Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет «Комп'ютерні технології і системи»

Кафедра «Комп'ютерні інформаційні технології»

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Проектування та розгортання маштабованоі системи для збору й аналізу телеметрії в Kubernetes за допомогою Elasticsearch і Grafana»

за освітньою програмою: «12 Інженерія програмного забезпечення»

зі спеціальності: «121 Інженерія програмного забезпечення»

Виконав: студент групи «ПЗ2016»

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |

(підпис студента) (Ім’я ПРІЗВИЩЕ)

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник: |  |  |  |

(підпис) (посада, Ім’я ПРІЗВИЩЕ)

|  |  |
| --- | --- |
| Засвідчую, що у цій роботі немає запозичень з праць інших авторів без відповідних посилань. | |
| Студент |  |

(підпис)

Дніпро – 2025 рік

Ministry of Education and Science of Ukraine

Ukrainian State University of Science and Technologies

Faculty «Computer technologies and systems»

Department «Computer information technology»

Explanatory Note

to Bachelor’s Thesis

on the topic: «Design and deployment of a scalable system for collecting and analyzing telemetry in Kubernetes using Elasticsearch and Grafana»

according to educational curriculum «12 Software engineering»

in the Speciality: «121 Software engineering»

|  |  |  |
| --- | --- | --- |
| Done by the student of the group PZ2016: |  |  |
| Scientific Supervisor: |  |  |
| Normative controller: |  |  |

Dnipro – 2025

Міністерство освіти і науки України

Український державний університет науки і технологій

Факультет: «Комп'ютерні технології і системи»

Кафедра: «Комп'ютерні інформаційні технології»

Рівень вищої освіти: бакалавр

Освітня програма: «12 Інженерія програмного забезпечення»

Спеціальність: «121 Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ

Завідувач кафедри КІТ

/Вадим ГОРЯЧКІН/

(підпис)

Дата

**ЗАВДАННЯ**

На кваліфікаційну роботу

бакалавра студенту

1. Тема роботи: «Проектування та розгортання маштабованоі системи для збору й аналізу телеметрії в Kubernetes за допомогою Elasticsearch і Grafana»

Керівник роботи: Шаповал Ірина Вікторівна, старший викладач

затверджені наказом № 328ст від 03.03.2025.

|  |  |  |
| --- | --- | --- |
| 2. Строк подання студентом роботи: | | 19.06.2025 |
| 3. Вихідні дані до роботи: |  |  |
| 4. Зміст пояснювальної записки (перелік питань, які потрібно опрацювати): | | |
| вступ, збір вимог до моніторингу, зовнішнє і внутрішнє | | |
| проектування, тестування та налагодження, висновки, література. | | |
| 5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): презентація, відео-демонстрація роботи програми. | | |

**КАЛЕНДАРНИЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
| 1 | Постановка задачі | 03.04.2025 – 05.04.2025 |  |
| 2 | Огляд літератури та аналіз аналогів | 06.04.2025 – 09.04.2025 |  |
| 3 | Розробка структур вхідних і вихідних даних | 10.04.2025 – 24.04.2025 |  |
| 4 | Визначення вимог до програми. Вибір та обґрунтування мови програмування | 25.04.2025 – 02.05.2025 |  |
| 5 | Узгодження та затвердження ТЗ | 03.05.2025 – 10.05.2025 | 30% |
| 6 | Розробка та програмування логіки програми | 11.05.2023 – 18.05.2023 |  |
| 7 | Розробка і реалізація графічного відоюраження метрик (Kibana, Grafana) | 19.05.2025 – 26.05.2025 | 60% |
| 8 | Відлагодження програми | 27.05.2025 – 02.06.2025 |  |
| 9 | Розробка, узгодження та затвердження програмної документації | 03.06.2025 – 18.06.2025 | 100% |
| 10 | Подання кваліфікаційної роботи до кафедри | 19.06.2025 – 26.06.2025 |  |
| 11 | Захист кваліфікаційної роботи на засіданні Екзаменаційної комісії | 27.06.2025 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент |  |  |  |

(підпис (Ім’я ПРІЗВИЩЕ)

|  |  |  |  |
| --- | --- | --- | --- |
| Керівник роботи |  |  |  |

**(підпис (Ім’я ПРІЗВИЩЕ)**

## РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра 82 с., 10 рис., 9 табл., 12 джерел.

**Метою роботи** було проектування та розгортання масштабованої системи для збору й аналізу телеметрії в Kubernetes з використанням Elasticsearch і Grafana. Основними завданнями було створення надійної та гнучкої архітектури, що дозволяє ефективно збирати, зберігати та візуалізувати дані про роботу контейнеризованих додатків та інфраструктури, а також налаштування алертингу для проактивного виявлення проблем. За результатами роботи було створено інтегроване рішення, що відповідає всім вимогам сучасних підходів до моніторингу розподілених систем.

Робота складається з 7 розділів:

* вступ — у цьому розділі описується актуальність проблеми моніторингу контейнеризованих додатків, мета та завдання дослідження. Складається з 3 сторінок;
* збір вимог до програмного забезпечення — у цьому розділі аналізуються існуючі системи моніторингу, розглядаються теоретичні основи контейнеризації та оркестрації, принципи роботи Kubernetes, системи збору та аналізу логів і метрик. Складається з 20 сторінок;
* зовнішнє і внутрішнє проектування — у цьому розділі описується функціональне призначення системи, визначаються вхідні і вихідні дані, проводиться проектування архітектури системи моніторингу, тестового додатку, системи алертів та дашбордів для візуалізації даних. Складається з 23 сторінок;
* реалізація та тестування — цей розділ містить інформацію про розгортання Kubernetes кластера, налаштування компонентів системи моніторингу (ELK Stack, Prometheus, Grafana), розгортання тестового додатку та результати тестування системи. Складається з 18 сторінок;
* висновки — підсумовуються результати роботи та визначаються перспективи подальших досліджень. Складається з 2 сторінок;
* список літератури — включає бібліографічний список використаної літератури. Складається з 2 сторінок;
* додатки — містять конфігураційні файли та код, що використовувався для розгортання та налаштування системи, а також приклади дашбордів. Складається з 14 сторінок.

Кількість таблиць: 9 штук. Кількість рисунків: 10 штук.

Ключові слова: KUBERNETES, МОНІТОРИНГ, ELASTICSEARCH, GRAFANA, PROMETHEUS, КОНТЕЙНЕРИЗАЦІЯ, ОРКЕСТРАЦІЯ, ЛОГИ, МЕТРИКИ, ТЕЛЕМЕТРІЯ, МАСШТАБОВАНІСТЬ, АНАЛІЗ ДАНИХ, ВІЗУАЛІЗАЦІЯ, АЛЕРТИНГ.

ЗМІСТ

[ВСТУП 9](#_Toc138003767)

[1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 11](#_Toc138003768)

[1.1 11](#_Toc138003769)

[1.1.1 Огляд платформи «DataDog»     12](#_Toc138003770)

[1.1.2 Огляд платформи «Dynatrace»     13](#_Toc138003771)

[1.1.3 Огляд стеку ELK](#_Toc138003772).

[1.1.4 Огляд Prometheus Stack     16](#_Toc138003773)

[1.2.1 Принципи роботи Kubernetes. 17](#_Toc138003774)

[Висновки до пункту 1 Error: Reference source not found](#_Toc138003775)

[2 ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ 22](#_Toc138003776)

[2.1 Зовнішнє проектування Error: Reference source not found](#_Toc138003777)

[2.1.1 Функціональне призначення Error: Reference source not found](#_Toc138003778)

[2.1.2 Експлуатаційне призначення Error: Reference source not found](#_Toc138003779)

[2.1.3 Функціональні вимоги Error: Reference source not found](#_Toc138003780)

[2.1.4 Вхідні та вихідні дані Error: Reference source not found](#_Toc138003781)

[2.1.4.1 Вхідні дані Error: Reference source not found](#_Toc138003782)

[2.1.4.2 Вихідні дані Error: Reference source not found](#_Toc138003783)

[2.1.5 Опис зовнішнього інформаційного середовища Error: Reference source not found](#_Toc138003784)

[2.2 Внутрішнє проектування Error: Reference source not found](#_Toc138003785)

[2.2.1 Аналіз зовнішніх специфікацій систем Error: Reference source not found](#_Toc138003786)

[2.2.1.1 Моделювання словника системи Error: Reference source not found](#_Toc138003787)

[3 ТЕСТУВАННЯ ТА НАЛАГОДЖЕННЯ Error: Reference source not found](#_Toc138003788)

[Висновки до пункту 3 Error: Reference source not found](#_Toc138003789)

[ВИСНОВОК Error: Reference source not found](#_Toc138003790)

[ЛІТЕРАТУРА Error: Reference source not found](#_Toc138003791)

[ДОДАТКИ Error: Reference source not found](#_Toc138003792)

[ДОДАТОК А – ТЕКСТ ПРОГРАМИ Error: Reference source not found](#_Toc138003793)

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ (програмне забезпечення) – програмний продукт, який застосовується на комп’ютерних системах таких як персональні комп’ютери, ноутбуки, мобільні пристрої, планшети, телефони а також деколи і на різній побутовій техніці на кшталт пральних машин, мікрохвильовок, холодильників та інше.

Кастомізація – підлаштовування, налаштування чогось під себе. Те ж саме що й персоналізація.

Нота – ступінь або звук, що має характеристики: октава, знак альтерації, протяжність та інші.

Тональність – це висотне розташування ладу. Усього існує 30 тональностей.

Лад – це сукупність звуків, які на основі спорідненості між ними об’єднані в систему, що має тоніку.

Тоніка – базова нота тональності або ладу, тобто та нота, від якої йде побудова тональності чи ладу.

Півтон – одиниця виміру музичних інтервалів, є найменшою відстанню між звуками за висотою. Зазвичай, візуально на фортепіано, у ролі півтону виступає одна клавіша.

Тон – одиниця виміру музичних інтервалів, містить в собі два півтони. Зазвичай, візуально на фортепіано, у ролі тону виступає дві клавіші підряд.

Ступінь – це звук музичної системи (звукоряду). Усього основних ступенів сім – «до», «ре», «мі», «фа», «соль», «ля», «сі».

Знак альтерації – знак, що відображає підвищення або зниження ступеню на півтона або тон. Таких знаків декілька: дієз (♯), дубль-дієз, бемоль (♭), дубль-бемоль, а також бекар (♮) – знак відміни дій інших знаків.

Нотний стан – це система з п’яти паралельних горизонтальних ліній, на якій розміщують ноти.

Октава – інтервал від «до» до «сі» включаючи обидві ноти, відображає звукоряд певної висоти. Таких октав може бути на різних інструментах різну кількість, і усі октави мають власні назви та власну висоту звучання.

Мажор – музичний лад, аккорди якого будуються на великій терції. Зазвичай цей лад має позитивне, радісне звучання.

Мінор – музичний лад, аккорди якого будуються на малій терції. Зазвичай цей лад має похмуре, сумне звучання.

# ВСТУП

Музика, як і у минулому, сьогодні, так і надалі залишатиметься невід’ємною частиною нашого життя. Технології просунулася вперед і тим самим змінилася сама музика – з’явились безліч нових інструментів та різноманітних жанрів, покращилась якість звучання, спростився процес створення нових композицій, пісень, аранжувань. Але будь-який інструмент у руках людини без гарної практики та базових знань ­– просто не розкритий інструмент, на якому неможливо відтворити щось дійсно приголомшливе. У всіх сферах є умовна "база знань", де потрібно досягти максимального рівня навичок, щоб отримати фундамент, завдяки якому людина стане майстром справи. У музиці важлива кожна нота, зіграна на інструменті, адже кожна нота задає тон та настрій композиції. Але все ж таки, чому музика настільки важлива для нашого світу? А важлива вона тим, що завдяки їй часто люди піднімають собі настрій, задають атмосферу у своєму повсякденному житті, мають різні спогади, пов’язані з окремими піснями. В тяжкі для людини часи, музика надає натхнення рухатись вперед та направляє на різноманітні вчинки, надає мотивацію, підтримує дух людини аби не дати їй впасти духом та поринути у депресію. Словом, «Музика дозволяє рухатись вперед і бути закоханим у неї». А ще є такі люди, як меломани, що просто не уявляють своє життя без музики, адже меломани обожнюють абсолютно різні жанри.

Завдяки технологіям нашого часу можна знайти різноманітні програми, що допомагають музикантам у вивченні та виявленні особистих проблем у музиці. Такі програми допомагають музикантові у його становленні, як висококваліфікованого фахівця. Тому, аби допомогти усім музикантам, а саме: як початківцям, так і більш досвідченим, – було вирішено створити ПЗ «Elegia», яка сприятиме у відточенні музичних навичок та знань музичної теорії, методом використовування запропонованих інструментів ПЗ для досягнення почуття, що передає та сама фраза: «Музика допомагає рухатися вперед…».

Завданням цієї дипломної роботи є вдосконалення навичок знань музиканта. Робота передбачає за собою написану музичну програму, що має тренажерні тести та інструментарій для музикантів із простим інтерфейсом для легкого використання. Для користувача ПЗ, здебільшого, є два режими у кожному розділі – режим тренування та режим інструменту, завдяки яким він зможе відточити свої знання в музичній теорії на практиці. Показувати результат тестування користувачеві – важлива складова навчання. Окрім зміцнення знань, «Elegia» матиме ряд інструментів із роботою для піаніно. Користувачем може бути будь-яка зацікавлена у музиці ​​людина – як музикант-початківець, так і професійний музикант зі стажем.

# 1 ЗБІР ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## 1.1 Перегляд аналогових конкурентів

Серед сучасних рішень для моніторингу контейнеризованих середовищ та інфраструктури, окрім Prometheus, варто виділити дві потужні комерційні платформи — DataDog та Dynatrace. DataDog представляє собою комплексну SaaS-платформу для моніторингу інфраструктури та додатків, що відрізняється швидким розгортанням завдяки готовим інтеграціям з понад 450 технологіями. Платформа використовує агентну модель збору даних, де легкий агент встановлюється на кожний сервер або контейнер і збирає метрики, логи та трейси, забезпечуючи візуальну кореляцію між ними. Особливу увагу варто звернути на розширені можливості DataDog для моніторингу Kubernetes, включаючи автоматичне виявлення подів, сервісів, відстеження життєвого циклу контейнерів та візуалізацію взаємозалежностей між компонентами системи. В свою чергу, Dynatrace пропонує унікальний підхід "один агент для всього" через технологію OneAgent, яка автоматично виявляє всі компоненти середовища та збирає необхідну телеметрію без ручного налаштування. Ключовою перевагою Dynatrace є запатентована система штучного інтелекту Davis, що аналізує мільярди залежностей для виявлення причинно-наслідкових зв'язків між подіями та автоматичного визначення кореневих причин проблем. Обидві платформи пропонують першокласну інтеграцію з Kubernetes, включаючи спеціалізовані оператори для розгортання агентів, глибокий моніторинг кластера та компонентів, а також потужні аналітичні інструменти. Проте, на відміну від відкритого Prometheus, ці рішення мають значні недоліки — високу вартість при масштабуванні, зберігання даних у хмарній інфраструктурі стороннього провайдера та закриту екосистему з обмеженими можливостями кастомізації.

## Огляд платформи DataDog

DataDog — це комплексна SaaS-платформа для моніторингу інфраструктури, додатків та цифрових користувацьких взаємодій. Заснована у 2010 році, сьогодні вона є одним із провідних рішень для спостереження (observability) та моніторингу в корпоративному сегменті.

### Архітектура та принципи роботи

DataDog використовує агентну модель збору даних. Легкий агент встановлюється на кожний сервер, контейнер або віртуальну машину. Він збирає метрики, логи та трейси, відправляючи їх у централізоване сховище DataDog для подальшої обробки та аналізу.

Основні компоненти архітектури DataDog:

1. **DataDog Agent** — легкий, відкритий програмний агент, написаний на Go, який збирає метрики, логи та трейси з хостів і сервісів.
2. **Інтеграції** — готові модулі для збору даних від популярних технологій та сервісів.
3. **Central DataDog Platform** — хмарна платформа для зберігання, обробки та візуалізації даних.
4. **API і SDK** — інструменти для інтеграції та кастомізації.

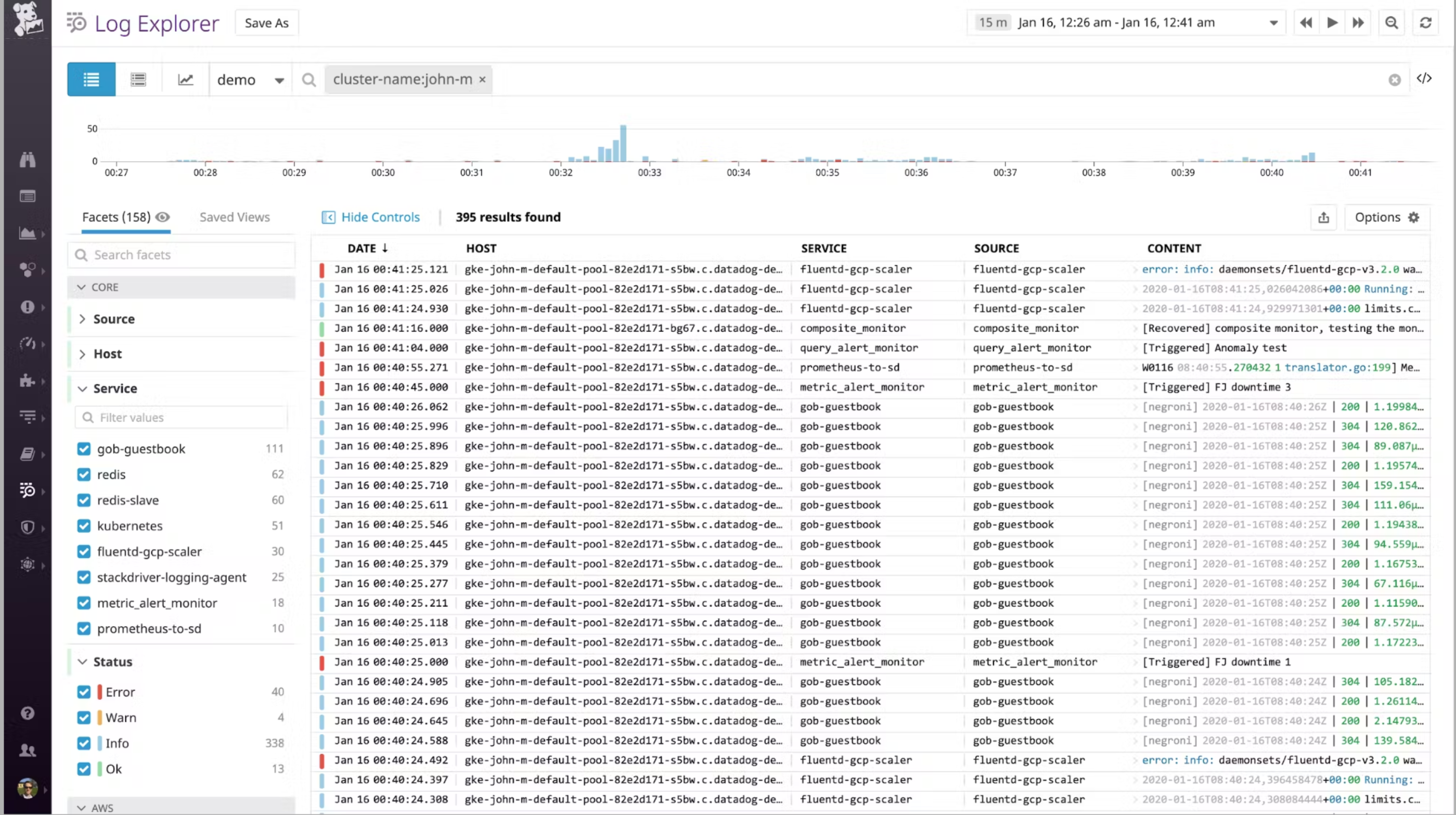


Рисунок 1.1 – Приклад дашборда DataDog

Переваги:

* **Швидкість впровадження**: готові інтеграції з понад 450 технологіями дозволяють швидко почати збір даних;
* **Уніфікований моніторинг**: єдина платформа для метрик, логів та трейсів;
* **Масштабованість**: здатність обробляти дані від десятків тисяч серверів;

Недоліки:

* **Висока вартість при масштабуванні**: ціна залежить від кількості хостів та функцій, що використовуються.
* **Зберігання даних у хмарі**: чутливі дані зберігаються на серверах DataDog, що може не відповідати вимогам безпеки деяких організацій.
* **Складність налаштування просунутих функцій**: деякі можливості вимагають глибокого розуміння платформи.

## 1.1.2 Огляд платформи Dynatrace

Dynatrace — це інтелектуальна платформа для моніторингу та обсерваційності, що використовує штучний інтелект для автоматичного виявлення та діагностики проблем у складних ІТ-середовищах. Заснована у 2005 році в Австрії, компанія Dynatrace зробила значний перехід від традиційного APM (Application Performance Management) до повноцінної платформи обсерваційності з власною AI-системою Davis. (сам приклад дашборду наведено на рис. 1.2).

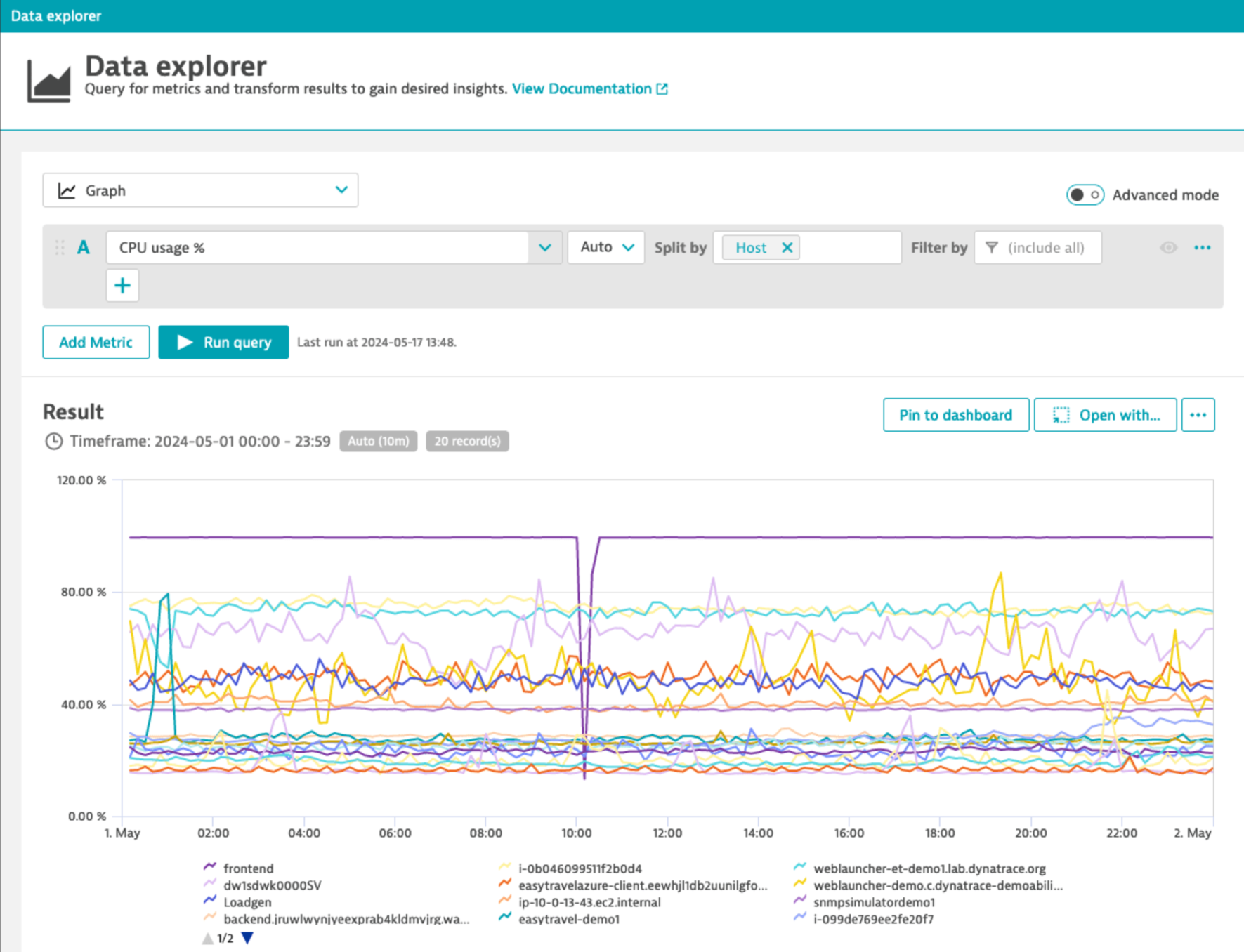


Рисунок 1.2 – Приклад дашборду «Dynatrace»

Переваги:

* **Автоматизація та AI** — повністю автоматичне виявлення та діагностика проблем без необхідності ручного налаштування;
* **Повна видимість** — єдина платформа для моніторингу від інфраструктури до користувацького досвіду;
* **Causation-Based AI** — визначення кореневих причин проблем, а не просто кореляцій між подіями;
* **Самоадаптація** — автоматична адаптація до змін в середовищі.

Недоліки:

* **Висока вартість ліцензії** — одне з найдорожчих рішень для моніторингу на ринку;
* **Складність впровадження** — потребує значного часу для повноцінного впровадження та налаштування;
* **Вимоги до ресурсів** — може потребувати значних ресурсів для роботи в великих середовищах;
* **Закрита екосистема** — менша гнучкість порівняно з відкритими рішеннями;
* **Складна крива навчання** — потребує часу для повного освоєння всіх можливостей;

## 1.1.3 Огляд стеку ELK.

ELK Stack представляє собою інтегроване рішення з відкритим вихідним кодом, спрямоване на комплексну роботу з логами: від їх збору та обробки до зберігання й візуалізації. Назва "ELK" походить від перших літер ключових компонентів: Elasticsearch, Logstash та Kibana. З розвиткомтехнології до стеку додалися легкі агенти Beats, що розширили його функціональність.

Рисунок 1.3 ілюструє архітектуру ELK Stack (Elasticsearch, Logstash, Kibana), доповнену компонентами Beats. Діаграма відображає повний потік даних у системі логування та моніторингу.

Зліва зображено сімейство Beats - легкі агенти для збору різних типів даних:

* Filebeat - для збору логів з файлів
* Winlogbeat - для збору логів подій Windows
* Metricbeat - для збору системних та прикладних метрик
* Packetbeat - для аналізу мережевого трафіку

Ці компоненти виконують функцію крайового збору даних (Edge Data Collection).

У центрі розташовано Logstash - систему обробки та трансформації даних, що представлена кількома робочими вузлами (Nodes) з увімкненими персистентними чергами (Persistent Queues Enabled). Logstash отримує дані від агентів Beats, обробляє їх та передає до Elasticsearch.

Справа розміщено два компоненти:

1. Elasticsearch - розподілене сховище для зберігання і пошуку даних, що складається з:
   * Master Nodes (3) - вузли управління кластером
   * Data Nodes - Hot (X) - вузли для зберігання активних даних
   * Data Nodes - Warm (X) - вузли для зберігання менш активних даних
2. Kibana - платформа візуалізації з кількома екземплярами (Instances (X))

Стрілки показують напрямок руху даних: від Beats до Logstash, і далі від Logstash до Elasticsearch, а Kibana підключається до Elasticsearch для відображення та аналізу зібраних даних.

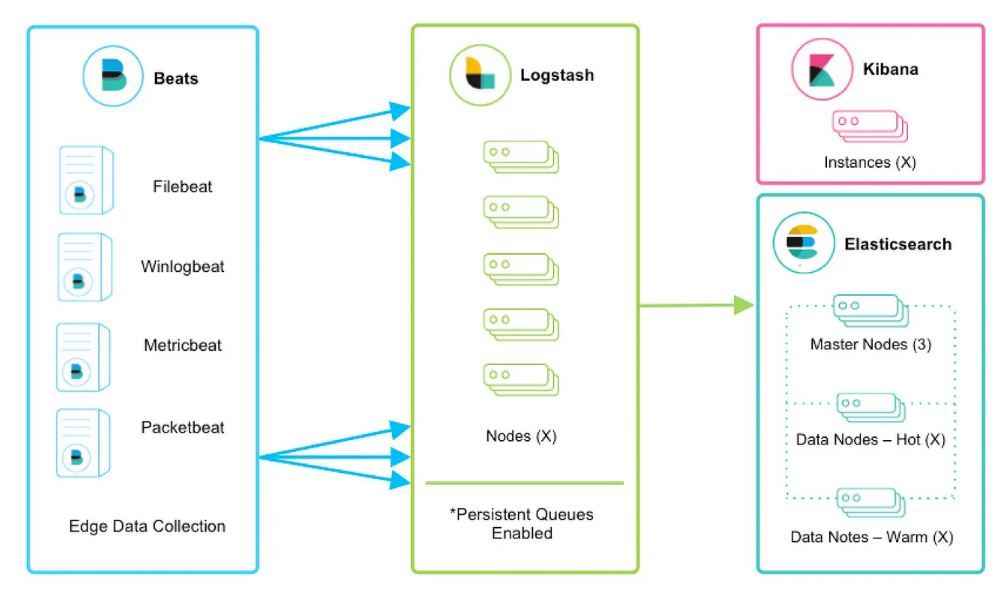


Рисунок 1.3 – ELK Stack

Переваги:

* **Прозорість реалізації**: можливість аудиту та розуміння внутрішньої роботи;
* **Спільнота розробників**: активна підтримка та розвиток компонентів;
* **Безкоштовне базове використання**: зниження витрат на впровадження;
* **Відсутність прив'язки до вендора**: незалежність від політики окремих компаній;
* **Можливість модифікації**: адаптація під специфічні потреби організації;

Недоліки:

* **Вимоги до ресурсів:** Одним з найсуттєвіших недоліків ELK Stack є високі вимоги до обчислювальних ресурсів;

### **Складність налаштування та підтримки:** ELK Stack вимагає значних зусиль для налаштування та підтримки;

### **Обмеження в роботі з метриками:** Незважаючи на можливість збору метрик через Metricbeat, ELK Stack має певні обмеження;

## 1.1.4 Огляд Prometheus Stack

Prometheus з Grafana формують потужний стек для моніторингу інфраструктури та додатків, який став стандартом де-факто для середовищ на базі Kubernetes. Цей стек пропонує спеціалізоване рішення для збору, зберігання, аналізу та візуалізації метрик.

## Архітектура та принципи роботи

Prometheus Stack побудований на кількох ключових компонентах:

* **Prometheus Server**: центральний компонент, що збирає та зберігає метрики як часові ряди
* **Exporters**: спеціалізовані агенти для надання метрик з різних систем
* **Alertmanager**: компонент для управління оповіщеннями
* **Pushgateway**: проміжний сервіс для короткоживучих завдань
* **Grafana**: платформа для візуалізації метрик у вигляді дашбордів

Архітектура стеку базується на моделі "pull", де Prometheus активно опитує (scrapes) цільові системи, а не чекає відправки даних, що забезпечує більшу надійність та контроль.

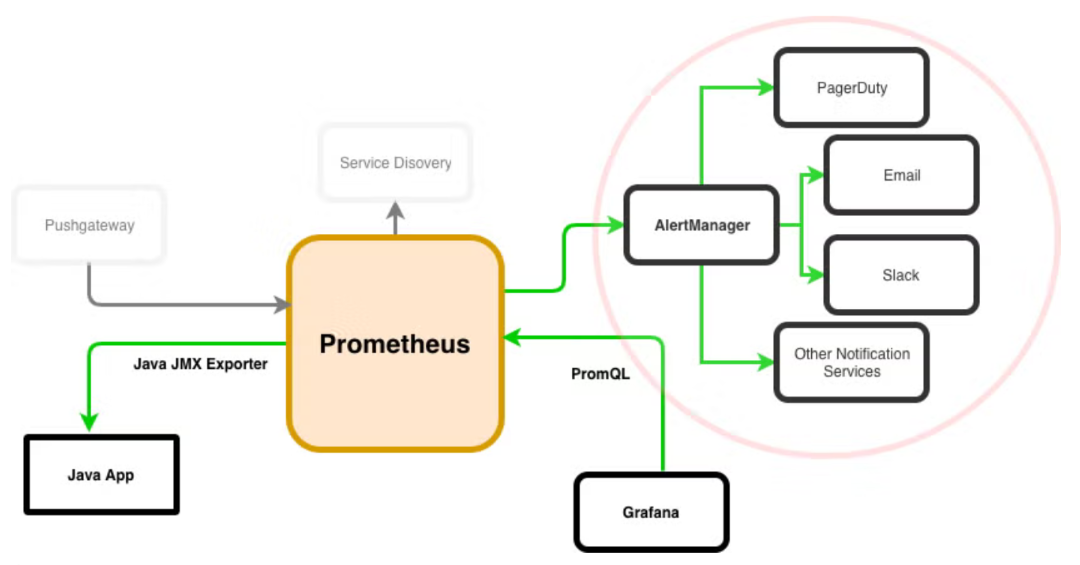


Рисунок 1.4 – Архітектура Prometheus

## 1.2.1 Принципи роботи Kubernetes.

Kubernetes (K8s) — це відкрита платформа для автоматизації розгортання, масштабування та управління контейнеризованими застосунками. Розроблена початково Google на основі внутрішньої системи Borg, Kubernetes стала стандартом де-факто для оркестрації контейнерів у промисловому масштабі.

#### Архітектура Kubernetes

Kubernetes має розподілену архітектуру типу "клієнт-сервер", що складається з наступних основних компонентів:

**Control Plane (Площина управління)**

Площина управління відповідає за глобальні рішення щодо кластера (наприклад, планування) та виявлення і реагування на події кластера. Компоненти площини управління включають:

1. **API Server (kube-apiserver)**: центральний компонент, що експонує API Kubernetes, обробляє запити та оновлює відповідні об'єкти в etcd. Всі взаємодії з кластером проходять через API Server.
2. **etcd**: розподілене сховище ключ-значення, яке зберігає всі дані кластера. Забезпечує надійне збереження конфігурації кластера та його стану.
3. **Scheduler (kube-scheduler)**: відповідає за відстеження новостворених подів, які не призначені до жодної ноди, та вибір ноди для їх запуску. Фактори для планування включають:
   * Вимоги до ресурсів
   * Обмеження апаратного/програмного забезпечення
   * Афініті та анти-афініті
   * Баланс навантаження між нодами
4. **Controller Manager (kube-controller-manager)**: запускає контролери, що обробляють рутинні завдання в кластері. Контролери включають:
   * Node Controller: моніторинг стану нод і реагування на їх відмови
   * Replication Controller: забезпечення правильної кількості подів
   * Endpoints Controller: зв'язування сервісів з подами
   * Service Account & Token Controllers: створення облікових записів і API токенів
5. **Cloud Controller Manager (cloud-controller-manager)**: інтегрує Kubernetes з API провайдерів хмарних послуг. Дозволяє основній кодовій базі Kubernetes залишатися незалежною від специфічних хмарних провайдерів.

**Worker Nodes (Робочі вузли)**

Worker ноди — це машини, які запускають контейнеризовані застосунки. Кожна нода містить:

1. **Kubelet**: агент, що запускається на кожній ноді. Забезпечує, що контейнери запускаються в поді. Kubelet:
   * Отримує специфікації подів від API сервера
   * Гарантує, що вказані контейнери працюють справно
   * Звітує про стан подів і ноди до API сервера
   * Запускає проби готовності та життєздатності
2. **Container Runtime**: програмне забезпечення, відповідальне за запуск контейнерів (containerd, CRI-O, Docker Engine та ін.)
3. **Kube-proxy**: мережевий проксі, що працює на кожній ноді. Підтримує мережеві правила та забезпечує зв'язок із подами. Реалізує частину концепції Kubernetes Service, керуючи перенаправленням з'єднань до відповідних подів.

#### Основні абстракції Kubernetes

Kubernetes оперує рядом абстракцій, що представляють стан системи:

**1. Pod (Под)**

Под — це найменша і найпростіша одиниця в Kubernetes, що представляє одиночний екземпляр застосунку. Под інкапсулює:

* Один або кілька контейнерів
* Спільний простір зберігання (томи)
* Специфічні налаштування мережі (IP-адреса, яку поділяють всі контейнери)
* Опції для запуску контейнерів

Поди, за своєю природою, є ефемерними (тимчасовими) і можуть бути замінені іншим подом при необхідності масштабування, оновлення або в разі збоїв.

**2. ReplicaSet**

ReplicaSet гарантує, що вказана кількість реплік подів запущена в будь-який момент часу. Забезпечує:

* Масштабованість застосунків
* Відмовостійкість через автоматичне відновлення подів
* Декларативне визначення бажаного стану

**3. Deployment**

Deployment — це вищий рівень абстракції над ReplicaSet, що надає декларативні оновлення для подів та ReplicaSets. Забезпечує:

* Стратегії оновлення (Rolling Update, Recreate)
* Відстеження історії розгортань
* Можливість відкату до попередніх версій
* Паузу та продовження розгортання

**4. Service**

Service — це абстракція, що визначає логічний набір подів і політику доступу до них. Сервіси дозволяють слабке зв'язування між залежними подами. Типи сервісів:

* ClusterIP: внутрішня IP-адреса, доступна лише всередині кластера
* NodePort: експонує сервіс на статичному порту кожної ноди
* LoadBalancer: використовує зовнішній балансувальник навантаження провайдера хмари
* ExternalName: повертає DNS CNAME запису

**5. Volume**

Volume вирішує дві проблеми: втрату даних, коли контейнер перезапускається, і спільне використання даних між контейнерами в поді. Типи томів:

* emptyDir: тимчасове порожнє сховище
* hostPath: монтування файлу або директорії з файлової системи ноди
* persistentVolume: постійне сховище, незалежне від життєвого циклу подів
* configMap/secret: монтування конфігурацій та секретів як файлів

**6. Namespace**

Namespace забезпечує віртуальне розділення кластера Kubernetes:

* Ізоляція ресурсів для різних команд, проектів або застосунків
* Керування доступом на рівні namespace
* Призначення квот ресурсів для окремих проектів

**7. StatefulSet**

StatefulSet керує розгортанням і масштабуванням набору подів з гарантіями щодо порядку розгортання та унікальності:

* Стабільні, унікальні імена мережі
* Стабільне, постійне сховище
* Упорядкований, плавний розгортання і масштабування
* Упорядковане оновлення та видалення

**8. DaemonSet**

DaemonSet забезпечує запуск копії поду на всіх (або деяких) нодах кластера:

* Ідеально для агентів моніторингу
* Збір логів на кожній ноді
* Агенти зберігання даних

**9. Job і CronJob**

Job створює один або кілька подів і гарантує, що вказана кількість успішно завершиться. CronJob створює Jobs за розкладом, подібно до crontab в Unix системах.

**10. ConfigMap і Secret**

ConfigMap зберігає конфігураційні дані у форматі ключ-значення для використання подами. Secret подібний до ConfigMap, але призначений для чутливих даних (паролі, токени, ключі).

# 2 ЗОВНІШНЄ І ВНУТРІШНЄ ПРОЕКТУВАННЯ

## 2.1 Зовнішнє проектування

### 2.1.1 Функціональне призначення

Функціональним призначенням розроблюваної системи є забезпечення повного циклу збору, зберігання, аналізу та візуалізації телеметричних даних у середовищі Kubernetes. Система призначена для:

* Збору метрик стану та продуктивності компонентів Kubernetes-кластера (вузлів, подів, сервісів)
* Збору логів контейнеризованих додатків з усіх вузлів кластера
* Централізованого зберігання та індексації телеметричних даних
* Аналізу зібраних даних з можливістю виявлення аномалій та тенденцій
* Візуалізації телеметрії через інтерактивні інформаційні панелі
* Налаштування системи оповіщень для проактивного виявлення проблем
* Масштабування компонентів системи відповідно до розміру кластера та обсягу даних

Система інтегрується з Kubernetes API для динамічного виявлення нових компонентів інфраструктури та застосовує принципи "zero-configuration" моніторингу, що дозволяє автоматично виявляти та починати збір даних з нових компонентів без ручного втручання.

## 2.1.2 Експлуатаційне призначення

Експлуатаційне призначення розроблюваної системи включає:

* Моніторинг стану та продуктивності Kubernetes-кластерів у режимі реального часу
* Діагностику проблем продуктивності та надійності розподілених додатків
* Аналіз трендів використання ресурсів для планування потужностей інфраструктури
* Ретроспективний аналіз інцидентів для визначення їх першопричин
* Забезпечення дотримання угод про рівень обслуговування (SLA) через моніторинг ключових показників
* Підтримку DevOps-практик через інтеграцію з CI/CD-системами
* Забезпечення аудиту безпеки через централізований збір та аналіз логів
* Підтримку мультитенантності для ізоляції даних у спільній інфраструктурі моніторингу

Система розрахована на використання різними категоріями фахівців:

* Адміністраторами Kubernetes для моніторингу здоров'я кластера
* Розробниками для діагностики та оптимізації додатків
* DevOps-інженерами для автоматизації операційних процесів
* Фахівцями з інформаційної безпеки для виявлення потенційних загроз

Очікується, що система функціонуватиме в режимі 24/7, забезпечуючи безперервний збір телеметрії та доступ до історичних даних.

## 2.1.3 Функціональні вимоги

Розроблювана система має відповідати наступним функціональним вимогам:

### Збір метрик:

* Автоматичне виявлення та моніторинг вузлів, подів, контейнерів, сервісів та інших об'єктів Kubernetes
* Збір системних метрик (CPU, пам'ять, мережа, диск) з усіх компонентів кластера
* Підтримка кастомних метрик додатків через Prometheus API

### Збір логів:

* Збір логів з усіх контейнерів у кластері незалежно від їх розташування
* Підтримка різних форматів логів (JSON, plain text, структурований)
* Корреляція логів з метриками за часовими мітками і ідентифікаторами

### Зберігання даних:

* Масштабоване зберігання метрик і логів в Elasticsearch
* Конфігурована політика ротації та видалення даних

### Візуалізація:

* Набір попередньо налаштованих дашбордів Grafana для моніторингу Kubernetes
* Інтерактивні графіки з можливістю налаштування користувачем

### Аналіз та алертинг:

* Налаштування порогових значень для ключових метрик
* Попередньо сконфігуровані правила оповіщень для типових проблем Kubernetes

### Масштабування та надійність:

* Горизонтальне масштабування всіх компонентів системи
* Відмовостійкість і автоматичне відновлення при збоях
* Буферизація даних при недоступності компонентів зберігання
* Механізми обмеження навантаження та захисту від перевантаження

## 2.1.4 Вхідні та вихідні дані

### 2.1.4.1 Вхідні дані

Вхідними даними для системи моніторингу та аналізу телеметрії в Kubernetes є:

1. **Метрики рівня інфраструктури:**
   * Утилізація CPU (відсоток використання, навантаження)
   * Використання оперативної пам'яті (загальний обсяг, використано, доступно, кеш)
   * Використання дискового простору (загальний обсяг, використано, доступно, I/O операції)
2. **Метрики Kubernetes:**
   * Стан вузлів (готовність, навантаження, кількість подів)
   * Стан подів (фаза життєвого циклу, рестарти, готовність)
   * Використання ресурсів на рівні подів та контейнерів
3. **Логи:**
   * Системні логи контейнерів (stdout, stderr)
   * Логи додатків з файлів всередині контейнерів
   * Логи компонентів Kubernetes (api-server, scheduler, controllers)
4. **Конфігураційні дані:**
   * Kubernetes API об'єкти (pods, deployments, services)
   * Конфігурація мережі та безпеки
   * Анотації та мітки ресурсів Kubernetes
   * Змінні середовища контейнерів
   * Секрети та конфігмапи (у безпечному форматі)
5. **Дані про події:**
   * Kubernetes events (створення, оновлення, видалення ресурсів)
   * Повідомлення про збої та відновлення

Формати вхідних даних включають:

* Prometheus метрики у форматі text/plain або OpenMetrics
* Логи у форматах plain text, JSON, structured logging
* API-дані у форматі JSON
* Конфігураційні файли у форматах YAML, JSON, INI

### 2.1.4.2 Вихідні дані

Вихідними даними системи моніторингу та аналізу телеметрії в Kubernetes є:

1. **Візуалізовані дані у вигляді дашбордів Grafana:**
   * Оперативні дашборди стану кластера з зведеною інформацією
   * Деталізовані графіки використання ресурсів за часовими рядами
   * Панелі перегляду та аналізу логів
2. **Аналітичні звіти:**
   * Звіти про використання ресурсів з прогнозом трендів
   * Аналіз аномалій та викидів у метриках
   * Аналіз помилок та їх кореляцій
3. **Оповіщення (алерти):**
   * Сповіщення про перевищення порогових значень метрик
   * Повідомлення про аномалії в роботі системи
4. **Дані для API та інтеграцій:**
   * JSON-відповіді на запити до API Grafana та Elasticsearch
   * Експортовані дані для CI/CD систем та автоматизації
5. **Експортовані файли:**
   * Вивантажені дані у форматах CSV, JSON, Excel
   * Архівовані логи та метрики для довгострокового зберігання
   * Резервні копії конфігурацій та правил оповіщень
6. **Дані аудиту та безпеки:**
   * Журнали доступу до системи моніторингу
   * Статистика доступу до даних за користувачами та ролями

Вихідні дані надаються через наступні інтерфейси:

* Веб-інтерфейс Grafana для інтерактивної взаємодії
* REST API Elasticsearch для прямого доступу до даних
* API Grafana для програмного доступу до дашбордів

## 2.1.5 Опис зовнішнього інформаційного середовища

Система збору й аналізу телеметрії в Kubernetes функціонує в складному інформаційному середовищі, взаємодіючи з різними компонентами інфраструктури та зовнішніми системами:

### Взаємодія з Kubernetes-кластером:

* **Kubernetes API Server** - основне джерело інформації про стан кластера та його ресурси. Система використовує API для отримання даних про поди, вузли, сервіси та інші об'єкти Kubernetes, а також для відстеження змін у конфігурації.
* **Kubelet** - агент на кожному вузлі кластера, який надає дані про стан вузла та контейнерів, що на ньому виконуються. Взаємодія відбувається через API або напряму через зчитування метрик з cAdvisor.
* **контейнери/поди** - основні одиниці розгортання додатків, які виробляють логи та метрики. Система збирає stdout/stderr потоки, а також експоновані метрики через HTTP endpoints.

### Середовище зберігання та обчислення:

* **Постійне сховище** (PersistentVolumes в Kubernetes) - використовується для зберігання даних Elasticsearch та конфігурацій системи.
* **Мережеве середовище** - включає внутрішню мережу Kubernetes та зовнішні мережеві компоненти (LoadBalancers, Ingress).
* **Обчислювальні ресурси** (CPU, пам'ять) - розподіляються між компонентами системи моніторингу з урахуванням пріоритетів та QoS.

### Інтерфейси взаємодії з користувачами:

* **Веб-інтерфейси** (Grafana, Kibana) - забезпечують візуальний доступ до даних моніторингу через HTTP/HTTPS.
* **API інтерфейси** - надають програмний доступ до даних та функцій системи моніторингу.
* **CLI інструменти** - забезпечують командний інтерфейс для взаємодії з системою моніторингу.

### Обмеження та вимоги середовища:

* **Політики безпеки** - визначають обмеження на збір та зберігання чутливих даних.
* **Мережеві політики** - регулюють доступ між компонентами системи та зовнішнім світом.
* **Квоти ресурсів** - обмежують використання обчислювальних ресурсів компонентами моніторингу.

Система спроектована з урахуванням усіх цих аспектів зовнішнього інформаційного середовища для забезпечення безперебійної роботи та інтеграції з існуючими IT-процесами організації.

## 2.2 Внутрішнє проектування

### 2.2.1 Аналіз зовнішніх специфікацій систем

При проектуванні системи збору й аналізу телеметрії в Kubernetes необхідно враховувати специфікації та API зовнішніх компонентів, з якими відбуватиметься інтеграція. Такий аналіз дозволяє забезпечити сумісність та ефективну взаємодію між різними частинами архітектури.

#### Kubernetes API

Kubernetes API є основним інтерфейсом для взаємодії з кластером і надає доступ до ресурсів через RESTful HTTP API:

* **Формат даних**: JSON
* **Методи аутентифікації**: сертифікати x509, Bearer tokens, OIDC
* **Версіонування API**: стабільні (наприклад, v1), бета (v1beta1) та альфа (v1alpha1) версії
* **Ресурси, що використовуються**:
  + /api/v1/nodes - інформація про вузли кластера
  + /api/v1/pods - інформація про поди
  + /api/v1/namespaces - доступ до просторів імен
  + /apis/apps/v1/deployments - дані про розгортання
  + /apis/metrics.k8s.io/v1beta1 - метрики ресурсів (через Metrics Server)

Для роботи з Kubernetes API система використовуватиме офіційні клієнтські бібліотеки та SDK.

#### Prometheus API

Prometheus надає API для збору та запиту метрик:

* **Формат даних**: специфічний текстовий формат Prometheus, JSON для HTTP API
* **Endpoint для scraping**: зазвичай /metrics на цільових сервісах
* **Query API**:
  + /api/v1/query - для виконання миттєвих запитів
  + /api/v1/query\_range - для запитів за часовим діапазоном
  + /api/v1/series - для пошуку серій часових рядів
* **Формат запитів**: PromQL (Prometheus Query Language)

#### Elasticsearch API

Elasticsearch забезпечує потужний RESTful API для зберігання, пошуку та аналізу даних:

* **Формат даних**: JSON
* **Основні endpoint'и**:
  + /{index}/\_doc - індексація документів
  + /{index}/\_search - пошук документів
  + /{index}/\_update - оновлення документів
  + /\_cluster/health - перевірка стану кластера
* **Мова запитів**: Elasticsearch Query DSL
* **API версіонування**: семантичне версіонування (7.x, 8.x)

#### Grafana API

Grafana надає HTTP API для взаємодії з дашбордами та іншими ресурсами:

* **Формат даних**: JSON
* **Аутентифікація**: API Keys, Basic Auth, OAuth
* **Основні endpoint'и**:
  + /api/dashboards - управління дашбордами
  + /api/datasources - управління джерелами даних
  + /api/alerts - управління оповіщеннями
  + /api/user - управління користувачами
* **Особливості**: підтримка організацій (multi-tenancy)

#### Протоколи взаємодії

Для забезпечення взаємодії між компонентами системи використовуються такі протоколи:

* **HTTP/HTTPS**: основний протокол для взаємодії з API
* **TCP/UDP**: для передачі логів (Syslog, Fluentd forward protocol)

Аналіз цих специфікацій дозволяє правильно спроектувати точки інтеграції, формати даних та механізми взаємодії між компонентами розроблюваної системи.