НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

«На правах рукопису»
УДК ___004.415.2___

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра

за освітньо-науковою програмою «Програмне забезпечення інформаційно-комунікаційних систем»

зі спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Автоматизована система резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері»

Виконала:
студентка VI курсу, групи ІТ-91мн
Цитовцева Анна Сергіївна _____

Керівник:
доцент кафедри АУТС, к.т.н., доцент
Катін Павло Юрійович _____

Рецензент:
Доцент кафедри ОТ, к.т.н., доцент
Волокита Артем Миколайович _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра автоматики та управління в технічних системах

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-наукова програма «Програмне забезпечення інформаційно-комунікаційних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту

Цитовцевій Анні Сергіївні

1. Тема дисертації «Автоматизована система резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері», науковий керівник дисертації Катін Павло Юрійович, к.т.н., доцент, затверджені наказом по університету від «12» березня 2021 р. №809-с

2. Термін подання студентом дисертації 11.05.2021

3. Об’єкт дослідження автоматизована система резервного копіювання даних сервісів в системах з мікросервісною архітектурою у Kubernetes кластері.

4. Предмет дослідження характеристики надійності резервування та створення консистентної резервної копії даних мікросервісів.

5. Перелік завдань, які потрібно розробити аналіз проблеми резервного копіювання даних в застосунках з мікросервісною архітектурою, огляд існуючих рішень, створення математичної моделі, розробка алгоритму створення та автоматизації процесу резервного копіювання даних в умовах роботи системі на базі мікросервісної архітектури, проведення експериментального дослідження.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу блок-схема алгоритму роботи менеджеру резервних копій, діаграма послідовності, діаграма класів, діаграма компонентів, діаграма розгортання, структурна схема, схема тестового середовища, діаграма прецедентів.

7. Орієнтовний перелік публікацій «Технічні науки і технології», «Проблеми інформатизації та управління», «WORLD SCIENCE: PROBLEMS, PROSPECTS AND INNOVATIONS».

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

9. Дата видачі завдання 01.02.2021

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1.	Огляд предметної області	01.02.2021	
2.	Аналіз існуючих рішень	08.02.2021	
3.	Аналіз та вибір технічних засобів для реалізації системи	15.02.2021	
4.	Розгортання середовища для розробки та тестування	22.02.2021	
5.	Розроблення програмного рішення	26.03.2021	
6.	Тестування розробленої системи	12.04.2021	
7.	Розроблення стартап – проекту	19.04.2021	
8.	Оформлення текстової документації	26.04.2021	
9.	Подання готової роботи	11.05.2021	

Студент Анна ЦИТОВЦЕВА

Науковий керівник Павло КАТІН

РЕФЕРАТ

Магістерська дисертація на тему «Автоматизована система резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері».

Дисертація містить 102 с. тексту, 34 рисунки, 21 таблиця, 21 джерело та 2 додатки.

Об’єктом розробки є дані сервісів в системах з мікросервісною архітектурою у Kubernetes кластері.

Метою магістерської дисертації є створення цілісної та консистентної резервної копії даних мікросервісів у кластері Kubernetes. В ході було розглянуто процес тестування на різних типах баз даних, досліджено дані, які дозволяє зберігати на поді оркестратор Kubernetes та можливість їх резервного копіювання. На практиці проведено перевірку працездатності системи, а також можливість конфігурування параметрів для певної інфраструктури та отримати на виході цілісну резервну копію.

В ході роботи було створено мікросервісний застосунок з використанням мови програмування Java та фреймворку Spring Boot з конфігурацією запуску всередині ноди Kubernetes кластеру.

Проведено моделювання системи резервування з використанням математичного апарату марковських процесів. Було досліджено потоки відмов і потоки відновлення, проведено розрахунки з використанням розробленої моделі.

Вихідна базова система має точки розширення для розвитку та може бути інтегрована у існуючу систему на базі мікросервісної архітектури. У ході отримання результатів розроблено план стартап-проекту.

Ключові слова: Kubernetes, мікросервіси, мікросервісна архітектура, резервне копіювання даних

ABSTRACT

Master's dissertation on "Automated data backup system in solutions with microservice architecture in the Kubernetes cluster."

Dissertation contains 102 pages of text, 34 figures, 21 tables, 21 sources and 2 appendices.

The object of the study is an automated system for backing up data services in systems with microservice architecture in the Kubernetes cluster.

The aim of the master's dissertation is to develop a system for creating a complete backup of microservice data in the Kubernetes cluster. The process of testing on different types of databases was considered, the data that allows the Kubernetes orchestrator to be stored on the podium and the possibility of their backup were investigated. In practice, the system is tested for efficiency, as well as the ability to configure parameters for a particular infrastructure and get a complete backup.

During the work, a microservice application was created using the Java programming language and the Spring Boot framework with a configuration to run inside the Kubernetes node of the cluster.

Modeling of the redundancy system using the mathematical apparatus of Markov processes is carried out. Failure flows and recovery flows were investigated, calculations were performed using the developed model.

The source base system has expansion points for development and can be integrated into an existing system based on a microservice architecture. In the course of obtaining the results, a plan of the startup project was developed.

Keywords: Kubernetes, microservices, microservice architecture, backup

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

API	Application user interface
CI/CD	Continuous integration та continuous delivery
DevOps	Development operation
GCP	Google Cloud Platform
GKE	Google Cloud Engine
GRPC	Google remote procedure call
GUI	Graphical user interface
HTTP	HyperText transfer protocol
HTTPS	Secured HyperText transfer protocol
ОС	Операційна система
PVC	Persistence Volume Claim
Под	Pod в системі Kubernetes

ВСТУП

Перехід до мікросервісної архітектури стає дедалі популярнішим. Такі великі компанії, як Amazon, Netflix, Spotify, а також малі та середні підприємства розробляють системи на базі мікросервісів. Мікросервісна архітектура має безліч різних переваг. Мікросервіси можуть бути розроблені на різних мовах програмування, можуть масштабуватися незалежно від інших служб і можуть бути розгорнуті на апаратному забезпеченні, яке найкраще відповідає їхнім потребам. Більше того, через розмір їх легше обслуговувати та роблять систему більш стійкою до несправностей, оскільки вихід з ладу однієї служби не порушить всю систему, що може статися в монолітній системі. Монолітна структура та переважно статична конфігурація компонентів додатків сьогодні обмежують можливості програми динамічно керувати внутрішніми компонентами, мати можливість адаптуватися до користувача та середовища та використовувати різні сервіси в мережі для поліпшення роботи програми.

Перехід до мікросервісної архітектури обіцяє зменшити складність, надати можливість незалежного масштабування, легко видаляти та розгортати незалежні частини системи, підтримувати використання різних платформ та мов, підвищувати загальну гнучкість системи і, нарешті, покращувати стійкість системи. Багато компаній вже використовують цей архітектурний підхід, впроваджуючи його у свої сервіси. Для інженерів програмного забезпечення підтримка продуктивності для таких архітектур ставить багато нових викликів, і з'явилася потреба у програмних підходах, що відповідають конкретним потребам середовищ мікросервісу.

Поява хмарних обчислень як нової парадигми, що надає постачальникам послуг можливість розгортати економічно ефективні рішення, сприяло розробці низки нових послуг, включаючи програми для зберігання даних в Інтернеті. Через економію масштабу хмарних служб зберігання даних, витрати, понесені кінцевими користувачами на передачу своїх даних у віддалене місце зберігання в Інтернеті,

значно зменшились. Таким чином, сервіси менеджменту резервного копіювання даних оберігають користувачів від більшої частини трудомістких неприємностей при резервному копіюванні даних: взаємодія з користувачем мінімальна, а у випадку втрати даних через аварію, відновлення вихідних даних − це безперебійна операція.

Об'єктом дослідження є дані сервісів в системах з мікросервісною архітектурою у Kubernetes кластері.

Предметом дослідження є автоматизована система резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері.

Метою магістерської дисертації є створення системи, що призначена для забезпечення цілісної та консистентної резервної копії даних мікросервісів у кластері Kubernetes.

Поставлена мета досягається засобами налагодження алгоритму та автоматизації процесу резервного копіювання даних у різних типах баз даних.

Конкретні завдання роботи:

- аналіз проблеми резервного копіювання даних в застосунках з мікросервісною архітектурою;
- огляд існуючих рішень;
- створення математичної моделі;
- розробка алгоритму та програми створення та автоматизації процесу резервного копіювання даних в умовах роботи системі на базі мікросервісної архітектури;
- проведення експериментального дослідження.

Беручи до уваги збільшення уваги та інтересу до впровадження мікросервісної архітектури у сучасні рішення та складність версіонування даних у Kubernetes кластері, тема магістерської дисертації є актуальною.

Наукова новизна полягає в тому, що вперше було розроблено алгоритм для створення цілісної та консистентної резервної копії даних мікросервісів у кластері Kubernetes.

Апробація роботи. Результати виконаної дослідницької роботи були опубліковані у збірнику VII Міжнародна науково-практичної конференції “WORLD SCIENCE: PROBLEMS, PROSPECTS AND INNOVATIONS” у доповіді «Способи захисту баз даних від sql ін’єкцій для забезпечення якості програмного забезпечення» у 2021 році в м. Торонто, Канада.

Публікації. Результати виконаної дослідницької роботи були опубліковані у статті «Особливості масштабування контейнерного навантаження на базі системи Kubernetes» в науковому журналі «Технічні науки і технології» у 2021 році в м. Києві, а також у статті «Аналіз доступності мікросервісів на базі системи управління та оркестрації контейнерів Kubernetes» в науковому журналі «Проблеми інформатизації та управління» у 2021 році в м. Києві.

1 АНАЛІЗ ТЕХНОЛОГІЙ ВІРТУАЛІЗАЦІЇ

У цьому розділі наведено необхідні відомості про архітектуру мікросервісів та схеми організації контейнерів. Архітектура мікросервісу дозволяє проектувати додатки як колекцію вільно пов'язаних незалежних служб на відміну від традиційної монолітної архітектури. Контейнери використовуються для інкапсуляції цих окремих послуг з хорошою ізоляцією.

1.1Аналіз мікросервісної архітектури

Термін «мікросервіс» вперше був уведений у травні 2011 року під час семінару архітекторів програмного забезпечення, що відбувся у Венеції [10]. З тих пір архітектура мікросервісів привертає велику увагу та інтерес серед розробників програмного забезпечення. У мікросервісній архітектурі додаток структуровано як сукупність слабо пов'язаних служб. Він складається з модулей, незалежних сутностей програмного забезпечення, які є дуже масштабованими. Залежності розділяються на логічні сутності, а функціональні проблеми розділяються. Кожна модульна сутність програмного забезпечення вирішує одну проблему. Кожна мікрослужба має чітко визначені інтерфейси, за допомогою яких інші організації можуть спілкуватися зі службою. Це дозволяє кожній службі змінити свою внутрішню реалізацію, в той час, як інтерфейс залишається сумісним з іншими службами.

1.1.1 Особливості мікросервісної архітектури

Філософія дизайну мікросервісної архітектури проста, “Зробіть один процес добре”. Архітектура мікросервісу повинна дотримуватися наступних принципів:

- зберігання послуги невеликими, щоб кожна з них вирішувала лише одну проблему [11];
- будь-яка організація, що займається розробкою програмного забезпечення, повинна застосувати автоматичну систему тестування, розгортання та доставки програмного забезпечення. Це дозволяє командам розробників та тестувань працювати над невеликими окремими одиницями розгортання програмного забезпечення [12].

Мікросервіси були визнані надійним програмним рішенням для розробки та масштабування великих програмних продуктів [14]. Архітектура мікросервісу зазвичай найкраще підходить для постійної інтеграції та постійної доставки [15]. Його модульний підхід полегшує цикли розробки та випуску програмного забезпечення, тим самим скорочуючи час виходу на ринок. DevOps (Розробка та експлуатація) – це набір практик, призначених мінімізувати час між внесенням змін до системи та спростити процес змін, які застосовуються у виробництві з високою якістю [16]. На рисунку 1.1 показані тенденції зростання DevOps та мікросервісів згідно з індексом пошуку ключових слів Google. Ця тенденція чітко вказує на зростаючу важливість архітектури мікросервісів щодо DevOps. Кілька великих організацій, таких як Netflix, застосували архітектуру мікросервісів [17] та успішно впровадили великі веб-продукти.

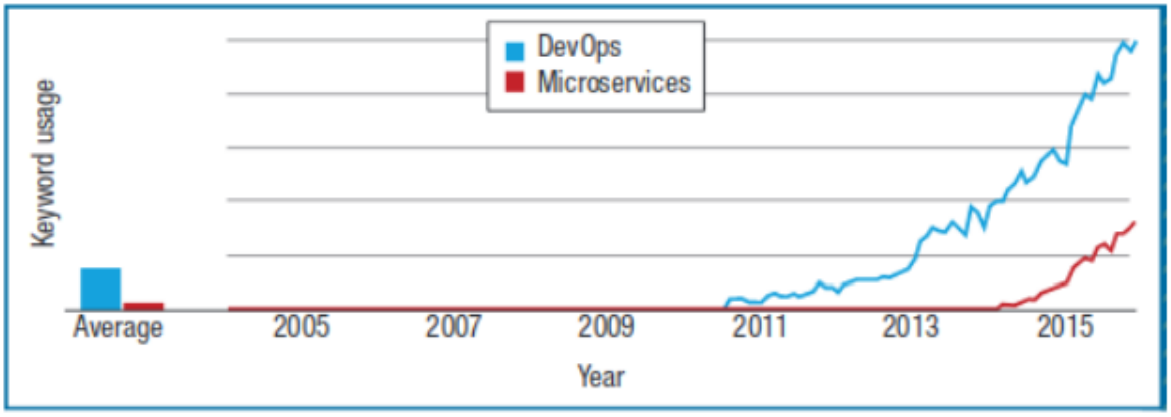

Рисунок 1.1 – Збільшення використання ключових слів DevOps та мікросервісів, відповідно до звіту Google Trends[13]

Кожен мікросервіс може бути розроблений на різній мові програмування в рамках, що надає розробникам програмного забезпечення свободу вибору. Це також дає додаткову перевагу при виборі правильної мови програмування та основи для бажаної функціональності. Мікросервіси взаємодіють між собою за допомогою чітко визначених інтерфейсів та методів зв'язку, таких як REST API або передача повідомлень. Набагато простіше врахувати зміни та запити на нові функції, оскільки лише відповідна мікрослужба повинна бути модифікована, підтримуючи послідовність визначень інтерфейсу. Масштабування за допомогою архітектури мікросервісів є простим та ефективним. Замість того, щоб масштабувати всю програму чи продукт, як у випадку монолітних програм, масштабуються та тиражуються лише бажані послуги. Це призводить до економії витрат на хмарні програми, оскільки хмарні обчислювальні ресурси оплачуються відповідно до їх використання. З цих причин архітектура мікросервісів зазвичай називається архітектурою, готовою до хмари [15].

Мікросервісна архітектура має свої обмеження. Архітектура мікросервісу представляє складність в умовах затримки мережі, мережевого зв'язку, балансування навантаження, стійкості до відмов, форматів повідомлень [18]. Розподілений характер архітектури вносить складність під час налагодження та ізоляції помилок [19]. Мережеві комунікації схильні до збоїв, і мікросервіси сильно залежать від мережевого зв'язку для міжмікросервісного зв'язку. Потік інформації між різними службами може становити проблему, оскільки кожна мікрослужба вирішує одну проблему, оскільки вона може стати інформаційними бар'єрами при спробі отримати повний огляд внутрішніх систем.

Мікросервіс, який залежить від іншого мікросервісу, повинен заблокувати себе, доки не буде готовий залежний мікросервіс. Зазвичай мікросервіси мають окремих власників (розробників), а прогалини в спілкуванні між різними розробниками додають додаткової затримки. Перевантажена мережа або погана пропускна здатність можуть мати глибокий вплив на продуктивність продукту, оскільки більша частина

функціональних можливостей залежить від взаємодії між мікросервісами. Виклик у пам’яті в монолітній програмі завжди швидший у порівнянні з архітектурою мікросервісу, яка покладається на мережевий зв’язок. Незважаючи на те, що додавання нових послуг сприймається як більш легке завдання, підтримка різних мікросервісів з різними стеками технологій може бути складною.

З точки зору організаційного управління, в кінцевому підсумку такий підхід вимагає декількох осіб, що володіють спеціальними технологічними знаннями для обслуговування та підтримки різних послуг, що може збільшити вартість. Модульне тестування та індивідуальне тестування рівня сервісу можна проводити комплексно з цією архітектурою. Складність побудови та підтримки інтеграції та тестових наборів систем зростає із збільшенням кількості послуг. У міру збільшення кількості служб у системі стан системи також ділиться між кількома окремими службами. Усі ці служби повинні працювати синхронно, щоб підтримувати колективний стан системи. Це призводить до більшої кількості помилок, оскільки одна неналежна послуга може порушити цикл роботи всієї програми.

1.1.2 Засоби оркестрації контейнерів

Контейнер Docker ідеально підходить для впровадження мікропослуг. Docker дозволяє запускати програми ізольовано в контейнері, фактично кожна служба може запускатися всередині окремого контейнера. Контейнери можуть бути на одній машині або на різних машинах. Docker має кілька функцій, таких як сховище зображень, що дозволяє надійно оновлювати та переходити на інші версії програми. Управління контейнерами Docker полегшується клієнтом Docker. Клієнт Docker може спілкуватися з декількома демонами і дозволяє користувачам контролювати та керувати контейнерами на хостах [20]. Docker вирішує більшість проблем, з якими стикається архітектура мікросервісів, таких як:

- автоматизація: створення та запуск контейнерів Docker піддається сценаріям, що сприяє автоматизації;
- незалежність: контейнери Docker автономні разом із програмою та необхідним середовищем виконання. Це дає можливість розробникам вибирати будь-який стек технологій, який вони вважають доцільними для бажаної функціональності;
- портативність: контейнери Docker є портативними та дозволяють тестувати окремі мікросервіси. Контейнери можна легко перенести на різні машини та середовища, це гарантує однакові середовища розробки та виконання;
- використання ресурсів: контейнери Docker містять мінімальний набір артефактів, необхідних для запуску програми, одночасно використовуючи ядро та інші ресурси операційної системи хоста з іншими контейнерами. Це максимізує утилізацію ресурсів;
- підтримка та популярність: докери широко застосовуються та підтримуються на різних платформах і це дозволяє командам розробників вибрати технологію, яка, на їх думку, найкраще підходить для даної мікросервісу, та розпочати її розробку.

Докер має кілька вбудованих інструментів з відкритим кодом, що сприяють швидкому розгортанню, автоматизації та простоті обслуговування мікросервісів, таких як реєстр Docker, Dockerfile та клієнт Docker. Файл Docker забезпечує автоматизацію та легкість розгортання мікросервісу з його зображенням, керованим реєстром Docker [55]. Клієнт Docker використовується для зв'язку та управління мікросервісом (контейнером) і дозволяє реєструвати та контролювати послугу. Контейнери Docker забезпечують швидке автоматичне розгортання та тестування програми. Автоматизоване розгортання та тестування є основними елементами сучасного конвеєру CI / CD. Таким чином, контейнери Docker стали технологією, що сприяє сучасним CI / CD. Популярні інструменти CI / CD, такі як Jenkins1 та Gitlab2,

підтримують Docker. Ці функції зробили контейнери Docker популярними та добре підходять для реалізації архітектури мікросервісу.

Контейнери Docker стали стандартним способом створення сучасних масштабованих програм. Однак архітектура мікросервісу вимагає збільшення кількості контейнерів для зростаючої кількості мікросервісів. Це породжує декілька викликів при керуванні контейнерами. Розподілений характер контейнерів ще більше ускладнює управління ними. Розгортання цих розподілених мікросервісів вимагає багатьох кроків ручного втручання системних адміністраторів. Автоматизація цих кроків призводить до довгих сценаріїв, які ускладнюються і важко підтримуються. Таким чином, зростає потреба у структурах та інструментах для управління контейнерами. Інструменти та фреймворки управління контейнерами відповідають наведеним нижче вимогам:

- автоматизоване розгортання контейнерів як у тестовому, так і у виробничому середовищі;
- масштабування та зменшення масштабу контейнерів має виконуватись згідно з шаблонами руху;
- гнучкість: контейнери повинні бути еластичними та завжди доступними, щоб програма могла працювати належним чином.

Фреймворки оркестрації контейнерів надають багато функцій, включаючи надання хостів, створення екземплярів контейнерів, зупинку контейнерів, перепланування невдалих контейнерів, зв’язування контейнерів через їх узгоджені інтерфейси, забезпечення стійкості, безпечне виставлення контейнерів поза їх кластером, масштабування контейнерів, оновлення зображень у контейнерах. Kubernetes, Docker Swarm та Nomad – найпопулярніші фреймворки організації контейнерів, доступні зараз.

1.2 Віртуалізація як підхід до розгортання застосунку

Віртуалізація – це методологія спільного використання та розподілу ресурсів комп’ютера на декілька середовищ виконання шляхом застосування однієї або декількох технологій, таких як розділення апаратного та програмного забезпечення, розподіл часу, часткове або повне моделювання машини, емуляція, якість обслуговування та багато інших [1]. В сучасній розробці надається перевага використанню віртуалізації системи в якості розгортання програмного забезпечення. Це дозволяє постачальникам програмного забезпечення та послуг обслуговувати безліч клієнтів фізичними та віртуальними ресурсами на вимогу.

За допомогою віртуалізації системи віртуалізовані ресурси можна масштабувати або зменшувати для того, щоб задовольнити збільшення попиту. Коли попит зменшується, віртуалізовані ресурси можна вилучити (або зменшити). Плавне масштабування дає змогу ресурсам бути безкоштовними та доступними для інших цілей, а також допомагає підтримувати низькі витрати на інфраструктуру для компанії.

Віртуалізація забезпечує однакове середовище розробки та виробництва програмного забезпечення. Це дозволяє запускати кілька віртуальних машин на одному серверному обладнанні. Машини працюють повністю ізольовано та забезпечують кращу масштабованість та використання базових апаратних ресурсів. Найпопулярнішими технологіями віртуалізації є віртуалізація на основі гіпервізора та віртуалізація на рівні операційної системи.

Віртуалізація на рівні операційної системи, також відома як контейнеризація або контейнерна віртуалізація, є легкою альтернативою віртуалізації на основі гіпервізора. Гіпервізор не задіяний, а віртуалізація здійснюється на рівні операційної системи (ОС). Усі віртуалізовані екземпляри, тобто контейнери, використовують спільне ядро ОС. З точки зору користувача, контейнери дають досвід роботи окремих операційних систем [2]. Оскільки немає задіяного гіпервізора, це значною мірою зменшує накладні витрати

на виконання, а спільне використання тієї самої операційної системи також зменшує накладні витрати на зберігання. Віртуалізація на основі контейнерів має слабку ізоляцію порівняно з рішеннями для віртуалізації на основі гіпервізора.

Як показано на малюнку 1.2, контейнери забезпечують рівень абстракції над ядром операційної системи хоста, що дозволяє кожному контейнеру поводитися як незалежна операційна система з ізоляцією.

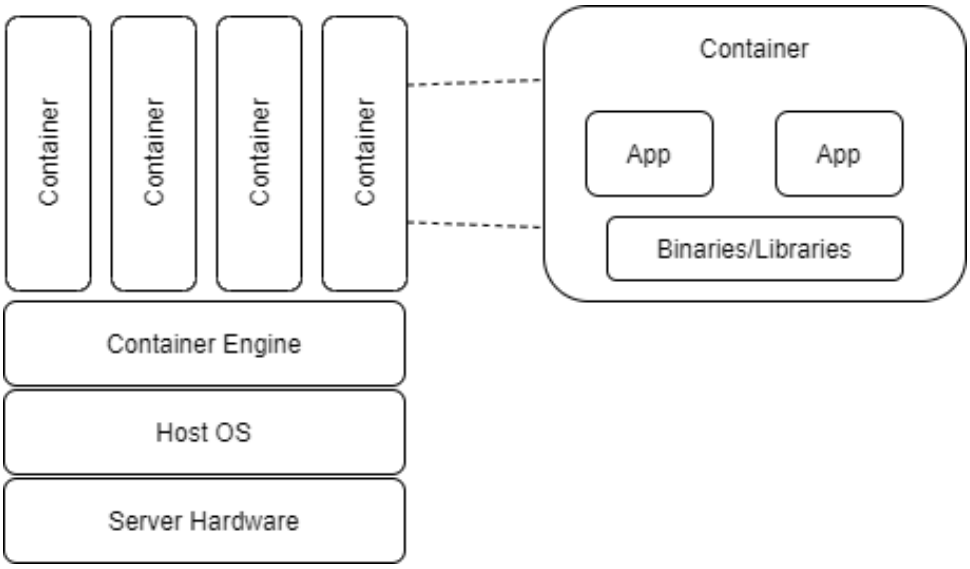

Рисунок 1.2 – Архітектура віртуалізації на основі контейнерів

Контейнери створюються на основі зображення. Зображення контейнера – це самостійний виконуваний пакет, який є легким і містить усі необхідні артефакти, необхідні для запуску контейнера, такі як: код, час виконання, системні ресурси. Цей пакет можна резервно копіювати та розповсюджувати. Кілька екземплярів контейнерів можуть бути виконані в одній і тій самій машині-хості або на різних машинах-хостах, залежно від вимог щодо відмовостійкості.

З точки зору користувача, контейнери створюють ілюзію таким чином, що процеси, що виконуються всередині них, працюють на різних фізичних машинах. Ядра Linux переважають у підтримці віртуалізації на основі контейнерів, а популярні рішення для віртуалізації на основі контейнерів, такі як Docker, покладаються на надані

ними функції. Інтерфейс управління, наданий ядром Linux, дозволяє контролювати, виконувати та адмініструвати контейнери. Ізоляція досягається за допомогою просторів імен, що дозволяє кожному контейнеру мати різний вигляд базової системи. Управління ресурсами, такими як центральний процесор, пам’ять та введення / виведення, управляється та контролюється за допомогою контрольних груп (cgroups).

1.3 Огляд компонентів системи оркестрації Kubernetes

Kubernetes – це розширювана платформа з відкритим кодом для управління робочими навантаженнями та послугами в контейнерах. Він підтримує як декларативну конфігурацію, так і автоматизацію [3]. Kubernetes побудований на основі перевіреної часом внутрішньої системи управління кластерами Google під назвою Borg. Система Borg використовується Google для запуску контейнерів у своїх центрах обробки даних вже більше десяти років. Це система управління кластерами, яка може запускати сотні тисяч завдань з різних додатків у багатьох кластерах. Kubernetes було представлено у 2014 році і з тих пір є найпопулярнішою системою управління та організації контейнерів.

Філософія дизайну Kubernetes полягає у створенні екосистеми з різних компонентів та інструментів, які дозволять легко розгортати, масштабувати та керувати програмами. Kubernetes дозволяє легко переносити програми в різних середовищах та забезпечує платформу управління, орієнтовану на контейнери.

1.3.1 Архітектура Kubernetes

На рисунку 1.3 показана архітектура високого рівня Kubernetes. Детально її компоненти будуть розглядатися далі.

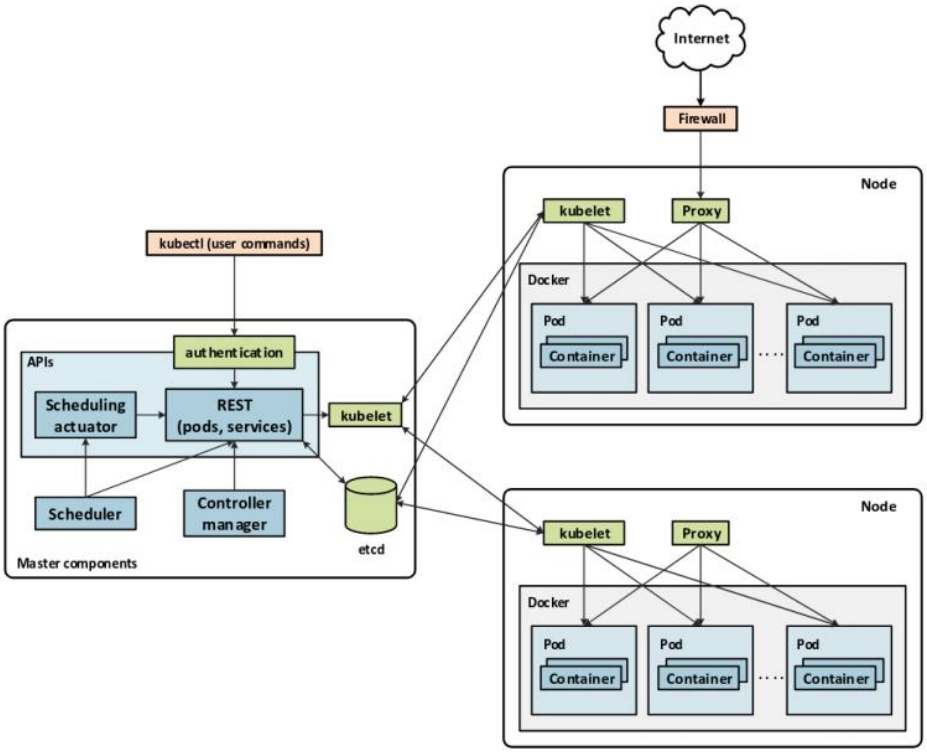

Рисунок 1.3 – Архітектура Kubernetes

Pod − це найменший і найпростіший блок, який можна створити та розгорнути в об'єктній моделі Kubernetes. Поди вважаються основними будівельними елементами Kubernetes. Простіше кажучи, под представляє запущений процес на кластері. Под може містити один контейнер або невеликий набір контейнерів, які мають спільні загальні ресурси, такі як мережа та сховище. Незважаючи на те, що найпоширенішим випадком використання є один контейнер на под, можуть існувати вимоги до кількох контейнерів, які повинні бути щільно з'єднані в одному поді. Контейнери всередині одного блоку завжди розташовуються на одному хості або на віртуальній машині. Контейнери всередині модуля взаємодіють між собою за допомогою localhost і спілкуються за межами модуля, використовуючи спільні мережеві ресурси, такі як порти. Кожному поду присвоюється унікальна IP-адреса, яка використовується для зв'язку із зовнішнім світом та іншими модулями. Kubernetes завжди планує та

організовує поди, а не контейнери. Отже, коли додатку потрібно масштабуватися, це означає додавання більше подів. Додавання подів називається "реплікацією" у Kubernetes. Поди є ефемерними сутностями, і їх можна створювати та знищувати за бажанням. Поди можуть бути припинені автоматично з різних причин, таких як відсутність ресурсів, тому важливо зберегти відсутність стану в них. Самостійно поди не працюють, швидше це середовище, де працюють контейнери.

Вузол, також відомий як minion, може бути фізичною або віртуальною машиною. Вузли забезпечують середовище виконання для подів і управляються головними компонентами. Важливими компонентами вузла є кубелет і кубе-проксі.

Kubelet – відомий контролер Kubernetes і керує рівнем виконання контейнера. Він реалізує API pod і node. Kubelet бере набір PodSpecs (представлення YAML або JSON, що описує pod) і гарантує, що контейнери, зазначені в специфікації PodSpec, працюють без будь-яких проблем. Слід зазначити, що kubelet не управляє контейнерами, створеними поза середовищем Kubernetes. Основні обов'язки Kubelet включають створення, постійний моніторинг та бережне завершення роботи контейнерів, що працюють на вузлі, та звітування про стан вузлів у кластер. PodSpec зазвичай надається Kubelet за допомогою сервера API. Існують три різні варіанти, крім сервера API, для надання PodSpec кубелету: шлях до файлу з командного рядка, кінцева точка HTTP як параметр у командному рядку, і нарешті, Kubelet може слухати запити HTTP та запитувати PodSpec. Інтервали оновлення для запиту на новий PodSpec можна налаштувати з командного рядка.

Kube-proxy запускається на кожному вузлі та програмує таблиці IP. Це дозволяє перенаправляти сервісний трафік у правильні серверні бази. Це додатково забезпечує високодоступне рішення для балансування навантаження та має низькі експлуатаційні витрати. Наприклад, запит на послугу, що надходить з вузла, найкраще обслуговується в тому самому вузлі. В основному він відображає окремі контейнери до однієї служби. Kube-proxy працює на рівні транспортного рівня і може виконувати просту

переадресацію потоків TCP або UDP. Він також може виконувати переадресацію на основі кругового руху, щоб досягти балансування навантаження. Кінцеві точки служби вирішуються та виявляються за допомогою DNS.

Мастер компоненти служать площиною управління кластером Kubernetes. Кластер відноситься до колекції вузлів і майстрів. Окремі мастер компоненти можна запускати на будь-якій машині кластера. Для простоти управління кластером основні мастер запускаються в одній машині. Контейнери користувачів не планується запускати на головних машинах. Kube-apiserver та ін .dd, kube-планувальник, kube-контролер-менеджер – це деякі з основних компонентів [4].

Контролер реплікації відповідає за те, щоб один под або набір подів були готовими і завжди доступними. Набір подів називається реплікою. Для масштабованості потрібні репліки, тобто під час горизонтального масштабування додається більше подів для обробки трафіку, а для високої доступності потрібно більше однієї репліки. ReplicationController здійснює нагляд за кількома подами через кілька вузлів. Поди, які контролюються та підтримуються контролером реплікації, автоматично замінюються новими, якщо деякі поди виходять з ладу.

Поди можуть бути видалені або припинені з різних причин. Це один із сильних переконливих факторів використання контролера реплікації, навіть якщо програма вимагає лише одного пода. Контролер реплікації використовує шаблони модулів (що містять набір правил) для управління модулями. Контролер реплікації також відповідає за видалення надлишкових подів, які можуть бути присутніми в кластері. Контролер може бути припинений разом із подами, якими він управляє, або контролер може бути припинений без припинення подів, і поди можуть бути від’єднанні від контролера.

Послуги − це абстракція, яка визначає логічний набір подів та політик доступу до них. Як видно раніше, поди є ефемерними, і контролер реплікації може видалити або замінити їх у будь-який момент часу. Таким чином, IP-адреса, пов’язана з модулем, не є фіксованою. Ці фактори ускладнюють спілкування зі подами. Ця проблема

вирішується за допомогою абстракції служби, де підгрупи, що надають однакові послуги, групуються за допомогою селектора міток. У Kubernetes служби є REST-об'єктами, такими як поди, і RESTful-операції можуть виконуватися над ними. І TCP, і UDP підтримуються протоколами для служб, для яких за замовчуванням використовується TCP. Об'єкт служби оновлюється, коли пов'язані поди змінюються. Об'єкт кінцевої точки служби створюється, коли служба створюється в Kubernetes і містить деталі модулів. Головною метою послуги є забезпечення постійного доступу користувачів до подів.

Аддони − це стручки та служби, що реалізують функції кластера [4]. Вони розширюють функціональність Kubernetes. Є кілька додатків, доступних для таких функцій, як: мережеві зв'язки та визначення мережевої політики, виявлення послуг, а також візуалізація та контроль. WeaveScope – це один з таких аддонів для графічної візуалізації контейнерів, скриньок, служб та інших об'єктів Kubernetes. Існує також аддон Dashboard, який забезпечує веб-інтерфейс для візуалізації та моніторингу кластера Kubernetes.

1.3.2 Вбудоване сховище пода Kubernetes

Управління сховищем – ще одна проблема, відмінна від керування обчислювальними екземплярами. Підсистема PersistentVolume надає API для користувачів та адміністраторів, який абстрагує детальну інформацію про те, як надається сховище та як воно споживається. Для цього ми представляємо два нові ресурси API: PersistentVolume та PersistentVolumeClaim.

PersistentVolume (PV) – це частина сховища в кластері, яка була надана адміністратором або динамічно надана за допомогою класів зберігання. Це ресурс у кластері так само, як вузол. PV – це плагіни для обсягу, як Volumes, але він має життєвий цикл, незалежний від будь-якого окремого Pod, який використовує PV. Цей

об'єкт API фіксує деталі реалізації сховища, будь то NFS, iSCSI або система зберігання даних для хмарного постачальника.

PersistentVolumeClaim (PVC) – це запит користувача на зберігання. Він схожий на под. Поди споживають ресурси вузлів, а PVC - ресурси PV. Поди можуть запитувати конкретні рівні ресурсів (ЦП та пам’ять). Claim можуть вимагати певного розміру та режимів доступу (наприклад, їх можна встановити ReadWriteOnce, ReadOnlyMany або ReadWriteMany)

Незважаючи на те, що PersistentVolumeClaims дозволяють користувачеві споживати абстрактні ресурси зберігання, зазвичай користувачі потребують PersistentVolumes з різними властивостями, такими як продуктивність для різних проблем. Адміністратори кластерів повинні мати можливість пропонувати різноманітні PersistentVolumes, які відрізняються не лише розмірами та режимами доступу, не показуючи користувачам подробиць того, як ці томи реалізовані. Для цих потреб існує ресурс StorageClass.

PV – це ресурси кластера. PVC – це запити на ці ресурси, які також виконують функцію перевірки заявок на ресурс. Існує два способи забезпечення PV: статичний чи динамічний.

- Статичний − адміністратор кластера створює ряд PV. Вони містять деталі реального сховища, яке доступне для використання користувачами кластера. Вони існують в API Kubernetes і доступні для споживання.
- Динамічний − коли жоден із статичних PV, створених адміністратором, не відповідає користувачеві PersistentVolumeClaim, кластер може спробувати динамічно забезпечити том спеціально для PVC. Це забезпечення базується на StorageClasses: PVC повинен запитувати клас зберігання, і адміністратор повинен створити та налаштувати цей клас для динамічного забезпечення. Претензії, які запитують клас, фактично вимикають для себе динамічне забезпечення.

Щоб увімкнути динамічне забезпечення зберігання на основі класу сховища, адміністратору кластера потрібно ввімкнути контролер прийому DefaultStorageClass на сервері API. Це можна зробити, наприклад, переконавшись, що DefaultStorageClass знаходиться серед упорядкованого списку значень, розділених комами, для прапора --enable-admission-plugins компонента сервера API. Для отримання додаткової інформації про прапорці командного рядка сервера API перевірте документацію kube-apiserver.

Користувач створює або у випадку динамічного забезпечення вже створив PersistentVolumeClaim із певним обсягом запитуваного сховища та з певними режимами доступу. Цикл управління в майстрі відстежує нові PVC, знаходить відповідний PV (якщо це можливо) і пов'язує їх між собою. Якщо PV було динамічно передбачено для нового PVC, нода завжди буде прив’язувати цей PV до PVC. В іншому випадку користувач завжди отримає принаймні те, про що просив, але обсяг може перевищувати запитуваний. Після прив’язки PersistentVolumeClaim є ексклюзивними, незалежно від того, як вони були пов’язані. Прив’язка PVC до PV − це співвідношення один до одного із використанням ClaimRef, яке є двонаправленим прив’язуванням між PersistentVolume та PersistentVolumeClaim.

Клейми залишатимуться незв’язаними на невизначений час, якщо відповідний обсяг не існує. Клейми будуть пов’язані у міру появи відповідних обсягів. Наприклад, кластер, забезпечений багатьма PV по 50Gi, не відповідав би PVC, що запитував 100Gi. PVC можна зв’язати, коли до кластера додано 100Gi PV.

Кластер перевіряє клейм, щоб знайти пов'язаний том і змонтує цей том для Pod. Для томів, які підтримують режими багаторазового доступу, користувач вказує, який режим бажаний, використовуючи свій клейм як том у Pod.

Як тільки користувач має клейм, і цей клейм зв’язаний, пов’язаний PV належить користувачеві стільки часу, скільки йому це потрібно. Користувачі планують Pod і

отримують доступ до заявлених PV, включаючи розділ persistentVolumeClaim в блок томів Pod.

Призначення функції «Захист об’єкта зберігання у використанні» полягає у тому, щоб PersistentVolumeClaims (PVCs), які активно використовуються Pod і PersistentVolume (PVs), пов’язані з PVCs, не вилучалися з системи, оскільки це може призвести до втрати даних. Якщо користувач видаляє PVC під активним використанням пода, PVC не видаляється негайно. Видалення PVC відкладається до тих пір, поки PVC більше не буде активно використовуватися жодними подами. Крім того, якщо адміністратор видаляє PV, прив’язану до PVC, PV не видаляється негайно. Видалення PV відкладається до тих пір, поки PV більше не буде прив’язаний до PVC. Можна бачити, що PVC захищений, коли статус PVC термінується, а список фіналізаторів включає kubernetes.io/pvc-protection (Рисунок 1.4):

```
kubectl describe pvc hostpath
Name:          hostpath
Namespace:     default
StorageClass:  example-hostpath
Status:        Terminating
Volume:
Labels:        <none>
Annotations:   volume.beta.kubernetes.io/storage-class=example-hostpath
               volume.beta.kubernetes.io/storage-provisioner=example.com/hostpath
Finalizers:    [kubernetes.io/pvc-protection]
...
```

Рисунок 1.4 – Процес термінації статусу PVC

Можна бачити, що PV захищений, коли статус PV термінується, а список фіналізаторів включає kubernetes.io/pv-protection (Рисунок 1.5):

```
kubectl describe pv task-pv-volume
Name:            task-pv-volume
Labels:          type=local
Annotations:     <none>
Finalizers:      [kubernetes.io/pv-protection]
StorageClass:    standard
Status:          Terminating
Claim:
Reclaim Policy:  Delete
Access Modes:    RWO
Capacity:        1Gi
Message:
Source:
    Type:          HostPath (bare host directory volume)
    Path:          /tmp/data
    HostPathType:
Events:            <none>
```

Рисунок 1.5 – Процес термінації статусу PV

Площина управління може прив’язати PersistentVolumeClaims до відповідних PersistentVolumes у кластері. Однак, якщо є необхідність, щоб PVC прив’язувався до певного PV, потрібно попередньо зв’язати їх. Вказуючи PersistentVolume у PersistentVolumeClaim, оголошується прив'язка між цим конкретним PV та PVC. Якщо PersistentVolume існує і не зарезервував PersistentVolumeClaims через своє поле requestRef, тоді PersistentVolume і PersistentVolumeClaim будуть пов'язані.

Прив'язка відбувається незалежно від деяких критеріїв відповідності обсягу, включаючи спорідненість вузлів. Площина управління все ще перевіряє, чи клас зберігання, режими доступу та запитаний розмір зберігання є дійсними. Підтримка розширення PersistentVolumeClaims (PVCs) тепер увімкнена за замовчуванням. Можна розширити такі типи томів:

- gcePersistentDisk,
- awsElasticBlockStore,
- Cinder,
- Glusterfs,

- Rbd,
- Azure File,
- Azure Disk,
- Portworx,
- FlexVolumes,
- CSI.

Розширити PVC можна лише в тому випадку, якщо для поля класу зберігання allowVolumeExpansion встановлено значення true (Рисунок 1.6).

```yaml
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: gluster-vol-default
provisioner: kubernetes.io/glusterfs
parameters:
  resturl: "http://192.168.10.100:8080"
  restuser: ""
  secretNamespace: ""
  secretName: ""
allowVolumeExpansion: true
```

Рисунок 1.6 – Приклад конфігурації PVC

Щоб подати запит на більший обсяг PVC, потрібно відредагувати об’єкт PVC та вказати більший розмір. Це викликає розширення тому, який підтримує базовий PersistentVolume. Новий PersistentVolume ніколи не створюється для задоволення клейму. Натомість змінюється розмір існуючого тому.

Підтримка розширення томів CSI ввімкнена за замовчуванням, але для підтримки розширення тому потрібен також певний драйвер CSI. Коли том містить файлову систему, розмір файлової системи змінюється лише тоді, коли новий Pod використовує PersistentVolumeClaim у режимі ReadWrite. Розширення файлової

системи здійснюється або під час запуску Pod, або під запуском Pod, і основна файлова система підтримує онлайн-розширення.

У цьому випадку не потрібно видаляти та відтворювати Pod або розгортання, що використовує наявний PVC. Будь-який використовуваний PVC автоматично стає доступним для його Pod, як тільки його файлова система буде розширена. Ця функція не впливає на PVC, які не використовуються модулем або розгортанням. Потрібно створити блок, який використовує PVC, перш ніж розширення буде завершено.

Кожен PV містить специфікацію та статус, що є специфікацією та статусом тому. Ім'я об'єкта PersistentVolume має бути дійсним іменем субдомену DNS (Рисунок 1.7).

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0003
spec:
  capacity:
    storage: 5Gi
  volumeMode: Filesystem
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Recycle
  storageClassName: slow
  mountOptions:
    - hard
    - nfsvers=4.1
  nfs:
    path: /tmp
    server: 172.17.0.2
```

Рисунок 1.7 – Приклад конфігурації імені PV

1.4 Огляд існуючих рішень

1.4.1 Аналіз Velero

Velero (раніше Heptio Ark) надає інструменти для резервного копіювання та відновлення ресурсів кластера Kubernetes та постійних томів та є рішенням з відкритою

кодовою базою [5]. Можна запустити Velero на загальнодоступній хмарній платформі або локально. Velero дозволяє:

- робити резервні копії кластера та відновити їх у разі втрати;
- перенести ресурси кластера на інші кластери;
- застосовувати виробничий кластер до кластерів для розробки та тестування.

Velero складається з:

- сервера, який працює на вашому кластері;
- клієнта командного рядка, який працює локально

Кожна операція Velero – резервне копіювання за необхідністю, резервне копіювання за розкладом, відновлення – це власний ресурс, який визначається за допомогою спеціального визначення ресурсів Kubernetes (CRD) і зберігається в etcd. Velero також включає контролери, які обробляють власні ресурси для виконання резервних копій, відновлення та всіх пов’язаних операцій.

Можна створити резервну копію або відновити всі об’єкти у кластері, або можна відфільтрувати об’єкти за типом, простором імен та / або міткою. Velero підходить для використання у випадку аварійного відновлення, а також для створення знімків стану вашого додатка перед виконанням системних операцій на кластері, таких як оновлення.

Операція резервного копіювання складається з наступних кроків:

- завантаження тарбалу скопійованих об’єктів Kubernetes у сховище хмарних об’єктів;
- виклик API постачальника хмарних послуг, щоб зробити знімки дисків постійних томів, якщо вказано.

За бажанням можна вказати хуки резервного копіювання, які будуть виконуватися під час резервного копіювання. Наприклад, вам може знадобитися сказати базі даних зберегти свої буфери в пам'яті на диск, перш ніж робити знімок.

Варто зауважити, що резервне копіювання кластерів не є суворо атомарним. Якщо об’єкти Kubernetes створюються або редагуються під час створення резервної

копії, вони можуть не бути включені до резервної копії. Шанси на отримання суперечливої інформації низькі, але це можливо.

Операція резервного копіювання за розкладом дозволяє створювати резервні копії даних через періодичні інтервали. Перше резервне копіювання виконується, коли розклад створюється вперше, а наступні резервні копії відбуваються через зазначений інтервал розкладу. Ці інтервали задаються виразом Cron.

Заплановані резервні копії зберігаються з назвою <НАЗВА ГРАФІКУ> - <TIMESTAMP>, де <TIMESTAMP> має формат YYYYMMDDhhmmss.

Операція відновлення дозволяє відновити всі об'єкти та постійні томи з раніше створеної резервної копії. Також можна відновити лише відфільтровану підмножину об'єктів та постійні томи. Velero підтримує повторне відображення простору імен - наприклад, за одне відновлення об’єкти у просторі імен “abc” можуть бути відтворені під простором імен “def”, а об’єкти у просторі імен “123” під “456”.

Ім’я відновлення за замовчуванням - <РЕЗЕРВНА ІМЯ> - <TIMESTAMP>, де <TIMESTAMP> має формат YYYYMMDDhhmmss. Також можна вказати власне ім'я. Відновлений об'єкт також включає мітку з ключем velero.io/restore-name та значенням <RESTORE NAME>.

За замовчуванням резервні місця зберігання створюються в режимі читання-запису. Однак під час відновлення можна налаштувати місце зберігання резервної копії в режимі лише для читання, що вимикає створення та видалення резервної копії для місця зберігання. Це корисно для того, щоб під час сценарію відновлення не створювались та не видалялись випадково резервні копії.

За бажанням можна вказати хуки відновлення, які будуть виконуватися під час відновлення або після відновлення ресурсів. Наприклад, може знадобитися виконати спеціальну операцію відновлення бази даних перед запуском контейнерів додатків бази даних.

Щоб переглянути структуру резервної копії, потрібно завантажити її з бакета в S3 або ввести команду Heptio Ark (Рисунок 1.8):

```
$ ark backup download nginx-simple
Backup nginx-simple has been successfully downloaded to $PWD/nginx-simple-data.tar.gz
```

Рисунок 1.8 – Команда для перегляду резервної копії

Після завантаження резервної копії локально, бачимо наступну структуру, наведену на Рисунок 1.9:

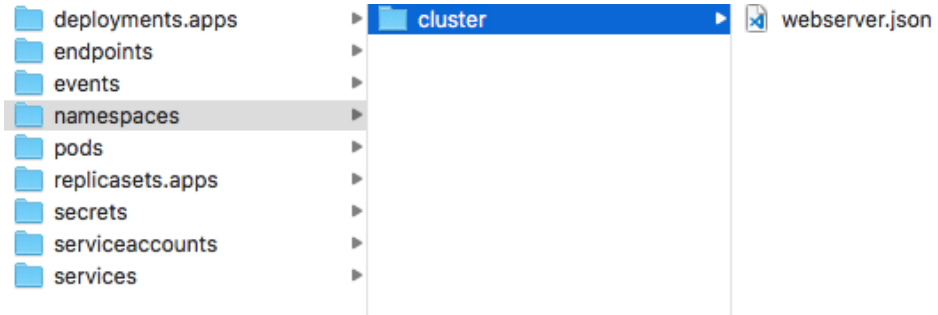

Рисунок 1.9 – Структура викачаного архіву Heptio Ark

Якщо не потрібно повне відновлення, можна відновити лише частину за допомогою команд Heptio Ark або перейти до бекапу напряму і відновити цю частину через kubectl.

Heptio Ark може виконувати заплановані завдання. Для цього потрібно знайти, які ресурси та простори потрібно включити в резервну копію або виключити з неї та коли потрібно виконувати бекап (Рисунок 1.10):

```
$ ark schedule create nginx-schedule --schedule="* 10 * * *" --include-namespaces webserver
Schedule "nginx-schedule" created successfully.
```

Рисунок 1.10 – Команда для створення запланованої задачі Heptio Ark

У цьому випадку резервна копія створюватиметься кожного дня через 10 годин і включатиме лише простір веб-сервера. У Heptio Ark CLI ми бачимо, що створено розклад і Heptio Ark вже створили перший бекап, результат якого наведено на Рисунку 1.11:

```
$ ark schedule get
NAME             STATUS    CREATED                          SCHEDULE       BACKUP TTL   LAST BA
CKUP   SELECTOR
nginx-schedule   Enabled   2018-07-08 17:49:00 +0200 CEST   * 10 * * *     720h0m0s     25s ago
<none>

$ ~/Downloads/ark backup get
NAME                            STATUS      CREATED                          EXPIRES   SELECTOR
nginx-schedule-20180708154900   Completed   2018-07-08 17:49:00 +0200 CEST   29d       <none>
nginx-simple                    Completed   2018-07-08 17:35:09 +0200 CEST   29d       <none>
```

Рисунок 1.11 – Результат першого створеного бекапу

У даному випадку вказано, що планові бекапи видаляються протягом 720 годин, тобто через 30 днів. Коли створюєте бекап або розклад, можна вказати час життя копії - TTL. Після цього періоду резервна копія буде видалена зі сховищ, наприклад, з сховища AWS.

1.4.2 Аналіз Handy Backup

Handy Backup – професійне програмне рішення, що дозволяє здійснювати резервне копіювання бази даних у автоматичному режимі [6]. Також програма дозволяє зберігати резервні копії інших типових даних. Для бекапу баз різних типів Handy Backup пропонує спеціалізовані плагіни, які ефективно створюють резервні копії BD та забезпечують їх зберігання на будь-яких сучасних носіях.

Програмне рішення представляє можливість бекапу наступних типових сховищ даних:

- Oracle;
- MySQL;
- 1C;
- MSSQL;
- PostgreSQL;
- MariaDB та ін.

Створення резервної копії баз даних за допомогою ODBC можливо в будь-якому СУБД, який у системі має відповідний драйвер. Для бекапу баз використовується універсальна плагінова база даних, яка дозволяє працювати як з SQL, так і з не-SQL орієнтованими СУБД:

- Бекап FoxPro – для збереження копій даних із популярної десктопної БД.
- FireBird бекап – створення бекапів для даних СУБД Firebird або InterBase.
- Резервне копіювання Access – створення резервних копій популярної СУБД.

Існує два різні методи резервного копіювання бази даних під управлінням будь-якого сучасного СУБД:

- Холодний бекап баз вимагає зупинки СУБД та всіх транзакцій.
- Горячий бекап баз – пряме копіювання даних без зупинки СУБД.

Усі плагіни для резервного копіювання баз даних у Handy Backup передбачають можливість гарячого бекапа. Програма може бути використана для гарячого копіювання баз даних як на фізичних, так і на віртуальних машинах.

Реплікація дозволяє розповсюджувати навантаження на базу даних між кількома серверами для створення декількох логічно пов'язаних копій баз даних. Хоча реплікація і кластеризація підвищують стабільність роботи БД, регулярний бекап вони не замінять. Зміни в базах даних, викликані їх пошкодженням, можуть зіпсувати всі дзеркала або кластери.

Існують два рішення Handy Backup, орієнтовані насамперед на комерційного користувача. Обидва цих рішення забезпечуються "з коробки" повним комплектом інструментів для роботи з базами даних різних типів та є платними.

1.4.3 Аналіз APBackup

APBackup − це програма для організації автоматичного резервного копіювання (бекапу), яка дозволить вам копіювати дані як на локальний диск, так і на мережевий ресурс і навіть на FTP / FTPS сервер. Програма може як просто копіювати дані, так і зберігати їх у форматі ZIP64. Так само програма дозволяє підключити для виконання завдання будь-який зовнішній архіватор [7].

Важливі переваги APBackup:

- вивантаження створених архівів на ftp-сервер,
- шифрування архівів ключем 256 біт (це означає, що ніхто крім вас не зможе скористатися резервними копіями),
- низька ціна (для корпоративного сегмента дуже незначна).

Схема архівування баз даних на сервері Windows Server виглядає складається з наступних кроків:

- створення бекапів sql-баз даних;
- створення за розкладом зашифрованого архіву з sql-бекапу і файловими базами даних;
- вивантаження архіву на ftp-сервер.

APBackUp має простий інтерфейс, Українська підтримка реалізована, рішення може повністю автоматично працювати за розкладом, потрібно тільки один раз налаштувати всі параметри. Архіви будуть зберігатись там, де вкаже користувач, наприклад CD диск, локальний диск, інший комп'ютер, FTP сервер. На рисунку 1.12 наведено інтерфейс створення автоматичної задачі:

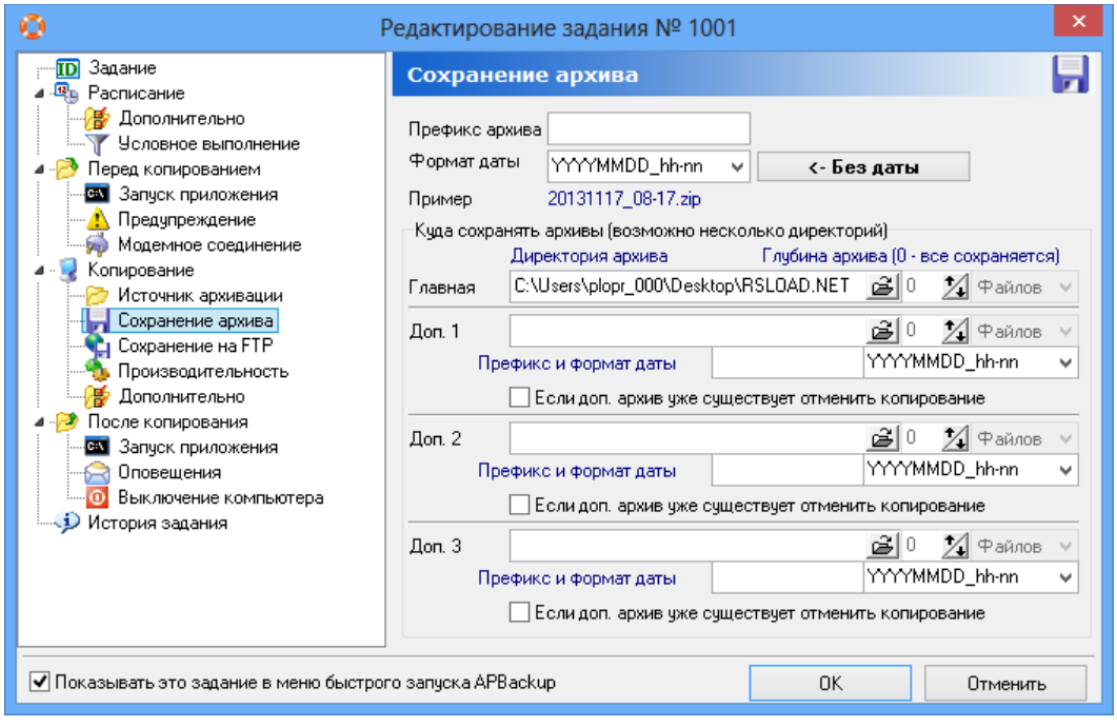

Рисунок 1.12 – Інтерфейс створення автоматичної задачі APBackup

Далі перераховані функціональні можливості APBackup:

- повністю автоматична робота програми за розкладом, не вимагає участі користувача;
- режими роботи: копіювання, архівація, синхронізація, копіювання на FTP сервер;
- архіви зберігаються в стандартному ZIP форматі (і новому ZIP64 форматі для архівів більше за 2 Гб). Крім того, програма дозволяє підключати будь-наявний у вас архіватор;
- встановлення пароля на ZIP архів;
- можливість перевірки архіву після архівації;
- можливість використання FTPS (FTP over TSL / SSL) протоколу для передачі архіву на FTP сервер;

- місцем збереження архіву може служити як локальний диск, CD-R, так і інший комп'ютер в локальній мережі, а так само FTP сервер, причому всі ці пункти призначення ви можете використовувати одночасно, дублюючи архів;
- можливість моніторингу каталогу і запуск завдання при зміні каталогу;
- адміністратор може отримувати листи від програми про хід виконання завдань;
- програма може зберігати задане число останніх архівів;
- широкі можливості по регулюванню навантаження на процесор – ви можете працювати на комп'ютері і практично не помічати, що в цей час APBackUp зберігає ваші дані.

Для використання резервного копіювання та відновлення потрібно, щоб у плану служби додатків був рівень Стандартний, Преміум або Ізольований. Рівні Преміум і Ізольований дозволяють створювати більшу кількість щоденних резервних копій в порівнянні з рівнем Стандартний.

Після створення однієї або декількох резервних копій додатка вони відображаються на сторінці Контейнери вашого профілю зберігання, як і ваш додаток. У облікового запису зберігання кожна резервна копія складається з файлу .zip з резервною копією даних і файлу .xml з маніфестом для вмісту файлу .zip. Ці файли можна розпакувати і переглянути, якщо необхідно отримати доступ до резервних копій без фактичного відновлення оригінальну програму.

Резервна копія бази даних програми зберігається в корені ZIP-файлу. Для бази даних SQL це BACPAC-файл (без розширення файлу), який можна імпортувати. Відомості про створення бази даних в базі даних SQL Azure на основі експорту BACPAC см. В розділі імпорт BACPAC-файлу для створення бази даних в базі даних SQL Azure.

Резервна копія може вміщати до 10 ГБ вмісту додатка і бази даних. Якщо розмір резервної копії перевищить цей ліміт, відобразиться повідомлення про помилку.

Резервні копії бази даних Azure для MySQL з підтримкою TLS не підтримуються. Якщо резервне копіювання налаштоване, резервні копії будуть несправними.

Резервні копії бази даних Azure для PostgreSQL з підтримкою TLS не підтримуються. Якщо резервне копіювання налаштоване, резервні копії будуть несправними.

Бази даних в додатку MySQL автоматично створюють резервні копії без будь-якого налаштування. Резервні копії можуть не працювати належним чином, якщо вручну задаються параметри бази даних в додатку MySQL, такі як додавання рядка підключення.

Можна також обрати існуючий додаток для відновлення резервної копії програми до іншої додаток в тій же групі ресурсів. Перш ніж використовувати цей параметр, необхідно створити іншу програму в групі ресурсів з конфігурацією бази даних, аналогічної конфігурації бази даних в резервної копії програми. Ви можете також створити додаток для відновлення вмісту.

1.5 Висновки

У цьому розділі розглядаються основні компоненти та елементи при розробці систем резервного копіювання даних у Kubernetes кластері.

Результати огляду компонентів Kubernetes та його архітектури, а також представлених на сучасному ринку існуючих програмних рішень буде використано в ході розробки структурних елементів даної системи та її програмної реалізації.

У першій частині розділу було представлено огляд складових мікросервісної архітектури, її порівняння з монолітною архітектурою. Наведено переваги та недоліки мікросервісного підходу проектування та виділено основні тенденції розвитку серед сучасних компаній, на прикладі детальніше описано ключові складові. У підрозділі 1.2

представлено такий підхід, як віртуалізація та як вона пов’язана у сучасному розгортанні мікросервісної архітектури.

У підрозділі 1.3 розглядаються компоненти системи оркестрації Kubernetes, проведено детальний огляд вбудованого сховища пода, методи та конфігурації для керування станами сховища, що стане у нагоді при наступному проектуванні алгоритму поведінки резервного копіювання даного сховища.

Крім того, проведено огляд існуючих рішень у галузі резервного копіювання даних. Було розглянуто такі рішення, як Velero,Handy Backup, APBackup. Два рішення мають корпоративну підписку, а Velero є рішенням з відкритою кодовою базою. Виявили недоліки або частини невирішеного завдання і намітили шлях його власного вирішення.

2 РОЗРОБКА СТРУКТУРНИХ ЕЛЕМЕНТІВ СИСТЕМИ

У даному розділі буде проаналізовано які вимоги висуваються до системи, та буде проведена формалізацію вимог.

2.1 Аналіз вимог до системи

Практичне застосування контейнерів в критично важливих робочих середовищах постійно збільшується, організації все частіше стикаються з питаннями безпеки, резервного копіювання, відновлення та іншими завданнями, що стосуються управління відповідними даними. Спочатку більшість контейнерних додатків не мали здатність зберігати стан, яке дозволяло б розгортати їх в загальнодоступному хмарному середовищі без особливих складнощів. Однак у міру того, як додатки, здатні зберігати стан і вимагають доступу до баз даних, перетворювалися в контейнерні, стала рости потреба в серйозному перегляді підходів, що стосуються резервного копіювання та відновлення. Як правило, переживання IT відділів щодо розгортаються контейнерних середовищ стосуються безпеки даних, втрати даних, планування відновлень після аварійних ситуацій та сталого функціонування бізнесу в цілому.

У контексті використання контейнери для того, щоб створювати нові монолітні додатки, або перетворювати вихідний код попередніх версій додатків, або впроваджувати нові динамічно розподілені додатки – потреба в резервному копіюванні структурованих даних нікуди не зникає. Організаціям необхідно укладати SLA угоди і гарантувати безперервну підтримку сервісів, заснованих на контейнерних середовищах, в справному стані. Таким чином, для ефективної роботи потрібно резервне копіювання і відновлення компонентів кластера і постійних даних навіть при використанні Kubernetes з мінімальною функціональністю.

Після огляду ситуацій на ринку та сучасних тенденцій розвитку необхідно визначити вимоги до програмного продукту. Серед функціональних вимог до процесу створення резервної копії виділено наступні:

1. Система повинна мати можливість створювати резервну копію для всього рішення.
2. Система повинна мати можливість автоматично запускати процедуру резервного копіювання відразу після розгортання.
3. Система повинна надавати адміністратору можливість запускати процедуру резервного копіювання вручну.
4. Система повинна надавати можливість лише адміністратору створювати та виконувати процедуру резервного копіювання.
5. Система повинна мати можливість запускати процедуру резервного копіювання за розкладом.
6. Система повинна надавати можливість адміністратору налаштовувати планувальник для процедури резервного копіювання.
7. Система повинна надавати можливість адміністратору вибирати місце для резервних копій файлів.
8. Система повинна надати можливість адміністратору зберігати файли резервних копій на внутрішньому носії (локальний жорсткий диск), на спеціальному файловому сервері (сервер SFTP), зовнішньому спільному диску.
9. Система повинна надавати можливість адміністратору перезавантажувати файли резервної копії з поточного місця в інше.
10. Система повинна надавати адміністратору можливість завантажувати в систему файл резервної копії.
11. Система повинна надавати можливість адміністратору налаштовувати, скільки резервних копій слід зберігати одночасно.
12. Система повинна мати можливість автоматично видаляти старі резервні копії.

13. Система повинна надавати можливість адміністратору видаляти резервні копії вручну.
14. Система повинна реєструвати всі операції резервного копіювання.
15. Система повинна надавати можливість адміністратору переглядати журнал операцій резервного копіювання.
16. Система повинна відображати адміністратору список існуючих резервних копій.
17. Система повинна відображати адміністратору такі атрибути існуючих резервних копій, як ім'я, джерела даних, розташування, шифрування, дата створення, розмір, тип резервної, статус (Завершено, Не вдалося, Обробка).

Наступні функціональні вимоги було висуноте до процесу відновлення резервної копії:

1. Система повинна мати можливість відновити повністю рішення із резервної копії.
2. Система повинна мати можливість виконати процедуру відновлення повністю автоматично.
3. Система повинна надавати можливість лише адміністратору розпочати процедуру відновлення.
4. Система повинна надавати можливість адміністратору вибирати файл резервної копії для процедури відновлення.
5. Система повинна надавати можливість адміністратору переглядати стан процедури відновлення.
6. Система повинна реєструвати всі операції відновлення.
7. Система повинна надавати можливість адміністратору переглядати журнал операцій відновлення.

2.2 Розробка сценаріїв використання системи

Після визначення функціональних та нефункціональних вимог, необхідно перейти до опису сценаріїв використання системи. Даний етап є досить важливим, адже правильне розуміння призначення системи допомагає досягти очікуваного результату роботи при розробці продукту.

Першим базовим сценарієм використання є ручне створення резервної копії, який описано у таблиці 2.1

Таблиця 2.1 – Сценарій створення резервної копії вручну

Назва сценарію	Створення резервної копії вручну
Опис	Цей сценарій описує, як адміністратор може створити резервну копію.
Передумови	Адміністратор увійшов у систему. Адміністратор має достатньо прав для запуску процедури резервного копіювання.
Тригер	Адміністратор натискає кнопку "Створити резервну копію".
Кроки	1. Адміністратор переходить на сторінку "Менеджер резервних копій". 2. Адміністратор натискає кнопку «Створити резервну копію». 3. Система відображає вікно "Створити нову резервну копію" з параметрами, які слід вказати.

Продовження таблиці 2.1

Кроки	4. Адміністратор визначає тип резервної копії, шлях для резервного копіювання. За бажанням адміністратор може вказати планувальник резервного копіювання, ліміт копій, шифрування або встановити прапорець "Почати відразу після створення". 5. Адміністратор натискає кнопку "Створити". 6. Система створює нову резервну копію та відображає її у списку резервних копій з усіма атрибутами.
Результат	Створено нову резервну копію, стан виконаної резервної копії "Завершено". Операція зареєстрована.
Альтернативний сценарій	1. Адміністратор може натиснути кнопку "Скасувати" у рядку для поточного резервного копіювання. 2. Система відображає наступне діалогове вікно "Процес резервного копіювання буде припинено. Резервну копію не буде створено. Будь ласка, підтвердьте скасування." 3. Адміністратор натискає кнопку "Підтвердити". 4. Система відображає сторінку "Backup Manager" без будь-яких змін. Статус поточної резервної копії "Скасовано".

Зворотною операцією до створення є відключення процедури резервного копіювання. Даний сценарій описано у таблиці 2.2

Таблиця 2.2 – Сценарій відключення резервного копіювання

Назва сценарію	Відключення резервного копіювання
Опис	Цей сценарій описує, як адміністратор може відключити процедуру резервного копіювання.
Передумови	Адміністратор увійшов у систему. Адміністратор має достатньо прав для відключення процедури резервного копіювання.
Тригер	Адміністратор перемикає статус резервної копії на "Вимкнути"
Кроки	1. Адміністратор переходить на сторінку "Менеджер резервних копій". 2. Адміністратор натискає вкладку «Налаштування». 3. Система відображає параметри процедури резервного копіювання. 4. Адміністратор перемикає статус резервної копії на "Вимкнути". 5. Система вимикає резервне копіювання.
Результат	Резервне копіювання вимкнено. Резервне копіювання більше не створюватиметься. Створені раніше резервні копії все ще існують. Статус поточної резервної копії - "Вимкнено". Операція зареєстрована.
Альтернативний сценарій	Відсутній

Функціональні вимоги визначають необхідність відображення повного списку доступних резервних копій для відновлення. Детальний сценарій використання описано у таблиці 2.3.

Таблиця 2.3 – Сценарій відображення списку існуючих резервних копій

Назва сценарію	Відобразити список існуючих резервних копій
Опис	Цей сценарій описує, як список показує ваше існуючі резервні копії і їх атрибути.
Передумови	Адміністратор увійшов у систему. У системі створено принаймні одну резервну копію.
Тригер	Адміністратор переходить на вкладку "Менеджер резервних копій".
Кроки	Адміністратор переходить на сторінку "Менеджер резервних копій". Система відображає список існуючих резервних копій із наступними полями: 1. Ім'я – назва файлу резервної копії 2. Статус – стан останнього виконання резервного копіювання. Можливі значення: завершено, помилка, ініціалізація, обробка. 3. Дата створення, наприклад 2021-06-04 18:24:54. 4. Час виконання – час, витрачений на процедуру резервного копіювання. 5. Розмір – розмір резервної копії, наприклад 1024 Мб 6. Шлях – шлях до місця, де зберігається файл резервної копії.

Продовження таблиці 2.3

Результат	Система відображає наявні резервні копії з їх атрибутами.
Альтернативний сценарій	Відсутній

При необхідності користувач може видалити певну необхідну резервну копію. Сценарій видалення резервної копії описано у таблиці 2.4.

Таблиця 2.4 – Сценарій видалення резервних копій

Назва сценарію	Видалення резервної копії
Опис	Цей сценарій описує, як адміністратор може видалити резервну копію.
Передумови	адміністратор увійшов у систему. У системі створено принаймні одну резервну копію. Адміністратор має достатньо прав на видалення резервної копії.
Тригер	Адміністратор натискає кнопку "Видалити".
Кроки	1. Адміністратор переходить на сторінку "Менеджер резервних копій". 2. Система відображає список існуючих резервних копій. 3. Адміністратор встановлює прапорець біля резервної копії (копій), яку він хоче видалити. 4. Адміністратор натискає кнопку «Видалити». 5. Система відображає попереджувальне повідомлення "Вибрану резервну копію буде видалено. Будь ласка, підтвердіте."

	6. Адміністратор натискає кнопку "Підтвердити".

Продовження таблиці 2.4

Кроки	7. Система видаляє вибрану копію. 8. Система відображає вкладку "Менеджер резервних копій" без видаленої копії.
Результат	Система видаляє резервну копію.
Альтернативний сценарій	1. Адміністратор натискає кнопку "Скасувати". 2. Система відображає вкладку "Менеджер резервних копій" без будь-яких змін.

Розроблювана система надає можливість запуску створення резервної копії за розкладом. Сценарій запуску наведено у таблиці 2.5.

Таблиця 2.5 – Сценарій створення резервної копії за розкладом

Назва сценарію	Запуск резервної копії за розкладом
Опис	Цей сценарій описує, як система виконує резервне копіювання за розкладом
Передумови	У системі створено принаймні одну резервну копію.
Тригер	Прийшов час, щоб запустити процедуру резервного копіювання за розкладом.
Кроки	1. Система чекає, поки не закінчяться операції з базою даних. 2. Під час цього процесу система відображає статус "Ініціює" для поточної резервної копії 3. Після завершення операцій з базою даних система запускає процедуру резервного копіювання.

Продовження таблиці 2.5

Кроки	4. Система відображає статус “Обробка” для поточної резервної копії на сторінці “Диспетчер резервних копій”. 5. Користувач може навести курсор миші на стан "Обробка", і система відобразить спливаюче вікно із наступною інформацією: скільки відсотків виконано та скільки часу минуло з моменту початку процедури. 6. Після завершення система відображає статус "Завершено" для поточної резервної копії на сторінці "Диспетчер резервних копій" та оновлює інші стовпці для створеної резервної копії
Результат	Запланована процедура резервного копіювання виконана. Створено нову резервну копію. Параметри виконаної процедури оновлені. Операція зареєстрована.
Альтернативний сценарій	Якщо процедура резервного копіювання не вдалася, статус процедури резервного копіювання має бути встановлений як "Не вдалося".

Після створення щонайменше однієї резервної копій в системі, адміністратор може запустити відновлення даних з обраної копії. Даний сценарій описано у таблиці 2.6.

Таблиця 2.6 – Сценарій відновлення даних з резервної копії

Назва сценарію	Відновлення даних з резервної копії

Опис	Цей сценарій описує, як система повинна відновити резервну копію.

Продовження таблиці 2.6

Передумови	Адміністратор увійшов у систему. Адміністратор має достатньо прав для запуску процедури відновлення. Створено принаймні одну резервну копію. Поки користувач не вибрав резервну копію, кнопка "Відновити" не активна. Якщо користувач вибрав дві резервні копії, кнопка "Відновити" стає неактивною. Користувач може вибрати лише одну резервну копію для запуску процедури відновлення.
Тригер	Адміністратор натискає кнопку «Відновити».
Кроки	1. Адміністратор переходить на сторінку "Менеджер резервних копій". 2. Система відображає список існуючих резервних копій. 3. Адміністратор вибирає резервну копію. 4. Адміністратор натискає кнопку «Відновити». 5. Система відображає вікно попередження «Усі поточні дані будуть втрачені. Ви впевнені, що хочете розпочати процедуру відновлення? " 6. Адміністратор натискає кнопку "Так". 7. Система запускає процедуру відновлення. 8. Система блокує інші дії резервного копіювання / відновлення.

Продовження таблиці 2.6

Кроки	9. Система відображає вікно з рядком стану та індикатора (має відображатися час від початку процедури відновлення та кількість виконаних відсотків) процедури відновлення. Також система повинна відображати повідомлення: Система буде відновлена до стану "Дата створення резервної копії". Також система відображає повідомлення "Система відновлює резервні копії:" Список резервних копій "" 10. Користувач може залишити сторінку "Менеджер резервних копій". 11. Функціональність інших сторінок інтерфейсу користувача повинна бути відключена. 12. Після завершення процедури відновлення система відображає повідомлення "Система успішно відновлена". 13. Адміністратор натискає кнопку “Ok”. 14. Система відображає сторінку “Менеджер резервних копій”.
Результат	Система відновлена з файлу резервної копії. Операція зареєстрована.
Альтернативний сценарій	А) Адміністратор натискає кнопку “Ні”. Б) Система відображає сторінку “Менеджер резервних копій”.

2.3 Розробка структури середовища тестування

Для екпериментального дослідження роботоздатності системи та оцінки покриття висунунтих вимог, необхідно розробити середовище розгортання, подібне до реальних умов використання. В додатку К, на якому зображено стуктурну схему тестового середовища, демонстує перелік необхідних компонентів.

Серед складових представлено:

- три типи баз даних, що розгорнуто у докер контейнері;
- кластер minikube, що повністю імітує Kubernetes кластер,до якого входять системний простір імен та простір імен за замовчуванням та мыкросервыси, даны яких потрыбно резервно скопыювати;
- розгортання розроблюваного додатку, що створює резервні копії.

2.4 Розробка структурної схеми системи резервного копіювання даних

Загальна стуктурна схема, розроблена в магістерській дисертації, представлена у додатку Ж. Далі буде описано її компоненти.

Адміністратор взаємодіє з системою через інтерфейс керування, який в свою чергу відправляє запити на сервер резервних копій для керування операціями. Адміністратор має можливість через інтерфейс керування взаємодіяти з налаштуванням кофігурації резервого копіювання, а саме переліку конкретних таблиць та сховищ, за бажанням налаштування рапуску резервного копіювання за розкладом, що в свою чергу відправляє оновлення до блоку запуску задач за розкладом.

Блок клієнт Kubernetes є певного рівня абстракцією над блоком управляння кластером, адже надає інтерфейс взаємодії для безпосереднії запитів на ресурси кластеру. Блок управління кластеру отримує інформацію про стан сервісів(розгортань) через їх API, характеристки PVC через PVC контролер та прив’язки хранилища поду

до нього. Крім того, даний блок надає можливість отримти додаткову інформацію про події Kubernetes.

У свою чергу на сервірі баз даних розташовано три типи баз даних, інформація яких підлягає резервному копіюванню. Взаємодія з базами даних відбувається через бекап воркер, який отримує команди від серверу резервних копій, та який взаємодіє з клієнтом Kubernetes для отримання додаткової інформації про кластер.

2.5 Висновки

У даному розділі проаналізовано вимоги, які висуваються до системи та формалізовано іх. Взято до уваги практичне застосування контейнерів та проаналізовано аскекти ефективного застосування в контейнерних середовищах. Після визначення функціональних та нефункціональних вимог, описано сценарії використання системи.

У підрозділі 2.2 детально розглянуто та описано сценарії використання системи, визначено передумови, тригер, кроки, результат та альтернативний сценарій для кожного з них.

У підрозділі 2.3 розроблено структуру середовища тестування та розглянуто її елементи.

У підрозділі 2.4 розроблено структурна схема системи резервного копіювання даних та описано призначення та взаємодію елементів схеми.

3 ВИБІР ТЕХНІЧНИХ ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ

Для того, щоб розпочати розробку складових системи, необхідно обрати технічні засоби для її реалізації. В цьому розділі буде розглянуто ключові засоби розробки, а саме:

- мова програмування – інструмент для розробки, який визначає ключові парадигми реалізації, особливості продуктивності, можливість масштабування, доступні бібліотеки для інтеграції з інфраструктурою Kubernetes;
- тестовий кластер розробки – середовище, де буде розгортатися та тестуватися сервіс;
- база даних – визначення набору типів сховищ, для яких буде доступна функція резервного копіювання.

3.1 Вибір мови програмування

З розвитком індустрії розробки програмного забезпечення різноманітність мов програмування неперервно зростає. Кожна з них особлива власним підходом до розробки, парадигмами, особливість застосування, швидкістю компіляції. Для того, щоб обрати мову під конкретну задачу, необхідно визначити перелік вимог. Основні з них:

- надання інструментів роботи з асинхронними запитами, адже основна логіка сконцентрована на роботі сервера;
- підтримка паралельної обробки процесів, що відобразиться на продуктивності роботи системи;
- інструменти для взаємодії з кластером Kubernetes – бібліотеки, які доступні для даної мови програмування для взаємодії з API Kubernetes.

Серед виділених мов, які задовольняють поставлені вимоги, будуть розглядатися Java, Python, Go. Дані мови надають описані вище можливості, активно використовуються та розвиваються. Проаналізуємо дані мови з точки зору інших показників.

3.1.1 Аналіз мови програмування Python

Python − широко використовувана, загальнодоступна, динамічна, розширювана мова програмування високого рівня. Його філософія дизайну підкреслює читабельність коду, а синтаксис дозволяє програмістам висловлювати концепції меншою кількістю рядків коду, ніж це було б можливо в таких мовах, як C. Дана мова набула привабливості та зібрала велику спільноту за рахунок великої кількості вбудованих бібліотек, динамічної типізації, що дає свої переваги під час написання скриптів і пришвидшує розробку додатків. Читабельність досягається за рахунок простого та лаконічного синтаксису, а отже це спрошує процес підтримки коду. Python має безкоштовні стандартні бібліотеки для використання на базі основних платформ, а також безкоштовний інтерпретатор, що вільно розповсюджується.

Процеси редагування та відлагождення в Python є дуже швидкими через відсутність кроку компіляції. У процесі відлагодження користувачу має можливість перевіряти значення локальних та глобальних змінних під час виконання програми, користуватися точками зупинки та переходити до необхідного рядка за один раз.

До переваг мови Python можна віднести:

- легкий процес прототипування;
- інтерпретатор, що спрощує процес редагування-тестування-відлагодження;
- велика спільнота;
- великий набір безкоштовних бібкіотек;
- підтримка асинхронності.

Виділимо недоліки мови Python:

- інтерпретованість робить її більш повільною;
- динамічна типізація не гарантує безпечність в даному контексті;
- розробка багатопоточниих програм може викликати проблеми.

Python Розглядається в контексті розробки система резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері. Серед виділених недоліків є проблеми при розробці багатопоточниих програм, що в нашому випадку було виділено як ключова вимога. Тому необхідно розглянути інші мови програмування в пошуках задоволення усіх умов.

3.1.2 Аналіз мови програмування Java

Java є високорівневою, надійною, об'єктно-орієнтованою та безпечною та стабільною мовою програмування, але вона не є чисто об'єктно-орієнтованою мовою, оскільки підтримує примітивні типи даних, такі як int, char тощо. Java є незалежною від платформи мовою, оскільки вона має середовище виконання, тобто JRE та API. Тут платформа означає апаратне або програмне середовище, в якому працює застосунок.

Java створена як мова для розподіленого програмування: він має вбудований механізм спільного використання даних та програмних кількох комп'ютерів, що підвищує продуктивність та ефективність праці.

Розробникам Java не потрібно вручну писати код для управління пам'яттю завдяки автоматичному управлінню пам'яттю (AMM). AMM також використовується в мові програмування Swift та при очищенні пам'яті додатків, які автоматично обробляють розподілення та звільнення пам'яті.

3.1.3 Аналіз мови програмування Go

Go – це мова програмування з відкритим кодом від Google. Перший стабільний реліз було зроблено у 2011 ріці. Go є статично типізованою, компільованою мовою програмування.

Go компілюється безпосередньо до машинного коду (якщо виключити посередницьку збірку Go), що є дуже швидким. Дизайн мови програмування побудований для швидкої компіляції з самого початку. Go компілюється міжплатформено для таких систем, як OS X, Linux, Windows, та багатьох інших. Після компіляції створюється лише один виконуваний файл без будь-яких залежностей, так що можна завантажити його куди завгодно, де підтримується Go, і просто запустити.

У Go добре організований процес роботи з пам’яттю – не потрібні об’єкти постійно моніторяться збирачем сміття та якщо знайдені, то автоматично видаляються. Окрім простоти написання та читання мови Go, вона має детальну та широку документацію, що є важливим моментом у використання.

3.1.4 Порівняння мов у контексті розроблюваної системи

Аналізуючи ключові вимоги до мови програмування, вибір Python було відхилено через надання недостатньо надійних інструментів для паралельної обробки процесів. У мові програмування Go також є проблеми при паралельній обробці даних, оскільки в Go неможливо створити незмінну структуру даних.

Мова програмування Java надає інструменти роботи з асинхронними запитами, підтримує паралельну обробку процесів, а також має бібліотеку для взаємодії з кластером Kubernetes, сама тому мову Java обрано для розробки системи.

3.2 Вибір кластеру Kubernetes для розробки та тестування

Оцінити роботоздатність системи на відповідність вимогам допомагає середовище для розробки та тестування. Опишемо вимоги до середовища:

- кластер Kubernetes, що дає змогу взаємодіяти зі всіма компонентами;
- необмежені права доступу до кластеру, адже процес розробки включає активну взаємодію;
- кількість вузлів не має суттєвого значення.

3.2.1 Аналіз Google Kubernetes Engine

Google Kubernetes Engine (GKE) є керованим, готовим до середовищем для запуску контейнерних програм на основі інфраструктури Google[8]. Декілька машин об’єднано в кластер, який автоматично генерується з вже налаштованими вузлами. Для взаємодії з кластером використовується стандартна утиліта командного рядка kubectl.

GKE працює з контейнерними програмами. Це додатки, упаковані в незалежні від платформи екземпляри ізольованого простору користувача, наприклад, за допомогою Docker. У GKE та Kubernetes ці контейнери, як для програм, так і для пакетних робіт, спільно називаються робочими навантаженнями. Перш ніж розгортати навантаження на кластері GKE, спочатку потрібно упакувати навантаження в контейнер[8].

Новий кластер створюється за допомогою одного кліку, для цього потрібно ввести параметри налаштування, такі як ім’я кластеру, версію кластеру розгортання і тд. Перевагою GKE серед інших хмарних платформ є можливість налаштування TPU – tensor processor unit для задач навчання нейронних мереж та Istio, проте в нашому випадку в цих параметрах немає необхідності.

3.2.2 Аналіз Minikube

Minikube – це локальний кластер Kubernetes, зосереджений на тому, щоб полегшити навчання та розвиток для Kubernetes. Все, що потрібно для запуску – це контейнер Docker (або аналогічно сумісний) або середовище віртуальної машини. Дане розгортання є повноцінним Kubernetes кластером, в яким включено стандартні компоненти, що будуть необхідно в процесі розробки.

Кількість подів регулюється кількістю локальних машин, тому в даному випадку маємо обмеження розгортання одного вузла. Потужність кластеру прирівнюється до потужності локальної машини, тому локальна машина повинна мати достатньо ресурсів для нормальної роботи кластеру.

Враховуючи те, що Minikube надає усі необхідні інструменти взаємодії з Kubernetes кластером, а також може бути розгорнутий на локальній машині і не потребує додаткових витрат, Minikube обрано в якості середовища для розробки та тестування.

3.3 Висновки

В даному розділі проведено аналіз технічних складових реалізації системи, а саме обрано мову програмування та обрано середовище для розробки та тестування.

Як мову програмування обрано мову Java, оскільки іі характеристики задовольняюсь поставлені критерії оцінки. В якості середовища розробки і тестування обрано Minikube, спираючи на те, що він створює кластер на локальній машині, є безкоштовним та включає все компоненти для взаємодії з обчислювальними ресурсами.

4 МАТЕМАТИЧНА МОДЕЛЬ

У розділах 2 – 3 описано розроблену автоматизовану систему резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері. Вона має програмну і апаратну частину резервного копіювання даних на основі технології віртуалізації. Система забезпечує трьократне резервування копіювання даних.

4.1 Припущення і обмеження

У більшості випадків системи даного типу працюють у горячому резерві. Припустимо, що технічні і надійнісні характеристики подів системи резервування мають однакові чисельні значення. Під час відмови одного з подів автоматично підключається резервний. Час підключення резервного поду не впливає на процес збереження даних. Тобто часом переключення будемо зневажати.

Показники надійності інших елементів системи значно кращі, ніж показники надійності системи резервування даних.

4.2 Постановка задачі

Практика розробки, налагодження, тестування і експлуатації програмного забезпечення системи резервного копіювання даних має приблизне значення інтестивності відмов λ для поду при ідеальних умовах.

Зважаючи з приведених припущень дана система представляє собою 3 паралельно працездатних поди. Один з даних подів є основним, інші 2 у гарячому резерві. Таким чином система може перебувати у чотирьох станах. Оголосимо стан кластеру:

s_0 – працюють усі три пода,

s_1 – працює два пода, один відновлюється,

s_2 – працює один под, два відновлюються,

s_3 – не працює жодний под (всі три пода відновлюються).

Граф технічних станів кластеру має вид, показаний на рисунку 4.1.

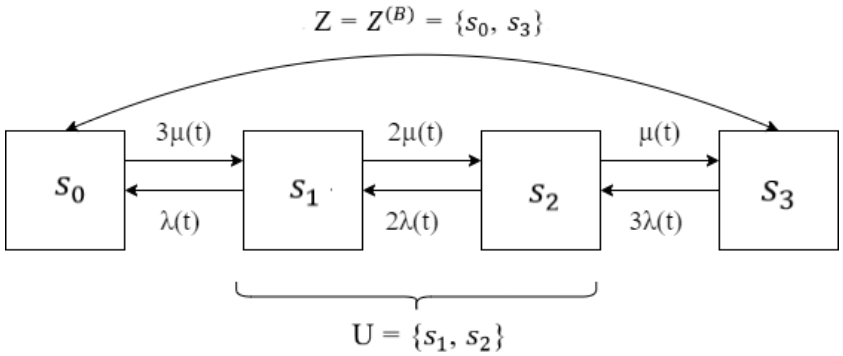

Рисунок 4.1 – Граф станів кластеру

Підмножина U = $\{s_1, s_2\}$, підмножина $Z^{(B)} = \{s_0, s_3\}$. Перетворений граф $\tilde{G}(\tilde{S})$ наведено на рисунку 4.2.

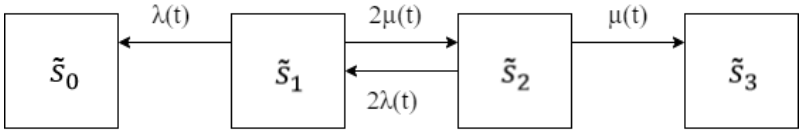

Рисунок 4.2 – Перетворений граф

З урахуванням припущень отримаємо надійнісні характеристики технічної системи. На кожний канал діє пуасоновский потік відмов з інтенсивністю λ(t) і пуасоновский потік ремоту(відновлення) поду з інтенсивністю μ(t) (в подальшому можно припустити, що він постійний і не залежить від часу). Постановка задачі передбачає, що для її рішення можна застосувати модель процесів гибелі та розмноження[21].

Система справна, якщо в ній всі поди працездатні – стан s_0. Система працездатна, коли вона знаходиться в стані s_1 або s_2. Система несправна, якщо знаходиться в стані s_3.

Знайти імовірність знаходження системи у $s_0 - s_3$, при тому що λ = μ = const.

4.2 Розрахунок імовірності знаходження системи у станах $s_0 - s_3$

Вирішемо цю задачу у загальному вигляді, користуючись формулами процесів гибелі та розмноження.

Граф станів кластеру, зображений на рисунку 4.1, показує що змін стані в кластері характерна для процесу гибелі та розмноження. Для визначення імовірності знаходження системи у заданих станах використаємо формули [21]:

$$p_0 = (1 + \frac{\lambda_{01}}{\lambda_{10}} + \frac{\lambda_{12}\lambda_{01}}{\lambda_{21}\lambda_{10}})^{-1},$$

$$p_1 = \frac{\lambda_{01}}{\lambda_{10}} p_0,$$

$$p_2 = \frac{\lambda_{12}\lambda_{01}}{\lambda_{21}\lambda_{10}} p_0,$$

$$p_3 = \frac{\lambda_{23}\lambda_{12}\lambda_{01}}{\lambda_{32}\lambda_{21}\lambda_{10}} p_0.$$

Значення λ_{k-1k} та λ_{kk-1} визначаються формулами:

$$\lambda_{k-1k} = (n - k)\,\lambda,$$

$$\lambda_{kk-1} = k\mu.$$

Розглянемо випадок, коли λ = μ = 1. Використовуючи формули, підставимо значення:

$$p_0 = (1 + \frac{2}{1} + \frac{1*2}{2*1})^{-1} = 0{,}25$$

$$p_1 = \frac{2}{1} * 0{,}25 = 0{,}5$$

$$p_2 = \frac{1*2}{2*1} * 0{,}25 = 0{,}25$$

$p_3 = 0$.

Розглянемо випадок, коли λ = 1, μ = 2. Використовуючи формули, підставимо значення:

$p_0 = (1 + \frac{2}{2} + \frac{1*2}{4*2})^{-1} = 0{,}444$

$p_1 = \frac{2}{2} * 0{,}444 = 0{,}444$

$p_2 = \frac{1*2}{4*2} * 0{,}444 = 0{,}111$

$p_3 = 0{,}001$.

4.2 Розрахунок математичного очікування

Знайдемо вираз для щільності розподілу $f_{1,2}$ випадкової величини $T_{1,2}$ – часу, коли буде справно працювати один або два пода, якщо в початковий момент часу t=0 працює один под, всі поди працюють незалежно.

Складемо систему дифернціальних рівнянь, які відповідають графу $\tilde{G}(\tilde{S})$ [21]:

$d\tilde{p}0(t)/dt = \lambda(t)\, \tilde{p}1(t)$,

$d\tilde{p}1(t)/dt = 2\lambda(t)\, \tilde{p}2(t) – (\lambda(t) + 2\mu(t))\tilde{p}1(t)$,

$d\tilde{p}2(t)/dt = 2\lambda(t)\, \tilde{p}1(t) – (2\lambda(t) + \mu(t))\tilde{p}2(t)$,

$d\tilde{p}3(t)/dt = \mu(t)\, \tilde{p}2(t)$.

Дану систему необхідно вирішувати за початкових умов $\tilde{p}1(0) = 1$, $\tilde{p}i(0)=0$, (i = 1, 2, 3).

При змінних інтенсивностях λ(t) та μ(t) вирішувати дану систему можна будь-яким чисельним методом на ЕОМ. Для прикладу розглянемо випадок, коли λ(t) = λ = const, μ(t) = μ = const. Система рівнянь для зображення по Лапласу ймовірностей станів буде мати вигляд [21]:

$x\tilde{\pi}0(x) = \lambda\tilde{\pi}1(x)$,

$x\tilde{\pi}1(x) = 2\lambda\tilde{\pi}2(x) – (\lambda + 2\mu)\tilde{\pi}1(x)$,

$x\tilde{\pi}2(x) = 2\lambda\tilde{\pi}1(x) - (2\lambda + \mu)\tilde{\pi}2(x)$,

$x\tilde{\pi}3(x) = \mu\tilde{\pi}2(x)$.

Вирішуючи дану систему, отримаємо $\tilde{\pi}1(x) = \frac{x+2\lambda+\mu}{x^2+C_1x+C_0}$, де $C_1 = 3\lambda + 3\mu$, $C_0 = 2\lambda^2 + \lambda\mu + 2\mu^2$[21]. Коріннями рівняння $p(x) = x^2 + C_1x + C_0 = 0$ будуть $a_{1,2} = \frac{-C_1 \pm \sqrt{C_1^2 - 4C_0}}{2}$ [21]. Обидва корні є дійсні, негативні та різні, отже, $\tilde{\pi}1(x) = \frac{x-a_3}{(x-a_1)(x-a_2)}$, де $a_3 = -(2\lambda + \mu)$ [21]. Застосовуючи формули, отримаємо $\tilde{p}1(t) = \frac{e^{a1t}(a_1-a_3)}{a_1-a_2} + \frac{e^{a2t}(a_3-a_2)}{a_1-a_2}$, $\tilde{\pi}2(x) = \frac{2\mu}{x-a_3}\ \tilde{\pi}1(x) = \frac{2\mu}{(x-a_1)(x-a_2)}$, звідки $\tilde{p}2(t) = 2\mu\left[\frac{e^{a1t}}{a_1-a_2} - \frac{e^{a2t}}{a_1-a_2}\right]$ [21].

Знайдемо щільність розподілу випадкової величини $T_{1,2}$ [21]:

$f_{1,2}(t) = \lambda\tilde{p}1(t) + \lambda\tilde{p}2(t) = \frac{\lambda}{a_1-a_2}[e^{a1t}(a_1 - a_3) - e^{a2t}(a_2 - a_3)] + \frac{2\mu^2}{a_1-a_2}[e^{a1t} - e^{a2t}] = q_1(-a_1)\, e^{a1t} + q_2(-a_2)\, e^{a2t}$ $(t > 0)$,

де $q_1 = \frac{\lambda(a_1-a_3)+2\mu^2}{(a_1-a_2)(-a_1)}$,

$q_2 = \frac{\lambda(a_2-a_3)+2\mu^2}{(a_1-a_2)a_2}$.

Вираз являє собою імовірнісну суміш двох показникових законів розподілу з параметрами $(-a_1)$ та $(-a_2)$ та з іморівностями q_1 та q_2 відповідно $(q_1 + q_2 = 1)$ [21].

Отже, $M[T_{1,2}] = \frac{q_1}{-a_1} + \frac{q_2}{-a_2}$,

$M[T_{1,2}^2] = \frac{2q_1}{a_1^2} + \frac{2q_2}{a_2^2}$,

$D[T_{1,2}] = M[T_{1,2}^2] - (M[T_{1,2}])^2$.

Розглянемо випадок, коли $\lambda = \mu = 1$. В цьому випадку $f_{1,2}(t) = e^{-t}$ $(t > 0)$, так як з кожного стану підмножини $U = \{s_1, s_2\}$ можна перейти в один із станів підмножини $Z = \{s_0, s_3\}$ і при цьому інтенсивність потоків подій, які переводять систему із підмножини $U = \{s_1, s_2\}$ і підмножину $Z = \{s_0, s_3\}$, постійна і дорівнює одиниці. Можна переконатися в тому, що для цього випадку $C_1 = 3\lambda + 3\mu = 6$,

C_0 = $2\lambda^2$ + λμ + 2 μ^2 = 5,

$a_1 = \frac{-6+\sqrt{36-20}}{2}$ = -1,

$a_2 = \frac{-6-\sqrt{36-20}}{2}$ = -5,

a_3 = -2λ - μ = -3,

$q_1 = \frac{1(-1+3)+2}{(-1+5)(1)}$ = 1,

$q_2 = \frac{1(-5+3)+2}{(-1+5)(-3)}$ = 0.

$M[T_{1,2}] = \sigma[T_{1,2}]$ = 1.

Розглянемо випадок, коли λ = 2, μ = 1. В цьому випадку C_1 = 9, C_0 = 12, a_1 = -1,628, a_2 = -7,372, a_3 = -5, q_1 = 0,9351, q_2 = 0,06832, = $\sigma[T_{1,2}]$ = 0,2413, $f_{1,2}$(t) = 1,522$e^{1,628t}$ + 0,478$e^{-7,372t}$ (t > 0), $M[T_{1,2}]$ = 0,6832.

Розглянемо найпростіший процес гибелі та розмноження(всі інтенсивності пуасоновських потоків постійні), який знаходиться в стаціонарному режимі. Блукання процесу по підмножині станів U може початися тільки із станів s_i або s_j. Отже, початкові умови для вирішення системи рівнянь задовольнятимуться співвідношенням $\tilde{p}$i(0) + $\tilde{p}$j(0) = 1, $\tilde{p}$k(0)= 0 (k = i-1, i+1, … , j-1, j+1) [21].

Граф станів підсистеми $\tilde{S}$i зображено на рисунку 4.3. Очевидно, закон розподілу часу знаходження в подмножині $\tilde{Z}$i системи $\tilde{S}$i і в підмножині Zi системи S в стаціонарному режимі буде однаковим, оскільки для обох систем початкові умови блукання однакові: $\tilde{p}$i-1(0) = pi-1(0) = 1, а самі підмножини однакові: Zi = $\tilde{Z}$i = {s_0, s_1, … , s_{i-1}}.

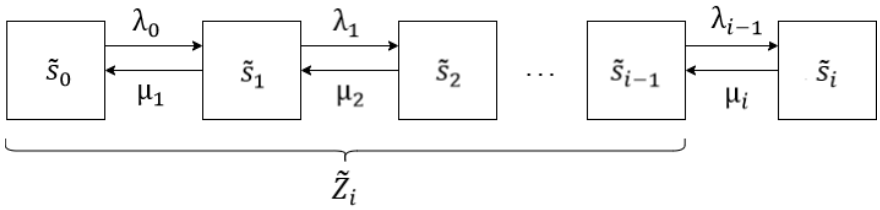

Рисунок 4.3 – Граф станів підсистеми $\tilde{S}$I [21]

Для графа, зображеного на рисунку 4.3, на основі ергодичної властивості маємо $\frac{\bar{t}_{0,i-1}}{(\bar{t}_{0,i-1}+\bar{t}_{i,i})} = \tilde{p}_{0,i-1}$, де $\tilde{p}_{0,i-1} = \sum_{l=0}^{i-1} \tilde{p}_l$ – вірогідність того, що система $\tilde{S}$i в стаціонарному режимі буде знаходитись в підмножині станів $\tilde{Z}$I [21].

$\bar{t}_{i,i} = \mathrm{M}[T_{i,i}] = \frac{1}{\mu_i}$ – математичне очікування часу однократного знаходження системи $\tilde{S}$i в стані s_{i} (для стаціонарного режиму): розподілено за показниковим законом з параметром μ. $\tilde{Z}_i$.

4.2 Висновки

У даному розділі побудовано математичну модель системи резервування на основі марковської теорії процесів гибелі та розмноження. Проаналізовано надійність системи на основі розрахунку імовірності знаходження системи у станах працездатності, справності та несправності, а також розрахунок математичного очікування часу перебування у підмножині станів s_0 – s_3. У ході моделювання побудовано граф станів кластеру.

Після виконаних розрахунків систему можна вважати надійною.

Додати до половини

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Процедури резервного копіювання та відновлення системи дозволяють створювати резервну копію баз даних і дампів файлів і повертатися до збереженої копії, якщо це необхідно. Створення резервної копії може знадобитися для таких типових цілей:

- резервне копіювання перед оновленням системи, з метою профілактики;
- резервне копіювання системи разом із резервним копіюванням віртуальних функцій для того, щоб мати стабільні резервні копії системи.

Дане рішення надає мікросервіс для послідовного резервного копіювання системи та відновлення з резервної копії.

Процеси резервного копіювання та відновлення представлені на наступній схемі, і вони складаються з кроків, що описано далі.

1. Адміністратор деактивує систему (блокує, тобто встановлює режим ReadOnly), щоб запобігти новим запитам та завершити поточні внутрішні операції.

2. Адміністратор запускає резервне копіювання (на головній стороні).

3. Система робить резервну копію всіх файлів з PVC.

4. Система робить резервну копію всіх баз даних у дамп-файлах.

5. Адміністратор надсилає дамп-файли до сховища файлів, якщо є спеціальне сховище для резервних файлів.

6. Адміністратор активує систему (розблоковує, тобто встановлює режим читання-запису), щоб відновити її нормальне функціонування.

7. Коли потрібно запустити відновлення, адміністратор отримує необхідні дамп-файли, з яких можна відновити, якщо файли резервної копії були надіслані до спеціального сховища, тоді адміністратор бере їх із сховища.

8. Адміністратор розміщує файли резервних копій на PVC, де Backup Manager зберігає файли резервних копій.

9. Адміністратор ініціює відновлення (на сайті резервної копії).

10. Система зупиняє служби, які використовують БД.

11. Система відновлює бази даних та PVC зі дампів та резервних файлів.

12. Система перевіряє готовність баз даних та з'єднувачів баз даних, запускає процеси переіндексації та чекає, поки процеси переіндексації не закінчяться.

13. Система відновлює служби, які були зупинені, відновлює всі процеси і дозволяє приймати нові вхідні запити.

14. Адміністратор активує відновлені служби (виконує команду розблокування) для належного функціонування відновлених служб.

5.1 Сценарій резервного копіювання даних

У цьому розділі описано кроки процедури резервного копіювання, що відображено на діаграмі послідовності (Додаток Б). Спочатку потрібно перевести систему в режим лише для читання, а потім запустити резервне копіювання та перевести систему назад у режим читання-запису. Приклади запитів та відповідей див. У "Прикладі виконання кроків резервного копіювання".

Даний сервіс автоматично визначає область блокування / розблокування та резервного копіювання, що включає послідовність, що описана далі.

- Задані мікросервіси, резервне копіювання яких здійснюється після переходу у режим лише читання. Спеціальна мікрослужба повинна бути належним чином налаштована в OpenShift із певними мітками та кінцевими точками.
- Усі PVC, за винятком дампів та файлів резервних копій, крім PVC каталогу NFV із зображеннями VM.
- PVC спеціальних мікросервісів на основі специфічної мітки в OpenShift.

5.1.1 Забезпечення консистентної копії даних

Переключити систему в режим лише для читання – за цією командою сервіс резервного копіювання даних просить мікросервіси змінити їх режим на режим лише для читання та завершити поточні дії. Режим лише для читання запобігає запитам "запис", які можуть змінити дані мікросервісів під час процедури резервного копіювання. Перевірити стан процесу можна за допомогою ідентифікатора блокування, що повертається у відповіді. Поведінка мікросервісів під час завершення поточної діяльності та в режимі "Лише для читання" є наступною: рекомендований статус – HTTP статус 423 "Заблокований". Проте поведінка мікросервісів у режимі "Лише для читання" визначається в їх користувацькій реалізації. Запити на читання обробляються як зазвичай.

Початком резервного копіювання є момент, коли після отримання запиту на резервне копіювання програма робить резервну копію. За потреби можна перевірити стан резервної копії за допомогою ідентифікатора резервної копії, що повертається у відповідь на запит на початок резервного копіювання. Одночасно можна запускати лише один процес. Під час запущеного процесу сервіс відхиляє інші запити на створення нового резервного копіювання. Передбачається, що під час або відразу після резервного копіювання системи, виконується резервне копіювання супутніх зовнішніх систем. Інакше передбачається, що під час резервного копіювання системи не буде змін у зовнішніх системах інтеграції, з якими працює система. Це необхідно для забезпечення послідовних резервних копій частин забезпечення та ресурсів.

Розроблений сервіс повертає систему назад у режим читання-запису, коли процедура резервного копіювання буде закінчена. За цією командою сервіс резервного копіювання перемикає мікросервіси назад у режим читання-запису. Перевірити стан процесу можна за допомогою ідентифікатора розблокування, що повертається у відповіді.

Під час процесу резервного копіювання всі операції з завершення / створення / масштабування тощо відхиляються за допомогою HTTP-коду 423. Ці запити не збираються в черги; натомість вони відкидаються. Усі операції, які вже виконувались до початку резервного копіювання, призупиняються до кінця резервного копіювання. У цьому випадку інтерфейс може не відображати подання процесів належним чином. Після завершення резервного копіювання всі ці операції відновляються та продовжують обробку.

Під час процесу резервного копіювання всі сповіщення від моніторингу (наприклад, сигнали тривоги) не можуть потрапляти на рівень оркестрації. Вони зберігаються в механізмі сповіщень і будуть доставлені після завершення резервного копіювання.

5.1.2 Алгоритм розпізнавання модулів блокування

Диспетчер резервних копій автоматично виявляє мікросервіси для блокування та резервного копіювання за такими конкретними мітками на своєму активному модулі під час розгортання OpenShift, описано у вигляді послідовності, що приведена нижче.

- Змінна lockRequired = "true" вказує на необхідність блокування для цього мікросервісу. Диспетчер резервних копій запустить блокування для мікросервісу, коли активується операція блокування.
- Змінна pollingRequired = "true" вказує на необхідність опитування (перевірка стану) після запуску операції блокування. Диспетчер резервних копій перевірятиме стан цього мікросервісу, доки він не заблокується.
- Змінна hasPostgres = "true" вказує на те, що служба використовує базу даних PostgreSQL. Менеджер резервних копій включить цю базу даних у область резервного копіювання та відновлення.

Диспетчер резервних копій використовує власні кінцеві точки для виконання блокування / розблокування, опитування та резервного копіювання додаткових мікросервісів, які повинні бути реалізовані та вказані у змінних середовища для відповідного розгортання OpenShift:

- LOCK_URL – URL − адреса кінцевої точки для блокування;
- UNLOCK_URL – URL − адреса кінцевої точки для розблокування;
- POLLING_TIMEOUT_MINUTES − необов’язково, визначає максимальний час опитування в хвилинах. Якщо його перевищено, буде розпочато відкат усього процесу блокування. Якщо його не заповнити, буде застосовано час очікування за замовчуванням.

5.2 Сценарій відновлення даних з резервної копії

Менеджер резервних копій надає можливість відновити дані мікросервісів із успішно створеної резервної копії. Адміністратор може розпочати операції відновлення з ідентифікатора резервної копії, отриманого під час операції резервного копіювання, і перевірити стан процесу відновлення за ідентифікатором відновлення, повернутим у відповідь. Можна отримати список процесів резервного копіювання та знайти ідентифікатор резервної копії. Після завершення відновлення адміністратор повинен виконати операцію розблокування для правильного відновлення служб, відновлених із резервної копії.

Зверніть увагу, що під час відновлення інтерфейс користувача може бути недоступним або не стабільним, що відображається як повідомлення про помилки в інтерфейсі користувача. Наприклад, користувач може отримати помилку 503 "Послуга недоступна".

Перед операцією відновлення менеджер резервних копій опускає та піднімає всі служби, які використовують Postgres, і йому може знадобитися певне налаштування

для пошуку та доступу до проектів, наприклад System Monitoring, якщо їх розгортання відрізняється від типового.

Якщо специфічні розгортання, наприклад System Monitoring, не мають власного проекту або він відсутній у розгортанні або має спеціальне ім'я, тоді для менеджеру резервних копій слід встановити змінні середовища, що описані нижче.

- Змінна CUSTOM_PROJECT_EXISTS – ця змінна за замовчуванням має значення "true". Встановіть значення "false", якщо проекту не існує. Наприклад, коли використовується prometheus, він може не мати власного проекту і знаходиться в загальному проекті, тому для змінної слід встановити значення "false".
- Змінна CUSTOM_PROJECT_NAME – при необхідності встановіть його для власного імені проекту.
- Змінна PROJECT_MONITORING_EXISTS – ця змінна за замовчуванням має значення "true". Встановіть значення "false", якщо проект моніторингу системи не існує.
- Змінна PROJECT_MONITORING_NAME – ця змінна має значення "system-monitoring" за замовчуванням. При необхідності встановіть його для власного імені проекту.

Менеджер резервних копій повинен мати роль адміністратора в додаткових заявлених проектах та System Monitoring. Щоб налаштувати цю роль, перейдіть до проекту та додайте обліковий запис служби "backup-manager" із роллю "admin" у "Ресурси"> "Членство"> "Облікові записи служби".

5.3 Очищення файлів резервної копії

Система забезпечує процедуру автоматичного очищення пошкоджених файлів резервних копій. Завдання очищення виконується за розкладом і вмикається за

замовчуванням. Виконання завдання очищення пропускається, коли система перебуває в режимі лише для читання.

Параметри розкладу встановлюються у таких змінних середовища для розгортання Backup Manager, що описані нижче.

- Змінна SCHEDULER_RETRY_TIMEOUT – за замовчуванням для цієї змінної встановлено значення 60. Він визначає період очищення, тобто час між двох послідовних виконанням процедури очищення.
- Змінна SCHEDULER_RETRY_TIME_UNIT – ця змінна за замовчуванням має значення MINUTES. Він визначає одиницю часу для періоду очищення. Допустимі такі значення: DAYS, HOURS, MINUTES, SECONDS, MILLISECONDS, MICROSECONDS, NANOSECONDS.

Процедуру автоматичного очищення можна вимкнути або ввімкнути за допомогою наступної змінної середовища:

- змінна CONSISTENCY_CHECK_ENABLED – ця змінна за замовчуванням має значення "true". Встановіть значення "false", якщо немає необхідності запускати завдання автоматичного очищення.

Можна запустити очищення вручну, використовуючи такі кінцеві точки:

- v1 / cleanup / consistency_check / start – запускає очищення з тайм-аутом та одиницею часу, встановленими у описаних вище змінних;
- v1 / cleanup / consistency_check / stop – зупиняє виконання періодичного завдання очищення.

Після повторного розгортання система буде виконувати очищення відповідно до змінної CONSISTENCY_CHECK_ENABLED, навіть якщо очищення було зупинено або розпочато через кінцеві точки.

Процедура очищення видаляє недійсні каталоги резервної копії, виявлені за одним із таких критеріїв:

- каталог не містить metafile.json – файлу, який зберігає загальний та детальний стан резервної копії;
- каталог містить metafile.json, який пошкоджений або має недійсну структуру;
- каталог містить metafile.json з допустимою структурою, але загальний стан резервної копії – IN_PROGRESS.

5.4 Висновки

В даному розділі описано програмну реалізацію автоматизованої системи резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері, описано вимоги до системи, а також сценарії використання.

В розділі 5.1 описано алгоритм створення резервної копії даних, умови забезпечення консистентної копії та метод розпізнавання модулів блокування.

В розділі 5.2 описано алгоритм відновлення даних з резервної копії, методи відновлення, параметри налаштування відновлення.

В розділі 5.3 описано умови для очищення файлів резервної копії, налаштування розкладу та інших параметрів конфігурації.

6 ТЕСТУВАННЯ СИСТЕМИ

У цьому розділі буде проведено тестування автоматизованої системи резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері. Розроблені сценарії покривають наступні фази роботи системи:

- блокування сервісів для забезпечення консистентної копії даних;
- резервне копіювання даних;
- відновлення даних з резервної копії.

6.1 Тестування функції резервного копіювання

Для створення резервної копії даних необхідною умовою є те, що копійовані дані не будуть змінюватись будь-якими взаємодіями з ними. Для досягання цієї мети менеджер резервного копіювання розпізнає за зазначеними критеріями сервіси, що потрібно зупинити на приймання запитів. Для цього виконаємо наступну команду, наведену на рисунку 6.1, де bm_ulr – зовнішній роут до мікросервісу, створений у Kubernetes роутах.

```
POST bm_ulr/v1/lock
```

Рисунок 6.1 – Команда для блокування сервісів

Даний запит є асинхронним, про що говорить http status code 202, тому після його виконання буде отримана наступна відповідь з ключем ідентифікатором, за яким можна перевірити статус виповнення операції блокування сервісів. Приклад відповіді на запит блокування наведено на рисунку 6.2.

```
Code: 202 Accepted Body: "2e4db355-e18c-414a-a96a-64d4d656ff41"
```

Рисунок 6.2 – Приклад відповіді на запит блокування

Наступним кроком необхідно дочекатися повного блокування всіх сервісів, про що скаже статус «SUCCESSFUL» для загального процесу блокування. Доступними статусами також є статус “IN_PROGRESS”, який каже про те, що на момент запиту статусу не всі сервіси ще були заблоковані. Приклад відповіді на запит наведено на рисунку 6.3, де показано блок зі статусом загального процесу.

```
Code: 200 OK
Body:

{
 "id": "2e4db355-e18c-414a-a96a-64d4d656ff41",
 "status": "SUCCESSFUL"
 "standardManoMicroservicesStatus": {
```

Рисунок 6.3 – Блок статусу загального процесу

На рисунку 6.4 наведено блок відповіді, яка містить весь список сервісів для блокування та їх статуси.

```
"monitoring-gateway": "COMPLETED",
"notification-engine": "COMPLETED",
"api-gateway": "COMPLETED",
"monitoring-console": "COMPLETED",
"trap-listener": "COMPLETED",
"alarm-processing-node": "NO_TASKS_RUNNING",
"mistral": {
"partOfRunningTasks": "unknown",
"status": "COMPLETED"
}
},
"additionalMicroservicesStatus": {
"custom-ms-1": "COMPLETED",
"custom-ms-2": "COMPLETED"
}
}
}
```

Рисунок 6.4 – Список сервісів та їх статуси

Після отримання успішного результату на блокування сервісів, можна розпочинати операцію резервного копіювання даних. Для цього потрібно скористатися ендпоінтом у менеджері резервних копій. Приклад запиту наведено на рисунку 6.5.

```
POST bm_url/v1/backup
```

Рисунок 6.5 – Запит на створення резервної копії

У якості відповіді буде отримано http status code та ідентифікатор резервної копії, створення який розпочато (рисунок 6.6).

```
Code: 200 OK Body: 2018-06-12_08-52-13
```

Рисунок 6.6 – Відповідь на запит створення резервної копії

Після отримання унікального ідентифікатора резервної копії є можливість динамічно спостерігати за процесом створення. На загальний статус резервної копії впливає статус виконання для кожного виду баз даних, множина яких характеризується множиною тих, які використовуються у мікросервісній системи. Приклад статусу резервної копії наведено на рисунку 6.7.

```
Code: 200 OK
Body:
{
 "id": "2018-06-12_08-52-13",
 "startTime": "2018-06-12 08:52:13.5213",
 "finishTime": "2018-06-12 08:54:04.544",
 "status": "SUCCESSFUL",
 "msProcesses": [{
  "subProcesses": [{
   "id": "backup-20180612T085223058788",
   "status": "SUCCESSFUL",
   "persistenceType": "PostgreSql"
  }],
  "microserviceName": "Backup_all_pg_dbs"
 }, {
  "subProcesses": [{
   "id": "20180612T085224",
   "status": "SUCCESSFUL",
   "databases": ["lambda_backup_tosca_composer",
"lambda_backup_workflow_service", ],
   "persistenceType": "MongoDb"
  }],
  "microserviceName": "Backup_all_mongo_dbs"
 }, {
  "subProcesses": [{
   "id": "RuntimeCatalog_2018-06-12_08-52-13",
   "status": "SUCCESSFUL",
   "targetPath": "/backup-manager/mano_backup",
   "sourcePath": "source/runtime-catalog/runtime_catalog_pvc",
   "microserviceName": "runtime-catalog",
   "persistenceType": "Pvc"
  }],
  "microserviceName": "runtime-catalog"
 }]
```


Рисунок 6.7 – Приклад статусу резервної копії

На успішне виконання резервної копії надається вичерпний час, який зазначається у конфігурабельній змінній середи для менеджеру резервних копій. Для того щоб побачити результати виконання у даний час, необхідно виконати запит, приклад якого наведено на рисунку 6.8.

```
GET bm_url/v1/backup/{backupId}
```

Рисунок 6.8 – Приклад запиту на отримання статусу резервної копії

Статус «SUCCESSFUL» свідчить про те, що резервна копія була успішно створена та може бути використана у подальшому для відновлення даних середовища. Після цього можна повертати систему знову до режиму читання-запису за продовжувати виконання операцій. Для розблокування мікросервісів необхідно виконати запит, приклад якого наведено на рисунку 6.9:

```
POST bm_ulr/v1/unlock
```

Рисунок 6.9 – Приклад запиту для розблокування мікросервісів

Даний запит є асинхронний про що свідчить http status code 202. У відповідь на запит буде надано ідентифікатор операції розблокування мікросервісів, який далі необхідно використати для моніторингу статусу операції. Приклад відповіді наведено на рисунку 6.10:

```
Code: 202 Accepted Body: "2e4db355-e18c-414a-a96a-64d4d656ff42"
```

Рисунок 6.10 – Приклад відповіді на запит розблокування мікросервісів

Як було зазначено, використаємо ідентифікатор операції для визначення її статусу, як покахано на рисунку 6.11:

```
GET bm_ulr/v1/unlock/{unlockId}
```

Рисунок 6.11 – Приклад запиту на статус операції

Результатом запису є JSON відповідь, яка містить у собі поля з ідентифікатором операції та її статус. Успішно завершена операція розблокування буде мати відповідь, наведену на рисунку 6.12:

```
Code: 200 OK
Body:
{
 "id": "2e4db355-e18c-414a-a96a-64d4d656ff42",
 "status": "SUCCESSFUL"
}
```

Рисунок 6.12 – Приклад успішно завершеної операції

6.2 Тестування функції відновлення даних з резервної копії

Після створення резервної копії, користувач може відновити данні з певної версії копії за її ідентифікатором або переглянути весь список резервних копій. Приклад запиту списку резервних копій наведено на рисунку 6.13:

```
GET bm_url/v1/backup
```

Рисунок 6.13 – Приклад запиту списку резервних копій

Результатом запису є JSON відповідь зі списком ідентифікаторів резервних копій, статусом бекапу для кожного виду баз даних, а також перелік баз, які включені в дану резервну копію. Приклад відповіді наведено на рисунку 6.14.

```
Code: 200 OK
Body:
[
  {
    "backupProcessId": "2018-07-04_11-05-06",
    "mongo": null,
    "postgre": null,
    "pvc": null,
    "status": "FAILED"
  },
  {
    "backupProcessId": "2018-07-04_13-48-29",
    "mongo": {
      "status": "SUCCESSFUL",
      "backupId": "20180704T134831",
      "dbs": [
        "mano_alarm_processing_node",
        "mano_artifact_manager",
        "mano_nel",
        "mano_nfv_catalog",
        "mano_tosca-composer",
        "test",
        "workflow-service",
        "config"
      ],
      "path": "null/granular/20180704T134831/20180704T134831.tgz"
    },
    "postgre": {
      "status": "SUCCESSFUL",
      "backupId": "backup-20180704T134829791517",
      "path": "null/granular/default/backup-20180704T134829791517"
    },
    "pvc": {
```


Рисунок 6.14 – Приклад частини відповіді, що містить перелік баз даних

Результат відповіді також має секцію, де описано PVC яких сервісів включено в резервну копію та шлях, де вони збережені. Приклад опису PVC наведено на рисунку 6.15.

```
      "status": "SUCCESSFUL",
      "restorePath": "/backup-manager/mano_backup/2018-07-04_13-48-29/pvc",
      "nameToPath": {
        "MonitoringConsole_2018-07-04_13-48-34": "source/monitoring-
console/config",
        "NfvCmEngine_2018-07-04_13-48-34": "source/nfv-cm-
engine/var/playbooks",
        "ResourceManager_2018-07-04_13-48-34": "source/resource-
manager/shared/adapters",
        "RuntimeCatalog_2018-07-04_13-48-34": "source/runtime-
catalog/runtime_catalog_pvc"
      }
    },
    "status": "SUCCESSFUL"
  },
```


Рисунок 6.15 – Приклад опису PVC

Після того, як користувач визначив ідентифікатор резервної копії для відновлення даних, необхідно виконати запит, наведений на рисунку 6.16.

```
POST bm_url/v1/restore/{backupId}
```

Рисунок 6.16 – Запит для відновлення даних

У відповідь буде отримано ідентифікатор відновлення даних, за допомогою якого користувач може слідкувати за процесом. Відповідь з ідентифікатором наведена на рисунку 6.17.

```
Code: 202 Accepted Body: 2018-06-12_08-52-20
```

Рисунок 6.17 – Ідентифікатор відновлення даних

Скориставшись ідентифікатором відновлення даних, користувач може виконати запит для отримання статусу процесу відновлення даних. Приклад запит наведено на рисунку 6.18.

```
GET bm_url/v1/restore/{restoreId}
```

Рисунок 6.18 – Приклад запиту статусу відновлення даних

Відповідь статусу процесу містить у собі ідентифікатор процесу відновлення, перелік підпроцесів та загальний статус операції. У переліку підпроцесів наведено шлях для кожного відновлення, тип сховища, а статус для цього підпроцесу. Приклад відповіді статусу відновлення резервної копії наведено на рисунку 6.19.


```
Code: 200 OK
Body:
{
  "id": "2018-10-06_08-38-41",
  "restoreSubProcesses": [
    {
      "restoreId": "/backup-manager/mano_backup/2018-10-05_13-34-
35/pvc/Custom2Pvc_2018-10-05_13-34-48.zip",
      "backupId": "Custom2Pvc_2018-10-05_13-34-48.zip",
      "status": "SUCCESSFUL",
      "persistenceType": "Pvc"
    },
    {
      "restoreId": "/backup-manager/mano_backup/2018-10-05_13-34-
35/pvc/ResourceServiceAdapters_2018-10-05_13-34-48.zip",
      "backupId": "ResourceServiceAdapters_2018-10-05_13-34-48.zip",
      "status": "SUCCESSFUL",
      "persistenceType": "Pvc"
    },
    {
      "restoreId": "restore-default-20181005T133442772394-
20181006T083845993815",
      "backupId": "backup-20181005T133442772394",
      "status": "SUCCESSFUL",
      "persistenceType": "PostgreSql"
    },
    {
      "restoreId": "869a63bd-4f45-4012-b889-9471d6dac2f5",
      "backupId": "20181005T133443",
      "status": "SUCCESSFUL",
      "dbs": [
        "mano_alarm_processing_node",
        "mano_artifact_manager",
        "mano_nel",
        "mano_nfv_catalog",
        "mano_tosca-composer",
        "mano_workflow_service",
        "config"
      ],
      "persistenceType": "MongoDb"
```


Рисунок 6.19 – Приклад відповіді статусу відновлення резервної копії

6.3 Висновки

В ході даного розділу розглянуто основні сценарії роботи системи, проведено тестування розробленої системи резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері, а саме таких сценаріїв, як:

- тестування функції резервного копіювання, підготовчих етапів та запитів для запуску процесу резервного копіювання у розділі 6.1;
- тестування функції відновлення даних з резервної копії, отримання списку резервної копій, використання ідентифікатора для керування процесом у розділі 6.2.

7 РОЗРОБКА СТАРТАП ПРОЕКТУ

В даному розділі описується рішення для монетизації продукту, що розробляється в даній магістерській дисертації. Проводиться аналіз ринку, а саме існуючих рішень, на основі якого формуються вимоги та ідеї для монетизації даного продукту. Розглядаються переваги і недоліки рішення в контексті комерціалізації.

7.1 Опис ідеї проекту

Головною метою даного стартап-проекту є розробка системи для автоматизації цілісного резервного копіювання даних, що орієнтована на роботу мікросервісними додатками в кластері Kubernetes. Дане рішення може бути використане замовниками в будь-якому оточенні Kubernetes в незалежності від платформи на різноманітних проектах. В таблиці 7.1 розглядається зміст ідеї, цільові сфери застосування та переваги продукту перед існуючими рішеннями.

Таблиця 7.1 – Опис ідеї стартап-проекту

Основний зміст ідеї	Сфери застосування	Переваги використання системи для користувача
Автоматизація процесу резервного копіювання мікросервісних додатків	Резервування файлів користувачів	В разі встрати файлів користувачів є можливість швидко розвернути останню резервну копію
	Біллінг коштів	Підвищення надійності критичних сервісів, втрата даних яких є небажаною
	Міграція даних	Даний продукт дозволяє в разі спотворення даних при невдалій міграції даних швидко відновити втрачену інформацію.

Огляд ринку показав, що фактично – схожих за функціоналом та надійністю – у розроблюваної системи немає потенційних конкурентів. Як аналог можна використовувати AP Backup або Velero. Проте, впровадження даних альтернативних варіантів є досить складним, слабкою інтеграцією с інструментами кластеру та надає меншу варіативність з точку зору зберігання. Більш докладно проаналізовані слабкі, нейтральні та сильні сторони продуктів у таблиці 7.2.

Таблиця 7.2 – Опис сильних, слабких та нейтральних сторін основної ідеї стартап-проекту

Техніко-економічні ознаки ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтра-льна сторона)	S (сильна сторона)
	Поточний проект	AP Backup	Velero			
Мова розробки	Java	C++	Go	+		
Архітектура	Мікро-сервісна	Моно-літна	Мікро-сервісна		+	
Тип ліцензії	MIT	MIT	Apache		+	
Сбосіб зберігання	ZIP, FTP, S3, бінарні файли	ZIP, FTP	etcd			+
Інтеграція з Kubernetes	+	-	-			+
Забезпечен-ня високої	+	-	-			+

надійності (НА)						

Продовження таблиці 7.2 – Опис сильних, слабких та нейтральних сторін основної ідеї стартап-проекту

Техніко-економічні ознаки ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтра-льна сторона) Поточ-ний проект	S (сильна сторона)
	Поточний проект	AP Backup	Velero			
Алгоритми захисту даних	AES-256	Відсутній	Відсутній			+

Перелік сильних сторін розробленого програмного забезпечення, наведений вище, становить основу для створення конкурентоспроможної бізнес-моделі для світового ринку розробки програмного забезпечення, оскільки ці критерії є критичними у будь-якій інформаційній системи. Основне завдання полягає лише у вдосконаленні спільноти користувачів, але з часом ця проблема буде вирішена.

7.2 Технологічний аудит ідеї проекту

У таблиці 7.3 проводиться технологічний аудит ідеї. Визначено набір технологій для розробки даного рішення, проводиться аналіз необхідних технологій у вільному доступі з огляду на вказані ліцензії.

Серед основних функцій проекту:

- налаштування резервного копіювання, що включає в себе розробку спеціальних CRD об’єктів для Kubernetes, які дозволять легко впроваджувати дану систему;
- управління резервними копіями – зберігання на різних сховищах, швидке отримання останніх резервних копій та їх розгортання на базі даних, перенос копії з одного сховище на інше, версіонування, видалення.

Таблиця 7.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології реалізації	Наявність технологій	Доступність технологій
1	Налаштування резервного копіювання	Kubernetes, Helm	Розгортання в кластері та налаштування за базовими принципами Kubernetes	Доступні
2	Управління резервними копіями	Java	Наявність власної реалізації	Необхідне доопрацювання
Технологія, обрана для реалізації ідеї проекту: Розробляється система автоматизованого резервного копіювання на основі Kubernetes з використанням мови програмування Java. Налаштування здійснюється з використанням загальнодоступних інструментів Kubernetes.				

7.3 Аналіз ринкових можливостей стартап-проекту

У даному розділі проводиться аналіз можливостей на ринку технологій у сфері автоматичного резервного копіювання даних. Також у розділі розглядаються загрози, які потенційно можуть мати негативні наслідки.

Таблиця7.4 – Аналіз загроз на ринку для стартап-проекту

№ п/п	Характеристика ринку	Критерій
1	Кількість головних конкурентів, од	~ 3
2	Обсяг продаж, грн/ум.од	Неможливо оцінити
№ п/п	Характеристика ринку	Критерій
3	Умови для розвитку	+
4	Обмеження для старту проекту	-
5	Проходження сертифікації для запуску	За бажанням
6	Динаміка ринку	Зростає
7	Рентабельність, %	Дуже висока, 90%

Виходячи з результатів проведеного аналізу, можна припустити, що дана технологічна ніша є доволі привабливою для запуску стартап-проекту.

У таблиці 7.5 наводяться головні характеристики потенційних клієнтів, де ЦА – цільова аудиторія, а ПЦГК – потенційна цільова група клієнтів, ШР – швидкість розгортання.

Таблиця 7.5 – Характеристика можливих клієнтів

№ п/п	Чинник	ЦА	Різниця у поведінці різних ПЦГК	Характеристики, яких потребують користувачі
1	Налаштування резервного копіювання	Адміністратори кластеру	Користувачі потребують гнучкої конфігурації	гнучка конфігурація розгортання, простота

Продовження таблиці 7.5

№ п/п	Чинник	ЦА	Різниця у поведінці різних ПЦГК	Характеристики, яких потребують користувачі
2	Управління резервними копіями	Розробники проекту чи сервісу	Користувачі потребують надійності в зберіганні даних	- висока надійність - доступність копій

На основі результатів аналізу груп потенційних клієнтів проводиться аналіз ринкового середовища. Потрібно розуміти, що на різні категорії клієнтів можуть впливати ті чи інші технологічні або фінансові чинники при виборі рішення. Наприклад, для різних категорій інженерів в пріоритеті можуть бути важливі для них фактори, як швидкість та безвідмовність роботи або простота у використанні.

Таблиці факторів, які можуть позитивно впливати на запуск проекту на ринку, та факторів, які навпаки можуть впливати негативно наведені в таблицях 7.6 та 7.7.

Таблиця 7.6 – Чинники загроз

№ п/п	Чинник	Аналіз загрози	Рішення	Узгодження

1	Поява конкурентів	Поява конкурентних рішень зі схожим функціоналом	Доопрацювання функціоналу рішення та підтримка клієнтської бази	-
2	Відсутність інвестицій	Складність у пошуку інвесторів	Покрокове залучення інвестицій з подальшим розширенням	Залучення міжнародних інвесторів

Продовження таблиці 7.6

№ п/п	Чинник	Аналіз загрози	Рішення	Узгодження
3	Висока ціна розробки	Можлива поява open source рішень, які не потребують фінансування	Розробка даного рішення потребує залучення висококваліфікованих спеціалістів, що потребує відповідних фінансових затрат. З іншого боку – open source, де розробники працюють на ентузіазмі	Побудова маркетингової кампанії. Розробка комплексного функціоналу. Залучення до безкоштовного плану

Таблиця 7.7 – Критерії можливостей

№ п/п	Чинник	Рішення	Узгодження

1	Відсутність монополії	На даний момент відсутні аналогічні рішення	-
2	Застосування конференцій як рекламних каналів	Пріоритет на рекламу через презентації на тематичних конференціях	Проводити презентації на тематичних конференціях з Kubernetes

У таблиці, конференцій, наводяться результати аналізу, які характеризують ознаки конкуренції в даній ніші на ринку (таблиця 7.8).

Таблиця 7.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється даний критерій	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
1. Вказати тип конкуренції Тип конкуренції: чиста	На світовому ринку – присутні схожі за функціоналом рішення	Постійне вдосконалення рішення з врахуванням зворотного зв'язку клієнтів проекту.
2. За рівнем конкурентної боротьби – міжнародний	Частково схожі рішення присутні на міжнародному ринку	Масштабувати проект на світовому ринку хмарних середовищ
3. За галузевою ознакою – міжгалузева	Інформаційні системи присутні у кожній галузі	Створення універсального рішення

Особливості конкурентного середовища	В чому проявляється даний критерій	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
4. Конкуренція за видами товарів – товарно-родова	Конкуренція відбувається на рівні технології задоволення потреб	Активна маркетингова кампанія

Продовження таблиці 7.8

Особливості конкурентного середовища	В чому проявляється даний критерій	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
5. За характером конкурентних переваг – нецінова	Клієнти готові платити вищу ціну за більш надійне та якісне рішення	Спрямування частини доходів на вдосконалення та розвиток рішення
6. За інтенсивністю – марочна	Клієнти довіряють компаніям, які є відомими на ринку інформаційний технологій	Просування продукту серед великих компаній

З результатів проведеного аналізу можна припустити, що конкуренція відсутня, тобто є можливість вільно зайняти дану нішу на ринку. Хоча і присутні рішення, які виконують частину функцій проекту, проте дане рішення робить ставку на комплексне застосування.

Основними перевагами при виборі цього продукту є простота у використанні, надійність та якість роботи, якої немає у конкурентів.

У таблиці 7.9 аналізуються дані, які показують, що можливість виходу на ринок досить висока, оскільки продукт пропонує унікальні рішення і технології, з якими буде досить складно конкурувати існуючим рішенням. Варто зазначити, що розроблений продукт – це проста у впровадженні система, є дуже надійною та задовольняє потреби більшості клієнтів.

Таблиця 7.9 – Аналіз конкуренції

	Прямі конкуренти	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
Складов і аналізу	Конкуренція на ринку України відсутня, на світовому присутні схожі рішення	Поява конкурентів на ринку є малоймовірною, окрім open source рішень	Відсутні	На ринку України - обмежені платоспро-можністю	Відсутні
Висновки:	Прями конкуренти відсутні	Моніторинг open source рішень	Відсутні	Спеціальний тариф з частковими функціями	Відсутні

У таблиці 7.10 проводиться аналіз конкуренції, без врахування особливостей ідеї, критеріїв вибору продукту споживачем та факторів маркетингового середовища, а також формується список факторів конкуренції.

Таблиця 7.10 – Переваги та недоліки стартап-проекту

№ п/п	Критерій конкурентоспроможності	Оцінка 1-30	Порівняння існуючих рішень у конкурентів						
			-3	-2	-1	0	1	2	3
1	Швидкість та простота впровадження	27	+						
2	Надійність	30	+						
3	Конфігурування	27				+			

Продовження таблиці 7.10

№ п/п	Критерій конкурентоспроможності	Оцінка 1-30	Порівняння існуючих рішень у конкурентів						
			-3	-2	-1	0	1	2	3
4	Звітність	25				+			
5	Висока собівартість	20				+			

SWOT є останньою частиною аналізу перед запуском стартап-проекту на ринку, так як в SWOT-аналізі припускається, що сильні та слабкі сторони найчастіше є внутрішніми, а можливості та загрози є зовнішніми. SWOT-аналіз наведено у таблиці 7.11.

Таблиця 7.11 – SWOT-аналіз

Плюси: – висока надійність;	Мінуси: – висока ціна продукту;

– швидкість та простота впровадження; – гнучка конфігурація; – звітність.	– для розробки потрібні високооплачувані спеціалісти; – підтримка оточень лише на базі Kubernetes.
Інші перспективи: – масштабування на інші хмарні оточення.	Інші ризики: – поява нових конкурентів; – open-source.

На основі результатів SWOT-аналізу можна сформувати інші способи реалізації стартап-проекту для виходу на ринок та орієнтовний термін реалізації. Альтернативні способи реалізації стартап-проекту, основні перспективи залучення необхідних ресурсів, а також терміни реалізації наведені в таблиці 7.12.

Таблиця 7.12 – Інші способи реалізації стартап проекту на ринку

№ п/п	Інші способи реалізації стартап проекту ринкової поведінки	Перспектива отримання ресурсів	Термін реалізації
1	Підвищення пізнаваності продукту за допомогою маркетингової кампанії	Ймовірно	4 місяці
2	Використання безкоштовних технологій для розробки стартап-проекту	Ймовірно	Протягом розробки
№ п/п	Інші способи реалізації стартап проекту ринкової поведінки	Перспектива отримання ресурсів	Термін реалізації
3	Підготовка базового тарифного плану зі зменшеною ціною для виходу на ринок	Ймовірно	6 місяців

Роблячи висновок з даних аналізу, найвигіднішим варіантом для початку є саме 3 пункт таблиці 7.12.

7.4 Ринкова стратегія

Для об’єктивної оцінки ринкової діяльності та розробки стратегії ринкової політики необхідно вивчити наступні умови: ринок, у сфері якого відбувається розробка продукту, які сегменти ринку є цільовими, потенційних та прямих конкурентів.

На етапі аналізу рингу необхідно в першу чергу визначити його продуктові та географічні кордони, суб’єкти – продавці та покупці, поточний об’єм та ємність ринку, кількісні та якісні показники структури ринку.

Опис ЦГ (цільових груп) потенційних клієнтів наведені у таблиці 7.13.

Таблиця 7.13 – Вибір ЦГ потенційних користувачів

№ п/п	Цільова група потенційних клієнтів	Зацікавленість введення проекту на ринок для споживача	Приблизний відсоток в межах ЦГ	Конкуренція в сегменті	Легкість входу у сегмент
1	IT компанії	Висока зацікавленість	80%	Конкуренція майже відсутня	Низька складність
2	Архітектори та адміністратори	Середня	60-70%	Низька	Середня складність
3	Державні установи	Середня зацікавленість	60%	Низька	Залежно від географічної

					зони. Висока
В результаті аналізу було обрано 1, 2 групи					

Після аналізу потенційних груп клієнтів необхідно обрати цільову групу, для яких буде пропонуватись продукт, та обирають стратегію охоплення ринку:

1) Стратегія масового маркетингу – підприємець не враховує відмінності між сегментами ринку та розглядає ринок як єдине ціле.

2) Стратегія диференційованого маркетингу – підприємець прагне охопити найбільшу можливу кількість сегментів ринку зі спеціально для них розробленими продуктами і окремою маркетинговою політикою для кожного з сегментів.

3) Стратегія концентрованого маркетингу – підприємець зосереджує свою увагу і ресурси на одному з сегментів ринку і пропонує продукт саме для обраної групи можливих клієнтів. Це стратегія «спеціалізації», в якій пропозиція, як правило, формується та розробляється спеціально під потреби конкретної цільової групи клієнтів, тому підприємець може, як правило, встановлювати на свій товар достатньо високі ціни.

Отже, для роботи в обраних вище сегментах ринку необхідно розробити базову стратегію розвитку (таблиця 7.14).

Таблиця 7.14 – Початкова стратегія конкурентної поведінки

№ п/п	Альтернатива розвитку	Стратегія охоплення ніші на ринку	Головні конкуренти стартап проекту відповідно до обраної альтернативи	Початкова стратегія розвитку

1	Розробка базового тарифного плана зі зменшеною ціною для виходу на ринок	Клієнтам не завжди потрібен повній пакет послуг, таким чином можливо охопити більше клієнтів	Velero – пропонує базову версію за зменшеною ціною	Диференціації

У таблиці 7.15 наведене визначення початкової стратегії конкурентної поведінки.

Таблиця 7.15 – Початкова стратегії конкурентної поведінки

№ п/п	Чи є проект «першопрохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурен-тної поведінки
1	Ні	Забирати у існуючих та шукати нових	Частина функціоналу охоплює аналоги	Стратегія лідера

7.5 Висновки до розділу

У даному розділі проведено аналіз потенційної маркетингової стратегії розробки рішення, а саме автоматизованої системи управління резервними копіями у кластері Kubernetes, а також перспективи впровадження даного продукту на ринку інформаційних послуг.

Рентабельність даної системи знаходиться на досить високому рівні, але надалі необхідно розвивати продукт як в плані технологій, так і маркетингу. Це пов’язано з тим, що розробка проводиться у сфері, де технології швидко застарівають та заміщуються, постійно постають все нові вимоги клієнтів. Більш того, для впровадження даного рішення у сферах банківських та медичних послуг, а також державних установах необхідно отримати відповідні сертифікати надійності та безпеки, що може дорого коштувати.

На основі результатів проведеного аналізу можна зробити висновок, що подальший розробка даного продукту є доцільною, оскільки в галузі є потреба в рішеннях такого роду, а обране середовище – Kubernetes – перспективним для розвитку страртап-проекту.

ВИСНОВКИ

У дисертації розглянуто проблему резервного копіювання даних в рішеннях з мікросервісною архітектурою у Kubernetes кластері. Проведений аналіз існуючих рішень виділив актуальність проблеми і допоміг сформувати вимоги і виділити основні напрямки дослідження.

Представлено комплексний підхід до рішення проблеми, присвячений задачі створення та автоматизації процесу резервного копіювання даних в умовах роботи системі на базі мікросервісної архітектури. Розроблене рішення надає можливість інтеграції у все існуючу, функціонуючу систему. Перевагою рішення є гнучкість налаштування конфігурації запуску, що стає точками розширення і надає можливість широкого використання. Створено модель надійності резервування та забезпечення консистентної резервної копії даних мікросервісів.

В ході було розглянуто процес тестування на різних типах баз даних, досліджено дані, які дозволяє зберігати на поді оркестратор Kubernetes та можливість їх резервного копіювання. На практиці проведено перевірку дієздатності системи, а також можливість конфігурування параметрів для певної інфраструктури та отримати на виході цілісну резервну копію.

Отже, розроблено систему, яка створює консистентну резервну копію даних мікросервісів, тим самим підвищуючи загальну надійність рішення. Також описано програмну реалізацію та архітектуру системи. Описано її основні модулі, аргументовано вибір технологій. На прикладі показано основні сценарії використання системи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Xavier, M. G., Neves, M. V., Rossi, F. D., Ferreto, T. C., Lange, T., and De Rose, C. A. Performance evaluation of containerbased virtualization for high performance computing environments. In Parallel, Distributed and Network-Based Processing (PDP) / M. G. Xavier – 21st Euromicro International Conference, 2013. –240 с.
2. Eric Evans. Domain-Driven Design: Tackling Complexity in the Heart of Software / Eric Evans – Addison-Wesley, 2003. – 332 с.
3. What is kubernetes? - [Електронний ресурс] – Режим доступу до ресурсу: https://www.nomadproject.io/intro/index.html
4. Kubernetes components - [Електронний ресурс] – Режим доступу до ресурсу: https://kubernetes.io/docs/concepts/ overview/components
5. Backup and migrate Kubernetes resources and persistent volumes - [Електронний ресурс] – Режим доступу до ресурсу: https://velero.io/blog/velero-1.5-for-and-by-community/
6. Програма для резервного копіювання баз даних - [Електронний ресурс] – Режим доступу до ресурсу: https://www.handybackup.ru/database-backup.shtml
7. APBackup - утиліта для автоматичного резервного копіювання за розкладом - [Електронний ресурс] – Режим доступу до ресурсу: http://avpsoft.ru/products/apbackup/
8. Google Kubernetes Engine - [Електронний ресурс] – Режим доступу до ресурсу: https://cloud.google.com/kubernetes-engine
9. Sam Newman. Building Microservices: Designing Fine-Grained System / Sam Newman – O'Reilly, 2015. – 122 с.
10. Fowler, M., and Lewis, J. Microservices. - [Електронний ресурс] – Режим доступу до ресурсу: https://martinfowler.com/articles/microservices.html.

11. Krause, L. Microservices: Patterns and Applications: Designing finegrained services by applying patterns / L. Krause – El Autor, 2015.
12. Joshi, S. Organization & cultural impact of microservices architecture. In International Conference on Applied Human Factors and Ergonomics / S. Joshi – Springer, 2017. – 89–95 с.
13. Microservices: Five Architectural Constraints. - [Електронний ресурс] – Режим доступу до ресурсу: https://www.nirmata.com/2015/02/02/microservices-five-architectural-constraints/
14. Singleton, A. The economics of microservices / A. Singleton – IEEE Cloud Computing, 2016. – 16–20 с.
15. Balalaie, A., Heydarnoori, A., and Jamshidi, P. Microservices architecture enables devops: Migration to a cloud-native architecture / A. Balalaie, A. Heydarnoori, P. Jamshidi – IEEE Software , 2016. – 42–52 с.
16. Bass, L., Weber, I., and Zhu, L. DevOps: A software architect’s perspective / L. Bass – Addison-Wesley Professional, 2015. – 11 с.
17. Lewis, J., and Fowler, M. Microservices: a definition of this new architectural term / J. Lewis – Mars, 2014.
18. Pautasso, C., Zimmermann, O., Amundsen, M., Lewis, J., and Josuttis, N. M. Microservices in practice, part 2: Service integration and sustainability / C. Pautasso – IEEE Software, 2017. – 97–104 с.
19. Jaramillo, D., Nguyen, D. V., and Smart, R. Leveraging microservices architecture by using docker technology / D. Jaramillo – SoutheastCon, 2016. – 1–5 с.
20. Docker Engine. - [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/engine/docker-overview/ #docker-engine.
21. Е.С. Вентцель, Л.А. Овчаров: Теория случайных процессов и её инженерные приложения / Вентцель Е.С., Овчаров Л.А. – Наука, 1991.

ДОДАТОК А

Перелік публікацій

WORLD SCIENCE: PROBLEMS, PROSPECTS AND INNOVATIONS

Abstracts of VII International Scientific and Practical Conference

Toronto, Canada

24-26 March 2021

Toronto, Canada

2021

2

УДК 62-503.5

СПОСОБИ ЗАХИСТУ БАЗ ДАНИХ ВІД SQL ІН'ЄКЦІЙ ДЛЯ ЗАБЕЗПЕЧЕННЯ ЯКОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Літвінова Наталія Олександрівна
Цитовцева Анна Сергіївна
Сопов Олексій Олександрович
студенти
Національний технічний університет України
«Київський політехнічний інститут
імені Ігоря Сікорського»
м. Київ, Україна

Анотація: Дана робота розкриває причини та наслідки виникнення такого виду хакерських атак на автоматизовані програмні системи з базами даних (БД), як SQL ін'єкція. Такий вид атаки ставить під загрозу якість програмного продукту в цілому: його надійність, цілісність та захищеність, що може спричинити виникнення уразливості для роботи будь-якої організації. Тут також розглянуті найбільш ефективні методи для захисту систем, що взаємодіють з БД, від такого роду атак. Представлені рішення дають можливість забезпечити вищий рівень якості програмної продукції.

Ключові слова: бази даних, безпека інформаційних систем, SQL ін'єкція, захист від кібератак, конфіденційність інформаційних ресурсів.

Постановка проблеми в загальному вигляді. Сучасні організацій, діяльність яких пов'язана зі сферою товарів та послуг, активно користуються досягненнями технічного розвитку по всьому світу. Сфера розробки програмного забезпечення набула широкого поширення в сучасному житті для всіх напрямків бізнесу: воно застосовується, як для введення внутрішнього діловодства, так і для просування своєї діяльності на ринку, наприклад, у цілях

маркетингу або заради надання зручного способу взаємодії для клієнтів. На даний момент провідні компанії замовляють інформаційні системи для клієнтів, такі як CRM, програмне забезпечення для управління бізнес-процесами – ERP – та програми на основі робочих процесів. Якою б не була мета програми, будь-яке програмне забезпечення передбачає передачу даних, значна частина з яких зазвичай зберігається у базах даних (БД).

Оскільки, БД слугують основним джерелом, де зберігається дані пов'язані з бізнес-процесами будь-якої організації (в тому числі й конфіденційні дані), то під час розробки програмних систем, що взаємодіють з БД, необхідно враховувати ряд загроз. Під загрозою прийнято розуміти будь-які обставини або події, що можуть бути причиною порушення політики безпеки інформації і/або нанесення збитків автоматизованій системі. Спробу реалізації загрози називають атакою.

Тут хакерська атака (кібератака) — спроба реалізації загрози на програмну систему. Тобто, це дії кібер-зловмисників (хакерів) або шкідливих програм, які спрямовані на захоплення інформаційних даних віддаленого комп'ютера, отримання контролю над ресурсами комп'ютера або на виведення системи з ладу.

SQL ін'єкція – один з найпоширеніших способів кібератак на сайти та програми, що працюють з базами даних, заснований на впровадженні в запит довільного SQL-коду. Вони визнані одними з найстаріших і найбільш небезпечних атак для веб-додатків.

Мета статті. Основною метою статті є розкриття шляхів викриття даних та отримання несанкціонованого доступу до БД за допомогою такого виду атаки, як SQL ін'єкція, та приведення можливих способів запобігання загроз за допомогою визначених у даній статті правил розробки програмного забезпечення, що здатні мінімізувати ризики для діяльності підприємства, передчасно захистити бізнес-дані.

Виклад основного матеріалу.

SQL — це мова запитів, призначена для управління даними, що

зберігаються в реляційних базах даних [1]. Її можна використовувати для доступу, редагування та видалення даних. Багато веб-додатків і веб-сайтів зберігають всі дані в базах даних SQL таких як, наприклад, MySQL, Oracle, SQL Server, Postgres тощо. У деяких випадках можна також використовувати команди SQL для запуску команд операційної системи. Тому успішна атака SQL Injection може мати дуже серйозні наслідки [2, 3]:

- Зловмисники можуть використовувати ін'єкції SQL, щоб отримати облікові дані інших користувачів бази даних. Потім вони можуть видати себе за цих користувачів. В найгіршому випадку, такий користувач може «виявитися» адміністратором бази даних з усіма привілеями до бази даних.
- SQL дозволяє вибирати та виводити дані з бази даних. Уразливість SQL ін'єкцій може дозволити зловмиснику отримати повний доступ до всіх даних на сервері баз даних, включаючи особисту інформацію, паролі, комерційні дані та інтелектуальну власність.
- SQL також дозволяє змінювати дані в базі даних і додавати нові дані. Наприклад, у фінансовій програмі зловмисник може використовувати SQL ін'єкцію для зміни залишків, анулювання транзакцій або переказу коштів на свій рахунок.
- Ви можете використовувати SQL для видалення записів з бази даних, навіть відкидання таблиць. Навіть якщо ви робити резервні копії баз даних, видалення даних може вплинути на доступність програми до відновлення бази даних. Крім того, резервні копії можуть не охоплювати останні дані.
- На деяких серверах баз даних можна отримати доступ до операційної системи, використовуючи сервер баз даних. Це може бути навмисно або випадково. У такому випадку зловмисник може використовувати SQL ін'єкцію як початковий вектор, а потім атакувати внутрішню мережу за брандмауером.

Прихованою небезпекою тут є те, що SQL запити можуть контролювати сервер БД, про що ваша програма і не дізнається [4].

Аналіз останніх досліджень і публікацій. Згідно з даними, взятими з останньої статистики найпопулярнішої системи керування веб-сайтами – Wordpress – кількість випадків SQL ін'єкцій налічує 21% від усіх здійснених хакерських атак (рис. 1). Інші дослідження також єдині у своїй думці, що цей вид атак посідає почесне друге місце серед атак на веб-додатки.

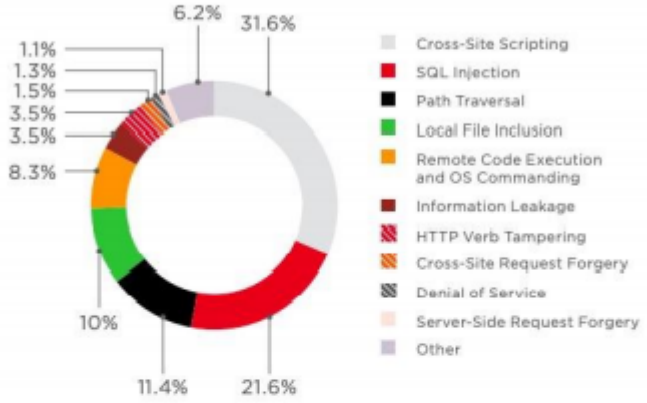

Рис. 1. Розподіл вразливостей за видом атак

Експертів з безпеки Wordpress стверджують наступні ключові висновки щодо загроз SQL ін'єкцій для веб-застосунків [5]:

1. Хоча більшість веб-застосунків проводять принаймні 4 кампанії з веб-атак на місяць, деякі сайти все ще піддаються атаці.

2. Електронна комерція переживає вдвічі більше атак SQL-ін'єкцій за інші галузі.

3. Кожен веб-сайт отримує 94 057 запитів на ін'єкцію SQL за один день. 94057 еквівалентно 1567 атакам SQLi на годину або 26 запитам на атаку за хвилину в середньому.

4. "Як зламати сайт WordPress за допомогою ін'єкції sql" — це широко шуканий запит, і його обсяг зріс на 200% у 2020 році. Це означає, що більше людей зацікавлені дізнатись більше про це.

Опис проблеми. Не зважаючи на таку статистику, багато розробники навіть не враховують таку загрозу і не зважають на те, що SQL-запити можуть

бути підроблені. Насправді такі запити, залежно від типу СКБД та умов впровадження, можуть обійти обмеження доступу, стандартну перевірку авторизації та аутентифікації чи дати можливість зловмиснику виконати довільний запит до бази даних (наприклад, прочитати вміст будь-яких конфіденційних даних, видалити, змінити або додати їх), отримати можливість читання та/або запису локальних файлів та виконання довільних команд на сервері.

Зловмисники можуть використовувати уразливості SQL ін'єкцій для різних цілей, проте основною причиною виникнення цього «слабкого місця» є зазвичай те, що вхідні дані від користувача не фільтруються (відсутня обробка спеціальних символів, перевірка типів тощо). Розглянемо для наочності наступні приклади.

Маємо форму пошуку товару. У якості параметрів пошуку є назва товару та верхня границя його ціни.

Приклад 1: У якості ціни користувач замість цифри «100» написав (не)навмисно «100'» (зверніть уваги на символ апострофа). Пошук за таким критерієм поверне непередбачену програмою помилку виконання. Отже, в даному випадку негативний наслідок – погіршення відмовостійкості та працездатності системи.

Приклад 2: У назву товару користувач включить свій SQL код, тобто основний запит на пошук буде містити внутрішній «злочинний» запит. Таким чином, крім виконання основної операції, передбаченої бізнес-логікою програми, виконуються інші несанкціоновані операції, які можуть порушити цілісність та конфіденційність даних, що зберігаються у БД, або навантажити БД так, що інші користувачі в кращому випадку будуть змушені довго очікувати виконання їх запитів.

На практиці не завжди можливо провести SQL ін'єкцію за допомогою полів для вводу даних, тому для веб-сайтів, які передають дані через GET-запити, наявний ще один випадок уразливості – редагування адресного рядку браузера. Наприклад, за допомогою SQL ін'єкції у адресному рядку хакер може

зчитати дані з будь-якої таблиці, використовуючи оператор UNION, наявний у більшості БД.

Результати. Для уникнення потенційних небезпек від такого виду загроз розробник програмних продуктів, що працюють з базами даних, повинен знати про таку уразливість і вживати заходів протидії впровадженню SQL на відповідному рівні [6].

Так, з урахування існуючих дослідження та тенденцій розробки програмного забезпечення, серед ефективних способів вирішення проблеми захисту від SQL ін'єкцій під час написання коду систем, виділено наступні:

– Використання параметризованих запитів

Багато серверів надають можливість відправки програмою параметризованих запитів (підготовлені вирази) до БД. Таким чином параметри зовнішнього походження відправляються на сервер окремо від самого запиту або автоматично екрануються клієнтською бібліотекою, що дозволяє автоматично уникнути шкідливих значень в параметрах.

– Фільтрація рядкових параметрів

Щоб впровадження коду було неможливо, для деяких СКБД, потрібно брати в лапки всі рядкові параметри. Так, у самому параметрі треба екранувати спецсимволи, тобто замінити лапки на \", апостроф на \', а зворотну косу риску на \\.

– Фільтрація цілочисельних параметрів

У випадку, коли параметром виступає цілочисельне значення, допомагає перевірка значення параметру на тип – якщо змінна не є числом, запит взагалі не повинен виконуватися.

– Усікання вхідних параметрів

Для внесення змін в логіку виконання SQL-запиту потрібно впровадження достатньо довгих рядків. Так, мінімальна довжина такого рядка найчастіше становить 8 символів («1 OR 1=1»). Якщо максимальна довжина коректного значення параметра невелика, то одним з методів захисту може

бути максимальне усікання значень вхідних параметрів.

– Зменшення кількості динамічних запитів

Також не варто забувати про запобіжні заходи, які ніколи не завадять для того, щоб захистити інформацію, що зберігається у БД. Тож, крім, контролю самих параметрів, розробник завжди має пам'ятати про обмеження доступу до системних таблиць, коректну обробку помилок та зберігання паролів у зашифрованому вигляді.

Висновки. Враховуючи викладений вище матеріал, забезпечення захисту даних у програмних системах з БД — важливий крок для покращення захищеності та цілісності бізнес-даних. Застосування приведених привентивних методів під час розробки застосунків матиме позитивний вплив на зменшення ризиків підприємств, що стосуються обмеження несанкціонованого доступу до джерел даних та захищеності конфіденційної інформації як організації, що придбала продукт, так і її клієнтів.

СПИСОК ЛІТЕРАТУРИ

1. Fehily C. SQL Database Programming / Chris Fihily. – Questing Vole Press, 2014.

2. Clarke J. SQL Injection Attacks and Defense / Justin Clarke. – Syngress Date, 2009.

3. Halde J. Basics of SQL Injection Analysis, Detection and Prevention / Jagdish Halde. – LAP LAMBERT Academic Publishing, 2014.

4. The Database Hacker's Handbook: Defending Database Servers / D. Litchfield, C. Anley, J. Heasman, B. Grindlay. – Wiley Date, 2005.

5. WordPress SQL injection – How to Fix & Prevent SQLi Hack [електронний ресурс]: https://secure.wphackedhelp.com/blog/wordpress-sql-injection-hack/.

6. Марков А. С., Миронов С. В., Цирлов В. Л. Выявление уязвимостей в программном коде // Открытые системы. 2005, № 12.

УДК 62-503.5

Цитовцева А.С.,
Сопов О.О.

АНАЛІЗ ДОСТУПНОСТІ МІКРОСЕРВІСІВ НА БАЗІ СИСТЕМИ УПРАВЛІННЯ ТА ОРКЕСТРАЦІЇ КОНТЕЙНЕРІВ KUBERNETES

Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського"

Постановка проблеми в загальному вигляді

Перехід до мікросервісної архітектури вже триває. Однак, як важливий атрибут якості для послуг оператора, доступність залишається проблемою. Доступність - це нефункціональна характеристика, що визначається як обсяг відключення послуги протягом певного періоду [5]. Висока доступність досягається, коли система доступна принаймні в 99,999% випадків. Отже, загальний час простою, дозволений протягом одного року для високодоступних систем, становить близько 5 хвилин [3]. Деякі характеристики мікросервісів та контейнерів, такі як малі та легкі, природно сприяли б підвищенню доступності [1]. Kubernetes забезпечує зцілення своїх керованих мікросервісних програм. Можливість відновлення Kubernetes полягає у перезапуску невдалих контейнерів та заміні або перепланованні контейнерів, коли їхні господарі виходять з ладу. Цілюща здатність також відповідає за відновлення нездорових контейнерів, поки вони знову не будуть готові. Ці функції також би природно покращили доступність послуг, що надаються програмами, розгорнутими Kubernetes. Питання в тому, яка доступність надається цими програмами?

Аналіз останніх досліджень і публікацій

Протягом останнього десятиліття з'явилась загальна тенденція до міграції до хмари [1]. У цьому контексті архітектурний стиль мікропослуг [2] привернув значну увагу. На відміну від монолітного архітектурного стилю, мікросервісна архітектура вирішує проблеми побудови власних хмарних додатків, використовуючи переваги хмари [3]. Незважаючи на те, що цей архітектурний стиль готовий здійснити революцію в IT-галузі, до цього часу він отримував обмежену увагу з боку наукових кіл.

Мета статті

У цій роботі було оцінено програми з мікросервісною архітектурою з точки зору доступності, оскільки наша кінцева мета – забезпечити високу доступність мікросервісів. Як продовження для початкової установки в приватній хмарі та конфігурації Kubernetes за замовчуванням, було досліджено інші архітектури, конфігурації та проведено серію експериментів з Kubernetes, виміряно час відключення для різних сценаріїв відмов. Метою було дати відповідь на такі дослідницькі запитання:

- Який рівень доступності Kubernetes може підтримувати для своїх керованих мікропослуг виключно завдяки своїм властивостям відновлення?
- Який вплив додавання надмірності на доступність можна досягти за допомогою Kubernetes?
- Якої доступності може досягти Kubernetes за найефективнішої конфігурації?
- Як доступність, досягнута за допомогою Kubernetes, порівнюється з існуючими рішеннями?

Експерименти було проведено в конфігурації Kubernetes за замовчуванням, а також у найбільш реагуючій. Для кращого позиціонування та характеристики отриманих результатів було обрано порівняння з існуючим рішенням для управління доступністю, Framework Management Framework (AMF) [1], перевіреною послугою проміжного програмного забезпечення для управління високою доступністю (HA).

Виклад основного матеріалу

1. Розгортання контейнерів у кластері Kubernetes, що працює в загальнодоступній хмарі.

У цьому розділі буде розглянуто кластер Kubernetes, що складається з віртуальних машин, що працюють у загальнодоступній хмарі. Kubernetes працює на всіх віртуальних машинах і створює єдиний вигляд кластера. Одна з віртуальних машин вибрана в якості ведучої, і вона відповідає за управління вузлами. Оскільки ми стурбовані високою доступністю, нам слід розглянути кластер HA, що складається з більш, ніж одного ведучого. Однак така установка все ще є експериментальною та незрілою для Кубернетеса. Таким чином, ми вирішили піти лише з одним майстром і не допустити відмови з боку майстра. Для простоти додаток тут складається лише з одного мікросервісу. Шаблон pod для контейнерної мікросервіси, а також бажана кількість реплік включені до специфікації контролера розгортання, яка розгортається в кластері. Ми обговоримо два способи виставлення послуг у кластерах Kubernetes, що працюють у загальнодоступній хмарі.

Сервіс типу Load Balancer: архітектура для розгортання програм у кластері Kubernetes з використанням служби типу Load Balancer в загальнодоступній хмарі показана на рис. 1. На додаток до IP кластера, послуги типу Load Balancer мають зовнішню IP-адресу, яка автоматично встановлюється як IP-адреса балансира навантаження хмарного провайдера. Використовуючи цю зовнішню IP-адресу, яка є загальнодоступною, можна отримати доступ до подів ззовні кластера.

Ingress: Може бути більше однієї служби, яку потрібно піддавати зовнішньому впливу, і за принципом попереднього методу, для кожного сервіру потрібен один балансир навантаження. З іншого боку, вхідний ресурс Kubernetes може мати декілька сервісів у якості резервних копій та мінімізувати кількість балансувальних навантажень [3]. У кластері Kubernetes, що працює в загальнодоступній хмарі, контролер входу розгортається та виставляється службою типу Load Balancer [3]. Отже, запити на всі послуги, що надсилаються на балансир навантаження хмарного провайдера, отримуються контролером входу та перенаправляються на відповідну службу на основі правил, визначених у ресурсі входу

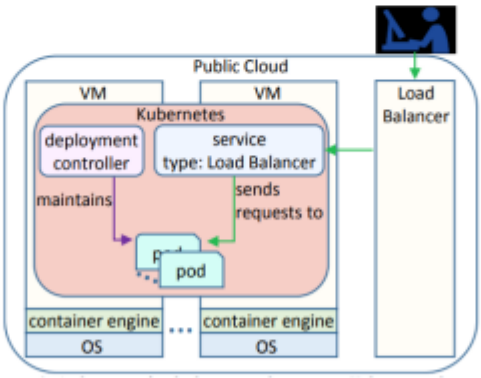

Рис. 1. Архітектура для розгортання програм на кластерах Kubernetes, що працюють у загальнодоступній хмарі

2. Розгортання контейнерів у кластері Kubernetes, що працює у приватній хмарі.

У цьому розділі буде описано налаштування експериментів, розглянуті сценарії відмов, а також показники доступності.

Ми встановили кластер Кубернетеса у приватній хмарі. Цей кластер складається з трьох віртуальних машин, що працюють у хмарі OpenStack. Ubuntu 16.04 - це ОС, що працює на всіх віртуальних машинах. Kubernetes 1.8.2 працює на всіх віртуальних машинах, а контейнерний двигун - Docker 17.09. Протокол мережевого часу (NTP) використовується для синхронізації часу між вузлами. Розгорнутою програмою є VideoLan Client (VLC). У кожному темплейті пода зазначено імедж, на якому встановлено VLC. Після розгортання пода на основі цього імеджа буде створений контейнер програми, який почне потокове передавання з файлу.

Kubernetes пропонує три рівні перевірки працездатності та механізмів для управління доступністю розгорнутих мікросервісів. По-перше, на рівні додатків Kubernetes гарантує, що програмні компоненти, що виконуються всередині контейнера, справні або за допомогою перевірки стану процесу, або заздалегідь визначених проб. В обох випадках, якщо Kubelet виявляє несправність, контейнер перезапускається. По-друге, на рівні поду Kubernetes відстежує відмови підсистеми та реагує відповідно до визначеної політики перезапуску. Нарешті, на рівні вузла, Kubernetes контролює вузли кластера через розподілені демони для виявлення відмов вузлів. Якщо вузол, що розміщує под, виходить з ладу, под переплановується на інший здоровий вузол. Що стосується цих рівнів перевірки працездатності, було зазначено три набори сценаріїв відмов. У першому наборі помилка програми спричинена помилкою процесу контейнера VLC. У другому наборі це пов'язано з відмовою процесу контейнера для подів, а в третьому наборі - через збій вузла. Для кожного набору експериментувалося з різними моделями надмірності, а також з конфігурацією Kubernetes за замовчуванням та найбільш адаптивною. Кожен сценарій повторювався 10 разів, і середнє значення вимірювань показано в Таблицях I - Таблиці V. Усі вимірювання, про які повідомляється в цьому документі, складаються у секундах.

Таблиця I – Експеримент з Кубернетисом – антинадлишкова модель та конфігурація за замовчуванням

Тригер відмови (одиниця: секунди)	Час реакції	Час репарації	Час відновлення	Час простою
Відмова контейнера VLC	0.716	0.472	1.050	1.766
Відмова контейнера пода	0.496	32.570	31.523	32.019
Відмова ноди	38.187	262.542	262.665	300.852

Час реакції: виявлено час між подією відмови, яку ми вводимо, і першою реакцією Кубернетеса, що відображає подію відмови.

Час репарації: Час між першою реакцією Кубернетеса та відновленням пода.

Час відновлення: Час між першою реакцією Kubernetes і часом, коли сервіс знову доступний.

Час простою: тривалість, протягом якої сервіс був недоступний. Він представляє суму часу реакції та часу відновлення

3. Експерименти, результати та аналіз

У цьому розділі представлено архітектури, експерименти, результати та аналіз для відповіді на питання дослідження, яке було поставлено у вступі.

Оцінка дій відновлення за допомогою конфігурації Kubernetes за замовчуванням для підтримки доступності

Рис. 2 показує архітектуру цих експериментів. Модель надмірності в даному випадку не є збитковою [2], а отже, кількість стручків у специфікації контролерів розгортання лише одна.

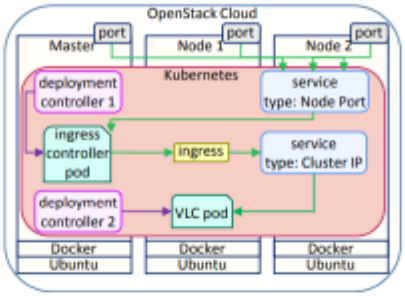

Рис. 2. Конкретна архітектура для розгортання програм за допомогою Kubernetes - модель надмірності без збитковості.

Збої в роботі через збій процесу контейнера VLC.

У цьому сценарії збій моделюється шляхом вимкнення процесу контейнера VLC з ОС. Коли контейнер VLC аварійно завершує роботу, Kubelet виявляє збій і приводить стручок до стану, коли він не отримуватиме нових запитів. У цей час, тобто час реакції, под вилучається зі списку кінцевих точок. Пізніше Kubelet перезапустить контейнер VLC, і відео почнеться з початку файлу. Цей час знаменує час ремонту. Час відновлення - це коли под знову потрапляє до списку кінцевих точок і готовий приймати запити.

Збої в роботі сервісу через збій процесу контейнера Pod.

Коли розгортається под, разом із контейнерами додатків, зазначеними в його шаблоні, створюється один додатковий контейнер, який є контейнером pod. Оскільки контейнер pod є процесом в ОС, можливо, він аварійно завершує роботу. У цьому випадку помилка моделюється шляхом вимкнення процесу контейнера підсистем з ОС. Коли процес контейнера для пода вбивається, Kubelet виявляє, що контейнер для пода відсутній, і це позначає час реакції. Коли новий под створено та запущено його контейнер VLC, відео почне транслюватися з початку файлу, і ми вважаємо под відремонтованим. Після цього Kubelet додасть новий под до списку кінцевих точок, і він буде готовий приймати нові запити, це означає час відновлення.

Збої в роботі служби через збій вузла.

У цьому випадку відмова вузла моделюється командою перезавантаження Linux на віртуальній машині, на якій розміщено под. Як вже згадувалося раніше, Kubelet відповідає за повідомлення звіту про стан вузла ведучим, і саме контролер вузла ведучого виявляє несправність вузла. Коли вузол, що розміщує под, виходить з ладу, він припиняє надсилання оновлень статусу ведучому, і майстер позначить вузол як не готовий після четвертого пропущеного оновлення стану. Цей час - час реакції. Коли вузол позначений як не готовий, підсистему VLC на вузлі планується припинити, а після його завершення буде створено новий. Час ремонту - це час запуску нового струму VLC і потокового передавання відео. Час відновлення - це коли под додається до списку кінцевих точок служби.

Результати та аналіз

Вимірювання та події цього набору експериментів наведені у таблиці I та на рис. 3 відповідно. На рис. 3 відмова контейнера VLC, контейнера pod або вузла, що розміщує Pod1, показана як перша подія. До цієї події IP-адреса Pod1 була в списку кінцевих точок, і сервіс був доступним. Після несправності сервіс стає недоступним. Однак, оскільки Kubernetes ще не виявив помилку, IP-адреса Pod1 залишається у списку кінцевих точок. Він вилучається зі списку кінцевих точок під час реакції.

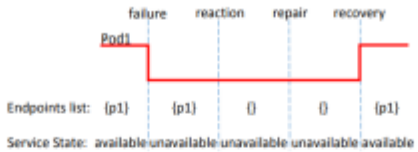

Рис. 3. Аналіз експериментів з Kubernetes з конфігурацією за замовчуванням та моделлю без збитковості – оцінка дій відновлення.

У даній архітектурі видалення IP-адреси Pod1 зі списку кінцевих точок як реакція Kubelet на збій контейнера VLC займає 0,716 секунди, а у випадку відмови контейнера pod - 0,496 секунди. Однак у разі відмови вузла, видалення IP od1 зі списку кінцевих точок, як реакція контролера вузла ведучого вимірювалася як 38,187 секунд. Причина полягає в тому, що за замовчуванням конфігурації Kubernetes, ведучому потрібно щонайменше 30 секунд, щоб виявити збій вузла. Оскільки Kubelet оновлює стан вузла кожні 10 секунд, а майстер дозволяє чотири пропущені оновлення стану, перш ніж позначити вузол як не готовий.

Час відновлення для всіх сценаріїв – це коли новий блок створюється знову і потокове відео починається знову. Як зазначається в таблиці I, час відновлення контейнера VLC або сценарії відмови контейнера пода суттєво відрізняються (0,472 секунди для першого та 32,570 секунд для останнього). Причина полягає в тому, що у випадку відмови контейнера pod, сигнал завершення надсилається до контейнера VLC, і Docker чекає 30 секунд, щоб він закінчився. Процес відновлення не розпочнеться, якщо контейнер VLC не буде припинено. Для сценарію виходу з ладу, як показано в таблиці I, час відновлення значно вищій. Причина полягає в тому, що за замовчуванням конфігурації Kubernetes, у разі відмови вузла, майстер чекає близько 260 секунд, щоб запустити новий под і відновити службу. Через такі високі терміни відновлення відключення обслуговування для сценаріїв відмови контейнера і вузла є значно вищіми - 32,019 секунди та 300,52 секунди відповідно.

Висновки

В рамках даної роботи було представлено та порівняно архітектури для розгортання програм на базі мікросервісів у кластерах Kubernetes, розміщених у загальнодоступних хмарах. Проведено експерименти в хмарному середовищі, розглядаючи різні сценарії відмов, конфігурації, моделі резервування щоб оцінити Kubernetes з точки зору доступності програм, що працюють на Kubernetes. Було проаналізовано результати експериментів і виявлено, що дії для відновлення, що забезпечує Kubernetes недосттні для забезпечення доступності, особливо високої доступності. Наприклад, конфігурація Kubernetes за замовчуванням призводить до значного відключення у разі відмови вузла. Kubernetes можна переконфігурувати, щоб уникнути цього значного відключення.

Література

1. *Sam Newman.* Building Microservices: Designing Fine-Grained System. – O'Reilly, 2015. – 251 p.

2. Design Patterns: Elements of Reusable Object-Oriented Software / Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. – O'Reilly, 2004. – 694 p.

3. *Pethuru Raj* Chelliah. Service Discovery and API Gateways. – Essentials of Microservices Architecture, 2019

4. *Eric Evans.* Domain-Driven Design: Tackling Complexity in the Heart of Software. – Addison-Wesley, 2003. – 560 p.

5. *Robert Martin*. Clean Code: A Handbook of Agile Software Craftsmanship. – Addison-Wesley, 2008. – 465 p.

Цитовцева А. С., Сопов О. О.

АНАЛІЗ ДОСТУПНОСТІ МІКРОСЕРВІСІВ НА БАЗІ СИСТЕМИ УПРАВЛІННЯ ТА ОРКЕСТРАЦІЇ КОНТЕЙНЕРІВ KUBERNETES

У статті досліджено проблеми оркестрації та проведено експерименти, щоб оцінити доступність, яку надає Kubernetes для керованих мікропослуг. Значну увагу приділено впливу додавання надмірності на доступність програм на базі мікросервісної архітектури та проведено експерименти з конфігурацією Kubernetes за замовчуванням, а також з найефективнішою. Перехід до архітектури мікропослуг триває і зараз. При такому підході система виділяється на менші модулі, які розроблені, розроблені та масштабовані окремо для побудови віртуалізованої функції. Але для постачальників функцій доступність залишається проблемою при переході до розгортання мікросервісів. Kubernetes - це рішення, яке має базу коду з відкритим кодом, воно визначає вибір розгорнутих частин, які разом надають інструменти для побудови, підтримки, масштабування та відновлення контейнерних функцій. Таким чином, Kubernetes приховує складність організації мікросервісів, щоб забезпечити управління їх доступністю. Для початку ми оцінюємо Kubernetes, використовуючи загальну конфігурацію з точки зору доступності в хмарних налаштуваннях. Представлені архітектури для державних та приватних хмар. Ми оцінюємо доступність, яка досягається цілющою силою Кубернету. Порівняльна оцінка була проведена за допомогою Framework Management Framework (AMF), тобто запропонованого механізму, що використовується в проміжному програмному забезпеченні для управління високою доступністю. Результати дослідження демонструють, що в деяких тестах відключення служби для програм, керованих Kubernetes, є значно високим.

Ключові слова: *мікросервіси, контейнери, оркестрація, докер, доступність.*

Tsytovtseva A., Sopov O.

ANALYSIS OF THE AVAILABILITY OF MICROSERVICE BASED ON THE MANAGEMENT AND ORCHESTRATION SYSTEM OF CONTAINERS KUBERNETES

The article explores problems of orchestration and executing test of performance to assess the availability of Kubernetes for supervised services. Considerable attention is paid to the sequence of managing exorbitance, on the availability of programs with on microservice architecture and tests have been conducted with the standard settings of Kubernetes and also with the most efficient. The move to a microservices architecture continues now. In this kind of approach the system is separeted to smaller modules which are designed, developed, and scaled separetely to build virtualized function. But, for function suppliers accessibility remains an issue in the transition to deploying microservices. Kubernetes is a solution which has code base an open source, it defines a selection of deployed parts that together gives instruments for building, supporting, scaling, and recovering containerized functions. In this way, Kubernetes hides the complexity of orchestrating microservices to provide management of their approachability. To start with, we estimate Kubernetes usinggeneral configuration in terms of approachability in cloud settings. It is presented architectures for public and private clouds. We are evaluating the availability achieved by the healing power of Kubernetes. A comparative assessment was executed using the Availability Management Framework (AMF), that is a suggested mechanism used in intermediate software for high availability management. The results of the research demonstrate that in some tests, disabling the service for applications managed by Kubernetes is significantly high.

Keywords*: microservices, containers, orchestration, docker, availability.*

ISSN 2411-5363 (print)
ISSN 2519-4569 (online)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЧЕРНІГІВСЬКА ПОЛІТЕХНІКА»

ТЕХНІЧНІ НАУКИ ТА ТЕХНОЛОГІЇ

НАУКОВИЙ ЖУРНАЛ

№ 1(23)

Чернігів 2021

ЗМІСТ

ТЕХНІЧНІ НАУКИ ТА ТЕХНОЛОГІЇ № 1(23), 2021

TECHNICAL SCIENCES AND TECHNOLOGIES

УДК 62-503.5
DOI: 10.25140/2411-5363-2021-1(23)-103-108

Олексій Сопов, Анна Цитовцева

ОСОБЛИВОСТІ МАСШТАБУВАННЯ КОНТЕЙНЕРНОГО НАВАНТАЖЕННЯ НА БАЗІ СИСТЕМИ KUBERNETES

На сьогодні технологія контейнеризації набуває широкого поширення, та проблема масштабування є однією з найбільш важливих для підвищення продуктивності систем. Kubernetes є передовим рішенням для керування контейнерами, проте проблема масштабування залишається слабо описаною та дослідженою. У даній роботі досліджено особливостей масштабування контейнерного навантаження на базі системи Kubernetes. Розкриті основні операції для досягнення горизонтального та вертикального масштабування. Описані властивості масштабованості системи Kubernetes. Стаття є оглядовою.

Ключові слова: *Kubernetes, мікросервіс, контейнерне навантаження, масштабування, Docker.*
Рис.: 4. Бібл.: 6.

Актуальність теми дослідження. На сьогодні виконання програмних застосунків в умовах контейнерної віртуалізації надає ряд переваг, як з фінансової сторони, так і зі сторони продуктивності, відмовостійкості та швидкості роботи. Тому, сфера розробки програмного забезпечення з використанням технології контейнеризації набуває широкого поширення в багатьох сферах бізнесу

З огляду на часті зміни навантаження на програмні продукти і різні умови їх функціонування, завдання масштабування корисного навантаження стає все більш актуальним. Проблема масштабування є однією й найбільш важливих для розширення обсягу виконаних задач за період часу, тому важливо використовувати правильні схеми масштабування застосунків.

Kubernetes є передовим рішенням при роботі із контейнерами та забезпечує відносно легкі для адаптації механізми керування, проте проблема масштабування залишається слабо описаною та дослідженою, що підвищує складність переходу бізнесу на контейнерні технології, зокрема, із використанням Kubernetes.

Постановка проблеми. В останній час триває перехід від класичної монолітної архітектури побудови систем до сервіс-орієнтованої, або ж, найчастіше, до мікросервісної архітектури задля забезпечення високого рівня доступності та відмовостійкості. Однією з найбільших переваг використання такого підходу є можливість масштабування для досягнення адекватної реакції, з огляду на часті зміни навантаження та різні умови функціонування системи в цілому.

Беручи до уваги переваги використання контейнерів, окремим мікросервісом найчастіше виступає один, або група об'єднаних контейнерів, які виконують ту, чи іншу задачу. Проте, керування такою системою з плином часу стає доволі складним, та вимагає значних зусиль. Сервіс Kubernetes [1] виконує більшість задач із керування контейнерами, їх життєвим циклом та версіями, тому є одним із найзручніших у даній області.

У Kubernetes корисне навантаження та інфраструктура концептуально розділені, тому є можливість виконувати масштабування на кожній складовій окремо, що призведе до появи більш ніж двох операцій для досягнення масштабування. Хоча система Kubernetes є достатньо гнучкою, проте вона була створена нещодавно, та такі особливості масштабування із розділенням концепцій є досі не описаними.

Аналіз останніх досліджень і публікацій. З аналізу літературних джерел можна дійти висновку, що в останній час з'явилась загальна тенденція міграції до хмари та використання віртуальних машин [2]. Проблема оптимального масштабування із використанням віртуальних машин вирішена достатньо широко [3, 4]. Архітектурний стиль мікропослуг та використання контейнерного навантаження привернули значну увагу, особливо на базі системи Kubernetes.

Виділення недосліджених частин загальної проблеми. Можна зробити висновок, що Kubernetes надає значні переваги у швидкості розробки та розгортанні застосунків на базі контейнерів, проте проблема масштабування до цього часу не отримувала значної уваги з боку наукових кіл.

Мета статті є дослідження особливостей масштабування контейнерного навантаження на базі системи Kubernetes. Розкрити основні операції для досягнення горизонтального та вертикального масштабування із розділенням концепцій корисного навантаження та інфраструктури. Описати властивість масштабованості системи Kubernetes.

Виклад основного матеріалу. У класичному розумінні масштабування застосунків поділяється на вертикальне та горизонтальне. Ці вектори масштабування є основоположними та мають різні варіації у різних системах.

Горизонтальне масштабування полягає у збільшенні кількості одиниць, що містять у собі додаток. Виконується розбиття системи на більш дрібні структурні компоненти та рознесення їх по окремим обчислювальним одиниця, або їх групам, і збільшення кількості одиниць, що паралельно виконують одну і ту ж функцію.

Вертикальне масштабування полягає у збільшенні ресурсів одиниці, на якій виконується застосунок. Тобто, збільшення продуктивності кожного компонента системи з метою підвищення загальної продуктивності.

При роботі в умовах контейнерної віртуалізації необхідно розуміти основні змінні, які можуть використовуватися для горизонтального та вертикального масштабування. У даному випадку одиницею, що виконує застосунок є, безпосередньо, контейнер, який розгорнуто із деякого зображення контейнера, тобто деяка ізольована оболонка вказаного програмного коду та його залежностей.

Горизонтальне масштабування у випадку із контейнерами полягає в наступному: додається новий вузол, а саме, додається новий контейнер, який розпочато із того самого зображення, з якого розпочато і перший контейнер. Контейнер може бути розгорнуто на будь-якій фізичній, або віртуальній машині, де досягнуті необхідні умови, наприклад, встановлено Docker daemon при роботі із системою контейнерної віртуалізації Docker [5].

Вертикальне масштабування полягає в зміні мінімальної/максимальної кількості ресурсів для контейнера. Можливість обмежити використання ресурсів окремим контейнером ж є однією з найважливіших складових контейнерної віртуалізації. Під ресурсами мається на увазі: процесорний час (CPU), кількість оперативної пам'яті (RAM), ресурси диску (iops). Обмеження ресурсів є необхідністю, адже необхідно контролювати максимальну кількість, яку використовує контейнер, щоб не допустити ситуації, коли один контейнер займає більшу частину ресурсів. За замовчуванням кількість ресурсів у контейнера необмежена, проте є можливість це налаштовувати за допомогою параметрів --cpus, --memory, --device-read-bps тощо.

Незважаючи на простоту розгортання контейнеру на будь-якій обчислювальній одиниці за допомогою спеціалізованих програмних засобі, таких як Docker, перед розробниками зазвичай стають більш складні задачі: підтримка великої кількості таких контейнерів, підтримка контейнерів різного розміру, різного навантаження та їх горизонтальне та вертикальне масштабування.

Для вирішення більшості вищенаведених задач були розроблені платформи, такі як Kubernetes, що значно полегшують роботу із великою кількістю контейнерів та їх контролем. Основні структурні компоненти Kubernetes зображено на рисунку 1 (задля простоти більшу частину, яка не приймає участь у масштабуванні було опущено).

Тобто, контейнери виконуються у середині так званих под (англ. Pod), що є найменшими одиницями керування у Kubernetes. Найчастіше, один под містить в собі лише один контейнер. Тобто, у Kubernetes контейнер знаходиться всередині окремої найменшої одиниці, що може бути розгорнута, пода. Саме Pod є об'єктом для керування при масштабуванні в Kubernetes.

У Kubernetes є можливість керувати ресурсами контейнера в середині поду. Аналогічно, як і у Docker можливо керувати основними ресурсами, а саме: процесорний час (CPU), кількість оперативної пам'яті (RAM), ресурси диску (iops) [6]. Проте, особливість роботи Kubernetes полягає в тому, що необхідно вказувати два параметри для ресурсів. Перший — ліміт, тобто, скільки максимально ресурсів буде використано контейнером. Другий — запит, тобто, скільки мінімально ресурсів необхідно для роботи контейнера та без яких ресурсів його не буде розгорнуто на кластері.

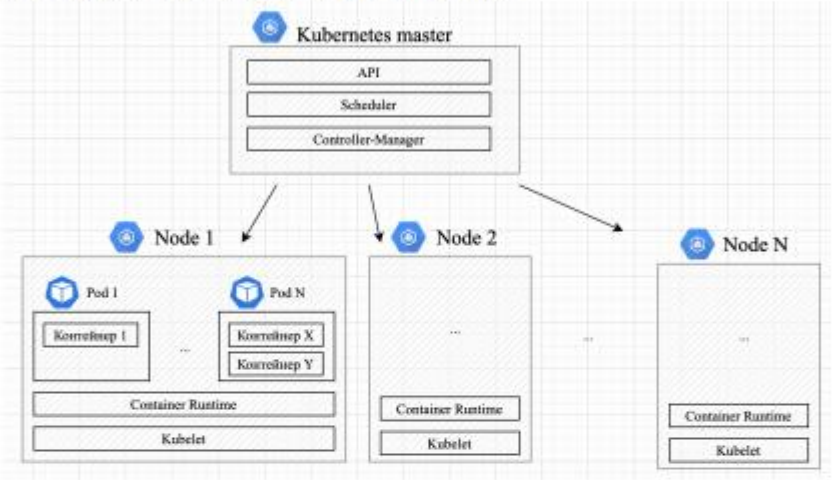

Рис. 1. Схема роботи Kubernetes

Тобто, однією з одиниць керування є ресурси контейнера всередині под: процесорний час, кількість оперативної пам'яті та ресурси диску, що можливо використовувати при виконанні масштабування.

З іншої сторони, поди розгортаються на так званих нодах. Нодами можуть виступити фізичні, або віртуальні машини, в залежності від конфігурації кластера. Фізичні та віртуальні машини мають власну ресурсну ємність, та визначають максимальну кількість под, що може бути розгорнута на них. Основними ресурсами керування для под є класичні технічні характеристики фізичної, або віртуальної машини: процесорний час, кількість оперативної пам'яті та ресурси диску.

Ресурси контейнера та ресурси ноди описують необхідні ресурси для заданого навантаження та ресурсну ємність кластера, та є основними змінними у роботі із кластером. Тому, для масштабування використано саме ці змінні.

Тобто, підсумовуючи, завдяки концепції Kubernetes у розділенні інфраструктури та додатку, два класичних типи масштабування: горизонтальне та вертикальне розширюються у двох розрізах: у розрізах інфраструктури(зміна ресурсів нод у кластері та їх кількості) та корисного навантаження(зміна ресурсів контейнерів та їх кількості).

Горизонтальне масштабування корисного навантаження в Kubernetes полягає у зміні кількості под із одним і тим самим додатком, тобто, його реплікація. Тобто, один і той самий додаток буде виконуватися паралельно, у різних подах, завдяки чому підвищується продуктивність та максимальна пропускна здатність.

Зі сторони інфраструктури горизонтальне масштабування полягає у додаванні нових нод до кластеру, на яких можливе буде виконання нових под. Тобто, збільшується кількість ресурсів, на яких можуть бути розташовані поди.

Масштабованість в цьому контексті означає можливість додавати до системи нові вузли: ноди та поди для збільшення загальної продуктивності. Це найпростіший спосіб масштабування, так як не вимагає ніяких змін в прикладних програмах, що працюють на таких системах. Схему горизонтального масштабування у Kubernetes у двох розрізах показано на рисунку 2. Як видно, основними компонентами для горизонтального масштабування є поди (у розрізі горизонтального масштабування корисного навантаження) та ноди (у розрізі горизонтального масштабування інфраструктури).

Отже, горизонтальне масштабування у Kubernetes може проводитися у двох розрізах: корисного навантаження та інфраструктурного та у двох сегментах: зміні кількості под та зміні кількості нод.

Рис. 2. Схема горизонтального масштабування у Kubernetes

Вертикальне масштабування корисного навантаження в Kubernetes полягає у зміні кількості ресурсів, які може зайняти контейнер. Тобто, у зміні кількості дозволеного використання процесорного часу, оперативної пам'яті, мережевих та дискових ресурсів.

Вертикальне масштабування у розрізі інфраструктури полягає у зміні типу віртуальної машини, на якій виконуються контейнери, тобто, зміні потужності та кількості оперативної пам'яті на даній віртуальній машині, що дозволить змінити кількість контейнерів, що можуть бути розгорнуті.

Масштабованість в цьому контексті означає можливість замінювати в існуючій системі автоматичного компоненти більш потужними і швидкими в міру зростання вимог і розвитку технологій. Цей спосіб масштабування може вимагати внесення змін до програми, щоб програми могли повною мірою користуватися дедалі більшій кількості ресурсів. Схему вертикального масштабування у Kubernetes у двох розрізах показано на рис. 3.

Отже, вертикальне масштабування у Kubernetes може проводитися у двох розрізах: корисного навантаження та інфраструктурного у двох сегментах: зміні ресурсів одиниці поду та зміні ресурсів одиниці ноди.

Дані схеми масштабування є основою для планування ресурсної ємності під задане навантаження в умовах контейнерної віртуалізації на платформі Kubernetes, тому можуть бути операціями для стратегій планування ресурсів та складовими у різних алгоритмах планування.

Рис. 3. Схема вертикального масштабування у Kubernetes

Підсумовуючи вищенаведені особливості масштабування, можна дійти висновку, що класичне масштабування (вертикальне та горизонтальне) у Kubernets виконується за допомогою чотирьох незалежних операцій (схематично такі операції показано на рис. 4):

- зміна кількості под (горизонтальне);
- зміна кількості нод (горизонтальне);
- зміна розміру контейнера, та, відповідно поду (вертикальне);
- зміна розміру машини, на якій підіймається контейнери (вертикальне).

Масштабованість Kubernetes проявляється у двох розрізах: корисного навантаження та інфраструктури. Масштабованість корисного навантаження полягає у можливості додати нові вузли, тобто, поди та ноди, а вертикального – у відповідному збільшенні ресурсів вузлів.

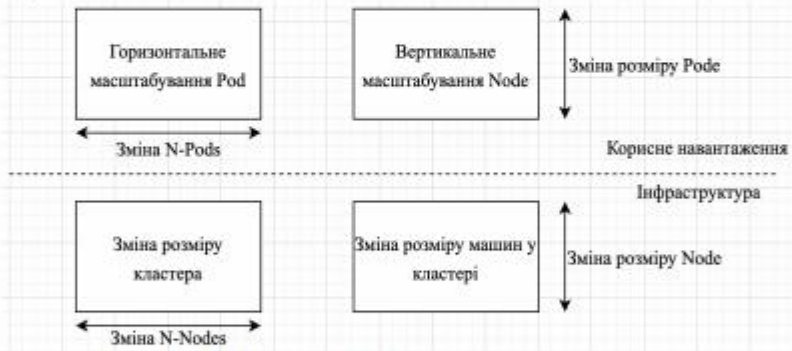

Рис. 4. Операції для досягнення масштабування у Kubernetes

Висновки. У даній статті були досліджені та описані особливості масштабування контейнерного навантаження на базі системи Kubernetes із розділенням концепцій масштабування інфраструктури та корисного навантаження.

Досліджено чотири основних операції масштабування у Kubernetes: у двох розрізах (корисного навантаження та інфраструктури) та у двох напрямках (горизонтально та вертикально). Горизонтальне масштабування досягається шляхом зміни кількості контейнерів, та, відповідно, под та зміні кількості машин у кластері, на яких можуть бути розгорнуті поди. Вертикальне масштабування полягає у зміні кількості ресурсів, як для под, так і для машин кластера.

Список використаних джерел

1. Офіційний портал Kubernetes. URL: https://kubernetes.io/.
2. Sam Newman. Building Microservices: Designing Fine-Grained System / Sam Newman – O'Reilly, 2015. – 251 с.

3. Теленик С. Ф. Управление распределением виртуальных машин в цод [Електронний ресурс] / С. Ф. Теленик, А. І. Ролик, Е. В. Жаріков. URL: https://ela.kpi.ua/bitstream/123456789/20434/1/64-13-Telenyk.pdf.

4. Жаріков Е. В. Динамічне розміщення віртуальних машин на основі навчання з підкріпленням в хмарних центрах обробки даних / Е. В. Жаріков, А. А. Коваль, Р. А. Терентьєв. // Наукові вісті Далівського університету. – 2017. – №13.

5. Офіційний портал Docker. URL: https://www.docker.com/.

6. Конфігурація ресурсів для Docker. URL: https://docs.docker.com/config/containers/resource_constraints/.

References

1. Kubernetes official portal. https://kubernetes.io/.

2. Sam Newman. Building Microservices: Designing Fine-Grained System / Sam Newman – O'Reilly, pp.251, 2015.

3. Telenyk S., Rolik A., Jarikov E., Managing the distribution of virtual machines in the data center. Retrieved from https://ela.kpi.ua/bitstream/123456789/20434/1/64-13-Telenyk.pdf.

4. Jarikov E., Koval A., Terentiev R., Dynamic deployment of virtual machines based on training with reinforcement in cloud data centers. *Scientific news of Daliv University*, №13, 2017

5. Docker official portal. https://www.docker.com/.

6. Docker resources configuration. https://docs.docker.com/config/containers/resource_constraints/.

UDC 62-503.5

Oleksii Sopov, Anna Tsytovtseva

FEATURES OF SCALING CONTAINER LOAD IN KUBERNETES SYSTEM

Today, the field of software development using container technology is becoming widespread in many areas of business, and the problem of scaling is one of the most important to achieve the expansion of the volume of tasks performed over time, so it is important to use the right scaling schemes. Kubernetes requires relatively easy-to-adapt container management mechanisms, but the problem of scaling them remains poorly described and researched, which increases the complexity of the business transition to container technologies and, in particular, the Kubernetes system.

Thanks to the separation of infrastructure and payload concepts in Kubernetes, the classic scaling methods – horizontal and vertical – are revealed in each component separately. Although the Kubernetes system is quite flexible, it has only recently been developed, and the features of scaling with concept separation have not yet been described.

Actual scientific researches and issues analysis showed that there is a general trend towards cloud migration recently. In this context, the architectural style of microservices and the use of container load has attracted considerable attention, especially based on the Kubernetes system. It can be concluded that Kubernetes has significant advantages in the speed of development and deployment of container-based applications, but the problem of scaling and features of horizontal and vertical scaling in Kubernetes has not received much attention from academia so far.

The aim of the work is to study the features of scaling the container load based on the Kubernetes system. To explain the basic operations to achieve horizontal and vertical scaling. To describe the scalability property of the Kubernetes system.

This work reveals the main architectural features of Kubernetes, the principles of working with applications in container virtualization. The main methods of regulating the resource capacity of containers are shown. The concepts of horizontal and vertical scaling in the Kubernetes system with an indication of scalability properties are revealed. The main operations for scaling in the Kubernetes system in the context of separation of payload and infrastructure concepts are given.

In this article the features of container load scaling based on the Kubernetes system with a separation of the concepts of payload and infrastructure scaling were investigated and described. This article is a review.

Keywords: *Kubernetes, microservice, application containers, scaling, docker.*

Fig.: 4. References.: 6.

Сопов Олексій Олександрович — студент-магістрант, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського" (пр-т Перемоги, 37, м. Київ, 03056, Україна).

Sopov Oleksii — master student, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (37 Peremohy Avenue, 03056 Kyiv, Ukraine).

E-mail: sopov.alax.a@gmail.com

ORCID: https://orcid.org/0000-0003-0389-3070

Цитовцева Анна Сергіївна — студент-магістрант, Національний технічний університет України "Київський політехнічний інститут імені Ігоря Сікорського" (пр-т Перемоги, 37, м. Київ, 03056, Україна).

Tsytovtseva Anna — master student, National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute" (37 Peremohy Avenue, 03056 Kyiv, Ukraine).