

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

УДК

«ПОГОДЖЕНО»

«ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ»

Декан факультету
інформаційних технологій

Завідувач кафедри комп'ютерних наук

Глазунова О.Г., д.п.н., професор

Голуб Б.Л., к.т.н., доцент

_____ 202_ р.

_____ 202_ р.

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему Система аналізу трафіку ігрових сервісів на прикладі мережі серверів Minecraft

Спеціальність _____

(код і назва)

Освітня програма _____

(назва)

Орієнтація освітньої програми _____

(освітньо-професійна або освітньо-наукова)

Гарант освітньої програми

_____ (науковий ступінь та вчене звання)

_____ (підпис)

_____ (ПІБ)

Керівник магістерської кваліфікаційної роботи

_____ (науковий ступінь та вчене звання)

_____ (підпис)

Ткаченко Олександр Миколайович
_____ (ПІБ)

Виконав

_____ (підпис)

Авер'янов Денис Сергійович
_____ (ПІБ студента)

КИЇВ-2024

ЗМІСТ

<u>ЗМІСТ</u>	2
<u>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ</u>	4
<u>ВСТУП</u>	5
<u>1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ</u>	8
1.1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ.....	8
1.2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ.....	9
1.3 ПОСТАНОВКА ЗАДАЧІ ДОСЛІДЖЕННЯ	12
<u>2. МОДЕЛЮВАННЯ СИСТЕМИ</u>	14
2.1 ОСНОВНІ ПОЛОЖЕННЯ МОДЕЛЮВАННЯ СИСТЕМИ.....	14
2.2 АРХІТЕКТУРА ПОТОКІВ ДАНИХ МІЖ МОДУЛЯМИ	15
2.3 ДІАГРАМА ПРЕЦЕДЕНТІВ	18
2.4 ДІАГРАМА ПОСЛІДОВНОСТІ.....	20
2.5 ДІАГРАМА АКТИВНОСТІ	22
2.5 ОПИС USER STORY	24
<u>3. РОЗРОБКА СИСТЕМИ</u>	27
3.1 ЗАГАЛЬНА АРХІТЕКТУРА СИСТЕМИ	27
3.2 ВПРОВАДЖЕННЯ ЗБОРУ ДАНИХ.....	30
3.3 ПОЧАТКОВА ОБРОБКА ЗІБРАНИХ ДАНИХ.....	34
3.4 РОБОТА З ДАНИМИ. ОБРОБКА ТА ЗБЕРІГАННЯ.....	36
3.5 ВІЗУАЛІЗАЦІЯ АНАЛІТИЧНИХ ДАНИХ	39
3.6 ІНФРАСТРУКТУРНІ РІШЕННЯ.....	40
<u>4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ</u>	43
4.1 АПАРАТНІ ТА ПРОГРАМНІ ВИМОГИ.....	43
4.2 ХІД ДОСЛІДЖЕННЯ.....	44

	3
4.3 Підсумкові результати дослідження	46
<u>ВИСНОВКИ</u>	49
<u>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ</u>	52
<u>ДОДАТКИ</u>	58

Перелік умовних позначень

API (Application Programming Interface) — інтерфейс для взаємодії між програмними компонентами.

DFD (Data Flow Diagram) — діаграма потоків даних у системі.

ORM (Object-Relational Mapping) — технологія для перетворення даних між об'єктами та реляційними базами.

JDBC (Java Database Connectivity) — API для роботи з базами даних у Java.

REST (Representational State Transfer) — стиль інтеграції систем через HTTP-запити.

ВСТУП

Minecraft — це одна з найпопулярніших багатокористувацьких онлайн-ігор, яка об'єднує мільйони гравців по всьому світу. У цій грі гравці можуть будувати та досліджувати віртуальні світи, взаємодіяти один із одним та виконувати різноманітні завдання. Популярність Minecraft створює значний потік даних, пов'язаний із поведінкою гравців, їхньою активністю та взаємодією з серверами. Ці дані є цінним джерелом інформації для адміністраторів проектів, аналітиків та маркетологів, які працюють над покращенням користувацького досвіду, оптимізацією роботи серверів і формуванням маркетингових стратегій.

Однак, ефективний аналіз трафіку ігрових серверів залишається складним завданням через великий обсяг, швидкість надходження та різноманітність даних. У цьому контексті розробка системи, яка дозволяє збирати, обробляти та візуалізувати дані в реальному часі, є надзвичайно актуальною. Така система здатна не лише допомогти у вирішенні технічних задач, пов'язаних із стабільністю серверів, але й надати цінні інсайти для управління проектами та розробки маркетингових стратегій.

Об'єкт та предмет дослідження

Об'єктом дослідження є процес аналізу трафіку ігрових серверів.

Предметом дослідження є методи та інструменти для збору, обробки, зберігання та візуалізації даних із ігрових серверів Minecraft.

Мета дослідження. Метою дослідження є розробка системи аналізу трафіку, яка дозволить ефективно обробляти великий обсяг даних у реальному часі, забезпечуючи інтерактивну візуалізацію метрик для прийняття стратегічних рішень у проектному та маркетинговому управлінні.

Завдання дослідження. Для досягнення поставленої мети було визначено такі завдання:

1. Провести системний аналіз предметної області для визначення основних вимог до системи.
2. Сформулювати наукові та технічні вимоги до системи аналізу трафіку.
3. Побудувати моделі предметної області та визначити архітектуру системи.
4. Розробити функціональні модулі системи для збору, обробки, зберігання та візуалізації даних.
5. Провести тестування системи, оцінити її ефективність та аналіз отриманих даних.
6. Сформулювати рекомендації щодо подальшого використання результатів.

Методи дослідження. Для виконання завдань було використано такі методи та технології:

- Для збору даних — Velocity Proxy API, Paper API.
- Для потокової обробки даних — Redpanda, Quarkus.
- Для зберігання даних — PostgreSQL, ClickHouse.
- Для візуалізації даних — Vue.js, Highcharts.
- Для моделювання — діаграми UML (діаграми прецедентів, послідовності, активності).
- Для управління інфраструктурою — Kubernetes, Terraform, Ansible.

Наукова новизна. Наукова новизна дослідження полягає в:

- вперше розробленій архітектурі системи аналізу трафіку для ігрових серверів, яка дозволяє обробляти великі обсяги даних у реальному часі;
- запропонованих удосконаленнях алгоритмів обробки подій для зменшення затримок і втрат даних;
- інтеграції сучасних інструментів потокової обробки та візуалізації даних із високонадійною інфраструктурою.

Апробація результатів дослідження. Результати дослідження було представлено на конференціях, а також відображено у наукових публікаціях.

Основні ідеї роботи обговорювалися в професійних спільнотах, що дало змогу врахувати сучасні тенденції та отримати цінні рекомендації щодо вдосконалення системи.

Структура магістерської роботи. Магістерська робота складається з XX сторінок, включає XX розділів, містить XX ілюстрацій, XX таблиць та XX додатків. У роботі використано XX джерел.

- У першому розділі подано аналіз предметної області та визначено основні вимоги до системи.
- У другому розділі розглянуто моделювання системи, представлено діаграми та описано архітектуру.
- Третій розділ присвячено реалізації системи, включаючи розробку функціональних модулів і тестування.
- У четвертому розділі представлено результати дослідження та висновки щодо практичного застосування системи.

Робота спрямована на розробку інноваційного рішення для аналізу трафіку ігрових серверів, що має важливе значення як у науковому, так і в практичному аспектах.

1. СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Предметною областю дослідження є аналіз трафіку в мережах ігрових серверів, зокрема на прикладі серверів Minecraft. Ця область охоплює сукупність процесів і технологій, спрямованих на збір, обробку та аналіз мережевих даних, які генеруються користувачами під час взаємодії з ігровою платформою.

Ігрові сервери, як правило, є основою багатокористувальницьких ігрових середовищ, забезпечуючи з'єднання між гравцями та обслуговуючи великі обсяги трафіку, що включає запити на авторизацію, передачу даних про дії гравців, обробку подій та синхронізацію ігрового процесу. Для серверів Minecraft трафік складається з повідомлень, що включають дані про мережеві з'єднання, місцезнаходження об'єктів, стан гравців, взаємодії між ними, а також інформацію про ігрові світи і сервери.

Ключовими характеристиками предметної області є:

- **динамічність трафіку:** значні пікові навантаження можуть виникати через синхронний вхід великої кількості гравців або активні ігрові події. Це вимагає ефективних механізмів для забезпечення надійності і швидкості обробки;
- **обсяг даних:** щохвилини генерується великий обсяг трафіку, який потребує обробки в реальному часі для забезпечення безперервності ігрового процесу;
- **необхідність у високій доступності:** сервери мають працювати безперебійно, навіть у випадках пікових навантажень, що вимагає масштабованої архітектури;
- **вимоги до аналітики:** важливо забезпечити можливість моніторингу активності користувачів, аналізу їхньої поведінки та виявлення можливих проблем, таких як затримки, баги або аномальна активність.

Окрім технічного забезпечення, аналітика трафіку є критично важливою для підвищення залучення гравців та покращення користувацького досвіду. Це

дозволяє зібрати корисну інформацію про вподобання аудиторії, створювати персоналізовані рекомендації та оптимізувати ігровий процес, орієнтуючись на реальні дані.

Таким чином, предметна область поєднує технічні, інфраструктурні та аналітичні аспекти, які створюють основу для ефективного управління ігровими проектами та покращення взаємодії з гравцями.

1.2 Аналіз існуючих рішень

У сучасному середовищі ігрових серверів Minecraft існують різні рішення для аналізу трафіку, моніторингу активності користувачів та управління мережею. Серед найбільш поширених інструментів виділяються Plan та Network Manager. Ці платформи мають свої унікальні підходи до збору й аналізу даних, проте їх функціональність обмежена у деяких аспектах.

Plan (Player Analytics) [1] — це інструмент для збору та аналізу даних про активність користувачів на сервері Minecraft (рис. 1.1).

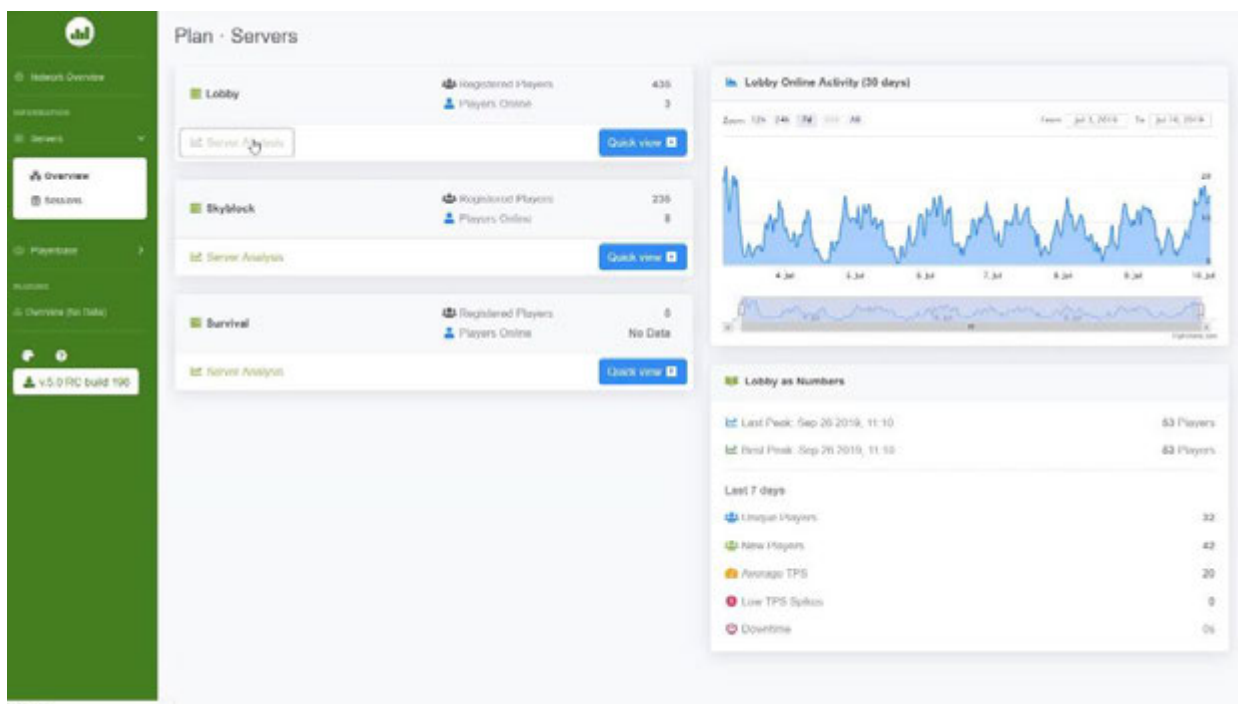


Рис. 1.1 Інтерфейс головного екрану Plan

Його основними функціями є:

- **аналітика гравців:** збір даних про ігрові сесії, час, проведений у грі, кількість входів і виходів, а також загальна активність кожного гравця;
- **моніторинг серверу:** відстеження продуктивності сервера, затримок, частоти кадрів (TPS) та використання ресурсів (пам'яті й процесора, диску);
- **візуалізація даних:** інтерактивні дашборди, які дозволяють зручно переглядати інформацію у вигляді графіків і таблиць.

Обмеження Plan:

- орієнтованість лише на дані гравців і продуктивність сервера без глибокої інтеграції з потоковими системами обробки трафіку;
- відсутність механізмів для аналізу взаємодії між серверами у кластері або багатосерверних топологіях;
- менша гнучкість для адаптації під специфічні потреби масштабних проектів.

Network Manager [2] — це інструмент для управління мережею серверів Minecraft, який надає можливість моніторингу трафіку і ефективного управління з'єднаннями між серверами, див рис. 1.2.



Рис. 1.2 Інтерфейс головного екрану Network

Основні функції:

1. **моніторинг трафіку між серверами:** аналіз вхідного і вихідного трафіку, виявлення аномальної активності;
2. **мерування мережею серверів:** оптимізація маршрутизації трафіку між окремими серверами в кластері;
3. **інтеграція з іншими плагінами:** забезпечення гнучкості в роботі із зовнішніми інструментами для обробки даних.

Обмеження Network Manager:

- відсутність розширеної візуалізації даних або механізмів збереження історичних даних для подальшого аналізу;
- орієнтованість на невеликі або середні мережі серверів, що ускладнює масштабування на великі платформи.

Основне обмеження існуючих інструментів — їх спеціалізація на окремих аспектах роботи серверів. Plan забезпечує базовий аналіз активності гравців, але не охоплює складні сценарії аналізу трафіку або інтеграцію з потоковими системами обробки даних. Network Manager фокусується на мережевих з'єднаннях і трафіку, але не має достатньо гнучкості для розширеного аналізу та візуалізації.

Тож, рішення має можливості поєднання обох підходів, які виходять за їх межі, використовуючи:

- потокову обробку даних у реальному часі (на базі Redpanda);
- систему збереження великих обсягів даних із низькими затримками (на базі ClickHouse);
- високодоступну архітектуру з кластерною підтримкою (Kubernetes) для масштабованості;
- динамічну візуалізацію результатів аналізу для глибшого розуміння поведінкових патернів користувачів.

Таким чином, це усуває обмеження існуючих рішень і пропонує більш комплексний підхід до аналізу трафіку мережі ігрових серверів Minecraft.

1.3 Постановка задачі дослідження

Проблематика аналізу трафіку ігрових серверів, зокрема серверів Minecraft, зумовлена швидким зростанням кількості онлайн-ігор, а також підвищенням вимог до якості обслуговування користувачів. Головною задачею в цьому контексті є створення ефективної системи, яка дозволить у реальному часі обробляти великі обсяги мережевих даних, надавати аналітичну інформацію для оптимізації роботи серверів та сприяти покращенню користувацького досвіду.

Виходячи з актуальності цієї тематики, сформульовано основну наукову проблему: розробка системи аналізу трафіку для моніторингу активності користувачів з метою ведення ефективної маркетингової стратегії та забезпечення безперервної роботи високонадійної аналітичної системи у масштабованій інфраструктурі з високим навантаженням.

Для вирішення поставленої проблеми визначено такі задачі:

1. провести аналіз існуючих рішень для моніторингу та обробки трафіку ігрових серверів Minecraft, виявити їхні обмеження та можливості;
2. розробити функціональну архітектуру системи, яка включатиме:
 - потокову обробку даних для аналізу у реальному часі;
 - надійні алгоритми для класифікації та агрегації подій користувачів;
 - моніторинг активності користувачів у реальному часі, для оцінки поведінкових патернів та визначати ключові метрики;
 - високонадійні обробку та збереження без втрат великих обсягів аналітичних даних з мінімальними затримками;
 - механізми візуалізації в обраному діапазоні та реальному часі, зі зручним представленням результатів аналізу.

3. створити механізми інтеграції системи з інструментами управління трафіком, такими як Velocity Proxy та Paper API, для покращення маршрутизації та збору даних;
4. впровадити систему відстеження ефективності маркетингових стратегій, враховуючи доступні аналітичні дані;
5. розробити високонадійну кластерну архітектуру на основі Kubernetes, яка забезпечить масштабованість і стійкість до пікових навантажень;
6. впровадити сучасні системи зберігання даних, зокрема ClickHouse для швидкої аналітики та PostgreSQL для реляційного зберігання, оптимізуючи їхнє використання;
7. провести дослідження за допомогою розробленої системи на реальних даних реальної мережі ігрових серверів.

Очікувані результати вирішення цих задач сприятимуть підвищенню ефективності роботи серверів Minecraft, створенню адаптивних маркетингових стратегій та покращенню користувальницького досвіду за рахунок глибшого розуміння поведінки гравців. Вирішуючи зазначені задачі, дослідження також формує основу для подальшого розвитку систем аналізу трафіку в інших галузях.

2. МОДЕЛЮВАННЯ СИСТЕМИ

2.1 Основні положення моделювання системи

Моделювання системи є ключовим етапом розробки, який дозволяє створити формалізоване уявлення про структуру, функціональність та взаємодію компонентів системи.

Об'єктно-орієнтоване моделювання є підходом до проектування, який зосереджується на поданні системи як сукупності об'єктів, що мають властивості (атрибути) і виконують певні дії (методи). У контексті розробки системи аналізу трафіку ігрових серверів Minecraft цей підхід дозволяє більш детально моделювати компоненти системи, їхні взаємодії та поведінку, забезпечуючи високу структурованість і гнучкість у поданні складних процесів. Для створення моделей системи застосовуються такі типи діаграм:

1. DFD (Data Flow Diagram) — це графічне представлення потоків даних у системі, яке описує, як дані переміщуються між процесами, зберіганнями, зовнішніми джерелами та споживачами [3]. Ця діаграма дозволяє візуалізувати основні процеси обробки інформації, а також точки передачі та трансформації даних у системі. Вона зосереджується на потоці даних, а не на їхній структурі, що робить її ефективним інструментом для моделювання та аналізу інформаційних систем. DFD використовується для планування архітектури системи, а також для виявлення "вузьких місць" або неефективних етапів у обробці даних;
2. діаграма прецедентів (Use Case Diagram) — це візуалізація, яка відображає взаємодію користувачів (акторів) із системою через різні сценарії використання (use cases) [4]. Вона дозволяє зрозуміти, які функції забезпечує система, та які зовнішні стейкхолдери (наприклад, гравці або аналітики) взаємодіють із цими функціями. Ця діаграма використовується для визначення ключових можливостей системи з точки зору користувачів;
3. діаграма послідовності (Sequence Diagram) — моделює динамічну взаємодію між компонентами системи у часовій послідовності. Вона

- показує, які об'єкти взаємодіють між собою, які повідомлення вони передають і в якому порядку. Такий підхід дозволяє деталізувати поведінку системи у певному сценарії, наприклад, як дані передаються від серверів через обробник до бази даних і потім до модулів візуалізації [5];
4. діаграма активності (Activity Diagram) — відображає послідовність виконання дій у межах одного чи кількох процесів. Вона моделює потік управління, розгалуження рішень та паралельне виконання операцій. Це інструмент для опису складних процесів, таких як обробка подій користувача, включаючи фільтрацію, агрегацію та запис даних у базу [6];
 5. опис User Story — слугує для опису сценаріїв взаємодії користувачів із системою з точки зору їхніх потреб. Показує, як користувачі виконують певні дії або досягають конкретних цілей у межах системи. Часто використовуються для формування вимог до системи та деталізації очікуваної поведінки [7].

Усі ці діаграми доповнюють одна одну, формуючи цілісну картину роботи системи: від високорівневих сценаріїв використання до деталізації процесів і динаміки взаємодії компонентів.

2.2 Архітектура потоків даних між модулями

Процес моделювання системи аналізу трафіку ігрових серверів Minecraft включає опис етапів, які відповідають ключовим завданням обробки даних. Основною метою моделювання є створення логічної структури, яка забезпечить коректну реалізацію збору, обробки, збереження та візуалізації даних. Це дозволяє гарантувати ефективність і надійність системи навіть у випадках високого навантаження.

Для опису передачі даних у системі використовується функціональний підхід, який дозволяє зосередитися на послідовності виконання дій, функціях кожного модуля та точках їхньої взаємодії. Такий підхід формує основи для

побудови функціональної моделі, яка візуалізує передачу даних між компонентами системи. Представлена модель (рис. 2.1) є основою для проєктування і реалізації надійної системи аналізу трафіку ігрових серверів Minecraft.

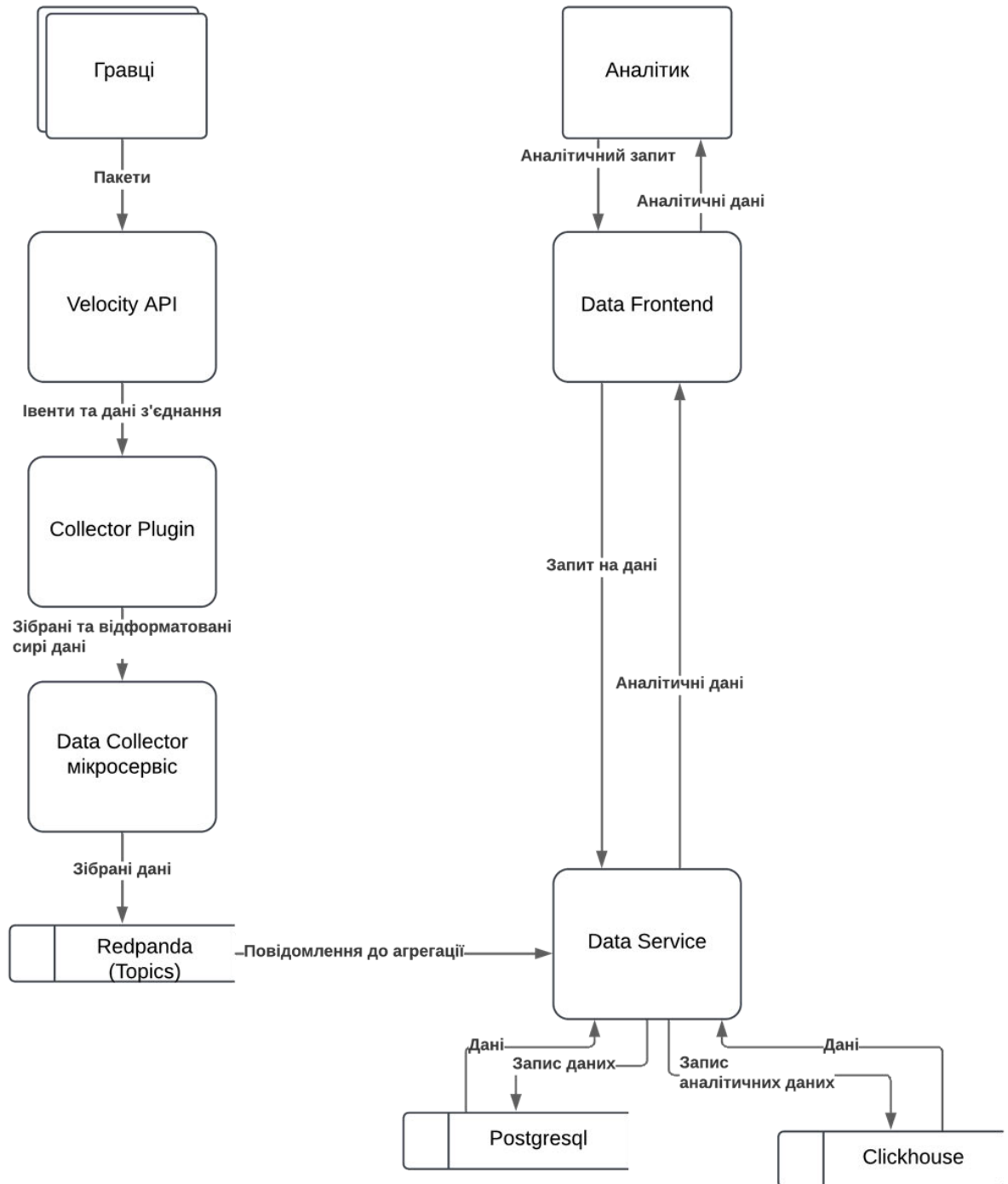


Рис. 2.1 DFD діаграма системи

Представлена діаграма демонструє функціональний потік даних у системі аналізу трафіку. Вона відображає взаємодію між ключовими компонентами системи та етапи обробки даних від джерела (гравці) до їхньої візуалізації та зберігання. Основні блоки діаграми та їхній опис:

1. гравці є джерелом даних, які генеруються під час взаємодії з ігровими серверами. Ці дані включають інформацію про події (вхід, вихід, переміщення, дії в грі) та метрики роботи серверів. Дані передаються у вигляді пакетів до системи через Velocity API;
2. проксі Velocity API отримує пакети даних від гравців, що містять сесійні дані, інформацію про події та з'єднання. Velocity API виконує роль інтерфейсу між ігровими серверами та системою збору даних;
3. встановлений Collector Plugin отримує дані від Velocity API, виконує їх первинну обробку та форматування. На цьому етапі дані структуруються для подальшої передачі до мікросервісу збору даних;
4. мікросервіс Data Collector приймає зібрані та відформатовані дані від Collector Plugin та передає їх у систему потокової обробки. Цей компонент забезпечує передачу даних у Redpanda для подальшої агрегації;
5. стрімінг Redpanda використовується для обробки даних у режимі реального часу. Вона створює канали (topics), через які дані передаються до наступних компонентів системи. Цей етап забезпечує надійну передачу та можливість обробки великих обсягів даних;
6. мікросервіс Data Service є основним аналітичним компонентом системи. Він отримує дані від Redpanda, виконує агрегацію, фільтрацію та обчислення ключових аналітичних метрик. Оброблені дані передаються до:
 - PostgreSQL для зберігання структурованих реляційних даних;
 - ClickHouse для зберігання великих обсягів аналітичних даних, оптимізованих для швидкого доступу.

Крім того, Data Service відповідає за обробку запитів від Data Frontend і передає аналітичні дані для їхньої візуалізації;

7. Інтерфейс Data Frontend виступає інтерфейсом для аналітичної системи. На основі запитів від аналітиків система генерує аналітичні звіти та візуалізує ключові метрики. Взаємодія з Data Service забезпечує отримання актуальних даних.

Ця діаграма наочно демонструє взаємодію компонентів системи на етапі моделювання. Вона описує логіку роботи ключових модулів і дозволяє зрозуміти, як будуть оброблятися та передаватися дані у майбутній реалізації системи.

Дані генеруються гравцями, передаються через Velocity API до Collector Plugin і мікросервісу Data Collector. Потім вони обробляються у Redpanda та передаються до Data Service, який здійснює аналітику. Оброблені дані зберігаються в базах даних (PostgreSQL і ClickHouse) або візуалізуються через Data Frontend.

2.3 Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) демонструє основні сценарії використання системи аналізу трафіку ігрових серверів Minecraft різними категоріями користувачів. Ця діаграма допомагає візуалізувати, які функції виконує система, як її компоненти взаємодіють із користувачами та які бізнес-процеси підтримуються.

Основні актори системи:

- Аналітик. Представляє користувача, який виконує аналіз даних. Аналітик використовує систему для збору інформації, її візуалізації та створення live-аналітики. Його основна мета – отримати детальні статистичні дані для підтримки управлінських рішень.
- Розробник. Цей актор відповідає за підтримку і масштабування системи. Розробник взаємодіє із системою для збору телеметрії, створення івентів та управління інфраструктурою.

- Гравець. Є джерелом даних для системи, генеруючи події під час гри. Гравець надсилає телеметрію, необроблені дані та проходить авторизацію для взаємодії із сервером.
- Проху. Представляє собою Velocity Proxy або Paper Server, який забезпечує обробку запитів від гравців, синхронізацію налаштувань та перевірку стану системи (health check).

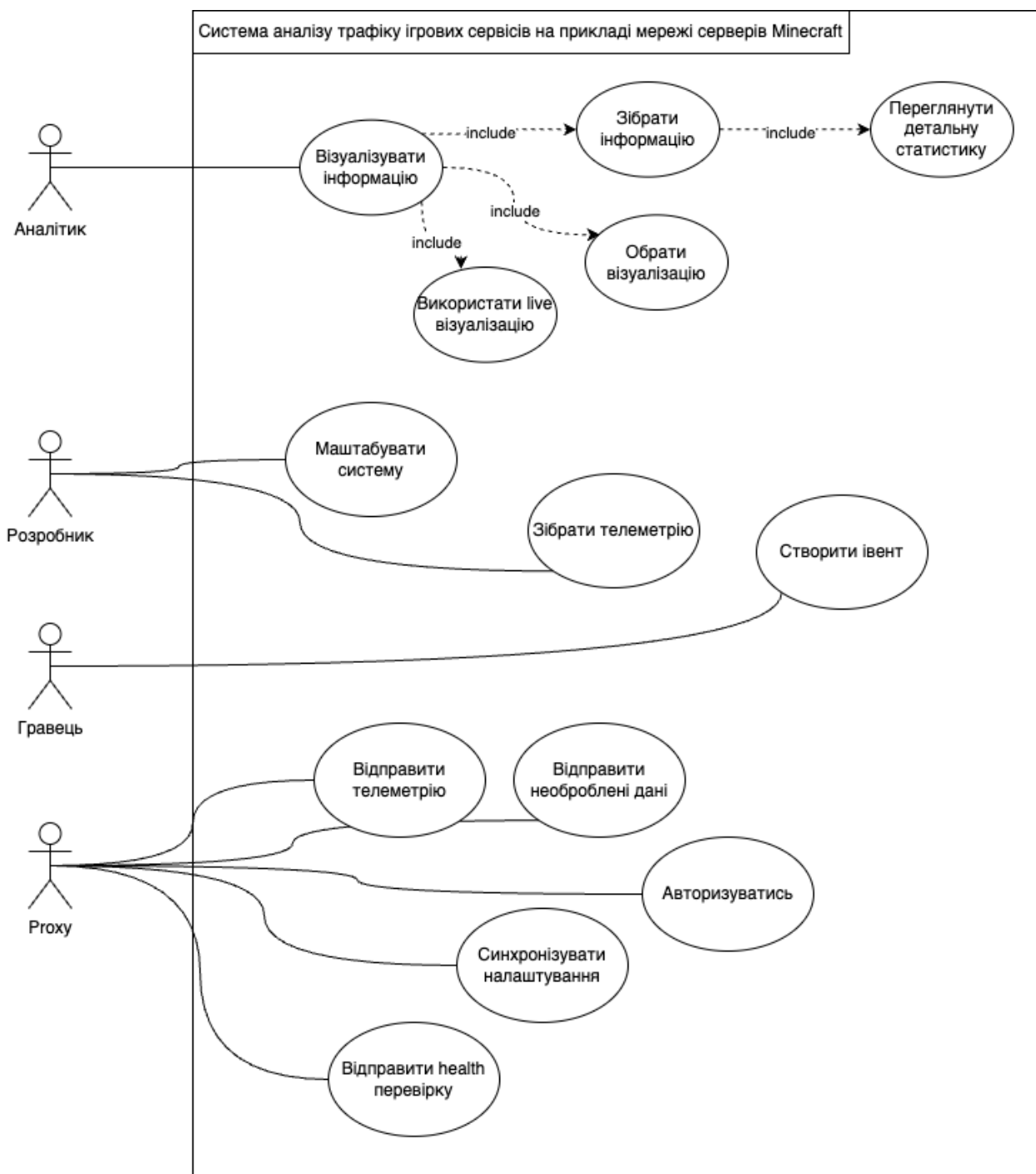


Рис. 2.2 Діаграма Use Case

Сценарії використання:

1. Візуалізація інформації. Аналітик збирає дані, обирає формат візуалізації та може використовувати live-аналітику для оперативного перегляду показників. Після цього він отримує доступ до детальної статистики, яка дозволяє виконати глибокий аналіз (use cases: "Зібрати інформацію", "Обрати візуалізацію", "Переглянути детальну статистику").
2. Масштабування системи. Розробник виконує дії, спрямовані на забезпечення стабільності системи та її розширення, включаючи збирання телеметрії для діагностики проблем і створення нових івентів для обробки специфічних подій.
3. Передача даних гравцем. Гравець генерує події у процесі гри, які передаються у вигляді телеметрії або необроблених даних. Це включає авторизацію для доступу до серверів (use cases: "Відправити телеметрію", "Відправити необроблені дані", "Авторизуватись").
4. Дії Proxy. Proxy забезпечує синхронізацію налаштувань між клієнтами, виконання перевірки стану системи (health check) та передачу телеметрії. Він діє як центральний вузол, що обробляє запити від гравців і передає їх на сервери.

2.4 Діаграма послідовності

Діаграма послідовності (Sequence Diagram) ілюструє динамічну взаємодію між компонентами системи аналізу трафіку під час виконання запиту на візуалізацію аналітичних даних. Вона демонструє, як дані передаються між модулями системи у часі, підкреслюючи порядок викликів і обміну повідомленнями (рис. 2.3). Учасники процесу:

1. аналітик – користувач системи, який ініціює запит на аналітику;
2. інтерфейс Data Frontend – модуль, що отримує запит від аналітика та відповідає за формування інтерфейсу користувача;

3. мікросервіс Data Service – центральний обробник запиту, який відповідає за збір, обробку та агрегацію даних;
4. сховище даних ClickHouse – система збереження аналітичних даних, оптимізована для виконання складних запитів;
5. база даних PostgreSQL – база даних для збереження реляційної інформації.

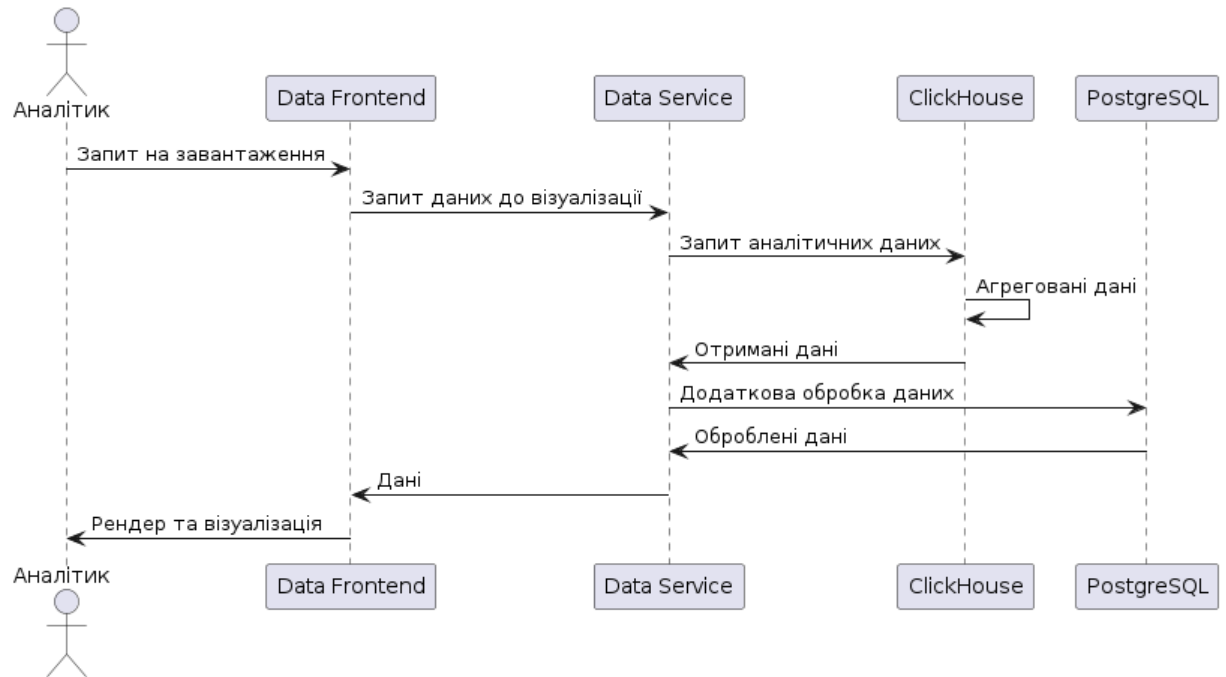


Рис. 2.3 Отримання аналітичних даних. Діаграма послідовності

Послідовність дій:

1. Аналітик ініціює запит через Data Frontend. Цей запит містить інформацію про необхідні метрики чи діапазон даних, які потрібно візуалізувати.
2. Data Frontend передає запит на візуалізацію до Data Service для обробки.
3. Data Service запитує аналітичні дані з бази даних ClickHouse. У цей момент відбувається перевірка та агрегування великих обсягів даних.
4. ClickHouse повертає агреговані дані до Data Service, які можуть потребувати додаткової обробки.
5. У разі потреби Data Service звертається до бази даних PostgreSQL для отримання додаткових реляційних даних, таких як параметри конфігурації чи користувацька інформація.

6. Оброблені дані повертаються в Data Frontend, який відповідає за їх рендеринг.
7. Data Frontend формує графіки та таблиці, використовуючи інструменти візуалізації, такі як Vue.js і Highcharts.
8. Аналітик отримує готову візуалізацію результатів.

Цей процес забезпечує інтегровану роботу компонентів для швидкого отримання та відображення аналітичних даних. Використання сховища даних ClickHouse для великих обсягів аналітики та бази даних PostgreSQL для структурованих метаданих дозволяє ефективно поєднати високу продуктивність і гнучкість у роботі з даними. Крім того, динамічний рендеринг інтерфейсу забезпечує зручність для користувачів, дозволяючи отримувати актуальні метрики в реальному часі.

Діаграма послідовності чітко відображає ключові етапи взаємодії між компонентами системи, акцентуючи увагу на передачі даних і логіці виконання запиту. Такий підхід забезпечує прозорість архітектури та дозволяє ідентифікувати потенційні точки оптимізації в роботі системи.

2.5 Діаграма активності

Діаграма активності на рис. 2.4 відображає шлях, який проходять дані від моменту отримання події від гравця до їх візуалізації аналітиком через інтерфейс системи. Процес починається з отримання події через Proху. Цей компонент виступає першим вузлом обробки, де подія фільтрується та передається до Collector Plugin. У цьому модулі виконується початкова перевірка валідності, яка визначає, чи є подія релевантною для подальшого аналізу. Нерелевантні події відхиляються на цьому етапі, що дозволяє системі зосередитися лише на критично важливих даних.

Якщо подія визнана валідною, дані передаються до Data Collector, який передає їх у Redpanda. Поточкова обробка через Redpanda дозволяє системі

швидко передати подію до Data Service. Data Service, у свою чергу, виступає основним аналітичним модулем, який отримує дані, виконує їх обробку, включаючи агрегацію та додавання метаданих, і готує їх до збереження. Оброблені дані зберігаються у двох базах даних: ClickHouse для аналітичних даних і PostgreSQL для реляційної інформації. Завершення цього етапу супроводжується підтвердженням про успіх, що гарантує коректність обробки.

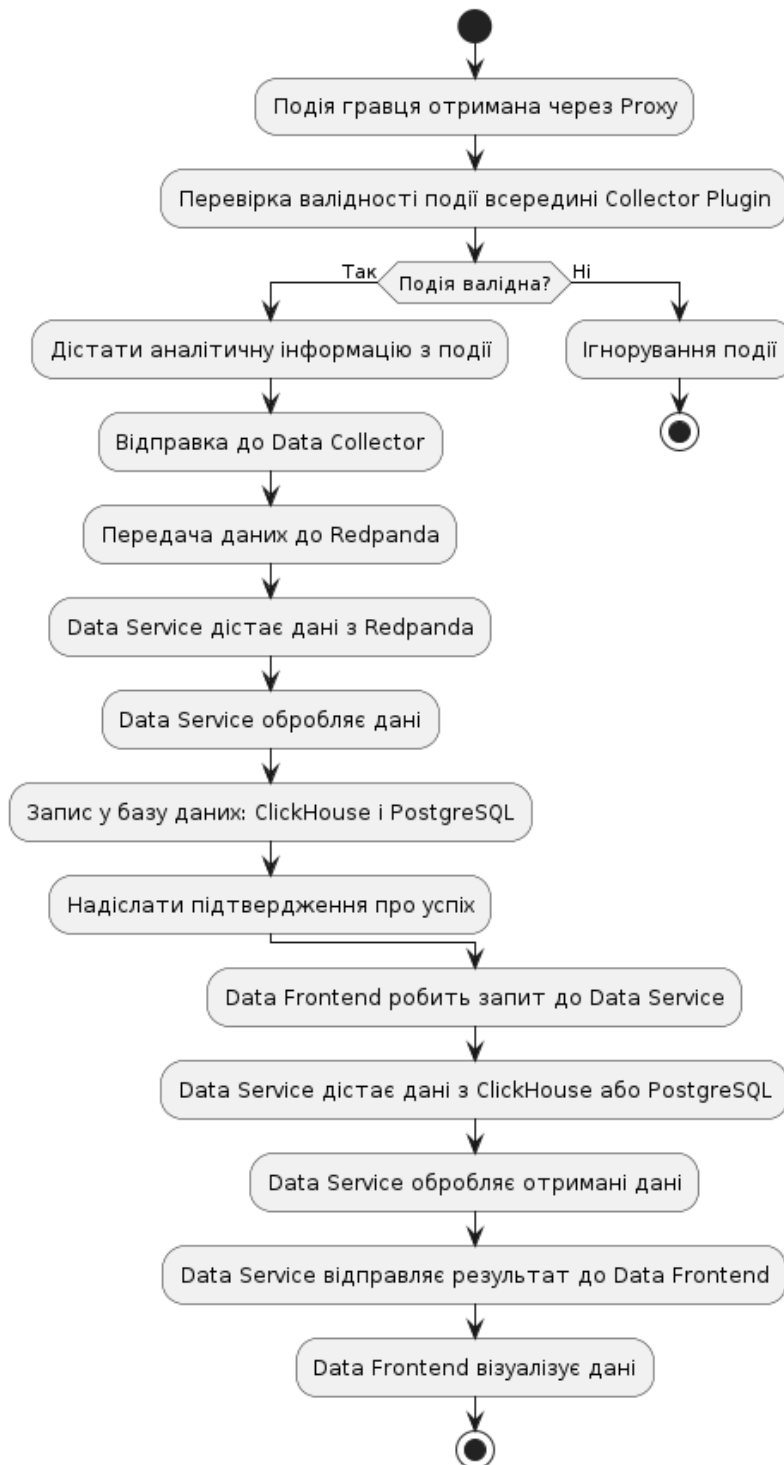


Рис. 2.4 Activity Diagram проходження від події до візуалізації

Для live візуалізації даних Data Frontend здійснює запит до Data Service, що є критично важливим етапом. Data Service отримує необхідну інформацію з баз даних, обробляє її відповідно до запиту та передає назад у Data Frontend. Цей модуль відповідає за створення графічного представлення даних у вигляді інтерактивних дашбордів. Завдяки цьому аналітик може в режимі реального часу спостерігати за ключовими метриками та використовувати їх для прийняття оперативних рішень.

Діаграма демонструє високу інтеграцію між модулями системи, що забезпечує ефективну обробку великого обсягу подій і швидкий доступ до аналітичних даних. Логіка побудови процесу враховує всі важливі етапи обробки, що робить систему стійкою до навантажень і здатною забезпечити live-аналітику для користувачів.

2.5 Опис User Story

User Story — це спосіб опису функціональності системи, який фокусується на кінцевих користувачах та їхніх потребах. Цей підхід фокусується на тому, що користувач хоче отримати від системи, а також на меті та критеріях прийнятності. Нижче наведено дві ключові історії користувачів, які стосуються візуалізації аналітичних даних та масштабування систем.

Описана в табл. 2.1 User Story описує сценарій взаємодії аналітика із системою. Основна задача аналітика — отримати чітку візуалізацію даних, що дозволяє швидко аналізувати ключові показники системи. Відповідно до цієї User Story, аналітик заходить у систему, переходить на сторінку аналітики, обирає бажаний тип візуалізації (графіки чи таблиці) та отримує відображення даних у реальному часі.

Дотримання чітких критеріїв прийнятності (Acceptance Criteria) забезпечує, що аналітик отримає зручний інтерфейс із можливістю вибору та коректним відображенням інформації.

Це дозволяє оперативно аналізувати дані та приймати обґрунтовані рішення.

Таблиця 2.1 — User Story отримання візуалізації аналітичних даних

US Code	1
As a	Аналітик
I want	Візуалізувати дані
In order to	Отримати розуміння ключових показників системи
Acceptance Criteria	
Given	Я заходжу в систему
Then	Я переходжу на сторінку аналітики
And	Обираю відповідний тип візуалізації
Then	Я бачу відображення потрібних даних у вигляді графіків чи таблиць

В табл. 2.2 User Story описує потребу розробника в масштабуванні системи під час високого навантаження. Система має залишатися стабільною навіть за умови зростання кількості запитів чи активних користувачів. Розробник отримує можливість аналізувати поточні метрики навантаження, виконувати запити на відображення цих метрик, а потім додавати нові вузли до кластеру для підтримки стабільної роботи.

Таблиця 2.2 — User Story масштабування системи

US Code	2
As a	Розробник
I want	Масштабувати систему
In order to	Забезпечити стабільну роботу під час високих навантажень
Acceptance Criteria	

Given	Система працює у режимі підвищеного навантаження
Then	Відправляю запити на відображення навантаження
And	Бачу метрики навантаження сервісів
Then	Додаю нові ноди

Ключові критерії прийнятності включають доступність даних про навантаження, можливість швидкого реагування на зміни та інтеграцію нових вузлів без перерви в роботі системи.

Описані User Story демонструють різні аспекти функціональності системи. Перша спрямована на зручність і точність аналітичного інтерфейсу для кінцевих користувачів, друга — на гнучкість і масштабованість системи для забезпечення її стабільної роботи. Такий підхід забезпечує відповідність системи потребам різних категорій користувачів, враховуючи як кінцевих аналітиків, так і технічний персонал.

3. РОЗРОБКА СИСТЕМИ

3.1 Загальна архітектура системи

Архітектура системи базується на інтеграції сучасних технологій, що забезпечують її масштабованість, продуктивність та високу надійність у роботі з великими обсягами трафіку. Система призначена для моніторингу активності користувачів на серверах Minecraft, обробки отриманих даних у реальному часі, їх збереження та подальшої візуалізації для прийняття стратегічних рішень.

Основними компонентами архітектури є:

1. Плагін збору даних:

- взаємодія з ігровими серверами через Velocity API та Paper API, що дозволяє отримувати дані про події, активність користувачів і навантаження у реальному часі;
- дані, зібрані цими API, передаються у потік для подальшої обробки.

2. Мікросервіс обробки даних:

- використання Quarkus як фреймворку для швидкої та оптимізованої обробки потоків даних;
- дані проходять етапи фільтрації, агрегації та класифікації подій для отримання ключових метрик активності користувачів.

3. Система збереження даних:

- ClickHouse використовується для роботи з великими обсягами даних, забезпечуючи швидкий доступ і виконання складних аналітичних запитів;
- PostgreSQL доповнює архітектуру, забезпечуючи реляційне зберігання метаданих, пов'язаних із користувачами та системними конфігураціями.

4. Інфраструктурна частина:

- Kubernetes відповідає за оркестрацію контейнерів, що забезпечує масштабованість системи та її стійкість до навантажень;

- Terraform використовується для автоматизації розгортання інфраструктури, а Ansible — для налаштування та управління серверами;

5. Мікросервіс візуалізації:

- інтерфейс системи побудований на основі Vue.js, що дозволяє створювати динамічні та інтерактивні веб-додатки;
- для побудови графіків і дашбордів використовується Highcharts, що забезпечує зручне представлення даних у вигляді візуалізації.

6. Модуль безпеки та маршрутизації:

- для забезпечення безпеки даних і оптимальної маршрутизації використовується Cloudflare DNS, що також захищає систему від DDoS-атак і оптимізує доступ користувачів.

Архітектура системи є модульною, що дозволяє легко інтегрувати нові компоненти або змінювати існуючі, див. рис. 2.1. Такий підхід забезпечує її гнучкість і готовність до розширення в умовах зростання навантаження або зміни вимог.

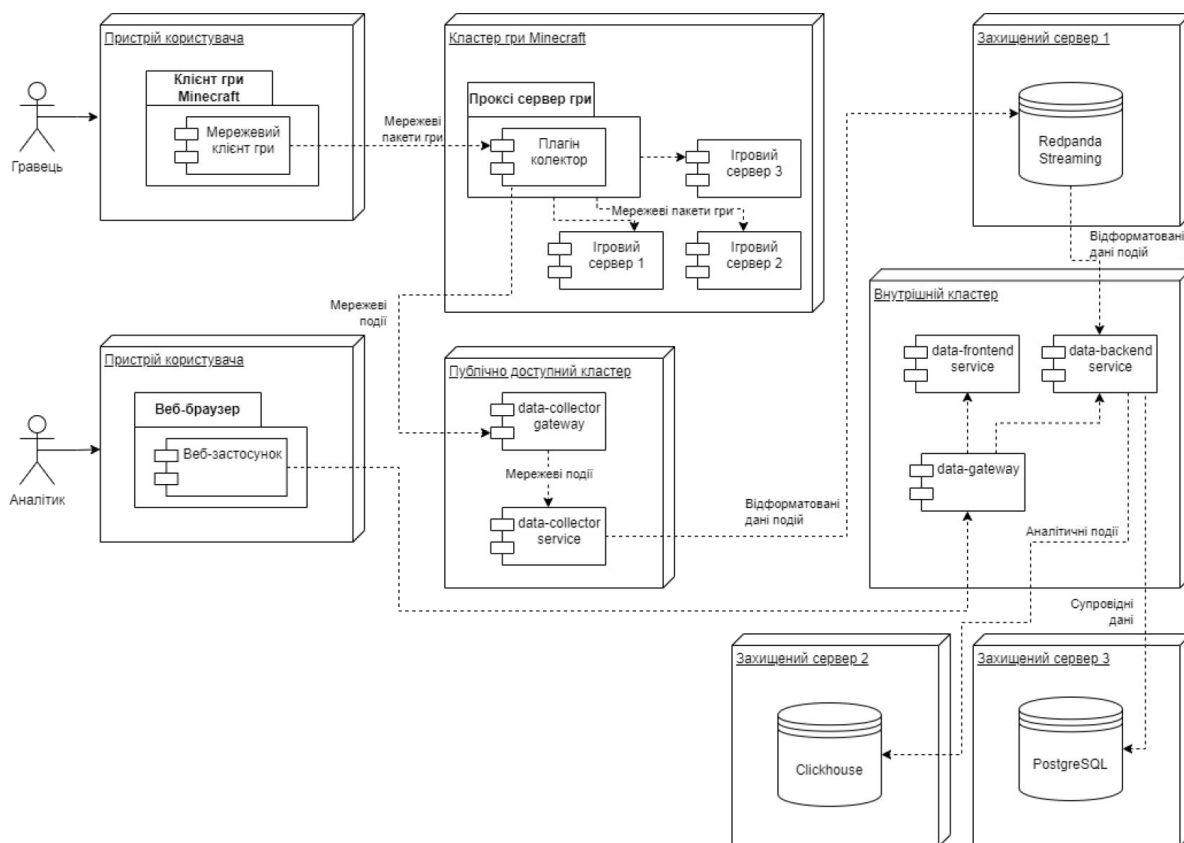


Рис. 3.1 Топологія системи

Сукупність використаних технологій забезпечує високий рівень автоматизації, стабільності та продуктивності, що робить систему надійним інструментом для аналізу ігрового трафіку в реальному часі.

Архітектура системи передбачає кілька ключових етапів обробки даних, які забезпечують їх ефективний збір, трансформацію, зберігання та візуалізацію,. Кожен етап є окремим функціональним модулем, що взаємодіє з іншими через стандартизовані інтерфейси.

Етапи обробки даних:

1. Збір даних. Дані отримуються із серверів Minecraft за допомогою інтеграції з Velocity API та Paper API. Ці API дозволяють фіксувати події користувачів, такі як вхід і вихід, мережеві дані, переміщення, використання об'єктів та інші взаємодії. Отримані дані передаються в обробник для подальшої фільтрації та агрегації.
2. Обробка даних. На цьому етапі дані проходять попередню обробку, яка включає:
 - Валідацію подій на коректний формат даних.
 - Відкидання подій, які не є релевантними для аналітики (наприклад, технічні системні події).
3. Агрегація даних. Фільтровані події групуються для формування ключових метрик, таких як:
 - Середня тривалість сесії користувача.
 - Кількість активних гравців у певний момент часу.
 - Частота викликів конкретних подій.
4. Збереження даних
 - Великі обсяги агрегованих даних записуються у ClickHouse, що дозволяє зберігати їх у стислому вигляді та виконувати складні аналітичні запити з низькими затримками.
 - Реляційні дані, наприклад профілі користувачів або системні мета-дані, зберігаються у PostgreSQL для швидкого доступу до специфічної інформації.

5. Візуалізація даних. Оброблені дані передаються до веб-інтерфейсу, створеного на основі Vue.js. Результати аналізу представлені у вигляді інтерактивних графіків і дашбордів, реалізованих через Highcharts. Це забезпечує зручний доступ до метрик та їх аналізу.

3.2 Впровадження збору даних

Для збору даних у системі було використано API Velocity Proxy, що є одним із ключових компонентів інфраструктури ігрових серверів. Velocity Proxy — це високопродуктивний проксі-сервер, розроблений для об'єднання кількох ігрових серверів у єдину мережу [8]. Його основною перевагою є здатність обробляти великий обсяг підключень із мінімальними затримками, що робить його ідеальним вибором для реалізації системи збору даних у реальному часі. Саме завдяки високій продуктивності та зручному API, Velocity Proxy було обрано як базу для роботи із серверними івентами.

Velocity — це більше, ніж оптимізація роботи з протоколом Minecraft. Розумна робота з протоколом дає неймовірний приріст продуктивності, але для більшої продуктивності нам піти набагато глибше.

Один із способів, за допомогою якого значно підвищується продуктивність і пропускна здатність, — це покращення швидкості стиснення пакетів, що надсилаються клієнту. На підтримуваних платформах (Linux x86_64 та aarch64) Velocity може замінити бібліотеку zlib (яка реалізує алгоритм стиснення, що використовується протоколом Minecraft) на libdeflate, яка вдвічі швидша за zlib, забезпечуючи аналогічний ступінь стиснення.

У контексті ігрових серверів Minecraft івенти — це події, структуровані представлення певної дії або зміни, яку виконав гравець, наприклад, авторизація, початок гри або взаємодія з об'єктом. API Velocity забезпечує створення подій, які містять ключові дані про стан гравця, ігрову сесію та інші дані. є ключовим джерелом інформації про активність користувачів, роботу серверів і взаємодії в

ігровому світі. Наприклад, івент може виникати під час підключення гравця до сервера, що і є потрібним для системи.

Івенти використовуються як основа для збору телеметрії, їхня структура визначає тип і формат даних, які передаються до системи для подальшої обробки. У Velocity API і Paper API івенти надають великий обсяг даних, котрі можливо використати після обробки для моніторингу та аналізу активності у режимі реального часу.

Крім того, було додано сумісність із Paper API для забезпечення гнучкості та додаткової функціональності. Paper [9] — це оптимізована версія серверної платформи Minecraft (форк Spigot), яка пропонує розширений API для роботи з ігровими подіями та об'єктами. Paper API надає інструменти для інтеграції специфічних функцій, таких як глибокий аналіз ігрової телеметрії, що дозволяє зібрати більше контексту під час обробки подій.

Spigot — це оптимізована серверна платформа для Minecraft, яка є форком (похідною версією) від Bukkit [10]. Вона призначена для покращення продуктивності серверів і розширення функціональності завдяки додатковим налаштуванням і API. Spigot забезпечує зниження затримок під час обробки ігрових подій, підтримку великої кількості плагінів і покращене управління ресурсами серверу, що робить його популярним серед адміністраторів великих мереж Minecraft.

Bukkit — це серверний API та платформа для Minecraft, яка дозволяє розробникам створювати плагіни для додавання або модифікації функціональності гри [11]. Bukkit став основою для багатьох інших серверних платформ, таких як Spigot і Paper, забезпечуючи стабільну базу для інтеграції кастомного функціоналу. Його гнучкість і легкість у використанні зробили його одним із найпоширеніших інструментів для створення модифікованих серверів Minecraft.

Spigot, як наступник Bukkit, зберігає сумісність із плагінами Bukkit, водночас надаючи значні покращення у продуктивності та стабільності.

Для збору та обробки івентів було розроблено спеціальний плагін, який працює за допомогою API Velocity. Цей плагін обробляє дані, що надходять із подій, таких як підключення гравців, взаємодії в грі або відправка системних повідомлень. Дані, які генеруються івентами Velocity, після початкової обробки плагіном стають структурованими та готовими для подальшої передачі в систему аналітики.

Передача оброблених даних до аналітичної системи здійснюється через REST API у Data Collector. REST API — це архітектурний стиль для взаємодії між компонентами системи через стандартні HTTP-запити [12]. Він був обраний завдяки його простоті, гнучкості та широкій підтримці у сучасних системах. Використання REST API забезпечує незалежність аналітичної системи від внутрішньої інфраструктури ігрових серверів, що дозволяє системі залишатися масштабованою та ізольованою від потенційних проблем у мережі серверів Minecraft.

Відправка даних через REST API дозволяє створити модульну архітектуру, де аналітична система працює як незалежний сервіс. Такий підхід полегшує її масштабування та інтеграцію з іншими сервісами в майбутньому, забезпечуючи високу продуктивність і стійкість до навантажень (див. рис. 3.2).

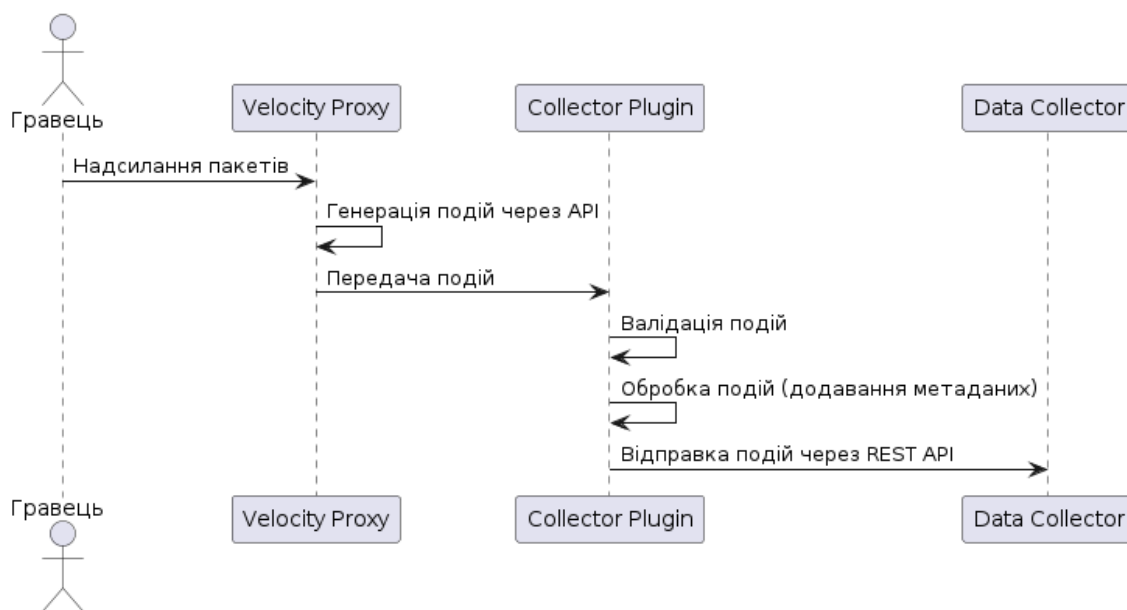


Рис. 3.2 Процес від створення пакету до відправки у вхідну точку аналітичної системи

Коли гравець взаємодіє із сервером Minecraft, процес обробки його пакетів у системі збору даних відбувається у кілька послідовних етапів. Ось як ці пакети передаються і обробляються:

1. Отримання пакетів гравця через Velocity Proxy. Після того як гравець підключається до мережі серверів, всі його пакети, включаючи дії в грі (рух, використання предметів, взаємодія), проходять через Velocity Proxy. Proxy виступає центральним вузлом, який забезпечує передачу трафіку між гравцем і ігровими серверами. На цьому етапі Proxy не виконує обробку даних, але фіксує всі вхідні запити та передає їх для подальшого аналізу.
2. Генерація подій у Velocity Proxy. Коли пакети досягають Velocity Proxy, вони автоматично перетворюються на події (івенти) за допомогою його API.
3. Слухання подій плагіном. Спеціально розроблений плагін, інтегрований із Velocity API, підписується на обробку певних типів подій пов'язаних з трафіком.
4. Валідація подій. Перш ніж події будуть оброблені далі, плагін виконує їхню валідацію. Це включає перевірку на відповідність структурі даних, відсіювання нерелевантних або дублікатних подій, а також виявлення потенційно некоректних даних. Наприклад, якщо подія не містить UUID гравця або виникла внаслідок аномальної активності, вона може бути відхилена.
5. Обробка подій у плагіні. Після валідації плагін виконує попередню обробку подій, додаючи метадані (наприклад, часові мітки, контекст гри) або агрегуючи дані, щоб зменшити обсяг переданих даних. Цей етап дозволяє створити компактну та оптимізовану структуру подій для передачі.
6. Відправка подій до аналітичної системи. Оброблені та валідовані події передаються в аналітичну систему через REST API Data Collector. Для

кожної передачі додається токен автентифікації, який підтверджує джерело даних.

7. Фінальний етап — надходження даних в аналітику. REST API Data Collector приймає оброблені події, зберігає їх у чергах Redpanda для подальшої потокової обробки.

Цей процес забезпечує високий рівень безпеки, ефективності та масштабованості, оскільки система збору даних працює незалежно від внутрішньої інфраструктури серверів Minecraft і може адаптуватися до збільшення обсягів даних.

3.3 Початкова обробка зібраних даних

Після того, як дані були зібрані плагіном із серверів Minecraft, вони спрямовуються мікросервісу Data Collector, який відповідає за їх початкову обробку. Data Collector побудований на основі Quarkus і використовує бібліотеки для інтеграції з потоковою системою Redpanda. Ця архітектура забезпечує ефективність обробки великих обсягів даних у реальному часі.

Quarkus — це сучасний фреймворк для розробки мікросервісів на мові Java, орієнтований на хмарні середовища та сервери, які використовують контейнеризацію. Quarkus забезпечує швидкий старт застосунків, мале споживання пам'яті та високу продуктивність, що робить його ідеальним для побудови масштабованих мікросервісів [13]. Його обрано через сумісність із широким спектром бібліотек для роботи з поточковими даними та оптимізацію для використання в середовищах, таких як Kubernetes.

Redpanda — це високопродуктивна платформа для потокової передачі (стрімінгу) даних, сумісна з Apache Kafka, але спроектована для кращої продуктивності та зниження затримок [14]. Redpanda дозволяє обробляти потоки даних із високою пропускнуою здатністю, забезпечуючи стабільну роботу навіть за умов великих навантажень. Її було обрано через простоту в налаштуванні,

низькі вимоги до ресурсів і сумісність із сучасними системами обробки потоків, див. рис. 3.3.

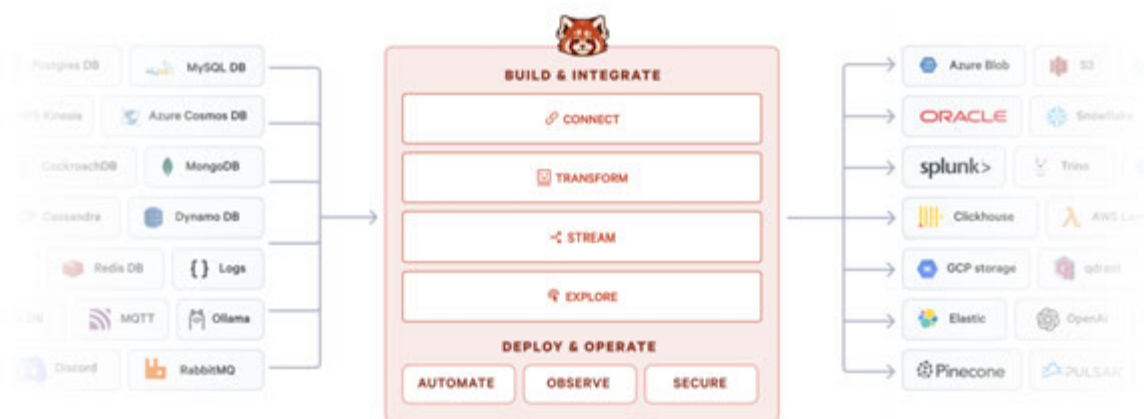


Рис. 3.3 Інтеграційні можливості Redpanda

Apache Kafka — це розподілена платформа для потокової обробки даних, яка дозволяє публікувати, підписуватися, зберігати та обробляти дані в реальному часі. Kafka використовується для побудови високонавантажених систем, де важливо забезпечити швидку і надійну передачу даних між компонентами [15]. Її головними перевагами є підтримка горизонтального масштабування та стійкість до збоїв завдяки реплікації даних.

Потоковий стрімінг (streaming) — це метод обробки даних у режимі реального часу. У цьому підході дані обробляються безпосередньо в момент їх надходження, що дозволяє системам реагувати на події миттєво. Потоковий стрімінг ідеально підходить для аналітичних систем, які обробляють великі обсяги подій, наприклад, івенти гравців, статистику продуктивності серверів або дії в грі.

Після надходження даних до Data Collector виконується перевірка ролі серверу, який надсилає ці дані, за допомогою механізму RBAC (Role-Based Access Control). RBAC — це модель управління доступом, яка дозволяє визначати права доступу на основі ролей [16]. У цьому випадку сервери з відповідними ролями (наприклад, "ігровий сервер", "тестовий сервер") отримують авторизацію для передачі даних. Якщо сервер не авторизовано, дані відхиляються для забезпечення безпеки.

У разі успішної авторизації до даних додається метадата, згенерована колектором. Ця метадата може включати часові мітки, географічні координати, джерело даних та інші контекстуальні параметри. Після цього дані перетворюються у формат JSON перед відправкою в Redpanda.

JSON (JavaScript Object Notation) — це легкий формат обміну даними, який є простим для читання і запису людьми, а також легко оброблюється машинами [17]. JSON забезпечує структуроване представлення даних, що дозволяє легко інтегруватися з іншими компонентами системи, зокрема з потоковими і аналітичними модулями.

Цей процес початкової обробки гарантує, що дані, які передаються до Redpanda, валідовані, захищені та підготовлені до подальшої аналітики у високопродуктивній потоковій системі. Завдяки використанню таких інструментів, як Quarkus та Redpanda, система забезпечує надійні збереження без втрат і обробку великих обсягів даних у режимі реального часу.

3.4 Робота з даними. Обробка та зберігання

Для обробки та зберігання даних у системі використовується мікросервіс Data Service, побудований на основі фреймворку Quarkus. Основна роль Data Service полягає в отриманні даних із Redpanda топиків, виконанні їхньої обробки та подальшому записі у бази даних. Data Service інтегрується з PostgreSQL для реляційних даних і ClickHouse для аналітичних даних, використовуючи різні інструменти ORM, драйвери та засоби керування міграціями бази даних, такі як Flyway.

PostgreSQL — це потужна реляційна база даних з відкритим кодом, яка підтримує розширені функції, такі як складні запити, транзакції, робота з JSON-даними, а також розширення для масштабованості. PostgreSQL є ідеальним вибором для зберігання структурованих даних, наприклад, метаданих і конфігурацій. У системі PostgreSQL використовується для управління

реляційною інформацією, яка часто оновлюється та є критично важливою для роботи аналітики [18].

ClickHouse — це аналітична база даних із колонковою організацією зберігання, яка оптимізована для швидкого виконання запитів до великих обсягів даних (рис. 3.4). Її головними перевагами є висока продуктивність, низькі затримки під час виконання складних аналітичних запитів та можливість обробки даних у реальному часі [19]. ClickHouse використовується в системі для зберігання агрегованих аналітичних даних, таких як статистика користувальницької активності чи метрики серверів.



Рис. 3.4 Інтеграції можливості ClickHouse

Hibernate — це фреймворк для об'єктно-реляційного відображення (ORM), який використовується для спрощення роботи з реляційними базами даних у Java-додатках. Він дозволяє розробникам працювати з базою даних через об'єкти Java, автоматично виконуючи трансляцію об'єктів у SQL-запити [20]. У системі Hibernate використовується для взаємодії з PostgreSQL, забезпечуючи швидко та зручну інтеграцію Data Service із реляційними даними.

JDBC (Java Database Connectivity) — це стандартний API для роботи з базами даних у Java-додатках. Він дозволяє виконувати SQL-запити, отримувати результати та керувати транзакціями [21]. У Data Service JDBC використовується

для інтеграції з ClickHouse, де необхідний прямий доступ до SQL-запитів без використання складних абстракцій.

HikariCP — це високопродуктивний пул з'єднань із базами даних, який забезпечує ефективне управління з'єднаннями між додатком і базою. HikariCP відомий своєю мінімальною затримкою та низькими витратами на ресурси, що робить його ідеальним для роботи у високонавантажених системах [22]. У Data Service HikariCP використовується разом із JDBC для забезпечення стабільної інтеграції з ClickHouse.

Flyway — це інструмент для управління версіями баз даних, який забезпечує автоматизацію процесу міграцій. Flyway використовує SQL-скрипти для впровадження змін у структуру бази даних, таких як створення нових таблиць, зміна колонок або індексів [23]. У системі Flyway інтегрується з PostgreSQL для автоматичного виконання міграцій під час розгортання або оновлення Data Service. Це гарантує, що структура бази даних відповідає поточній версії додатку, запобігаючи можливим проблемам несумісності.

Data Service отримує дані з Redpanda топиків, які обробляються у кілька етапів: перевірка валідності, агрегація та додавання метаданих. Потім ці дані записуються:

- У PostgreSQL для зберігання структурованих реляційних даних таких, як інформації про сесії чи налаштування.
- У ClickHouse для збереження великих обсягів аналітичних даних, які використовуються для побудови візуалізацій і виконання складних аналітичних запитів.

Цей підхід забезпечує гнучкість системи та дозволяє ефективно працювати з великими обсягами даних у реальному часі. Завдяки використанню сучасних інструментів, таких як Hibernate, JDBC і HikariCP, система поєднує високу продуктивність із зручністю розробки та підтримки.

3.5 Візуалізація аналітичних даних

Система візуалізації аналітичних даних побудована для забезпечення зручного та інтерактивного інтерфейсу для користувачів, які працюють з аналітичною інформацією у режимі реального часу. Основу інтерфейсу становить Vue.js, а для побудови графіків і візуальних компонентів використано Highcharts. Дані для візуалізації беруться безпосередньо з Data Service через запити, що забезпечує їх актуальність і відповідність запитам користувачів.

Vue.js — це прогресивний фреймворк JavaScript, призначений для побудови інтерфейсів користувача. Vue.js вирізняється гнучкістю, легкістю в освоєнні та високою продуктивністю [24]. Завдяки компонентній структурі він дозволяє легко створювати інтерактивні інтерфейси з можливістю повторного використання коду. У даній системі Vue.js забезпечує побудову динамічного інтерфейсу, який адаптується до вимог користувачів і дозволяє виконувати запити до аналітичних даних безпосередньо через Data Service.

Highcharts — це бібліотека JavaScript для побудови інтерактивних графіків і діаграм. Вона підтримує широкий спектр типів візуалізації (див. рис. 3.5), включаючи лінійні графіки, гістограми, діаграми розсіювання, теплові карти тощо [25].

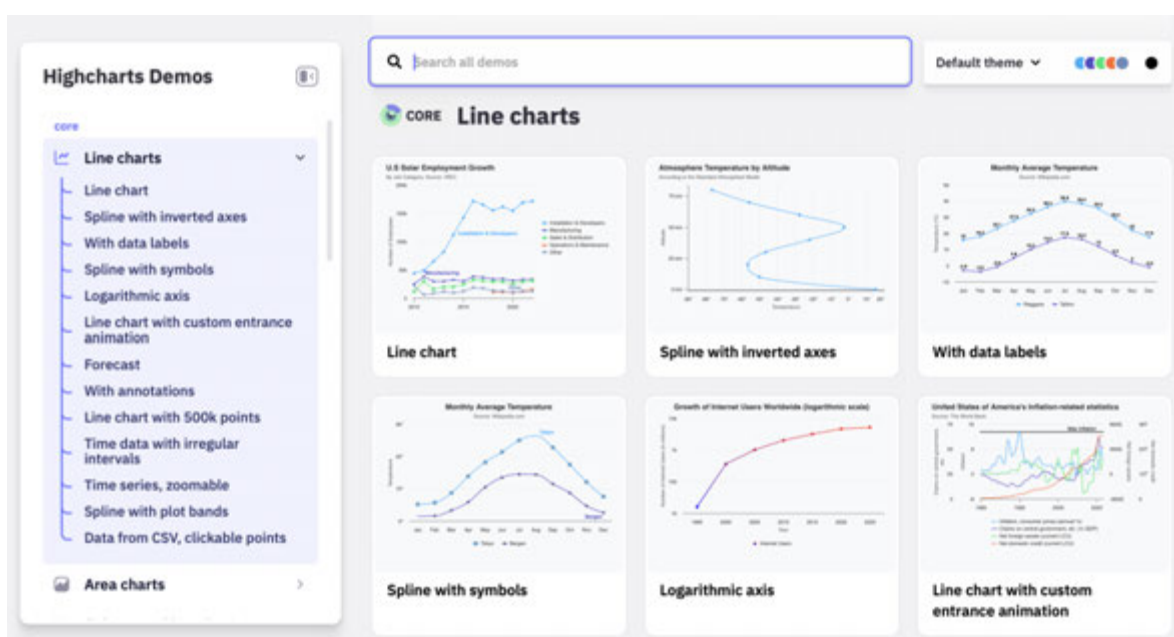


Рис. 3.5 Можливості Highcharts

Highcharts була обрана для системи через її багатофункціональність, налаштовуваність і сумісність із Vue.js. Бібліотека дозволяє легко інтегрувати дані з Data Service у візуальні компоненти, забезпечуючи зручний доступ до ключових метрик.

Візуалізація здійснюється шляхом надсилання запитів із інтерфейсу (Vue.js) до Data Service. На основі цих запитів система:

1. Отримує дані з відповідних баз (PostgreSQL або ClickHouse) через Data Service.
2. Обробляє їх відповідно до вимог користувача (агрегація, фільтрація, сортування).
3. Повертає результати у форматі, придатному для відображення у Highcharts.

Інтерактивний підхід до візуалізації даних дозволяє користувачам (аналітикам, адміністраторам) легко отримувати доступ до потрібної інформації, переглядати метрики в реальному часі, проводити порівняльний аналіз і будувати прогнози. Такий спосіб інтеграції інтерфейсу та аналітики значно спрощує управління проектами і прийняття рішень.

3.6 Інфраструктурні рішення

Інфраструктурна частина системи була побудована з акцентом на надійність, масштабованість і безпеку. Для забезпечення ефективного розгортання компонентів та управління системою використовуються сучасні інструменти, які інтегруються в єдину екосистему.

Kubernetes — це платформа для оркестрації контейнерів, яка автоматизує розгортання, управління і масштабування контейнеризованих додатків. Kubernetes дозволяє забезпечити стабільність і гнучкість роботи навіть за умов високого навантаження. Його використання гарантує, що всі сервіси системи

можуть працювати незалежно один від одного, а також дозволяє легко додавати нові сутності (рис. 3.6) [26].

Кластер було розгорнуто на базі Talos Linux за допомогою Terraform і Ansible. Talos — це мінімалістична операційна система для роботи з Kubernetes, орієнтована на безпеку, автоматизацію та простоту управління. Вона не має традиційного SSH-доступу, що значно знижує ризик атак [27].

Terraform — це інструмент для автоматизації інфраструктури, який дозволяє описувати її у вигляді коду (Infrastructure as Code) [28]. Це робить процес створення, оновлення та масштабування кластера простим і надійним.

Ansible — це інструмент автоматизації, який використовується для управління конфігураціями та налаштувань серверів [29]. У даній системі Ansible забезпечує автоматичне налаштування всіх вузлів кластера.

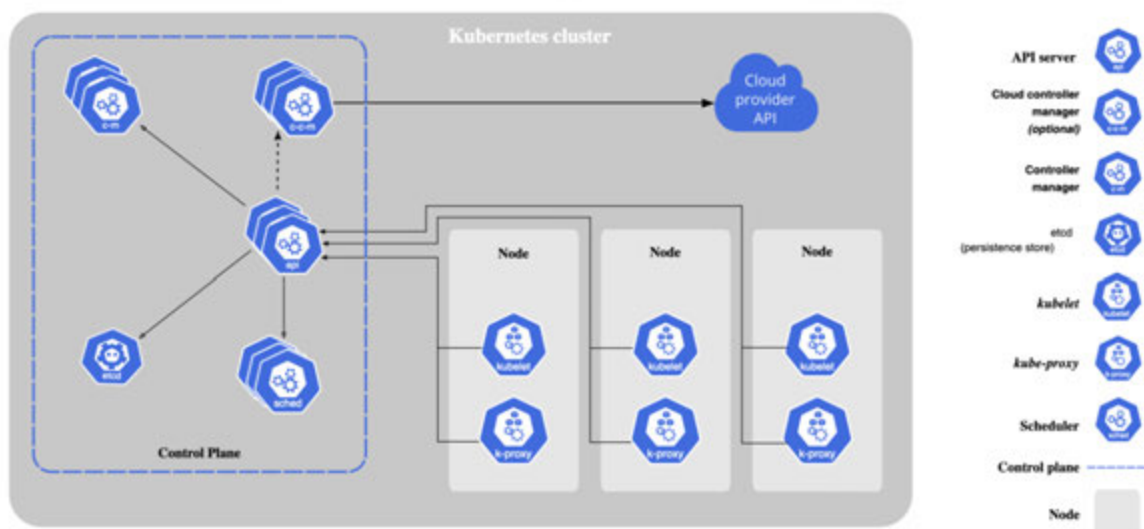


Рис. 3.6 Компоненти кластеру Kubernetes

Уся інфраструктура, пов'язана з обробкою даних, розгорнута на окремих машинах із Ubuntu [30], які мають додатковий захист і налаштовані резервні копії (бекапи). Це дозволяє гарантувати цілісність даних навіть у випадку технічних збоїв.

Для управління процесом розгортання застосовується Helm — інструмент для управління пакетами в Kubernetes [31]. Helm дозволяє спростити та

автоматизувати деплой додатків, забезпечуючи при цьому можливість версіонування конфігурацій.

GitLab CI/CD — це платформа для автоматизації процесу тестування, збірки та деплойменту додатків [32]. CI/CD забезпечує швидке і контрольоване розгортання нових версій сервісів, що знижує ризик помилок і дозволяє швидше інтегрувати зміни в систему.

Для управління доменними іменами використовується Cloudflare DNS, який забезпечує швидке і надійне розв'язання DNS-запитів [33]. Крім того, Cloudflare пропонує додаткові функції безпеки, такі як захист від DDoS-атак і автоматичне шифрування трафіку.

Для роботи з ігровим трафіком за потреби може бути використано HAProxy — високопродуктивний балансувальник навантаження та проксі-сервер. HAProxy дозволяє рівномірно розподіляти ігровий трафік між серверами, забезпечуючи стабільність системи під час пікових навантажень і знижуючи затримки [34].

Завдяки використанню описаних інструментів і технологій вдалося створити інфраструктуру, яка відповідає сучасним стандартам безпеки та надійності. Автоматизація процесів деплойменту та управління дозволяє швидко масштабувати систему та адаптувати її до змін у навантаженні чи конфігурації.

4. РЕЗУЛЬТАТИ ДОСЛІДЖЕННЯ

4.1 Апаратні та програмні вимоги

Для забезпечення стабільної роботи системи аналізу трафіку ігрових серверів Minecraft були визначені мінімальні апаратні та програмні вимоги, що враховують потреби кожного з компонентів системи.

Кластер Kubernetes потребує мінімум трьох воркер нод для забезпечення високої доступності. Це дозволяє рівномірно розподілити навантаження між сервісами та забезпечити їхню стійкість до відмов. Кожна нода має бути обладнана:

- 2 ГБ оперативної пам'яті (RAM);
- 1 віртуальним процесором (vCPU).

Для обробки та зберігання даних потрібні три сервери для баз даних із більш високими вимогами до ресурсів, оскільки вони обробляють великі обсяги інформації та виконують складні запити. Кожен сервер повинен мати:

- 8 ГБ оперативної пам'яті (RAM);
- 2 віртуальні процесори (vCPU).

Для коректної маршрутизації запитів потрібен домен і доступ до налаштування DNS-записів. Це дозволить налаштувати маршрутизацію трафіку між компонентами системи та забезпечити доступ до інтерфейсу аналітичних даних через зручну адресу.

Для автоматизації процесів тестування, збірки та розгортання необхідно налаштувати CI/CD інфраструктуру. Це включає:

- Сервер з GitLab CI/CD або подібним інструментом для реалізації конвеєра CI/CD;
- Мінімум 2 ГБ RAM і 1 vCPU для роботи CI/CD;
- Доступ до репозиторію з вихідним кодом системи.

Особливістю системи є мінімальні вимоги до ресурсу плагіна для збору метрик, встановленого на ігрових серверах. Цей плагін потребує лише:

- 50 мілікорів (0.05 CPU);
- 70 МБ оперативної пам'яті (RAM).

Його низькі вимоги забезпечують легку інтеграцію навіть на сервери з обмеженими ресурсами без шкоди для продуктивності ігрового процесу.

Додатково потрібен існуючий проект серверів Minecraft, який буде джерелом даних для системи. Ці сервери повинні підтримувати інтеграцію з Velocity Proxy і надавати доступ до ігрових подій для збору інформації. Це включає встановлення плагінів, які передають телеметрію до системи.

4.2 Хід дослідження

Дослідження було проведено у кілька етапів, які включали аналіз предметної області, моделювання системи, розробку інфраструктури, реалізацію функціональних модулів, а також тестування і аналіз результатів. Нижче наведено опис ключових етапів ходу дослідження.

На початковому етапі було визначено основні цілі та задачі дослідження. Проведено аналіз існуючих рішень для моніторингу трафіку ігрових серверів, зокрема систем Plan і Network Manager, для виявлення їхніх переваг та недоліків. Це дозволило сформулювати основну проблему: створення системи аналізу трафіку, яка забезпечує реальний час роботи, мінімізує втрати даних і дозволяє проводити глибокий аналіз.

На основі отриманих вимог було створено кілька типів діаграм:

- Діаграма прецедентів (Use Case Diagram), що відобразила сценарії використання системи.
- Діаграма послідовності (Sequence Diagram), яка описала динамічну взаємодію компонентів під час обробки запитів.
- Діаграма активності (Activity Diagram), що відобразила етапи обробки даних.

- Діаграма User Story, яка деталізувала потреби користувачів та їхню взаємодію із системою.

Ці моделі дозволили сформувати чітку архітектуру системи, визначивши ключові компоненти, їхню взаємодію та способи інтеграції.

Для розгортання системи було створено кластер Kubernetes із використанням Talos Linux. Інфраструктура автоматизована за допомогою Terraform та Ansible, що дозволило швидко налаштувати високонадійну архітектуру. Сервери для баз даних і потокової передачі даних було розгорнуто окремо для забезпечення безпеки і стабільності роботи системи.

Розробка системи включала кілька ключових компонентів:

1. Плагін для збору метрик на базі Velocity Proxy і Paper API, який виконує валідацію подій та передає їх до Data Collector через REST API.
2. Data Collector, реалізований на Quarkus, забезпечує інтеграцію з Redpanda для потокової обробки даних.
3. Data Service, який обробляє потоки даних із Redpanda, зберігає їх у PostgreSQL і ClickHouse за допомогою Hibernate та JDBC.

Було проведено кілька типів тестування:

- Функціональне тестування перевіряло правильність роботи основних модулів.
- Навантажувальне тестування забезпечувало стабільність роботи під час пікових навантажень.
- Інтеграційне тестування гарантувало коректну взаємодію між компонентами.

На завершальному етапі було проведено аналіз результатів роботи системи. Система продемонструвала високу ефективність обробки ігрового трафіку, забезпечила точну візуалізацію даних у режимі реального часу та відповідає поставленим вимогам щодо стабільності та масштабованості.

Хід дослідження відобразив послідовну та систематичну роботу над створенням системи аналізу трафіку. Кожен етап був спрямований на вирішення конкретних задач, що забезпечило досягнення всіх цілей дослідження.

4.3 Підсумкові результати дослідження

Підсумком проведеного дослідження стало створення системи аналізу трафіку ігрових серверів Minecraft, яка демонструє високу ефективність у зборі, обробці, зберіганні та візуалізації даних. Система дозволяє отримувати як загальні, так і деталізовані аналітичні дані, які надають глибоке розуміння поведінки користувачів, ефективності маркетингових кампаній і загального стану ігрових серверів. Завдяки інтерактивному інтерфейсу та широким можливостям аналізу, система може використовуватися для оптимізації проєктів, прийняття стратегічних рішень і моніторингу ключових показників.

Перший аналітичний результат представлений у вигляді кругової діаграми на рис. 4.1, яка демонструє розподіл джерел трафіку. Найбільшу частку, 32%, складають відвідувачі, залучені через Monitoring ADS. Це підкреслює значення рекламних оголошень на платформах моніторингу серверів як найбільш ефективного інструменту для залучення нової аудиторії. Значну роль також відіграють Discord (15%) та рекламні пости (15%), що свідчить про важливість соціальних платформ у підтримці активності спільноти.



Рис. 4.1 Кругова діаграма джерел походження трафіку

Система аналізу дозволяє не лише визначати джерела трафіку, а й аналізувати географічну статистику гравців, відображаючи розподіл користувачів по країнах на інтерактивній мапі. Цей інструмент допомагає виявити ключові регіони зростання, що є важливим для таргетування маркетингових кампаній.

Другий аналітичний результат демонструється на лінійному графіку (див. рис. 4.2), який ілюструє динаміку відвідувань нових і конверсію відвідувачів в постійних гравців. Графік чітко показує пікову активність на вихідних, що є типовою поведінкою для більшості ігрових спільнот. Інші піки на графіку співпадають із проведенням маркетингових кампаній, таких як акції, розіграші або запуск реклами. Це свідчить про ефективність маркетингових стратегій і можливість їхнього точного впливу на поведінку користувачів.



Рис. 4.2 Лінійний графік динаміки відвідувань і конверсії в постійних гравців

Крім того, система здатна визначати індивідуальну статистику гравців, включаючи їхній внесок у загальну активність, тривалість сесій і взаємодії в грі. Вона також може виявляти аномальні патерни поведінки, які можуть бути ознаками зловживань, наприклад, використання ботів чи іншого забороненого софту. Це додає рівень захисту для ігрової платформи та покращує якість досвіду користувачів.

Система також передбачає можливість впровадження реферальної системи, яка дозволить відстежувати ефективність реферальних програм і мотивувати користувачів залучати нову аудиторію. Аналіз таких даних може дати додаткові інсайти для маркетингових рішень.

Завдяки цим результатам система аналізу трафіку не лише успішно виконує поставлені задачі, але й надає розширені можливості для стратегічного управління проектами, таргетування аудиторії та моніторингу поведінки гравців. Вона забезпечує як загальний огляд стану серверів, так і можливість детального аналізу, що робить її універсальним інструментом для розробників, маркетологів і адміністраторів.

ВИСНОВКИ

У межах виконання магістерської роботи було розроблено систему аналізу трафіку для моніторингу активності користувачів ігрових серверів Minecraft, яка відповідає сформульованій меті дослідження. У роботі охоплено всі етапи створення системи, від розробки архітектури до реалізації її компонентів, проведення тестування та аналізу отриманих результатів. Основні результати та висновки дослідження наведено нижче.

У роботі реалізовано комплексну систему аналізу трафіку, яка включає такі компоненти:

- Модуль збору даних: інтеграція з Velocity Proxy і Paper API для отримання подій від серверів Minecraft. Розроблено плагін, який забезпечує валідацію, попередню обробку даних і передачу подій до аналітичної системи.
- Модуль обробки даних: впроваджено мікросервіс Data Collector, який використовує Quarkus і Redpanda для потокової обробки даних у режимі реального часу.
- Система зберігання даних: реалізовано збереження реляційних даних у PostgreSQL і аналітичних даних у ClickHouse із використанням Hibernate та JDBC.
- Інструменти візуалізації: створено інтерактивний інтерфейс на базі Vue.js і Highcharts для відображення ключових метрик та аналітики в реальному часі.
- Інфраструктура: розроблено автоматизовану систему розгортання на основі Kubernetes, Terraform і Ansible, яка забезпечує масштабованість та високу доступність.

Огляд висновків за результатами дослідження:

- Ефективність системи: Використання Redpanda забезпечило низькі затримки у передачі та обробці подій, що є критично важливим для аналізу

ігрового трафіку в реальному часі. Тестування підтвердило стабільність роботи системи під високим навантаженням.

- Динаміка активності гравців: Графічний аналіз показав пікову активність гравців на вихідних, що відповідає загальним тенденціям поведінки аудиторії багатокористувацьких ігор. Додаткові сплески активності нових користувачів підтверджують ефективність проведених маркетингових кампаній.
- Розподіл джерел трафіку: Найбільшу частку трафіку становлять Monitoring ADS (32%), Discord (15%) та рекламні пости (15%), що підтверджує їхню ключову роль у залученні аудиторії.
- Інтеграція інструментів: Використання сучасних технологій забезпечило надійність, гнучкість і масштабованість системи, що дозволяє адаптувати її до змінних умов роботи.
- Аналітичні можливості: Система дозволяє аналізувати географічний розподіл гравців, визначати індивідуальні патерни поведінки та виявляти аномалії, що можуть свідчити про зловживання.

Рекомендації щодо можливого подальшого використання отриманих результатів

- Адаптація для інших проектів: Розроблену систему можна адаптувати для використання в інших багатокористувацьких іграх або онлайн-платформах, що дозволить ефективно моніторити активність користувачів.
- Інтеграція реферальної системи: Впровадження реферальних програм на основі отриманих даних сприятиме залученню нових гравців та підвищенню лояльності поточної аудиторії.
- Розширення аналітичних можливостей: Подальший розвиток системи може включати прогнозування поведінки гравців, побудову рекомендаційних систем та автоматизацію виявлення аномалій.

- Оптимізація маркетингових стратегій: Отримані дані дозволяють більш ефективно планувати та аналізувати маркетингові кампанії, зокрема з урахуванням географічного розподілу аудиторії.

Таким чином, результати дослідження демонструють, що розроблена система здатна виконувати всі поставлені задачі, забезпечуючи високу продуктивність, точність і гнучкість у роботі з аналітичними даними. Отримані висновки мають значний науковий та практичний потенціал, створюючи базу для подальших досліджень у сфері аналізу даних і вдосконалення багатокористувацьких платформ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Plan [Електронний ресурс]. – Режим доступу: <https://www.playeranalytics.net> – Дата звернення: 27.10.2024.
2. Network Manager [Електронний ресурс]. – Режим доступу: <https://www.spigotmc.org/resources/networkmanager.28456/> – Дата звернення: 27.10.2024.
3. Data Flow Diagram [Електронний ресурс]. – Режим доступу: <https://www.lucidchart.com/pages/data-flow-diagram> – Дата звернення: 27.10.2024.
4. Use Case Diagram [Електронний ресурс]. – Режим доступу: <https://creately.com/diagram/example/ikv4e94o5/use-case-diagram> – Дата звернення: 27.10.2024.
5. Sequence Diagram [Електронний ресурс]. – Режим доступу: <https://plantuml.com/sequence-diagram> – Дата звернення: 27.10.2024.
6. Activity Diagram [Електронний ресурс]. – Режим доступу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/> – Дата звернення: 27.10.2024.
7. User Story Diagram [Електронний ресурс]. – Режим доступу: <https://www.mountangoatsoftware.com/agile/user-stories> – Дата звернення: 27.10.2024.
8. Velocity [Електронний ресурс]. – Режим доступу: <https://velocitypowered.com/> – Дата звернення: 27.10.2024.
9. Paper [Електронний ресурс]. – Режим доступу: <https://papermc.io/> – Дата звернення: 27.10.2024.
10. Spigot [Електронний ресурс]. – Режим доступу: <https://www.spigotmc.org/> – Дата звернення: 27.10.2024.
11. Bukkit [Електронний ресурс]. – Режим доступу: <https://bukkit.org/> – Дата звернення: 27.10.2024.

12. REST API [Электронный ресурс]. – Режим доступа: <https://restfulapi.net/> – Дата звернення: 27.10.2024.
13. Quarkus [Электронный ресурс]. – Режим доступа: <https://quarkus.io/> – Дата звернення: 27.10.2024.
14. Redpanda [Электронный ресурс]. – Режим доступа: <https://redpanda.com/> – Дата звернення: 27.10.2024.
15. Kafka [Электронный ресурс]. – Режим доступа: <https://kafka.apache.org/> – Дата звернення: 27.10.2024.
16. RBAC [Электронный ресурс]. – Режим доступа: <https://auth0.com/docs/authorization/rbac/> – Дата звернення: 27.10.2024.
17. JSON [Электронный ресурс]. – Режим доступа: <https://www.json.org/> – Дата звернення: 27.10.2024.
18. PostgreSQL [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/> – Дата звернення: 27.10.2024.
19. ClickHouse [Электронный ресурс]. – Режим доступа: <https://clickhouse.com/> – Дата звернення: 27.10.2024.
20. Hibernate [Электронный ресурс]. – Режим доступа: <https://hibernate.org/> – Дата звернення: 27.10.2024.
21. JDBC [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> – Дата звернення: 27.10.2024.
22. HikariCP [Электронный ресурс]. – Режим доступа: <https://github.com/brettwooldridge/HikariCP> – Дата звернення: 27.10.2024.
23. Flyway [Электронный ресурс]. – Режим доступа: <https://flywaydb.org/> – Дата звернення: 27.10.2024.
24. Vue.js [Электронный ресурс]. – Режим доступа: <https://vuejs.org/> – Дата звернення: 27.10.2024.
25. Highcharts [Электронный ресурс]. – Режим доступа: <https://www.highcharts.com/> – Дата звернення: 27.10.2024.

26. Kubernetes [Електронний ресурс]. – Режим доступу: <https://kubernetes.io/> – Дата звернення: 27.10.2024.
27. Talos [Електронний ресурс]. – Режим доступу: <https://www.talos.dev/> – Дата звернення: 27.10.2024.
28. Terraform [Електронний ресурс]. – Режим доступу: <https://www.terraform.io/> – Дата звернення: 27.10.2024.
29. Ansible [Електронний ресурс]. – Режим доступу: <https://www.ansible.com/> – Дата звернення: 27.10.2024.
30. Ubuntu [Електронний ресурс]. – Режим доступу: <https://ubuntu.com/> – Дата звернення: 27.10.2024.
31. Helm [Електронний ресурс]. – Режим доступу: <https://helm.sh/> – Дата звернення: 27.10.2024.
32. GitLab CI/CD [Електронний ресурс]. – Режим доступу: <https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/> – Дата звернення: 27.10.2024.
33. Cloudflare DNS [Електронний ресурс]. – Режим доступу: <https://www.cloudflare.com/dns/> – Дата звернення: 27.10.2024.
34. HAProxy [Електронний ресурс]. – Режим доступу: <https://www.haproxy.org/> – Дата звернення: 27.10.2024.

Вы сказали:

Можеш ще назви підшаманити? Примірно щоб трохи було згадки як тайтл на сайті

ChatGPT сказав:

Список використаних джерел

1. Plan - Advanced Analytics for Minecraft Servers [Електронний ресурс]. – Режим доступу: <https://www.playeranalytics.net> – Дата звернення: 27.10.2024.
2. Network Manager - Efficient Management for Minecraft Networks [Електронний ресурс]. – Режим

- доступу: <https://www.spigotmc.org/resources/networkmanager.28456/> – Дата звернення: 27.10.2024.
3. Data Flow Diagram - Visualize Data Processes [Електронний ресурс]. – Режим доступу: <https://www.lucidchart.com/pages/data-flow-diagram> – Дата звернення: 27.10.2024.
 4. Use Case Diagram - Simplified System Scenarios [Електронний ресурс]. – Режим доступу: <https://creately.com/diagram/example/ikv4e94o5/use-case-diagram> – Дата звернення: 27.10.2024.
 5. Sequence Diagram - Understand System Interactions [Електронний ресурс]. – Режим доступу: <https://plantuml.com/sequence-diagram> – Дата звернення: 27.10.2024.
 6. Activity Diagram - Workflow and Processes Made Clear [Електронний ресурс]. – Режим доступу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/> – Дата звернення: 27.10.2024.
 7. User Story Diagram - Visualizing Agile Stories [Електронний ресурс]. – Режим доступу: <https://www.mountangoatsoftware.com/agile/user-stories> – Дата звернення: 27.10.2024.
 8. Velocity - High-Performance Minecraft Проху [Електронний ресурс]. – Режим доступу: <https://velocitypowered.com/> – Дата звернення: 27.10.2024.
 9. Paper - Optimized Minecraft Server Software [Електронний ресурс]. – Режим доступу: <https://papermc.io/> – Дата звернення: 27.10.2024.
 10. Spigot - Open Source Minecraft Server [Електронний ресурс]. – Режим доступу: <https://www.spigotmc.org/> – Дата звернення: 27.10.2024.
 11. Bukkit - Minecraft Plugin Platform [Електронний ресурс]. – Режим доступу: <https://bukkit.org/> – Дата звернення: 27.10.2024.
 12. REST API - Design Patterns for Web Services [Електронний ресурс]. – Режим доступу: <https://restfulapi.net/> – Дата звернення: 27.10.2024.
 13. Quarkus - Kubernetes Native Java Framework [Електронний ресурс]. – Режим доступу: <https://quarkus.io/> – Дата звернення: 27.10.2024.

- 14.Redpanda - High-Performance Streaming Data Platform [Электронный ресурс]. – Режим доступа: <https://redpanda.com/> – Дата звернення: 27.10.2024.
- 15.Apache Kafka - Event Streaming Platform [Электронный ресурс]. – Режим доступа: <https://kafka.apache.org/>– Дата звернення: 27.10.2024.
- 16.RBAC - Role-Based Access Control Explained [Электронный ресурс]. – Режим доступа: <https://auth0.com/docs/authorization/rbac/> – Дата звернення: 27.10.2024.
- 17.JSON - Lightweight Data Format [Электронный ресурс]. – Режим доступа: <https://www.json.org/> – Дата звернення: 27.10.2024.
- 18.PostgreSQL - Advanced Open Source RDBMS [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/> – Дата звернення: 27.10.2024.
- 19.ClickHouse - Fast Analytics Database [Электронный ресурс]. – Режим доступа: <https://clickhouse.com/> – Дата звернення: 27.10.2024.
- 20.Hibernate - Object-Relational Mapping Tool [Электронный ресурс]. – Режим доступа: <https://hibernate.org/> – Дата звернення: 27.10.2024.
- 21.JDBC - Java Database Connectivity [Электронный ресурс]. – Режим доступа: <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> – Дата звернення: 27.10.2024.
- 22.HikariCP - High-Performance Connection Pool [Электронный ресурс]. – Режим доступа: <https://github.com/brettwooldridge/HikariCP> – Дата звернення: 27.10.2024.
- 23.Flyway - Version Control for Databases [Электронный ресурс]. – Режим доступа: <https://flywaydb.org/> – Дата звернення: 27.10.2024.
- 24.Vue.js - Progressive JavaScript Framework [Электронный ресурс]. – Режим доступа: <https://vuejs.org/> – Дата звернення: 27.10.2024.
- 25.Highcharts - Interactive JavaScript Charts [Электронный ресурс]. – Режим доступа: <https://www.highcharts.com/> – Дата звернення: 27.10.2024.

26. Kubernetes - Automating Deployment, Scaling, and Management [Электронный ресурс]. – Режим доступа: <https://kubernetes.io/> – Дата звернення: 27.10.2024.
27. Talos - Secure Kubernetes Operating System [Электронный ресурс]. – Режим доступа: <https://www.talos.dev/> – Дата звернення: 27.10.2024.
28. Terraform - Infrastructure as Code Tool [Электронный ресурс]. – Режим доступа: <https://www.terraform.io/> – Дата звернення: 27.10.2024.
29. Ansible - Simplified IT Automation [Электронный ресурс]. – Режим доступа: <https://www.ansible.com/> – Дата звернення: 27.10.2024.
30. Ubuntu - Linux Distribution for Servers [Электронный ресурс]. – Режим доступа: <https://ubuntu.com/> – Дата звернення: 27.10.2024.
31. Helm - Kubernetes Package Manager [Электронный ресурс]. – Режим доступа: <https://helm.sh/> – Дата звернення: 27.10.2024.
32. GitLab CI/CD - Automating Development Pipelines [Электронный ресурс]. – Режим доступа: <https://about.gitlab.com/stages-devops-lifecycle/continuous-integration/> – Дата звернення: 27.10.2024.
33. Cloudflare DNS - Managed Domain Name System [Электронный ресурс]. – Режим доступа: <https://www.cloudflare.com/dns/> – Дата звернення: 27.10.2024.
34. HAProxy - Reliable Load Balancing and Proxying [Электронный ресурс]. – Режим доступа: <https://www.haproxy.org/> – Дата звернення: 27.10.2024.

ДОДАТКИ