ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Мікросервісна архітектура для аналізу криптовалютного ринку та автоматизованої торгівлі»

на здобуття освітнього ступеня магістра
зі спеціальності 123 Комп'ютерна інженерія
освітньо-професійної програми «Комп'ютерні системи та мережі»

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

_____ Роман Мироненко
(підпис)

Виконав: здобувач вищої освіти групи КСДМ-61
Роман МИРОНЕНКО

Керівник: Віталій ВОЛОХІН
к.т.н., доцент

Рецензент: _____
науковий ступінь, вчене звання *Ім'я, ПРІЗВИЩЕ*

Київ 2025

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерна інженерія

Ступінь вищої освіти Магістр

Спеціальність 123 Комп'ютерна інженерія

Освітньо-професійна програма «Комп'ютерні системи та мережі»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Комп'ютерної інженерії
_____ Наталія ЛАЩЕВСЬКА
«_____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Мироненку Роману Олександровичу

1. Тема кваліфікаційної роботи: «Мікросервісна архітектура для аналізу криптовалютного ринку та автоматизованої торгівлі»

керівник кваліфікаційної роботи Віталій ВОЛОХІН, к.т.н., доцент,

затверджені наказом Державного університету інформаційно-комунікаційних технологій від «___» _____ 2025 р. № _____.

2. Строк подання кваліфікаційної роботи «17» листопада 2025 р.

3. Вихідні дані до кваліфікаційної роботи: Науково-технічна література з проєктування мікросервісних архітектур, методи побудови розподілених систем для збору, обробки та зберігання потокових даних, підходи до забезпечення продуктивності, масштабованості та надійності високонавантажених програмних систем.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Дослідження предметної області криптовалютного ринку та підходів до алгоритмічної торгівлі.

2. Аналіз мікросервісних архітектур для обробки великих обсягів фінансових даних у реальному часі.
3. Проєктування мікросервісної архітектури програмної системи.

5. Перелік ілюстративного матеріалу: *презентація*
   1. Актуальність теми дослідження.
   2. Математична оцінка точності відстеження об’єктів для глибокого навчання.
   3. Постановка задачі.
   4. Порівняння технологій.
   5. Прийняті архітектурні та технологічні рішення.
   6. Архітектура мікросервісної системи.
   7. Висновки до магістерської роботи.

6. Дата видачі завдання «01» вересня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз наявної науково-технічної літератури та документації	*21.10 - 25.10.2025*	
2	Вивчення матеріалів для аналізу розвитку технологій обробки криптовалютних ринкових даних	*21.10 – 25.10.2025*	
3	Дослідження методів аналізу ринкових даних та автоматизованої торгівлі	*21.10 - 25.10.2025*	
4	Порівняння технологій, які будуть використовуватись	*21.10 - 25.10.2025*	
5	Вибір технологій на основі аналізу	*21.10 - 25.10.2025*	
6	Розробка мікросервісної архітектури для аналізу криптовалютного ринку та автоматизованої торгівлі	*01.11 - 20.11.2025*	
7	Оформлення роботи: вступ, висновки, реферат	*01.12 - 20.12.2025*	
8	Розробка демонстраційних матеріалів	*01.12 - 20.12.2025*	
9	Попередній захист роботи	*01.12 - 20.12.2025*	

Здобувач вищої освіти _____ Роман МИРОНЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Віталій ВОЛОХІН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 79 стор., 10 рис., 18 джерел.

Мета роботи – розробка та дослідження мікросервісної архітектури для аналізу криптовалютного ринку та автоматизованої торгівлі з метою підвищення продуктивності, масштабованості та надійності обробки ринкових даних.

Об'єкт дослідження – процес обробки та аналізу потокових і історичних даних криптовалютного ринку в розподілених програмних системах.

Предмет дослідження – методи та архітектурні підходи до побудови мікросервісних систем для збору, зберігання, обробки та аналізу ринкових даних з подальшою автоматизацією торгових операцій.

У роботі використано методи проєктування розподілених програмних систем, мікросервісну архітектуру, асинхронну та потокову обробку даних, а також сучасні брокери повідомлень і системи зберігання даних.

Проведено аналіз сучасних архітектурних рішень для побудови високонавантажених систем обробки даних у реальному часі, зокрема систем обміну повідомленнями, форматів зберігання даних та баз даних для часових рядів.

Розроблено мікросервісну архітектуру системи аналізу криптовалютного ринку, що забезпечує масштабування окремих компонентів, ізоляцію функціональних сервісів та підвищення відмовостійкості. Реалізовано сервіси збору ринкових даних, їх обробки, зберігання та формування торгових сигналів.

КЛЮЧОВІ СЛОВА: МІКРОСЕРВІСНА АРХІТЕКТУРА, РОЗПОДІЛЕНІ СИСТЕМИ, ПОТОКОВА ОБРОБКА ДАНИХ, КРИПТОВАЛЮТНИЙ РИНОК, АВТОМАТИЗОВАНА ТОРГІВЛЯ, БРОКЕРИ ПОВІДОМЛЕНЬ, МАСШТАБОВАНІСТЬ.

ABSTRACT

Text part of the master's qualification work: 79 pages, 10 pictures, 18 sources.

The aim of the work is to develop and study a microservice architecture for cryptocurrency market analysis and automated trading in order to improve performance, scalability, and reliability of market data processing.

The object of the study is the process of processing and analyzing streaming and historical data of the cryptocurrency market in distributed software systems.

The subject of the study is methods and architectural approaches to building microservice systems for collecting, storing, processing, and analyzing market data with subsequent automation of trading operations.

The work employs methods of designing distributed software systems, microservice architecture, asynchronous and streaming data processing, as well as modern message brokers and data storage systems.

An analysis of modern architectural solutions for building high-load real-time data processing systems has been carried out, including messaging systems, data storage formats, and time-series databases.

A microservice architecture for a cryptocurrency market analysis system has been developed, providing scalability of individual components, isolation of functional services, and improved fault tolerance. Services for market data collection, processing, storage, and generation of trading signals have been implemented.

KEYWORDS: MICROSERVICE ARCHITECTURE, DISTRIBUTED SYSTEMS, STREAMING DATA PROCESSING, CRYPTOCURRENCYMARKET, AUTOMATED TRADING, MESSAGE BROKERS, SCALABILITY.

Зміст

Вступ 9
1. Аналіз предметної області 10
1.1 Криптовалюта та трейдинг 10
1.2 Сучасний стан криптовалютного ринку 12
1.3 Проблеми аналізу та автоматизованої торгівлі 13
1.4 Роль бек-тестингу у розробці стратегій 14
2. Визначення цілей 17
2.1 Визначення цілей 17
2.2 Підготовка до побудови архітектури 18
3. Порівняння технологій та бірж 20
3.1 Вступ 20
3.2 Аналіз архітектурних підходів 20
3.3 Аналіз технологій, потрібних для архітектури 22
3.4 Порівняння БД 27
3.5 Порівняння форматів файлів 33
3.6 Порівняння брокерів повідомлень 39
3.7 Порівняння мов програмування 46
3.8 Порівняння бірж 50
3.9 Огляд біржі Binance 57
4. Розробка архітектури 64
4.1 Обрані технології 64
4.2 Детальний опис сервісів 69
CandleProcessor 69
FeatureCalculator 71
RealTimeDataManager 73
DBSaver 74
StrategyRunner 75
StrategyManager 76
TradingEngine 77
BTRunner 78
BTManager 79
DataDownloader 80
Висновки 81
Перелік посилань 83
Додаток А. Демонстраційні матеріали(Презентація) 84

Вступ

Сьогодні все більш популярною стає криптовалюта і все, що з нею пов’язано. Цифрові активи поступово перетворюються з вузькоспеціалізованого інструмента у повноцінний елемент світової фінансової системи. Зростає кількість бірж, збільшується обсяг торгів, а технології, що лежать в основі криптовалют, знаходять застосування у різноманітних сферах. Також активно розвивається й інфраструктура для аналізу та автоматизації торгівлі: трейдери та розробники створюють нові торгові стратегії, сервіси збору та обробки даних, системи прогнозування й аналітики. У таких умовах постає потреба у гнучких та надійних інструментах, які дозволять ефективно працювати з великими обсягами історичних і реальних ринкових даних, тестувати потенційні стратегії та використовувати їх у реальному часі.

Крім актуальності у контексті криптовалют, дана тема є важливою і з погляду побудови сучасних програмних архітектур. Розробка системи, здатної обробляти великі обсяги даних, працювати з різними форматами, взаємодіяти з кількома джерелами інформації та масштабуватися під змінне навантаження, є складним і водночас цікавим викликом. Побудова мікросервісної архітектури для задач бектестингу, аналізу й автоматичної торгівлі потребує грамотного вибору технологій, продуманого розподілу відповідальностей між сервісами та створення надійних механізмів взаємодії. Таким чином, ця тема виходить за межі лише фінансової сфери і стає повноцінним інженерним кейсом, що поєднує роботу з великими даними, високонавантаженими компонентами та системами реального часу.

Саме тому в даній роботі я зосереджуюсь на побудові мікросервісної архітектури, яка об’єднає механізми отримання даних, їх структурування, тестування стратегій та можливість подальшої реальної торгівлі. Такий підхід дозволить створити гнучку, масштабовану та практично корисну систему, що відповідає сучасним вимогам.

1. Аналіз предметної області

1.1 Криптовалюта та трейдинг

Криптовалюта – це валюта, яка є цифровою чи віртуальною і захищена криптографією. Вона використовується для безпечних транзакцій в інтернеті. Головна особливість криптовалюти – децентралізація. В основі криптовалюти лежить блокчейн.

Блокчейн – це розподілена база даних, яка зберігає ланцюжок блоків, який постійно стає довшим. Кожен блок складається з позначки, хешу попереднього блоку та даних транзакцій, які подані як хеш-дерево. Інформація про транзакції відкрита(зазвичай) і не шифрується. Захист досягається шляхом включення хешу блоку у наступний блок, таким чином внесення змін в один блок вимагає відповідних змін в усіх блоках після нього.

Кожна транзакція може бути перевірена у блокчейні, при цьому особистість користувача є анонімною завдяки використанню криптографічних адрес.

Блокчейн-технологія забезпечує прозорість операцій, що дозволяє трейдерам отримувати достовірні дані про ринок у режимі реального часу. Це створює передумови для побудови аналітичних моделей і стратегій, які базуються на історичних транзакціях.

Першою криптовалютою вважається Bitcoin, створений у 2009 році під псевдонімом Satoshi Nakamoto. Після цього почали з’являтися інші криптовалюти, одними з найпопулярніших є Ethereum, Solana. Нині існує понад 10000 активних криптовалют, що торгуються на сотнях бірж в усьому світі.

Трейдинг – це торгівля фінансовими активами, метою якого є заробіток на зміні ціни(якщо спрощено - купити дешевше, продати дорожче). Бувають різні типи трейдингу за часом, способом тощо.

На відміну від інвестування, ціллю якого є довгостроковий прибуток, трейдинг більше направлений на активне використання коливань у коротких або середніх часових проміжках.

Типи трейдингу за часовим проміжком:

- Скальпінг – надкороткострокова торгівля, коли угоди тривають від кількох секунд до кількох хвилин. Мета – отримати невеликий прибуток на дрібних коливаннях, компенсуючи його кількістю операцій.
- Денний трейдинг – усі позиції відкриваються та закриваються протягом одного торгового дня. Трейдер працює на внутрішньоденних коливаннях.
- Свінг трейдинг – утримання позицій протягом кількох днів або тижнів для отримання прибутку на середньострокових тенденціях.
- Позиційний трейдинг – довгострокова торгівля з акцентом на фундаментальний аналіз і великі ринкові тренди.

Типи трейдингу за способом:

- Ручний – усі рішення приймає людина. Використовується технічний і фундаментальний аналіз, графіки, індикатори.
- Автоматизований – операції виконуються за допомогою алгоритмів і торгових роботів, які самостійно аналізують дані, визначають сигнали та відкривають або закривають угоди.

Оскільки швидкість ринку і обсяги торгів зростають, ручний трейдинг все більше поступається місцем автоматизованому. Використання автоматичних алгоритмів для торгівлі дозволяє:

- Швидко реагувати на події на ринку(за мілісекунди)
- Одночасно працювати з великою кількістю торгових пар

- В реальному часі розраховувати значення індикаторів
- Автоматично перевіряти нові стратегії на історичних даних

Саме розвиток автоматизованих систем і став передумовою для створення архітектур для трейдингу.

Типи трейдингу за класом активів:

- Spot trading – купівля/продаж активу за поточним курсом.
- Margin trading – торгівля з використанням позикових коштів.
- Futures trading – угоди на майбутню ціну активу.
- Options trading - похідні контракти з правом, але не зобов’язанням купівлі/продажу.
- Arbitrage trading – отримання прибутку з різниці цін одного активу на різних біржах.

Таким чином, розвиток криптовалют, а також блокчейну призвели до появи алгоритмічного трейдингу, який поєднує в собі такі механізми як обробка великих даних, машинне навчання і автоматизоване управління ризиками.

1.2 Сучасний стан криптовалютного ринку

Станом на 2025 рік загальна капіталізація криптовалютного ринку перевищує 2 трильйони доларів США, а щоденний обсяг торгів сягає понад 100 мільярдів доларів. Найбільшу частку ринку займають біржі Binance, Coinbase, OKX та Bybit.

Біржі відіграють ключову роль у забезпеченні ліквідності та доступності даних. Через відкриті API (наприклад, Binance API або Binance Vision) розробники отримують можливість завантажувати історичні дані свічок, обсяги торгів і поточні котирування в режимі реального часу.

Волатильність – це показник, що описує швидкість зміни певного фінансового активу з плином часу. Чим волатильність вища тим більші коливання і відповідно більші ризики, а також більші можливості заробити.

Криптовалютний ринок характеризується високою волатильністю через простоту трейдингу. З одного боку це спрощує купівлю чи продаж, а з іншого ускладнює поведінку ринку. Традиційні методи ручної торгівлі не дозволяють вчасно реагувати на швидку зміну цін, тому автоматизована торгівля стала одним з основних напрямків розвитку фінтех технологій.

Фактори, які впливають на ринок:

- глобальні економічні події (регуляторна політика США, ЄС, Китай);
- технологічні новини (оновлення Ethereum);
- активність великих гравців (так званих китів);
- вплив медіа та соціальних мереж.

Через велику кількість інформаційних факторів ринок криптовалют має риси хаотичної системи, що ускладнює побудову стабільних торгових стратегій.

З розвитком автоматизованих систем виникає прагнення до мінімізації людського втручання, однак повна автоматизація є складною через необхідність врахування численних позаринкових факторів (новини, регуляторні зміни тощо).

1.3 Проблеми аналізу та автоматизованої торгівлі

При вирішенні задач аналізу та автоматизованої торгівлі полягають в великому обсязі та швидкості надходження даних, неоднорідність даних між біржами, проблеми з якістю та кількістю історичних даних(відсутність, пропуски). Тобто, перша проблема полягає в отриманні актуальних та дійсних даних. Друга проблема полягає в складності(та часі) обчислення певних значень, індикаторів, оскільки перед запуском потрібно перевірити стратегії перед

реальним запуском. Також є безпекові питання такі як: ліміти, механізми захисту від трат, управління ключами, захист API.

Проблема неоднорідності API. кожна біржа має власний формат API, різні обмеження на запити та інтервали історичних даних. Це створює складнощі при побудові універсальної системи збору даних.

Тестування і продуктивність. Обчислення індикаторів та симуляція стратегій на великих обсягах історичних даних вимагає оптимізації алгоритмів і використання паралельної обробки. Без цього процес бек-тестингу може тривати години або дні.

Автоматизована торгівля повинна враховувати ризики втрати зв’язку, повторних запитів або зависання процесів. Для цього використовуються черги повідомлень, логування, механізми повторного виконання та обмеження ризику (stop-loss, take-profit).

Отже, основні виклики при створенні системи автоматизованої торгівлі полягають у забезпеченні швидкодії, надійності, узгодженості даних і безпеки виконання операцій.

1.4 Роль бек-тестингу у розробці стратегій

Бек-тестинг – це перевірка торгової стратегії на історичних даних. Основна проблема в тому, що просто історичних цін недостатньо для більш точного бектесту, бажано для цього мати ще orderbooks, тобто існуючі лімітні ордери в кожен момент часу, оскільки біржа працює таким чином, що від розміру позиції може залежати середня ціна.

Також, через затримку виконання трейда, бектест може неправильно реагувати на сигнали, оскільки доволі складно на історичних даних змоделювати такі затримки.

Крім цього потрібен певний підхід до оцінки стратегії одразу по декількох критеріях: прибутковість, ризикованість, задіюванність балансу.

Типи бек-тестингу:

- Single-run backtest – один запуск на фіксованих параметрах.
- Walk-forward testing – покрокова перевірка з переналаштуванням стратегії.
- Forward testing (paper trading) – тестування у реальному часі без фактичного виконання ордерів.

Поєднання цих методів дозволяє отримати більш об'єктивну оцінку стратегії.

Метрики ефективності:

- ROI (Return on Investment) – прибутковість;
- Sharpe ratio – співвідношення прибутку до ризику;
- Max Drawdown – максимальне падіння капіталу;
- Win rate, Profit factor, Expectancy.

Завдяки цим метрикам можна комплексно оцінити не лише прибутковість, а й стабільність торгової стратегії.

У межах даної роботи бек-тестинг реалізується як окремий сервіс, що дозволяє незалежно обробляти великі обсяги історичних даних і швидко оцінювати ефективність нових стратегій.

Бек-тестинг є обов’язковим етапом розробки будь-якої торгової системи, оскільки дозволяє мінімізувати ризики втрат і виявити потенційні слабкі сторони алгоритму до його запуску на реальному ринку.

Висновки до розділу 1

У цьому розділі було проаналізовано предметну область криптовалютного ринку, визначено його сучасний стан, ключові проблеми автоматизації та роль бек-тестингу в побудові торгових систем.

Отримані результати показують, що сучасний криптовалютний ринок характеризується високою волатильністю, великим обсягом даних і потребує високопродуктивних рішень.

Тому доцільним є створення мікросервісної архітектури, яка забезпечить масштабованість, гнучкість і можливість незалежної розробки та розгортання окремих компонентів системи.

2. Визначення цілей

2.1 Визначення цілей

Основною ціллю розробки системи є створення гнучкої, масштабованої та надійної мікросервісної архітектури, яка дозволить працювати з торговими стратегіями на всіх ключових етапах їхнього життєвого циклу: від отримання історичних даних та їх попередньої обробки, до тестування цих стратегій на минулих ринкових умовах, а також подальшої реальної роботи з онлайн-даними біржі. Система повинна забезпечувати можливість як офлайнового аналізу, так і повноцінної автоматичної торгівлі в реальному часі, що потребує високого рівня організації компонентів, чіткої взаємодії між сервісами та здатності до подальшого розширення.

Однією з ключових задач архітектури є реалізація якісного механізму бектестингу. Для повноцінної оцінки торгових стратегій система має вміти швидко завантажувати, структурувати та аналізувати великі обсяги історичних даних. Це не просто технічна можливість — це фундамент, на якому будується розуміння того, як стратегія поводилася б у реальних ринкових сценаріях. Саме тому архітектура повинна бути достатньо швидкою, щоб обробляти значні обсяги даних без зайвих затримок, та достатньо простою, щоб розробники могли без труднощів додавати нові стратегії чи змінювати існуючі.

Ще однією важливою ціллю є здатність системи працювати з реальними, онлайн-ринковими даними. Мікросервісний підхід дозволяє окремим компонентам відповідати за вузькі та чітко визначені задачі — наприклад, отримання ринкових свічок, обробку сигналів або прийняття торгових рішень — і при цьому не впливати на роботу інших частин системи. Такий підхід забезпечує як високу надійність, так і безперервність роботи: відмова одного компонента не повинна зупиняти роботу всієї системи.

Безпека та автоматизація є ще одними важливими напрямами. Архітектура повинна передбачати можливість автоматичного виконання повторюваних задач, таких як оновлення історичних даних, перезапуск бектестів або регулярне оновлення параметрів стратегій. Автоматизація зменшує вплив людського фактору та робить систему більш стабільною при довготривалій роботі. Водночас слід враховувати необхідність захисту даних, безпечного доступу до біржі та запобігання несанкціонованим операціям, що особливо важливо у контексті роботи з реальними коштами.

Окремою ціллю є гнучкість системи. Архітектура має легко адаптуватися до нових вимог: додавання нових бірж, нових форматів даних, нових стратегій або нових типів аналізу. Для цього використовується підхід мікросервісів, коли кожен компонент виконує свою окрему функцію, а взаємодія між сервісами організовується через чітко визначені інтерфейси. Це дозволяє розвивати систему поступово, не переписуючи її з нуля при появі нових потреб.

У результаті вся система повинна бути достатньо швидкою для обробки великих даних, достатньо гнучкою для розширення, достатньо безпечною для роботи з реальними біржами та достатньо простою для підтримки та розвитку. Такий баланс дозволить не лише протестувати стратегії у контрольованих умовах, але й використовувати їх у реальній торгівлі, забезпечуючи стабільну та зручну платформу для подальшого вдосконалення.

2.2 Підготовка до побудови архітектури

Перед тим як переходити до розробки архітектури майбутньої системи, важливо провести детальний аналіз технологій, які можуть бути використані під час її створення. Оскільки система складається з декількох мікросервісів, кожен з яких має свої завдання та вимоги, вибір бази даних, брокера повідомлень, формату файлів, мови програмування та навіть біржі, з якої буде отримуватися

інформація, безпосередньо впливає на ефективність, швидкість та надійність роботи всієї архітектури.

Мікросервісний підхід передбачає, що кожен сервіс повинен бути автономним, легко масштабованим та незалежним від інших компонентів. Але для того, щоб ця автономність була реальною, необхідно обрати такі технології, які не лише підходять окремим сервісам, а й дозволяють їм ефективно взаємодіяти між собою. Наприклад, вибір брокера повідомлень визначає спосіб обміну подіями між сервісами, вибір бази даних — спосіб збереження та доступу до результатів, а типи файлів та формат зберігання історичних даних — продуктивність роботи сервісів, що виконують бектест.

Крім того, для роботи з фінансовими ринками важливо враховувати специфіку бірж і форматів, у яких вони надають дані. Деякі біржі надають дані у вигляді CSV, інші — у JSON, деякі підтримують WebSocket-стріми, а інші надають архіви через HTTP. Відповідно, саме порівняння бірж дозволяє зрозуміти, яка з них найкраще підходить для задач бектестингу та реальної торгівлі на початковому етапі.

Тому розділ 3 присвячений саме порівнянню ключових технологій. Це дозволить сформувати обґрунтовану основу для подальшого проектування системи: визначити, які інструменти найбільше відповідають вимогам швидкості, простоти реалізації, можливості горизонтального масштабування, сумісності між сервісами та подальшої підтримки. Без цього порівняння вибір технологій був би випадковим і міг би призвести до обмежень у роботі системи вже на етапі реалізації або масштабування.

Таким чином, порівняння у розділі 3 є необхідним аналітичним кроком, який дозволяє перейти від загальних вимог до конкретних технічних рішень, забезпечуючи фундамент для побудови ефективної та надійної мікросервісної архітектури.

3. Порівняння технологій та бірж

3.1 Вступ

Розробка мікросервісної архітектури для торговлі та бектестингу торгових стратегій – це комплексна задача, оскільки вона вирішує багато проблем, включає взаємодію з великими обсягами даних, алгоритми для аналізу та обробки інформації, має бути масштабованим. На цьому етапі важливо визначити архітектурні рішення, сформувати вимоги до продуктивності, оцінити можливі ризики, придумати структуру майбутньої системи, щоб вона легко масштабувалася в майбутньому, а також інтегрувалася з іншими сервісами загальної мікросервісної архітектури.

В цьому розділі проаналізуємо існуючі підходи, технології, визначимо вимоги і обгрунтуємо вибір технологій.

3.2 Аналіз архітектурних підходів

Існує 2 основних архітектурних підходи: монолітний та мікросервісний.

Монолітний підхід

В монолітній системі всі необхідні функції – завантаження даних, обробка, розрахунок, виконання стратегій, візуалізація – об’єднані в одному програмному модулі.

Переваги моноліту:

- Простота початкової реалізації
- Швидкість передачі даних між об’єктами однієї програми швидша ніж мережею
- Менша кількість точок відмови
- Простіша логіка деплойменту

Недоліки:

- Складність масштабування окремих частин
- Занадто великі системи важче підтримувати
- Можлива велика зв’язність між компонентами.
- Складність тестування окремих компонентів

Мікросервісний підхід

Мікросервісна архітектура передбачаж поділ всієї системи на окремі незалежні компоненти(сервіси), кожен з яких відповідає за чітко визначену функцію.

Переваги мікросервісів:

- Можливість збільшувати продуктивність лише тих частин, які цього потребують
- Ізольованість помилок – збій в роботі одного з сервісів не зупинить роботу всіх інших
- Можливість оновлення окремого компоненту

Недоліки:

- Збільшена складність взаємодії між сервісами
- Швидкість передачі даних між сервісами

Висновок

Монолітна архітектура дає швидкість роботи програми, але більш складна в підтримці та більш ресурсозатратна(при необхідності збільшити пропускну здатність одного компоненту – потрібно масштабувати всю систему). В той час коли мікросервісна архітектура дає можливість масштабувати окремі частини, але швидкість передачі даних між різними сервісами менша.

3.3 Аналіз технологій, потрібних для архітектури

Система працюватиме з великою кількістю історичних даних та даних в реальному часі, тому швидкість отримання і обробки таких даних є важливою.

Почнемо з отримання даних:

Система може отримувати історичні дані з:

- Бази даних
- Локальне сховище файлів
- API ендпоінти
- Черга повідомлень
- Стріми(потокові дані)

Кожен підхід має свої переваги та недоліки. Оцінимо існуючі технології.

База даних

Переваги:

- Простота виборки, фільтрації даних за допомогою запитів.
- Вся логіка отримання чи запису даних на стороні БД, ми лише використовуємо готове рішення
- Можливість конкурентного доступу без ризику втрати чи дублювання даних.
- Зручність масштабування

Недоліки

- Вимагає постійного підключення до БД
- Для дуже великих обсягів даних можуть бути затримки
- Зростає складність резервного копіювання і відновлення при великих об’ємах даних.

Найкраще підходить для

Частого доступу до даних середнього об’єму, коли важлива можливість фільтрації, швидкої вибірки та транзікційності.

Локальні файли

Переваги:

- Простота реалізації, дані можна організувати у вигляді деревовидної структури
- Деякі біржі віддають історичні дані архівами і це найшвидший спосіб завантажити великий об’єм
- Висока швидкість читання для окремих форматів, для таких файлів оптимізовано послідовний та вибірковий доступ
- Незалежність від мережі – можна працювати офлайн

Недоліки

- Дублювання при використанні різними сервісами
- Відсутність механізмів одночасного доступу
- Складніше виконувати фільтрацію

Найкраще підходить для

Даних великого об’єму які потрібно швидко і часто зчитувати.

API-ендпоінти

Переваги

- Простота інтеграції, можна одразу отримувати актуальні дані після чого обробляти без необхідності збереження
- Зазвичай найпростіший(якщо не єдиний) спосіб отримати свіжі дані від бірж

- Не потребує власного збереження чи обслуговування інфраструктури для історії

Недоліки

- Обмеження кількості і частито запитів, що ускладнює завантаження великих обсягів історичних даних
- Залежність від стабільності інтернету
- Дані повертаються «порціями», тому для великих об’ємів будуть витрати на обробку
- Деякі біржі мають обмежену глибину історії для запитів через API

Найкраще підходить для

Отримання останніх оновлень, синхронізація нових свічок, роботи з малими або середніми обсягами даних.

Черга повідомлень

Переваги

- Асинхронність
- Масштабованість, можна з легкість відправляти дані на різні сервіси, конкретний сервіс, розподіляти навантаження
- Обробка даних в потоковому режимі
- Зменшення навантажень на основні сервіси, оскільки всі операції передачі і розподілення бере на себе брокер.
- Гарантована доставка(в залежності від брокера і налаштувань)

Недоліки

- Ускладнене налагодження і необхідність окремої інфраструктури для брокера
- Не призначення для довготривалого зберігання

Найбільше підходить для

Потокових даних які потрібно розділити між різними сервісами.

Стріми(потокові дані)

Стрімінг є ключовим способом отримання актуальних ринкових даних в режимі реального часу

Переваги:

- Найактуальніші дані(за останню годину, хвилину, секунду)
- Мінімальна затримка
- Можливість інтеграції з чергами повідомлень

Недоліки

- Відсутність історичних даних, лише дані в реальному часі
- Можливе накопичення дрібних повідомлень
- Необхідність обробки out-of-order подій, коли дані можуть надходити з пропусками або не в певному порядку(наприклад, порядок пар)

Найкраще підходить для

Отримання останніх даних для торгівлі. Прийняття рішень для торгівлі.

Висновок

Різні технології мають свої переваги та недоліки. Найкращого результату не можна досягти використовуючи лише щось одне, для цього потрібно вдало комбінувати різні підходи, щоб отримувати якомога більше переваг і щоб недоліки були якомога менш помітними.

3.4 Порівняння БД

Порівняємо найбільш релевантні БД для трейдингу.

- PostgreSQL
- TimescaleDB
- ClickHouse
- MongoDB
- SQLite
- QuestDB

PostgreSQL

PostgreSQL – потужна open-source реляційна БД, яка активно розробляється більше 35 років. Відома своєю стабільністю та відповідністю SQL стандартам.

Була створена для забезпечення гнучкої, розширюваної СУБД з підтримкою складних типів даних, а також з можливістю створення користувацьких функцій.

Переваги

- Підтримка складного SQL, індексів, агрегацій
- Транзакційність і цілісність даних
- Дуже висока стабільність і зрілість технології
- Проста інтеграція

Недоліки

- Не оптимізована для часових рядів
- При дуже великих таблицях падає продуктивність
- Обмежене горизонтальне масштабування

Оцінка

- Великі об’єми: середньо.
- Середні об’єми: відмінно.
- Частий доступ: добре.

- Рідкий доступ: добре.

TimescaleDB

TimescaleDB – це розширення PostgreSQL для високопродуктивної аналітики часових рядів в режимі реального часу.

Була створена для роботи з часовими рядами, для швидкого отримання таких даних і обробки у великих об’ємах.

Як зазначено на github розробника – зменшує розмір часових рядів і даних подій більш ніж на 90%.

Переваги

- Оптимізована для роботи з часовими рядами
- Підтримка SQL
- Покращена швидкість вибірок порівняно з PostgreSQL
- Можливість компресії даних

Недоліки

- Складніша конфігурація
- Частина функцій доступна лише в комерційній редакції
- Не підходить для екстремальних навантажень

Оцінка

Великі об’єми: добре.

Середні об’єми: відмінно.

Частий доступ: відмінно.

Рідкий доступ: добре.

ClickHouse

ClickHouse – це аналітична база даних з дуже високою швидкістю обробки.

Була створена як СУБД, здатна виконувати важкі аналітичні запити по мільярдах рядків за секунди з мінімальним споживанням ресурсів.

Переваги

- Колонкове зберігання – читає лише необхідні колонки
- Дуже велика швидкість для великих вибірок
- Добре стискає дані

Недоліки

- Розроблена компанією Яндекс
- Не підходить для частих одиничних вставок
- Складна конфігурація та архітектура

Оцінка

Великі об’єми: відмінно.

Середні об’єми: задовільно.

Частий доступ: добре.

Рідкий доступ: добре.

MongoDB

MongoDB – документоорієнтована NoSQL система керування базами даних, в якій дані зберігаються у форматі JSON-подібних документів.

Була створена для випадків коли структура змінюється з часом і для сценаріїв коли важливі масштабування та швидкість розробки.

Переваги

- Відсутність жорсткої схеми
- Просте горизонтальне масштабування
- Гарно працює з вкладеними структурами
- Швидкі вставки документів

Недоліки

- Гірша продуктивність для аналітики
- Займає більше пам’яті

Оцінка

Великі об’єми: погано.

Середні об’єми: добре.

Частий доступ: добре.

Рідкий доступ: добре.

SQLite

SQLite – це легка вбудована СУЬД яка працює як один файл без сервера. Часто використовується в моб застосунках або в невеликих локальних системах.

Була створена для простого способу зберігання структурованих даних без сервера БД.

Переваги

- Простота використання
- Мінімальна конфігурація
- Працює в файлі

Недоліки

- Не витримує високих навантажень
- Проблеми з одночасним доступом
- Обмежена продуктивність при великих таблицях

Оцінка

Великі об’єми: дуже погано.

Середні об’єми: погано.

Частий доступ:задовільно.

Рідкий доступ: добре.

QuestDB

QuestDB – високопродуктивна колонкова БД, оптимізована для фінансових потоків та IoT. Має SQL-подібний синтаксис.

Була створена як СУБД, що поєднує ефективну аналітику по часових інтервалах зі швидкістю потокового запису.

Переваги

- Підтримка SQL
- Колонкове зберігання і високий рівень стиснення
- Природна оптимізація під timestamp

Недоліки

- Менша екосистема ніж у Postgres
- Складніша робота з великими JSON

Оцінка

Великі об’єми: дуже добре.

Середні об’єми: відмінно.

Частий доступ: добре.

Рідкий доступ: добре.

3.5 Порівняння форматів файлів

Порівняємо формати файлів, для зберігання ринкових даних. А саме

- CSV
- JSON
- Parquet
- Feather
- HDF5
- Zstd/gzip

CSV

CSV – це простий текстовий формат, де кожен рядок містить значення, розділені певним символом(зазвичай комою)

Був створений для універсального переносу табличних даних між різними системами.

Біржа Binance зберігає свої історичні дані саме в цьому форматі.

Переваги

- Універсальність. Можна читати в будь-якому середовищі, будь-якою мовою/бібліотекою
- Це текстовий формат, тому легко читається людиною
- Легкий для налагодження

Недоліки

- Великий розмір файлу
- Повільне читання для великих наборів
- Відсутність типізації даних
- Не призначений для ефективних фінансових історій

Оцінка

Великі об’єми: погано

Середні об’єми: задовільно

Частий доступ: погано

Рідкий доступ: нормально

JSON

JSON – текстовий формат для зберігання структурованих даних у вигляді вкладених об’єктів.

Був створений для обміну структурованими даними між веб-сервісами

Переваги

- Зберігає вкладені структури
- Може читатись людиною
- Велике поширення у вебі

Недоліки

- Великий розмір(особливо коли багато даних з однією структурою – велика кількість дублювання ключів)
- Повільне читання

Оцінка

Великі об’єми: погано

Середні об’єми: задовільно

Частий доступ: погано

Рідкий доступ: добре

Parquet

Parquet – формат файлів, орієнтований на ефективне зберігання і отримання даних з відкритим кодом.

Був створений для швидкого читання фрагментів у великих наборах, де потрібно читати лише частину колонок

Переваги

- Колонкове зберігання → зчитує тільки потрібні колонки
- Дуже гарне стиснення
- Швидке читання
- Наявність фільтрації
- Можливість розбиття одного файлу на деревовидну структуру директорій з файлами з фільтрами
- Швидке читання у pandas

Недоліки

- Оптимізований під читання, а не під запис

Оцінка

Великі об’єми: відмінно

Середні об’єми: добре

Частий доступ: дуже добре

Рідкий доступ: відмінно

Feather

Feather - бінарний формат на основі Apache Arrow, оптимізований для дуже швидкого зчитування у pandas.

Був створений для передачі даних між мовами та фреймворками.

Переваги

- Швидке читання у pandas
- Баланс розміру та швидкості
- Мінімальні накладні витрати

Недоліки

- Гірше стиснення ніж у parquet

Оцінка

Великі об’єми: добре

Середні об’єми: відмінно

Частий доступ: відмінно

Рідкий доступ: дуже добре

HDF5

HDF5 – формат для зберігання великих масивів даних, який використовується в наукових обчисленнях в ML.

Переваги

- Швидкий доступ до частини даних
- Підтримка груп, дерев, складних структур

Недоліки

- Складний формат
- Більша ймовірність пошкодження файлу при збоях

Оцінка

Великі об’єми: дуже добре

Середні об’єми: добре

Частий доступ: добре

Рідкий доступ: добре

Zstd/gzip

Zstd/gzip - це не формат таблиці, а спосіб стискання файлів.

Бінанс саме так і зберігає свої csv файли.

Переваги

- Суттєве зменшення розміру файлів CSV/JSON. Для CSV стискання зазвичай зменшує розмір у 5–10 разів.
- Зменшує навантаження на дисковий простір, особливо при довготривалому зберіганні (5+ років історії).
- Добре підходить для передачі по мережі, бо трафік менший.

Недоліки

- Щоб зчитати файл потрібно його розпакувати
- Гірше працює з уже стиснутими форматами, наприклад Parquet. Додатковий виграш у розмірі – мінімальний (<5%).
- Час доступу збільшується, особливо для великих архівів (по кілька ГБ).

Таким чином для довгостроковго зберігання чи передачі файлів між сервісами стиснення може бути гарним рішенням, але не завжди, оскільки деякі формати «з коробки» мають стиснення.

Висновок

При виборі формату файлу варто враховувати як технічні характеристики, функціональені можливості так і призначення. Наприклад, Binance зберігає файли в не дуже оптимізованому форматі – CSV, але це дає можливість легко його читати з будь-якого середовища. Можна навіть просто відкрити в текстовому редакторі. Але при цьому щоб зменшити розмір – ці файли архівуються. Це гарний приклад збалансованого рішення.

3.6 Порівняння брокерів повідомлень

Брокери повідомлень зручні для потокових даних, тому порівняємо найпопулярніші з них:

- RabbitMQ
- Kafka
- Redis Streams
- NATS / NATS JetStream
- ZeroMQ (не зовсім брокер, але дуже популярний у трейдингу)

RabbitMQ

RabbitMQ – класичний брокер повідомлень, що використовує протокол AMQP.

Був розроблений для забезпечення гарантованої доставки, архітектури черг та складної доставки повідомлень.

Функціональні можливості

- Гарантована доставка повідомлень
- Виконання подій в правильному порядку
- Повторна доставка при збої
- Queuing, exchange, routing keys, topic routing

Переваги

- Зріла і надійна технологія
- Гнучкі патерни маршрутизації
- Гарна документація

Недоліки

- Низька пропускна здатність в порівнянні з іншими
- Вища затримка

Оцінки

- Швидкість: 50-150 тис. пов./с
- Великі об’єми: погано
- Середні об’єми: добре
- Live-stream: частково підходить для сигналів, але не order-book.

Kafka

Kafka – розподілений брокер подій, побудований як лог потоку. Створений компанією LinkedIn для обробки великих потоків даних, де традиційні черги не справлялися. Оптимізований під Big Data та стрімінг в реальному часі.

Функціональні можливості

- Паралельне читання
- Повторне «програвання» історії
- Розподіл обробки між багатьма сервісами

Переваги

- Найвища пропускна здатність серед брокерів
- Горизонтальне масштабування

Недоліки

- Важка інфраструктура
- Більша затримка в порівнянні з NATS, ZeroMQ

Оцінки

- Швидкість: понад 1млн пов./с

- Великі об’єми: відмінно
- Середні об’єми: відмінно
- Live-stream: найкращий варіант

Redis Streams

Redis Streams – розширення Redis, що додає потоки подій, збереження в лог, групи споживачів.

Був створений для дуже швидкого отримання і відправлення повідомлень з низькою затримкою

Функціональні можливості

- Швидка реакція без складної інфраструктури
- Повність im-memory обробка

Переваги

- Висока швидкість запису
- Низька затримка
- Простий API
- Універсальність
- Оптимальний для real-time логіки

Недоліки

- Погано підходить для великих обсягів
- Не зберігає довгострокові дані
- Менша надійність при збоях(якщо не налаштовані snapshot/replica)

Оцінки

- Швидкість: 300-600 тис. пов./с
- Великі об’єми: задовільно
- Середні об’єми: відмінно
- Live-stream: дуже добре

NATS / NATS JetStream

NATS – надшвидка система для низьколатентних повідомлень.

Був створений для миттєвої реакції в мікросервісах.

Функціональні можливості

- Автоматичне горизонтальне масштабуваня
- Проста модель pub/sub

Переваги

- Дуже низька затримка
- Висока швидкеість
- Проста інфраструктура

Недоліки

- Невелика кількість функцій і інструментів
- Підходить не для всіх типів зберігання потоків

Оцінки

- Швидкість: > 1 млн пов/с
- Великі об’єми: добрк
- Середні об’єми: відмінно
- Live-stream: чудово

ZeroMQ

ZeroMQ – надшвидкий транспортний рівень для обміну повідомленнями між процесами.

Створено для мінімальної латентності та максимальної продуктивності.

Функціональні можливості

- Пряме P2P з’єднання
- Передача подій без центрального сервера

Переваги

- Найбільша швидкість серед усіх систем
- Дуже низька затримка
- Легкий протокол

Недоліки

- Відсутність централізованого маршрутизатора
- Потрібно вручну описувати поведінку в випадку збоїв

Оцінки

- Швидкість: 2-3 млн пов/с
- Великі об’єми: погано
- Середні об’єми: добре, якщо локально
- Live-stream: чудово для HFT, але не для розподіленої системи.

Висновок

Швидкість передачі повідомлень між сервісами грає велику роль в системах для автоматизованої торгівлі, однак обирати брокер лише за цим показником – помилка. Не менш важливою є масштабованість, яка визначає на скільки система може обробляти зростаючий обсяг даних. Гнучкість також є ключовим фактором: різні брокери пропонують різні моделі доставки, типи маршрутизації та механізми роботи зі стрімами. Крім того слід враховувати надійність системи, доступність механізмів відновлення і простота супроводу.

Отже вибір брокера повідомлень – це компроміс між продуктивністю, надійністю, гнучкістю і простотою підтримки.

3.7 Порівняння мов програмування

Порівняємо найрелевантніші мови:

- Python
- Go
- Rust
- Java
- C++

Python

Python –високорівнева мова з акцентом на читабельність коду. Спочатку створювалась як мові для швидкої розробки та автоматизації.

Переваги

- Найбільша кількість бібліотек для роботи з даними
- Швидкий розвиток
- Є потужні бібліотеки для бектесту

Недоліки

- Нижча продуктивність ніж в Go
- GIL – обмеження багатопоточності

Оцінки

Швидкість: Середня

Великі об’єми: добре

Простота розробки: відмінно

Go

Go – мова від Google, створена для високопродуктивних мереж і розподілених систем.

Створена для мікросервісів, високонавантажених бекендів.

Переваги

- Дуже швидкі та легкі корутини(мільйони потоків)
- Стабільна продуктивність
- Низька затримка

Недоліки

- Слабша екосистема для роботи з даними
- Обмежені можливості для складних обчислень
- Відсутня гнучкість як у Python

Оцінки

Швидкість: висока

Великі об’єми: добре

Простота розробки: добре

Rust

Rust – сучасна системна мова з упором на безпеку пам’яті без garbage collector.

Була створена для високопродуктивних сервісів, системного програмування, як безпечна альтернатива C++.

Переваги

- Дуже висока продуктивність
- Низька затримка

Недоліки

- Висока складність
- Більший час розробки
- Менша кількість бібліотек для роботи з даними

Оцінки

Швидкість: дуже висока

Великі об’єми: відмінно

Простота розробки: низька

Java

Java – одна з найпопулярніших корпоративних мов.

Переваги

- Висока стабільність
- JVM – висока продуктивність і хороша оптимізація

Недоліки

- Довгий час розробки
- Громіздкий стек

Оцінки

Швидкість: висока

Великі об’єми: дуже добре

Простота розробки: середня

C++

C++ - системна мова з максимальною продуктивністю. Широко використовується в HFT.

Переваги

- Найвища продуктивність серед усіх вказаних мов
- Прямий контроль над пам’яттю

Недоліки

- Довга розробка
- Високий рівень помилок(memory leaks)
- Велика складність коду

Оцінки

Швидкість: найвища

Великі об’єми: відмінно

Простота розробки: низька

3.8 Порівняння бірж

Порівняємо найпопулярніші біржі які існують на сьогодні:

- Binance
- Bybit
- OKX
- KuCoin
- Kraken
- Coinbase Advanced

Для кожної ключові аспекти:

- Якість та доступність історичних даних
- Якість та швидкість API
- Технічні обмеження
- Переваги/недоліки

Binance

Binance

Історичні дані

Binance Vision – найкращий і найбільш повний архів історичних даних.

Вони надають:

- Свічкові дані за різними таймфреймами по великій кількості пар
- Futures
- Option

Якість: відмінна

API для реальної торгівлі

Binance має

- WebSocket з реалтайм даними з низькою затримкою
- Стабільні стріми
- REST API з останніми даними
- Потужну документацію
- Найбільшу ліквідність на ринку

Якість: відмінно.

Переваги

- Найкраща якість історичних даних серед усії
- Потужний WebSocket
- Велика ліквідність

- Зручна структура файлів
- Продуманий API

Недоліки

- Висока конкуренція ботів
- Ліміти на websocket
- Ліміти на кількість саб-акаунтів(для реальної торгівлі)

Bybit

Bybit

Історичні дані

Історичні дані цієї біржі теж є, але в меншому обсязі, з гіршою документацією і менш структуровані.

Якість: гарна

API для реальної торгівлі

Має гарні стріми WebSocket, але

- Інколи більша затримка
- Інколи бувають ліміти
- Futures API стабільний, Spot API гірший

Якість: гарна.

Переваги

- популярна серед трейдерів
- зручний API для ф'ючерсів
- хороший WebSocket

Недоліки

- історичні дані не повні

- менша ліквідність
- складність у симуляції складних сценаріїв

OKX

OKX

Історичні дані

OKX має дуже якісні дані, але вони:

- не такі зручні у завантаженні
- не такі структуровані
- частково платні

Якість: непогана

API для реальної торгівлі

OKX має дуже стабільні та швидкі WebSocket-и.

Фактично це друга за якістю інфраструктура після Binance.

Якість: гарна, але лише для даних в реальному часі

Переваги

- хороша ліквідність
- стабільні стріми
- прозора документація
- низька затримка

Недоліки

- історичні дані не такі структуровані
- заплутана специфікація futures/spot

KuCoin

KuCoin

Історичні дані

Історичні дані є, але:

- неповні
- інколи з пропусками
- немає order book updates за всі періоди

Для серйозних бектестів не рекомендується.

Якість: Низька

API для реальної торгівлі

- часті rate limits
- нестабільні WebSocket у пікові моменти
- робота з ордерами добра, але не найшвидша

Якість: непогана

Переваги

- проста інтеграція
- низькі вимоги KYC
- універсальна документація

Недоліки

- низька ліквідність
- гірша якість стрімів
- історичні дані слабкі

Kraken

Kraken

Історичні дані

Є архіви, але вони часто:

- з затримками
- з пропусками
- з меншою деталізацією

Якість: погана

API для реальної торгівлі

- менш оптивізований WebSocket
- Повільніший REST

Якість: погана

Переваги

- Безпека
- Стабільність

Недоліки

- Низька ліквідність
- Слабкі історичні дані

Може використовуватись як додаткова біржа, але не як основна

Coinbase Advanced

Coinbase Advanced

Історичні дані

- order book історії немає
- ticks доступні не завжди

- API сильно урізане

Якість: погана

API для реальної торгівлі

- стабільний
- стандартизований
- безпечний
- але не найшвидший

Якість: нормальна

Переваги

- регульована біржа
- безпечна

Недоліки

- дуже мало історичних даних
- низька ліквідність для altcoins
- API слабший за Binance/OKX/Bybit

Висновок

Проаналізувавши всі біржі, для початку розробки архітектури підійде біржі Binance, по наступних причинах:

- Найбільша доступна кількість історичних даних
- Великі обсяги торгів
- Потужне API для торговлі та отримання історичних даних

3.9 Огляд біржі Binance

Binance Vision

Binance Vision – це архів історичних даних. Але це не API, а звичайний сайт. Тому щоб завантажити історичні дані програмно – потрібно дослідити як можна це зробити.

Спочатку знайдемо необхідні дані через графічний інтерфейс, а потім розберемось як дістати їх з програми.

Заходимо на https://data.binance.vision/

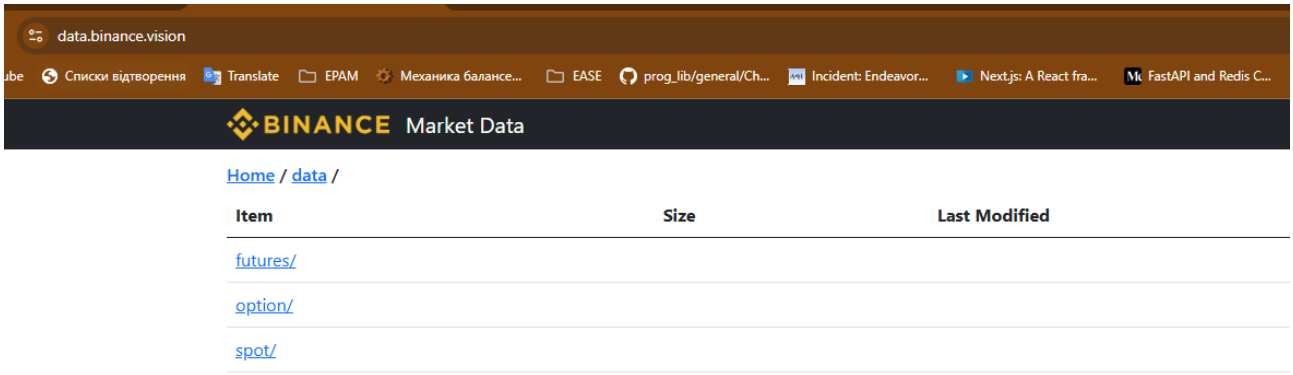


Рис .3.9.1 – головна сторінка сайту

Бачимо 3 директорії futures, options, spot.

Нас цікавлять спотові дані, тому заходимо в директорію spot.

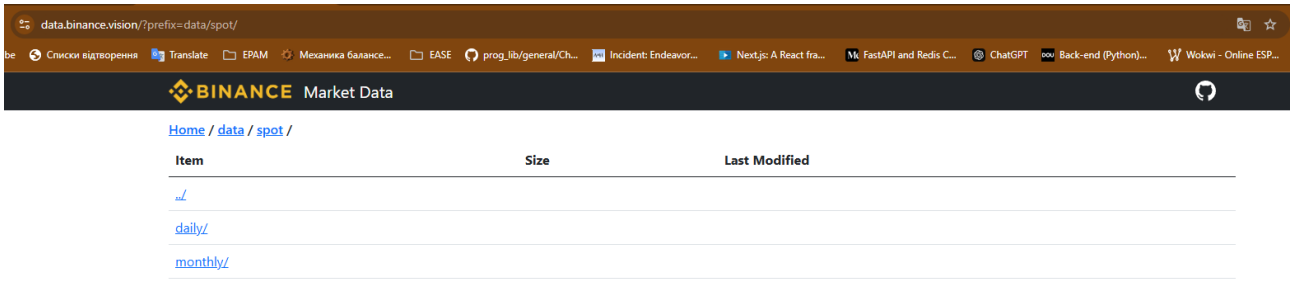


Рис .3.9.2 – структура data/spot

Тепер бачимо дві директорії daily та monthly.

Daily – в одному файлі будуть дані за 1 день

Monthly – в одному файлі дані за 1 місяць

Також помічаємо як змінилось посилання в адресному рядку: https://data.binance.vision/?prefix=data/spot/

Переходимо в daily(для monthly всередині все буде аналогічно).

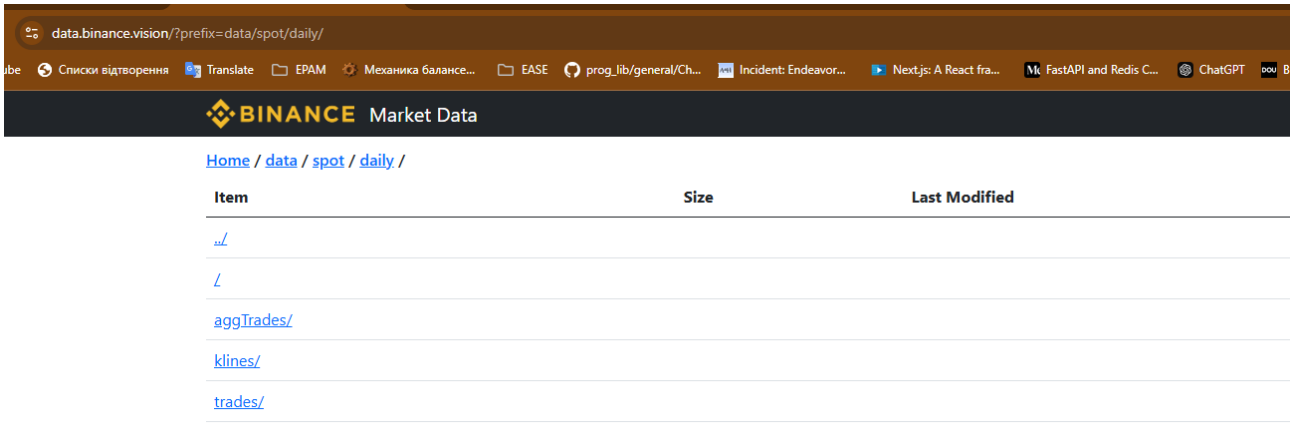


Рис .3.9.3 – структура data/spot/daily

url: https://data.binance.vision/?prefix=data/spot/daily/

Бачимо aggTrades, klines, trades.

Це різні дані, нас цікавлять свічкові дані, тому ми переходимо в klines/.

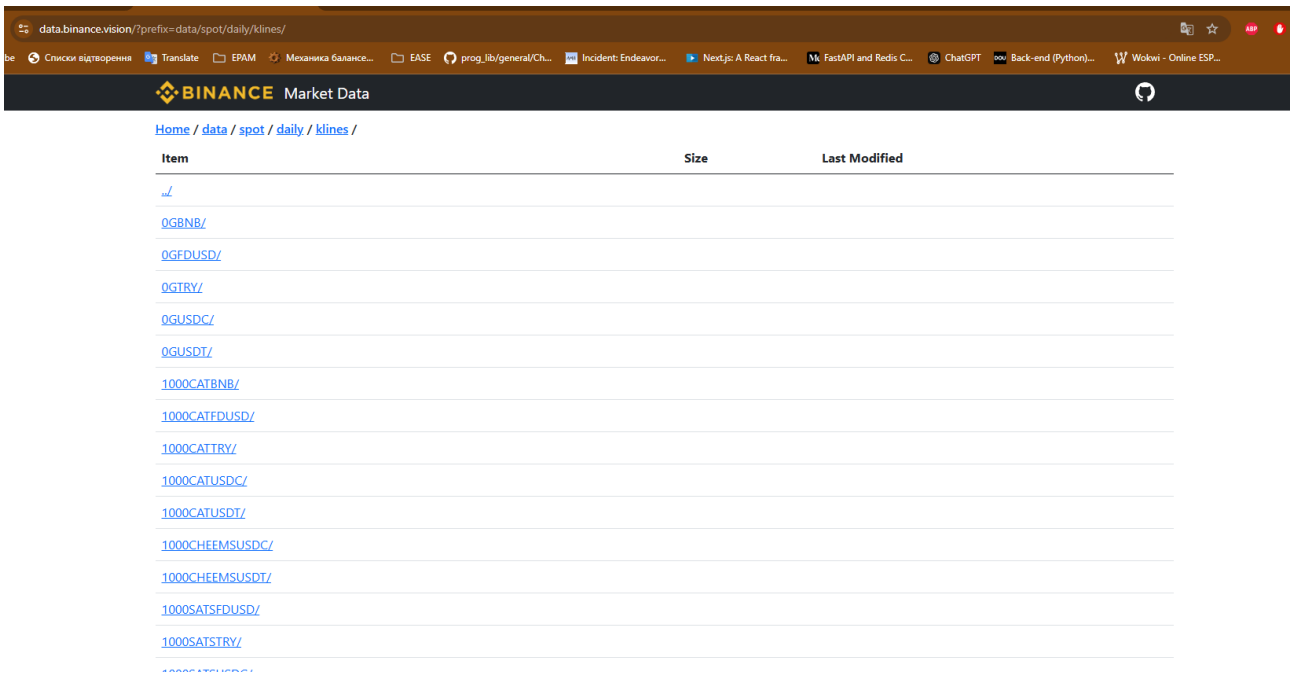


Рис .3.9.4 – структура data/spot/daily/klines

url: https://data.binance.vision/?prefix=data/spot/daily/klines/

Тут вже бачимо багато директорій.

Кожна з них – це торгова пара, перейдемо в BTCUDT

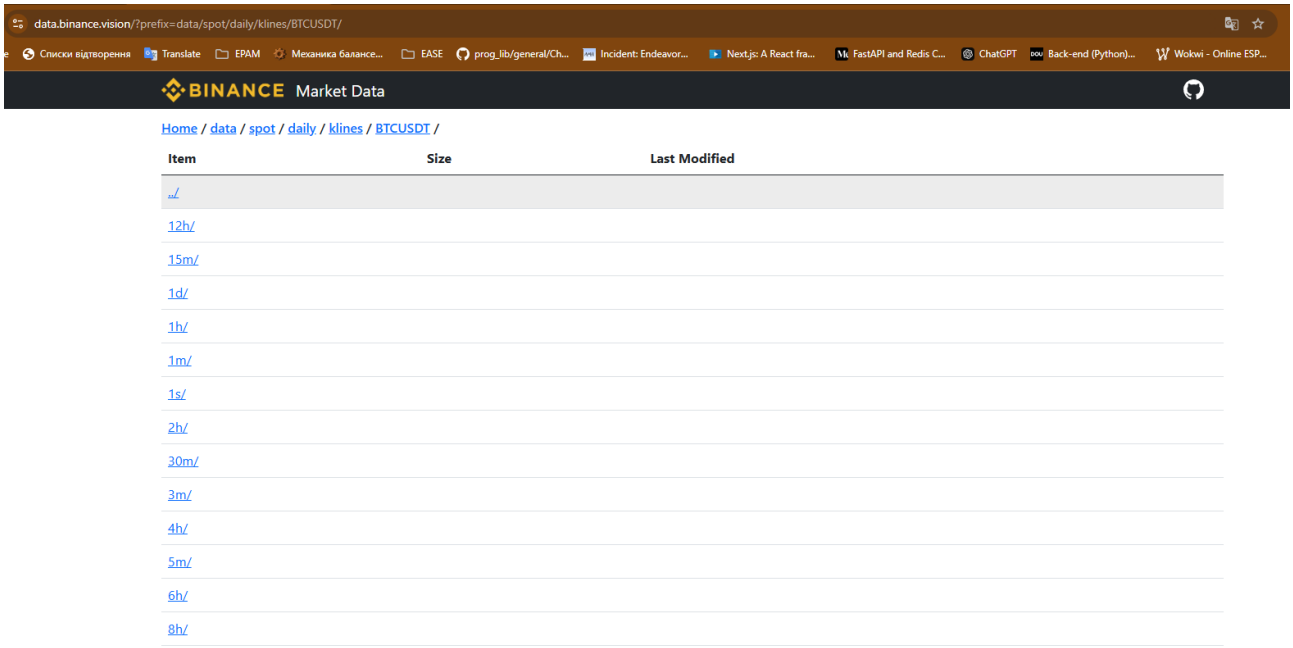


Рис .3.9.5 – структура data/spot/daily/klines/BTCUSDT

url: https://data.binance.vision/?prefix=data/spot/daily/klines/BTCUSDT/1m/

Тепер бачимо директорії з часовими інтервалами. Кожен часовий інтервал вказано для однієї свічки. Наприклад, 1m – означає що одна свічка вміщує дані за одну хвилину.

Переходимо в 1m/

url: https://data.binance.vision/?prefix=data/spot/daily/klines/BTCUSDT/1m/

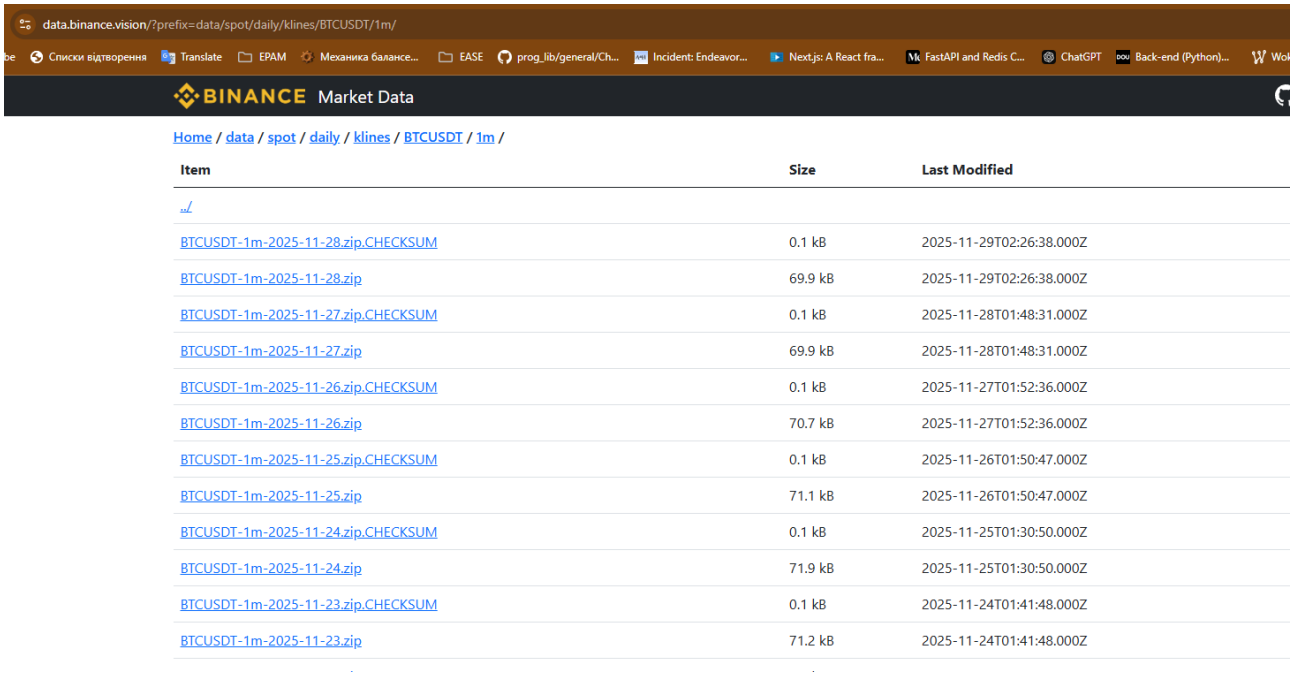


Рис .3.9.6 – структура data/spot/daily/klines/BTCUSDT/1m

Бачимо файли .zip та .zip.CHECKSUM

Zip – архів з csv файлом з даними

CHECKSUM – контрольна сума, можна використовувати для перевірки цілісності файлів.

Завантажимо zip файл, розпакуємо і перевіримо вміст

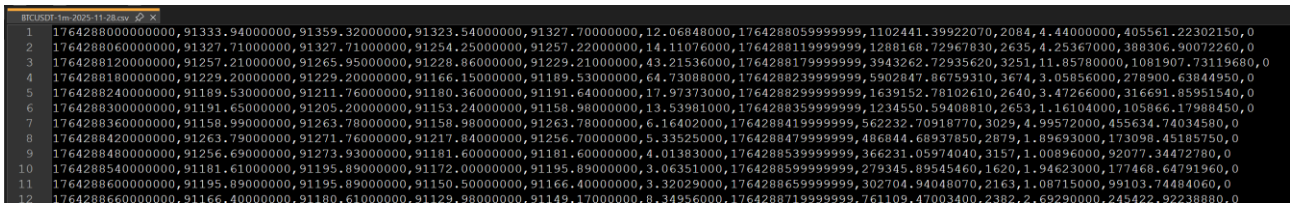


Рис .3.9.7 – вміст csv файлу

Бачимо значення, розділені комою

Перейшовши на гітхаб binance, можна знайти назву колонок

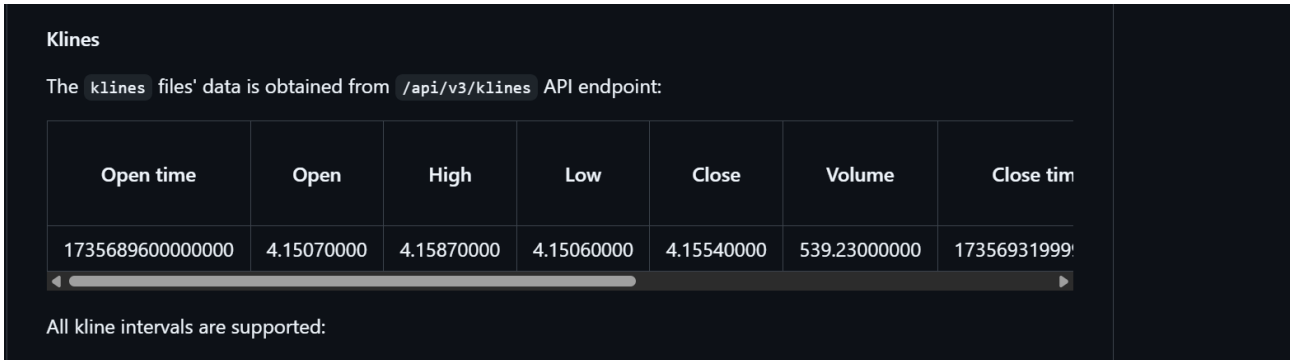


Рис .3.9.7 – опис даних в csv файлі

Отже, в файлі маємо значення, розділені комою в такому порядку

Open time – час відкриття свічки

Open – ціна відкриття свічки

High – найвища ціна за період свічки

Low – найнижча ціна за період свічки

Close – ціна закриття свічки

Volume – обсяг проторгованого базового активу(BTCUSDT – BTC)

Close Time – час закриття свічки

Quote asset volume – обсяг торгів у котируваному активі(BTCUSDT – USDT_

Number of trades – кількість угод за цей час

Taker buy base asset volume – показує скільки базового активу було куплено

Taker buy quote asset volume – скільки котируваного активу було витрачено

Ignore – колонка, яку можна ігнорувати, залишилась для зворотньої сумісності

Тепер розберемось які запити відбуваються щоб отримати ті чи інші дані.

Натискаємо F12 в браузері, обираємо вкладку Network і переходимо на https://data.binance.vision/?prefix=data/

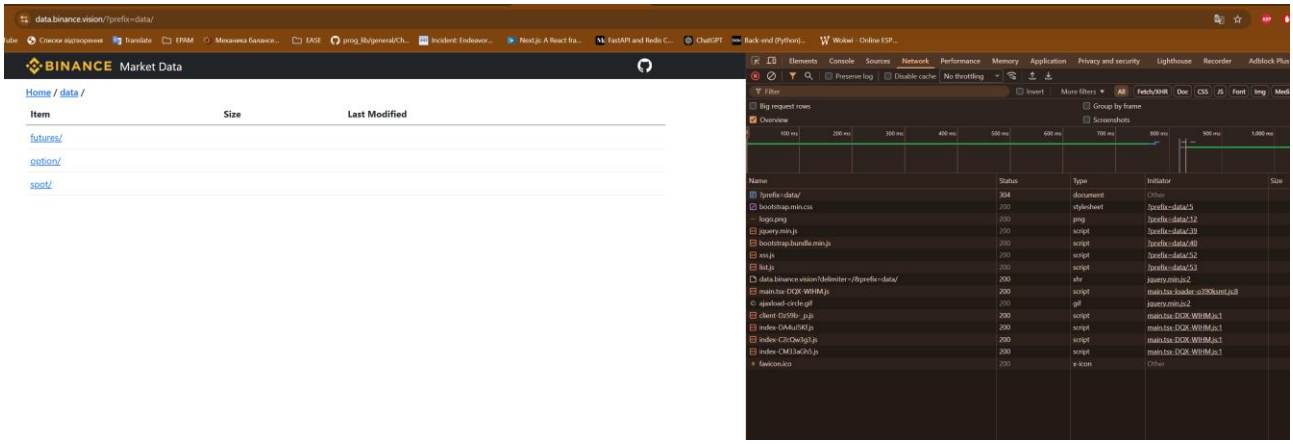


Рис .3.9.8 – запити, які робить binance vision

Знаходимо запит на

https://s3-ap-northeast-1.amazonaws.com/data.binance.vision?delimiter=/&prefix=data/

в відповіді якого лежить XML.

Проаналізувавши який можна зрозуміти, що тут і повертаються «наступні шляхи», де вказані наступні директорії (spotm fitires, options).

Проаналізувавши інші сторінки, можна зрозуміти що така логіка працює для всіх вкладок. Єдина різниця – остання сторінка з файлами, самі файли лежать в інший секції, але формат той самий.

Таким чином, щоб отримати доступні активи, таймфрейми по активу тощо потрібно зробити дві речі:

1) Побудувати «префікс»
2) Зробити запит на https://s3-ap-northeast-1.amazonaws.com/data.binance.vision, передавши правильний префікс

Таким чином ми отримаємо всі необхідні дані для отримання доступних опцій, або посилання для завантаження файлів.

Binance API

При дослідженні Binance API було виявлено, що бінанс надає інтерфейс, якого повністю достатньо для торговлі в реальному часі:

1) Websocket – отримання ціни в реальному часу
2) Ендпоінти
    - GET /api/v3/ticker/bookTicker — ціна
    - GET /api/v3/account — баланс
    - POST /api/v3/order — створити ордер
    - GET /api/v3/order — перевірити статус

Для базової торговлі цього більш ніж достатньо.

Бінанс має певні ліміти на запити, а також на streams. Тому потрібно це враховувати.

Також, для торговлі API потрібно створювати саб-акаунти

Саб-акаунт – це акаунт, який можна створити з основного акаунта, використовувати в торгівлі, він має власний баланс, не залежний від основного акаунту. Кількість саб-акаунтів обмежена і залежить від рівня акаунта Binance.

Відповідно, щоб ізолювати баланс кожної стратегії – потрібно буде самому створити логіку керування балансом, слід це враховувати.

4. Розробка архітектури

4.1Обрані технології

Якщо коротко, вся архітектура має виконувати дві основні задачі:
1. Бектестинг стратегій

2. Реальна автоматична торгівля

Для початку потрібен бектест, оскільки перед запуском реальної стратегії її потрібно протестувати.

Група сервісів, пов’язаних з бектестом має виконувати задачі:

- Завантаження історичних даних
- Симуляція торгівлі на історичних даних
- Підрахунок та збереження результатів і метрик

Група сервісів, пов’язаних з торгівлею має виконувати задачі:

- Збереження і обробка реалтайм даних
- Торгівля
- Зберігання трейдів
- Отримання і зберігання історичних даних за останній період

Архітектура

Проаналізувавші всю раніше описану інформацію і не тільки, я спроектував наступну архітектуру(схема на малюнку).

Я обрав:

Мова програмування - Python

Насправді, гарних кандидатів є 2 – Go та Python, але Python має набагато більше готових бібліотек для задач, пов’язаних з трейдингом та обчисленнями. Хоча Go є більш швидкою мовою, та для поточних задач вистачить Python, при потребі можна буде окремі сервіси переписати на Go. Але початкова реалізація на Python, оскільки він має бібліотеки, які написані на C для роботи з даними, а також розробка буде значно швидшою ніж на Go.

Брокер повідомлень - Kafka

Оскільки він має найбільшу пропускну здатність і функціоналу який він надає більш ніж достатньо. Нам не потрібна велика кількість способів маршрутизацій як у Rabbit, чи інші функції. Тому це найкращий варіант.

База даних – PostgreSQL

Вона була обрана через свою надійність, стабільність і універсальність. Хоча й існують СУБД, які більше підходять для часових даних, я обрав постгрес, оскільки в моїх архітектурі немає необхідності в швидкісній БД, яка забере на себе частину обчислень. Мені важлива надійність, щоб не втратити дані, а також підтримка різних форматів. Тому Postgres – найкращий варіант.

Також використовуватиметься **Redis** для збереження стану стратегії(наприклад для холодного старту) або обміну даних між декількома стратегіями.

Сервіси:

CandleProcessor – сервіс, який слухатиме потоки даних з бінанс і кластиме в Kafka. Таким чином велика кількість сервісів зможе підписатися на певні дані у Kafka і ми не залежатимемо від лімітів бінанс.

FeatureCalculator – сервіс, який на основі даних з Kafka рахуватиме певні індикатори і кластиме ці дані назад в Kafka(для інших сервісів).

DBSaver – зберігатиме дані з Kafka в БД для довгострокового зберігання.

StrategyManager – відповідатиме за запуск StrategyRunner. Автоматично масштабуватиме, передаватиме конфіг для запуску, керуватиме під час роботи.

StrategyRunner – сервіс, який запускатиме «стратегії». Його задача на основі вхідних даниї(з Kafka) прийняти рішення(купівля, продаж, нічого не робити).

TradingEngine – відповідатиме за виконання трейдів, також керуватиме бюджетом, розбиватиме один саб-акаунт бінанс на багато внутрішніх акаунтів. Таким чином нам не заважатиме ліміт по сабакаунтах і ми зможемо виділяти окремий ізольований бюджет для кожної стратегії.

BTManager – по аналогії з StrategyManager, але для бектесту.

BTRunner – запускатиме бектести стратегій, зберігатиме результати, рахуватиме необхідні показники.

DataDownloader – завантажуватиме дані з binance.vision і зберігатиме в NFS(network file system).

RealTimeDataManager - по аналогії з StrategyManager для CandleProcessor та FeatureCalculator.

Також на схемі зображені:

Зовнішні ресурси:

Binance WS, Binance API – API бінанса, розділено оскільки дані з вебсокету ми використовуємо на початку пайплайна, а API в кінці для виконання трейдів.

Binance vision – архів історичних даних Binance.

Інфраструктурні компоненти:

Postgres – БД, в якій зберігатимуться всі довгострокові дані.

Kafka – брокер повідомлень, який використовуватиметься для передачі даних між сервісами(свічки, трейди, індикатори).

NFS – сховище для зберігання історичних даних в зручному форматі(parquet)

Redis – in-memory база даних для стратегій в реальному часі. Використовуватиметься для збереження стану, обміну даними між різними стратегіями.

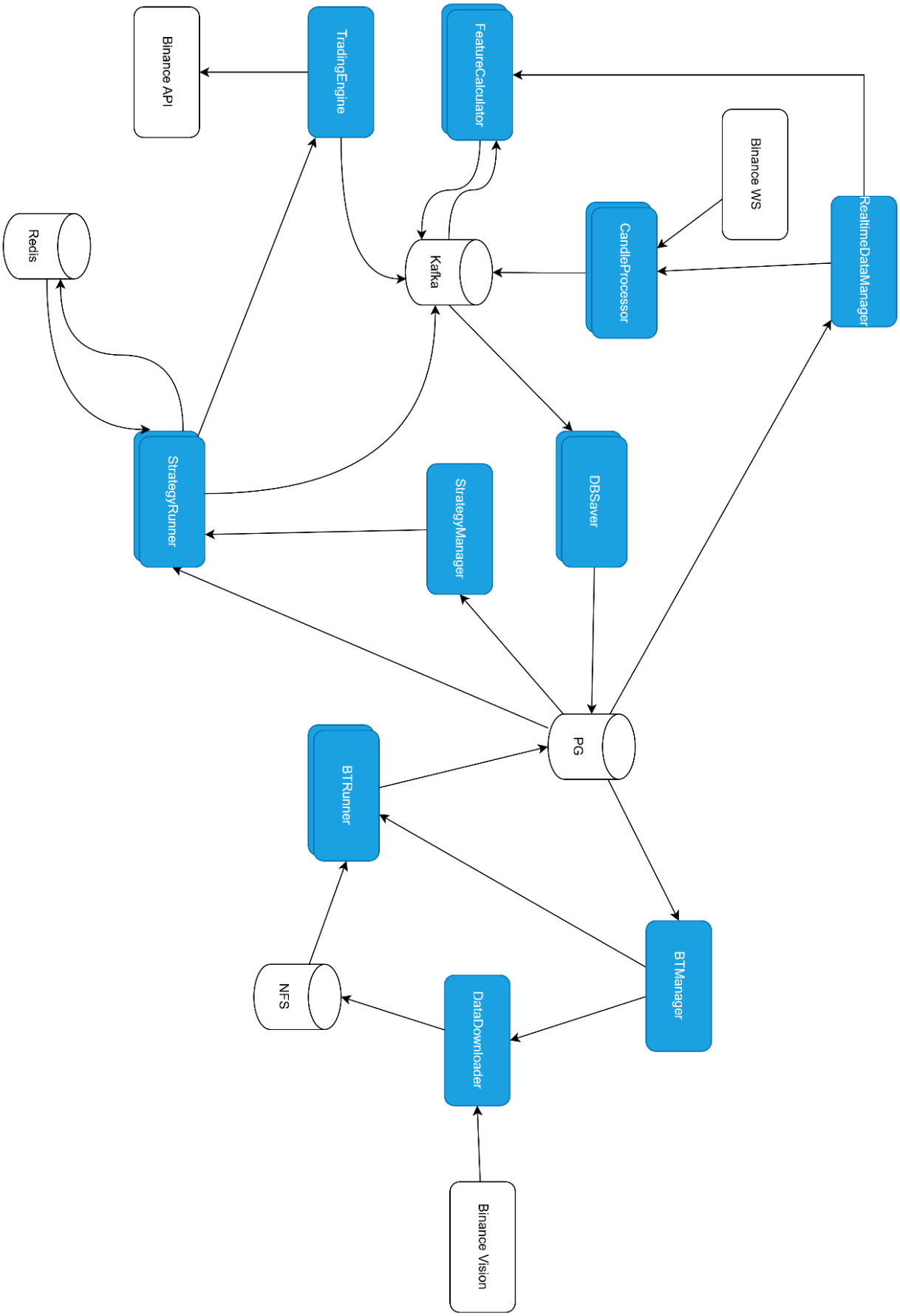


Рис. 4.1.1 - Архітектура мікросервісів для аналізу і торгівлі на криптовалютних біржах

4.2Детальний опис сервісів

CandleProcessor

CandleProcessor — це сервіс, який вирішує проблему з лімітами Binance на кількість одночасних потоків даних у реальному часі. Замість того, щоб кожен сервіс сам відкривав власний вебсокет, CandleProcessor один раз підключається до стріму Binance, слухає всі події і ретранслює їх у Kafka. Таким чином ми уникаємо дублювання підписок і зменшуємо навантаження.

Основні функції

Прийом даних з Binance (WebSocket)

Сервіс підписується на kline-стріми Binance та отримує "сирі" свічкові дані в реальному часі.

Уніфікація структури даних

Кожне kline-повідомлення приводиться до єдиного JSON-формату з такими ключами:

- open_time
- open
- high
- low
- close
- volume
- close_time
- quote_asset_volume
- number_of_trades
- taker_buy_base_asset_volume
- taker_buy_quote_asset_volume
- pair — поле, яке сервіс додає самостійно.

Тобто, у Kafka передається вся інформація, яка приходить по вебсокету, плюс ідентифікатор торгової пари.

Передача даних у Kafka

Дані з однієї біржі відправляються в окремий Kafka topic.

Наприклад, для Binance spot-свічок використовується topic binance_spot_klines.

Завдяки цьому при додаванні нової біржі достатньо створити для неї окремий topic і почати відправляти туди її свічки — логіка сервісів при цьому не змінюється.

Динамічне керування підписками

Сервіс повинен вміти додавати нові торгові пари до прослуховування або видаляти існуючі, не перезавантажуючись і не зупиняючи роботу.

API сервісу

CandleProcessor матиме мінімалістичне API для керування списком торгових пар:

1) POST /klines_pair
    - Додає нову пару до прослуховування.
    - Сервіс одразу підписується на відповідний вебсокет-канал Binance.

Параметри(JSON):

- pair
- timeframe

2) DELETE /klines_pair
    - Видаляє пару з активних підписок.
    - CandleProcessor відписується від стріму і припиняє слати повідомлення для цієї пари в Kafka.

Параметри(JSON):

- pair
- timeframe

FeatureCalculator

Цей сервіс відповідатиме за обрахунок індикаторів, які можуть перевикористовуватись в різних стратегіях. Він братиме дані з Kafka, обчислюватиме їх і кластиме назад в Kafka.

Кожен індикатор має наступні вхідні дані:

- топіки вхідних даних
- код для обчислення
- топік куди класти результат

Основна логіка роботи:

1) Читання даних з Kafka
2) Обчислення індикаторів
3) Публікація результатів в Kafka

Організація реплік та воркерів

Масштабування

Сервіс може мати довільну кількість реплік, залежно від навантаження та кількості індикаторів.

- Якщо індикаторів мало, або при обчисленні є однакові кроки — один воркер може виконувати всі.
- Якщо індикаторів багато — можна рознести їх між різними репліками.
- Якщо якесь джерело даних має надзвичайно великий обсяг подій — використовуються кілька воркерів для одного індикатора.

Групи Kafka consumer

Семантика груп визначає, як саме розподіляється навантаження:

- Всі репліки мають різні consumer groups — це потрібно, щоб вони незалежно обробляли повідомлення з однакових топіків і не заважали одна одній.
- Якщо один топік перевантажений, і потрібно кілька воркерів, щоб рахувати фічі швидше — такі воркери мають однакову consumer group, щоб ділити партиції між собою.

Таким чином:

- різні репліки = різні групи → незалежна обробка;
- воркери для одного й того ж індикатора = одна група → горизонтальне масштабування.

Закріплення індикаторів

Кожна репліка має свій список індикаторів, які вона рахує. Це дозволяє:

- ізолювати важкі обчислення на окремих репліках;
- зменшити конкуренцію за ресурси;
- чітко контролювати використання ресурсів і навантаження.

Назви всіх топіків з індикаторами(фічами) починаються на «features_» або «feature_».

RealTimeDataManager

Цей сервіс відповідатиме за масштабування, налаштування та керування сервісами FeatureCalculator та CandleProcessor. Він підніматиме нові інстанси, керуватиме конфігом, керуватиме цими сервісами, додаватиме репліки, розподілятиме «відповідальність» між репліками. Вмикатиме та вимикатиме потрібна пари, індикатори.

Цей сервіс потрібен лише для зменшення «ручної роботи», щоб не піднімати репліки і не прописувати їм конфіг щоразу руками, він це робитиме «по кліку».

Також RealTimeDataManager бере на себе логіку розподілу «відповідальності» між репліками. Наприклад, якщо потрібно обробляти 200 торгових пар, він може рівномірно розподілити їх між доступними CandleProcessor, щоб уникнути ситуацій, коли один інстанс переповнений, а інший — простоює. Те ж саме стосується і FeatureCalculator: різні індикатори можуть оброблятись у різних репліках, або одна репліка може відповідати лише за певний сегмент пар. Такий підхід дозволяє оптимізувати ресурси й підвищує загальну продуктивність.

Взаємодія з іншими сервісами:

Керуватиме FeatureCalculator та CandleProcessor

DBSaver

Задача цього сервісу – зберігати дані з Kafka в БД. В нього буде auto-scaling і в усіх реплік однакова група. Таким чином масштабування буде автоматизовано. Вони записуватимуть дані з потрібних топіків(буде налаштовуватись в конфізі) в БД для довгостроковго зберігання. Вони знатимуть «правила» запису в БД, які теж будуть задаватися в конфігі, звідти вони будуть брати інформацію які топіки слухати, які дані записувати, в яку таблицю тощо. DBSaver

DBSaver буде виконувати роль універсального сервісу для збереження потокових даних у постійному сховищі. На відміну від інших компонентів, які фокусуються на логіці стратегії, аналізу або підготовці даних, DBSaver не впливає на бізнес-логіку — він лише гарантує надійний, структурований і передбачуваний запис у базу даних. Його робота повинна бути стабільною, масштабованою та ізольованою від інших елементів системи, щоб великі обсяги даних не створювали вузького місця.

Для опису які дані і в якому форматі зберігати – буде розроблена спеціальна структура «конфігу».

Кожна репліка отримує конфігурацію, в якій описано:

- які Kafka-топіки потрібно слухати;
- які саме поля з повідомлень мають бути збережені;
- у яку таблицю бази даних їх записувати;
- який формат використовувати для розбору повідомлення;
- правила трансформації або нормалізації даних перед збереженням.

Взаємодія з іншими сервісами:

Взаємодіятиме з TradingEngine, FeatureCalculator, StrategyRunner через Kafka, зберігатиме ці дані в БД.

StrategyRunner

Цей сервіс відповідатиме за «прийняття рішень» про купівлю, продаж чи бездіяльність. Кожна репліка запускатиме певний код, в який входить отримання даних, обробка, підрахунок фіч(якщо їх нема в Kafka) і прийняття рішень.

Кожна стратегія складається з

- Коду
- Опису вхідних даних(топіки Kafka)
- Обчислення
- Алгоритм прийняття рішення.

Кожна стратегія автоматично зберігатиме свій стан в Redis щоб після перезапуску вона могла продовжити виконання з того ж місця, де закінчила. Також стратегії можуть обмінюватися даними якщо це потрібно(наприклад, якщо одна стратегія приймає рішення на основі рішень інших стратегій).

Коли результатом стратегії є транзакція – вона робить відповідний запит на TradingEngine, вказавши деталі транзакції, а також ім’я стратегії.

StrategyManager

Цей сервіс відповідатиме за масштабування **StrategyRunner.** Він вмітиме піднімати нові репліки, керувати, установлювати та змінювати конфіг, зупиняти виконання сервісів, видаляти делістнуті пари, розбивати одну стратегію на декілька под.

По суті, StrategyManager є “керівним центром” для всіх екземплярів StrategyRunner. Якщо StrategyRunner відповідає за безпосереднє виконання торгових стратегій (роботу з даними, генерацію сигналів, виставлення ордерів), то StrategyManager займається тим, щоб ці стратегії були запущені у правильній кількості, з правильними параметрами та на правильних торгових парах. Він не обчислює сигнали самостійно, але приймає рішення про те, скільки інстансів потрібно запустити, де їх запустити і з якими налаштуваннями.

Взаємодія з іншими сервісами:

Запускає інстанси StrategyRunner.

TradingEngine

TradingEngine — це сервіс, який відповідає за виконання торгових операцій, але не обмежується лише простим відправленням ордерів на біржу. Його задача — стати повноцінним “execution layer”, який забезпечує безпеку, контроль, фінансову ізоляцію та узгоджене керування балансами між стратегіями.

Функції сервісу:

- Виконання транзакцій

  ТЕ відправляє запити безпосередньо на біржу(наприклад, Binance) та контролює повний життєвий цикл ордера(надсилання, відслідковування, обробка результату, логування)

- Управління балансом

  Сервіс веде власний внутрішній облік балансу, контролює кошти для кожної стратегії, ліміти використання коштів тощо.

- Безпека транзакцій

  Перевірка, що запит не прийшов запізно, що ми не виконаємо трейд за поганою для нас ціною тощо

Взаємодія з іншими сервісами:

- StrategyExecutor генерує торгові сигнали;
- TradingEngine перевіряє, чи можна їх виконати, надсилає ордери, результат виконання повертає StrategyExecutor, а також кладе в Kafka(topic trades)
- DBSaver зберігає історію всіх операцій;

BTRunner

Цей сервіс відповідає за тестування стратегій на історичних даних. Він отримує параметри для запуску, завантажує потрібні історичні дані і починає «тестувати стратегію».

У процесі роботи BTRunner не просто проганяє стратегію по свічках — він відтворює повноцінний процес торгівлі, включаючи облік комісій, відкриття та закриття позицій, управління ризиками, проведення угод та реакцію стратегії на зміни ринку. Таким чином він фактично симулює реальну роботу торгової системи, але на вже відомих історичних даних, що дозволяє повністю оцінити поведінку стратегії у минулому.

Крім інформації про те, скільки стратегія заробила, сервіс також повертає дані про поведінку балансу, максимальну просадку, ризикованість, використання балансу, кількість угод, середню дохідність, волатильність результату, а також інші показники, які допоможуть оцінити стратегію з різних сторін. Це дозволяє зрозуміти не лише абсолютний фінансовий результат, а й те, наскільки стабільною була стратегія, наскільки агресивно вона використовувала ресурси, чи були ризиковані періоди та як вона реагувала на різні ринкові умови.

Взаємодія з іншими сервісами:

- BTDownloader завантажує історичні дані і зберігає їх в partitioned parquet в NFS
- BTManager запускає BTRunner
- BTRunner читає файли з NFS
- BTRunner зберігає результати запуску стратегії в PostgreSQL

BTManager

На відміну від компонентів, які працюють із реальними ринковими даними, BTManager керує процесами, пов’язаними саме з бектестингом. Його основне завдання — організація та координація виконання великої кількості тестів стратегій, які можуть запускатися паралельно та вимагати значних ресурсів. Якщо BTRunner безпосередньо виконує тестування однієї стратегії, то BTManager визначає, скільки таких запусків потрібно підняти, як їх розподілити, які з них мають вищий пріоритет, та контролює повний життєвий цикл цих процесів.

Перш за все, BTManager отримує завдання на проведення бектесту. Це може бути запит від користувача, автоматичний запуск по розкладу або повторне тестування після зміни параметрів стратегії. BTManager не виконує тест самостійно — він формує чергу задач, розподіляє їх між доступними BTRunner та контролює завантаження кожного інстансу. Завдяки цьому система може виконувати багато запусків одночасно без перевантаження окремих вузлів. У разі збільшення кількості задач BTManager може масштабувати кількість BTRunner, піднімаючи нові екземпляри для обробки черги.

Взаємодія з іншими сервісами:

- BTManager запускає BTDownloader
- BTManager запускає BTRunner

DataDownloader

Цей сервіс завантажуватиме дані з Binance Vision, розпаковуватиме їх і зберігатиме в NFS в форматі partitioned parquet.

DataDownloader виконує роль основного джерела історичних ринкових даних для всієї системи. Його задача — автоматично отримувати сирі файли з Binance Vision, які зазвичай надаються у вигляді архівів CSV, і перетворювати їх у зручний та оптимізований формат для подальшої обробки. Завдяки цьому інші сервіси, такі як BTRunner або будь-які дослідницькі інструменти, можуть працювати з уже структурованими та готовими до використання даними без необхідності повторного завантаження або ручної конвертації.

Для кожного символу, таймфрейму та часових діапазонів DataDownloader самостійно визначатиме, які файли потрібно завантажити, перевірятиме наявність уже існуючих даних у NFS та уникатиме дублювання. У разі відсутності певних ділянок історії він дозавантажуватиме їх автоматично. Це дозволяє підтримувати повний і безперервний набір партиційованих даних для всіх інструментів, з якими працює система.

Після завантаження DataDownloader розпаковує ZIP-файли, читає CSV-файли, виконує валідацію даних і конвертує їх у формат Parquet. Використання Parquet є важливою частиною архітектури: він забезпечує значно швидше читання, компактне зберігання та можливість ефективно працювати з колонками, що особливо корисно під час фільтрації чи агрегації у процесі бектестингу або аналізу.

Висновки

В ході виконання дипломної роботи було розроблено архітектуру мікросервісної системи, призначеної для отримання, обробки, зберігання та аналізу великих обсягів фінансових даних, а також тестування та запуску торгових стратегій в реальному часі.

Було виконано порівняльний аналіз технологій, включаючи бази даних, брокери повідомлень, формати файлів, мови програмування та біржі, що дозволило свідомо обрати найбільш підходящі рішення для конкретних задач системи. Це забезпечило обґрунтований вибір інструментів і створило фундамент для побудови надійної та масштабованої інфраструктури.

На основі отриманих досліджень була спроектована архітектура з чітким розподіленням відповідальностей між сервісами. Кожен компонент вирішує чітку проблему. DataDownloader відповідає за завантаження й підготовку історичних даних; BTRunner здійснює тестування торгових стратегій; DBSaver забезпечує довготривале зберігання даних. Крім задач трейдингу, деякі сервіси виконують роль «керівника» іншими сервісами, це потрібно для більш зручного керування та розгортання: RealTimeDataManager, StrategyManager та BTManager виконують функції координації, керування конфігураціями та масштабування відповідних підсистем. Такий підхід дозволяє досягти високої гнучкості, рівня ізоляції та можливості незалежного розгортання й масштабування кожного сервісу.

Система побудована з виконанням вимог до продуктивності, надійності та безпеки, враховуючи важливість швидкості розробки. Було використано комбінації технологій таким чином, щоб отримати максимум позитивних якостей кожної і щоб негативні були не помітні. Використання Kafka дає високу швидкість передачі повідомлень і її гнучкості повністю вистачає для поточної архітектури. Parquet – чудовий формат для швидкого читання, але повільний для запису, саме тому він буде записуватись лише раз і читатися декількома репліками одного з сервісів.

Розроблена архітектура є гнучкою, масштабованою і здатною до розширення: можна як збільшувати кількість реплік існуючої системи для збільшення продуктивності, так і додавати нові біржі, підходи до розрахунків. Використання мікросервісів дозволяє легко збільшити продуктивність тієї частини системи де це потрібно.

Також, при потребі пришвидшення системи(наприклад, для високочастотного трейдингу) можна деякі сервіси перенести на мову Go, зберігши формат повідомлень. Таким чином вся система продовжить працювати.

При виконанні роботи було досягнуто поставленої мети: розроблено архітектуру сучасної мікросервісної системи, яка забезпечує повний цикл роботи з торговими стратегіями, я гнучкою, масштабованою і достатньо продуктивною.

Перелік посилань

1. PostgreSQL : офіційна документація: https://www.postgresql.org/ (дата звернення: 21.10.2025).
2. TimescaleDB : розширення для часових рядів PostgreSQL: https://github.com/timescale/timescaledb (дата звернення: 21.10.2025).
3. ClickHouse : аналітична колоночна система керування базами даних: https://clickhouse.com/ (дата звернення: 21.10.2025).
4. MongoDB : документоорієнтована система керування базами даних: https://www.mongodb.com/ (дата звернення: 21.10.2025).
5. SQLite : вбудована реляційна база даних: https://sqlite.org/ (дата звернення: 21.10.2025).
6. QuestDB : високопродуктивна база даних для часових рядів: https://questdb.com/ (дата звернення: 21.10.2025).
7. Comma-separated values (CSV): https://en.wikipedia.org/wiki/Comma-separated_values (дата звернення: 21.10.2025).
8. JSON (JavaScript Object Notation): https://en.wikipedia.org/wiki/JSON (дата звернення: 21.10.2025).
9. Apache Parquet : колоночний формат зберігання даних: https://parquet.apache.org/ (дата звернення: 21.10.2025).
10. Apache Arrow Feather : формат обміну даними: https://arrow.apache.org/docs/python/feather.html (дата звернення: 14.11.2025).
11. HDF5 : формат та бібліотека для зберігання великих обсягів даних: https://www.hdfgroup.org/solutions/hdf5/ (дата звернення: 14.11.2025).
12. Apache Kafka : платформа розподіленої обробки потокових даних: https://kafka.apache.org/ (дата звернення: 14.11.2025).
13. RabbitMQ : брокер повідомлень для розподілених систем: https://www.rabbitmq.com/ (дата звернення: 14.11.2025).
14. Redis Streams : механізм потокової обробки даних у Redis: https://redis.io/docs/latest/develop/data-types/streams/ (дата звернення: 14.11.2025).
15. NATS JetStream : система потокових повідомлень: https://docs.nats.io/nats-concepts/jetstream (дата звернення: 14.11.2025).
16. ZeroMQ : бібліотека асинхронного обміну повідомленнями: https://zeromq.org/ (дата звернення: 14.11.2025).
17. Binance API : офіційна документація API криптовалютної біржі Binance: https://www.binance.com/en/binance-api (дата звернення: 05.12.2025).
18. Binance Spot API Documentation : технічна документація: https://github.com/binance/binance-spot-api-docs (дата звернення: 05.12.2025).

Додаток А. Демонстраційні матеріали(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП ’ЮТЕРНОЇ ІНЖЕНЕРІЇ

Магістерська робота

«Мікросервісна архітектура для аналізу криптовалютного ринку та автоматизованої торгівлі»

Виконав : студент групи КСДМ-61 Мироненко Роман Олександрович

Керівник : к.т.н., доц., доцент кафедри КІ Волохін Віталій Васильович

Київ - 2025

Актуальність теми

Тема актуальна з двох причин:

1. Зростання популярності криптовалюти і автоматизованої торгівлі
2. Цікава інженерна задача, оскільки включає роботу з даними в реальному часі, історичними даними, масштабування системи і роботу з багатьма технологіями.

Постановка задачі

Задачі потрібно вирішити(крім очевидних як масштабованість, швидкість, керування ризиками, торгівля).

Задачі:

- Завантаження і зберігання історичних даних
- Перевірка стратегії на історичних даних
- Вирішення проблем з обмеженнями біржі, а саме:
  - Кількість акаунтів(для розподілання бюджету)
  - Кількість з’єднань
- Керування балансом
- Простота запуску
- Зберігання історії

Порівняння технологій

Було порівняно наступні технології:

- Мікросервіс чи моноліт
- БД
- Брокери повідомлень
- Типи файлів
- Мови програмування
- Біржі





4

Прийняті ріше

Було обрано баланс між швидкістю роботи і швидкістю розробки.

- Мікросервіс, бо його простіше масштабувати та підтримувати і його вистачає для моїх задач(не HFT, а торгівля на свічкових даних)
- БД – PostgreSQL, через універсальність, перевіреність часом, стабільність. Для задач в моїй архітектурі вона повністю підходить.
- Kafka – ідеально підійшов. Необхідний набір правил маршрутизації без зайвих і максимальна швидкість.
- Parquet – оптимізований для читання, типи даних, колонкове зберігання, можливість розбиття файлі на деревовидну структуру директорій з файлами з фільтрами, наявність фільтрації
- Python – найбільша екосистема для вказаних задач, найкраща швидкість розробки. В майбутньомуможна частково використати Go
- Binance – найкраще API, найбільший архів історичних даних, великі обсяги торгів, гарна документація.

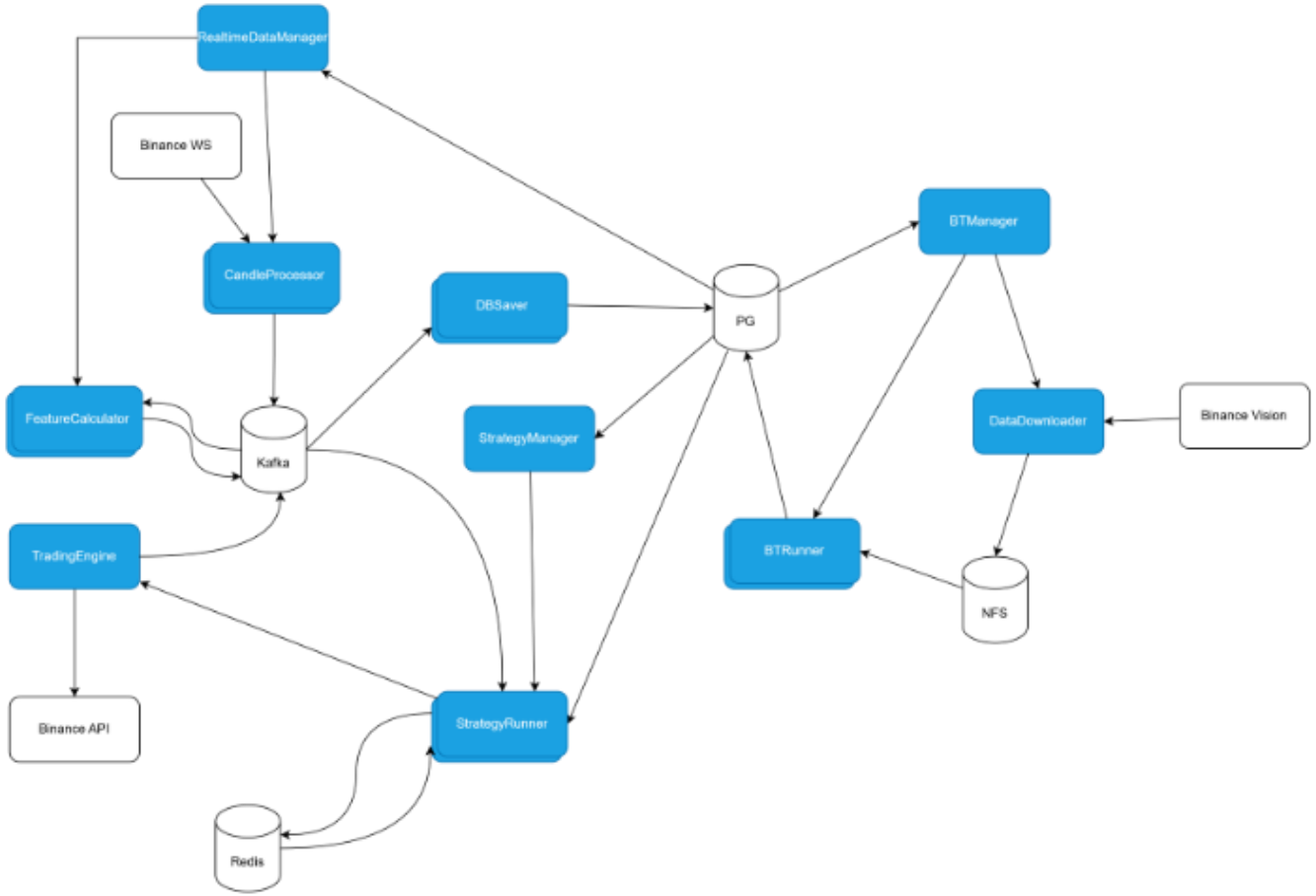


Рисунок 1 – Архітектурамікросервісів для аналізу і торгівлі на криптовалютних біржах

Висновки

- Розроблено мікросервісну архітектуру системи аналізу криптовалютного ринку та автоматизованої торгівлі.
- Реалізовано розподілену систему збору, обробки та зберігання потокових і історичних даних.

7

Публікації та апробація роботи

Статті:

- Antonenko, A. V. ., Buriak, M. S., Vostrikov, S. O., Balvak, A. A., Korotin, D. S., & Myronenko, R. O. (2025). МОЖЛИВОСТІ ПІДВИЩЕННЯ ЕФЕКТИВНОСТІ МЕРЕЖ В СИСТЕМАХ КЛІЄНТ-СЕРВЕР. Таврійський науковий вісник. Серія: Технічні науки, 1(4), 3-14. https://doi.org/10.32782/tnv-tech.2025.4.1.1

Тези доповідей

- Бутенко В.О., Мироненко Р.О. Роль автоматизованого аналізу даних у забезпеченні продовольчої безпеки країни// Робота подана до друку, ще не опублікована.
- Мироненко Р.О., Бутенко В.О. Використання алгоритмів машинного навчання для прогнозування попиту // Робота подана до друку, ще не опублікована.