|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ** | | | | | | | |
| **НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЗАХИСТУ ІНФОРМАЦІЇ** | | | | | | | |
|  |  |  | | |  | |  |
| **КАФЕДРА ІНФОРМАЦІЙНОЇ ТА КІБЕРНЕТИЧНОЇ БЕЗПЕКИ** | | | | | | | |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
| Пояснювальна записка | | | | | | | |
| до бакалаврської роботи | | | | | | | |
|  | на тему: | | | | | |  |
| **«ДОСЛІДЖЕННЯ ШЛЯХІВ ТА РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВИКОРИСТАННЯ SERVICE MESH У ПІДТРИМЦІ МАСШТАБОВАНИХ І СТІЙКИХ РОЗГОРТАНЬ KUBERNETES»** | | | | | | | |
|  | | | | | | | |
|  | | | | | | | |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  | Виконав студент 4 курсу, групи БСД-42 | | | | | |
|  |  | спеціальності 125 Кібербезпека | | | | | |
|  |  | освітньо-професійної програми «Інформаційна та кібернетична безпека» | | | | | |
|  |  | (шифр і назва спеціальності) | | | | | |
|  |  | Баранов А.А. | | | | | |
|  |  | (прізвище та ініціали) | | | | | |
|  |  | Керівник | Довженко Н.М. | | | | |
|  |  |  | | | (прізвище та ініціали) | | |
|  |  | Рецензент | |  | | | |
|  |  | (прізвище та ініціали) | | | | | |
|  |  | Нормоконтролер | | | | Чумак Н.С. | |
|  |  |  | | | (прізвище та ініціали) | | |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  |  |  | | |  | |  |
|  | КИЇВ – 2023 | | | | | |  |

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Інститут | ННІЗІ | | |  |  |  |
| Кафедра | Інформаційної та кібернетичної безпеки | | | | | |
| Ступінь вищої освіти | | | Бакалавр | |  |  |
| Спеціальність | | 125 Кібербезпека | | |  |  |
| Освітньо-професійна програма | | | | Інформаційна та кібернетична безпека | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  | ЗАТВЕРДЖУЮ | |
|  |  |  | Завідувач кафедри ІКБ  Гайдур Г.І | |
|  |  |  | “\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2023 року | |
|  |  |  |  |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **З А В Д А Н Н Я** | | | | | |  |
|  | **НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ** | | | | | |  |
|  |  | |  | |  | |  |
| Баранову Андрію Андрійовичу | | | | | | | |
| (прізвище, ім’я, по батькові) | | | | | | | |
| 1. Тема бакалаврської роботи: | | «Дослідження шляхів та розробка рекомендацій | | | | | | |
| щодо використання Service Mesh у підтримці масштабованих і стійких розгортань | | | | | | | |
| Kubernetes» | | | | | | | |
| керівник бакалаврської роботи | | | Довженко Надія Михайлівна, к.т.н., доц. | | | | |
|  |  | | (прізвище, ім’я, по батькові, науковий ступінь, вчене звання) | | | | |
| затверджені наказом закладу вищої освіти від «24 » лютого 2023 року № 26. | | | | | | | |
|  |  | |  | |  | |  |
| 2. Строк подання студентом бакалаврської роботи | | | | | | 29.05.2023р. | |
|  |  | |  | |  | |  |
| 3. Вихідні дані до бакалаврської роботи | | | |  | | |  |
| програмний комплекс для віртуалізації; | | | | |  | |  |
| інструментарій для встановлення Kubernetes,Service Mesh; | | | | | | |  |
| наукова та технічна література, експлуатаційна документація, нормативні | | | | | | | |
| документи, міжнародні стандарти. | | | | |  | |  |
|  |  | |  | |  | |  |
| 4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) | | | | | | | |
| 1. Аналіз поняття контейнеризація. | | | | | | | |
| 2.Дослідження найпоширеніших методів контейнеризації та видів сервісних сіток. | | | | | | | |
| 3. Розробка рекомендацій щодо використання Service Mesh для підтримки безпеки, моніторингу та управління мережею в середовищі Kubernetes. | | | | | | | |
|  | | | | | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| 5. Перелік графічного матеріалу | |  |  |
| 1. Тема бакалаврської роботи. | | | |
| 2. Об’єкт, предмет, мета та наукові завдання дослідження. | | | |
| 3. Результати аналізу важливості ролі контейнеризації. | | | |
| 4. Результати аналізу інформаційної панелі Istio Grafana. | | | |
| 5. Рекомендації щодо налаштувань безпеки за допомогою Istio Dashboard Grafana. | | | |
| 6. Висновки за результатами роботи. | | | |
|  | | | |
| 6. Дата видачі завдання | 25.02.2023 р. | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/ п | Назва етапів бакалаврської роботи | Строк виконання етапів  бакалаврської роботи | Примітка |
| 1. | Аналіз науково-технічної літератури | 25.02.2023 р. | виконано |
| 2. | Аналіз принципу роботи контейнеризації | 03.03.2023 р. | виконано |
| 3. | Дослідження можливостей використання Service Mesh для підтримки вимог до безпеки, моніторингу та управління мережею в середовищі Kubernetes | 04.04.2023 р. | виконано |
| 4. | Розробка рекомендацій щодо використання Service Mesh для підтримки безпеки, моніторингу та управління мережею в середовищі Kubernetes | 29.04.2023 р. | виконано |
| 5. | Оформлення результатів дослідження | 25.05.2023 р. | виконано |
| 6. | Підготовка презентації до захисту | 28.05.2023 р. | виконано |

Студент Баранов А.А.

( підпис ) (прізвище та ініціали)

Керівник бакалаврської роботи Довженко Н.М.

( підпис ) (прізвище та ініціали)

**РЕФЕРАТ**

Текстова частина бакалаврської роботи: 56 сторінка, 33 рисунка, 20 джерел.

*Об’єкт дослідження* – технологія мікросервісної архітектури Service Mesh.

*Предмет дослідження* – використання технології Service Mesh в підтримці масштабованих та стійких розгортань Kubernetes.

*Мета роботи* – розробка рекомендацій щодо оптимального використання мікросервісної архітектури Service Mesh.

*Методи дослідження* – вивчення технічної документації, проведення тестування різних інструментів та порівняння різних сервісів для аналізу їхніх функцій та ефективності.

Аналіз і порівняння різних реалізацій Service Mesh, зокрема Istio та Linkerd, дослідження їх можливостей та функціоналу. Також будуть проведені експерименти з використанням різних Service Mesh на платформі Kubernetes, дослідження їх впливу на швидкість, стійкість та масштабованість розгортань. Для аналізу будуть використані методи порівняльного аналізу, експериментального дослідження та аналізу експлуатаційної документації.

*Галузь використання* – кібербезпека.

КІБЕРБЕЗПЕКА, Мікросервіси, БЕЗПЕКА МЕРЕЖІ, Service Mesh, Istio,GRAFANA, Linkerd, DASHBOARD, KUBERNETES, Операційна система.

**ABSTRACT**

Bachelor’s thesis: 56 pages, 33 figures, 20 sources.

*Object of research –* Service Mesh microservice architecture technology.

*Subject of research -* use of Service Mesh technology in supporting scalable and stable deployments of Kubernetes.

*The aim of research -* develop recommendations for optimal use of the Service Mesh microservice architecture.

*Research methods* – study of technical documentation, testing of various tools and comparison of various services to analyze their functions and effectiveness.

Analysis and comparison of different implementations of Service Mesh, in particular Istio and Linkerd, study of their capabilities and functionality. Experiments will also be conducted using different Service Meshes on the Kubernetes platform, researching their impact on the speed, stability and scalability of deployments. The methods of comparative analysis, experimental research and analysis of operational documentation will be used for the analysis.

*Field of use –* cybersecurity.

CYBERSECURITY, MICROSERVICES, NETWORK SECURITY, SERVICE MESH, ISTIO, GRAFANA, LINKERD, DASHBOARD, KUBERNETES, OPERATING SYSTEM.

.

**ЗМІСТ**

**ВСТУП……………………………………………………………………………….....9**

1. **ОГЛЯД ТЕХНОЛОГІЇ КОНТЕРНІЗАЦІЇ ТА ПРИНЦИП РОБОТИ SERVICE MESH…………………………………………………………………...... 11**
   1. Поняття та принципи роботи контейнерів…………….…………….. 11
   2. Огляд популярних платформ контейнеризації, зокрема Kubernetes…14
   3. Поняття та принципи роботи Service Mesh……………………….…..17
   4. Огляд популярних реалізацій Service Mesh, зокрема Istio та Linkerd..21

**Висновки до першого розділу………………………………………………….… 25**

1. **ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ ВИКОРИСТАННЯ SERVICE MESH ДЛЯ ПІДТРИМКИ МАСШТАБОВАНИХ ТА СТІЙКИХ РОЗГОРТАНЬ KUBERNETES …………………………………………………….17**
   1. Аналіз особливостей Kubernetes, які підтримують Service Mesh ..…26
   2. Дослідження можливостей використання Service Mesh для підтримки вимог до безпеки, моніторингу та управління мережею в середовищі Kubernetes…………………………………………………………….....37

**Висновки до другого розділу………………………………………………….…..43**

1. **РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВИКОРИСТАННЯ SERVICE MESH У ПІДТРИМЦІ МАШТАБОВНИХ ТА СТІЙКИХ РОЗГОРТАНЬ KUBERNETES ……………………………………………………………………... 44**
   1. Визначення оптимальних параметрів розгортання Service Mesh в середовищі Kubernetes...........................................................................................44
   2. Розробка рекомендацій щодо використання Service Mesh для підтримки безпеки, моніторингу та управління мережею в середовищі Kubernetes……………………………………………………………………………..57

**Висновки до третього розділу………………………………………………….....61**

**ВИСНОВКИ………………………………………………………………………..…62**

**ПЕРЕЛІК ПОСИЛАНЬ…………………………………………..………………..63**

**ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)….………………………...65**

**ВСТУП**

*Актуальність дослідження*. Дослідження шляхів та розробка рекомендацій щодо використання Service Mesh у підтримці масштабованих і стійких розгортань Kubernetes є дуже актуальною темою для дослідження в сучасному світі ІТ.

Kubernetes став популярним інструментом для автоматизації розгортання, керування та масштабування контейнерів. Хоча Kubernetes забезпечує багато можливостей для розгортання і управління контейнерами, він не надає інструментів для керування мережею, яка забезпечує з'єднання між контейнерами. Це може призвести до проблем зі стійкістю і масштабованістю додатків.

У такому контексті, Service Mesh, який є інфраструктурою для керування мережею, може допомогти вирішити ці проблеми. Він забезпечує мережеві функції, такі як маршрутизація, балансування навантаження та безпека, для всіх контейнерів, що працюють в Kubernetes-середовищі. Це дозволяє підтримувати масштабованість і стійкість розгортання додатків.

Отже, дослідження може виявитися корисним для тих, хто працює з Kubernetes і хоче забезпечити високу стійкість та масштабованість своїх додатків. Розробка рекомендацій щодо використання Service Mesh у підтримці масштабованих і стійких розгортань Kubernetes може стати корисним керівникам проектів, розробникам програмного забезпечення та системним архітекторам.

*Об’єкт дослідження* – технологія мікросервісної архітектури Service Mesh.

*Предмет дослідження* – використання технології Service Mesh в підтримці масштабованих та стійких розгортань Kubernetes.

*Мета роботи –* розробка рекомендацій щодо оптимального використання мікросервісної архітектури Service Mesh.

*Завдання бакалаврської роботи:*

* Огляд літератури та аналіз існуючих підходів до керування мережею в Kubernetes;
* Аналіз Service Mesh, які можливості надає дана технологія для керування мережею в Kubernetes, як вона працює, і які переваги вона може забезпечити для масштабованих та стійких розгортань;
* Дослідження методики тестування, для того щоб перевірити ефективність Service Mesh у реальному середовищі;
* Розробити рекомендації щодо застосування технології Service Mesh.

*Методи дослідження* – вивчення технічної документації, проведення тестування різних інструментів та порівняння різних сервісів для аналізу їхніх функцій та ефективності.

*Практичне значення одержаних результатів*: рекомендації щодо використання Service Mesh для керування мережею в Kubernetes-середовищі. Це може допомогти забезпечити кращу масштабованість, стійкість та безпеку додатків Kubernetes у сфері забезпечення кібербезпеки у персональних та корпоративних інформаційних системах.

1. **ОГЛЯД ТЕХНОЛОГІЇ КОНТЕРНІЗАЦІЇ ТА ПРИНЦИП РОБОТИ SERVICE MESH**
   1. **Поняття та принципи роботи контейнерів**

Контейнери — це тип технології віртуалізації, який дозволяє розробникам створювати та розгортати програмні додатки в автономному та портативному середовищі. Контейнери забезпечують ізольоване середовище виконання для програм разом із усіма залежностями та бібліотеками, необхідними для їх роботи. Це дозволяє розробникам швидко та легко створювати та розгортати програми, не турбуючись про базову інфраструктуру чи проблеми сумісності.

Контейнери споживають ресурси (такі як обчислювальна потужність

центрального процесора та пам’ять) ефективніше порівняно з такими варіантами, як віртуальні машини. Він також забезпечує переваги дизайну як у тому, як він відокремлює програму, так і в тому, як абстрагує рівень програми від рівня інфраструктури.

На рівні проекту ця абстракція дозволяє командам і членам команд працювати над кожною частиною проекту без ризику втручання в інші частини проекту. Розробники додатків можуть зосередитися на розробці додатків, а інженери інфраструктури — на інфраструктурі. У цьому випадку розділення командних обов’язків також розділяє код. Іншими словами, код програми не може зламати код інфраструктури, і навпаки.

Контейнери роблять перехід між розробкою та виробництвом для команд

більш плавним. Наприклад, якщо вам потрібно запустити програми Node.js на вашому сервері, одним із варіантів є встановлення Node.js безпосередньо. Це просте рішення, якщо ви один розробник, який працює на одному сервері. Однак, коли ви починаєте працювати з кількома розробниками та розгортаєте в кількох середовищах, одна інсталяція Node.js може відрізнятися для різних членів команди, які використовують дещо різні середовища розробки.[4]

На рівні розробки портативність контейнерів дозволяє створювати

інфраструктурні проекти, які не повинні турбуватися про непередбачуване виконання коду програми. У цьому контексті контейнери дозволяють розробникам:

* Запустити програму однаково на кількох машинах. Це означає, що локальні контейнери розробки та виробництва можна надійно зробити передбачуваними та спільно використовувати розробниками.
* Запустити кілька контейнерів як підключений стек програм. Це означає, що можна розробити свою програму та розділити її на мікросервіси, а не створити монолітний дизайн.

Ці функції можуть зробити контейнери схожими на віртуальні машини, але

різниця полягає в їх базовому дизайні та подальшій ефективності. Сучасна контейнерна технологія спеціально розроблена, щоб уникнути високих вимог до ресурсів для віртуальних машин. Контейнери мають однакові принципи переносимості та відтворюваності, але розроблені на різних рівнях абстракції. Контейнери пропускають віртуалізацію апаратного забезпечення та ядра, найбільш ресурсомістких частин віртуальної машини, і натомість покладаються на базове обладнання та ядро ​​хост-машини.

Як наслідок, контейнери відносно легкі та потребують низьких ресурсів. З тих пір контейнерна технологія сприяла розвитку багатої екосистеми, яка підтримує контейнерно-орієнтований розвиток, включаючи:

* Побудову образу контейнера.
* Збереження зображення в реєстрах і сховищах.
* Забезпечення стійкості даних за допомогою томів даних.
* Інтегровані мережеві рішення.

Строго кажучи, ці інструменти та переваги є допоміжними функціями на межі контейнерних технологій. Однак ці потреби настільки повсюдні, що їх часто об’єднують у комплексне контейнерне рішення. Принципи проектування контейнерів зазначено на рис.1.1.

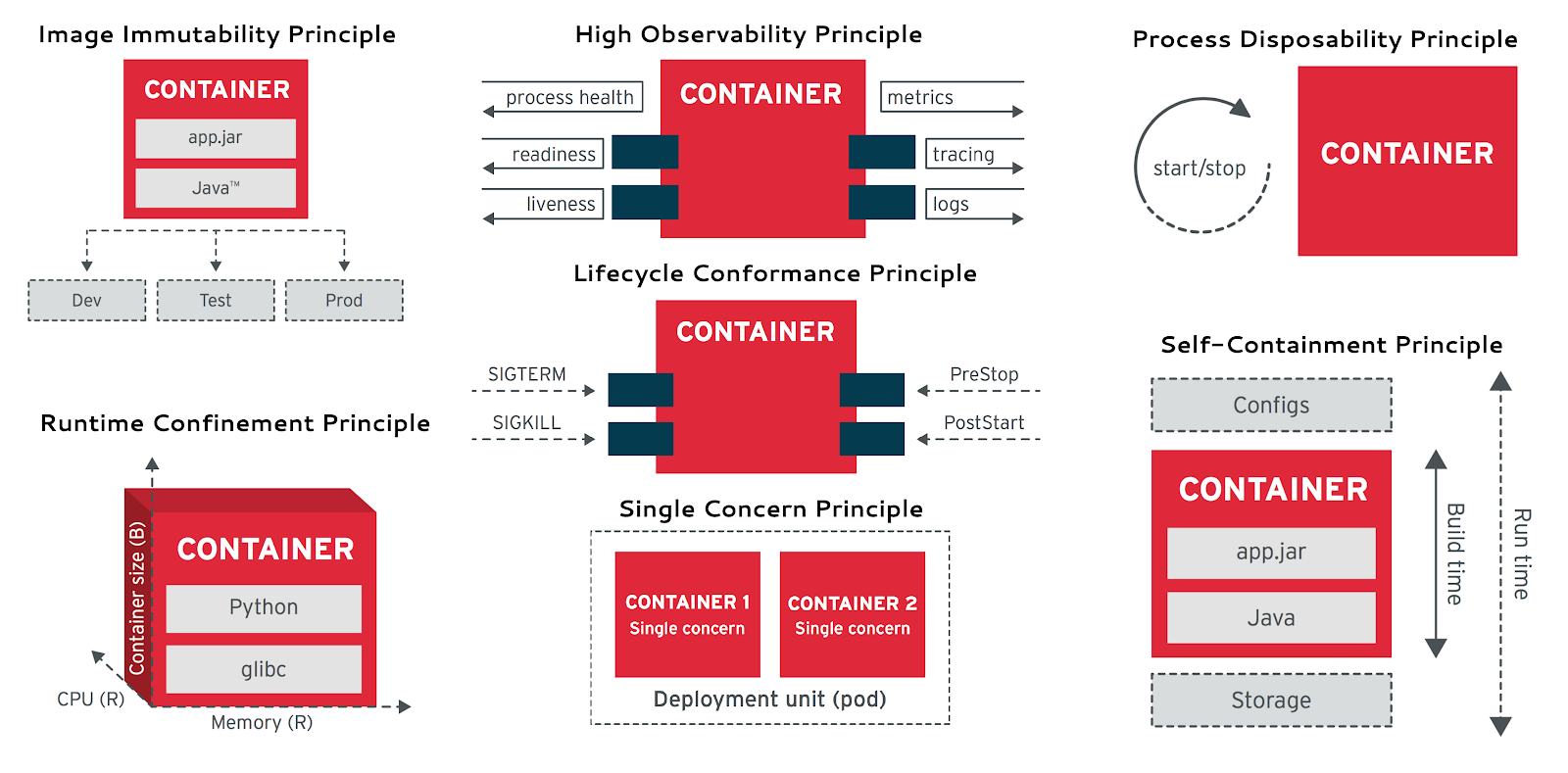


Рис.1.1. Принципи проектування контейнерів

Деякі з ключових понять і принципів роботи контейнера включають:

* Контейнерізація: Контейнерізація – це процес створення контейнера шляхом упаковки програми з усіма її залежностями та бібліотеками. Це забезпечує стабільну роботу програми в різних середовищах.
* Зображення: образ контейнера — це легкий, автономний виконуваний пакет, який містить усе необхідне для запуску частини програмного забезпечення, включаючи код, бібліотеки та системні інструменти. Зображення контейнерів створюються з Dockerfile, який є сценарієм, який визначає кроки, необхідні для створення образу.
* Середа виконання контейнера: середовище виконання контейнера — це програма, яка відповідає за створення контейнерів і керування ними. Найпопулярнішим середовищем виконання контейнерів є Docker.
* Оркестровка: Оркестровка контейнерів — це процес керування та масштабування контейнерів у кластері машин. Зазвичай це робиться за допомогою інструменту оркестровки контейнера, наприклад Kubernetes або Docker Swarm.
* Реєстр контейнерів: Реєстр контейнерів — це сховище для зберігання та спільного використання зображень контейнерів. До популярних реєстрів контейнерів належать Docker Hub і Google Container Registry.
* Мікросервіси: Контейнери часто використовуються в контексті мікросервісів, що є архітектурним стилем, який передбачає розбиття великих монолітних програм на менші, більш модульні сервіси. Кожен сервіс може бути упакований в окремий контейнер, що спрощує його розгортання, масштабування та керування.

Загалом контейнери — це потужна технологія, яка революціонізувала спосіб створення, розгортання та керування програмним забезпеченням. Забезпечуючи легке та портативне середовище виконання, контейнери дозволяють розробникам зосередитися на створенні чудового програмного забезпечення, не загрузаючи в деталях управління інфраструктурою.

* 1. **Огляд популярних платформ контейрнізації, зокрема Kubernetes**

Контейнерна платформа — це програмне рішення, яке дозволяє керувати контейнерними програмами. Він пропонує автоматизацію, оркестровку, безпеку, налаштування, підтримку контейнерних архітектур та багато іншого.

Є кілька видів контейнерних платформ:

* Контейнерні механізми, такі як Docker і Docker Enterprise Edition, забезпечують середовище виконання контейнерів, яке дозволяє створювати контейнери, керувати зображеннями контейнерів і виконувати основні операції.
* Оркестратор контейнерів, наприклад Kubernetes, який може керувати та автоматизувати контейнери в масштабі.
* Платформа керованого контейнера, наприклад Google Kubernetes Engine. Доповніть контейнери та механізми оркестровки додатковими службами, такими як керування оркестровкою та основними апаратними ресурсами.

Для того, щоб краще дослідити контейнерні платформи, варто звернути увагу на особливості OpenShift. Завдяки їх вивченню можна зрозуміти, як працюють контейнерні платформи, як вони допомагають забезпечувати більш ефективний розгортання і масштабування додатків, і як їх можна використовувати для власних проектів. Безперечно, ознайомлення з OpenShift дозволить збільшити рівень розуміння контейнерної технології в цілому, а також вивчити ряд корисних практик, які можна використовувати при роботі з іншими контейнерними платформами.[8]

OpenShift від Red Hat (рис.1.2.) — це контейнерна платформа, яка пропонує комплексне середовище для розробки, керування та розгортання контейнерних програм. Він побудований на технологіях з відкритим кодом, включаючи Docker і Kubernetes, які відповідають стандартам відкритого контейнера. OpenShift спрощує створення, розгортання та керування контейнерними програмами за допомогою цих стандартів.

Крім того, OpenShift надає численні функції, такі як автоматизація та масштабування розгортання програм, моніторинг, журналювання та інтеграція з іншими інструментами розробки, серед іншого. OpenShift можна встановити на приватній хмарній службі або на локальному сервері, дозволяючи вам керувати своїми даними в безпечному середовищі відповідно до ваших вимог.

Також, OpenShift полегшує розгортання та керування програмами в контейнерах, надаючи доступ до попередньо створених шаблонів програм. Ця функція спрощує процес розгортання та керування програмами в контейнерах, тим самим роблячи його менш громіздким.

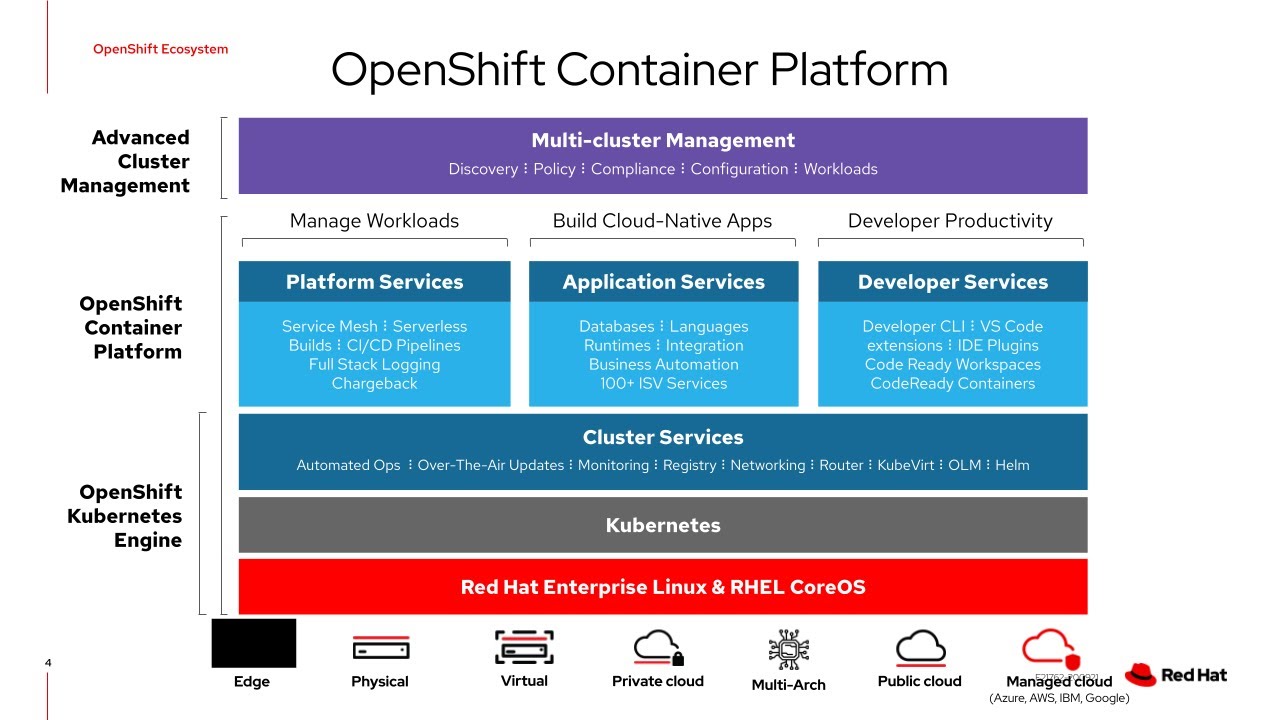


Рис.1.2. Контейнерна платформа OpenShift

Red Hat's OpenShift є однією з найбільш популярних контейнерних платформ, що дозволяє керувати контейнерами в різних хмарних середовищах.Red Hat's OpenShift має складну структурну складову, яка включає декілька компонентів. Основним елементом є Kubernetes, який використовується для керування контейнерами і є ядром платформи OpenShift.

Структурні компоненти контейнерної платформи Red Hat OpenShift такі:

* Головні вузли: ці вузли утворюють площину керування кластером OpenShift, відповідальним за керування розгортанням, масштабуванням і моніторингом контейнерних програм. Вони забезпечують сервер API, менеджер контролера та інші компоненти.
* Вузол: це робочий вузол, який запускає контейнерну програму разом з іншими службами, такими як Docker engine, Kubelet і Proxy.
* Реєстр: Реєстр відповідає за зберігання зображень контейнерів і керування ними. OpenShift за замовчуванням надає власний інтегрований реєстр, але його також можна налаштувати на використання зовнішніх реєстрів, таких як Docker Hub, Quay.io або Google Container Registry.
* CLI: інтерфейс командного рядка (CLI) — це інструмент, який розробники використовують для взаємодії з платформою OpenShift. CLI дозволяє користувачам створювати, розгортати, масштабувати та керувати контейнерними програмами.
* Source-to-Image (S2I): цей компонент дозволяє розробникам автоматично створювати зображення контейнерів із вихідного коду. S2I надає структуру для створення образів конструктора, які містять необхідні інструменти та бібліотеки для створення програм, а також образ середовища виконання, який містить необхідні залежності для запуску програми.
* Оператори: оператори — це спосіб упаковки, розгортання та керування програмами Kubernetes. OpenShift забезпечує структуру для створення та розгортання операторів, що спрощує керування складними програмами.
* Service Mesh: OpenShift Service Mesh — це платформа для керування додатками на основі мікросервісів. Він надає набір інструментів для керування трафіком між службами, моніторингу та налагодження служб, а також захисту служб за допомогою mTLS.

Разом ці компоненти утворюють ядро платформи Red Hat OpenShift, забезпечуючи надійне та гнучке середовище для створення, розгортання та керування контейнерними програмами.

**1.3 Поняття та принципи роботи Service Mesh**

Service Mesh — це концепція, яка застосовується до мікросервісів. Це спеціальний рівень інфраструктури, розгорнутий для оптимізації міжсервісного зв’язку та досягнення підвищеної видимості, надійності та безпеки. За останні роки сервісна мережа набула популярності та стала стандартним компонентом стеку хмарних технологій.

Мікросервіси — це архітектурний підхід до розробки програмного забезпечення, в якому основна увага приділяється розділенню моноліту на набір сервісів. Ця концепція стала популярною, оскільки все більше організацій почали досліджувати її потенціал, щоб допомогти їм подолати проблеми із застарілими системами та платформами

Залежно від розміру та складності програми можуть бути сотні чи тисячі мікросервісів. І щоб забезпечити бажані результати або покращити взаємодію з клієнтами, ці служби підключаються та спілкуються одна з одною.

Часто ці мікросервіси розгортаються всередині кластера Kubernetes. Проте доступність служби та безпечний зв’язок залишаються проблемою..

Почнемо з того, що коли кількість і складність мікросервісів зростає, їх виявлення стає проблемою. Оскільки мікросервіси можна розгорнути на будь-якому сервері, одній службі (скажімо, службі A) може знадобитися допомога у взаємодії з іншою службою (скажімо, службі B), коли є тисячі мікросервісів.

Крім того, коли в мікросервіси впроваджується небізнес-логіка, це призводить до плутанини між операторами та розробниками. Команда розробників також може почуватися втраченою, оскільки призначена послуга включає бізнес-логіку (BL) і небізнес-логіку. Крім того, зв’язок між службами в традиційній архітектурі мікросервісів, як правило, небезпечний.

Kubernetes намагається вирішити деякі з цих проблем, керуючи службами, що працюють як модулі в мережі кластерів Kubernetes. Проте проблеми залишаються та впливають на продуктивність програми, насамперед у зв’язку між послугами та безпеці.

Щоб вирішити ці проблеми, ІТ-команди використовують новий набір інструментів: сервісну мережу. Мета впровадження сітки сервісу дуже проста: не перевантажувати код сервісу додатковими деталями, пов’язаними з інфраструктурою/мережею, і дозволити йому піклуватися лише про бізнес-функції, які він має виконувати.[5]

Ось кілька ключових понять і принципів роботи сервісної сітки:

* Площина даних сервісної мережі: площина даних відповідає за обробку мережевого трафіку між службами. Він складається з набору побічних проксі-серверів, розгорнутих поряд із кожним екземпляром служби. Ці проксі-сервери перехоплюють весь вхідний і вихідний трафік і застосовують набір настроюваних правил, політик і поведінки.
* Площина керування сіткою обслуговування: площина керування відповідає за керування конфігурацією, розгортанням і моніторингом проксі-серверів площини даних. Він забезпечує централізоване уявлення про топологію сітки служби, стан працездатності, моделі трафіку та політики безпеки.
* Архітектура сітки служби: архітектура сітки служби зазвичай складається з набору взаємопов’язаних кластерів, кожен з яких містить кілька служб. Сітка служби контролює трафік між цими кластерами, запроваджуючи політики безпеки та обробляючи сценарії відновлення після відмови.
* Можливість спостереження за сіткою служби: сітка служби забезпечує видимість продуктивності та поведінки кожного екземпляра служби в реальному часі. Це включає такі показники, як затримка, частота помилок і обсяг трафіку, а також розподілену інформацію про відстеження, що дозволяє розробникам швидко виявляти та вирішувати проблеми.
* Безпека сітки служби: сітка служби забезпечує набір функцій безпеки, таких як шифрування, автентифікація та авторизація, щоб забезпечити конфіденційність, цілісність і доступність мережевого трафіку між службами.
* Масштабованість сітки служби: сітка служби може динамічно збільшувати або зменшувати кількість проксі-серверів залежно від обсягу трафіку та попиту на послуги. Це забезпечує оптимальне використання ресурсів і знижує ризик простою служби через обмеження ресурсів.
* Конфігурація сітки служби: сітка служби забезпечує гнучкий і централізований спосіб налаштування та керування поведінкою проксі-серверів площини даних. Сюди входять політики маршрутизації трафіку, балансування навантаження, розриву ланцюга та повторних спроб тощо.



Рис 1.3. Принцип роботи Service Mesh

Service Mesh надає потужний набір інструментів і абстракцій для підвищення надійності, безпеки та видимості архітектур мікросервісів, дозволяючи розробникам і операторам зосередитися на створенні цінності для бізнесу.

Використання сітки сервісу усуває складність використання сторонніх бібліотек або компонентів, які розробникам потрібно кодувати в кожній службі; натомість сервісна сітка дозволяє цим командам створювати проксі-сервіс для всі ці компоненти. Як наслідок, ці проксі-сервіси тепер становлять мережу сервісів і забезпечують більшу спостережуваність, безпеку та надійність.

**1.4 Огляд популярних реалізацій Service Mesh, зокрема Istio та Linkerd**

На сьогоднішній день доступно кілька популярних реалізацій Service Mesh, кожна з яких має власний набір функцій, можливостей і компромісів. Ось огляд двох найпопулярніших реалізацій Service Mesh:

* Istio — це мережева платформа служби з відкритим вихідним кодом, яка забезпечує керування трафіком, безпеку та функції спостереження для додатків мікросервісів. Спочатку він був розроблений Google, IBM і Lyft, а зараз є частиною Cloud Native Computing Foundation (CNCF).
* Istio побудовано на основі Envoy, високопродуктивного проксі-сервера, розробленого Lyft, і інтегрується з Kubernetes для оркестровки контейнерів. Він надає багатий набір функцій, таких як маршрутизація трафіку, ін’єкція помилок, розрив ланцюга та розподілене трасування.[11]

Ось деякі з ключових особливостей Istio:

* Traffic Management: Istio надає потужну структуру керування трафіком для додатків мікросервісів. Це дозволяє користувачам впроваджувати вдосконалені стратегії маршрутизації трафіку та балансування навантаження, встановлювати цілі рівня обслуговування (SLO) і застосовувати точні політики контролю трафіку, такі як повторні спроби, тайм-аути та обмеження швидкості.
* Security: Istio надає комплексну систему безпеки для додатків мікросервісів. Він шифрує весь мережевий трафік між службами, забезпечує взаємну автентифікацію TLS і реалізує політики авторизації на основі ідентифікації та ролей служби. Istio також надає потужну структуру для впровадження безпечного зв’язку в кількох кластерах і хмарах.
* Observability: Istio надає багатий набір функцій спостереження для додатків мікросервісів. Він надає метрики та журнали в реальному часі для моніторингу продуктивності та поведінки сервісу, розподілене трасування для виявлення та діагностики проблем у межах сервісу, а також графік сервісу для візуалізації топології сітки сервісу.
* Platform support: Istio розроблено для бездоганної роботи з Kubernetes, але також підтримує інші платформи оркестровки контейнерів, такі як Mesos і Docker Swarm. Istio також можна розгорнути на різноманітних хмарних платформах, включаючи Amazon Web Services (AWS), Google Cloud Platform (GCP).

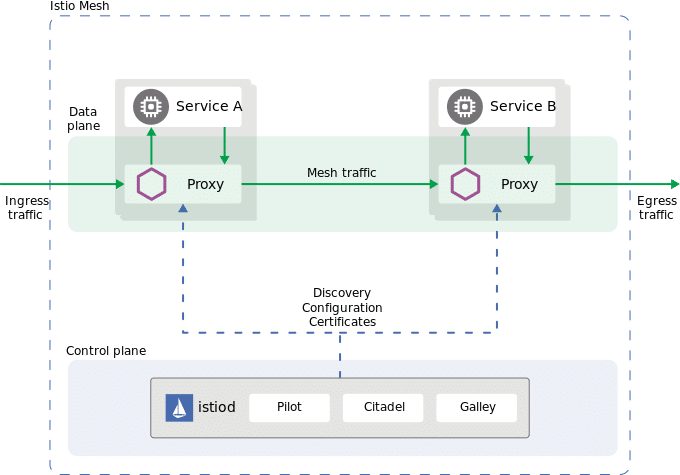


Рис 1.4. Принцип роботи Istio Service Mesh

Linkerd — це надлегка сервісна сітка для Kubernetes, яка зосереджена на простоті та зручності використання. Це проект із відкритим вихідним кодом, розроблений компанією Buoyant, яка розробляє програмне забезпечення в Сан-Франциско[1].

Linkerd надає простий та інтуїтивно зрозумілий інтерфейс користувача для налаштування маршрутизації трафіку, балансування навантаження та політики безпеки. Він використовує власний проксі Linkerd2-proxy, який створено на основі мови програмування Rust. Проксі розроблений як легкий і швидкий, з невеликим обсягом пам’яті та низькою затримкою.

Ось деякі з ключових функцій Linkerd:

* Traffic Management: Linkerd надає розширені можливості маршрутизації трафіку та балансування навантаження для програм Kubernetes. Це дозволяє користувачам налаштовувати розподіл трафіку, повторні спроби, тайм-аути та інші політики на рівні обслуговування. Linkerd також підтримує автоматичне балансування навантаження та перевірку працездатності, що полегшує керування великими та складними розгортаннями мікросервісів.
* Security: Linkerd надає потужні функції безпеки для програм Kubernetes. Він підтримує шифрування mTLS (взаємне TLS) між службами та може автоматично змінювати сертифікати TLS для підвищення безпеки. Linkerd також надає структуру для впровадження політик контролю доступу на основі посвідчень служби.
* Observability: Linkerd надає багатий набір функцій спостереження для програм Kubernetes. Він підтримує розподілене трасування та збір показників, що полегшує моніторинг і налагодження складних розгортань мікросервісів. Linkerd також надає зручну інформаційну панель, яка дозволяє користувачам візуалізувати залежності сервісів і показники продуктивності.
* Lightweight and easy to use: Linkerd розроблений, щоб бути легким і простим у використанні. Він має невеликий обсяг пам’яті та низьку навантаження на ЦП, що робить його придатним для середовищ з обмеженими ресурсами. Linkerd також легко встановити та налаштувати, завдяки простому та інтуїтивно зрозумілому інтерфейсу користувача.

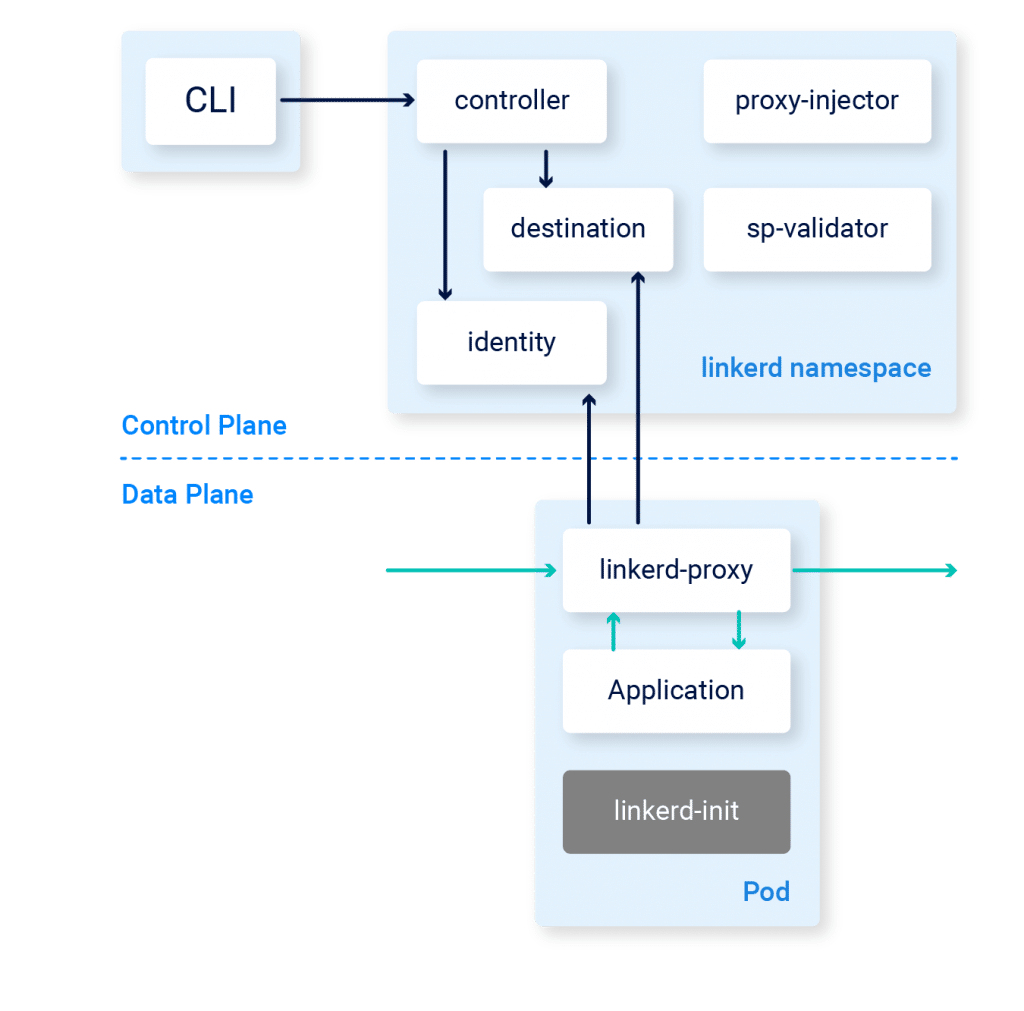


Рис 1.5. Принцип роботи Linkerd Service Mesh

Linkerd — це проста та легка сервісна сітка, яка надає розширені функції керування трафіком, безпеки та спостереження для програм Kubernetes. Це популярний вибір для розробників і команд DevOps, які хочуть спростити розгортання та керування додатками мікросервісів.

Ось порівняльна таблиця для двох популярних сервісних мережевих платформ - Istio і Linkerd:

*Таблиця 1.1. Порівняння Istio та Linkerd за основними характеристиками*

|  |  |  |
| --- | --- | --- |
| Особливості | ***Istio*** | ***Linkerd*** |
| Мова програмування | Go | Rust |
| Архітектура | Площина керування та площина даних | Площина керування та площина даних |
| Проксі служба | Envoy | Linkerd2-proxy |
| Traffic Management | Розширена маршрутизація трафіку, балансування навантаження та детальні політики керування трафіком | Розширена маршрутизація трафіку, балансування навантаження, повторні спроби та тайм-аути на рівні обслуговування |
| Security | Функції безпеки, шифрування mTLS і політики авторизації на основі ідентифікаційних і службових ролей | Функції безпеки, шифрування mTLS і політики контролю доступу на основі ідентифікаційних даних служби |
| Observability | Метрики та журнали в реальному часі, розподілене трасування та графік служби для візуалізації топології сітки служби | Розподілене відстеження, збір показників і зручна інформаційна панель для моніторингу залежностей і продуктивності служби |
| Platform support | Kubernetes, Mesos, Docker Swarm і численні хмарні платформи | Kubernetes and OpenShift |
| Community support | Велика та зростаюча спільнота з багатою екосистемою плагінів і розширень | Активна спільнота, зосереджена на простоті та зручності використання |

**Висновки до першого розділу**

Проаналізовано структурних компонентів контейнерних платформ,які дозволяють ефективно керувати та оркеструвати контейнерні додатки, а також проаналізавано структуру Service Mesh, її використання,для чого створена і т.д.

Досліджено поняття та принципи роботи контейнерів, їх структурні компоненти, принцип роботи Service Mesh, на прикладі Istio та Linkerd.Складено порівняльну таблицю та більш поглибленно розкрита робота кожного з цих сервісних сіток.

Проаналізовано популярні платформи контернізації на прикладі Red Hat`s OpenShift, його компонентів, принципи роботи двох сервісних сіток, а також порівняльну таблицю для розуміння різниці їх використання, для кого з корисутвачів чи корпортивних компаній підійде краще та чи інша сітка

1. **ДОСЛІДЖЕННЯ МОЖЛИВОСТЕЙ ВИКОРИСТАННЯ SERVICE MESH ДЛЯ ПІДТРИМКИ МАСШТАБОВАНИХ ТА СТІЙКИХ РОЗГОРТАНЬ KUBERNETES**

**2.1 Аналіз особливостей Kubernetes, які підтримують Service Mesh**

Kubernetes — популярна платформа оркестровки контейнерів з відкритим вихідним кодом, яка надає багатий набір функцій для розгортання додатків мікросервісів і керування ними. Однією з найбільш захоплюючих тенденцій у розробці сучасних додатків є використання сервісних сіток для забезпечення розширеного керування трафіком, безпеки та можливостей спостереження для додатків мікросервісів. Розглянемо, як Kubernetes надає кілька функцій, які підтримують розгортання та роботу сервісних сіток.

1.Контейнерізація

Контейнерізація — це метод упаковки програми та її залежностей в єдиний контейнер, який можна розгортати в різних середовищах. Контейнери ізольовані один від одного та від головної операційної системи, що робить їх портативними, легкими та простими в керуванні. Контейнери забезпечують узгоджене та передбачуване середовище виконання для програм, що полегшує розробку та розгортання програм у масштабі.

Контейнеризація революціонізувала спосіб розробки, розгортання та керування сучасними програмами. З розвитком архітектури мікросервісів контейнери стали важливим інструментом як для розробників, так і для операційних груп. Однією з найпопулярніших платформ для контейнеризації є Kubernetes, платформа оркестровки контейнерів з відкритим кодом, яка надає багатий набір функцій для розгортання та керування контейнерними програмами.[3]

Переваги контейнеризації

Контейнеризація пропонує кілька переваг порівняно з традиційною віртуалізацією та моделями розгортання на голому металі:

* Портативність: контейнери портативні та можуть розгортатися в різних середовищах, включаючи локальні центри обробки даних, публічні хмари та гібридні середовища. Це полегшує переміщення додатків між середовищами та зменшує ризик блокування постачальника.
* Масштабованість: контейнери мають невелику вагу, і їх можна швидко збільшити або зменшити залежно від попиту. Це полегшує роботу зі стрибками трафіку та зменшує потребу в надлишкових ресурсах.
* Узгодженість: контейнери забезпечують узгоджене та передбачуване середовище виконання для програм. Це спрощує керування та підтримку програм, зменшує ризик помилок і підвищує загальну надійність програм.
* Безпека: Контейнери забезпечують додатковий рівень безпеки, ізолюючи програми одна від одної та від головної операційної системи. Це ускладнює зловмисникам скомпрометувати систему та зменшує ризик витоку даних.

Kubernetes і контейнеризація

Kubernetes надає багатий набір функцій для розгортання та керування контейнерними програмами. Ось деякі з ключових функцій Kubernetes, які підтримують контейнеризацію:

* Підтримка середовища виконання контейнерів: Kubernetes підтримує кілька середовищ виконання контейнерів, зокрема Docker, container і CRI-O. Це означає, що користувачі можуть вибрати середовище виконання контейнера, яке найкраще відповідає їхнім потребам.
* Оркестровка контейнерів: Kubernetes забезпечує вбудовану підтримку оркестровки контейнерів, що означає, що користувачі можуть легко керувати та масштабувати контейнерні програми. Kubernetes може автоматично розгортати та керувати контейнерами в кластері вузлів, керувати мережевою маршрутизацією та балансуванням навантаження, а також керувати розподілом ресурсів.
* Керування ресурсами та масштабування: Kubernetes надає вбудовані можливості керування ресурсами та масштабування. Користувачі можуть вказувати запити на ресурси та обмеження для кожного контейнера, а Kubernetes автоматично керує розподілом ресурсів на основі попиту. Kubernetes також може автоматично масштабувати кількість реплік для служби на основі використання ресурсів та інших показників.
* Поточні оновлення та відкоти: Kubernetes дозволяє виконувати повільні оновлення та відкоти контейнерних програм. Це означає, що нові версії програми можна розгортати поступово, а Kubernetes автоматично керує переходом між старою та новою версіями. Це допомагає мінімізувати час простою та знизити ризик помилок.
* Виявлення служб і балансування навантаження: Kubernetes надає вбудовані можливості виявлення служб і балансування навантаження. Сервіси визначаються як логічна абстракція для набору пакетів, і Kubernetes автоматично розподіляє трафік між усіма пакетами, які належать до сервісу. Це спрощує масштабування додатків і впорається зі стрибками трафіку.

Підсумовуючи, контейнеризація революціонізувала спосіб розробки, розгортання та керування сучасними програмами. Kubernetes надає багатий набір функцій для розгортання та керування контейнерними програмами, включаючи підтримку кількох середовищ виконання контейнерів, оркестрацію контейнерів, керування ресурсами та масштабування, поточні оновлення та відкати, а також виявлення служб і балансування навантаження. Використовуючи Kubernetes як платформу для контейнеризації, розробники та робочі команди можуть скористатися багатим набором функцій і легко керувати та масштабувати контейнерні програми.

2. Виявлення служби та балансування навантаження

Виявлення служб і балансування навантаження є важливими компонентами сучасних архітектур додатків. Kubernetes надає вбудовану підтримку для виявлення служб і балансування навантаження, що полегшує керування та масштабування програм у розподіленому середовищі.Розглянемо концепції виявлення служб і балансування навантаження в Kubernetes, а також те, як їх можна використовувати для керування та масштабування контейнерних програм.

Виявлення служби в Kubernetes

Виявлення служб — це процес визначення розташування служб у розподіленій системі. У Kubernetes служби визначаються як логічна абстракція для набору модулів. Служби надають стабільну IP-адресу та ім’я DNS, які можна використовувати для доступу до модулів, які належать до служби.

Kubernetes пропонує кілька методів виявлення служб:

* DNS: Kubernetes надає вбудований DNS-сервер, який можна використовувати для визначення DNS-імен служб за їхніми IP-адресами. Це дозволяє додаткам легко знаходити та спілкуватися з іншими службами в кластері Kubernetes.
* Змінні середовища: Kubernetes автоматично встановлює змінні середовища для кожного контейнера, який містить IP-адресу та номер порту служб, від яких залежить контейнер. Це дозволяє програмам виявляти інші служби та спілкуватися з ними за допомогою змінних середовища.
* Бібліотеки виявлення служб: багато мов програмування та фреймворків надають бібліотеки, які можна використовувати для виявлення служб і взаємодії з ними в кластері Kubernetes. Ці бібліотеки зазвичай використовують Kubernetes API для отримання інформації про служби та надання абстракції для зв’язку з ними.

Балансування навантаження в Kubernetes

Балансування навантаження – це процес розподілу мережевого трафіку між набором серверів або контейнерів, щоб забезпечити рівномірний розподіл робочого навантаження та високу доступність служби. У Kubernetes балансування навантаження досягається за допомогою служби Kubernetes.

Коли служба створюється в Kubernetes, їй призначається віртуальна IP-адреса. Kubernetes автоматично направляє трафік до модулів, які належать службі, за допомогою балансувальника навантаження. Балансувальник навантаження розподіляє трафік на основі циклічного алгоритму, забезпечуючи рівномірний розподіл робочого навантаження між усіма модулями, які належать до служби.

Kubernetes пропонує кілька методів балансування навантаження:

* Циклічний баланс навантаження: Kubernetes використовує циклічний алгоритм для розподілу трафіку між усіма модулями, які належать до служби. Це гарантує рівномірний розподіл робочого навантаження та те, що кожен модуль отримує справедливу частку трафіку.
* Приналежність до сеансу: Kubernetes підтримує приналежність до сеансу, що гарантує, що запити від одного клієнта направляються до одного модуля. Це може бути корисно для програм, які вимагають сеансів із збереженням стану.
* Зовнішні балансувальники навантаження: Kubernetes може інтегруватися із зовнішніми балансувальниками навантаження, такими як AWS Elastic Load Balancing або Google Cloud Load Balancing, щоб забезпечити масштабоване та високодоступне балансування навантаження.

Виявлення служб і балансування навантаження є важливими компонентами сучасних архітектур додатків. Kubernetes надає вбудовану підтримку для виявлення служб і балансування навантаження, що полегшує керування та масштабування контейнерних програм. Завдяки Kubernetes розробники та робочі команди можуть скористатися перевагами вбудованих можливостей балансування навантаження, щоб забезпечити високу доступність їхніх послуг і рівномірний розподіл робочого навантаження. Використовуючи Kubernetes як платформу для виявлення служб і балансування навантаження, організації можуть створювати стійкі та масштабовані програми, які можуть легко справлятися зі сплесками трафіку та розвиватися відповідно до потреб бізнесу.

3. Поточні оновлення та відкат

Поточні оновлення та відкат є важливими функціями Kubernetes, які допомагають гарантувати, що ваші програми завжди запущені та працюють із мінімальним простоєм. Розглянемо концепції постійних оновлень і відкатів у Kubernetes і те, як їх можна використовувати для керування розгортанням контейнерних програм.

Поступові оновлення в Kubernetes

Поступове оновлення — це процес поступового оновлення набору модулів до нової версії вашої програми, гарантуючи, що ваша програма залишається доступною для користувачів під час процесу оновлення. У Kubernetes постійні оновлення виконуються шляхом створення нового набору реплік з оновленою версією вашої програми та поступового збільшення кількості модулів у новому наборі реплік, одночасно зменшуючи кількість модулів у старому наборі реплік.

Наступні кроки описують процес виконання поточного оновлення в Kubernetes:

1. Створіть нову версію програми та створіть новий набір реплік з оновленою версією.

2. Поступово збільшуйте кількість пакетів у новому наборі реплік, одночасно зменшуючи кількість пакетів у старому наборі реплік.

3. Слідкуйте за процесом оновлення та переконайтеся, що ваша програма залишається доступною під час оновлення.

Kubernetes надає кілька функцій, які допомагають з постійними оновленнями, зокрема:

* Зонди готовності та живучості: ці зонди використовуються для визначення того, чи готовий модуль приймати трафік і чи він ще живий. Використовуючи ці зонди, Kubernetes може автоматично визначати, чи готовий модуль до додавання чи видалення з набору реплік під час поточного оновлення.
* Стратегії оновлення: Kubernetes пропонує кілька стратегій оновлення, як-от поточні оновлення та синьо-зелені розгортання, які можна використовувати для керування процесом оновлення.

Відкати в Kubernetes

Відкат — це процес повернення розгортання до попередньої версії у разі будь-яких проблем або помилок, які могли бути представлені в новій версії програми. У Kubernetes відкат виконується шляхом створення нового набору реплік із попередньою версією програми та поступового збільшення кількості модулів у новому наборі реплік, одночасно зменшуючи кількість модулів у старому наборі реплік.

Наступні кроки описують процес виконання відкоту в Kubernetes:

* Визначте версію програми, яку потрібно відкотити.
* Створіть новий набір реплік із попередньою версією програми.
* Поступово збільшуйте кількість модулів у новому наборі реплік, одночасно зменшуючи кількість пакетів у поточному наборі реплік.

Kubernetes надає кілька функцій, які допомагають виконувати відкат, зокрема:

* Відстеження історії: Kubernetes відстежує історію всіх розгортань і наборів реплік, що полегшує визначення версії вашої програми, яку потрібно відкотити.
* Стратегії відкоту: Kubernetes надає кілька стратегій відкоту, як-от автоматичні та ручні відкоти, які можна використовувати для керування процесом відкоту.

Поточні оновлення та відкат є важливими функціями Kubernetes, які допомагають гарантувати, що ваші програми завжди запущені та працюють із мінімальним простоєм. За допомогою постійних оновлень і відкатів ви можете безпечно та легко керувати розгортанням ваших контейнерних програм, гарантуючи, що ваші користувачі завжди матимуть доступ до останньої версії вашої програми. Завдяки Kubernetes розробники та робочі команди можуть використовувати переваги вбудованих функцій для виконання поточних оновлень і відкатів, забезпечуючи плавний і ефективний процес розгортання своїх програм.

4. Керування ресурсами та масштабування

Керування ресурсами та масштабування є критично важливими аспектами керування контейнерними програмами в Kubernetes. Kubernetes надає низку інструментів і функцій, які дозволяють розробникам і робочим групам ефективно керувати використанням ресурсів і масштабувати свої програми за потреби. У цій статті ми розглянемо концепції управління ресурсами та масштабування в Kubernetes, а також функції, доступні для підтримки цих завдань.

Управління ресурсами в Kubernetes

Управління ресурсами передбачає розподіл і керування ресурсами, доступними для контейнерної програми. Це включає керування ЦП, пам’яттю, сховищем і мережевими ресурсами.

Kubernetes надає кілька функцій, які допомагають керувати використанням ресурсів, зокрема:

* Запити на ресурси та обмеження: Kubernetes дозволяє розробникам вказувати мінімальний і максимальний обсяг ресурсів, необхідних для контейнера, використовуючи запити на ресурси та обмеження. Вказуючи запити на ресурси та обмеження, Kubernetes може ефективно розподіляти ресурси між різними контейнерами в модулі, гарантуючи, що контейнери мають доступ до ресурсів, необхідних для належного функціонування.
* Horizontal Pod Autoscaler (HPA): HPA — це функція Kubernetes, яка автоматично масштабує кількість модулів у розгортанні на основі використання ЦП або інших спеціальних показників. Використовуючи HPA, Kubernetes може гарантувати, що додатки матимуть доступ до ресурсів, необхідних для належного функціонування, уникаючи при цьому браку ресурсів або втрат.
* Пріоритет і випередження модулів: Kubernetes надає функцію під назвою Priority and Preemption, яка дозволяє розробникам визначати важливість різних модулів у розгортанні. Після цього Kubernetes може автоматично визначати пріоритети та випереджати пакети залежно від їх важливості, гарантуючи, що критичні пакети мають доступ до ресурсів, необхідних для належного функціонування.

Масштабування в Kubernetes

Масштабування включає коригування ресурсів, виділених програмі, на основі її поточних потреб. У Kubernetes масштабування можна виконувати як вертикально, так і горизонтально.

* Вертикальне масштабування: передбачає збільшення або зменшення ресурсів, виділених для контейнера, наприклад ЦП або пам’яті. Kubernetes надає такі функції, як запити ресурсів контейнера та обмеження, які можна використовувати для налаштування ресурсів, виділених для контейнера.
* Горизонтальне масштабування: передбачає збільшення або зменшення кількості модулів, на яких запущено певне розгортання або набір реплік. Kubernetes надає такі функції, як HPA, який можна використовувати для автоматичного масштабування кількості модулів на основі використання ЦП або інших спеціальних показників.

Керування ресурсами та масштабування є критично важливими аспектами керування контейнерними програмами в Kubernetes. Ефективно керуючи використанням ресурсів і масштабуючи додатки відповідно до своїх потреб, розробники та операційні команди можуть забезпечити безперебійну роботу своїх додатків із ресурсами, необхідними для належного функціонування. За допомогою Kubernetes розробники та робочі команди можуть скористатися перевагами вбудованих функцій для ефективного керування ресурсами та масштабування своїх додатків, забезпечуючи плавний та ефективний процес розгортання своїх контейнерних додатків.

5. Безпека

Kubernetes — це потужна платформа оркестровки контейнерів, яка дозволяє розробникам розгортати, масштабувати та керувати контейнерними програмами. Як і з будь-якою іншою технологією, безпека є критичною проблемою, коли йдеться про Kubernetes. Розглянемо функції безпеки Kubernetes і те, як вони можуть допомогти захистити ваші контейнерні програми.

Контроль доступу на основі ролей (RBAC)

Однією з основних функцій безпеки Kubernetes є контроль доступу на основі ролей (RBAC), що зазначено на рисунку 2.1. RBAC дозволяє адміністраторам визначати ролі та дозволи користувачів і керувати ними, забезпечуючи детальний контроль над тим, хто може отримувати доступ до ресурсів у кластері Kubernetes і змінювати їх. RBAC дозволяє адміністраторам обмежувати доступ до критично важливих ресурсів і даних, захищаючи від несанкціонованого доступу та витоку даних.

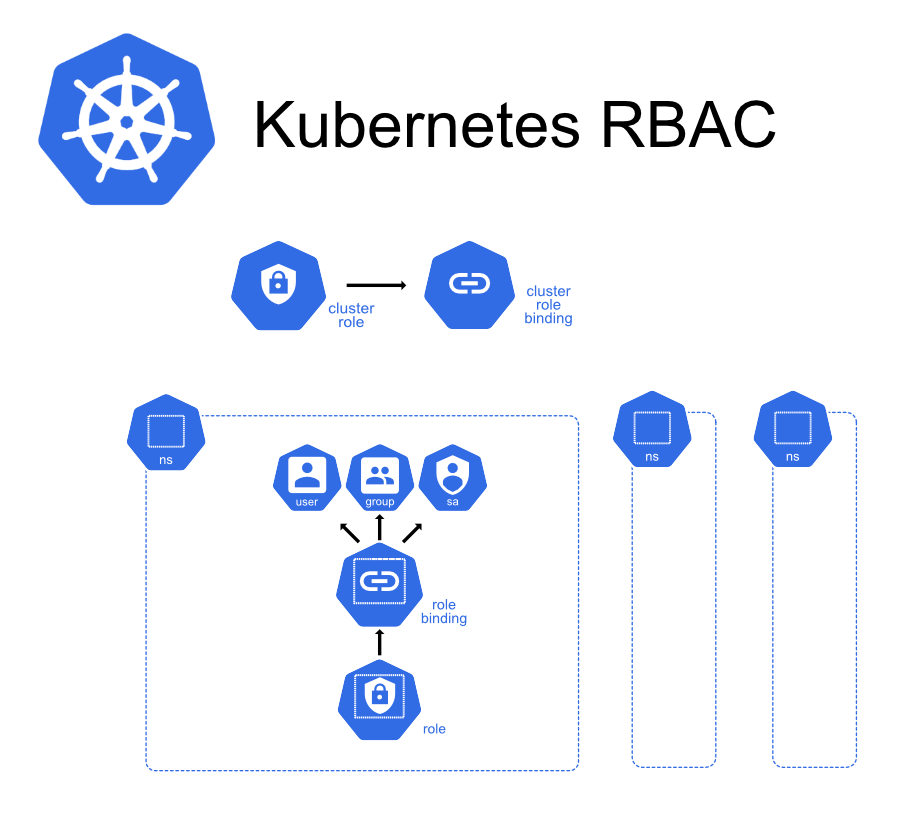


Рис 2.1. Схема налаштувань безпеки Kubernetes RBAC

Мережні політики

Kubernetes також надає мережеві політики, які дозволяють адміністраторам контролювати потік мережевого трафіку між модулями в кластері. Мережеві політики дозволяють адміністраторам створювати правила, які визначають, які модулі можуть спілкуватися один з одним і як. Це забезпечує додатковий рівень безпеки, захищаючи від несанкціонованого доступу та зловмисних атак.

Управління секретами

Kubernetes забезпечує безпечний спосіб керування секретами, такими як ключі API, паролі та сертифікати, через керування секретами. Секрети зберігаються як об’єкти Kubernetes і доступ до них мають лише авторизовані модулі та користувачі. Це гарантує безпеку конфіденційних даних і захист від несанкціонованого доступу.

Безпека зображення контейнера

Kubernetes підтримує різні реєстри зображень контейнерів, зокрема Docker Hub і Google Container Registry. Ці реєстри можна використовувати для зберігання та розповсюдження зображень контейнерів. Kubernetes надає функцію під назвою ImagePolicyWebhook, за допомогою якої можна гарантувати, що в кластері розгортаються лише довірені образи контейнерів. ImagePolicyWebhook дозволяє адміністраторам визначати політики перевірки зображень, забезпечуючи використання лише зображень із надійних джерел.

Безпека виконання

Kubernetes надає низку функцій безпеки під час виконання, зокрема AppArmor, SELinux і seccomp. Ці функції дозволяють адміністраторам обмежувати дії, які може виконувати контейнер, захищаючи від зловмисних атак і вразливостей.

Kubernetes надає надійний набір функцій безпеки, які можуть допомогти захистити ваші контейнерні програми. Використовуючи RBAC, мережеві політики, керування секретами та інші функції, розробники та робочі групи можуть гарантувати, що їхні кластери Kubernetes безпечні та захищені від несанкціонованого доступу та зловмисних атак. Як і з будь-якою іншою технологією, важливо бути в курсі найновіших найкращих практик безпеки та регулярно переглядати та оновлювати свої політики безпеки та конфігурації Kubernetes.

Kubernetes надає надійний набір функцій безпеки, які можуть допомогти захистити ваші контейнерні програми. Використовуючи RBAC, мережеві політики, керування секретами та інші функції, розробники та робочі групи можуть гарантувати, що їхні кластери Kubernetes безпечні та захищені від несанкціонованого доступу та зловмисних атак. Як і з будь-якою іншою технологією, важливо бути в курсі найновіших найкращих практик безпеки та регулярно переглядати та оновлювати свої політики безпеки та конфігурації Kubernetes.

**2.2 Дослідження можливостей використання Service Mesh для підтримки вимог до безпеки, моніторингу та управління мережею в середовищі Kubernetes**

Середовище Kubernetes стало популярним для розгортання мікросервісних додатків, оскільки воно забезпечує автоматичну оркестрацію, масштабування та керування контейнерами. Однак, з поширенням мікросервісної архітектури, з’явилася потреба в нових інструментах для керування мережею, моніторингу та безпеки, що призвело до появи технології Service Mesh.

Service Mesh - це мережевий шар, розташований між додатком та мережею. Він забезпечує функції моніторингу, балансування навантаження, безпеки та керування трафіком між мікросервісами. Це дозволяє зменшити складність управління мережею та підвищити безпеку та стійкість додатку.[6]

Одним зі засобів Service Mesh є Istio, що забезпечує широкий набір можливостей для керування мережею, моніторингу та безпеки. Istio забезпечує функції моніторингу та логування, які дозволяють виявляти та вирішувати проблеми у реальному часі. Крім того, Istio забезпечує можливість налаштування маршрутизації трафіку, балансування навантаження та обмеження швидкості передачі даних, що дозволяє підвищити стійкість додатку та зменшити його залежність від окремих сервісів.

Однією з ключових функцій Istio є забезпечення безпеки між мікросервісами. Istio дозволяє налаштовувати правила безпеки для кожного сервісу, використовуючи механізми аутентифікації, авторизації та шифрування. Це дозволяє запобігнути багато з потенційних проблем безпеки, які можуть виникнути в середовищі мікросервісів. За допомогою Istio можна налаштувати правила доступу до ресурсів, обмежити доступ до конфіденційної інформації та уникнути атак типу "Man in the Middle" або "Cross-site scripting".

Іншою важливою функцією Istio є моніторинг мережі. Istio забезпечує збір та аналіз логів, що дозволяє відслідковувати проблеми мережі та шукати причини їх виникнення. Istio також забезпечує моніторинг трафіку мережі та стану сервісів, що дозволяє оперативно виявляти проблеми та забезпечувати максимальну доступність сервісів.

Останньою, але не менш важливою функцією Istio є управління мережею. Istio забезпечує автоматичну маршрутизацію між сервісами, що дозволяє покращити швидкість та доступність сервісів. Крім того, Istio забезпечує можливість балансування навантаження та автоматичного масштабування, що дозволяє збільшувати потужність мережі за необхідності та зменшувати її, коли потрібне менше ресурсів.

Istio забезпечує безпеку мережі, моніторинг її стану та автоматичне управління, що є дуже важливим для забезпечення максимальної доступності та ефективності сервісів.[10]

Насамперед, Istio забезпечує безпеку мережі за допомогою механізмів аутентифікації, авторизації та шифрування. Це дозволяє забезпечити конфіденційність, цілісність та доступність даних, що передаються між сервісами. Istio також підтримує різні методи аутентифікації, включаючи мутуальну аутентифікацію TLS, аутентифікацію на основі токенів та інтеграцію з існуючими системами аутентифікації.

Крім того, Istio забезпечує моніторинг мережі за допомогою збору метрик та логів з кожного сервісу. Це дозволяє операторам мережі отримувати повну картину про стан мережі та сервісів, що допомагає вчасно виявляти та вирішувати проблеми.

Нарешті, Istio забезпечує автоматичне управління мережею за допомогою механізмів трафіку-менеджменту та балансування навантаження. Це дозволяє автоматично розподіляти трафік між сервісами, підтримувати резервні копії та автоматично масштабувати сервіси в залежності від навантаження. Все це допомагає забезпечити максимальну доступність та ефективність сервісів.

Istio надає кілька механізмів аутентифікації та авторизації для забезпечення безпеки мережі:

1. Mutual TLS (mTLS) - Istio може налаштувати взаємну аутентифікацію сервісів з використанням TLS-шифрування. Кожен сервіс буде мати свій SSL-сертифікат та приватний ключ, які використовуються для шифрування трафіку між сервісами.

Mutual TLS (mTLS) - це механізм аутентифікації та шифрування, який дозволяє забезпечити безпеку між мікросервісами у мережі Istio.mTLS гарантує, що всі з'єднання між мікросервісами будуть зашифровані та перевірені на аутентичність з обох боків, тобто як відправник, так і отримувач будуть перевірені на те, що вони є частинами довіреної мережі.

Для використання mTLS у Istio, кожному мікросервісу присвоюється сертифікат TLS, що дозволяє проводити аутентифікацію та шифрування. Крім того, Istio автоматично генерує та управляє кореневим сертифікатом для кожної мережі, що дозволяє гарантувати безпеку всієї мережі.

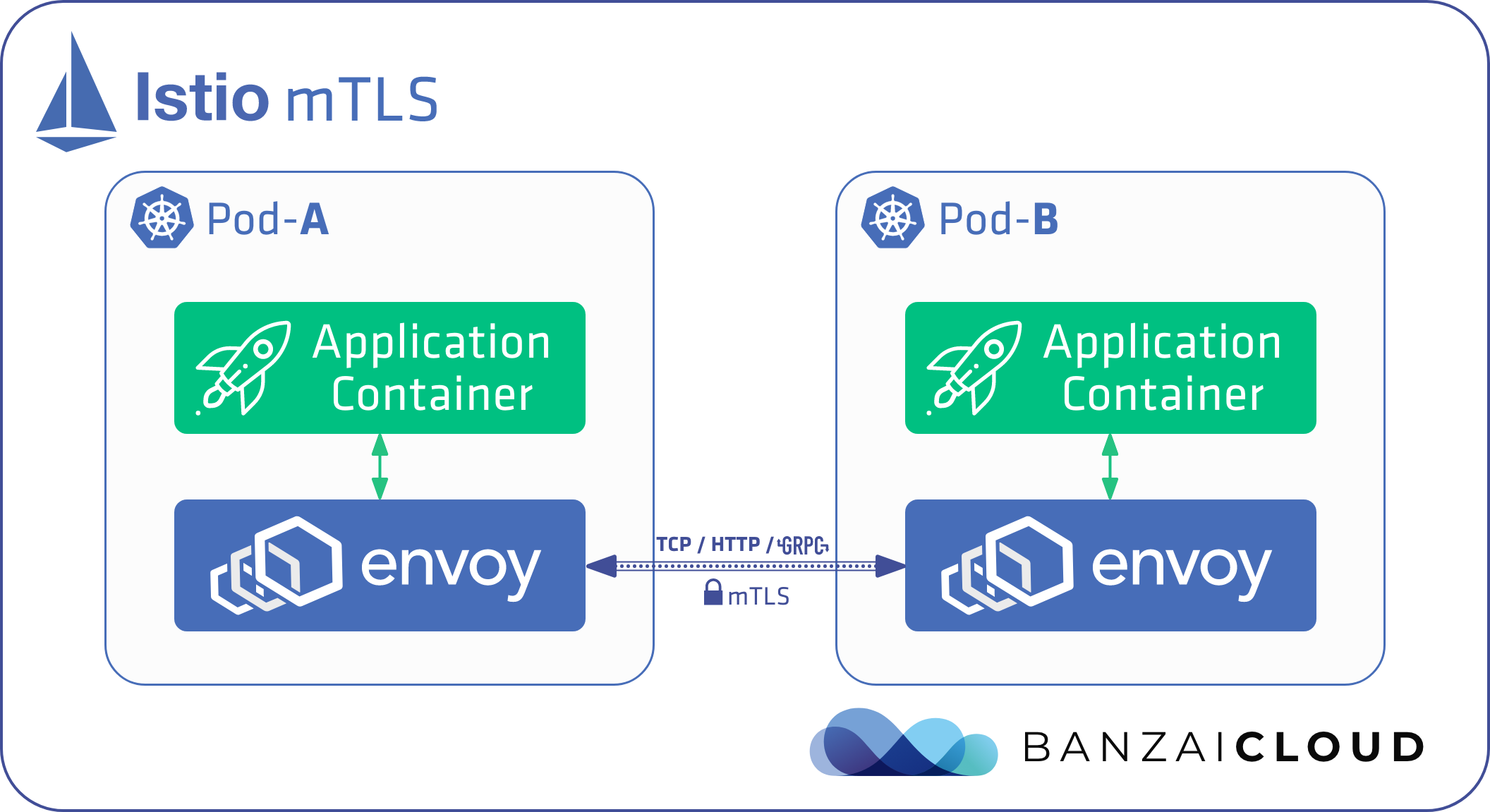


Рис. 2.2. Мутуальна аутентифікація TLS

1. JSON Web Token (JWT) - Istio підтримує JWT-аутентифікацію, яка дозволяє сервісам перевіряти токени, які відправляються в заголовках запитів. Це може бути корисно для перевірки джерела запиту та забезпечення авторизації на основі ролей. JSON Web Token (JWT) - це стандарт для безпечної передачі між сторонами інформації в форматі JSON.

В контексті Istio, JWT використовується для аутентифікації користувачів та авторизації доступу до ресурсів.Ідея полягає у тому, що користувач аутентифікується під час входу до системи та отримує JWT-токен, який містить інформацію про його ідентифікатор та права доступу.

Далі, при спробі доступу до захищеного ресурсу, користувач передає JWT-токен у заголовку запиту, і Istio перевіряє його валідність та дозволяє або забороняє доступ до ресурсу в залежності від прав доступу, зазначених в токені.Для реалізації JWT-аутентифікації в Istio потрібно налаштувати політики безпеки (Security Policies) та користуватись інструментами, які дозволяють генерувати та перевіряти JWT-токени, наприклад, OpenID Connect або JSON Web Token (JWT) плагін для Envoy.

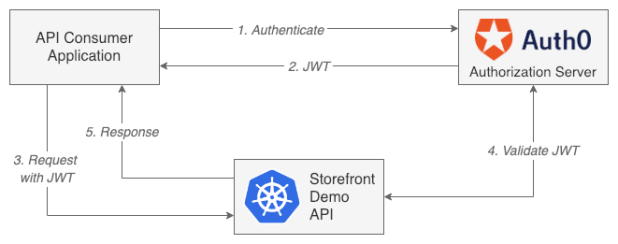


Рис. 2.3. Принцип аутентифікації користувача через JWT-токен

1. Access Control Lists (ACLs) - це механізм контролю доступу в Istio, що дозволяє налаштувати правила доступу до сервісів і ресурсів в мережі. За допомогою ACL можна визначити, які користувачі або групи користувачів мають доступ до конкретних ресурсів та які дії вони можуть виконувати з ними.

Access Control Lists (ACL) - Istio дозволяє налаштовувати списки контролю доступу (ACL) для обмеження доступу до конкретних ресурсів або сервісів на основі IP-адрес, підсітей або ідентифікаторів користувачів.

ACL використовується в Istio для реалізації політик авторизації на основі різноманітних атрибутів, таких як ім'я користувача, IP-адреса, роль користувача та інші. Для визначення прав доступу можна використовувати різні типи правил, такі як allow / deny або заборонити доступ до певних ресурсів.

Наприклад, ACL може бути налаштований для того, щоб дозволити лише користувачам з певної групи або з певними привілеями доступ до певного сервісу в мережі Istio. Крім того, ACL може бути використаний для забезпечення доступу до конфіденційної інформації тільки певним користувачам або групам користувачів.

Всі правила ACL використовуються для генерації політики авторизації в Istio, що дозволяє забезпечувати безпеку в мережі за допомогою точної настройки доступу до сервісів та ресурсів.

1. Role-Based Access Control (RBAC) - це механізм для регулювання доступу до ресурсів на основі ролей, призначених окремим користувачам або групам. Istio використовує RBAC для забезпечення детального контролю над доступом до мікросервісів у сітчастому середовищі сервісу.

В Istio політики RBAC можна визначити на рівні сервісу або на рівні робочого навантаження. Політики рівня обслуговування застосовуються до всіх робочих навантажень, пов’язаних із певною службою, тоді як політики рівня робочого навантаження застосовуються лише до окремих робочих навантажень.

Політики RBAC в Istio визначаються за допомогою набору спеціальних ресурсів, включаючи ресурси AuthorizationPolicy і ServiceRole. Ці ресурси дозволяють адміністраторам визначати правила контролю доступу на основі ряду критеріїв, включаючи ідентифікаційну інформацію користувача або групи, які надсилають запит, використовуваний метод (наприклад, GET, POST) і шлях запиту.

RBAC в Istio також можна використовувати в поєднанні з іншими механізмами автентифікації, такими як mTLS і JWT, щоб забезпечити багаторівневий підхід до безпеки. Наприклад, політики RBAC можна використовувати для обмеження доступу до певної служби для певного набору користувачів або груп, тоді як mTLS можна використовувати для забезпечення підключення до служби лише авторизованих клієнтів.

Загалом RBAC в Istio надає потужний і гнучкий механізм для контролю доступу до мікросервісів у сітчастому середовищі сервісу. Це дозволяє адміністраторам визначати детальні політики, які враховують широкий спектр факторів, і може використовуватися в поєднанні з іншими механізмами автентифікації та авторизації для забезпечення багаторівневого підходу до безпеки.

1. Web Application Firewall (WAF) - Istio може використовувати WAF для захисту від атак типу "Cross-site scripting" та інших вразливостей веб-додатків.

Istio також надає брандмауер веб-додатків (WAF) як частину своїх функцій безпеки. WAF може допомогти захистити веб-додатки від поширених атак, таких як впровадження SQL, міжсайтовий сценарій та інші веб-експлойти.

Istio WAF реалізовано за допомогою проксі-сервера Envoy, який перехоплює та перевіряє вхідний трафік HTTP, перш ніж він досягне програми. Він використовує правила та фільтри для виявлення та блокування потенційно зловмисних запитів.

WAF можна налаштувати на використання різних наборів правил і ланцюжків фільтрів залежно від конкретних потреб безпеки програми. Він також може генерувати докладні журнали та показники, щоб допомогти з налагодженням і моніторингом.

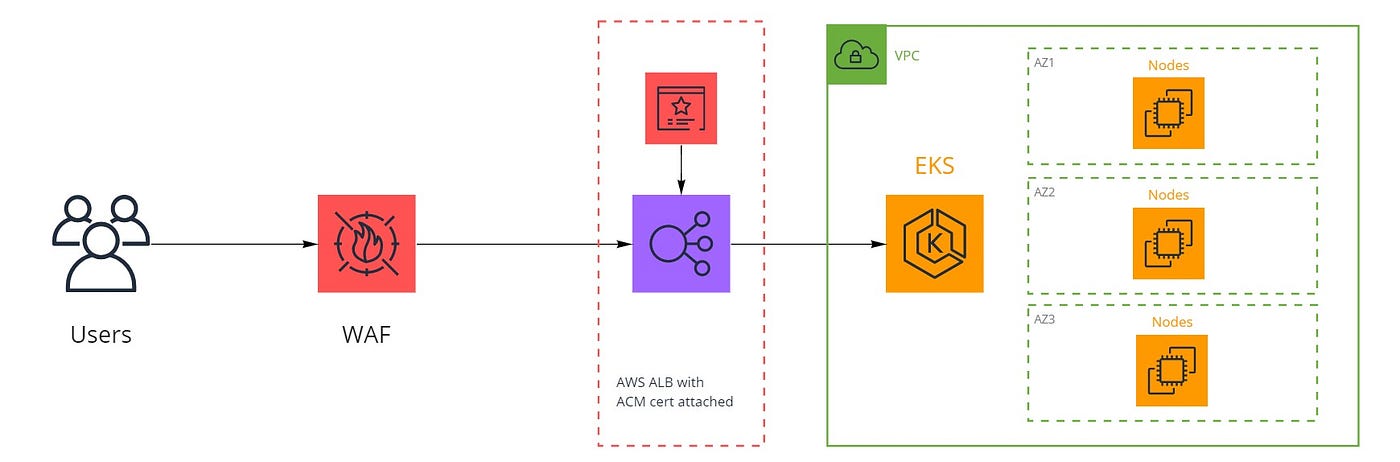


Рис. 2.4. Принцип роботи Web Application Firewall Istio Envoy

**Висновки до другого розділу**

Проаналізовано особливостей Kubernetes,які підтримують Service Mesh. Також проаналізовано функціонал Service Mesh для розгортання та керування контейнерними програмами

Досліджено поняття та принципи роботи Istio Service Mesh, функції безпеки Kubernetes і те, як вони можуть допомогти захистити ваші контейнерні програми.Складено детальний список функціоналу безпеки та більш поглибленно розкрита робота кожного з цих функцій.

Проаналізовано одну з найпопулярніших сіток Istio Service Mesh, її механізмів безпеки.Розглянуто кілька механізмів аутентифікації та авторизації для забезпечення безпеки мережі, які підходять як для корпоративної мережі так і для локальної,а також для персональної.

1. **РОЗРОБКА РЕКОМЕНДАЦІЙ ЩОДО ВИКОРИСТАННЯ SERVICE MESH У ПІДТРИМЦІ МАШТАБОВНИХ ТА СТІЙКИХ РОЗГОРТАНЬ KUBERNETES**
   1. **Визначення оптимальних параметрів розгортання Service Mesh в середовищі Kubernetes**

Оптимальні параметри розгортання Service Mesh в середовищі Kubernetes залежать від вимог до масштабованості, надійності, продуктивності, безпеки та інших параметрів.[12]

Основні параметри, які можна оптимізувати при розгортанні Service Mesh в середовищі Kubernetes:

1. Вибір платформи Service Mesh: Istio, Linkerd, Consul, Maesh та інші. При виборі необхідно враховувати функціональні можливості, продуктивність, масштабованість, наявність інструментів моніторингу та управління мережею.
2. Конфігурування мережевої топології: вибір типу мережі (безпроводова, проводова, гібридна), конфігурування вузлів, налаштування маршрутизації мережевого трафіку.
3. Налаштування безпеки: встановлення правил доступу до ресурсів, шифрування мережевого трафіку, аутентифікація та авторизація користувачів та сервісів.
4. Налаштування моніторингу та логування: налаштування механізмів моніторингу мережі та сервісів, збір та аналіз логів.
5. Налаштування масштабування та управління: налаштування механізмів автоматичного масштабування сервісів, управління версіями сервісів, балансування навантаження.
6. Вибір платформи кластера Kubernetes: вибір між публічними та приватними хмарами, віртуальними машинами та контейнерними сервісами.

Оптимальні параметри розгортання Service Mesh в середовищі Kubernetes можуть бути різними для кожного проекту, в залежності від вимог до масштабованості, надійності, продуктивності, безпеки та інших параметрів. Одним з основних параметрів, які можна налаштувати при розгортанні Service Mesh в Kubernetes, є:

1. Розмір мережі - мережа Service Mesh може бути масштабована в залежності від потреб проекту. Якщо мережа занадто мала, можуть виникати проблеми з масштабованістю і надійністю, а якщо занадто велика - можуть виникати проблеми з продуктивністю.
2. Кількість боксів (sidecar) - бокси є складовою частиною кожного мікросервісу в Service Mesh і відповідають за забезпечення мережевої безпеки, моніторингу та управління мережею. Від кількості боксів залежить продуктивність та надійність мережі.
3. Використання протоколу мережевого взаємодії - Istio підтримує різні протоколи мережевого взаємодії, такі як HTTP, gRPC, TCP, UDP тощо. Вибір протоколу залежить від потреб проекту.
4. Використання механізмів безпеки - Istio підтримує різні механізми безпеки, такі як аутентифікація, авторизація, шифрування трафіку тощо. Вибір механізмів безпеки залежить від потреб проекту.
5. Моніторинг - Istio має вбудовані механізми моніторингу мережі, такі як Prometheus, Grafana тощо. Вибір механізмів моніторингу залежить від потреб проекту.
6. Сумісність з іншими інструментами - Istio повинен бути сумісним з іншими інструментами, які використовуються в проекті, такими як CI/CD інструменти, Docker-контейнери тощо.

При виборі оптимальних параметрів розгортання Service Mesh в середовищі Kubernetes слід враховувати всі потреби проекту, включаючи:

* Масштабність: Потрібно визначити, які масштабність інфраструктури має підтримувати Service Mesh, і як це можна досягти. Також слід звернути увагу на кількість інстансів Istio Control Plane та їх розміщення.
* Продуктивність: Для досягнення високої продуктивності слід вибрати оптимальні налаштування Service Mesh, такі як використання мінімальної кількості проксі-серверів Envoy та відповідних налаштувань мережі.
* Надійність: Service Mesh має забезпечувати високу надійність сервісів, тому слід звернути увагу на налаштування аварійного відновлення та механізми резервування.
* Безпека: Service Mesh має забезпечувати безпеку між сервісами, тому слід враховувати параметри безпеки, такі як налаштування автентифікації, авторизації, шифрування трафіку та контроль доступу.
* Моніторинг: Service Mesh має забезпечувати моніторинг стану сервісів та мережі. Тому слід звернути увагу на відповідні налаштування моніторингу, такі як інтеграція з системами моніторингу та логування.
* Вартість: Слід враховувати вартість розгортання та підтримки Service Mesh, щоб забезпечити оптимальне співвідношення вартості та продуктивності проекту.
* Комплексність: Service Mesh має забезпечувати комплексне управління мережею та сервісами, тому слід визначити, наскільки глибоко можна налаштувати.

Для початку розгортання Istio Service Mesh та його налаштування, потрібно інсталювати кластер Kubernetes та встановити Docker.

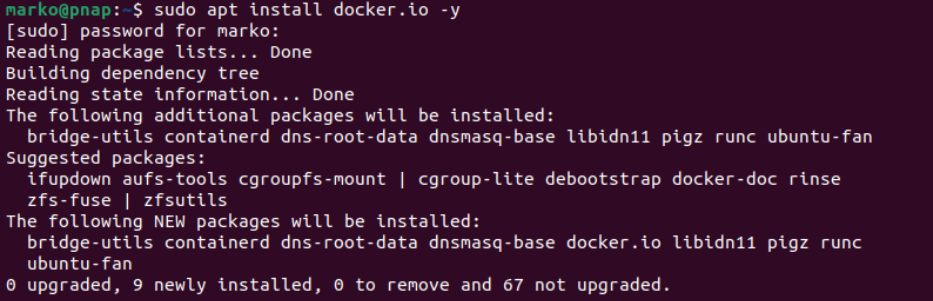


Рис. 3.1. Завантаження Docker

Далі потрібно подивитись чи працює Docker належним чином, то ж звернимось до команди sudo systemctl status docker.

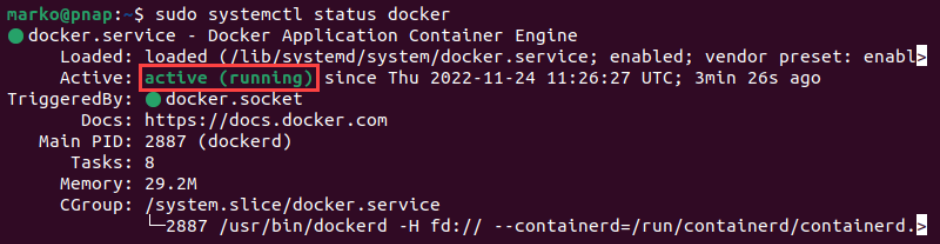
****

Рис. 3.2. Перевірка Docker

Як бачимо Docker працює належним чином, він активний, а отже, наступним кроком буде установка кластера Kubernetes.

Для того щоб встановити кластер потрібно скористатись інструментами для його встановлення,тож використаємо команду sudo apt install kubeadm kubelet kubectl.

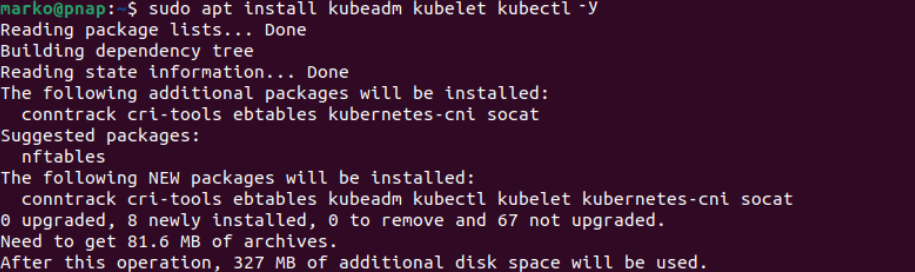


Рис. 3.3. Встановлення інструментів Kubernetes

Почекавши декілька хвилин на встановлення, особливо важливо перевірити чи встановились всі інструменти належним чином,тож для цього використаємо команду kubeadm version.



Рис. 3.4. Перевірка встановлення інструментів Kubernetes

Наступний крок для встановлення кластеру - підготовка серверів. Спочатку потрібно вимкнути пам’ять підкачки, робиться це для того щоб зменшити навантаження на систему, команда - sudo swapoff –a.

Далі, встановлення контейнерних модулів - sudo nano /etc/modules-load.d/containerd.conf.

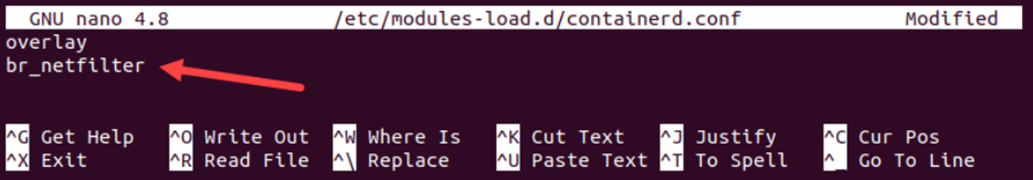


Рис. 3.5. Установка контейнерних модулів

Після установки контейнерних модулів,потрібно налаштувати мережу.Для цього відкриваємо файл конфігурації за допомогою команди - sudo nano /etc/sysctl.d/kubernetes.conf , і далі по порядку вводимо:

* net.bridge.bridge-nf-call-ip6tables = 1
* net.bridge.bridge-nf-call-iptables = 1
* net.ipv4.ip\_forward = 1

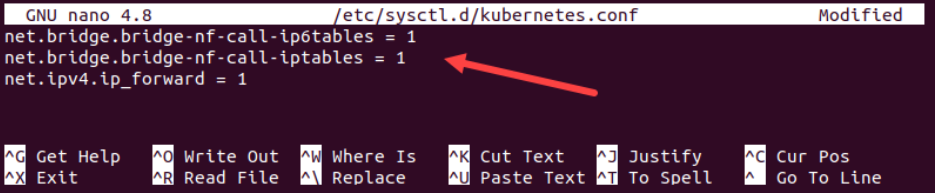


Рис. 3.6. Налаштування мережі Kubernetes

Налаштувавши мережу, потрібно дати ім’я хоста на кожен вузол сервера, це робиться за допомогою команд: sudo nano /etc/default/kubelet та KUBELET\_EXTRA\_ARGS="--cgroup-driver=cgroupfs". Подальші кроки, це ініціалізація самого Kubernetes. В терміналі ми додаємо блок конфігурації.

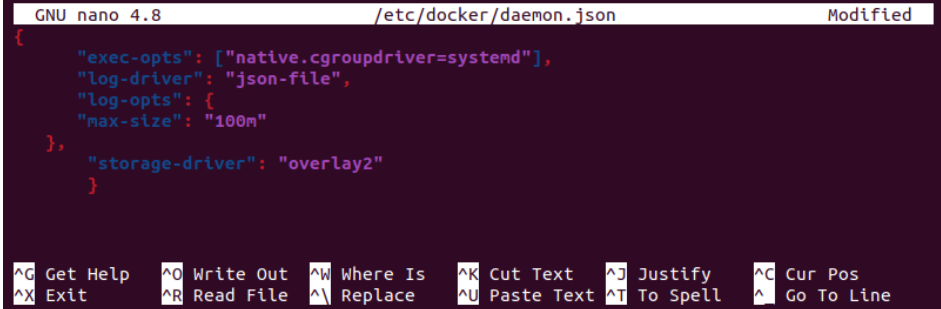


Рис. 3.7. Налаштування конфігурації через Docker

Зберігаємо,перезавантажуємо кластер і вводимо команду – Environment = "KUBELET\_EXTRA\_ARGS=--fail-swap-on=false".

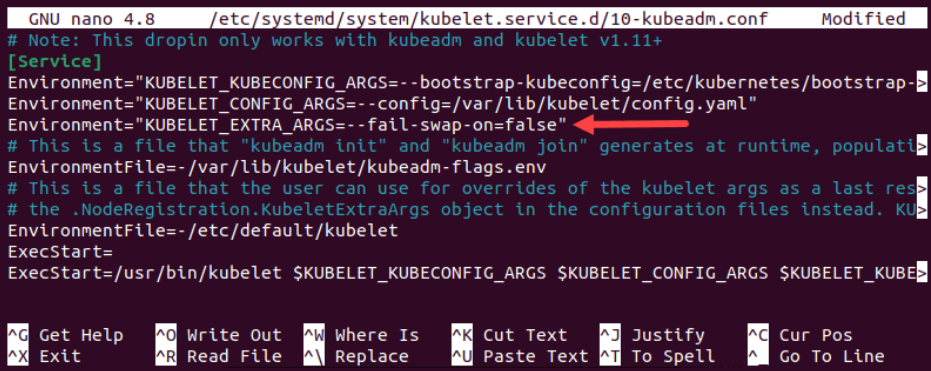


Рис. 3.8. Файл конфігурації Kubelet

Перезавантажуємо Kubelet,вводимо команду systemctl restart kubelet , а далі ініціалізуємо Kubernetes - sudo kubeadm init --control-plane-endpoint=[master-hostname] --upload-certs.

В кінці ми повинні побачити дану картину,що означає вдале встановлення кластеру на робочу машину.

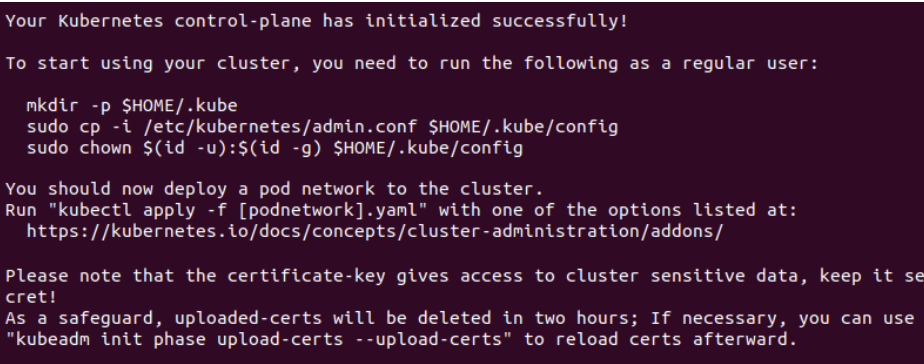


Рис. 3.9. Завершення встановлення Kubernetes

Після успішного встановлення кластеру Kubernetes, почнемо встановлювати Istio Service Mesh.Для початку перейдемо на офіційний вебсайт продукту і встановимо ресурс версії,яка підходить нам – 1.2.4.

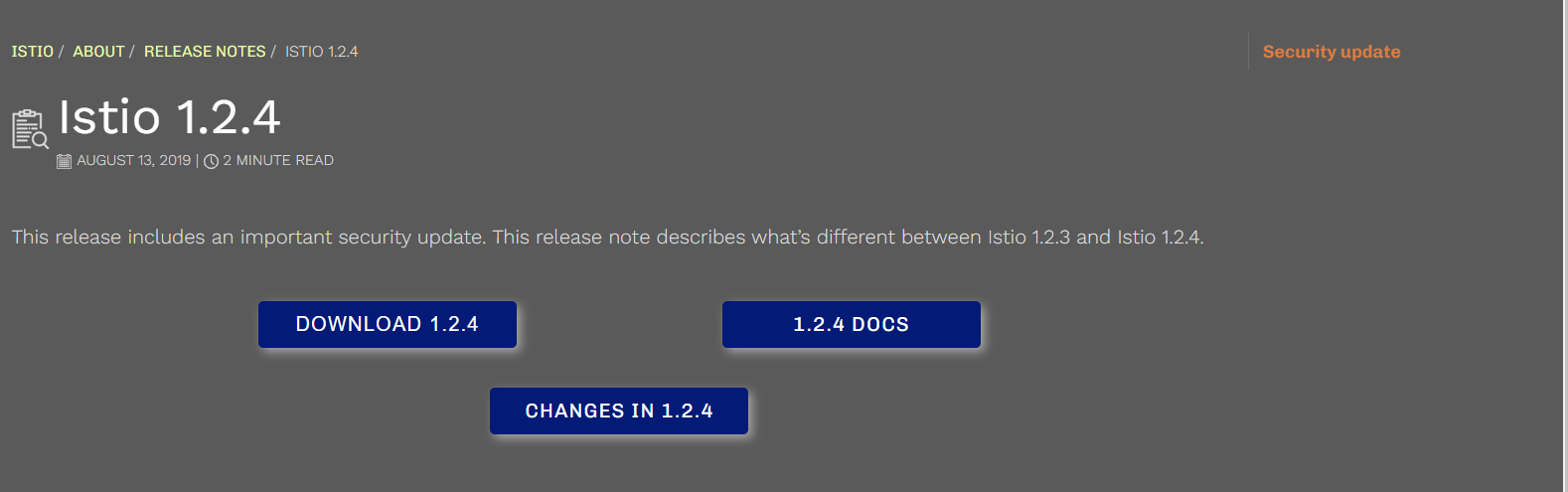


Рис. 3.10. Встановлення образ-ресурсу Istio Service Mesh

Після успішного завантаження образу , звернемось до терміналу та розпакуємо образ.

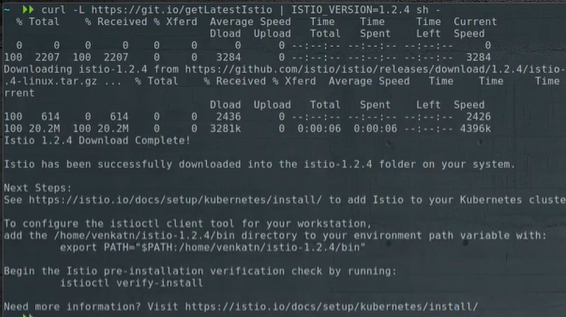


Рис. 3.11. Розпаковка образу Istio Service Mesh

Далі перевіримо чи не було проблем при встановлені образу і всі частини кластеру працюють відповідно – istioctl verify install.



Рис. 3.12. Перевірка правельності роботи частин кластеру Kubernetes

Перевірка пройшла успішно тож наступним кроком буде створення профілю, який надасть змогу користуватись сервісами даного ресурсу такі,як:

* + Grafana;
  + Jaeger;
  + Prometeus;
  + Kiali.

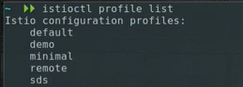


Рис. 3.13. Створення профілю

Командою istioctl manifest apply –set profile=demo можемо спостерігати,які сервіси нам будуть доступні.

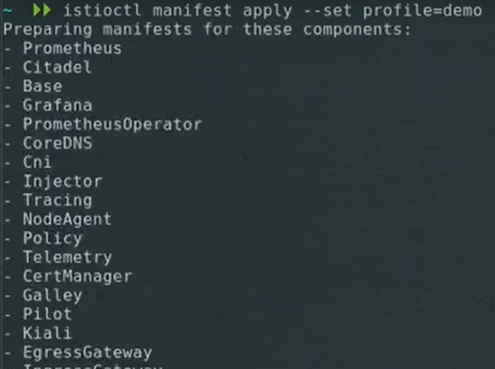


Рис. 3.14. Активація сервісів Service Mesh

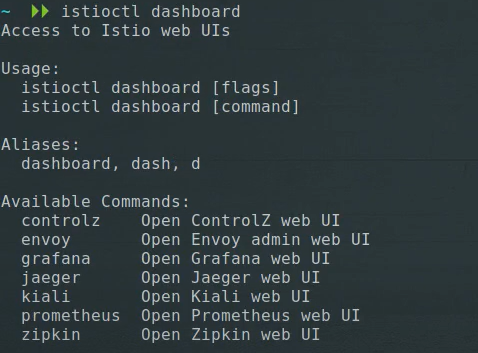


Рис. 3.15. Сервіси,які доступні для роботи

Так,як ми активували дані сервіси, спробуємо підключитись до 1 з них.



Рис. 3.16. Підключення через localhost до сервісу Kiali

Після підключення ,вводимо логін та пароль, вони стандартні – admin,admin. При успішному підключення нас буде чекати ось така платформа.

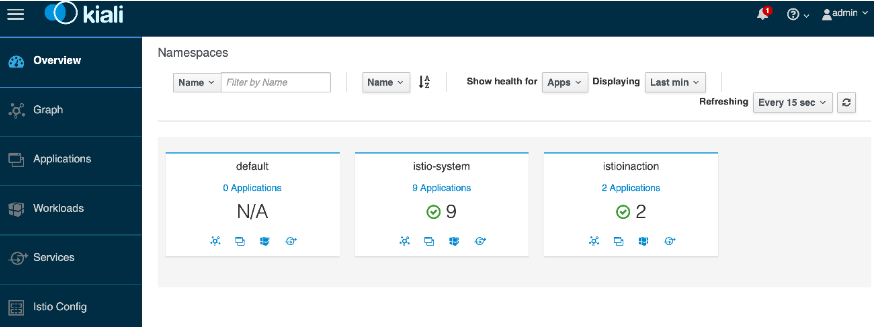


Рис. 3.17. Istio Service Mesh Kiali Dashboard

За допомогою данної інформаційної панелі, ми можемо спостерігати подіями які відбуваються на робочій машині користувача.Дана платформа може допомогти більш коректно налаштувати функції безпеки , а також заздалегідь попередити користувача про якийсь кіберінцедент або кібератаку. Перед тим,як перевірити дані сервіси на налаштування безпеки , розгорнемо ще 1 інформаційну панель – Grafana рис 3.18.



Рис. 3.18. Підключення до сервісу Grafana

При успішному підключенні бачимо дану інформаційну панель

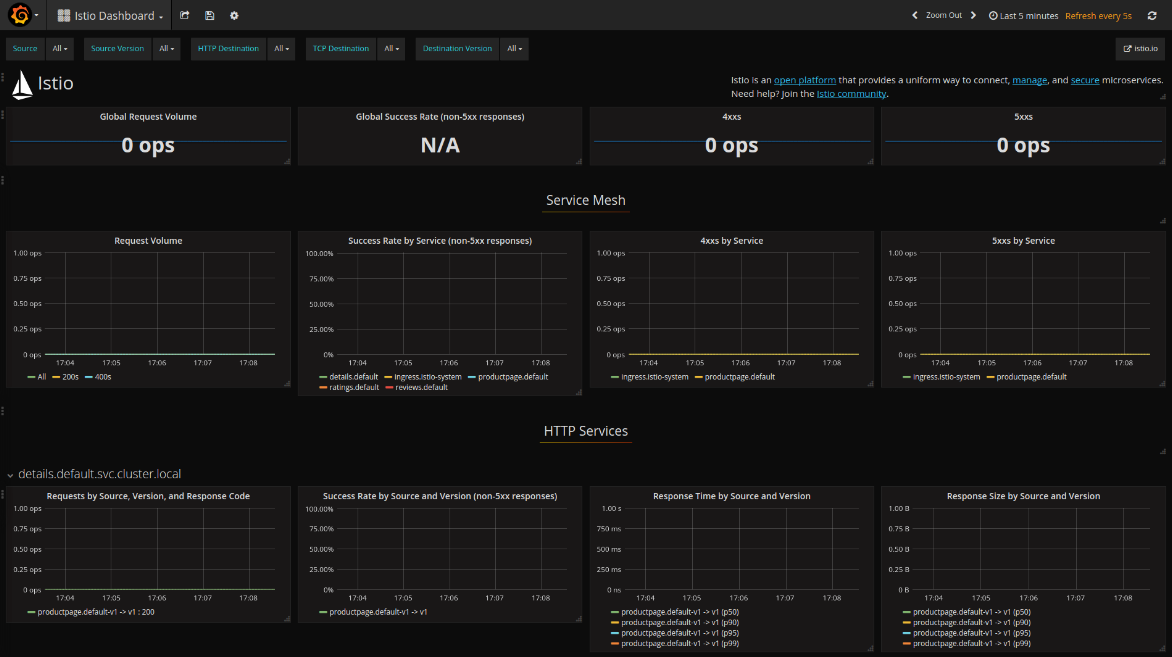


Рис.3.19. Grafana Istio Dashboard

Так,як основною задачею даного сервісу є надання сигналів про інциденти, блокування невідомих URL-посилань , а також блокування невідомих HTTP-запитів, налаштуємо саме дану інформаційну панель і перевіримо, як добре вона буде справлятись з даними завданнями безпеки. Бачимо,що сама панель зараз ніяких функцій не виконує,так як не налаштована.[15

Першим кроком до налаштування буде створення так званих сигналів,які будуть надавати нам інформацію про всю активність робочої машини.

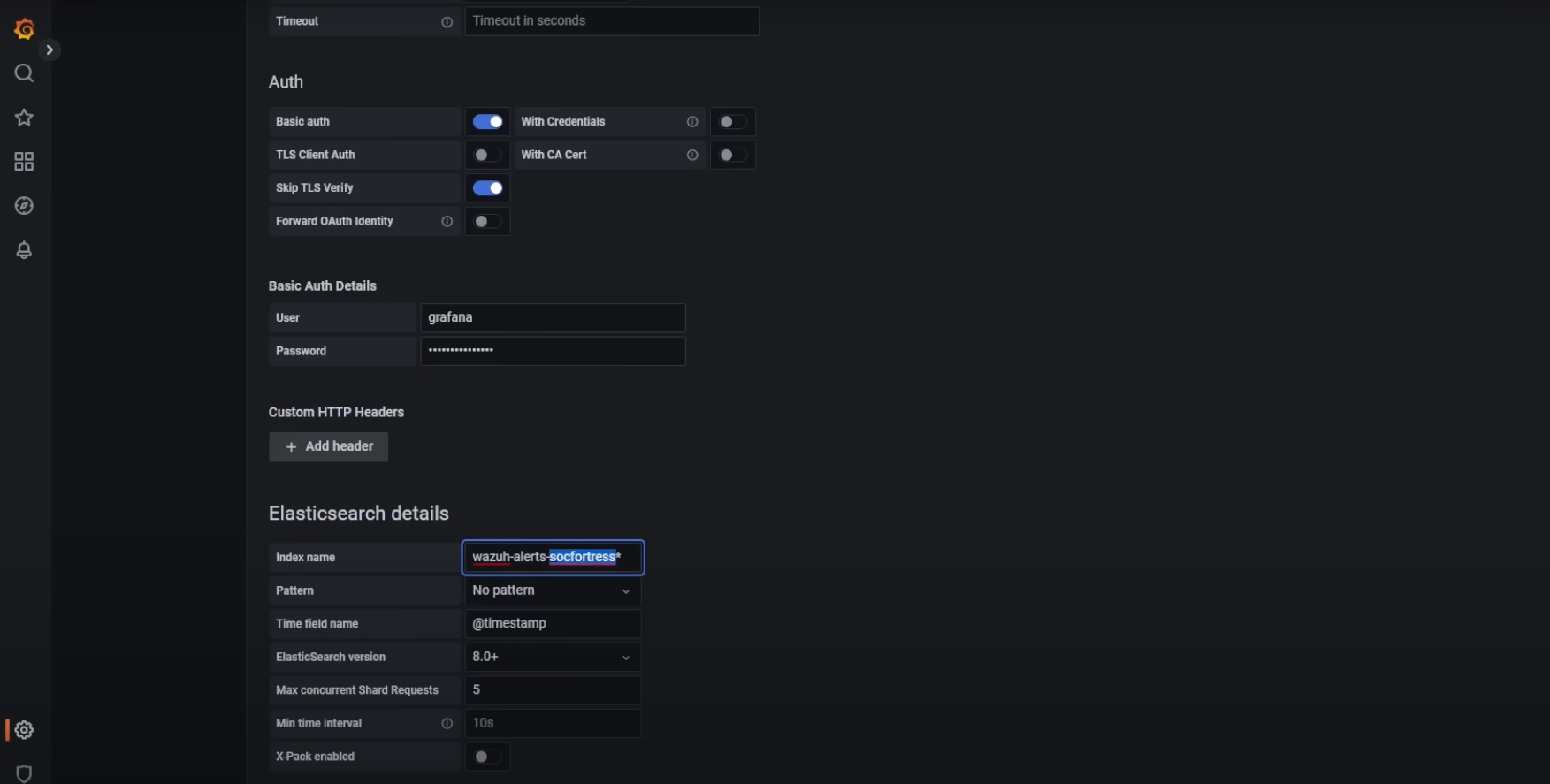


Рис 3.20. Налаштування Alerts в Grafana

В графу URL, додаємо код,який забезпечать в подальшому нас відносно стабільною URL-фільтрацією.

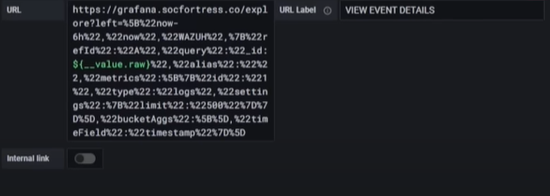
****

Рис.3.21. Налаштування URL-фільтрації

Після активації бачимо, що наша інформаційна панель почала роботу і дає нам вже перші посилання на роботу логів.



Рис.3.22. Панель роботи логів Grafana



Рис 3.23. Фільтрація подій робочої машини

На рисунку 3.23. бачимо,що почалась реєстрація та фільтрація подій робочої машини, інформаційна панель за допомогою основного робочого хоста аналізує всі моменти,які відбуваються і сповіщає користувача про ту чи іншу подію.

Дана інформаційна панель дає ще змогу налаштувати декілька панелей одночасно,які будуть передавати інформацію про логи,як було зазначено раніше,про навантаженність мережі, або навантаженність системи робочої машини.



Рис 3.24. Інформаційна панель навантаженності мережі

**3.2 Розробка рекомендацій щодо використання Service Mesh для підтримки безпеки, моніторингу та управління мережею в середовищі Kubernetes**

У сучасному світі, коли мікросервісна архітектура набуває все більшої популярності, а середовище Kubernetes виступає як одна з найефективніших платформ для керування контейнерами, ми стикаємося з рядом викликів щодо безпеки, моніторингу та управління мережею. Використання Service Mesh може стати відповіддю на ці виклики, забезпечуючи нам необхідні інструменти та практики. Одним з головних переваг використання Service Mesh є забезпечення безпеки. Він надає нам можливість автоматично захищати мікросервіси за допомогою вбудованих механізмів шифрування, аутентифікації та авторизації. Безпека є критично важливим аспектом у мікросервісній архітектурі, і Service Mesh дозволяє нам уникнути багатьох потенційних загроз, забезпечуючи захищену комунікацію між сервісами.

Крім того, Service Mesh дозволяє нам забезпечити ефективний моніторинг мережі та додатків. Він надає механізми збору метрик, трасування запитів та аналізу логів на рівні мікросервісів. Це дозволяє нам отримати детальну інформацію про роботу кожного окремого сервісу, виявляти проблеми та здійснювати швидкий реагування на них. Моніторинг Service Mesh допомагає нам збільшити доступність та продуктивність наших додатків, а також забезпечити належне використання ресурсів мережі.[11]

Управління мережею є ще однією важливою складовою, яку Service Mesh надає. Завдяки різноманітним функціям, таким як маршрутизація, балансування навантаження, контроль трафіку та автоматична підміна сервісів, ми можемо легко керувати та масштабувати наші мікросервіси безпосередньо з Service Mesh. Це дозволяє нам реалізувати гнучку та динамічну архітектуру, а також забезпечує легкість впровадження та зміни сервісів без впливу на весь додаток.

Беручи до прикладу Istio Service Mesh та його внутрішні інструменти,можна сказати,що Istio Dashboard Grafana — це потужний інструмент, який надає графічний інтерфейс користувача для моніторингу та візуалізації показників, зібраних Istio Service Mesh.

Grafana — це популярна платформа для аналітики та моніторингу з відкритим кодом, яка дозволяє користувачам створювати настроювані інформаційні панелі для візуалізації даних із різних джерел.

Інтеграція Istio Dashboard Grafana дозволяє користувачам отримати уявлення про продуктивність і поведінку їхніх мікросервісів, запущених на Istio. Він пропонує багатий набір попередньо створених візуалізацій і віджетів, які можна налаштувати для відображення в реальному часі та історичних даних про трафік служби, затримку, рівень помилок та інші важливі показники.

Використовуючи функції спостереження Istio, такі як розподілене трасування та збір показників, Grafana може отримувати та відображати детальну інформацію про мережеву інфраструктуру служби. Це дозволяє користувачам детально вивчати конкретні служби або кінцеві точки, аналізувати тенденції з часом і визначати потенційні вузькі місця або проблеми в системі.

За допомогою Istio Dashboard Grafana користувачі можуть створювати інтерактивні та динамічні панелі інструментів, які надають повне уявлення про середовище мікросервісу. Вони можуть налаштовувати різні панелі та графіки для моніторингу ключових показників продуктивності, налаштовувати сповіщення на основі конкретних порогових значень і отримувати цінну інформацію про загальний стан і поведінку своїх програм.[19]

Крім того, Grafana пропонує гнучкість для інтеграції з іншими джерелами даних і сторонніми плагінами, дозволяючи користувачам включати додаткові показники та інформацію на свої інформаційні панелі Istio. Ця можливість розширення робить його універсальним інструментом для моніторингу не лише Istio Service Mesh, але й інших компонентів і систем у середовищі Kubernetes.

Інтеграція Istio Dashboard Grafana забезпечує зручний і настроюваний інтерфейс для моніторингу та візуалізації показників, зібраних Istio Service Mesh. Це дає змогу користувачам отримувати цінну інформацію, вирішувати проблеми та оптимізувати продуктивність своїх мікросервісів у середовищі Kubernetes.

Проаналізувавши,можна сказати, що дана інформаційна панель несе в собі декілька переваг,якщо порівнювати її з іншими панелями,наприклад Prometeus чи Kiali:

1. Візуалізація мережевої статистики: Grafana Istio Dashboard дозволяє візуалізувати статистику мережевого трафіку, що допомагає операторам мережі зрозуміти, як працює мережа, і виявляти проблеми в реальному часі.
2. Моніторинг мікросервісів: Grafana Istio Dashboard дозволяє моніторити мікросервіси, які працюють на Istio, і виявляти проблеми з швидкістю, доступністю та іншими метриками.
3. Легке налаштування: Grafana Istio Dashboard має легкий інтерфейс для налаштування, що дозволяє користувачам мережі швидко і легко встановити та налаштувати панель моніторингу Istio.
4. Підтримка інших додатків: Grafana Istio Dashboard може інтегруватися з іншими додатками моніторингу, такими як Prometheus та Jaeger, для більш детального аналізу мережі.
5. Отримання інформації у реальному часі: Grafana Istio Dashboard надає користувачу мережі доступ до реального часу інформації про мережу та мікросервіси, що дозволяє швидко реагувати на проблеми та вдосконалювати мережу.

Так,як дана панель включає в собі налаштування безпеки,можна навести декілька рекомендації для коректного налаштування параметрів безпеки,що допоможуть користувачеві в стабільному та безпечному користуванні данної платформи:

1. Захистіть доступ до самої панелі управління Grafana. Встановіть потужні паролі для адміністраторського облікового запису та переконайтесь, що вони не викладені у відкритий доступ. Кращим варіантом є використання автентифікації з двофакторним підтвердженням.
2. Обмежте доступ до Istio Grafana Dashboard за допомогою налаштувань мережевої політики Istio. Встановіть правила, які дозволяють доступ до панелі управління лише певним користувачам або IP-адресам. Ви можете використовувати правила включення/виключення для точного контролю над доступом.
3. Постійно оновлюйте Grafana та Istio до останніх версій. Перевіряйте наявність оновлень та застосовуйте їх для корекції виявлених безпекових уразливостей.
4. Моніторинг системи та журналізація. Підтримуйте моніторинг подій та логів, щоб швидко виявляти підозрілу активність та потенційні атаки на систему.[18]

**Висновки до третього розділу**

Реалізовано віртуальне середовище для встановлення кластеру та сервісів для подальшого їх застосування.

Проаналізовано існуючі сервісні сітки та їх функціонал. Застовуючи Istio Service Mesh, як віртуальне середовище, були проведені налаштування інформаційної панелі Grafana, за допомогою якої, досліджено функціонал і можливості самої сервісної сітки.

Розроблено рекомендації щодо використання Service Mesh для підтримки безпеки, моніторингу та управління мережею в середовищі Kubernetes.

**ВИСНОВКИ**

В бакалаврській роботі отримано наступні наукові та науково-практичні результати:

* Досліджено структурні компоненти контейнерних платформ,які дозволяють ефективно керувати та оркеструвати контейнерні додатки, а також проаналізавано структуру Service Mesh, її використання.
* Проведена аналітична робота щодо переваг використання Service Mesh у Kubernetes-середовищі, досліджено популярні рішення та їх функціональні можливості.
* Досліджено поняття та принципи роботи Istio Service Mesh, функції безпеки Kubernetes і те, як вони можуть допомогти захистити ваші контейнерні програми та складено детальний список функціоналу безпеки та більш поглибленно розкрита робота кожного з цих функцій
* Проведена робота зі збірки та конфігурації Service Mesh на основі Istio в Kubernetes-середовищі.
* Досліджено різні методики відслідковування трафіку та моніторингу мережі в Service Mesh, включаючи використання інструментів моніторингу, таких як Kiala та Grafana.
* Розроблено рекомендації щодо використання Service Mesh у Kubernetes середовищі для підтримки масштабованих і стійких розгортань, з урахуванням безпеки, моніторингу та оптимізації продуктивності.

**ПЕРЕЛІК ПОСИЛАНЬ**

1. What is Linkerd? [Електронний ресурс] – Режим доступу: https://www.trustradius.com/products/linkerd/reviews#overview
2. Introduction to Containers [Електронний ресурс] – Режим доступу: https://www.digitalocean.com/community/conceptual-articles/introduction-to-containers
3. Principles of Container-based Application Design [Електронний ресурс] – Режим доступу: https://kubernetes.io/blog/2018/03/principles-of-container-app-design/
4. What are Container Platforms? [Електронний ресурс] – Режим доступу: https://www.aquasec.com/cloud-native-academy/container-platforms/container-platforms-6-best-practices-and-15-top-solutions/
5. What is a Service Mesh? [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchitoperations/definition/service-mesh
6. What is a service mesh, and why do you needit?[Електронний ресурс] – Режим доступу: https://www.kellton.com/kellton-tech-blog/what-is-a-service-mesh-and-why-do-you-need-it
7. A Kubernetes Service Mesh Comparison [Електронний ресурс] – Режим доступу: https://www.toptal.com/kubernetes/service-mesh-comparison
8. Kubernetes Service Mesh: A Comparison of Istio, Linkerd, and Consul [Електронний ресурс] – Режим доступу: https://platform9.com/blog/kubernetes-service-mesh-a-comparison-of-istio-linkerd-and-consul/
9. List of Best Service Mesh Tools For Microservices [Електронний ресурс] - Режим доступу: https://devopscube.com/service-mesh-tools/
10. Service meshes: an in-depth introduction [Електронний ресурс] – Режим доступу: https://www.jetstack.io/blog/service-meshes-a-deeper-introduction/
11. Istio Service Mesh in 2020 [Електронний ресурс] – Режим доступу: https://www.cncf.io/blog/2020/05/25/istio-service-mesh-in-2020/
12. Find the right service mesh tool for Kubernetes environments [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchitoperations/tip/Find-the-right-service-mesh-tool-for-Kubernetes-environments
13. How to implement Docker image scanning with open source tools [Електронний ресурс] – Режим доступу: https://sysdig.com/blog/docker-image-scanning/
14. How to install Kubernetes. [Електронний ресурс] – Режим доступу: https://ubuntu.com/kubernetes/install
15. Deploy and Access the Kubernetes Dashboard [Електронний ресурс] – Режим доступу: https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/
16. How To Install Kubernetes Cluster On Ubuntu. [Електронний ресурс] – Режим доступу: https://www.edureka.co/blog/install-kubernetes-on-ubuntu
17. Guided Exercise: Installing Istio on a Minikube Cluster [Електронний ресурс] – Режим доступу: https://kubebyexample.com/learning-paths/istio/install
18. Advantages of Using Kubernetes [Електронний ресурс] – Режим доступу: https://www.linode.com/docs/guides/kubernetes-use-cases/
19. Enable Istio in the Cluster [Електронний ресурс] – Режим доступу: https://ranchermanager.docs.rancher.com/v2.5/how-to-guides/advanced-user-guides/istio-setup-guide/enable-istio-in-cluster
20. Download Istio [Електронний ресурс] – Режим доступу: https://istio.io/latest/docs/setup/getting-started/

**ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)**