Київський національний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЕКТ

на тему:

«Розробка мікросервісної архітектури хмарного середовища на платформі KUBERNETES»

Студента 2м курсу, 6з групи, спеціальності 121 «Інженерія програмного забезпечення» спеціалізації «Інженерія програмного забезпечення» _____ підпис студента — Петліченка Дмитра Георгійовича

Науковий керівник старший викладач кафедри програмної інженерії та кібербезпеки _____ підпис керівника — Десятко Альона Миколаївна

Гарант освітньої програми доктор технічних наук, професор кафедри інженерії програмного забезпечення та кібербезпеки _____ підпис гаранта — Криворучко Олена Володимирівна

КИЇВ - 2020

Київський національний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь магістр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
8 листопада 2019 р.

Завдання
на випускний кваліфікаційний проект студентові

Петліченку Дмитру Георгійовичу
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проекту «Розробка мікросервісної архітектури хмарного середовища на платформі KUBERNETES»
Затверджена наказом ректора від "24" грудня 2019 р. № 4440

2. Строк здачі студентом закінченої проекту 01 грудня 2020 р.

3. Цільова установка та вихідні дані до проекту *Мета проекту полягає у розробці мікросервісної архітектури хмарного середовища.*
Об'єкт дослідження розгортання інфраструктури у хмарному середовищі, розгортання у цьому середовищі кластеру Kubernetes, та наповнення кластеру функціоналом.
Предмет дослідження є методи continuous integration та continuous delivery.

4. Консультанти проекту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проекту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. ХМАРНІ СЕРЕДОВИЩА, ПІДХОДИ, ЩО ЛЕЖАТЬ В ЇХ ОСНОВІ.

ВИБІР ПРОВАЙДЕРА ХМАРНИХ ПОСЛУГ

1.1. Основи хмарних середовищ

1.2. Варіанти надання обчислювальних потужностей

1.3. Моделі розгортання хмари

1.4. Вибір провайдера хмарних послуг для розгортання kubernetes кластера

1.4.1. Порівняльна характеристика послуг українських провайдерів

1.4.2. Порівняльна характеристика послуг західних провайдерів

1.4.3. Сервіси компанії Amazon

1.5. Висновки за розділом

РОЗДІЛ 2. ОСНОВИ KUBERNETES, ІСТОРІЯ ПОЯВИ ТЕХНОЛОГІЇ ТА ПОРІВНЯННЯ ЇЇ З АНАЛОГАМИ

2.1. Основи KUBERNETES

2.2. Історія еволюції KUBERNETES

2.3. Порівняння основних конкурентів технології

2.4. Висновки за розділом

РОЗДІЛ 3. РОБОТА З AMAZON ПРОВАЙДЕРОМ. РОЗГОРТАННЯ KUBERNETES КЛАСТЕРУ ТА НАПОВНЕННЯ ЙОГО ФУНКЦІОНАЛОМ

3.1. Робота з серверами та сіткою у хмарі

3.2. Розгортання та конфігурація KUBERNETES кластеру

3.3. Наповнення кластеру функціоналом

3.3.1. Автентифікація

3.3.2. Моніторинг

3.3.3. Сервіс документообігу

3.4. Висновки за розділом

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проекту

№ пор.	Назва етапів випускного кваліфікаційного проекту	Строк виконання етапів проекту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускної кваліфікаційної роботи*	*20.09.2019*	*20.09.2019*
2.	*Розробка та затвердження завдання на проект магістра*	*08.11.2019*	*08.11.2019*
3.	*Вступ та перелік літературних джерел*	*24.01.2020*	*24.01.2020*
4.	*Наукова стаття*	*01.09.2020*	*01.09.2020*
5.	*Технічне завдання*	*28.02.2020*	*28.02.2020*
6.	*Розділ 1. Хмарні середовища, підходи, що лежать в їх основі. вибір провайдера хмарних послуг*	*25.06.2020*	*25.06.2020*
7.	*Розділ 2. Основи kubernetes, історія появи технології та порівняння її з аналогами*	*07.09.2020*	*07.09.2020*
8.	*Розділ 3. Робота з Amazon провайдером. розгортання kubernetes кластеру та наповнення його функціоналом*	*19.10.2020*	*19.10.2020*
9.	*Висновки та пропозиції*	*30.10.2020*	*30.10.2020*
10.	*Здача випускного кваліфікаційного проекту на кафедру (перша перевірка)*	*05.11.2020*	*05.11.2020*
11.	*Підготовка автореферату та презентації доповіді*	*05.11.2020*	*05.11.2020*
12.	*Попередній захист випускного кваліфікаційного проекту*	*25.11.2020-27.11.2020*	*25.11.2020-27.11.2020*
13.	*Зовнішнє рецензування випускної кваліфікаційної роботи*	*01.12.2020*	*01.12.2020*
14.	*Підготовка до публічного захисту випускної кваліфікаційної роботи*	*10.12.2020-11.12.2020*	

7. Дата видачі завдання «8» листопада 2019 р.

8. Науковий керівник випускного кваліфікаційного проекту Десятко А. М.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Криворучко О. В.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Петліченко Д. Г.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проекту

_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____
_____

Науковий керівник випускного кваліфікаційного проекту _____
(підпис, дата)

Відмітка про попередній захист _____ Десятко А. М. _____
(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проект

Випускний кваліфікаційний проект студента Петліченко Д. Г.
(прізвище, ініціали)
може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____ Криворучко О. В. _____
(прізвище, ініціали, підпис)

Завідувач кафедри _____ Криворучко О. В. _____
(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Відповідно до мети дослідження робота присвячена створенню конфігурації, на основі досліджень, поєднання двох технологій та використання їх ефективного симбіозу.

В результаті порівняльного аналізу аналогічних рішень визначимо чому поєднання хмарних технологій та технологій контейнерних оркестраторів є дуже актуальним і чому такий симбіоз є ефективним при управлінні ресурсами.

Розробка конфігурації інфраструктури виконана у хмарному середовищі компанії Amazon - Amazon Web Services. Kubernetes кластер розгорнутий на основі серверів, замовлених у сервісі EC2 у хмарному середовищі Amazon. Все налаштування мережі для функціонування кластеру виконано у сервісі VPC у хмарному середовищі Amazon. Налаштування фаерволів, вхідного трафіку та безпеки створене у сервісі EC2 за допомогою інструментів load balancer, security group, network access control list.

Готова інфраструктура, функціонал кластеру та мікросервісна архітектура була успішно протестована відповідно до функціональних вимог.

Ключові слова: Kubernetes, Docker, кластер, оркестратор, хмарне середовище, Amazon Web Services.

ABSTRACT

In accordance with the purpose of the research, the work is devoted to creating a configuration, based on research, a combination of two technologies and the use of their effective symbiosis.

As a result of comparative analysis of similar solutions, was determined why the combination of cloud technologies and technologies of container orchestrators is very relevant and why such a symbiosis is effective in resource management.

Infrastructure configuration development was performed in Amazon's cloud environment - Amazon Web Services. The Kubernetes cluster is deployed based on servers ordered from the EC2 service in the Amazon cloud environment. All network settings for cluster operation are performed in the VPC service in the Amazon cloud environment. The setting of firewalls, incoming traffic and security is created in the EC2 service using the tools - load balancer, security group, network access control list.

The finished infrastructure, cluster functionality and microservice architecture were successfully tested in accordance with the functional requirements.

Keywords: Kubernetes, Docker, cluster, orchestrator, cloud environment, Amazon Web Services.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ - Програмне забезпечення

БД - База даних

ОС - Операційна система

ІТ - Інформаційні технології

K8ts - Kubernetes

*Nix - Unix, Linux

ПК - Персональний ком’пютер

API - Application Program Interface

LMS - Learning Management System

IAC - Infrastructure As Code

CF - Configuration Management

AWS - Amazon Web Service

CD - Continuous Delivery

MaaS - Metal as a Service

SaaS - Software as a service

PaaS - Platform as a service

IaaS - Infrastructure as a service

ЗМІСТ

ВСТУП

Актуальність. Стрімкий розвиток інформаційних технологій на кінці XX - початку XXI століття призвів до того, що сьогодні дуже важко знайти людину, яка б щодня не користувалася інтернетом та не використовувала додатки роблячи найпростіші побутові задачі. Всі додатки, сайти та їх функціональні частини на початку 2010 почали мігрувати до мікросервісної архітектури. На сьогодні ми можемо побачити іншу тенденцію - це міграція у хмарні середовища, побудова ізольованої інфраструктури віддалено та використання в інфраструктурі не просто контейнерів, а й їх оркестраторів.

Наприклад на дані технології перейшли вже багато компаній, як то Twitter, Facebook, Netflix – усі ці компанії об'єднує великий обсяг даних та необхідність забезпечення доступ до ресурсів.

Велика кількість великих корпорацій навіть інвестують гроші в розробку власних мікросервісних оркестраторів та інших інструментів для реалізації доставки коду до кластерів. Як приклад - компанія Netflix, що є лідером серед провайдерів медійних послуг, інвестувала грощі та розробила власний інструмент для розгортання додатків у Kubernetes - Spinaker.

Така сама ситуація зараз відбувається і з хмарними технологіями. Сьогодні, в 2020 році, складно знайти людину, яка б не могла похвалитися інстаграмом, ел-поштою, картами і пробками в телефоні. І все це наразі зберігається, обробляється та функціонує у хмарному середовищі.

					КНТЕУ 121 06з-10.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		24.01.20	*Розробка мікросервісної архітектури хмарного середовища на платформі KUBERNETES*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Десятко А.М.		24.01.20		*В*	*5*	*64*
Гарант		Криворучко О.В.		24.01.20		Факультет інформаційних технологій 2м курс, 6з група		
Розробив		Петліченко Д.Г.		24.01.20				
					Вступ			

Хмарні середовища наразі використовують у кожній сфері як виробництва, так і надання послуг. Готельні системи, автомобільні додатки, навігатори, фармацевтична сфера, обчислення на промислових виробництвах, магазини та навіть фінансові структури - всі вони користуються хмарними середовищами для свого функціонування.

Побудова ефективного хмарного середовища - не проста задача. Так як всюди присутній принцип “плати по факту використання” - для зменшення бюджетів повинно буту якнаймога менше самого “використання”. Отже, вважаємо за доцільне розробити у хмарному середовищі мікросервісну архітектуру із використанням найпопулярнішого оркестратора.

Дослідженням питань хмарних технологій та мікросервісних оркестраторів займалися такі вітчизняні науковці як Мамзелев А.І., Монахов Д.Н., Монахов Н.В., Прончев Г.Б., тощо.

Незважаючи на це, питання у розрізі, симбіозу хмарних технологій та контейнерних аркестраторів є недостатньо розкритими і потребують більш глибокого дослідження.

Відсутність достатньої кількості документації та великий попит на поєднання двох технологій на сучасному IT ринку України зумовило визначення об'єкта і предмета дослідження.

Мета дослідження: створення конфігурації, на основі досліджень, поєднання двох технологій та дослідження їх ефективного симбіозу.

Об'єкт дослідження: розгортання інфраструктури у хмарному середовищі, розгортання у цьому середовищі кластеру Kubernetes, та наповнення кластеру функціоналом.

*Предмет дослідження***:** створення рекомендацій, конфігурацій та налаштувань хмарного середовища, кластера Kubernetes та його логічного наповнення.

У відповідності з метою дослідження поставлені наступні *завдання*:

1) дослідити хмарних провайдерів, як вітчизняних так і закордонних;

2) дослідити розвиток та причини популяризації контейнерних оркестраторів при побудові микросервісної архітектури;

3) обґрунтувати вибір провайдера хмарних технологій та технології оркестрування мікросервісною архітектурою;

4) створити інфраструктуру, мережу, кластер та наповнити кластер функціоналом;

5) провести тестування готового рішення перевіривши коректне функціонування інфраструктури.

Методи дослідження, що були використані у роботі: аналіз, абстрагування, порівняння, графічний метод і моделювання.

Наукова новизна дослідження полягає в удосконаленій концепції поєднання двох технологій та дослідження їх ефективного симбіозу.

*Практичне значення дослідження***:** готовий кластер що розгорнутий у хмарному середовищі та корректно функціонує готовий для використання і розгортання на його основі мікросервісної архітектури.

РОЗДІЛ 1.

ХМАРНІ СЕРЕДОВИЩА, ПІДХОДИ, ЩО ЛЕЖАТЬ В ЇХ ОСНОВІ. ВИБІР ПРОВАЙДЕРА ХМАРНИХ ПОСЛУГ

1.1. Основи хмарних середовищ

В ході аналізу визначень у багатьох джерелах мною було сформовано загально визначення хмарного середовища. Хмарне середовище - це середовище для зберігання і обробки даних, яка об'єднує в собі апаратні засоби, програмне забезпечення, канали зв'язку і підтримку користувачів. Робота в такому середовищі спрямована на зниження витрат і збільшення ефективності роботи будь-якого підприємства.

Хмарне середовище - це не тільки віртуалізація. Віртуалізація серверів та інфраструктури становить основу приватних хмарних обчислень, самі віртуалізація і управління середовищем ще не є приватною хмарою [1].

Віртуалізація більше структурує, об'єднує в пул і динамічно надає ресурси інфраструктури: сервери, персональні комп'ютери, носії, мережеве обладнання, сполучене програмне забезпечення [1]. Однак щоб середовище могло вважатися хмарним, потрібні ще й інші складові, такі як віртуальні машини, операційні системи, високостійкі операційні системи, програмне забезпечення для відволікання ресурсів зберігання, кошти для масштабування і розбиття множини об'єктів на групи.

Приватна хмара на відміну від загальнодоступної або гібридної відноситься до ресурсів, що використовуються єдиною організацією або означає, що хмарні ресурси організації повністю приховані в хмарі від інших.

Хмарні технології - необов'язково джерело економії. Вони можуть економити гроші, але це не означає, що буде обов'язково так. Приватна хмара більш ефективно перерозподіляє ресурси, для задоволення корпоративних вимог, також може зменшити величезні витрати на обладнання [1]. Даний вид хмари потребує автоматизації. Через це, зниження витрат не головна перевага цієї моделі. Першочерговим завданням повинна стояти швидкість випуску продукції на ринок, а не економія.

Приватна хмара не завжди впроваджена у замовника. Вона означає запобігання витоку інформації. Деякі провайдери виділяють ресурс одному замовнику, ігноруючи спільне використання. Якщо центри обробки даних об'єднувати в пул ресурси різних замовників, то ізолювати один від одного їх можна за допомогою Virtual Private Network (VPN) [3].

Приватна хмара (як і публічна хмара) - це не тільки інфраструктурні сервіси. Приватна хмара зводиться не тільки до інфраструктури як послуга (IaaS), для розробки і тестування нового програмного забезпечення використовується високорівнева платформа як послуга (PaaS) [2].

Найшвидше росте сегмент хмарних обчислень - IaaS. Він надає нижчі рівні ресурсів центру обробки даних в простий для використання формі, при цьому, не змінюючи основних принципів роботи. Для створення нових додатків, призначених для хмари і надають абсолютно нові послуги, які можуть дуже відрізнятися від того, що давали колишні додатки, розробникам зручніше використовувати PaaS.

Приватна хмара може перестати бути приватною. Здавалося б, приватна хмара має переваги хмари: швидкість перебудови, масштабованість і ефективність, має відносно високий рівень безпеки. Але в той же час, рівень обслуговування, безпеку і контроль дотримання вимог буде підвищуватися і в загальнодоступних хмарних сервісах. Тому деякі приватні хмари, швидше за все, цілком перейдуть в категорію загальнодоступних. Більшість же сервісів приватного хмари, напевно, будуть перерости в гібридні хмарні.

Сервіси, розширюють можливості за рахунок використання загальнодоступних хмарних послуг і інших сторонніх ресурсів.

У документі "The NIST Definition of Cloud Computing" визначають наступні характеристики хмар [1]:

- висока ступінь автоматизованого самообслуговування системи з боку провайдера
- наявність системи Broad Network Access
- зосередженість ресурсів в окремих місцях для ефективного розподілу
- ресурси можуть необмежено виділятися з великою швидкістю в залежності від потреб
- система управління хмарою сама контролює і оптимізує виділення ресурсів.

Самообслуговування на вимогу (On-demand self-service). У споживача є можливість отримати доступ до надаваних обчислювальних ресурсів в односторонньому порядку в міру потреби, автоматично, без необхідності взаємодії з співробітниками кожного постачальника послуг.

Широкий мережевий доступ (Broad network access). Надані обчислювальні ресурси доступні по мережі через стандартні механізми [3].

Для різних платформ і клієнтів (мобільних телефонів, планшетів, ноутбуків, робочих станцій).

Об'єднання ресурсів в пули (Resource pooling). Обчислювальні ресурси провайдера об'єднуються в пули для обслуговування багатьох споживачів по багато орендної (multi-tenant) моделі. Пули включають в себе різні фізичні та віртуальні ресурси, які можуть бути динамічно призначені і перепризначені відповідно до споживчими запитами [3]. Немає необхідності в тому, щоб споживач знав точне місце розташування ресурсів, проте можна зазначити їх місцезнаходження на більш високому рівні абстракції (наприклад, країна або регіон). Прикладами такого роду ресурсів можуть бути системи зберігання, обчислювальні потужності, пам'ять, пропускна здатність мережі.

Миттєва еластичність (Rapid elasticity). Ресурси можуть бути легко виділені і звільнені, в деяких випадках автоматично, для швидкого масштабування пропорційно попиту [2]. Для споживача можливості надання ресурсів бачаться як необмежені, тобто вони можуть бути присвоєні в будь-якій кількості і в будь-який час.

Вимірюваний сервіс (Measured service). Хмарні системи автоматично керують і оптимізують ресурси за допомогою засобів вимірювання, реалізованих на рівні абстракції стосовно різного роду сервісів (наприклад, керування зовнішньою пам'яттю, обробкою, пропускною здатністю або активними призначеними для користувача сесіями). Використані ресурси можна відстежувати і контролювати, що забезпечує прозорість як для постачальника, так і для споживача, що використовує сервіс [2].

1.2. Варіанти надання обчислювальних потужностей

Варіанти надання обчислювальних потужностей в хмарних системах сильно відрізняються один від одного. У зв'язку з тим, що одним з головних

підходів в основі хмарних технологій це представлення всього як сервіс, до назв варіантів надання послуг прийнято додавати словосполучення "as a service", що в перекладі означає "у виді сервісу".

SaaS (Software as a service), або програми у вигляді сервісів - варіант, при якому пропонується використовувати конкретне ПО, наприклад, корпоративну систему, у вигляді сервісу по підписці [14]. Якщо підприємство не має можливості мати внутрішній Exchange - сервер для роботи пошти і календарів його можна придбати віддалено, вже налаштованим з урахуванням всієї необхідної специфіки.

PaaS (Platform as a service) - на відміну від SaaS, який призначений більше для кінцевого користувача, цей варіант призначений більше для розробників [14]. У разі PaaS, в хмарі функціонує набір програм, основних сервісів і бібліотек, на основі яких пропонується розробляти свої додатки. Прикладом цього варіанту надання послуг є платформа для створення додатків Google AppEngine. Крім цього, під PaaS розуміють також і окремі частини складних систем, такі як системи бази даних або комунікацій.

HaaS (Hardware as a service) - один з перших термінів, що означають надання деяких базових апаратних функцій і ресурсів у вигляді сервісів. Замість прямої оренди сервера використовується віртуалізація [14]. У разі HaaS, під конкретним апаратним забезпеченням маються на увазі деякі абстрактні сутності, аналогічні фізичним, такі як місце для зберігання інформації, процесорний час в еквіваленті якого або реального CPU, пропускна здатність.

IaaS (Infrastructure as a service) - вважається, що цей термін прийшов на зміну HaaS, піднявши його на новий рівень. Цим терміном називають надання

комп'ютерної інфраструктури як послуги [14]. У даній роботі розглядається цей варіант надання потужностей.

CaaS (Communication as a service) - надання послуг зв'язку як сервісу. Під послугами зв'язку, зазвичай мають на увазі IP-телефонію, пошту або миттєві комунікації, такі як чати або служби обміну миттєвими повідомленнями. [2]

Порівняльна характеристика послуг, які надаються кожною моделлю кінцевому користувачеві приведена в Таб.1.1.

Прикладами SaaS є Gmail, Google Docs, Netflix, Photoshop.com, Acrobat.com, Intuit QuickBooks Online, IBM LotusLive, Unyte, Salesforce.com, Sugar CRM і WebEx.

Прикладами послуг платформи PaaS служать IBM SmartCloud Application Services, Amazon Web Services, Windows Azure, Boomi, CastIron.

Прикладами послуг Iaas служать IBM SmartCloud Enterprise, VMWare, Amazon EC2, Windows Azure, Google Cloud Storage, Parallels Cloud Server і багато інших.

Таблиця 1.1.

Порівняння моделей хмарних служб

Тип	**Споживач**	**Служба, яка надається хмарою**	**Область дії рівня обслуговування**	**Налаштування**
SaaS	Кінцеві користувачі	Готовий додаток	Час роботи додатка Продуктивність додатка	Мінімальна або відсутня Можливості визначаються ринком або постачальником
PaaS	Власник додатка	Середовище виконання для коду програми Хмарне сховище Інші хмарні служби, такі як інтеграція	Доступність середовища Швидкодія середовища Не поширюється на додатки	Високий рівень настройки на рівні додатків в межах пропонованих служб
IaaS	Власник додатка або IT забезпечує	Віртуальний сервер Хмарне сховище	Доступність віртуального сервера Час для підготовки до роботи Не поширюється на платформу або додатки	Мінімальні обмеження для додатків, встановлених в стандартизованих віртуальних збірках ОС

Джерело: створено автором на основі джерел [2],[14].

1.3. Моделі розгортання хмари

Незалежно від класу обслуговування, хмарні технології можуть бути розцінена як загальнодоступні, приватні, співтовариство, або гібридні, заснована на моделі розгортання. Загальнодоступна хмара - це хмара доступна широкому загалу, тобто клієнтом сервісів може стати як будь-яка компанія, так і будь-який користувач. Тут пропонується легкий і доступний спосіб розгортання веб-сайту або бізнес-системи з великими можливостями масштабування. Приклади: онлайн-сервіси Amazon EC2 і Amazon Simple Storage Service (S3), Google Apps / Docs, Salesforce.com, Microsoft Office Web. Приватна хмара - це центр обробки даних для бізнесу або іншої організації з вузьким колом доступу [15].

У більшості випадків, створення приватної хмари означає реструктуризацію існуючої інфраструктури шляхом додавання віртуалізації і хмарних обчислень. Це дозволяє користувачам взаємодіяти з локальним центром даних, використовуючи переваги загальнодоступних хмар. Перевага приватного хмари перед загальнодоступним полягає в тому, що у приватної хмари більш детальний контроль ресурсів. Звичайно ж, коли роботу не можна довірити загальнодоступній хмарі, приватні хмари - ідеальні. Хмара спільноти спільно використовується кількома організаціями та підтримується певною спільнотою. Гібридна хмара формується, коли до приватної хмари додана обчислювальна здатність від відкритої хмари [3]. Недоліком гібридної хмари є складність її управління і правильного створення. Отримання послуг відбувається з різних джерел, при цьому потрібно організовувати їх так, ніби джерело одне. Взаємодія між приватним і загальнодоступною хмарою може ще більше ускладнити рішення.

Моделі розгортання (загальні або виділені, також розміщуються всередині або поза організації) визначаються як володіння архітектурним проектом і управління ним, а також рівень доступною настройки ним [14].

Різні моделі розгортання ресурсів можуть бути оцінені за трьома стандартами: вартість, контроль і масштабованість.

Загальнодоступна хмара - це пул комп'ютерних послуг, що поставляються через Інтернет [3]. Воно пропонується постачальником, який зазвичай використовує модель оплати за фактичне використання або вимірювану службу. Загальнодоступні хмарні обчислення мають наступні потенційні переваги: оплата проводиться тільки за спожиті ресурси, маневреність досягається за допомогою швидкого розгортання, є можливості швидкого масштабування потужностей і всі служби поставляються з поліпшеними однаковими характеристиками доступності, відмовостійкості, безпеки і керованості. Існують такі різновиди загальнодоступного хмари:

1) Спільно використовується загальнодоступна хмара. Спільно використовується загальнодоступне хмара забезпечує перевагу швидкого впровадження, масового масштабування і низькі витрати на організацію виробництва. Вона надається в загальному середовищі, в якій архітектура, настройка і рівень безпеки проектуються і управляються постачальником відповідно до орієнтованими на ринок специфікаціями [14].

2) Виділена загальнодоступна хмара. Виділена загальнодоступна хмара забезпечує функціональні можливості, схожі на ті, які є в спільно використовуваний, за винятком того, що надає виділену інфраструктуру. Безпека, швидкодія, а іноді і можливості настройки краще в виділеній, ніж спільно використовуваний загальнодоступній хмарі [14]. Її архітектура і рівні обслуговування, можуть відрізнятися залежно, і її вартість може бути вище,

ніж у спільно використовуваної загальнодоступної хмари, в залежності від обсягу.

3) Приватна хмара - це пул комп'ютерних ресурсів, що надаються як стандартний набір служб, який визначається, проектується і контролюється конкретним підприємством [2].

Шлях до приватного хмари часто лежить через необхідність контролювати середу доставки, викликану наявністю застарілих додатків, вимогами до продуктивності, необхідністю відповідності регулятивним нормам або унікальністю компанії. Наприклад, банки та урядові установи мають проблеми з безпекою даних, які можуть перешкоджати використанню наявних служб загальнодоступного хмари. Існують такі різновиди приватної хмари:

1) Самостійно розміщена приватна хмара. Самостійно розміщена приватна хмара забезпечує переваги з точки зору контролю над архітектурою і операціями, в ньому використовуються наявні інвестиції в персонал і устаткування, і вона забезпечує виділену локальну середовище, яке проектується, розміщується і управляється всередині компанії.

2) Розміщена приватна хмара. Розміщена приватна хмара - це виділена середовище, яке проектується всередині компанії, а розміщується і управляється за її межами. У ній поєднуються переваги управління службою та архітектурним проектом з перевагами аутсорсингу.

3) Приватна хмара на основі пристрою. Приватна хмара на основі пристрою - це виділена середовище, яке закуповується у постачальника і спроектована їм з орієнтованими на постачальника послуг і ринок функціями і контролем над архітектурою [15].

Таблиця 1.2.

Порівняння моделей розгортання хмари

Тип розгортання	**Місце розміщення**	**Загальна або виділена**	**Управління архітектурою**	**Масштабованість**	**Необхідні інвестиції**
Спільно використовується Загальнодоступна хмара	Зовнішнє	Загальна	Постачальник або ринок	Мінімальні обмеження	Оплата за фактичне використання
Виділена загальнодоступна хмара	Зовнішнє	Частково або повністю виділена	Постачальник або ринок	Обмежено контрактом	Оплата за фактичне використання
Самостійно розміщена приватна хмара	Внутрішнє	Повністю виділене	Самостійне	Обмежено капітальними інвестиціями	Створення хмари, загальні служби
Розміщена приватна хмара	Зовнішнє	Повністю виділене	Самостійне	Обмежено капітальними інвестиціями чи контрактом	Залежить від контракту може або впливати або не впливати на капітал
Приватне хмара на основі пристрою	Внутрішнє	Повністю виділене	Постачальник	Обмежено пропозицією	Залежить від контракту може або впливати або не впливати на капітал

Джерело: створено автором на основі джерел [2], [3], [15].

1.4. Вибір провайдера хмарних послуг для розгортання kubernetes кластера

1.4.1. Порівняльна характеристика послуг українських провайдерів

Так як «хмарний» ринок України тільки почав розвиватися, такого розмаїття послуг, як зарубіжний, він на жаль не пропонує. Для проведення порівняння за повним переліком критеріїв за аналогією із західними провайдерами немає достатньої інформації, так як провайдери надають послуги які не підпадають під загальну характеристику.

На сьогоднішній день на ринку України серед провайдерів Iaas можна виділити дві компанії Tucha і Volia.

Tucha управляє своєю хмарою за допомогою платформи CloudStack 2.3 [10]. Дана платформа забезпечує безперервність, доступність і безперебійність роботи віртуальної інфраструктури, а також використовується більшістю компаній, що управляють публічними і приватними хмарами.

Порівняльні характеристики по продуктам кожної компанії розглянуті у додатку 1 [10][11].

Другий з найбільших провайдерів на ринку України, за поданням хмарних сервісів - ВОЛЯ. Управляє своїм хмарою за допомогою платформи від VMware [11].

Структура «хмари» включає два кластери - ресурсний і керуючий. Тому нарощування числа ресурсів практично нічим не обмежена зі збереженням безперервності роботи всього проекту в цілому. Створено з використанням апаратних рішень Cisco, EMC і Netapp, а також віртуалізаційних платформи VMware. Дані для білінгу надходять з останньої в власний додаток «Волі», за допомогою якого і будуть виставлятись рахунки. Інтегратором проекту виступила компанія DeNovo.

Одним з найбільш затребуваних і розвинених сервісів є бекап файлів в хмару VoliaCLOUD.

Епоха жорстких дисків і відновлення даних пройшла. Сьогодні компанії потребують захисту своєї інформації на більш високому рівні. Хмарні технології пропонують керовані сховища даних - сервіси, які дозволяють зручно, просто і надійно зберігати резервні копії і файли (бази даних, зображення, аудіо, відео). Ваші персональні дані зберігаються на численних розподілених в мережі серверах, що гарантує компанії безвідмовний доступ до своєї інформації [11].

При створенні резервних копій в хмарі VoliaCLOUD:

Ви можете відновити дані на будь-якому ноутбуці, настільному комп'ютері або сервері з виходом в Інтернет, або ж створити резервну копію безпосередньо в хмарі;

Ви платите тільки за ті ресурси, які використовуєте;

Хмарне резервне копіювання можна легко налаштувати за допомогою консолі управління з веб-інтерфейсом. Роботу з резервним копіюванням в мережевому сховище можна повністю автоматизувати.

Так само дана компанія надає послуги SaaS [11], такі як:

- 1С до VoliaCLOUD
- перенесення CRM до VoliaCLOUD
- перенесення пошти до VoliaCLOUD

1.4.2. Порівняльна характеристика послуг західних провайдерів

Для даного порівняння ми звернемо увагу на провайдерів IaaS, послуги яких можуть бути куплені в онлайн без укладення контракту з ким би то не було. Сформовано список з 11 компаній, від великих і відомих до маленьких і недавно вийшовших на ринок.

Для порівняння необхідно ввести деякі критерії, які змогли б відобразити важливі аспекти хмарних обчислень, такі як гарантія скорочення витрат і економія на масштабування, рівень сервісу, гнучкість в конфігурації серверів і цікаві для користувачів моменти типу безпеки, надійності і підтримки. В результат було виділено 15 критеріїв [16]:

1. План оплати (Pricing). Провайдери пропонують плани оплати типу:

- плати-за-фактом-використання (зазвичай в погодинному режимі)

- членські знижки (коли користувач отримує знижку на послуги в обмін на річне обслуговування) або комбінацію. Чим більше опцій - тим краще, однак модель оплати типу-плати-за-фактом-використання є найбільш цікавою окремої опцією, оскільки надає гнучкий контроль над використанням. Розглядалися тільки найбільш помітні плани оплати.

2. Середньомісячна ціна. Орієнтовна вартість в доларах за хмарний сервер 1 CPU, 2GB RAM (і хороші аналоги), усереднена для дата центрів і серверів Windows / Linux. При доступній інформації погодинна оплата розраховувалася на основі 730 годинного місяці. Інакше використовувалася місячна ціна, виключаючи ціни за передачу трафіку.

3. Service Level Agreement (SLA). Угоди по рівню сервісу (незалежно від попередньої продуктивності) в процентах.

4. Кількість дата центрів (Datacenters). Кількість дата центрів, що використовують для хмарних серверів.

5. Сертифікації (Certifications). У тому випадку, якщо вендор має різні сертифікати безпеки і надійності типу PCI або SAS70.

6. Вертикальне масштабування (Scale Up). По можливості вертикальне масштабування виконується додаванням більшої кількості пам'яті, CPU і об'єм серверу.

7. Горизонтальне масштабування (Scale Out). Швидке розгортання нових серверів.

8. Підтримка обслуговування користувачів (Support). Передбачена трьохрівнева суб'єктивна шкала оцінювання:

◦ Poor (погано) - компанії, що надають безкоштовну підтримку тільки он-лайн (форуми і т.д.) - інша підтримка повинна оплачуватися;

◦ Average (середньо) - компанії, що надають один тип 24x7 підтримки безкоштовно (телефон або он-лайн чат) + форуми;

◦ Extensive (високо) - компанії, що надають набір пропозицій, включѐнних в базовую цену.

9. Моніторинг (Monitoring). Пропонується трьохрівнева суб'єктивна шкала оцінювання:

◦ Poor (погано) - компанії без інтегрованих рішень моніторингу / сповіщення. Необхідно розгортання сторонніх інструментів або покупка додаткових послуг;

◦ Average (середньо) - компанії з дуже простим набором простих інтегрованих засобів (кілька індикаторів або без оповіщення);

◦ Extensive (високо) - компанії з повноцінними безкоштовними інтегрованими інструментами моніторингу.

10. API. Чи є API для взаємодії з серверами чи ні.

11. Free Tier. Є безкоштовна trial-версія , яку клієнт міг би використовувати для тестування роботи з сервісом.

12. Підтримувані ОС (Oss). Кількість підтримуваних операційних систем незалежно від версії, доступних у вигляді налаштованих образів.

13. Різноманітність типів серверів (Instance Types). Кількість різних конфігурацій серверів. Деякі провайдери пропонують повністю налаштованих сервера в термінах CPU (обозначені як - конфігуруються , -configurable).

14. Вартість вихідного трафіку (Data Transfer out (/ GB)). Вартість в доларах кожний гігабайт вихідного трафіку. Компанії, що надають безкоштовний вихідний трафік, мають значення ціни.

15. Вартість вхідного трафіку (Data Transfer in (/ GB)). Те ж саме, але для вхідного трафіку.

1.4.3. Сервіси компанії amazon

Розглянемо більш докладно сервіси найвідомішого провайдера хмарних технологій - Amazon. Amazon Web Services (AWS).

AWS являє собою конструктор, з якого можна зібрати як завгодно складну, розподілену мережеву інфраструктуру. Крім цього Amazon надає перший рік користування сервісом безкоштовно протягом року, за умови, що ви не перевищите лімітів сервісу (при перевищенні доведеться оплачувати за звичайним тарифним планом). Цілком достатньо, щоб спробувати хмарний хостинг безкоштовно. AWS дуже зручний при необхідності розгортати багато однакових серверів. Безкоштовний пакет AWS Free Usage Tier включає в себе [9]:

- EC2 (інстанси - віртуальні машини ОС)

- 750 годин використання віртуальної машини з Linux або Windows Server (613 Мб ОЗУ, 32 бітна або 64-бітна платформа) - достатня кількість годин для роботи інстанса щомісяця
- 750 годин Elastic Load Balancer плюс 15 Гб обробки трафіку
- 30 Гб Amazon Elastic Block Storage, плюс 2 мільйони операцій введення / виводу і 1 Гб для зберігання снепшотів.
- 15 Гб трафіку
- S3 (файлове сховище)
- 5 Гб Amazon S3 стандартного сховища, 20000 Get запитів і 2000 Put запитів
- Relational Database Service (служба реляційних баз даних, RDS)
- 750 годин сервісу для запуску MySQL, Oracle BYOL або SQL Server
- 20 Гб сховище бази даних
- 10 мільйонів операцій введення / виводу
- 20 Гб сховище для бекапів для автоматичного резервного копіювання вашої бази даних і можливістю створити снепшот бази даних

На Amazon є калькулятор для розрахунку споживання послуг, що надаються. Є три типи тарифів: on-demand, spot, reserved. On-demand - це звичайний VPS на віртуалізації Xen. Spot - це те ж саме, що on-demand, тільки не гарантується такий же високий uptime. Spot працює, поки ціна, яку ви запропонували, вищою за середню ціну за цей же інстанс. A reserved це знижка при довгому користуванні інстанса on-demand, яку можна придбати. Умовно кажучи, якщо ви довгий час використаєте on-demand інстанси за 23 долари, то щоб перейти на тариф reserved і платити по 12 доларів в місяць потрібно заплатити 50 одноразово [9].

Крім цього варто згадати про глобальну інфраструктуру Амазону, а саме про дата-центрах. У Північній Америці у Амазону 12 дата-центрів, в Європі - 8, в Південній Америці - 1, в Азії - 5 [9]. Тобто на Амазон можна отримати пристойну швидкість і пінг практично з будь-якої точки світу.

1.5. Висновки за розділом 1

Хмарні обчислення (англ. Cloud computing) - технологія распределѐнной обробки даних, в якій комп'ютерні ресурси і потужності надаються користувачеві як Інтернет-сервіс. Суть концепції хмарних обчислень полягає в наданні кінцевим користувачам віддаленого динамічного доступу до послуг, обчислювальних ресурсів і додатків (включаючи операційні системи і інфраструктуру) через інтернет.

На сьогоднішній день не більше 15% вітчизняних організацій на практиці застосовують хмарні технології з метою оптимізації своїх IT-інфраструктур. Фахівці продовжують запевняти, що Cloud Computing відкриває доступ до потужних ресурсів, а також дає реальну можливість одним стрибком подолати технологічну прірву, яка відділяє нас від більш розвинених країн.

Після аналізу послуг, пропонованих відомими провайдерами сервісу IaaS (вітчизняних і зарубіжних), порівняльної характеристики використовуваних ними платформ, увага була зупинена на хмарних сервісах від Amazon - Amazon Web Services. Саме вони були обрані основою для розгортання kubernetes кластеру.

РОЗДІЛ 2.

ОСНОВИ KUBERNETES, ІСТОРІЯ ПОЯВИ ТЕХНОЛОГІЇ ТА ПОРІВНЯННЯ ЇЇ З АНАЛОГАМИ

2.1. Основи KUBERNETES

Kubernetes є проектом з відкритим вихідним кодом, призначеним для управління кластером серверів Linux як єдиною системою. Kubernetes управляє і запускає контейнери Docker на великій кількості серверів, а так само забезпечує спільне розміщення та реплікацію великої кількості контейнерів. Проект був розпочатий Google і тепер підтримується багатьма компаніями, серед яких Microsoft, RedHat, IBM і Docker [29].

Компанія Google користується контейнерної технологією вже більше десяти років. Вона починала з запуску більш 2 млрд контейнерів протягом одного тижня. За допомогою проекту Kubernetes компанія отримала дуже позитивний досвід створення відкритої платформи, призначеної для масштабування та запуску контейнерів.

Проект переслідує дві мети. Якщо компанія користуєтеся контейнерами Docker, виникає питання про те, як масштабувати і запускати контейнери відразу на великій кількості хостів, а також як виконувати їх балансування. У kubernetes пропонується високорівнева API, що визначає логічне групування контейнерів, що дозволяє визначати пули контейнерів, балансувати навантаження, а також задавати їх розміщення.

Концепції Kubernetes [32]:

Nodes (node.md): Нода це машина/сервер в кластері Kubernetes.

Pods (pods.md): Pod це група контейнерів із загальними розділами, що запускаються як єдине ціле.

Replication Controllers (replication-controller.md): replication controller гарантує, що певна кількість «реплік» pod'и будуть запущені в будь-який момент часу.

Services (services.md): Сервіс в Kubernetes це абстракція яка визначає логічний об'єднаний набір pod і політику доступу до них.

Volumes (volumes.md): Volume (розділ) це директорія, можливо, з даними в ній, яка доступна в контейнері.

Labels (labels.md): Label'и це пари ключ / значення які прикріплюються до об'єктів, наприклад pod'ам. Label'и можуть бути використані для створення і вибору наборів об'єктів.

Kubectl Command Line Interface (kubectl.md): kubectl інтерфейс командного рядка для управління Kubernetes.

Працюючий кластер Kubernetes включає в себе агента, запущеного на нодах (kubelet) і компоненти майстра (APIs, scheduler, etc), поверх рішення з розподіленим сховищем. Наведена схема показує бажане, в кінцевому підсумку, стан, хоча все ще ведеться робота над деякими речами, наприклад: як зробити так, щоб kubelet (всі компоненти, на самом деле) самостійно запускався в контейнері, що зробить планувальник на 100%, що підключається. Графічна схема Kubernetes наведена у додатку 2.

Архітектура Kubernetes [34]:

- Нода Kubernetes. При погляді на архітектуру системи ми можемо розбити його на сервіси, які працюють на кожній ноді і сервіси рівня управління кластера. На кожній ноді Kubernetes запускаються сервіси, необхідні для управління нодою з боку майстра і для запуску додатків.

Звичайно, на кожній ноді запускається Docker. Docker забезпечить розгортання образів і запуск контейнерів.

- Kubelet. Kubelet управляє pod'амі їх контейнерами, образами, розділами, etc.
- Kube-Proxy. Також на кожній ноді запускається простий proxy-балансувальник. Цей сервіс запускається на кожній ноді і налаштовується в Kubernetes API. Kube-Proxy може виконувати найпростіше перенаправлення потоків TCP і UDP (round robin) між набором бекенд.
- etcd. Стан майстра зберігається в екземплярі etcd. Це забезпечує надійне зберігання конфігураційних даних і своєчасне оповіщення інших компонентів про зміну стану.
- Kubernetes API Server. Kubernetes API забезпечує роботу api-сервера. Він призначений для того, щоб бути CRUD сервером з вбудованою бізнес-логікою, реалізованої в окремих компонентах або в плагінах. Він, в основному, обробляє REST операції, перевіряючи їх і оновлюючи відповідні об'єкти в etcd (і подієво в інших сховищах).
- Scheduler. Scheduler прив'язує не запущені pod'и до нод через виклик / binding API.; планується підтримка множинних scheduler'ов і призначених для користувача scheduler'ов.
- Kubernetes Controller Manager Server. Всі інші функції рівня кластера представлені в Controller Manager. Наприклад, Ноди управляються і контролюються засобами node controller. Ця сутність в результаті може бути розділена на окремі компоненти, щоб зробити їх незалежно підключаються. Replication Controller - це механізм, який базується на pod API. В кінцевому рахунку планується перевести її на загальний механізм plug-in, коли він буде реалізований.

2.2. Історія еволюції KUBERNETES

З самого початку розвитку інформаційних технологій організації запускали додатки на фізичних серверах. Не було ніякого способу визначити межі ресурсів для додатків на фізичному сервері, і це викликало проблеми з розподілом ресурсів. Наприклад, якщо кілька додатків виконуються на фізичному сервері, можуть бути випадки, коли один додаток буде займати більшу частину ресурсів, і в результаті чого інші додатки будуть працювати гірше [33]. Рішенням цього було запустити кожен додаток на іншому фізичному сервері. Але це не масштабувати, оскільки ресурси використовувалися не повністю, через що організаціям було складно підтримувати безліч фізичних серверів.

Після почалася ера віртуального розгортання. Для вирішення була представлена віртуалізація. Вона дозволила запускати кілька віртуальних машин (ВМ) на одному фізичному сервері. Віртуалізація ізолює додатки між віртуальними машинами і забезпечує певний рівень безпеки, оскільки інформація одного додатка не може бути вільно доступна іншому додатку.

Віртуалізація дозволяє краще використовувати ресурси на фізичному сервері і забезпечує кращу масштабованість, оскільки додаток можна легко додати або оновити, крім цього знижуються витрати на обладнання і багато іншого. За допомогою віртуалізації можна перетворити набір фізичних ресурсів в кластер одноразових віртуальних машин.

Кожна віртуальна машина являє собою повноцінну машину, на якій виконуються всі компоненти, включаючи власну операційну систему, поверх віртуалізованого обладнання [19].

Ера контейнерів - контейнери схожі на віртуальні машини, але у них є властивості ізоляції для спільного використання операційної системи (ОС) між додатками. Тому контейнери вважаються легкими. Подібно віртуальній машині, контейнер має свою власну файлову систему, процесор, пам'ять,

простір процесу і багато іншого [58]. Оскільки вони не пов'язані з базовою інфраструктурою, вони переносяться між хмарами і збірками ОС.

Контейнери стали популярними через таких додаткових переваг як:

- Гнучке створення і розгортання додатків: простота і ефективність створення образу контейнера в порівнянні з використанням образу віртуальної машини.
- Безперервна розробка, інтеграція і розгортання: забезпечує надійну і чисту збірку і розгортання образу контейнера з швидким і простим відкатом (завдяки незмінності способу).
- Поділ завдань між Dev і Ops: створюйте образи контейнерів додатків під час складання / релізу, а не під час розгортання, тим самим відділяючи додатки від інфраструктури.
- Ідентичність довкілля при розробці, тестуванні та релізі: на ноутбуці працює так само, як і в хмарі.
- Працює на Ubuntu, RHEL, CoreOS, on-prem, Google Kubernetes Engine і в будь-якому іншому місці.
- Управління, орієнтоване на додатки: підвищує рівень абстракції від запуску ОС на віртуальному обладнанні до запуску додатка в ОС з використанням логічних ресурсів.
- Слабко зв'язаний, розподілені, гнучкі, виділені мікросервіси: замість монолітного стека на одній великій виділеної машині, додатки розбиті на більш дрібні незалежні частини, які можна динамічно розгортати і керувати.
- Ізоляція ресурсів: передбачувана продуктивність програми.
- Грамотне використання ресурсів: висока ефективність і компактність.

Для кращого розуміння різниці між традиційним розгортанням, віртуалізацією та контейнеризацією наведений Рис.2.1.

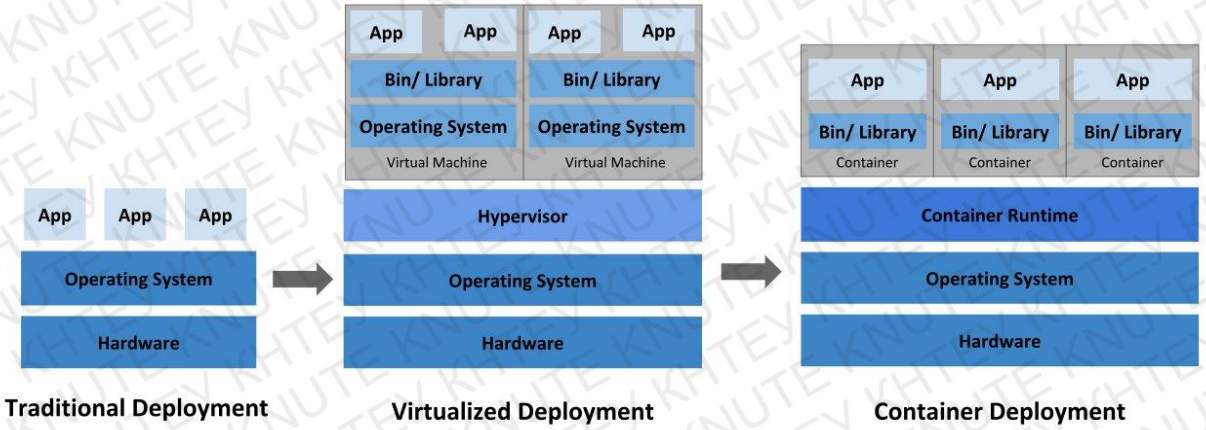


Рис. 2.1. Порівняння традиційне розгортанням, віртуалізацію та контейнеризацію

Джерело: створено автором на основі джерела [36].

Контейнери - відмінний спосіб зв'язати і запустити ваші програми. У виробничому середовищі необхідно управляти контейнерами, які запускають додатки, і гарантувати відсутність простоїв. Наприклад, якщо контейнер виходить з ладу, необхідно запустити інший контейнер. Не було б простіше, якби така поведінка обробляється системою?

Ось тут Kubernetes приходить на допомогу. Kubernetes дає вам фреймворк для гнучкої роботи розподілених систем. Він займається масштабуванням і обробкою помилок в додатку, надає шаблони розгортання і багато іншого.

Kubernetes надає:

- Моніторинг сервісів і розподіл навантаження. Kubernetes може виявити контейнер, використовуючи ім'я DNS або власний IP-адреса. Якщо трафік в контейнері високий, Kubernetes може збалансувати навантаження і розподілити мережевий трафік, щоб розгортання було стабільним.

- Оркестрації сховища Kubernetes дозволяє автоматично монтувати систему зберігання за вашим вибором, таку як локальне сховище, провайдери загальнодоступного хмари і багато іншого.
- Автоматичне розгортання і відкати. Використовуючи Kubernetes можна описати бажаний стан розгорнутих контейнерів і змінити фактичний стан на бажаний. Наприклад, ви можете автоматизувати Kubernetes на створення нових контейнерів для розгортання, видалення існуючих контейнерів і розподілу всіх їх ресурсів в новий контейнер.
- Автоматичний розподіл навантаження. Kubernetes може розмістити контейнери на ваших вузлах так, щоб найбільш ефективно використовувати ресурси.
- Самоконтроль. Kubernetes перезапускає, замінює і завершує роботу контейнерів, які не проходять певну перевірку працездатності, і не показує їх клієнтам, поки вони не будуть готові до обслуговування.
- Управління конфіденційною інформацією і конфігурацією. Kubernetes може зберігати і управляти конфіденційною інформацією, такою як паролі, OAuth-токени і ключі SSH. Ви можете розгортати і оновлювати конфіденційну інформацію і конфігурацію додатка без змін образів контейнерів і не розкриваючи конфіденційну інформацію в конфігурації стека.

2.3. Порівняння основних конкурентів технології

На даний момент існує декілька основних (популярних та підтримуваних) оркестраторів для мікросервісних архітектур, які безпосередньо є конкурентами для технології kubernetes. в них також можна кластеризувати розгортання додатків та сервісів. Серед найпопулярніших це:

- Docker swarm
- Docker Swarmmode

- Kubernetes
- Mesos

Таблиця 2.1.

Коротка інформація згідно сайтів оркестраторів

Оркестратор	**Розробник**	**Рік випуску**	**Статус**	**Підтримка**	**Розмір спільноти**
DockerSwarm	Docker	2015	Активний, але перенесений у основний Docker Engine	+	4k+Stars,800+ Forks,150+ Contributors
Kubernetes	CNCF	2015	Активний	Від третіх сервісів	20k+ Stars, 7k+ Forks,1k+Contributors
Mesos	Apache Software Foundation	2016	Активний	+	2k+ Stars, 1k+Forks,200+ Contributors

Джерело: створено автором на основі джерел [34]-[35] включно.

Повне та детальне порівняння оркестраторів наведене в додатку 3.

На основі таблиці порівнянь оберемо основних претендентів, це:

- Kubernetes
- DockerSwarm
- Mesos DCOS

Трохи детальніше розглянемо Kubernetes та Mesos, як основні, оскільки Docker Swarm, хоч і є доволі простим та легким до опанування, але не надасть усього необхідного функціоналу, в основному нас більше цікавить можливість розгортати поди Kubernetes, а також гнучкість керування DCOS.

Перед детальним розгляданням продуктів, сформуємо належним чином потреби мережі:

- Гнучкість розгортання
- Надійність сервісу
- Балансування навантаження
- Ізольованість ресурсів
- Гранулярність доступу
- Легкість опанування
- Розвинутий ком'юніті
- Сервіс має бути безкоштовним

Сформовані вимоги є мінімальними можуть не відображати усіх потреб мережі, проте є необхідними та достатніми для мережі.

Docker Swarm:

- Доволі стабільна робота, через постійну підтримку та механізм само поновлення (перезапуск контейнерів, що зупинилися)
- Внутрішній балансувальник –відсутній
- Внутрішнє нативне логування відсутнє
- Моніторинг – нативний відсутній
- Присутній нативный драйвер мережі
- Автоматичне масштабування – відсутнє
- Документація - доволі заплутана, через переніс сервісу у ядро іншого

Kubernetes:

- Стабільний
- Присутній внутрішній балансувальник подів.
- Логування – присутнє, проте так само найкращим виходом буде використання ELK

- Моніторинг – базовий присутній, що надає запис базових метрик у бекенд, вказаний користувачем
- Нативний драйвер мережі –відсутній
- Автоматичне масштабування –присутнє
- Документація- доволі структурована

Mesos:

- Стабільний
- Присутній внутрішній балансувальник родів.
- Логування – присутнє, проте так само найкращим виходом буде використання ELK
- Моніторинг – базовий присутній, що надає запис базових метрик у бекенд, вказаний користувачем
- Нативный драйвер мережі –відсутній
- Автоматичне масштабування – залежить від фреймворку
- Документація- дуже детальна та структурована

2.4. Висновки за розділом 2

В ході роботи над розділом мною був описаний процес розвитку та еволюції технологій які оркеструють контейнерами, надають багато додаткових переваг при управлинні мікросервісною архітектурою та спрощують роботу безпосередньо сервісом.

Проаналізувавши потреби та порівнявши аналоги – мною був обґрунтований вибір платформи Kubernetes, за наступними критеріями:

Популярність – інструмент постійно підтримується.

Найкращий із представлених механізм самовідновлення – контейнери, що, з якоїсь причини перестали працювати - відновлюються одразу ж після

відключення, на відміну від того, що інші оркестратори просто відмічають, контейнер як нефункціонуючий.

Легкість користування – описувати інфраструктуру сервісів значно простіше, так само просто описувати конфігурацію різних інструментив що виконують супутні функції.

Найстабільніший API серед претендентів – 99% відповідей гарантовані, найдовший час відповіді – 1 секунда.

Ефективний розподіл ресурсів - так як розгортання кластеру буде відбуватися у хмарному середовищі - дуже важливо створити систему що буде використовувати мінімальну кількість ресурсів, відповідно і мінімальну кількість грошей.

Максимальна самостійність системи - Kubernetes перезапускає, замінює і завершує роботу контейнерів, які не проходять певну перевірку працездатності, і не показує їх, поки вони не будуть готові до обслуговування. Також в Kubernetes є багато інструментів для автоматизації функціонування системи. Відповідно, при правильному налаштуванні - мікросервісна архітектура не потребує втручання і функціонує повністю самостійно.

РОЗДІЛ 3.

РОБОТА З AMAZON ПРОВАЙДЕРОМ. РОЗГОРТАННЯ KUBERNETES КЛАСТЕРУ ТА НАПОВНЕННЯ ЙОГО ФУНКЦІОНАЛОМ

3.1. Робота з серверами та сіткою у хмарі

Кластер Kubernetes буде встановлений, як група віртуальних машин на AWS EC2 сервісах. У якості мережі та балансувальника навантаження буде використанний load balancer у зв'язку із VPC та security groups. Сервіс документообігу буде розгорнутий у середовищі Kubernets. Kubernetes буде встановлений на операційній системі ubuntu 16.04, з огляду через офіційну підтримку та легкість конфігурації.

Первинна конфігурація буду додана до серверу прі первинній конфігурації. Технологія себе гарно зарекламувала у сервісі, як EC2 від AWS, де необхідні команди, у тестовому вигляді, передаються через “User Data” – поле у формі створення сервера. Ці команди одразу виконуються після завантаження системи. Перевагою існування цього поля є те, що за допомогою різноманітних інструментів Configuration Management, наприклад Ansible, чи то Infrastructure As a Code, наприклад Terraform, можна одразу підняти велику кількість віртуальних хостів, не витрачаючи часу на індивідуальне налаштування.

Для початку роботи потрібно зайти на https://aws.amazon.com/ і пройти веріфікацію. Після цього можна потрапити до основного меню AWS, що можна побачити на знімку з екрану рис. 3.1.

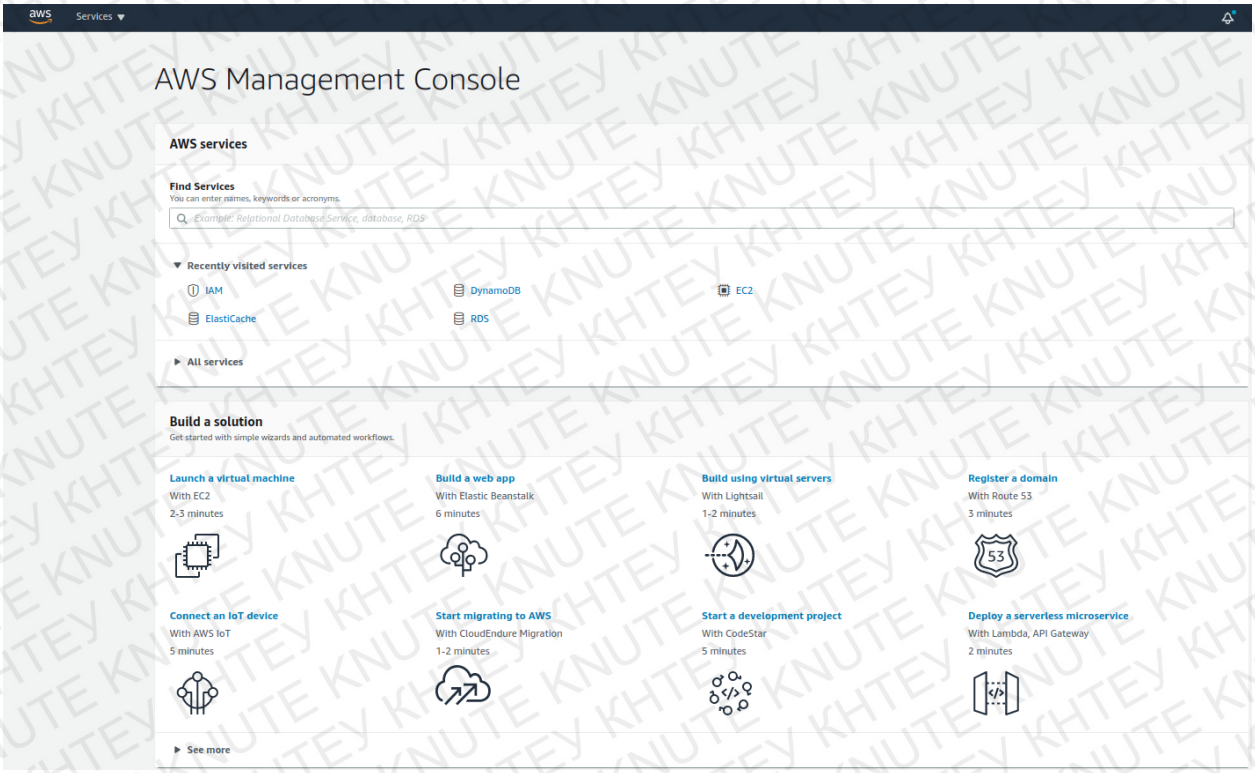


Рис. 3.1. Знімок з екрану Основне меню у AWS

Джерело: створено автором.

Починаємо створювати пулл віртуальних машин EC2. для цього потрібно відкрити консоль управління і перейти на вкладку Amazon EC2.

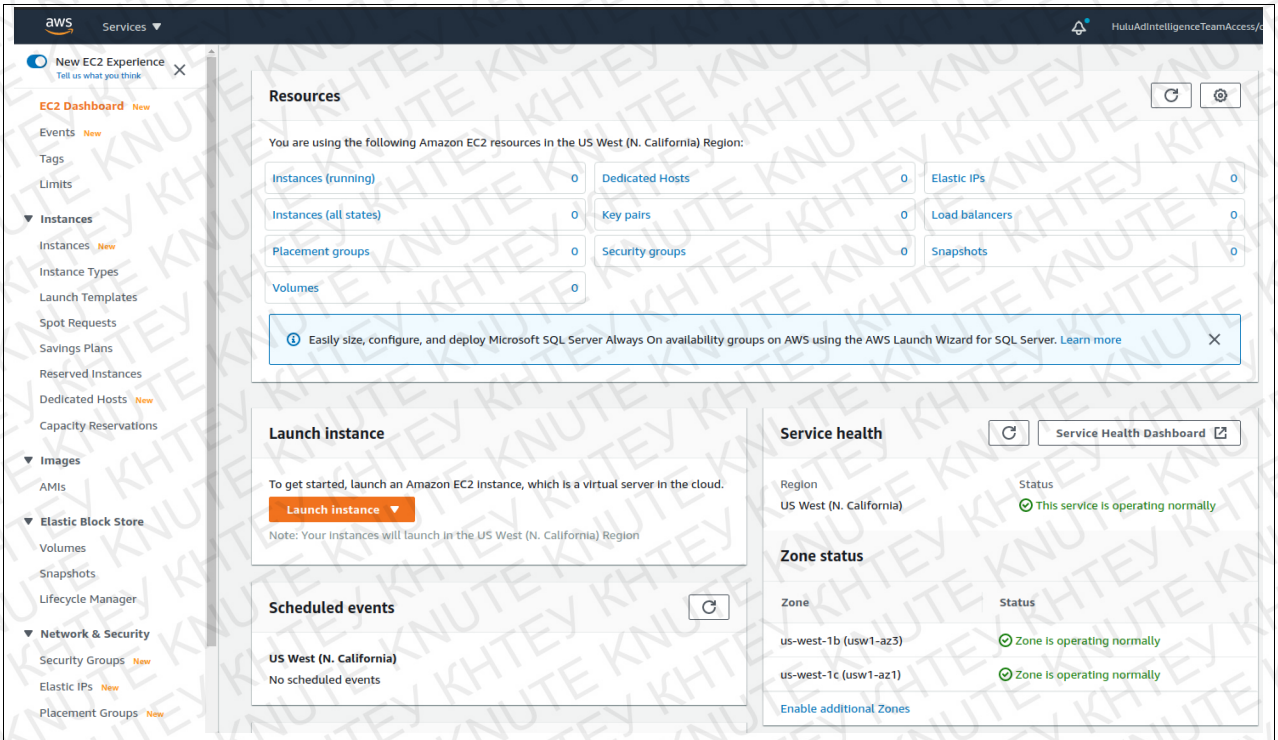


Рис. 3.2. Знімок з екрану Меню EC2 у AWS

Джерело: створено автором.

Для початку, необхідно вибрати регіон в якому буде працювати EC2 сервер і натиснути на кнопку Launch Instance.

У термінології Амазон, регіон (Region) - це регіональне розташування дата-центрів в яких будуть працювати віртуальні машини. Регіони, в свою чергу, поділяються на зони доступності (Availability Zone). Для забезпечення максимального рівня відмовостійкості застосування потрібно розташовувати його в різних регіонах, а всередині регіону розподіляти EC2 екземпляри по різних зонах доступності.

Сервіси AWS доступні в дата-центрах розташованих в США, Ірландії, Сінгапурі та Японії. Для користувачів з України, логічним буде використання регіону в Франкфурті, Німеччина (eu-central-1).

Далі необхідно вибрати тип операційної системи EC2. Можна створити свою або використовувати вже готові образи (AMI). Приклад операційних

систем наведений на знімку з екрану рис. 3.3. Для простоти давайте візьмемо готовий образ типу Basic 32-bit Amazon Linux AMI.

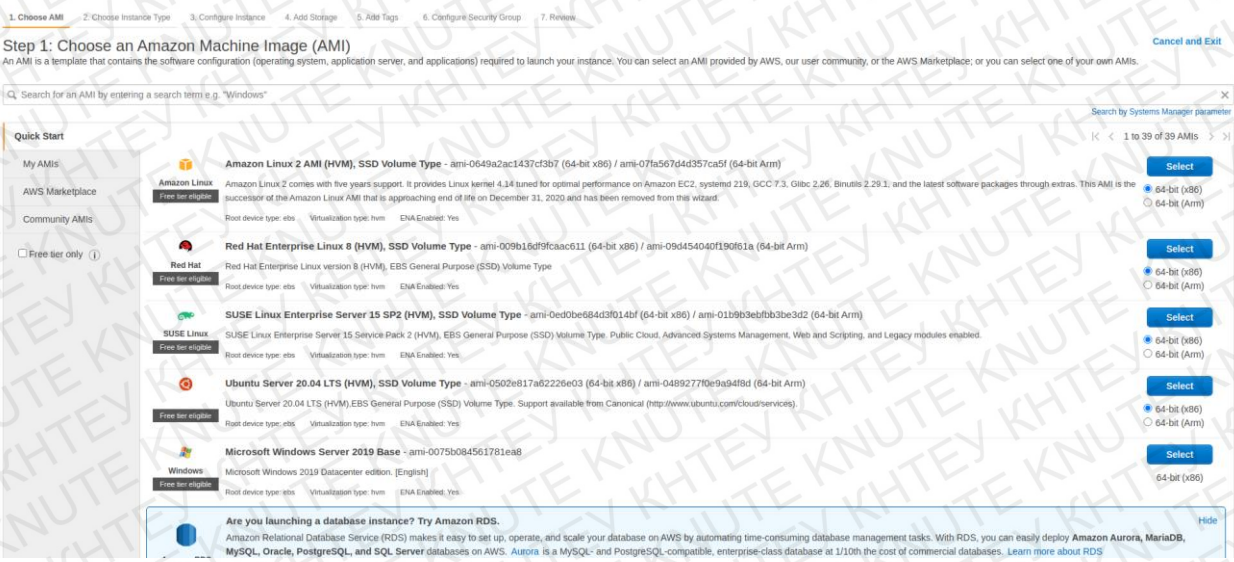


Рис. 3.2. Знімок з екрану Меню Amazon Machine Image у AWS

Джерело: створено автором.

На наступному кроці вибираємо тип та об'єм пам'яті. Інші параметри при створенні мною були залишені без змін.

На наступних двох кроках можна внести зміни в розширені налаштування сервера. Я залишу значення цих параметрів за замовчуванням і перейдемо до діалогу створення приватного SSH ключа, який, надалі, ми будемо використовувати для доступу до віртуальної машини.

На кроці Create Key Pair, потрібно ввести будь-яку вподобану ім'я для SSH ключа і після переходу за посиланням Create & Download your Key Pair, ключ буде створено і буде запропоновано зберегти його на локальному комп'ютері.

Наступне, що необхідно буде зробити - це створити групу безпеки (Security Group) для EC2 сервера. Для цього, на кроці Configure Firewall,

вибираємо Create a new Security Group, і даємо цій групі будь-яку вподобану назву. Це можна побачити на знімку з екрану рис. 3.4.

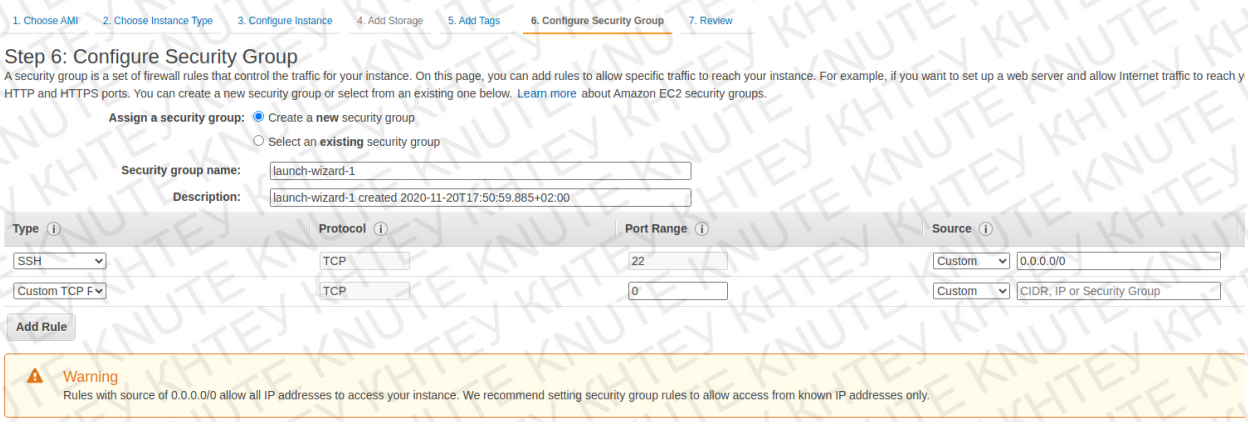


Рис. 3.4. Знімок з екрану Меню створення Security Group у AWS

Джерело: створено автором.

Фактично, група безпеки (Security Group) - це засіб забезпечує фільтрацію мережевих з'єднань для EC2 сервера. Іншими словами - це файрвол. Відповідно для забезпечення доступу до сервісів, які будуть працювати у EC2 сервері необхідно дозволити доступ до певних TCP портів. Для Kubernetes потрібно зробити сервер доступним для вхідних з'єднань по протоколах HTTP, HTTPS і SSH. За бажанням, можна додати додаткові правила.

Список ресурсів використовуваних створеним EC2 сервером можна подивитися в панелі управління EC2. Приклад можна побачити на рис. 3.5.

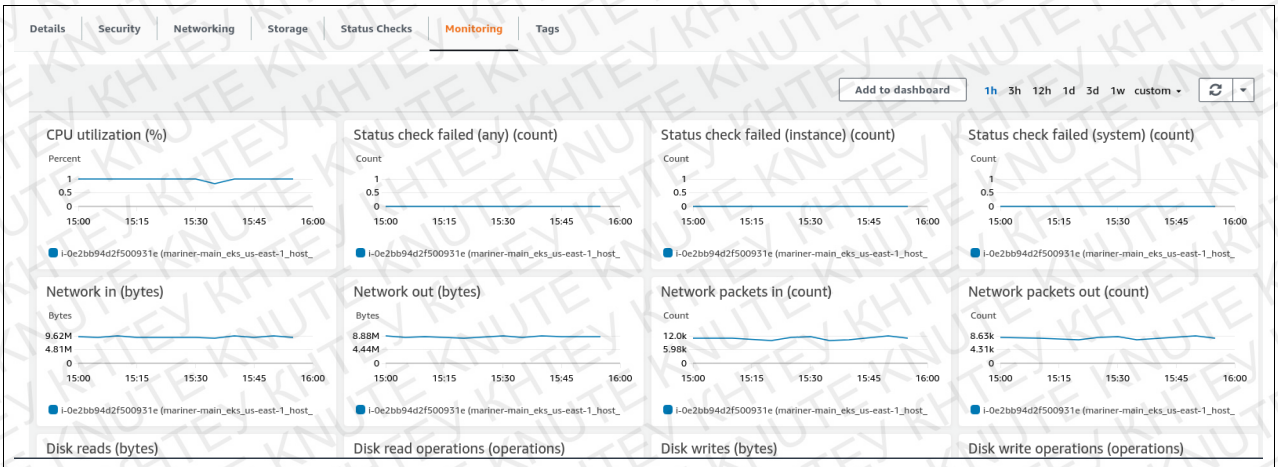


Рис. 3.5. Знімок з екрану Меню моніторингу EC2 сервера у AWS

3.2. Розгортання та конфігурація KUBERNETES кластеру

Для інсталяції Kubernetes потрібно спочатку створити дисковий простір для мастера та трьох нод. Команди для цього виглядають наступним чином:

sudo su

mkdir -p /var/lib/libvirt/images/

cd /var/lib/libvirt/images/ git clone -b blog_post_v2

https://github.com/andrewmichaelsmith/xen-coreos- kube.git coreos

cd coreos

wget https://beta.release.core-os.net/amd64-usr/current/coreos_production_xen_image.bin.bz2 -O

| bzcat > coreos_production_xen_image.bin

#To pad out extra space in the image. #TODO: Undoubtedly a better way than this dd if=/dev/zero of=tempfile bs=1G count=2

cat tempfile >> coreos_production_xen_image.bin

cp coreos_production_xen_image.binmaster1.bin

cp coreos_production_xen_image.bin node1.bin

cp coreos_production_xen_image.bin node2.bin

cp coreos_production_xen_image.binnode3.bin

Та інсталяція на машини. Також код наведений нижче:

```
KEY=$(cat ~/.ssh/id_rsa.pub)
sed"s#SSH_KEY#
$KEY#g" < master1/ubuntu/latest/user_data.tmpl >
master1/ubuntu/latest/user_data
sed "s#SSH_KEY#$KEY#g" < node1/ubuntu/latest/user_data.tmpl >
node1/ubuntu/latest/user_data
sed "s#SSH_KEY#$KEY#g" < node2/ubuntu/latest/user_data.tmpl >
node2/ubuntu/latest/user_data
sed"s#SSH_KEY#
$KEY#g" < node3/ubuntu/latest/user_data.tmpl >
node3/ubuntu/latest/user_data
```

Код для генерації сертифікатів:

```
cd certs
openssl genrsa -out ca-key.pem 2048
openssl req -x509 -new -nodes -key ca-key.pem -days 10000 -out ca.pem -subj "/CN=kube-ca"
openssl genrsa -out apiserver-key.pem 2048 openssl req -new -key apiserver-key.pem -out
apiserver.csr -subj "/CN=kube-apiserver" -config openssl.cnf
openssl x509 -req -in apiserver.csr -CA ca.pem - CAkey ca-key.pem -CAcreateserial -out
apiserver.pem
-days 365 -extensions v3_req -extfile openssl.cnf cd ..
```

Та їх встановлення на мастер ноди:

```
#Total hack, so it's indented correctly when we move it in to .yml
sed-i's/^/ /'certs/*.pem
sed -i $'/CA.PEM/ {r certs/ca.pem\n d}' master1/openstack/latest/user_data
sed -i $'/APISERVER.PEM/ {r certs/apiserver.pem\n d}' master1/openstack/latest/user_data
sed -i $'/APISERVER-KEY.PEM/ {r certs/apiserver- key.pem\n d}'
master1/openstack/latest/user_data
```

Для перевірки налаштування:

```
curl 'https://validate.core-os.net/validate' -XPUT
```

```
--data-binary'@master1/openstack/latest/user_data'
| python -mjson.tool
curl 'https://validate.core-os.net/validate' -X PUT
--data-binary '@node1/openstack/latest/user_data' |
curl 'https://validate.core-os.net/validate' -X PUT
--data-binary '@node2/openstack/latest/user_data' | python -mjson.tool
curl 'https://validate.core-os.net/validate' -X PUT
--data-binary '@node3/openstack/latest/user_data' | python -mjson.tool
```

Якщо усі команди передали “null” – результат є успішним. Тоді Kubelete запуститься і завантажить hypercube.

Описаний вище процес запустить 4 віртуальні машини із cloud-config. Сам процес, Kubelete запуститься і завантажить hypercube, контейнери для kube-proxy стартують на нодах, запустяться контейнери для:

- Api server
- Controllermanager
- Scheduler намастері

Для моніторинг процесу – потрібно підключитися до консолі та промоніторити завантаження ноди.

Для становлення kubectl:

```
curl -O https://storage.googleapis.com/kubernetes-
release/release/v1.2.3/bin/linux/amd64/kubectl chmod +x kubectl
mv kubectl /usr/local/bin/kubectl
```

Після чого з'являється можливість комунікації та роботи із кластером.

Перевіримо статус кластера із системи:

```
root@xen# kubectl -s http://192.168.122.254:8080 get nodes
NAME STATUS AGE192.168.122.2 Ready 1m
192.168.122.254 Ready 1m
Ready 1m
```

Тепер створимо іменний простір(Namespace), та назвемо його kube-system.

```
curl -H "Content-Type: application/json" -XPOST -
d'{"apiVersion":"v1","kind":"Namespace","metadata":
{"name":"kube-system"}}' "http://192.168.122.254:8080/api/v1/namespaces"
```

Таким чином на гіпервізор і встановлені 4 віртуальних машини із ubuntu в якості операційної системи. На них у свою чергу встановлений кластер, що складається з одного мастеру та трьох нод. Схему кластеру можна побачити на рис.3.6.

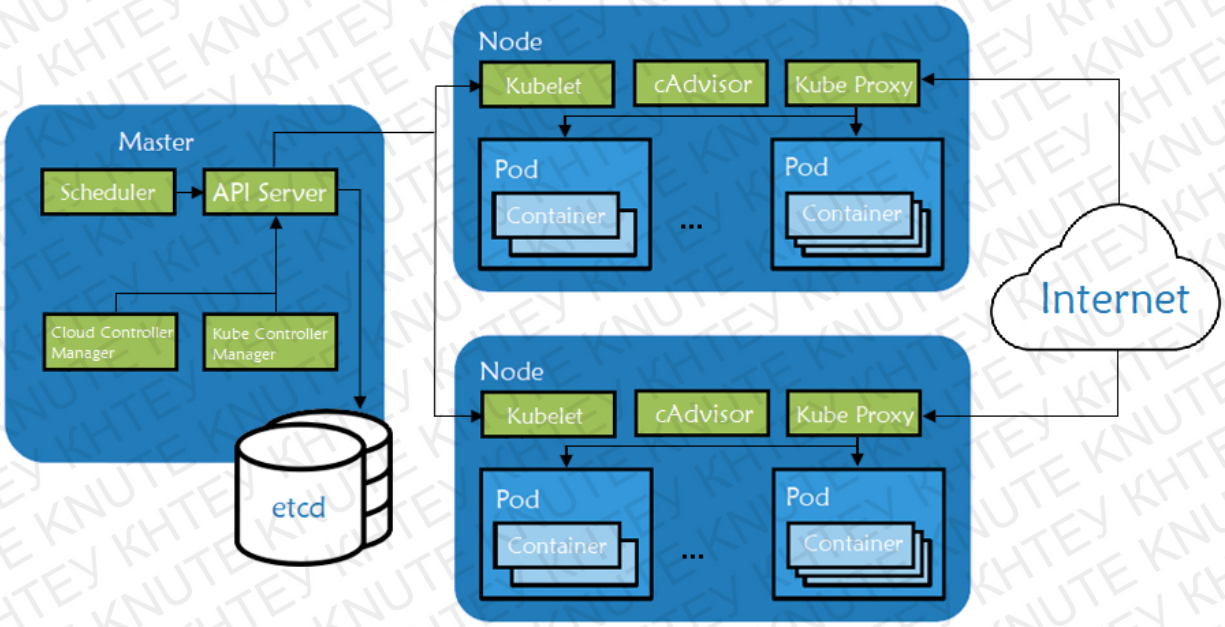


Рис. 3.6. Схема створеного Kubernetes кластеру

Джерело: створено автором.

Функціонально кластер виглядає наступним чином:

- Flannel service - обслуговує однорангову мережу. Дозволяє контейнерам обмінюватись трафіком.
- Etcd service – Відповідає за збереження стану kubernetes

- Docker service – власне кажучи сервіс, за допомогою якого kubernetes запускає контейнери
- Kubelet service – об'єднує ноди у кластер, дозволяє запуск інших додатків Kubernetes

3.3. Наповнення кластеру функціоналом

3.3.1. Автентифікація

У сам kubernetes та сервіси, що запущені на ньому буде необхідно забезпечити доступ. Цей пункт реалізації доволі простий, оскільки все що буде потрібно – це підключення до вже існуючого домену, що є абсолютно тривіальним:

Для виконання цього пункту – виконаємо наступне:

Встановимо на віртуальній машині Xen keystone service, що надасть доступ до дерева домену Kubernetes із наступними налаштуваннями:

Генеруємо секретний токен SSL:

openssl rand -hex 10

Що у даному випадку дорівнює:

«eb18653b1907c4c0e97f»

/etc/keystone/keystone.conf

[DEFAULT]

admin_token = eb18653b1907c4c0e97f public_bind_host

admin_bind_host

[identity] driver = ldap

[ldap]

chase_referrals =false page_size = 1000 query_scope =sub

suffix = CN=Administrator, DC=cs-lab,DC=cad,DC=ntu- kpi,DC=kiev,DC=ua

url = ldap://dragon2:389 use_tls = false

#If use_tls = true then the following parameters will need to be uncommented

#tls_req_cert = demand #tls_cacertfile =<path-to-cert-file>

user =Administrator@kiev.uapassword = “користувацький_пароль” user_allow_create = false

user_allow_update = false user_attribute_ignore = enabled user_id_attribute = CN
user_mail_attribute = mail user_name_attribute = sAMAccountName user_objectclass = organizationalPerson
user_tree_dn = OU=Users,DC=cs-lab,DC=cad,DC=ntu- kpi,DC=kiev,DC=ua
[eventlet_server_ssl] enable = True
certfile = /etc/keystone/ssl/certs/keystone.pem keyfile = /etc/keystone/ssl/private/keystonekey.pem ca_certs = /etc/keystone/ssl/certs/ca.pem
cert_required = False
[ssl]
ca_key = /etc/keystone/ssl/private/cakey.pem key_size = 2048
valid_days = 3650 cert_subject=/C=IN/ST=Administrator/

Налаштування кубернетес для авторизації:

v1
kind: Config preferences: {} clusters:
- cluster:
certificate-authority-data:LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUR1VENDQXFHZ0F3SUJBZ0lKQUw4NFMxUwUTNjTUEwR0NTcUdTSWIzRFFFQkN3VUFNRUl4UURBK0JnTlYKQkFNV U56RTVNaTR4TmpndU1USXlMakUzTXl4SlVEb3hOekl1TVRZdU1DNHlMRWxRT2pFM01 pNHhOaTQxT0M0dwpRREUwmprNE9ERTFOelV3SGhjTk1UWXdOek13TVRJeU5qRTFXa GNOTWpZd056STRNVEl5TmpFMVdqQkNNVUF3ClBnWURWUVFERkRjeE9USXVNVFk 0TGpFeU1pNHhOek1zU1ZBNk1UY3lMakUyTGpBdU1peEpVRG94TnpJdU1UWXUKTlRnd U1FQXhORF
server: https://kauth.cad.ntu-kpi.kiev.ua:443 name: cube.cluster
contexts:
context:
cluster: cube.cluster namespace: administrator user: administrator
name: administrator.cube.cluster users:
name:administrator user:
username: administrator password: password
current-context: administrator.cube.cluster

Таким чином налаштований кластер для автентифікації.

3.3.2. Моніторинг

Сучасні кластерні мікросервісні архітектури, що використовує Kubernetes, де слабозв'язані контейнери створюються та знищуються, в залежності від того, коли і де вони необхідні, вимагає нового підходу до ефективного моніторингу. Вже недостатньо, моніорити навантаження і активність на конкретній машині, коли робота, що призначається цій машини може змінюватися згідно вимог кластеру і додатку в цілому. Якщо ж архітектура розроблена таким чином, що процес може зникнути, не можна сказати, чи є відмова критичною, спостерігаючи за одним тільки процесом.

Так само, як Kubernetes змушує архітекторів більше думати про додатки та навантаження, аніж про процеси, так само і процес моніторингу. Кластеро-центричний моніторинг надає перевагу у тому, що користувач може реагувати на тренди поведінки на рівні додатків, реагувати швидко та отримувати менше інформаційного шуму.

Prometheus– відкритий продукт, що був спеціально створений за для моніторингу кластерних систем, таких як Kubernetes, зокрема модель назначень та керування нодами. Наразі Prometheus підтримує Api Kubernetes, що дуже зручно, оскільки запит напряму до кластера від системи моніторингу буде швидшим, аніж використання третіх інструментів, також слід зауважити, що Kubernetes має набір заздалегідь підготовлених метрик, що ними може скористатись Prometheus.

Для встановлення потрібно створити наступні конфігураційні файли: deployment.yaml - що зображений на знімку з екрану рис. 3.7., та configmap.yaml зі всіма конфігураціями - що зображено на знімку з екрану рис. 3.8.

```
---
apiVersion: extensions/v1beta1
kind: Deployment
metadata:
  labels:
    name: prometheus-deployment
  name: prometheus
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: prometheus
    spec:
      containers:
      - image: quay.io/prometheus/prometheus:v1.0.1
        name: prometheus
        command:
        - "/bin/prometheus"
        args:
        - "-config.file=/etc/prometheus/prometheus.yml"
        - "-storage.local.path=/prometheus"
        - "-storage.local.retention=24h"
        ports:
        - containerPort: 9090
          protocol: TCP
        volumeMounts:
        - mountPath: "/prometheus"
          name: data
        - mountPath: "/etc/prometheus"
          name: config-volume
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
          limits:
            cpu: 500m
            memory: 2500Mi
      volumes:
      - emptyDir: {}
        name: data
      - configMap:
          name: prometheus-config
        name: config-volume
---
```

Рис. 3.7. Знімок з екрану Deployment конфігурація для prometheus

Джерело: створено автором.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: prometheus-config
data:
  prometheus.yml: |
    global:
      scrape_interval: 30s
      scrape_timeout: 30s
    scrape_configs:
    - job_name: 'prometheus'
      static_configs:
        - targets: ['localhost:9090']
    - job_name: 'kubernetes-cluster'
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - api_servers:
        - 'https://kubernetes.default.svc'
        in_cluster: true
        role: apiserver
    - job_name: 'kubernetes-nodes'
      scheme: https
      tls_config:
        ca_file: /var/run/secrets/kubernetes.io/serviceaccount/ca.crt
        insecure_skip_verify: true
      bearer_token_file: /var/run/secrets/kubernetes.io/serviceaccount/token
      kubernetes_sd_configs:
      - api_servers:
        - 'https://kubernetes.default.svc'
        in_cluster: true
        role: node
      relabel_configs:
      - action: labelmap
        regex: __meta_kubernetes_node_label_(.+)
    - job_name: 'kubernetes-service-endpoints'
      scheme: https
      kubernetes_sd_configs:
      - api_servers:
        - 'https://kubernetes.default.svc'
        in_cluster: true
        role: endpoint
      relabel_configs:
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scrape]
        action: keep
        regex: true
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_scheme]
        action: replace
        target_label: __scheme__
        regex: (https?)
      - source_labels: [__meta_kubernetes_service_annotation_prometheus_io_path]
        action: replace
        target_label: __metrics_path__
        regex: (.+)
      - source_labels: [__address__, __meta_kubernetes_service_annotation_prometheus_io_port]
        action: replace
        target_label: __address__
        regex: (.+)(?::\d+);(\d+)
        replacement: $1:$2
```

Рис. 3.8. Знімок з екрану Configmap конфігурація для prometheus

Джерело: створено автором.

За необхідності - сконфігуруємо сховище, куди прометеус зберігає дані, для цього необхідно замінити emptyDir на абсолютний шлях у конфігурації.

1) У стоковій конфігурації скористаймося прокидуванням портів

```
$ kubectl get pods-lapp=prometheus -o name | \ sed's/^.*\///'| \ xargs -I{} kubectl port-forward {}9090:9090
```

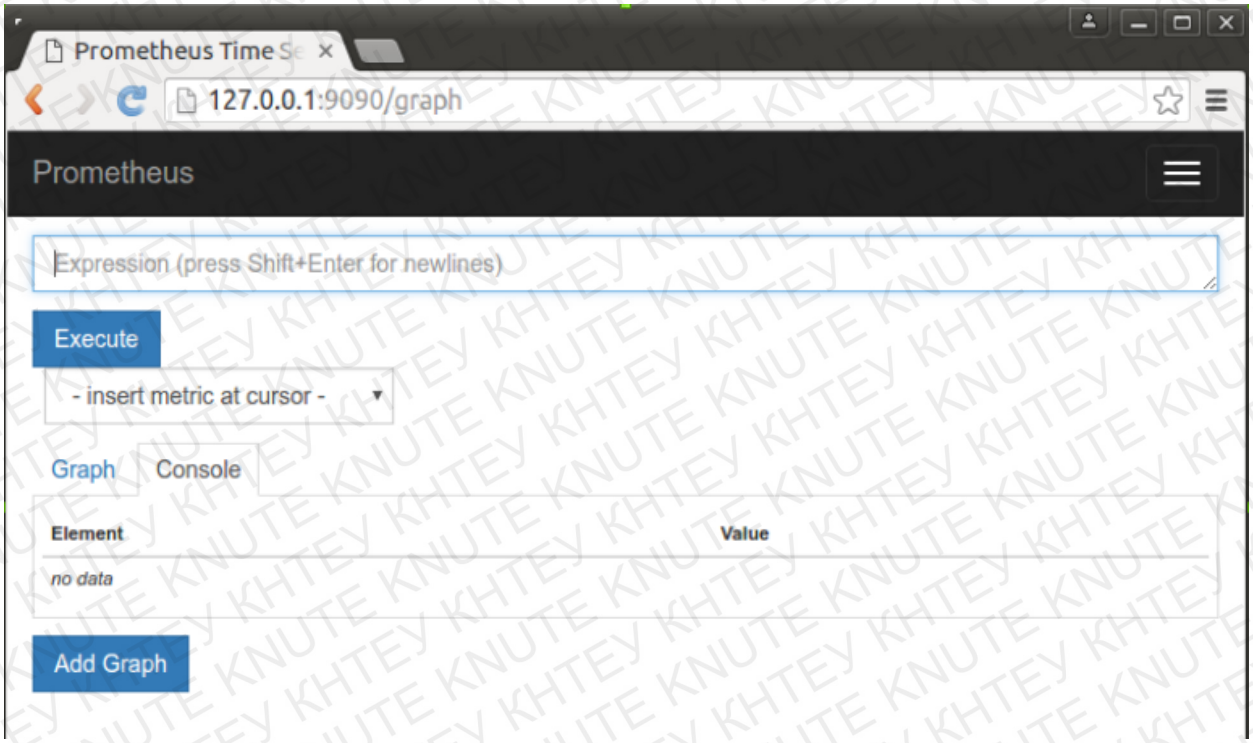


Рис. 3.9. Знімок з екрану Приклад інтерфейсу сервісу prometheus

Джерело: створено автором.

Система піднята і готова до налаштування, що можна побачити на знімку з екрану 3.9.

Тепер створимо декілька запитів для ілюстрації ідеї. Можна відобразити зайнятість CPU, RAM, вводу-виводу, та інше, за допомогою запиту:

```
container_memory_usage_bytes{image="CONTAINER:VERSION"}
```

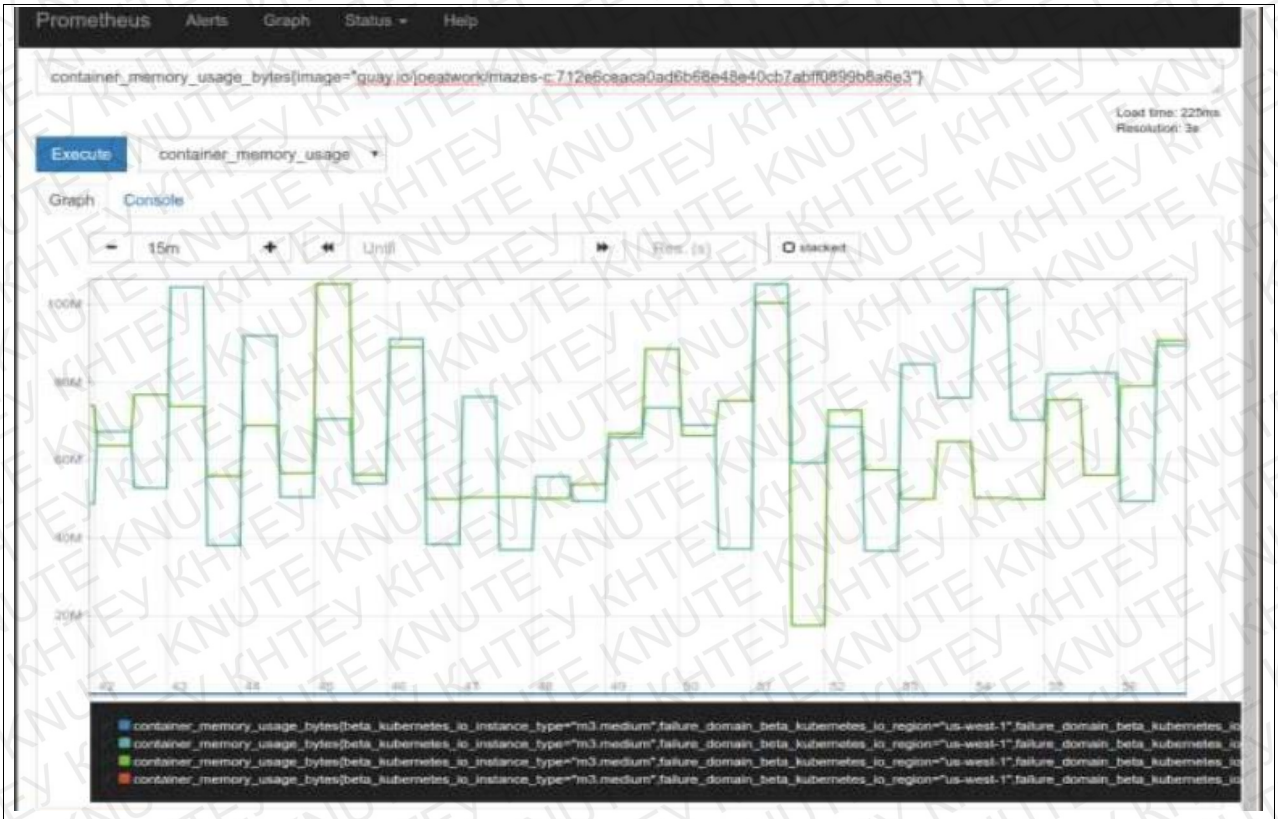


Рис. 3.10. Знімок з екрану Графіки роботи контейнера

Джерело: створено автором.

Також можна моніторити сам kubernetes, наприклад:

```
sum(container_memory_usage_bytes{kubernetes_namespace="kube-system"})
sum(container_memory_usage_bytes{kubernetes_namespace="kube-system",
kubernetes_pod_name=~"kube-dns-v11.*"})
```

Дані запити відобразять кількість ресурсів, що використовує неймспейс kube-system у першому випадку, та скільки використовує окремий под.

Рис. 3.11. Знімок з екрану Графік роботи неймспейсу kube-system

Джерело: створено автором.

Таким чином у нас є налаштована система моніторингу, що можна використовувати. Вона легковісна, може сповіщати користувача через канали пошти, чи інші зручні мессенжери. Дозволяє гнучке налаштування.

3.3.3. Сервіс документообігу

У якості сервісу документообігу буде запропонований ownCloud, через швидкість налаштування та легкість використання. Також сервіс можна встановити таким чином, щоб він був веб формою до вже існуючого сховища.

Скористаємось для демонстрації скористаємось наступними параметрами:

Mysql, є реляційною базою даних, що буде містити користувацькі логіни та паролі, у разі відмови доступу до ActiveDirectory.

Owncloud – власне система розподіленого доступу.

```
kind: Service apiVersion: v1 metadata:
name: mysql spec:
ports:
- name: mysql port: 3306 protocol: TCP
selector: app: mysql

---
apiVersion: extensions/v1beta1 kind: Deployment
metadata: name: mysql
spec:
replicas: 1 template:
metadata: labels:
app: mysql spec:
containers:
name: mysql
image: mysql:5.7.12 ports:
containerPort: 3306 env:
name: MYSQL_ROOT_PASSWORD
value: rootpwd
name: MYSQL_DATABASE
value: owncloud
name: MYSQL_USER value: owncloud
name: MYSQL_PASSWORD
value: owncloud
```

Рис. 3.12. Знімок з екрану Налаштування MYSQL

Джерело: створено автором.

Для налаштування ownCloud спочатку буде потрібно встановити на комп'ютер helm:

```
$ curl -fsSL -o get_helm.sh https://raw.githubusercontent.com/helm/helm/master/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

Тепер за допомогою Helm можна легко встановити ownCloud.

Створивши конфігураційні файли запускаємо контейнери:

```
kubectl create –f mysql.yaml
helm repo add bitnami https://charts.bitnami.com/bitnami
helm install my-release bitnami/owncloud
```

Таким чином маємо запущений owncloud, перевіримо його:

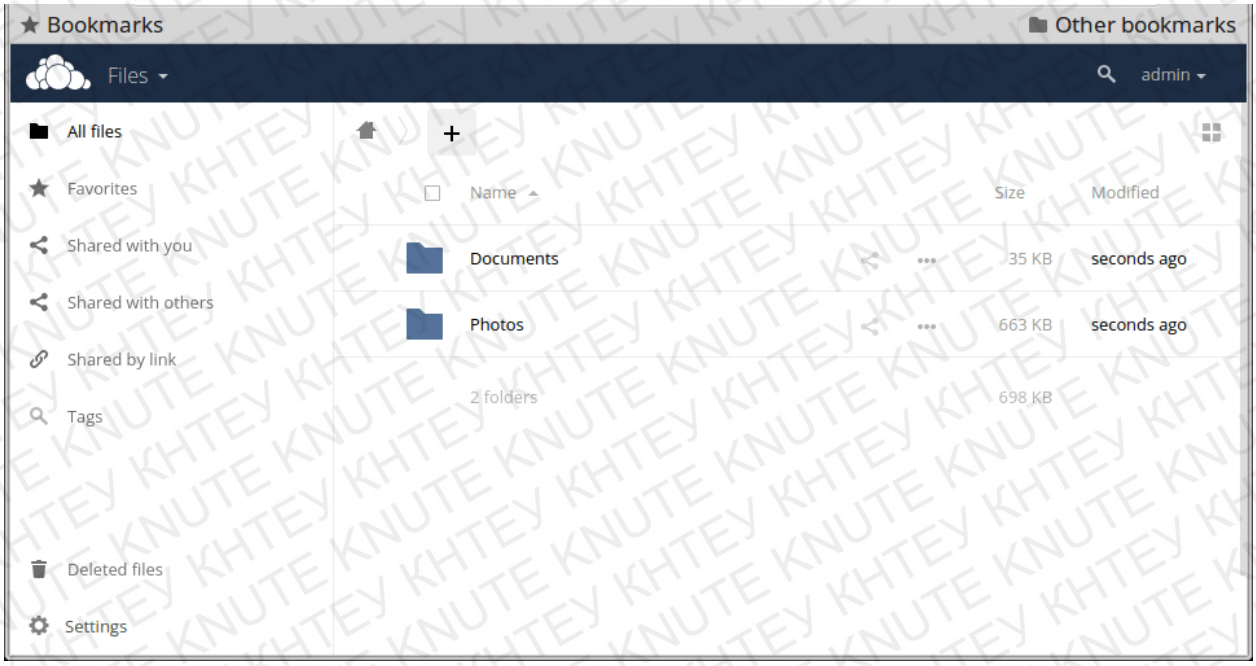


Рис. 3.13. Знімок з екрану Приклад інтерфейсу сервісу документообігу ownCloud

Джерело: створено автором.

Сервіс працює успішно що можна побачити на знімку з екрану 3.13.

3.4. Висновки за розділом 3

Отже мною була створена мережа із чотирьох серверів у хмарному середовищі Amazon Web Services, враховуючи подальшу конфігурацію для кластера Kubernetes.

Із фінансової точки зору – кластер виходить у районі 100 Евро на місяць, якщо розгортати його у клауд провайдері, також він дозволяє гнучко поділяти ядра (12 на сервері, що перетворюються у 40 логічних у кластері).

Кластер був наповнений мінімальним логічним функціоналом для функціонування. Був встановлений сервіс аутентифікації, за допомогою якого можна розподіляти доступ до кластера, а також через неї можливо роздавати доступ до функціоналу інших контейнерів. Була встановлена та швидко налаштована система документообігу, та обміну файлами, а також система моніторингу, що має функції логування, візуалізації та нотифікації. Всі системи швидко розгортаються, а також стійкі до похибок, оскільки налаштовування Kubernetes такі, що при відмові контейнера буде одразу створений новий. Таким чином було продемонстрована гнучкість кластеру мікросервісів, швидкість налаштування нових рішень, та гнучкість розробки логіки мережі.

ВИСНОВКИ ТА ПРОПОЗИЦІЇ

З проведених досліджень можна зробити наступні висновки:

1. Проведено аналіз найпопулярніших провайдерів хмарних послуг, як закордонних та і вітчизняних. У ході порівняння було визначено що рівень розвитку вітчизняних провайдерів набагато відстає від закордонних. Відповідно для виконання роботи вибір пав на найширший по функціоналу та сервісам - Amazon Web Services.

2. В процесі виконання роботи було доведено чим саме була зумовлена причина історичного розвитку віртуалізації, контейнеризації та контейнерних оркестраторів.

3. Після якісного аналізу, у якості сервісу оркестратору був обраний Kubernetes. Цей окрестратор надає найширші можливості із розгортання, конфігурування та побудови відказостійкої системи. У зв'язку з хмарними технологіями - це поєднання і є новизною роботи адже воно є дуже простим у використанні, оптимальним з фінансової сторони та раціональним у використанні ресурсів.

4. В процесі виконання магістерської роботи була розроблена архітектура хмарного середовища, де було встановлено Kubernetes кластер. Сам кластер був наповнений оптимально необхідним функціоналом та повністю підготовлений до розміщення у ньому будь-якого мікросервісного додатку.

					КНТЕУ 121 06з-10.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		30.10.20	*Розробка мікросервісної архітектури хмарного середовища на платформі KUBERNETES*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Десятко А.М.		30.10.20		*ВП*	*58*	*64*
Гарант		Криворучко О.В.		30.10.20		Факультет інформаційних технологій 2м курс, 6з група		
Розробив		Петліченко Д.Г.		30.10.20				
					Висновки та пропозиції			

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cloud Computing : Global (2010 – 2015) [Электронный ресурс] / marketsandmarkets.com. – Б. м., 2010. Режимдоступа: http://www.marketsandmarkets.com/Market-Reports/cloud-computing-234.html (30.10.2011).
2. Монахов Д.Н.,Монахов Н.В.Прончев Г.Б.,Кузьменков Д.А .— Облачные Технологии. Теория и практика книга МАКС Пресс, МГУ, 2013 г. –128 с.
3. Грейс Уокер, "Основы облачных вычислений", Справочник IBM.
4. Pierre Audoin Consultants [Электронныйресурс] - /https://www.pac-online.com/
5. Офіційний сайт компанії Apache CloudStack™ [Электронныйресурс] - /http://cloudstack.apache.org
6. Офіційний сайт компанії Eucalyptus [Электронный ресурс] /https://www.eucalyptus.com/
7. Офіційний сайт компанії [Електронний ресурс] -/http://www.vmware.com/
8. Офіційний сайт компанії Openstack [Електронний ресурс] - /http://www.ubuntu.org/
9. Офіційний сайт компанії Amazon [Електронний ресурс] - /https://www.amazon.com
10. Офіційний сайт компанії Tucha [Электронный ресурс] /http://tucha.ua/

					КНТЕУ 121 06з-10.МР		
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*			
Зав. каф.		Криворучко О.В.		24.01.20	*Розробка мікросервісної архітектури хмарного середовища на платформі KUBERNETES*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Десятко А.М.		24.01.20		*СВД*	*59*	*64*
Гарант		Криворучко О.В.		24.01.20		Факультет інформаційних технологій 2м курс, 6з група		
Розробив		Петліченко Д.Г.		24.01.20				
					Список використаних джерел			

11. Офіційний сайт компанії Volia [Электронный ресурс] - /http://cloud.volia.com/
12. Облачные технологии [Электронный ресурс]. http://www.ixbt.com/cm/cloud-computing.shtml(обращение 22 июня2016).
13. Т.В. Батура, Ф.А. Мурзин, Д.Ф. Семич, ОБЛАЧНЫЕ ТЕХНОЛОГИИ: ОСНОВНЫЕ ПОНЯТИЯ, ЗАДАЧИ И ТЕНДЕНЦИИ РАЗВИТИЯ, 2014 [Электронный ресурс]. -http://swsys-web.ru/cloud-computing-basic-concepts-problems.html(обращение 22 июня 2016).
14. Фундаментальные исследования. – 2015. – № 11 (часть 5) – С. 1048-1053 [Электронный ресурс].-http://www.fundamental-research.ru/ru/article/view?id=39558(обращение 22 июня2016).
15. Мамзелев А.И., Основные принципы концепции облачных вычислений, 2011 [Электронный ресурс]. -http://cyberleninka.ru/article/n/osnovnye-printsipy-kontseptsii-oblachnyh-vychisleniy(обращение 22 июня2016).
16. Облачные вычисления (Cloud computing), 2015 [Электронный ресурс]. - http://www.tadviser.ru/index.php
17. All PTC Mathcad Tutorials [Электронный ресурс]. http://learningexchange.ptc.com/tutorials/by_product/ptcmathcad/product_id:4?langen.
18. Buyya R., Broberg J., Goscinski A., Cloud Computing. Principle sand Paradigms, John Wiley & Sons, Inc., New Jersey, pp.637.
19. Focus Group on Cloud Computing Technical Report, 2012, Part 1:Introduction to the cloud ecosystem: definitions, taxonomies, use cases and high-level requirements, ver.1.0,c.62.
20. Khaled S., Boutaba R., Estimating service response time for elastic cloud applications, Proc. Of the International Conference on Cloud Networking
21. Блог учаснику проекту Mesos [Електронний ресурс] – Режим доступу https://www.quora.com/profile/Florian-Leibert

22. Репозиторії учасника проекту Мезос [Електронний ресурс] - Режим доступу - https://github.com/ FlorianLeibert/
23. Блог із інформацією про кластер кубернетес [Електронний ресурс] - Режим -http://stytex.de/
24. Блог із ілюстартивним прикладом кластеру Кубернетес [Електронний ресурс] - Режим доступу - https://andrewmichaelsmith.com/2016/05/my-kubernetes-setup/
25. Репозиторій блогера, що працює з кубернетес [Електронний ресурс] - Режим доступу -https://github.com/stytexde
26. Матеріал дослідження мережі для кластеру кубернетес, блог компанії Цитрікс [Електронний ресурс] - Режим доступу - https://www.citrix.com/blogs/2017/04/17/netscaler-and-kubernetes-an-enabler-for-digital
27. Матеріал дослідження мережі для кластеру кубернетес, репозиторій компанії Цитрікс [Електронний ресурс] - Режим доступу - https://github.com/citrix/kube-ingress-citrix-netscaler
28. Матеріал дослідження мережі для кластеру кубернетес, приклад мережево налаштування, резпозиторій компанії Кубернетес [Електронний ресурс] – Режим доступу - https://github.com/kubernetes/contrib/tree/master/ingress/controllers/nginx/examples
29. Блог із матеріалами для налаштуваннями кластеру кубернетес [Електронний ресурс] - Режим доступу - https://andrewmichaelsmith.com/2016/05/my-kubernetes-setup/
30. Стаття на feed stream, Матеріал порівняння оркестраторів [Електронний ресурс] - Режим доступу- https://medium.com/@mustwin/a-handy-guide-to-the-mesos-kubernetes-swarm-jungle-ad6bc086c736#.b9s4qvu66

31. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу - https://docs.docker.com/swarm/discovery/
32. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу - https://docs.docker.com/engine/swarm/
33. Матеріал порівняння оркестраторів, офіційний репозиторій проекту Мезос [Електронний ресурс] - Режим доступу - https://mesosphere.github.io/mesos-dns/
34. Матеріал порівняння оркестраторів, документація проекту Кубернетес [Електронний ресурс] - Режим доступу - https://kubernetes.io/docs/user-guide/connecting-applications/
35. Матеріал порівняння оркестраторів, документація проекту Кубернетес [Електронний ресурс] - Режим доступу - https://kubernetes.io/docs/user-guide/services/#environment-variables
36. Матеріал порівняння оркестраторів, блог компанії Докер [Електронний ресурс] - Режим доступу - https://blog.docker.com/2017/02/docker-secrets-management/
37. Матеріал порівняння оркестраторів, документація проекту Кубернетес [Електронний ресурс] - Режим доступу -https://kubernetes.io/docs/user-guide/secrets/
38. Матеріал порівняння оркестраторів, репозиторій проекту Мезос - Режим доступу [Електронний ресурс] - https://github.com/mesosphere/marathon/issues/2295
39. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу - https://docs.docker.com/compose/environment-variables/

40. Матеріал порівняння оркестраторів, документація проекту Кубернетес [Електронний ресурс] - Режим доступу -https://kubernetes.io/docs/user-guide/configmap/
41. Матеріал порівняння оркестраторів, документаціяя проекту Кубернетес [Електронний ресурс] - Режим доступу - https://kubernetes.io/docs/user-guide/environment-guide/
42. Матеріал порівняння оркестраторів, документація проекту Мезос [Електронний ресурс] - Режим доступу - https://mesos.apache.org/documentation/latest/logging/
43. Матеріал порівняння оркестраторів, стаття-диспут моніторнигу кластерів [Електронний ресурс] - Режим доступу - https://technologyconversations.com/2016/11/07/collecting-metrics-and-monitoring-the-cluster/
44. Матеріал порівняння оркестраторів, документація проекту Мезос - Режим доступу [Електронний ресурс] - http://mesos.apache.org/documentation/latest/monitoring/
45. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу -https://docs.docker.com/
46. Матеріал порівняння оркестраторів, документація фреймворку Марафон [Електронний ресурс] - Режим доступу - https://mesosphere.github.io/
47. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу - https://docs.docker.com/swarm/networking/
48. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу - https://docs.docker.com/engine/reference/commandline/stack_deploy/
49. Матеріал порівняння оркестраторів, документація проекту Докер [Електронний ресурс] - Режим доступу -https://docs.docker.com/

50. Офіційний сайт документації Кубернетес [Електронний ресурс] - Режим доступу https://kubernetes.io/docs/user-guide/horizontal-pod-autoscaling/
51. Офіційний сайт компанії Докер [Електронний ресурс] - Режим доступу - Docker.
52. Офіційний сайт проекту Кубернетес [Електронний ресурс] - Режим доступу - Kubernetes.io
53. Офіційний сайт проекту Мезос [Електронний ресурс] - Режим доступу - mesos.apache

ДОДАТКИ

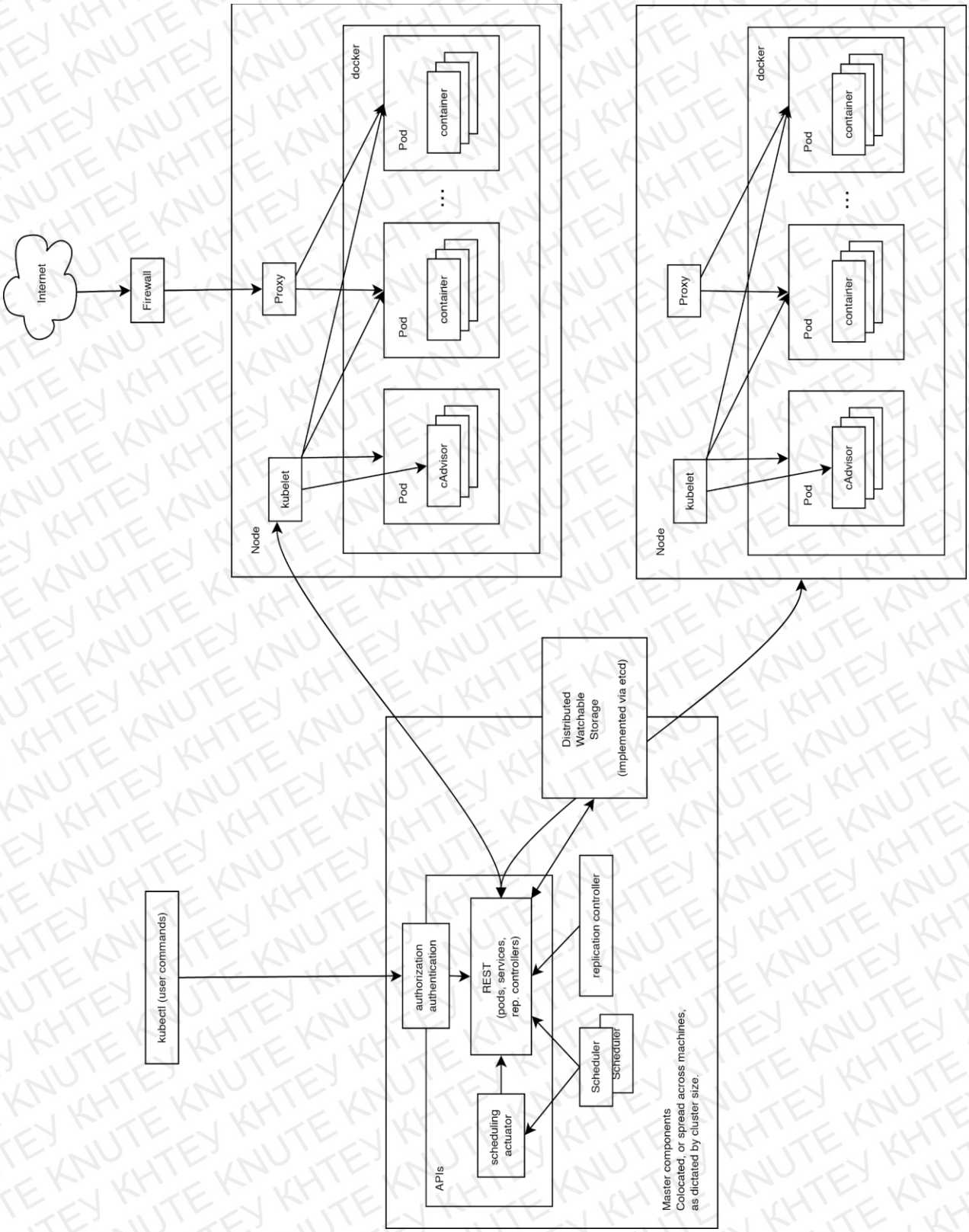
Додаток А

Суть послуги	**Хостинг сайтів і пошти в хмарі**	**Хостинг серверів в хмарі (VPS)**	**Хмарний офіс в коробці**	**Хмарний ЦОД**
Для чого потрібна	Розмістити сайти і пошту на віртуальному сервері за межами України	Розгорнути сервер на надійному хостингу з можливістю виділення ресурсів на вимогу	Отримати готове рішення («хмара під ключ») для невеликої компанії (від 5 до 30 осіб)	Використовувати гнучке, безпечне рішення для створення віртуальних центрів обробки даних (ЦОД) на базі європейської хмари Tucha
Розміщення серверів	Німеччина / Нідерланди	Німеччина / Нідерланди	Німеччина / Нідерланди	Німеччина / Нідерланди
доступність	99,9% (в кластері), 99,0% (в режимі гипервизора)	99,9% (в кластері), 99,0% (в режимі гипервизора)	99,9% (в кластері)	99,9% (в кластері)
Можливість вибору серверної ОС	Платформа LAMP (Linux, Apache, MySQL, PHP)	Linux або Microsoft Windows Server Web Edition	Microsoft Windows Server	Будь-яка, включаючи власну ОС замовника
Можливість оренди ОС	-	+	+	+
Можливість вибору клієнтського ПО	Можливість встановити будь-яку CMS	Повний root-доступ	OpenOffice / MS Office, власне ПО замовника	Будь-яке, включаючи власне ПО замовника
Можливість оренди ПО	-	+	+	+

Продовження дод. А

Купівля додаткового дискового простору	Можливий перехід на інший тарифний план	Можлива (в кластері)	Можлива (в кластері)	Можлива (в кластері)
Базова підтримка	+	+	+	+
розширена підтримка	-	Host Expert	Expert	Expert
Додаткові можливості	Доменні імена, виділені IP-адреси, кластер	IP-адреси, доменні імена, моніторинг роботи сервера, розширена підтримка, дисковий простір, ліцензійне ПО	Розширена підтримка, дисковий простір, ліцензійне ПЗ, моніторинг продуктивності співробітників	IP-адреси, інтернет-канал, розширена підтримка, дисковий простір, ліцензійне ПЗ, моніторинг продуктивності співробітників
Вартість базової конфігурації, євро / міс.	2	від 8	від 7,25 за 1 місце	42,36

Додаток Б


Internet
Firewall
Proxy
kubelet
docker
Pod
container
cAdvisor
Node
Distributed
Watchable
Storage
(implemented via etcd)
kubectl (user commands)
authorization
authentication
REST
(pods, services,
rep. controllers)
replication controller
Scheduler
scheduling
actuator
APIs
Master components
Colocated, or spread across machines,
as dictated by cluster size.

Додаток В

Features	**DockerSwarm**	**Docker Swarm Mode**	**Kubernetes**	**Mesos**
Архітектура Планувальника	Монолітний	Розподілений Стан	Розподілений Стан	Дворівневий
Агностичність	-	-	Docker RKT	Docker, RKT, Інші
Service Discovery	Відсутня, потрібні треті сервіси	Нативна підтримка	Нативна підтримка за допомогою вбудованого DNS чи за допомогою змінних оточення.	Так, за допомогою Mesos-DNS
Secret management	-	Присутня, за допомогою Docker Secret Management	Нативна підтримка	Підтримка можлива, якщо присутня у фреймворку
Configuration management	Через змінні середовища у compose файлі	Через змінні середовища у compose файлі	Нативна підтримка через Config Map чи за допомогою змінних оточення.	Нативно відсутня, підтримка можлива, якщо присутня у фреймворку

Продовження дод. В

Logging	Потребує налаштування драйвера логування, що буде передавати логи на треті сервіси	Потребує налаштування драйвера логування , що буде передавати логи на треті сервіси	Можно тільки відправити логи на інші сервіси	За допомогою Container Logger або ж якщо його надає фреймворк
Monitoring	Потребує треті сервіс для моніторингу усіх контейнерів	Потребує треті сервіси для моніторингу усіх контейнерів	За допомогою Hipsters може проводити базовий моніторинг	Надсилає метрики на моніторинг третіх сервісів
High-Availability	Нативна підтримка створення багатьох менеджерів	Нативна підтримка	Нативна підтримка завдяки реплікації майстрів	Нативна підтримка, із вибором через ZooKeeper
Load balancing	-	Можлива, шляхом балансування портів сервісами балансування	За потреби зовнішній балансувальник створюється перед сервісом	Надається фреймворком, наприклад Marathon
Networking	+	+	Потребує інших сервісів для створення оверлей мережі	Потребує третіх сервісів для мережевої взаємодії

Продовження дод. В

Application definition	Docker Compose файли	Docker Compose стек та файли	Використовує yaml для створення будь-яких об’єктів	Залежить від фреймворку
Deployment	Docker Compose	Підтримує роллбек та апдейт стратегії	Нативна підтримка розгортки, якщо вказана стратегія	Залежить від фреймворку
Auto-scaling	-	але має простий механізм ручного масштабування	Нативна підтримка в рамках, вказаних для пода	Залежить від фреймворку
Self-healing	-	+	+	Залежить від фреймворку
Stateful support	Використання томів даних	Використання томів даних	За допомогою Stateful Sets чи використання томів даних	Використання томів даних
Documentations	Плутана, через переніс Swarm у основний Docker Engine	Плутана, через переніс Swarm у основний Docker Engine	Плутана через переніс документаціїі з github.io на kubernetes.io	Детальна, централізована