НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
(повна назва інституту/факультету)
Кафедра автоматики та управління в технічних системах
(повна назва кафедри)

«До захисту допущено»
завідувач кафедри АУТС
_____ О.І.Ролік
«___»_____20__ р.

Магістерська дисертація

зі спеціальності (спеціалізації) 126 «Інформаційні системи та технології»
(код і назва спеціальності)

на тему: Розподілена система аналізу якості сервісів хмарної IT-інфраструктури

Виконав: Студент 6 курсу, групи ІА-82мп
(шифр групи)

Ярощук Тарас Володимирович _____
(прізвище, ім'я, по батькові)

Науковий керівник:
зав. каф. АУТС, д.т.н., проф., Ролік О.І._____ _____
(посада, нуковий ступінь, вчене звання, прізвище та ініціали)

Рецензент: _____ _____
(посада, нуковий ступінь, вчене звання, прізвище та ініціали)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ - 2019 рік

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет (інститут) Інформатики та обчислювальної техніки
(повна назва)

Кафедра Автоматики та управління в технічних системах
(повна назва)

Освітньо-кваліфікаційний ступінь другий (магістерський) за освітньо-професійною програмою

Спеціальність 126 Інформаційні системи та технології
(код і назва)

Спеціалізація Інтегровані інформаційні системи
(назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.І. Ролік
(підпис) (ініціали, прізвище)
«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Ярощуку Тарасу Володимировичу
(прізвище, ім'я, по батькові)

1. Тема дисертації Розподілена система аналізу якості сервісів хмарної IT-інфраструктури
Науковий керівник Ролік Олександр Іванович, зав. каф. АУТС, д.т.н., проф.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «____» _____ 2019 р. № _____

2. Строк подання студентом дисертації 3 грудня 2019 р.

3. Об'єкт дослідження система управління якістю корпоративної IT-інфраструктури

4. Зміст пояснювальної записки а) аналіз існуючих рішень; б) розробка аналітичної моделі прогнозування; в) розробка системи управління якістю; г) розробка алгоритму прогнозування; д) розробка системи аналізу якості; е) розробка стартап-проекту.

5. Перелік завдань, які потрібно розробити: проаналізувати поняття якості сервісу, проаналізувати існуючі рішення систем аналізу якості сервісів, розробити модель

прогнозування якості, розробити структурну схему системи, протестувати розроблену систему, провести маркетинговий аналіз стартап проекту.

6. Перелік графічного (ілюстративного) матеріалу: структурна схема системи аналізу якості сервісів, алгоритм прогнозування партості IT-інфраструктури, діаграма прецедентів, діаграма послідовності створення нового сервісу, діаграма класів системи аналізу, діаграма розгортання системи, схема бази даних, алгоритм створення нового сервісу.

7. Орієнтовний перелік публікацій:

8. Консультанти розділів дисертації:

9. Дата видачі завдання 05 вересня 2019

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Строк виконання етапів дисертації	Примітка
1	Аналіз предметної області, огляд літератури	15.09.2019	
2	Дослідження методів аналізу якості сервісів	18.09.2019	
3	Розробка моделі машинного навчання для прогнозування якості	30.09.2019	
4	Технічне рішення та реалізація системи аналізу якості	20.10.2019	
5	Тестування роботи системи	10.11.2019	
6	Розробка стартап-проекту	20.11.2019	
7	Оформлення матеріалів дисертації	04.12.2019	

Студент _____ Т. В. Ярощук
(підпис) (ініціали, прізвище)

Науковий керівник дисертації _____ О.І. Ролік
(підпис) (ініціали, прізвище)

РЕФЕРАТ

Магістерська дисертація 108 с., 30 рис., 20 табл., 9 дод, 22 дж.

Тема магістерської дисертації “Розподілена система аналізу якості сервісів хмарної IT інфраструктури”.

Актуальність магістерської дисертації полягає в тому, що кожна компанія що надає послуги онлайн має свою IT-інфраструктуру із запущеними в ній сервісами, якість яких постійно повинна в автоматичному режимі підтримуватись на узгодженому рівні.

Об’єктом дослідження є система управління якістю IT-інфраструктури.

Предметом дослідження є показники якості сервісів корпоративної IT-інфраструктури.

Метою дисертації є створення розподіленої системи аналізу якості сервісів корпоративної IT-інфраструктури, яка в автоматичному режимі управляє ресурсами, що надані сервісам, створює звіти прогнозування вартості інфраструктури та надає користувачам можливість виконувати аналітичні запити до даних. Задачею роботи є:

1. дослідити предметну область;

2. дослідження методів аналізу числових рядів;

3. розробка аналітичної моделі прогнозування вартості IT-інфраструктури;

4. розробити систему аналізу якості сервісів з використанням технологій обробки великих даних;

5. дослідити ефективність розробленого методу у порівнянні з існуючими.

Ключові слова: машинне навчання, великі дані, система аналізу, якість сервісів, корпоративна IT-інфраструктура.

ABSTRACT

Master's Thesis 108 p., 30 figures, 20 tables, 9 appendixes, 22 sources.

Theme of the master's thesis "Distributed quality analysis system for services in a cloud IT-infrastructure".

The reason for the relevance of a master's thesis is that each company providing online services has its own IT infrastructure with services running in it, the quality of which must be maintained automatically at a consistent level.

The object of the study is the quality management system of the IT infrastructure.

The subject of the study is the quality of corporate IT infrastructure services.

The purpose of this work is to create a distributed system of quality analysis of corporate IT infrastructure services, which automatically manages the resources provided to the services, creates infrastructure cost forecasting reports and enables users to perform analytical data queries. The task of the work is:

1. explore the subject area;

2. research of methods of analysis of numerical series;

3. development of an analytical model for forecasting the cost of IT infrastructure;

4. Develop a system for analyzing the quality of services using Big Data processing technologies;

5. investigate the effectiveness of the developed method in comparison with existing ones.

Keywords: machine learning, big data, analysis system, quality of services, corporate IT-infrastructure.

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ У РОБОТІ СКОРОЧЕНЬ

БД – база даних;

СУБД - система керування базами даних;

ЦОД – центр обробки даних;

API - Прикладний програмний інтерфейс;

DWH - Data Warehouse;

JS - JavaScript

JSON - JavaScript Object Notation;

HTTP - HyperText Transfer Protocol;

HDFS - Hadoop Ditributed File System

ВСТУП

Кожна компанія, що надає послуги онлайн, зацікавлена в підтримці якості цих сервісів, оскільки в умовах конкуренції клієнти мають вибір використовувати ті сервіси, які працюють швидше, краще та без перебоїв у роботі. При роботі з корпоративними клієнтами сторонами складається угода про рівень надання послуг, при порушенні умов якої компанія, що надає послугу, зобов'язана відшкодувати завдані збитки.

З ростом компанії її корпоративна IT-інфраструктура стає все більш складною в підтримці, зростає кількість системних адміністраторів та співробітників служби підтримки, тому використання хмарних обчислень стає все більш популярною альтернативою власному ЦОД. Безумовно, захист персональних даних користувачів та захищеність даних компаній є однією з найголовніших цілей для успішного ведення бізнесу, проте більшість з лідерів провайдерів хмарних послуг гарантує це на документальному рівні. Також, існує можливість створити власне корпоративне хмарне середовище, доступ до якого буде можливий лише для співробітників компанії.

Наявність бізнес-аналітиків у штаті компанії, неодмінно, збільшує її конкурентоспроможність, оскільки для компанії відкривається можливість приймати рішення на основі аналізу історичних даних та прогнозуванні. Класичним підходом для виконання аналітичних звітів є використання тих сами баз даних, які містять і інформацію про транзакції та користувачів компанії. При зростанні кількості користувачів та співробітників, традиційні реляційні бази даних не витримують навантаження і стають все менш ефективними, тому використання додаткового аналітичного сховища даних є хорошою альтернативою повільним записам до СУБД.

З ростом популярності підходу розбивання великого корпоративного сервісу на більш дрібні мікро-сервіси зростає також і складність у конфігурування та моніторингу кожного з них. Для розгортання монолітного застосунко небхідно створити та підтримувати всього один скрипт запуску, проте для кожного мікро-

сервісу необхідно написати свою програму для створення образу, завантаження на сервер та старта самого сервіса, що значно ускладнює підтримку системи в цілому.

Метою дисертації є створення розподіленої системи аналізу якості сервісів корпоративної IT-інфраструктури, яка в автоматичному режимі управляє ресурсами, що надані сервісам, створює звіти прогнозування вартості інфраструктури та надає користувачам можливість виконувати аналітичні запити до даних.

Дана система повинна просто і зручно інтегруватись з усіми хмарними провайдерами та з будь-якою власною IT-інфраструктурою компаній.

Для досягнення поставленої мети потрібно:

– розглянути переваги та недоліки розгортання та підтримки сервісів в контейнерах, у віртуальному середовищі та на окремих серверах компанії;

– дати визначення поняттю “якості сервісу”, охарактеризувати показники якості сервісу;

– дати огляд існуючих рішень в області аналізу показників якості сервісів хмарної IT-інфраструктури;

– розробити систему управління функціональністю IT-інфраструктури;

– розробити підсистему збору показників якості сервісів;

– створити користувацький веб-застосунок для управління та моніторингу сервісів в корпоративній IT інфраструктурі.

1 АНАЛІЗ ЯКОСТІ СЕРВІСІВ ЗА ДОПОМОГОЮ КОРПОРАТИВНОГО СХОВИЩА ДАНИХ

1.1 Задача підтримки якості сервісів на узгодженому рівні

В умовах все зростаючої конкуренції, робота над якістю послуг є невід'ємною частиною сервісного бізнесу. Клієнти вимагають від компаній-постачальників чітких гарантій і домовленостей, в разі порушення яких клієнту будуть компенсовані завдані збитки. Такою гарантією в сучасному світі стала угода про рівень надання послуг (він англ. - Service Level Agreement). Така угода є зовнішнім документом (існуючий між замовником і виконавцем), що описує параметри послуги, що надається. "Відповідність SLA" еквівалентно тому, що сервіс працює так, що реальні параметри відповідають узгодженим в угоді значенням метрик. Хоча сам термін SLA з'явився в IT, сьогодні такі документи використовуються для опису найрізноманітніших послуг, як в сфері інформаційних технологій, так і в інших сегментах B2B, наприклад, в обслуговуванні комерційної нерухомості, при ремонті спеціалізованого обладнання, тощо.

Угоди SLA активно застосовуються там, де виконавець і замовник послуг автономні по відношенню один до одного. Це можуть бути відносини між компаніями або навіть підрозділами одного бізнесу, якщо одне з них позиціонується, як внутрішній клієнт. У таких випадках, правда, передбачена інша абревіатура з трьох букв: OLA - Operational Level Agreement - аналогічний SLA внутрішній документ компанії, що визначає зони відповідальності підрозділів при взаємодії в процесі формування якоїсь зовнішньої послуги.

Договір SLA не бюрократизує роботу IT чи іншого сервісного підрозділу, а навпаки, формалізує та робить більш прозорим взаємодія зі споживачами послуг. SLA не тільки містить опис послуг, а й ставить межі відповідальності в рамках певного сервісу. Наприклад, якщо одна сторона домовилася на обслуговування типової IT-інфраструктури, але вже після укладення договору в ній з'являється специфічне обладнання, а разом з ним - і заявки на його налаштування. Відповідний фахівець коштує дорого, а його зарплату не була закладена при розрахунку вартості даної послуги. З іншого боку, клієнт вважає, що це частина інфраструктури, а

значить ваша відповідальність. Або компанія обслуговуєте онлайн касу, а у клієнта проблеми з 1С, які він намагається перенаправити на компанію. SLA дозволяє формалізувати це питання, обмеживши область взаємодії з користувачами тільки заздалегідь оголошеними об'єктами або продуктами [1].

Угода містить інформацію про те, в якому випадку послуга вважається виконаним (коли відповідальність виконавця припиняється) - це значить організації, яка обслуговує, припустимо, комерційну нерухомість, вже не доведеться ремонтувати дах за свій рахунок, якщо нові проблеми розкрилися через 2 роки після попереднього ремонту , а заявлена гарантія на роботи - 1 рік.

В SLA прописуються параметри послуги і їх допустимі коливання - рівень SLA. Наприклад, в угоді можна прописати, що фахівець підтримки має право відповідати протягом 4-х годин після реєстрації заявки, а не миттєво, а також не відповідати у вихідні та святкові дні. Однак щоб SLA не перетворилося на головний біль для всіх зацікавлених сторін, важливо вказувати там реально досяжні параметри послуг. І якщо вони когось не влаштовують, краще обговорювати їх на етапі підписання - розставити всі крапки над "і", щоб ні у кого не було завищених очікувань. До речі, час - далеко не єдино можливе джерело метрик для послуг, однак пропонувати надто багато параметрів або використовувати якісь непрямі показники, слабо корелюють з діями виконавця, не варто. Вони тільки ускладнюють роботу.

Зміст SLA з боку сервісного відділу або компанії - це набір цільових метрик, до яких прагнуть виконавці (на відміну від KPI, що представляють собою фактичне вимір послуги, але часто змішуються з поняттям SLA). Недотримання заявлених умов - причина почати внутрішнє розслідування і депремірovать винних. Вибір правильних показників для контролю, як вибір правильних метрик, вимагає досвіду і розуміння ситуації. Наприклад, не можна бездумно мотивувати співробітників вирішувати завдання клієнта швидше - так постраждає якість рішення.

Створення системи аналізу показників якості сервісів, неодмінно, є в інтересах компанії-постачальника, адже бізнес-аналітики компанії в будь-який момент зможуть подивитися на історичні дані показники сервісів, створити модель

їх прогнозу на майбутнє, розрахувати піки навантаження на сервіси, спрогнозувати кількість ресурсів, що будуть використовуватися всіми сервісами компанії, та таким чином розрахувати вартість інфраструктури, необхідність придбання додаткових серверів, тощо.

1.2 Аналіз компонентів сучасної корпоративної IT-інфраструктури

Сучасна корпоративна IT-інфраструктура, зазвичай, складається із внутрішніх та зовнішніх сервісів. Внутрішніми називають сервіси, які використовуються виключно співробітниками компанії та, зазвичай, є доступними тільки з корпоративної мережі шляхом фізичного знаходження співробітника на території офісу, або ж завдяки підключення до корпоративної мережі завдяки технології VPN - віртуальної приватної мережі. Таким чином, створюється “тунель” між комп'ютером співробітника і сервером, в якому вся інформація зашифрована, і провайдер не розуміє, на який сайт переходить користувач. Зовнішні сервіси, в свою чергу, доступні через мережу “Інтернет” будь-якому користувачу, якому компанія надає послуги (з яким укладено контракт на надання послуг).

Зазвичай, кожен сервіс, незалежно від доступності зовнішнім користувачам, взаємодіє зі своєю базою даних, в якій зберігається “бізнес” інформація. Наприклад, список користувачів, список контрактів, товарів, доставок, тощо. Такі бази даних називають OLTP (Online Transaction Processing) - обробкою онлайн транзакцій. Основна функція подібних систем полягає у виконанні великої кількості коротких транзакцій. Самі транзакції виглядають відносно просто, наприклад, "зняти суму грошей з рахунку A, додати цю суму на рахунок B". Проблема полягає в тому, що, по-перше, транзакцій дуже багато, по-друге, виконуються вони одночасно (до системи може бути підключено кілька тисяч одночасно працюючих користувачів), по-третє, при виникненні помилки, транзакція повинна цілком відкотитися і повернути систему до стану, який був до початку транзакції (не повинно бути ситуації, коли гроші зняті з рахунку A, але не надійшли на рахунок B). Практично всі запити до бази даних в OLTP-додатках складаються з команд вставки, оновлення, видалення. Таким чином, критичним для OLTP-додатків є швидкість і

надійність виконання коротких операцій оновлення даних. Чим вище рівень нормалізації даних в OLTP-додатку, тим воно, як правило, швидше і надійніше [4].

Іншим типом систем зберігання даних є, так звані, OLAP-додатки (On-Line Analytical Processing - оперативна аналітична обробка даних). Це узагальнений термін, що характеризує принципи побудови систем підтримки прийняття рішень (Decision Support System - DSS), сховищ даних (Data Warehouse), систем інтелектуального аналізу даних (Data Mining). Такі системи призначені для знаходження залежностей між даними (наприклад, можна спробувати визначити, як пов'язаний обсяг продажів товарів з характеристиками потенційних покупців), для проведення подальшого аналізу. OLAP-додатки оперують великими масивами даних, вже накопиченими в OLTP-додатках, взятими їх електронних таблиць або з інших джерел даних. Додавання в систему нових даних відбувається відносно рідко великими блоками (наприклад, раз в квартал завантажуються дані за підсумками квартальних продажів з OLTP-додатків). Також, дані, додані в систему, ніколи не видаляються [5].

Перед завантаженням дані проходять різні процедури "очищення", пов'язані з тим, що в одну систему можуть надходити дані з багатьох джерел, що мають різні формати представлення для одних і тих же понять, дані можуть бути некоректними, помилкові. Запити до системи є не регламентованими і, як правило, досить складними. Швидкість виконання запитів важлива, але не критична, оскільки найголовнішою задачиєю таких систем є підтримка даних у консистенстному стані у будь-який момент часу.

Дані OLAP-додатків, зазвичай, представлені у вигляді таблиці фактів та однією або більше таблиць вимірів. Наприклад, можна побудувати таблицю фактів продажів товарів, і таблиці вимірів типу товару, відділення компанії, інформація про покупця, тощо. Такий спосіб організації даних буде містити інформацію про продажі різних типів товарів по підрозділам, товарам. Грунтуючись на цих даних, можна відповідати на питання на зразок "у якого підрозділу найкращі обсяги продажів в поточному році?", Або "які тенденції продажів відділень Південно-Західного регіону в поточному році в порівнянні з попереднім роком?".

Повертаючись до проблеми нормалізації даних, можна сказати, що в системах OLAP, що використовують реляційну модель даних (ROLAP), дані доцільно зберігати у вигляді слабо нормалізованих відносин, що містять заздалегідь обчислені основні підсумкові дані. Велика надмірність і пов'язані з нею проблеми тут не страшні, тому що оновлення відбувається тільки в момент завантаження нової порції даних. При цьому відбувається як додавання нових даних, так і перерахунок підсумків [6]. OLAP системи, на відміну від OLTP, набагато швидше виконують аналітичні запити, часто надають користувачам можливість створення звітів у різних форматах даних.

1.2.2 Системи аналізу даних на базі “Озера даних”

«Озеро даних» (data lake) - це елемент інфраструктури Big Data, сховище великої кількості неструктурованих даних, генерованих або зібраних однією компанією або державною установою. Дані в “озерах” зберігаються, як правило, в несистематизированном (неструктурованому) вигляді в форматах даних, які надаються постачальником даних і, можливо, не є зручними для подальшого опрацювання та аналізу [7]. Шаблони проектування представлені на рисунку 1.1.

Компанії створюють озера даних з кількох причин, серед яких: необхідність мати всі матеріали на випадок перевірки, потенційна цінність даних в майбутньому, вимоги закону і інші.

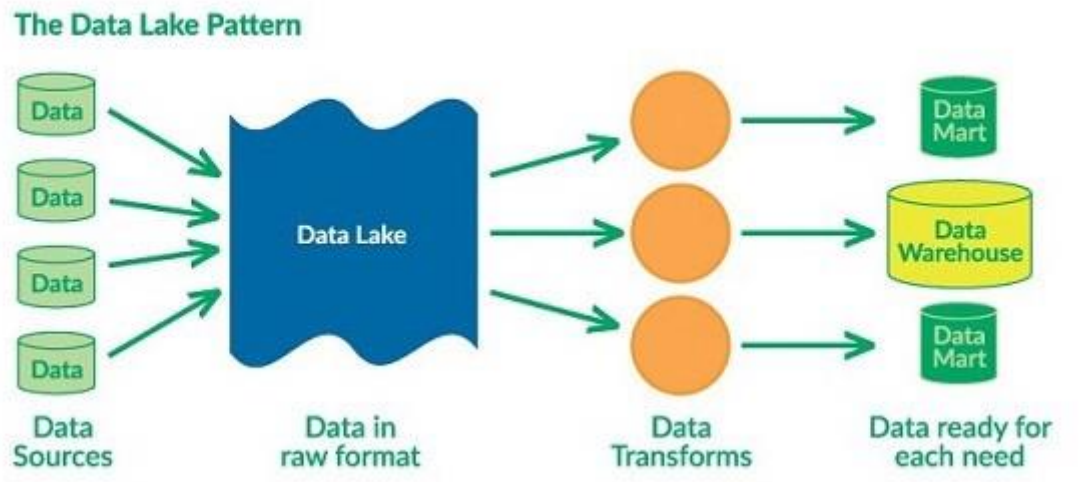

Рисунок 1.1 - Шаблон проектування озера даних

Озера даних можуть знаходитися на серверах самої компанії або в хмарному сховищі. Доступ до даних мають, як правило, всі співробітники, а ступінь захищеності озер низька. Зберігання таких даних, зазвичай, не коштує багато грошей, оскільки на цьому етапі дані зберігаються в вигляді файлів на розподіленій файловій системі. Також, такі дані доцільно архівувати для зменшення вартості зберігання, проте якщо вони використовуються часто, то час, витрачений на архівування та розархівування, може значно перевищувати час опрацювання.

Зберіганням і адмініструванням озер даних нині займаються спеціалізовані фірми: Teradata, Zaloni, HVR, Podium Data, Snowflake і інші. Більшість компаній надають не тільки потужності для зберігання, але і інструменти для структуризації озер і обробки даних [8].

Згідно з прогнозом Markets and Markets, до 2021 року ринок озер даних виросте до $ 8,81 млрд із річним темпом росту 28,3%. Сьогодні озера є необхідною частиною будь-якої корпоративної інфраструктури Big Data.

Головна проблема озер даних, як і природних водойм, в тому, що вони можуть забруднюватись і перетворюватися на болота. Іншими словами, сховища бувають настільки структуровані і завалені неоднорідними даними, що розібратися у всьому цьому і тим більше отримати цінну інформацію не є можливим, тому слід призначати значну увагу структурі озера, створити на рівні компанії правила використання озера, періодично видаляти старі дані, які більше не використовуються.

1.2.3 Корпоративне сховище даних

Сховище даних - це центральний репозиторій інформації, який можна аналізувати для прийняття більш обгрунтованих рішень. Сховище має чітку структуру (рис 1.2) та складається з декількох слоїв. Дані надходять в сховище з транзакційних систем, реляційних баз даних і інших джерел - як правило, з певною періодичністю. Бізнес-аналітики, фахівці по роботі з даними і особи, відповідальні

за прийняття рішень, отримують доступ до даних за допомогою інструментів бізнес-аналітики, SQL-клієнтів і інших додатків для аналітики [9].

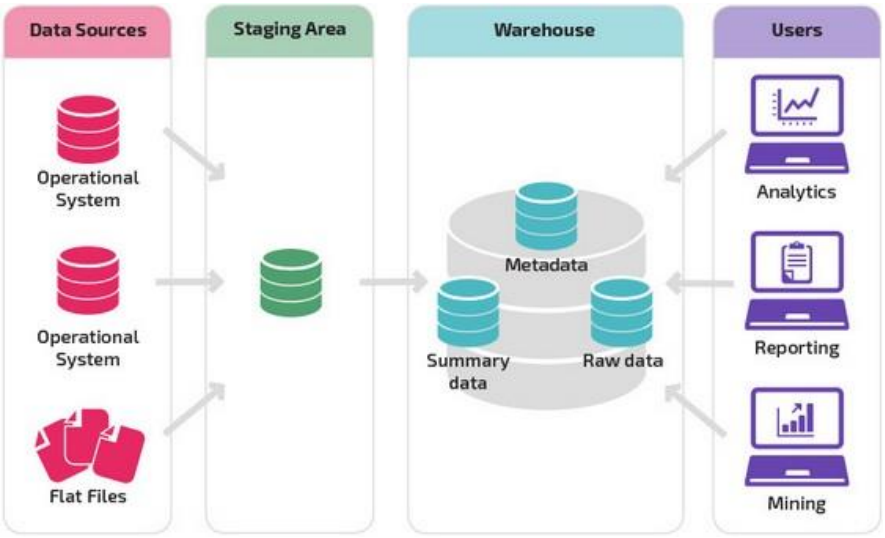

Рисунок 1.2 - Структура корпоративного сховища даних

Дані і інструменти аналітики незамінні для компаній, які прагнуть зберегти переваги перед конкурентами. Щоб перетворювати дані в корисну аналітичну інформацію, стежити за ефективністю ведення бізнесу і приймати обґрунтовані рішення, компанії використовують звіти, панелі управління і різні аналітичні інструменти. За всіма цими звітами, панелями управління та аналітичними інструментами стоять сховища даних. У них дані зберігаються більш ефективно, з меншою кількістю операцій читання і запису, а результати запитів до сховищ блискавично доставляються сотням і тисячам користувачів одночасно [10].

Архітектура сховища даних складається з трьох рівнів. Нижній рівень архітектури - сервер бази даних, що відповідає за завантаження і зберігання даних. Середній рівень - аналітичний механізм, який використовується для доступу до даних і їх аналізу. Верхній рівень - інтерфейсний клієнт, що надає результати з використанням інструментів формування звітів, пошуку та аналізу даних [11].

Для упорядкування даних в сховищі даних використовується схема, що описує розташування і типи даних (наприклад, ціле число, поле даних або рядок). Після надходження дані зберігаються в різних таблицях, описаних в цій схемі. З її

допомогою інструменти запитів визначають, до яких таблиць даних слід звернутися для аналізу.

1.3 Переваги та недоліки будування інфраструктури в хмарному середовищі

Рівень надійності сучасних хмар корпоративного рівня набагато вище, ніж надійність власної фізичної IT-інфраструктури. Для забезпечення безперервності бізнес-процесів в будь-яких ситуаціях професійний сервіс-провайдер обов'язково резервує компоненти інформаційної системи, в тому числі в георозподілених. Крім того, це можуть бути центри обробки даних не тільки на території України, але і за кордоном, забезпечуючи максимальний рівень доступності інформаційних систем за рахунок синхронної і асинхронної реплікації даних на рівні систем зберігання, віртуальних машин і додатків. Крім резервування, для побудови свого хмари професійний сервіс-провайдер використовує перевірені рішення від світових лідерів IT-індустрії, які надають якісну проактивний підтримку.

Крім надійної технологічної бази, хмари сьогодні мають достатню інформаційну безпеку на всіх рівнях архітектури віртуального дата-центру як відповідно до світових стандартів, так і вимогами регуляторів. Для побудови захищених систем застосовуються сертифіковані засоби захисту інформації для каналів зв'язку, периметра мережі, засобів віртуалізації, операційних систем, середовища адміністрування віртуального дата-центру і т.д. Таким чином, використовуючи хмари корпоративного рівня, замовник може бути впевнений у виконанні вимог законодавства в області захисту конфіденційної інформації і персональних даних.

Серед переваг хмар можна виділити масштабованість і гнучкість, тобто замовник залежно від бізнес-завдань може оперативно змінити споживані IT-ресурси, які виділяються для компанії із загального пулу обчислювальних ресурсів корпоративного хмари сервіс-провайдера. Таким чином, хмара дозволяє гнучко збільшити або зменшити обчислювальні потужності і сховище даних в залежності від сезонності попиту або пікових навантажень. При цьому протягом декількох хвилин з будь-якої точки світу можна створювати віртуальні машини довільних

конфігурацій і управляти конфігурацією мереж за допомогою програмно визначаються комутаторів і маршрутизаторів без зупинки бізнес-процесів. Важливим є те, що вартість володіння інфраструктурою в цьому випадку визначається лише кількістю фактично спожитих ресурсів. Цей принцип в цілому знижує рівень витрат компанії на володіння IT-інфраструктурою, при цьому самі витрати переходять з капітальних в операційні. В лідерах ринку провайдерів хмарних послуг можна знайти найбільші IT-компанії світового рівня (рис 1.3), що означає активну підтримку цієї галузі зі сторони компаній, що формують IT ринок світу.

Для різних потреб бізнесу використовується гібридна архітектура інформаційних систем з приватного, корпоративного і публічного хмари, відповідно, великий сервіс-провайдер може забезпечити доступ до глобальних світових сервісів, створюючи високошвидкісний канал з'єднання між даними компонентами гібридної інфраструктури.

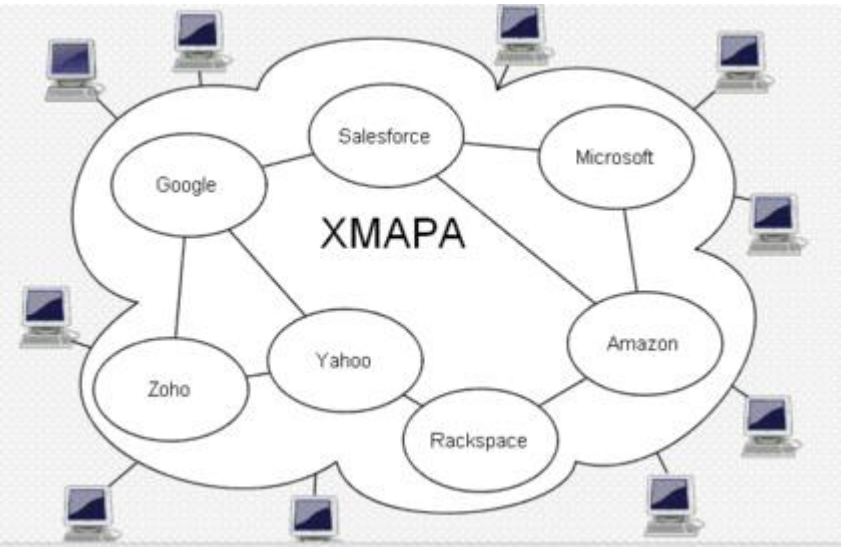

Рисунок 1.3 - Основні провайдери хмарних послуг

Під технологією хмарних обчислень (cloud computing) розуміється інноваційна технологія, яка дозволяє об'єднувати IT-ресурси різних апаратних платформ в єдине ціле і надавати користувачеві доступ до них через локальну мережу або глобальну мережу Інтернет. Хмарні сервіси від різних провайдерів пропонують користувачам через мережу Інтернет доступ до своїх ресурсів за

допомогою безкоштовних або умовно безкоштовних хмарних додатків, апаратні і програмні вимоги яких не припускають наявності у користувача високопродуктивних і ресурсопотребляемих комп'ютерів.

Загальна тенденції розвитку описаної галузі зростає, адже розвиваються усі напрямки розроблення хмарних послуг, в тому числі обчислювальних ресурсів як сервіс (Infrastructure as a service), систем зберігання даних (Storage as a service), сервісів по відновленню даних (Disaster Recovery Service), сервісів інформаційної безпеки, програмного забезпечення по підписці (Software as a service), адміністрування і т.д. За нашими прогнозами, ще протягом декількох років ми будемо спостерігати інтенсивне зростання ринку хмарних послуг - не менше 25-30% в рік, в тому числі у віддалених регіонах цей процес буде пов'язаний з проникненням високошвидкісного Інтернету. При цьому буде збільшуватися частка хмарних технологій в гібридних архітектур інформаційних систем компаній, так як буде накопичуватися експертиза у використанні хмар, кількість успішних бізнес-кейсів і, відповідно, рівень довіри. Одночасно з розширенням використання хмар буде підвищуватися затребуваність захищених приватних хмарних рішень.

Галузь хмарних обчислень розвивається в напрямку загальносвітових тенденцій, в тому числі і з боку сервіс-провайдерів. Створюється все більше механізмів стандартизації і регламентації даної галузі, наприклад складання угоди про рівень надання послуг стає неодмінною частиною процесу впровадження хмарних послуг у компанії. З боку держави будуть формуватися регулюють галузь нормативні документи, а з технологічної точки зору хмари будуть все більш продуктивними (максимальна швидкість обчислення при обробці великого обсягу інформації) і менш коштовними. Є й інша сторона розвитку хмарних технологій: експоненціальне зростання даних - основний об'єм інформації припадатиме на архівні копії, які не будуть затребувані і не будуть видалені.

В довгостроковій перспективі розвиток хмарних сервісів буде пов'язане з впровадженням систем машинного навчання: штучного інтелекту, нейронних мереж, доповненої реальності, а також нейроінтерфейси. Так, вже зараз знаходиться в глибокій розробці концепція туманних обчислень, яка передбачає

використовувати для зберігання та аналізу даних не центральні вузли мережі дата-центрів, а ресурси великої кількості географічно розподілених персональних пристроїв (ПК, планшетів, гаджетів, дронів, побутових приладів і т. д.), по суті, реалізуючи принцип розподілу обчислювальної здатності практично на всі навколишні нас пристрою.

1.4 Порівняльний аналіз існуючих рішень систем аналізу якості сервісів в хмарному середовищі

До найвідоміших та найбільш використовуваних постачальників хмарних послуг можна віднести Amazon Web Services, Google Cloud Platform та Microsoft Azure. Кожен з цих провайдерів надає користувачам можливість налаштовувати свою систему моніторингу та реакцій.

Amazon CloudWatch - це сервіс моніторингу і спостереження зі зручним користувацьким інтерфейсом (рис 1.4), що створений для інженерів DevOps, розробників, інженерів по надійності сайтів та IT-менеджерів. CloudWatch надає дані і дієві аналітичні відомості для моніторингу додатків, реагування на зміни продуктивності в масштабах системи, оптимізації використання ресурсів та отримання єдиного уявлення про працездатність системи. Завдяки інтеграції Amazon CloudWatch з сервісом AWS Identity and Access Management (IAM) можна вказувати, які дії CloudWatch користувач може виконувати в рамках аккаунта AWS. Наприклад, можна створити політику IAM, яка дозволить тільки певним користувачам організації використовувати API GetMetricStatistics. Тоді вони зможуть виконувати дію для отримання даних по хмарним ресурсів.

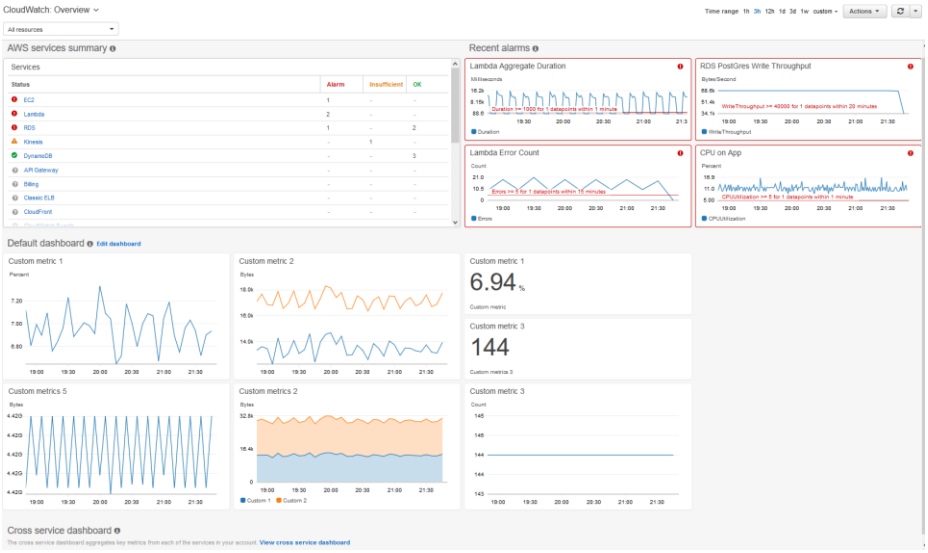

Рисунок 1.4 - Користувацький інтерфейс продукту Amazon CloudWatch

CloudWatch збирає дані моніторингу та операційні дані у вигляді журналів, метрик і подій, допомагаючи отримати єдине уявлення додатків, сервісів і ресурсів AWS, що працюють на платформі AWS, а також в локальному середовищі. За допомогою CloudWatch користувачі можуть виявляти аномальну поведінку в своїх середовищах, налаштовувати попередження, створювати спільні візуальні уявлення журналів і метрик, виконувати автоматизовані дії, усувати неполадки, а також дізнаватися корисні відомості, які допоможуть вам підтримувати стабільну роботу додатків. З даним сервісом легко інтегрується ще один, який дає змогу налаштовувати нотифікації для подальшого реагування [15].

Amazon Simple Notification Service (SNS) - це високодоступний, надійний, безпечний, повністю керований сервіс відправки повідомлень за моделлю «видавець - передплатник» (Pub / Sub), за допомогою якого можна ізолювати мікросервіси, розподілені системи і бессерверной додатки. В Amazon SNS можна використовувати теми для високопродуктивної розсилки push-повідомлень відразу багатьом одержувачам. Використовуючи теми Amazon SNS, системи публікацій можуть розсилати повідомлення великій кількості кінцевих передплатників для паралельної обробки, включаючи черги Amazon SQS, функції AWS Lambda і об'єкти HTTP / S webhooks. Крім того, SNS можна використовувати для розсилки інформації кінцевим користувачам за допомогою мобільних push-повідомлень, SMS-повідомлень і електронних листів. Даний сервіс відправки нотифікацій також

підтримує інтеграцію з популярними корпоративними месенджерами, наприклад, з системою Slack.

Служба Azure Monitor забезпечує максимальну доступність і продуктивність додатків і служб, надаючи повноцінне рішення для збору, аналізу і обробки даних телеметрії з хмарних і локальних середовищ. Вона допоможе вам зрозуміти, як виконуються програми, а також заздалегідь визначити проблеми, що впливають на них, і ресурси, від яких вони залежать. Azure Monitor дозволяє виявляти і діагностувати проблеми і залежності між додатками за допомогою Application Insights. Також, сервіс надає можливість зіставляти проблеми інфраструктури з Azure Monitor для віртуальних машин і Azure Monitor для контейнерів. Однією з переваг можна назвати здатність аналізувати дані моніторингу за допомогою Log Analytics для усунення неполадок і виконання глибокої діагностики [16].

На наведеній нижче схемі показано високорівневе представлення Azure Monitor (рис 1.5). У центрі схеми знаходяться сховища даних для метрик і журналів - двох основних типів даних, що використовуються службою Azure Monitor. У лівій частині знаходяться джерела даних моніторингу, що наповнюють ці сховища даних. Справа знаходяться різні операції, які Azure Monitor виконує із зібраними даними (наприклад, аналіз, оповіщення та потокова передача в зовнішні системи). Даний сервіс інтегрується з усіми популярними сервісами хмарного провайдера Azure. Крім можливості відслідковувати інформацію про метриках і події використовуваних компонентів Azure також є додавання власних custom metrics & events.

Завдяки такому компоненту як Azure Service Health з'являється можливість своєчасно дізнаватися про технічні роботах і збої в інфраструктурі Azure, які можуть торкнутися доступність розгорнутих в хмарі сервісів і ресурсів. Як сповіщень можуть виступати повідомлення поштою, телефоном або webhook.

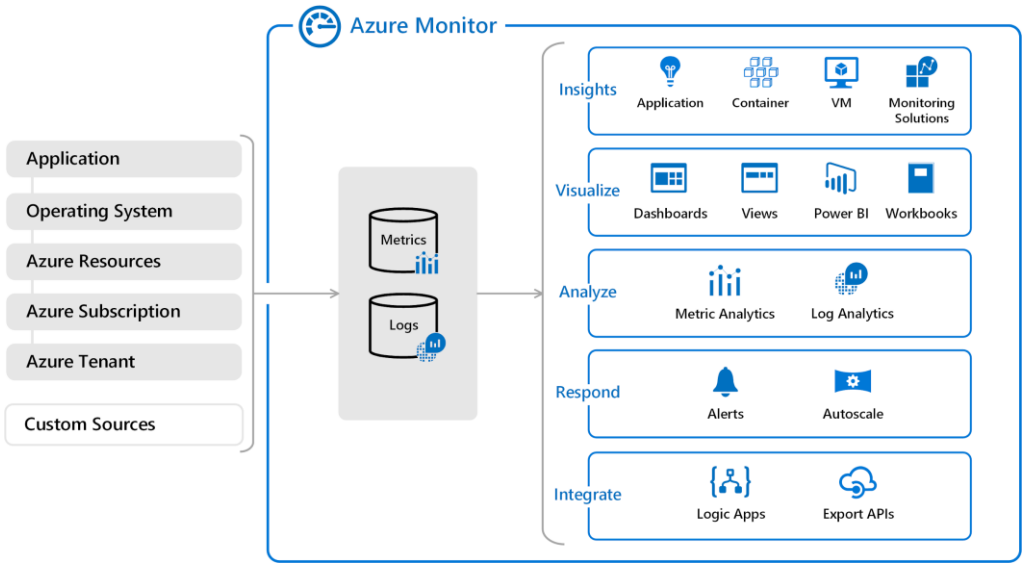

Рисунок 1.5 - Структура продукту Azure Monitor

Всі дані, що збираються службою Azure Monitor, відповідають одному з двох основних типів, тобто представляють собою метрики або журнали. Метрики - це числові значення, що описують конкретний аспект системи в певний момент часу. Вони займають невеликий обсяг, і їх можна використовувати в сценаріях, які передбачають роботу в режимі практично реального часу. Журнали містять дані різних типів, впорядковані по записах з різними наборами властивостей для кожного типу. Крім даних продуктивності в системі (у вигляді журналів) зберігаються дані телеметрії, наприклад події та трасування, так що можна об'єднати всі ці дані для аналізу.

Для багатьох ресурсів Azure дані, зібрані службою Azure Monitor, відображаються безпосередньо на сторінці "Огляд" порталу Azure. Зверніть увагу на будь-яку віртуальну машину, і ви побачите кілька діаграм з метриками продуктивності. Клацніть будь-який графік, щоб відкрити дані в браузері метрик на порталі Azure, де ви можете відзначати значення декількох метрик з плином часу. Ви можете переглядати графіки в інтерактивному режимі або закріпити їх на панелі моніторингу, щоб переглядати їх разом з іншими візуалізаціями.

Моніторинг Stackdriver - це потужний і універсальний інструмент моніторингу хмар, який тісно інтегрований практично з будь-яким сервісом на платформі Google Cloud. Даний сервіс підтримує безперебійну перевірку HTTP,

HTTPS і TCP. Зонди, що надсилаються цими чеками, регулюються правилами брандмауера VPC, тому вони також повинні бути налаштовані правильно.

Google Cloud Platform дає користувачам можливість моніторити додатки завдяки сервісу Stackdriver, який забезпечує видимість продуктивності, тривалості роботи та загального стану здоров'я хмарних додатків. Stackdriver збирає метрики, події та метадані з Cloud Cloud Platform, Amazon Web Services, розміщує зонди часу безперебійного користування, інструментарій програм та різноманітних загальних компонентів додатків, зокрема Cassandra, Nginx, веб-сервер Apache, Elasticsearch та багато інших. Stackdriver отримує дані та генерує статистику за допомогою інформаційних панелей, діаграм та сповіщень.

Моніторинг Stackdriver автоматично виявляє та відстежує ваші хмарні ресурси, незалежно від того, працюєте ви на хмарній платформі Google або веб-службах Amazon. Послуга включає в себе панелі інформаційних панелей за замовчуванням для багатьох служб Google Cloud Platform та AWS, яка містить не тільки метрики, але й критичні метадані від цих провайдерів, що полегшує розуміння взаємозв'язку між компонентами та моніторинг кластерів автоматичного масштабування.

Розширені можливості оповіщення, включаючи швидкість змін, агрегацію кластерів та політику щодо багатьох умов, допомагають гарантувати вам повідомлення про виникнення критичних проблем, зменшуючи при цьому ймовірність помилкових позитивних результатів.

Гнучкі інформаційні панелі та багаті інструменти для візуалізації допомагають визначити нагальні проблеми. Звіт про аномалію, виявлення шаблонів та прогнозування виснаження дають уявлення про довгострокові тенденції, які можуть вимагати уваги. Зручний користувацький інтерфейс продукту дозволяє клієнтам сервісу швидко та зручно створювати звіти моніторингу.

1.5 Висновок до розділу

В даному розділі була розглянута угода про надання послуг та описана важливість задачі підтримки якості сервісів на узгодженому рівні. З ростом

популярности надання послуг онлайн, зросла і необхідність в захисті прав користувачів таких сервісів. Таким захистом часто виступає угода про рівень надання послуг, у якому чітко прописані збитки для компанії у разі якщо послуги було надано несвоєчасно або ж не в повному обсязі. В свою чергу, компанії-постачальники стають все більш зацікавлені в захисті своїх прав, а отже автоматизація процесу підтримки сервісів на заданому рівні та створення системи аналізу показників якості сервісів стають все більш пріоритетними задачами великих компаній.

Були проаналізовані компоненти сучасної IT-інфраструктури, була обґрунтована необхідність будування озера даних і корпоративного сховища даних для задач аналітики. Із ростом компанії задача аналізу даних стає все більш актуальною, оскільки з ростом користувацької бази постає необхідність прогнозувати навантаження на сервіси, прогнозувати кількість обчислювальних ресурсів на наступний квартал, інше.

Також, були проаналізовані переваги та недоліки будування хмарної IT-інфраструктури, описані визначення та моделі надання хмарних послуг.

Наостанок, були проаналізовані існуючі рішення систем аналізу якості сервісів в хмарному середовищі для найбільших провайдерів хмарних послуг - Amazon Web Services, Google Cloud Platform та Microsoft Azure [17].

Кожен провайдер надає користувачам можливість налаштовувати систему моніторингу та реакції подій, проте найголовнішим недоліком таких рішень є прив'язка до сервісів певного провайдера. Тобто провайдер зацікавлений надавати зручні сервіси моніторингу, які дуже легко інтегруються з іншими сервісами саме цього провайдера хмарних послуг. Таким чином провайдери заохочують використовувати свої і тільки свої послуги та сервіси.

На початкових етапах формування компанії використання публічних хмарних провайдерів є хорошою ідеєю, оскільки на початкових стадіях важко спрогнозувати кількість обладнання, яке буде необхідним в майбутньому. Проте для успішних компаній використання публічних хмарних провайдерів стає все більш і більш коштовним, тому компанії на певному етапі розвитку замислюються про перехід на власні технологічні розробки. Повний перехід не є обов'язковий, проте слід завжди

приділяти увагу коштам, витраченим на кожен із сервісів провайдера хмарних послуг, адже деякі сервіси, наприклад сервіс моніторрингу, можуть коштувати набагато більше, ніж аналоги продуктів сторонніх по відношенню до хмарного провайдера компаній.

2 РОЗРОБЛЕННЯ СИСТЕМИ УПРАВЛІННЯ ФУНКЦІОНУВАННЯМ КОРПОРАТИВНОЇ ІТ-ІНФРАСТРУКТУРИ

2.1 Розроблення структурної схеми системи управління функціональності

Система управління функціонуванням складається із користувацького інтерфейсу, серверної частини, бази даних та агентських додатків (рис 2.1).

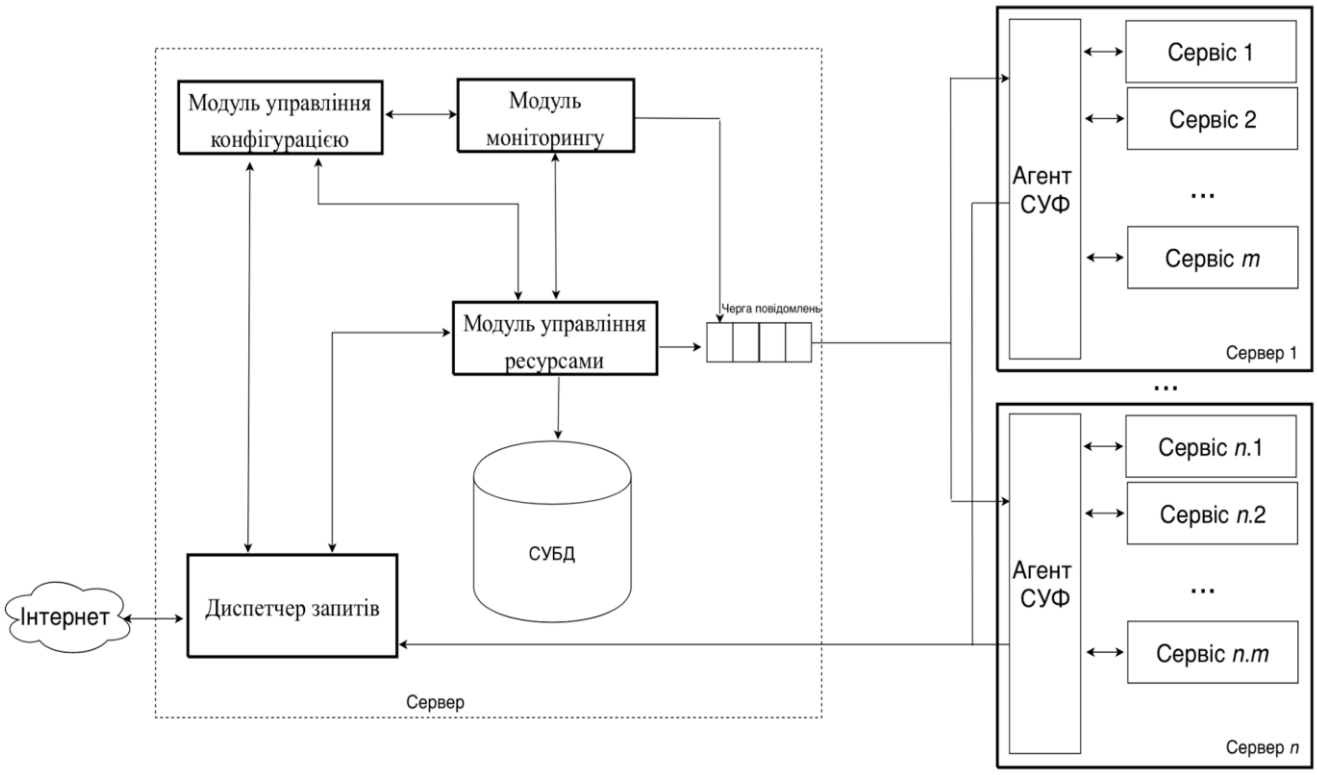

Рисунок 2.1 – Структурна схема системи управління функціональності

Агентські додатки виконують встановлюються на кожен сервер корпоративної ІТ-інфраструктури. Після встановлення програмного забезпечення агентського додатку на сервер, в автоматичному режимі відбувається реєстрування серверу в системі. Це досягається за допомогою спеціального програмного продукту під назвою “Програма виявлення сервісів”. Принцип дії даного продукту таких - в код кожного агентського додатку завантажена IP адреса та порт програми виявлення сервісів. Після першого старту агента, автоматично надсилається на нтого запит реєстрації нового агенту в системі. Після отримання відповіді про успішну

самореєстрацію, агентський додаток починаю збирати статистику про сервер системи, на якому він встановлений.

Модуль моніторингу періодично взаємодіє із агентськими додатками та збирає від них інформацію про стан серверу та сервісів, що запущені на ньому.

Модуль управління ресурсами, в свою чергу, відповідає підтримку показників якості сервісів на узгодженому рівні шляхом надання додаткових ресурсів відповідному сервісу.

Функціональне призначення диспетчеру запитів полягає в тому, що він є інтерфейсом взаємодію системи з клієнтськими додатками (рис 2.2), тобто він приймає запити від програми користувацького інтерфейса та перенаправляє їх на внутрішні IP адреси відповідної підсистеми, в залежності від типу запиту.

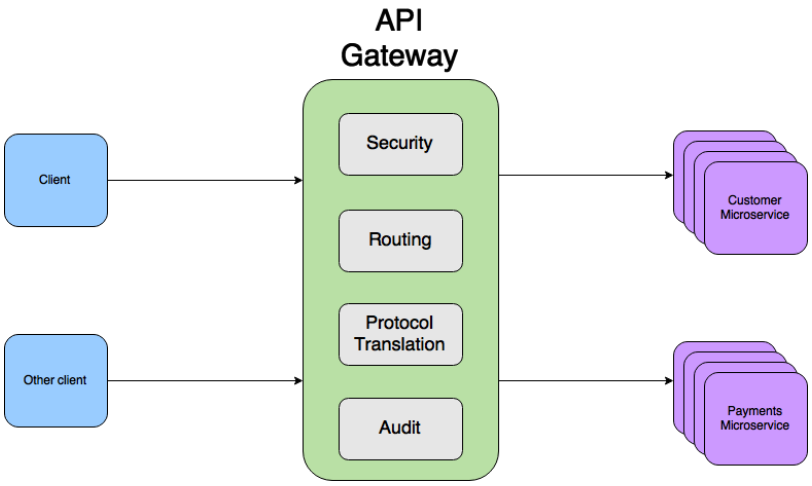

Рисунок 2.2 - Функції шлюзу прикладного програмного інтерфейсу

Для взяємодії серверного та агентських додатків було обрано використання черги повідомлень, адже такий елемент системи дозволяє динамічно додавати агентськи додатки, виступаючи в ролі Service Discovery - програми виявлення сервісів. При першому, агентський додаток реєструє себе у системі в модулі управління ресурсами, після чого усі запити до створеного агенту надсилаються до черги повідомлень із спеціальним ідентифікатором додатку.

2.2 Вибір системи віртуалізації для сервісів корпоративної IT-інфраструктури

Одне з найважливіших завдань розробленої системи - ізолювати корпоративні сервіси один від одного та управляти їх ресурсами (рис 2.3). Для такої задачі доцільно використати систему віртуалізації ресурсів операційної системи, адже віртуалізація дозволяє вирішити такі проблеми досить швидку, не жертвуючі при цьому продуктивністю самого сервісу.

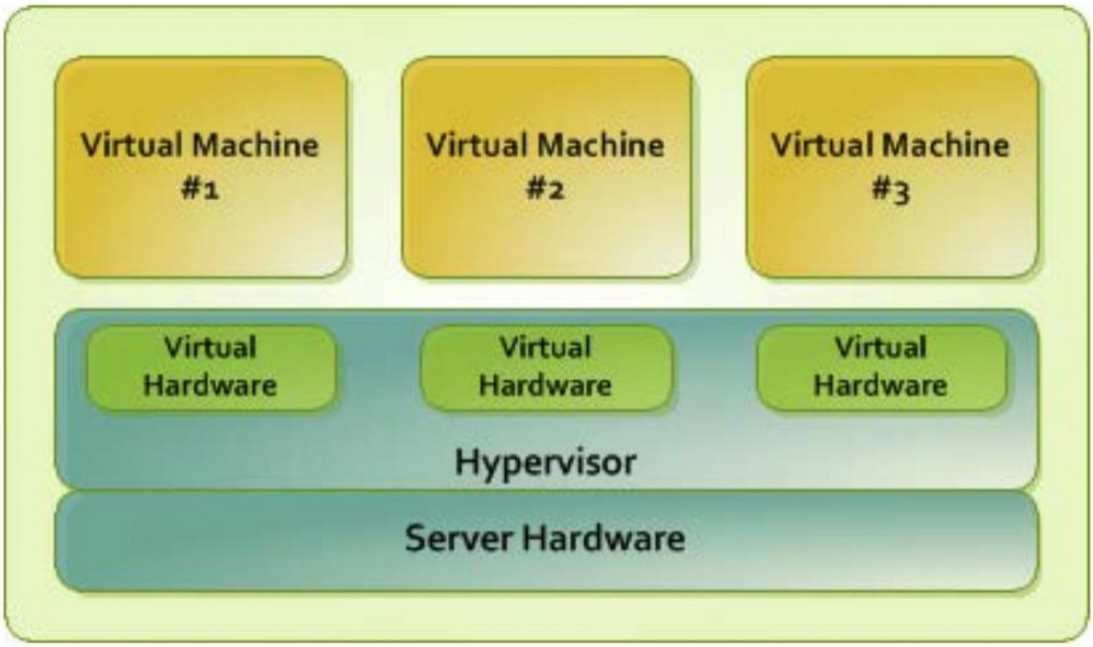

Рисунок 2.3 - Віртуалізація на рівні операційної системи

До переваг віртуалізації також можна віднести економію коштів. Завдяки консолідації ця технологія дозволяє скоротити кількість серверів. Якщо раніше для виконання якогось завдання було потрібно кілька фізичних машин, то тепер можна запустити потрібне число гостьових операційних систем в віртуальному середовищі на одному сервері. Це дозволяє економити на підтримці обладнання. Зменшення кількості фізичних серверів веде до зменшення займаного ними місця. Завдяки цьому зменшується орендна плата за стойки в дата-центрі. Якщо компанія має свій власний ЦОД, то це означає загальне зниження енергоспоживання і тепловиділення системи, отже, з'являється можливість закуповувати менш потужні і більш дешеві системи охолодження, що неодмінно позначиться в рахунках на електрику.

Також варто відзначити, що віртуалізація дозволяє знизити витрати на адміністрування інфраструктури. Одним з головних переваг є можливість

віддаленого доступу до консолі управління. Так, щоб перезавантажити сервер, більше не потрібно йти в машинний зал (або використовувати KVM-перемикачі на базі IP), подати команду перезавантаження можна через консоль. Точно так само можна підключити додаткові обчислювальні ресурси.

Крім цього, ще одним плюсом віртуалізації є простота клонування віртуальних машин. Якщо серверна інфраструктура компанії стандартизована і являє собою групу серверів з однаковими настройками а також керівництвом організації буде прийнято рішення відкрити додатковий офіс, то розгортання нової інфраструктури зведеться до копіювання образів і налаштування програмного забезпечення.

До недоліків віртуалізації можна віднести необхідність перебудови підходу до роботи з надійністю системи. Дійсно, так як на одному фізичному сервері одночасно запущені кілька віртуальних машин, то вихід з ладу хоста призводить до одночасного відмови всіх ВМ і працюють на них додатків. Тому доцільно використовувати відмовостійкі рішення, наприклад на базі відмов кластерів.

У цьому випадку два або кілька серверів працюють в групі і для користувача виглядають як один сервер, який обробляє запити і відповідає за роботу додатків. У разі виходу з ладу одного з вузлів кластера, призначені для користувача програми виконують failover, Автоматичний перезапуск на працездатних вузлах, а застосунок або не припиняє роботу, або припиняє на досить короткий час.

При використанні віртуалізації сервісів перед розробниками постає додаткова задача - балансування навантаження. Наприклад, якщо ВМ використовує багато обчислювальних ресурсів процесора (або пам'яті), то це позначається на роботі інших ВМ хоста, яким також потрібно процесорний час (пам'ять). Адміністраторам доводиться розподіляти навантаження, встановлюючи правила, за якими запущені віртуальні машини будуть автоматично переміщатися на менш навантажені сервера або ж «розвантажувати» завантажені.

Технології віртуалізації покликані вирішити питання з недостатнім ступенем утилізації ресурсів фізичних систем, але тепер з'явилася небезпека вдаритися в іншу крайність. Люди почали створювати ВМ при кожному зручному випадку, часто забуваючи видаляти старі і не використовувані машини. Це веде до складнощів з

нестачею дискового простору, підвищеного споживання обчислювальних ресурсів і перебоїв в роботі [17].

Використання віртуальних машин дозволяє розробнику повністю зімітувати інфраструктуру, в якій проект буде розгорнуто, що знову ж таки дозволяє знизити витрати з відомою проблемою "на моїй машині все працює" і тримати кілька версій ПЗ для різних проектів (якщо проект А використовує PHP 5.3, а проект Б використовує PHP 7.0, то замість жонглювання версіями на робочій машині розробник може їх тримати в окремих ізольованих машинах).

Порівнюючи зв'язку Vagrant та VirtualBox з системою Docker дозволяють його реалізувати, тому принципової різниці немає, тому має сенс просто описати концепції обох інструментів.

VirtualBox сам по собі є просто менеджером віртуальних машин, який дозволяє запускати їх з образів. Тут все просто: береться образ файлів системи, емулюється процесор, оперативна пам'ять, жорсткий диск, віртуальна машина шукає бут-сеткор на віртуальному диску, і машина далі стартує, як ніби вона реальна.

Vagrant є менеджером коштів віртуалізації (рис 2.4), і насправді може запускати не тільки VirtualBox, але й Docker. Саме в цьому розрізі Vagrant дозволяє автоматизувати створення віртуальної машини (або віртуальних машин) VirtualBox з образу чистої операційної системи, застосувавши так звану провізію - ініціалізувати машину за допомогою shell-скрипта або менеджера конфігурації.

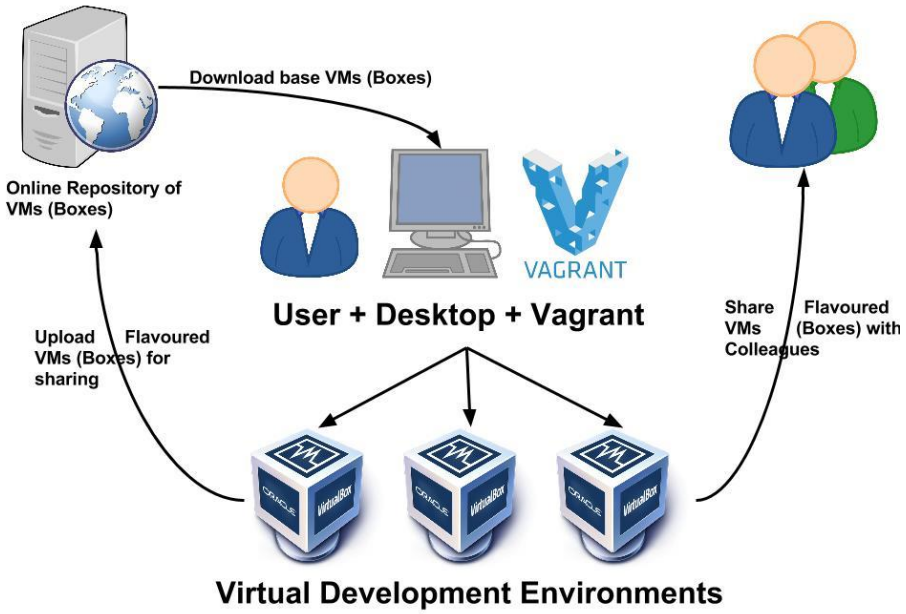

Рисунок 2.4 - Призначення системи Vagrant

Docker сам по собі надає рівно такий же функціонал віртуалізації, але дуже сильно відрізняється в плані концепції. Наприклад, він використовує Copy-On-Write систему шарів для реалізації файлової системи. Кінцева файлова система збирається з декількох шарів, в результаті чого зображення можуть ділити нижні шари, а для кожної віртуальної машини використовується свій власний шар (верхній в цій композиції), тому машини можуть використовувати загальну базу, яку не потрібно копіювати для старту машини. Також, Docker використовує систему просторів імен в ядрі Linux (знову ж важко сказати, як це організовано в Mac / Windows).

Це дозволяє ставитися до машин, як до видаткового матеріалу і пересоздавать їх сотні раз в секунду (це дуже швидкий процес в порівнянні з підняттям повноцінної віртуальної машини). Завдяки цьому сформувалася філософія Docker, в якій прийнято правило process-per-container (один процесс на контейнер), і замість поширення віртуальних машин концепція Docker має на увазі поширення доданків, кожне з яких загорнуте в свій образ віртуальної машини для ізоляції [18].

З точки зору віртуалізації всередині відбувається практично те ж саме що і зі звичайною віртуальною машиною: запускається кореневої процес, який піднімає всі інші. В образах Vagrant-VirtualBox зазвичай використовується звичний init, в

Docker-образах - сам додаток, таким чином виходить, що у віртуальній машині Vagrant-Virtualbox запускається менеджер ОС, що стежить за всім, що відбувається, а в Docker-образі - нічого, крім кінцевого додатки.

Vagrant - це інструмент, що дозволяє уніфікувати середовище розробки і середу в якій цей код буде розгортатися, використовуючи технологію віртуалізації. Даний продукт розроблений компанією HashiCorp, що спеціалізується на інструментах для автоматизації розробки і експлуатації. Він дозволяє створювати і конфігурувати легковагі, повторювані і переносяться оточення для розробки.

Таким чином Virtualbox, Vagrant і Docker розрізняються головним чином концепцією, яка не несе функціональної різниці. Проте, не можна не відзначити, що ком'юніті для кожного інструменту більше розвинене в тому напрямку, в якому використовується сам інструмент: використовуючи Docker, користувач можете користуватися загальнодоступними ізольованими машинами з одним процесом в кожній машині і конфігурацією через змінні оточення, використовуючи Vagrant, користувачі отримують доступ до машин, які налаштовані через configuration manager (chef, puppet, saltstack) або і зовсім просто через shell-скрипт і готовим до установки великої кількості сервісів на одну машину [19]. Також, існує розширення для Docker під назвою Docker-compose, завдяки якому користувачу надана можливість створювати один файл з описом усіх залежностей сервісів між собою та запускати декілька сервісів однією однією командою.

2.3 Імплементація програмного забезпечення системи управління функціональності

2.3.1 Вибір інструментів розробки

Для реалізації системи управління була обрана мова програмування Java версії 1.10, оскільки дана мова програмування має низку ключових переваг перед конкурентними рішеннями. Наприклад, Java є об'єктно-орієнтованою мовою програмування. В Java все є об'єктом. Доповнення може бути легко розширено, тому що він заснований на об'єктної моделі.

Також, до переваг обраної мови програмування можна віднести її кросплатформеність, легкість читання і ясність. Крім того, Java підтримує сучасні механізми багаторазового використання програмного коду з арсеналу об'єктно-орієнтованого і функціонального програмування. У складі Java поставляється велика кількість зібраних і переносяться функціональних можливостей, що становлять стандартну бібліотеку.

Весь код було написано в програмне забезпечення JetBrains IntelliJ IDEA (рис 2.5). Даний продукт - це провідна середовище швидкої розробки на мові Java. IntelliJ IDEA являє собою високотехнологічний комплекс тісно інтегрованих інструментів програмування, що включає інтелектуальний редактор вихідних текстів з розвиненими засобами автоматизації, потужні інструменти рефакторинга коду, вбудовану підтримку технологій J2EE, механізми інтеграції з середовищем тестування Ant / JUnit і системами управління версіями, унікальний інструмент оптимізації та перевірки коду Code Inspection, а також інноваційний візуальний конструктор графічних інтерфейсів. Також, до складів продуктів компанії JetBrains входить зручний інструмент роботи з реляційними базами даних JetBrains DataGrid. Даний продукт представляє собою редактор таблиць баз даних, через який користувачу надається можливість створювати, переглядати та редагувати таблиці. DataGrip надає інструменти для роботи з об'єктами бази даних. При створенні або зміні таблиці, додаванні або зміні колонки, індексу, ключа в уже існуючі при цьому генерітся відповідний скрипт його можна відразу виконати в базі, а можна скопіювати згенерований DDL-запит в редактор і працювати вже безпосередньо з кодом.

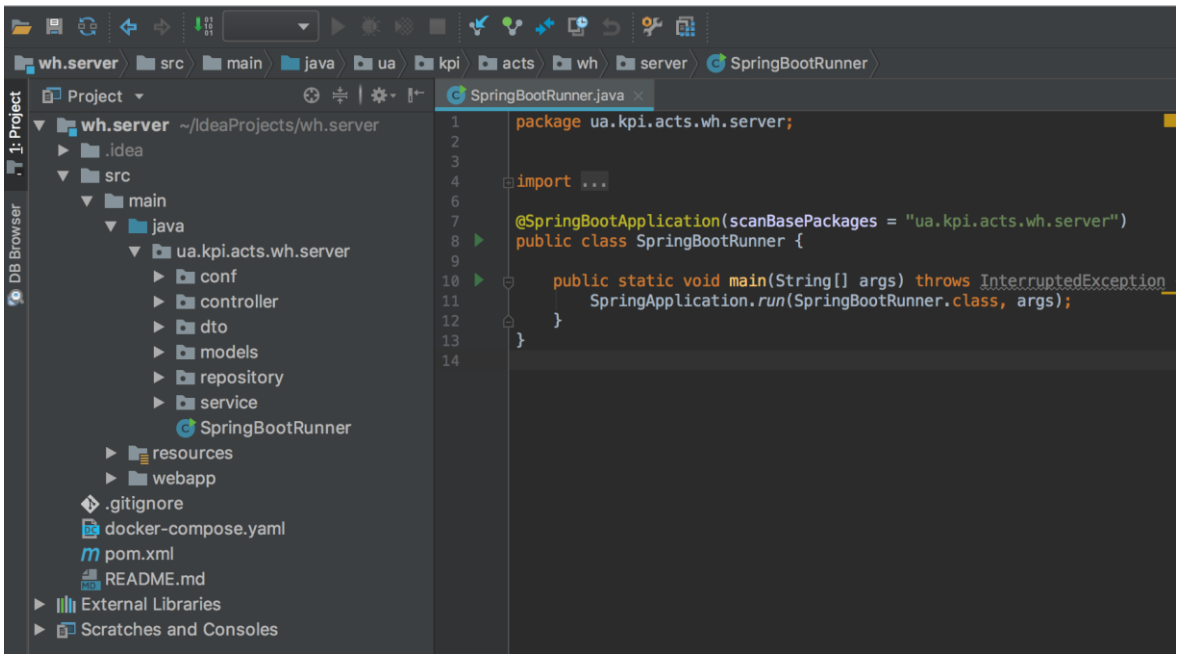

Рисунок 2.5 – Структура проекту у середовищі Intelliji Idea

Для імплементації прикладного програмного інтерфейсу (REST API) на мові програмування Java доцільно використовувати фреймворк Spring, а саме модулі Spring Core, Spring Data, Spring MVC, Spring Test (рис 2.5). Spring Framework забезпечує комплексну модель розробки і конфігурації для сучасних бізнес-додатків на Java - на будь-яких платформах (рис 2.6). Ключовий елемент Spring - підтримка інфраструктури на рівні програми: основна увага приділяється конфігуруванню бізнес-додатків, тому розробники можуть зосередитися на бізнес-логіці без зайвих налаштувань в залежності від середовища виконання.

Spring - це архітектурний фреймворк, тому що він придуманий не стільки для виконання якоїсь прикладної задачі (як, скажімо, Struts, який потрібен, щоб писати web-додатки), а для забезпечення кращої масштабованості, можливості більш простого тестування і більш простий інтеграції з іншими фреймворками (наприклад, вже згаданим Struts або Hibernate). Завдяки цьому писати великі програми стає простіше - розробники просто уникають ряду проблем, пов'язаних зі створенням enterprise-додатків, замість того, щоб їх вирішувати [20].

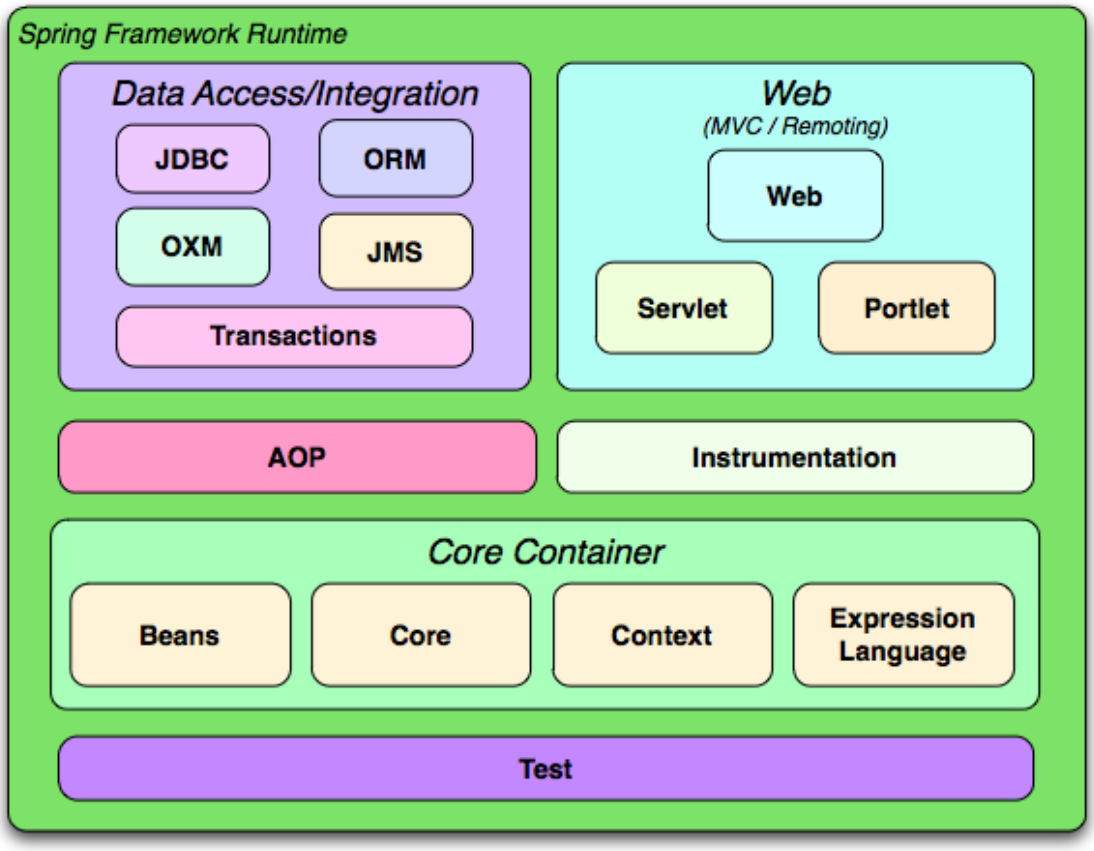

Рисунок 2.6 – Доступні модулі Spring Framework

Для передачі інформації між будь-якими компонентами системи, інформація подається у форматі JSON. Легкочитаємий і компактний, JSON являє собою гарну альтернативу XML і вимагає куди менше форматування контенту. Це інформативний посібник допоможе вам швидше розібратися з даними, які ви можете використовувати з JSON і основною структурою з синтаксисом цього ж формату.

Черга повідомлень для передачі даних між агентським та серверними додатками була реалізована завдяки програмному продукту RabbitMQ (рис 2.7). На відміну від ZeroMQ, який вбудовується в додатки, RabbitMQ - сервіс-посередник. Він розмежовує права доступу, підтримує шифрування, збереження повідомлень на диск (щоб пережити планове відключення електрики), роботу в кластерах і навіть дублювання сервісів для підвищення працездатності. До того ж він написаний на Erlang, за що автоматично стає невбиваним і підтримуваним на більшості популярних ОС.

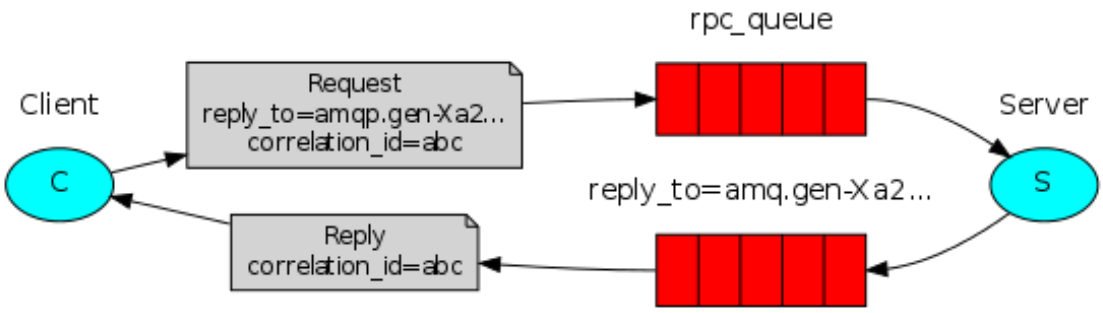

Рисунок 2.7 – Реалізація RPC за допомогою системи RabbitMQ

2.3.2 Реалізація серверного застосування

В якості протокола передачі данних між користувацьким інтерфейсом на серверним застосуванням був обраний HTTP, що є стандартом де-факто на ринку створення серверних застосувань, реалізуючих архітектурний підхід REST API.

Серверний застосунок обмінюється повідомленнями з програмою користувацького інтерфейса за допомогою прикладного програмного інтерфейсу, що імплементує архітектуру REST (REpresentational State Transfer). Дана архітектура описує принципи побудови розподілених гіпермедіа систем, того що іншими словами називається World Wide Web, включаючи універсальні способи обробки і передачі станів ресурсів по HTTP. Інформація про поточний стан агентів зберігається у БД [21].

2.3.3 Реалізація агентського застосування

Агент представляє собою Spring Boot застосунок, який постійно слухає нові повідомлення з черги RabbitMQ по назві унікального імені агента, виконує Docker команди та відповідає результатом їх виконання. Конфігурація агентського додатку включає в себе унікальну назву агента та адресу серверного застосування.

При першому старті агент реєструє себе у серверному додатку за допомогою посилання HTTP запиту, а при подальших запусках агента надсилається запрос на оновлення статусу, передаючи у тілі запроса значення “ACTIVE” по ключу “status”

об’єкта JSON. Дана техніка дозволяє агентським додаткам реєструвати себе самостійно, не перекладаючи цю відповідальність на основний веб-застосунок системи.

Агентське застосування не повинно навантажувати сервер, тому його було реалізованно у вигляді тонкого віддаленого клієнту до системи Docker, що запущена на сервері за який відповідає даний агент. Єдиною задачею агента є приймання запитів із Docker командами від серверного додатку, виконання цієї комнади за допомогою Docker клієнта та повернення результату у зручному вигляді.

Агентський застосунок надає можливість виконувати на сервері наступні Docker команди:

– PS – відобразити інформацію про стан запущених на сервері контейнерів;
– START/STOP – відновити роботу або зупинити роботу контейнера. Якщо контейнер було зупинено, то його в будь-який момент часу можна запустити, не втративши ніяких даних контейнера;
– STATS – вивести на екран статистику контейнера, включаючи імена всіх завантажених сервісів, імена зупинених сервісів, кількість оперативної пам'яті, кількість ядер, що виділені контейнеру;
– RM – видалити контейнер із усією його інформацією;
– RUN – запустити новий контейнер;
– LOGS – повернути записи журналювання сервісу в запущеному контейнері;
– EXEC – найбільш універсальна команда, яка дозволяє запустити будь-яку команду всередині операційної системи запущенного контейнера.

2.3.4 Імплементація користувацького інтерфейсу

Користувацький інтерфейс представляє собою веб-сторінки, ще реалізована за допомогою мови розмітки гіпертексту HTLM. Вбудований в сторінку код JavaScript надсилає запити на програмний інтерфейс REST серверного додатку. Веб-додаток отримує запит, обробляє його та повертає результат у вигляді JSON та відображає його на веб-сторінці (рис 2.8).

Легкочитаємий і компактний, JSON являє собою гарну альтернативу XML і вимагає куди менше форматування контенту. Це інформативний посібник допоможе вам швидше розібратися з даними, які ви можете використовувати з JSON і основною структурою з синтаксисом цього ж формату.

Для оформлення сторінок було обрано фреймворк Bootstrap - вільний набір інструментів для створення сайтів і веб-додатків. Включає в себе HTML- і CSS-шаблони оформлення для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення. Bootstrap включає в себе HTML і CSS-шаблони оформлення для типографіки, веб-форм, кнопок, міток, блоків навігації та інших компонентів веб-інтерфейсу, включаючи JavaScript-розширення.

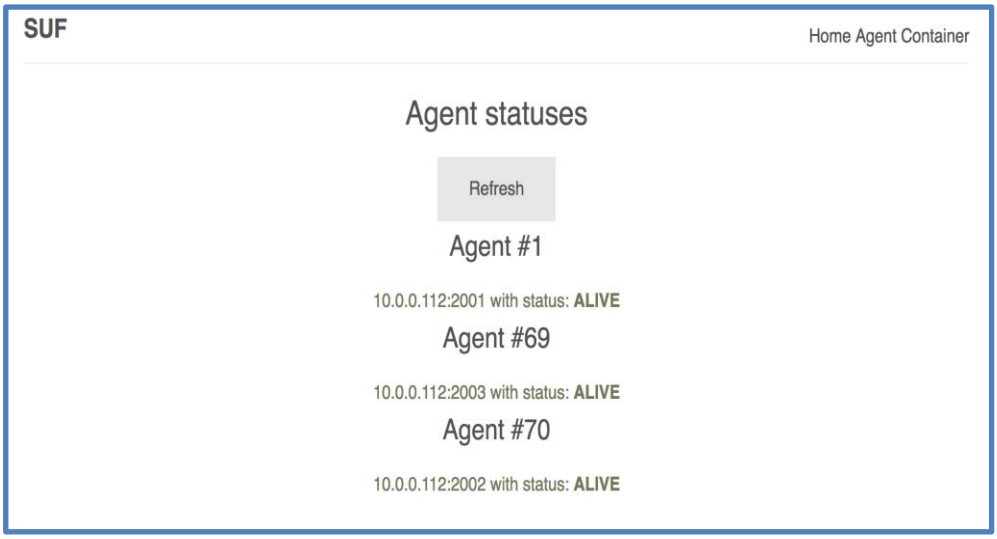

Рисунок 2.8 - Головна сторінка СУФ

Bootstrap використовує сучасні напрацювання в області CSS і HTML, тому необхідно бути уважним при підтримці старих браузерів, оскільки підтримка таких браузерів як Internet Explorer не є першим пріоритетом для більшості розробників сучасних фреймворків.

2.4 Розгортання системи управління в хмарній IT-інфраструктурі

Оскільки кожен сервіс корпоративної IT-інфраструктури виконується в своїй власній віртуальній операційній системі за використанням системи Docker, то образи таких сервісів можна з легкістю завантажувати на сервери корпоративної IT-інфраструктури за допомогою команд реєстру Docker образів (рис 2.9).

При рості кількості сервісів збільшується і кількість образів операційних систем, а отже у компаній постає питання про створення реєстру даних образів. Система віртуалізації Docker також надає користувачам можливість створювати власний приватний реєстр образів. Таким чином, з будь-якого сервісу можна зробити образ на одному з серверів корпоративної IT-інфраструктури.

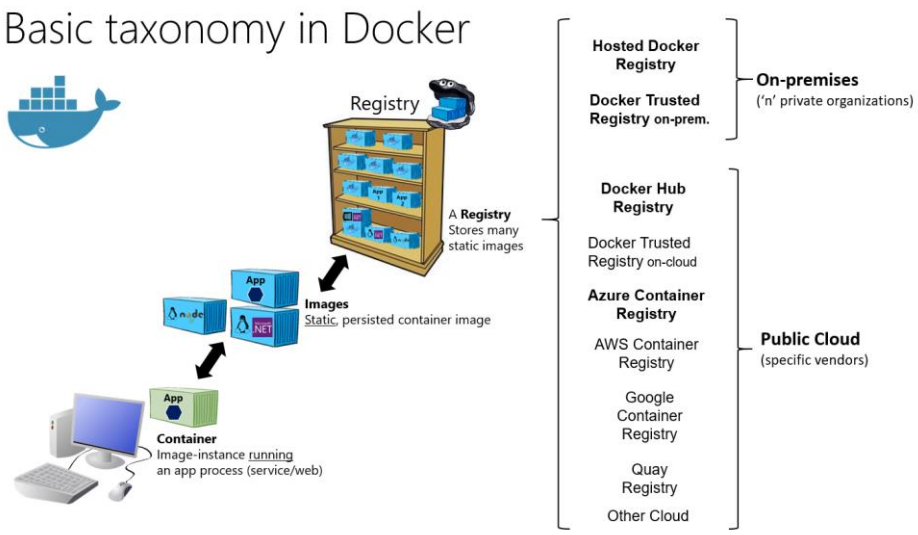

Рис 2.9 - Структура зберігання образів в реєстрі

Реєстр Docker образів доцільно розміщувати на окремому сервері, до яким неможливо управляти із зовнішньої мережі інтернет, тобто доступ повинен бути тільки у адміністраторів компанії із серверів, які підключені до корпоративної мережі.

Docker registry це звичайний docker контейнер, який виконує роль сховища. Це дає нам можливість підняти репозиторій для docker образів, який можна розмістити де завгодно, прив'язати доменне іменне, сертифікат, обмежити доступ до сервера і інші переваги. Звичайно, щоб доступ був звідки завгодно бажано на сервері (або комп'ютері) мати статичний адресу. Можна звичайно це все обійти, з'єднавши наприклад комп'ютери за допомогою VPN мереж.

В основі роботи Docker легітизований стандартизований здатність виконувати коди. Docker - це операційна система для контейнерів. Підтверджений тим, що віртуальна машина створює віртуальне представлення пристроїв безпеки пристроїв (тож потрібна необхідна можливість безпосередньо управляти таким), контейнери створюють віртуальне представлення серверної операційної системи. Після установки на кожен сервер Docker надає доступ до простих команд, необхідних для збору, використання або залишків контейнерів.

2.5 Висновки до розділу

В даному розділі була розроблена структурна схема системи управління функціональністю корпоративної IT-інфраструктури, що складається із агентських додатків, що встановлюються на кожен сервер, та веб-додатку управління ресурсами IT-інфраструктури. До задач агентських додатків можна віднести моніторинг запущених на сервері сервісів, управління їми. Веб-застосунок відповідальний за взаємодію с агентськими додатками шляхом отримання запитів від користувачів та пересилки команд до відповідного агента шляхом посилки повідомлення в чергу. Також, для зручності було створено користувацький веб-інтерфейс, через який доступні всі операції над контейнеризованими сервісами.

Для реалізації описаних додатків було обрано використовувати мову програмування Java та фреймворк для будування корпоративних рішень Spark, а саме модулі Spring Web, Spring Security, Spring Data. Дані інструменті є безкоштовними, зручними в використанні та універсальними. Агентські додатки взаємодіють із основним веб-додатком шляхом пересилки та отримання повідомлень до черги.

Для розгортання кожного с компонентів системи було обрано систему контейнеризації Docker, адже він задовольняє всі потреби уніфікації процесу побудови та доставки образу сервісу на сервіс корпоративної IT-інфраструктури. Система віртуалізації Docker також надає користувачам можливість створювати власний приватний реєстр образів, таким чином образ кожного з компонентів

створеної системи безпечно зберігається на сервері ІТ-інфраструктури компанії та не є доступним з зовнішньої мережі.

3 ПРОЕКТУВАННЯ КОРПОРАТИВНОГО СХОВИЩА ДАНИХ У ХМАРНОМУ СЕРЕДОВИЩІ

3.1 Вибір розподіленої файлової системи для зберігання корпоративних даних

При проектуванні корпоративного сховища даних у хмарному середовищі, першим постає питання про те, як саме дані будуть фізично зберігатися розподілено.

3.1.1 Hadoop Distributed File System

HDFS (Hadoop Distributed File System) - розподілена файлова система, яка використовується в проекті Apache Hadoop призначена для зберігання файлів великих розмірів, по блоках розподілених між вузлами обчислювального кластера. Всі блоки в HDFS (крім останнього блоку файлу) мають однаковий розмір, і кожен блок може бути розміщений на декількох вузлах, розмір блоку і коефіцієнт реплікації (кількість вузлів, на яких повинен бути розміщений кожен блок) визначаються в налаштуваннях на рівні файлу. Завдяки реплікації забезпечується стійкість розподіленої системи до відмов окремих вузлів.

Файли в HDFS можуть бути записані лише одного разу (модифікація не підтримується), а запис в файл в один час може вести тільки один процес. Організація файлів в просторі імен - традиційна ієрархічна: є кореневий каталог, підтримується вкладення каталогів, в одному каталозі можуть розташовуватися і файли, і інші каталоги.

Розгортання примірника HDFS передбачає наявність центрального вузла імен (англ. Name node), що зберігає метадані файлової системи і метаінформацію про розподіл блоків, і серії вузлів даних (англ. Data node), безпосередньо зберігають блоки файлів. Вузол імен відповідає за обробку операцій рівня файлів і каталогів - відкриття і закриття файлів, маніпуляція з каталогами, вузли даних безпосередньо відпрацьовують операції по запису і читання даних. Вузол імен та вузли даних забезпечуються веб-серверами, що відображають поточний статус вузлів і

дозволяють переглядати вміст файлової системи (рис 3.1). Адміністративні функції доступні з інтерфейсу командного рядка.

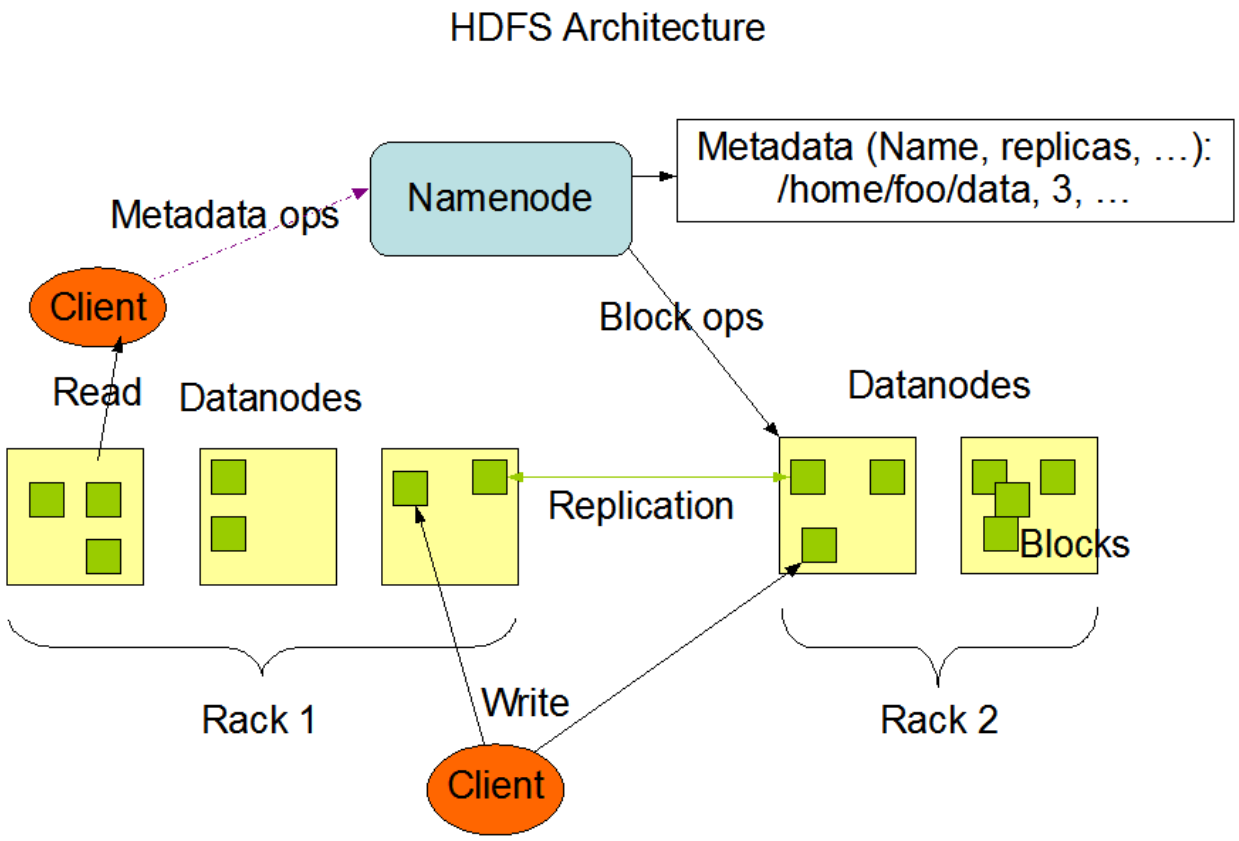

Рисунок 3.1 - Архітектура файлової системи Hadoop

HDFS призначена для підтримки дуже великих файлів. Додатки, сумісні з HDFS, - це програми, що працюють з великими наборами даних. Ці додатки записують свої дані тільки один раз, але читають їх один або кілька разів і вимагають, щоб ці читання виконувалися зі швидкістю потокової передачі. HDFS підтримує семантику «записав один раз - прочитав багато» для файлів. Типовий розмір блоку, який використовується HDFS, становить 64 МБ. Таким чином, файл HDFS розбивається на фрагменти по 64 МБ, і, якщо можливо, кожен блок буде знаходитися на різних вузлах DataNode.

Клієнтський запит на створення файлу не відразу досягає NameNode. Фактично спочатку клієнт HDFS кеширує дані файлу в тимчасовий локальний файл. Записи додатки прозоро перенаправляються в цей тимчасовий локальний файл. Коли локальний файл накопичує дані вартістю більш як один розмір блоку HDFS,

клієнт зв'язується з NameNode. NameNode вставляє ім'я файлу в ієрархію файлової системи і виділяє для нього блок даних. NameNode відповідає на запит клієнта ідентифікатором DataNode і цільовим блоком даних. Потім клієнт пересилає блок даних з локального тимчасового файлу в зазначений DataNode. Коли файл закритий, залишилися неперезапісанние дані в тимчасовому локальному файлі передаються в вузол даних. Потім клієнт повідомляє NameNode, що файл закритий. На цьому етапі NameNode фіксує операцію створення файлу в постійному сховищі. Якщо NameNode відключається до закриття файлу, файл втрачається.

Вищевказаний підхід був прийнятий після ретельного розгляду цільових програм, що працюють на HDFS. Ці додатки потребують потокової записи в файли. Якщо клієнт пише в віддалений файл безпосередньо, без будь-якої буферизації на стороні клієнта, швидкість мережі і перевантаження в мережі значно впливають на пропускну здатність. Цей підхід не без прецеденту. Раніше розподілені файлові системи, наприклад AFS використовує кешування на стороні клієнта для підвищення продуктивності. Вимога POSIX було пом'якшено для досягнення більш високої продуктивності завантаження даних.

Коли клієнт записує дані в файл HDFS, його дані спочатку записуються в локальний файл, як описано в попередньому розділі. Якщо файл HDFS має коефіцієнт реплікації три, то при накопиченні локальним файлом повного блоку призначених для користувача даних, клієнт отримує список вузлів даних з вузла імен. Цей список містить вузли даних, в яких буде розміщуватися репліка цього блоку. Потім клієнт скидає блок даних в перший DataNode. Перший DataNode починає отримувати дані невеликими порціями (4 КБ), записує кожну частину в свій локальний репозиторій і передає цю частину в другій DataNode в списку. Другий DataNode, в свою чергу, починає отримувати кожну частину блоку даних, записує цю частину в свій репозиторій і потім скидає цю частину в третій DataNode. Нарешті, третій DataNode записує дані в свій локальний репозиторій. Таким чином, DataNode може отримувати дані з попереднього в конвеєрі і в той же час пересилати дані наступного в конвеєрі. Таким чином, дані передаються від одного DataNode до наступного.

3.1.2 Вибір файлового формату зберігання даних

Загалом, для зберігання великих об'ємів даних виділяють 2 категорії файлових форматів: лінійні та колоночні (рис 3.2).

Row-oriented: rows stored sequentially in a file

Key	Fname	Lname	State	Zip	Phone	Age	Sales
1	Bugs	Bunny	NY	11217	(123) 938-3235	34	100
2	Yosemite	Sam	CA	95389	(234) 375-6572	52	500
3	Daffy	Duck	NY	10013	(345) 227-1810	35	200
4	Elmer	Fudd	CA	04578	(456) 882-7323	43	10
5	Witch	Hazel	CA	01970	(567) 744-0991	57	250

Column-oriented: each column is stored in a separate file
Each column for a given row is at the same offset.

Key
1
2
3
4
5

Fname
Bugs
Yosemite
Daffy
Elmer
Witch

Lname
Bunny
Sam
Duck
Fudd
Hazel

State
NY
CA
NY
CA
CA

Zip
11217
95389
10013
04578
01970

Phone
(123) 938-3235
(234) 375-6572
(345) 227-1810
(456) 882-7323
(567) 744-0991

Age
34
52
35
43
57

Sales
100
500
200
10
250

Рисунок 3.2 - Категорії файлових форматів

В лінійних форматах (AVRO, Sequence) рядки даних одного типу зберігаються разом, утворюючи безперервне сховище. Навіть якщо необхідно отримати лише деякі значення з рядка, все одно вся рядок буде зчитана з диска в пам'ять. Лінійний спосіб зберігання даних обумовлює знижену швидкість операцій читання і виконання виборчих запитів, а також більша витрата дискового простору. Лінійні формати, на відміну від колоночного, не є строго типізований (рис. 3.3). Наприклад, Apache AVRO зберігає інформація про тип кожного поля в розділі метаданих разом зі схемою, тому для читання серіалізовать даних інформації не потрібно попереднє знання схеми. Бінарний формат файлів послідовностей (Sequence File) для зберігання Big Data у вигляді серіалізовать пар ключ / значення в екосистемі Apache Hadoop також містить метадані в заголовку файлу.

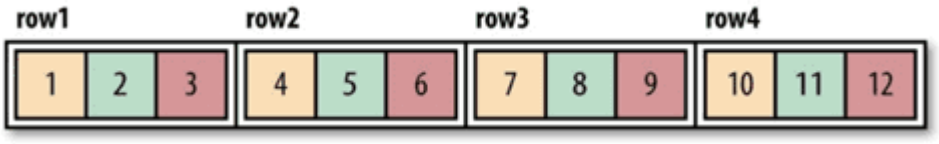

Рисунок 3.3 - Лінійний файловий формат

Формат Sequence File відмінно забезпечує паралелізм при виконанні задач фреймворку MapReduce, тому що різні порції одного файлу можуть бути розархівовані і використані незалежно один від одного. Проте, ступінь стиснення інформації у строкових форматів нижче, ніж у столбцових. Однак, саме лінійно-орієнтовані формати найкраще підходять для потокової записи, тому що в разі збою інформація може бути відновлена (повторно синхронізована) з останньої точки синхронізації.

У стовпчик-орієнтованих форматах (Parquet, RCFile, ORCFile) файл розрізається на декілька стовпців даних, які зберігаються разом, але можуть бути оброблені незалежно один від одного (рис 3.4). Такий метод зберігання інформації дозволяє пропускати непотрібні стовпці при читанні даних, що істотно прискорює читання даних і відмінно підходить в разі, коли необхідний невеликий обсяг рядків або виконуються виборчі запити, як, наприклад, в СУБД Apache Hive. Але такий формат читання і запису займає більше місця в оперативній пам'яті, оскільки, щоб отримати стовпець з кількох рядків, кешується кожен рядок.

Також стовпчиково-орієнтовані формати не використовуються в середовищах потокової обробки (Apache Kafka, Flume), тому що після збою записи поточний файл не може бути відновлений через відсутність точок синхронізації. Однак, стовпчик файли займають менше місця на жорсткому диску внаслідок більш ефективного стиснення інформації за стовпцями.

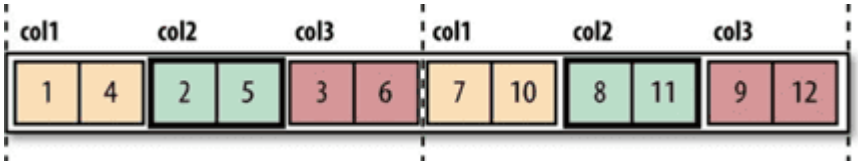

Рисунок 3.4 - Стовпчиковий файловий формат

Отже, вищеописані архітектурні відмінності колоночного і лінійних форматів Big Data файлів обумовлюють їх різну ефективність при зберіганні, читанні і запису інформації. Тому примітивне порівняння швидкості обробки інформації в тому чи іншому форматі щодо інших не зовсім коректно, тому що оцінюються не тільки і не

стільки формати, скільки алгоритмічне якість використовують їх прикладних систем.

Sequence File (файл послідовностей) - це двійковий формат для зберігання Big Data у вигляді серіалізовать пар ключ / значення в екосистемі Apache Hadoop, що дозволяє розбивати файл на ділянки (порції) при стисненні. Це забезпечує паралелізм при виконанні завдань MapReduce, тому що різні порції одного файлу можуть бути розпаковані і використані незалежно один від одного. Поряд з Apache Avro, Sequence File вважається лінійно-орієнтованим (строковим) форматом Big Data, на відміну від колоночного (столбцових) форматів (RCFile, Apache ORC і Parquet).

Відзначимо, що кожен постачальник Big Data продукту, перш за все, прагне просувати свої власні формати даних і ті, які найбільше підходять для використання з його рішенням, наприклад, сертифікаційний іспит Data Engineer по дистрибутива Apache Hadoop від компанії Cloudera (CHD, Cloudera's Distribution of Hadoop) включає теми по Parquet, тоді як аналогічна сертифікація по Hortonworks Data Platform більше приділяє уваги формату ORC. Також Parquet широко використовується в Apache Impala, Drill і Big Data платформі MAPR. А у розробника комерційної версії Apache Kafka, корпорація Confluent, в пріоритеті формат AVRO.

Таким чином, при виборі формату даних, слід, перш за все, мати на увазі практичні завдання, які необхідно вирішити з його допомогою в рамках функціональних можливостей конкретної Big Data системи. Наприклад, лінійний формат AVRO забезпечує високу швидкість запису інформації, і тому відмінно підходить обробки потоків Big Data в Apache Kafka, Flume і корпоративних озерах даних (Data Lake), а також добре вирішує завдання повного читання всіх полів запису, що потрібно в ETL- сховищах і вітрин даних.

У свою чергу, стовпчиково-орієнтовані формати (Parquet, RCFile, ORCFile) швидше зчитати дані з файлу за рахунок пропуску непотрібних стовпців і займають менше місця на диску. Тому формати стовпчикового типу активно застосовуються в файлових сховищах і СУБД на основі Apache Hadoop, зокрема, Hive. В контексті застосування SQL-запитів варто відзначити деякі ключові особливості формату ORC в порівнянні з іншими стовпчиковими файлами:

– Індексація блоків кожного стовпчика, що прискорює операції введення-виведення;
– зчитування метаданих на рівні стовпця дозволяє оптимізувати SQL-запити;
– підтримка ACID-вимог до транзакцій (Atomicity - Атомарність, Consistency - Узгодженість, Isolation - Ізольованість, Durability - Довговічність);
– більш високий ступінь стиснення файлів економить місце на жорсткому диску.

А до унікальних переваг формату Apache AVRO відноситься читаємість людиною, адже JSON формат знайомий багатьом розробникам. Також до переваг відносять забезпечення сумісності та підтримка еволюції схем даних, коли зміни обробляються шляхом пропуску, додавання або модифікації окремих полів. Серед стовпчикових форматів файлів, ORC забезпечує найкраще стиснення даних.

Отже, порівняльний аналіз найбільш поширених файлових форматів Big Data, в черговий раз підтверджує, що найкращий ефект дає застосування саме того інструменту, який орієнтований на конкретне завдання.

3.1.3 Вибір способу структури сховища даних

В сучасних корпоративних сховищах даних виділяють два способи структурування даних - «зірка» та «сніжинка».

Схема типу «зірка» має централізоване сховище даних, яке зберігається в таблиці фактів (рис 3.5). Схема розбиває таблицю фактів на ряд денормалізованних таблиць вимірів. Таблиця фактів містить агреговані дані, які будуть використовуватися для складання звітів, а таблиця вимірювань описує збережені дані.

Денормалізовані проекції менш складні, тому що дані згруповані. Таблиця фактів використовує тільки одну посилання для приєднання до кожної таблиці вимірювань. Більш проста конструкція зіркоподібній схеми значно спрощує написання складних запитів.

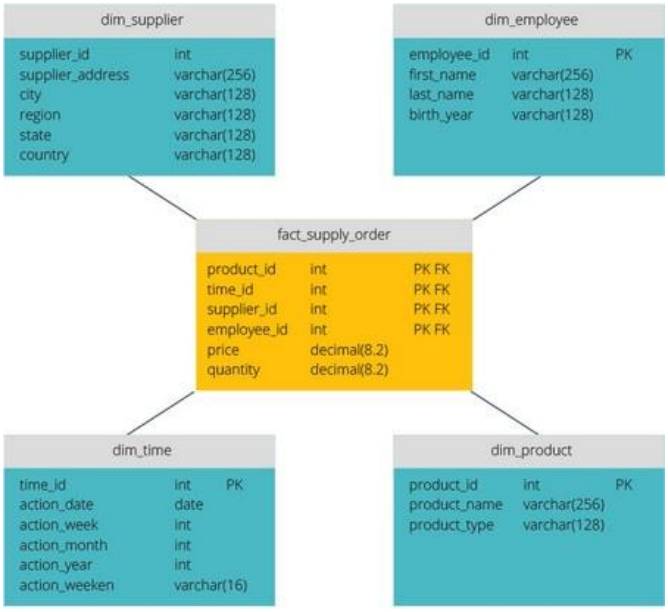

Рисунок 3.5 - Приклад схеми типу “Зірка”

Дані зберігаються у денормалізованому вигляді, оскільки операція join є дуже важкою для виконання на великих масивах даних у розподіленому середовищі. Таким чином, ми жертвуємо об'ємом збережених даних, проте операція зчитування відбувається набагато швидше, оскільки непотрібно виконувати операцію join, для якої в розподіленому середовище необхідне пересилання даних через мережу.

Схема типу «сніжинка» відрізняється тим, що використовує нормалізовані дані. Нормалізація означає ефективну організацію даних так, щоб всі залежності даних були визначені, і кожна таблиця містила мінімум надмірності (рис 3.6). Таким чином, окремі таблиці вимірювань розгалужуються на окремі таблиці вимірювань.

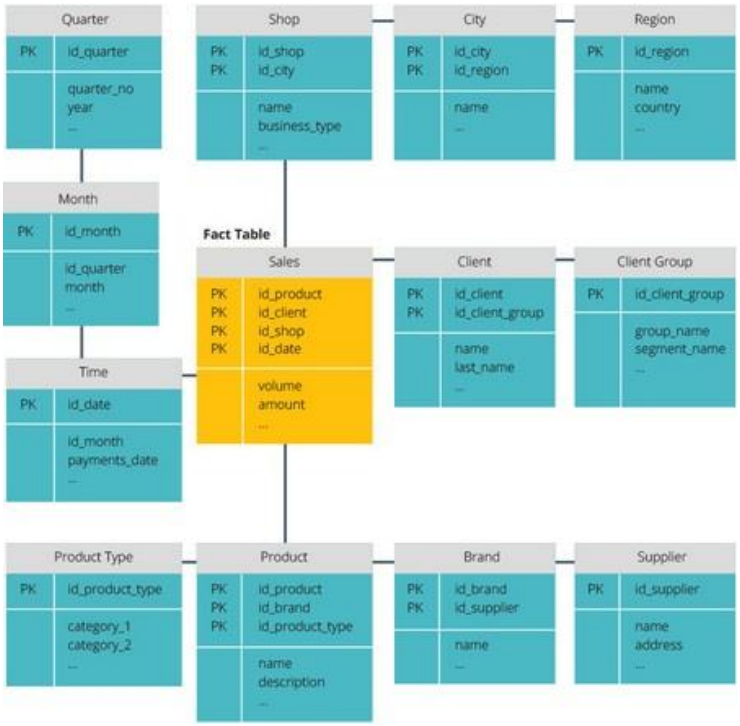

Рисунок 3.6 - Приклад схеми типу “Сніжинка”

Схема «сніжинки» використовує менше дискового простору і краще зберігає цілісність даних. Основним недоліком є складність запитів, необхідних для доступу до даних - кожен запит повинен пройти кілька з'єднань таблиць, щоб отримати відповідні дані, а операція join в розподіленому середовище призводить до пересилання велокого обєму даних через мережу. Як відомо, будь-яке пересилання даних через мережу відбувається в сотні раз повільніше, ніж обробка даних в пам'яті, та в десятки разів повільніше, ніж запис і зчитування даних з дискового простору.

3.2 Вибір інструментів для розподіленого аналізу даних

З ростом популярності використання технологій Big Data зростає і кількість інструментів для розподіленої обробки даних. До найбільш розповсюджених можна віднести такі інструменти як Spark, Presto, Flink, Storm.

Apache Storm - обчислювальне середовище на основі Open Source для роботи в реальному часі. Даний фреймворк легко інтегрується з іншими компонентами, має високі показники продуктивності і досить просто розгортається, створюючи розподілену систему RT-обчислень для швидкої обробки великих потоків даних,

додаючи можливості обробці в Apache Hadoop. Спочатку проект створений в BackType, потім куплений Twitter і в вересні 2013 року переведений в інкубатор Apache. Використання Storm в кластері Hadoop забезпечує ефективну обробку всього спектра робочих навантажень: від роботи в реальному часі до інтерактивної і обробки групами.

Зараз Storm розробляється по відкритій моделі, серед учасників співробітників компаній Hortonworks, Twitter, Verisign, Yahoo і інші. Тестування Apache Storm при обробці мільйона 100-байтових повідомлень в секунду на один вузол (node) показало відмовостійкість додатки, масштабованість між вузлами кластера і простоту роботи. Apache Storm підтримує інтеграцію з СУБД, розпаралелювання, поділ і повторні спроби при помилках, коли це необхідно.

Проект Apache Spark останнім часом привертає до себе величезну увагу, про нього написано велику кількість маленьких практичних статей, він став частиною Hadoop 2.0. Також, він швидко обріс додатковими фреймворками, такими, як Spark Streaming, SparkML, Spark SQL, GraphX, а крім цих «офіційних» фреймворків з'явилося море проектів - різні коннектори, алгоритми, бібліотеки і так далі. Spark це проект Apache, який позиціонується як інструмент для «блискавичних кластерних обчислень». Проект розробляється процвітаючим вільним спільнотою, зараз є найбільш активним з проектів Apache (рис 3.7). Spark надає швидку та універсальну платформу для обробки даних. У порівнянні з Hadoop Spark прискорює роботу програм в пам'яті більш ніж в 100 разів, а на диску - більш ніж в 10 разів.

Крім того, код на Spark пишеться швидше, оскільки тут у вашому розпорядженні буде більше 80 високорівневих операторів. При вивченні Apache Spark варто відзначити ще один важливий аспект: тут надається готова інтерактивна оболонка (REPL). За допомогою REPL можна протестувати результат виконання кожного рядка коду без необхідності спочатку програмувати і виконувати всі завдання цілком. Тому написати готовий код вдається набагато швидше, крім того, забезпечується ситуативний аналіз даних.

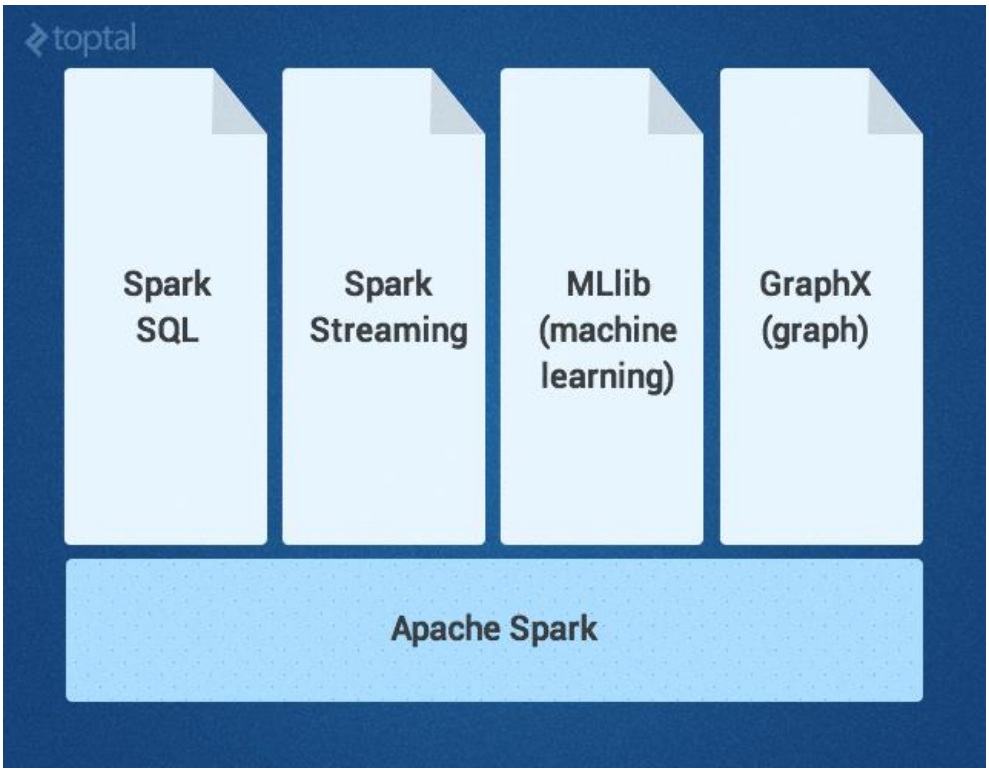

Рисунок 3.7 - Модулі фреймворку Apache Spark

Ядро Spark доповнюється набором потужних високорівневих бібліотек, які легко інтегруються з ним в рамках того ж додатка. В даний час до таких бібліотек відносяться SparkSQL, Spark Streaming, MLlib (для машинного навчання) і GraphX. Зараз також розробляються інші бібліотеки та розширення Spark.

Для інтерактивних запитів біль доцільно використовувати Presto - це відкритий джерело розподіленого SQL-запиту для запуску інтерактивних аналітичних запитів проти джерел даних будь-яких розмірів, починаючи від гігабайт до петабайт.

Presto був розроблений і написаний з нуля для інтерактивної аналітики і наближається до швидкості зберігання комерційних сховищ даних, одночасно масштабуючи розміри таких організацій, як Facebook.

Проект Presto, що належить громаді, підтримується Apache Foundation, незалежною некомерційною організацією з відкритим та нейтральним управлінням, що розміщується в рамках Linux Foundation.

Presto дозволяє запитувати дані, де він живе, включаючи Hive, Cassandra, реляційні бази даних або навіть власні сховища даних. Один запит Presto може поєднувати дані з декількох джерел, що забезпечує аналітику для всієї організації.

Presto орієнтований на аналітиків, які очікують часу реакції в межах від другої секунди до хвилин. Presto порушує помилковий вибір між швидкою аналітикою з використанням дорогого комерційного рішення або використанням повільного "безкоштовного" рішення, яке вимагає надмірного обладнання.

Facebook використовує Presto для інтерактивних запитів проти декількох внутрішніх сховищ даних, включаючи їх сховище даних 300PB. Понад 1000 співробітників Facebook щодня використовують Presto для запуску понад 30 000 запитів, які в сукупності сканують петабайт кожен день.

Хоча комерційної підтримки Apache Flink ще належить набрати критичну масу, цей інструмент обіцяє заповнити пробіл, залишений іншими потоковими двигунами з відкритим вихідним кодом - додати в застосунок потокової обробки даних можливість повторів і відкатів.

Майже п'ять років тому інструменти потокової обробки даних почали ставати все більш поширеними, тому що «той же вибух росту даних, який зумовив актуальність великих даних, породжує і попит на перетворення даних в основу для негайних дій». А на початку поточного року було задекларовано, що «Інтернет речей (IoT) представляє ту область застосування, яка проштовхне на передній план передачу потоків в реальному часі».

Flink, на відміну від потокових движків з відкритим вихідним кодом Apex, Storm або Heron, здійснює не тільки передачу потоків. Він більше нагадує Apache Spark навпаки. В обох випадках один і той же движок здійснює передачу як в реальному часі, так і пакетами, позбавляючи від необхідності в архітектурі Lambda. У обох є API-інтерфейси для запиту даних з таблиць і API-інтерфейси або бібліотеки для обробки в реальному часі і пакетної обробки, а також є графи і машинне навчання.

Flink є обчислювальним інструментом, створеним для передачі потоків і розширений для передачі пакетів, тоді як Spark був створений для передачі пакетів і власної версії - мікропакетов (ця ситуація, в кінці кінців, зміниться). Певною мірою ця метафора може бути застосована до Storm (з його API-інтерфейсом Trident), але у нього відсутні бібліотеки і підтримка таких найважливіших функцій, як строго однократна обробка подій (вона, наприклад, гарантує, що подія буде

оброблено лише одного разу), а також можуть бути встановлені обмеження щодо масштабування.

Хоча прихильники Flink стверджують, що це один з п'яти головних проектів Apache Big Data, у спільноти розробників Spark є прихильники, діяльність яких спрямована на те, щоб цей проект був визнаний лідером ринку. Крім того, Spark отримав підтримку комерційних виробників. Він підтримується практично всіма збірками Hadoop і основними хмарними провайдерами. Зростає і список наявних в Spark інструментів аналізу та підготовки даних.

Flink націлюється на додатки з запам'ятовуванням станів, оскільки на відміну від інших потокових движків з відкритим вихідним кодом він підтримує фіксацію часу подій. Це означає, що програми, що працюють з каналами передачі даних, можуть виробляти відкат і повтор.

При цьому Flink націлюється на функціонал, який зазвичай резервується за СУБД - підтримку додатків із запам'ятовуванням станів (рис 3.8). Додатки, що розміщуються на Flink як мікросервісов, будуть керувати станом. Для працюючих в реальному часі додатків відмова від характерних для СУБД операцій введення-виведення скорочує затримки і накладні витрати. Ідея не такі вже й нова. В n-рівневий додатках станами часто управляло написане на Java ПО проміжного шару.

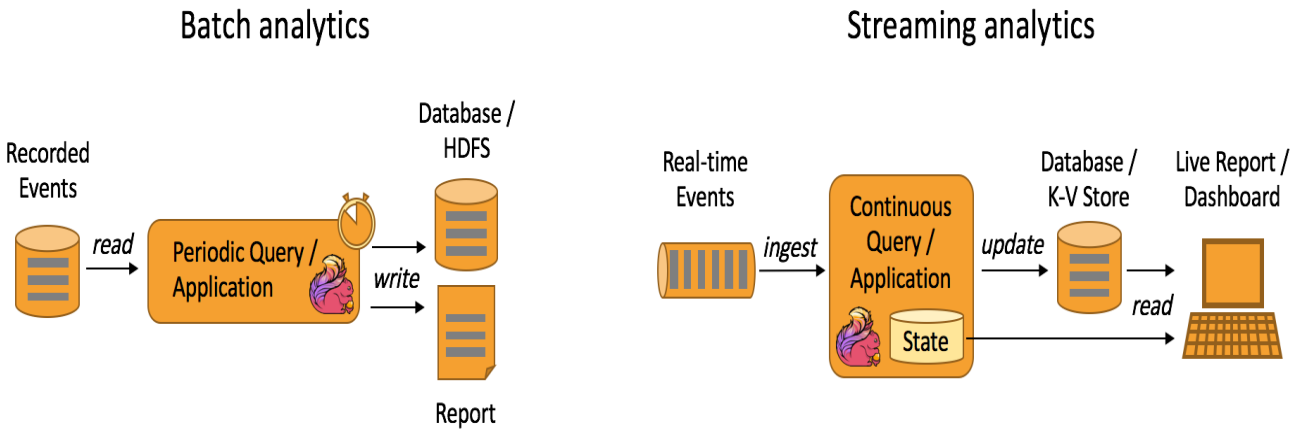

Рисунок 3.8 - Види аналітичних систем

В епоху застосування швидких СУБД, що використовують зберігання в оперативній пам'яті і на флеш-масивах, залишається ряд відомих областей обробки

в реальному часі, в яких як і раніше може мати значення винесення збереження станів за межі СУБД.

Прикладами можуть служити управління мережами IoT, відображення актуального стану угод, що укладаються за допомогою електронної комерції, видача рекомендацій в реальному часі, резервування квитків і управління підключеними до Інтернету автомобілями. Це не означає, що Flink замінить СУБД, оскільки дані (або витримка з них) можуть, в кінцевому підсумку, зберігатися. Різниця в тому, що ключова операція обробки деяких транзакцій реального часу (але не тих, які вимагають повної підтримки).

3.3 Інтерфейси доступу до корпоративних даних

В розробленому корпоративному сховищі даних дані зберігаються в форматі Parquet, тобто в спеціальному форматі даних, адаптованому для аналітичних запитів. Самі по собі файли на розподіленій файловій системі Hadoop Distributed File System не є корисними, проте завдяки інструментам, описаним у минулому розділі, з'являється можливість виконувати аналітичні запити до даних.

Слід зазначити, що в сучасних системах аналізу виділяють запити, результат яких очікується якомога швидше (так звані, ad hoc запити) та запити, результат яких не є критичним до отримання в момент виконання. Запити першого типу, зазвичай, не є складними, часто потребують дані тільки з однієї таблиці, результатом виконання може бути декілька рядків згрупованих даних. Зазвичай, не критичні запити набагато складніші, в них можуть бути присутні елементи бізнес-логіки, взаємодія з зовнішніми сховищами даних, інше. Результатом виконання такого запиту може бути дуже багато рядків, тобто такі запити більш схожі на звіти. Для даних двох видів запитів доцільно використовувати різні інструменти, адже для кожного типу існують свої обмеження. Розглянемо детальніше кожен із видів запитів.

Прикладом ad hoc запитів є аналіз значень даних, запити на унікальні значення тієї чи іншої колонки, перевірка конкретного рядка (запит по уникальному ідентифікатору) та інше. Дана задача досить легко може бути представлена у

вигляді SQL запиту до однієї таблиці. Серед наведених вище розподілених інструментів аналізу даних для такої задачі найбільше адаптований фреймворк Presto.

Для користувача (наприклад, бізнес-аналітика) дані, що зберігаються в спеціальному форматі на розподіленій файловій системі, представлені у вигляді таблиць, до котрих через клієнтську бібліотеку можна посилати звичайні SQL запити.

```
[hadoop@ip-10-167-87-171 ~]$ presto-cli --catalog hive --schema default
presto:default> SELECT language, page_title, AVG(hits) AS avg_hits
             -> FROM default.wikistats
             -> WHERE language = 'en'
             -> AND page_title NOT IN ('Main_Page',  '404_error/')
             -> AND page_title NOT LIKE '%Special%'
             -> AND page_title NOT LIKE '%index%'
             -> AND page_title NOT LIKE '%Search%'
             -> AND NOT regexp_like(page_title, '\%20')
             -> GROUP BY language, page_title
             -> ORDER BY avg_hits DESC
             -> LIMIT 10;
 language |         page_title          |      avg_hits
----------+-----------------------------+--------------------
 en       | Usher_feat._Young_Jeezy     |            20966.1
 en       | Ghetto_84                   |             5641.0
 en       | 14_Nitrous_Oxide            |  4945.666666666667
 en       | Wiki                        |             4532.0
 en       | Blue_moon                   |             4502.9
 en       | Musume_Morning              |             4259.0
 en       | Dick_Clark                  |  4139.533333333334
 en       | File:0670041882.jpg         |             4058.0
 en       | Sergio_Fiorentino_Pianist   |             3877.0
 en       | Albinoni,_Tomaso_Giovanni   | 3499.3333333333335
(10 rows)

Query 20150929_211856_00031_8qniv, FINISHED, 9 nodes
Splits: 177 total, 177 done (100.00%)
0:32 [128M rows, 5.64GB] [4.02M rows/s, 181MB/s]
```

Рисунок 3.9 - Клієнт командний рядок Presto

Таким чином користувачі повністю абстраговані від усієї складності розподіленої системи і можуть працювати з даними, ніби вони збережені у звичайній реляційній базі даних. Для підключення до Presto користувачу необхідно завантажити клієнт для командного рядка (рис 3.9) і вказати адресу і порт координатора, після чого користувачам надається можливість виконувати звичні SQL запити.

Для більш складних запитів та звітів описаний вище інструмент не зовсім підходить, адже результат виконання запиту на генерування звіту може бути досить великим, його не буде зручно переглядати у консольному клієнті. Також, слід зазначити що запити на генерування звітів, зазвичай, потребують набагато більше ресурсів та часу, адже часто необхідні звіти генеруються на даних за великий

проміжок часу, наприклад, квартальні звіти, річні, тощо. Наостанок, складні звіти досить важко описати стандартною мовою програмування SQL. Ця задача не є неможливою, проте при зростанні складності звіту, зростає і кількість рядків самого запиту SQL, його стає важче читати та тестувати.

Серед описаних в підрозділі 3.1.3 розподілених інструментів аналізу великих масивів даних найбільш доцільно використовувати фреймворк розподілених обчислень Apache Spark.

Для генерування звітів при використанні фреймворку Apache Spark достатньо написати невелику программу на мові програмування Python, Scala або Java, після чого необхідно вказати адресу менеджера ресурсів YARN та саме запустити програму.

Програма, написана за допомогою фремворку Spark, складається з коду, який виконується на програмі-драйвері та коду, що виконується на розподілених програмах-виконавцях (рис 3.10). Весь програмний представляється у вигляді направленого ациклічного графу виконання, після чого Spark автоматично виконує ряд оптимізацій. Прикладами такої оптимізації є фільтрація даних на початку виконання, а не в місці реального виклику операції фільтрування, передача предикату фільтрування на рівень файлів, інше.

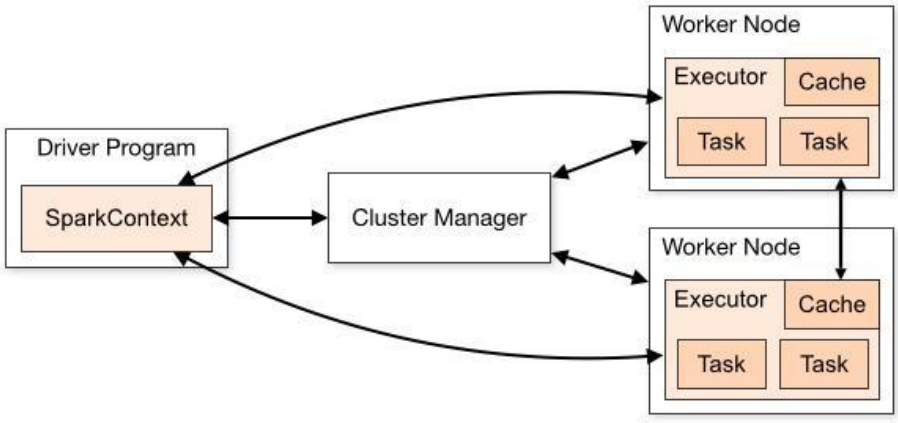

Рисунок 3.10 - Схема виконання програм Spark

Програмний застосунок розкладається на один або більше етапів, кожен з яких буде виконаний локально на одному з серверів. Між різними етапами проходить операція передачі даних по мережі, тобто програміст зацікавлений в

зменшенні кількості етапів виконання, адже передача даних по мережі є дуже коштовною операцію, на відміну від обробки даних в оперативній пам'яті. Кожна програма на Spark завершується записом результату в сховище, наприклад, в корпоративне сховище даних, реляційну базу даних, інше.

Операції в Spark поділяються на два типи - трансформації та дії. Прикладом трансформації може виступати операція перетворення даних з одного виду в інший, операція фільтрування. Дією може виступати запис результату на дисковий простір, в базу даних, в сховище даних.

При використанні інструменту Spark одним з важливих етапів є моніторинг виконання завдань, адже при обробці великих масивів даних програма може працювати годинами, тож в будь-який момент програміст повинен мати можливість відслідкувати прогрес виконання, проаналізувати обсяг даних, що обробляється. Однією з переваг даного фреймворку є те, що його постачальник також надає доступ до користувацького інтерфейсу моніторингу виконання програм (рис 3.11), де програміст може отримати повну картину того, що відбувається на кластері програм-виконавців.

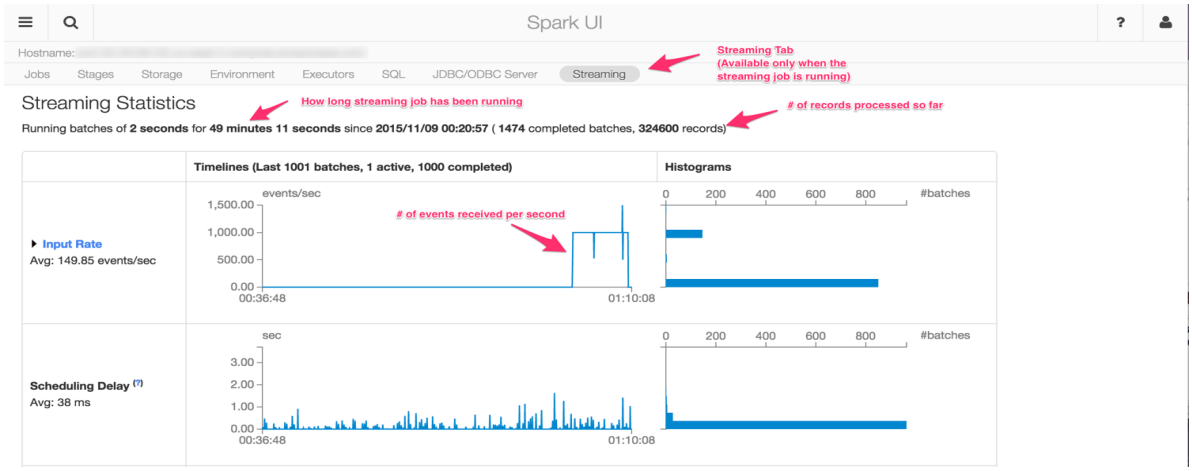

Рисунок 3.11 – Користувацький інтерфейс Spark

Програми на Spark виконуються “ліниво”, тобто код починаю виконуватися не одразу, а тільки після того, як інструмент проаналізує написаний код, зробить усі оптимізації. Такий спосіб виконання також дає змогу перезапустити весь код обробки для невеликої частини даних. Описана техніка є важливим елементом

виконання, оскільки розподілені обчислення мають високий ризик того, що обладнання одного з серверів вийде зі строю і необхідно буде проаналізувати на якому етапі виконання програма завершилась та перезапустити виконання не для всіх даних, а тільки для невеликої частини, заощадживши при цьому багато часу. Дана концепція є дуже важливою в розподіленому середовищі, адже основним способом оптимізувати час виконання розподілених програм є зменшення кількості даних, що пересилаються мережею.

Apache Spark підтримує всі перераховані вище формати даних (AVRO, Sequence, Parquet, ORC, RCFile), але найбільш оптимально працює з колоночного файлами в режимі онлайн-аналітики та із строковими при обробці інформаційних потоків. Проте, розробники Spark також створили можливість маніпулювати даними у всіх описаних форматах.

3.4 Висновки до розділу

У даному розділі було спроектовано корпоративне сховище даних, що складається з розподіленої файлової системи та інструментів розподіленого обчислення, що оперують даними, збереженими на ній. В якості розподіленої файлової системи було обрано продукт Hadoop Distributed File System, який зарекомендував себе як надійну і захищену файлову систему, що здатна масштабуватися до десятків тисяч серверів без втрати якості виконання.

Також, в розділі було проаналізовано та обрано сучасні розподілені інструменти обробки даних. Були описані інструменти обробки потокових даних, інструменти, що здатні виконувати запити SQL на розподіленій файловій системі Hadoop Distributed File System та універсальні фреймворки розподілених обчислень.

Для виконання цілеспрямованих запистів було обрано фреймворк Presto, завдяки якому користувачі можуть швидко та легко маніпулювати даними за допомогою мови побудови запитів SQL. Даний інструмент дозволяє повністю абстрагуватися від складнощів розподілених обчислень та сконцентрувати увагу на написанні скриптів маніпулювання даними. При використанні даного інструменту складається враження, що запити виконуються на сервері класичної реляційній базі

даних, що значно зменшує час на навчання нових співробітників, адже мова виконання запитів SQL є стандартом де факто вже багато років і є знайомою для багатьох бізнес-аналітиків та розробників програмного забезпечення.

В систему також було інтегровано модуль генерування аналітичних звітів, в який можна швидко додавати програми написані за допомогою фреймворку Spark. Користувачам надається можливість налаштовувати час та дату, у які необхідно запустити програму звіту а також вказати адресу електронної пошти, куди згенерований звіт необхідно надіслати.

4 РОЗРОБКА СИСТЕМИ АНАЛІЗУ ЯКОСТІ СЕРВІСІВ КОРПОРАТИВНОЇ ІТ-ІНФРАСТРУКТУРИ

4.1 Розробка підсистеми моніторингу та збору показників якості сервісів

Першим етапом розробки системи аналізу показників якості сервісів є створення модулю їх моніторингу та збереження. Основною задачею даного модулю є постійне опитування черги повідомлень на предмет нових подій, їх завантаження та збереження в корпоративному сховищі даних. Показники можна розглядати як нескінченний потік подій, тому черга повідомлень в даному випадку є доцільним рішенням. Сервісів корпоративної інфраструктури може бути досить багато, тому модуль моніторингу також повинен бути розподілений та масштабуватися горизонтально Також, для забезпечення відмовостійкості черга повідомлень не повинна видаляти повідомлення після їх прочитання. Слід зазначити, що показники якості розглядаються як будь-які інші дані, тому їх також можна зберігати в корпоративному сховищі даних та аналізувати. Всі описані вище вимоги імплементовані в продукті Apache Kafka (рис 4.1). Даний програмний комплекс є диспетчером повідомлень на Java платформі.

Рисунок 4.1 - Структура компонентів Kafka

Одним з ключових понять Kafka є тема повідомлення в яку видавці пишуть повідомлення і є слухачі в темах, які читають ці повідомлення. Задля забезпечення відмовостійкості все події в процесі диспетчеризації пишуться на диск, незалежно від того, чи є вони прочитані підписниками на тему.

До складу Kafka входять набір утиліт по створенню тим, розділів, готові видавці, слухачі для прикладів і ін. Для роботи Kafka необхідний координатор

«ZooKeeper», тому спочатку необхідно встановити ZooKeeper (zkServer.cmd) потім сервер Kafka (kafka-server-start.bat), командні файли знаходяться в відповідних папках bin.

Для простого тестування можна використовувати що входять до складу готові програми «kafka-console-consumer» і «kafka-console-producer» (рис 4.2). Слухачі на практиці об'єднують в групи, це дозволить різним програмам читати повідомлення з теми паралельно.

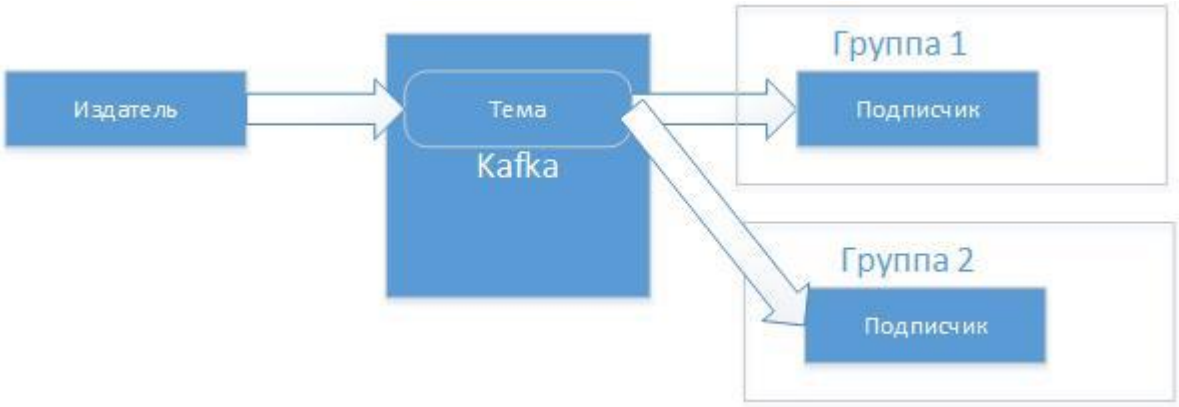

Рисунок 4.2 - Отримання повідомлень групою підписників

Для кожної програми буде організована своя чергу, читаючи з якої виконується переміщення покажчика останнього прочитаного повідомлення (offset), це називається фіксацією (commit) читання. І так якщо видавець відправить повідомлення в тему, то воно буде гарантовано прочитано одержувачем цієї теми якщо він запущений або, як тільки він підключиться. Причому якщо є різні клієнти (client.id), які читають з однієї теми, але в різних групах, то повідомлення вони отримають незалежно один від одного і в той час, коли будуть готові. Так можна уявити послідовник повідомлень і незалежне читання їх споживачами з однієї теми (рис 4.3).

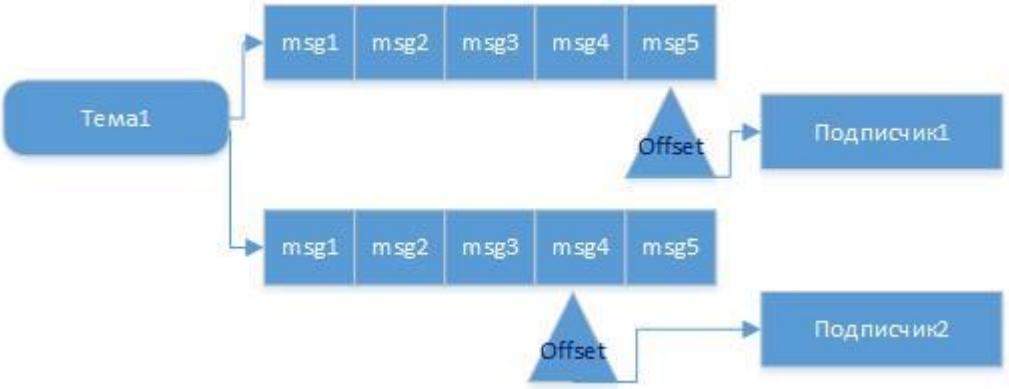

Рисунок 4.3 - Зберігання зміщення повідомлень

Але є ситуація, коли повідомлення в тему можуть почати надходити швидше ніж йти, тобто споживачі обробляють їх довше. Для цього в темі можна передбачити розділи (partitions) і запускати споживачів в одній групі для цієї теми.

Тоді відбудеться розподіл навантаження і не всі повідомлення в темі і групі підуть через одного споживача. І тоді вже буде обрана стратегія, як розподіляти повідомлення по розділах. Є кілька стратегій: round-robin - це по колу, по хеш значенням ключа, або явне вказівку номера розділу куди писати. Отримувачі в цьому випадку розподіляються рівномірно по розділах. Якщо, наприклад, підписників буде в групі буде більше ніж розділів, то хтось не отримає повідомлення. Таким чином розділи робляться для поліпшення масштабованості.

Отже, потоки в Kafka це послідовність подій, які отримують з теми, над якою можна виконувати певні операції, трансформації і потім результат віддати далі, наприклад, в іншу тему або зберегти в БД, в загальному куди завгодно. Операції можуть бути як наприклад фільтрації (filter), перетворення (map), так і агрегації (count, sum, avg). Для цього є відповідні класи KStream, KTable, де KTable можна уявити як таблицю з поточними агрегованими значеннями які постійно оновлюються в міру надходження нових повідомлень в тему.

Видавець пише в тему повідомлення (рис 4.4), Kafka всі повідомлення зберігає в журналі повідомлень на дисковому просторі, який має політику зберігання (Retention Policy), наприклад 7 днів. Наприклад події зміни котирування це потік, далі необхідно дізнатися середнє значення, а отже доцільно створити

Stream який візьме історію з журналу і вважатиме середнє, де ключем буде акція, а значенням - середнє (це вже таблиця зі станом).

Основною особливістю операції агрегування, на відміну від операцій, наприклад, фільтрації, є збереження стану. Тому знову надходять повідомлення (події) в тему, будуть схильні до вирахування, а результат буде зберігатися (state store), далі знову вступники писатися в журнал, Stream їх буде обробляти, додавати зміни до вже збереженого стану. Наприклад, операції фільтрації та зміни формату даних не вимагають збереження стану. В даному випадку stream буде зберігати стан на розподіленій файловій системі незалежно від видавця, оскільки цього вимагає сам фреймворк.

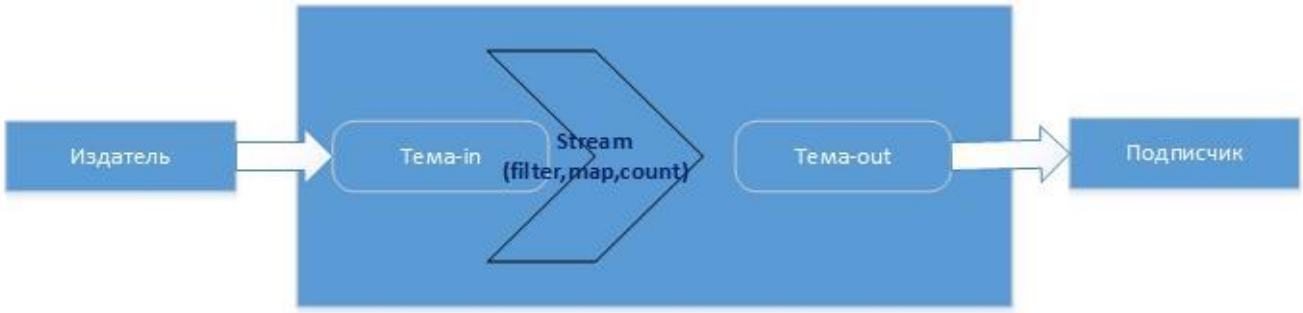

Рисунок 4.4 - Структура Kafka Streams

Наприклад, видавець пише повідомлення, а програма - stream в цей час не працює, нічого не пропаде, все повідомлення будуть збережені в журналі і як тільки програма-stream стане активною, вона зробить обчислення, збереже стан, виконає зміщення для прочитаних повідомлень (позначить що вони прочитані) і в подальшому вона вже до них не повернеться, більш того ці повідомлення підуть з журналу (kafka-logs). Додаток Stream буде читати цю тему вважати кількість однакових слів, які не явно для нас зберігати стан і перенаправляти в іншу тему out-topic. Тому слід приділити наголошенню на зв'язок журналу і стану (state store).

4.2 Реалізація алгоритмів прогнозування вартості IT-інфраструктури

Розроблена система може бути використана для побудування більшості сучасних аналітичних моделей на алгоритмів машинного навчання. Наприклад,

завдяки зберіганню показників якості сервісів у корпоративному сховищі даних можна створити програму, що прогнозує кількість ресурсів, що буде використана всіми сервісами інфраструктури. Модуль управління якістю сервісів підтримує кожен сервіс в працездатному стані шляхом своєчасного збільшення обчислювальних ресурсів, що надані сервісу, тому загальна кількість ресурсів не є статичною і може змінюватись протягом дня. Таким чином, спершу слід розрахувати загальну кількість ресурсів у кожен момент часу на історичних даних, після чого побудувати аналітичну модель прогнозування даного показника на майбутнє. Отримавши значення кількості обчислювальних ресурсів у наступний проміжок часу, можна з легкістю порівняти його із загальною кількістю зарезервованих ресурсів та отримати рішення щодо необхідності реєстрування нових серверів задля забезпечення ресурсами усіх запущених сервісів.

Серед модулей аналізу часових рядів за допомогою python було обрано statsmodels. Даний модуль надає широкий набір засобів і методів для проведення статистичного аналізу та економетрики. До основних етапів аналізу рядів включено будування моделі ARIMA. Для початку, необхідно завантажити дані показників якості (рис 4.5).

```
1  from pandas import read_csv
2
3  dataset = read_csv('history_resources.csv', ';',
4                     index_col=['date_oper'],
5                     parse_dates=['date_oper'],
6                     dayfirst=True)
7  dataset.head()
8
```

Рисунок 4.5 - Завантаження даних показників якості

Функція read_csv, в даному випадку, крім вказання параметрів, які задають використовувані колонки і індекс, можна помітити ще 3 параметра для роботи з датою. Функція parse_dates задає імена стовпців, які будуть перетворені в тип DateTime. Варто відзначити, що якщо в даному стовпці будуть порожні значення парсинг не вдасться і повернеться стовпець типу object. Щоб цього уникнути треба додати параметр keep_default_na = False. Заключний параметр dayfirst вказує

функції парсинга, що перше в рядку першим йде день, а не навпаки. Якщо не поставити цей параметр, то функція може не правильно перетворювати дати і плутати місяць і день місцями (рис 4.6).

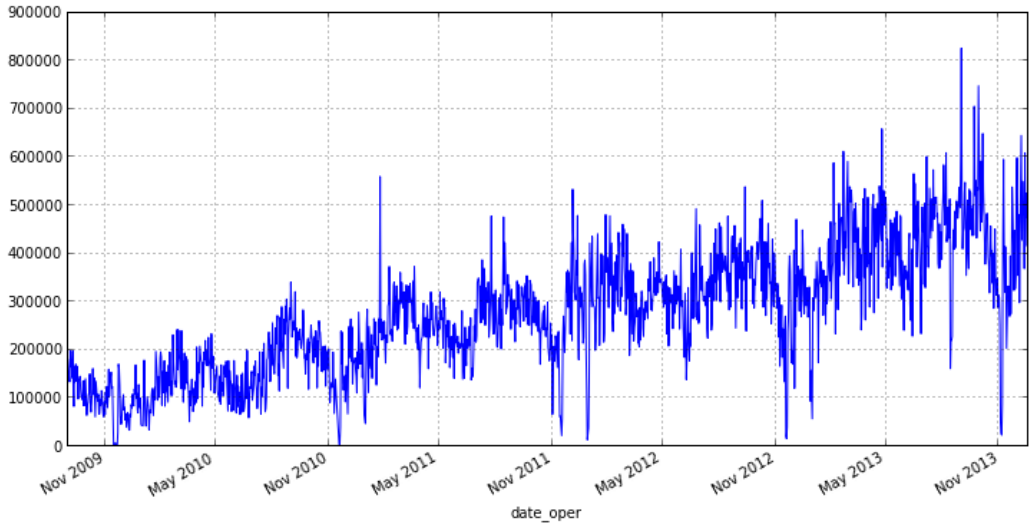

Рисунок 4.6 - Візуалізація історичних даних показників якості сервісів

З графіка видно, що ряд має невелику кількість залишків, які впливають на розподіл. Крім того аналізувати ресурси за кожен день не зовсім вірно, тому що, наприклад, в кінці тижня будуть дні в які кількість активних користувачів значно більше, ніж в інші (рис 4.7). Тому є сенс перейти до тижневого інтервалу і середнього значення використаних ресурсів на ньому, це позбавить нас від викидів і зменшить коливання нашого ряду. У pandas для цього є зручна функція resample, в якості параметрів їй передається період округлення і агрегатна функція.

```
otg = otg.resample('W', how='mean')
otg.plot(figsize=(12,6))
```

Рисунок 4.7 - Візуалізація ряду без залишків

З нового графіку видно відсутність немає яскравих викидів і має яскраво виражений тренд. З цього можна зробити висновок про те, що ряд не є стаціонарним. Для визначення нормальності розподілу необхідно провести тест Харки - Бера, адже це дозволяє підтвердити припущення про однорідність. Даний тест перевіряє помилки спостережень на нормальність за допомогою звірки їх третього моменту (асиметрія) і четвертого моменту (ексцес) з моментами нормального розподілу даного ряду (рис 4.8). Критерії Дарбіна - Уотсона і Голдфелда - Квандта є точними (неасімптотіческімі) в тому сенсі, що вони безпосередньо враховують кількість спостережень у вибірці. На противагу цьому критерії Харке - Бера, Бройша - Годфрі і Уайта є асимптотическими і добре наближаються розподілом хі-квадрат тільки при великому обсязі спостережень. Тому цілком покладатися на результати застосування останніх можна тільки при великій кількості даних у вибірці.

```
row =  [u'JB', u'p-value', u'skew', u'kurtosis']
jb_test = sm.stats.stattools.jarque_bera(otg)
a = np.vstack([jb_test])
itog = SimpleTable(a, row)
print itog
```

```
=============================================================
     JB          p-value          skew         kurtosis
-------------------------------------------------------------
5.60453241944 0.0606724103035 0.111910758759 2.25991843713
-------------------------------------------------------------
```

Рисунок 4.8 - Розрахунок показників моделі

Значення цієї статистики свідчить про те, нульова гіпотеза про нормальність розподілу відкидається з малою вірогідністю (probably> 0.05), і, отже, наш ряд має нормального розподілу. Функція SimpleTable служить для оформлення висновку. У даному випадку на вхід їй подається масив значень (співвідношення не більше 2) і список з назвами стовпців або рядків (рис 4.9).

Більшість методів і моделей засновані на припущеннях про стаціонарності ряду, але як було відмічено раніше наш ряд таким швидше за все не є. Тому для перевірки перевірки стаціонарності доцільно провести узагальнений тест Діккі-Фуллера на наявність поодиноких коренів. Для цього в модулі statsmodels є функція adfuller.

Тест Дікі-Фуллера - це методика, яка використовується в прикладній статистиці і економетрики для аналізу часових рядів для перевірки на стаціонарність. Був запропонований в тисяча дев'ятсот сімдесят девятому році Девідом Дікі і Вейном Фуллером. Тест виконується послідовно для кожної обраної змінної. Розраховується ADF-статистика і порівнюється з N процентним (N = 1%, 5%, 10%) рівнем значущості. На основі даного порівняння робиться висновок про стаціонарності або нестаціонарності ряду. Результати виводяться в табличній формі.

```
test = sm.tsa.adfuller(otg)
print 'adf: ', test[0]
print 'p-value: ', test[1]
print'Critical values: ', test[4]
if test[0]> test[4]['5%']:
    print 'ряд є стаціонарним'
else:
    print 'одиничних коренів немає, ряд не є стаціонарним'
```

```
adf: -1.38835541357
p-value: 0.58784577297
Critical values: {'5%': -2.8753374677799957, '1%': -3.4617274344627398, '10%': -2.5741240890815571}
есть единичные корни, ряд не стационарен
```

Рисунок 4.9 - Розрахунок стаціонарності ряду

Проведений тест підтвердив припущення про нестаціонарність ряду. У багатьох випадках взяття різниці рядів дозволяє це зробити. Якщо, наприклад, перші різниці ряду стаціонарні, то він називається інтегрованим рядом першого порядку. Функція diff обчислює різницю вихідного ряду з рядом із заданим зміщенням періоду. Період зміщення передається як параметр period. Оскільки в різниці перше значення вийти невизначеним, то нам треба позбутися його для цього і використовується метод dropna. Висока p-value дає нам можливість стверджувати, що нульова гіпотеза про рівність середніх вірна, що свідчить про стаціонарності ряду.

Після побудування нового графіку, можна побачити що тренд дійсно відсутній, таким чином ряд перших різниць є стаціонарним, а вихідний ряд - інтегрованим рядом першого порядку (рис 4.10). Початковий (вихідний) ряд є інтегрованим рядом першого порядку, коли його перші різниці утворюють стаціонарний ряд динаміки. Якщо для формування стаціонарного часового ряду потрібно отримати ряд других різниць, то вихідний ряд називається інтегрованим рядом другого порядку і т.д.

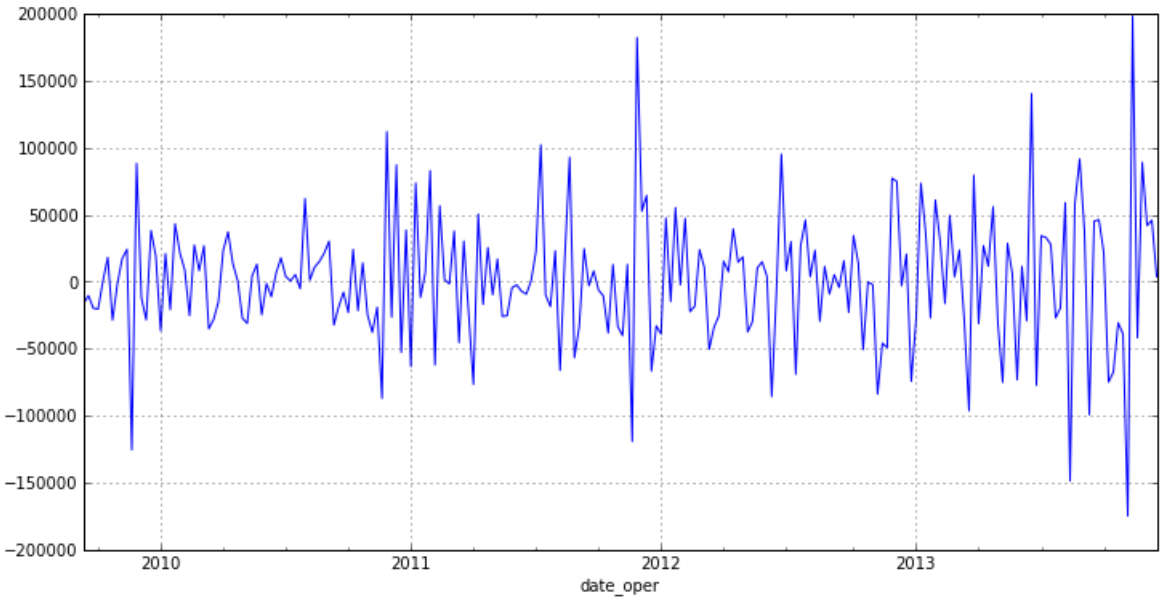

Рисунок 4.10 - Візуалізація ряду після прибирання тренду

Для моделювання доцільно використовувати модель ARIMA, побудовану для ряду перших різниць. Для того, щоб побудувати модель, необхідно знати її порядок, що складається з параметрів p (порядок компоненти AR), d (порядок інтегрованого ряду) і q (порядок компонетни MA).

Параметр d дорівнює 1, залишилося визначити p і q. Для їх визначення нам треба вивчити авторкорреляціонную (ACF) і частково автокорреляционную (PACF) функції для ряду перших різниць. ACF допоможе нам визначити q, оскільки по її коррелограмм можна визначити кількість автокореляційних коефіцієнтів сильно відмінних від 0 до моделі MA. PACF допоможе нам визначити p, оскільки по її коррелограмм можна визначити максимальний номер коефіцієнта сильно відмінний від 0 до моделі AR.

Щоб побудувати відповідні коррелограмми, в пакеті statsmodels є функції plot_acf і plot_pacf. Дані функції візуалізують графіки ACF і PACF, у яких по осі X відкладаються номери лагів, а по осі Y значення відповідних функцій. Потрібно відзначити, що кількість лагів у функціях і визначає число значущих коефіцієнтів числового ряду (рис 4.11).

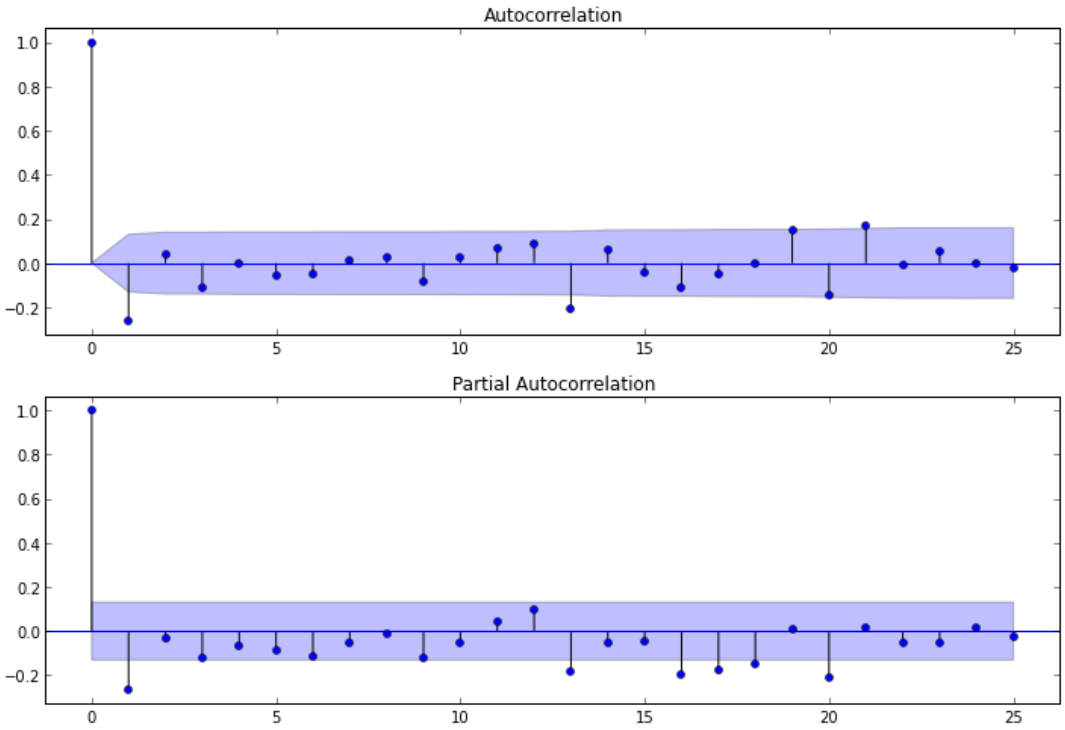

Рисунок 4.11 - Графіки функції автокореляції та часткової автокореляції

С коррелограмми PACF можна зробити висновок, що p дорівнює одиниці, тому що на ній тільки 1 лаг сильно відмінний від нуля. За коррелограмм ACF можна побачити, що q дорівнює одиниці, тому що після лага 1 значенні функцій різко падають. Отже, при відомих значеннях всіх параметрів доцільно побудувати модель, але для її побудови необхідно взяти не всі дані, а тільки частину. Дані з частини не потрапили до модель залишимо для перевірки точності прогнозу моделі (рис 4.12).

```
src_data_model = otg[:'2013-05-26']
model = sm.tsa.ARIMA(src_data_model, order=(1,1,1), freq='W').fit(full_output=False, di
sp=0)
```

Рисунок 4.12 - Отримання моделі ARIMA

З отриманого результату видно, що у отриманої моделі всі коефіцієнти значимі, а отже доцільно перейти до оцінки моделі (рис 4.12).

ARIMA Model Results

Dep. Variable:	D.y	No. Observations:	194
Model:	ARIMA(1, 1, 1)	Log Likelihood	-2326.028
Method:	css-mle	S.D. of innovations	38615.075
Date:	Tue, 24 Dec 2013	AIC	4660.057
Time:	02:12:47	BIC	4673.128
Sample:	09-13-2009 - 05-26-2013	HQIC	4665.350

	coef	std err	z	P>|z|	[95.0% Conf.	Int.]
const	1588.2266	142.728	11.128	0.000	1308.484	1867.969
ar.L1.D.y	0.6660	0.055	12.166	0.000	0.559	0.773
ma.L1.D.y	-1.0000	0.014	-72.214	0.000	-1.027	-0.973

Roots

	Real	Imaginary	Modulus	Frequency
AR.1	1.5015	+0.0000j	1.5015	0.0000
MA.1	1.0000	+0.0000j	1.0000	0.0000

Рисунок 4.12 - Статистичні показники моделі

Перш за все, необхідно перевірити залишки даної моделі на відповідність «білого шуму», а також проаналізуємо коррелограму залишків, оскільки це може допомогти у визначенні важливих для включення і прогнозування елементів регресії. Отже перш за все необхідно провести Q-тест Льюнг - Боксу для перевірки гіпотези про те, що залишки випадкові, тобто є «білим шумом». Даний тест проводиться на залишках моделі ARIMA (рис 4.13). Таким чином, нам треба спочатку отримати залишки моделі і побудувати для них ACF, а потім до вийшов коефіцієнтам примітити тест.

```
q_test = sm.tsa.stattools.acf(model.resid, qstat=True) #
print DataFrame({'Q-stat':q_test[1], 'p-value':q_test[2]})
```

Рисунок 4.13 - Отримання залишків моделі

Значення цієї статистики і p-values, свідчать про те, що гіпотеза про випадковості залишків не відкидається, і швидше за все цей процес представляє «білий шум».

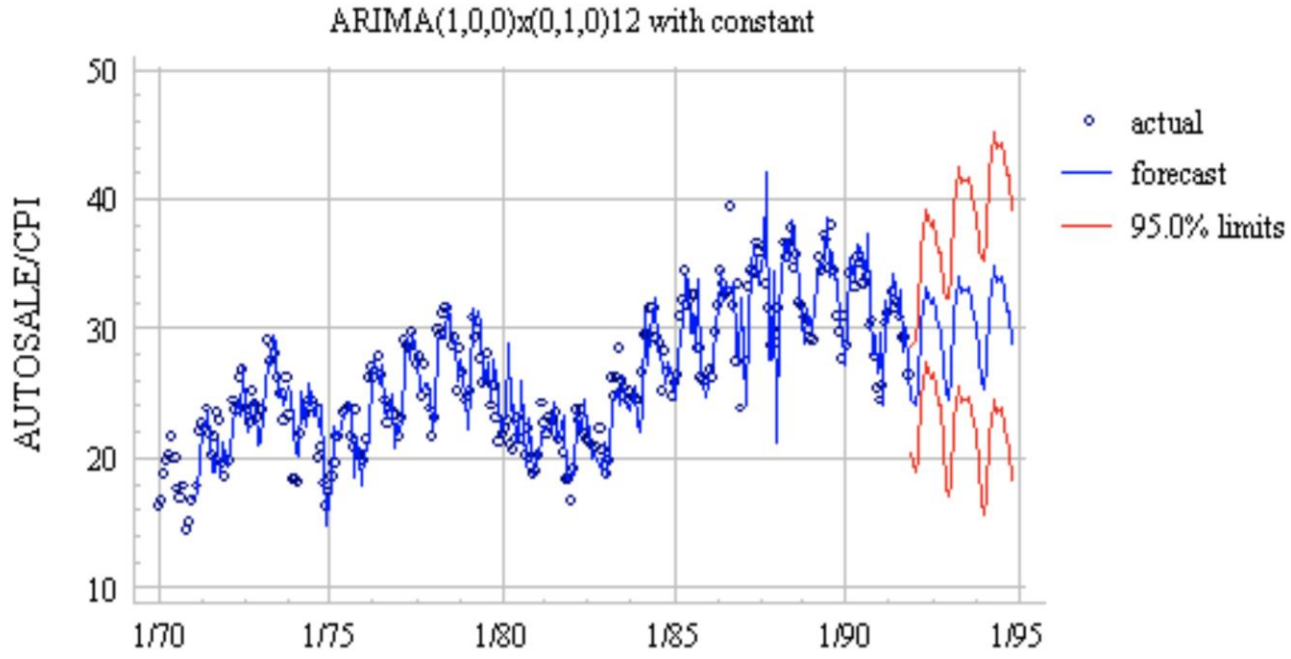

Рисунок 4.14 - Візуалізація прогнозу даних

Використовуючи побудованій моделі ARIMA в компанії є можливість прогнозувати кількість обчислювальних ресурсів, що буде необхідна в наступні проміжки часу, що виражені у тижнях (рис 4.14).

Таким чином, системні адміністратори компанії мають можливість заздалегідь зарезервувати необхідну кількість обчислювальних ресурсів у хмарного провайдера, або ж придбати необхідну кількість власного серверного обладнання. Таким чином, компанія заощаджує кошти, оскільки при бронюванні серверів заздалегідь ціна на них є нижче ринкової. Також, компанія не завдає збитків при пікових навантаженнях, оскільки в корпоративній IT-інфраструктури завжди є резерв вільних обчислювальних ресурсів, що можуть бути надані сервісам за необхідності.

4.3 Реалізація системи бізнес-аналітики та візуалізації даних

Візуалізація даних, неодмінно, є невід'ємною частиною будь-якого сховища даних. Одним із потенційних користувачів сховища є бізнес-аналітики, якім необхідно надати зручний інтерфейс до даних із можливістю створення графіків різних типів. Наприклад, аналітик може побудувати графік росту або падіння продажів за квартал та наглядно продемонструвати менеджерам компанії.

Superset дозволяє користувачам споживати дані різними способами: писати запити SQL, створювати нові таблиці, створювати візуалізацію (фрагмент), додавати цю візуалізацію до однієї або багатьох інформаційних панелей та завантажувати CSV. SQL Lab є частиною Superset і забезпечує багатий редактор SQL, який дозволяє користувачам як запитувати, так і візуалізувати дані. Користувачам надається можливість досліджувати та переглядати таблиці в Presto, без особливих зусиль складати SQL запити для доступу до даних. Звідти ви можете експортувати файл CSV або негайно візуалізувати свої дані у вікні Superset "Explore".

Superset розроблений так, щоб він був легко інтегрованим в хмарне середовище, оскільки продукт був розроблений для масштабних, розподілених повинен працює всередині контейнерів (рис 4.15). Продуктивність даного продукту можна протестувати на локальному комп'ютері, проте для масштабування платформи практично немає меж, оскільки продукт є горизонтально масштабований. Superset може бути налаштований для будь-якої компанії, оскільки існує можливість обирати веб-сервер (Gunicorn, Nginx, Apache), двигун бази даних метаданих (MySQL, Postgres, MariaDB,…), чергу повідомлень (Redis, RabbitMQ, SQS,…), результати пошуку (S3, Redis, Memcached,…), рівень кешування (Memcached, Redis,…) добре працює з такими сервісами, як NewRelic, StatsD і DataDog, і має можливість запускати аналітичні навантаження на більшості популярні технології баз даних.

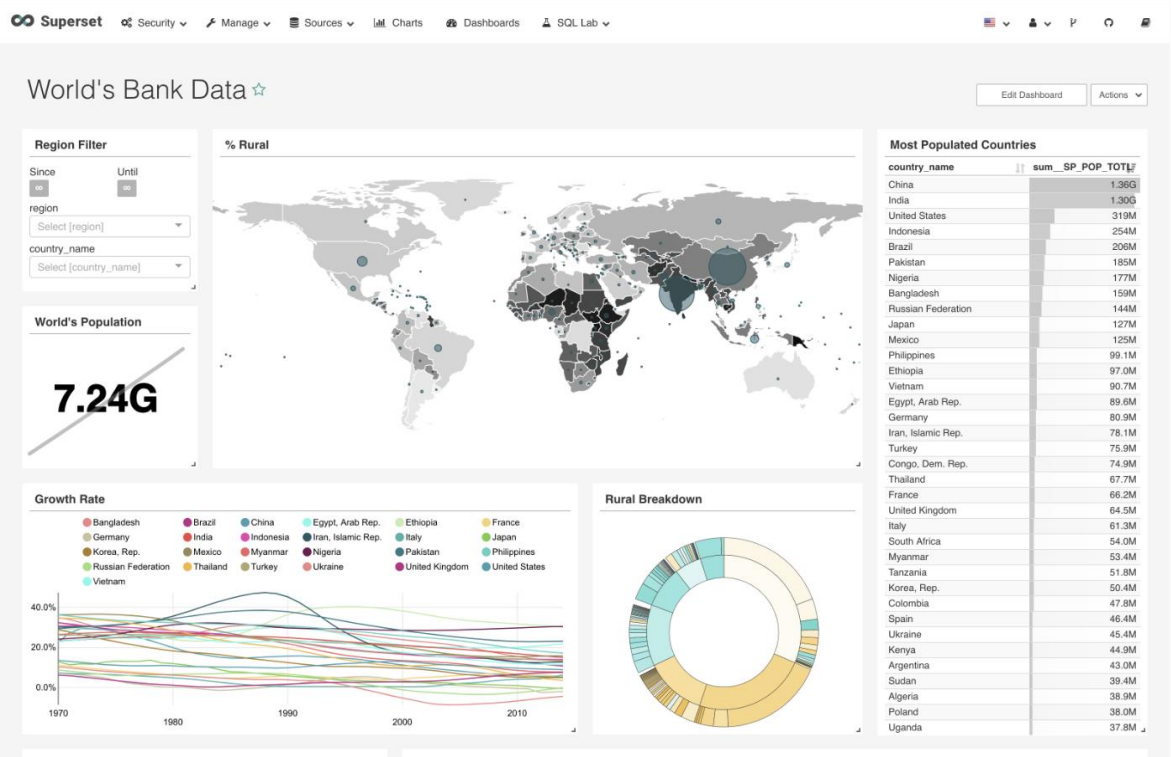

Рисунок 4.15 - Веб-інтерфейс додатку Superset

Однією з найбільш використовуваних можливостей даного продукту є лабораторія SQL запитів (SQL Labs), яка надає користувачу можливість через користувацький інтерфейс веб-додатку створювати запити на мові SQL (рис 4.4). У даному підрозділі веб-додатку вбудована перевірка синтаксису запитів, а також автодоповнення ключових слів за стандартом SQL-92. Створивши текст запиту: користувачам надається можливість зберегти його для подальшого використання, вказавши назву, або ж інтерактивно виконати запит, отримавши результат виконання в вигляді таблиці. Також, отриманий результат можно фільтрувати, зберегти в форматах cvs, json або в звичайному текстовому форматі (рис 4.16).

Взаємодія користувача з підсистемою аналітичної звітності реалізовано за допомогою функціональності Apache Superset:

– даний програмний продукт є безкоштовним для використання під ліцензією Apache Foundation;
– взаємодія відбувається в web-середовищі, тобто сервіс може бути запущений у приватній корпоративній IT-інфраструктурі;
– звіти формуються користувачем в зручному візуальному конструкторі звітів;
– інформація візуалізується у вигляді Звітів і Інформаційних панелей.

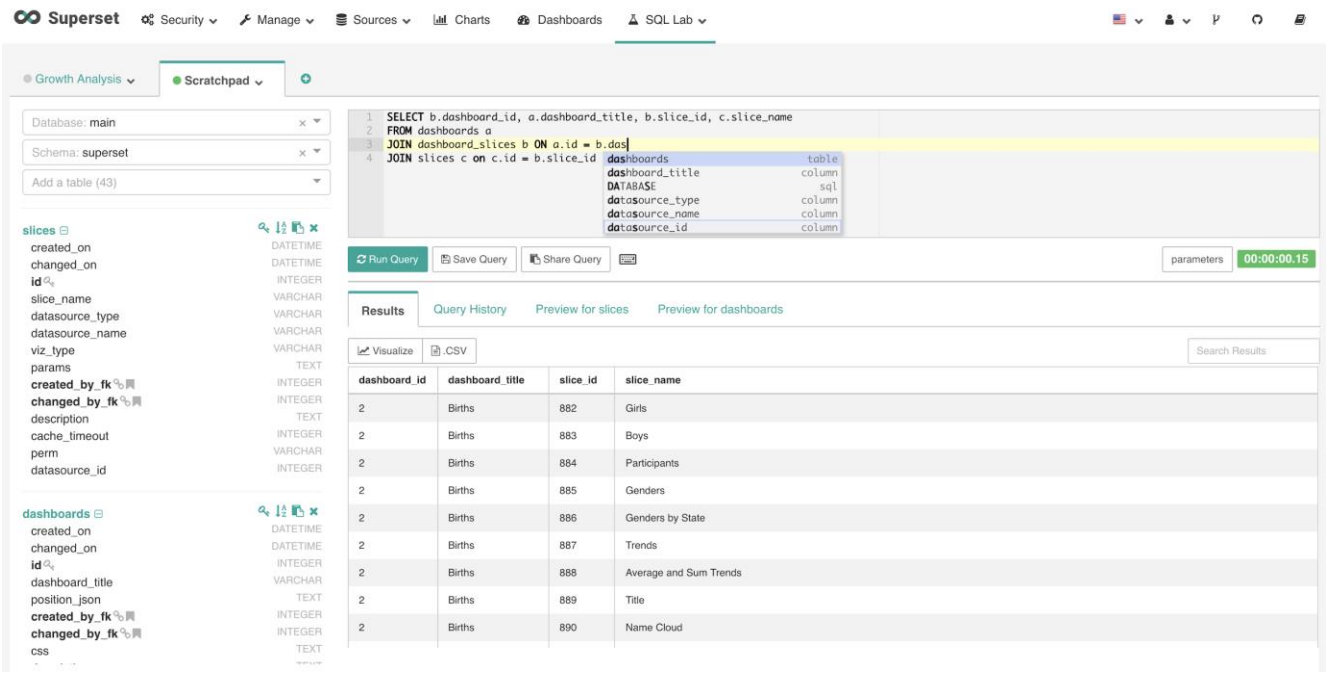

Рисунок 4.16 - Веб-застосунок модулю SQL Labs

Підсистема «Аналітичної звітності» надає функціональні можливості забезпечення можливості отримання даних зі сховища даних і формування на їх базі необхідних користувачам аналітичних звітів будь-якого типу. В даному продукті вже реалізовано ряд найбільш використовуваних видів представлення даних: графік, кругова діаграма, столбчатая діаграма, таблиця зрізу, лінійна таблиця.

В інструменті автоматизоване створення довільних форм звітності засобами вбудованого конструктора формування звітів (конструктора) самостійно користувачами без необхідності використання спеціалізованих мов і алгоритмів. Також, слід зазначити що в Superset є можливість об'єднання показників в папки для подальшого зручності, вибору показників в звіти.

Даний продукт легко та зручно інтегрується з рядом популярних продуктів баз даних, наприклад, з Postgres, MySQL, Redshift, Vertica. Для побудування більш універсальних звітів вбудована формування параметризованих звітів на основі значень параметрів, переданих користувачем через форму веб-інтерфейсу. Для більшої персоналізації в продукті присутня інтеграція с сервісами ідентифікації співробітників компанії, завдяки чому кожен користувач може створити для себе папки зі звітами, найбільш використовувані запити, віджити та інше.

4.4 Висновки до розділу

В даному розділі була розроблена підсистема моніторингу показників якості сервісів корпоративної IT-інфраструктури. В якості черги повідомлень було обрано продукт RebbitMQ, оскільки він є зручним, легко масштабованим.

Була побудована модель ARIMA для прогнозу числового ряду показників сервісів, було розраховано необхідні коефіцієнти та побудовано графік прогнозу. Для реалізації системи бізнес-аналізу було обрано Apache Superset - сервіс візуалізації та побудови звітів із інтерактивним середовищем виконання SQL запитів.

5 РОЗРОБКА СТАРТАП ПРОЕКТУ

Стартап або стартап-компанія (від англ. Start-up - запускати) - це тип бізнесу, спрямований на отримання доходу шляхом реалізації принципово нової ідеї. Термін стартап став популярним за часів доткомів, коли було створено велику кількість інтернет-компаній, що займаються новітніми технологіями і надання послуг, які раніше не існували. Нові проекти в галузях високих технологій часто називають хайтек-стартап. На відміну від звичайного бізнесу який людина тільки відкриває, стартап має інноваційну основу, тобто відкривається бізнес який раніше не існував взагалі.

Термін стартап застосовується для всіх галузей економіки. На прикладі сфери інформаційних технологій та інтернет-проектів, в IT стартапами називають нові компанії, тільки з'являються або плановані до створення. IT-стартапи не прив'язані до агресивних маркетингових і піар-методам, спрямованим на швидку розкрутку фірми. IT-стартапи можуть створюватися на тривалий термін і з великим терміном виходу на ринок (наприклад, створення продукту протягом двох років і тільки після цього вихід на ринок і просування).

Від класичного бізнесу стартап відрізняється тим, що практично відразу потребує коштів інвесторів. Тобто інвестори вкладають гроші в компанію не в той момент, коли вона вже успішно працює, а тільки на самому початку її зародження. Стартап ґрунтується на ідеях, і саме в ці ідеї і інвестори вкладають гроші. Основні ідеї для стартапу знаходяться в сфері IT. Тому застосування технологій в новій бізнес-моделі важливо. Навіть в традиційних галузях медицини або сільського господарства можуть застосовуватися високотехнологічні ідеї. Нові проекти розвиваються куди швидше. Якщо ідея зацікавила користувачів, то буквально за рік компанія перетвориться в лідера і досягне значних розмірів.

5.1 Опис ідеї проекту

Система аналізу якості сервісів, що легко інтегрується в існуючу хмарну корпоративну IT-інфраструктуру. Дана система дає користувачам можливість зручного доступу до даних з корпоративного сховища даних, дозволяє будувати та виконувати аналітичні моделі, створювати звіти на основі історичних даних та виконувати інтерактивні запити до даних. Опис основної ідеї стартапу наведено в таблиці 5.1.

Таблиця 5.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрями застосування	Вигоди для користувача
Система аналізу якості сервісів, що легко інтегрується в існуючу хмарну корпоративну IT-інфраструктуру	Оптимізація ресурсів корпоративної IT-інфраструктури	Економія часу та коштів на самостійне побудування системи аналізу для корпоративної IT-інфраструктури, широкий вибір аналітичних моделей та рішень підтримки якості сервісів на узгодженому рівні

На ринку можна знайти три основних конкурентів, що пропонують схожий функціонал. Основними недоліками є те, що ці рішення створені популярними провайдерами хмарних послуг, а отже сервіс є інтегрованим тільки з одним провайдером хмарних послуг.

Порівняльний аналіз потенційних техніко-економічних сильних, слабких та нейтральних характеристик рішення із основними конкурентами в області аналізу якості сервісів корпоративної IT-інфраструктури наведено в таблиці 5.2.

Таблиця 5.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

№ п/п	Техніко-економічні характеристики ідеї	(потенційні) товари/концепції конкурентів			W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		AWS CoudWatch	GCP Stackdriver	Azure Monitor			
1	Вартість	Середня вартість	Середня вартість	Середня вартість			+
2	Візуалізація даних	Так	Так	Так		+	
3	Запуск аналітичних моделей	Так	Так	Ні			+
4	Універсальний збір метрик	Ні	Так	Ні			+
5	Наявність клієнтської бази	Ні	Так	Так	+		

5.2 Технологічний аудит ідеї проекту

Для створення системи доцільно провести технічний аудит, що представлений в таблиці 5.3.

Таблиця 5.3 – Технологічна здійсненність ідеї проекту

№ п/п	Ідея проекту	Технології	Наявність технологій	Доступність технологій
1	Back-end частина системи управління ресурсами	Java, Spring Boot, Spring Security, Spring Data	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ
2	Front-end частина системи управління ресурсами	HTML, CSS, Bootstrap, JavaScript	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ
3	Корпоративне сховище даних	Hadoop, Parquet, Airflow, Spark	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ
4	Система бізнес-аналізу	Presto, Superset	Наявні, дороблювати не потрібно.	Доступні, відкритий доступ
Для створення системи у вільному доступі знаходяться усі необхідні технології: Java, Spring Framework, HTML, CSS, Hadoop, Presto, Spark.				

5.3 Аналіз ринкових можливостей запуску

Аналіз попиту із вказаними показниками ринку наведений в таблиці 5.4.

Таблиця 5.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1	Кількість головних конкурентів, од	На даному ринку не виявлено видимих конкурентів. На ринку України вони відсутні зовсім, а на світовому ринку їх до 5
2	Загальний обсяг продаж, грн/ум.од	2 млрд. ум. од.
3	Динаміка ринку (якісна оцінка)	Зростає
4	Наявність обмежень для входу (вказати характер обмежень)	Без обмежень
5	Специфічні вимоги до стандартизації та сертифікації	Немає
6	Середня норма рентабельності в галузі (або по ринку), %	Невисока, тридцять відсотків за однин місяць платної підписки.

Визначення потенційних клієнтів наведено в таблиці 5.5.

Таблиця 5.5 – Характеристика потенційних клієнтів стартап-проекту

№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних цільових груп клієнтів	Вимоги споживачів до товару
1	Моніторинг сервісів IT-інфраструктури	Розробники сервісів корпоративної IT-інфраструктури	Потребують моніторингу різних показників	Легкість користування, інтуїтивний інтерфейс, вартість, автоматизація
2	Управління ресурсами сервісів IT-інфраструктури	Адміністратори корпоративної IT-інфраструктури	Потребують горизонтального та вертикального масштабування сервісів	Інтегрованість з хмарними провайдерами, зручний користувацький інтерфейс

Після визначення потенційних груп клієнтів проводиться аналіз ринкового середовища: складаються таблиця 5.7 та факторів, що сприяють ринковому впровадженню проекту, та таблиця 5.6 факторів, що йому перешкоджають

Таблиця 5.6 – Фактори загроз

№	Фактор	Зміст загрози	Можлива реакція компанії
1	Поява нових конкурентів	Поява конкурентів з подібними рішеннями	Зміна ціни та вирішення більшої кількості проблем
2	Зниження цін провайдерів хмарних послуг	Хмарні провайдери можуть знижувати ціну для корпоративних клієнтів	Розширення функціональності

До позитивних факторів слід віднести вільний ринок аналізу даних клієнтів.

Таблиця 5.7 – Фактори можливостей

№	Фактор	Зміст можливості	Можлива реакція компанії
1.	Вільний ринок України	На ринку немає готових рішень	Простота виходу на ринок
2.	Вихід на міжнародний ринок	Оскільки сервіс надається онлайн, то вихід на міжнародний ринок є реалізованим за короткий проміжок часу	Реклама, прямі продажі

Надалі проводиться аналіз пропозиції: визначаються загальні риси конкуренції на ринку, що показано в таблиці 5.8.

Таблиця 5.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції: ринок України – монополістична, світовий – чиста	На ринку України відсутні рішення систем аналізу даних сервісів	Максимально розширювати межі та захоплювати сегменти компаній, що потребують даний продукт
Рівень боротьби: національний	Ринок систем аналізу даних сервісів є по всій території України	Використання продукту на території України
За галузевою ознакою: внутрішньо-галузева	Конкуренція тільки в системах аналізу даних сервісів	Моніторинг ринку
За характером пере-ваг: нецінова	ІС систем аналізу даних сервісів має перевагу в автоматизації та інтеграції з провайдерами хмарних послуг	Розширення інтеграцій з іншими популярними хмарними провайдерами

Модель 5 сил конкуренції Портера використовується для розуміння структури галузі, аналізу її привабливості з точки зору отримання прибутку,

оцінки конкуренції і розробки стратегії бізнесу. Призначення моделі: організація повинна здійснити пошук такої сфери діяльності, в якій вона захищена від дії конкурентних сил, або створити унікальну бізнес-модель і отримувати прибуток вище, ніж в середньому по галузі.

Таблиця 5.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Постачальники	Клієнти	Товари-замінники
	Прямі конкуренти на ринку України відсутні, на світовому частково – GCP Monitor, AWS CloudWatch.	Вихід світових лідерів на ринок України	Постачальники відсутні	Обмеження платоспроможністю	Товари-замінники відсутні
Висновки	Конкуренція несформована	AWS CloudWatch, GCP Monitor, Azure Montoring	Постачальники відсутні	Можлива відмова від рішення	Товари-замінники відсутні

Далі необхідно проаналізувати та визначити фактори конкурентоспроможності компанії, що наведені в таблиці 5.10.

Таблиця 5.10 – Обґрунтування факторів конкурентоспроможності

№	Фактор конкурентоспро-можності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1	Універсальність	Система інтегрується з корпоративною IT-інфраструктурою будь-якого типу та є доступною через прикладний програмний інтерфейс та користувацький веб інтерфейс
2	Ціна	По відношенню до конкурентів, дана система маю меншу собівартість, а це означає, що вартість використання системи є меншою за конкурентів
3	Інновації та технології	Система використовує інноваційні технології машинного навчання та великих даних, розподіленого обчислення та зберігання даних.
4	Масштабованість	Для забезпечення конкурентоспроможності система має бути масштабована і готова до змін та розміру бізнесу клієнта.
5	Швидкість впровадження	Для бізнесу завжди важлива швидкість, щоб бути попереду конкурентів

За визначеними факторами конкурентоспроможності було проведено аналіз сильних та слабких сторін стартапу.

Таблиця 5.11 – Порівняльний аналіз сильних та слабких сторін «Інтелектуальна система фільтрації коментарів за допомогою машинного навчання»

№	Фактор конкурентоспроможності	Рейтинг товарів-конкурентів у порівнянні						
		-3	*-2*	*-1*	*0*	*1*	*2*	*3*
1	Універсальність		✓					
2	Ціна			✓				
3	Інновації та технології				✓			
4	Масштабованість				✓			
5	Швидкість			✓				

За допомогою SWOT-аналізу можна оцінити сильні (Strengths) і слабкі (Weaknesses) сторони, можливості (Opportunities) і ризики (Threats) того чи іншого проекту. Ця модель сходить до досліджень Стенфордського університету 1960-х років, в ході якого вивчалися найбільш успішні компанії США. Результат був такий: розрив між планами компаній і їх конкретним здійсненням становив 35%. Проблема полягала не в низькій компетентності співробітників, а в нечітко поставлених завданнях. Багато співробітників взагалі не уявляли, що і навіщо вони робили. В результаті вчені розробили систему SWOT-аналізу, щоб допомогти учасникам того чи іншого проекту отримати більш чітке уявлення про нього. SWOT наведений у таблиці 5.12.

Таблиця 5.12 – SWOT-аналіз стартап-проекту

Сильні сторони: 1. універсальність; 2. технологія, інновація	Слабкі сторони: 1. ІТ фахівці. 2. досвід
Можливості: 1. несформована конкуренція; 2. попит на системи аналізу показників якості сервісів	Загрози: 1. поява нових конкурентів; 2. стан економіки середовища

Альтернативи ринкового впровадження наведені в таблиці 5.13.

З означених альтернатив обирається та, для якої:

а) отримання ресурсів є простішим та ймовірним;

б) строки реалізації – меньшими.

Таблиця 5.13 – Альтернативи ринкового впровадження стартап-проекту

№	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1	Спільні продажі	Дуже висока. У випадку партнера-гіганта	1 рік
2	Індивідуальні продажі	Середня	1,5-2 роки

5.4 Розроблення ринкової стратегії проекту

Сервіс монетизується шляхом продажу самого рішення компаніям та подальшого оформлення платної підписки. Для оформлення платних підписок на сервіс необхідно визначити цільову аудиторію та групу потенційних споживачів. Опис таких груп наведеного в таблиці 5.14.

Аналіз груп дає можливість сформувати стратегію охоплення ринку для виходу на нього.

Таблиця 5.14 – Вибір цільових груп потенційних споживачів

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1	Великі компанії із власною IT-інфраструктурою що не розміщується в хмарному середовищі	Готові	Зростаючий попит	Низька	Низький бар’єр входу

Продовження таблиці 5.14

№	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
2	Компанії з ІТ-інфраструктурою побудованою в хмарному середовищі	Готові	Зростаючий попит	Низька	Високий бар'єр входу
3	Компанії без власної ІТ-інфраструктури	Не готові	Низький попит	Низька	Високий бар'єр входу
Які цільові групи обрано: Компанії що мають власну ІТ-інфраструктуру та компанії, що користуються послугами хмарних провайдерів для розміщення ІТ-інфраструктури					

Для виходу на ринок необхідно визначити базову стратегію розвитку, охоплення ринку та ключові конкурентоспроможні позиції. Дана інформація наведена в таблиці 5.15.

Таблиця 5.15 – Визначення базової стратегії розвитку

№	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до	Базова стратегія розвитку

			обраної альтернативи	
1	Надання рішення великим компаніям що має ІТ-інфраструктуру	Вибірковий розподіл	Здатність протистояти прямим конкурентам	Стратегія диференціації

За вимогами споживачів у таблиці 5.16 розробляється стратегія конкурентної поведінки. В таблиці 5.17 визначена стратегія позиціонування.

Таблиця 5.16 – Визначення базової стратегії конкурентної поведінки

№	Чи є проект «першо-прохідцем» на ринку?	Чи буде компанія шукати нових споживачів, або забирати існуючих?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки*
1.	Ні	Шукати нових	Ні	Стратегія спрямованості

Таблиця 5.17 – Визначення стратегії позиціонування

№	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)

1	1. Простота використання 2. Ціна 3. Наявність бажаного функціоналу 4. Масштабованість 5. Інтеграція 6. Якість	Стратегія спрямованості	1. Цінова перевага 2. Іноваційне рішення 3. Простота розгортання 4. Простота впровадження 5. Інтеграція 6. Масшабованість	1. Підвищення прибутку 2. Збільшення клієнтів 3. Оптимізація витрат

5.5 Розроблення маркетингової програми

Маркетингова програма - це документ, що викладає, обґрунтовує і деталізує маркетингові цілі і маркетингові плани компанії. Маркетингова програма - результати виконаної роботи відділу або служби маркетингу і в той же час заділ на майбутнє. Першим кроком є формування маркетингової концепції товару, який отримує споживач. Для цього у таблиці 5.18 потрібно підсумувати результати попереднього аналізу конкурентоспроможності товару.

Таблиця 5.18 – Визначення ключових переваг концепції потенційного товару

№	Потреба	Вигода, яку про-понує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити

1	Збільшення прибутку	Спрямування ресурсів лише на ті групи клієнтів, які більше приносять прибутку	Вартість, технологія, відсутність зайвих функцій
2	Оптимізація витрат	Зменшення витрат на пошук прихованих груп клієнтів	
3	Підвищення якості оцінки коментарів	Підвищення якості роботи інтелектуальної системи	Якість надання послуг

Наступним кроком є розробка трирівневої маркетингової моделі товару, для чого уточнюється ідея послуг та продуктів, що наведено таблиця 5.19.

Таблиця 5.19 – Опис трьох рівнів моделі товару

<table>
<tr><td>Рівні товару</td><td colspan="3">Сутність та складові</td></tr>
<tr><td>І. Товар за задумом</td><td colspan="3">Система аналізу показників якості сервісів хмарної корпоративної ІТ-інфраструктури із використанням технологій машинного навчання та великих даних</td></tr>
<tr><td rowspan="4"></td><td>Властивості/характеристики</td><td>М/Нм</td><td>Вр/Тх /Тл/Е/Ор</td></tr>
<tr><td>Кількість</td><td></td><td>1 шт.</td></tr>
<tr><td colspan="3">Якість: стандарти якості постачання програмних продуктів</td></tr>
<tr><td colspan="3">Пакування: через веб-інтерфейс</td></tr>
<tr><td></td><td colspan="3">Марка: Antihate</td></tr>
<tr><td rowspan="2"></td><td colspan="3">Програмний продукт</td></tr>
<tr><td colspan="3">Програмний продукт, технічна підтримка та підписка на розширений функціонал</td></tr>
<tr><td colspan="4">За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності</td></tr>
</table>

Наступним кроком є визначення цінових меж на потенційний товар, яке передбачає аналіз ціни на товари-аналоги, а також аналіз рівня доходів цільової групи споживачів, що наведене в таблиці 5.20.

Таблиця 5.20 – Визначення меж встановлення ціни

№	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар/послугу
1.	Від 20$ в місяць за кожен сервіс	Від 94$ в місяць	Від 1955$ в місяць	Нижня:0$ Верхня: 200$

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення, таблиця 5.21:

− проводити збут власними силами або залучати сторонніх посередників (власна або залучена система збуту);

− вибір та обґрунтування оптимальної глибини каналу збуту;

− вибір та обґрунтування виду посередників.

Таблиця 5.21 – Формування системи збуту

№	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1	Оформлення платної підписки через веб-додаток	Доступ до додаткових функцій сервісу	Канал одного рівня	Селективна з використанням комбінованого каналу збуту

Останньою складової маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів, що наведено в таблиці 5.22.

Таблиця 5.22 – Концепція маркетингових комунікацій

№	Специфіка поведінки цільових клієнтів	Канали комунікацій, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного звернення
1.	Управління ресурсами сервісів корпоративної IT-інфраструктури	Прямі офіційні	Доступність та об’єктивність інформації про фірму і товар Унікальність послуги	Розширення аудиторії	Вказати на використання інновацій та сучасних технологій машинного навчання та кейси вирішення проблем

5.6 Висновки до розділу

В даному розділі було проведено маркетинговий аналіз із завданням визначити конкурентоспроможність розробленого рішення, можливостей виходу на ринок з проектом інтелектуальної системи аналізу показників якості сервісів корпоративної IT-інфраструктури.

За результатами проведеного аналізу було встановлено, що існує велика ймовірність успішного виходу на ринок та швидкої монетизації продукту. На ринку України немає прямих конкурентів рішення, тому розробка та старт проекту реалізується більш легко. За результатами проведеного маркетингового дослідження було обрано стратегії конкурентоспроможності стартапу, а також зроблено порівняльний аналіз рішення із конкурентами, визначено цільову аудиторію та потенційних клієнтів.

Основною перевагою рішення є використання сучасних та інноваційних рішень машинного навчання для аналізу та систем обробки великих даних для зберігання та опрацювання показників якості сервісів. Дане рішення націлено на B2B сегмент, тому найкращим каналом збуту є прямі продажі. Сильними сторонами продукту є його інтегрованість з популярними хмарними провайдерами та легка масштабованість. Виходячи з цього можна стверджувати про доцільність подальшої реалізації продукту.

ВИСНОВКИ

В результаті виконання магістерської дисертації було розроблено та протестовано систему аналізу якості сервісів корпоративної IT-інфраструктури. Також була розроблена структурна схема системи управління функціональністю корпоративної IT-інфраструктури, що складається із агентських додатків, що встановлюються на кожен сервер, та веб-додатку управління ресурсами IT-інфраструктури. Описана система реалізована використовуючи мову програмування Java та фреймворк для будування корпоративних рішень Spark, а саме модулі Spring Web, Spring Security, Spring Data. Дані інструменті є безкоштовними, зручними в використанні та універсальними. Агентські додатки взаємодіють із основним веб-додатком шляхом пересилки та отримання повідомлень до черги. Веб-застосунок реалізований в виді RESTful сервісу, що обмінюється HTTP повідомленнями з веб-сторінкою.

Для розгортання кожного с компонентів системи було обрано систему контейнеризації Docker, адже він задовольняє всі потреби уніфікації процесу побудови та доставки образу сервісу на сервіс корпоративної IT-інфраструктури. Система віртуалізації Docker також надає користувачам можливість створювати власний приватний реєстр образів, таким чином образ кожного з компонентів створеної системи безпечно зберігається на сервері IT-інфраструктури компанії та не є доступним з зовнішньої мережі.

Було спроектовано та реалізовано корпоративне сховище даних, що складається з розподіленої файлової системи та інструментів розподіленого обчислення, що оперують даними, збереженими на ній. В якості розподіленої файлової системи було обрано продукт Hadoop Distributed File System, який зарекомендував себе як надійну і захищену файлову систему, що здатна масштабуватися до десятків тисяч серверів без втрати якості виконання.

В систему також було інтегровано модуль генерування аналітичних звітів, в який можна швидко додавати програми написані за допомогою фреймворку Spark. Була розроблена підсистема моніторингу показників якості сервісів корпоративної

IT-інфраструктури. В якості черги повідомлень було обрано продукт RebbitMQ, оскільки він є зручним, легко масштабованим.

Була побудована модель ARIMA для прогнозу числового ряду показників сервісів, було розраховано необхідні коефіцієнти та побудовано графік прогнозу. Для реалізації системи бізнес-аналізу було обрано Apache Superset - сервіс візуалізації та побудови звітів із інтерактивним середовищем виконання SQL запитів.

В останньому розділі було проведено маркетинговий аналіз із завданням визначити конкурентоспроможність розробленого рішення, можливостей виходу на ринок з проектом інтелектуальної системи аналізу показників якості сервісів корпоративної IT-інфраструктури. За результатами проведеного аналізу було встановлено, що існує велика ймовірність успішного виходу на ринок та швидкої монетизації продукту. На ринку України немає прямих конкурентів рішення, тому розробка та старт проекту реалізується більш легко.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Best practices for service-level agreement [Електронний ресурс] – Режим доступу до ресурсу: https://www.cio.com/article/2438284/outsourcing-sla-definitions-and-solutions.html.

2. Optimal CPU Utilization [Електронний ресурс]. – 2014. – Режим доступу до ресурсу: https://turbonomic.com/blog/on-turbonomic/optimal-cpu-utilization-depends/.

3. Service Level Agreement (SLA) [Електронний ресурс] – Режим доступу до ресурсу: http://www.legalservicesboard.org.uk/Projects/pdf/service_level_agreement_guidance_final_version.pdf.

4. Теленик С.Ф. УПРАВЛІННЯ НАВАНТАЖЕННЯМ І РЕСУРСАМИ ЦЕНТРІВ ОБРОБЛЕННЯ ДАНИХ ПРИ ВИДІЛЕНИХ СЕРВЕРАХ [Електронний ресурс] / Теленик С.Ф., Ролік О.І., Букасов М.М. – Режим доступу до ресурсу: http://aaecs.org/telenik-sf-rolko-bukasov-mm-rimarrv-rolk-ko-upravlnnya-na-vantajennyam--resursami-centrv-obroblennya-danih-pri-vidlenih-serverah.html.

5. Azure monitor overview [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/azure/azure-monitor/overview

6. Теленик С.Ф., Ролік А.А, Букасов М.М. Моделі управління розподілом обмежених ресурсів в інформаційно-телекомунікаційній мережі // Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. — К.: Екотех. — 2006. — №44. — С. 243—246.

7. Ролик А.И. Модель управления перераспределением ресурсов информационно-телекоммуникационной системы при изменении значимости бизнес-процессов// Автоматика. Автоматизация. Электротехнические комплексы и системы. ХГТУ, 2007. — №2(20).— С. 73—82.

8. ТЕЛЕНИК С.Ф. МОДЕЛІ УПРАВЛІННЯ ВІРТУАЛЬНИМИ МАШИНАМИ ПРИ СЕРВЕРНІЙ ВІРТУАЛІЗАЦІЇ [Електронний ресурс] /

ТЕЛЕНИК С.Ф., РОЛІК О.І., БУКАСОВ М.М.. – 2009. – Режим доступу до ресурсу: http://it-visnyk.kpi.ua/wp-content/uploads/2011/03/51_26.pdf.

9. Теленик С.Ф., Ролік А.А., Букасов М.М. Технологія управління IT-інфраструктурою на основі ресурсного підходу// Вісник ЖДТУ. — 2008.— №4(47). — с. 180—189.

10. Теленик С.Ф., Ролік О.І., Букасов М.М., Соколовський Р.Л. Система управління інформаційно-телекомунікаційною системою корпоративної АСУ// Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка. — К.: «ВЕК+», — 2006. — № 45.— С. 112—126.

11. A minimal Docker image based on Alpine Linux with a complete package index and only 5 MB in size! [Електронний ресурс] – Режим доступу до ресурсу: https://hub.docker.com/_/alpine/.

12. Private Docker Registry [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.aquasec.com/wiki/display/containers/Docker+Registries+101.

13. Swarm mode [Електронний ресурс] – Режим доступу до ресурсу: https://docs.docker.com/engine/swarm/.

14. Amazone ECS [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/ecs/.

15. The NIST Definition of Cloud Computing [Електронний ресурс] // Computer Security Division Information Technology Laboratory National Institute of Standards and Technology Gaithersburg, MD 20899-8930. – 2011. – Режим доступу до ресурсу: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-145.pdf.

16. WHAT IS PAAS [Електронний ресурс] – Режим доступу до ресурсу: https://azure.microsoft.com/ru-ru/overview/what-is-paas/.

17. Опрацювання запитів до сервісу [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: http://ena.lp.edu.ua/bitstream/ntb/20129/1/3-11-13.pdf.

18. Теленик С.Ф. Швидке розроблення застосувань в адаптивній технології SmartBase / С.Ф. Теленик, О.А. Амонс, В.С. Хмелюк, К.О. Крижова // Проблеми програмування. – 2006. – №1–2. Спец. випуск. – С. 299 – 305.

19. Zaghdoudi T. Bank Failure Prediction with Logistic Regression [Текст] / T Zaghdoudi // International Journal of Economics and Financial Issues, Vol. 3, No. 2, 2013, – pp.537-543

20. Yehuda B., Brooks C. Survey Response Rate Levels and Trends in Organizational Research, 2008. – 61 с.