МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему:

«Контейнеризований python додаток в кластері Kubernetes»

Завідувач випускаючої кафедри — **Довбиш А.С.**

Керівник роботи — **Проценко О.Б**

Студент групи ІК.м-91 — **Медведєв В.І.**

СУМИ 2020

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *Інформаційно-комунікаційні технології*

Затверджую:
зав. кафедри _____

"_____" _____ 20___р.

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТОВІ

Медведєву Владиславу Ігоровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Контейнеризований python додаток в кластері Kubernetes*
затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи) _____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1) Інформаційний огляд;

2) Постановка задачі;

3) Вибір методів рішення та програмних засобів;

4) Розробка дизайну, та програмна реалізація.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1	*Аналіз проблеми та постановка задачі*		
2	*Проектування схеми додатку*		
3	*Реалізація додатку*		
4	*Оформлення кваліфікаційної магістерської роботи*		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис

РЕФЕРАТ

Записка: 73 стор., 16 рис., 3 додатка, 12 джерел.

Об’єкт дослідження — автоматичне тестування сервісів в кластері Kubernetes.

Мета роботи — розробка додатку для автоматизованого тестування встановленого до кластеру Kubernetes.

Методи дослідження — використана комбінація технологій для створення додатків python, flask, html, css, bootstrap, docker .

Результати — проведений інформаційний огляд, були обрані технології для виконання поставлених задач, сформована схема додатку та виконана програмна реалізація.

КОНТЕЙНЕРИЗАЦІЯ, BOOTSTRAP, DOCKER, KUBERNETES, DOCKER ОБРАЗ, КОНТЕЙНЕР, POD, ВИСОКА ДОСТУПНІСТЬ, ХАОС ІНЖИНІРИНГ

ЗМІСТ

ВСТУП

Тема хмарних обчислень дуже актуальна на даному етапі розвитку комп'ютерної індустрії. У сучасному світі інформація та передача даних є дуже важливим ресурсом і супроводжує нас щодня: на роботі, в навчанні, навіть в простому повсякденному спілкуванні. І то наскільки швидко ми маємо можливість передавати інформацію істотно впливає на наш образ існування. Людство потребує постійного розвитку, і тому так активно користується попитом інтернет і він стає тим важливим полем діяльності для створення чогось нового більш досконалого в плані зручності доступу і швидкості доступу - це найбільш бажані характеристики інтернету - потреба в них і підштовхнула розробників до створення хмарних обчислень.

Осмислення і впровадження хмарних обчислень зараз набирає обертів. Термін «хмара» - це звичайно ж метафора, яку можна розтлумачити, як приховування від користувача за «хмарою» всієї складності інфраструктури хмарного інтернет-сервісу.

За допомогою хмарних обчислень доступ до необхідної користувачеві інформації або технологій стає можливим в будь-який час з будь-якої точки доступу в інтернет з будь-якого комп'ютера, незважаючи на його програмне забезпечення і нічого додатково не встановлюючи на власний комп'ютер. Підрахунок споживання послуг ведеться автоматично і за рахунок автоматизації та масштабності істотна і економія користувача на вартості послуг. Користувачеві немає необхідності модернізувати власну апаратну інфраструктуру, він користується програмним забезпеченням, платформою і інфраструктурою постачальника хмарних послуг. Тоді як постачальник з великими об'єднаними ресурсами і з можливістю перерозподілу навантажень може використовувати менші апаратні ресурси, ніж було б потрібно при виділених апаратних потужностях для кожного окремого користувача.

Самі ж сервіси, які знаходяться в хмарі можуть виконувати безліч функцій: це може бути як інтернет магазин, соціальна мережа чи просто веб-

сайт, так і складна система яка забезпечує телекомунікаційну мережу, електронно-платіжну систему, правову і так далі. І в сучасному світі існує безліч аналогів та подібних рішень, тому в користувача є великий вибір, що обрати. Головними критеріями являється зручність, функціональність та висока доступність, адже будь-який додаток повинен бути доступним для користувачів завжди. Саме через це одним із найважливіших факторів у розробці програмного забезпечення є тестування. Воно може бути як і ручним так і автоматичним. І сама автоматичне тестування має більш вагому пріоритетність адже дозволяє уникнути людського фактору, прискорити швидкість тестування та і дозволити уникнути монотонної роботи людям. Саме це і стало вагомими критеріями при розробці додатків.

1. ІНФОРМАЦІЙНИЙ ОГЛЯД

Якщо підійти з технічного боку, хмарні технології - спосіб організації фізичних та програмних засобів, а також набір інструментів, за допомогою яких користувач отримує обчислювальні потужності, щоб виконувати поставлені перед ним завдання. Також це можна описати як технологію розподіленої обробки цифрових даних, за допомогою яких комп'ютерні ресурси надаються інтернет-користувачеві як онлайн-сервіс. Програми запускаються і видають результати роботи в вікні web-браузера на локальному комп'ютері. При цьому всі необхідні для роботи програми та їх дані знаходяться на віддаленому інтернет-сервері і тимчасово кешуються на клієнтській стороні. Користувач має доступ тільки до власних даних і не може управляти програмною і апаратною інфраструктурою, що лежить в основі системи. Термін "Хмара" використовується як метафора, заснована на зображенні мережі Інтернет на діаграмах комп'ютерних мереж, або як образ складної інфраструктури, яка прихована під спеціальним програмним рівнем.

Найбільш вдалими прикладами хмарних послуг є:

• пошта: gmail, hotmail;

• віддалена робота з документами: Google-документи, Office Web Apps;

• зберігання даних: Google Drive, OneDrive, Dropbox;

• редагування зображень в режимі реального часу: Figma;

• сервіси для створення заміток, спільної роботи над завданнями: Trello, Jira, Evernote;

• онлайн-магазини додатків: Google Play, App Store і Microsoft Store;

• хмарний хостинг - розміщення свого сайту в «хмарі».

У перерахованих сервісів є набір послуг для пересічних користувачів і хмарні рішення для бізнесу. У першому випадку ви отримуєте мінімальний набір функцій, якого вистачить для вирішення повсякденних завдань. Для роботи підприємства потрібен хмарний сервіс для бізнесу, тому що

функціонал там ширше. В принципі Cloud Computing лежать декілька підходів.

Перший - доступність додатків через Інтернет. Підхід не відноситься до закритих інфраструктур, в них Інтернет замінюється локальними мережами, але частина сервісів, все одно доступні з глобальної мережі.

Другий підхід - віртуалізація. Віртуалізація дає простоту та легкість масштабування. Завдяки віртуалізації кожен користувач може отримати стільки потужності, скільки йому необхідно при цьому маючи можливість масштабування в майбутньому.

Третій підхід - Cloud Computing є послугою. У Cloud computing використовується сучасний підхід. Використані ресурси для користувача це набір послуг, які споживаються і, в разі надання комерційним постачальником, оплачуються. Наприклад, хостинг для даних з доступом через HTTP REST API. У користувача є потрібний обсяг даних, який доступний за допомогою зручного інтерфейсу. Дані зберігаються на фізичних серверах, захищені від втрати і територіально розподілені для високої доступності.

Четвертий підхід - простота і стандартизація. Важлива властивість для нової технології. Все, що пропонується всередині хмари доступно через прості виклики API. Величезну популярність здобув протокол REST, за допомогою якого всі операції з даними можна робити через користувача запити. Застосовуються і інші рішення, для різних мов програмування вже доступні бібліотеки для написання подібних систем роботи з даними.

1.1 Принцип роботи та можливості хмарних технологій

Хоча хмара представляється як щось абстрактне, за нею стоїть цілком конкретний набір «заліза», програмного забезпечення і своя архітектура.
Хмарні обчислення будуються на базі серверного та мережевого обладнання. Устаткування об'єднано програмним рішенням і в ньому є призначений для користувача інтерфейс для управління послугою.

Щоб зрозуміти, як працюють хмарні технології, уявіть прохолодне приміщення, де в спеціальних шафах розміщені сервери - потужні комп'ютери з великим запасом пам'яті і дисків для зберігання і обробки інформації. Щоб користувач отримав доступ до цих комп'ютерів, в них встановлено мережеве обладнання - світчі, роутери, комутатори.

Кожна одиниця устаткування може працювати сама по собі. Хмарні системи - це коли всі елементи працюють як єдине ціле, як добре налагоджений механізм. Щоб хмарний сервіс працював саме так, йому потрібен набір спеціального програмного забезпечення, яке буде як диригент керувати всіма процесами.

Кінцевий користувач бачить вже готовий продукт - можливість відкрити сайт і скористатися сервісом: перевірити пошту, встановити додаток на телефон, управляти проектом або отримати доступ до віддаленої бази даних.

Компанії надають хмарні послуги для бізнесу та окремих користувачів у вигляді сервісу. Для зручності види сервісів позначають абревіатурою. Найпоширеніші з них:

• SaaS - Software as a Service, або ПЗ як сервіс;

• PaaS - Platform as a Service, або платформа як сервіс;

• IaaS - Infrastructure as a Service, або інфраструктура як сервіс;

• FaaS - Function as a service, або функція як сервіс.

SaaS - тут ховається Software as a Service, дослівно - програмне забезпечення як сервіс. Клієнт використовує програмне забезпечення провайдера, яке працює в хмарній інфраструктурі. При цьому підході створюються облікові записи клієнта: в пошті, навчальних курсах, інструментах для дизайнерів, в календарі. Завдяки цьому програми доступні з будь-яких пристроїв.

Хоча під SaaS за замовчуванням мають на увазі саме програмний продукт, за цією абревіатурою може стояти Storage-as-a-Service, або

зберігання як сервіс. Хмарні ресурси також використовують для зберігання даних, наприклад, в Google Drive, Dropbox.

Platform as a Service - ви отримуєте комп'ютерну платформу, аналог комп'ютера з операційною системою, яку використовуєте для розгортання(установки) своїх додатків.

PaaS як Process as a Service, або процес як сервіс, - все частіше йде з приставкою Business - використовує хмарні ресурси для управління і автоматизації складних бізнес-процесів.

Infrastructure as a Service - інфраструктура як сервіс - передбачає, що ви отримуєте буквально шматочок хмарної інфраструктури, в якому самі встановлюєте потрібні програми.

Information as a Service, або інформація як сервіс, відкриває доступ до масиву мінливої інформації. Сюди можна віднести біржові котирування, курси валют.

Function as a service - функція як сервіс - дозволяє розробляти, запускати програмні продукти і керувати ними. Основна особливість - запускає певні функції в момент, коли виконується задана умова.

Ще одна відмінність в тому, що з вас будуть знімати не абонплату за місяць, а гроші за обсяг використовуваного дискового простору і кількість операцій в місяць, тобто за активний час користування.

За способом використання хмари виділяють такі види:

• приватне;

• публічне;

• гібридне.

Публічна хмара - це інфраструктура, яка доступна великій кількості компаній і сервісів. Cloud-хостинг по своїй суті і є публічна хмара.

Приватну хмару створюють і підтримують для конкретної організації, яка видає доступ своїм користувачам. Власником платформи буде або сама організація, або постачальник послуг. У випадку з гібридною хмарою

частина ресурсів виділяється для загального користування, а частина «бронюється» для окремого замовника.

Хмарний сервіс - це технологія, яка покликана спростити життя і зробити сервіси доступнішими.

Головні переваги хмарних технологій:

• можливість працювати з особистими акаунтами і даними з будь-якого пристрою;

• не потрібно зберігати інформацію на флешку або інший накопичувач;

• кілька користувачів можуть одночасно редагувати документи і файли;

• хмарні сервіси працюють в браузері, тому не має значення, яка операційна система стоїть на вашому телефоні, планшеті або комп'ютері;

• інформація зберігається на хмарному сервері - навіть якщо ПК або телефон зламається, ви не втратите дані;

• ви користуєтеся найсвіжішою версією програми: постачальник послуги сам стежить за її оновленням;

• можете ділитися інформацією видалено, не пересилаючи великий обсяг даних, наприклад, надати доступ до папки з документами або фотографіями;

• не потрібно купувати потужний комп'ютер для розробки і розгортання програм - користуєтеся можливостями хмари і економите;

• не потрібно бути гуру програмування і адміністрування - хмарні обчислення доступні як людям з досвідом, так і для новачків.

Основні мінуси:

• потрібен стабільний інтернет - без нього не скористаєтеся послугою;

• не будь-який продукт можна налаштувати під свої цілі і завдання;

хоча провайдери надійно захищають хмари, завжди є ризик злому;

• створювати власну хмару дорого, тому малим підприємствам вигідніше користуватися приватною хмарою або навіть публічною, якщо мова йде про приватних підприємців;

• багато сервісів доступні безкоштовно, але не факт, що вони будуть такими завжди. Задумайтесь, чи готові потенційно платити за послугу і скільки.

Поки що переваги хмарних технологій переважують їх недоліки та несуть більше вигоди, ніж ризиків.

Хмарні рішення використовують малі та великі організації в різних сферах. Цілі у них теж різні:

• резервне копіювання даних з подальшим їх відновленням;

• розробка та тестування програм;

• аналіз великих масивів інформації;

• робота з електронною поштою і настройка віддалених робочих столів;

• зберігання додатків для кінцевого користувача.

Компанії по-різному використовують переваги хмари. Розробники відеоігор відкрили для своїх користувачів можливість грати по мережі і спілкуватися між собою. Фінансові компанії відстежують шахрайські схеми в режимі реального часу. Охоронні організації і власники магазинів бачать, що відбувається в торговому залі і оперативно реагують на ситуацію. Але хмара використовує не тільки в цих цілях.

SaaS, PaaS і IaaS, про які ми говорили раніше, різняться тим, наскільки сильно ви можете впливати на роботу хмари. Якщо ви бізнес-користувач і купуєте програмне забезпечення як послугу, SaaS, то можливостей вплинути на її роботу мінімум. Можна налаштувати її під свої потреби, або звернутися до постачальника з проханням додати якийсь функціонал. Коли таких запитів набереться достатня кількість, її впровадять - чи ні. Якість роботи програмного продукту і доступність даних теж залежать від постачальника.

Великий плюс SaaS як послуги для бізнесу - не потрібно розбиратися в технічних деталях або наймати фахівця, який буде підтримувати працездатність програми.

Приклади рішень для бізнесу:

• Microsoft Office 365 і Google Drive;

• хмарні обчислення для складського обліку;

• адміністрування бізнес-процесів;

• збір статистики Google Analytics і платформа для маркетингу.

PaaS - платформа як послуга - підходить для розробників. Купуючи певний обсяг ресурсів - оперативної пам'яті, дискового сховища, центрального процесора - отримуєте свій комп'ютер в хмарі. Тут можете встановлювати, налаштовувати і додавати будь-яке ПЗ.

Платформи надають великі постачальники послуг, завдяки цьому ваш віртуальний комп'ютер буде швидко і стабільно працювати. Плюс в тому, що не обов'язково точно визначати обсяг ресурсів, які знадобляться в майбутньому. Якщо знадобиться розширити масштаби, це легко зробити в хмарі. Багато провайдерів пропонують тарифні плани, де ви оплачуєте тільки обсяг використаних ресурсів: пам'яті, місця на диску, і кількість операцій.

Такий тип послуги підходить найчастіше середньому і великому бізнесу: щоб створити не один віртуальний сервер, а наприклад, групу серверів і запустити серйозний додаток, потрібні великі ресурси, за які доведеться платити. Оскільки компанія сама налаштовує всі складові, потрібно наймати штат фахівців, які будуть цим займатися.

Приклади послуг для бізнесу:

• обчислювальні потужності;

• хостинг додатків;

• бази даних;

• сховища даних.

Найбільш популярні постачальники послуг в цьому напрямку: Amazon Web Services, Windows Azure, Google Cloud Platform.

IaaS - інфраструктура як послуга - означає, що ви орендуєте сервер: виділений фізичний, віртуальний або навіть віртуальний датацентр. Варіант підходить для досвідчених IT-фахівців або компаній, в штаті яких такі є.

Постачальник послуги забезпечує стабільну роботу заліза і програм **віртуалізації**. Ви отримуєте налаштований сервер і доступ до управління, а

також право встановити будь-яку операційну систему, програми та самостійно ними управляти. Якщо сервер перестав підходити, завжди можна змінити його на інший, не переживаючи, що витратилися на обладнання.

1.2 Місце віртуалізації та контейнеризація додатків

Ключове місце в концепції хмарних обчислень займає віртуалізація, оскільки тільки завдяки їй можна побудувати по-справжньому ефективну "хмару", яке матиме наступні характеристики: прийнятна вартість володіння, ефективне управління ресурсами і гарантований рівень обслуговування користувачів.

Почнемо з того, як можна визначити Cloud Computing. Хмарні обчислення зазвичай розуміють як набір апаратних ресурсів або IT-сервісів, що надаються користувачам на вимогу з глобальної (External Cloud) або локальної (Internal Cloud) мережі таким чином, що споживачі IT-ресурсів не замислюються про характер і місце їх походження. В цілому, Cloud Computing характеризується трьома основними трендами сучасних IT.

Три кити Cloud Computing.

По-перше, Utility Computing. Це поняття має на увазі відношення до споживання IT-ресурсів як до отримання електрики з розетки або води з крана. Тобто необхідні апаратні потужності виділяються на вимогу з хмари. Коли потужності не потрібні, вони не споживаються користувачами, а відповідно, гроші за них не стягуються.

По-друге, Software as a Service (SaaS). Це фундаментальне поняття Cloud Computing передбачає зміну моделі доставки програмного забезпечення користувачам таким чином, що компанія припиняє купувати програмне забезпечення як коробкові або замовні продукти, а починає "споживати" його з хмари на основі передплати. Типовий приклад - це електронна пошта Gmail від Google, якою користуються багато компаній, не витрачаючи при цьому на обладнання для поштових серверів, серверне ПЗ і адміністраторів.

По-третє, віртуалізація. Це платформа для хмарних обчислень, яка дозволяє гнучко розділяти і гарантувати ресурси, надавати їх на вимогу і контролювати їх використання. За допомогою віртуалізації можливе створення ізольованих віртуальних оточень користувачів, які реалізують певні IT-сервіси, а також поділ ресурсів (мережа, пристрої зберігання) на логічному рівні для більш гнучкого управління.

Одна з основних особливостей Cloud Computing - це зміна моделі доставки IT-сервісів кінцевим користувачам. Якщо раніше пріоритетом вендорів були продаж якомога більшої кількості IT-продукції в дата-центри замовника, то тепер основна мета - "посадити" замовника на гачок підписки на свої послуги, які доставляються в необхідному йому обсязі на вимогу з дата-центрів провайдерів IT- послуг. А яке ж місце віртуалізації в цьому процесі? Дуже просто: віртуалізація робить Cloud Computing реалізованим на практиці з точки зору технології та економічної ефективності [8].

Віртуалізація - надання набору обчислювальних ресурсів або їх логічного об'єднання, абстраговані від апаратної реалізації, і забезпечує при цьому логічну ізоляцію один від одного обчислювальних процесів, які виконуються на одному фізичному ресурсі. Прикладом використання віртуалізації є можливість запуску декількох операційних систем на одному комп'ютері: при тому кожен з примірників таких гостьових операційних систем працює зі своїм набором логічних ресурсів (процесорних, оперативної пам'яті, пристроїв зберігання), наданням яких із загального пулу, доступного на рівні обладнання, управляє хостова операційна система - гіпервізор.

Якщо говорити про технології, то віртуалізація - це в широкому сенсі відділення уявлення системи від її реалізації. Тобто віртуалізація присутня в будь-якому аспекті IT, включаючи операційні системи, сховища даних, веб-браузери та багато іншого. Однак в концепції Cloud Computing найбільше значення мають віртуалізація серверів, віртуалізація додатків і віртуалізація настільних комп'ютерів як основні інструменти підвищення ефективності доставки IT-послуг корпоративним користувачам.

Всі ці технології дозволяють використовувати віртуальні машини або віртуальні додатки, які "відв'язані" від апаратного забезпечення і конкретної ОС, а значить, є більш мобільними і гнучкими одиницями надання послуг. Для віртуальних машин, на відміну від фізичних, може динамічно виділятися і обмежуватися ємність споживаних ресурсів, забезпечуватися задані параметри рівнів обслуговування (SLA) і, головне: користувач отримає тільки ті ресурси, які йому потрібні в даний момент (і буде за них платити відповідно). Говорячи простими словами, користувач формулює свої потреби на рівні необхідних потужностей, якості послуг і необхідних сервісів - і отримує їх на вимогу від провайдера у вигляді віртуальних машин. А вже на стороні Cloud-провайдера віртуалізація дозволяє динамічно розподіляти віртуальні машини по обладнанню, забезпечувати відмовостійкість сервісів і здійснювати управління і моніторинг з єдиної точки.

Базисом хмарної системи на рівні вузла є гіпервізор - програмне або вбудоване програмне забезпечення, що дозволяє віртуалізувати системні ресурси. Гіпервізор бувають двох типів. Гіпервізор типу 1 працюють безпосередньо на обладнанні системи. Гіпервізор типу 2 працюють поверх базової операційної системи, яка забезпечує служби віртуалізації, такі як підтримка пристроїв введення / виведення і управління пам'яттю. На Рис. 2.1 показані відмінності гіпервізора двох типів [8].

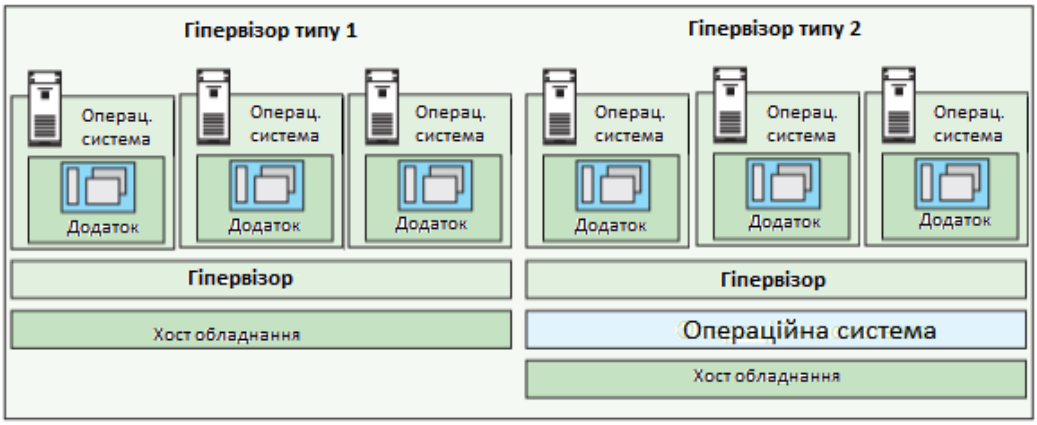

Рисунок 2.1 – Різниця між гіпервізорами двох типів

На відміну від фізичного сервера віртуалізація дозволяє простіше ставитися до питання планування ресурсів, її масштабування більш швидке, гнучке і дешеве. Якщо обчислювальної потужності фізичного сервера не вистачає, то в цьому випадку змінюють або додають певні комплектуючі: жорсткий диск, оперативну пам'ять, процесор і інші компоненти обчислювальної системи. Жорсткий диск можна замінити, а для заміни інших елементів потрібно вимикати обладнання. Це тягне за собою простій сервера, збої в його роботі, невдоволення з боку керівництва і користувачів, так як на розбирання і збирання сервера потрібно чимало часу. Іноді пошук потрібних комплектуючих займає більше одного тижня, протягом якої система працює в режимі неповної потужності, що призводить до збитків у компанії і падіння репутації.

Збільшити продуктивність віртуальної машини (VM) простіше і швидше. Хмарний провайдер постійно надає необхідний запас продуктивності, завдяки чому розширюються можливості віртуального сервера. Усі зміни можна поміняти за пару хвилин. Цей процес здійснюється за допомогою панелі управління без відключення або перезавантаження сервера. Такий підхід дозволяє проводити модернізацію дуже точно, виділяючи строго необхідні ресурси. Необов'язково замінювати процесор, якщо вам будуть потрібні, наприклад, 2 обчислювальних ядра. А для того, щоб збільшити обсяг диска, достатньо додати віртуальній машині необхідний обсяг і не витрачати гроші на дорогий фізичний диск.

Крім того, віртуальні ресурси не вимагають обслуговування з боку клієнта - ремонтом апаратної платформи займається хмарний провайдер. Він надає резервування обчислювальних потужностей, тому якщо сервер, на якому функціонує віртуальна машина, дав збій, то вона в автоматичному режимі перекладається на інший, безперебійно працюючи. Що стосується фізичного обладнання, то його модернізація менш гнучка, тому адміністратори створюють запаси обчислювальних ресурсів, які часто

простоюють. Хмарний провайдер спочатку купує обладнання з надлишковою продуктивністю, щоб розміщувати безліч віртуальних машин, тому клієнту не потрібно створювати запаси - він отримує ресурси на вимогу в будь-який момент.

Коли ми розібралися с питанням віртуалізації, саме час переходити до питання, а як же працюють все сервіси та додатки на віртуальних машинах і в цьому їм допомагає контейнеризація. Контейнеризація - це легка віртуалізація і ізоляція ресурсів на рівні операційної системи, яка дозволяє запускати додаток і необхідний йому мінімум системних бібліотек в повністю стандартизованому контейнері, що з'єднують з хостом або чим-небудь зовнішнім по відношенню до нього за допомогою певних інтерфейсів. Контейнер не залежить від ресурсів або архітектури хоста, на якому він працює. Всі компоненти, необхідні для запуску програми, упаковуються як один образ і можуть бути використані повторно. Додаток в контейнері працює в ізольованому середовищі і не використовує пам'ять, процесор або диск хостової операційної системи. Це гарантує ізольованість процесів всередині контейнера.

Існує декілька основних варіантів віртуалізаії, а саме два підходи за допомогою яких ми маємо змогу створювати незалежні простори, які в свою чергу будуть ізольовані один від одного на одному фізичному сервері. Першим є створення віртуальних машин, для роботи яких потрібен гіпервізор, а другим підходом є створення віртуальних контейнерів. У випадку з віртуальними машинами для кожної використовується власна гостьова ОС, а у випадку з віртуальними контейнерами застосовується ядро хостової ОС. Завдяки цьому, перший підхід дозволяє створювати неоднорідні середовища для обчислень на одному комп'ютері, другий підхід - лише однорідні.

Але оскільки віртуальні машини включають в себе операційну систему, їх розмір буває занадто великим і може досягати кілька гігабайт. Ще одним недоліком віртуальних машин можна вважати те, що для завантаження

операційної системи і встановлення програми, яке в ній розміщено, потрібно доволі багато часу в порівнянні з контейнерами. В свою чергу, контейнери більш легкі і, в більшості випадків, їх розмір вимірюється в мегабайтах. В порівнянні продуктивності контейнерів з віртуальними машинами, контейнери можна запускати майже миттєво. Тому при виборі між контейнерами та віртуальними машинами слід враховувати цілі, які намагаєтеся досягти.

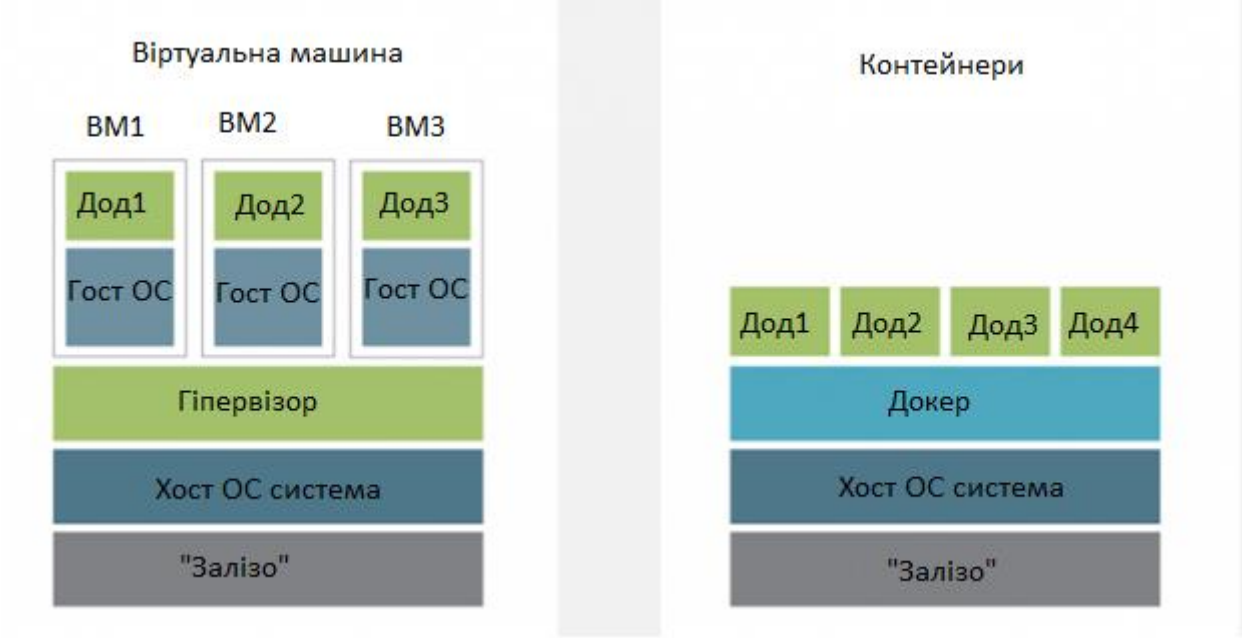

Рисунок 3.1 - Різниця між віртуальною машиною та контейнером

Багато проблем виникає при зміні середовища де запускаються додатки. Це може виникнути, наприклад, коли розробник розробляє і запускає код в тестовому середовищі. Зміни можуть бути присутні не тільки з обчислювальними ресурсами, але також і з мережами. Політики безпеки, топології мережі, і багато іншого може відрізнятися. Саме завдяки контейнерам можна досягти необхідної ізоляції.

Переваги контейнерів:

- Гнучке середовище. Основна перевага у використанні контейнерів полягає в їх швидкості створення, на відміну екземплярів віртуальних машин.

- Підвищена продуктивність. Швидкість розробки досягається за рахунок того, що контейнери усувають мережеві залежності й конфлікти. Кожен контейнер може бути незалежно оновлений без будь-яких проблем тому їх можна розглядатися як окремий мікросервіс.
- Версіонування. Управління версіями дозволяє стежити за відмінностями.
- Переносимість. Контейнери інкапсулюють всі необхідні залежності, такі як бібліотеки, необхідні для запуску програми. Це дозволяє переносити контейнера з одного середовища в інше.
- Стандартизація. Контейнери засновані на відкритих стандартах і можуть працювати в основних дистрибутивах Linux, Microsoft.
- Безпека. Контейнери ізолюють процеси один від іншого і від базової операційної системи. Таким чином, будь-яке оновлення не впливає на роботу іншого контейнера.

Недоліки контейнерів:

- Підвищена складність: при великій кількості контейнерів, які працюють з додатком, збільшується складність. Управління безліччю контейнерів досить складне завданням у виробничому середовищі. Такі інструменти, як Kubernetes і Mesos, полегшують управління великою кількістю працюючих контейнерів.
- Також до мінусів можна віднести складність, яка полягає в тому, що зазвичай в контейнер додається більше ресурсів, ніж потрібно - це призводить до розростання образу і великому розмірі контейнера.
- Підтримка Native Linux: основна кількість контейнерних технологій, таких як Docker, основані на Linux-контейнерах. Тому виконання цих контейнерів в середовищі Microsoft – досить складний процес, а їх щоденне використання викликає складності в порівнянні з початковим запуском цих додатків на Linux.
- Незрілість: контейнери - доволі нова технологія на ринку.

Віртуалізація на рівні операційної системи - це спосіб віртуалізації, при якому операційна система допускає роботу декількох ізольованих контейнерів, а не тільки одного. Вони виглядають як справжні комп'ютери з точки зору програм, запущених в них. Підхід корисний, коли необхідно налаштувати декілька операційних систем з ідентичними параметрами. Різні програми можуть бути встановлені і працювати ідентично, як якщо ми запускаємо додаток на операційній системі хоста. Ресурси, призначені контейнеру, доступні лише йому. Для створення контейнерів операційної системи ми можемо використовувати такі контейнерні технології, як VServer, Jails ,LXC, OpenVZ, Solaris , Linux, BSD.

Контейнери додатків: віртуалізація додатків - це програмна технологія, яка інкапсулює комп'ютерні програми з базової операційної системи, на якій вона виконана. Повністю віртуалізований додаток не встановлюється в традиційному сенсі, хоча він все одно виконується так, як якщо б він був встановлений. Додаток поводиться під час виконання, так як ніби він безпосередньо взаємодіє з вихідної операційною системою і всіма ресурсами, якими він управляє, але може бути ізольованим в різному ступені.

Контейнери додатків призначені для упаковки і запуску служб як одного процесу, тоді як в контейнерах ОС можуть виконуватися кілька сервісів і процесів. Контейнерні технології, такі як Docker і Rocket, є прикладами контейнерів для додатків.

1.3 Оркестрація контейнерів та порівняння оркестраторів

Контейнерні платформи, такі як Docker, зараз є дуже популярними для упаковки додатків, які базуються на мікросервісній архітектурі. 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 [10].

Існують різні платформи для оркестрації контейнерів. Вони дозволяють реалізувати зручні та ефективні засоби розгортання контейнерних систем, побудови єдиної централізованої консолі для застосування політик управління. Давайте розглянемо найбільш популярні та розберемося в їх перевагах та недоліках.

Найбільш популярним є **Kubernetes** - платформа з відкритим вихідним кодом, спочатку розроблена Google і в даний час підтримувана Cloud Native Computing Foundation. Kubernetes підтримує як декларативну конфігурацію, так і автоматизацію. Це може допомогти автоматизувати розгортання, масштабування і управління контейнерної робочим навантаженням і послугами. API Kubernetes допомагає встановити зв'язок між користувачами, компонентами кластеру і зовнішніми компонентами сторонніх виробників. Рівень управління Kubernetes і самі вузли виконуються на групі вузлів, які разом утворюють кластер. Робоче навантаження додатка складається з одного або декількох модулів, які виконуються на вузлі (вузлах) Worker. Рівень управління контролює групи контейнерів (Pod-и) і робочі вузли. До особливостей можна віднести :

- служби виявлення і балансування навантаження;
- оркестрації системи зберігання даних;
- автоматизовані розгортання і відкати;
- горизонтальне масштабування;
- управління секретом і конфігурацією;
- самовідновлення;
- пакетне виконання;
- подвійний стек IPv4 / IPv6.

Наступним по полярності є **OpenShift**.

Redhat пропонує OpenShift Container Platform як сервіс (PaaS). Він допомагає автоматизувати додатка на безпечних і масштабованих ресурсах в гібридних хмарних середовищах. Він надає платформи корпоративного рівня для створення, розгортання та управління контейнерними додатками. Сервіс побудований на основі Redhat Enterprise Linux і Kubernetes. Openshift має різні функціональні можливості для управління кластерами через інтерфейс користувача і інтерфейс командного терміналу.

Nomad - це зручний, гнучкий і простий у використанні оркестратор робочого навантаження для розгортання контейнерів і неконтейнерних додатків і управління ними не залежно від того розташовані вони в хмарному або в локальному середовищі. Nomad працює як єдиний двійковий файл з невеликим ресурсом (35MB) і підтримується в macOS, Windows, Linux. Розробники використовують декларативну інфраструктуру як код (IaC) для розгортання своїх додатків і визначають спосіб розгортання програми. Nomad автоматично відновлює додатки після збоїв. Nomad підходить для оркестрації будь-якого типу додатків (не тільки контейнери). Він забезпечує першокласну підтримку Docker, Windows, Java, віртуальних машин і багато чого іншого.

Переваги:

- простота і надійність;
- модернізація застарілих додатків без перезапису;
- перевірена масштабованість;
- підтримка роботи з декількома хмарами;
- вбудована інтеграція з Terraform, Consul і Vault.

GKE надає повністю кероване рішення для оркестрації контейнерних додатків на Google Cloud Platform. Кластери GKE створені на основі Kubernetes. Ви можете взаємодіяти з кластерами за допомогою Kubernetes CLI. Команди Kubernetes можна використовувати для розгортання додатків і управління ними, виконання завдань адміністрування, установки політик і

моніторингу працездатності розгорнутих робочих навантажень. Розширені функції управління Google Cloud також доступні з кластерами GKE, такими як балансування навантаження Google Cloud, пули вузлів, автоматичне масштабування вузлів, автоматичне оновлення, автоматичне відновлення вузлів, ведення журналу та моніторинг за допомогою операційного пакету Google Cloud. Google Cloud надає інструменти CI / CD, що допомагають створювати і обслуговувати контейнери додатків. Cloud Build можна використовувати для створення образів контейнерів (наприклад, Docker) з різних репозиторіїв вихідного коду, а Container Registry - для зберігання образів контейнерів. GKE - готове для підприємства рішення з попередньо розробленими шаблонами розгортання.

AWS EKS - повністю керований сервіс Kubernetes. AWS дозволяє запускати кластер EKS за допомогою AWS Fargate, який є безсерверною потужністю для контейнерів. Fargate усуває необхідність у виділенні ресурсів і управлінні серверами, дозволяючи платити за ресурс як додаток. AWS дозволяє використовувати додаткові функції EKS, такі як Amazon CloudWatch, Amazon Virtual Private Cloud (VPC), AWS Identity, групи автоматичного масштабування і управління доступом (IAM), додатки моніторингу, масштабування і балансування навантаження. EKS інтегрується з сіткою AWS App та пропонує власний досвід Kubernetes. EKS працює під управлінням останнього Kubernetes і сертифікований як сумісний з Kubernetes.

1.4 Тестування хмарних сервісів

Хмарне тестування - це форма тестування програмного забезпечення, при якій веб-додатки використовують середу хмарних обчислень («хмара») для імітації реального трафіку користувача. З технічної точки зору тестування хмарних сервісів або хмарних додатків схоже на тестування традиційного програмного забезпечення, але є і своя специфіка - крім таких видів тестування, як модульне, інтеграційне, функціональне і навантажувальне, для

хмарних проектів додаються ще два етапи тестування: локальне і тестування в реальному хмарному середовищі [5]. Набір тестів на кожному етапі може бути різним: модульні тести можуть проводитися локально, тоді як тестування навантаження рекомендується проводити на реальному експлуатаційному середовищі. При тестуванні хмарних додатків можуть виникнути деякі складності в порівнянні зі звичайним програмним забезпеченням:

- часті оновлення та релізи залишають мало часу на перевірку правильності і безпеки додатків;
- часом компоненти бекенда, пов'язані з призначеним для користувача інтерфейсом додатку, залишаються без перевірки;
- забезпечення безпеки даних;
- важливо визначити найдоступніші зони і провести призначене для користувача тестування за участю якомога більшої кількості людей з різних місць;
- під час інтеграції і перенесення SaaS-додатків нерідко виникають складнощі зі збереженням даних тестування;
- для нових релізів перевіряються всі аспекти, які стосуються ліцензування, в т.ч. кількість користувачів і функціональність програми;
- відсутність стандартизації додатків.

До основних видів тестування можна віднести наступні.

Модульне тестування дозволяє перевірити на коректність окремі частини програми, ізолюючи їх від всієї програми. З точки зору тестування хмарних сервісів основна увага при модульному тестуванні слід зосередити на поведінці конкретного об'єкта класу, а не на взаємодії цього об'єкта з іншими компонентами програми. Наприклад, для Windows Azure будь-який тест, що залежить від сховища Windows Azure, вимагає складної настройки і додаткової логіки для забезпечення доступності коректних даних при виконанні тестування. Внаслідок цього розробники спеціально для

тестування часто створюють спеціальний модуль доступу до даних. Зокрема, цей засіб дозволяє виконувати тестування класів зберігання даних незалежно від хмарного сховища Windows Azure [4].

Інтеграційне тестування призначене для перевірки взаємодії та інтерфейсів між компонентами, підсистемами. Після того як розроблені основні компоненти хмарного додатку, необхідно створити інтеграційні тести. Якщо у додатка немає web-інтерфейсу, то інтеграційні тести можуть бути і функціональними. Для розробки інтеграційних тестів можуть бути використані різні інструменти. Треба відзначити, що для проведення інтегрального тестування необхідно, щоб всі модулі програми вже були розгорнуті в локальному або хмарному середовищі. Тому для проведення таких тестів важлива автоматизація збірки і публікація рішення.

В ході функціонального тестування визначається, чи вирішує додаток завдання, для яких воно було розроблено, чи відповідає вимогам і очікуванням користувачів. Зазвичай функціональне тестування включає також тестування користувальницького інтерфейсу.

Основна мета навантажувального тестування полягає в тому, щоб, створивши очікуване в системі навантаження (наприклад, за допомогою імітації роботи віртуальних користувачів), переконатися в коректному функціонуванні системи. Динамічна природа хмарних додатків ускладнює створення реалістичних навантажувальних тестів і виділення необхідної для тестування кількості обчислювальних ресурсів. У підсумку результати навантажувального тестування для додатків часто бувають некоректними або переоцінені. Ще однією відмінною рисою багатьох хмарних додатків є георозподіленість їх користувачів, а отже, при тестуванні необхідно враховувати фактор, що швидкість підключення і звернення реальних користувачів до додатку може сильно варіюватися.

Фактично тестування навантаження - це єдиний спосіб, що дозволяє побачити, чи приводить невисока швидкість підключення великої кількості користувачів до споживання додатком більшої кількості ресурсів. А це, в

свою чергу, може зменшити число одночасно працюючих користувачів, на яке розрахований хмарний додаток. Крім цього, тестування навантаження допомагає з'ясувати, наскільки оптимальне правило розбиття сховища на розділи бази даних.

Chaos Engineering - це дисципліна проведення експериментів з системою з метою створення впевненості в її здатності протистояти турбулентним умов в процесі виробництва. Досягнення в області великомасштабних розподілених програмних систем змінюють правила гри в програмну інженерію. Як галузь, швидко застосовуються методи, що підвищують гнучкість розробки і швидкість розгортання. Навіть коли всі окремі служби в розподіленій системі працюють належним чином, взаємодія між цими службами може привести до непередбачуваних результатів. Непередбачувані результати, які ускладнюються рідкісними, але руйнівними подіями реального світу, що впливають на виробничу середу, роблять ці розподілені системи хаотичними.

Нам необхідно виявити слабкі місця, перш ніж вони проявляться в користувачів. Системні недоліки можуть проявлятися в наступних формах: неправильні настройки відкату при недоступності послуги; повторюваність запитів з неправильно налаштованих тайм-аутами; простої, коли система отримує занадто багато трафіку; каскадні відмови при виході з ладу єдиної точки відмови. Ми повинні активно усувати найбільш суттєві недоліки, перш ніж вони вплинуть на клієнтів в процесі виробництва. Потрібен спосіб впоратися з хаосом, властивий цим системам, скористатися перевагами збільшення гнучкості і швидкості і бути впевненими в успішному встановленні, незважаючи на всю складність, яку додатки представляють.

Емпіричний, системний підхід усуває хаос в розподілених системах в масштабі і зміцнює впевненість у здатності цих систем протистояти реальним умовам. Дізнаємося про поведінку розподіленої системи, спостерігаючи за нею під час керованого експерименту. Це називається Chaos Engineering.

Щоб вирішити проблему невизначеності розподілених систем в масштабі, Chaos Engineering можна розглядати як засіб сприяння експериментів по виявленню системних слабких місць. Ці експерименти складаються з чотирьох етапів:

1. Почніть з визначення «стійкого стану» як деякого вимірюваного виходу системи, який вказує на нормальну поведінку.

2. Припустімо, що це стійкий стан збережеться як в контрольній групі, так і в експериментальній групі.

3. Введіть змінні, які відображають реальні події в світі, такі як збої серверів, несправності жорстких дисків, розірвані мережеві з'єднання.

4. Спробуйте спростувати гіпотезу, шукаючи різницю в стійкому стані між контрольною групою і експериментальної групою.

Чим складніше порушити стійкий стан, тим більше ми впевнені в поведінці системи. Якщо слабке місце виявлено, тепер є мета для вирішення, перш ніж це поведінка проявиться в системі в цілому.

1.5 Постановка задачі

Основною задачею даної роботи є створення python додатку, який буде встановлений в кластер Kubernetes. Сам додаток має виконувати роль автоматизованого менеджера для тестування додатків використовуючи підхід хаотичного видалення працюючих компонентів сервісу при цьому емулюючи роботу в хмарі. Він буде корисним для інженерів з програмного забезпечення для автоматизованого тестування своїх додатків, матиме досить широкий спектр можливостей включаючи вибір режиму тестування для емуляції різного виду проблем, які можуть виникати під час роботи. Головними критеріями для додатку є:

- простота встановлення та використання;
- можливість вибору режиму роботи;
- запуск роботи за графіком у встановлений час;
- наявність звіту тестування для подальшого аналізу.

Для досягнення поставлених цілей в обов’язкову порядку необхідно провести аналіз предметної області, створення моделі додатку, вибір найбільш підходящих технологій для реалізації та тестування готового продукту.

2. ВИБІР ТЕХНОЛОГІЙ РЕАЛІЗАЦІЇ

Вибір технологій реалізації завжди є дуже відповідальною і складною справою, адже саме від цього вибору залежить наскільки продукт вийде якісним і як багато часу та сил розробники витратять на його створення.

2.1 Технології Kubernetes для оркестрації додатків

Kubernetes є проектом з відкритим вихідним кодом, призначеним для управління кластером контейнерів Linux як єдиною системою. Kubernetes управляє і запускає контейнери Docker на великій кількості хостів, 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 [10].

Компанія Google користується контейнерною технологією вже більше десяти років. Вона починала з запуску більш 2 млрд контейнерів протягом одного тижня. За допомогою проекту Kubernetes компанія ділиться своїм досвідом створення відкритої платформи, призначеної для масштабованого запуску контейнерів.

Проект переслідує дві мети. Якщо ви користуєтеся контейнерами Docker, виникає наступне питання про те, як масштабувати і запускати контейнери відразу на великій кількості хостів Docker, а також як виконувати їх балансування. У проекті пропонується високорівнева API, що визначає логічне групування контейнерів, що дозволяє визначати пули контейнерів, балансувати навантаження, а також задавати їх розміщення. 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 оброблялося системою? Ось тут Kubernetes приходить на допомогу! Kubernetes надає вам фреймворк для гнучкої роботи розподілених систем. Він займається масштабуванням і обробкою помилок в додатку,

надає шаблони розгортання і багато іншого. Наприклад, Kubernetes може легко управляти розгортанням вашої системи. Kubernetes надає вам:

- Моніторинг сервісів і розподіл навантаження. Kubernetes може виявити контейнер, використовуючи ім'я DNS або власний IP-адреса. Якщо навантаження трафіку в контейнері високе, Kubernetes може збалансувати навантаження і розподілити мережевий трафік між подібними, щоб робота додатку була більш стабільною.
- Оркестрації сховища. Kubernetes дозволяє вам автоматично змонтувати систему зберігання за вашим вибором, таку як локальне сховище чи загальнодоступну хмару 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. Ви надаєте Kubernetes кластер вузлів, який він може використовувати для запуску контейнерних завдань. Ви вказуєте Kubernetes, скільки ЦП і пам'яті потрібно для кожного контейнера. Kubernetes може розмістити контейнери на ваших вузлах так, щоб найбільш ефективно використовувати ресурси.
- Самоконтроль. Kubernetes перезапускає проблемні контейнери, замінює і завершує роботу контейнерів, які не проходять певну користувачем 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, такий 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 стеку.

2.2 Технологія контейнеризації Docker

Docker (Докер) - програмне забезпечення з відкритим вихідним кодом, що застосовується для розробки, тестування, доставки і запуску веб-додатків в середовищах з підтримкою контейнеризації. Він потрібен для більш ефективного використання системи і ресурсів, швидкого розгортання готових програмних продуктів, а також для їх масштабування і перенесення в інші середовища з гарантованим збереженням стабільної роботи. Розробка Docker була розпочата в 2008 році, а в 2013 році він був опублікований як вільно доступний під ліцензією Apache 2.0. В якості тестового додатка Docker був включений в дистрибутив Red Hat Enterprise Linux 6.5. У 2017 році була випущена комерційна версія Docker з розширеними можливостями. Docker працює в Linux, ядро яких підтримує cgroups, а також ізоляцію простору імен. Для інсталяції та використання на платформах, відмінних від Linux, існують спеціальні утиліти Kitematic або Docker Machine [3].

Основний принцип роботи Docker - контейнеризація додатків. Цей тип віртуалізації дозволяє упаковувати програмне забезпечення по ізольованим середах - контейнерів. Кожен з цих віртуальних блоків містить всі необхідні елементи для роботи програми. Це дає можливість одночасного запуску великої кількості контейнерів на одному хості. Docker-контейнери працюють в різних середовищах: локальному центрі обробки інформації, хмарі, персональних комп'ютерах [2].

Переваги використання Docker.

- Мінімальне споживання ресурсів - контейнери не віртуалізують всю операційну систему (ОС), а використовують ядро хоста і ізолюють програму на рівні процесу. Останній споживає набагато менше ресурсів локального комп'ютера, ніж віртуальна машина.

- Швидкісне розгортання - допоміжні компоненти можна не встановлювати, а використовувати вже готові docker-образи (шаблони). Наприклад, не має сенсу постійно встановлювати і налаштовувати Linux Ubuntu. Досить 1 раз її інсталювати, створити образ і постійно використовувати, лише оновлюючи версію при необхідності.
- Зручне приховування процесів - для кожного контейнера можна використовувати різні методи обробки даних, приховуючи фонові процеси.
- Робота з небезпечним кодом - технологія ізоляції контейнерів дозволяє запускати будь-який код без шкоди для ОС.
- Просте масштабування - будь-який проект можна розширити, запровадивши нові контейнери.
- Зручний запуск - додаток, що знаходиться всередині контейнера, можна запустити на будь-якому docker-хості.
- Оптимізація файлової системи - образ складається з шарів, які дозволяють дуже ефективно використовувати файлову систему.

2.3 Python фреймворк Flask

Flask - це невеликий і легкий веб-фреймворк, написаний на мові Python, що пропонує корисні інструменти і функції для полегшення процесу створення веб-додатків з використанням Python. Він забезпечує гнучкість і є більш доступним фреймворком для нових розробників, так як дозволяє створити веб-додаток швидко, використовуючи тільки один файл Python. Flask - це розширювана система, яка не зобов'язує використовувати конкретну структуру директорій і не вимагає складного шаблонного коду перед початком використання [1]. Flask використовує механізм шаблонів Jinja для динамічного створення HTML-сторінок з використанням знайомих понять в Python, таких як змінні, цикли, списки. А також на всіх веб-сторінках можна використовувати CSS, Bootstrap, JavaScript щоб зробити

додаток візуально привабливим та надати можливість використовувати на мобільних браузерах.

До переваг можна віднести:

1. Flask вважається кращим веб-фреймворком для створення легких веб-додатків і невеликих статичних сайтів.
2. Легкий для розуміння, зрозуміти, як працювати з фреймворком, зможе навіть початківець програміст. Flask має просту структуру і інтуїтивно зрозумілий синтаксис. Крім того, на відміну від інших фреймворків Flask дозволяє програмісту повністю контролювати процес розробки.
3. Гнучкий, лише деякі частини фреймворка недоступні для модифікацій, тому що вони вже максимально прості і оптимізовані. Програміст може самостійно редагувати більшу частину інструментів фреймворка під свої потреби.
4. Разом з Flask програміст отримує шаблон, який дозволяє використовувати один і той же інтерфейс користувача на декількох сторінках.
5. Хороші інструменти для тестування, у Flask інтегровані інструменти для тестування і налагодження. Програміст має в розпорядженні повноцінні unit тести, вбудований сервер розробки, відладчик і обробник запитів.

Але також можна виокремити і деякі мінуси даного фреймворку.

1. Не підтримка асинхронність. Flask обробляє всі запити в один потік, тобто кожен запит блокує потік на час свого виконання, а потім передає в чергу наступний запит. Якщо провести аналогію, то це як лише одна працююча каса в супермаркеті, при великому кількість покупців (запитів) каса не справлятиметься і утворюється велика черга.
2. Недолік можливостей, якщо говорити про розробку великих веб-додатків, то у Flask просто не вистачає можливостей.

2.4 Технологія Chaos Engineering

Chaos Engineering - це певний підхід до «навмисного пошкодження речей», метою якого є дізнатися щось нове про систему, роблячи експерименти над нею. Безумовно, наша мета полягає в необхідності виявити приховані проблеми, які можуть виникнути у виробництві. Тільки тоді ми зможемо реагувати на слабкі сторони системи і робити її стійкою до помилок. Chaos Engineering виходить за рамки традиційного тестування (помилки) в тому, що він не тільки підтверджує припущення. Він також допомагає нам в дослідженні багатьох непередбачуваних речей, які могли б статися, і відкрити нові властивості наших спочатку хаотичних систем. Хоча згадані концепції не є новими, першим, хто спочатку легалізував Chaos Engineering як дисципліну, став Netflix. (Netflix, як виявилося, - щось більше, ніж сервіс відео трансляції. Компанія, що стоїть за сервісом - родоначальник в сфері автоматизованого тестування помилок, який розробив такі утиліти, як відомий Chaos Monkey [5].

У принципах Chaos Engineering Netflix дало дисципліні наступне визначення: Chaos Engineering - дисципліна, експериментує на розподіленій системі, з метою створити впевненість в здатності системи витримати турбулентні умови у виробництві. Як Chaos-інженер ви тестуєте здатність системи справлятися з подіями реального світу - помилки сервера, стрибки трафіку, пошкоджені повідомлення і т.п. - в серії контрольованих експериментів. У відповідність з принципами ці «хаотичні» експерименти зазвичай включають в себе 4 ступені: визначення нормальної поведінки системи - її стабільний стан – оснований на початку перед початком тестування, такому як загальна пропускна здатність, частота виникнення помилок. Побудова діаграми, яка буде відображати різницю між станом до початку тестування і після (з практичної точки зору обидві групи можуть містити одну і ту ж систему, але в різний проміжок часу, тобто до або під час експерименту). Запропонуйте експериментальній групі моделювати події реального світу, такі як збої сервера, неправильні відповіді сервера або

сплески трафіку. Порівняйте дані початку и кінця. Чим менша різниця, тим ми можемо бути більш впевнені в тому, що наша система стійка. Не варто говорити, що рішення будь-яких проблем, виявлених у такий спосіб - хороша ідея. Сенс знаходження потенційних катастрофічних подій знову і знову полягає в необхідності зробити їх неважливими для нашої інфраструктури і її операторів.

В якості базового прикладу, скажімо, що ви хочете знати, що відбувається, якщо з якоїсь причини ваша база даних MySQL недоступна. Навмисне введення в систему помилок так, що окремі частки вашої інфраструктури стають недоступними, є відмінним способом дізнатися, як система взаємодіє. Обережність: ви ніколи не повинні проводити хаос-експеримент на готовому продукті в замовника, якщо ви напевно знаєте, що він завдасть серйозної шкоди, ймовірно при цьому впливаючи на замовників - а разом з ними і на вашу репутацію. Насамперед необхідно виправити відомі проблеми. Chaos Engineering вимагає початковий рівень стійкості. Повертаючись до експерименту. Ви припускаєте, що при відсутності бази даних, ваш веб-додаток перестане опрацьовувати запитами, повертаючи помилку. Щоб зімітувати подію, вам потрібно заблокувати доступ до сервера бази даних. Існує кілька способів досягти цього. Наприклад, ви могли б додати правило iptable для перенесення трафіку на порт 3306 або відповідним чином змінити групи безпеки хмарного провайдера. Однак під час експерименту щось несподіване трапляється: здається, що веб-додаток відповідає постійно. Ваші системні метрики підтверджують підозру, що щось не так. Після деякого дослідження, ви знайдете причину - невірно налаштований тайм-аут клієнта - і виправите її за лічені секунди.

Отже, початкові гіпотези виявилися неправильними. На щастя, це хороша проблема, тому що тільки що виявили нову здатність системи - недолік, насправді - про який не знали до експерименту! Дешевше вирішити ці проблеми зараз, ніж чекати, поки вони виявляться в реальному робочому процесі.

3. ПРОГРАМНА РЕАЛІЗАЦІЯ

Для реалізації поставленої задачі ми використовуємо велику кількість технологій та інструментів. Та перш за все потрібно визначитися з середою розробки та планом виконання. Оскільки наш додаток повинен бути високо доступним, а також його призначення тестування мікросервісів в кластері Kubernetes потрібно підготувати інфраструктуру. Для мінімального кластеру потрібна хоча б одна віртуальна машина, у вирішені цього питання допоможе Oracle VM VirtualBox.

3.1 Віртуальна машина в Oracle VM VirtualBox

VirtualBox — це програма віртуалізації для операційних систем, створена німецькою фірмою innotek, яка зараз належить до Oracle Corporation. Вона встановлюється на поточну операційну систему, яка називається хостовою, усередину даної програми встановлюється інша операційна система, яку називають гостьовою операційною системою. Підтримується основними операційними системами Linux, FreeBSD, Mac OS X, OS/2 Warp, Microsoft Windows, які підтримують роботу гостьових операційних систем FreeBSD, Linux, OpenBSD, OS/2 Warp, Windows і Solaris. Починаючи з 2007 року, за спостереженнями DesktopLinux.com, VirtualBox займає третє місце за популярністю серед програмних засобів, які дозволяють запуск Windows-програм на базі Linux.

Перш за все потрібно скачати образ та встановити на локальний комп’ютер. Після успішного встановлення створюємо гостьову машину для базі образу Ubuntu. Оскільки дана віртуальна машина створюється для запуску і роботи менеджера додатків Kubernetes вона повинна підходити під мінімальні реквізити по оперативній пам’яті, потужності процесора та виділеній кількості диску.

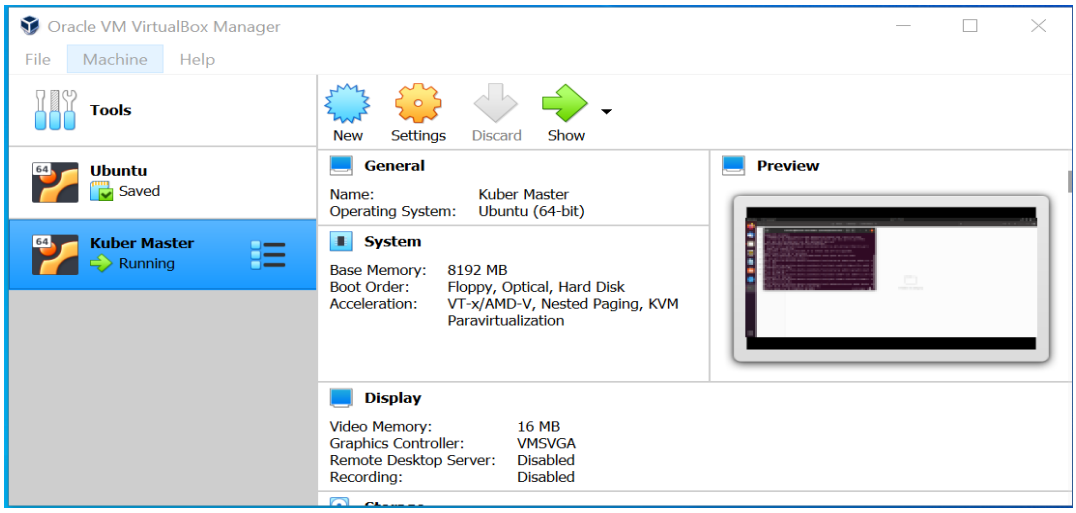

Рисунок 3.1 – Віртуальна машина в Oracle VM

Для роботи потрібно запустити її та встановити образ Ubuntu проходячи процес встановлення та конфігурації.

3.2 Встановлення кластеру Kubernetes

Підготувавши віртуальну машину перейдімо до встановлення оркестретору. Повноцінна інструкція знаходиться на офіційному сайті з послідовністю шагів. Спочатку потрібно встановити Docker адже Kubernetes також є платформою, яка складається з набору контейнерів. Для цього виконуємо команду «sudo apt-get install docker.io» в терміналі, та запускаємо docker. Далі йде набір команд, які встановлюють всі необхідні інструменти для повноцінного запуску та роботи кластеру. Список команд наступний:

```
sudo systemctl enable docker
sudo systemctl status docker
sudo systemctl start docker
sudo apt-get install curl
curl -s https://packages.cloud.google.com/apt/doc/apt-key
.gpg | sudo apt-key add
sudo apt-add-repository "deb http://apt.kubernetes.io/ ku
bernetes-xenial main"
sudo apt-get install kubeadm kubelet kubectl
sudo apt-mark hold kubeadm kubelet kubectl
sudo swapoff -a
sudo hostnamectl set-hostname master-node
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```
kubernetes-master:~$ mkdir -p $HOME/.kube
kubernetes-master:~$ sudo cp -i /etc/kubernetes/admin.con
f $HOME/.kube/config
kubernetes-master:~$ sudo chown $(id -u):$(id -g) $HOME/.
kube/config
sudo kubectl apply -f https://raw.githubusercontent.com/c
oreos/flannel/master/Documentation/kube-flannel.yml
kubectl get pods --all-namespaces
kubectl get nodes
```

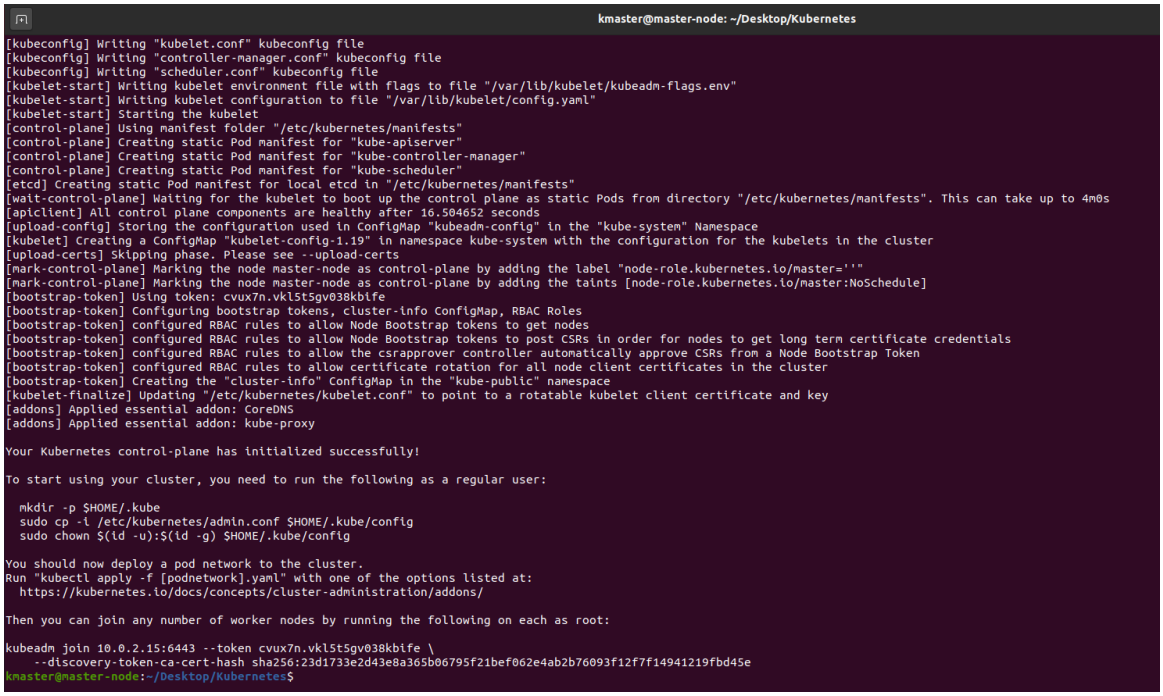

Рисунок 3.2 – Ініціалізація кластера Kubernetes

Виконавши дану інструкцію ми отримуємо набір працюючих контейнерів, який в своїй сукупності надає менеджер додатків Kubernetes. Наступним шагом потрібно встановити дашборду для того, щоб мати доступ до інтерфейсу.

```
sudo kubectl apply -f https://raw.githubusercontent.com/k
ubernetes/dashboard/v2.0.0/aio/deploy/recommended.yaml
kubectl proxy
```

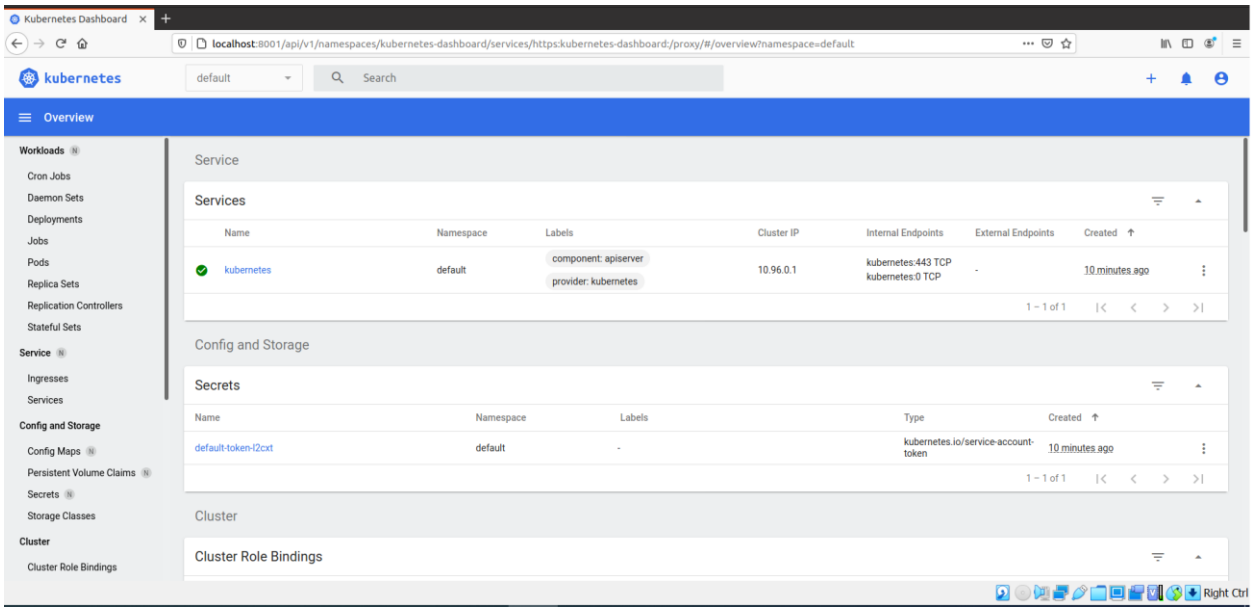

Рисунок 3.3 –Kubernetes юзер інтерфейс

Наступним шагом отримаємо токен для доступу в кластер:

```
cat <<EOF | kubectl apply -f -
apiVersion: v1
kind: ServiceAccount
metadata:
  name: admin-user
  namespace: kubernetes-dashboard
EOF

cat <<EOF | kubectl apply -f -
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: admin-user
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: cluster-admin
subjects:
- kind: ServiceAccount
  name: admin-user
  namespace: kubernetes-dashboard
EOF


kubectl -n kubernetes-dashboard $ ( kubectl -n kubernetes-dashboard  | grep admin-
user | awk ' {print $ 1} ' )
```

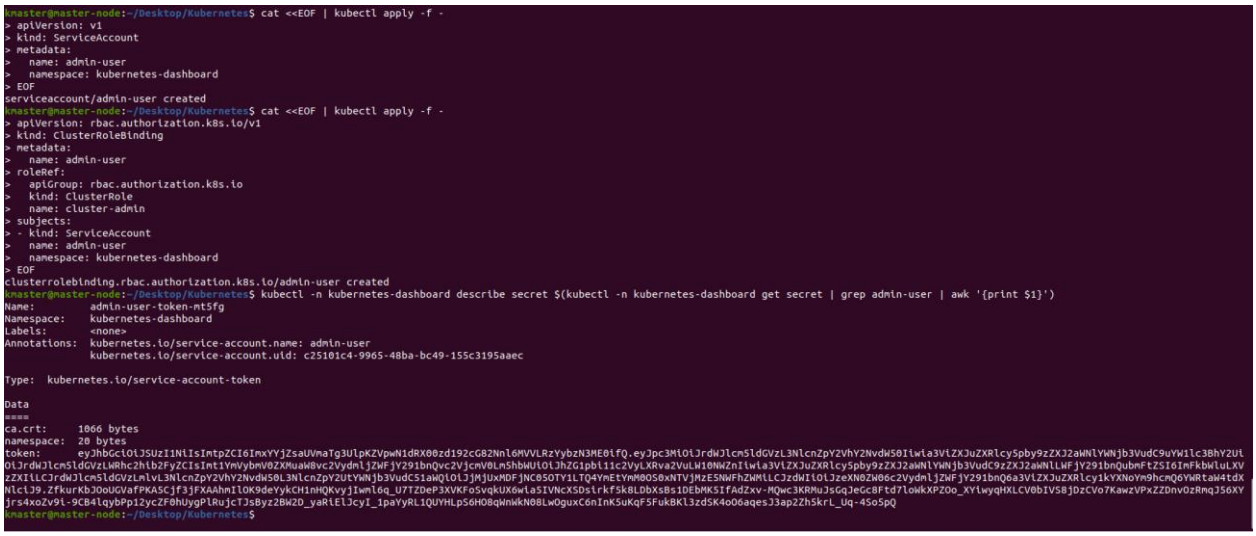

Рисунок 3.4 – Створення дашборди та отримання токену доступу

Кластер Kubernetes готовий до використання. На разі в ньому можна розробляти та тестувати нові додатки.

3.3 Контейнеризований додаток для тестування сервісів

Концепція контейнеризації та ізоляції процесів існує вже давно, але поява в 2013 році Docker прискорило впровадження цієї технології. Контейнеризація - це форма віртуалізації операційної системи, при якій додатки запускаються в ізольованих призначених для користувача просторах, званих контейнерами, і всі вони використовують одну і ту ж загальну операційну систему (ОС). По суті контейнер являє собою повністю упаковану і портативну обчислювальну середу. Все, що необхідно з додатком для запуску - бібліотеки, конфігураційні файли і залежності - інкапсулюється і ізолюється в контейнері. Сам контейнер абстрагується від хостової ОС, маючи лише обмежений доступ до базових ресурсів - подібно легковагій віртуальній машині. В результаті контейнеризований додаток може бути запущено на різних типах інфраструктури: на порожніх машинах, всередині ВМ, в хмарі, без необхідності рефакторінга для кожного середовища. При запуску не потрібно налаштовувати окрему гостьову ОС для кожної програми, так як всі вони використовують ядро однієї операційної системи. Завдяки такій високій ефективності, контейнеризація зазвичай

використовується для упаковки безлічі окремих мікросервісів, з яких складаються сучасні програми [4].

«Chaos Testing App» являється додатком в концепцію якого лягає підхід хаотичного тестування, який дає змогу емулювати проблеми, які можуть виникнути при роботі docker додатків в хмарі і вирішити їх до того як додатком починають користуватися користувачі.

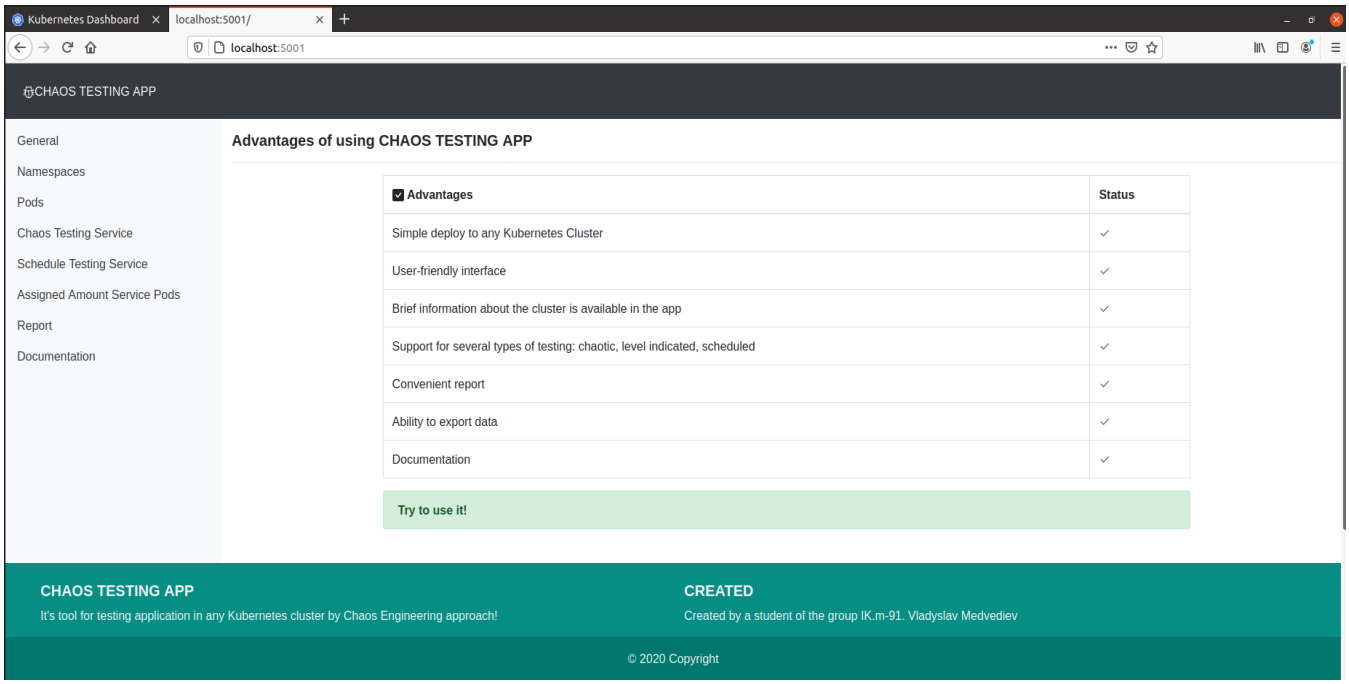

Рисунок 3.5 – Головна сторінка додатку

Додаток має декілька головних переваг в порівнянні з аналогами.

1. По-перше це його простота встановлення. Для цього потрібно лише зберегти декілька головних файлів та запустити дві команди в вашому кластері куди ви хочете його встановити.
2. Наступною перевагою є те, що він має зручний та простий інтерфейс. І користувачу не потрібно розбиратися в безлічі команд для роботи як в більшості аналогів.
3. В додатку доступна стисла інформація про кластер та встановлених в ньому додатків, а це економія часу при його використанні.

4. Декілька підтриманих видів тестування, які дозволять експериментувати з типом тестування та рівнем.
5. Зручний звіт, який укаже тип тестування та список сервісів де відбулося тестування.
6. Підтримка можливості зберегти собі звіт для того щоб пізніше його перевірити.
7. Документація – вказана прямо в додатку.

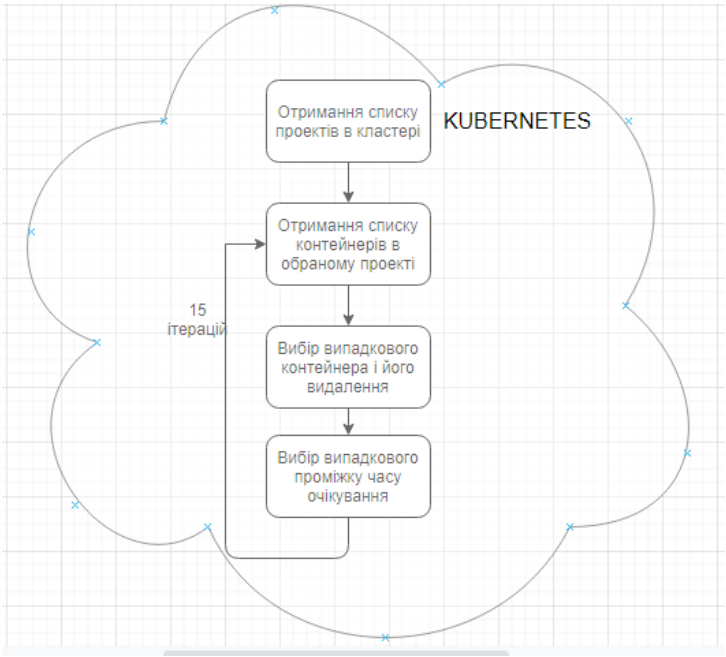

Рисунок 3.6 – Схема алгоритму

Ось такий вигляд має схема роботи алгоритму додатку.

3.4 Навігація та інформаційні сторінки

Оскільки однією з головних переваг додатку є його зручність, то на всіх сторінках передбачена навігація. За її допомогою користувач може відразу перейти на необхідну сторінку[1].

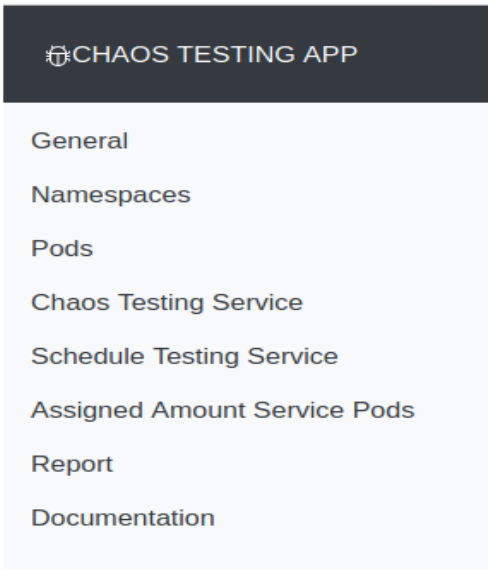

Рисунок 3.7 – Навігаційне меню

Наступним не менш важливим фактором є те, що перед початком тестування чи під час в користувача може виникнути необхідність проаналізувати поточний статус кластера та встановлених в ньому сервісів: доступні проекти, встановлені контейнери та їх статус. Все це можна побачити прямо з додатку – перейти на сторінки «Namespaces» чи «Pods».

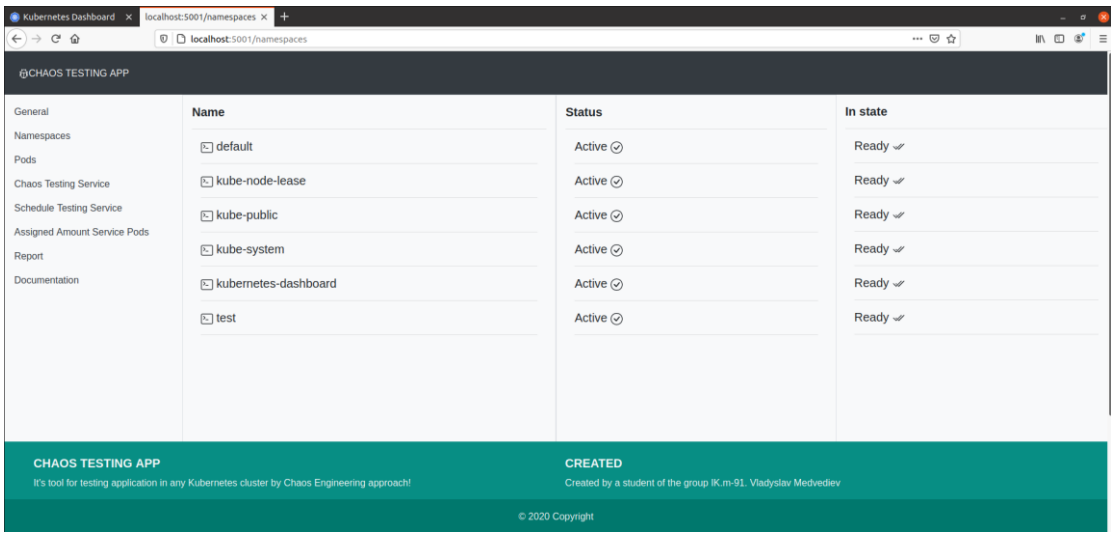

Рисунок 3.8 – Сторінка проектів кластера

Pods є термінологією Kubernetes і являється аналогом контейнеру.

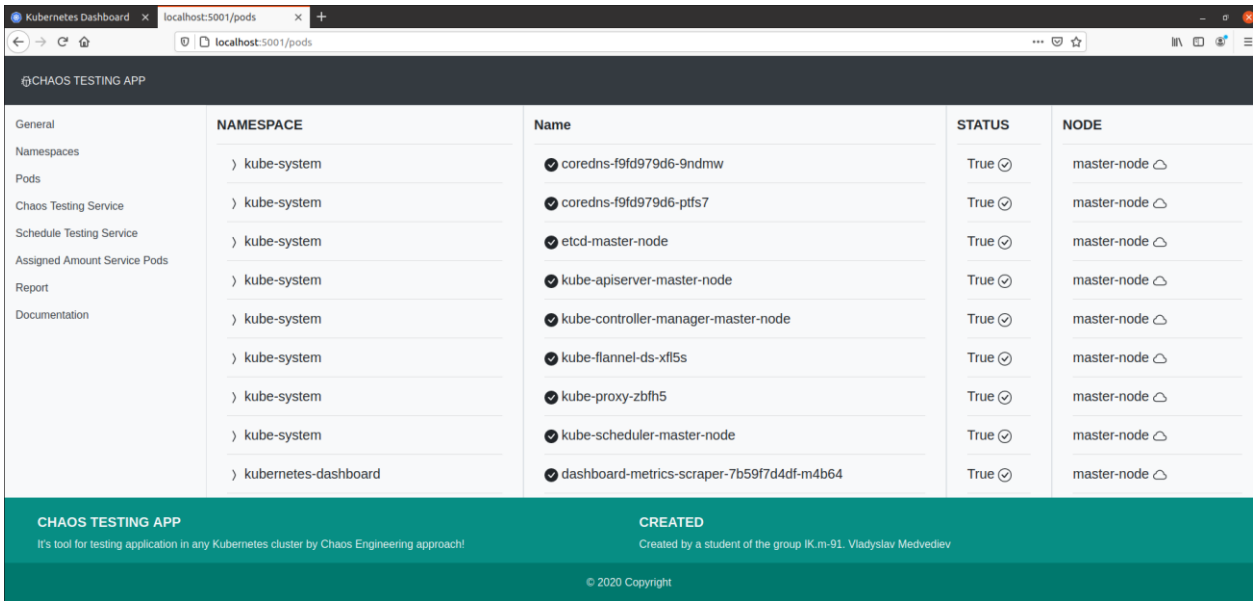

Рисунок 3.9 – Статус pods в проектах

Саме їх статус відображає в якому статусі зараз знаходиться додаток чи компонента додатку [6].

3.5 Тестування за допомогою підходу хаотичного інжинірингу

Головною ціллю даного додатку є хаотичне тестування мікросервісів. В основі реалізації лежить клієнт python, який дозволяє виконувати команди аналогічні діям адміністратора в кластері, чи емулювати різного виду проблеми. Принцип роботи: хаотичний вибір контейнеру додатку чи декількох(в дозволеному проекті) та видалення цього контейнера. Kubernetes як менеджер додатків слідкує за статусом подів та у разі проблеми створення аналогічного для відновлення роботи додатку. В «Chaos Testing App» передбачено три види такого тестування.

Перший найбільш простий, адміністратор обирає проекти в яких дозволяється виконати хаотичне тестування і розпочинає його.

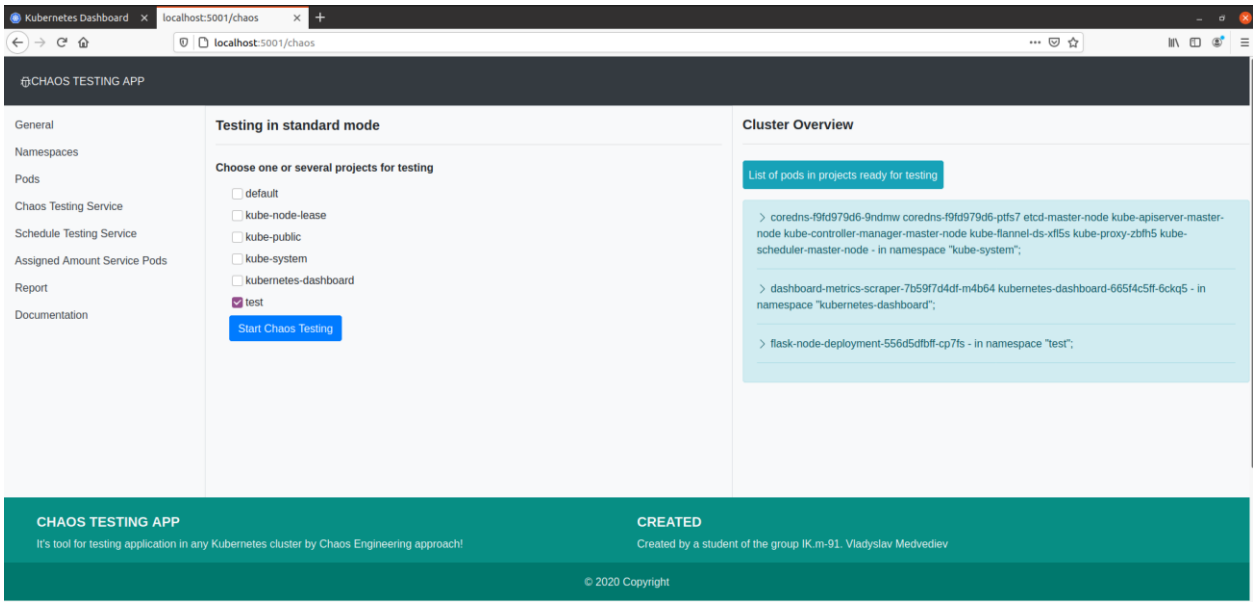

Рисунок 3.10 – Тестування в стандартному режимі

Додаток обирає будь-який под з обраних проектів і видаляє його, потім обирається час без дії від 15 до 150 секунд і ітерація повторюється ще раз. Таких ітерацій передбачено 15.

Наступним видом є аналогічне тестування як і попереднє, але встановлюється час початку тестування. Це є дуже зручним, наприклад, коли кластер активно використовується для тестування чи розробки, а хаотичне тестування потрібно розпочати у встановлений час, наприклад, вночі. Для цього передбачена інша сторінка з додатковим полем, яке вказує дату та час. Приклад в якому форматі вказаний також.

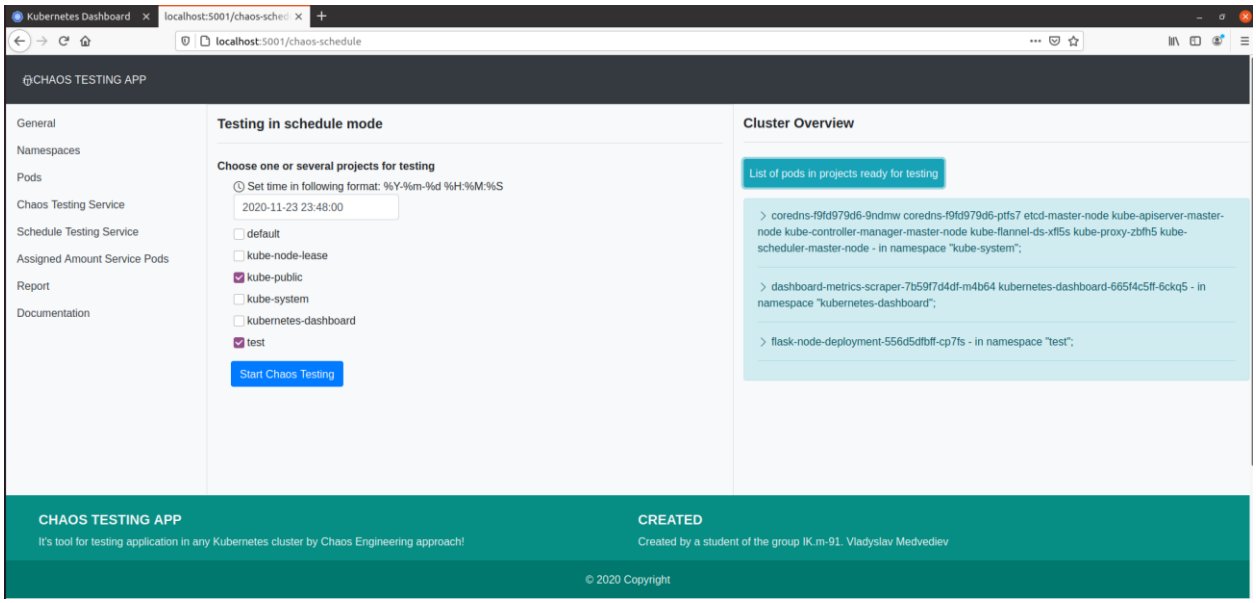

Рисунок 3.11 – Тестування за встановленим часом

І останнім видом є тестування, коли за одну ітерацію видаляється декілька подів, а не лише один. Для цього передбачено додаткове поле, яке приймає ціле число більше 1.

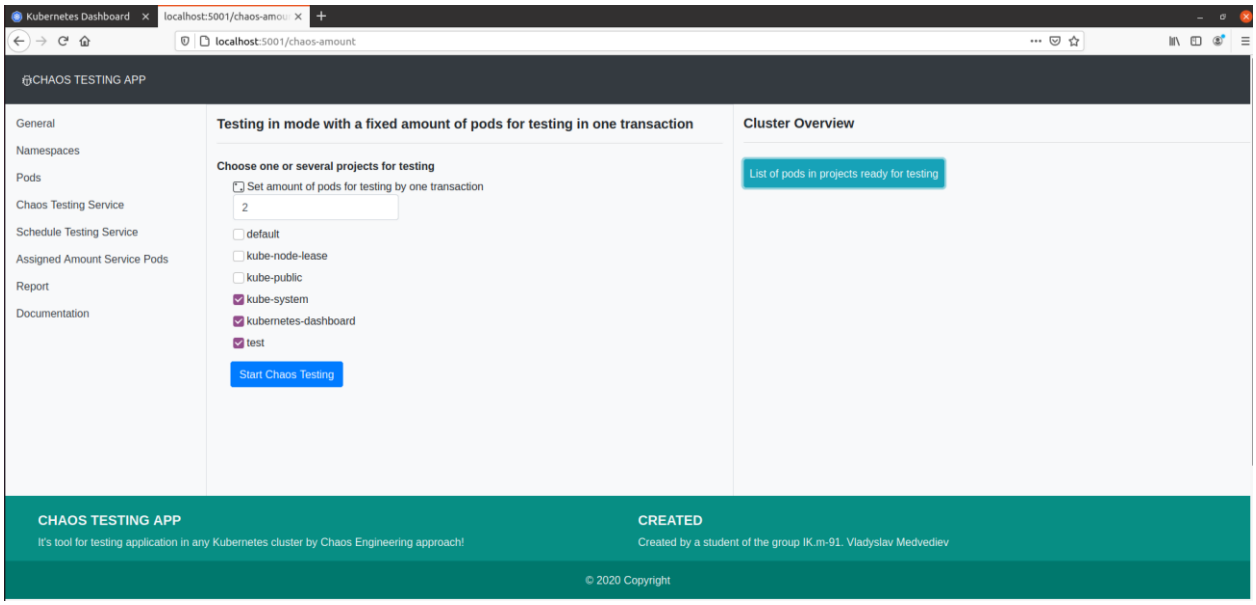

Рисунок 3.12 – Тестування с обраною кількість подів

Всі ці можливості дозволяють виконувати тестування за підходом хаотичного інжинірингу в автоматичному режимі, а також в форматі який більше підходить вам. Маємо деякі можливості у виставленні конфігурації, які підійдуть саме для ваших цілей.

3.6 Формування звіту тестування

Оскільки головною задачею даного додатку є знаходження проблемних місць у сервісах при нестандартних ситуаціях потрібно формувати звіт, який буде в точності відображати дії, які були виконані нашим додатком. Для того, щоб в подальшому мати можливість відновити дії, які приводять до проблеми. Для рішення даної проблеми було створено сторінку для звіту. Реалізовано на основі бази даних SQLite модуля python. SQLite - це С бібліотека, яка реалізує легковажну дискову базу даних (БД), яка не потребує окремого серверного процесу і дозволяє отримати доступ до БД з використанням мови запитів SQL. Деякі додатки можуть використовувати SQLite для внутрішнього зберігання даних. Також можливо створити прототип додатка з використанням SQLite, а потім перенести код в більш багатофункціональну БД, таку як PostgreSQL або Oracle.

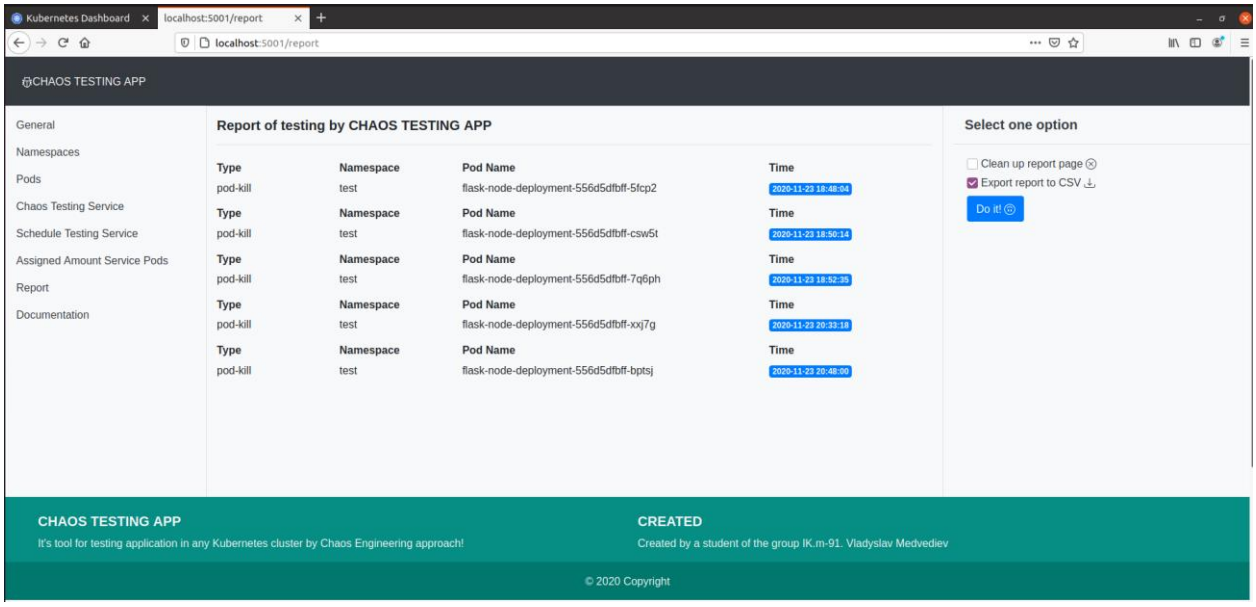

Рисунок 3.13 – Сторінка звіту

Сторінка звіту має приємний та легких дизайн, відображає назви подів, дії які з ними відбувалися та час, що дає змогу в подальшому аналізі та виявленні проблем.

Також має дві необхідних функції. Перша по очистці сторінки звіту, наприклад, перед запуском нового тестування. А також можливість зберегти результати на свій локальний комп’ютер в форматі SCV.

Отже, даний додаток має всі необхідні можливості для виконання автоматичного хаотичного тестування та аналізу дій у зручній формі.

3.7 Встановлення додатку в кластер Kubernetes

Перш за все потрібно створити docker image нашого додатку. Docker-образ (Docker-image) – файл, що включає залежності, відомості, конфігурацію для подальшого розгортання і ініціалізації контейнера. Для його створення потрібно додати декілька обов`язкових файлів: Dockerfile, requirements. Деякі пояснення відносно них. Dockerfile – це файл, який описує набір правил по збірці образу, в якому перший рядок вказує на базовий образ. Наступні команди виконують копіювання файлів і установку програм для створення певного середовища для розробки [12]. В нашому випадку він виглядає наступним чином.

```
FROM python:3.6.2
RUN apt-get update
RUN mkdir /app
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 5000
ENTRYPOINT [ "python3" ]
CMD [ "app.py" ]
```

Файл requirements – описує набір бібліотек необхідних для нашого додатку, які будуть включені в наш образ.

```
APScheduler==3.6.3
Flask==1.1.2
Jinja2==2.11.2
kubernetes==12.0.1
openshift==0.6.0
requests==2.25.0
schedule==0.6.0
```

Після їх створення потрібно виконати команду для побудови образу.

```
sudo docker build -t app:latest .
```

В результаті ми отримуємо docker image нашого додатку, який в подальшому можна використовувати для створення контейнерів. Для того, щоб ми мали

можливість використовувати його в нашому кластері ми маємо зберегти образ в будь-якому docker reqistry – це місце, де зберігаються образи. Для наших цілей ми використовуємо локальний реєстр, який буде розташований на нашій віртуальній машині. Для створення реєстру маємо запустити контейнер реєстру, створити тег для нашого образу додатку, та зберегти образ в локальному реєстрі. Після цього образ нашого додатку буде доступний в нашому кластері Kubernetes і на його основі ми матимемо змогу створити сервіс.

Для встановлення сервісу до кластеру нам знадобиться декілька конфігураційних файлів: deployment, service, cluster role, role binding, service-account. Вся конфігурація і описання цих файлів буде знаходитися в додатках роботи. А для створення об’єктів потрібно виконати команду:

`kubectl apply -f /deploy` , де /deploy – шлях до всіх файлів.

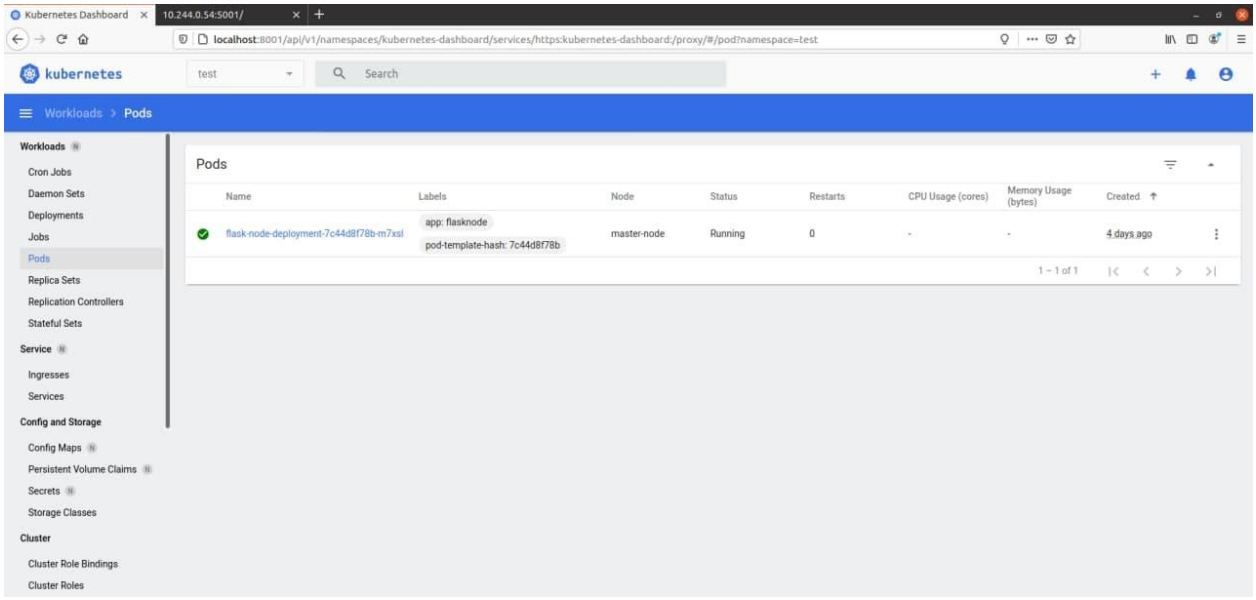

Рисунок 3.14 – Запущений додаток в Kubernetes

В результаті ми отримаємо под в якому буде запущений та працюючий додаток. Доступний додаток буде по адресі http://localhost:5051.

ВИСНОВКИ

Основною задачею даної роботи було створення контейнеризованого python додатку в основі якого лежить принцип хаотичного тестування на базі Chaos Engineering підходу та встановлення його до Kubernetes кластеру. Для реалізації цього був виконаний детальний аналіз предметної області, було визначено найбільш зручні та перспективні технології, а також зіставлення та порівняння подібних рішень. В результаті був складений список технологій для реалізації, який описується в ході роботи. Додаток був створений та відповідає всім необхідним критеріям. Проведено тестування, яка дозволяє роботи висновок, що всі необхідні функції виконуються правильно та додаток має перелік переваг, а саме:

- простота встановлення та використання;
- можливість вибору режим роботи;
- запуск роботи за графіком у встановлений час;
- наявність звіту тестування для подальшого аналізу.

Подальшою перспективою розвитку проекту є підтримка більшої кількості видів конфігурацій для тестування, що дозволить розширити можливість експериментів та підібрати необхідний режим. А також оскільки даний додаток має можливість виключно в тестуванні встановлених додатків, потрібно підтримати можливість тестування інфраструктури у вигляді віртуальних машин, адже стабільність роботи інфраструктури та спроможності переживати різного виду проблеми залежить стабільність роботи всього продукту.

СПИСОК ЛІТЕРАТУРИ

1. Гринберг, М. Разработка веб-приложений с использованием Flask на языке Python - М.: ДМК, 2016. - 272 с.

2. R. Chamberlain and J. Schommer Using Docker to Support Reproducible Research. – California: O'Reilly Media, 2014 - 420c.

3. Эдриен М. Использование Docker. Разработка и внедрение программного обеспечения при помощи технологии контейнеров. Руководство - М.: ДМК Пресс, 2017. - 427 с.

4. Дремина М. Проектный подход к разработке и внедрению систем менеджмента качества - М.: Лань, 2017**.** - 142 с.

5. Casey Rosenthal and Lorin Hochstein Chaos Engineering. Building Confidence in System Behavior through Experiments. – California: O'Reilly Media, 2017 - 320c.

6. Фрейен Б. HTML5 и CSS3. Разработка сайтов для любых браузеров и устройств - М.: Питер, 2014. - 304 с.

7. Spurlock J. Bootstrap. Responsive Web-Development. – California: O'Reilly Media, 2014. — 128 с.

8. Лэнгоун, Д. Виртуализация настольных компьютеров с помощью VMware View 5: моногр.- М.: ДМК Пресс, 2013. - 268 с.

9.Docker [Електронний ресурс]//URL: - https://eternalhost.net/blog/razrabotka

10. Kubernetes open-source system [Електронний ресурс]//URL: - https://kubernetes.io/

11. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.

12. Docker reference documentation [Електронний ресурс]//URL: - https://docs.docker.com/reference/

Додатки

```
import kubernetes
import random
import time
import sqlite3
import sched
import os
import csv
import time as time_module
from flask import Flask, render_template, request, url_for, flash, redirect
from kubernetes import client, config
app = Flask(__name__)
kub_token =
'eyJhbGciOiJSUzI1NiIsImtpZCI6IkNRY0k1T1VVLVFISFFsSndnX0xtZTB0X3J
PSXMxSFlHNmdTSG4wa1Y5bDAifQ.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY
2VhY2NvdW50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc
3BhY2UiOiJrdWJlcm5ldGVzLWRhc2hib2FyZCIsImt1YmVybmV0ZXMuaW8vc
2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJhZG1pbi11c2VyLXRva2VuL
WZzNWRkIiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9zZXJ2aWNl
LWFjY291bnQubmFtZSI6ImFkbWluLXVzZXIiLCJrdWJlcm5ldGVzLmlvL3Nlc
nZpY2VhY2NvdW50L3NlcnZpY2UtYWNjb3VudC51aWQiOiIxYTVmNjgxZC0
xOWQ1LTQwNzItYWY4Ny0xNzk4NmZjYTBiZGQiLCJzdWIiOiJzeXN0ZW06
c2VydmljZWFjY291bnQ6a3ViZXJuZXRlcy1kYXNoYm9hcmQ6YWRtaW4tdX
NlciJ9.qfNl80Eu-
qeUYrs5L3xr4hVtDH0Br1ocE2q5JDwz6Q_OA9r8qX9rbLjgPwqtfEFD0icvc2Xn
CcAudX0xR5hGlKWXhblwkPhSICXW6-
noADRnT8GZe1IKteCxFjJnh_FFvZmfN3fAAgh01RI7VodWw0QRFImywMhS0
DjWPuv-
Mq6imCAsrz4f8MYffsSbV7Ia7qhgOeIxcbQoNz3fC7eiE5ugUk6dgeo1wTa1DSM
```

```
GRpOUPCmc64Iqfn0wj0F9KC0Y1JdwUjUhTMY-
hK_quexyItY6Kd_tyeMOTVg1T5qi2Nhdxf-djo4ozTrEV36F5l-yzb-
IRxw263BXuqOXfup1AQ'

def kuber(cluster='http://localhost:8001', token=kub_token):
    cluster = None
    if cluster is None:
        try:
            config.load_incluster_config()
            _core_v1_api = client.CoreV1Api()
            return _core_v1_api
        except config.ConfigException as e:
            print("Can't load incluster kubernetes config. This script is intended to use
inside of kubernetes")
    else:
        conf = client.Configuration()
        conf.host = cluster
        conf.verify_ssl = False
        conf.api_key = {"authorization": "Bearer " + token}
        _api_client = client.ApiClient(conf)
        # _apps_v1_api = client.AppsV1Api(self._api_client)
        _core_v1_api = client.CoreV1Api(_api_client)
        return _core_v1_api

def list_pods(namespace='test'):
    client = kuber(cluster='http://localhost:8001', token=kub_token)
    all_pods = client.list_namespaced_pod(namespace=namespace)
    pod_name = []
    for pod in all_pods.items:
        pod_name.append(pod.metadata.name)
```

```
    return pod_name

def information_pod():
    client = kuber(cluster='http://localhost:8001', token=kub_token)
    list_n = list_projects()
    pod_name = []
    pod_namespace = []
    pod_status = []
    pod_node = []
    for n in list_n:
        all_pods = client.list_namespaced_pod(namespace=n)
        for pod in all_pods.items:
            pod_namespace.append(pod.metadata.namespace)
            pod_name.append(pod.metadata.name)
            pod_status.append(pod.status.conditions[0].status)
            pod_node.append(pod.spec.node_name)
    return pod_name, pod_namespace, pod_status, pod_node

def list_projects():
    client = kuber(cluster='http://localhost:8001', token=kub_token)
    all_namespaces = client.list_namespace()
    namespaces = []
    for ns in all_namespaces.items:
        namespaces.append(ns.metadata.name)
    print(namespaces)
    return namespaces

def delete_pod_by_name(pod, namespace='test'):
    client = kuber(cluster='http://localhost:8001', token=kub_token)
    body = kubernetes.client.V1DeleteOptions()
```

```
    client.delete_namespaced_pod(name=pod, namespace=namespace, body=body)
    print('Pod: ' + pod + ' was deleted')
    add_to_db(pod, namespace)

def random_deleting_pods(namespace='test'):
    for i in range(15):
        list_namespaced_pods = list_pods(namespace)
        print('List pods in selected ptoject: ' + str(list_namespaced_pods))
        random_pod = random.choice(list_namespaced_pods)
        print('Random pod: ' + str(random_pod))
        delete_pod_by_name(random_pod, namespace)
        print('Random pod was deleted')
        sleep_time = random.randint(15, 150)
        print('Sleep time: ' + str(sleep_time))
        time.sleep(sleep_time)

def random_deleting_several_pods(namespace='test', amount=1):
    for i in range(15):
        for j in range(amount):
            print('Iteration for deleting pod: ' + str(j))
            list_namespaced_pods = list_pods(namespace)
            print('List pods in selected ptoject: ' + str(list_namespaced_pods))
            random_pod = random.choice(list_namespaced_pods)
            print('Random pod: ' + str(random_pod))
            delete_pod_by_name(random_pod, namespace)
            print('Random pod was deleted')
        sleep_time = random.randint(15, 150)
        print('Sleep time: ' + str(sleep_time))
        time.sleep(sleep_time)
```

```
def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn

def add_to_db(name, content):
    conn = get_db_connection()
    conn.execute('INSERT INTO report (name, content) VALUES (?, ?)', (name,
content))
    conn.commit()
    conn.close()

def read_from_db():
    conn = get_db_connection()
    all_from_db = conn.execute('SELECT * FROM report').fetchall()
    conn.close()
    return all_from_db

def delete_all_from_db():
    conn = get_db_connection()
    conn.execute('DELETE FROM report')
    conn.commit()
    conn.close()

# Export from DB to CSV file
def export_csv():
    print("******Employee Table Data*******")
    conn = sqlite3.connect('database.db')
    cursor = conn.cursor()
    cursor.execute("select * from report")
```

```
    with open("employee_data.csv", "w") as csv_file:
        csv_writer = csv.writer(csv_file, delimiter="\t")
        csv_writer.writerow([i[0] for i in cursor.description])
        csv_writer.writerows(cursor)
    dirpath = os.getcwd() + "/employee_data.csv"
    conn.close()
    print('Data exported Successfully into {}'.format(dirpath))

@app.route('/')
def index():
    return render_template('base.html')

@app.route('/namespaces')
def namespaces():
    list_n = list_projects()
    all_namepsaces_kuber = ''
    for n in list_n:
        all_namepsaces_kuber = all_namepsaces_kuber + str(n)
    return render_template('namespaces.html', namespaces=list_n)

@app.route('/pods')
def pods():
    pod_name, pod_namespace, pod_status, pod_node = information_pod()
    print(pods)
    return render_template('pods.html', pod_name=pod_name,
pod_namespace=pod_namespace, pod_status=pod_status, pod_node=pod_node)

@app.route('/pods/delete')
def pods_delete():
    name = delete_pod_by_name()
```

```
    return 'Following pods were deleted' + str(name)


@app.route('/chaos', methods=('GET', 'POST'))
def chaos():
    namespaces = list_projects()
    list_objects = []
    for nm in namespaces:
        b_pods = ''
        all_pods_project = list_pods(nm)
        if len(all_pods_project) > 0:
            for j in all_pods_project:
                b_pods = b_pods + str(j) + ' '
            b_pods = b_pods + ' - in namespace "' + str(nm) + '";'
            list_objects.append(b_pods)
    if request.method == 'POST':
        selected_project = request.form.getlist("users")
        print('Select: ' + str(selected_project))
        # title = request.form['title']
        # content = request.form['content']
        for i in range(len(selected_project)):
            random_deleting_pods(namespace=selected_project[i])
        return redirect('base.html')
    return render_template('chaos.html', objects=list_objects, projects=namespaces)


@app.route('/chaos-schedule', methods=('GET', 'POST'))
def chaos_schedule():
    namespaces = list_projects()
    list_objects = []
    for nm in namespaces:
        b_pods = ''
```

```
        all_pods_project = list_pods(nm)
        if len(all_pods_project) > 0:
            for j in all_pods_project:
                b_pods = b_pods + str(j) + ' '
            b_pods = b_pods + ' - in namespace "' + str(nm) + '";'
            list_objects.append(b_pods)
    if request.method == 'POST':
        selected_project = request.form.getlist("users")
        selected_time = request.form['time']
        print('Select project: ' + str(selected_project))
        print('Select time: ' + str(selected_time))
        for i in range(len(selected_project)):
            scheduler = sched.scheduler(time_module.time, time_module.sleep)
            t = time_module.strptime(selected_time, '%Y-%m-%d %H:%M:%S')
            t = time_module.mktime(t)
            scheduler_e = scheduler.enterabs(t, 1, random_deleting_pods,
argument=(selected_project[i],))
            scheduler.run()
            print('Schedule started')
            return redirect('/')
    return render_template('chaos-schedule.html', objects=list_objects,
projects=namespaces)

@app.route('/chaos-amount', methods=('GET', 'POST'))
def chaos_amount():
    namespaces = list_projects()
    list_objects = []
    for nm in namespaces:
        b_pods = ''
        all_pods_project = list_pods(nm)
```

```
        if len(all_pods_project) > 0:
            for j in all_pods_project:
                b_pods = b_pods + str(j) + ' '
            b_pods = b_pods + ' - in namespace "' + str(nm) + '";'
            list_objects.append(b_pods)
    if request.method == 'POST':
        selected_project = request.form.getlist("users")
        selected_amount = request.form['amount']
        print('Select project: ' + str(selected_project))
        print('Select time: ' + str(selected_amount))
        for i in range(len(selected_project)):
            random_deleting_several_pods(namespace=selected_project[i],
amount=int(selected_amount))
            print('Schedule started')
            return redirect('/')
    return render_template('chaos-amount.html', objects=list_objects,
projects=namespaces)

@app.route('/report', methods=('GET', 'POST'))
def reports():
    if request.method == 'POST':
        options = request.form.getlist("repor")
        print('Options: ' + str(options))
        if options[0] == 'Clean':
            delete_all_from_db()
            print('DB is empty!')
            return redirect('/report')
        elif options[0] == 'Export':
            print('CSV!!!')
            export_csv()
```

```
            mess = 'Data exported Successfully'
            return redirect('/report')
    rep = read_from_db()
    return render_template('report.html', posts=rep)

@app.route('/doc')
def docs():
    return render_template('doc.html')

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5001)
```

HTML та Bootstrap

```
<!doctype html>
<html lang="en">
<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

    <title>{% block title %} {% endblock %}</title>
</head>
<body style="padding-bottom:150px;">
```

```
<div class="container-fluid p-4 bg-dark text-white">
  <svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-bug" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
    <path fill-rule="evenodd" d="M4.355.522a.5.5 0 0 1 .623.333l.291.956A4.979 4.979 0 0 1 8 1c1.007 0 1.946.298 2.731.811l.29-.956a.5.5 0 1 1 .957.29l-.41 1.352A4.985 4.985 0 0 1 13 6h.5a.5.5 0 0 0 .5-.5V5a.5.5 0 0 1 1 0v.5A1.5 1.5 0 0 1 13.5 7H13v1h1.5a.5.5 0 0 1 0 1H13v1h.5a1.5 1.5 0 0 1 1.5 1.5v.5a.5.5 0 1 1-1 0v-.5a.5.5 0 0 0-.5-.5H13a5 5 0 0 1-10 0h-.5a.5.5 0 0 0-.5.5v.5a.5.5 0 1 1-1 0v-.5A1.5 1.5 0 0 1 2.5 10H3V9H1.5a.5.5 0 0 1 0-1H3V7h-.5A1.5 1.5 0 0 1 1 5.5V5a.5.5 0 0 1 1 0v.5a.5.5 0 0 0 .5.5H3c0-1.364.547-2.601 1.432-3.503l-.41-1.352a.5.5 0 0 1 .333-.623zM4 7v4a4 4 0 0 0 3.5 3.97V7H4zm4.5 0v7.97A4 4 0 0 0 12 11V7H8.5zM12 6H4a3.99 3.99 0 0 1 1.333-2.982A3.983 3.983 0 0 1 8 2c1.025 0 1.959.385 2.666 1.018A3.989 3.989 0 0 1 12 6z"/>
  </svg>CHAOS TESTING APP
</div>
<div class="row">
  <!--Navbar-->
  <div class="col-md-2 bg-light bottom" style="height: 100vh;">
    <!--Navbar-->
    <nav class="navbar text-white">
      <!-- Links -->
      <ul class="navbar-nav h-100">
        <li class="nav-item">
          <a class="nav-link text-dark display-5" href="/">General</a>
        </li>
        <li class="nav-item">
          <a class="nav-link text-dark display-5" href="/namespaces">Namespaces</a>
```

```
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark display-5" href="/pods">Pods</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark display-5" href="/chaos">Chaos Testing Service</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark display-5" href="/chaos-schedule">Schedule Testing Service</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark display-5" href="/chaos-amount">Assigned Amount Service Pods</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark display-5" href="/report">Report</a>
            </li>
            <li class="nav-item">
              <a class="nav-link text-dark display-5" href="/doc">Documentation</a>
            </li>
          </ul>
        </nav>
      </div>
      <div class="col-md-10">
        <h5 class="font-weight-bold pt-3 mb-3">Advantages of using CHAOS TESTING APP </h5>
        <hr>
```

```
<div class="container">
  <table class="table table-bordered">
    <thead>
    <tr>
      <th><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-check-square-fill" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
        <path fill-rule="evenodd" d="M2 0a2 2 0 0 0-2 2v12a2 2 0 0 0 2 2h12a2 2 0 0 0 2-2V2a2 2 0 0 0-2-2H2zm10.03 4.97a.75.75 0 0 0-1.08.022L7.477 9.417 5.384 7.323a.75.75 0 0 0-1.06 1.06L6.97 11.03a.75.75 0 0 0 1.079-.02l3.992-4.99a.75.75 0 0 0-.01-1.05z"/>
      </svg>  Advantages</th>
      <th>Status</th>
    </tr>
    </thead>
    <tbody>
    <tr>
      <td>Simple deploy to any Kubernetes Cluster</td>
      <td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
        <path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7 7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0 1 .708 0z"/>
      </svg></td>
    </tr>
    <tr>
      <td>User-friendly interface</td>
      <td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
```

<path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7 7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0 1 .708 0z"/>
</svg></td>
</tr>
<tr>
<td>Brief information about the cluster is available in the app</td>
<td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
<path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7 7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0 1 .708 0z"/>
</svg></td>
</tr>
<tr>
<td>Support for several types of testing: chaotic, level indicated, scheduled</td>
<td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
<path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7 7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0 1 .708 0z"/>
</svg></td>
</tr>
<tr>
<td>Convenient report</td>
<td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">

```
                <path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7
7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0
1 .708 0z"/>
            </svg></td>
        </tr>
        <tr>
            <td>Ability to export data</td>
            <td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi
bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
                <path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7
7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0
1 .708 0z"/>
            </svg></td>
        </tr>
        <tr>
            <td>Documentation</td>
            <td><svg width="1em" height="1em" viewBox="0 0 16 16" class="bi
bi-check2" fill="currentColor" xmlns="http://www.w3.org/2000/svg">
                <path fill-rule="evenodd" d="M13.854 3.646a.5.5 0 0 1 0 .708l-7
7a.5.5 0 0 1-.708 0l-3.5-3.5a.5.5 0 1 1 .708-.708L6.5 10.293l6.646-6.647a.5.5 0 0
1 .708 0z"/>
            </svg></td>
        </tr>
        </tbody>
    </table>
    <div class="alert alert-success">
        <strong>Try to use it!</strong>
    </div>
  </div>
</div>
```

```
<!-- Footer -->
<footer class="page-footer font-small green pt-4 fixed-bottom">
  <div class="container-fluid text-center text-md-left">
    <div class="row" style="background-color:#078e84;">
      <div class="col-md-6 mt-md-0 mt-3 text-light pt-4 pl-5">
        <h5 class="text-uppercase font-weight-bold">CHAOS TESTING APP</h5>
        <p>It's tool for testing application in any Kubernetes cluster by Chaos Engineering approach!</p>
      </div>
      <hr class="clearfix w-100 d-md-none pb-3 text-light">
      <div class="col-md-6 mb-md-0 mb-3 text-light pt-4">
        <h5 class="text-uppercase font-weight-bold">CREATED</h5>
        <p>Created by a student of the group IK.m-91.
          Vladyslav Medvediev</p>
      </div>
    </div>
  </div>
  <div class="footer-copyright text-center py-3 text-light" style="background-color:#00786d;">© 2020 Copyright
  </div>
</footer>
<!-- Footer END-->

<!-- Optional JavaScript -->
<!-- jQuery first, then Popper.js, then Bootstrap JS -->
<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>
```

```
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHND
z0W1" crossorigin="anonymous"></script>
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM
" crossorigin="anonymous"></script>
</body>
</html>
```

YAML templates

```
apiVersion: apps/v1
 kind: Deployment
 metadata:
   name: flask-node-deployment
 spec:
   replicas: 1
   selector:
     matchLabels:
       app: flasknode
   template:
     metadata:
       labels:
         app: flasknode
```

```
    spec:
#       serviceAccountName: app-test
      containers:
      - name: flasknode
        image: localhost:5000/app
        imagePullPolicy: Always
        ports:
        - containerPort: 5000
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: app-test
rules:
  - apiGroups:
      - ""
    resources:
      - pods
      - services
      - configmaps
      - configmap
      - secrets
      - serviceaccounts
      - namespaces
    verbs:
```

```
    - '*'
  - apiGroups:
    - apps
    resources:
    - deployments
    - replicasets
    - statefulsets
    verbs:
    - '*'
  - apiGroups:
    - ""
    resources:
    - pods
    verbs:
    - get
  - apiGroups:
    - "
    resources:
    - namespaces
    verbs:
    - '*'
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
```

```
  name: app-test
subjects:
  - kind: ServiceAccount
    name: default
    namespace: test2
roleRef:
  kind: Role
  name: app-test
  apiGroup: rbac.authorization.k8s.io
apiVersion: v1
 kind: Service
 metadata:
   name: flask-node-deployment
 spec:
   ports:
   - port: 5000
     targetPort: 5000
   selector:
     app: flasknode
apiVersion: v1
kind: ServiceAccount metadata:
  name: app-test
```