи розгортання веб-додатка в кластерному середовищі в основі якої буде контейнерний оркестратор Kubernetes.

Для організації роботи потрібно вибрати сервер, за допомого якого буде можливим розгортання системи. Центральний елемент системи є орекстратор контейнерів Kubernetes який буде встановленний на сервері, для створення CI частини був вибратний інструмент Jenkins, як система контролю версій веб-додатку був вибраний git. Для розробки тестового веб-додатку було вибрано фреймворк flask.

В роботі були розглянуті різні контейнерні оркестратори. В результаті аналізу слабких і сильних сторін існуючих оркестраторів було вибрано саме Kubernetes. В якості операційної системи для сервера було вибратно unix подібну операційну систему Ubuntu 20.04 LTS на основі ядра Debian з підтримкою веб-інтерфейсу для інтерактивності.

Розроблено Kubernetes кластер та веб-додаток, також побудовано автоматизований конвеєр для деплою веб-додатку.

КЛАСТЕРНЕ СЕРЕДОВИЩЕ, KUBERNETES, LINUX, GIT, СЕРВЕР, JENKINS, ВЕБ – ДОДАТОК, ВЕБ – СЕРВЕР, СКРИПТОВА МОВА YAML, BASH, МОВА ПРОГРАМУВАННЯ PYTHON.

# ABSTRACT

Explanatory note: 86 pages., 27 figures., 20 reference., 4 addition

Purpose: To develop an automated system and select methods for deploying a web application in a clustered environment based on the Kubernetes container orchestrator.

To organize the work you need to choose a server with which it will be possible to deploy the system. The central element of the system is the Kubernetes container orchestrator which will be installed on the server, to create the CI part was the Jenkins tool of choice, as the version control system of the web application was chosen git. The flask framework was chosen to develop the test web application.

Various container orchestrators were considered in the work. As a result of the analysis of the weaknesses and strengths of the existing orchestrators, Kubernetes was chosen. Unix-like Debu kernel-based Ubuntu 20.04 LTS operating system with support for web interactivity was chosen as the operating system for the server.

Developed the Kubernetes cluster and web application, an automated pipeline for web application development was also built.

CLUSETRS, KUBERNETES, LINUX, GIT, SERVER, JENKINS, WEB-SERVER, WEB-APPLICATION, SCRIPTING LANGUAGES YAML, BASH, LANGUAGE OF PROGRAMMING PYTHON.

ЗМІСТ

# ПЕРЕЛІК СКОРОЧЕНЬ

IAC – Інфраструктура як код (англ., Infrastructure as code);

API – Інтерфейс програмування додатків (англ., Application Programming Interface);

SCM – Системні конфігураційні менеджери; (англ., System Configuration Management);

ПК – Персональний комп’ютер;

AWS – Додатки від Amazon (англ., Amazon Web Services);

CI – Неперервна інтеграція (англ., Continuous Integration) ;

CD – Неперервна доставка (англ., Continuous Delivery,) ;

CD – Неперервне розгортання (англ., Continuous Deployment) ;

DevOps – Development and Operations;

QA – забезпечення якості (англ., Quality Assurance) ;

GKE – Google Kubernetes Service;.

AKS – Amazon Kubernetes Service;

PV – Persistent volume;

PVC – Persistent volume claim.

# ВСТУП

На сьогоднішній день існує проблема зі швидкістю, складністю та ефективністю розробки кінцевого програмного продукта, багато років розробники писали великі веб-додатки так названим монолітним методом, це коли розробляється великий додаток який в собі зберігає всі модулі та частинки коду, це було і є досить зручно в написанні, але такий метод має певні недоліки, перш за все якщо в додатку вийде з ладу тий чи інший модуль тоді працездатність всього додатка стане під загрозою, що є критичним аспектом, якщо дивитись з позиції бізнесу то як недолік варто віднести досить важкий процес відлатки та оновлення додатку.

Мікросервісна архітектура – це архітекура сьогодення львина доля всіх найпопулярнійших ресурсів, або проектів використовує саме цю систему методику побудови додатків. Принцип мікросервісної архітектури в тому, що весь додаток розподіляється на мікросервіси, наприклад додаток має авторизацію, в сервісній архітектурі його можливо зробити окремим мікросервісом. Переваги такого принципу є стабільність додатку, якщо один мікросервісом вийде з ладу його буде легко полагодити, також легко оновлювати версію додатку. Як недолік можна віднести досить важке налаштування і в цілому міжсервісна взаємодія.

Kubernetes в свою чергу – це платформа, яка надає всі можливості для інтеграції з мікросервісними додатками та забазпечує їх взаємодію між собою, а також прискорює та зменшує ризики при оновлення додатка.

# 1 INFRASTRUCTION AS CODE

Модель Infrastruction as Code (IaC , Інфраструктура як код), яку іноді називають «програмованої інфраструктурою», – це модель, за якою процес налаштування інфраструктури аналогічний процесу програмування ПЗ. [1] По суті, вона поклала початок усунення кордонів між написанням додатків і створенням середовищ для цих додатків. Додатки можуть містити скрипти, які створюють свої власні віртуальні машини і керують ними. Це основа хмарних обчислень і невід'ємна частина DevOps.

Інфраструктура як код дозволяє управляти віртуальними машинами на програмному рівні це більш детально розглянуто на рис. 1.1. Це виключає необхідність ручного налаштування і оновлень для окремих компонентів обладнання. Інфраструктура стає надзвичайно "еластичною", тобто відтворюється і масштабованої. Один оператор може виконувати розгортання і управління як однієї, так і тисячами машинами, використовуючи один і той самий код. Серед гарантованих переваг інфраструктури як коду – швидкість, економічність і зменшення ризику.

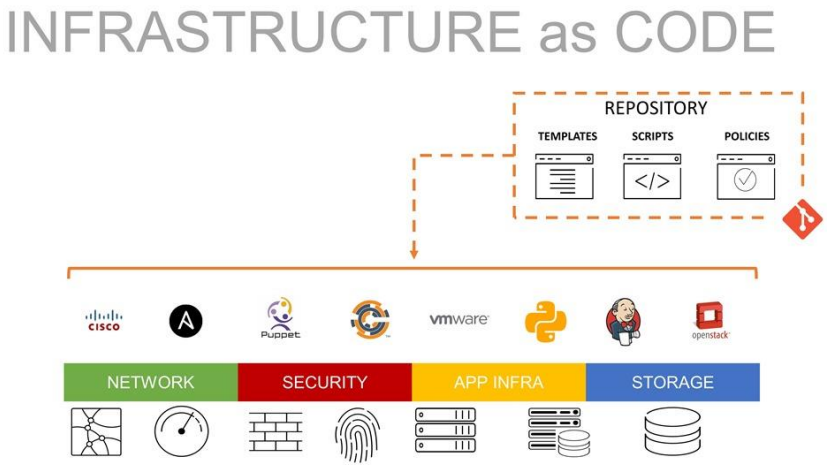


Рисунок 1.1 – Інфраструктура як код

Концепція інфраструктури як коду подібна до програмних сценаріїв, які використовуються для автоматизації IT– процесів. Однак, сценарії в основному використовуються для автоматизації серії статичних кроків, які необхідно повторювати багаторазово на декількох серверах.

## 1.1 Скриптові мови та їх порівняння

Назва «bash» є акронімом від Bourne–again–shell ( «ще–одна–командна–оболонка–Борна») і являє собою гру слів: Bourne–shell – одна з популярних різновидів командної оболонки для UNIX (sh), автором якої є Стівен Борн (1978), вдосконалена в 1987 році Брайаном Фоксом. Прізвище Bourne (Борн) перегукується з англійським словом born, що означає «народжений», звідси: народжена–знову–командна оболонка.

Bash – це командний процесор, що працює, як правило, в інтерактивному режимі в текстовому вікні. Bash також може читати команди з файлу, який називається скриптом (або сценарієм). Як і всі Unix–оболонки, він підтримує автодоповнення імен файлів і директорій, підстановку виведення результату команд, змінні, контроль за порядком виконання, оператори розгалуження і циклу. Ключові слова, синтаксис і інші основні особливості мови були запозичені з «sh». Bash в основному задовольняє стандарту POSIX, але з рядом розширень.

Script – це сценарна мова (мова сценаріїв, скриптова мова, від англ. Scripting language) – високорівнева мова сценаріїв (script) – коротких описів дій, виконуваних системою. Сценарій – це програма, що має справу з готовими програмними компонентами.

Користувацька оболонка bash може працювати в двох режимах – інтерактівному та, відповідно, неінтерактівному. Відкрити оболонки в Ubuntu можна комбінацією клавіш “Ctrl + Alt + F1”, звичайний графічний інтерфейс зникне, а перед вами відкриється один з семи віртуальніх терміналів, доступних в дістрібутіві Ubuntu. Як і в більшості мов програмування bash не є

виключенням, він включає такі функції як:

Масиви та списки – в bash також є можливістю працювати з масивами. При роботі з масивами часто користуються змінною оточення IFS – роздільника полів для вхідних рядків (IFS – Input Field Separator). За замовчуванням IFS рівний символу «пробіл» , але може бути змінений для розбиття рядка на елементи масиву, наприклад, за комами. Зверніть увагу, що для формування змінних оболонки, які доступні через «$1», «$2» тощо., використовується саме змінна IFS, тобто введений після ім’я скрипта рядок аргументів, буде розділений саме за першим символом, який зберігається в цій змінній.

Потоки – файл з якого відбувається читання, називають стандартним потоком введення, а в який відбувається запис, відповідно – стандартним потоком виводу.

Канали – стандартні потоки можна перенаправляти не лише в файли, але й на вхід інших сценаріям. З’єднання потоку виведення одної програми з потоком введення іншої називають каналом, або пайпом «|» (pipe).

Конвеери – це команди, що з’єднані операторами «&&»,«||»для виконання в певній послідовності.

Умовні оператори – в скриптовій мові bash підтримуються два оператори розгалуження: «if» та «else». Оператор «if», як і в інших мовах, виконує певний блок вказівок, в залежності від умови. Умову огортають в подвійні квадратні дужки «[[]]» , які bash розглядає як один елемент з кодом виходу. Всередині блоку операторів огорнутого в «[[]]» можемо використовувати оператори «&&» та «||»

Цикли – мова оболонки дає користувачеві можливість організовувати циклічне виконання інструкцій з допомогою циклів: «while», «for», «select».

Функції – в сценаріях оболонки можливе оголошення та виклик функцій. Варто зазначити, що саме поняття функцій в bash дещо урізане. Насправді, функції в bash – це іменована група команд, які виконаються під час звертання до функції. В будь–якому випадку функціями слід користуватись всюди, де є

код, що повторюється з невеликими варіаціями.

PowerShell - це командна оболонка з мовою сценаріїв, спочатку створена на основі платформи .NET Framework, а пізніше - на .NET Core. На відміну від приймаючої і повер3тають текстові дані оболонок, Windows PowerShell працює з класами .NET, це є основною різницею між двома оболнками bash та powershell у яких є властивості і методи. PowerShell дозволяє виконувати звичайні команди, а також дає доступ до об'єктів COM, WMI і ADSI. У ній використовуються різні сховища, на зразок файлової системи або реєстру Windows, для доступу до яких створені т.зв. постачальники (providers). Варто відзначити можливість вбудовування виконуваних компонентів PowerShell в інші додатки для реалізації різних операцій, в т.ч. через графічний інтерфейс. Вірно і зворотне: багато додатків для Windows надають доступ до своїх інтерфейсів управління через PowerShell.

## 1.2 Системи конфігурування інфраструктури

На сьогоднішній день є багато різних конфігураційних інструментів, але найбільшою популярністю мають попит такі як:

- chef;
- puppet;
- ansible;
- saltStack.

Chef – система управління конфігураціями, написана на Ruby (клієнтська частина) і Erlang (серверна частина), з використанням предметно–орієнтованої мови для опису конфігурацій. Використовується для спрощення завдань налаштування і підтримки безлічі серверів і може інтегруватися в хмарні платформи, такі як Rackspace і Amazon EC2, для автоматизації управління поточними та автоматизації процесу настройки нових серверів [2].

Користувач Chef створює певні «рецепти» з описом того, як управляти серверними додатками (наприклад, Apache, MySQL або Hadoop) і їх

налаштувань.

«Рецепт» – це опис стану ресурсів системи, в якому вона повинна знаходитися в конкретний момент часу, включаючи встановлені пакети, запущені служби, створені файли. Chef перевіряє, що кожен з ресурсів системи налаштований правильно і намагається виправити стан ресурсу, якщо воно не відповідає очікуваному.

Chef може працювати як в режимі клієнт–сервер, так і в режимі автономної конфігурації, званому «chef–solo». У режимі клієнт–сервер клієнт посилає на сервер різні властивості хоста, на якому він розташований. На стороні сервера використовується Solr для індексування властивостей і надання API для запиту інформації клієнтом. «Рецепти» можуть запитувати ці властивості і використовувати отримані дані для настройки хоста.

Зазвичай використовується для управління Linux–вузлами, але останні версії підтримують Windows .

Puppet – багатоплатформовий клієнт–серверний додаток, який дозволяє централізовано керувати конфігурацією операційних систем та програм, встановлених на кількох комп'ютерах. Puppet написано мовою програмування Ruby.

Puppet дозволяє просто налаштувати і згодом швидко керувати майже кожною мережею на базі будь–якої операційної системи Red Hat, CentOS, Fedora, Debian, Ubuntu, OpenSUSE , Solaris, BSD, Mac OS X і Microsoft Windows (через cygwin).

Система Puppet досить поширена у світі IT, де у своїй роботі її використовують такі компанії як Google, Fedora Project, Стенфордський університет, Red Hat, Siemens IT Solution, Badoo і SugarCRM.

Puppet дозволяє автоматизувати роботу з адміністрування групи серверів, уможливлюючи централізоване керування користувачами, установку пакетів, оновлення конфігурації тощо. Вузли мережі, керовані за допомогою Puppet, час від часу опитують сервер, отримують і застосовують зміни конфігурації внесені адміністратором. Для описання конфігурації

вживається особлива декларативна мова.

SaltStack – це програмне забезпечення з відкритим вихідним кодом на основі Python для автоматизації IT–подій, дистанційного виконання завдань і керування конфігурацією. Підтримка підходу "Infrustructure As Code" до систем обробки даних і мереж розгортання та управління мережами даних, автоматизації конфігурації, оркестровки SecOps, виправлення вразливостей та керування гібридною хмарою.

SaltStack походить від необхідності високошвидкісного збору даних і виконання завдань для системних адміністраторів центрів обробки даних, які керують масштабним масштабом інфраструктури і, в результаті, складністю.

Ця система керування конфігурацією зберігає всі дані конфігурації (стану) всередині легко зрозумілої структури даних, що використовує YAML.

Нові віртуальні машини та екземпляри хмари автоматично підключаються до Salt Master після створення.

Salt Cloud підтримує 25 публічних і приватних хмарних систем, включаючи AWS, Azure, VMware, IBM Cloud і OpenStack. Salt Cloud надає інтерфейс для Salt для взаємодії з хост–хмарами та функціональністю хмари, такою як DNS, сховище, балансування навантаження тощо.

Chef, Puppet, Ansible і SaltStack – це всі інструменти керування конфігурацією, тобто вони призначені для встановлення та керування програмним забезпеченням на існуючих серверах. CloudFormation або Terraform є «інструментами оркестрації», що означає, що вони призначені для надання самим серверам, залишаючи завдання налаштування цих серверів на інші інструменти.

Ці дві категорії не є взаємовиключними, оскільки більшість інструментів керування конфігурацією може мати певний ступінь підготовки, і більшість інструментів для оркестрування може мати певний рівень керування конфігурацією. Але акцент на управлінні конфігурацією або оркестровці означає, що деякі інструменти будуть краще підходити до певних типів завдань.

Інструменти керування конфігураціями, такі як Chef, Puppet, Ansible і SaltStack, типово використовуються як змінна інфраструктурна парадигма. Наприклад, якщо спромогтися на Chef встановити нову версію OpenSSL, вона запуститиме оновлення програмного забезпечення на існуючих серверах і зміни відбудуться на місці.

З часом, коли застосовується все більше і більше оновлень, кожен сервер створює унікальну історію змін. Це часто призводить до явища, відомого як дрейф конфігурації, де кожен сервер стає дещо іншим, ніж всі інші, що призводить до витоків, які важко діагностувати і майже неможливо відтворити.

- chef, Puppet і SaltStack за замовчуванням використовують архітектуру клієнт–сервер. Клієнт, який може бути веб–інтерфейсом або інструментом CLI, є тим, що використовується для видачі команд (наприклад, «розгортання X»). Ці команди переходять до сервера, який відповідає за виконання ваших команд і зберігання стану системи. Щоб виконати ці команди, сервер розмовляє з агентами, які повинні бути запущені на кожному сервері, який потрібно налаштувати. Це має ряд недоліків:

- потрібно встановити і запустити додаткове програмне забезпечення на кожному з ваших серверів;

- потрібно розгорнути додатковий сервер (або навіть кластер серверів для високої доступності) тільки для керування конфігурацією;

- необхідно не тільки встановити це додаткове програмне забезпечення та апаратне забезпечення, але й потрібно його підтримувати, оновити його, робити резервні копії, відстежувати його та відновлювати у випадку відключень;

- оскільки клієнт, сервер і агенти повинні обмінюватися даними через мережу, потрібно відкрити для них додаткові порти та налаштувати способи їх аутентифікації один до одного, і всі вони збільшують площу поверхні для зловмисників;

- всі ці додаткові рухомі частини вводять в інфраструктуру велику

кількість нових режимів відмови. Коли отримуємо звіт про помилку в 3 ранку, тоді доводиться з'ясувати, чи є це помилка в коді програми, коді IAC, клієнтському програмному забезпеченні керування конфігурацією або програмному забезпеченні агента керування конфігурацією або програмному забезпеченні сервера керування конфігурацією. або портів, які використовують всі ті елементи керування конфігурацією, або спосіб, у який вони аутентифікують один одного;

- ansible і Terraform використовують архітектуру тільки для клієнта. Клієнт Ansible працює, підключаючись безпосередньо до серверів через SSH. Terraform використовує API інфраструктури постачальника послуг хмари, тому не існує нових механізмів аутентифікації, що виходять за межі того, що вже використовується з постачальником хмарних послуг, і немає необхідності в прямому доступі до серверів. Виявляється, що це найкращий варіант з точки зору простоти використання, безпеки та ремонтопридатності.

- але кожний інструмент має свої переваги та недоліки в технічному плані порівняльна характеристика інструментів зображена на рис. 1.2.

| | Ansible | Chef | Puppet | SaltStack |
|---|---|---|---|---|
| Programming language | Python | Ruby, Erlang | C++, Clojure | Python |
| Configuration language | YAML, JSON | Ruby | Proprietary | YAML |
| Database | n/a | PostgreSQL | PuppetDB | n/a |
| Transport | ssh | RabbitMQ | Mcollective | ZeroMQ |
| Deployment method | push | pull | pull | push |
| Master | A, B, L, O, S | Linux | Linux | L, B |
| Agent | n/a | A, B, L, O, S, W | A, B, L, O, S, W | A, B, L, O, S, W |
| Agentless | yes | no | no | yes |
| Public Cloud | AM | AM/AZ/PR | no | no |
| Cloud management | AM, AZ, OS, GCP | Fog driver | AM, AZ, VM, GCP | Salt Cloud |
| Architecture | server | client, server | client, server | client, server |
| Step-by-step | yes | yes | no | no |
| Enterprise GUI | Ansible Tower | Opscode Manage | Puppet Enterprise | SaltStack Enterprise |
| Opensource GUI | Semaphore | Chef Manage | Foreman | Saltpad, Saltshaker |
| | | | | |
| Github Stars | 25,283 | 4,979 | 4,628 | 8,080 |
| Github Forks | 8,790 | 2,074 | 1,901 | 3,762 |
| Contributors | 2,957 | 519 | 455 | 1,885 |
| Commits | 32,760 | 19,692 | 25,816 | 85,485 |
| Enterprise version | yes | yes | yes | yes |

Рисунок 1.2 – Порівняльна характеристика інструментів.

## 1.3 Методологія DevOps

DevOps (акронім від англ. Development і operations) – це методологія розробки ПЗ, сфокусована на гранично активній взаємодії і інтеграції в одній упряжці програмістів, тестувальників і адмінів, синхронізоване обслуговуючих загальний для них сервісів або продуктів. Головна мета цього – створення єдиного циклу взаємозалежності розробки, експлуатації та розгортання програмного забезпечення, щоб в кінцевому рахунку допомагати організаціям (сервісів, стартапам) швидше і безболісніше створювати і оновлювати їх програмні продукти і сервіси, експлуатовані в режимі реального часу або «в продакшені».

DevOps формує «безшовний» цикл розробки на рис 1.3, тим самим допомагаючи прискорити випуск програмного продукту [3].

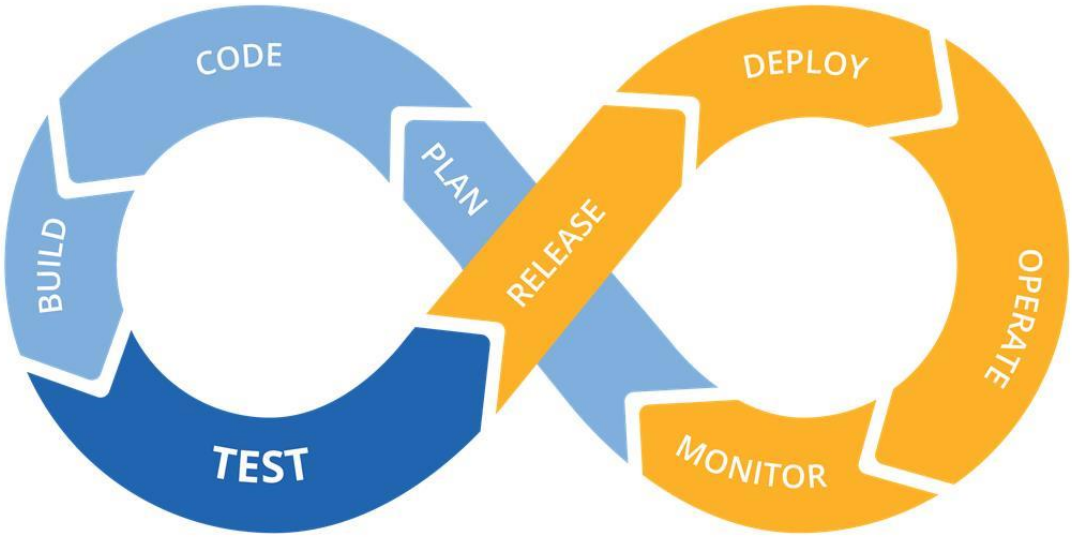


Рисунок 1.3 – DevOps цикл розробки

Прискорення досягається за рахунок впровадження систем автоматизації. Плюс програмісти починають брати участь в налаштуванні

серверів і пошуку багів, наприклад, вони можуть писати автоматизовані тести.

Таким чином налагоджується взаємодія між відділами. Співробітники починають краще розуміти, які етапи проходить програмний продукт перед тим, як потрапити в руки користувача.

Коли розробник розуміє, з чим стикається адміністратор при налаштуванні сервера, він намагається згладити можливі «гострі кути» в коді.

Сьогодні колись дуже чіткі межі рис.1.4 між розробкою і експлуатацією почали зникати. Це призводить до прискорення всього життєвого циклу програмного забезпечення, більш коротким сесій QA і багатьом іншим змінам. У нас навіть з'явилися нові процеси, такі як управління безперервної інтеграцією і безперервним розгортанням. Багато великі програмні рішення, розроблені за допомогою методики DevOps, розгортаються по кілька разів на день, а не кілька разів на рік.

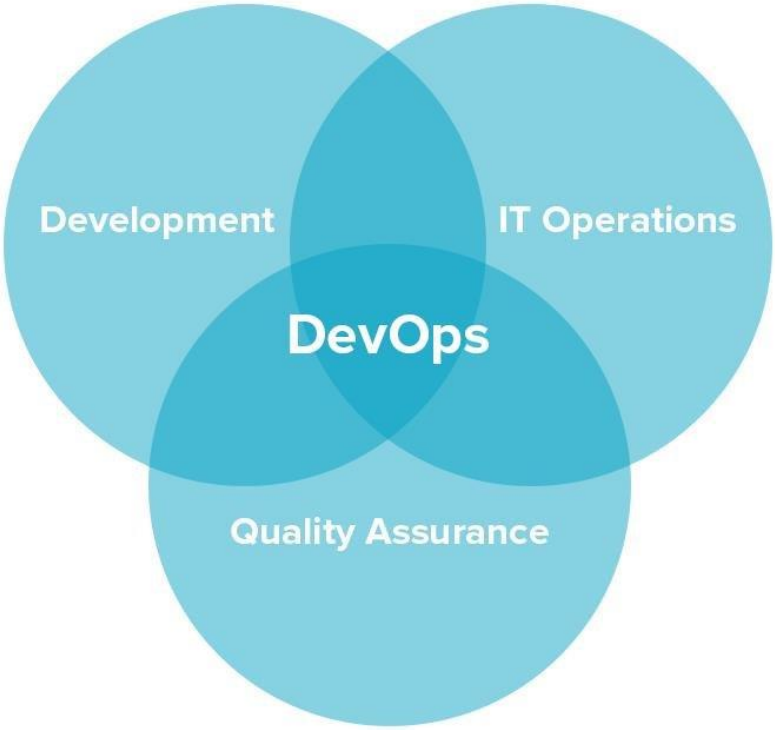


Рисунок 1.4 – Межі компетенції DevOps

Ця тенденція в автоматизації робочих процесів прогресувала поступово протягом останніх 10 років, поки програмне забезпечення та процеси не досягли своєї зрілості.

Методологія DevOps зародилася в 2009 році, але його витоки сягають ще глибше. Так, фахівці в якості прикладу інструментарію–прабатька називають популярний кроссплатформений клієнт–серверний додаток Puppet, перша версія якого з'явилася ще в 2005 році. Люку Канісу (Luke Kanies), нині засновнику компанії Puppet Labs і розробнику на Ruby в той час, набридло вручну налаштовувати Linux і вносити зміни в файли конфігурації. Він мріяв про спосіб, який би дозволив надавати і налаштовувати Unix–подібні системи більш програмним і відтвореним способом. Таким чином, він написав скрипт на Ruby, який робив це замість нього, і назвав його Puppet.

Пізніше на ринку з'явився аналогічний інструментарій, включаючи Chef, Ansible, SaltStack і багато інших. Крім того, навколо цих інструментів утворилися спільноти; розробники і системні адміністратори акумулювали свої знання у вигляді «рецептів», що дозволяють вам легко і швидко налаштувати програмне забезпечення незалежно від використовуваного основного дистрибутива Unix.

За допомогою подібного інструментарію розробники можуть створювати самодостатні програмні сценарії того, як запустити додаток. Вони можуть включати всі залежності і запускатися на різних дистрибутивах Unix за допомогою простого запуску скрипта. Те, що раніше займало кілька тижнів настройки в ручному режимі і виконувалося тільки висококваліфікованими фахівцями, в даний час робиться в лічені години за допомогою звичайного скрипта.

Хоча розробники і отримали можливість розгортати свій код швидше і простіше, ніж раніше, вони зіткнулися з іншою проблемою. Так як програмісти стали менше залежати від своїх колег з операційного відділу, на розробників лягла велика відповідальність за працездатність їх власних запущених додатків.

Ця проблема спричинила за собою появу на ринку ще одного інструментарію, який згодом набув широкого поширення: модель надання хмарних обчислень «платформа як послуга» PaaS. Хоча спочатку цей інструментарій просувався такими компаніями, як Salesforce і Google, перші реалізації PaaS вимагали від розробників писати спеціалізований код, який замикав додатки на їхній платформі. PaaS платформа не була широко популяризована, поки компанія Heroku не надала однойменну хмарну PaaS–платформу, що підтримує ряд мов програмування. З її допомогою розробники могли запускати свій код лише з невеликою кількістю істотних модифікацій, якщо такі взагалі були необхідні.

Системи PaaS виявилися на вершині принципів автоматизації DevOps. Більш того, в наші дні більшість представлених на ринку PaaS–платформ використовують DevOps–інструменти для настройки і запуску. Різниця в тому, що на PaaS–платформі додатки, які запущені на ній, повністю керовані. Можливо запускати, зупиняти, а також здійснювати масштабування і моніторинг програм в рамках PaaS–платформи через інтерфейс програмування додатків API. У DevOps мається можливим створити набір інструментів для управління вашими додатками, а з PaaS отримуємо вже попередньо налаштований інструментарій.

Найбільшим недоліком використання моделі надання хмарних обчислень PaaS є те, що вона дуже строго визначає архітектуру програми. За бажанням є можливість отримати більше контролю над середовищем виконання, технологія Linux контейнерів вам її наддасть, чи не вплинувши при цьому на швидкість і гнучкість виконання коду. Створення середовища з нуля за допомогою інструментарію Chef або Puppet може зайняти кілька годин, причому не має можливим бути до кінця впевненими в тому, що в результаті ви працюєте з точною її копією. Використання ж Linux контейнерів дозволить розробникам в лічені секунди відтворити середу Linux, при цьому можливо бути впевненими, що це її точна копія.

Інженери отримали величезний виграш до своєї продуктивності за

рахунок багаторічного розвитку та поліпшення інструментарію DevOps. Дні розробників, які не звертають увагу на проблеми операційного функціонування і масштабування своїх додатків, добігають кінця, так як мистецтво запуску додатків неухильно перетворюється в повністю автоматизований комп'ютерний код.

Необхідною концепцією у DevOps методології є безперервна інтеграція та доставка (CI/CD).

CI / CD – концепція, яка реалізується як конвеєр, полегшуючи злиття тільки що «закоміченного» коду в основну кодову базу. Концепція дозволяє запускати різні типи тестів на кожному етапі (виконання інтеграційного аспекту) і завершувати його запуском з розгортанням «закоміченного» коду в фактичний продукт, який бачать кінцеві користувачі (виконання доставки).

Continuous Integration (CD, Безперервна інтеграція) Розробники, що практикують постійну інтеграцію, зливають свої зміни до основної гілки репозиторію якомога частіше. Зміни розробника перевіряються, створюючи автоматизовані тести для зборки та запускаючи їх. Роблячи це, уникаючи «інтеграційного пекла», що зазвичай відбувається, коли люди чекають день релізу, щоб об'єднати свої зміни в кінцевій гілці репозиторію.

Безперервна інтеграція приділяє велику увагу автоматизації тестування, щоб перевірити, що програма не порушена, коли нові збірки інтегровані в основну гілку репозиторію.

Continuous Delivery (CD, безперервна доставка) – це продовження безперервної інтеграції, щоб переконатися, що можливо швидко і стабільно випускати нові зміни для ваших клієнтів. Це означає, що на додаток до автоматизованого тестування, також ще автоматизувався процес випуску, і можливо розгорнути свою програму в будь–який момент часу, натиснувши кнопку.

Теоретично, при безперервній доставці, можливо вирішити коли буде день релізу, щотижня, два тижні або що завгодно відповідає вашим діловим вимогам. Однак, якщо дійсно потрібно отримати переваги від постійної

доставки, потрібно розгорнутись на виробництві якомога раніше, щоб переконатися, що ви пустилися невеликі партії, які легко виправити у разі виникнення проблеми.

Continuous Deployment (CD, Постійне розгортання) йде на крок далі, ніж безперервна доставка. З цією практикою кожна зміна, яка проходить всі стадії вашого виробничого конвеєра, передається вашим клієнтам. Ніякого людського втручання немає, і лише невдалий тест не дозволить розгорнути нові зміни.

Безперервне розгортання є відмінним способом прискорити цикл зворотного зв'язку зі своїми клієнтами та відійти від тиску з команди, оскільки більше не існує дня релізу. Розробники можуть зосередитися на створенні програмного забезпечення, і вони бачать, що їхня робота відбувається через кілька хвилин після того, як вони закінчать роботу над ним [4].

На зображенні на рис.1.5 нижче проілюстровано як безперервна інтеграція, безперервна доставка та безперервне розгортання взаємодіє один з одним:



Рисунок 1.5 – Взаємодія CI, CD та CD

Вартість впровадження безперервної інтеграції(CI):

- команді потрібно буде написати автоматизовані тести для кожної нової функції, поліпшення або виправлення помилок;

- команді потрібен безперервний сервер інтеграції, який може відстежувати головне сховище і автоматично виконувати тести для кожного нового натискання;

- розробникам необхідно об'єднувати свої зміни якомога частіше, принаймні раз на день.

Що отримується:

- менші кількості помилок передаються у виробництво, оскільки регресії забираються на ранній стадії за допомогою автоматизованих тестів;

- побудова релізу просте, оскільки всі питання інтеграції були вирішені на ранній стадії;

- менше перемикання контексту, оскільки розробники попереджають, як тільки вони зламають побудову і можуть працювати над його виправленням, перш ніж перейти до іншого завдання;

- витрати на тестування різко скорочуються – ваш сервер CI може виконати сотні тестів за лічені секунди;

- команда QA витрачає менше часу на тестування і може зосередитися на значних покращеннях якості культури.

Вартість впровадження безперервної доставки(CD):

- команді потрібен міцний фундамент для безперервної інтеграції, і тестовий набір команди має покривати достатню кількість кодової бази;

- розгортання необхідно автоматизувати. Тригер залишається вручну, але після запуску розгортання не повинно бути необхідності втручання людини;

- швидше за все, команді доведеться використовувати прапори перемикання, щоб неповні функції не впливали на клієнтів у продакшині.

Що отримується:

- складність розгортання програмного забезпечення зникла. Команді більше не доведеться витрачати дні на підготовку до випуску;

- команда може випускати релізи частіше, тим самим прискорюючи контур зворотного зв'язку з клієнтами;

- існує набагато менше тиску на рішення для невеликих змін, а отже, заохочує швидше переходити.

- Вартість впровадження безперервного розгортання:

- культура тестування команди повинна бути найкращою. Якість тестового набору визначатиме якість ваших версій;

- процес документації потребуватиме швидкого оновлення з приводу розгортання;

- прапори перемикання стають невід'ємною частиною процесу випуску значних змін, щоб переконатися, що можливо координувати свою діяльність з іншими відділами (підтримка, маркетинг, PR).

Що отримується:

- команда можете розвиватися швидше, оскільки немає необхідності призупиняти розробку для релізів. Конвеєри розгортання спрацьовують автоматично для кожної зміни.

- випуски менш ризиковані і легше виправити в разі виникнення проблем, коли розгортаються невеликі партії змін.

- клієнти бачать безперервний потік поліпшень, а якість збільшується щодня, а не кожен місяць, квартал або рік.

# 2 ОРГАНІЗАЦІЯ СЕРВЕРА

Сервер це - виділений або спеціалізований комп'ютер для виконання сервісного програмного забезпечення (в тому числі серверів тих чи інших завдань).

Сервером називається комп'ютер, виділений з групи персональних комп'ютерів (або робочих станцій) для виконання будь-якої сервісної завдання без безпосередньої участі людини. Сервер і робоча станція може мати однакову апаратну конфігурацію, так як розрізняються лише по участі в своїй роботі людини за консоллю.

Деякі сервісні завдання можуть виконуватися на робочій станції паралельно з роботою користувача. Таку робочу станцію умовно називають невиділений сервером.

Консоль (зазвичай – монітор, клавіатура, миша) і участь людини необхідні серверів тільки на стадії первинної настройки, при апаратно-технічному обслуговуванні та управлінні в позаштатних ситуаціях (штатно, більшість серверів управляються віддалено). Для позаштатних ситуацій сервери зазвичай забезпечуються одним консольним комплектом на групу серверів (з комутатором, наприклад, KVM-перемикачем, або без такого).

Серверне обладнання найчастіше призначене для забезпечення роботи сервісів в режимі 24/7, тому часто комплектується дублюючими елементами, що дозволяють забезпечити «п'ять дев'яток» (99,999%; час недоступності сервера або простий системи становить менше 6 хвилин в рік). Для цього конструкторами при створенні серверів створюються спеціальні рішення, відмінні від створення звичайних комп'ютерів.

Пам'ять забезпечує підвищену стійкість до збоїв. Наприклад для i386-сумісних серверів, модулі оперативної пам'яті і кеша має посилену технологію

корекції помилок (англ. Error Checking and Correction, ECC). На деяких інших платформах, наприклад SPARC (Sun Microsystems), корекцію помилок має вся пам'ять. Для власних мейнфреймів IBM розробила спеціальну технологію Chipkill [5].

Підвищення надійності сервера досягається резервуванням, в тому числі з гарячими підключенням і заміною (англ. Hot-swap) критично важливих компонентів:

при необхідності вводиться дублювання процесорів (наприклад, це важливо для безперервності виконання сервером завдання довготривалого розрахунку - в разі відмови одного процесора обчислення не обриваються, а тривають, нехай і на меншій швидкості);

- блоків живлення;
- жорстких дисків в складі масиву RAID і самих контролерів дисків;
- груп вентиляторів, що забезпечують охолодження компонентів сервера.

У функції апаратного моніторингу вводять додаткові канали для контролю більшої кількості параметрів сервера: датчики температури контролюють температурні режими всіх процесорів, модулів пам'яті, температуру в відсіках з встановленими жорсткими дисками; електронні лічильники імпульсів, вбудовані в вентилятори, виконують функції тахометрів і дозволяють, залежно від температури, регулювати швидкість їх обертання; постійний контроль напруги живлення компонентів сервера дозволяє сигналізувати про ефективність роботи блоків живлення; сторожовий таймер не дозволяє залишитися непоміченим зависання системи, автоматично виробляючи примусову перезавантаження сервера.

## 2.1 Серверні компоненти

До компонетів серверного обладнання в дипломій роботі було взято ноутбук який зображений на рис.2.1, який будет виконувати роль серверу,

його ресурсів буде достатньо щоб розгорнути тестову інфраструктуру. Основні характеристики сервера:

- процесор intel i5 3220U 2C\4T;
- оперативна пам’ять 8 DDR3 гігабайтів;
- дискова система SSD 120 гігабайтів.

## 2.2 Порівняльна характеристика операційних систем

Операційна система Linux - це велике сімейство операційних систем, яке розробляється як приватними компаніями, так і спільнотами вільних розробників. На відміну від інших операційних систем, Linux не існує в еталонному вигляді - всі види цієї операційної системи, або як їх називають, дистрибутиви, повністю розробляються своїми власними розробниками. Дистрибутиви розрізняються як за призначенням (для комп'ютерів, для серверів, для вбудованих пристроїв і т. Д.), Так і по компонентах у багатьох дистрибутивів свій власний набір додатків і утиліт. У нинішній час, кількість дистрибутивів Linux більше декількох сотень, і це без урахування занедбаних і приватних збірок.

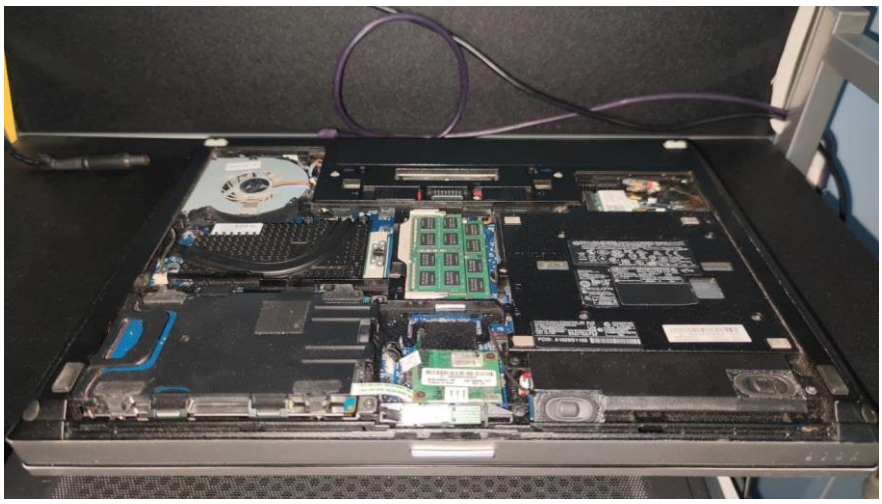

Рисунок 2.1 – Сервер

Розробка Ubuntu почалася в ті часи, коли дистрибутивів Linux, придатних для простого користувача можна було перерахувати по пальцях. Розробники Ubuntu відразу орієнтувалася на звичайних користувачів, ставлячи перед собою мету зробити зручну і регулярно оновлювану операційну систему. На сегодняший день Ubuntu є найпопулярнішим дистрибутивом Linux для персональних комп'ютерів - кількість його користувачів перевищує 20 мільйонів. Цей дистрибутив багатьма рекомендується як самий доброзичливий і простий в освоєнні Linux дистрибутив. За рахунок величезної бази користувачів, в інтернеті можна легко знайти відповідь майже на будь-яке питання [7].

Ubuntu підтримує мінімальну установку, при якій з програм буде присутній тільки найнеобхідніші системні утиліти і веб-браузер на рис 2.2 зображений веб-інтерфейс Ubuntu 20.04, а так само "повну" установку, в разі чого будуть встановлені так само різні додаткові програми, на кшталт офісного пакету LibreOffice, мультимедіа програвача і багато чого іншого .

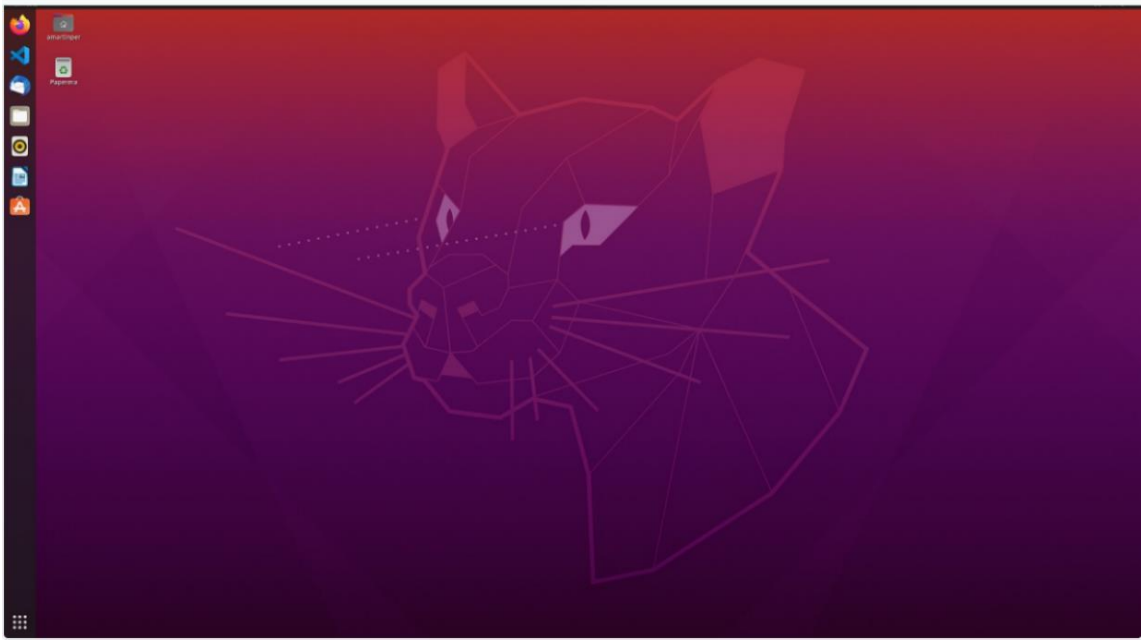

Рисунок 2.2 – Веб-інтерфейс ОС Ubuntu 20.04

Для Ubuntu розробники випускають нову версію дистрибутива кожні 6 місяців, підтримка яких триває 9 місяців, після чого для отримання оновлень безпеки потрібно оновиться на нову версію. Крім цього, кожні два роки виходить спеціальна версія LTS (підтримка протягом тривалого періоду), яка буде підтримуватися оновленнями безпеки протягом наступних 5 років.

Основний компонент кожної операційної системи - це її ядро зображеня ядра на рис 2.3. І навіть з цього, між операційними системами є величезні відмінності. Ядро Linux - монолітне, воно складається з одного файлу, а для розширення його функціональності можна використовувати модулі [8].

Всі програми спілкуються з ядром через системні виклики, вони стандартизовані, тому одні й ті ж програми без переписування зможуть працювати на різних платформах під управлінням Linux, наприклад, x86 і ARM.

Всі драйвери вбудовані в ядро, але зате більшість програм знаходяться в просторі користувача, в тому числі графічна оболонка. Монолітна структура дає більше безпеки, оскільки якщо на етапі складання ядра відключити підтримку модулів, виконати свій код на рівні ядра буде неможливо.

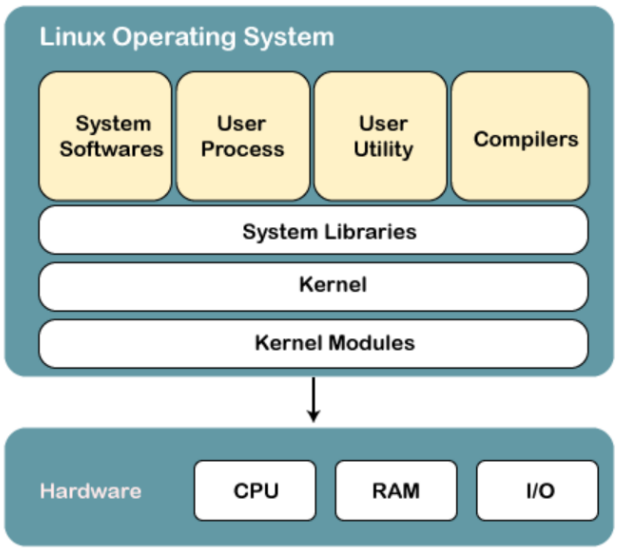


Рисунок 2.3 – Ядро системи Linux

Це головна, але не очевидна відмінність Linux від Windows. Windows має зовсім інший тип ядра. Тут використовується гібридне ядро, яке складається з безлічі невеликих частин - бібліотек dll, кожна з яких відповідає строго за свою функцію.Системні виклики не використовуються, замість них призначені для користувача програми змушені звертатися до документованих бібліотекам user32.dll, gdi32.dll, kenel32.dll, advapi32.dll. Ці бібліотеки викликають функції з ntdll.dll, яка безпосередньо пов'язана з ядром. Драйверами управляє бібліотека hal.dll і підключаються вони до ядра окремо. Висновком на екран управляє графічна підсистема ядра, туди входить вся робота з графікою, в тому числі і з оболонкою зображеня ядра winodws на рис 2.4.

Можливість використання призначеного для користувача режиму ядра дозволяє легко адаптувати систему до будь-якого типу програм, наприклад win16 або POSIX. Але за цю гнучкість доводиться платити продуктивністю [6].

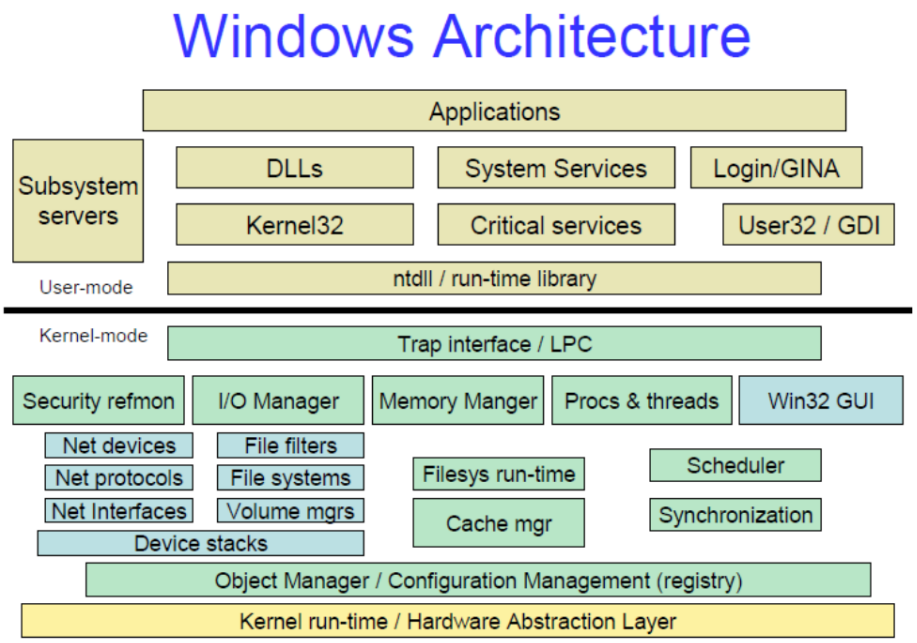


Рисунок 2.4 – Ядро системи Windows

Те, що сильно відрізняє операційну систему linux від windows так це структурою файлової системи на рис 2.5. Linux надає файлову систему більш реально, такою як вона є насправді. Починається структура файлової системи з кореня, або, іншими словами, основного каталогу системного розділу, а вже туди підключаються всі інші диски по потрібних підкаталогах [9].

Файли сортуються по каталогах в залежності від типу, наприклад, виконувані - в / bin /, настройки - / etc /, а ресурси - в / usr /. Виходить що одна програма розділена по всій файлової системи, але це не вивідує труднощів через пакетного менеджера.

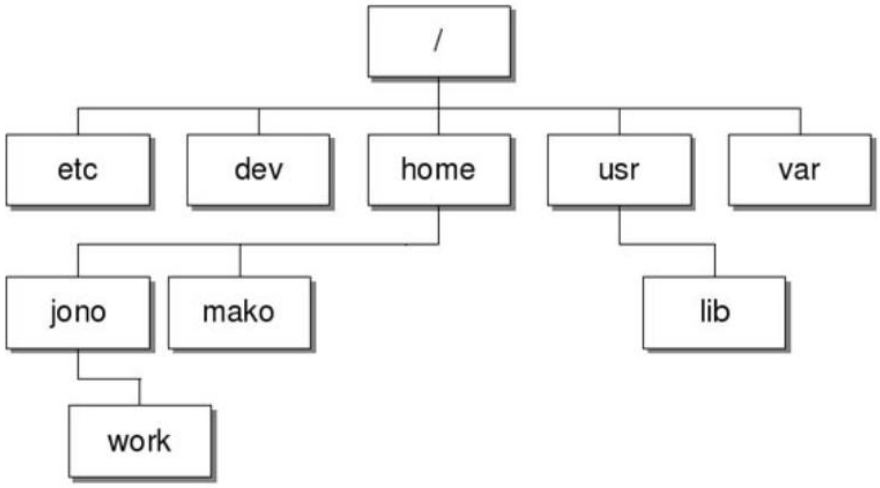


Рисунок 2.5 – Файлова система Linux

Пристрої зберігання в Linux іменуються за алфавітом, а розділи на них - цифрами. Наприклад, перший жорсткий диск буде мати ім'я sda, другий - sdb. А розділи на першому нумеруватимуться - sda1, sda2, sda3 і так далі. Розділи можуть вільно монтуватися в будь-яку потрібну папку, наприклад, в якості домашнього каталогу або / var /.

Windows створює додаткову абстракцію. Хоча диски і розділи називаються схожим чином, як і в Linux, але все це приховано операційною

системою. Користувачеві ж надається така абстракція, як диск C :, D :, E :, F: і так далі. Кожен з них - це розділ на жорсткому диску, а більш детальну інформацію від користувача система приховує. Це й на краще для новачків. Що стосується розподілу файлів, то одна програма знаходиться в одній папці, з усіма виконуваними файлами, настройками і ресурсами [10].

У Linux все настройки зберігаються в звичайних файлах, які розташовані в файлової системі. Глобальні файли налаштувань знаходяться в папці / etc /. Вони застосовні до всіх користувачів, які використовують цей комп'ютер. Налаштування користувальницьких програм знаходяться в прихованих підкаталогах домашнього каталогу користувача. Таке зберігання досить зручно, оскільки конфігураційні файли легко перенести на інший комп'ютер, а децентралізованого збільшує надійність системи. Кожна програма створює свій конфігураційний файл, зі своїм синтаксисом, і редагуються вони, в основному, вручну. Майже всі налаштування можна виконати через графічний інтерфейс, але часто графічні утиліти створюють дуже заплутані конфігурації. Ручна робота завжди виглядає краще.

Це теж важлива відмінність linux від windows. Windows зберігає всі настройки додатків, системи та драйверів в спеціальній базі даних, під назвою реєстр Windows який зображений на рис 2.6. Всі налаштування розділені по гілках і ключах, а програми можуть дуже швидко отримати до них доступ.

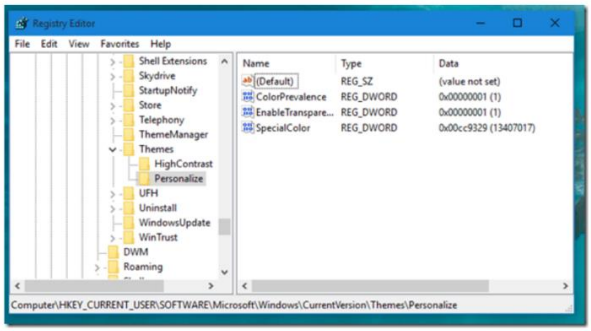


Рисунок 2.6 – Реєстр Windows

Такий спосіб надання дає за замовчуванням безпеку налаштувань, можливість віддаленого зміни і легкого їх зміни за допомогою графічних програм. Але тут криються і великі недоліки: настройки можна перенести на інший комп'ютер, централізована система налаштувань може бути пошкоджена і це може вивести систему ладу. До того ж програми дуже швидко переповнює реєстр і він починає займати занадто багато, тому на його початкове завантаження потрібно багато часу.

Linux спочатку був розроблений як багато користувачів система. Файли мають три категорії доступу - це користувач-власник, група користувачів і всі інші. Також є три параметра доступу - читання, запис і виконання. За допомогою комбінації цих простих параметрів і здійснюється контроль доступу до всіх файлів в системі, а оскільки в Linux - все є файл, значить до всього. Згодом почали вважати що така система застаріла і були доопрацьовані списки доступу ACL, SELinux і AppArrmor - вони повністю задовольняють всі потреби в безпеці. Але вони так і не набрали великої популярності.

Windows була призначена для роботи тільки одного користувача, спочатку через це виникало дуже багато проблем з безпекою. Але потім система користувачів була доопрацьована розрахована на багато користувачів система, яка, крім власника, групи та інших включає докладні ACL списки доступу. Можна сказати, що тут відмінність windows і linux не таким значним.

Продовжимо порівняння windows і linux. Управління програмами і їх оновленням це величезна різниця windows і linux, настільки все реалізовано по-іншому.

У Linux існують репозиторії пакетів програм. Там є якщо не все, то майже всі необхідні програми, драйвера і компоненти системи. У Linux качати програми з інтернету нема особливого сенсу, хоча така можливість теж є використання централізованих репозиторіїв дає велику безпеку і надійність, а також можливість поновлення. Як тільки нова версія програми з'явилася в репозиторії, ви можете її відновити. Процес оновлення виконується однією командою відразу для всієї системи, тоді, коли вам це зручно.

У Windows немає репозиторіїв, вам доведеться шукати всі необхідні програми в інтернеті і встановлювати їх вручну. Кожна програма буде оновлюватися сама, коли вважатиме за потрібне, в тому числі і система. Для оновлення системи знадобитися перезавантаження.

# 3 ОРКЕСТРАТОРИ КОНТЕЙНЕРІВ

Використання контейнерів - один з найбільш активно розвиваються сегментів IT-ринку, орієнтований на корпоративне використання хмарних послуг [11].

Контейнери на ринку віртуалізації з'явилися відносно недавно. В основному завдяки технології Docker, що дозволяє запускати додатки в контейнері. Це схоже на звичайну віртуальну машину, але набагато краще, оскільки досягається практично повна незалежність від інфраструктури і знижене споживання ресурсами. Саме тому зараз можна помітити зсув від традиційних віртуальних машин в сторону контейнерів. Великі підприємства використовують їх при створенні хмарної IT-інфраструктури.

Оркестрації - це координація взаємодії декількох контейнерів. Звичайно, можна працювати і без оркестрації - ніхто не забороняє створити контейнер, в якому будуть запущені всі необхідні процеси. Однак в цьому випадку ви будете позбавлені гнучкості, масштабованості, а також виникнуть питання безпеки, оскільки запущені в одному контейнері процеси не будуть ізольовані і зможуть впливати друг на друга.

Оркестрації дозволяє створювати інформаційні системи з безлічі контейнерів, кожен з яких відповідає тільки за одну певну задачу, а спілкування здійснюється через мережеві порти і загальні каталоги. При необхідності кожен такий контейнер можна замінити іншим, що дозволяє, наприклад, швидко перейти на іншу версію бази даних при необхідності.

Існують різні платформи для оркестрації контейнерів. Вони дозволяють реалізувати зручні та ефективні засоби розгортання контейнерних систем, побудови єдиної централізованої консолі для застосування політик управління. Найбільш відомі такі системи: Kubernetes, Docker Swarm і Apache

Mesos та багато інших.

## 3.1 Апаратна віртуалізація

Процес апаратної віртуалізації включає в себе установку гіпервизора або диспетчера віртуальних машин (VMM). Саме вони створюють рівень абстракції між програмним забезпеченням і безпосереднім «залізом» [12]. Після установки гипервизора програмне забезпечення покладається на віртуальні уявлення обчислювальних компонентів - тобто, наприклад, на віртуальні CPU замість фізичних. До найбільш популярним гіпервізором можна віднести рішення VMware та vSphere на основі ESXi і Microsoft Hyper-V. Віртуалізовані обчислювальні ресурси об'єднуються в ізольовані «групи», звані віртуальними машинами (ВМ). На підготовлену ВМ можна встановити операційну системи, а потім необхідні додатки. Віртуалізовані системи можуть мати відразу кілька віртуальних машин одночасно, при цьому кожна віртуальна машина логічно ізольована від своїх «сусідів». Таким чином, якщо одна з ВМ піддасться атаці вірусу, буде перевантажена або зламана, це ніяк не позначиться на інших машинах всередині системи. Віртуалізація дозволяє бізнесу скоротити витрати, пов'язані з придбанням серверного обладнання. Наприклад, замість покупки 10 різних серверів для розміщення 10 додатків, кожному з яких для роботи потрібна окрема ОС, досить побудувати віртуалізувати систему, яка буде містити необхідну кількість ВМ, в рамках всього одного досить потужного сервера на рис 3.1 зображено різницю між традиційною архітектурою віртуалізаццією.

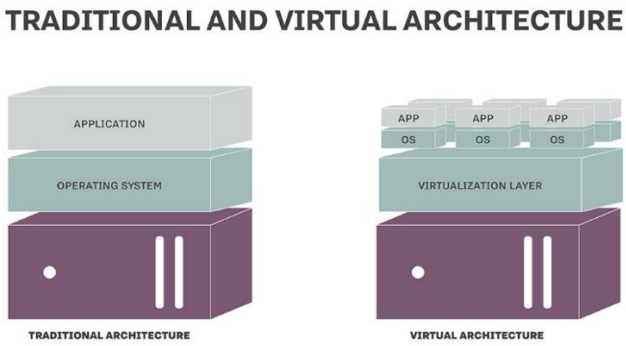


Рисунок 3.1– Класична архітектура в порівнянні з віртуалізацією

Оскільки гипервизор встановлюється безпосередньо на сервер, а операційні системи і ПО додаються пізніше, такий підхід часто називають виртуализацией «на голому залізі». В останні роки Гіпервізор стали самостійними вузько заточеними ОС. Є й альтернативний підхід, який працює в разі, якщо ви не можете / не бажаєте працювати з «голим залізом» - спершу встановлюється головна операційна система, а вже в ній розгортається гипервизор і все ВМ. Цей метод часто називають виртуализацией з хост-системою. Зараз він найчастіше використовується для віртуалізації контейнерів.

Існує кілька типів апаратної віртуалізації. Серед них - повна віртуалізація, паравіртуалізація і віртуалізація з апаратною підтримкою. Коротко поговоримо про кожного з них. Повна віртуалізація повністю імітує необхідну обладнання для гостьової операційної системи. У підсумку виходить среда, аналогічна ОС, що працює на окремому сервері. Використання повної віртуалізації дозволяє адміністраторам запускати віртуальні середовища будь-яких конфігурацій без будь-яких додаткових апаратних хитрощів. Паравіртуалізацією називається підхід, при якому на ВМ запускається особливим чином підготовлена (змінена та / або перекомпілировать) версія гостьовий ОС. Іншими словами, системні адміністратори повинні дотримуватися вихідні вимоги ОС до заліза лише

частково. Віртуалізація з апаратною підтримкою використовує апаратне забезпечення комп'ютера в якості «архітектурної підтримки» для створення повністю виртуализированной ВМ і управління нею. Віртуалізація з апаратною підтримкою була вперше представлена IBM в 1972 році з IBM System / 370. Віртуалізація з апаратною підтримкою на сьогоднішній день є найбільш поширеною формою віртуалізації. Технологія віртуалізації дала життя популярним зараз хмарних сервісів, таким наприклад як:

- віртуальніVDS / VPS-сервери;
- IaaS, інфраструктура як послуга;
- термінальні сервери в хмарі;
- VDI, віртуальна інфраструктура робочих столів;
- cloud gaming.

## 3.2 Віртуалізація на рівні операційної системи

Контейнеризація або віртуалізація на рівні операційної системи – метод віртуалізації, при якому ядро операційної системи підтримує кілька ізольованих екземплярів простору користувача замість одного. Ці екземпляри (зазвичай звані контейнерами або зонами) з точки зору користувача повністю ідентичні окремому екземпляру операційної системи. Ядро забезпечує повну ізольованість контейнерів, тому програми з різних контейнерів не можуть впливати один на одного.

На відміну від апаратної віртуалізації, при якій емулюється апаратне оточення і може бути запущений широкий спектр гостьових операційних систем, в контейнері може бути запущений екземпляр операційної системи тільки з тим же ядром, що і у хостовой операційної системи (всі контейнери вузла використовують загальне ядро). При цьому при контейнеризації відсутні додаткові ресурсні накладні витрати на емуляцію віртуального обладнання та запуск повноцінного екземпляру операційної системи, характерної при апаратної віртуалізації. Абстрагування хост–системи від контейнеризованих

додатків.

Контейнери задумані бути повністю стандартизованими. Це означає, що контейнер з'єднується з хостом або чим–небудь зовнішнім по відношенню до нього за допомогою певних інтерфейсів. Контейнеризований додаток не повиннен покладатися або якимось чином залежати від ресурсів або архітектури хоста, на якому воно працює. Це спрощує припущення про середовищі виконання програми в процесі розробки. Аналогічно, з точки зору хоста, кожен контейнер являє собою «чорний ящик». Хосту немає діла до того, що за додаток всередині.Простота масштабування.

Одним з переваг абстрагування між операційною системою хоста і контейнерами є те, що при правильному проектуванні програми, масштабування може бути простим і прямолінійним. Сервіс–орієнтована архітектура (буде розглянута далі) в комбінації з контейнеризованими додатками забезпечує основу для легкого масштабування.

Розробник може запустити кілька контейнерів на своїй робочій машині, при цьому та ж система може бути горизонтально масштабувати, наприклад, на тестовому майданчику. Коли контейнери запускаються в експлуатацію (продакшн), вони знову можуть бути масштабовані.

Контейнери дозволяють розробнику цієї програми було або компонент додатка з усіма його залежностями і далі працювати з ними як з єдиним цілим. Хосту не треба турбуватися про залежності, необхідних для запуску конкретного додатка. Якщо хост може запустити Docker, він може запустити будь–який Docker–контейнер.

Це робить легким управління залежностями і також спрощує управління версіями програми. Хост–системи більше не повинні відповідати за управління залежностями додатки, тому що, за винятком випадків залежності одних контейнерів від інших контейнерів, все залежно повинні міститися в самому контейнері надзвичайно легкі, ізольовані середовища до виконання.

Не дивлячись на те, що контейнери не надають такого ж рівня ізоляції та управління ресурсами, як технології віртуалізації, вони мають надзвичайно

легкої середовищем виконання. Контейнери ізольовані на рівні процесів, працюючи при цьому поверх одного і того ж ядра хоста. Це означає, що контейнер не включає в себе повну операційну систему, що призводить до практично миттєвого його запуску. Розробники можуть легко запустити сотні контейнерів зі своєю робочою машини без будь–яких проблем.

Контейнери легкі ще і в тому сенсі, що вони зберігаються «пошарово» на рис. 3.2. Якщо кілька контейнерів засновані на одному і тому ж шарі, вони можуть спільно використовувати цей базовий шар без дублювання, що призводить до мінімальному завантаженні дискового простору в наступних образах.

Docker–файли дозволяють користувачам задати конкретні дії, необхідні для створення нового образу контейнера. Це дозволяє задавати настройки середовища виконання так, як ніби–то це код, при бажанні зберігаючи ці настройки в системі контролю версій. Однаковий Docker–файл, зібраний в одному і тому ж оточенні, завжди створить ідентичний образ контейнера.Використання Docker–файлів для повторюваних, сумісних збірок (білдів).

Коли є можливість створити образи контейнерів в інтерактивному режимі, зазвичай краще записати кроки конфігурації в Docker–файл. Docker–файли це прості файли збірки (build–файли), які описують процес створення образу контейнера з відомої початкової точки.

Docker–файли неймовірно корисні і ними досить легко почати користуватися. Ось лише деякі з переваг, які вони надають:

- простота роботи з версіями, тобто Docker–файли можуть бути збережені в системі контролю версій для відстеження змін і «відкатування» помилок;

- передбачуваність, тобто бірка образів з Docker–файлу допомагає позбутися від людських помилок в процесі створення образів;

- контрольованість, тобто якщо хтось плануєте ділитися своїм образом, зазвичай хорошим тоном є надання Docker–файлу, який створює даний образ,

щоб інші користувачі могли проконтролювати процес. Таким чином, по суті, надається історія зроблених для побудови образу кроків;

- гнучкість, тобто створення образів з Docker–файлу дозволяє перевизначити налаштування, які задані за замовчуванням в інтерактивному режимі. Це означає, що Вам треба виробляти менше дій, щоб отримати правильно функціонуючий образ.

Docker–файли є відмінним інструментом автоматизації побудови образів контейнерів для створення повторюваного процесу їх побудови.

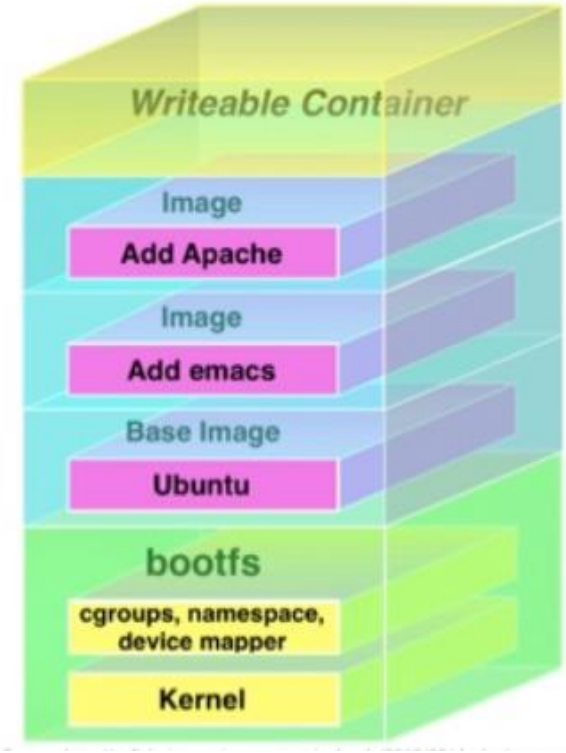


Рисунок 3.2 – Контейнер у вигляді шарів

## 3.3 Віртуалізація та контейнеризація

Контейнери Docker і віртуальні машини - це обидва способи розгортання програм у середовищах, які ізольовані від базового обладнання. Головна відмінність - рівень ізоляції.

За умови виконання контейнера, такого як Docker, додаток знаходиться в ізольованому середовищі всередині функцій ізоляції, які надає контейнер, але все одно використовує те саме ядро, що й інші контейнери на тому ж хості. Як результат, процеси, що виконуються всередині контейнерів, видно із хост-системи (їм надано достатньо привілеїв для переліку всіх процесів). Наприклад, якщо запустити контейнер MongoDB за допомогою Docker, і виконати команду ps -e | grep mongo у звичайній оболонці на хості (не в Docker), процес буде видно. Наявність кількох контейнерів спільного використання ядра дозволяє кінцевому користувачеві збирати багато та багато контейнерів на одній машині з майже моментальним часом запуску. Крім того, як наслідок, контейнери, яким не потрібно вбудовувати повну ОС, вони дуже легкі, зазвичай близько 5-100 МБ.

На відміну від цього, у віртуальній машині все, що працює у віртуальній машині, не залежить від операційної системи хоста або гіпервізора. Платформа віртуальної машини запускає процес (який називається монітором віртуальної машини, або VMM) з метою управління процесом віртуалізації для конкретної ВМ, а хост-система виділяє деякі свої апаратні ресурси у ВМ. Однак, що принципово відрізняється від віртуальної машини, це те, що під час запуску вона завантажує нове, виділене ядро для цього віртуального середовища і запускає (часто досить великий) набір процесів операційної системи на рис 3.3 зображено порівняння архітектури віртуальної машини від контейнеру. Це робить розмір віртуальної машини набагато більшим, ніж типовий контейнер, який містить лише додаток. Остнні роки набирає популярності ьезсерверні обчислення, безсерверні додатки розбиваються на функції і розміщуються у сторонніх постачальників, які стягують плату з розробника додатків лише залежно від часу, який виконує кожна функція. Це направлення останніми роками дуже активно розвивається, але в будь-якому разі ця технологія використовує контейнери в яких працюють додатки.

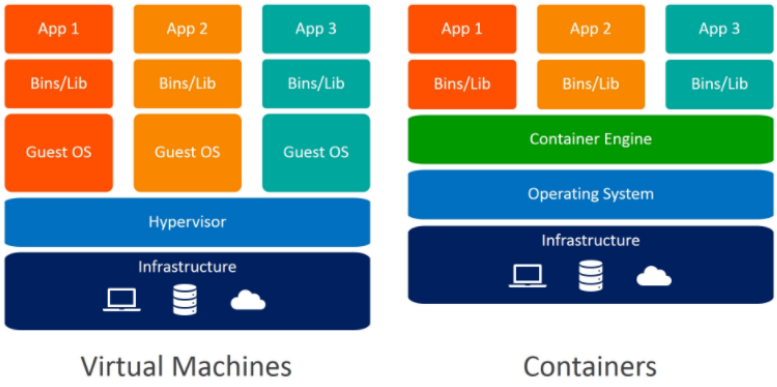


Рисунок 3.3 – Архітектура віртуальної машини і контейнера

Запуск спеціального ядра та ОС має кілька переваг. Безпека ізоляції є однією з основних. Наприклад, з точки зору головної системи, неможливо дізнатися, що працює у віртуальній машині. Оскільки ядро ділиться між контейнерами на одному хості, існує більший (проте, все ще дуже малий) ризик того, що злоумисник зможе уникнути його стримування та отримати доступ до основного хоста. З віртуальною машиною це складніше, оскільки ядра не використовуються спільно, що зменшує поверхню атаки до базового обладнання.

Розгянемо детальніше технічні подробиці про різницю між часом роботи контейнера та гіпервізором віртуальної машини, або про те, як реалізовано знімок сховища контейнера порівняно з тим, як ви зберігаєте дані у віртуальній машині; але ці відмінності, як правило, не настільки важливі для того, хто вирішує, використовувати контейнери або віртуальну машину. В яких випадках можливо вибрати Docker, а також ті, де віртуальні машини краще підходять, або в яких випадках є можливим вибрати обидва.

Контейнери - хороший вибір для більшості навантажень програми. Контейнери більш підходять, зокрема, якщо наступне є пріоритетом:

- час початку;

- Контейнери Docker зазвичай запускаються за кілька секунд або менше, тоді як віртуальні машини можуть зайняти хвилини. Таким чином, робочі навантаження, які потрібно розпочати дуже швидко, або які включають постійне обертання додатків вгору-вниз, можуть добре підійти для Docker.

- ефективність;

- оскільки контейнери Docker ділять багато своїх ресурсів із хост-системою, для їх запуску потрібно встановити менше речей. Порівняно з віртуальною машиною, контейнер, як правило, займає менше місця та споживає менше оперативної пам'яті та процесорного часу. З цієї причини ви часто можете вмістити більше програм на одному сервері, використовуючи контейнери, ніж виртуальні машини. Подібним чином, через нижчий рівень споживання ресурсів, контейнери можуть допомогти заощадити гроші на хмарних обчисленнях.

- ліцензування;

- більшість основних технологій, необхідних для розгортання контейнерів Docker, включаючи час роботи контейнерів та організаторів, таких як Kubernetes, є безкоштовними та з відкритим кодом. Це може призвести до економії витрат, одночасно збільшуючи гнучкість. (Але варто зазначити, що в багатьох випадках організації використовують комерційний дистрибутив Docker або Kubernetes, щоб спростити розгортання та отримати послуги професійної підтримки.)

- повторне використання коду.

Кожен запущений контейнер базується на зображенні контейнера, яке містить двійкові файли та бібліотеки, необхідні контейнеру для запуску даної програми. Зображення контейнерів легко створити за допомогою файлів Dockerfiles. Їх можна спільно використовувати та використовувати повторно, використовуючи реєстри контейнерів, які в основному є сховищами, що містять образи контейнера. Ви можете створити внутрішній реєстр для спільного використання та повторного використання контейнерів у вашій

компанії. Тисячі готових зображень можна безкоштовно завантажити із загальнодоступних реєстрів (наприклад, Docker Hub або Quay.io) і використовувати як основу для створення власних контейнерних програм.

Звичайно, віртуальні машини також можуть бути упаковані у зображення, і цими зображеннями також можна ділитися, але не настільки ефективно та легко, як контейнери. Крім того, зображення віртуальних машин не так легко автоматично створити, і вони, як правило, більших розмірів. Крім того, оскільки вони зазвичай включають операційні системи, їх перерозподіл може юридично ускладнитися. (У більшості випадків ви не можете легально завантажити та запустити образ віртуальної машини з попередньо встановленою Windows, наприклад, не маючи ліцензії Windows.)

Розглянемо деякі причини, чому варто відмовитись від Docker і дотримуватися своїх віртуальних машин.

Віртуальні машини більш ізольовані один від одного та від хост-системи, ніж контейнери Docker. Це пояснюється тим, що, віртуальні машини безпосередньо не діляться ядром та іншими ресурсами з основною системою.

З цієї причини віртуальні машини в цілому більш безпечні, ніж контейнери. Незважаючи на те, що Docker пропонує різні інструменти, які допомагають ізолювати контейнери та запобігти проникненню порушень в одному контейнері в інші, в кінці кінців контейнери не ізолюються з точки зору безпеки так само, як і віртуальні машини.

Сьогодні більшість платформ віртуальних машин працюють на всіх основних операційних системах, та стає можливим запускати будь-який тип операційної системи, який потрібно на віртуальних машинах. Таким чином, можете розгорнути віртуальну машину Windows на Linux, або навпаки. Ця портативність зручна, якщо є інфраструктура, де потрібно мати можливість розгорнути один тип операційної системи на іншому.

Docker не такий портативний. Хоча певним чином Docker зменшує залежність від вашої операційної системи (наприклад, можемо запускати один і той же контейнер Docker на Ubuntu або CentOS, хоча кожен з цих

дистрибутивів Linux використовує інший тип системи управління пакетами), Docker не забезпечує портативність в операційних системах. Контейнери Docker для Linux працюють лише на хостах Linux, і те саме стосується і Windows. (Крім того, Docker працює лише на певних версіях Windows, що є ще одним обмеженням переносимості.)

Багато сучасних платформ віртуальних машин дозволяють легко “робити знімки” віртуальних машин та “відкатувати” машину за бажанням. Це може бути корисним при вирішенні питань пошкодження даних або порушень безпеки, серед інших питань.

Docker не пропонує однотипних функціональних можливостей. Можливо відкотити контейнерні зображення, але оскільки контейнери в більшості випадків зберігають свої дані поза зображенням, відкатування зображення не допоможе вам відновити дані, втрачені запущеною програмою. Це також не обов’язково допоможе вам зупинити порушення безпеки, якщо тільки порушення не було спричинене проблемою в певній версії образу контейнера.

## 3.4 Відмінності між PaaS, SaaS, IaC

IaaS, PaaS або SaaS - це моделі надання хмарних сервісів. Те, як вони співвідносяться один з одним, часто зображують у вигляді піраміди з різним рівнем контролю інформації. Вершина - це кінцевий користувач, який працює з особистими даними, «загорнутими» у вигляді програми або сервісу з зручним інтерфейсом. Програма або сервіс розгортаються на якійсь технологічній платформі, це другий рівень піраміди. Нарешті, її основа - це інфраструктура: віртуальні сервери, обчислювальні потужності, накопичувачі і канали зв'язку [13].

SaaS (Software-as-a-Service). Ця хмарна модель - найпоширеніша. Програми та сервіси розробляє і обслуговує провайдер, розміщує їх в хмарі і пропонує кінцевому користувачеві через браузер або додаток на його ПК.

Клієнт лише вносить абонплату (або користується сервісом безкоштовно), оновленням і технічною підтримкою програм займається провайдер. SaaS-сервіси можуть надавати місце для зберігання файлів (Dropbox), офісний пакет документів для роботи (Google Doc, Microsoft Office 365), допомагати організовувати фотографії (Flickr) або спілкуватися з іншими людьми (Facebook). Основний клієнт SaaS-сервісів - звичайний користувач на рис 3.4 зображено моделі надання хмарних послуг.

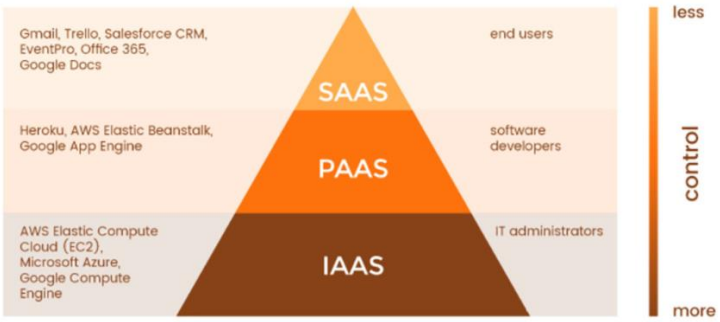


Рисунок 3.4 – Модель надання хмарних послуг

PaaS (Platform-as-a-Service). В цьому випадку хмарний провайдер надає доступ до операційних систем, засобів розробки і тестування, системам управління базами даних. Провайдер контролює не тільки сервери, системи зберігання даних і обчислювальні потужності, але також пропонує користувачеві на вибір певні платформи і засоби управління ними. Приклади PaaS: Google App Engine, IBM Bluemix, Microsoft Azure, VMWare Cloud Foundry. Користувачі PaaS-сервісів - це розробники ПЗ.

IaaS (Infrastructure-as-a-Service). При цій моделі споживач отримує інформаційно-технологічні ресурси - віртуальні сервери з певною обчислювальною потужністю та обсягами пам'яті. Всім «залізом» займається провайдер. Він встановлює на нього ПО для створення віртуальних машин, але не займається установкою і підтримкою ПЗ користувача. Провайдер

контролює тільки фізичну та віртуальну інфраструктуру. Приклади IaaS: IBM Softlayer, Hetzner Cloud, Microsoft Azure, Amazon EC2, GigaCloud. Клієнти IaaS - це системні адміністратори компаній.

З точки зору кінцевого користувача SaaS - сама зрозуміла і зручна хмарна модель. Часто простіше і ефективніше використовувати готовий SaaS-сервіс, який вже відповідає певним вимогам. Але готові рішення не завжди існують, і в такому випадку моделі PaaS і IaaS - незамінні.

## 3.5 Порівняльна характеристика контейнерних оркестраторів

Перед вибором оркестротора потрібно розуміти, що не всі інструменти оркестрації однакові, діло не стільки в тому, що одні оркестратори краще інших, скільки в тому, що визначені рішення для оркестрації мають певні сильні та слабкі сторони та специфічні особливості, які можуть мати або не мати відносин до кінцевих задач, в особливості при виборі орекстратору важливим чинником є гнучкість та переносимість інфраструктури. Нижче перелік деяких найпопулярних рішень.

Docker Swarm - це власне рішення Docker, розроблене для оркестрації та управління кластерами, введене в основний двигун Docker як  в 2016 році. Swarm, по суті, дозволяє користувачам брати кластери хостів Docker і об'єднувати їх в одну віртуальну систему або хост, який Docker викликає "swarm", який сам по собі складається з одного або декількох вузлів менеджера та робочого Докера.

Основні компоненти Docker включають:

- manager node - зазвичай обробляє оркестрацію та управління кластерами;
- worker node - запуск завдань, доручених вузлами менеджера.

Docker Swarm дозволяє копіювати, оновлювати та розгортати програми у якості служб, при цьому надаючи системним адміністраторам можливість збільшувати або зменшувати ітерації контейнерів, щоб адаптуватися до змін

вимог комп'ютера. Користувачі можуть взаємодіяти безпосередньо з Docker API, що надає їм доступ до Docker Tools та командного рядка Docker, які використовуються для розгортання контейнерів. За словами Докера, Swarm здатний одночасно запускати до 30000 контейнерів і скупчуватися до 1000 вузлів, не порушуючи працездатності системи.

Однією з найпомітніших переваг Swarm є те, як легко налаштувати та запустити.

Kubernetes, який, як вважають, є найближчим конкурентом Docker Swarm та найпопулярнішим інструментом організації контейнерів. Розроблений Google, Kubernetes - це платформа з відкритим кодом, яка організовує розгортання, управління та масштабування контейнерних програм. Основні компоненти Kubernetes включають:

- cluster складається з вузлів, які запускають конфігурацію ресурсів, що використовуються інфраструктурою Kubernetes для запуску програм.

- вузли, контейнерні хости, що взаємодіють із службами кластера. Група контейнерів в одному стручку може спілкуватися один з одним через localhost;

- labels призначені контейнерам, які ідентифікують їх як частину стручка;

- replication controller планує стручки через кластер Kubernetes.

Разом ці компоненти дозволяють перетворювати контейнери в найрізноманітнійші програми.

Хоча Kubernetes є досить не простим при першому погляді, це одне з найбільш зручних рішень для організації контейнерів. Інструмент пропонує функціональні можливості, які забезпечують доступність, портативність, масштабованість та безпеку. Крім того, Kubernetes - це випробувана платформа, що базується на більш ніж 15-річному досвіді, накопиченому під час запуску робочих навантажень Google. З точки зору надійності, Kubernetes схвалений Фондом хмарних обчислювальних технологій (CNCF) і має 1200 співробітників та понад 50 000 комісій.

Kubernetes дозволяє користувачам швидко реагувати на вимоги комп'ютера шляхом масштабування або, навпаки, зменшення масштабу функцій. Це дає свободу вибору операційних систем, хмарних платформ, середовищ виконання контейнерів та архітектур процесів, серед усіх інших, що дозволяє покращити продуктивність, розподіляючи робоче навантаження по доступних ресурсах.

Amazon Elastic Container Services (ECS) є масштабованим і потужним рішенням для організації контейнерів, яке дозволяє користувачам контролювати, керувати та розгортати контейнери Docker та запускати контейнерні програми на веб-службах Amazon (AWS).

Слід зазначити, що контейнери, організовані Amazon ECS, можна запускати лише на Amazon Web Services EC2, оскільки немає підтримки сторонньої інфраструктури.

Найбільшою перевагою використання Amazon ECS для організації контейнерів є його безперервна інтеграція з існуючими службами AWS.

Amazon ECS дозволяє командам користувачів створювати практично будь-який тип контейнерних додатків, починаючи від архітектур мікросервісів та тривалих програм, закінчуючи технологіями машинного навчання та пакетних завдань.

Отже для поставлених задач, для нас підійде будь який оркестратор з розглянутих, але в кваліфійкаційній роботі я обрав Kubernetes тому що він повністю безкоштовний, та досить популярний.

Варто зазначити розглядаючи контейнерні оркестратори неможливо не задіти такі досить популярні оркестратори як Kubernetes та Openshift маючи однакову структуру, вони обидва досить різні.

Google розробив Kubernetes як відкриту та портативну платформу Container as-a-a-Service, що дозволяє організації адмініструвати послуги для: автоматизованого розгортання, масштабування та управління робочими навантаженнями додатків. Kubernetes, прийнятий Фондом хмарних нативних обчислень (CNCF), надає такі нестандартні функції, як:

- автоматизація процесів;
- балансування навантаження;
- самоконтроль;
- організація зберігання.

OpenShift Red Hat - це хмарна платформа для контейнерів, яка функціонує як платформа як послуга, так і як механізм оркестрації контейнерів. За своєю суттю OpenShift - це інструмент з відкритим кодом, який використовує платформу Kubernetes для управління контейнерами Docker для послідовного:

- управління навантаженням;
- самоконтроль;
- централізоване забезпечення політики.

За допомогою OpenShift розробники можуть розгортати контейнерні програми в інтегрованому середовищі розробки (IDE) на рис 3.5 зображено архітектурні особливості Openshift, одночасно використовуючи Kubernetes для управління ними.

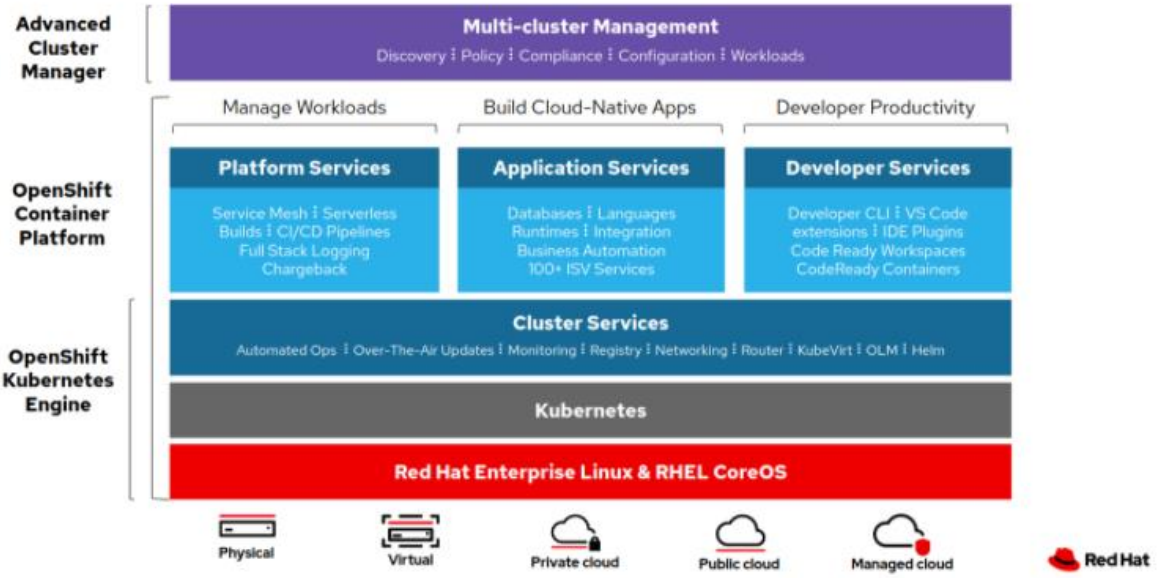


Рисунок 3.5 – Архітектурні особливості Openshift

Kubernetes, і OpenShift працюють на ліцензії Apache 2.0 та допомагають у масштабному управлінні та розгортанні додатків. Однак існують принципові відмінності в тому, як кожна з цих технологій забезпечує свою функціональність.

Kubernetes був створений як фреймворковий проект з відкритим кодом. Управління K8s здійснюється завдяки постійній співпраці між користувачами світової спільноти розробників. Хоча це означає, що підтримка робиться своїми руками, ви також отримуєте вигоду від знань та співпраці в рамках спільноти з відкритим кодом [14].

OpenShift доступний як:

- комерційний продукт (контейнерна платформа OpenShift);
- публічна хмара (OpenShift Online і OpenShift Dedicated).

Контейнерна платформа OpenShift розробляється, управляється та адмініструється розробниками Red Hat Inc. і постачається з платною підпискою на адміністрування та управління інфраструктурою. Це означає, що користувачі отримують спеціальну підтримку з періодичними оновленнями. З іншого боку, версія OpenShift з відкритим кодом, також відома як OKD, є платформою спільноти, яка обмежена «самопідтримкою».

Зі зростанням популярності та випадків використання Kubernetes, список інструментів, сумісних з Kubernetes, зріс в геометричній прогресії. Це залишається важливим фактором серед розробників, який, по суті, пропонує свободу вибору щодо платформи, складності та модернізації.

Що ще важливіше, ширше прийняття Kubernetes пов’язано також з тим, що Kubernetes керує службами на всіх трьох основних публічних хмарних платформах:

- GKE для GCP від Google;
- AKS для Azure;
- EKS для Amazon AWS.

Навпаки, OpenShift пропонує обмежені можливості установки, оскільки ви можете встановити його лише на трьох дистрибутивах Linux:

- для OpenShift 3 необхідно буде використовувати Red Hat Atomic або Red Hat Enterprise Linux (RHEL);
- щоб встановити OpenShift 4, необхідно RedHat CoreOS;
- для встановлення версії з відкритим кодом (OKD) потрібно CentOS або RHEL.

Kubernetes не має вбудованих можливостей для автентифікації та авторизації. Це обумовлює необхідність ручного створення процедур автентифікації, таких як токени для безпеки. Якщо протокол безпеки в Kubernetes не є чітко визначеним, трафіку в кластері Kubernetes за замовчуванням також бракує шифрування. Як результат, нестандартний екземпляр Kubernetes зазвичай вважається більш сприйнятливим до векторів атаки. OpenShift має сувору і чітко визначену політику безпеки. Наприклад, OpenShift обмежує роботу контейнерів Docker як прості зображення. Щоб підтримувати мінімальний рівень безпеки на OpenShift, вам потрібно мати певні права адміністратора. Більше того, OpenShift пропонує інтегрований сервер для полегшення автентифікації та авторизації.

Веб-інтерфейс Kubernetes складний і не рекомендується новачкам. Щоб отримати доступ до веб-графічного інтерфейсу Kubernetes, вам потрібно буде встановити інформаційну панель Kubernetes і переслати адресу порту локальної машини на сервер кластера за допомогою kube-proxy. Оскільки на інформаційній панелі відсутня сторінка входу, вам також доведеться створити маркери-носії, що полегшують авторизацію та автентифікацію.

І навпаки, OpenShift має інтуїтивно зрозумілу веб-консоль зі сторінкою входу в один дотик. Консоль надає простий інтерфейс на основі форм, що дозволяє легко змінювати, додавати та видаляти ресурси. Також простіше візуалізувати свої ролі кластера, проекти та сервери.

Kubernetes - це фреймворк з відкритим кодом, який можна розгорнути на:

- будь-яка основна загальнодоступна хмарна платформа: Azure, AWS або GCP;

- будь-який дистрибутив Linux, включаючи Ubuntu та Debian.

Щоб розгорнути контейнери Kubernetes, вам знадобляться об’єкти Kubernetes, які можуть легко обробляти паралельні оновлення. Для розгортання OpenShift вам знадобляться дистрибутиви RHEL, Red Hat CoreOS або CentOS Linux. Розгортання виконується за допомогою команди DeploymentConfig, яка не може бути реалізована за допомогою контролерів, а може використовуватися за допомогою виділених логічних логік. Хоча DeploymentConfig не підтримує одночасні оновлення, такі як об'єкти Kubernetes, розгортання OpenShift, однак, має певні переваги, включаючи керування версіями та автоматичні тригери розгортання.

Kubernetes не пропонує комплексного рішення CI / CD. Однак ви можете поєднати його з такими інструментами, як автоматизований моніторинг, тестування та сервери CI, щоб допомогти вам створити цілий конвеєр CI / CD. Крім того, сторонні плагіни, такі як CircleCI, можуть допомогти в швидкому побудуванні швидших конвеєрів CI / CD у Kubernetes.

Хоча OpenShift також не є повноцінним рішенням CI / CD, вбудований сертифікований контейнер Jenkins виконує роль сервера CI [15]. На рис 3.6 показано типовий конвеєр CI / CD, сформований з Jenkins (для двигуна CI / CD), Git, сховища Nexus та SonarQube (для SAST).

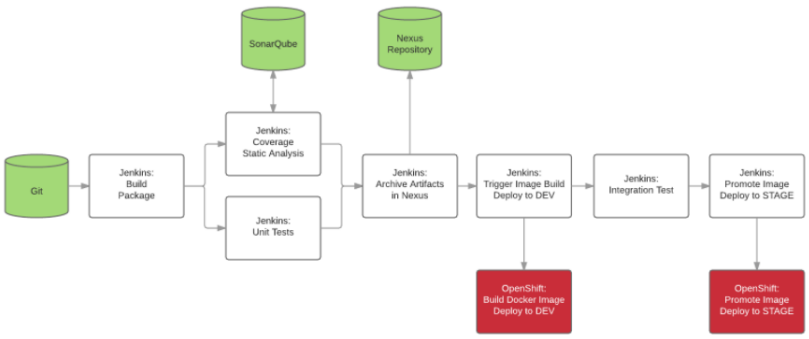


Рисунок 3.6 – Типовий конвеєр CI / CD

Можливо налаштувати власний реєстр Docker у Kubernetes, але платформа не має інтегрованого реєстру зображень. Kubernetes, однак, дозволяє витягувати зображення з приватного реєстру, щоб створити власні блоки.

Навпаки, завдяки вбудованому реєстру зображень OpenShift легко поєднується з DockerHub або Red Hat. Розробники можуть легко шукати та керувати зображеннями контейнерів за допомогою потоків зображень.

Kubernetes підтримує багаторазові оновлення, що відбуваються одночасно. Оновлення є простим, оскільки вам потрібно лише викликати команду оновлення kubeadm, щоб отримати останню версію. Завжди рекомендується зробити резервну копію існуючих інсталяційних файлів перед оновленням до нової версії.

OpenShift не підтримує кілька одночасних оновлень автоматично. Вам потрібно буде отримати доступ до системи управління пакетами Red Hat Enterprise Linux, за допомогою якої ви зможете встановити останню версію OpenShift.

Kubernetes, і OpenShift - чудові варіанти широкомасштабного розгортання контейнерних програм. Хоча Kubernetes популярний у більшості організацій завдяки своїм гнучким варіантам розгортання, OpenShift пропонує спеціальну підтримку та має безліч вбудованих компонентів, що спрощують контейнеризацію додатків, роблячи його популярним як у методологіях Agile, так і в DevOps.

Kubernetes, безперечно, є кращим вибором для додатків з високим попитом, які потребують постійного оновлення; OpenShift пропонує комплексне рішення, яке ідеально підходить для компаній, які потребують постійної та цілеспрямованої підтримки.

### 3.5.1 Порівняльна характеристика систем котролю версій коду

Система контролю версії відноситься до процесів, що вивчають систематизацію версій, об'єднаних при редагуванні та спільній роботі. але

контроль версії, як термін, розглядається в контексті розробки програмного забезпечення, власне делегації, він необхідний для професіоналів різних відраслей. Для найбільших проектів за розробником програмного забезпечення системи контролю версії відслідковують безліч змін у вихідному коді [17].

Усі системи контролю версії мають наступні можливості:

- дозволяють команді програмістів одночасно працювати над одним і тим же проектом;
- мінімізують конфлікти між розробниками, які працюють над одним проектом;
- автоматично створюють архів кожної версії, включаючи в себе всі зміни проекту.

Існує три основні групи управлінням версіями:

- згідно з розподілом репозиторію: централізовані та розподілені;
- відповідно до методів перевірки злиття та передачі коду: блокування, використання злиття до фіксації та виконання фіксації до слівання;
- система управління версіями може виконувати невеликі операції або операції з файлами.

CVS, є найпопулярнішою та широко застосовуваною системою контролю версії на сьогоднішній день. Після випуску в 1986 році вона швидко стала загальноприйнятим стандартом. CVS придбала популярність завдяки простої системі підтримки файлів та ревізії в актуальному стані.

Існує ряд IDE для CVS, включаючи Xcode (Mac), Eclipse, NetBeans та Emacs.

Переваги:

- це перевірена система часу, яка використовується більше трьох десятиліть;
- існує багато IDE, які використовують CVS.

Недоліки:

- переміщення або переіменування файлів не включається в оновлення

версій;

- передостання символічних посилань на файли, пов'язані з деякими ризиками безпеки;

- відсутність підтримки атомарних операцій може сприяти пошкодженню вихідного коду;

- повільні операції установи міток та розгалуджння;

- слаба підтримка двоїчних файлів.

- SVN, ще одна поширена система управління версіями. Більшість проектів з відкритим вихідним кодом та найбільшими платформами, такі як Ruby, Python Apache, використовують SVN. З-за величезної популярності існує безліч версій та доступних IDE.

Переваги системи контролю версії SVN:

- нова і значно покращена система, заснована на CVS;

- допускає атомарні операції;

- операції в гілці проекту малозатратні;

- доступні різні плагіни IDE.

Недоліки:

- може видавати помилки при зміни назви файлів та каталогів;

- недостатньо команд для управління репозиторієм;

- SVN працює повільнійше за порівнянням з іншими системами управління версіями.

GIT, Завдяки розподіленій формі управління без необхідності використання оригінального програмного забезпечення багатьох проектів з відкритим вихідним кодом віддають перевагу Git.

Переваги:

- майже всі негативні риси CVS / SVN усунені;

- висока швидкість роботи розподіленої системи контролю версії;

- легкість проведення різних операцій з гілками проекту;

- користувачі можуть отримати доступ до повної деревної історії в режимі офлайн;

- пропонує високу розподілену одноразову модель.

Недоліки:

- високий поріг входження для користувачів SVN;
- обмежена підтримка Windows порівнюючи з Linux.

Mercurial, вважається ефективної для великих проектів, в яких бере участь багато розробників і дизайнерів. Mercurial - це високопродуктивна система, яка пропонує оптимальну швидкість. Вона також відома своєю простотою і докладною документацією.

Переваги Mercurial системи контролю версій:

- низький поріг входження в порівнянні з Git;
- детальна документація;
- розподілена модель;
- високопродуктивна система з відмінною швидкістю.

Недоліки:

- не можна об'єднати дві батьківські гілки;
- заснована на розширеннях, а не сценаріях;
- недостатньо гнучка, щоб виконувати операції за замовчуванням.

Bazaar, унікальна тим, що може використовуватися з розподіленою і централізованою базою коду. Це робить її універсальною системою контролю версій. Крім цього Bazaar дозволяє використовувати детальний рівень управління. Її можна легко розгорнути в самих різних сценаріях, що робить її адаптивної і гнучкою для всіляких проектів.

Переваги:

- ідеально підходить для різноманітних проектів;
- надає набагато більш просунутий набір команд і підтримку IDE;
- включає в себе настроюється набір функцій, який підходить для різноманітних проектів.

Недоліки:

- є новою і недостатньо проробленою системою управління версіями;
- відсутня підтримка IDE.

### 3.5.2 Вибір інструментів для CI частини

В якості основного агрегатора CI частини в кваліфікаційній робі було вибрано систему Jenkins [16].

Дженкінс пропонує простий спосіб встановити середовище безперервної інтеграції або безперервної доставки (CI / CD) для майже будь-якої комбінації мов та сховищ вихідних кодів, що використовують конвеєри, а також автоматизує інші рутинні завдання розробки. Незважаючи на те, що Дженкінс не усуває необхідності створювати сценарії для окремих кроків, він дає швидший та надійніший спосіб інтегрувати весь ланцюжок інструментів побудови, тестування та розгортання.

Сьогодні Дженкінс є провідним сервером автоматизації з відкритим кодом і має близько 1600 плагінів для підтримки автоматизації всіх видів завдань розробки. Проблема, яку спочатку намагався вирішити Кавагучі, безперервна інтеграція та безперервна доставка коду Java (тобто створення проектів, запуск тестів, аналіз статичного коду та розгортання) - це лише один із багатьох процесів, які люди автоматизують за допомогою Дженкінса. Ці 1600 плагінів охоплюють п’ять областей: платформи, інтерфейс, адміністрування, управління вихідним кодом і, найчастіше, управління збірками.

Jenkins поширюється як архів WAR та як пакети інсталятора для основних операційних систем, як пакет Homebrew, як образ Docker та як вихідний код. Вихідним кодом є переважно Java, з кількома файлами Groovy, Ruby та Antlr.

Jenkins можливо запустити WAR окремо або як сервлет на сервері програм Java, наприклад Tomcat. У будь-якому випадку він створює веб-інтерфейс користувача та приймає виклики до свого REST API.

Jenkins створює адміністративного користувача з довгим випадковим паролем, який можно вставити на початкову веб-сторінку, щоб розблокувати інсталяції, інтерфейс jenkins зображено на рис 3.7.

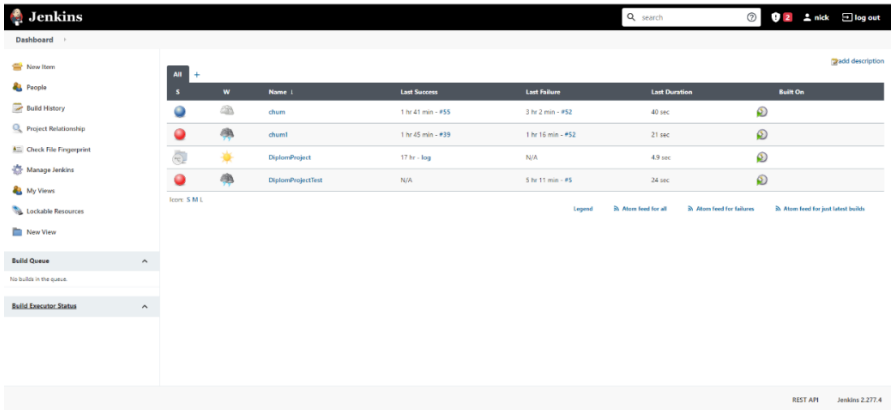


Рисунок 3.7 – Веб інтерфейс утиліти jenkins

Jenkins найпопулярнійша утиліта для автоматизації CI та CD процесів, він не складний у розумінні так як використовує декларативну могу програмування, а також Groovy та має зручний веб-інтерфейс.

Завдяки великої кількості плагінів його можна інтегрувати майже під будь-яку задачу та архітектуру.

# 4 ОРГАНІЗАЦІЯ KUBERNETES КЛАСТЕРУ

Для того щоб почати встановлювати Kubernetes на потрібно встановити залежні пакети на Ubuntu 20.04. Треба увімкнути апаратну віртуалізаію перед початком виконання команд на сервері.

Базові пакети для підготовки системи до встановлення kubernetes:

- початкове оновлення системи – sudo apt update;
- встановлення необхідних системних утиліт – sudo apt install net-tools;
- встановлення ssh серва – sudo apt install openssh-server;
- встановлення python версії 2 та 3 – sudo apt install python3 та sudo apt install python;
- встановлення virtualBox sudo apt install virtualbox;
- встановлення ansible sudo apt install ansible;
- встановлення vagrant sudo apt install vagrant;
- встановлення git sudo apt install git.

Існує багато способів встановлення kubernetes кластеру, в кваліфікаційній роботі був обраний метод розгортання kubernetes кластеру за допомогою kubeadm цей метод є класичним та універсальним, цей метод реомендований Google для Kubernetes. Кластер буде розгорнутий за допомогою vagrant файлу , який першим этапом підніме віртуальні машини на яких в подальшому буде встановлений kubernetes.

## 4.1 Початкове налаштування Kubernetes

Для початку необхідно з’ясувати які технічні вимоги повнні мати ноди на яких буде встановлено в kubernetes cluser в атестаційні роботі необхідно розробити конвеєр на збірку невеликого додатку на flask, для даних цілей

необіхдно мінімальна кількість ресурсів. Jenkins необхідно 1 гігабайти оперативної пам`яті та один процесор. Для додатку на flask необхідно 512 мегабайтів оперативною пам’яті та 200 мілісекунд процесорного часу. Найбільше навантаження на ноди будуть у момент збіркі docker образу та може сягати всього ліміту процесорного часу та приблизно 500-1000 мегабайтів оперативної пам’яті. А також варто пам’ятати що за замовчуванням операційна система з Kuberntetes кластером використовує 1 гігабайт оперативної пам’яті та приблизно 500 мілісекунд процесорного часу. Тому виходячи із технічних можливостей сервера оптимальною конфігурацією буде мастер нода з двома процесорами та 4 гігабайтами оперативною пам’яті. В конфігурації vagrantfile прокидується порт з віртуальної машини master 30301 на сервер 8080 на який можливо заходить в локальній мережі, це було зроблено для зручності роботи з jenkins, так як за замовчуванням jenkins використовує порт 8080 в подальшому сервіс NodePort можливо замінити на LoadBalancer якщо необхідно.

Першим етапом встановлення кластеру є:

- створення vagrant файлу зображено на рис 4.1;
- запуск vagrant файлу командою vagrant up;

Технічні характеристики нод на яких буде праювати kubernetes див на рис 4.2 вхід на мастер або робочу ноду, яка створена за допомогою vagrant файлу sudo vagrant ssh master [19]. Також варто відмітити метод встановлення kubernetes за допомогою kubespray це досить прости, та зручний спосіб, але він має певні недолікі, а саме:

- нестабільність роботи сервісів;
- важка для розуміння конфігурація написан на мові програмкванні ruby;
- не гнуність конфігурації.

До переваг варто відмітити:

- легкість встановлення;
- не треба нічого налаштовувати окремо.

```
hosts=[
      {
        :hostName => "master",
        :box => "ubuntu/focal64",
        :subnet => "subnet1",
        :cpu => 2,
        :ram => 4096,
        :ip => "172.100.100.100",
      },
       {
         :hostName => "node-1",
         :box => "ubuntu/focal64",
         :subnet => "subnet1",
         :cpu => 1,
         :ram => 1024,
         :ip => "172.100.100.101",
       }
]

Vagrant.configure(2) do |config|
    hosts.each do |host|
            config.vm.define host[:hostName] do |node|
                node.vm.box = host[:box]
                node.vm.hostname = host[:hostName]
                node.vm.network "private_network" , ip:
"#{host[:ip]}"
                if host[:hostName] == 'master'
                    node.vm.network "forwarded_port", guest:
30301, host: 8080,
                        auto_correct: true
                end
                node.vm.provider "virtualbox" do |vb|
                    vb.memory = host[:ram]
                    vb.cpus = host[:cpu]
                    vb.name = host[:hostName]
                end
            end
    end
end
```

Рисунок 4.1 – Код Vagrantfile

## 4.2 Створення кластеру та конфігурування кластеру Kubernetes

Другим етапапом необхідно встановити kubernetes на мастер ноду, перелік команд для встановлення зображено на рис 4.2. Після розгортання віртуальних машин необхідно залогінитсь на мастер ноду, та виконати команди на рис 4.2 [18]. Процес встановлення kubernetes займаэ приблизно 10-15 хвилин на данному серверу, основними етапами встановлення kubernetes:

- встановлення docker engine;
- встановлення kubernetes engine ;
- встановлення kubectl;
- встанволення мережного плагіну Kalico.

```
Встановлення Kubernetes на master and worker nodes
 Вімкнути Firewall
ufw disable
-     Вімкнути swap
swapoff -a; sed -i '/swap/d' /etc/fstab
-     Оновити sysctl settings for Kubernetes networking
cat >>/etc/sysctl.d/kubernetes.conf<<EOF
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
-     Встановлення docker engine
{
  apt install -y apt-transport-https ca-certificates curl gnupg-
agent software-properties-common
  curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-
key add -
  add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
  apt update
  apt install -y docker-ce=5:19.03.10~3-0~ubuntu-focal
containerd.io
}
-     Встановлення Kubernetes
{
  curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg |
apt-key add -
  echo "deb https://apt.kubernetes.io/ kubernetes-xenial main" >
/etc/apt/sources.list.d/kubernetes.list
}
-     Встановлення Kubernetes компонентів
apt update && apt install -y kubeadm=1.18.5-00 kubelet=1.18.5-00
kubectl=1.18.5-00
-     Налаштування мастера Kuberenetes
-     Ініціалізація Kubernetes Cluster
kubeadm init --apiserver-advertise-address=172.100.100.100 --pod-
network-cidr=192.168.0.0/16  --ignore-preflight-errors=all
-     Встановлення плагіну Calico network
kubectl --kubeconfig=/etc/kubernetes/admin.conf create -f
https://docs.projectcalico.org/v3.14/manifests/calico.yaml
```

Рисунок 4.2 – Встановлення kubernetes cluster

```
-     Команда приєднаянн кластеру
kubeadm token create --print-join-command
-     Для того щоб використовувати kubernetes не від
адміністратору
mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

-     Для того що приєднати ноду до кластеру
token create --print-join-command і ввести це в воркер ноду.
-     Верифікація від кластеру
kubectl get nodes
-     Подивитися на статус kubernetes-controller
kubectl get cs
```

Рисунок 4.3 – Приєднання worker-node до Kubernetes cluster

Після встановлення Kubernetes за замовчуванням піднімається дві ноди, але врахувуючи системні вимоги та тривале навантаження сервера було прийнято рішення відключити worker-node. Якщо використовувати створений кластер в “production” тоді необхідно продублювати master ноду

на справді, залишати одну ноду “master” не дуже правильне рішення так як, у випадку відказу “master” ноди весь kubernetes кластер вийде з ладу до того моменту, коли “master” нода не буде доступна знову, тому в реальній ситуації справжнього “production” рішення потрібно залишати мінімум дві “master” ноди, а також дублювати etcd яка зберігає всю інформацію про кластер.

Команди виключення однієї ноди:

- kubectl delete node node-1;
- sudo vagrant destroy node-1.

Перевіряємо працездатность командою “kubectl get nodes”

Наступним шляхом потрібно потрібно розділити існючі ноди для зручності, використовується метод “міток” на ноду “master” повісимо мітку “cicd”, ця нода використовується для інструментів на ноду “worker-node” повісимо мітку “applications” на якій в свою чергу буде працювати web-application. В подальшому при створенні ресурсів Kubernetes буде використовано мітки.

Команди для надання міток нодам:

- показує які мітки існують на нодах kubectl get nodes --show-labels;
- kubectl label nodes node-1 tools=jenkins.

Наступним шяхом необхідно стсворити “namespaces” в Kubernetes для розподілення навантаження на ноди, та для зручності управління.

Команда для створення “namespaces”:

- kubectl create namespace cicd.

# 5 ПОБУДОВА КОНВЕЄРУ CICD

Безперервна інтеграція - це методологія розробки і набір практик, при яких в код вносяться невеликі зміни з частими коммітов. І оскільки багато найновіших задач розробляються з використанням різних платформ та інструментів, то з'являється необхідність в механізмі інтеграції та тестуванні внесених змін.

З технічної точки зору, мета CI забезпечити послідовний і автоматизований спосіб збирання, упаковки та тестування додатків. При налагодженому процесі безперервної інтеграції розробники з більшою ймовірністю будуть робити часті комміти, що, в свою чергу, буде сприяти поліпшенню комунікації і підвищення якості програмного забезпечення в якості CI інструменту в кваліфікаційній роботі було обрано інструмент jenkins який буде посередником між серверною складовою kubernetes та девелоперами в додатку Г зображена структурна схема процесу CICD. Кінцевим пунктом в реалізації CI частини в кваліфікаційній робот буде компіляція додатка.

В кваліфікаційній роботі було розроблено частину CD, яка починається після успішного проходженя етапа компіляції додатка, далі починається етап збірки docker образу, а потім його доставка до dockerhub реєстру. Останнім кроком є деплой додатка до kubernetes кластуру.

## 5.1 Встановлення та налаштування Jenkins

Для побудови конвеєра необіхдно встановити jenkins на kubernetes кластер.

Існує декілька методів деплою найпопулярні рішенням це за допомогою helm утиліти та за допомогою утиліти kubectl. В кваліфікаційній роботі було

обрано метод деплою Jenkins за допомогою kubectl.

Для початку траба з’ясувати які компоненти будуть входити для встановлення:

- deployment;
- service;
- pv;
- pvc;
- serviceAccount.

Основна конфігурація jenkins написана в deployment.yaml яка зображена на рис 5.1. Всі конфігураційні файли створюється за допомогою команди kubectl create -f somefile.yaml.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: jenkins
  labels:
    app: jenkins
spec:
  selector:
    matchLabels:
      app: jenkins
  replicas: 1
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  template:
    metadata:
      labels:
        app: jenkins
    spec:
      serviceAccountName: jenkins
      containers:
      - name: jenkins
        image: jenkins/jenkins:2.235.1-lts-alpine
        imagePullPolicy: IfNotPresent
        env:
        - name: JAVA_OPTS
          value: -Xmx2048m -Dhudson.slaves.NodeProvisioner.MARGIN=
50 -Dhudson.slaves.NodeProvisioner.MARGIN0=0.85
        ports:
        - containerPort: 8080
          protocol: TCP
        - containerPort: 50000
          protocol: TCP
        volumeMounts:
        - mountPath: /var/jenkins_home
          name: jenkins
      restartPolicy: Always
      securityContext:
        runAsUser: 0
      terminationGracePeriodSeconds: 30
      volumes:
      - name: jenkins
        persistentVolumeClaim:
          claimName: jenkins-claim
```

Рисунок 5.1 – Основний конфігураційний файл для створення jenkins

Наступним шляхом необхідно створити service.yaml для міжмережової взаємодії Jenkins з Kubernetes кластером, а також для порт-форвардінгу. Створення service.yaml зображено на рис 5.2.

```
```
apiVersion: v1
kind: Service
metadata:
  name: jenkins
spec:
  type: NodePort
  ports:
    - port: 8080
      targetPort: 8080
      nodePort: 30301
  selector:
    app: jenkins

---

apiVersion: v1
kind: Service
metadata:
  name: jenkins-jnlp
spec:
  type: ClusterIP
  ports:
    - port: 50000
      targetPort: 50000
  selector:
    app: jenkins
```

Рисунок 5.2 – Компонент service

Необхідно створити сховища, для зберігання конфігурації jenkins. Створення pv.yaml та pvc.yaml зображено на рис 5.3. В данній роботі як основне сховище було створено persistent values на чотири гігабайти, сховище нобхідно для jenkins щоб збрігати всю конфігурацію. Невід’ємною частиною persistent values є persistent values claim який гарантує що певна кількість гігабайтиів буде доступна для певного persistent values.
```

````
#### Створення pv
```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: jenkins
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 4Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/mnt/data"
```
#### Створення pvc
```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: jenkins-claim
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 4Gi
```
````

Рисунок 5.3 – Компонент persistence voulume та persistence voulume claim

У Kubernetes облікові записи служби використовуються для надання ідентифікаційних даних для подів. Структури, які хочуть взаємодіяти з сервером API, пройдуть автентифікацію за допомогою певного облікового запису служби. За замовчуванням програми автентифікуються як обліковий запис служби за замовчуванням у просторі імен, в якому вони працюють.

Отже, у Kubernetes є два типи облікових записів - обліковий запис користувача та обліковий запис служби. Обліковим записом користувача користуються люди, а обліковим записом послуг - машини. Обліковий запис користувача може бути для адміністратора, який отримує доступ до кластера для виконання адміністративних завдань, або для розробника, який звертається до кластера для розгортання програм тощо.

Тобто для забезпечення прав певному користовачу або неймспейсу необхідно створити serviceAccount.yaml зображенао на рис.5.4 який зберігає

правила які дозволені певному користовачеві.

```
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: jenkins
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: jenkins
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/exec"]
  verbs: ["create","delete","get","list","patch","update","watch"]
- apiGroups: [""]
  resources: ["pods/log"]
  verbs: ["get","list","watch"]
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["create","delete","get","list","patch","update"]
- apiGroups: [""]
  resources: ["configmaps"]
  verbs: ["create","delete","get","list","patch","update"]
- apiGroups: ["apps"]
  resources: ["deployments"]
  verbs: ["create","delete","get","list","patch","update"]
- apiGroups: [""]
  resources: ["services"]
  verbs: ["create","delete","get","list","patch","update"]
- apiGroups: [""]
  resources: ["ingresses"]
  verbs: ["create","delete","get","list","patch","update"]

---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: jenkins
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: jenkins
subjects:
- kind: ServiceAccount
  name: jenkins
  namespace: jenkins
```

Рисунок 5.4 – Компонент serviceAccount

## 5.2 Створення пайплайнів для автоматизації

Jenkins pipline було написано на мові програмування Groovy зображено на рис 5.5 . Jenkins пайплай нскладається з декількох етапів:

- checkout на якому стягується остання версія коду;
- compile на якому компілюється web-додаток;
- build && Push image на цьому етапі проходить створення образу додатка та просування його до dockerhub реєстру;
- deploy останній етап на якому деплоїться додаток до kubernetes cluster.

```
podTemplate(yaml: '''
apiVersion: v1
kind: Pod
spec:
  containers:
  - name: docker
    image: docker:19.03.1-dind
  name: test-container
    volumeMounts:
    - mountPath: /usr/bin/kubectl
      name: kubectl
    - mountPath: /usr/bin/python
       name: python
  volumes:
  - name: python
  - name: kubectl
    securityContext:
      privileged: true
    env:
      - name: DOCKER_TLS_CERTDIR
        value: ""
''') {
    node(POD_LABEL) {
        container('docker') {
            stage(" Checkout"){
                checkout([$class: 'GitSCM', branches: [[name:
'*/master']],
                userRemoteConfigs: [[url:
'https://github.com/TiestoVarn/test1337.git']]])
            }
            stage("Compile"){
                sh "python Mykola.py ."
            }
            stage('Build && Push image') {

docker.withRegistry('https://registry.hub.docker.com',
'DockerHubCreeds') {
                def newApp = docker.build "tiestovarn/docker:
${env.BUILD_TAG}"
                newApp.push()
                }
            }
            stage("Deploy"){
                sh "kubectl apply -f myapp.yaml . -n jenkins"
            }
        }
    }
}
```

Рисунок 5.5 – Jenkins pipline

## 5.3 Створення веб-додатку

Додаток створено на python за допомогою фреймворка flask, ціль даного додатко виключно для тестування строенної інфраструктури, код додатку та файл home.html зображено на рис 5.6. Результат запуску контейнеру є web сторінка з рисунком на якому написано “hello world” [20].

C: > Users > Mykola_Khodakivskyi > Desktop > diploma > diplom > Mykola.py > ...

```python
from flask import Flask, render_template, request, redirect, url_for

import os

Mykola = Flask(__name__)

@Mykola.route('/')
def home():
    return render_template('home.html')

if __name__ == "__main__":
    Mykola.run(host="0.0.0.0", port="1337", debug=True)

```

C: > Users > Mykola_Khodakivskyi > Desktop > diploma > diplom > templates > home.html > ...

```html
<!DOCTYPE html>
<html lang="en" dir="ltr">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=4, shrink-to-fit=no">
    <!-- CSS only -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" re
    <!-- JavaScript Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/js/bootstrap.bundle.min.
    <title>Installer</title>
  </head>
  <style>
    body{
      background-image: url('static/bgfull.jpg');
      background-repeat: no-repeat;
      background-attachment: fixed;
      background-size: cover;
    }
  </style>
```

Рисунок 5.6 – Веб-додаток та html сторінка

# ВИСНОВКИ

В даній роботі було розглянуто основні принципи побудови кластерного середовища в основі якого лежить Kubernetes кластер, була розроблена архітектура. Розроблено веб-додаток на мові програмування python який служив в якості тестового додатка для перевірки працездадтності системи.

Таким чином досліджені технології оркестрації контейнерів, а також методи автоматизації розробки веб-додатків дозволили збільшити продуктивність розробки кінцевого продукта в декілька раз.

Слід зазначити, що на молодіжному форумі «РАДІОЕЛЕКТРОНІКА ТА МОЛОДЬ В XXI СТОРІЧЧІ» в 2020 р. та в 2021 р. були представлені тези за темами впровадження хмарної мікросервісної архітектури на прикладі контейнорної програми openshift та хмарної мікросервісної архітектури на прикладі контейнорної програми kubernetes, що суттєво допомогли у розробці даної кваліфікаційної роботи . У Додатках Б та В наведено повний текст тез, що був поданий до друку.

# ПЕРЕЛІК ПОСИЛАНЬ

1. Поняття Infrastructure as Code [Електроний ресурс:]: https://www.redhat.com/en/topics/automation/what-is-infrastructure-as-code-iac.

2. Різниця між системами конфігурування інфраструктурою [Електроний ресурс:]: https://blog.gruntwork.io/why-we-use-terraform-and-not-chef-puppet-ansible-saltstack-or-cloudformation-7989dad2865c.

3. Методологія devops [Електроний ресурс:]: https://stackpulse.com/blog/the-devops-methodology-principles-for-successful-implementations/.

4. DevOps основні задачі[Електроний ресурс:]: https://www.altexsoft.com/blog/engineering/devops-principles-practices-and-devops-engineer-role/.

5. Задачі сервера [Електроний ресурс:]: https://galtsystems.com/blog/start/chto_takoe_server_i_dlya_chego_on_nuzhen/.

6. Порівняльна характеристика операційних систем [Електроний ресурс:]: https://www.guru99.com/linux-differences.html.

7. Операційна система ubuntu [Електроний ресурс:]: https://sysadmin.ru/articles/best-linux-distros.

9. Основні характеристики linux [Електроний ресурс:]: https://www.javatpoint.com/what-is-linux.

10. Основні відмінності між windows та лінкус операційними системами[Електроний ресурс:]: https://ualinux.com/ru/stream/osnovnye-otlichiya-windows-i-linux.

11. Поняття віртуалізації та контейнеризації [Електроний ресурс:]: https://www.weave.works/blog/a-practical-guide-to-choosing-between-docker-containers-and-vms.

12. Апаратна віртуалізація особливості [Електроний ресурс:]:

https://www.it-grad.ru/blog/kak-rabotaet-apparatnaya-virtualizaciya.

13. Відмінності Saas, PaaS, IaaS [Електроний ресурс:]: https://itglobal.com/ru-ru/company/blog/iaas-paas-saas/

14. OpenShift [Електроний ресурс:]: https://redhat.axoft.ru/files/OpenShift.pdf.

15. Порівняльна характеристика Kubernetes та Openshift [Електроний ресурс:]: https://www.redhat.com/en/topics/containers/red-hat-openshift-kubernetes.

16. Вибір інструмента для сі частини [Електроний ресурс:]: https://www.infoworld.com/article/3239666/what-is-jenkins-the-ci-server-explained.html.

17. Вибір системи контролю версій [Електроний ресурс:]: https://www.internet-technologies.ru/articles/newbie/top-5-sistem-upravleniya-versiyami-s-otkrytym-ishodnym-kodom.html.

18. Встановлення Kubernetes cluster [Електроний ресурс:]: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/create-cluster-kubeadm/.

19. Особливості написання пайплаїнів [Електроний ресурс:]: https://www.jenkins.io/doc/book/pipeline/.

20. Побудова вед-додатку[Електроний ресурс:]: https://flask.palletsprojects.com/en/2.0.x/.