МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Дипломна магістерська робота

на тему: **Розробка програмного забезпечення для «Системи ідентифікації студентів»**

Виконав: студент групи МгІТ 1-20
спеціальності 122 Комп'ютерні науки
освітньої програми Комп'ютерні науки

Віталій ГЛЕМБОЦЬКИЙ

Керівник: к.т.н., доц. Тетяна АСТІСТОВА

Рецензент: д.т.н. проф. Володимер ЩЕРБАНЬ

Київ 2021

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Спеціальність **122 Комп'ютерні науки**
Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КНТ
_____проф.Щербань В.Ю.
"_____" _____2021 року

З А В Д А Н Н Я

НА ДИПЛОМНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Глембоцькому Віталію Сергійовичу

1. Тема роботи *Розробка програмного забезпечення для «Системи ідентифікації студентів»*,
науковий керівник роботи *Астістова Т.І., к.т.н.,доцент,*
затверджені наказом вищого навчального закладу від "04" листопада 2021 року наказ №286
2. Строк подання студентом роботи 20.11.2021р.
3. Вихідні дані до роботи Розробка кафедри комп'ютерних наук.
Розробка програмного забезпечення для системи ідентифікації студентів
4. Зміст дипломної роботи Розділ1. Технології та хмарні платформи; Розділ 2. Архітектурне забезпечення та взаємодія компонетів сервера; Розділ 3. Розробка системи додатку.

5. Консультанти розділів дипломної магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Вступ	*Тетяна АСТІСТОВА к.т.н., доц*		
Розділ1	*Тетяна АСТІСТОВА к.т.н., доц*		
Розділ 2	*Тетяна АСТІСТОВА к.т.н., доц*		
Розділ 3	*Тетяна АСТІСТОВА к.т.н., доц*		

6. Дата видачі завдання 10.2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної магістерської роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	05.10.2021	
2	Розділ 1 Технології та хмарні платформи	25.10.2021	
3	Розділ 2 Архітектурне забезпечення та взаємодія компонетів сервера	30.10.2021	
4	Розділ 3 Розробка системи додатку	10.11.2021	
5	Висновки	10.11.2021	
6	Оформлення дипломної магістерської роботи (чистовий варіант)	15.11.2021	
7	Здача дипломної магістерської роботи на кафедру для рецензування (за 14 днів до захисту)	*10.12.21*	
8	Перевірка дипломної магістерської роботи на наявність ознак плагіату (за 10 днів до захисту)	*14.12.21*	
9	Подання дипломної магістерської роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	*17.12.21*	

Студент Віталій ГЛЕМБОЦЬКИЙ

Науковий керівник роботи Тетяна АСТІСТОВА

Директор НМЦУПФ Олена ГРИГОРЕВСЬКА

АНОТАЦІЯ

Глембоцький В.С. Розробка програмного забезпечення для «Системи ідентифікації студентів»

Дипломна магістерська робота за спеціальністю 122 - «Компю'терні науки». – Київський національний університет технологій та дизайну, Київ, 2021 рік.

Дипломна робота присвячена розробці програмного забезпечення для додатку «Системи ідентифікації студентів». Під час виконання роботи було розроблено архітектуру програмного забезпечення що задовольняє принципи «Cloud Computing Architecture» та змодельовано систему роботи архітектури.

Для досягнення результату було створенно Amazon ECS cluster за допомогою інструмента Terraform який довзоляє декларативно керувати інфраструктурою та створювати потрібні нам інстанси та мережі.

Дипломна робота демонструє повноцінну картину взаємодію між усіма елементами системи, що дає комплексне представлення про хмарні технології, їхні ресурси та переваги які вони надають користувачеві.

Ключові слова: Програмне запезпечення, архітектура, Amazon ECS cluster, Terraform

ANNOTATION

Glembotskiy V.S. Software development for Student Identification System It is Manuscript.

Diploma master's degree work on speciality 122 are « Computer Science ». It is the Kiev a national university of technologies and design, Kiev, 2021 year.

Thesis is devoted to the development of software for the application of the student identification system. In the course of the work, a software architecture was

developed that satisfies the principles of "Cloud Computing Architecture" and a system of architecture was modeled.

To achieve the result, the Amazon ECS cluster was created using the Terraform tool, which allows you to declaratively manage the infrastructure and create the instances and networks we need.

The diploma project demonstrates a complete picture of the interaction between all elements of the system, which gives a comprehensive view of cloud technologies, their resources and the benefits they provide to the user.

Keywords: Software, architecture, Amazon ECS cluster, Terraform

ЗМІСТ

ВСТУП 7

Розділ 1 Технології та хмарні платформи 10

1.1 Віртуальний сервер у хмарі Elastic Compute Cloud 10

1.2 Сервіс керування контейнерами Elastic Container Service 13

1.3 Балансувальник навантаження на сервер Elastic Load Balancing 16

1.4 Огляд баз данних 19

1.5 Документо-орієнтованаі бази даних MongoDB та DocumentDB 23

1.6 Інструмент написання інфраструктури сервера Terraform 29

1.7 Висновок до розділу 33

Розділ 2 Архітектурне забезпечення та взаємодія компонентів сервера 34

2.1 Загальна концепція компонентів програмного запезпечення 34

2.2 Модуль Kubernetes API 40

2.3 Графічний інтерфейс Kubernetes Lens 43

2.4 Пакетний менеджер модулів Helm Chart 49

2.5 Висновок до розділу 52

Розділ 3 Розробка системи додатку **53**

3.1 Структура системи ідентифікації студентів 53

3.2 Написання коду для створення сервера 55

3.3 Налаштування серверу для роботи з додатком 61

3.6 Висновок до розділу 65

ВИСНОВКИ 66

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ 67

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 68

ВСТУП

Актуальність. Подальший розвиток інформаційних технологій України залежить від того, наскільки вона відповідатиме вимогам інформаційно-індустріального суспільства XXI століття. Попит на IT-технології в Україні невпинно зростає: продовжується автоматизація різних виробничих процесів, розробка додатків, збільшуються потреби в «хмарних» сервісах. Тому, я переконаний, що додаток «Система ідентифікації студентів» прискорить та полегшить взаємодію студентів та викладачів і оптимізує процес їхньої взаємодії.

Мета і завдання. Метою роботи є автоматизація системи обліку відвідувань занять студентами. Завдання – створити функціонал для мобільного додатку, сайт, базу данних та сервер освітнього процесу для зберігання інформації .

Об'єкт та предмет дослідження. Відсутність достатньої ефективності та практичності використання ресурсів для проведення обліку занять стдудентів. В результаті ми отримуємо зниження ефективності навчання студентів.

Методи та засоби дослідження. Для досягнення поставлених цілей було вирішенно створити повноцінний додаток для студентів та викладачів. Стек технологій для реалізації: Frontend розробка із використанням реактивності, документоорієнтована база даних MongoDB, технологія QR-код, AWS, Terraform та Helm.

Наукова новизна та практичне значення отриманих результатів. Новизною даного дослідження є створення та розгортання інфраструктури сервера для «Системи ідентифікації студентів», що поєднує у собі новітні технології хмарних серверів та простоту використання мікросерівсів. Використання технологій Kubernetes та Terraform додає унікальності данній розробці.

Мета проєкту. Метою проєкту є створення простого у використанні додатку, що поєднує у собі новітні технології хмарних обчислень, потужних та швидких серерів та простоту використання.

Результати дослідження. Розрізняють статичні та динамічні QR-коди. Статичний QR-код містить інформацію, яку зазначили при його генеруванні. Динамічний QR-код є багатофункціональним: до нього можна підключати додаткові функції, які будуть виконуватись одночасно. Різновидами QR-кодів є DataMatrix та Aztec Code. Сучасні мобільні телофни, смартфони та планшети мають вбудоване програмне забезпечення для зчитування та розпізнавання QR-коду.

Для написання програми було обрано один із фреймворків JS, а саме React-JS. Бібліотека React була вперше випущена компанією Facebook у 2013 році. React це бібліотека для створення користувацьких інтерфейсів. Однією з неї.

Характерними рисами є можливість використовувати мову JSX. Ця мова програмування з близьким до HTML синтаксисом, який компілюється у JavaScript. Створені компоненти можуть бути легко змінені і використані заново в нових проєктах.

Для створення бази дннаих було обранно MongoDB - це нереляційна база даних і її легко масштабувати. Це база даних з відкритим кодом, написана на C++. Основні переваги MongoDB:

- Відсутність схеми
- Дана БД заснована на колекціях різних документів. Кількість полів, зміст і розмір цих документів може відрізнятись. Тобто різні сутності не повинні бути ідентичні за структурою.
- Вкрай зрозуміла структура кожного об'єкта.
- Легко масштабується
- Для зберігання використовуваних на даний момент даних використовується внутрішня пам'ять, що дозволяє отримати більш швидкий доступ.
- Дані зберігаються у вигляді JSON документів

- MongoDB підтримує динамічні запити документів (document-based query)
- Відсутність складних JOIN запитів

Для створення підтримки та інтеграції нашого сервера обранно AWS (Amazon Web Service) – це платформа хмарних обчислень, яка дозволяє користувачам отримувати доступ до обчислювальних служб на вимогу, таких як сховище бази даних, віртуальний сервер, VPS, S3 Bucket та багато інших.

Теоретична цінність. Теоретична цінність проведеної роботи полягає в тому, що в ньому зроблено аналіз дійсних моделей систем ідентифікації студентів та запропоновано новітній цифровий варіант, який може бути використаний в ході подальшого вивчення цього питання.

Практична цінність. Практична цінність полягає у розробці додатку який допоможе оптимізувати освітній процес відвідування студентами занять та зробить його ефективнішим.

Апробація результатів роботи.

Результати роботи були апробовані на міжнародній конференції та в статті.

1. Астістова Т.І., Розробка системи ідентифікації студентів/ Т.І. Астістова, В.С. Глембоцький // Тези V Міжнародної науково-практичної конференції «Мехатронні системи: інновації та інжиніринг – «MSIE-2021» К. КНУТД , 4 листопада 2021р. - С. 153-154
2. Астістова Т. І., Глембоцький В.С., Програмне забезпечення для системи ідентифікації студентів// Т.І. Астістова, В.С.Глембоцький // Інформаційні технології в науці, виробництві та підприємництві: зб. наук. праць молодих вчених, аспірантів, магістрів кафедрикомп’ютерних наук та технологій. – К. : Освіта України, 2021 р. – С 208 – 211

Розділ 1 ТЕХНОЛОГІЇ ТА ХМАРНІ ПЛАТФОРМИ

1.1 Віртуальний сервер у хмарі Elastic Compute Cloud

При аналізі ринку хмарних платформ та порівнянні якості та швидкодії сервісів вибір впав на Amazon Web Services. AWS почали працювати з користувачами в якості платформи з технологічною інфраструктурою в 2006 р. В даний час більше мільйона активних користувачів використовують AWS для вирішення найрізноманітніших завдань.

Однією з найбільших переваг AWS є те, що він може обслуговувати організації будь-якого розміру. Такі великі компанії як Netflix і Expedia покладаються на AWS для надання послуг по всьому світу. І малий бізнес теж може знайти все необхідне. Серед плюсів Amazon можна виділити такі:

- З AWS легко масштабувати – він може підтримувати практично необмежену кількість користувачів.
- Гнучкість налаштування та підтримки сторонніх інтеграцій – практично будь-яка організація може обслуговуватися через AWS.
- Аналітика в реальному часі та рішення для великих даних, доступні через фірмові програми Amazon Kinesis Streams та Firehose. Потрібна невелика команда розробників, яка зможе їх налаштувати.
- Оновлення та постійні нові функції дуже важливі, оскільки ця платформа продовжує оцінювати потреби клієнтів.

Обчислювальна «хмарна» технологія Amazon Elastic Compute Cloud (Amazon EC2) – це веб-сервіс, що надає безпечні масштабовані обчислювальні ресурси в хмарі. Він допомагає розробникам, спрощуючи проведення обчислень у хмарі масштабу всього Інтернету. Простий веб-інтерфейс Amazon EC2 дозволяє отримати доступ до обчислювальних ресурсів і налаштувати їх з мінімальними зусиллями. Він надає користувачам повний контроль над обчислювальними ресурсами, а також перевірене обчислювальне середовище Amazon для роботи. Amazon EC2 пропонує обчислювальну платформу з найбільш широкими та досконалими функціональними можливостями, яка

дозволяє вибрати процесор, сховище, мережу, операційну систему та модель пок. AWS пропонує найшвидші процесори в хмарі і є єдиним простором хмари з мережею Ethernet, пропускна здатність якої становить 400 Гбіт/с. Сервіс має найпотужніші інстанси на базі графічних процесорів для організації курсів з машинного навчання та графічного відображення робочих навантажень, а також інстансами з найнижчими витратами на кожен логічний висновок інстансів у хмару. На AWS виконується більше робочих навантажень SAP, HPC, машинного навчання та Windows, ніж у будь-якій іншій хмарі. Балансування навантаження та автомасштабування є дуже важливими функціями EC2. Ви можете створити правила, при яких можна автоматично збільшити кількість серверів, наприклад, якщо один або кілька серверів не справляються з навантаженням. Контроль за здоров'ям серверів веде ще один сервіс AWS – Amazon Cloud Watch. За допомогою цього сервісу можна створювати різні перевірки — checks — за допомогою яких контролюються найважливіші показники роботи.

Amazon EC2 надає наступні функції:

- Віртуальні обчислювальні середовища, відомі як екземпляри
- Попередньо налаштовані шаблони для ваших інстансів, відомі як образи машин Amazon (AMI), які упаковують біти, необхідні для вашого сервера (включаючи операційну систему та додаткове програмне забезпечення)
- Різні конфігурації ЦП, пам'яті, сховища та мережевої ємності для ваших екземплярів, відомі як типи екземплярів.
- Захистіть дані для входу в свої екземпляри за допомогою пар ключів (AWS зберігає відкритий ключ, а ви зберігаєте закритий ключ у безпечному місці)
- Томи сховища для тимчасових даних, що видаляються при зупинці, переведенні в сплячий режим або завершенні роботи екземпляра, відомі як томи сховища екземплярів.

- Тома постійного сховища для ваших даних за допомогою Amazon Elastic Block Store (Amazon EBS), відомі як томи Amazon EBS
- Декілька фізичних прихильностей для ваших ресурсів, таких як інстанси та томи Amazon EBS, відомі як регіони та зони доступності.
- Брандмауер, який дозволяє вам вказувати протоколи, порти та діапазони вихідних IP-адрес, які можуть досягати ваших екземплярів за допомогою груп безпеки.
- Статичні IPv4-адреси для динамічних хмарних обчислень, відомі як еластичні IP-адреси.
- Метадані, відомі як теги, які ви можете створювати та призначати своїм ресурсам Amazon EC2.
- Ви можете створювати віртуальні мережі, які логічно ізольовані від решти хмари AWS і які ви можете за бажання підключати до своєї власної мережі, відомої як віртуальні приватні хмари (VPC).

Amazon EC2 підтримує створення ресурсів за допомогою AWS CloudFormation. Ви створюєте шаблон у JSON або YAML, який описує ваші ресурси AWS, а AWS CloudFormation надає та настроює ці ресурси за вас. Ви можете повторно використовувати свої шаблони CloudFormation для надання одних і тих же ресурсів кілька разів, будь то в одному регіоні та обліковому записі або в декількох регіонах та облікових записах.

Amazon EC2 надає API запит. Ці запити є запитами HTTP або HTTPS, в яких використовуються команди HTTP GET або POST та параметр запиту з ім'ям Action. Для отримання додаткових відомостей про дії API для Amazon EC2 див. розділ «Дії» у Довіднику Amazon EC2 . Для створення програми з використанням API-інтерфейсів для конкретних мов, а не надсилання запитів через HTTP або HTTPS, AWS надає бібліотеки, зразки коду, посібники та інші ресурси для розробників програмного забезпечення. Ці бібліотеки надають базові функції, які автоматизують такі завдання, як криптографічний підпис

наших запитів, повторна спроба запитів та обробка відповідей про помилки, що спрощує початок роботи.

1.2 Сервіс керування контейнерами Elastic Container Service

Amazon Elastic Container Service – це високопродуктивний сервіс керування контейнерами з широкими можливостями масштабування. Він підтримує контейнери Docker і дозволяє легко запускати програми в автоматично керованому кластері інстансів Amazon Elastic Compute Cloud (Amazon EC2).

Контейнери дозволяють легко упаковувати код програми, його налаштування та залежності у зручні компонувальні блоки, що забезпечують однаковість середовища, ефективність виконання операцій, продуктивну розробку та можливість керування версіями. Контейнери забезпечують швидке, надійне та одноманітне розгортання програм незалежно від середовища розгортання.

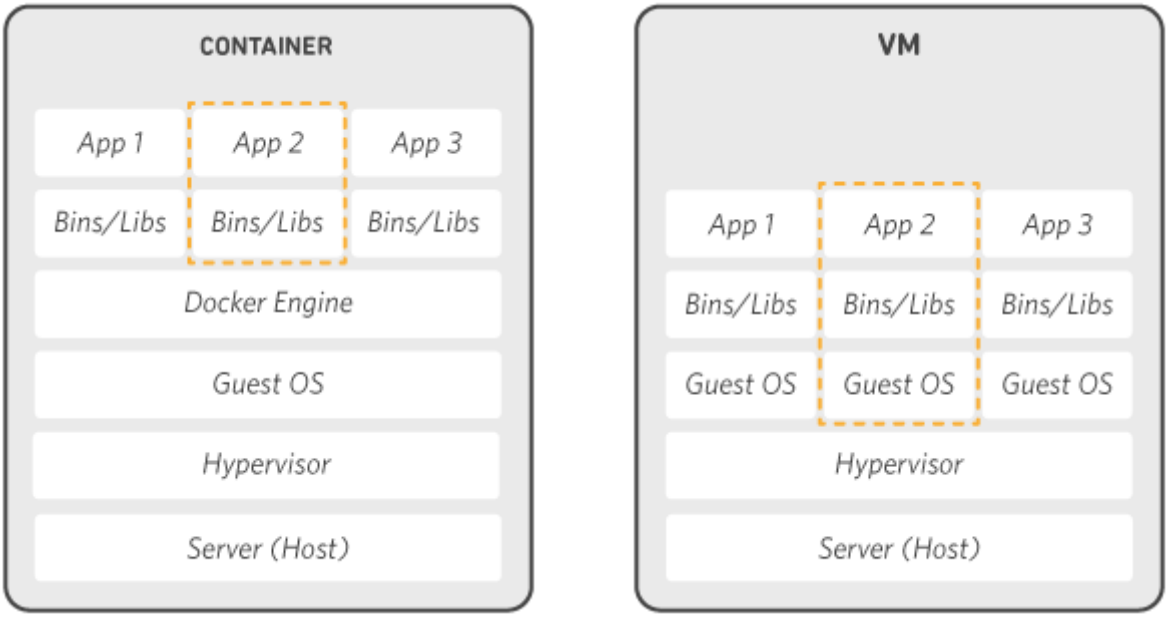


Рис. 1.1. Вміст контейнерів

Запуск контейнера з новим кодом можна виконати без значних додаткових витрат на розгортання. Швидкість роботи покращена, оскільки код, створений у контейнері на локальній машині розробника, можна легко перемістити на тестовий сервер – достатньо перемістити контейнер. Під час створення цей

контейнер можна зв'язати з іншими контейнерами, які необхідні для запуску стека програми.

Образ контейнера Docker – це своєчасне захоплення коду програми та залежностей. Це дозволяє проєктній організації створити стандартний конвеєр для життєвого циклу програми. Нижче наведено відповідні приклади.

1. Розробники створюють та запускають контейнер на локальному комп'ютері.
2. Сервер безперервної інтеграції запускає той же контейнер та виконує з ним інтеграційне тестування для перевірки відповідності вимогам.
3. Цей контейнер передається в проміжне середовище, де його поведінку під час роботи можна перевірити за допомогою навантажувального або ручного тестування.
4. Цей же контейнер передається у виробниче середовище.

Можливість створювати, тестувати, передавати та запускати один і той же контейнер на всіх етапах процесу інтеграції та розгортання значно спрощує розробку високоякісних та надійних додатків

Контейнери сприяють підвищенню ефективності використання ресурсів, дозволяючи дещо різнорідним процесам працювати в одній системі. Ефективність використання ресурсів – це природний результат використання контейнерами методів ізоляції та розподілу. Для контейнерів можна встановлювати граничні значення використання ЦПУ та пам'яті хоста. З'ясувавши, які ресурси необхідні для контейнера і які ресурси доступні на базовому сервері хоста, можна визначити обчислювальні ресурси для використання з меншими хостами або збільшити щільність процесів, запущених на одному великому хості. Це підвищує рівень доступності та дозволяє оптимізувати використання ресурсів.

Гнучкість контейнерів Docker заснована на їхній мобільності, простоті розгортання та невеликому розмірі. На відміну від віртуальної машини вони не вимагають встановлення та налаштування. Сервіси стиснення, вбудовані в контейнери, дозволяють легко переміщувати їх між хостами, ізолювати від

збоїв інших суміжних сервісів та захищати від помилкових виправлень або оновлення програмного забезпечення в центральній системі.

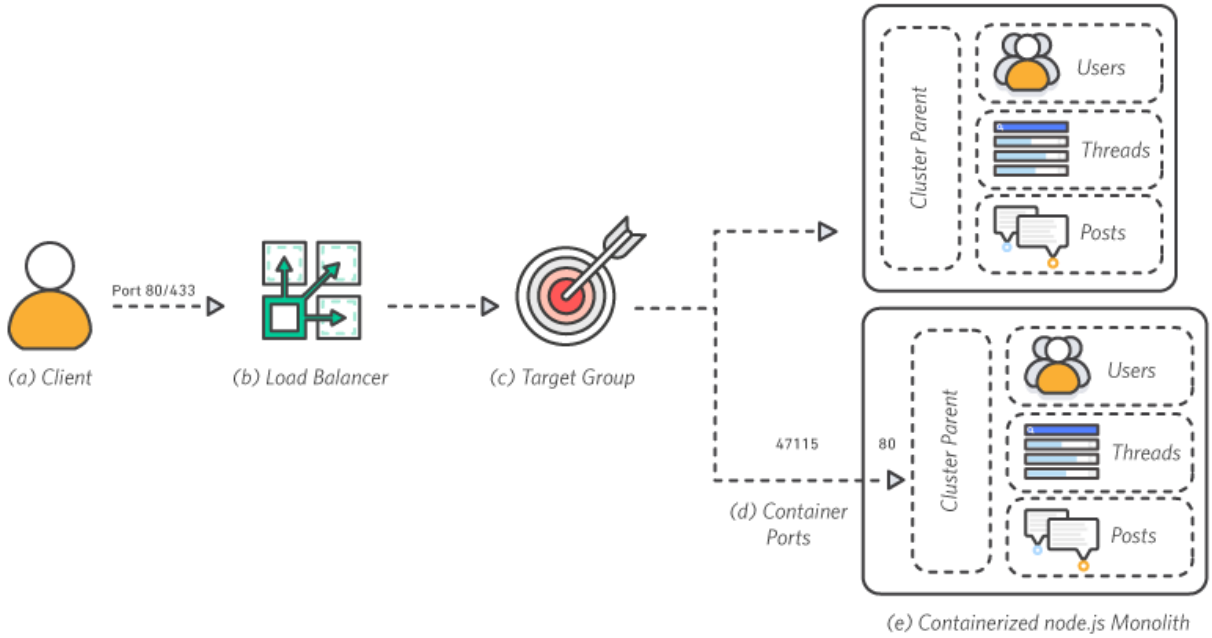


Рис. 1.2. ECS архітектура

a. Клієнт

Клієнт здійснює запити до балансувальника навантаження через порт 80

b. Балансувальник навантаження

Балансувальник навантаження розподіляє запити на всі доступні порти.

c. Цільові групи

Інстанси зареєстровані в цільових групах програми.

d. Порти контейнерів

Кожен контейнер запускає єдиний процес застосунку, який прив'язує батьківський елемент кластера node.js до порту 80 всередині його простору імен.

e. Контейнеризований монолітний додаток node.js

Батьківський об'єкт кластера node.js відповідає за розподіл трафіку за виконавцями всередині монолітної програми. Ця архітектура є контейнеризованою, але так само і монолітною, оскільки кожен контейнер виконує одні й самі функції.

1.3 Балансувальник навантаження на сервер Elastic Load Balancing

Elastic Load Balancing автоматично розподіляє вхідний трафік додатків за декількома цільовими об'єктами та віртуальними пристроями в одній або декількох зонах доступності.

Рис. 1.3. Взаємодія компонентів

Розглянем основні типи балансувальників:

Classic Load Balancer. Перша версія балансувальника від AWS, працює як на четвертому, так і на сьомому рівні OSI, він підтримує HTTP, HTTPS, TCP і SSL. Він підтримує базове балансування навантаження між декількома інстансами Amazon EC2 та працює як на рівні запитів, так і на рівні з'єднання. Насьогодні балансер вважається застарілим, тому рекомендується використовувати лише в окремих випадках. Наприклад, для програм, які були побудовані в мережі EC2-Classic

Network Load Balancer. Підходить для високого навантаження, працює на 4-му рівні OSI (можна використовувати в EKS та ECS), підтримуються TCP, UDP та TLS. Network Load Balancer спрямовує трафік на цільові об'єкти в Amazon VPC і здатний обробляти мільйони запитів за секунду при наднизьких затримках. Крім того, він оптимізований для обробки моделей трафіку з раптовим і змінним навантаженням.

Application Load Balancer. Працює на 7-му рівні, має підтримку Lambda, підтримує правила на рівні заголовків та шляхів, підтримуються HTTP та HTTPS.Забезпечує розширену маршрутизацію запитів, яка орієнтована на доставку додатків, побудованих на базі сучасних архітектур, у тому числі мікросервіси та контейнери. Надсилає трафік на цільові об'єкти в Amazon VPC, спираючись на вміст запиту.

Загальні компоненти Load Balance

- Access Logging Policy - журнали доступу до ELB
- Scheme внутрішній чи зовнішній балансувальник. Ідея полягає в тому, чи повинен LoadBalancer отримати зовнішні адреси, щоб бути доступним зовні, чи це може бути внутрішній балансувальник;
- Security Groups контроль доступу до балансувальника. Якщо описати простіше , це високорівневий файрвол.
- Subnets підмережі які знаходяться всередині VPC. Subnets вказується під час створення. Коли VPC обмежені регіоном, Subnets також обмежений по зонах доступності. Якщо створюєте Load Balancer, краще створювати його принаймні у двох під мережах
- Listeners протоколи балансувальника. Як зауважити сказанне раніше, для Classic Load Balancer це може бути HTTP, HTTPS, TCP і SSL, для Network Load Balancer — TCP, UDP і TLS, для Application Load Balancer — HTTP і HTTPS.

Тепер більш детально розглянемо балансувальники Application Load Balancer та Network Load Balancer. Ці балансувальники мають свої компонентні особливості. Зокрема, з'явилося таке поняття, як Target Groups – інстанси (та функції). За допомогою цього компоненту з'явилася можливість вказувати на який з Target Groups потрібно спрямовувати трафік. Тепер проаналізуємо наступний компонент - Elastic IP. Він надає єдину IP-адресу, яку можна зв'язати з різними екземплярами EC2. Якщо екземпляр EC2 має

Elastic IP-адресу і цей екземпляр завершено або зупинено, можна невідкладно зв'язати новий екземпляр EC2 з адресою Elastic IP. При цьому поточна програма не припинить роботу, оскільки програми бачать все ту ж IP-адресу, навіть якщо реальний EC2 змінився.

ELB нажає автоматичний розподіл вхідного трафіку по кількох цільових об'єктах (контейнери, інстанси Amazon EC2, IP-адреси та функції Lambda). ELB спроможний розподіляти трафік зі змінним навантаженням як у одній зоні доступності, і між кількома зонами доступності. Користувач може вибрати з трьох типів балансувальників, що забезпечують і високу доступність, автомасштабування, і непоганий захист. Все це важливо для забезпечення відмовостійкості програм.

Основні переваги EBL:

- Висока доступність. У угоді про обслуговування йдеться про 99,99% доступності для балансувальника навантаження. Приміром, декілька зон доступності гарантує, що трафік буде оброблено лише справними об'єктами. Власне кажучи, можна балансувати навантаження і по всьому регіону, здійснюючи перенаправлення трафіку на справні цільові об'єкти у різних зонах доступності.
- Безпека. ELB працює з Amazon VPC, надаючи різні можливості забезпечення безпеки - це і інтегроване управління сертифікатами, і автентифікація користувачів, і розшифровка SSL/TLS. Все разом забезпечує централізоване та гнучке управління налаштуваннями TLS.
- Еластичність. ELB спроможний виконувати обробку раптових змін мережного трафіку. А глибока інтеграція з Auto Scaling дає додатку потрібну кількість ресурсів, якщо змінюється навантаження, причому ручне втручання не потрібно.
- Гнучкість. Дозволено використовувати IP-адреси для маршрутизації запитів до цільових об'єктів ваших програм. Це

гарантує гнучкість при віртуалізації цільових програм, даючи таким чином можливість розміщувати відразу кілька програм на одному інстансі. Так як програми можуть використовувати один мережевий порт і мають окремі групи безпеки, спрощується взаємодія між програмами, коли у нас архітектура, наприклад, на основі мікросервісів.

- Моніторинг та аудит. Можна здійснювати моніторинг програм у реал-таймі, використовуючи функції Amazon CloudWatch. Йдеться про метрики, журнали, відстеження запитів. Говорячи простою мовою, ви зможете виявляти проблеми та досить точно визначати вузькі місця продуктивності
- Гібридне балансування навантаження. Можливість балансування навантаження між локальними ресурсами та AWS із застосуванням одного й того ж балансувальника спрощує міграцію або розширення локальних додатків у хмару. Спрощується та обробка відмов із застосуванням хмари.

1.4 Огляд баз данних

База даних відноситься до набору логічно пов'язаної інформації, організованої таким чином, щоб її можна було легко отримати, керувати та оновлювати. Доступ до баз даних здійснюється в електронному вигляді з комп'ютерної системи та зазвичай контролюється системою управління базами даних (СУБД). Адміністратор бази даних (DBA) - це особа, відповідальна за управління базами даних, включаючи безпеку бази даних, контроль доступу, резервне копіювання та аварійне відновлення.

Існує чотири різні типи об'єктів бази даних, які допомагають користувачам компілювати, вводити, зберігати та аналізувати дані в різних форматах:

1. Таблиці
2. Запити
3. Форми

4. Звіти

Розробники повинні використовувати той тип бази даних, який відповідає їхнім вимогам та потребам. Існують різні типи структур баз даних. Ієрархічна база даних: ієрархічна база даних слідує порядку ранжування або батьківсько-дочірнім відносинам для структурування даних. Мережева база даних: мережева база даних схожа на ієрархічну базу даних, але з деякими змінами. Мережева база даних дозволяє дочірньому запису з'єднуватися з різними батьківськими записами, таким чином, забезпечуючи двосторонні стосунки.

Об'єктно-орієнтована база даних: в об'єктно-орієнтованій базі даних інформація зберігається об'єктно-подібним чином. Реляційна база даних: реляційна база даних орієнтована на таблиці, де кожен біт даних пов'язанний з кожним іншим бітом даних. Нереляційна база даних чи база даних NoSQL: база даних без SQL використовує різні формати, такі як документи, графіки, широкі стовпці, що забезпечує велику гнучкість і масштабованість для оформлення бази даних. Бази даних широко поділяються на два основних типи або категорії, а саме: реляційні або послідовні бази даних та нереляційні або непослідовні бази даних або бази даних без SQL. Розробники можуть використовувати їх окремо або разом, залежно від характеру даних та потрібних функцій.

Реляційна база даних – найпоширеніший тип бази даних. Він використовує схему, яка є шаблоном, який використовується для визначення структури даних, що зберігається в базі даних. Наприклад, компанія, що продає продукти своїм клієнтам, повинна мати деяку форму знань про те, куди ці продукти йдуть, кому і в якій кількості. Для кожного підходу можна використовувати різні типи реляційних баз даних. Наприклад, перша таблиця може використовуватися для відображення основної інформації про клієнтів, друга - для кількості проданих продуктів, а третя - для перерахування, хто і де купив цей продукт. Існують ключі, пов'язані з таблицями в реляційній базі даних. Вони надають коротке зведення бази даних або доступ до конкретного рядка або стовпця, які ви хочете перевірити.

Таблиці, що також називаються об'єктами, пов'язані один з одним. У таблиці з інформацією про клієнтів може бути зазначений конкретний ідентифікатор для кожного клієнта, який може позначати все, що потрібно знати про цього клієнта, наприклад, його адресу, ім'я та контактну інформацію. Крім того, у таблиці з описом продукту кожному продукту можна надати певний ідентифікатор. У таблиці, де зберігаються всі замовлення, достатньо буде записати ці ідентифікатори та їх кількість. Будь-яка зміна цих таблицях вплине на всі вони, але передбачуваним і систематичним чином. Реляційні бази даних мають свої переваги і недоліки, які варто врахувати, перш ніж у них вкладатися.

Переваги:

- Реляційні бази даних наслідують строгу схему, а це означає, що кожен новий запис повинен мати різні компоненти, які дозволяють їй вписуватися в цей попередньо сформований шаблон. Це робить дані передбачуваними і легко оцінюються.
- Відповідність ACID є обов'язковою для всіх баз даних РСУБД, що означає, що вони повинні забезпечувати атомарність, узгодженість, ізоляцію та довговічність.
- Вони добре структуровані та значно знижують ймовірність помилок.

Недоліки:

- Ретельний характер, суворі схеми та обмеження реляційних баз даних унеможливлюють зберігання данних в необхідних кількостях на сьогоднішній день.
- Неможливо масштабувати по горизонталі, оскільки реляційні бази даних наслідують певну схему. Хоча вертикальне масштабування здається очевидним вирішенням проблеми, це насправді не так. Вертикальне масштабування має межу, і дані, що збираються через Інтернет щодня, просто надто великі, щоб уявити, що вертикальне масштабування працюватиме довго.

- Обмеження схеми також перешкоджають міграції даних у різні СУБД та з них. Вони мають бути ідентичними; інакше це просто не спрацює.

Інший поширений тип бази даних – нереляційна. Нереляційна форма організації бази даних більш економна за своєю структурою та формою, ніж реляційні бази даних. Замість таблиць зі стовпцями та рядками у них є колекції різних категорій, наприклад, користувачів та замовлення, ілюстровані документами. Таким чином, одна колекція може мати кілька документів. Крім того, вони можуть слідувати або не дотримуватися будь-якого конкретного шаблону або схеми. Документ може мати назву, адресу та продукт у колекції; водночас інший документ може мати лише назву та продукт у тій самій колекції, оскільки для цих документів немає конкретної схеми.

Крім того, різні колекції не обов'язково можуть мати стосунки між собою. До різних типів нереляційних баз даних відносяться. Сховище пар ключ-значення. Цей тип тільки зберігає та надає швидкі та прості знання про пари "ключ-значення". Це простий і легкий спосіб зберігати дані та отримувати доступ до них. Деякі приклади - Amazon DynamoDB та Redis. Сховище з широкою колоною. Цей тип також можна назвати багатовимірним сховищем ключів та значень. Він зберігає та керує величезними обсягами даних у таблицях або кількох стовпцях. Кожен із цих стовпців може діяти як запис, що допомагає масштабувати петабайти даних.

Яскравими прикладами є Scylla, HBase та Cassandra. Сховища документів Тут однакова структура перестає бути необхідністю для записів. Вони можуть мати широкий спектр типів і значень і всі вони можуть бути вкладеними. Дані зберігаються в документах JSON, і ці документи схожі на документи типу "ключ-значення" та "широкий стовпець". Деякі з найвідоміших баз даних NoSQL потрапляють у цю категорію, а саме Couchbase та MongoDB. Пошукові системи. Вони відрізняються від сховищ документів тим, що допомагають зробити дані доступними за допомогою

простого пошуку. Деякі приклади: Solr, Splunk та Exasticsearch. Графічні бази даних. Бази даних графіків показують зв'язок між різними точками даних. Вони застосовуються для аналізу різних типів даних та його взаємозв'язку одне з одним. Вони представлені як мережі пов'язаних об'єктів чи вузлів. Прикладами є Datastax Enterprise Graph та Neo4J. Нереляційні бази даних не є досконалими і мають певні переваги, але також деякі недостатки.

Переваги:

- Їхня властивість яка довзоляє користуватися без схем полегшує керування та зберігання величезних обсягів даних. Їх також можна легко масштабувати по горизонталі.
- Дані не є надто складними і можуть бути розподілені між кількома виділеними вузлами для кращого доступу.

Недоліки:

- Оскільки у них немає певної структури або схеми для даних, що зберігаються, ви не можете покладатися на свої дані для певного поля, тому що воно може не мати його.

1.5 Документо-орієнтованаі бази даних MongoDB та DocumentDB

Для створення нашого додатку було обранно документо-орієнтовану базу данних MongoDB адже вона є гнучкою у використанні, може здійснювати обробку великих данних та використовує оперативну пам’ять для зберігання данних, що робить доступ до них більш швидким. Усі ці переваги покривають наші потреби при розробці нашого програмного забезпечення.

MongoDB – це програма управління базами даних NoSQL з відкритим вихідним кодом. NoSQL використовується як альтернатива традиційним реляційним базам даних. Бази даних NoSQL дуже корисні для роботи з великими наборами розподілених даних. MongoDB – це інструмент, який може керувати документально-орієнтованою інформацією, зберігати чи витягувати інформацію. MongoDB підтримує різноманітні форми даних. Це одна з багатьох технологій нереляційних баз даних, які виникли в середині 2000-х років під прапором NoSQL – зазвичай для використання у додатках з

великими даними та інших завданнях обробки, пов'язаних із даними, які не вписуються у жорстку реляційну модель. Замість використання таблиць та рядків, як у реляційних базах даних, архітектура MongoDB складається з колекцій та документів.

Розробники можуть використовувати Mongo DB для спеціальних запитів, індексації, балансування навантаження, агрегації, виконання серверного JavaScript та інших функцій. MongoDB використовує записи, що складаються з документів, які містять структуру даних, що складається з пар полів та значень. Документи – це основна одиниця даних у MongoDB. Документи схожі на нотацію JavaScript, але використовують варіант під назвою Binary JSON (BSON). Перевага BSON полягає в тому, що він підтримує більшу кількість типів даних. Поля у цих документах аналогічні стовпцям у реляційній базі даних. Відповідно до посібника користувача MongoDB, значення, що містяться, можуть бути різними типами даних, включаючи інші документи, масиви та масиви документів. Документи також включатимуть первинний ключ як унікальний ідентифікатор.

Набори документів називаються колекціями, що функціонують як еквівалент таблиць реляційної бази даних. Колекції можуть містити дані будь-якого типу, але обмеження полягає в тому, що дані в колекції не можуть бути розподілені по різним базам даних.

Оболонка mongo є стандартним компонентом дистрибутивів MongoDB із відкритим вихідним кодом. Після встановлення MongoDB користувачі підключають оболонку mongo до своїх екземплярів MongoDB. Оболонка mongo діє як інтерактивний інтерфейс JavaScript для MongoDB, який дозволяє користувачам запитувати та оновлювати дані, а також виконувати адміністративні операції. Двійкове представлення документів, подібних до JSON, забезпечується форматом зберігання документів та обміну даними BSON. Автоматичне сегментування - ще одна ключова функція, яка дозволяє розподіляти дані в колекції MongoDB за декількома системами для горизонтальної масштабованості в міру збільшення обсягів даних та вимог до

пропускної спроможності.

СУБД NoSQL використовує єдину головну архітектуру для забезпечення узгодженості даних із вторинними базами даних, які підтримують копії первинної бази даних. Операції автоматично реплікуються у ці вторинні бази даних для автоматичного відпрацювання відмови.

Як і інші бази даних NoSQL, MongoDB не потребує визначених схем. У ньому зберігаються дані будь-якого типу. Це дозволяє користувачам створювати будь-яку кількість полів у документі, спрощуючи масштабування баз даних MongoDB в порівнянні з реляційними базами даних. Однією з переваг використання документів і те, що ці об'єкти зіставляються з власними типами даних у ряді мов програмування. Крім того, наявність вбудованих документів знижує потребу в об'єднанні баз даних, що може знизити витрати.

Основна функція MongoDB - горизонтальна масштабованість, що робить її корисною базою даних для компаній, що запускають програми для роботи з великими даними. Крім того, сегментування дозволяє базі даних розподіляти дані по кластеру машин. Нові версії MongoDB також підтримують створення зон даних на основі ключа уламка. MongoDB підтримує ряд механізмів зберігання та надає API-інтерфейси механізмів зберігання, що дозволяють третім сторонам розробляти власні механізми зберігання для MongoDB.

СУБД також має вбудовані можливості агрегування, які дозволяють користувачам запускати MapReduce код безпосередньо в базі даних, а не запускати MapReduce на Hadoop. MongoDB також включає власну файлову систему GridFS, схожу на розподілену файлову систему Hadoop (HDFS). Файлова система використовується в першу чергу для зберігання файлів, розмір яких перевищує обмеження BSON 16 МБ на документ. Ці подібності дозволяють використовувати MongoDB замість Hadoop, хоча програмне забезпечення бази даних інтегрується з Hadoop, Spark та іншими середовищами обробки даних. Хоча MongoDB має деякі цінні переваги, є й недоліки.

Завдяки стратегії автоматичного перемикання при відмові користувач настроює лише один головний вузол у кластері MongoDB. Якщо майстер виходить з ладу, інший вузол автоматично перетворюється на нового майстра. Цей перемикач обіцяє безперервність, але не миттєво це може зайняти до хвилини. Для порівняння, база даних Cassandra NoSQL підтримує кілька головних вузлів, тому якщо один майстер виходить з ладу, інший залишається в резерві для високодоступної інфраструктури бази даних. Єдиний головний вузол MongoDB також обмежує швидкість запису даних у базі даних. Запис даних має бути записаний на головному вузлі, а запис нової інформації до бази даних обмежена ємністю цього головного вузла. Інша потенційна проблема полягає в тому, що MongoDB не забезпечує повну цілісність посилань за рахунок використання обмежень зовнішнього ключа, що може вплинути на узгодженість даних.

Крім того, автентифікація користувачів за замовчуванням не включена до баз даних MongoDB, що свідчить про популярність технології серед розробників. Однак зловмисні хакери націлилися на велику кількість незахищених систем MongoDB в атаках з викупом, що призвело до додавання параметра за замовчуванням, який блокує підключення до баз даних, якщо вони не були налаштовані адміністратором бази даних.

Amazon DocumentDB (з сумісністю з MongoDB) – це швидка, масштабована, високодоступна та повністю керована служба бази даних документів, яка підтримує робочі навантаження MongoDB. В якості бази даних документів Amazon DocumentDB дозволяє легко зберігати, запитувати та індексувати дані JSON. Amazon DocumentDB розроблений з нуля, щоб забезпечити продуктивність, масштабованість та доступність, необхідні при масштабній роботі з критично важливими робочими навантаженнями MongoDB. У Amazon DocumentDB сховище та обчислювальні ресурси розділені, що дозволяє масштабувати кожен незалежно один від одного, і ви можете збільшити ємність читання до мільйонів запитів за секунду, додавши

до 15 реплік читання з малою затримкою за лічені хвилини, незалежно від розміру ваших даних. Amazon

DocumentDB розрахований на доступність 99,99% та реплікує шість копій ваших даних у трьох зонах доступності (AZ). Розробники можуть використовувати Amazon Database Migration Service щоб легко перенести свої локальні бази даних або бази даних MongoDB з Amazon Elastic Compute Cloud (EC2) до Amazon DocumentDB практично без простоїв.

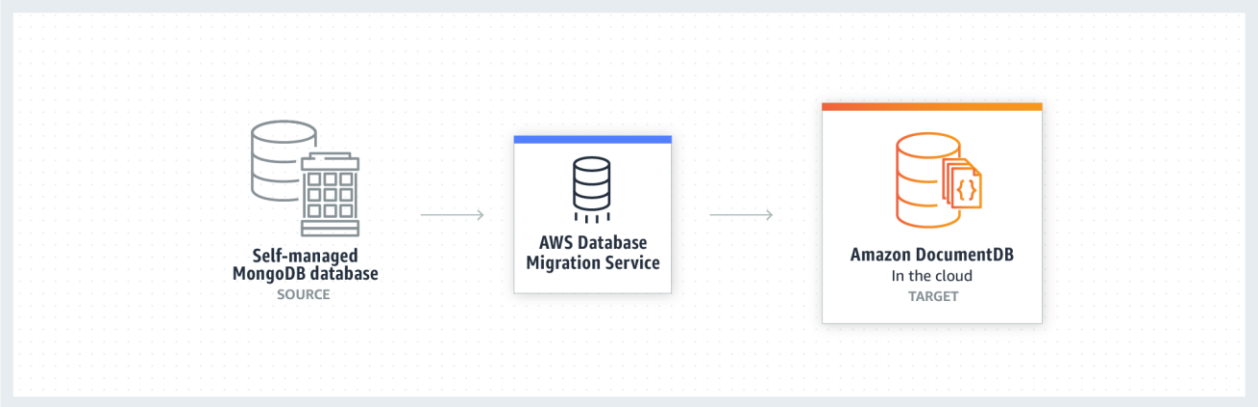


Рис. 1.4. З’єднання MongoDB з DocumentDB

Amazon DocumentDB спочатку створювався з використанням хмарної архітектури бази даних. Його унікальна архітектура поділяє сховище та обчислювальні ресурси, тому кожен рівень може масштабуватись незалежно. Amazon DocumentDB використовує спеціально створену розподілену відмовостійку систему зберігання із самовідновленням, яка забезпечує високу доступність та надійність за рахунок шести способів реплікації даних у трьох зонах доступності AWS (AZ).. На наступних схемах показано поділ обчислень та сховищ в архітектурі Amazon DocumentDB та шість способів реплікації даних у трьох зонах доступності.

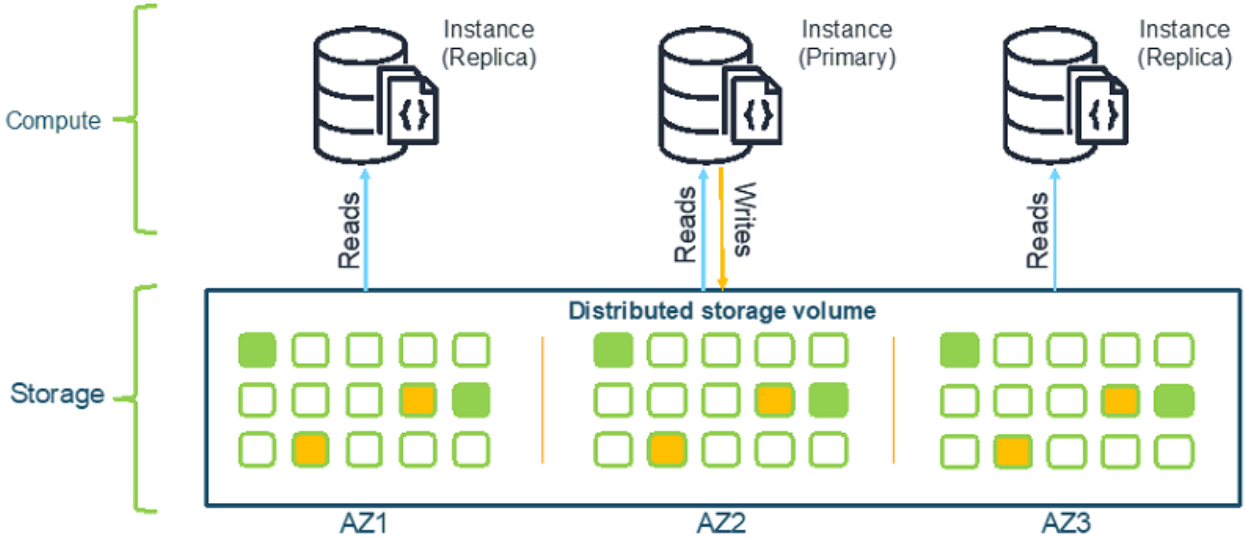


Рис. 1.5. Архітектура DocumentDB

Оскільки том сховища відокремлений від обчислювальних примірників, обчислювальні примірники не покладаються на приєднане сховище, унікальне для примірника. Кожен екземпляр у кластері включає том розподіленого сховища; тому при додаванні нових екземплярів копіювання даних не потрібне. Це вигідно для вас, тому що ви можете додати додатковий екземпляр репліки у свій кластер або масштабувати екземпляри за лічені хвилини, щоб збільшити пропускну здатність до мільйонів операцій читання за секунду, незалежно від розміру даних. Так само ви можете зменшувати і масштабувати так само легко, не впливаючи на продуктивність інших екземплярів.

У Amazon DocumentDB рівень зберігання відповідає за реплікацію та надійність даних. На відміну від традиційних архітектур баз даних, екземпляри реплік в Amazon DocumentDB не містять даних і не беруть участь у протоколі реплікації для досягнення кворуму для забезпечення надійності. В результаті ви можете масштабувати операції читання на екземплярах реплік, щоб підвищити продуктивність обчислювальних ресурсів. У Amazon DocumentDB всі оператори CRUD (findAndModify, update, insert, delete) гарантують атомарність і узгодженість, навіть для операцій, які модифікують кілька документів.

1.6 Інструмент підготовки інфраструктури сервера Terraform

Один з найкращих сторонніх ресурсів для роботи з AWS є Terraform, адже він дуже простий у використанні та надає можливість повністю контролювати усі процеси написання нашого серверу, що оптимізує та прискорює нашу роботу.

Terraform - це інфраструктура з відкритим вихідним кодом, як інструмент коду, розроблений HashiCorp. Він використовується для визначення та надання повної інфраструктури за допомогою простої в використанні декларативної мови. Це інструмент підготовки інфраструктури, в якому можна зберегти налаштування хмарної інфраструктури у вигляді кодів. Він дуже схожий на такі інструменти, як CloudFormation, які розробники використовували для автоматизації інфраструктури AWS, але його можна використовувати тільки на AWS. З Terraform ви можете використовувати його на інших хмарних платформах. Нижче наведено деякі з переваг використання Terraform.

- Забезпечує оркестрування, а не лише керування конфігурацією
- Підтримує кількох постачальників, таких як AWS, Azure, GCP, DigitalOcean та багато інших.
- Підтримує незмінну інфраструктуру, в якій конфігурація змінюється плавно
- Використовує просту для розуміння мову, HCL (мова конфігурації HashiCorp)
- Інфраструктуру можна легко перемістити на іншного провайдера
- Підтримує клієнтську архітектуру, тому немає необхідності додаткового управління конфігурацією на сервері

Нижче наведено основні концепції, що використовуються в Terraform:

- Змінні: також використовується як вхідні змінні, це пара ключ-значення, використовувана модулями Terraform для налаштування.
- Провайдер: це плагін для взаємодії з API служби та доступу до пов'язаних із нею ресурсів.

- Модуль: це папка з шаблонами Terraform, де визначено всі конфігурації.
- Етап: складається з кешованої інформації про інфраструктуру, керовану Terraform, та пов'язаних з нею конфігураціях.
- Ресурси: це відноситься до блоку з одного або декількох об'єктів інфраструктури (обчислювальні екземпляри, віртуальні мережі тощо), які використовуються при налаштуванні та управлінні інфраструктурою.
- Джерело даних: реалізовано провайдерами для повернення інформації про зовнішні об'єкти в терраформування.
- Вихідні значення: це значення модуля terraform, що повертаються, які можуть використовуватися іншими конфігураціями.
- План: це один із етапів, на якому визначається, що необхідно створити, оновити або знищити, щоб перейти від реального/поточного стану інфраструктури до бажаного стану.
- Використання: це один із етапів, на якому він застосовує зміни реального/поточного стану інфраструктури, щоб перейти до бажаного стану.

Життєвий цикл Terraform складається з ініціалізації, планування, застосування та знищення.

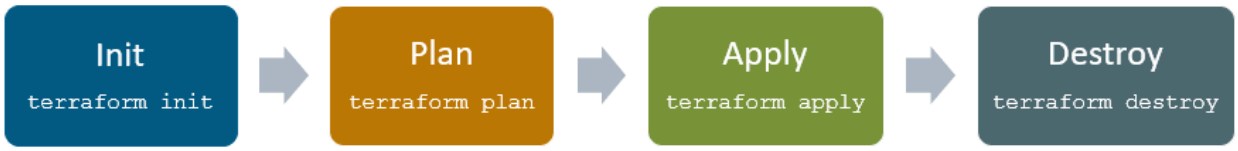


Рис. 1.6. Життєвий цикл Terraform

Terraform init ініціалізує робочий каталог, який складається з усіх конфігураційних файлів. План Terraform використовується для створення плану виконання для досягнення бажаного стану інфраструктури. Зміни файлів конфігурації виконуються для досягнення бажаного стану. Потім Terraform apply вносить зміни до інфраструктури, як визначено в плані, і

інфраструктура входить у бажаний стан. Terraform destroy використовується для видалення всіх старих інфраструктурних ресурсів, які позначаються як застарілі після фази застосування.

Terraform складається з двох основних компонентів, що становлять його архітектуру:

- Терраформ Ядро
- Провайдери

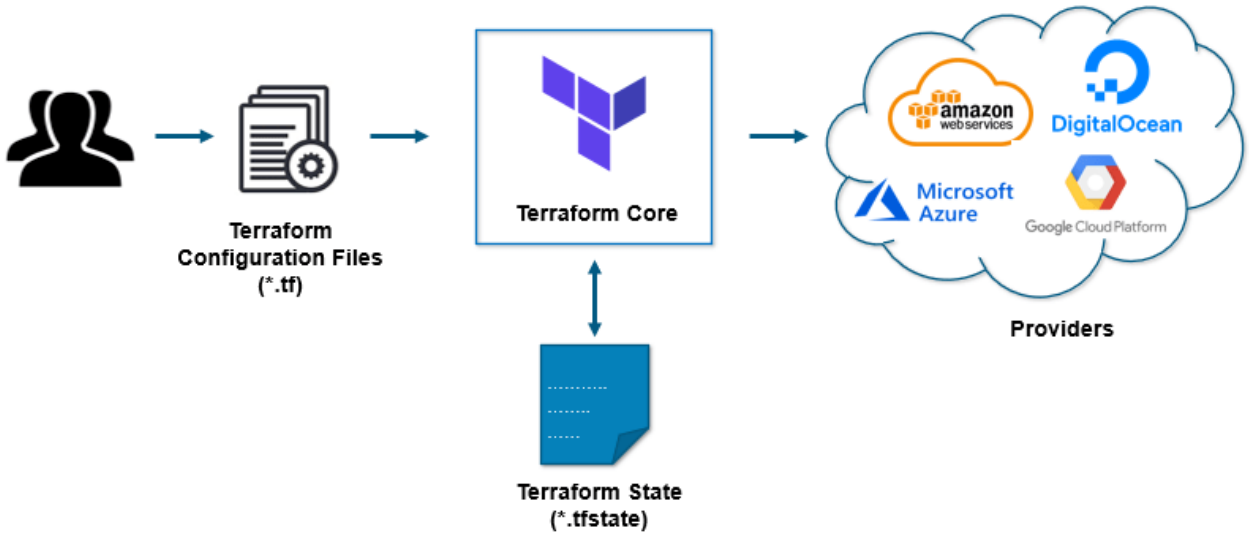


Рис. 1.7. Архітектура Terraform

Ядро Terraform використовує два джерела введення для виконання своєї роботи. Перше джерело введення – це конфігурація Terraform, яку розробники, як користувач, настроюєте. Тут визначається, що потрібно створити чи підготувати. І друге джерело вхідних даних - це стан, в якому terraform зберігає актуальний стан того, як виглядає поточне налаштування інфраструктури.

Отже, ядро terraform приймає вхідні дані та визначає план того, що необхідно зробити. Він порівнює поточний стан та бажану конфігурацію в кінцевому результаті. Ядро визначає, що потрібно зробити, щоб досягти бажаного стану у конфігураційному файлі. Він показує, що потрібно

створити, що потрібно оновити, що потрібно видалити, щоб створити та підготувати інфраструктуру.

Другий компонент архітектури – постачальники конкретних технологій. Це можуть бути хмарні провайдери, такі як AWS, Azure, GCP або інша інфраструктура як сервісна платформа. Він також є постачальником більш високорівневих компонентів, таких як Kubernetes або інших інструментів платформи, навіть деяке програмне забезпечення як інструмент самообслуговування.Це дає можливість створювати інфраструктуру на різних рівнях. Наприклад, створити інфраструктуру AWS, потім розгорнути Kubernetes поверх неї, а потім створити сервіси/компоненти всередині цього кластера Kubernetes.

Terraform має більше сотні постачальників для різних технологій, і кожен постачальник потім надає користувачеві terraform доступ до своїх ресурсів. Так, наприклад, через постачальника AWS розробники мають доступ до сотень ресурсів AWS, таких як екземпляри EC2, користувачі AWS. З постачальником Kubernetes розробники отримують доступ до ресурсів, таких як розгортання.

1.7 Висновки до першого розділу

Основна концепція технічної сторони проєкту – використовувати новітніші хмарні технології, що дозволять задіювати великі обичлювальні потужності, підтримувати мобільність да доступність з будь-якої точки планети. Завдяки новітнім технологіям ми можемо контролювати навантаження на наш сервер та розподіляти його за потреби. За допомогою сервісів які ми маємо у своє розпорядження ми можемо в режимі реального часу аналізувати потік данних на наший ресур та відстужвати усі змінни які відбуваються на ньому. Ми можемо спостерігати за кількістю запитів чи помилок які надходяь на сервер та швидко реагувати для аналізу та покращення ефективності нашого ресурсу.

Розділ 2 АЛГОРИТМІЧНЕ ЗАБЕЗПЕЧЕННЯ ТА ВЗАЄМОДІЯ КОМПОНЕНТІВ СЕРВЕРА

2.1 Загальна концепція компонентів програмного забезпечення

Контейнеризація додатків – один із головних трендів сучасних IT-розробок. Проте, у контейнерів є один істотний недолік для масового споживача — складне настроювання масштабування. Рішенням стали автоматичні системи управління контейнеризацією, найпопулярнішою з яких є Kubernetes.

Kubernetes - це інструмент оркестровки з відкритим вихідним кодом, розроблений Google для управління мікросервісами або контейнерними додатками в розподільному кластері узлов. Kubernetes забезпечує високостійку інфраструктуру з можливістю розгортання з нульовим часом простого, автоматичним відкатом, масштабуванням і самовосстановленням контейнерів (що включає автоматичне розміщення, автоматичний перезапуск, автоматичну реплікацію та масштабування контейнерів у залежності від використання ЦП). Основна ціль Kubernetes - скрити складність управління парком контейнерів, наданий REST API для необхідних функцій. Kubernetes є портативним за своєю природою, він може працювати на різних публічних або приватних хмарних платформах, таких як AWS, Azure, OpenStack або Apache Mesos. Він також може працювати на голих машинах. Kubernetes наслідує архітектуру клієнт-сервер. Можливе встановлення з кількома головними серверами (для забезпечення високої доступності), але за замовчуванням є один головний сервер, який діє як керуючий вузол та точка контакту.

Головний сервер складається з різних компонентів, включаючи kube-apiserver, сховище etcd, kube-controller-manager, хмарний-контролер-менеджер, kube-scheduler та DNS-сервер для сервісів Kubernetes. Компоненти вузла включають kubelet та kube-proxy поверх Docker.

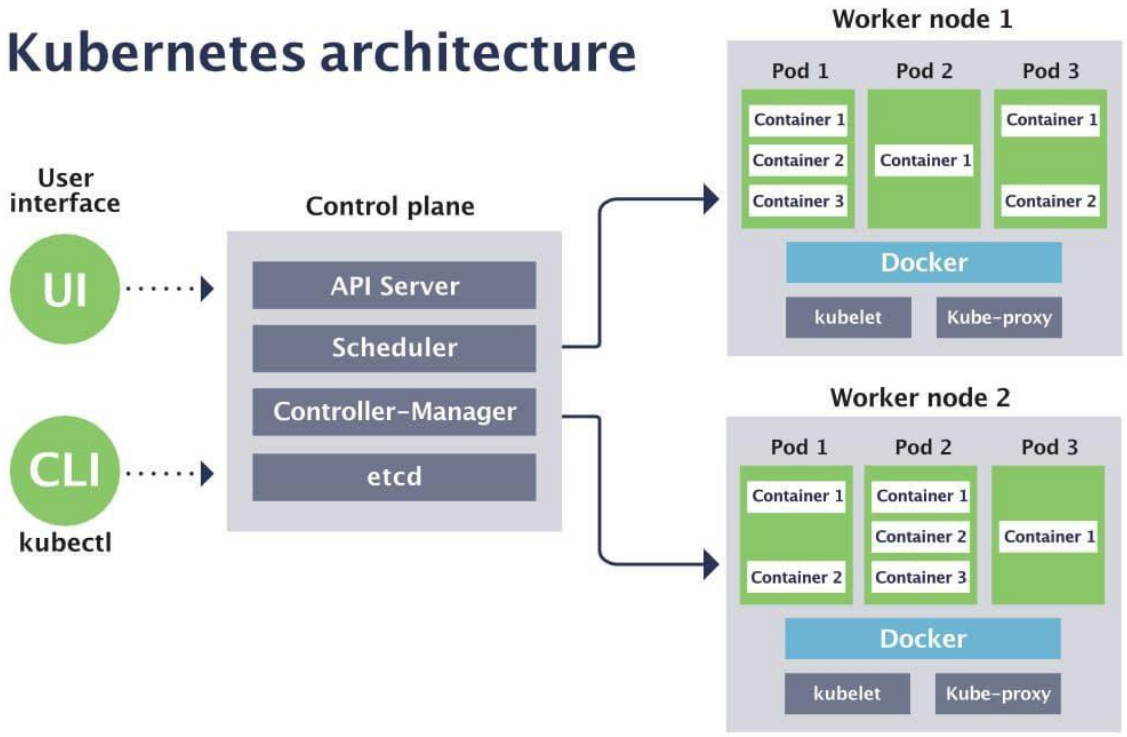


Рис. 2.1. Архітектура Kubernetec

Нижче наведено основні компоненти, виявлені на головному вузлі:

- etcd cluster - просте розподілене сховище значень ключів, яке використовується для зберігання даних кластера Kubernetes (таких як кількість модулів, їх стан, простір імен тощо), об'єктів API та відомостей про виявлення служб. Він доступний лише з сервера API з міркувань безпеки. etcd включає сповіщення кластера про зміни конфігурації за допомогою спостерігачів. Повідомлення – це запити API на кожному вузлі кластера etcd для запуску оновлення інформації у сховищі вузла.
- kube-apiserver – сервер API Kubernetes – це центральний керуючий об'єкт, який отримує всі запити REST для модифікацій (для модулів, служб, наборів/контролерів реплікації тощо), виступаючи як зовнішній інтерфейс для кластера. Крім того, це єдиний компонент,

який взаємодіє із кластером etcd, перевіряючи, що дані зберігаються в etcd та відповідають деталям служби розгорнутих модулів.

- kube-controller-manager - запускає ряд окремих процесів контролера у фоновому режимі (наприклад, контролер реплікації контролює кількість реплік у модулі, контролер кінцевих точок заповнює об'єкти кінцевих точок, такі як служби та модулі, та інші) для регулювання загального стану модуля кластер та виконувати рутинні завдання. Коли змінюється конфігурація служби (наприклад, заміна образу, з якого запущені модулі, або зміна параметрів у файлі конфігурації yaml), контролер виявляє цю зміну і починає працювати в напрямку нового бажаного стану.
- cloud-controller-manager - відповідає за управління процесами контролера із залежностями від основного хмарного провайдера (якщо застосовується). Наприклад, коли контролеру необхідно перевірити, чи завершено вузол, або налаштувати маршрути, балансувальники навантаження або томи у хмарній інфраструктурі, це обробляється диспетчером хмарного контролера.
- kube-scheduler - допомагає планувати поди (спільно розташовану групу контейнерів, усередині яких працюють процеси нашої програми) на різних вузлах залежно від використання ресурсів. Він зчитує експлуатаційні вимоги служби та планує їх на найбільш підходящому вузлі. Наприклад, якщо програма потребує 1 ГБ пам'яті і 2 ядра ЦП, то модулі для цієї програми будуть заплановані на вузлі, принаймні, з цими ресурсами. Планувальник запускається щоразу, коли виникає потреба запланувати поди. Планувальник повинен знати спільні доступні ресурси, а також ресурси, виділені для робочих навантажень на кожному вузлі.

Нижче наведено основні компоненти, виявлені на (робочому) вузлі:

- kubelet - основна служба на вузлі, що регулярно приймає нові або змінені специфікації модулів (в основному через kube-apiserver) та

забезпечує працездатність модулів та їх контейнерів у бажаному стані. Цей компонент також повідомляє майстра про працездатність хоста, на якому він працює.

- kube-proxy - проксі-сервіс, який запускається на кожному робочому вузлі для роботи з окремими підмережами хоста та надання послуг зовнішньому світу. Він виконує пересилання запитів у потрібні модулі/контейнери у різних ізольованих мережах кластера.

Команда kubectl – це лінійний інструмент, який взаємодіє з kube-apiserver та відправляє команди на головний вузол. Кожна команда перетворюється на виклик API. Використання Kubernetes вимагає розуміння різних абстракцій, які він використовує для представлення стану системи, таких як служби, модулі, томи, простору імен та розгортання.

- Pod - зазвичай відноситься до одного або кількох контейнерів, які повинні керуватися як одна програма. Pod інкапсулює контейнери додатків, ресурси зберігання, унікальний мережевий ідентифікатор та іншу конфігурацію того, як запускати контейнери.
- Сервісні модулі - є непостійними, тобто Kubernetes не гарантує, що цей фізичний модуль залишатиметься активним (наприклад, контролер реплікації може вбити та запустити новий набір модулів). Натомість служба є логічним набором модулів і діє як шлюз, дозволяючи (клієнтським) модулям відправляти запити до служби без необхідності відстежувати, які фізичні модулі фактично становлять службу.
- Том - схожий на том у контейнері в Docker, але том Kubernetes застосовується до всього модуля і монтується на всіх контейнерах модуля. Kubernetes гарантує збереження даних під час перезапуску контейнера. Том буде видалено лише тоді, коли капсула буде знищена. Крім того, з модулем може бути пов'язано кілька томів (можливо різних типів).

- Простір імен - віртуальний кластер (один фізичний кластер може запускати кілька віртуальних), призначений для середовищ із великою кількістю користувачів, розподілених по декількома командам чи проєктами, для ізоляції проблем. Ресурси всередині простору імен повинні бути унікальними та не мати доступу до ресурсів в іншому просторі імен. Крім того, простору імен може бути виділена квота ресурсів, щоб не споживати більше, ніж його частка у загальних ресурсах фізичного кластера
- Розгортання - визначає бажаний стан модуля або набору реплік у файлі yaml. Потім контролер розгортання поступово оновлює середовище (наприклад, створюючи або видаляючи репліки), доки поточний стан не відповідатиме бажаному стану, вказаному у файлі розгортання. Наприклад, якщо файл yaml визначає 2 репліки для модуля, але зараз працює тільки одна, буде створено додаткову. Зверніть увагу, що репліки, що керуються через розгортання, не повинні керуватися безпосередньо, лише через нові розгортання.

Kubernetes був розроблений для підтримки функцій, необхідних для високодоступних розподілених систем, таких як (автоматичне) масштабування, висока доступність, безпека та переносимість.

Масштабованість – Kubernetes забезпечує горизонтальне масштабування подів на основі завантаження ЦП. Поріг використання ЦП можна налаштувати, і Kubernetes автоматично запустить нові поди, якщо буде досягнутий поріг. Наприклад, якщо граничне значення для ЦП становить 70%, але додаток фактично зростає до 220%, то зрештою буде розгорнуто ще 3 модулі, так що середнє завантаження ЦП знову стане нижче 70%. Коли є кілька модулів для конкретної програми, Kubernetes забезпечує можливість балансування навантаження між ними.

Kubernetes також підтримує горизонтальне масштабування модулів з відстеження стану, включаючи бази даних NoSQL і СУБД за допомогою наборів з відстеження стану. Набір з відстеження стану аналогічний концепції

розгортання, але забезпечує сталість і стабільність сховища навіть при видаленні модуля.

Висока доступність - Kubernetes забезпечує високу доступність як на рівні програм, так і на рівні інфраструктури. Набори реплік забезпечують виконання бажаної (мінімальної) кількості реплік модуля без збереження стану для цієї програми. Набори з відстеження стану виконують ту ж роль для модулів з відстеження стану. На рівні інфраструктури Kubernetes підтримує різні серверні частини розподіленого сховища, такі як AWS EBS, Azure Disk, Google Persistent Disk, NFS та інші. Додавання надійного та доступного рівня зберігання в Kubernetes забезпечує високу доступність робочих навантажень із відстеженням стану. Крім того, кожен із головних компонентів може бути налаштований для багатовузлової реплікації (multi-master), щоб забезпечити більш високу доступність.

Безпека - Kubernetes забезпечує безпеку на кількох рівнях: кластер, додаток та мережу. Кінцеві точки API захищені за допомогою безпеки транспортного рівня (TLS). Тільки авторизовані користувачі (облікові записи служб або звичайні користувачі) можуть виконувати операції у кластері (через запити API). На рівні програми секрети Kubernetes можуть зберігати конфіденційну інформацію (наприклад, паролі або токени) для кожного кластера (віртуальний кластер при використанні просторів імен, інакше фізичний). Зауважте, що секрети доступні з будь-якого модуля в тому ж кластері. Мережеві політики для доступу до модулів можна визначити у розгортанні. Мережева політика визначає, як подам дозволено взаємодіяти один з одним та іншими кінцевими точками мережі.

. Перенесення у Kubernetes проявляється у виборі операційної системи (кластер може працювати в будь-якому основному дистрибутиві Linux), архітектури процесора (віртуальні машини або «голого заліза»), хмарних провайдерів (AWS, Azure або Google Cloud Platform) і нових середовищ виконання контейнерів. , крім Docker, також можуть бути додані.

2.2 Модуль Kubernetes API

Kubernetes API – це програмний інтерфейс на основі ресурсів (RESTful), що надається через HTTP. Він підтримує отримання, створення, оновлення та видалення первинних ресурсів за допомогою стандартних HTTP-команд (POST, PUT, PATCH, DELETE, GET). Для деяких ресурсів API включає додаткові ресурси, які дозволяють виконувати детальну авторизацію (наприклад, відокремити деталі перегляду модуля від отримання його журналів) і можуть приймати та обслуговувати ці ресурси в різних уявленнях для зручності чи ефективності. Kubernetes підтримує ефективні сповіщення про зміни ресурсів за допомогою годинника. Kubernetes також забезпечує узгоджені операції зі списком, щоб клієнти API могли ефективно кешувати, відстежувати та синхронізувати стан ресурсів. Користувачі отримують доступ до Kubernetes API за допомогою kubectl, клієнтських бібліотек або за допомогою запитів REST. Для доступу до API можуть бути авторизовані як користувачі, так і сервісні акаунти Kubernetes. Коли запит досягає API, він проходить кілька етапів, показаних на наступній діаграмі:

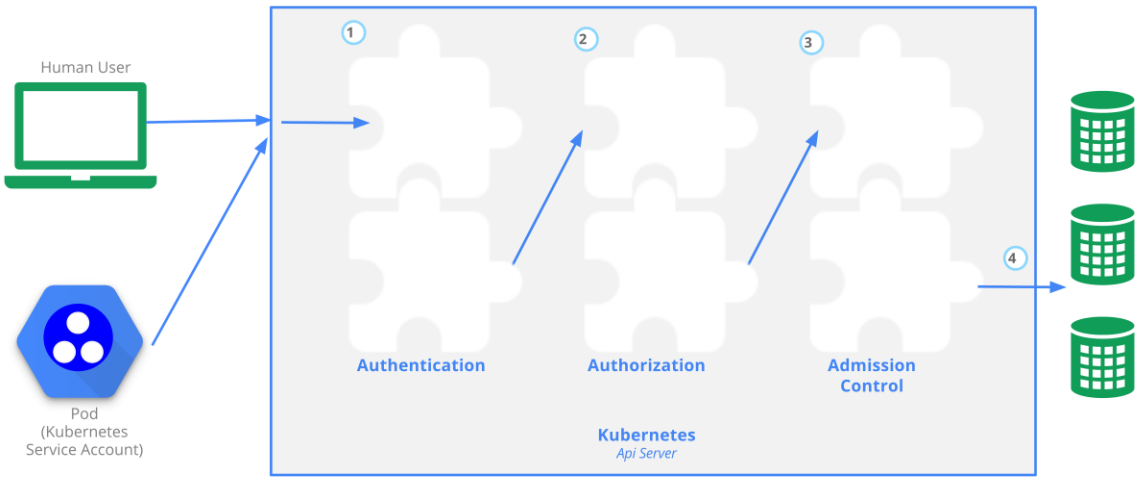


Рис. 2.2. Архітектура Kub-API

У типовому кластері Kubernetes API обслуговує порт 443 захищений TLS. Сервер API представляє сертифікат. Цей сертифікат може бути

підписаний з використанням приватного центру сертифікації (ЦС) або на основі інфраструктури відкритого ключа, пов'язаної із загальновизнаним ЦС. Якщо ваш кластер використовує приватний центр сертифікації, вам знадобиться копія цього сертифіката CA, налаштована у вашому ~/.kube/config на клієнті, щоб ви могли довіряти з'єднанню та бути впевненим, що воно не було перехоплено. На цьому етапі ваш клієнт може подати сертифікат клієнта TLS. Після того, як TLS встановленне, HTTP-запит переходить на етап аутентифікації. Це показано як крок 1 на схемі. Сценарій створення кластера або адміністратор кластера налаштовує API сервера для запуску одного або кількох модулів Authenticator.

Вхідними даними на етапі аутентифікації є весь HTTP-запит; однак зазвичай він перевіряє заголовки та / або сертифікат клієнта. Модулі аутентифікації включають в себе клієнтські сертифікати, пароль і прості токени, токени початкового завантаження та веб-токени JSON (використовуються для облікових записів служби). Можна вказати кілька модулів аутентифікації, і в цьому випадку кожен з них буде перевірятися послідовно, поки один з них не буде успішним. Якщо запит не може бути аутентифікований, він відхиляється з кодом стану HTTP 401. У іншому випадку користувач аутентифікується як конкретне ім’я користувача, а ім’я користувача стає доступним для подальших кроків використання в їх рішеннях. Деякі аутентифікатори також забезпечують членство користувача в групах.

Хоча Kubernetes використовує імена користувачів для прийняття рішень для керування доступом та введення журналів запитів, він не має об’єкта користувача та не зберігає імена користувачів або інформацію про користувачів у вашому API. Після підтвердження автентичності запиту, що надходить від певного користувача, запит має бути авторизований. Це показано як крок 2 на схемі. Запит має містити ім’я користувача запитувача, запитувану дію та об’єкт, на який ця дія впливає. Запит є авторизованим, якщо наявна політика оголошує, що користувач має дозволи на виконання

запитаної дії. Наприклад, якщо Боб має наведену нижче політику, то він може читати модулі лише в просторі імен projectCaribou:

```
{
    "apiVersion": "abac.authorization.kubernetes.io/v1beta1",
    "kind": "Policy",
    "spec": {
        "user": "bob",
        "namespace": "projectCaribou",
        "resource": "pods",
        "readonly": true
    }
}
```

Якщо Боб робить такий запит, запит авторизується, оскільки йому дозволено читати об’єкти в просторі імен projectCaribou:

```
{
  "apiVersion": "authorization.k8s.io/v1beta1",
  "kind": "SubjectAccessReview",
  "spec": {
    "resourceAttributes": {
      "namespace": "projectCaribou",
      "verb": "get",
      "group": "unicorn.example.org",
      "resource": "pods"
    }
  }
}
```

Якщо Боб робить запит на запис (створення або оновлення) об'єктів у просторі імен projectCaribou, його авторизація відхиляється. Якщо Боб робить запит на читання (отримання) об'єктів в іншому просторі імен, такому як projectFish, його авторизація відхиляється. Авторизація Kubernetes вимагає, щоб ви використовували спільні атрибути REST для взаємодії з існуючими системами контролю доступу в масштабах усієї організації або хмарного провайдера. Важливо використовувати форматування REST, тому що ці системи управління можуть взаємодіяти з іншими API, крім Kubernetes API.

Kubernetes підтримує кілька модулів авторизації, таких як режим ABAC, режим RBAC та Webhook. Коли адміністратор створює кластер, він настроює модулі авторизації, які мають використовуватися на сервері API. Якщо налаштовано більше одного модуля авторизації, Kubernetes перевіряє кожен модуль, і якщо будь-який модуль авторизує запит, запит може бути продовжено. Якщо всі модулі відхиляють запит, запит відхиляється (код стану HTTP 403). Попереднє обговорення застосовується до запитів,

надісланих на захищений порт сервера API (типовий випадок). Сервер API може фактично обслуговувати 2 порти.За промовчанням сервер Kubernetes API обслуговує HTTP на 2 портах.

Порт localhost:

- призначений для тестування та початкового завантаження, а також для інших компонентів головного вузла (планувальника, диспетчера-контролера) для взаємодії з API
- ні TLS
- за замовчуванням порт 8080
- IP за промовчанням - localhost, змініть його за допомогою прапорця --insecure-bind-address.
- запит обходить модулі автентифікації та авторизації.
- запит обробляється модулями контролю допуску.
- захищений необхідністю мати доступ до хосту

Захищений порт:

- використовується по можливості
- використовує TLS. Встановлює сертифікат за допомогою --tls-cert-file та ключ із прапором --tls-private-key-file.
- за замовчуванням порт 6443, змінити за допомогою прапорця --secure-port.
- IP за замовчуванням - це перший мережний інтерфейс, відмінний від localhost, змінити його можна за допомогою прапорця --bind-address.
- запит обробляється модулями автентифікації та авторизації.
- запит обробляється модулями контролю допуску.
- модулі аутентифікації та авторизації запускаються.

2.3 Графічний інтерфейс Kubernetes Lens

Kubernetes Lens – ефективне середовище IDE з відкритим вихідним кодом для Kubernetes. Lens спрощує роботу з Kubernetes, допомагаючи керувати кластерами та контролювати їх у режимі реального часу. Він був

розроблений Kontena, Inc., а потім придбаний Mirantis у 2020 році, який потім відкрив його і зробив доступним для безкоштовного завантаження. Lens - це окрема програма, яку можна встановити в macOS, Windows та деяких версіях Linux. З Kubernetes Lens розробники можуть отримати доступ до будь-якого кластера Kubernetes у будь-якому місці. Kubernetes Lens націлений на розробників, SRE та розробників програмного забезпечення в цілому. Швидше за все це єдина платформа, яка вам знадобиться для управління кластерною системою в Kubernetes. Його підтримують ряд компаній та хмарних екосистем, таких як Apple, Rakuten, Zendesk, Adobe, Google та інші. Є багато функцій, які роблять Kubernetes Lens дуже привабливим інструментом. Ось огляд деяких із них.

Управління кластерами в Kubernetes може бути важким, але з Kubernetes Lens розробники можуть працювати з кількома кластерами, зберігаючи контекст з кожним із них. Lens дозволяє налаштовувати, змінювати та перенаправляти кластери одним клацанням миші, організовуючи та розкриваючи всю робочу систему в кластері, надаючи метрики . Маючи цю інформацію, розробники можуть легко та дуже швидко редагувати зміни та застосовувати їх. Додати кластер Kubernetes у Lens дуже просто. Все, що вам потрібно зробити, це вказати локальний / онлайн файл kubeconfig на Lens, і він автоматично виявить його і підключиться до нього.

За допомогою Lens можна перевіряти всі ресурси, що працюють всередині вашого кластера, від простих модулів і розгортань до типів, що налаштовуються вашими додатками. Вбудована візуалізація та метрики. Kubernetes Lens поставляється з вбудованою настройкою Prometheus, яка має розраховану на багато користувачів функцію, яка забезпечує управління доступом на основі ролей (RBAC) для кожного користувача. Це означає, що в кластері користувачі можуть отримувати доступ тільки до візуалізацій, до яких вони мають дозвіл.

З Prometheus розробникі отримують доступ до графіків в реальному часі, діаграм використання ресурсів і метриків використання, таких як ЦП,

пам'ять, мережа, запити і т.д., які інтегровані в панель управління Lens. Ці графіки та показники відображаються у контексті конкретного кластера, який проглядається в даний момент, у режимі реального часу.

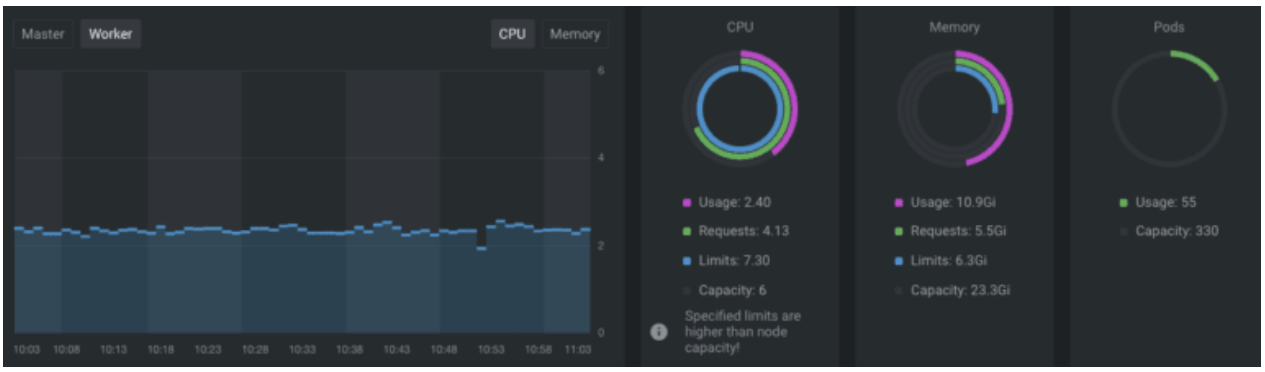


Рис. 2.3. Відображення метриків сервера

Kubernetes Lens також інтегрується з Helm, що спрощує встановлення та управління діаграмами та випусками Helm у Kubernetes. Kubernetes Lens дозволяє використовувати доступні репозиторії Helm з Artifact Hub і автоматично додає репозиторій, якщо інші репозиторії ще не налаштовані. Якщо вам потрібно додати інші репозиторії, їх можна додати вручну через командний рядок. Зверніть увагу, що настроєні репозиторії Helm глобально додаються до комп'ютера користувача, тому інші процеси також можуть їх бачити. Усі графіки з налаштованих репозиторіїв Helm будуть перераховані в розділі "Програми".

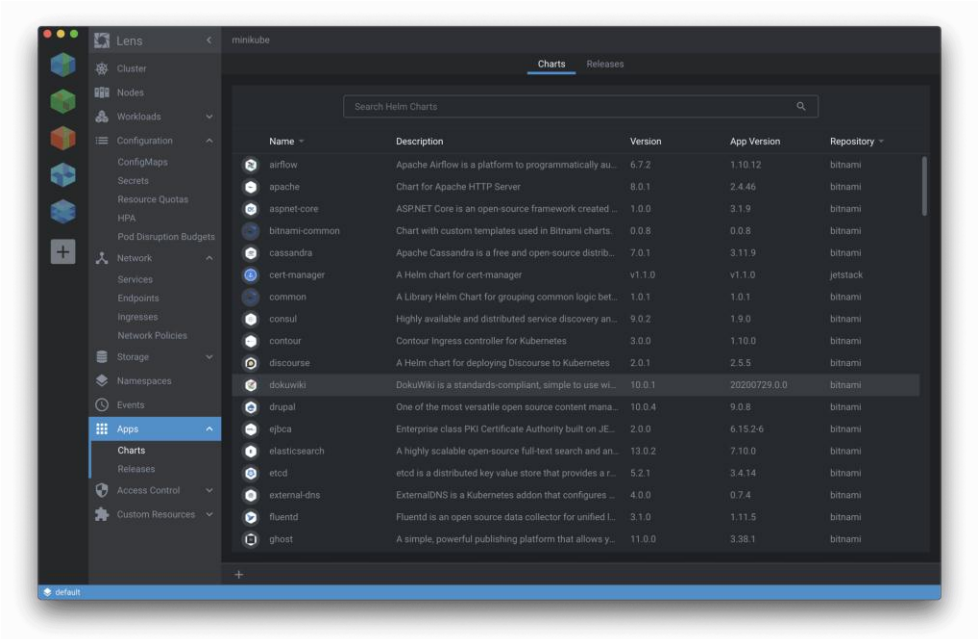


Рис. 2.4. Helm Charts

Kubernetes Lens також дозволяє створювати власні розширення за допомогою API-інтерфейсів Lens. Вони підтримують додавання нових деталей об'єкта, створення настроюваних сторінок, додавання елементів рядка стану та інші модифікації інтерфейсу користувача. Розширення можуть бути опубліковані в npm для створення посилання на архів, який може посилатися екран установки Kubernetes Lens.

Lens надає спосіб керування Kubernetes через графічний інтерфейс, оскільки керування кількома кластерами на різних платформах означає розшифровку інших складних елементів, режимів та методів множинного доступу для організації кластерів, компонентів, вузлів та інфраструктури. Вирішити все це з командного рядка складно, повільно. Це пов'язано насамперед із постійним збільшенням кількості кластерів та додатків, не кажучи вже про їх конфігурації та вимоги.

За допомогою графічного інтерфейсу Kubernetes Lens можна робити кілька речей:

1. Розробники можуть додавати кластери вручну, переглядаючи їх kubeconfigs, і відразу ж визначати файли kubeconfig на локальному комп'ютері.
2. За допомогою Lens розробникі можуть об'єднати ці кластери в робочі групи, хоч би як вони з ними взаємодіяли.
3. Lens надає візуальні дані про стан об'єктів, таких як модулі, розгортання, простору імен, мережу, сховище і навіть настроювані ресурси у кластері. Це спрощує виявлення та усунення будь-яких проблем із кластером.

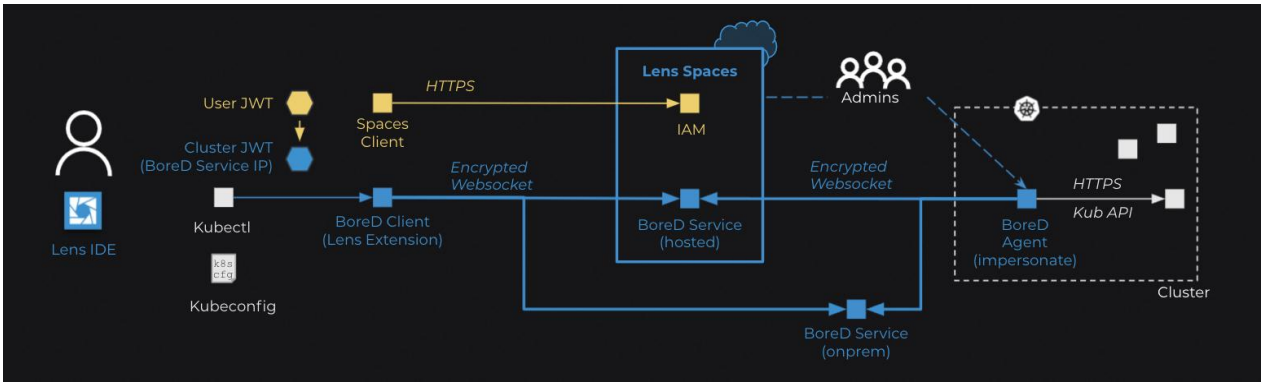
Розробникі також можуть викликати його вбудований термінал та виконувати команди за допомогою командного рядока kubectl. Вбудований термінал використовує версію kubectl, сумісну з API кластера. Lens дає вам доступ та дозволяє працювати з широким спектром кластерів Kubernetes у будь-якій хмарі в єдиному уніфікованому середовищі IDE. Кластери можуть бути локальними (наприклад minikube або Docker Desktop) або зовнішніми

(наприклад, Docker Enterprise, EKS, AKS, GKE, Rancher або OpenShift). Кластери можна додати, просто імпортуючи kubeconfig з деталями кластеру . Kubernetes Lens сприяє командній роботі та співпраці за допомогою функції Spaces. Це простір для спільної роботи команд та проєктів хмарних розробників. З простором для об'єктива розробникі можуть:

1. Легко організовувати та отримувати доступ до кластерів команди з будь-якого місця: GKE, EKS, AKS, локально або локальний кластер розробників.
2. Легкий доступ та безпечний загальний доступ до всіх кластерів у просторі Cluster Connect

У Kubernetes важко розділити доступ до різних кластерів. При роботі в якості адміністратора з різними постачальниками, які вимагають від розробників використання тих самих інструментів, або при спробі отримати доступ до файлів kubectl, потрібно щоб ці файли працюватли з вашим kubectl. Потім підключити файл kubectl до тієї ж мережі, що й API цільового кластера. Однак розробникам потрібно буде використовувати VPN, щоб бути в тій же мережі, що й провайдер, а в деяких випадках вам також потрібно використовувати інші провайдери IAM. Це ризики безпеки, тому що користувачі можуть оминути передові методи безпеки. Lens використовує Cluster Connect для спільного доступу до кластера без шкоди для безпеки кластера. За допомогою

Kubernetes Lens Spaces розробникі можетуть надсилати та отримувати доступ на запрошення до інших кластерів. Всі запрошення поєднуються, а потім надаються за допомогою проксі-сервера Lens Kubernetes. Щоб отримати доступ до кластерів, завантажуєте агент Cluster Connect у бажаний кластер. Потім агент дозволяє підключатися до кластерів з Lens Spaces, використовуючи наскрізне шифрування для захисту з'єднань між вами та кластерами, усуваючи необхідність VPN і необхідність включення вхідного порту на брандмауері. Це також означає, що ви можете легко отримати доступ до їх кластерів Kubernetes і працювати з ними з будь-якого місця.



2.5. Cluster Connect

Lens об'єднує кластери в логічні групи, які називаються робочими просторами. Це допомагає DevOps та SRE, яким доводиться керувати кількома (навіть сотнями) кластерами. Зазвичай один робочий простір містить список кластерів та їхню повну конфігурацію. Kubernetes Lens - один з найефективніших інтерфейсів Kubernetes. Він підтримує CRD Helm 3 та має зручний графічний інтерфейс. Lens, звичайно, також виконає налаштування кластеру. Kubernetes Lens забезпечує ситуаційну поінформованість про все, що працює в Kubernetes, знижуючи бар'єр для входу для розробників, які тільки-но починають працювати. Це ідеальне рішення з багатьох причин, зокрема:

1. Це дає впевненість у тому, що кластери правильно встановлені та налаштовані.
2. Підвищена наочність, статистика в реальному часі, потоки журналів та засоби прямого усунення несправностей.
3. Можливість швидко та легко організовувати кластери повністю підвищує продуктивність та швидкість бізнесу.
4. EKS, AKS, GKE, Minikube, Rancher, K0s і т.д. - будь-які Kubernetes, які розробникі можуть використовувати, - всі працюють з Lens. Потрібно тільки імпортувати kubeconfigs для відповідних кластерів.
5. Kubernetes Lens побудований на основі відкритого вихідного коду з активною спільнотою, що підтримується Kubernetes та хмарними екосистемами

2.4 Пакетний менеджер модулів Helm Chart

Kubernetes користується великою популярністю, і розробники шукають додаткові способи та методи для розгортання додатків у кластері цієї системи. Навіть командний рядок kubectl став сприйматися, як інструмент низького рівня, при цьому користувачі продовжують шукати ще простіші способи взаємодії з кластером. Draft, Gitkube, Helm, Ksonnet, Metaparticle і Skaffold - ось лише деякі інструменти, які допомагають розробникам створювати та розгортати програми в Kubernetes. Для розробки нашого додатку ми обрали Helm.

Helm – це менеджер пакетів для Kubernetes (apt або yum). Він працює шляхом поєднання кількох маніфестів в один пакет, який називається діаграмою. Helm також підтримує зберігання діаграм у віддалених або локальних репозиторіях Helm, які працюють як реєстри пакетів, такі як Maven Central, Ruby Gems, реєстр npm тощо.

Helm в даний час є єдиним рішенням, що підтримує:

- Базова підтримка шаблонів та значень для маніфестів Kubernetes
- Оголошення залежності між програмами (діаграма діаграм)
- Реєстр доступних програм для розгортання (репозиторій Helm)
- Тип кластера Kubernetes на рівні програми / діаграми
- Управління встановленням / оновленням графіків загалом
- Вбудований відкат діаграми до попередньої версії без повторного запуску конвеєра CI/CD.

Написання та підтримка маніфестів Kubernetes YAML для всіх необхідних об'єктів Kubernetes може бути трудомістким та стомлюючим завданням. Для найпростішого розгортання вам знадобиться щонайменше 3 YAML-маніфести з дубльованими та жорстко запрограмованими значеннями. Helm спрощує цей процес та створює єдиний пакет, який можна розгортати у вашому кластері.

Helm - це клієнт-серверна програма, яка донедавна покладалася на Tiller (сервер управління) для розгортання у вашому кластері. Він встановлюється

під час встановлення / ініціалізації helm на вашому клієнтському комп'ютері. Tiller просто отримує запити від клієнта та встановлює пакет у ваш кластер. Helm можна легко порівняти з RPM пакетів DEB в Linux, надаючи розробникам зручний спосіб упаковати та надіслати програму кінцевим користувачам для встановлення.

Helm складається з двох частин:

- Клієнт (CLI), який знаходиться на вашому локальному компютеру.
- Сервер (Tiller), який знаходиться в кластері Kubernetes для виконання того, що потрібно.

Ідея полягає в тому, що ви використовуєте CLI для просування необхідних вам ресурсів, а tiller стежитиме за тим, щоб цей стан дійсно був виконаний, створюючи/оновлюючи/видаляючи. Три основні концепції Helm:

- Діаграма: пакет попередньо налаштованих ресурсів Kubernetes.
- Реліз: конкретний екземпляр діаграми, розгорнутий у кластері за допомогою Helm.
- Репозиторій: група опублікованих діаграм, які можуть бути доступні іншим користувачам.

Використання Helm дозволяє програмному забезпеченню розгортати свої тестові середовища одним натисканням кнопки. Прикладом цього може бути те, що для тестування нової функції інженеру необхідна база даних SQL. Замість того, щоб виконувати процес локальної установки програмного забезпечення, створення необхідних баз даних та таблиць, інженер може просто запустити одну команду Helm Install, щоб створити та підготувати базу даних, готову до тестування.

Після того, як діаграма побудована один раз, її можна використовувати щоразу знову і знову. Той факт, що розробник може використовувати ту саму діаграму для будь-якого середовища, знижує складність створення чого-небудь для розробки та тестування. Розробник може просто налаштувати діаграму та переконатися, що вона готова до застосування у будь-якому середовищі. Так ми отримуємо перевагу використання готової до

застосування діаграми. Helm Charts спрощує встановлення перевизначених значень за умовчанням у файлі values.yaml , дозволяючи постачальникам програмного забезпечення або адміністраторам діаграм визначати базові налаштування.

Розробники та користувачі діаграм можуть змінити ці налаштування під час встановлення діаграми відповідно до своїх потреб. Якщо потрібна установка за умовчанням, перевизначення не потрібне.

Коли ви створюєте нову діаграму, Helm має певну структуру:

- .helmignore : містить усі файли, які слід ігнорувати при упаковці діаграми.
- Chart.yaml : тут розробник поміщає всю інформацію про діаграму, яку він укладає. Так, наприклад, номер версії діаграми і т. д.
- Values.yaml : тут розробник визначаєте всі значення, які хоче додати до шаблонів.
- Діаграми: тут зберігаються інші діаграми, від яких залежить діаграма. Можливо, розробник викликає іншу діаграму, яка має правильно функціонувати.
- Шаблони: в цю папку потрібно поміщати фактичний маніфест, який розгортається разом із діаграмою. Наприклад, розробник може розгортати nginx, якому потрібна служба, конфігураційна картка та секрети. У каталозі шаблонів будуть файли deployment.yaml, service.yaml, config.yaml і secrets.yaml. Всі вони отримають значення з values.yaml зверху.

2.5 Виснокок до другого

У проєкті було використано новітні технології для створення розгортання та підтримки сервера сторенного в хмарній платформі Amazon Web Services. Платформа Kubernetes дозволяє користувачам запускати масштабовані, високодоступні контейнерні робочі навантаження. Враховуючи унікальні проблеми, що виникають у світі, Інтернету речей все більше і більше інструментів стають стандартизованими в екосистемі Kubernetes для підвищення безпеки, видимості та зручності використання.

Графічний інтерфейс Lens дозволяє налаштовувати, змінювати та перенаправляти кластери одним натисканням миші, організовуючи та розкриваючи всю робочу систему в кластері, надаючи метрики. Ми можемо спостерігати за роботою наших сервісів у режимі реального часу.

Helm Chart дозволяє програмному забезпеченню швидко розгортати наші середовища одним натисканням клавші, що підвищує ефективність та зменшує затрати часу.

Розділ 3. РОЗРОБКА СИСТЕМИ ДЛЯ ДОДАТКУ

3.1 Структура системи ідентифікації студентів

Основна частина «Системи ідентифікації студентів» скаладається з таких частин:

- Elastic Load Balancig
- Proxy-nginx
- Admin-panel
- Client-site
- Back-office
- Back-End
- RabbitMQ
- Redis
- MongoDB
- DataDog
-

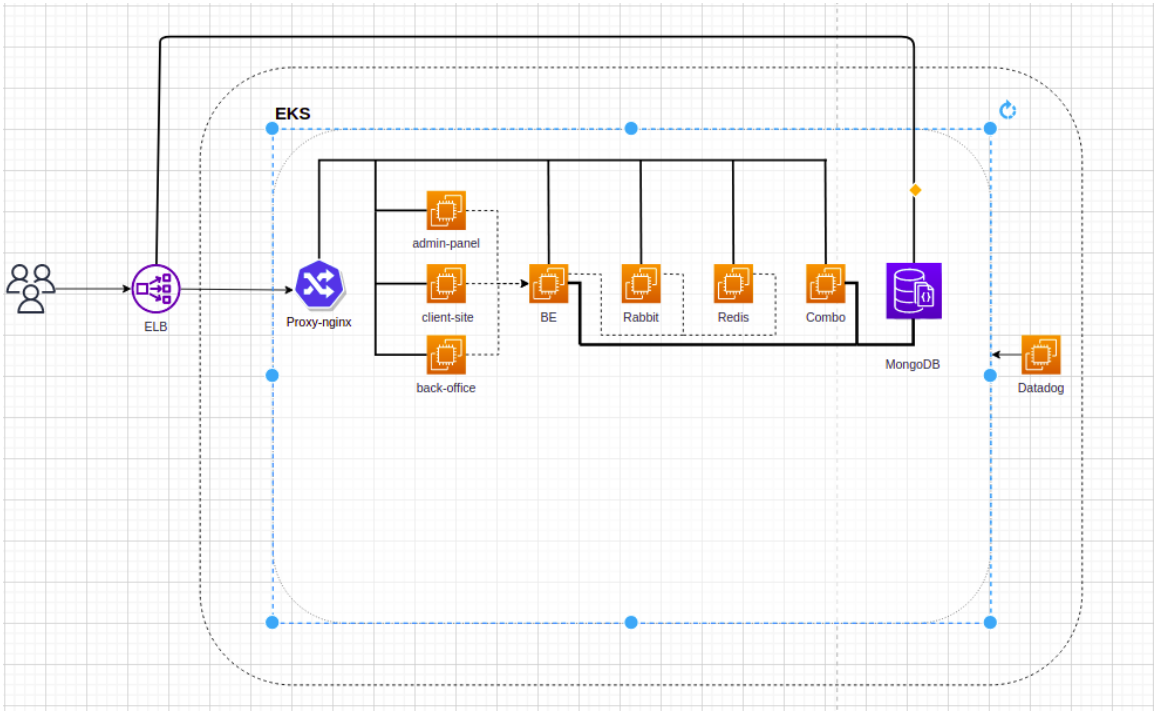


Рис. 3.1. Інфраструктура додатку

Elastic Load Balancing допомагає нам автоматично розподіляти трафік від вхідних програм за різними напрямками, такими як наші інстанси Amazon

EC2, контейнери, IP-адреса. Ми можемо контролювати змінне навантаження трафіку наших програм в одній зоні або кількох зонах доступності. Elastic Load Balancing має необхідний рівень високої доступності, автоматичної масштабованості та безпеки, щоб робить нашу програму стійкою до помилок.

Proxy-nginx - однією з причин для використання проксування Nginx в нашому проєкті є можливість масштабування інфраструктури. Nginx вміє одночасно керувати кількома паралельними з'єднаннями. Це робить його ідеальним сервером для контакту з клієнтами. Сервер може передавати запити на будь-яку кількість бекенд-серверів для обробки основного масиву трафіку, що надходить у нашу інфраструктуру. Це також забезпечує гнучкість при додаванні або заміні бекенд-серверів у міру необхідності.

Admin-panel - графічна панель управління сайтом де адміністратор зможе керувати контентом нашого додатку, добавляти викладачів, групи студентів та списки предметів. За допомогою цього інструмента є можливість редагувати розклад занять та час.

Client-site – скаладється з сайту для викладача та сайту для студента. У додатку викладача буде розклад його занять час та QR-код для сканування студентами. Також викладач має змогу переглянути кількість відвідин його занять конкретним студентом. У додатку студента буде також розклад занять їх час так сканувальник QR-кодів.

Back-end – це створення та рендер нашого додатку на стороні сервера. Тут відбувається взаємодія з базою данних та API, бекенд відповідає за зберігання та організацію данних. Бекенд взаємодіє із зовнішнім інтерфейсом, відправляючи та отримуючи інформацію, яка відображатиметься у вигляді веб-сторінки. Щоразу, коли буде відбуватися будь-яка взаємодія з користувачем на стороні клієнта, браузер надсилає запит на сторону сервера, який повертає інформацію у вигляді коду зовнішнього інтерфейсу, який браузер може інтерпретувати та відображати.

RabbitMQ - по суті є менеджером черг. Деколи у веб-застосунках з'являється необхідність виконати складні ресурсомісткі завдання, які не

можуть бути поміщені в короткому часовому інтервалі HTTP запиту. І тут на допомогу приходять черги. Основна ідея черг – уникнути виконання ресурсомістких завдань безпосередньо після надсилання запиту. Натомість завдання ставиться в чергу для подальшого виконання в асинхронному режимі. Тобто. при отриманні запиту від клієнта ми інкапсулюємо завдання як повідомлення та відправляємо його в чергу, а вже обробник черги дістає повідомлення в порядку їхнього слідування та обробляє належним чином.

Redis - це не реляційна структура даних у пам'яті, що використовується як база даних. Дані зберігаються як пара ключ-значення. І при цьому сховище вміє масштабуватись шляхом реплікації між серверами. Redis зберігає всі дані в пам'яті, що дозволяє зробити доступ до даних максимально швидким у порівнянні з іншими базами. Ми будемо використовувати Redis як кеш.

MongoDB – наша основна база данних. MongoDB дозволяє нам динамічні зміни схеми після яких легко вносити гнучкі зміни без необхідності переробляти існуючу базу даних для підтримки нових полів. Крім того, ієрархія документів легко зіставляється з ієрархіями об'єктів у коді програми, спрощуючи операції створення, читання, оновлення та видалення.

DataDog - це платформа моніторингу та аналітики на основі SaaS для великомасштабних програм та інфраструктури. Поєднуючи в реальному часі метрики серверів, контейнерів, баз даних та додатків з наскрізним трасуванням, Datadog надає дієві попередження та потужну візуалізацію, щоб забезпечити повне спостереження за нашим додатком. Datadog відстежує хмарні програми для комп за допомогою аналізу даних, серверів моніторингу, інструментів, баз даних та різних сервісів, щоб допомогти розробникам максимізувати продуктивність та покращити взаємодію з користувачем.

3.2 Написання коду для створення сервера

Для написання нашої інфраструктури було обрано інструмент Terraform. Він допоможе автоматизувати всі основні процеси для повноцінної роботи нашого сервера. Terraform використовує мову HashiCorp Configuration Language. Він сумісний із JSON, що означає, що він сумісно з іншими

системами між лінійними продуктами Terraform. При об'єднанні підфайли утворюють добре структурований файл конфігурації HCL. Ця структура допомагає точно і легко описувати конфігурації середовища, необхідних інструменту Terraform.

Перший файл це autoscaler. Він потрібний для авто-масштабування, що дозволяє автоматично збільшувати та зменшувати робочі навантаження залежно від використання ресурсів.

autoscaler.tf

common > autoscaler.tf

```
module "eks-cluster-autoscaler" {
  source  = "lablabs/eks-cluster-autoscaler/aws"
  version = "1.5.0"
  # insert the 3 required variables here
  cluster_identity_oidc_issuer     = module.eks-cluster.cluster_oidc_issuer_url
  cluster_identity_oidc_issuer_arn = module.eks-cluster.oidc_provider_arn
  cluster_name                     = local.cluster_name
}
```

Рис. 3.2. Autoscaler.tf

Наступний файл eks це керований контейнерний сервіс для запуску та масштабування додатків Kubernetes у AWS

common > eks.tf

```
locals {
  cluster_name = "${var.project_name}-${var.env}-eks"

  worker_group = [
    {
      name                          = "worker-${var.env}-${var.instance_type}"
      instance_type                 = var.instance_type
      additional_userdata           = ""
      asg_desired_capacity          = 2
      additional_security_group_ids = [aws_security_group.worker_group_mgmt_one.id]
    }
  ]
}

module "eks-cluster" {
  source = "./module/eks-cluster"

  cluster_name    = local.cluster_name
  cidr_block      = var.cidr_block
  private_subnets = module.vpc.private_subnets
  public_subnets  = module.vpc.public_subnets
  worker_group    = local.worker_group
  region          = var.region
  vpc_id          = module.vpc.vpc_id

  aws_profile = var.aws_profile
}

data "aws_eks_cluster" "cluster" {
  name = module.eks-cluster.cluster_id
}
data "aws_eks_cluster_auth" "cluster" {
  name = module.eks-cluster.cluster_id
}
```

Рис. 3.3. Eks.tf

Ще один необхідний файл це security-groups який являює собою правила доступу для захисту віртуальних машин ECS

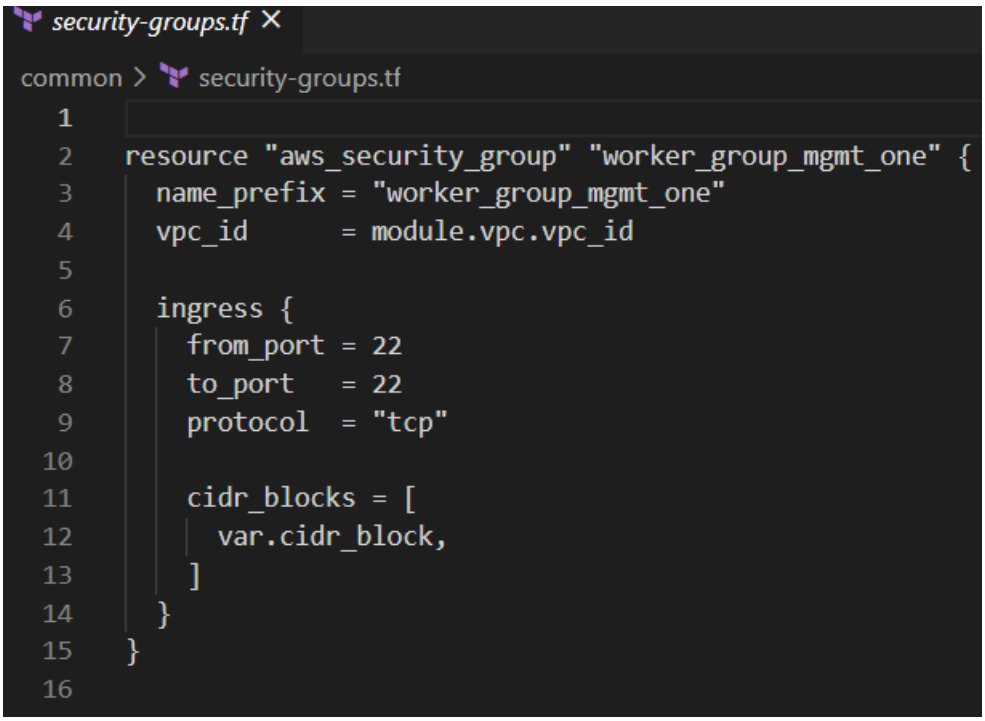


Рис. 3.4. Security-groups.tf

Route 53 – це високодоступний та масштабований хмарний веб-сервіс системи доменних імен (DNS). Ми використовуємо його як дуже надійний та ефективний метод перенаправлення кінцевих користувачів до інтернет-додатків, переводячи доменні імена у формат цифрових IP-адресів, зрозумілих для комп'ютера. Route 53 також повністю сумісний із протоколом IPv6. Сервіс Amazon Route 53 надсилає запити користувачів до інфраструктури AWS, наприклад до інстансів Amazon EC2, балансувальників навантаження Elastic Load Balancing або кошиків Amazon S3.

Крім того, він може використовуватися для перенаправлення користувачів на інфраструктуру за межами AWS. Amazon Route 53 можна використовувати як для організації підключень тільки до «здорових» адресів (з використанням перевірок DNS), так і для незалежного моніторингу стану програми та її кінцевих точок.

route53.tf

prod > route53.tf

```
resource "aws_route53_zone" "system-identification-com-ua" {
# Imported with ./tf.sh import aws_route53_zone.system-identification-com-ua K00355791G36FEH259K8DQ
    name = "system-identification.com.ua"
    tags = {
        "Name" = "dev"
    }
}

module "zones" {
  source = "./module/aws-route53/modules/zones"

  zones = {

    "system-identification.com.ua" = {
      comment = "prod"
      tags = {
        Name = "prod"
      }

    }
  }
}

resource "aws_route53_record" "system_identificationl_com_ua" {
  zone_id = module.zones.this_route53_zone_zone_id["system_identificationl.group"]
  name    = "bbb.system_identificationl.group"
  type    = "A"
  ttl     = "3600"
  records = ["32.79.205.266"]
}

resource "aws_route53_record" "turn_system_identificationl_com_ua" {
  zone_id = module.zones.this_route53_zone_zone_id["system_identificationl.group"]
  name    = "bbb-turn.system_identificationl.group"
  type    = "A"
  ttl     = "3600"
  records = ["12.169.102.289"]
```

Рис. 3.5. Route-53.tf

Kubernetes.tf використовується для взаємодії з ресурсами, що підтримуються Kubernetes.

kubernetes.tf

common > module > eks-cluster > kubernetes.tf

```
# Kubernetes provider
# https://learn.hashicorp.com/terraform/kubernetes/provision-eks-cluster#optional-configure-terraform-kubernetes-provider
# To learn how to schedule deployments and services using the provider, go here: https://learn.hashicorp.com/terraform/kubernetes/deploy

# The Kubernetes provider is included in this file so the EKS module can complete successfully. Otherwise, it throws an error when creat
# You should **not** schedule deployments and services in this workspace. This keeps workspaces modular (one for provision EKS, another

provider "kubernetes" {
  host                   = data.aws_eks_cluster.cluster.endpoint
  token                  = data.aws_eks_cluster_auth.cluster.token
  cluster_ca_certificate = base64decode(data.aws_eks_cluster.cluster.certificate_authority.0.data)
}

/*
provider "kubectl" {
  host                   = data.aws_eks_cluster.cluster.endpoint
  cluster_ca_certificate = base64decode(data.aws_eks_cluster.cluster.certificate_authority.0.data)
  token                  = data.aws_eks_cluster_auth.cluster.token
  load_config_file       = false
}

*/
```

Далі необхідно написати сценарій який буде об’єднувати нашу структуру та запускати створення усіх необхідних ресурсів.

common > tf.sh

```
#!/usr/bin/env bash

ACTION=$@

TERRAFORM_VERSION="1.0.5"
TERRAFORM_PLATFORM=linux
case "$OSTYPE" in
darwin*) TERRAFORM_PLATFORM=darwin ;;
linux*) TERRAFORM_PLATFORM=linux ;;
esac

TERRAFORM_BASE_DIR="$(pwd)/.terraform"
TERRAFORM_BIN_DIR="${TERRAFORM_BASE_DIR}/bin"
TERRAFORM_ENV_DATA_DIR="${TERRAFORM_BASE_DIR}/data"

if [ -d "${TERRAFORM_BIN_DIR}" ] && [ -x "${TERRAFORM_BIN_DIR}/terraform" ]; then
  echo "terraform executable file found..."
else
  echo "terraform executable file not found, installing..."
  mkdir -p ${TERRAFORM_BIN_DIR}
  #wget https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terraform_${TERRAFORM_VERSION}_${TERRAFORM_PLATFORM}
  wget https://releases.hashicorp.com/terraform/${TERRAFORM_VERSION}/terraform_${TERRAFORM_VERSION}_${TERRAFORM_PLATFORM}_
  unzip -o /tmp/terraform.zip -d ${TERRAFORM_BIN_DIR} >/dev/null
  if [ -d "${TERRAFORM_BIN_DIR}" ] && [ -x "${TERRAFORM_BIN_DIR}/terraform" ]; then
    echo "terraform executable file installed successfully..."
  else
    echo "error while terraform executable file installation exiting..."
    exit 1
  fi
fi

${TERRAFORM_BIN_DIR}/terraform "$@"
```

Рис. 3.6. Terrform.sh

Після створення нашої інфраструктури нам необхідно отримати AssumeRole. Це являє собою механізм автентифікації в AWS IAM, що дозволяє отримати тимчасові дані доступу для виконання запитів до ресурсів. Для цього необхідно написати сценарій для отримання цих доступів.

```
function aws_configure_profile_admin() {
  local arn=$(aws configure get profile.SystemIdentificationAdminRole.role_arn)
  if [ -z $arn ] || [ "$arn" != "$masterrole" ]; then
    aws configure set profile.SystemIdentificationAdminRole.output json
    aws configure set profileSystemIdentificationRole.region ${region}
    aws configure set profile.SystemIdentificationRole.role_arn ${masterrole}
    aws configure set profile.SystemIdentificationRole.source_profile default
    aws configure set profile.SystemIdentificationRole.duration_seconds 3600
  else
    aws configure get profile.SystemIdentificationRole.role_arn
  fi
}
function aws_configure_profile_master() {
    aws configure set profile.MasterRole.output json
    aws configure set profile.MasterRole.region ${region}
}
function aws_sts_assume_master_role() {
  printf "${RED}"
  local stats=$(aws sts assume-role --role-arn ${masterrole} --role-session-name=MasterRole-local-$USER > $ASSUME_ROLE_PATH ; echo $?)
  printf ${NC}
  if [[ $stats -ne 0 ]]; then
      echo -e "${RED}Problem with assume your user to master role.${NC}"
      echo -e "${PURPLE}To fix problem, please add your user aws to trusted list.${NC}"
      exit 5
  fi
  aws configure set profile.MasterRole.aws_access_key_id '$(_parse_assume_role_json AccessKeyId)'
  aws configure set profile.MasterRole.aws_secret_access_key $(_parse_assume_role_json SecretAccessKey)
  aws configure set profile.MasterRole.aws_session_token $(_parse_assume_role_json SessionToken)
}
function apply_cluster_config(){
  printf "${RED}"
  local stats=$(aws eks --region ${region} update-kubeconfig --name ${clustername} >/dev/null ; echo $?)
  printf ${NC}
  if [[ $stats -ne 0 ]]; then
      echo -e "${RED}Problem with eks cluster.${NC}"
      exit 7
  fi
}
```

Рис. 3.7. Assumerole.sh

Після цього ми можемо під’єднатися до нашого кластера за допомогою графічного інтерфейсу Lens де можемо спостерігати успішне створення нашої інфраструктури.

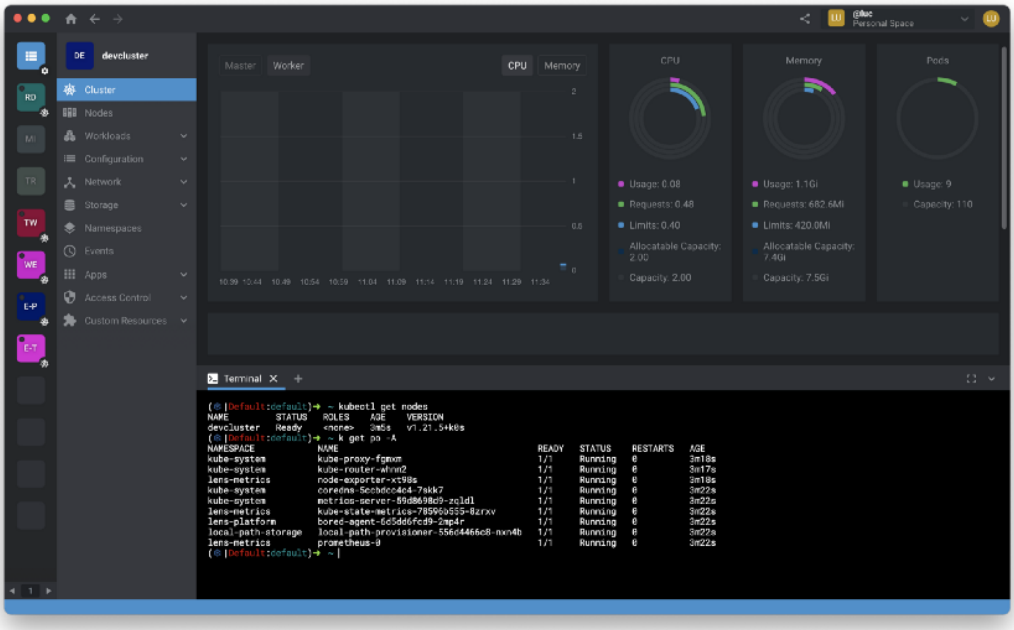


Рис. 3.8. Інфраструктура в Lens

3.3 Налаштування серверу для роботи з додатком

Далі нам необхідно встановити ресурси які потрібні для повноцінної роботи «Системи ідентифікації студентів». Для цього ми скористаємось менеджером пакетів для Kubernetes який називається Helm. Спочатку нам необхідно написати файли які описують ресурси, необхідні для запуску програми, інструменту або служби всередині кластера Kubernetes.

Наші мікросервіси це Redis, RabbitMQ, Datadog, Nginx та ще декілька потрібних для роботи інфраструктури. Тому нам необхідно стоворити файли їхньої конфігурації. Під час інсталяції пакета Helm об'єднує шаблони пакета із заданою нами конфігурацією та значеннями за замовчуванням із файлу value.yaml. Для цих пакетів виконується рендеринг у маніфестах Kubernetes, які потім розгортаються у Kubernetes API. При цьому створюється реліз, тобто конкретна конфігурація і розгортання конкретного пакета.

common > ! redis.yaml

```yaml
annotations:
  category: Database
apiVersion: v2
appVersion: 6.2.6
dependencies:
  - name: common
    repository: https://charts.bitnami.com/bitnami
    tags:
      - bitnami-common
    version: 1.x.x
description: Open source, advanced key-value store. It is often referred to as a data structure server since keys can contain strings,
engine: gotpl
home: https://github.com/bitnami/charts/tree/master/bitnami/redis
icon: https://bitnami.com/assets/stacks/redis/img/redis-stack-220x234.png
keywords:
  - redis
  - keyvalue
  - database
maintainers:
  - email: containers@bitnami.com
    name: Bitnami
  - email: cedric@desaintmartin.fr
    name: desaintmartin
name: redis
sources:
  - https://github.com/bitnami/bitnami-docker-redis
  - http://redis.io/
version: 15.6.4
```

Рис. 3.9. Створення конфігурації для Redis

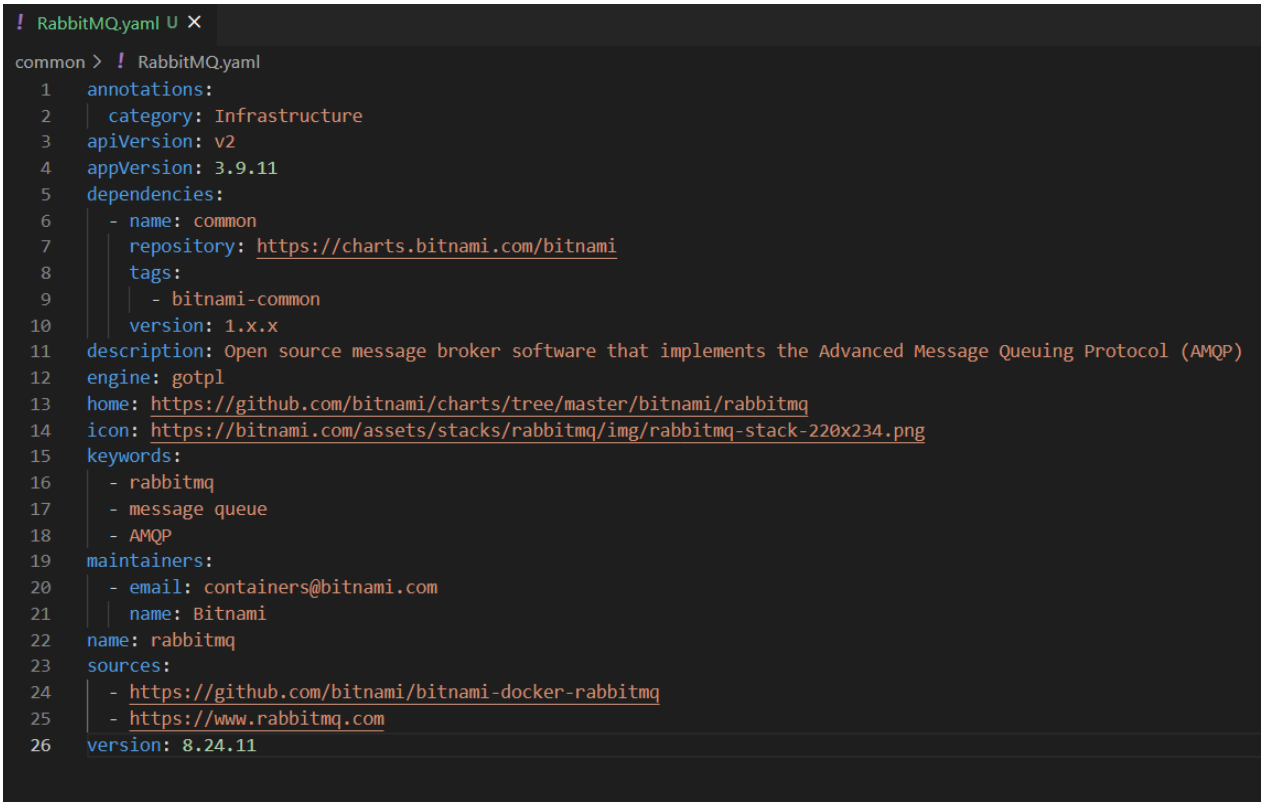


Рис. 3.10. Створення конфігурації для RabbitMQ

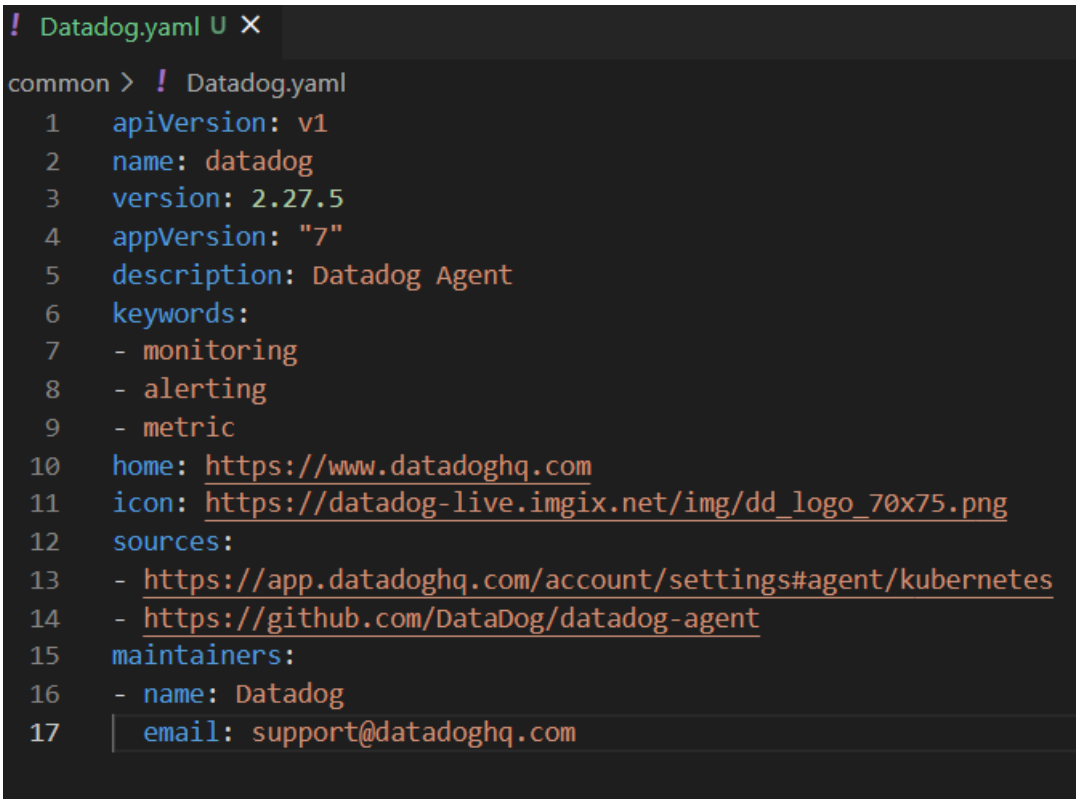


Рис. 3.11. Створення конфігурації для Datadog

Після того, як ми створили наші файли нам потрібно встановити мікросервіси на сервер. Це виконується за допомогою команди :

```
helm install -f myvalues.yaml myredis ./redis
```

Тепер відкривши наш графічний інтерфейс Lens ми можемо побачити, що наші мікросервіси успішно встановилися і наша інфраструктура повністю готова до використання.

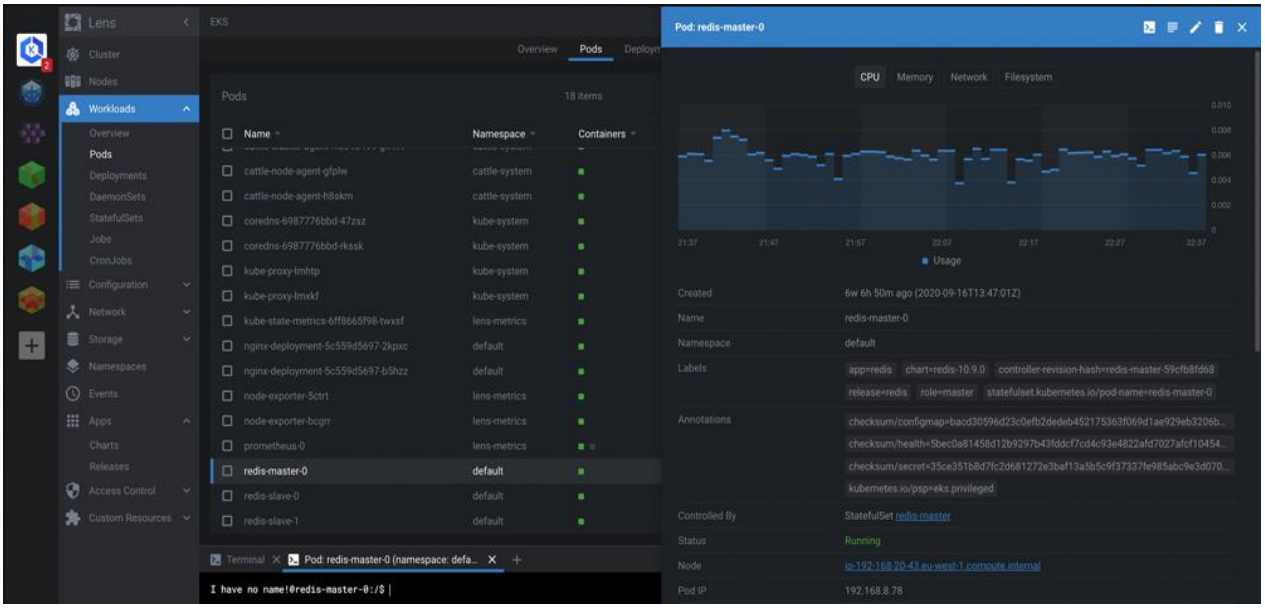


Рис. 3.12. Мікросервіси на сервері

3.4 Висновок до третього розділу

Ми створили та розгорнули нашу інфраструктуру сервера та баз даних для «Системи ідентифікації студентів», що б забезпечити повноцінну та захищенну роботу системи за допомогою новітніх технологій в області розробки програмного забезпечення, що дозволяє легко керувати, масштабувати та змінювати її структуру.

За допомогою графічного інтерфейсу Lens можна відстежувати усі зміни які відбуваються на сервері. Це довзоляє керувати навантаженням на сервер, пам'ять, процесор, модулі, мережу та встановленні ресурси. Таким чином це спрощує виявлення та усунення помилок.

Пакетний менеджер Helm дозволяє з легкістю встановлювати додаткові модулі необхідні для ефективнішої роботи структури.

ВИСНОВКИ

Під час реалізації проєкту було створенно програмне забезпечення для додатку «Системи ідентифікації студентів». Завдяки основній концепції технічної сторони проекту, ми використовувати новітніші хмарні технології. За допомогою новітніх технологій ми можемо контролювати навантаження на наш сервер та розподіляти його за потреби. Сервісіви, які ми маємо у своє розпорядженні, дають можливість в режимі реального часу аналізувати потік данних на наш ресур та відстужвати усі змінни, які відбуваються на ньому.

Платформа Kubernetes, яку ми використовували при розробці програмного забезпечення, дозволяє нам запускати масштабовані, високодоступні контейнерні робочі навантаження. Наш графічний інтерфейс Lens дозволяє налаштовувати, змінювати та перенаправляти кластери одним натисканням миші, організовуючи та розкриваючи всю робочу систему в кластері, надаючи метрики.

Інфраструктура має продуману архітектуру. Її можна легко масштабувати та змінювати при необхідності. Встановлені модулі допомагають ефективніше здійснювати роботу системи та надають можливість отримання повної інформації того, що відбувається, як на сервері так і на клієнтській стороні. Це дозволяє нам ефективно аналізувати отриманні дані для покращення розуміння потреб наших користувачів та подальшої оптимізації додатку, це робить наш додаток для системи ідентифікації студентів зручним та простим у використанні. Сучасні технології дозволяють швидко обробляти запити та надавати потрібну інформацію. Надійний захист протоколів безпеки гарантує збереження конфіденційної інформації користувачів. Усі встановленні модулі можна легко замінювати або добавляти нові, що додає гнучкість нашій системі.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

1. AWS– Amazon Web Services
2. API – Application Programming Interface
3. IP – інтернет протокол
4. HTTP - протокол передачі гіпер-текстових документів
5. СУБД - система управління базами даних
6. EKS- Elastic Kubernetes Service
7. ЦС – центр сертифікації
8. VPN - віртуальна приватна мережа
9. SSL - рівень захищених сокетів
10. DNS - система доменних імен

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронний ресурс – Режим доступу: https://www-terraform-io.translate.goog/docs/language/providers/index.html?_x_tr_sl=en&_x_tr_tl=uk&_x_tr_hl=ru
2. Електронний ресурс-Режим доступ : https://proselyte.net/tutorials/mongodb/advantages// Руководство по MongoDB.
3. Роберт Мартин, Чистая архитектура. Искусство разработки программного обеспечения.// СПб.
4. Електронний ресурс. – Режим доступу: http://www.informika.ru/ State Institute of Information Technologies and Telecommunications
5. Нейгард М., Проєктирование и дизайн ПО для тех, кому не всё равно./ М.Нейгард.// СПб.: Питер, 2016. – 320 с.: ил. – (Серия «Библиотека программиста»)
6. Грицюк Ю. І., Аналіз вимог до програмного забезпечення. Навчальний посібник/ Ю. І. Грицюк .// Київ,2018, С.425
7. Майкл Віттіг. «Amazon Web Services in Action ». США, Manning, 1237 – 1469 с.
8. Електронний ресурс – Режим доступу: https://docs.aws.amazon.com/wellarchitected/latest/high-performance-computing-lens/wellarchitected-high-performance-computing-lens.pdf
9. Гордєєв, А. В. Системне програмне забезпечення / А.В. Гордєєв, А.Ю. Мовчанів. – М.: Пітер, 2016. – 736 с.
10. Майєрс, Г. Надійність програмного забезпечення/Г. Майєрс. – К.: Світ, 2018. – 360 с.
11. Васильєв, А. Є. Мікроконтролери. Розробка додатків, що вбудовуються / А.Є. Васильєв. - М: БХВ-Петербург, 2012. - 304 с..
12. Добринін, В. Ю. Технології компонентного програмування/В.Ю. Добринін. - Москва: Гостехіздат, 2004. - 216 с.

13. Соболь, Б.В. Методи оптимізації. Практикум/Б.В. Соболь. - М: Фенікс, 2009. - 650 с.
14. Електронний ресурс – Режим доступу: https://kubernetes.io/uk/docs/setup
15. Лукін, В. В. Технологія розробки програмного забезпечення. Навчальний посібник/В.В. Лукін, В.М. Лукін, Т.В. Лукін. - Москва: Гостехвидав, 2015. - 286 з
16. Мілов, А. В. Основи програмування у завданнях та прикладах / А.В. Милів. - М: Фоліо, 2002. - 400 с.
17. Оліфер Н.А., Оліфер В.Г. Мережеві операційні системи. - СПб.: Пітер, 2001
18. Вірт Н. Алгоритми та структури даних. - М.: Світ, 1989. - 360 с.
19. Мадник С., Донован Дж. Операційні системи
20. Юрагов, Є. А. Програмне забезпечення вимірювальних процесів. Частина 2. Програмування плат збору даних та розробка вимірювальних додатків/Є.А. Юрагів. - М: МГОУ, 2013. - 212 с.
21. JavaScript - [Електронний ресурс] URL:https://developer.mozilla.org
22. Addy Osmani. Developing Backbone.js Applications. US: O'Reilly Media, 2013
23. Прамодкумар Дж. Садападж, Мартін Фаулер. NoSQL. Нова методологія розробки нереляційні бази даних. Москва: видавництво "Вільямс", 2013.
24. Буторін Д.М. Розробка баз даних у MongoDB. Красноярськ: КДПУ, 2013.
25. Angular.Туловський А. Документація з API Angular.js. Електронна версія, 2013.
26. Кайл Бенкер. MongoDB у дії. Москва: видавництво "ДМК Прес", 2012
27. Стів Макконел. Ідеальний код, 2-е видання. Санкт-Петербург: видавництво "Пітер", 2012.
28. С.Архіпенков. Лекції з управління програмними проєктами. Москва, 2009.

29. А. Якобсон, Г. Буч, Дж. Рамбо. Уніфікований процес розробки програмного забезпечення. Санкт-Петербург: видавництво "Пітер", 2002.
30. Мартін Дж. Планування розвитку автоматизованих систем. - М.: Фінанси та статистика, 2005. - 191с.
31. Маклаков С. В. BPwin та ERwin. CASE-засоби розробки інформаційних систем. - М.: Діалог - МІФІ, 2007. - 125с.
32. Крёнке Д. Теорія та практика побудови баз даних. - СПб.: Пітер, 2007. - 800с.
33. Інформаційні технології. Навчальний посібник/Под ред. А. К. Волкова. - М.: Видавництво ІНФРА, 2008. - 256с.
34. Девід А. Марка, Клемент МакГоуен. Методологія структурного аналізу та проєктування SADT. - М.: ДІАЛОГ-МІФІ, 2005. - 243 с.
35. Орлов С.А. Технологія розробки програмного забезпечення. 2-ге видання. - СПб: Пітер - прес, 2008. - 379с.