МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-практичної програми «Інформатика»
на тему: «Інформаційна технологія моніторингу хмарних додатків із мікросервісною архітектурою»
здобувача групи ІН.м-25 Бекова Олексія Ангеловича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Олексій БЕКОВ
(підпис)

Керівник,
старший викладач кафедри
компьютерних наук, к.т.н. _____ Борис КУЗІКОВ
(підпис)

СУМИ 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри

_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
на здобуття освітнього ступеня магістр
зі спеціальності 122 - Комп'ютерних наук, освітньо-практичної програми «Інформатика»
здобувача групи ІН.м-25, Бекова Олексія Ангеловича

1. Тема роботи: «Інформаційна технологія моніторингу хмарних додатків із мікросервісною архітектурою»
затверджую наказом по СумДУ від «06» грудня 2023 р. №1412-VI
2. Термін здачі здобувачем кваліфікаційної роботи до16.12.2023 р.
3. Вхідні дані до кваліфікаційної роботи
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження.
2) Огляд технологій, що використовуються для моніторингу хмарних рішень 3) Розробка системи моніторингу, яка забезпечує вчасне інформування відказу комунікацій між мікросервісами і зовнішними системами 4) Аналіз результатів.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
6. Консультанти до проєкту (роботи), із значенням розділів проєкту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 2023___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	01.10-10.10.2023	
2	*Огляд технологій, що використовуються для оцінки якісних показників софтверних проєктів*	11.10-14.10.2023	
3	*Розробка інформаційної технології для оцінки якісних показників імплементаційних софтверних проєктів*	15.10-31.10.2023	
4	*Аналіз отриманих результатів*	1.11-20.11.2023	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	21.11-02.12.2023	

Здобувач вищої освіти _____ Керівник _____
(підпис) (підпис)

РЕФЕРАТ

Записка: 54 с., 12 рис., 3 табл., 29 літературних джерел, 3 додатки.

Обґрунтування актуальності теми роботи — Тема кваліфікаційної роботи є актуальною, оскільки присвячена підвищенню сопстережуваності (observability) інформаційних систем побудованих за мікросервсіною архітектурою, що є складовою забезпечення їх захисту і підтримки критеріїв якості експлуатації.

Об'єкт дослідження — супровід інформаційних систем.

Мета роботи — розробити інформаційну технологію моніторингу хмарних додатків з метою стаблізації роботи системи при комунікації з зовнішніми і внутрішніми сервісами для отримання завчасних нотифікацій при помилках комунікації між ними.

Методи дослідження — аналіз та синтез програмних систем.

Результати — розроблено модель моніторингу хмарних рішень з мікросервісною архітектурою з перевіркою стабільності комунікації мікросервісів між собою і зовнішними системами, облікових данних, валідності сертифікатів.

АУТЕНТИФІКАЦІЯ І АВТОРИЗАЦІЯ, МІКРОСЕРВІСНА АРХІТЕКТУРА, МОНІТОРИНГ, FTP, GRAFANA, HTTP/HTTPS, KUBERNETES, PYTHON, REST API, SSH, SFTP, SOAP OVER HTTP

ЗМІСТ

ВСТУП

Актуальність. Типовим для сучасних інформаційних систем є використання хмарних середовищ та мікросервісного підходу. Підхід є популярним через гнучкість та широкі можливості до масштабування. Мікросервіси прискорюють цикл розробки, але водночас створюють труднощі в моніторингу та управлінні розподіленими компонентами, що може призводити до проблем з узгодженістю даних та управлінням транзакціями. На тлі цього виникає потреба в розробці інформаційної технології моніторингу розподілених інформаційних систем для забезпечення найближчого до нульового часу на простій в умовах великої кількості зав'язків між зовнішніми та внутрішніми сервісами.

Моніторинг у хмарному рішенні є проактивною та стратегічною практикою, яка дозволяє організаціям підтримувати високопродуктивну, безпечну та економічно ефективну IT-інфраструктуру. Це забезпечує видимість у реальному часі, швидке вирішення проблем і постійне вдосконалення відповідно до динамічної природи хмарних середовищ.

Об'єкт дослідження. Супровід інформаційних систем.

Предмет дослідження. Інформаційна технологія моніторингу розподілених інформаційних систем у хмарних середовищах.

Гіпотеза. Врахування особливостей хмарних середовищ і способів взаємодії можуть дозволити побудувати засоби моніторингу із часом реагування близьким до реального часу

Новизна. На відміну існуючих систем моніторингу реалізовано багатоканальну активну перевірку робото спосібності сервісів (heathchech), враховано особливості будови інформаційних систем у сфері OSS/BSS.

Структура. Робота складається зі вступу, аналізу засобів та підходів до моніторингу додатків, постановки задачі дослідження, вибір методики та

інструментів для рішення поставленої проблеми, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

1 ІНФОРМАЦІЙНИЙ ОГЛЯД

1.1 Огляд існуючих підходів до моніторінгу

Моніторинг хмарних додатків з архітектурою мікросервісів зазвичай передбачає поєднання інструментів і практик для забезпечення продуктивності, надійності та безпеки системи [1]. Основні методи та їх комбінації:

Логування. Використання фреймворків логування допомагає фіксувати події та дії в кожному мікросервісі. Інструменти відстеження надають уявлення про потік запитів у різних мікросервісах, допомагаючи визначати вузькі місця продуктивності та розуміти наскрізний потік транзакцій.[2]

Розподілені інструменти моніторингу. Оскільки мікросервіси розподілені між різними компонентами, використання інструментів моніторингу, здатних впоратися зі складністю розподіленої системи, є вирішальним. Для цієї мети зазвичай використовуються такі інструменти, як Prometheus, Grafana та Jaeger [3].

Якщо мікросервіси розгортаються всередині контейнерів, якими керують платформи оркестровки, такі як Kubernetes, інструменти моніторингу, специфічні для цих середовищ (наприклад, Kubernetes Dashboard, Prometheus Operator), використовуються для нагляду за робочими навантаженнями в контейнерах.

Моніторинг безпеки. Впровадження інструментів моніторингу безпеки для виявлення й реагування на потенційні загрози або вразливості в архітектурі мікросервісів має вирішальне значення для захисту конфіденційних даних і підтримки відповідності [4].

Моніторинг API. Оскільки мікросервіси спілкуються через API, моніторинг end-points API є життєво важливим. Такі інструменти, як Postman, Swagger або спеціальні служби моніторингу API, допомагають забезпечити належне функціонування та ефективність цих інтерфейсів. Найбільша

проблема це якість самих API, яка вимагає постійної перевірки їх роботи, кількість API в продуктовому телеком або білінгові рішенні, які використовуються між мікросервісами може в сереньому сягати від 150 до 1500 мікросервісів [5], моніторінг такої кількості може бути реально проблематичним.

Моніторинг безперервної інтеграції/безперервного розгортання (CI/CD). Моніторинг конвеєра CI/CD допомагає забезпечити плавність процесу розгортання, а будь-які проблеми, які виникли під час розгортання, можна виявити та оперативно вирішити [6].

Метрики та сповіщення. Збір показників щодо різних аспектів, таких як використання ресурсів, час відповіді, частота помилок і пропускна здатність, є важливим. Налаштування сповіщень на основі попередньо визначених порогових значень дозволяє швидко виявляти аномалії чи проблеми в мікросервісах і реагувати на них. Можливо зробити сповіщення на кожну проблему, але деталізувати проблему і виявити критичні для бізнеса помилки буде вкрай важко, тому дуже критично розуміти які сповіщення повинні бути з найвищим пріоритетом і як правильно будувати нотифікації для команди спостереження [7].

Поєднуючи ці методи та інструменти моніторингу, компанії можуть отримати вичерпну інформацію про стан і продуктивність своїх хмарних додатків за допомогою архітектури мікросервісів. Регулярний перегляд та оновлення стратегії моніторингу має важливе значення для адаптації до мінливих потреб системи.

Розглянемо комбінацію методів для яких на разі є прогалини, які зазначені вище в комплексній реалізації в існуючих рішеннях і від яких залежить стабільна робота всієї системи: Моніторинг конвеєра CI/CD, Моніторинг API, Метрики та сповіщення.

1.2 Моніторинг API

Інструменти моніторингу API відіграють роль у забезпеченні доступності, надійності та продуктивності API. Моніторинг API має вирішальне значення з кількох причин, що відображає важливість інтерфейсів прикладного програмування (API) у сучасних архітектурах програмного забезпечення та бізнес-операціях. Він є невід'ємною частиною підтримки надійності, продуктивності та безпеки сучасних програмних систем. Це дозволяє організаціям забезпечувати безперебійну роботу користувачів, відповідати очікуванням продуктивності та забезпечувати безперервну та ефективну роботу своїх програм [8].

В архітектурі мікросервісів API (інтерфейси прикладного програмування) відіграють вирішальну роль у полегшенні зв'язку та взаємодії між окремими мікросервісами. API визначають, як різні служби можуть запитувати або обмінюватися даними одна з одною. Ось кілька аспектів, які зазвичай обробляються API між мікросервісами:

- API дозволяють мікросервісам спілкуватися один з одним через мережу. Служби надають API, які визначають end-points та методи, за допомогою яких інші служби можуть взаємодіяти.
- API визначають формати обміну даними, такі як JSON або XML, які використовуються для зв'язку між мікросервісами. Це забезпечує загальне розуміння структури даних.
- API визначають протоколи для надсилання запитів і отримання відповідей. Це включає вказівку методів HTTP (GET, POST, PUT, DELETE) або інших протоколів зв'язку, таких як gRPC.
- API надають конкретні end-point, які представляють функції або ресурси, надані мікросервісом. Ці end-points — це URL-адреси або URI, які інші мікросервіси можуть викликати для виконання дій.

- API служать договірними контрактами між мікросервісами. Вони визначають умови, за яких один мікросервіс може взаємодіяти з іншим, включаючи очікувані вхідні параметри та формат повернених даних.
- API можуть бути задіяні в механізмах виявлення сервісів, де мікросервіси можуть динамічно виявляти розташування та end-points інших сервісів. Це особливо важливо в динамічних і масштабованих середовищах.
- API обробляють автентифікацію та авторизацію, щоб забезпечити доступ до певних функцій лише авторизованим мікросервісам або користувачам. Це вкрай важливо для підтримки безпеки в архітектурі мікросервісів [9].
- API визначають, як обробляються помилки під час зв'язку. Це включає вказування кодів помилок, повідомлень про помилки та очікуваної поведінки в разі виникнення помилки.
- API можуть сприяти асинхронному зв'язку між мікросервісами. Це часто реалізується за допомогою черг повідомлень або керованих подіями архітектур, що дозволяє службам публікувати події та підписуватися на них.
- API обробляють версії для керування змінами в інтерфейсі з часом. Це гарантує, що існуючі споживачі не будуть перервані під час розгортання нових версій мікросервісів.
- API можуть надавати механізми кешування даних для оптимізації продуктивності. Кешування може зменшити потребу в надлишкових запитах, покращуючи ефективність зв'язку мікросервісів.
- API можуть запроваджувати обмеження швидкості, щоб контролювати кількість запитів, які мікросервіс може зробити протягом визначеного періоду часу. Це допомагає запобігти зловживанням або надмірному використанню.

Підсумовуючи, API в архітектурі мікросервісів визначають правила та протоколи, які керують зв'язком між службами. Вони діють як контракт, який дозволяє мікросервісам злагоджено працювати разом, забезпечуючи рівень

абстракції та незалежності між окремими компонентами. Чіткість і надійність API мають вирішальне значення для успіху системи на основі мікросервісів.

Відсутність моніторингу API у хмарному рішенні може впливають на загальний стан систем, продуктивність і безпеку системи. Компанії можуть зіткнутися з труднощами, пов’язаними з забезпеченням безперебійної взаємодії з користувачем, оптимізацією використання ресурсів і оперативним вирішенням проблем, що виникають у їхній екосистемі API.

Щоб пом’якшити ці ризики, впровадження надійних практик моніторингу API є важливим у хмарних рішеннях, але інструменти моніторингу не можуть дати якості без впровадження метрик та сповіщення про падіння або помилок самих API.

Обираємо ключові критерії для моніторингу API з вище вказаних характеристик:

- End-points – корректність значень;
- Payload структура, або мандаторні парамтри API;
- Облікові данні, для контроля успішності автентифікації, авторизації;
- Характеристики throttling, для контроля кількості запросів;
- Коди помилок.

1.3 Моніторинг безперервної інтеграції/безперервного розгортання (CI/CD)

Безперервна інтеграція (CI) і безперервне розгортання (CD) — це практики розробки програмного забезпечення, спрямовані на автоматизацію та оптимізацію процесу створення, тестування та розгортання змін програмного забезпечення. Ці практики зазвичай називають разом CI/CD.

Хмарні платформи пропонують масштабовані ресурси, що забезпечує швидку розробку та розгортання. Кількість мікросервісів не дозволяє витрачати час на мануальне розгортання та встановлення окремо кожного з

них. CI/CD автоматизує життєвий цикл розробки, дозволяючи часті випуски. Це ідеально узгоджується з масштабованістю хмари, забезпечуючи швидке надання нових функцій і оновлень користувачам. Практики CI/CD гарантують, що додатки можуть легко використовувати переваги хмарної масштабованості. Автоматичне розгортання дозволяє ефективно збільшувати або зменшувати масштаб у відповідь на зміни робочого навантаження.

CI/CD має ключове значення для гнучкого розгортання мікросервісів. Він підтримує швидку та незалежну розробку, тестування та розгортання окремих служб, сприяючи загальній гнучкості архітектури. Зміни інфраструктури розглядаються як код, перевіряються версіями та тестуються разом із кодом додатка, забезпечуючи послідовність і надійність у розгортаннях [10].

Таким чином, синергія між CI/CD і хмарними обчисленнями є невід'ємною частиною досягнення сучасного, ефективного та швидкого життєвого циклу розробки програмного забезпечення. Це поєднання дозволяє організаціям використовувати весь потенціал хмарних платформ, надаючи програмне забезпечення швидко, надійно та в масштабах.

CI/CD є основною практикою DevOps. Він сприяє співпраці між командами розробки та операцій, сприяє спільній відповідальності та заохочує культуру постійного вдосконалення та автоматизації. Вводиться термін CMDB (База даних керування конфігурацією) – База даних керування конфігурацією забезпечує розуміння критичних активів організації та їх зв'язків, таких як інформаційні системи, джерела активів та цілі для підпорядкованих активів [11].

В хмарних рішеннях роль CMDB або бази даних управління конфігурацією важлива і виконує кілька ключових функцій:

1. CMDB служить центральним зберігачем інформації про всі складові конфігурації в хмарному середовищі. Вона включає в себе дані про сервери,

мережеві пристрої, зберігання даних, програмне забезпечення, мережеві з'єднання та інші компоненти.

2. CMDB дозволяє відстежувати залежності між різними елементами конфігурації. Наприклад, вона може показати, які додатки залежать від певних серверів чи які сервіси використовують конкретні мережеві ресурси.

3. CMDB може бути використана для відстеження змін у хмарному середовищі. Вона дозволяє контролювати та реєструвати всі зміни конфігурації, такі як внесення нових ресурсів, модифікації параметрів чи вилучення складових.

4. CMDB може допомагати у відстеженні версій різних компонентів та конфігурацій, забезпечуючи можливість швидко повертатися до попередніх версій або відновлювати конфігурації у випадку проблем.

5. CMDB надає засоби для ефективного планування ресурсів в хмарному середовищі. Це може включати в себе розподіл ресурсів, визначення оптимальних конфігурацій та уникнення конфліктів ресурсів.

6. На основі інформації з CMDB можна розробляти автоматизовані процеси управління конфігурацією, які спрощують рутинні завдання, такі як розгортання нових екземплярів або масштабування ресурсів.

7. CMDB дозволяє проводити аналіз конфігурацій та генерувати звіти для прийняття рішень. Наприклад, можна вивчити завантаження ресурсів, визначити найбільш обтяжені елементи і планувати майбутні інвестиції.

CMDB важлива для ефективного управління конфігураціями в хмарному середовищі, де швидке масштабування та зміни можуть впливати на безпеку, доступність і продуктивність.

1.4 Метрики та сповіщення

Інформаційна панель у контексті моніторингу відноситься до візуального представлення ключових показників ефективності (KPI) для системи, програми або інфраструктури. Інформаційні панелі створюються за допомогою інструментів моніторингу, таких як Prometheus, Nagios, Grafana,

Splunk, що дозволяє користувачам створювати настроювані та інтерактивні інформаційні панелі [12]. Опис поширених типів моніторів, які можна знайти на інформаційній панелі моніторингу нижче (Рис.1.1)[13]:

- Монітор безвідмовної роботи відображає поточний стан безвідмовної роботи критично важливих служб або систем. Ключові показники: відсоток доступності, випадки простою та час після останнього інциденту.
- Монітор показників продуктивності надає інформацію про продуктивність контрольованої системи або програми. Ключові показники: використання ЦП, використання пам’яті, дисковий ввід/вивід і затримка мережі.
- Монітор помилок відстежує кількість помилок або збоїв у додатку чи службі. Ключові показники: відсоток частоти помилок, кількість помилок і відомості про конкретні типи помилок.
- Монітор логування візуалізує журнали та події, створені системою або програмою. Ключові показники: записи журналу, повідомлення про помилки та шаблони, виявлені в журналах.
- Монітор безпеки відстежує події та аномалії, пов’язані з безпекою. Ключові показники: кількість інцидентів безпеки, невдалих спроб входу та сповіщень від систем безпеки.
- Моніторинг використання ресурсів відстежує використання системних ресурсів для виявлення потенційних вузьких місць. Ключові показники: ЦП, пам’ять, дисковий простір і використання пропускної здатності мережі.
- Монітор сповіщень відображає сповіщення, створені системою моніторингу або зовнішніми джерелами сповіщень. Ключові показники: статус сповіщення, приоритет і деталі сповіщень.

- Власний монітор адаптований до конкретних показників або умов, що відповідають потребам організації. Ключові показники: спеціальні показники та показники на основі унікальних вимог.

- Монітор продуктивності бази даних фокусується на продуктивності та справності баз даних. Ключові показники: час виконання запиту, час відповіді бази даних і стан з'єднання.

Рисунок 1.1 Grafana Dashboard монітори [13]

Ці монітори часто відображаються як панелі на інформаційних панелях Grafana, забезпечуючи консолідоване уявлення про стан і продуктивність системи в режимі реального часу. Користувачі можуть налаштовувати макет, вибирати типи візуалізації (наприклад, графіки, датчики або таблиці) і налаштовувати сповіщення (Рис.1.2), які надсилатимуться при виконанні певних умов. [13]

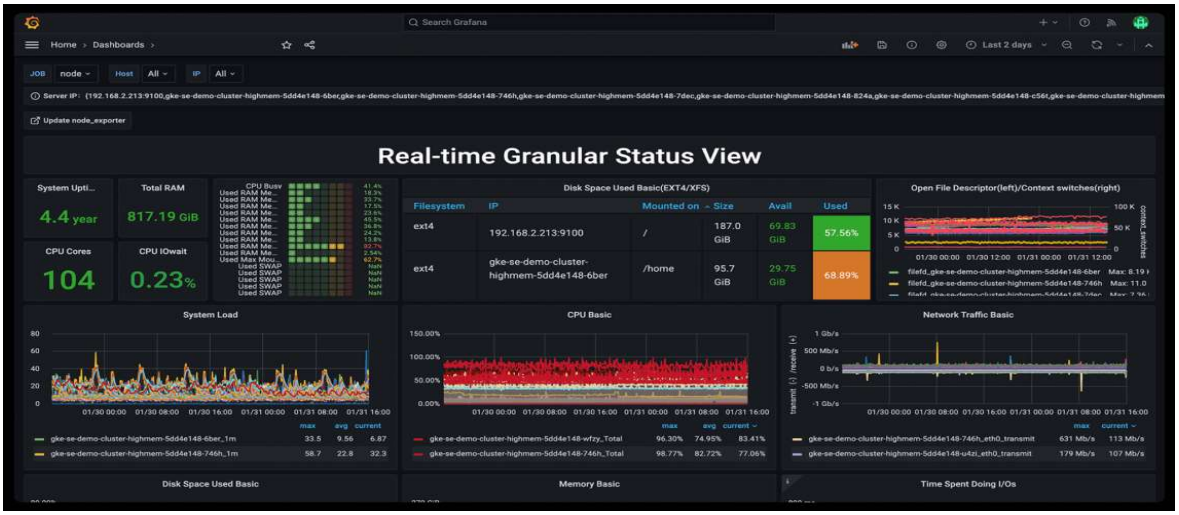


Рисунок 1.2 Grahana Dashboard користувача [13]

1.5 Постановка задачі

Створити комплекс заходів для моніторингу роботи хмарного ентерпрайз рішення, архітектура якого дозволятиме стабілізувати його роботу і отримати мінімальни показники часу простою системи. Результати аналізу та методи вирішення повинні бути візуалізовані у зручному та зрозумілому для кінцевого користувача вигляді.

Даний комплекс повинен складатися з наступних конфігурацій моніторингу:

- Архітектура системи моніторинга хмарного сервіса;
- API моніторінг з зовнішніми сервісами, CMDB валідатор конфігурації після інсталяції;
- Health-Checker зовнішніх інтерфейсів;
- Репорт з метриками та відповідной пріотиризацієй.

Дані для аналізу будемо брати з прикладу конфігурації моніторінгу для OSS/BSS рішення (Operation Support System/Business Support System) для продажу і активації в мережі мобільних ліній, девайсів, підтримки користувачів в магазинах та онлайн, а також біллінг операцій.

2. ВИБІР МЕТОДІВ ВИРІШЕННЯ

2.1 Структурно-функціональне моделювання архітектури моніторингу

Розглянемо діаграму Enterprise Архітектури одного з BSS/OSS рішень для Телеком оператора (Данія), як приклад хмарного рішення. На діаграмі (рис. 2.1) показані основні модулі і мікросервіси кожного модуля, а також тип транспорту комунікацій між мікросервісами. Як показано майже усі сервіси пов’язані між собою і взаємодіють у єдиному процесі закупки послуг, девайсів, організації оплати рахунків і активації послуг у мережі. При такій архитектурі вимоги до стабільності системи дуже жорсткі. Основна складова стабільності – моніторінг критичних процесів і вчасна нотифікація при виявлені ризиків [14].

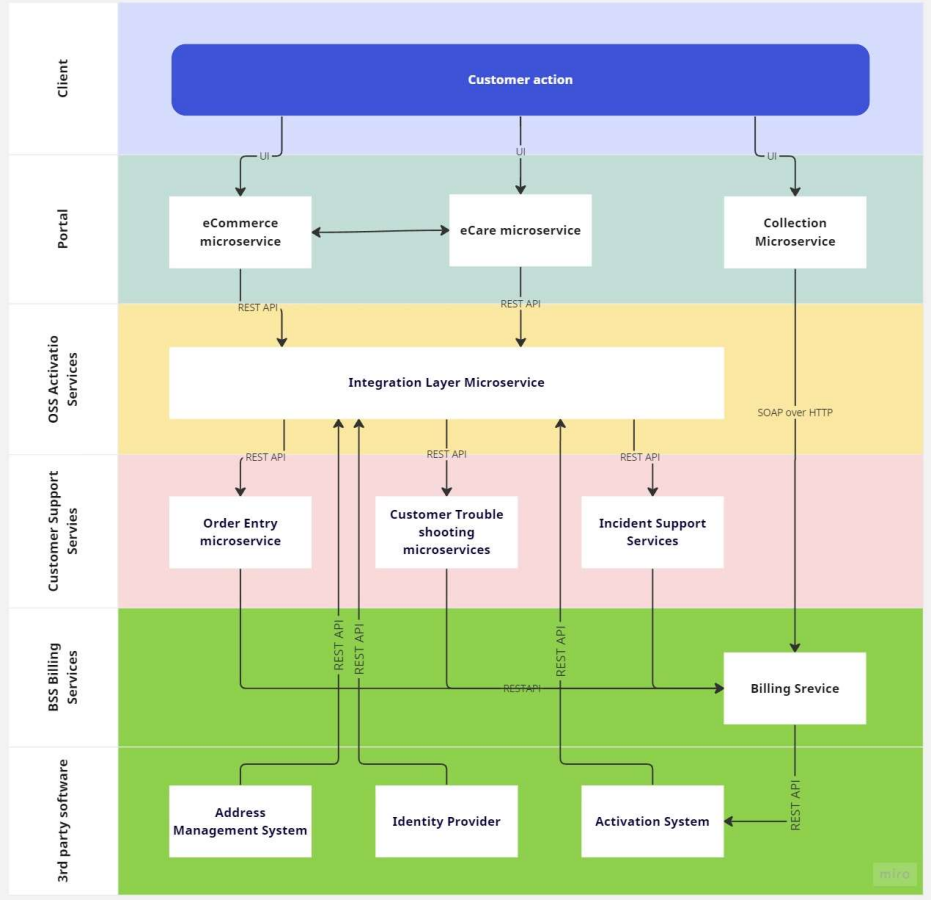


Рисунок 2.1. Архитектура OSS/BSS Telecom Solution

Архітектура системи моніторингу для хмарного рішення розроблена для забезпечення повної видимості стану, продуктивності та безпеки всієї інфраструктури та програм. Використовувані конкретні компоненти та інструменти можуть відрізнятися залежно від вимог і використовуваних технологій.

Функціональний моніторинг хмарного рішення передбачає оцінку поведінки та продуктивності функціональних компонентів системи, щоб переконатися, що вони відповідають запланованим вимогам і забезпечують очікувану функціональність [15]. Структуруємо основні етапи реалізації моніторинга для рішення вище з метою забезпечення безперебійної роботи хмарного сервіса як цілого:

1. Цілі рівня обслуговування (SLO) та індикатори рівня обслуговування (SLI):
    a. Визначити чіткі SLO, які представляють бажані рівні продуктивності та функціональності.
    b. Встановити SLI, які є показниками, які кількісно визначають продуктивність певних функцій.
2. API та моніторинг end-point'ів:
    a. Відстежити стан і продуктивність API і end-point-ів
    b. Відстежити час відповіді, рівень помилок і доступність критичних end-point-ів API.
3. Моніторинг транзакцій:
    a. Впровадити моніторинг транзакцій, щоб відстежувати потік і успіх критичних транзакцій.
    b. Переконатися, що наскрізні процеси функціонують належним чином.
4. Функціональне тестування у виробництві:

a. Виконуйте синтетичні транзакції у виробничому середовищі для імітації взаємодії користувачів.

b. Використовуйте автоматизовані сценарії, щоб перевірити, чи критичні функції працюють належним чином.

5. Моніторинг робочого процесу:

a. Відстежуйте наскрізні робочі процеси, щоб забезпечити безперебійне виконання.

b. Відстежуйте ключові кроки в бізнес-процесах і виявляйте будь-які аномалії або відхилення.

6. Сповіщення та ще раз сповіщення:

a. Налаштуйте сповіщення на основі попередньо визначених порогів для функціональних показників.

b. Отримуйте сповіщення, коли функціональність падає нижче прийнятного рівня.

c. Оберіть валідну приоритизацію сповіщень.

7. Моніторинг помилок:

a. Відстежуйте та реєструйте помилки, які виникають під час взаємодії користувача або системних процесів.

b. Аналізуйте рівень помилок, типи та тенденції, щоб виявити та вирішити проблеми.

8. Моніторинг залежностей:

a. Відстежуйте залежності від зовнішніх служб і API.

b. Переконайтеся, що зміни в залежностях не впливають негативно на функціональність системи.

9. Маштабованість і тестування навантаження:

a. Виконайте навантажувальне тестування, щоб оцінити, як система поводиться за різних рівнів трафіку.

b. Відстежуйте здатність системи масштабувати та обробляти збільшені навантаження, зберігаючи функціональність.

10. Моніторинг доставки вмісту:

a. Якщо можливо, відстежуйте мережі доставки вмісту (CDN), щоб забезпечити ефективне розповсюдження вмісту.

b. Відстежуйте затримку, звернення до кешу та ефективність доставки.

11. Журнали та аудит:

a. Використовуйте журнали та інструменти аудиту для відстеження дій користувачів і системних подій.

b. Переконайтеся, що журнали фіксують відповідну інформацію для аудиту та усунення несправностей.

12. Інформаційна панель і звітність:

a. Використовуйте інформаційні панелі для візуалізації ключових функціональних показників і показників ефективності.

b. Створення звітів про функціональну продуктивність протягом певного часу.

13. Постійне вдосконалення:

a. Збирайте відгуки від функціонального моніторингу, щоб визначити області для покращення.

b. Регулярно переглядайте та оновлюйте конфігурації моніторингу на основі змінних вимог.

14. Безпека:

a. Перевіряйте облікові данні та права доступу.

Ця архітектура гарантує, що система моніторингу надає інформацію в реальному часі, сприяє проактивному вирішенню проблем і підтримує загальну працездатність і безпеку хмарного рішення. Важливо пристосувати архітектуру до конкретних хмарних послуг постачальника, архітектур додатків (монолітів, мікросервісів) і організаційних потреб.

2.2 Аналіз та вибір алгоритмів моніторингу

Звернемо увагу, що безперебійна робота сервісів не означає 100% покриття усіх можливіх ризиків. Головна задача покрити критичні реалізації функціоналу, які можуть мати вплив на більшу кількість бізнес сценарієв [16]. Таким чином зробити коректний фокус на піднятті вірних флагів для найшвидшого вирішення дефектів і проблем солюшена.

Таким чином, сформуємо дорожну карту реалізації моніторингу:

1. API та моніторинг end-point'ів: частина CI/CD моніторинга конфігурації CMDB. Готових рішень на разі не існує, адже конфігурація end-point-ів задача специфічна для кожного сервісу. Створемо еталони значень параметрів, з якими будемо перевірять CMDB параметри та end-point-и після інсталяції нового білда. При виявлені відхилень флаг додається в репорт, рівень відхилення – червоний.

2. Моніторинг транзакцій: комплексне рішеня, логування транзакції роботи API, налаштування роботи на Error only логування. При отриманні Error підняти флаг. Рівень відхилень в залежності від категорії транзакцій. Безпека, Аутентифікація – червоний, Активація і Оплата – Жовтий, інше – Зелений.

3. Функціональне тестування у виробництві: готове рішення відсутнє. Необхідне формування найбільш критичних сценаріїв в кількості 2-3 шт. Реалізація автотестів з окремим репортом. Рівень відхилень – червоний,

використовувати під час CI/CD процесу, одразу після оновлення яких небудь сервісів.

4. Моніторинг робочого процесу: Grafana моніторинг з налаштуванням робочих процесів.

5. Сповіщення та ще раз сповіщення: Grafana Dashboard з вірними рівнями відхилень, окремі репорти під час CI/CD оновлень.

6. Моніторинг помилок. Комплексне рішення з корректними рівнями флагів. Для прикладу, при отримані http 401 відповіді флаг повинен підніматися – червоний рівень відхилення, так як повязаний з автонтефікацією.

7. Інформаційна панель і звітність: формуємо якійсний дашборд та репорти.

2.3 Формування репорту моніторингу CI/CD та HelathChecker.

Формування ефективного репортінгу моніторингу базується на декількох ключових принципах [17], які спрямовані на надання корисної та зрозумілої інформації для прийняття рішень:

1. Обирайте ключові метрики, які найкраще відображають стан системи чи сервісу. Це дозволяє уникнути інформаційного перенавантаження та надає основний фокус на важливих показниках.

2. Використовуйте графіки, діаграми, таблиці, кольорові виділення та інші візуальні засоби для легшого сприйняття інформації. Інтерактивні дашборди дозволяють користувачам взаємодіяти з даними.

3. Розумійте, хто буде користувачем репортів. Адаптуйте формат і зміст для аудиторії, забезпечуючи потрібний рівень деталізації та максимально зрозумілий зміст.

4. Визначте заздалегідь, які показники чи умови слід вважати критичними і вигідними для репортування.

5. Надайте контекст до представлених даних. Інформація має бути зрозумілою та корисною, і контекст допоможе відвідувачам розуміти, як вони можуть взаємодіяти з даними.

6. Порівнюйте поточні дані з минулим періодом часу. Це допомагає виявляти тенденції, а також оцінювати ефективність вжитих заходів.

7. Використовуйте автоматичне збирання та відображення даних для зменшення помилок та забезпечення актуальності.

8. Дотримуйтесь Принципів "Single Source of Truth". Гарантуйте, що дані в репортах однакові та відображають правдивий стан системи. Уникайте відмінностей в інформації з різних джерел.

10. Слідкуйте за тим, як користувачі використовують репорти та вносьте зміни відповідно до їхніх потреб.

Важливо підкреслити, що правильно налаштований моніторинг та ефективний репортінг є критичними компонентами для забезпечення стабільності, доступності та продуктивності систем та сервісів. Розглянуті принципи є фундаментальними кроками у напрямку створення інструментів, які надають користувачам (в тому числі адміністраторам, розробникам та керівникам) необхідну інформацію для ефективного управління та вирішення завдань.

Моніторинг дозволяє виявляти проблеми та тренди, забезпечуючи можливість передбачення та попередження можливих відмов. Репортінг, у свою чергу, перетворює накопичені дані у зрозумілу, візуальну інформацію, яка допомагає приймати обгрунтовані рішення та визначати стратегії подальшого розвитку.

Ретельно вибрані метрики, правильно сформований звіт та врахування особливостей аудиторії дозволяють створити ефективні та корисні

інструменти для моніторингу та управління. Автоматизація збору даних та виведення звітів сприяє швидкому реагуванню та вирішенню проблем [18].

Загальною метою є створення екосистеми, де моніторинг та репортінг стають не просто інструментами спостереження, але й ефективними засобами для оптимізації роботи та досягнення стратегічних цілей.

3. РЕАЛІЗАЦІЯ

3.1 Огляд загальної архітектури системи моніторинга

Мета створення комплексу для моніторингу та аналізу ентерпрайз хмарного рішення – це отримання стабільної самодостатньої системи для вчасної нотифікації при виявленні критичних проблем системи, попередженню інфраструктурних проблем та виконання базової аналітики критерієв стабільнотсі роботи системи. Результати аналізу та роботи моніторингу повинні бути подані та візуалізовані у зручному та зрозумілому для кінцевого користувача вигляді.

Компекс має багатокомпоненту структуру [19], кожна компонента відповідає за покриття ключових етапів роботи системи:

- CI/CD моніторинг – перевірка конфігураційних параметрів всіх зовнішніх та внутрішніх інтерфейсів і порівняння значень CMDB із значеннями еталонів;
- Healhchecker – перевірка доступності зовнішніх контрактних інтерфейсів;
- Integration tracker – перевірка статусу інтеграцій, будемо використовувати Grafana monitor dashboard;
- Пріоритезація нотифікацій – розрозбка правил нотифікацій з системою пріоритетів для саппорт команди.

Таким чином покриємо усі зв’якзки між мікросервісами в середині хмари і всі зв’язки із зовнішніми інтерфейсами (рис.3.1).

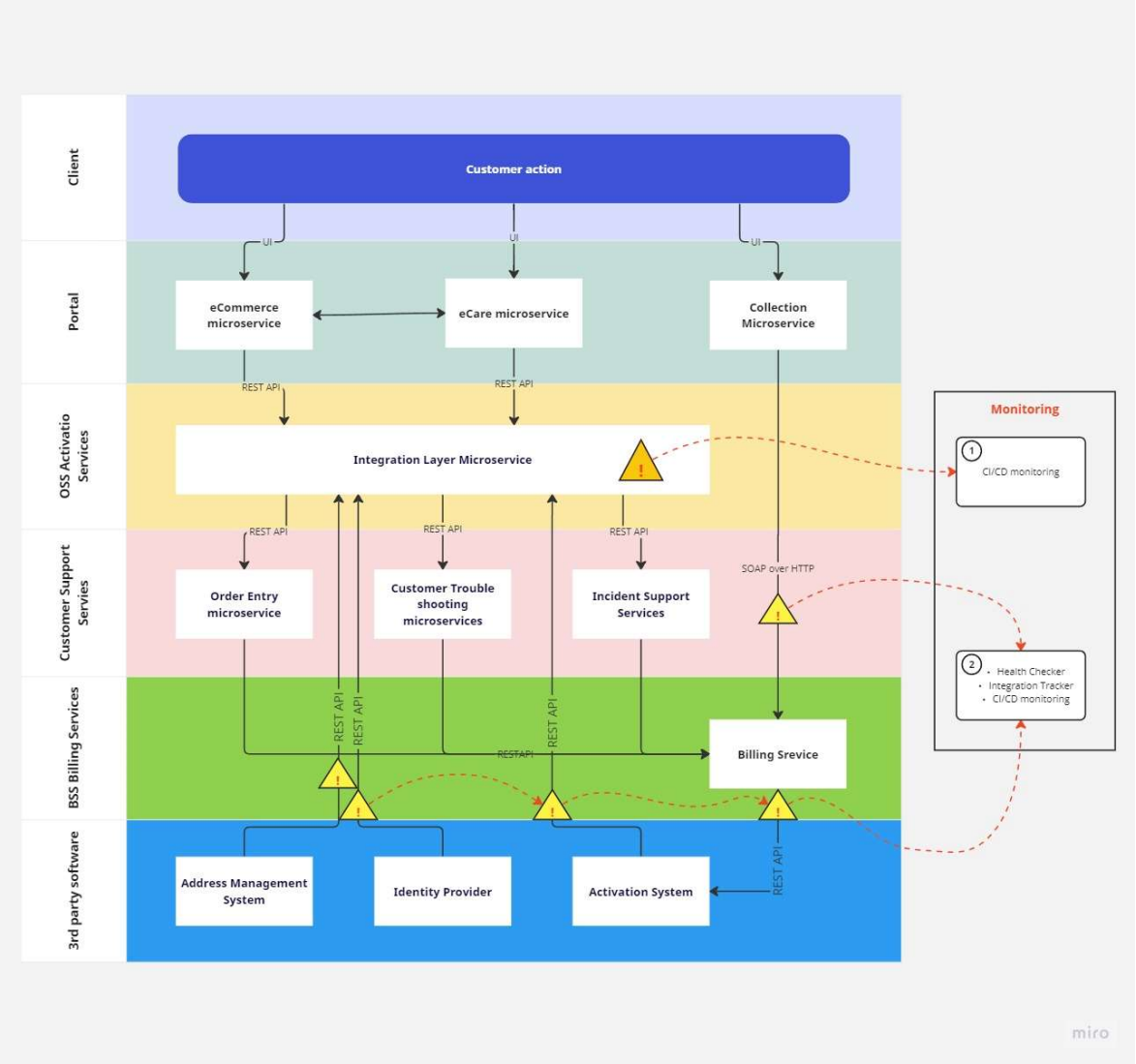


Рисунок 3.1 Моніторинг інтерфійсів в хмарі

Пояснення для вище зображеної схеми (Рис. 3.1).

Розділемо рішення для якого робимо моніторинг на 6 умовних шарів (виділені різним кольором):

1 шар – клієнт та його дії, маніпуляції і робота користувача системи;

2 шар – клієнтський портал, веб-застосунок або веб-сайт, який забезпечує користувачам централізований доступ до різноманітних сервісів, інформації чи можливостей. Простий інтерфейс для доступу до різних послуг або розширене середовище для співпраці та обміну інформацією.

3 шар – інтеграційний. Компонента системи, призначених для об'єднання і взаємодії різних частин системи чи окремих систем. Цей шар використовується для забезпечення сприятливої взаємодії між різнорідними компонентами, мікросервісами та зовнішними додатками або сервісами, незалежно від того, чи вони розташовані в межах однієї системи, чи вони є частинами різних систем. Інтеграційний шар допомагає забезпечити гнучкість, масштабованість та легкість обслуговування системи, дозволяючи їй ефективно взаємодіяти з різними компонентами, сервісами чи зовнішніми системами. Цей мікросервіс являє собою кровяну систему хмарного рішення. Забезпечує різноманітні протоколи та механізми зв'язку для обміну даними між різними компонентами системи. Використовуються для забезпечення асинхронної комунікації та передачі повідомлень між компонентами без прямого їхнього зв'язку. Забезпечує механізми безпеки для збереження конфіденційності і цілісності даних, передаваних між компонентами. Включає в себе сервіси реєстрації компонентів, які можуть бути доступні для інших, і відкриті API для взаємодії з системою. Максимально важливо забезпечити безперервну роботу сервісу і налаштувати моніторинг для найшвидшого вирішення проблем. Сфокусуємось на можливості покриття моніторингу CI/CD та формування правил нотифікацій та інтгераційного монторинга [20].

4 шар – сапорт сервіс. Системи підтримки роботи всього рішення. Вся робота мікросервісів організована через інтеграційний шар, або напряму з зовнішніми системами. В нашому випадку організуємо моніторинг інтеграційного шару і кожного зонвішнього інтерфейса окремо.

5 шар – білінговий сервіс. Відповідає за облік та розрахунок витрат або платежів у контексті певного бізнесу чи послуги. Цей сервіс грає ключову роль в управлінні фінансами та обліком вартості використання різних ресурсів. Роботи данного сервіса асинхронні і використовує файлові інтерфейси обміну данними.

6 шар – зовнішні допоміжні системи. В залежності від локації і покриття мікросервісами ці системи можуть підтримувати різноманітний функціонал: адресні, активаційні, ідентифікаційні сервіси, сервіси оплати, тощо. Складності роботі кожного сервісу ще додає індивудальний контракт для кожного інтерфейсу: різноманітний транспорт, ауторизація, формат даних, шифрування. До моніторингу зовнішніх інтерфейсів підійдемо комплексно. По-перше, організуємо перевірку доступності інтерфейсів (healthchecker) для найшвидшої нотифікації при відсутності з'єднання, або зміни паролів доступу, або протухання сертифікатів. По-друге, сформуємо чіткі правила приоритезації нотифікацій. Зробимо двохрівневую пріоритизацію: за важливістю зовнішньої системи в рамках бізнес процесів, наприклад активаційний сервіс буде пріоритетнішим за сервіс друкування інвойсів, та за рівнем роботоспроможності інтерфейса, якщо помилка одноразова, то пріоритет буде нижчим за постійну, або за відстуність доступа до зовнішнього інтерфейса (помилки з'єднання, аутинтефікації). По-третє, сконфігуремо моніторинг CI/CD, адже запобігання помилки набагато дешевше виправленню. Сконфігуруємо еталон end-point-ів, паролів, ключових параметров, з яким в автоматичному режимі буде перевірка після інсталяції наступного реліза на хмарне рішення і до запуску в продакшн режимі.

3.2 CI/CD моніторинг. CMDB параметри.

Збірка конфігурації CMDB є складним процесом, який включає конвеєр gitlab, конструктор середовища і хмарний конструктор. Для спрощення існує так званий конвеєр візуалізації (рис.3.2). Також важливо розуміти, що в репозиторії шаблонів CMDB немає атвоматичної фіксації, а збирання конфігурації CMDB для певного середовища завжди слід викликати вручну [21].

Пряме підключення до головної гілки репозиторію cmdb-templates заборонено. Щоб внести будь-які зміни, створюється окрема гілка з master, вносяться необхідні зміни, робиться запит на merge (злиття) та об’єднується з master. Merge дозволено лише для успішних конвеєрів merge запросів.

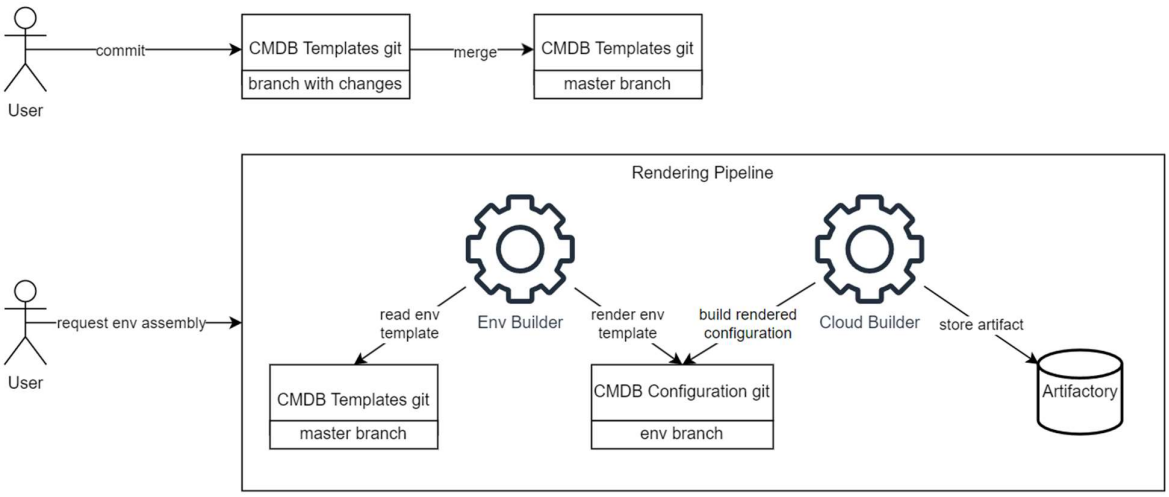


Рисунок 3.2. Конвеєр візуалізації.

Імпорт артефакту наборів параметрів — це операція, яка застосовується до всього клієнта та впливає на всі хмари та середовища цього клієнта і може вплинути на інші середовища, якщо їх декілька у відповідному клієнті та хмарі. Таким чином кожен конфігураційний параметр має версію і ці зміни можно відмінити. Але і робити контроль за змінами дуже важко, враховуючи велику кількість мікросервісів і зонвішних інтерфейсів.

Розглянемо приклад встановлення програми і імпорту параметрів CMDB в хмару і в deployment мікросервіса.

Щоб установити програму, збірку потрібно завантажити в середовище, куди потрібно імпортувати CMDB. Після завантаження артефакту в середовище потрібно налаштувати програму для цього артефакту (рис.3.3)

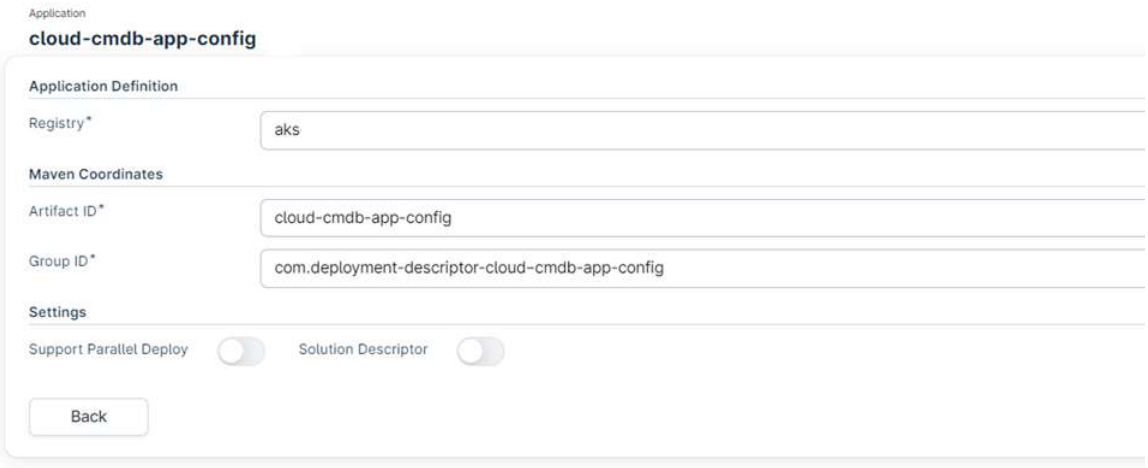


Рисунок 3.3 приклад cloud-cmdb-app-config.

Перевіремо, що параметрам інтерфейсу №1 зроблені, відкриваємо конфігурацію інтерфейсу з флагом “Show Parameters”. Необхідно впевнитися, що параметри примінилися (рис. 3.4).

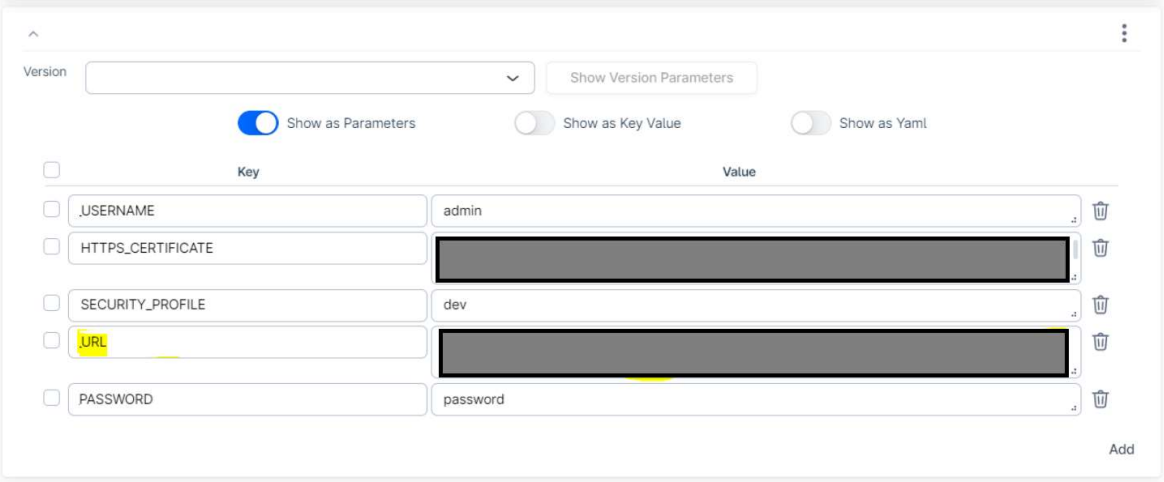


Рисунок 3.4 CMDB конфінурація інтерфейс №1.

Запускаємо імпорт параметрів, таким чином усі параметри розілються по ключовим значенням на мікросервіси нашого хмарного рішення (рис. 3.5).

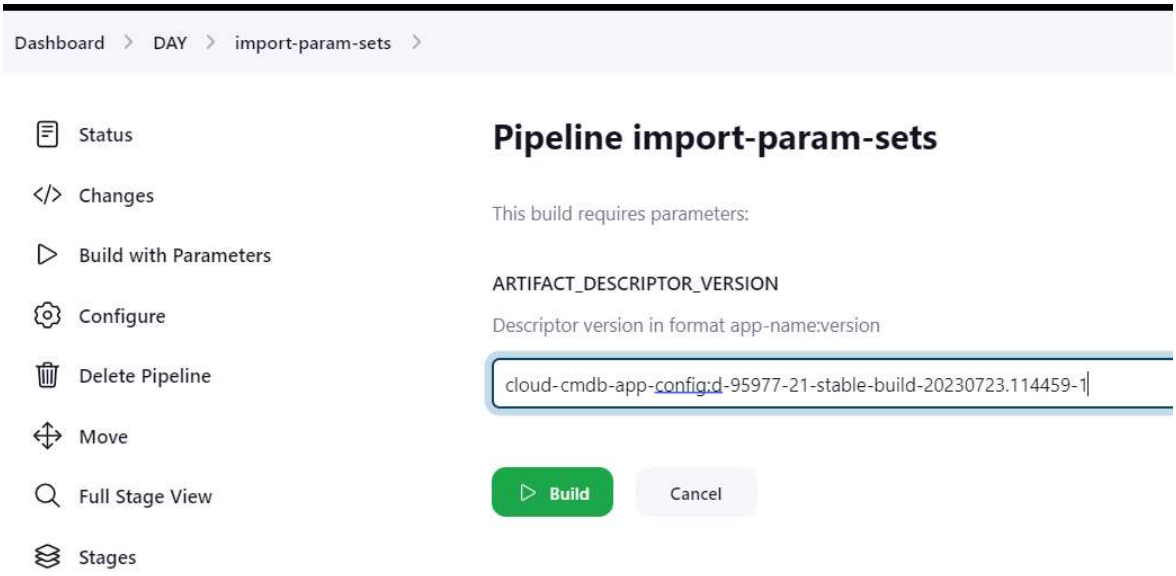


Рисунок 3.5. Pipeline import-param-sets

Навігуємось в конфігуратор деплоймента і секретів мікросервісу, параметри вдало проставились.

Таким чином вся конфігурація середовища та взаємодії між мікросервісами являється частиною CMDB. Найбільш ефективними є два варіанти моніторингу: перевірка CMDB конфігурації до інсталяції, або перевірка CMDB конфігурації після інсталяції середовища, отримавши параметри безпосередньо з конфігурації деплоймента та секретів.

3.3 CI/CD моніторинг. Формування Еталону.

Так як CMDB представляє собою базу даних елементів конфігурації (CI) та їх зв'язок з інфрастукутрою мікросервісів і зовнішніх інтерфейсів, таких як активаційні сервіси, телевізійні сервіси, сервіси локацій (google map) та ін. Для того, щоб зрозуміти які параметри будуть ключовими для моніторингу виявимо можливий транспорт між сервісами і основні характеристики. В нашому випадку ентерпрайз телеком рішення забезпечує можливість активації мобільного, звязку, інтернету, сервісів ммс, прослуховування музики, підключення сервісів нетфлікс, дісней, перевірки кредоспроможності

клеєнта, оплати карткою, за допомогою банківського рахунку та ще декілька сервісів, які підтримують існуючі оператори зв'язку [22].

Виділемо 4 группи зовнішніх і внутрішніх інтерфейсів:

1. API інтерфейси між мікросервісами і REST API зовнішних сервісів. Мабудь найполпулярніший транспорт. Простий в імпелементації, легко масштабується та дозволяє підтримувати високу швидкість передачі даних, так як використовується json формат [23]. Підтримка сучасних технологій аутентифікації/авторизації, як oAuth 2.0 [24];
2. SOAP over HTTP зовнішні інтерфейси. Транспорт не такий гнучкий, як REAT API, але має перевагу в тому, що стандартизован [25]. Аторизація зазвичай проходе через параметри в тілі header's, які називаються тег.
3. Файлові інтерфейси. Зазвичай використовуються для передачи великої кількості інформації, як платежі, деталі по користуванню сервісам
4. Редіректи, або авторизаційні інтерфейси. Використовуються для переміщенням між UI сервісами із збереженням деталей користувача, щоб отримати плавний перехід від однієї системи до іншої.

Групи основних параметрів в залежності від інтерфейсів представимо в таблиці 3.1

Таблиця 3.1 Типи інтерфейсів і транспорт

#	Тип інтерфейу	End-points формат	Деталі Аутентифікації	Транспорт
1	FileBased	sftp://127.0.0.1/upload	ssh key	SFTP
2	FileBased	ftp://127.0.0.1/upload	login/password	FTP
3	FileBased	ftp://127.0.0.1/upload	login/password	SFTP

4	SOAP over HTTP	https://url.com/resource	login/password in the header TLS1.2	https
5	SOAP over HTTP	https://url.com/resource	login/password in the header	http
6	REST API	https://url.com/resource	Basic authorization TLS 1.2	https
7	REST API	https://url.com/resource	Basic authorization	http
8	REST API	https://url.com/resource	oAuth 2.0	http
9	REST API	https://url.com/resource	oAuth 2.0 TLS	https

Збираємо еталон вищезгаданих параметрів для кожного інтерфейсу. В нас вийшло 9 групп порівняння. В залежності від типу end-point-ів та авторизації, виводимо наступні конфігурації еталонів: SFTP+SSH, SFTP+Login/Password, SOAP over HTTP та REST API+Basic Auth, REST API+oAuth.

SFTP+SSH. Використання ключів SSH для SFTP (протокол безпечної передачі файлів) додає додатковий рівень безпеки, увімкнувши автентифікацію на основі ключів замість того, щоб покладатися виключно на паролі. Це налаштування передбачає використання OpenSSH, найпоширенішу реалізацію SSH.

Приклад з'єднання:

ssh -i /path/to/private-key username@hostname

Таким чином в нас наступні параметри для еталону: username, hostname ssh-key.

SFT+Login/Password. Варіант з індивідуальним паролем вважається меньш захищенни, але деякі інтерфейси все ще продовжують використання данного типа зєднання.

ssh username@hostname/password

таким чинов в нас наступні параметри для еталону: **username, hostname, password**

SOAP over HTTP та REST API+Basic Auth. Авторизація в протоколі SOAP (Simple Object Access Protocol) зазвичай використовується для захисту доступу до веб-сервісів. Є кілька способів реалізації авторизації в SOAP:

HTTP Basic Authentication: Можна використовувати заголовок HTTP Authorization для передачі інформації про авторизацію. Зазвичай це використовується разом із стандартом HTTP Basic Authentication, де логін та пароль закодовані та включаються у заголовок запиту, або SOAP Headers зі спеціальними елементами безпеки (WS-Security). Але обидва варіанти будуть мати спеціальні теги з для авторизації. Аналогічний механізм існує і для REST API (табл.3.2), тому об’єднаємо конфігурацію еталона, як для Basic авторизації.

Отримуємо наступні параметри для конфігурації еталону:

Client_URI, Login, Password.

Таблиця 3.2 Приклади авторизаційних хедерів

Тип авторизації	Приклад Авторизаційних хедерів
SOAP over HTTP Basic Authentication	<soapenv:Header> <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"> <wsse:UsernameToken> <wsse:Username>your_username</wsse:Username> <wsse:Password>your_password</wsse:Password> </wsse:UsernameToken>

	</wsse:Security> </soapenv:Header> <soapenv:Header>
SOAP over HTTP ws security	<wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext"> <wsse:UsernameToken> <wsse:Username>your_username</wsse:Username> <wsse:Password>your_password</wsse:Password> </wsse:UsernameToken> </wsse:Security> </soapenv:Header>
REST API Basic Authorization	POST 'https://rspcom.pidemia.io/gsma/rsp/es2plus/confirmOrder' \--header 'Authorization: Basic cm9vdDptYXZlbmly'

REST API+oAuth. OAuth 2.0 (Open Authorization 2.0) - це протокол авторизації, який надає стандартні механізми для інтерактивного отримання доступу до ресурсів в інтернет-сервісах від інших сервісів або додатків. Його основною метою є дозвіл користувачам надавати обмежений доступ до своїх ресурсів (наприклад, профілю, фотографій чи контактів) іншим сервісам без передачі свого пароля [24].

В данному випадку авторизація виконується в два етапи. Спочатку йде аутентифікація, клієнт отримує доступ до авторизаційного сервера та отримує дозвіл від ресурсу власника. За валідного дозволу, клієнт обмінює авторизаційний код на токен доступу та, можливо, токен оновлення. Клієнт використовує токен доступу для отримання доступу до ресурсів на сервері ресурсів (Таблиця 3.3). Токен, або JSON Web Token (JWT) - це стандарт відкритого формату [26], який визначає спосіб безпечного передавання інформації між двома сторонами у форматі JSON. JWT може використовуватися для автентифікації та передавання структурованої інформації між сторонами. Токен отримується в зашифрованому форматі і може мати в собі параметризовані данні, для отримання доступу.

JWT складається з трьох частин:

- Header (Заголовок) який вказує тип токена та алгоритм шифрування, які використовуються для підпису чи шифрування токена. Заголовок закодований в Base64Url.

- Payload (Навантаження): містить параметризовану інформацію (корисну для застосунка) та стандартні дані. Призначене для передачі інформації між двома сторонами. Payload також закодований в Base64Url.

- Signature (Підпис): створюється на основі заголовка та навантаження з використанням секретного ключа (який відомий тільки серверу). Підпис додається до токена і використовується для перевірки цілісності та автентичності токена.

Такий токен може бути використаний для передачі інформації про користувача або авторизації у безпечний спосіб між клієнтом і сервером. Важливо зазначити, що вміст токена може бути декодований, але підпис забезпечує його цілісність, і будь-яке змінення токена буде помітно.

Отримуємо наступні параметри для конфігурації еталону:
Client_URI, Client_id, Client_secret.

Таблиця 3.3

Крок авторизації	Приклад Авторизаційних хедерів
Аутентифікація	curl --location --request GET 'https://login.microsoftonline.com/cf5f5248-9b41-11ee-b9d1-0242ac120002/oauth2/v2.0/token' \ --header 'Content-Type: application/x-www-form-urlencoded' \ --data-urlencode 'client_id=f4529e16-9b41-11ee-b9d1-0242ac120002' \ --data-urlencode 'client_secret=fb2b604c~9b4111eeb9d10242ac120122' \

	--data-urlencode 'grant_type=client_credentials' \ --data-urlencode 'response_type=token' \ --data-urlencode 'scope=/.default'
Отримання токена	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9. eyJzdWIiOiAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpvaG4gRG9lIiwgImlhdCI6IDE1MTYyMzkwMjJ9. SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
Запрос з авторизаційним хедером	curl --location 'https://public-gateway-sample.url.com/api/graphql-server/graphql' \ --header 'Content-Type: application/json' \ --header 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiAiMTIzNDU2Nzg5MCIsICJuYW1lIjogIkpvaG4gRG9lIiwgImlhdCI6IDE1MTYyMzkwMjJ9.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c' \

3.4 CI/CD моніторинг. Реалізація.

Є два варіанти перевірки CMDB параметрів з еталонами: перевірка власне параметрів до початку інсталяції, це нам дасть змогу впевнитися раніше, якщо параметри не валідні та зробити необхідні корегування, та перевірка після інсталяції, тут моніторинг дасть інформацію пізніше, але це будуть значення в деплойменті енвайренмента, що дасть нам більш прозору інформацію, чи конфігурація валідна, чи ні.

Для нашого рішення обираємо другий варіант, робимо перевірку на енвайренменті, як більш точну перевірку.

В конфігурації мікросервіса можемо виділити 3 артефакта які нам знадобляться для реалізації функціоналу моніторинга: pod (власне піднятий

мікросервіс), deployment (конфігурація мікросервіса), secrets (сертифікати, паролі доступу) (Рис.3.3).

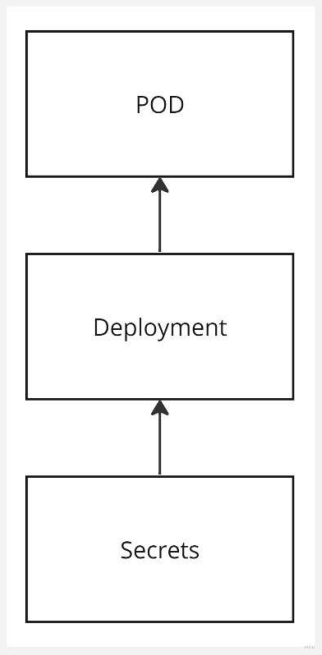


Рисунок 3.6 Артефакти мікросервіса.

Для отримання параметрів з сервера використаємо REST API запрос по namespace pod в deployment для отримання інформації по сконфігурованим end-point'ам та в secrets для отримання інформації по сконфігурованим секретам. На рисунках 3.4, 3.5 представлені параметри, які нам треба порівняти з еталонами.

Приклад коду для передачі і аналізу сформованих реквестів і порівняння з еталонами в Додатку А.

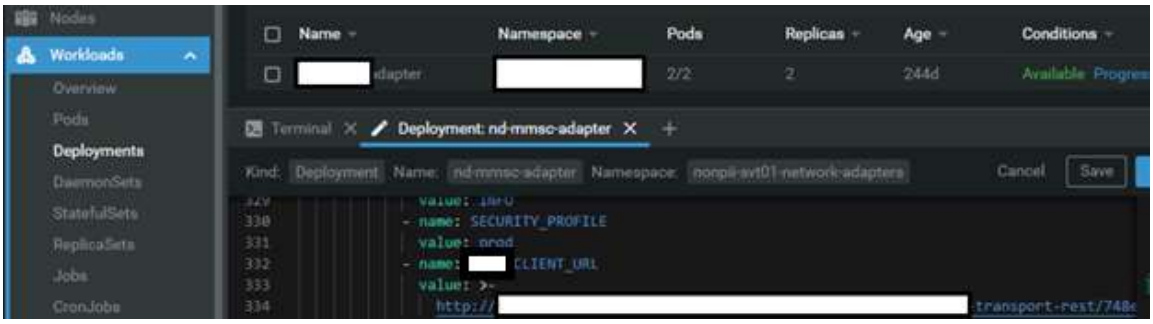


Рисунок 3.7. Deployment конфігурація

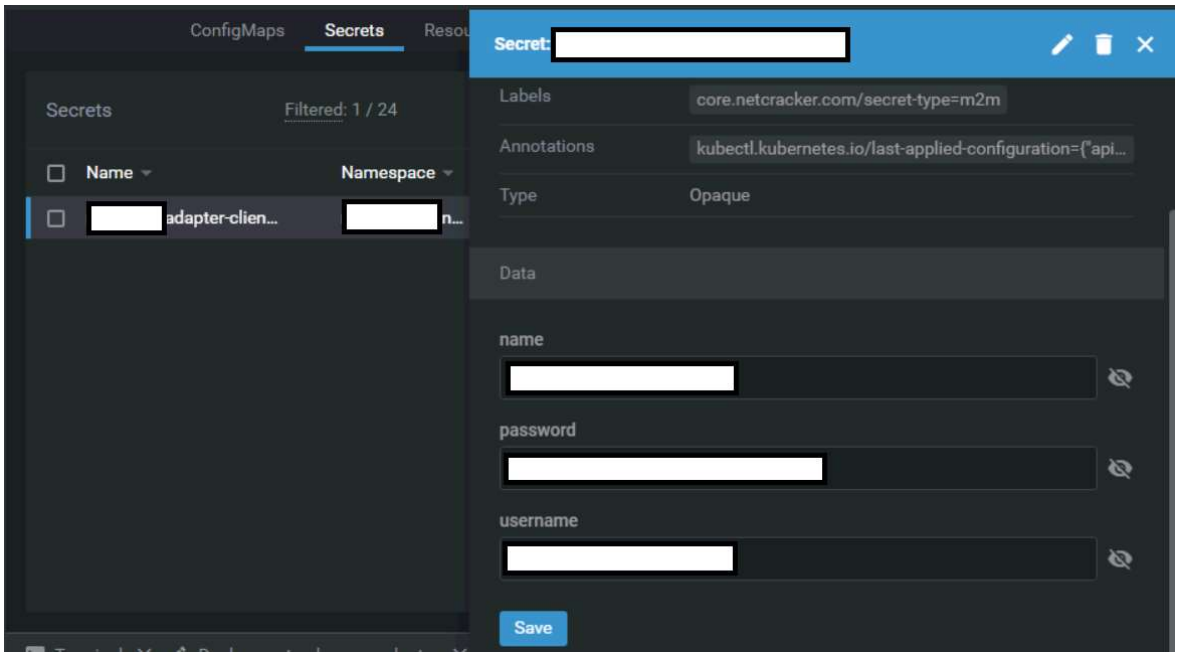


Рисунок 3.8. Secrets конфігурація

Результи перевірки формуємо в репорт в автоматичному режимі, котрий буде надсилатися для аналізу сапортній команді. Частота запуску після кожного оновлення серверу. Репорт повинен складатися з наступних деталей:

- Тема листа повинна містити назву теста і дату виконання;
- Перелік інтерфейсів і статус ОК або NOK;
- У випадк NOK деталі у вигляді посилання на помилку.

Доступ до результатів моніторингу має мати тільки кваліфікований персонал з відповідним рівнем доступу, тому деталі виконання перевірки не повинні бути в репорті, тільки посилання для перегляду.

Приклад репорту можете переглянути в Додатку Б.

3.5. Healthchecker зовнішніх систем

У світі постійно зростаючих технологій та динамічного хмарного середовища, де системи піддаються постійним змінам та масштабуванню,

розробка та впровадження ефективного health checker для зовнішніх інтерфейсів є критично важливим завданням.

Цей механізм не тільки забезпечує стабільність та надійність системи, але й дозволяє оперативно реагувати на можливі проблеми, забезпечуючи максимальну доступність для кінцевих користувачів. Перевірка доступності, затримки, пропускної здатності та безпеки стає ключовим елементом стратегії управління конфігураціями та утриманням здоров'я систем.

Автоматизоване відновлення та реагування на помилки підсилюють рівень автономності системи, зменшуючи час простою та забезпечуючи безперебійну роботу. Забезпечення безпеки та відповідність стандартам гарантують, що інтерфейси залишаються надійними та відповідають вимогам.

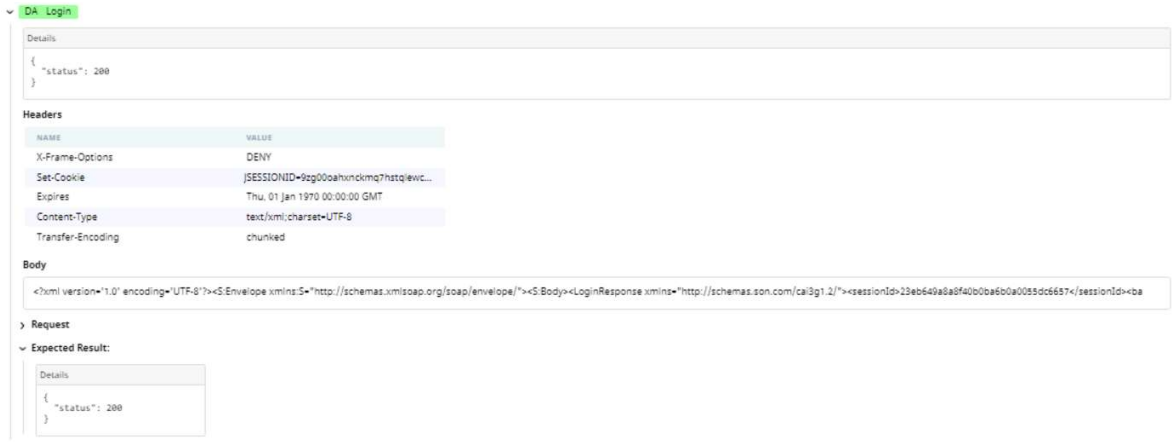
Ось у світі швидких змін та вимогливих користувачів, health checker стає не просто інструментом технічного моніторингу, але стратегічним елементом для досягнення високої продуктивності та конкурентоспроможності в сучасному IT-ландшафті.

Головна причина імплементації helthchecker зовнішніх інтерфейсів – це неможливість контролювати їх роботоспроможність при максимальній залежності цілого рішення від них. Для найшвидшої ідентифікації недоступності і найшвидшої нотифікації для вирішення дефектів з 3-ї сторони.

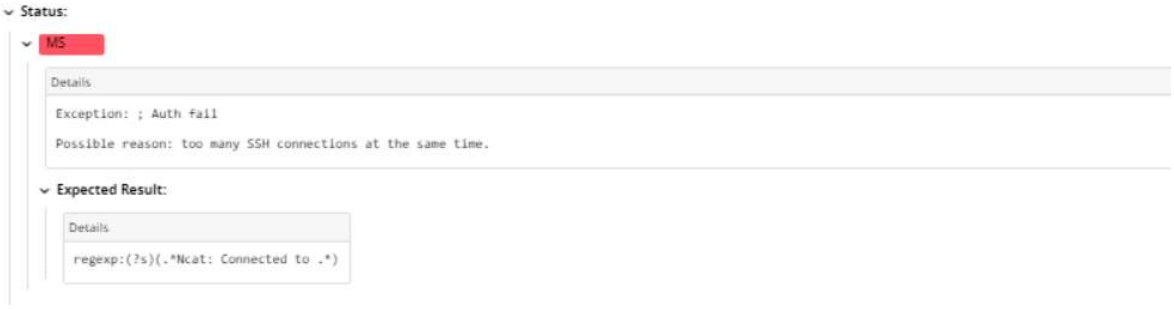
Формуємо правила для перевірки зовнішніх систем. Головна ідея – не нашкодити, перевірка повинна бути швидка, не навантажувати середовище і не використовувати тест дату. Найбільш відповідний варінт до цих вимог, який можемо масштабувати незалежно від транспорту інтерфейсу – перевірка з’єднанання між системами в мережі. Є декілька варіантів перевірки: прості http REST запроси, які можно реалізувати за допомогою команд curl [27], команди передачи пакетів данних ping, telnet, netcat [28], або створення шифрованого з’єднання через ssh (Secure Shell) [29]. В залежності від доступу до зовнішнього інтерфейсу можемо використовувати комбінацію з вищезгаданих команд.

Формуємо запроси та очікуванні відповіді в разі успіху, Expected Result (ER) і додаємо обробку результатів команди, якщо результат відмінний від відповіді підкреслюємо червоним та додаємо в репорт.

Варіанти позитивних і негативних результатів перевірки показані на рисунках 3.9,3.10.



Рисунко 3.9 позитивна сценарій healthcheck, REST API



Рисунко 3.9 негативний сценарій healthcheck, SSH

Результати роботи healthcheck оформлюємо в автоматизовний репорт. Частота запуска – індивідуальне рішення для кожної системи, в залежності від стабільності системи. На разі встановлюємо кожні 4 години, але важливо мати засоби або механізми, які можуть динамічно адаптувати частоту healthchecks в залежності від обставин (наприклад, збільшуючи частоту під час періодів великої активності або автоматично зменшуючи її під час періодів невеликої активності).

Варіант репорту можливо переглянути в додатку Б.

ВИСНОВКИ

В роботі були розглянуті існуючі підходи до моніторингу в хмарних рішеннях, а також ретельно проаналізували системи моніторингу, визначивши інтеграційний моніторинг як ключовий компонент контролю функціоналу в хмарних рішеннях. Проведений аналіз дозволив виявити слабкі сторони існуючих систем та визначити перспективи їх поліпшення за допомогою додаткового покриття методами прогностичної валідації конфігурації мікросервісів і систематичної перевіркі доступності сервісів за допомогою функціоналу healthchecker .

Спроектовано інформаційну систему моніторингу хмарних додатків із мікросервісною архітектурою, яка дозволятиме стабілізувати його роботу і отримати мінімальні показники часу простою системи.

Розроблено архітектуру комплекса моніторинга.

Реалізовно інтеграційний моніторинг, який складається з API моніторинга зовнішніх та внутрешніх сервісів, CMDB валідатор конфігурації end-point'ів та параметрів доступу мікросервісів після інсталяції систем. Також реалізовано функціонал healthchecker'а для перевірки доступності зовнішніх систем інтегрованих з розглянутим хмарним рішенням. Реалізація виконена за допомогою ланцюгових запитів REST API, telnet, netcat, ssh виконаних в визначеній послідовності та з додатковою перевіркою результатів.

Приклади репорту моніторинга з переліком інтерфейсів, статусом і результатом перевірки представлені в додатках Б, В.

Мета та поставлені задачі для кваліфікаційної роботи досягнуті та виконані у повному обсязі.

СПИСОК ЛІТЕРАТУРИ

1. Microservice Architecture, [Електронний ресурс]. // Режим доступу: https://www.fiorano.com/platform?utm_source=gads&utm_medium=search&utm_campaign=WebsiteTraffic+-+Search+Ads+%28%2723%29+-+Fiorano%3A+Q3&utm_term=microservices&utm_content=ad1&gclid=Cj0KCQiAsburBhCIARIsAExmsu5OSX6A3cY7ywC9rmPwGx_JE3TMnQwz6sHBV9nwWhCqrgVqiKO1YesaAjrJEALw_wcB

2. Almeida M. G. de, Canedo E. D. Authentication and Authorization in Microservices Architecture: A Systematic Literature Review // Applied Sciences (Switzerland). 2022. Т. 12. №

3. 13 cloud monitoring tools to ensure optimal cloud performance and drive business success, [Електронний ресурс]. // Режим доступу: https://www.digitalocean.com/resources/article/cloud-monitoring-tools

4. Arie Bregman, 6 Best Practices for Effective Monitoring Alerts [Електронний ресурс]. // Режим доступу: https://medium.com/@bregman.arie/6-best-practices-for-effective-monitoring-alerts-a585bfc0d830

5. Panel: the Correct Number of Microservices for a System Is 489. [Електронний ресурс]. // Режим доступу https://infoq.com/presentations/number-microservices-system/

6. Sinde S. P. и др. Continuous Integration and Deployment Automation in AWS Cloud Infrastructure // Int J Res Appl Sci Eng Technol. 2022. Т. 10. № 6.

7. Morabito R., Kjällman J., Komu M. Hypervisors vs. lightweight virtualization: A performance comparison // Proceedings - 2015 IEEE International Conference on Cloud Engineering, IC2E 2015. : Institute of Electrical and Electronics Engineers Inc., 2015. С. 386–393.

8. Surwase V. REST API Modeling Languages -A Developer's Perspective // IJSTE -International Journal of Science Technology & Engineering. 2016. Т. 2. № 10.

9. Soni A., Ranga V. API features individualizing of web services: REST and SOAP // International Journal of Innovative Technology and Exploring Engineering. 2019. Т. 8. № 9 Special Issue.
10. Khan M. O. Fast Delivery, Continuously Build, Testing and Deployment with DevOps Pipeline Techniques on Cloud // Indian J Sci Technol. 2020. Т. 13. № 5.
11. CMDB configuration, [Електронний ресурс]. // Режим доступу: https://web.archive.org/web/20210128121541/https://searchdatacenter.techtarget.com/definition/configuration-management-database
12. Febriana R. M. Implementasi Sistem Monitoring Menggunakan Prometheus Dan Grafana // Seminar Nasional Telekomunikasi dan Informatika (SELISIK 2016). 2020. Т. 13. № 1.
13. Grafana tool from Grafana labs, [Електронний ресурс] // Режим доступу: https://grafana.com/
14. Sahu M. L. и др. Cloud-Based Remote Patient Monitoring System with Abnormality Detection and Alert Notification // Mobile Networks and Applications. 2022. Т. 27. № 5.
15. Alcaraz Calero J. M., Gutiérrez Aguado J. Comparative analysis of architectures for monitoring cloud computing infrastructures // Future Generation Computer Systems. 2015. Т. 47.
16. Marques G. и др. Proactive resource management for cloud of services environments // Future Generation Computer Systems. 2024. Т. 150.
17. Chhetri T. R. и др. A Combined System Metrics Approach to Cloud Service Reliability Using Artificial Intelligence // Big Data and Cognitive Computing. 2022. Т. 6. № 1.
18. Lakshmi Narayanan K., Naresh R. An efficient key validation mechanism with VANET in real-time cloud monitoring metrics to enhance cloud storage and security // Sustainable Energy Technologies and Assessments. 2023. Т. 56.
19. Poniszewska-Marańda A., Czechowska E. Kubernetes cluster for automating software production environment // Sensors. 2021. Т. 21. № 5.

20. Reile C. и др. Bunk8s: Enabling Easy Integration Testing of Microservices in Kubernetes // Proceedings - 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2022. , 2022.
21. Abhishek M. K., Rao D. R., Subrahmanyam K. Framework to Deploy Containers using Kubernetes and CI/CD Pipeline // International Journal of Advanced Computer Science and Applications. 2022. Т. 13. № 4.
22. Sturm R. A., Drogseth D., Twing D. CMDB Systems: Making Change Work in the Age of Cloud and Agile. , 2015.
23 The JavaScript Object Notation (JSON) Data Interchange Format, [Електронний ресурс] // Режим доступу: https://datatracker.ietf.org/doc/html/rfc8259
24. Singh J., Chaudhary N. K. OAuth 2.0: Architectural design augmentation for mitigation of common security vulnerabilities // Journal of Information Security and Applications. 2022. Т. 65.
25. Simple Object Access Protocol (SOAP) 1.1, [Електронний ресурс] // Режим доступу: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/
26. RFC7519 JSON Web Token (JWT), [Електронний ресурс] // Режим доступу: https://datatracker.ietf.org/doc/html/rfc7519
27. ComputerHope. Linux curl command help and examples.
28. OpenBSD. OpenBSD manual pages [www.openbsd.org].
29. Ylonen T. SSH Protocol – Secure Remote Login and File Transfer // Proceedings of the 6th USENIX Security Symposium. 1996.

ДОДАТОК А

Порівняння CMDB параметрів з еталоном

```
import sys
import json
import requests
import warnings
import base64
import time
warnings.filterwarnings("ignore")

global verbose_run
global TOTAL_FAILED
global api_timeout_sec

global env_prefix
global kuber_api_url
global kuber_api_header

global gw_postfix
global gw_login
global gw_header

global INT_params_target

time_start = time.time()
debug_mode = False


if debug_mode:
    verbose_run = 'y'
    api_timeout_sec = 3

    env_prefix = 'test'
    kuber_api_url = 'https://api.kuber.cloud:6443'
    kuber_api_token =
'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJPbmxpbmUgSldUIEJ1
aWxkZXIiLCJpYXQiOjE3MDI4MjQyMjYsImV4cCI6MTczNDM2MDIyNiwiY
XVkIjoid3d3LmV4YW1wbGUuY29tIiwic3ViIjoianJvY2tldEBleGFtcGxlLmNvb
```

```
SIsIkdpdmVuTmFtZSI6IkpvaG5ueSIsIlN1cm5hbWUiOiJSb2NrZXQiLCJFbWFp
bCI6Impyb2NrZXRAZXhhbXBsZS5jb20iLCJSb2xlIjpbIk1hbmFnZXIiLCJQcm9
qZWN0IEFkbWluaXN0cmF0b3IiXX0.pfsOk7XOjg2uF9W8jaJODF8-
0Z0dRbD8AGW8_Cs9I9s'
   kuber_api_header = {'Authorization': f"Bearer {kuber_api_token}"}

   gw_postfix = "auth/openid-connect/token"
   gw_username = "hello"
   gw_password = "world"
   gw_login =
f"password={gw_password}&username={gw_username}&grant_type=password&
client_id=end"
   gw_auth_header = {'Content-Type': 'application/x-www-form-urlencoded'}

   cloud_url_midfix = 'server'
   cloud_url_postfix = 'managed.cloud.server'

   int_params_target = {
      "INT_PR_SERVICE_USERID": "****",
      "INT_BS_USERNAME": "****",
      "INT_BS_PASSWORD": "****",
      "INT_KS_USERNAME": "****",
      "INT_KS_PASSWORD": "****",
      "INT_BO_TENANT": "****"
   }
else:
   verbose_run = sys.argv[1]
   api_timeout_sec = float(sys.argv[2])

   env_prefix = sys.argv[3]
   kuber_api_url = sys.argv[4]
   kuber_api_token = sys.argv[5]
   kuber_api_header = {'Authorization': f"Bearer {kuber_api_token}"}

   gw_postfix = sys.argv[6]
   gw_username = sys.argv[7]
   gw_password = sys.argv[8]
   gw_login =
f"password={gw_password}&username={gw_username}&grant_type=password&
client_id=end"
```

```
    gw_auth_header = json.loads(sys.argv[9])

    INT_params_target = json.loads(sys.argv[10])

if verbose_run == 'y':
    verbose_run = True
else:
    verbose_run = False
TOTAL_FAILED = False


def print_if_verbose(mssg):
    if verbose_run:
        print(mssg)


def print_error(mssg):
    global TOTAL_FAILED
    print(mssg)
    TOTAL_FAILED = True


def api_get(api_host, api_postfix, auth_type, header, api_timeout_sec):
    api_url = f'{api_host}/{api_postfix}'
    try:
        if auth_type == 'bearer':
            response = requests.get(api_url, headers=header, verify=False,
timeout=api_timeout_sec)
    except requests.exceptions.RequestException:
        print(f"ERROR: no response in {api_timeout_sec} sec. from URL:
{api_url}")
        return None
    if not response.ok:
        try:
            response.raise_for_status()
        except Exception as exc:
            print(f"ERROR: {exc}\n  > {response.text}")
            return None
    else:
        return response.json()
```

```
def post_api_gw(api_prefix, api_postfix, data, headers):
    api_timeout_sec = 20
    api_url = f'{api_prefix}/{api_postfix}'
    try:
        response = requests.request("POST", api_url, data=data, headers=headers,
verify=False, timeout=api_timeout_sec)
    except requests.exceptions.RequestException:
        print(f"ERROR: no response in {api_timeout_sec} sec. from URL:
{api_url}")
        return None
    if not response.ok:
        try:
            response.raise_for_status()
        except Exception as exc:
            print(f"ERROR: {exc}\n > {response.text}")
            return None
    else:
        return response


def get_idp_token(private_gw_host):
    api_response = post_api_gw(private_gw_host, gw_postfix, gw_login,
gw_auth_header)
    if api_response:
        return api_response.json()['access_token']


def INT_params_validate(INT_params_target, INT_common_variables):

    def replace_substring(json_obj, substring, replacement):
        if isinstance(json_obj, dict):
            for key in list(json_obj.keys()):
                if substring in key:
                    new_key = key.replace(substring, replacement)
                    json_obj[new_key] = json_obj.pop(key)
        return json_obj

    def get_kube_INT_secured_variables():
```

```
        api_postfix = f'api/v1/int-secured-variables'
        secured_variables_obj = api_get(kuber_api_url, api_postfix, 'bearer',
kuber_api_header, api_timeout_sec)
        if not secured_variables_obj:
            return None
        else:
            return secured_variables_obj['data']

    def decode_secure_values(obj):
        for key in obj:
            obj[key] = base64.b64decode(obj[key]).decode('utf-8')
        return obj

    def INT_params_compare(INT_params_er, INT_params_ar, search_type):
        unrecognized_params = {}
        for param_name in INT_params_er:
            if param_name not in INT_params_ar:
                print_if_verbose(f"INFO: '{param_name}' param is not found in INT
{search_type} variables..")
                unrecognized_params.update({param_name:
INT_params_er[param_name]})
            else:
                if INT_params_er[param_name] != INT_params_ar[param_name]:
                    print_error(f"FAIL: '{param_name}' {search_type} param value
mismatch:\n  AR: {INT_params_ar[param_name]}\n  ER:
{INT_params_er[param_name]}")
                else:
                    print_if_verbose(f"PASS: '{param_name}' {search_type} param value
matches:\n  AR: {INT_params_ar[param_name]}")
        return unrecognized_params

    print_if_verbose(f"INFO: searching target parameters among INT common
ones...")
    INT_params_target = replace_substring(INT_params_target, 'INT_', '')
    unrecognized_params = INT_params_compare(INT_params_target,
INT_common_variables, search_type="common")
    if unrecognized_params:
        print_if_verbose(f"INFO: searching unrecognized (presumably secured)
params in INT secret:\n{json.dumps(unrecognized_params, indent=2)}")
        secured_variables_obj = get_kube_INT_secured_variables()
```

```
        if secured_variables_obj:
            secured_variables_obj = decode_secure_values(secured_variables_obj)
            unrecognized_params = INT_params_compare(unrecognized_params,
secured_variables_obj, search_type="secured")
            if unrecognized_params:
                print_error(f"FAIL: following target parameters weren't found neither in
secured nor in common variables:\n{json.dumps(unrecognized_params,
indent=2)}")


api_postfix = f'apis/networking.k8s.io/v1/ingresses'
ingresses_obj = api_get(kuber_api_url, api_postfix, 'bearer', kuber_api_header,
api_timeout_sec)
private_gw_host = f"https://{[item['spec']['rules'][0]['host'] for item in
ingresses_obj['items'] if item['metadata']['name'] == 'private-gateway'][0]}"
idp_header = {'Authorization': f'Bearer {get_idp_token(private_gw_host)}'}

api_postfix = f'api/v1/common-variables'
public_gw_host = f"https://{[item['spec']['rules'][0]['host'] for item in
ingresses_obj['items'] if item['metadata']['name'] == 'public-gateway'][0]}"
INT_common_variables = api_get(public_gw_host, api_postfix, 'bearer',
idp_header, api_timeout_sec)

# print_if_verbose(f"INFO: INT target
variables:\n{json.dumps(INT_params_target, indent=2)}")
# print_if_verbose(f"INFO: INT common
variables:\n{json.dumps(INT_common_variables, indent=2)}")
INT_params_validate(INT_params_target, INT_common_variables)
if TOTAL_FAILED:
    print(f"\nGENERAL_FAIL: some checks are either failed or not finished")
else:
    print("GENERAL_PASS: all the validations were executed and passed")
print_if_verbose(f'INFO: execution time: {time.strftime("%H:%M:%S",
time.gmtime(time.time() - time_start))}')
```

ДОДАТОК Б

Приклад репорту Health-Check.

Health Check status:

Sl.no	Integrated Systems	Status	Link to Execution	Details
1	INT 1. BO	Pass		
2	INT 2. BA	Cancelled		
3	INT 3. KI	Pass		
4	INT 4. KL	Pass		
5	INT 1. KR	Failed	TICK-001	timeout received
6	INT 1. BS	Pass		
7	INT 1. BI	Pass		
8	INT 1. BL	Pass		
9	INT 1. BN	Pass		
10	INT 1. BE	Pass		
11	INT 1. AR	Pass		
12	INT 1. AA	Pass		
13	INT 1. AB	Pass		
14	INT 1. AT	Pass		
15	INT 1. AI	Pass		
16	INT 1. AL	Pass		
17	INT 1. CE	Pass		
18	INT 1. CC	Pass		
19	INT 1. CL	Pass		
20	INT 1. CK	Pass		
21	INT 1. CR	Pass		
22	INT 1. CP	Pass		
23	INT 1. XE	Pass		
24	INT 1. XL	Pass		

ДОДАТОК В

Приклад перевірки інтеграційного репорту.

Status of the Integrations:

Sl.no	Integration Systems	Status	End Points Validation	Credentials Validation
1	INT 1. BO	Pass	ER=AR	ER=AR
2	INT 2. BA	Pass	ER=AR	ER=AR
3	INT 3. KI	Pass	ER=AR	ER=AR
4	INT 4. KL	Pass	ER=AR	ER=AR
5	INT 1. KR	Failed	TICK-001	ER=AR
6	INT 1. BS	Pass	ER=AR	ER=AR
7	INT 1. BI	Pass	ER=AR	ER=AR
8	INT 1. BL	Pass	ER=AR	ER=AR
9	INT 1. BN	Pass	ER=AR	ER=AR
10	INT 1. BE	Failed	ER=AR	TICK-001
11	INT 1. AR	Pass	ER=AR	ER=AR
12	INT 1. AA	Pass	ER=AR	ER=AR
13	INT 1. AB	Pass	ER=AR	ER=AR
14	INT 1. AT	Pass	ER=AR	ER=AR
15	INT 1. AI	Pass	ER=AR	ER=AR
16	INT 1. AL	Pass	ER=AR	ER=AR
17	INT 1. CE	Pass	ER=AR	ER=AR
18	INT 1. CC	Pass	ER=AR	ER=AR
19	INT 1. CL	Pass	ER=AR	ER=AR
20	INT 1. CK	Pass	ER=AR	ER=AR
21	INT 1. CR	Pass	ER=AR	ER=AR
22	INT 1. CP	Pass	ER=AR	ER=AR
23	INT 1. XE	Pass	ER=AR	ER=AR
24	INT 1. XL	Pass	ER=AR	ER=AR