ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ДИСТРИБУТИВНОЇ СИСТЕМИ ВИКОНАННЯ ВІДКЛАДЕНИХ ЗАВДАНЬ МОВОЮ JAVA»**

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Зайцев І.С.
(прізвище та ініціали)
Керівник Гаманюк І.М.
(прізвище та ініціали)
Рецензент_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Напрям підготовки - 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
О.В. Негоденко
“__” _____2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ
ЗАЙЦЕВУ ІВАНУ СЕРГІЙОВИЧУ
(прізвище, ім‘я, по батькові)

1. Тема роботи: «Розробка дистрибутивної системи виконання відкладених завдань мовою Java»
Керівник роботи Гаманюк І.М.
(прізвище, ім‘я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від “ 24 ” лютого 2023 року №26

2. Строк подання студентом роботи “ 1 ” червня 2023 року

3. Вхідні дані до роботи:
3.1 Науково-технічна література, пов’язана з розробкою дистрибутивних систем.
3.2 Технічна документація баз даних та комунікації між системами.

4. Зміст розрахунково-пояснювальної записки(перелік питань які потрібно розробити).
4.1 Аналіз обов’язків розроблюваної системи
4.2 Аналіз та дослідження існуючих аналогів
4.3 Розробка системи виконання відкладених завдань та веб-інтерфесу
4.3.1 Розробити модуль для виконання завдань
4.3.2 Розробити модуль для балансування запитів між вузлами
4.3.3 Розробити модуль для реєстрації вузлів
4.3.4 Розробити модуль для обміну повідомленнями між вузлами
4.3.5 Розробити модуль для зберігання завдань
4.3.6 Розробити модуль для перегляду завдань

4.4 Тестування системи виконання відкладених завдань
4.5 Підключення моніторингу продуктивності до системи
4.6 Розробка інфраструктури, розгортання та запуск системи
4.6.1 Налаштувати гіпервізор Proxmox
4.6.2 Встановити Kubernetes кластер
4.6.3 Розробити Terraform інфраструктуру

5. Перелік демонстраційного матеріалу
5.1 Мета, об’єкт та предмет дослідження
5.2 Аналоги
5.3 Технічне завдання
5.4 Стек технологій
5.5 Висновки

6. Дата видачі завдання “ 25 ” лютого 2023 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	25.02.23-27.02.23	Виконано
2	Аналіз та дослідження існуючих аналогів	28.02.23-10.03.23	Виконано
3	Моделювання, проектування системи	13.03.23-24.03.23	Виконано
4	Розробка основного функціоналу системи	27.03.23-28.04.23	Виконано
5	Тестування системи	01.05.23-05.05.23	Виконано
6	Розробка інфраструктури та розгортання системи	08.05.23-12.05.23	Виконано
7	Вступ, реферат, висновки	15.05.23-19.05.23	Виконано
8	Розробка основних демонстраційних матеріалів	22.05.23-26.05.23	Виконано
9	Попередній захист роботи	25.05.23	Виконано
10	Здача роботи	01.06.23	Виконано

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина бакалаврської роботи: 45 с., 3 табл., 29 рис., 2 дод., 13 джерел.

JAVA, MAVEN, SPRING, MONGODB, POSTGRESQL, ZOOKEEPER, KAFKA, ELASTICSEARCH, KIBANA, ANGULAR, TERRAFORM, KUBERNETES, PROXMOX.

Об’єкт дослідження - функціювання єдиного інформаційного простору.

Предмет дослідження - виконання відкладених завдань в дистрибутивній системі.

Мета роботи - виконання відкладених завдань в дистрибутивній системі шляхом застосування розробленої системи.

Методи дослідження - методи проектування та розробки програмного забезпечення, методи опрацювання та аналізу отримання результатів, методи тестування програмного забезпечення.

В ході виконання дипломної роботи була розроблена дистрибутивна системи виконання відкладених завдань та веб-інтерфейс для моніторингу. Був застосований повний цикл розробки програмного забезпечення який включає:

— планування архітектури та підбір технології;

— розробка системи;

— тестування системи;

— підключення моніторингу до системи;

— розробка інфраструктури та розгортання системи;

Обрано оптимальні засоби розробки та підходи до проектування. Система має гнучку модульну архітектуру з використанням провідних шаблонів проектування. Систему було налагоджено, протестовано та випробувано, що показало, коректність роботи програмного забезпечення. Система є ефективною та вирішує всі поставлені задачі.

У процесі розробки було реалізовано всі необхідні підсистеми проекту:

— основний модуль системи (Core);

— модуль для виконання завдань (Worker);

— модуль для балансування запитів між вузлами (Gateway);

— модуль для реєстрації вузлів (Discovery Provider);

— модуль для обміну повідомленнями між вузлами (Queue Provider);

— модуль для зберігання завдань (Database Provider);

— модуль веб-інтерфейсу для перегляду завдань (UI);

Система виконання відкладених завдань має широку сферу застосування в багатьох галузях, де необхідно автоматизувати процеси виконання завдань з певною затримкою, наприклад:

— інформаційні технології.

Система може використовуватися для автоматичного виконання певних процесів на серверах, які потребують затримки. Наприклад, автоматичний запуск резервного копіювання бази даних або обробка великої кількості даних, що вимагає багато часу;

— фінансова сфера.

Система може використовуватися для автоматичного виконання платежів з певною затримкою або для автоматизації процесу збору платежів від клієнтів;

— медична сфера.

Система може використовуватися для автоматичного відправлення нагадувань про прийом ліків або повідомлення про результати аналізів з певною затримкою;

— логістика.

Система може використовуватися для автоматичного запуску процесу доставки товару з певною затримкою, залежно від часу замовлення;

Ці сфери є лише деякими з можливих застосувань системи виконання відкладених завдань, і насправді застосування може бути широким та різноманітним в залежності від конкретної потреби та вимог.

Система виступає я якості бібліотеки та підтримує можливість інтеграції в наявну інфраструктуру та надає можливість створювати та контролювати відкладені завдання.

ЗМІСТ

ВСТУП 10
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ 12
1.1 Системи виконання відкладених завдань 12
1.2 Масштабування системи 12
1.3 Огляд та аналіз існуючих аналогів 14
2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ 16
2.1 Бекенд - Система виконання відкладених завдань 16
2.1.1 Мова програмування Java 16
2.1.2 Інструмент збірки Maven 16
2.1.3 Фреймворк Spring 17
2.1.4 Виявлення та ідентифікація сервісів Zookeeper Service Discovery 17
2.1.5 Черга для обміну повідомлень Kafka 20
2.1.6 База даних для зберігання стану MongoDB та PostgreSQL 22
2.1.7 Пошуковий двигун Elasticsearch 24
2.2 Фронтенд - Система перегляду вузлів та завдань 24
2.2.1 Мова програмування TypeScript 24
2.2.2 Фреймворк Angular 25
2.2.3 Система моніторингу Kibana 26
2.3 Інфраструктура 26
2.3.1 Terraform 26
2.3.2 Docker та Kubernetes 27
2.3.3 Гіпервізор Proxmox 27
3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ 29
3.1 Структура проекту 29
3.2 Реалізація 32
3.2.1 Система виконання відкладених завдань 32
3.2.2 Система для перегляду завдань 36
3.2 Тестування 39
3.3 Моніторинг 42
4 ІНФРАСТРУКТУРА ТА РОЗГОРТАННЯ 45
4.1 Інфраструктура 45
4.2 Розгортання 46
ВИСНОВКИ 50
ПЕРЕЛІК ПОСИЛАНЬ 51
ДОДАТОК А. ПРЕЗЕНТАЦІЯ 52
ДОДАТОК Б. КОД СИСТЕМИ 58

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

RSS — really simple syndication — використовується для публікації та постачання інформації, що часто змінюється.

UI — user interface — інтерфейс користувача.

APM — application performance management — моніторинг та менеджмент продуктивності програмного забезпечення.

ВСТУП

Робота системи виконання відкладених завдань зазвичай полягає в тому, що користувач системи створює завдання, яке необхідно виконати, задає параметри його виконання, такі як дата та час, та інші параметри. Після цього система призначає виконання завдання відповідно до встановлених параметрів.

При одночасному виконанні великої кількості завдань система отримує велике навантаження, затримки виконання завдань та не може гарантувати відмовостійкості в разі відмови серверу. Для вирішення цієї проблеми використовується масштабування.

Традиційні системи не підтримують горизонтальне масштабування та виконання завдань на декількох вузлах. Це не дає змогу виконувати одночасно велику кількість завдань.

Системи які підтримують горизонтальне масштабування також мають недоліки такі як:

— кожен із вузлів опитує базу даних на наявність завдань.

Цей спосіб створює більше навантаження на систему збільшуючи затримку при виконанні завдань;

— один із вузлів опитує базу даних на наявність завдань.

Цей спосіб зменшує пропускну здатність збільшуючи затримку при виконанні завдань;

Об’єкт дослідження - функціювання єдиного інформаційного простору.

Предмет дослідження - виконання відкладених завдань в дистрибутивній системі.

Мета роботи - виконання відкладених завдань в дистрибутивній системі шляхом застосування розробленої системи.

Методи дослідження - методи проектування та розробки програмного забезпечення, методи опрацювання та аналізу отримання результатів, методи тестування програмного забезпечення.

Мета і завдання роботи:

— розробити систему яка буде підтримувати горизонтальне масштабування та розподілення навантаження між вузлами системи шляхом розподілення бази даних на логічні розділи, які будуть опитуватися кожним із вузлів;

— протестувати систему використовуючи юніт, інтеграційні, та тести продуктивності;

— підключити моніторинг продуктивності до системи, використовуючи Elastic APM;

— розробка інфраструктури та розгортання системи, використовуючи Kubernetes та Terraform;

Розроблювана система буде виступати в якості бібліотеки та буде підтримувати можливість інтеграції в наявну інфраструктуру та надавати можливість створювати та контролювати відкладені завдання.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Системи виконання відкладених завдань

Один з ключових елементів систем виконання відкладених завдань - це компонент який займається плануванням, він визначає час виконання кожного завдання, що забезпечує виконання завдань вчасно та відповідно до заданих параметрів. Завдання запускається на окремому потоці виконання, які створюються системою виконання завдань.

Додатково, системи виконання відкладених завдань можуть забезпечувати інші функції, такі як моніторинг та логування. Наприклад, система може повідомляти про стан виконання завдань, сповіщати про непередбачувані події, які виникають під час виконання завдань, або збирати статистику про виконання завдань для подальшого аналізу та вдосконалення роботи системи.

Види завдань які можуть виконуватись системами відкладених завдань:

— відкладені нагадування. Наприклад при створенні події в календарі система автоматично сповістить у відповідний час;

— відкладені платежі. При оплаті підписки на стримінговому сервісі або інтернет провайдера гроші будуть автоматично списуватись або буде відправлятись лист з нагадуванням про оплату;

— серверні завдання такі як резервне копіювання баз даних, моніторинг ресурсів системи, та інші;

— інтеграція з зовнішніми системами для періодичного запиту RSS змін;

1.2 Масштабування системи

Існують два основні типи для масштабування - вертикальне та горизонтальне.

Також існують такі стратегії для масштабування як мануальне та автоматичне.

Вертикальне масштабування - це процес збільшення потужності системи за допомогою додавання більш потужнішого обладнання (процесор, оперативна пам’ять, диск).

При використанні вертикального масштабування зазвичай використовують мануальну стратегію, коли потрібно збільшити продуктивність системи.

Переваги вертикального масштабування:

— низька складність, не потребує додаткового програмного забезпечення;

Недоліки вертикального масштабування:

— межа ресурсів при якій досягається максимальна потужність апаратного забезпечення;

— не може гарантувати безперебійну роботу системи у випадку відмови апаратного забезпечення;

На рисунку 1.1 зображено приклад вертикального масштабування.

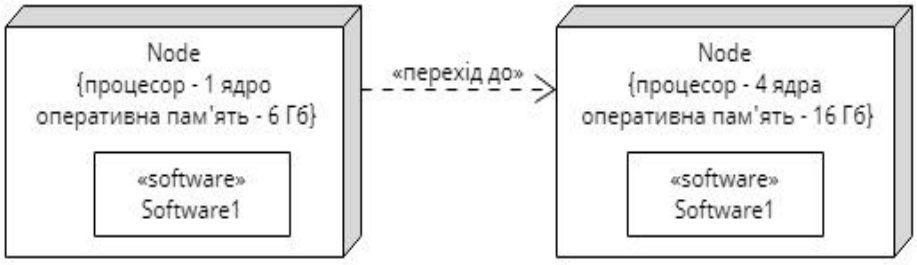


Рисунок 1.1 - Приклад вертикального масштабування

Горизонтальне масштабування - це процес збільшення потужності системи за допомогою додавання нових вузлів системи з ідентичним функціоналом та розподіленням їх між окремими віртуальними або фізичними серверами.

При використанні горизонтального масштабування зазвичай застосовують автоматичну стратегію в залежності від навантаження на систему автоматично створюються нові вузли між якими буде розподілено навантаження.

Переваги горизонтального масштабування:

— висока надійність, доступність та продуктивність. Оскільки в разі відмови одного сервера, інші сервери можуть продовжувати роботу;

Недоліки горизонтального масштабування:

— складність підтримки та управління, оскільки вимагає додаткового програмного забезпечення для підтримки та управління розподілу навантаження;

На рисунку 1.2 зображено приклад горизонтального масштабування.

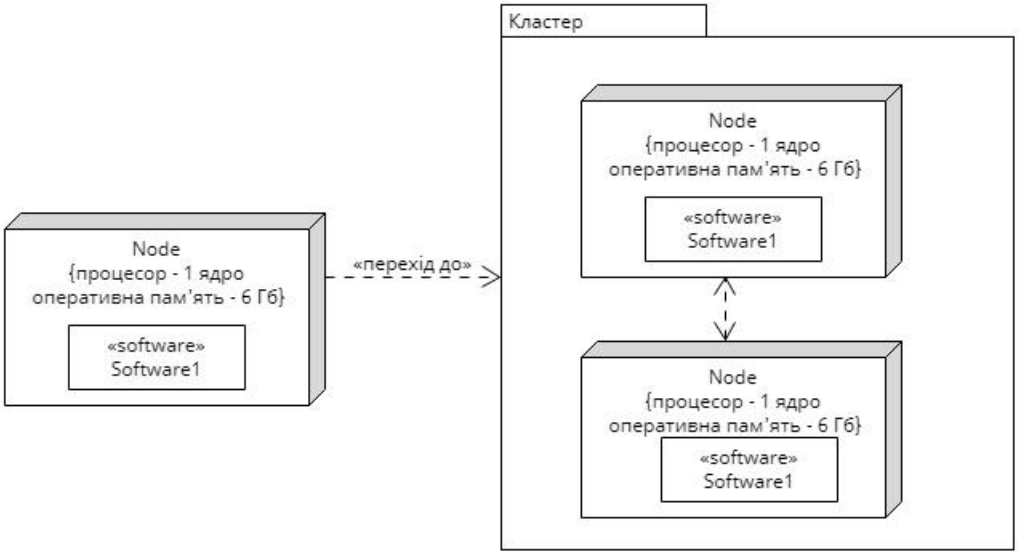


Рисунок 1.2 - Приклад горизонтального масштабування

Для системи виконання відкладених завдань було вибрано підхід горизонтального масштабування для підтримки дистрибутивного виконання завдань на різних вузлах системи.

1.3 Огляд та аналіз існуючих аналогів

Quartz та JobRunr це системи для планування завдань та виконання їх в заданий час. Системи можуть виконувати завдання, що повторюються з певною періодичністю, а також одноразові завдання в зазначений час. Системи підтримують різні бази даних, що дозволяє зберігати інформацію про завдання та їх стан. Системи підтримують механізми блокування завдань, що забезпечує виконання завдань тільки один раз.

Таблиця 1.1 - Список аналогів

	Quartz	**JobRunr**
Схема роботи	Кожен із вузлів опитує базу даних на наявність всіх завдань	Кожен із вузлів опитує базу даних на наявність всіх завдань
Система моніторингу	Немає	Тільки система перегляду завдань
Виконання завдань	Затримка від 1 секунд перед виконанням завдання	Затримка від 5 секунд перед виконанням завдання
Підтримка різних систем управління базами даних	Так	Так

На таблиці 1.1 зображено список аналогів.

Опитування бази даних на наявність всіх завдань на кожному вузлі створює більше навантаження на систему тим самим збільшуючи затримку при виконанні завдань.

2 АНАЛІЗ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Бекенд - Система виконання відкладених завдань

2.1.1 Мова програмування Java

Java - це об’єктно-орієнтована мова програмування, що використовується для розробки різноманітних додатків, від малих мобільних програм до великих корпоративних серверних застосунків.

Java базується на принципах об’єктно-орієнтованого програмування, таких як інкапсуляція, наслідування та поліморфізм. Вона використовує віртуальну машину Java (JVM), що дозволяє виконувати код на різних платформах без необхідності перекомпіляції.

Java має велику бібліотеку стандартних класів, яка надає розширені можливості для розробки додатків. Також існують багато сторонніх бібліотек та фреймворків, які дозволяють розширити функціональність Java та спростити процес розробки.

Загалом, Java є ідеальним варіантом для розробки серверної частини для системи виконання відкладених завдань.

2.1.2 Інструмент збірки Maven

При розробці мульти-модульного проекту з використанням сторонніх залежностей виникає проблема менеджменту, конфігурації, збірки та підтримки проекту.

Apache Maven є популярним інструментом збірки та управління залежностями в проектах Java. Він базується на моделі об'єкту проекту (POM), яка описує конфігурацію проекту та його залежностей.

Maven дозволяє зручно та автоматично керувати залежностями проекту, збирати його з використанням заданих конфігурацій, запускати тестування та генерувати звіти про проект.

Maven має велику кількість плагінів, які дозволяють розширити його функціональність та забезпечити його взаємодію з іншими інструментами. Maven є часто використовуваним інструментом для збирання та керування залежностями в проектах Java, особливо в середовищі розробки інтегрованих платформ, таких як Eclipse.

2.1.3 Фреймворк Spring

При розробці будь-якого функціоналу виникає потреба для розробки додаткового коду для зв'язування класів та модулів між собою, Spring вирішує цю проблему.

Spring - це популярний фреймворк для розробки додатків на платформі Java. Він дозволяє швидко створювати потужні та масштабовані додатки, зосереджуючись на бізнес-логіці.

Spring має багато модулів та підпроектів, які дозволяють використовувати тільки ті частини фреймворку, які необхідні для проекту. Найбільш відомими модулями Spring є:

Spring Core: базовий модуль, який забезпечує функціональність інверсії залежностей та управління бінами (Inversion of Control).

Spring MVC: модуль, який дозволяє створювати веб-додатки та RESTful API.

Spring Data: модуль, який забезпечує простий та швидкий доступ до різних типів баз даних.

Spring також підтримує інші технології, такі як Spring Boot, який дозволяє швидко створювати та запускати самостійні додатки.

2.1.4 Виявлення та ідентифікація сервісів Zookeeper Service Discovery

При розробці застосунку який повинен масштабуватися виникає декілька проблем:

— автоматичного додавання нових вузлів при збільшенні навантаження без необхідності додаткового втручання. (Auto Scaling);

— автоматичного видалення вузлів при проблемах з мережею або ресурсами віртуальної машини. (Fault tolerance);

— рівномірного розподілення навантаження між усіма вузлами застосунку. (Load Balancing);

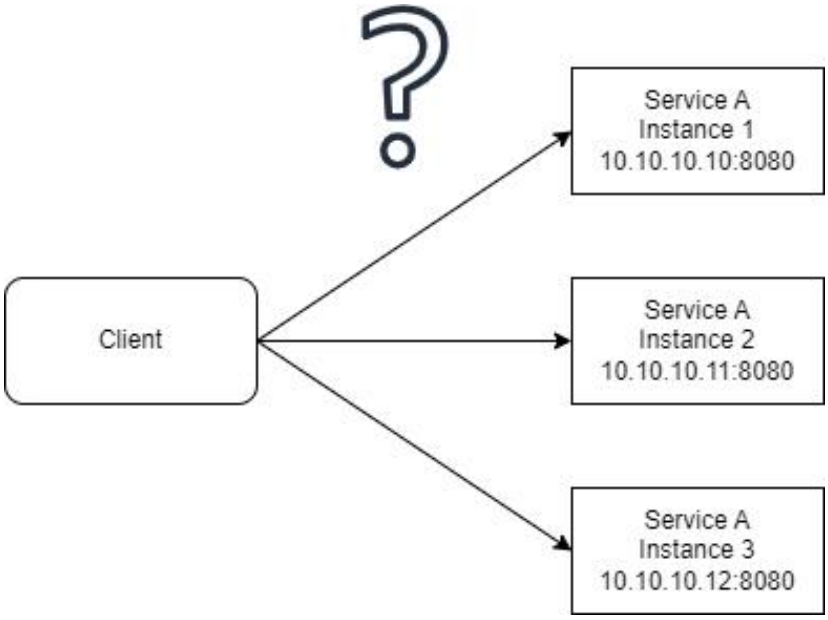


Рисунок 2.1 - Приклад проблеми балансування запитів

На стороні клієнта вести облік IP адресів проблематично:

— за звичай віртуальній машині назначається динамічний IP;

— якщо один із вузлів буде недоступний то продуктивність системи починає деградувати;

Service Discovery - це процес автоматичного виявлення та ідентифікації сервісів (пристроїв, застосунків, ресурсів).

Кожен вузол застосунку реєструється в Service Registry та час від часу відправляє свій статус. Service Registry зберігає інформацію про всі зареєстровані вузли. Кожен вузол може отримати інформацію про сусідні зареєстровані вузли для можливості балансування запитів.

Види Service Discovery:

— на стороні клієнта (Client-Side);

Клієнтська сторона отримує всі зареєстровані вузли застосунку Service A та відповідальна за розподілення запитів.

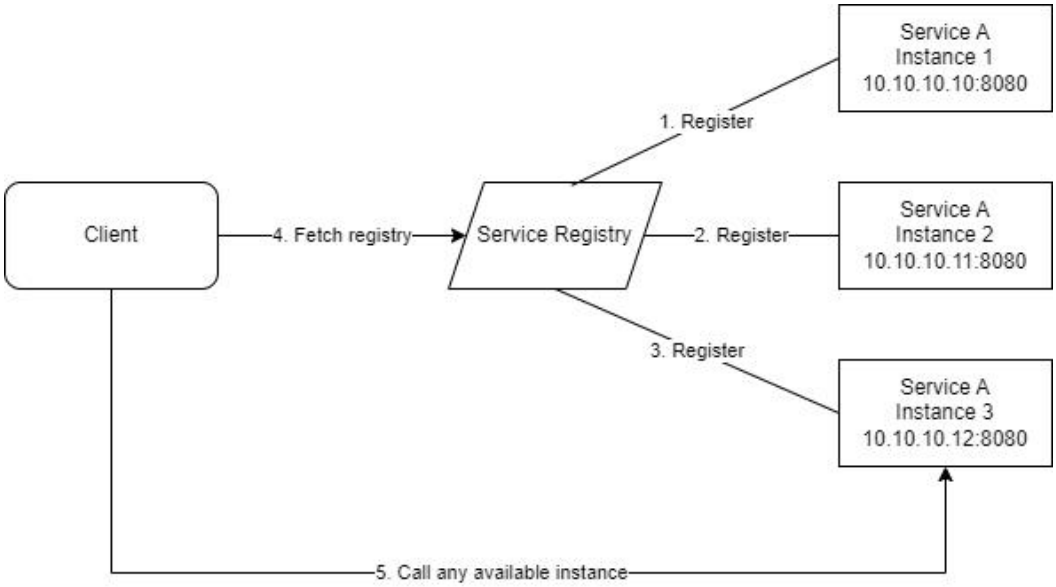


Рисунок 2.2 - Приклад балансування запитів на клієнтський стороні

— на стороні сервера (Server-Side);

На серверній стороні створюється механізм який відповідальний за розподілення запитів, клієнтська сторона знає тільки про один сервіс, наприклад Gateway.

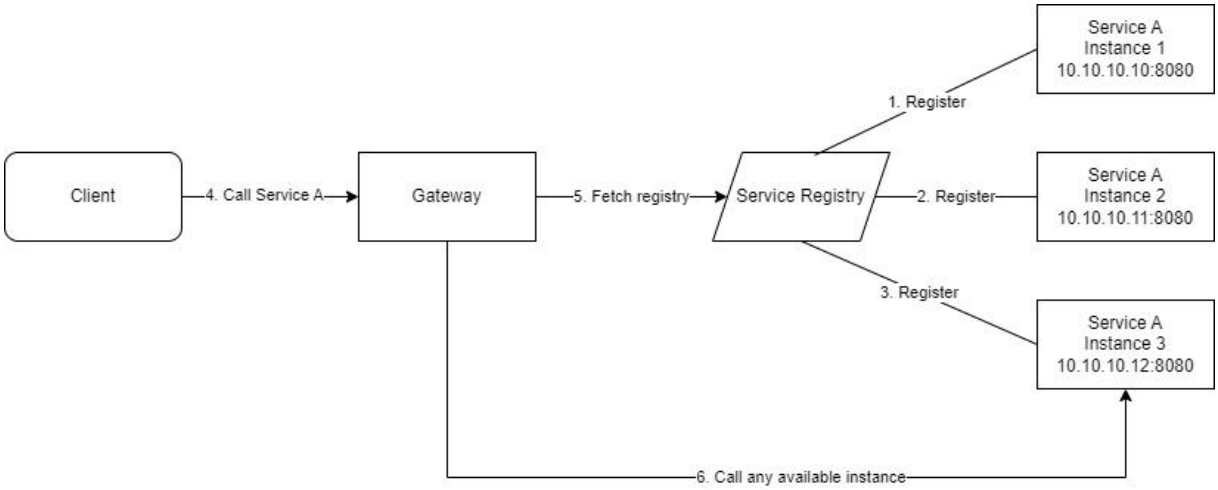


Рисунок 2.3 - Приклад балансування запитів на серверній стороні

Можливі алгоритми балансування:

— випадковим чином вибирається один із вузлів, наприклад Round Robin;

— в залежності від регіону;

— в залежності навантаження Service Registry може відсортувати вузли по пріоритетності та запропонувати клієнту який вузол краще використовувати;

Для розробки дистрибутивної системи виконання відкладених завдань було вибрано Service Discovery з використанням Apache Zookeeper, та підхід балансування запитів на серверній стороні.

2.1.5 Черга для обміну повідомлень Kafka

При розробці застосунку який повинен масштабуватися виникає декілька проблем:

— комунікації та обміну повідомлення між вузлами та гарантії доставки повідомлень. (Delivery guarantee);

— рівномірного розподілення навантаження між усіма вузлами застосунку. (Load Balancing);

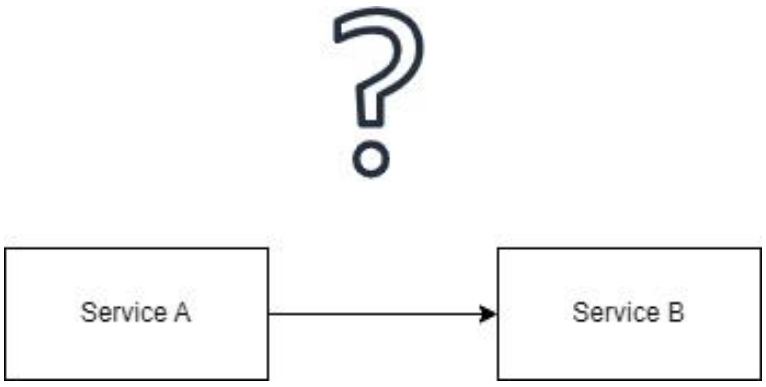


Рисунок 2.4 - Приклад проблеми комунікації

Використовувати традиційний HTTP для комунікації та Service Discovery для виявлення сервісів проблематично:

— немає гарантії що повідомлення буде доставлено або що взагалі не буде доставлено декілька раз в разі проблеми з мережею або якщо відправник або отримувач буде не доступний;

— немає високої пропускної здатності та низької затримки;

Apache Kafka - це дистрибутивна система для обміну повідомленнями, яка дозволяє ефективно передавати великі об'єми даних між різними застосунками та системами.

При використанні Kafka застосунки не комунікують на пряму, замість цього один застосунок відправляє повідомлення в Kafka брокер, другі застосунки підписуються на певні повідомлення та можуть отримувати їх, це дає змогу одному застосунку відправляти повідомлення одночасно для декількох отримувачів.

Рисунок 2.5 - Приклад комунікації використовуючи Kafka

Kafka зберігає всі отримані повідомлення на протязі періоду який можливо сконфігурувати, якщо якийсь застосунок був не доступний на момент відправки повідомлення то в нього є час для того щоб прочитати повідомлення в майбутньому. (Data retention)

Так як архітектурно додатки не комунікіють на пряму це дає змогу підтримувати високу пропускну здатність та низьку затримку. (High throughput and lower latency)

Kafka підтримує масштабування, декілька вузлів може бути запущено для балансування навантаження між брокерами Kafka. Навіть якщо один із вузлів вийде з ладу та буде не доступний Kafka може продовжувати працювати. (Fault tolerance)

Всі дані в Kafka зберігаються в топіках (topic) та розділах (partition). Топік це як таблиця в базі даних, відправник має змогу відправити повідомлення в топік, а підписники мають змогу підписатися на отримання повідомлень з конкретного топіку. Розділи використовуються для того щоб розподілити навантаження між кількома брокерами Kafka та забезпечити паралельну обробку даних між

підписниками, також це є ключовий механізм для підтримки масштабування. (Scalability)

В Kafka є декілька режимів гарантії доставки повідомлень:

— Максимум один раз (At most once).

При використанні цього підходу повідомлення можуть бути втрачені, якщо під час обробки станеться збій. Ця гарантія забезпечує найвищу пропускну здатність, але не гарантує 100% доставку повідомлень;

— Принаймні один раз (At least once).

При використанні цього підходу повідомлення можуть дублюватися, якщо під час обробки станеться збій, але є гарантія що вони 100% не будуть втрачені;

— Точно один раз (Exactly once).

При використанні цього підходу повідомлення не будуть втрачені або дубльовані, якщо під час обробки станеться збій. Щоб досягнути цієї гарантії Kafka використовує транзакції (transaction) та ідемпотентних отримувачів (Idempotent consumer);

2.1.6 База даних для зберігання стану MongoDB та PostgreSQL

При розробці застосунку який повинен масштабуватися виникає проблема зберігання даних.

При використанні реляційних баз даних (SQL) виникає проблема що більшість із них не підтримують масштабування та розподілення даних між серверами. SQL бази даних гарантують ACID та доступність (Availability) але вони не надають можливості поділу (Partitioning) між декількома серверами.

SQL

Atomicity - база даних гарантує те що при використанні транзації часткового результату збережено не буде, вся транзакція виконається повністю, або всі дані не будуть збережені.

Consistency - база даних гарантує що не буде збережено невідповідних даних які порушують обмеження таблиць (constraint), відносини між таблицями (primary-foreign keys).

Isolation - при виконанні транзакцій паралельно база даних гарантує що при використанні різних ступенів ізоляції одна транзакція не повинна повиливати на другі якщо вона не була збережена.

Існують такі рівні ізоляції транзакцій:

— READ UNCOMMITTED;

— READ COMMITTED;

— REPEATABLE READ;

— SERIALIZABLE;

Кожен із рівнів вирішує різні проблеми (Dirty Reads, Nonrepeatable Reads, Phantom Reads) які можуть виникнути при паралельному виконанні транзакцій.

Durability - база даних веде лог транзакцій та гарантує якщо раптово станеться помилка системи то збережені транзакції не буде втрачено.

NoSQL

NoSQL бази даних підтримують поділ між вузлами та масштабування, але в основному в них немає підтримки ізоляції транзакцій та в деяких базах взагалі немає підтримки консистенції даних.

Різні NoSQL бази даних імплементують різні алгоритми зберігання та обробки даних. Немає універсального інструменту який би можна було використовувати для всіх задач. При підборі бази даних потрібно ретельно обирати під які задачі база даних підходить краще за все.

Основні переваги NoSQL баз даних:

— Partitioning.

Дані зберігаються на диску в залежності від розділу, це дає змогу зберігати велику кількість даних, а також швидко отримувати дані по розділу. Наприклад як розділ можна використовувати країну або регіон, та зберігати користувачів на різних розділах диску;

— Sharding.

Алгоритм зберігання Sharding схожий з Partitioning але дані зберігаються не на одному диску та на одному сервері, а на різних серверах та на різних розділах

диску. Це дає змогу розподілити навантаження на різні вузли бази даних для розділів;

— Fault tolerance.

Так як NoSQL бази даних мають змогу масштабуватися то навіть якщо один із вузлів вийде з ладу та буде не доступний база даних може продовжувати працювати без проблем доступності;

Один з ключових недоліків NoSQL баз даних це більша затримка при роботі з базою даних, тому що дані можуть знаходитися на фізично різних серверах.

Для розробки дистрибутивної системи виконання відкладених завдань було вибрано NoSQL базу даних - MongoDB. MongoDB це баланс між підтримкою ACID та підтримкою Partitioning, Sharding.

2.1.7 Пошуковий двигун Elasticsearch

При розробці будь-якої системи виникає потреба моніторингу стану системи та зберігання/аналізу логів, створення сповіщень в разі критичних помилок.

Elasticsearch - це пошуковий двигун, який дозволяє зберігати та аналізувати великі обсяги даних в реальному часі. Elasticsearch можна використовувати для логування, моніторингу, аналіз веб-трафіку та багато іншого. Elasticsearch підтримує масштабування. Даний пошуковий двигун можна масштабувати горизонтально за допомогою кластерів, що дозволяє збільшити продуктивність та надійність системи.

2.2 Фронтенд - Система перегляду вузлів та завдань

2.2.1 Мова програмування TypeScript

TypeScript - мова програмування, яка є розширенням мови JavaScript. TypeScript включає в себе всі можливості JavaScript, але додає додаткову підтримку для статичної типізації, яка дозволяє підвищити якість коду та зменшити кількість помилок.

Основним призначенням TypeScript є полегшення розробки складних веб-додатків, які містять велику кількість коду та функцій. TypeScript надає можливість визначати типи для змінних, функцій та інших елементів коду, що дозволяє вірно передбачати поведінку коду та запобігати виникненню помилок.

Крім того, TypeScript має ряд інших переваг, таких як підтримка класів, інтерфейсів та модулів, яка полегшує розробку та підтримку коду великих проектів. TypeScript також підтримує асинхронні функції, що полегшує розробку складних додатків, які використовують багатопоточну обробку даних.

У загальному, TypeScript є потужним інструментом для розробки веб-додатків, який дозволяє підвищити якість та надійність коду та спростити процес розробки та підтримки проектів.

2.2.2 Фреймворк Angular

Для розробки системи перегляду вузлів, завдань та їх статусу був вибраний фреймворк Angular.

Angular - це потужний фреймворк для розробки веб-додатків, створений компанією Google. Він дозволяє створювати високоякісні та масштабовані додатки за допомогою TypeScript та HTML. Основна мета Angular полягає в полегшенні розробки складних веб-додатків, які мають велику кількість даних та функцій.

Angular має багато функцій та інструментів, які дозволяють створювати додатки швидко та ефективно. Наприклад, фреймворк має вбудовану підтримку для компонентів, що дозволяє створювати елементи інтерфейсу користувача які можна перевикористати. Angular також має підтримку для директив, що дозволяє змінювати поведінку елементів DOM, та сервісів, що дозволяють створювати та використовувати загальні функції та дані.

Крім того, Angular має вбудовану підтримку для створення односторінкових додатків, що дозволяють користувачам переходити між сторінками без перезавантаження сторінки. Фреймворк також має підтримку для HTTP-запитів, що дозволяє додаткам взаємодіяти з сервером та отримувати дані.

У загальному, Angular є потужним фреймворком для розробки веб-додатків, який дозволяє створювати високоякісні та масштабовані додатки швидко та ефективно.

2.2.3 Система моніторингу Kibana

Kibana - це візуальний інтерфейс, який дозволяє здійснювати пошук, візуалізацію та аналіз даних, збережених у Elasticsearch. Відображати дані можна в різних форматах, включаючи таблиці, діаграми та графіки. Крім того, Kibana дозволяє створювати та налаштовувати різні види дошок та панелей для відображення важливих даних у режимі реального часу.

2.3 Інфраструктура

2.3.1 Terraform

Для того щоб розробити універсальний автоматичний спосіб розгортання системи було вибрано інструмент Terraform.

Terraform - це інструмент інфраструктурного програмування (Infrastructure as Code), який дозволяє автоматизувати процес створення та управління інфраструктурою. Terraform використовує декларативний підхід до інфраструктурного програмування, де вміст описує бажаний стан інфраструктури, а Terraform створює та управляє необхідними ресурсами, щоб досягти цього стану.

Однією з ключових переваг Terraform є його гнучкість та підтримка багатьох провайдерів хмарних сервісів та інших інфраструктурних систем. Terraform може працювати з провайдерами, такими як AWS, Google Cloud Platform, Microsoft Azure, а також з власними серверами, кластерами Kubernetes та іншими інфраструктурними системами.

Також, Terraform забезпечує можливість версіонування конфігурації інфраструктури, що дозволяє легко відслідковувати зміни та повертатися до попередніх версій конфігурації.

Загалом, Terraform дозволяє ефективно керувати інфраструктурою та забезпечити її повну автоматизацію, що допомагає зменшити ризики людських помилок та прискорити розгортання та управління інфраструктурою.

2.3.2 Docker та Kubernetes

Для того щоб розробити універсальний спосіб запуску системи було вибрано Doker для контейнерезації та Kubernetes для оркестрування.

Docker - це платформа для створення, розпакування та запуску контейнерів. Контейнери є ізольованими віртуальними середовищами, які містять застосунок та всі залежності, необхідні для його роботи. Docker забезпечує стандартизацію процесу розгортання додатків, що дозволяє ефективно використовувати ресурси та забезпечує більш просте та швидке розгортання додатків.

Kubernetes - це система управління контейнерами, яка дозволяє автоматизувати розгортання, масштабування та керування додатками, які запущені в контейнерах. Kubernetes забезпечує автоматичне масштабування додатків, балансування навантаження та автоматичне відновлення після відмови. Використання Kubernetes дозволяє забезпечити високу доступність та надійність додатків.

Разом Docker та Kubernetes дозволяють легко та ефективно розгорнути, масштабувати та керувати додатками в хмарних середовищах. Використання цих технологій дозволяє забезпечити високу продуктивність та ефективне використання ресурсів сервера.

2.3.3 Гіпервізор Proxmox

Для розміщення Kubernetes системи було вибрано локальний сервер та Proxmox гіпервізор.

Proxmox - це віртуалізаційна платформа з відкритим вихідним кодом, яка дозволяє встановлювати та управляти віртуальними машинами та контейнерами на фізичних серверах. Proxmox базується на гіпервізорі KVM, що дозволяє

забезпечити ефективну віртуалізацію різних типів ресурсів, таких як обчислювальна потужність, пам'ять та мережеві ресурси.

Proxmox має простий та зрозумілий інтерфейс користувача, що дозволяє швидко налаштувати та управляти віртуальними машинами та контейнерами. Крім того, Proxmox підтримує клонування та резервне копіювання віртуальних машин та контейнерів, що забезпечує зручне управління та зменшує час на відновлення в разі непередбачуваних ситуацій.

Proxmox також підтримує віддалене управління та моніторинг за допомогою зручного API та різноманітних інтеграцій з іншими інструментами та платформами. Загалом, Proxmox є потужним та зручним інструментом для віртуалізації та управління інфраструктурою, який дозволяє ефективно використовувати ресурси.

3 РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ

3.1 Структура проекту

При розробці системи було передбачено підтримку горизонтального масштабування для Core компоненту. Компонент Worker це ключовий компонент системи, вузол який займається обробкою завдань, він включає компонент Core та імплементує методи для обробки завдань.

Для забезпечення балансування HTTP запитів між декількома Worker вузлами було розроблено Gateway компонент. Кожен вузол системи використовує Discovery provider компонент для реєстрації, після цього Gateway компонент також використовує Discovery provider для отримання всіх реєстрацій вузлів. Discovery provider також використовується для ключового алгоритму поділу на розділи та розподілення навантаження на базу даних.

Для підтримки комунікації між вузлами використовується компонент Queue provider, який надає змогу підписуватись на повідомлення одним вузлам, та відправляти їх другим вузлам системи.

Для зберігання стану завдань було розроблено Database provider який використовується всіма вузлами системи.

Для забезпечення перегляду поточних вузлів системи та їх статусу, а також списку завдань було розроблено компонент UI.

При дизайні архітектури системи було використано Java Maven модулі, що дає змогу підключати:

— різні Discovery провайдери, наприклад Zookeeper, Kubernetes, Eureka;

— різні Queue провайдери, наприклад Kafka, RabbitMQ;

— різні Database провайдери, наприклад MongoDB, PostgresSQL, Redis;

На рисунку 3.2 зображена діаграма компонентів системи.

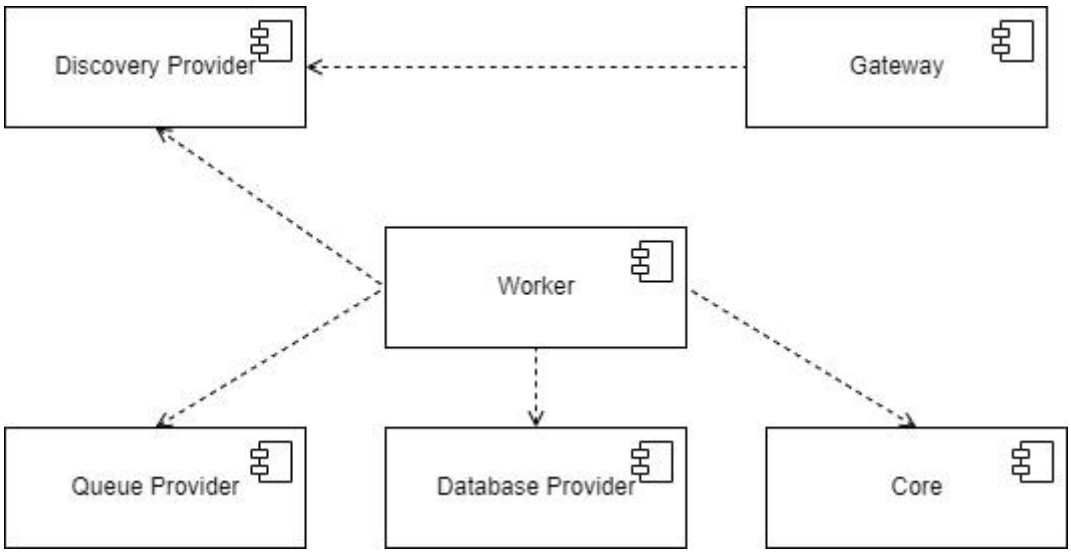


Рисунок 3.2 - Діаграма компонентів системи

Головні компоненти системи:

— Core.

Проект: scheduler-core.

Технології: Java 17, Maven, Spring Boot 3;

— Gateway.

Проект: scheduler-rest-api-gateway.

Технології: Spring Cloud Gateway, Spring Cloud Zookeeper, Elastic APM;

— Worker.

Проект: scheduler-worker-mongodb.

Технології: Zookeeper Discovery Provider, MongoDB Database Provider, Elastic APM.

Проект: scheduler-worker-postgresql.

Технології: Zookeeper Discovery Provider, PostgreSQL Database Provider, Elastic APM;

Модульні компоненти системи:

— Discovery provider.

Проект: scheduler-discovery-provider-discovery-zookeeper.

Технології: Spring Cloud Zookeeper;

— Queue provider.

Проект: scheduler-queue-provider-kafka.

Технології: Spring Kafka;

— Database provider.

Проект: scheduler-database-provider-mongodb.

Технології: Spring Data MongoDB.

Проект: scheduler-database-provider-postgresql.

Технології: Spring Data PostgreSQL;

Користувацький інтерфейс:

— UI.

Назва: scheduler-ui.

Технології: Angular 15;

На рисунку 3.1 зображена високорівнева діаграма архітектури системи виконання відкладених завдань.

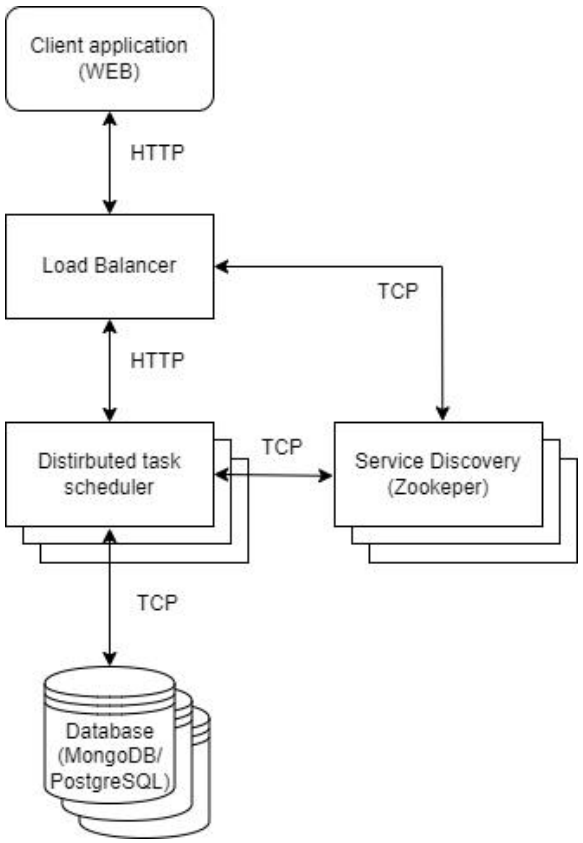


Рисунок 3.1 - Діаграма високорівневої архітектури

Основна функціональність системи:

— Консистентність даних;

Система гарантує виконання завдання тільки один раз. Завдання не повинно пропасти в разі помилки або бути виконано декілька раз. Система виявляє випадки коли те ж саме завдання було розміщене декілька раз.

— Низька затримка.

Система забезпечує максимально низьку затримку для обробки та виконання завдань;

— Відмовостійкість.

Система продовжує працювати в випадках відмови декількох вузлів;

— Розподілення даних.

Система зберігає дані на різних вузлах бази даних, кожен вузол системи комунікує тільки з певними вузлами бази даних;

— Моніторинг.

До системи підключено Elastic Application Performance Monitoring що надає змогу слідкувати за станом системи;

3.2 Реалізація

3.2.1 Система виконання відкладених завдань

Кожен в вузлів опитує базу даних використовуючи розділ який було присвоєно при старті кожні 2 секунди на наявність завдань які повинні бути виконані, якщо є такі завдання тоді вони блокується використовуючи Optimistic Locking та відправляється в чергу Kafka, один з вільних вузлів який підписаний на отримання завдань починає його виконувати.

Алгоритм присвоєння розділів для кожного вузла:

При старті кожен вузол прораховує над якими розділами бази даних потрібно працювати. Наприклад якщо в системі 6 роздлів та запущено один вузол тоді цей вузол буде працювати зі всіма 6 розділами, якщо запустити ще один вузол то два вузла будуть обробляти рівну кількість розділів, перший вузол 3

розділи і другий вузол також 3 розділи. Якщо запустити більше вузлів чим розділів в системі тоді такі вузли не будуть брати участь в обробці завдань.

Алгоритм блокування завдання:

При додаванні нових вузлів присвоєння розділів відбувається не миттєво, і є ймовірність що на декілька секунд два вузла можуть почати обробляти завдання з одного розділу, щоб вирішити цю проблему використовується Optimistic Locking, кожен вузол перед початком обробки завдання блокує його, якщо воно заблоковано то інший вузол не зможе почати обробляти його.

Алгоритм відправки завдань в чергу:

Для забезпечення рівномірного виконання завдань кожним із вузлів використовується черга повідомлень Kafka. Кожен вузол при опитуванні бази даних на наявність вільних завдань не починає зразу їх виконувати, а відправляє їх в Kafka, після цього всі вузли отримають завдання в рівній кількості.

На рисунку 3.3 зображено діаграму діяльності взаємодії вузлів.

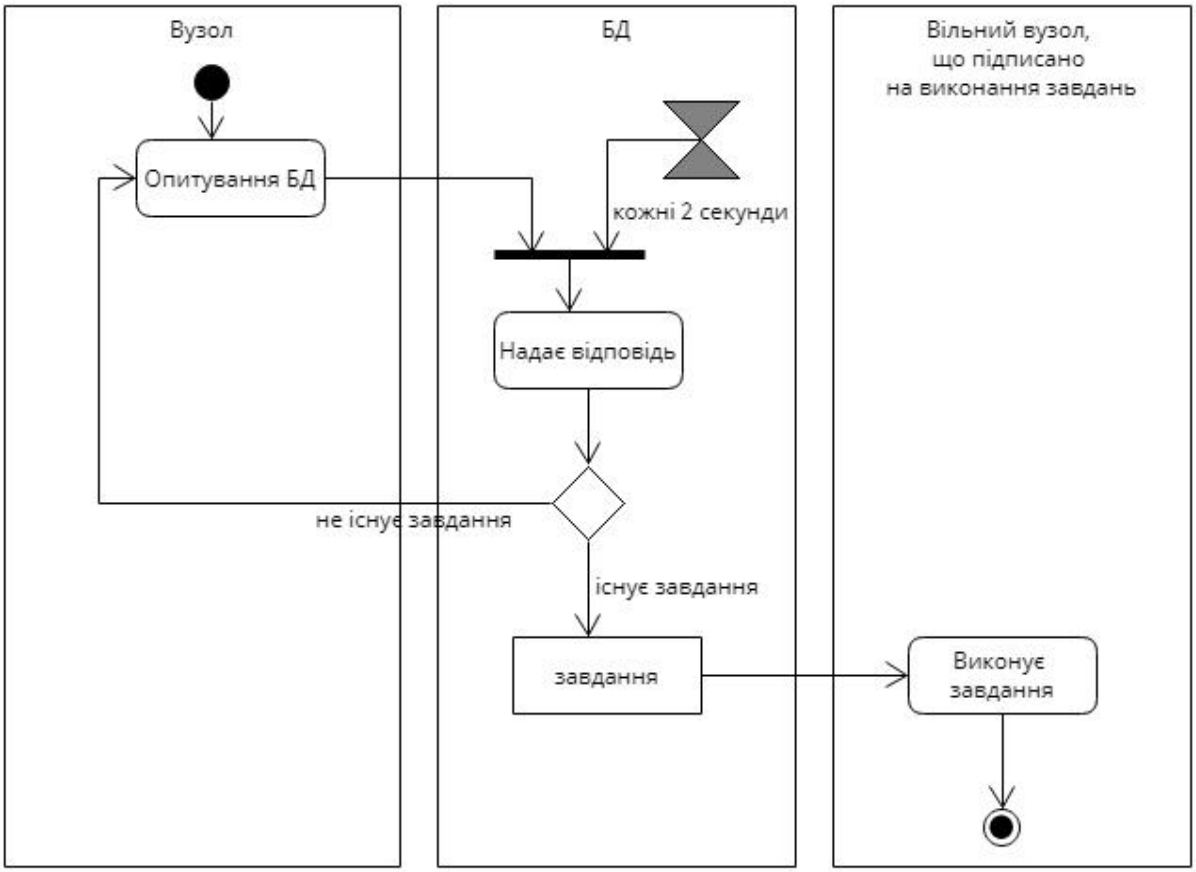


Рисунок 3.3 - Діаграма діяльності взаємодії вузлів

На рисунку 3.3 зображена діаграма основних класів системи виконання відкладених завдань.

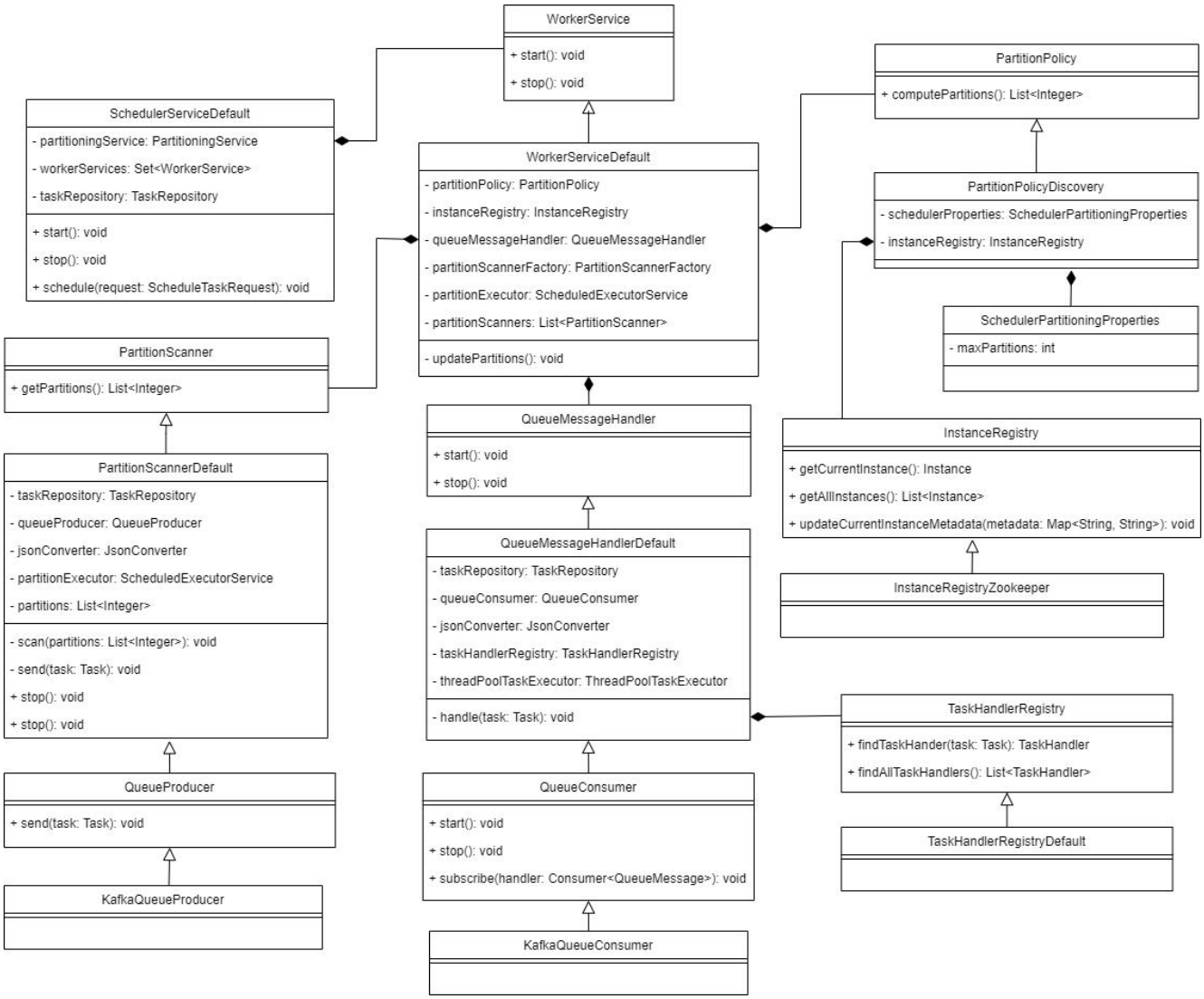


Рисунок 3.3 - Діаграма основних класів системи виконання завдань

При розробці системи було використано такі дизайн та архітектурні патерни:

— Domain Model - об'єкти домену винесені в окремі класи які репрезентують їх стан;

— Service Layer - основна логіка системи для обробки інформації винесена в окремий шар;

— Repository Layer - логіка для комунікації з базою даних винесена в окремий шар;

— Controller - логіка комунікації з HTTP, Kafka винесені в окремі шари;

На рисунку 3.4 зображено діаграму прецедентів системи виконання завдань.

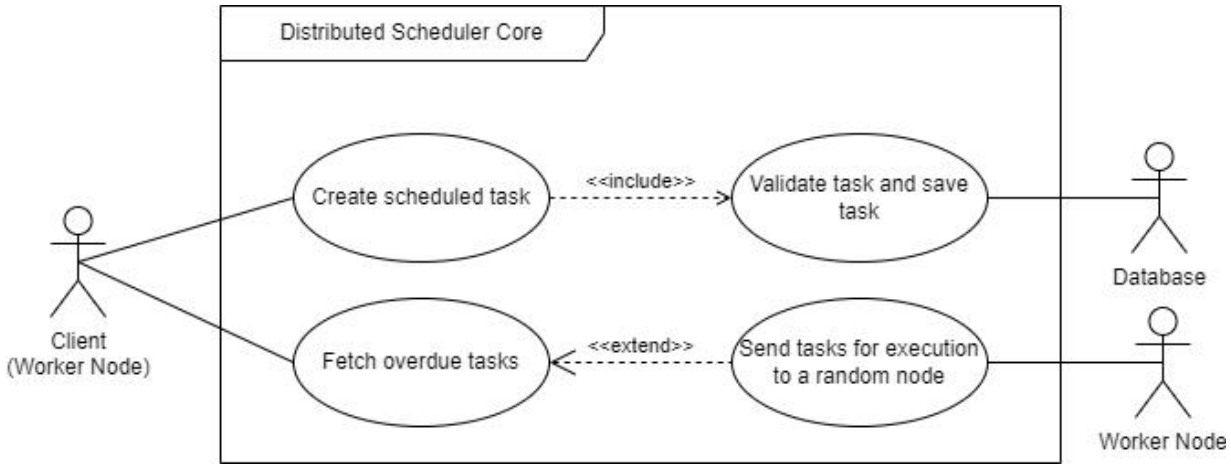


Рисунок 3.4 - Діаграму прецедентів системи виконання завдань

Таблиця 3.1 - Схема бази даних

Назва поля	Тип	Можливі значення
partition	int	
id	UUIDv7	
status	enum	SCHEDULED, SUBMITTED, PROCESSING, SUCCEEDED, FAILED
executeAt	timestamp	
name	varchar	
data	varchar	

На таблиці 3.1 зображена схема бази даних.

Поле partition було додано для поділу бази даних на логічні розділи, що забезпечить поліпшення продуктивності запитів та оптимізації зберігання даних.

Для поля id був вибраний тип UUIDv7, він генерується враховуючи поточний час що забезпечує меншу вірогідність колізій та можливість сортування.

Поле status використовується для можливості відображення поточного статусу завдання.

Поле executeAt використовується для зберігання часу коли повинно бути виконано завдання.

Поле name використовується для ідентифікації завдання та створення механізму виконання завдань.

Поле data може містити довільний текст, та використовується для передачі стану при створенні завдання та його виконанні.

3.2.2 Система для перегляду завдань

Для зручного перегляду стану системи було розроблено веб-інтерфейс використовуючи фреймворк Angular.

На рисунку 3.5 зображено список активних вузлів системи та їх стан.

При додаванні нового вузла в систему він буде автоматично відображений в системі перегляду завдань. Веб-інтерфейс комунікує із вузлами системи використовуючи компонент Gateway який балансує запити між усіма вузлами, кожен вузол системи знає про всі сусідні вузли.

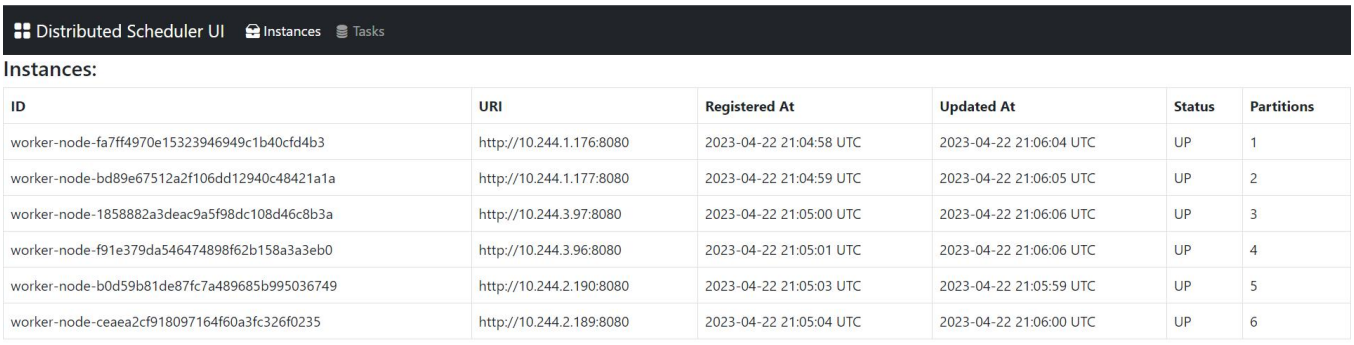


Рисунок 3.5 - Система перегляду завдань, список вузлів

На рисунку 3.6 зображено список завдань та їх стан.

Для реалізації пагінації був використаний тип keyset pagination, що на відмінну від стандартного offset pagination підходу забезпечить більшу продуктивність та не потребує повного скану бази даних для пошуку відповідної сторінки.

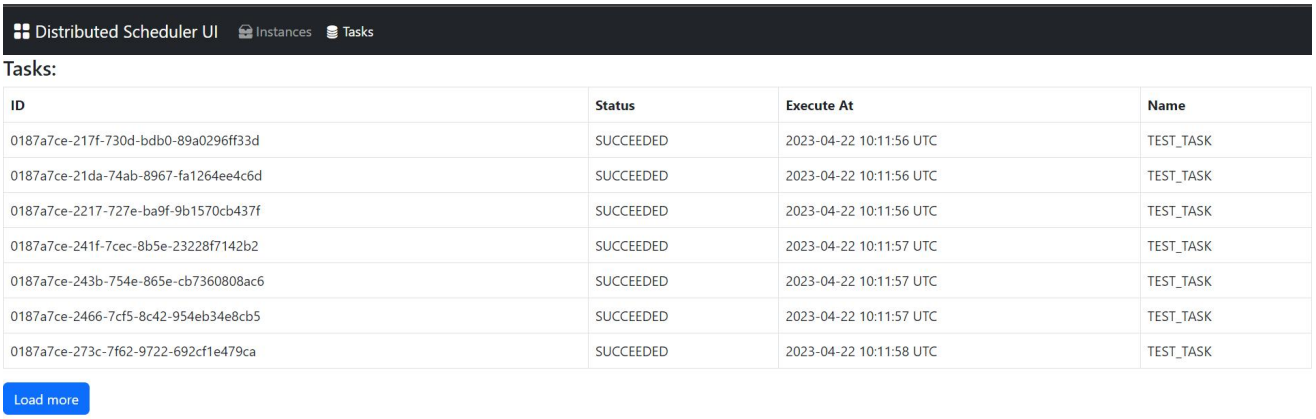


Рисунок 3.6 - Система перегляду завдань, список завдань

При створенні веб-інтерфесу було застосовано патерн MVVM (Model-View-ViewModel). Model використовуються для передачі інформації між контролерами та відображенням. View відображає дані за допомогою HTML. ViewModel дозволяє зв'язати два шари використовуючи потрібну логіку.

На рисунку 3.7 зображено діаграма пакетів для моделювання MVVM.

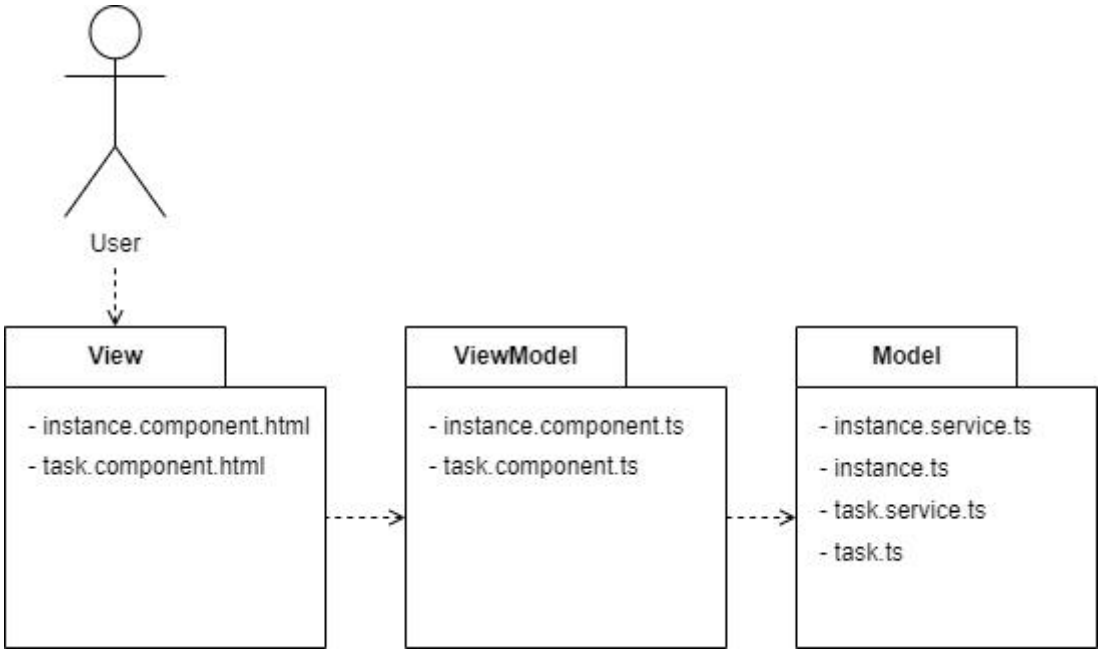


Рисунок 3.8 - Діаграма пакетів MVVM

На рисунку 3.8 зображена діаграма послідовності системи перегляду завдань яка описує логіку звернення використовуючи Load Balancer для рівномірного розподілення запитів між вузлами системи.

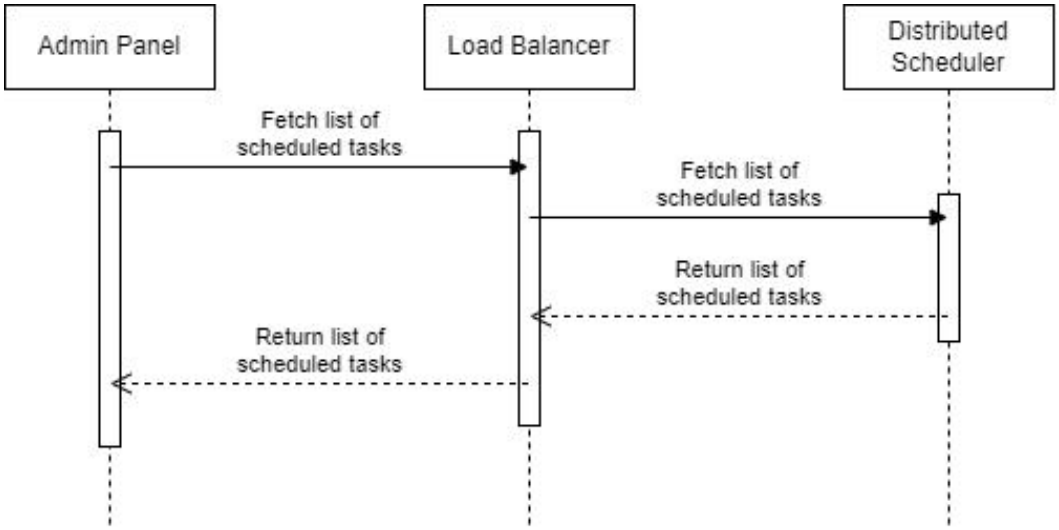


Рисунок 3.8 - Діаграма послідовності системи перегляду завдань

На рисунку 3.9 зображена діаграма станів меню системи перегляду завдань.

Веб-інтерфейс має статичне меню вгорі сторінки та підтримує переходи для перегляду списку вузлів та завдань.

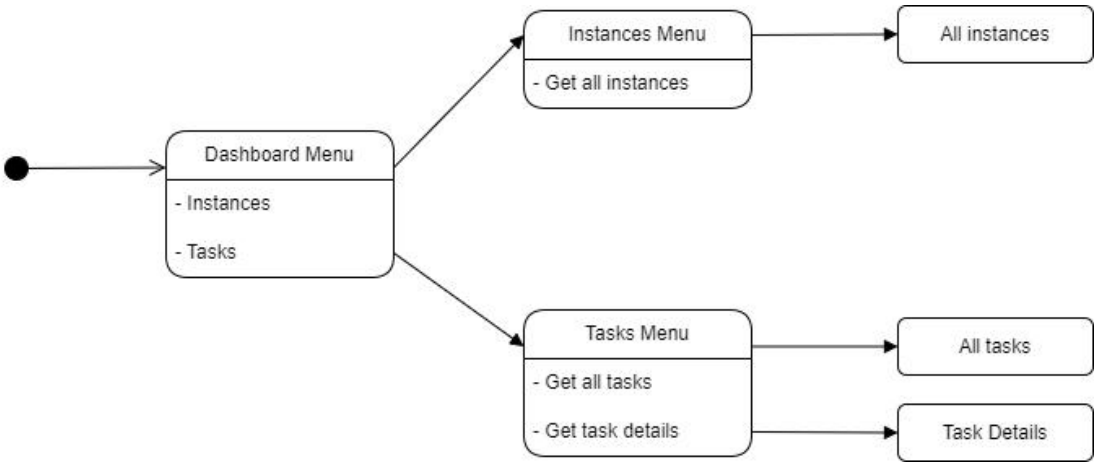


Рисунок 3.9 - Діаграма станів меню системи перегляду завдань

На рисунку 3.10 зображено діаграму прецедентів системи перегляду завдань.

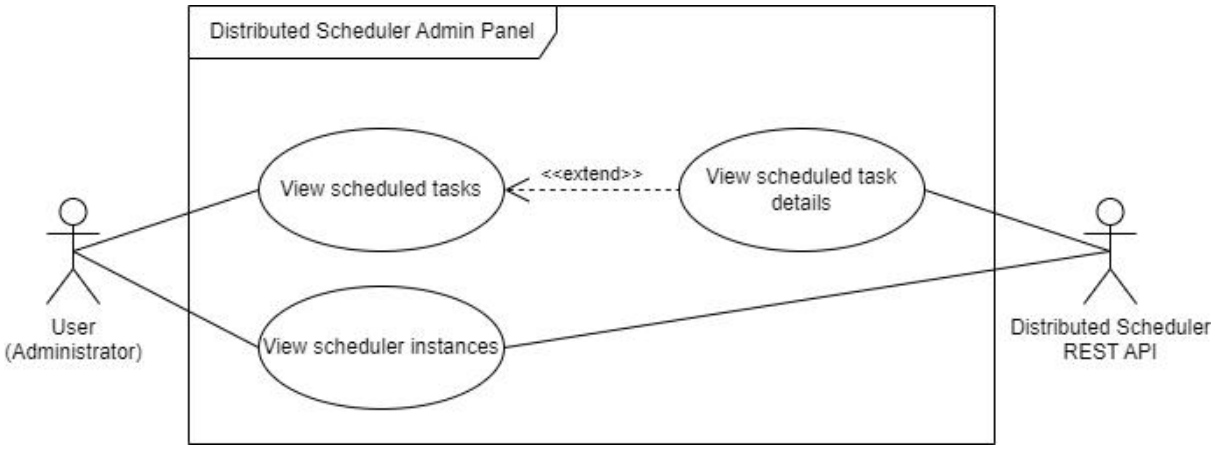


Рисунок 3.10 - Діаграму прецедентів системи перегляду завдань

3.2 Тестування

При розробці дистрибутивної системи виконання відкладених завдань було розроблено тести для перевірки якості програмного забезпечення. Для основного критичного функціоналу використано unit та інтеграційні тести. Для перевірки стійкості системи було проведено тестування продуктивності системи.

Unit тестування - це метод тестування програмного забезпечення, який полягає в тестуванні окремих функцій або методів коду програми з метою визначення їх коректності та відповідності вимогам.

Кожен тест має заданий вхідний набір даних та очікуваний результат, і виконується автоматично. Після виконання тесту зазвичай генерується звіт про результати тестування використовуючи який можна виявити та виправити помилки у коді.

Інтеграційне тестування - це процес тестування, який виконується з метою перевірки взаємодії між компонентами системи. Це можуть бути, наприклад, окремі модулі, компоненти, сервіси або підсистеми.

Основна мета інтеграційного тестування полягає у виявленні помилок, що виникають при взаємодії різних компонентів системи.

Тестування продуктивності - це процес тестування програмного забезпечення з метою визначення швидкості, масштабованості, стійкості та ефективності роботи системи в умовах реального навантаження.

Тестування продуктивності допомагає виявити проблеми, що виникають при тривалому користуванні системою з великою кількістю даних.

Для тестування системи було використано такі технології: JUnit, Mockito, Testcontainers.

JUnit дозволяє створювати тест кейси та в автоматичному режимі перевіряти логіку класів які тестуються.

Mockito дозволяє підміняти реалізації класів що спрощує подальше тестування.

Testcontainers дозволяє коректно протестувати методи використовуючи реальну MongoDB/PostgreSQL базу даних та Kafka брокер.

На рисунку 3.10 зображено протестовані методи системи виконання завдань.

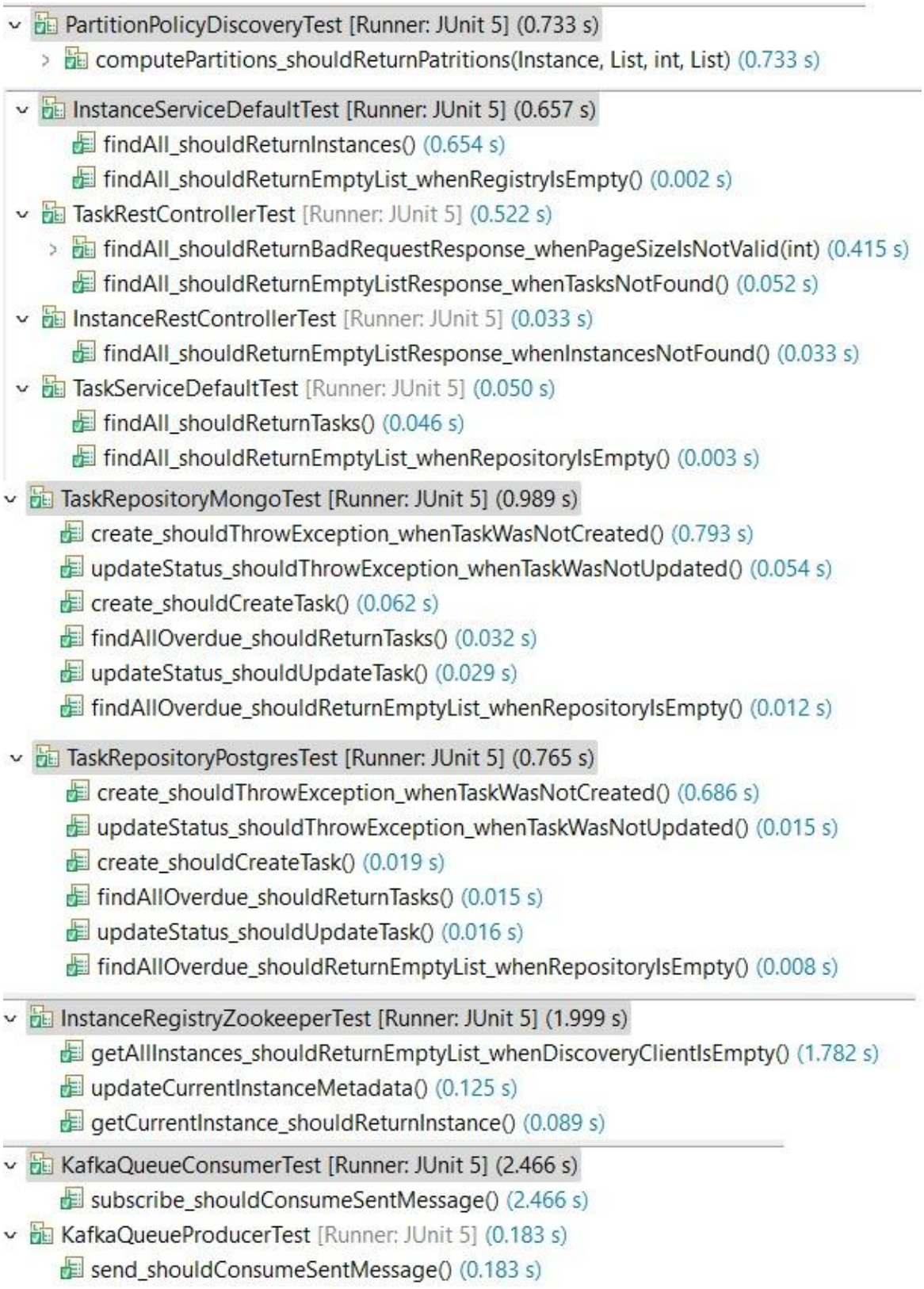


Рисунок 3.10 - Протестовані методи системи виконання завдань

3.3 Моніторинг

Систему було протестовано під навантаженням використовуючи дві різні NoSQL та SQL бази даних.

— MongoDB.

Результати з використанням бази даних MongoDB зображено на рисунку 3.11. Загалом використовуючи базу даних MongoDB система виконувала 30-70 завдань за секунду одним вузлом системи виконання відкладених завдань, в загальній кількості було запущено 6 вузлів.

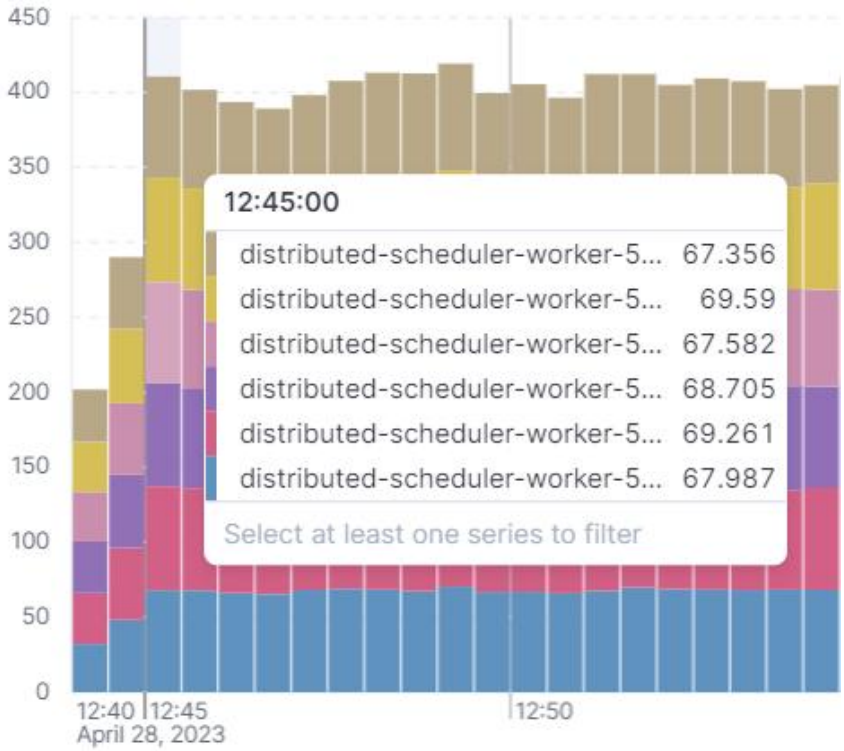


Рисунок 3.11 - Показники виконання завдань за 1 секуну з використанням бази даних MongoDB

База даних MongoDB забезпечує зберігання великої кількості даних за допомогою поділу даних на різних вузлах системи, також гарантує продовжувати працювати в разі відмови одного із вузлів системи, але цей функціонал додає додаткову затримку при комунікації з базою даних так як дані фізично можуть зберігатися на різних серверах.

На рисунку 3.12 зображена транзакція пошуку завдань та відправки їх на виконання. В середньому щоб обробити 1000 завдань використовуючи MongoDB зайняло 12-15 секунд;

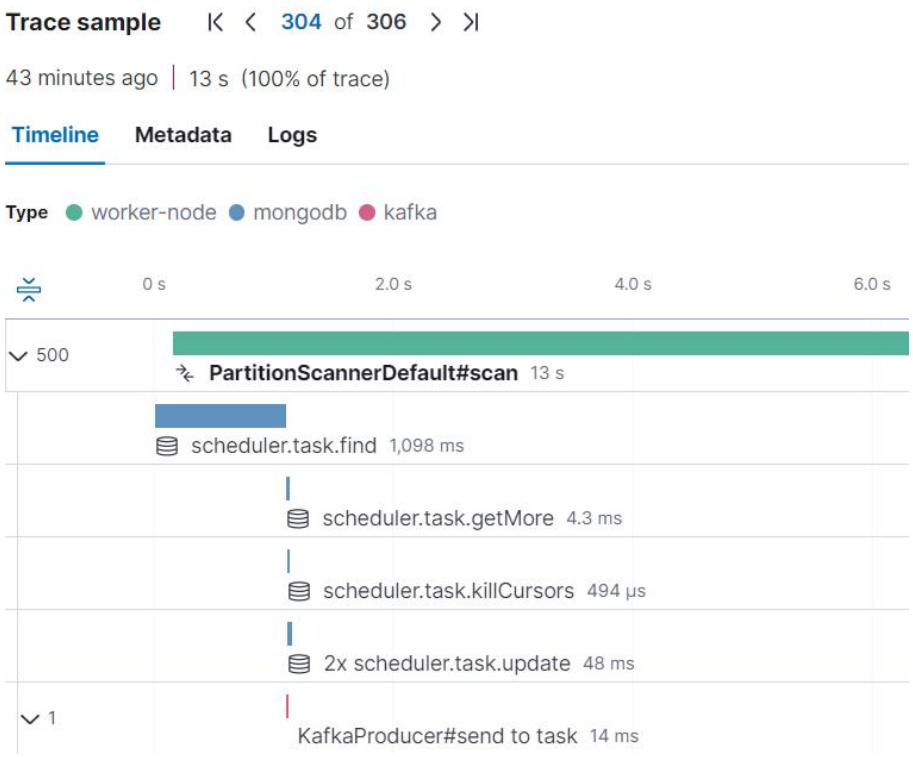


Рисунок 3.12 - Транзакція пошуку завдань з використанням бази даних MongoDB

— PostgreSQL.

Результати з використанням бази даних PostgreSQL зображено на рисунку 3.13. Загалом використовуючи базу даних PostgreSQL система виконувала 90-190 завдань за секунду одним вузлом системи виконання відкладених завдань, в загальній кількості було запущено 6 вузлів.

База даних PostgreSQL зберігає всі дані на одному сервері, що забезпечує низьку затримку, але не гарантує що вразі відмови серверу база даних буде продовжувати працювати.

На рисунку 3.14 зображена транзакція пошуку завдань та відправки їх на виконання. В середньому щоб обробити 1000 завдань використовуючи PostgreSQL зайняло 5-8 секунд;

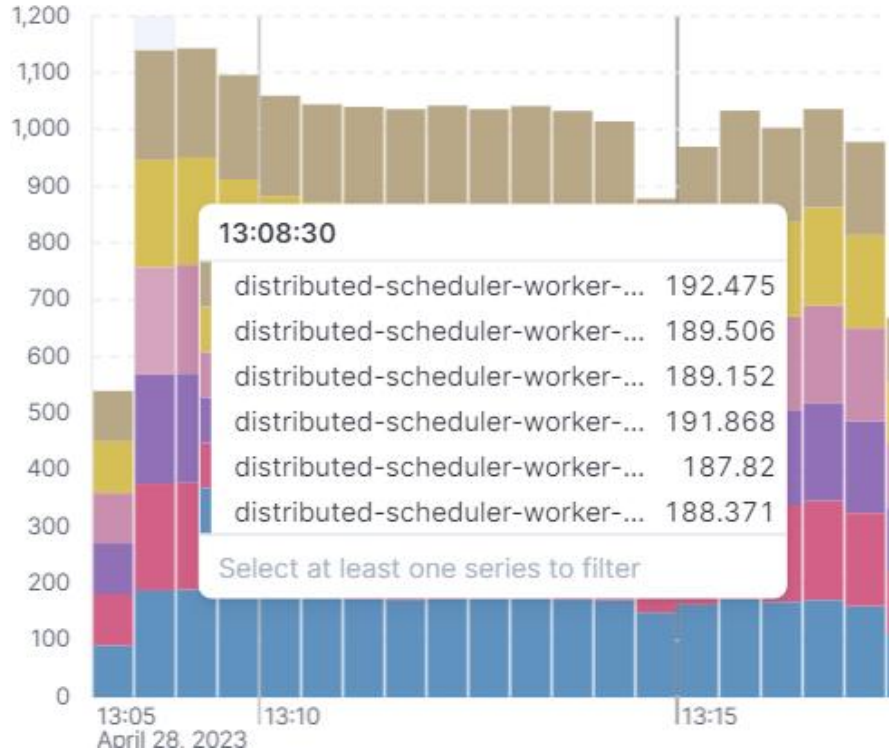


Рисунок 3.13 - Показники виконання завдань за 1 секуну з використанням бази даних PostgreSQL

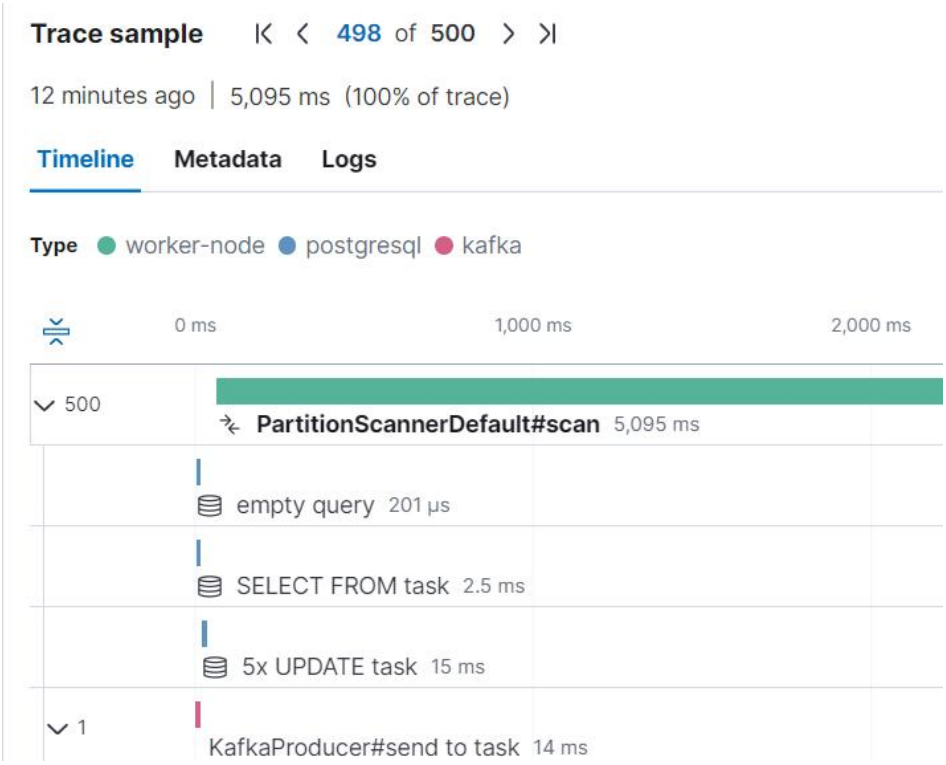


Рисунок 3.14 - Транзакція пошуку завдань з використанням бази даних PostgreSQL

4 ІНФРАСТРУКТУРА ТА РОЗГОРТАННЯ

4.1 Інфраструктура

Для розгортання системи було встановлено та налаштовано Proxmox Virtual Environment 7.4 Hypervisor використовуючи локальний сервер.

Характеристики серверу: 16 CPU, 64 RAM, 2 TB SSD, 100 Mbps Ethernet.

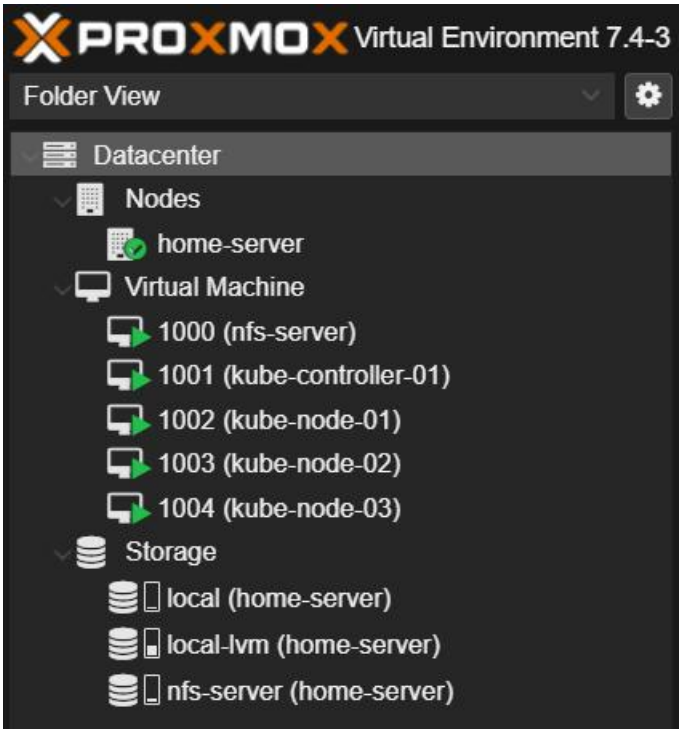


Рисунок 4.1 - Список віртуальних машин

— nfs-server.

Було налаштовано NFS сервер для підтримки спільного доступу всіх віртуальних машин до одного мережевого сховища. Характеристики віртуальної машини: 2 CPU, 8 RAM, 1 TB SSD;

— kube-controller-01.

Було налаштовано головний сервер для управління Kubernetes кластеру. Характеристики віртуальної машини: 2 CPU, 4 RAM, 32 GB SSD;

— kube-node-01, kube-node-02, kube-node-03.

Було налаштовано 3 сервери для вузлів Kubernetes на яких будуть запускатися компоненти системи. Характеристики кожної віртуальної машини: 4 CPU, 16 RAM, 32 GB SSD;

4.2 Розгортання

Таблиця 4.1 - Список розгорнутих компонентів

Назва продукту	Назва компоненту	Кількість копій	Метод розгортання
Kafka		3	Terraform, Kubernetes
Zookeeper		3	Terraform, Kubernetes
MongoDB	Mongos	2	Terraform, Kubernetes
MongoDB	Config Server	2	Terraform, Kubernetes
MongoDB	Shard 1	2	Terraform, Kubernetes
MongoDB	Shard 2	2	Terraform, Kubernetes
MongoDB	Shard 3	2	Terraform, Kubernetes
PostgreSQL		1	Terraform, Kubernetes
Elasticsearch		3	Terraform, Kubernetes
Elasticsearch	APM	3	Terraform, Kubernetes
Elasticsearch	Kibana	1	Terraform, Kubernetes
Система обробки відкладених завдань		6	Kubernetes
REST API Gateway		1	Kubernetes
Веб інтерфейс		1	Локально

На Таблиці 4.1 зображено компоненти які було розгорнуто на раніше встановлений Kubernetes кластер.

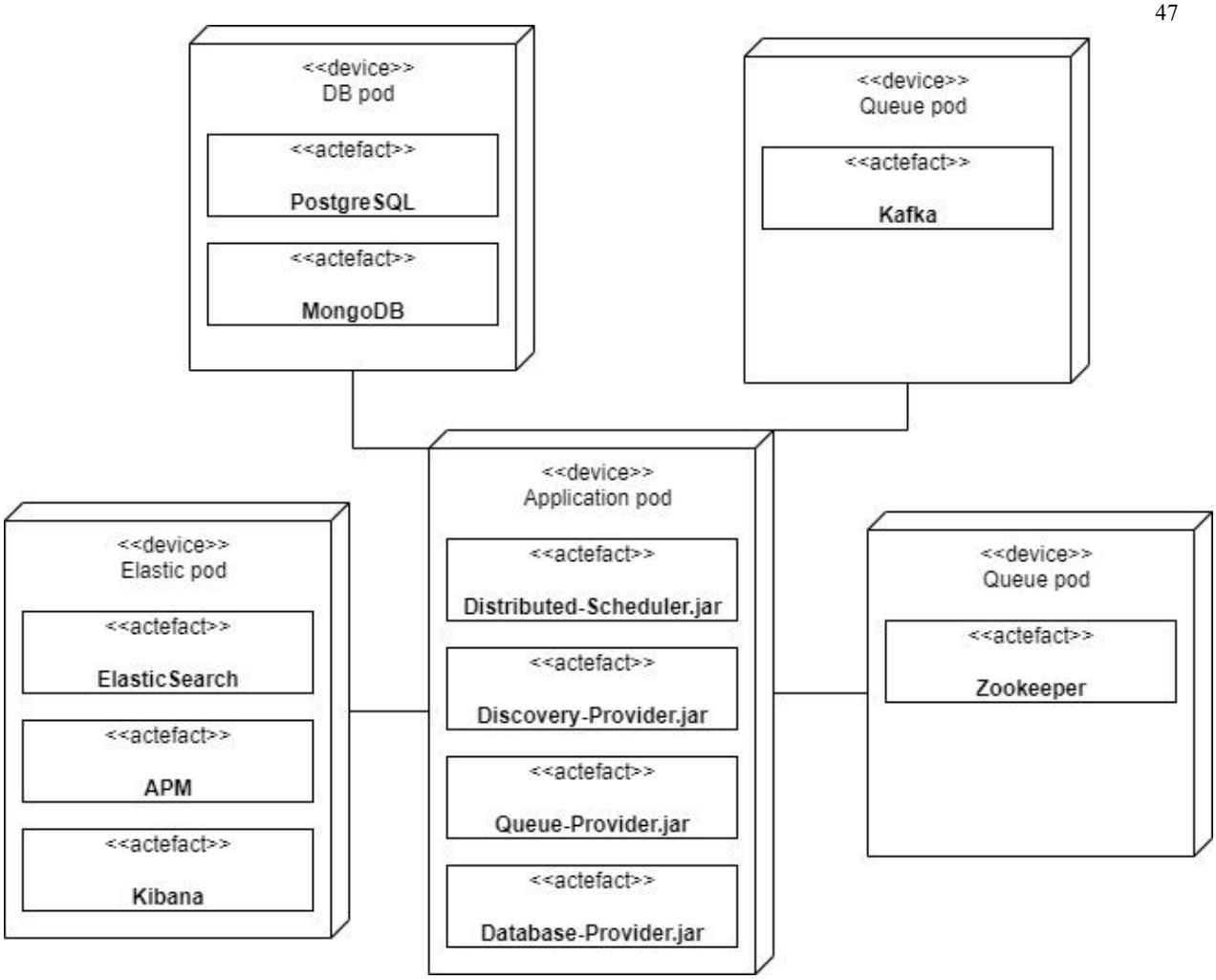


Рисунок 4.2 - Діаграма розгортання

На рисисунку 4.2 зображена діаграма розгортання.

Всі компоненти системи було розгорнуто використовуючи Kubernetes кластер. Для розгортання системи виконання відкладених завдань було додатково розгорнуто приватний Docker Registry щоб була можливість зібрати JAR файли додатку в Docker контейнер та в подальшому розгорнути використовуючи Kubernetes. Всі інші компоненти вже знаходилися в публічному Docker Registry.

```
NAME                                                READY   STATUS
pod/apm-server-apm-server-fbc7f5d6c-zxwmq           1/1     Running
pod/elasticsearch-es-default-0                      1/1     Running
pod/elasticsearch-es-default-1                      1/1     Running
pod/elasticsearch-es-default-2                      1/1     Running
pod/kafka-0                                         1/1     Running
pod/kafka-1                                         1/1     Running
pod/kafka-2                                         1/1     Running
pod/kibana-kb-65d4d49fbd-qf6wb                      1/1     Running
pod/mongo-mongodb-sharded-configsvr-0               1/1     Running
pod/mongo-mongodb-sharded-configsvr-1               1/1     Running
pod/mongo-mongodb-sharded-mongos-5c547f67f5-2589f   1/1     Running
pod/mongo-mongodb-sharded-mongos-5c547f67f5-64w58   1/1     Running
pod/mongo-mongodb-sharded-shard0-data-0             1/1     Running
pod/mongo-mongodb-sharded-shard0-data-1             1/1     Running
pod/mongo-mongodb-sharded-shard1-data-0             1/1     Running
pod/mongo-mongodb-sharded-shard1-data-1             1/1     Running
pod/mongo-mongodb-sharded-shard2-data-0             1/1     Running
pod/mongo-mongodb-sharded-shard2-data-1             1/1     Running
pod/postgres-postgresql-0                           1/1     Running
pod/zookeeper-0                                     1/1     Running
pod/zookeeper-1                                     1/1     Running
pod/zookeeper-2                                     1/1     Running
```

Рисунок 4.2 - Список Kubernetes подів компонентів системи виконання відкладених завдань

```
NAME                              TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)
apm-server-apm-http               ClusterIP      10.98.234.172    <none>          8200/TCP
apm-server-apm-http-headless      ClusterIP      None             <none>          8200/TCP
elasticsearch-es-default          ClusterIP      None             <none>          9200/TCP
elasticsearch-es-http             ClusterIP      10.103.118.242   <none>          9200/TCP
elasticsearch-es-internal-http    ClusterIP      10.99.34.163     <none>          9200/TCP
elasticsearch-es-transport        ClusterIP      None             <none>          9300/TCP
kafka                             ClusterIP      10.97.80.12      <none>          9092/TCP
kafka-headless                    ClusterIP      None             <none>          9092/TCP,9093/TCP,9095/TCP
kibana-kb-http                    LoadBalancer   10.103.112.220   192.168.31.10   5601:31700/TCP
mongo-mongodb-sharded             ClusterIP      10.111.100.49    <none>          27017/TCP
mongo-mongodb-sharded-headless    ClusterIP      None             <none>          27017/TCP
mongo-mongos-sharded-headless     ClusterIP      None             <none>          27017/TCP
postgres-postgresql               ClusterIP      10.103.218.185   <none>          5432/TCP
postgres-postgresql-hl            ClusterIP      None             <none>          5432/TCP
zookeeper                         ClusterIP      10.96.114.63     <none>          2181/TCP,2888/TCP,3888/TCP
zookeeper-headless                ClusterIP      None             <none>          2181/TCP,2888/TCP,3888/TCP
```

Рисунок 4.3 - Список Kubernetes сервісів компонентів системи виконання відкладених завдань

На рисунках 4.2, 4.3, 4.4, 4.5 зображено розгорнуті компоненти системи виконання відкладених завдань.

```
NAME                                                        READY   STATUS
pod/distributed-scheduler-rest-api-gateway-755b9657b-qj45p  1/1     Running
pod/distributed-scheduler-worker-56458fd995-5mnf9           1/1     Running
pod/distributed-scheduler-worker-56458fd995-ggp2b           1/1     Running
pod/distributed-scheduler-worker-56458fd995-gq7mr           1/1     Running
pod/distributed-scheduler-worker-56458fd995-qlhnr           1/1     Running
pod/distributed-scheduler-worker-56458fd995-v9lz4           1/1     Running
pod/distributed-scheduler-worker-56458fd995-wwdrt           1/1     Running
```

Рисунок 4.4 - Список Kubernetes подів системи виконання відкладених завдань

```
NAME                                  TYPE           CLUSTER-IP       EXTERNAL-IP     PORT(S)
distributed-scheduler-rest-api-gateway LoadBalancer  10.108.150.50    192.168.31.12   8000:30792/TCP
distributed-scheduler-worker          ClusterIP      10.109.34.248    <none>          8080/TCP
```

Рисунок 4.5 - Список Kubernetes сервісів системи виконання відкладених завдань

Kubernetes дозволяє розширювати систему за допомогою додавання нових вузлів до кластеру. Також є можливість збільшувати кількість системних сервісів в ручному або в автоматичному режимі в залежності від навантаження на систему.

Terraform допомагає автоматизувати процес створення та управління інфраструктурою за допомогою написання інфраструктурного коду.

ВИСНОВКИ

В ході виконання дипломної роботи була розроблена дистрибутивна системи виконання відкладених завдань та веб-інтерфейс для моніторингу. Був застосований повний цикл розробки програмного забезпечення який включає:

1. Аналіз обов’язків розроблюваної системи;
2. Планування архітектури та підбір технології;
3. Розробка системи;
4. Тестування системи;
5. Підключення моніторингу продуктивності до системи;
6. Розробка інфраструктури та розгортання системи;

Під час проведення аналізу предметної області були обрані та описані засоби для розробки дистрибутивних систем такі як: Java, Maven, Spring, MongoDB, PostgreSQL, Zookeeper, Kafka, Elasticsearch, Kibana, Angular, Terraform, Kubernetes, Proxmox.

У процесі розробки було реалізовано всі необхідні підсистеми проекту:

1. Основний модуль системи (Core);
2. Модуль для виконання завдань (Worker);
3. Модуль для балансування запитів між вузлами (Gateway);
4. Модуль для реєстрації вузлів (Discovery Provider);
5. Модуль для обміну повідомленнями між вузлами (Queue Provider);
6. Модуль для зберігання завдань (Database Provider);
7. Модуль веб-інтерфейсу для перегляду завдань (UI);

Система має гнучку модульну архітектуру з використанням провідних шаблонів проектування. Систему було налагоджено, протестовано та випробувано, що показало, коректність роботи програмного забезпечення. Система є ефективною та вирішує всі поставлені задачі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 615 p.

2. Java. [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/java/javase/17/docs/api/index.html

3. Maven. [Електронний ресурс]. – Режим доступу: https://maven.apache.org/

4. Spring Framework. [Електронний ресурс]. – Режим доступу: https://spring.io

5. MongoDB Sharding. [Електронний ресурс]. – Режим доступу: https://www.mongodb.com/docs/manual/sharding/

6. PostgreSQL. [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/docs/current/transaction-iso.html

7. Apache Zookeeper, Service Discovery. [Електронний ресурс]. – Режим доступу: https://curator.apache.org/curator-x-discovery/index.html

8. Apache Kafka. [Електронний ресурс]. – Режим доступу: https://kafka.apache.org

9. Elasticsearch, Kibana. [Електронний ресурс]. – Режим доступу: https://www.elastic.co/guide/en/cloud-on-k8s/current/index.html

10. Angular. [Електронний ресурс]. – Режим доступу: https://angular.io/docs

11. Terraform. [Електронний ресурс]. – Режим доступу: https://www.terraform.io

12. Kubernetes. [Електронний ресурс]. – Режим доступу: https://kubernetes.io/docs/home

13. Proxmox. [Електронний ресурс]. – Режим доступу: https://www.proxmox.com/en/proxmox-ve

ДОДАТОК А. ПРЕЗЕНТАЦІЯ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка дистрибутивної системи виконання відкладених завдань мовою Java

Виконав студент 4 курсу
групи ПД-43
Зайцев Іван Сергійович
Керівник роботи
Гаманюк Ігор Михайлович

Київ – 2023

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи - підвищення швидкості та кількості виконання відкладених завдань за рахунок використання дистрибутивної системи розробленої мовою Java.

Об'єкт дослідження - виконання відкладених завдань.

Предмет дослідження - дистрибутивна система для виконання відкладених завдань.

2

ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Аналіз обов'язків розроблюваної системи.
2. Аналіз та дослідження існуючих аналогів.
3. Розробка системи виконання відкладених завдань та веб-інтерфесу.
   3.1. Розробити модуль для виконання завдань.
   3.2. Розробити модуль для балансування запитів між вузлами.
   3.3. Розробити модуль для реєстрації вузлів.
   3.4. Розробити модуль для обміну повідомленнями між вузлами.
   3.5. Розробити модуль для зберігання завдань.
   3.6. Розробити модуль для перегляду завдань.
4. Тестування системи виконання відкладених завдань.
5. Підключення моніторингу продуктивності до системи.
6. Розробка інфраструктури, розгортання та запуск системи.
   6.1. Налаштувати гіпервізор Proxmox.
   6.2. Встановити Kubernetes кластер.
   6.3. Розробити Terraform інфраструктуру.

3

АНАЛІЗ АНАЛОГІВ

	Quartz	JobRunr	Розроблена система
Схема роботи	Кожен із вузлів опитує базу даних на наявність всіх завдань	Кожен із вузлів опитує базу даних на наявність всіх завдань	Кожен із вузлів опитує базу даних на наявність завдань тільки свого логічного розділу
Система моніторингу	Немає	Тільки система перегляду завдань	Система перегляду завдань та система моніторингу продуктивності
Виконання завдань	Затримка від 1 секунд перед виконанням завдання	Затримка від 5 секунд перед виконанням завдання	Затримка від 0.1 секунди пред виконанням завдання
Підтримка різних систем управління базами даних	Так	Так	Так

4

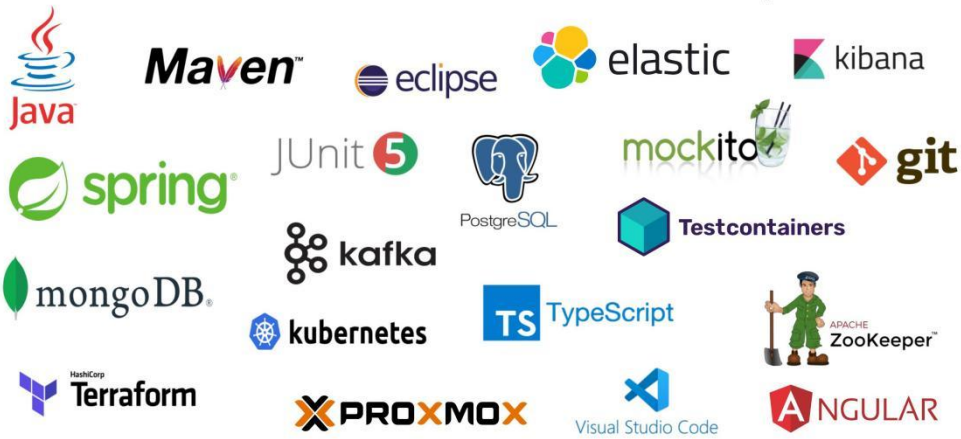
ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги:
1. Можливість створення відкладених завдань.
2. Можливість підключення різних модулів та баз даних.
3. Веб-інтерфейс для перегляду основної інформації.
4. Можливість моніторингу продуктивності.

Не функціональні вимоги:
1. Підтримка горизонтального маштабування.
2. Завдання не повинно пропасти або бути виконано декілька раз в разі помилки.
3. Безперебійне продовження роботи у випадках відмови декількох вузлів.
4. Забезпечення низької затримки для обробки та виконання завдань.

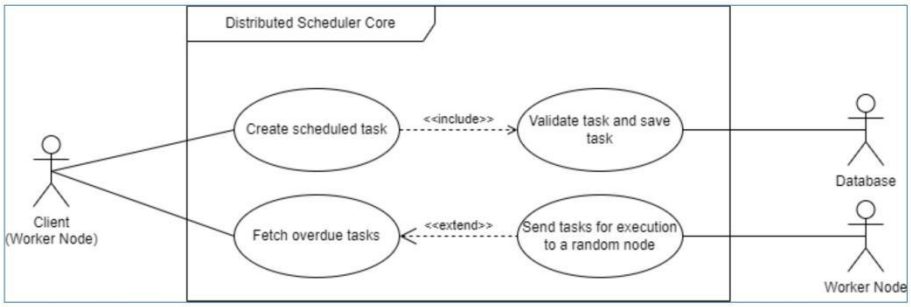
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



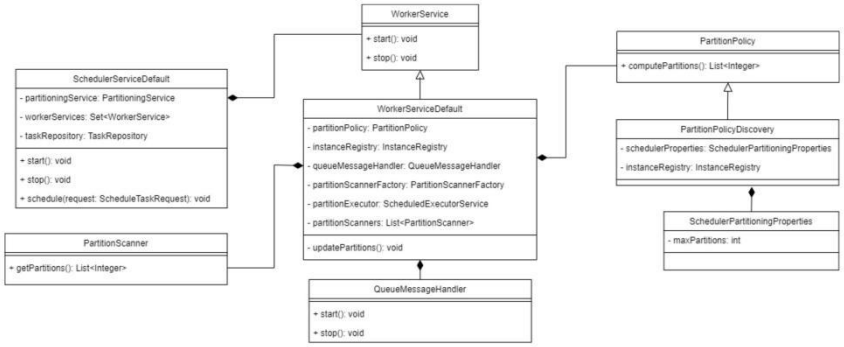
6

ДІАГРАМА ПРЕЦЕДЕНТІВ



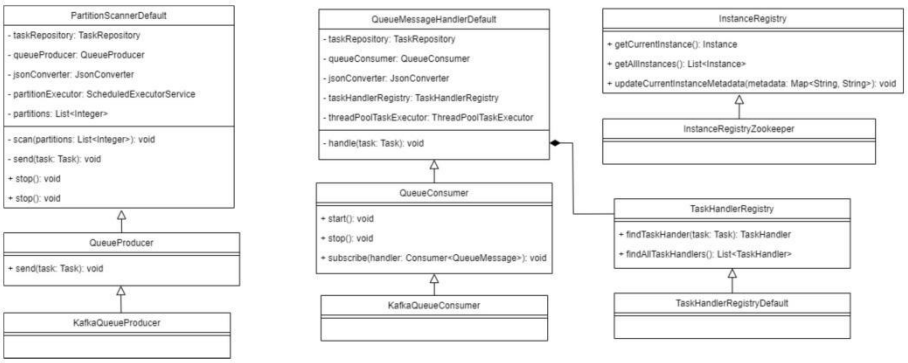
7

РЕАЛІЗАЦІЯ БІЗНЕС ЛОГІКИ ВИКОНАННЯ ЗАВДАНЬ



8

РЕАЛІЗАЦІЯ БІЗНЕС ЛОГІКИ ВИКОНАННЯ ЗАВДАНЬ



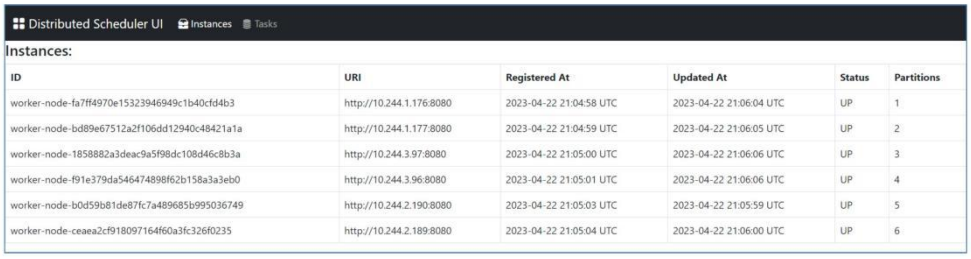
9

СХЕМА БАЗИ ДАНИХ

Таблиця бази даних tasks		
Назва поля	Тип	Можливі значення
partition	int	
id	UUIDv7	
status	enum	SCHEDULED, SUBMITTED, PROCESSING, SUCCEEDED, FAILED
executeAt	timestamp	
name	varchar	
data	varchar	

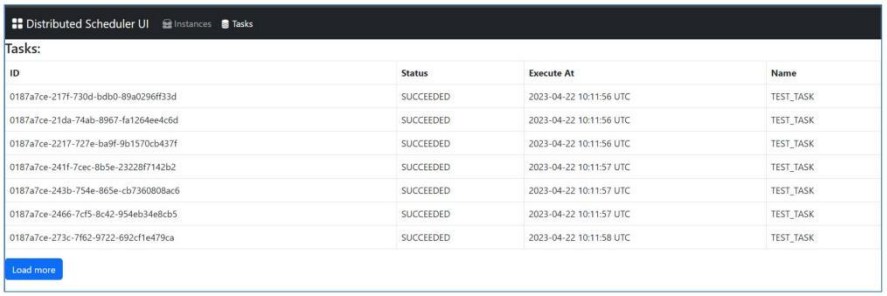
10

СПИСОК ВУЗЛІВ СИСТЕМИ



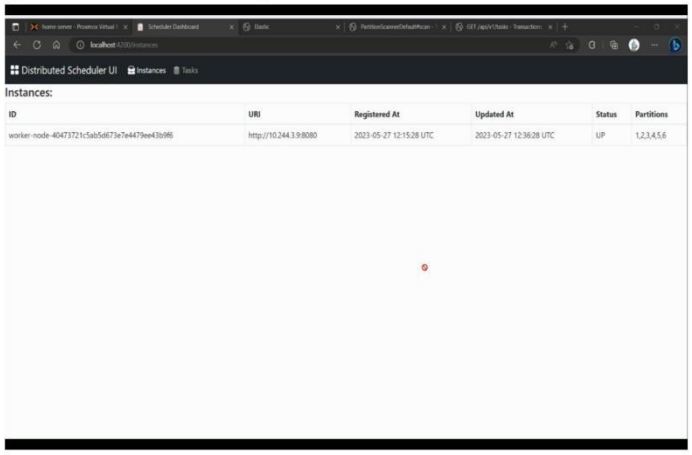
11

СПИСОК ЗАВДАНЬ СИСТЕМИ



12

ДЕМОНСТРАЦІЯ РОБОТИ СИСТЕМИ



13

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

1. Зайцев І.С. Використання Service Discovery для горизонтального масштабування при створенні дистрибутивної системи виконання відкладених завдань / І.С. Зайцев, І.М. Гаманюк // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез 20.04.2023 ДУТ, м. Київ — К.: ДУТ, 2023. — С. 56.

2. Зайцев І.С. Використання NoSQL баз даних для горизонтального масштабування при створенні дистрибутивної системи виконання відкладених завдань / І.С. Зайцев, І.М. Гаманюк // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез 20.04.2023 ДУТ, м. Київ — К.: ДУТ, 2023. — С. 96.

3. Зайцев І.С. Використання Kafka для горизонтального масштабування та комунікації при створенні дистрибутивної системи виконання відкладених завдань / І.С. Зайцев, І.М. Гаманюк // Всеукраїнська науково-технічна конференція «Застосування програмного забезпечення в ІКТ». Збірник тез 20.04.2023 ДУТ, м. Київ — К.: ДУТ, 2023. — С. 142.

14

ВИСНОВКИ

1. Проведено детальний аналіз предметної галузі та аналогічних систем які використовуються на поточний момент для дистрибутивного виконання відкладених завдань. Наявні системи не можуть забезпечити швидкого виконання відкладених завдань в великій кількості.
2. Розроблено фронтенд частину мовою Typescript та бекенд частину мовою Java. Система розроблена з використанням модулів що дозволяє підключати різні системи управління базами даних такі як PostgreSQL, MongoDB та інші, різні системи обміну повідомленнями такі як Kafka та інші.
3. Протестовано систему з використанням юніт, інтеграційних, та тестів продуктивності. Використано провідні бібліотеки тестування такі як JUnit, Mockito, Testcontainers.
4. Підключено моніторинг Elastic APM та Kibana для аналізу продуктивності в реальному часі.
5. Розроблено інфраструктуру використовуючи інструмент інфраструктурного програмування Terraform.
6. Розгорнуто систему використовуючи систему управління контейнерами Kubernetes та Proxmox гіпервізор.

15

ДЯКУЮ ЗА УВАГУ!

16

ДОДАТОК Б. КОД СИСТЕМИ

zookeeper-kafka-postgresql/application/scheduler-rest-api-gateway/src/main/java/ua/ivan909020/scheduler/worker/Application.java

```
package ua.ivan909020.scheduler.worker;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import co.elastic.apm.attach.ElasticApmAttacher;

@SpringBootApplication
public class Application {

   public static void main(String[] args) {
      ElasticApmAttacher.attach();

      SpringApplication.run(Application.class, args);
   }

}
```

zookeeper-kafka-postgresql/application/scheduler-rest-api-gateway/src/main/resources/application.yaml

```
# server configuration
server:
  port: 8000

# spring configuration
spring:
  application:
    name: rest-api-gateway

  cloud:
    zookeeper:
      connect-string: zookeeper-headless.scheduler.svc.cluster.local:2181
      discovery:
        prefer-ip-address: true

    gateway:
      globalcors:
        cors-configurations:
```

```
        '[/**]':
          allowed-origins: '*'
          allowed-methods: '*'
          allowed-headers: '*'

      routes:
        - id: worker-node
          uri: lb://worker-node
          predicates:
            - Path=/worker-node/**
          filters:
            - RewritePath=/worker-node/(?<segment>.*), /$\{segment}
```

zookeeper-kafka-postgresql/application/scheduler-rest-api-gateway/src/main/resources/elasticapm.properties

```
service_name=scheduler-rest-api-gateway-zookeeper-kafka-postgresql
application_packages=ua.ivan909020
server_url=http://apm-server-apm-http-headless.scheduler.svc.cluster.local:8200
log_sending=true
use_path_as_transaction_name=true
```

zookeeper-kafka-postgresql/application/scheduler-rest-api-gateway/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.5</version>
        <relativePath />
    </parent>

    <groupId>ua.ivan909020.scheduler</groupId>
    <artifactId>distributed-scheduler-rest-api-gateway-zookeeper-kafka-
postgresql</artifactId>
    <version>1.0.0</version>

    <properties>
```

```
    <java.version>17</java.version>
    <maven.compiler.source>${java.version}</maven.compiler.source>
    <maven.compiler.target>${java.version}</maven.compiler.target>

    <spring.cloud.version>2022.0.1</spring.cloud.version>
    <elastic.apm.version>1.37.0</elastic.apm.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-webflux</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-zookeeper-discovery</artifactId>
    </dependency>

    <dependency>
        <groupId>co.elastic.apm</groupId>
        <artifactId>apm-agent-attach</artifactId>
        <version>${elastic.apm.version}</version>
    </dependency>
</dependencies>

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring.cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```
    <build>
        <finalName>distributed-scheduler-rest-api-gateway</finalName>

        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

zookeeper-kafka-postgresql/application/scheduler-worker/src/main/java/ua/ivan909020/scheduler/worker/Application.java

```
package ua.ivan909020.scheduler.worker;

import java.time.Duration;
import java.time.Instant;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.atomic.AtomicInteger;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.event.InstanceRegisteredEvent;
import org.springframework.context.annotation.Bean;
import org.springframework.context.event.EventListener;

import co.elastic.apm.attach.ElasticApmAttacher;
import ua.ivan909020.scheduler.core.model.domain.task.ScheduleTaskRequest;
import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.service.core.SchedulerService;
import ua.ivan909020.scheduler.core.service.handler.TaskHandler;

@SpringBootApplication
public class Application {

    private final SchedulerService schedulerService;

    public Application(SchedulerService schedulerService) {
        this.schedulerService = schedulerService;
    }
```

```
    @EventListener(InstanceRegisteredEvent.class)
    public void handleInstanceRegisteredEvent() {
        ExecutorService executorService = Executors.newFixedThreadPool(8);

        Instant timestamp = Instant.now().plus(Duration.ofMinutes(5));
        AtomicInteger counter = new AtomicInteger(0);

        for (int i = 0; i < 100000; i++) {
            executorService.submit(() -> {
                ScheduleTaskRequest request = new ScheduleTaskRequest();
                request.setExecuteAt(timestamp);
                request.setName(String.format("TEST_TASK_%d",
counter.incrementAndGet()));
                request.setData("");

                schedulerService.schedule(request);
            });
        }

        schedulerService.start();
    }

    @Bean
    public TaskHandler taskHandler() {
        return new TaskHandler() {

            @Override
            public boolean supports(Task task) {
                return task.getName().startsWith("TEST_TASK");
            }

            @Override
            public void handle(Task task) {
                System.out.println("HANDLED: " + task);
            }

        };
    }

    public static void main(String[] args) {
        ElasticApmAttacher.attach();

        SpringApplication.run(Application.class, args);
    }
```

```
}
```

zookeeper-kafka-postgresql/application/scheduler-worker/src/main/resources/application.yaml

```
# server configuration
server:
  port: 8080

# spring configuration
spring:
  application:
    name: ${scheduler.group-id}

  cloud:
    zookeeper:
      connect-string: zookeeper-headless.scheduler.svc.cluster.local:2181
      discovery:
        instance-id: ${scheduler.instance-id}
        prefer-ip-address: true

  kafka:
    bootstrap-servers: kafka-headless.scheduler.svc.cluster.local:9092
    consumer:
      group-id: ${scheduler.group-id}
    producer:
      batch-size: 1MB
      properties:
       linger.ms: 10

  datasource:
    url: jdbc:postgresql://postgres-postgresql-hl.scheduler.svc.cluster.local:5432/scheduler
    username: postgres
    password: password

  liquibase:
    change-log: classpath:migration/changelog.xml

# scheduler configuration
scheduler:
  group-id: worker-node
  instance-id: ${scheduler.group-id}-${random.value}
  max-partitions: 6
  task-fetch-interval: 2s
```

```
  task-fetch-limit: 1000
  queue-topic: task
```

zookeeper-kafka-postgresql/application/scheduler-worker/src/main/resources/elasticapm.properties

```
service_name=scheduler-worker-zookeeper-kafka-postgresql
application_packages=ua.ivan909020
server_url=http://apm-server-apm-http-headless.scheduler.svc.cluster.local:8200
log_sending=true
use_path_as_transaction_name=true
trace_methods=ua.ivan909020.scheduler.core.service.worker.scanner.PartitionScanner
Default#scan
```

zookeeper-kafka-postgresql/application/scheduler-worker/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.0.5</version>
        <relativePath />
    </parent>

    <groupId>ua.ivan909020.scheduler</groupId>
    <artifactId>distributed-scheduler-worker-zookeeper-kafka-postgresql</artifactId>
    <version>1.0.0</version>

    <properties>
        <java.version>17</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>

        <distributed.scheduler.version>1.0.0</distributed.scheduler.version>
        <elastic.apm.version>1.37.0</elastic.apm.version>
    </properties>

    <dependencies>
```

```
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>

    <dependency>
      <groupId>ua.ivan909020.scheduler</groupId>
      <artifactId>distributed-scheduler-rest-api</artifactId>
      <version>${distributed.scheduler.version}</version>
    </dependency>
    <dependency>
      <groupId>ua.ivan909020.scheduler</groupId>
      <artifactId>distributed-scheduler-discovery-provider-zookeeper</artifactId>
      <version>${distributed.scheduler.version}</version>
    </dependency>
    <dependency>
      <groupId>ua.ivan909020.scheduler</groupId>
      <artifactId>distributed-scheduler-queue-provider-kafka</artifactId>
      <version>${distributed.scheduler.version}</version>
    </dependency>
    <dependency>
      <groupId>ua.ivan909020.scheduler</groupId>
      <artifactId>distributed-scheduler-database-provider-postgresql</artifactId>
      <version>${distributed.scheduler.version}</version>
    </dependency>

    <dependency>
      <groupId>co.elastic.apm</groupId>
      <artifactId>apm-agent-attach</artifactId>
      <version>${elastic.apm.version}</version>
    </dependency>
  </dependencies>

  <build>
    <finalName>distributed-scheduler-worker</finalName>

    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
```

```
        </plugins>
    </build>

</project>
```

scheduler-core/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>ua.ivan909020.scheduler</groupId>
        <artifactId>distributed-scheduler-parent</artifactId>
        <version>1.0.0</version>
        <relativePath>../pom.xml</relativePath>
    </parent>

    <artifactId>distributed-scheduler-core</artifactId>
    <packaging>jar</packaging>

    <properties>
        <uuid.generator.version>4.1.0</uuid.generator.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-configuration-processor</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-commons</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.integration</groupId>
            <artifactId>spring-integration-core</artifactId>
```

```
        </dependency>

        <dependency>
            <artifactId>jackson-databind</artifactId>
            <groupId>com.fasterxml.jackson.core</groupId>
        </dependency>
        <dependency>
            <groupId>com.fasterxml.uuid</groupId>
            <artifactId>java-uuid-generator</artifactId>
            <version>${uuid.generator.version}</version>
        </dependency>
    </dependencies>

</project>
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/configuration/WorkerAutoConfiguration.java

```
package ua.ivan909020.scheduler.core.configuration;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import com.fasterxml.jackson.databind.ObjectMapper;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.queue.QueueConsumer;
import ua.ivan909020.scheduler.core.queue.QueueProducer;
import ua.ivan909020.scheduler.core.repository.TaskRepository;
import ua.ivan909020.scheduler.core.service.converter.JsonConverter;
import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;
import ua.ivan909020.scheduler.core.service.handler.TaskHandlerRegistry;
import ua.ivan909020.scheduler.core.service.worker.WorkerServiceDefault;
import ua.ivan909020.scheduler.core.service.worker.policy.PartitionPolicy;
import ua.ivan909020.scheduler.core.service.worker.policy.PartitionPolicyDiscovery;
import ua.ivan909020.scheduler.core.service.worker.scanner.PartitionScannerFactory;
import
ua.ivan909020.scheduler.core.service.worker.scanner.PartitionScannerFactoryDefault;
import
ua.ivan909020.scheduler.core.service.worker.scanner.handler.QueueMessageHandler;
```

```
import
ua.ivan909020.scheduler.core.service.worker.scanner.handler.QueueMessageHandlerDe
fault;

@Configuration
public class WorkerAutoConfiguration {

   @Bean
   @ConditionalOnMissingBean
   public JsonConverter jsonConverter(ObjectMapper objectMapper) {
      return new JsonConverter(objectMapper);
   }

   @Bean
   @ConditionalOnMissingBean
   public PartitionPolicy partitionPolicyDiscovery(
         SchedulerProperties schedulerProperties,
         InstanceRegistry instanceRegistry) {

      return new PartitionPolicyDiscovery(schedulerProperties, instanceRegistry);
   }

   @Bean
   @ConditionalOnMissingBean
   public PartitionScannerFactory partitionScannerFactory(
         SchedulerProperties schedulerProperties,
         TaskRepository taskRepository,
         QueueProducer queueProducer,
         JsonConverter jsonConverter) {

      return new PartitionScannerFactoryDefault(schedulerProperties, taskRepository,
queueProducer, jsonConverter);
   }

   @Bean
   @ConditionalOnMissingBean
   public QueueMessageHandler queueMessageHandler(
         TaskRepository taskRepository,
         QueueConsumer queueConsumer,
         JsonConverter jsonConverter,
         TaskHandlerRegistry taskHandlerRegistry,
         ThreadPoolTaskExecutor taskHandlerExecutor) {

      return new QueueMessageHandlerDefault(taskRepository, queueConsumer,
jsonConverter,
```

```
            taskHandlerRegistry, taskHandlerExecutor);
    }

    @Bean
    @ConditionalOnMissingBean
    public WorkerServiceDefault workerServiceDefault(
            PartitionPolicy partitionPolicy,
            InstanceRegistry instanceRegistry,
            PartitionScannerFactory partitionScannerFactory,
            QueueMessageHandler queueMessageHandler) {

        return new WorkerServiceDefault(partitionPolicy, instanceRegistry,
                partitionScannerFactory, queueMessageHandler);
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/configuration/TaskAutoConfiguration.java

```
package ua.ivan909020.scheduler.core.configuration;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.integration.util.CallerBlocksPolicy;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import ua.ivan909020.scheduler.core.service.handler.TaskHandlerRegistry;
import ua.ivan909020.scheduler.core.service.handler.TaskHandlerRegistryDefault;

@Configuration
public class TaskAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public ThreadPoolTaskExecutor taskHandlerExecutor() {
        ThreadPoolTaskExecutor taskHandlerExecutor = new ThreadPoolTaskExecutor();
        taskHandlerExecutor.setCorePoolSize(8);
        taskHandlerExecutor.setMaxPoolSize(8);
        taskHandlerExecutor.setQueueCapacity(8);
        taskHandlerExecutor.setRejectedExecutionHandler(new
CallerBlocksPolicy(Integer.MAX_VALUE));
```

```
        taskHandlerExecutor.afterPropertiesSet();
        return taskHandlerExecutor;
    }

    @Bean
    @ConditionalOnMissingBean
    public TaskHandlerRegistry taskHandlerRegistry(ApplicationContext
applicationContext) {
        return new TaskHandlerRegistryDefault(applicationContext);
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/configuration/SchedulerAutoConfiguration.java

```
package ua.ivan909020.scheduler.core.configuration;

import java.util.Set;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.repository.TaskRepository;
import ua.ivan909020.scheduler.core.service.core.PartitionService;
import ua.ivan909020.scheduler.core.service.core.PartitionServiceDefault;
import ua.ivan909020.scheduler.core.service.core.SchedulerService;
import ua.ivan909020.scheduler.core.service.core.SchedulerServiceDefault;
import ua.ivan909020.scheduler.core.service.worker.WorkerService;

@Configuration
@EnableConfigurationProperties
public class SchedulerAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public SchedulerProperties schedulerProperties() {
        return new SchedulerProperties();
    }

    @Bean
```

```
    public PartitionService partitionService(SchedulerProperties schedulerProperties) {
        return new PartitionServiceDefault(schedulerProperties);
    }

    @Bean
    @ConditionalOnMissingBean
    public SchedulerService schedulerService(
            PartitionService partitionService,
            Set<WorkerService> workerServices,
            TaskRepository taskRepository) {

        return new SchedulerServiceDefault(partitionService, workerServices,
taskRepository);
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/configuration/properties/SchedulerProperties.java

```
package ua.ivan909020.scheduler.core.configuration.properties;

import java.time.Duration;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties("scheduler")
public class SchedulerProperties {

    private String groupId;

    private String instanceId;

    private int maxPartitions;

    private Duration taskFetchInterval;

    private int taskFetchLimit;

    public String getGroupId() {
        return groupId;
    }

    public void setGroupId(String groupId) {
```

```
        this.groupId = groupId;
    }

    public String getInstanceId() {
        return instanceId;
    }

    public void setInstanceId(String instanceId) {
        this.instanceId = instanceId;
    }

    public int getMaxPartitions() {
        return maxPartitions;
    }

    public void setMaxPartitions(int maxPartitions) {
        this.maxPartitions = maxPartitions;
    }

    public Duration getTaskFetchInterval() {
        return taskFetchInterval;
    }

    public void setTaskFetchInterval(Duration taskFetchInterval) {
        this.taskFetchInterval = taskFetchInterval;
    }

    public int getTaskFetchLimit() {
        return taskFetchLimit;
    }

    public void setTaskFetchLimit(int taskFetchLimit) {
        this.taskFetchLimit = taskFetchLimit;
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/domain/instance/Instance.java

```
package ua.ivan909020.scheduler.core.model.domain.instance;

import java.time.Instant;
import java.util.List;
```

```
import org.springframework.cloud.client.ServiceInstance;

public class Instance {

    public static final String UPDATED_AT = "updated_at";
    public static final String PARTITIONS = "partitions";

    private ServiceInstance serviceInstance;

    private Instant registeredAt;

    private Instant updatedAt;

    private InstanceStatus status;

    private List<Integer> partitions;

    public ServiceInstance getServiceInstance() {
        return serviceInstance;
    }

    public void setServiceInstance(ServiceInstance serviceInstance) {
        this.serviceInstance = serviceInstance;
    }

    public Instant getRegisteredAt() {
        return registeredAt;
    }

    public void setRegisteredAt(Instant registeredAt) {
        this.registeredAt = registeredAt;
    }

    public Instant getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(Instant updatedAt) {
        this.updatedAt = updatedAt;
    }

    public InstanceStatus getStatus() {
        return status;
    }
```

```
    public void setStatus(InstanceStatus status) {
        this.status = status;
    }

    public List<Integer> getPartitions() {
        return partitions;
    }

    public void setPartitions(List<Integer> partitions) {
        this.partitions = partitions;
    }

    @Override
    public String toString() {
        return "Instance [" +
                "serviceInstance=" + serviceInstance +
                ", registeredAt=" + registeredAt +
                ", updatedAt=" + updatedAt +
                ", status=" + status +
                ", partitions=" + partitions + "]";
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/domain/instance/InstanceStatus.java

```
package ua.ivan909020.scheduler.core.model.domain.instance;

public enum InstanceStatus {

    UP, OUT_OF_SERVICE;

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/domain/page/KeysetPage.java

```
package ua.ivan909020.scheduler.core.model.domain.page;

public record KeysetPage(Integer size, String cursor) {

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/domain/page/PagedList.java

```
package ua.ivan909020.scheduler.core.model.domain.page;

import java.util.List;

public record PagedList<T>(List<T> content, String nextCursor) {

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/domain/queue/QueueMessage.java

```
package ua.ivan909020.scheduler.core.model.domain.queue;

import java.util.Objects;

public class QueueMessage {

    private String key;

    private String value;

    public String getKey() {
        return key;
    }

    public void setKey(String key) {
        this.key = key;
    }

    public String getValue() {
        return value;
    }

    public void setValue(String value) {
        this.value = value;
    }

    @Override
    public int hashCode() {
        return Objects.hash(key, value);
    }
```

```
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        QueueMessage other = (QueueMessage) obj;
        return Objects.equals(key, other.key) && Objects.equals(value, other.value);
    }

    @Override
    public String toString() {
        return "QueueMessage [key=" + key + ", value=" + value + "]";
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/domain/task/ScheduleTaskRequest.java

```
package ua.ivan909020.scheduler.core.model.domain.task;

import java.time.Instant;

public class ScheduleTaskRequest {

    private Instant executeAt;

    private String name;

    private String data;

    public Instant getExecuteAt() {
        return executeAt;
    }

    public void setExecuteAt(Instant executeAt) {
        this.executeAt = executeAt;
    }

    public String getName() {
```

```
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getData() {
        return data;
    }

    public void setData(String data) {
        this.data = data;
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/entity/Task.java

```
package ua.ivan909020.scheduler.core.model.entity;

import java.time.Instant;
import java.util.Objects;

public class Task {

    private Integer partition;

    private String id;

    private Long version;

    private TaskStatus status;

    private Instant executeAt;

    private String name;

    private String data;

    public Integer getPartition() {
        return partition;
    }

    public void setPartition(Integer partition) {
```

```
    this.partition = partition;
}

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public Long getVersion() {
    return version;
}

public void setVersion(Long version) {
    this.version = version;
}

public TaskStatus getStatus() {
    return status;
}

public void setStatus(TaskStatus status) {
    this.status = status;
}

public Instant getExecuteAt() {
    return executeAt;
}

public void setExecuteAt(Instant executeAt) {
    this.executeAt = executeAt;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public String getData() {
    return data;
```

```
    }

    public void setData(String data) {
        this.data = data;
    }
    @Override
    public int hashCode() {
        return Objects.hash(data, executeAt, id, name, partition, status, version);
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        Task other = (Task) obj;
        return Objects.equals(data, other.data) &&
                Objects.equals(executeAt, other.executeAt) &&
                Objects.equals(id, other.id) &&
                Objects.equals(name, other.name) &&
                Objects.equals(partition, other.partition) &&
                status == other.status &&
                Objects.equals(version, other.version);
    }

    @Override
    public String toString() {
        return "Task [" +
                "partition=" + partition +
                ", id=" + id +
                ", version=" + version +
                ", status=" + status +
                ", executeAt=" + executeAt +
                ", name=" + name +
                ", data=" + data + "]";
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/model/entity/TaskStatus.java

```
package ua.ivan909020.scheduler.core.model.entity;

public enum TaskStatus {

    SCHEDULED,
    SUBMITTED,
    PROCESSING,
    SUCCEEDED,
    FAILED

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/queue/QueueConsumer.java

```
package ua.ivan909020.scheduler.core.queue;

import java.util.function.Consumer;

import ua.ivan909020.scheduler.core.model.domain.queue.QueueMessage;

public interface QueueConsumer {

    void start();

    void stop();

    void subscribe(Consumer<QueueMessage> handler);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/queue/QueueProducer.java

```
package ua.ivan909020.scheduler.core.queue;

import ua.ivan909020.scheduler.core.model.domain.queue.QueueMessage;

public interface QueueProducer {

    void send(QueueMessage message);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/repository/TaskRepository.java

```
package ua.ivan909020.scheduler.core.repository;

import java.time.Instant;
import java.util.List;

import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;

public interface TaskRepository {

    void create(Task task);

    void updateStatus(Task task);

    List<Task> findAllOverdue(List<Integer> partitions, List<TaskStatus> statuses,
Instant timestamp, int limit);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/repository/TaskRepositoryExtended.java

```
package ua.ivan909020.scheduler.core.repository;

import java.util.List;

import ua.ivan909020.scheduler.core.model.domain.page.KeysetPage;
import ua.ivan909020.scheduler.core.model.domain.page.PagedList;
import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;

public interface TaskRepositoryExtended extends TaskRepository {

    PagedList<Task> findAll(List<Integer> partitions, List<TaskStatus> statuses,
KeysetPage page);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/converter/JsonConverter.java

```
package ua.ivan909020.scheduler.core.service.converter;
```

```
import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;

public class JsonConverter {

   private final ObjectMapper objectMapper;

   public JsonConverter(ObjectMapper objectMapper) {
      this.objectMapper = objectMapper;
   }

   public String convertToString(Object value) {
      try {
         return objectMapper.writeValueAsString(value);
      } catch (JsonProcessingException e) {
         throw new IllegalStateException("Failed to convert object to string", e);
      }
   }

   public <T> T convertToObject(String value, Class<T> type) {
      try {
         return objectMapper.readValue(value, type);
      } catch (JsonProcessingException e) {
         throw new IllegalStateException("Failed to convert string to object", e);
      }
   }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/PartitionConverter.java

```
package ua.ivan909020.scheduler.core.service.converter;

import java.util.Arrays;
import java.util.List;
import java.util.stream.Collectors;

public class PartitionConverter {

   public static String toString(List<Integer> partitions) {
      return partitions.stream().map(String::valueOf).collect(Collectors.joining(","));
   }

   public static List<Integer> toList(String partitions) {
```

```
        if (partitions.isEmpty()) {
            return List.of();
        }
        return
Arrays.stream(partitions.split(",")).mapToInt(Integer::parseInt).boxed().toList();
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/core/PartitionService.java

```
package ua.ivan909020.scheduler.core.service.core;

import java.util.List;

public interface PartitionService {

    List<Integer> getAll();

    int generate();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/core/PartitionServiceDefault.java

```
package ua.ivan909020.scheduler.core.service.core;

import java.util.List;
import java.util.Random;
import java.util.stream.IntStream;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;

public class PartitionServiceDefault implements PartitionService {

    private final Random random = new Random();

    private final SchedulerProperties schedulerProperties;

    public PartitionServiceDefault(SchedulerProperties schedulerProperties) {
        this.schedulerProperties = schedulerProperties;
    }
```

```
    @Override
    public List<Integer> getAll() {
        return IntStream.rangeClosed(1,
schedulerProperties.getMaxPartitions()).boxed().toList();
    }

    @Override
    public int generate() {
        return random.nextInt(schedulerProperties.getMaxPartitions()) + 1;
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/core/SchedulerService.java

```
package ua.ivan909020.scheduler.core.service.core;

import ua.ivan909020.scheduler.core.model.domain.task.ScheduleTaskRequest;

public interface SchedulerService {

    void start();

    void stop();

    void schedule(ScheduleTaskRequest request);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/core/SchedulerServiceDefault.java

```
package ua.ivan909020.scheduler.core.service.core;

import java.util.Set;

import com.fasterxml.uuid.Generators;

import ua.ivan909020.scheduler.core.model.domain.task.ScheduleTaskRequest;
import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.core.repository.TaskRepository;
```

```
import ua.ivan909020.scheduler.core.service.worker.WorkerService;

public class SchedulerServiceDefault implements SchedulerService {

    private final PartitionService partitionService;

    private final Set<WorkerService> workerServices;

    private final TaskRepository taskRepository;

    public SchedulerServiceDefault(
            PartitionService partitionService,
            Set<WorkerService> workerServices,
            TaskRepository taskRepository) {

        this.partitionService = partitionService;
        this.workerServices = workerServices;
        this.taskRepository = taskRepository;
    }

    @Override
    public void start() {
        workerServices.forEach(WorkerService::start);
    }

    @Override
    public void stop() {
        workerServices.forEach(WorkerService::stop);
    }

    @Override
    public void schedule(ScheduleTaskRequest request) {
        Task task = new Task();
        task.setPartition(partitionService.generate());
        task.setId(Generators.timeBasedEpochGenerator().generate().toString());
        task.setVersion(1L);
        task.setStatus(TaskStatus.SCHEDULED);
        task.setExecuteAt(request.getExecuteAt());
        task.setName(request.getName());
        task.setData(request.getData());

        taskRepository.create(task);
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/discovery/InstanceRegistry.java

```
package ua.ivan909020.scheduler.core.service.discovery;

import java.util.List;
import java.util.Map;

import ua.ivan909020.scheduler.core.model.domain.instance.Instance;

public interface InstanceRegistry {

    Instance getCurrentInstance();

    void updateCurrentInstanceMetadata(Map<String, String> metadata);

    List<Instance> getAllInstances();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/handler/TaskHandler.java

```
package ua.ivan909020.scheduler.core.service.handler;

import ua.ivan909020.scheduler.core.model.entity.Task;

public interface TaskHandler {

    boolean supports(Task task);

    void handle(Task task);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/handler/TaskHandlerRegistry.java

```
package ua.ivan909020.scheduler.core.service.handler;

import java.util.List;
```

```
import ua.ivan909020.scheduler.core.model.entity.Task;

public interface TaskHandlerRegistry {

    TaskHandler findTaskHandler(Task task);

    List<TaskHandler> findAllTaskHandlers();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/handler/TaskHandlerRegistryDefault.java

```
package ua.ivan909020.scheduler.core.service.handler;

import java.util.ArrayList;
import java.util.List;

import org.springframework.context.ApplicationContext;

import ua.ivan909020.scheduler.core.model.entity.Task;

public class TaskHandlerRegistryDefault implements TaskHandlerRegistry {

    private ApplicationContext applicationContext;

    public TaskHandlerRegistryDefault(ApplicationContext applicationContext) {
        this.applicationContext = applicationContext;
    }

    @Override
    public TaskHandler findTaskHandler(Task task) {
        List<TaskHandler> taskHandlers = findAllTaskHandlers().stream()
                .filter(taskHandler -> taskHandler.supports(task))
                .toList();

        if (taskHandlers.isEmpty()) {
            throw new IllegalStateException("Found none task handlers for task: " +
task.getId());
        }
        if (taskHandlers.size() > 1) {
            throw new IllegalStateException("Found too many task handlers for task: " +
task.getId());
        }
```

```
        return taskHandlers.get(0);
    }

    @Override
    public List<TaskHandler> findAllTaskHandlers() {
        return new
ArrayList<>(applicationContext.getBeansOfType(TaskHandler.class).values());
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/worker/WorkerService.java

```
package ua.ivan909020.scheduler.core.service.worker;

public interface WorkerService {

    void start();

    void stop();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/worker/WorkerServiceDefault.java

```
package ua.ivan909020.scheduler.core.service.worker;

import static java.util.concurrent.TimeUnit.SECONDS;
import static
ua.ivan909020.scheduler.core.model.domain.instance.Instance.PARTITIONS;

import java.time.Duration;
import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.function.Consumer;

import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;

import ua.ivan909020.scheduler.core.service.converter.PartitionConverter;
import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;
import ua.ivan909020.scheduler.core.service.worker.policy.PartitionPolicy;
import ua.ivan909020.scheduler.core.service.worker.scanner.PartitionScanner;
import ua.ivan909020.scheduler.core.service.worker.scanner.PartitionScannerFactory;
import
ua.ivan909020.scheduler.core.service.worker.scanner.handler.QueueMessageHandler;

public class WorkerServiceDefault implements WorkerService {

    private static final Duration REPARTITION_INTERVAL = Duration.ofSeconds(10);

    private final Logger logger = LoggerFactory.getLogger(getClass());

    private final PartitionPolicy partitionPolicy;

    private final InstanceRegistry instanceRegistry;

    private final QueueMessageHandler queueMessageHandler;

    private final PartitionScannerFactory partitionScannerFactory;

    private final ScheduledExecutorService partitionExecutor;

    private List<PartitionScanner> partitionScanners;

    public WorkerServiceDefault(
            PartitionPolicy partitionPolicy,
            InstanceRegistry instanceRegistry,
            PartitionScannerFactory partitionScannerFactory,
            QueueMessageHandler queueMessageHandler) {

        this.partitionPolicy = partitionPolicy;
        this.instanceRegistry = instanceRegistry;
        this.queueMessageHandler = queueMessageHandler;
        this.partitionScannerFactory = partitionScannerFactory;
        this.partitionExecutor = Executors.newScheduledThreadPool(1);
        this.partitionScanners = new ArrayList<>();
    }

    @Override
    public void start() {
        logger.info("Starting");
```

```
        queueMessageHandler.start();

        partitionExecutor.scheduleWithFixedDelay(() -> {
            try {
                updatePartitions();
            } catch(Exception e) {
                logger.warn("Failed to update partitions", e);
            }
        }, 0, REPARTITION_INTERVAL.toSeconds(), SECONDS);
    }

    @Override
    public void stop() {
        logger.info("Stopping");

        consumePartitionScanners(PartitionScanner::stop);
        queueMessageHandler.stop();

        partitionExecutor.shutdownNow();
    }

    private void updatePartitions() {
        List<Integer> newPartitions = partitionPolicy.computePartitions();
        List<Integer> currentPartitions = getCurrentPartitions();

        logger.info("Starting repartition, current: {}, new: {}", currentPartitions,
newPartitions);

        if (!newPartitions.equals(currentPartitions)) {
            rectratePartitions(newPartitions);
        }

        instanceRegistry.updateCurrentInstanceMetadata(Map.of(PARTITIONS,
PartitionConverter.toString(newPartitions)));
    }

    private List<Integer> getCurrentPartitions() {
        return partitionScanners.stream()
                .map(PartitionScanner::getPartitions)
                .flatMap(Collection::stream)
                .toList();
    }

    private void rectratePartitions(List<Integer> newPartitions) {
```

```
        logger.info("Recreate partitions");

        consumePartitionScanners(PartitionScanner::stop);
        if (newPartitions.isEmpty()) {
            partitionScanners = List.of();
        } else {
            partitionScanners = List.of(partitionScannerFactory.create(newPartitions));
        }
        consumePartitionScanners(PartitionScanner::start);
    }

    private void consumePartitionScanners(Consumer<PartitionScanner> consumer) {
        partitionScanners.forEach(scanner -> {
            try {
                consumer.accept(scanner);
            } catch (Exception e) {
                logger.warn("Failed to consume scanner, {}", scanner, e);
            }
        });
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/worker/policy/PartitionPolicy.java

```
package ua.ivan909020.scheduler.core.service.worker.policy;

import java.util.List;

public interface PartitionPolicy {

    List<Integer> computePartitions();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/worker/policy/PartitionPolicyDiscovery.java

```
package ua.ivan909020.scheduler.core.service.worker.policy;

import java.util.ArrayList;
import java.util.Comparator;
```

```
import java.util.List;
import java.util.stream.IntStream;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.model.domain.instance.Instance;
import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;

public class PartitionPolicyDiscovery implements PartitionPolicy {

   private final SchedulerProperties schedulerProperties;

   private final InstanceRegistry instanceRegistry;

   public PartitionPolicyDiscovery(SchedulerProperties schedulerProperties,
InstanceRegistry instanceRegistry) {
      this.schedulerProperties = schedulerProperties;
      this.instanceRegistry = instanceRegistry;
   }

   @Override
   public List<Integer> computePartitions() {
      List<String> instanceIds = instanceRegistry.getAllInstances().stream()
            .sorted(Comparator.comparing(Instance::getRegisteredAt))
            .map(instance -> instance.getServiceInstance().getInstanceId())
            .toList();

      int instancesCount = instanceIds.size();
      int currentInstanceIndex = instanceIds
            .indexOf(instanceRegistry.getCurrentInstance().getServiceInstance().getInstan
ceId());

      if (instancesCount == 0 || currentInstanceIndex == -1) {
         return List.of();
      }

      List<Integer> partitions = IntStream.rangeClosed(1,
schedulerProperties.getMaxPartitions()).boxed().toList();
      return partition(partitions, instanceIds.size()).get(currentInstanceIndex);
   }

   private List<List<Integer>> partition(List<Integer> partitions, int partitionSize) {
      List<List<Integer>> result = new ArrayList<>(partitionSize);

      int quotient = partitions.size() / partitionSize;
      int reminder = partitions.size() % partitionSize;
```

```
        int offset = 0;
        for (int i = 0; i < partitionSize; i++) {
            int size = quotient + (i < reminder ? 1 : 0);
            result.add(partitions.subList(offset, offset + size));
            offset += size;
        }
        return result;
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/scanner/PartitionScanner.java

```
package ua.ivan909020.scheduler.core.service.worker.scanner;

import java.util.List;

public interface PartitionScanner {

    List<Integer> getPartitions();

    void start();

    void stop();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/scanner/PartitionScannerDefault.java

```
package ua.ivan909020.scheduler.core.service.worker.scanner;

import static java.util.concurrent.TimeUnit.SECONDS;
import static ua.ivan909020.scheduler.core.model.entity.TaskStatus.SCHEDULED;

import java.time.Instant;
import java.util.List;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
```

```
import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.model.domain.queue.QueueMessage;
import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.core.queue.QueueProducer;
import ua.ivan909020.scheduler.core.repository.TaskRepository;
import ua.ivan909020.scheduler.core.service.converter.JsonConverter;

public class PartitionScannerDefault implements PartitionScanner {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    private final SchedulerProperties schedulerProperties;

    private final TaskRepository taskRepository;

    private final QueueProducer queueProducer;

    private final JsonConverter jsonConverter;

    private final ScheduledExecutorService partitionExecutor =
Executors.newScheduledThreadPool(1);

    private final List<Integer> partitions;

    public PartitionScannerDefault(
            SchedulerProperties schedulerProperties,
            TaskRepository taskRepository,
            QueueProducer queueProducer,
            JsonConverter jsonConverter,
            List<Integer> partitions) {

        this.schedulerProperties = schedulerProperties;
        this.taskRepository = taskRepository;
        this.queueProducer = queueProducer;
        this.jsonConverter = jsonConverter;
        this.partitions = partitions;
    }

    @Override
    public List<Integer> getPartitions() {
        return partitions;
    }
```

```
    @Override
    public void start() {
        logger.info("Starting, partitions: {}", partitions);

        partitionExecutor.scheduleWithFixedDelay(() -> {
            try {
                scan(partitions);
            } catch(Exception e) {
                logger.warn("Failed to scan partitions", e);
            }
        }, 0, schedulerProperties.getTaskFetchInterval().toSeconds(), SECONDS);
    }

    @Override
    public void stop() {
        logger.info("Stopping, partitions: {}", partitions);

        partitionExecutor.shutdownNow();
    }

    private void scan(List<Integer> partitions) {
        List<Task> tasks = taskRepository.findAllOverdue(
                partitions, List.of(SCHEDULED), Instant.now(),
schedulerProperties.getTaskFetchLimit());

        logger.info("Scanning, partitions: {}, found tasks: {}", partitions, tasks.size());

        for (Task task : tasks) {
            try {
                send(task);
            } catch (Exception e) {
                logger.warn("Failed to send a task: {}", task, e);
            }
        }
    }

    private void send(Task task) {
        task.setStatus(TaskStatus.SUBMITTED);
        taskRepository.updateStatus(task);

        QueueMessage message = new QueueMessage();
        message.setKey(String.valueOf(task.getId()));
        message.setValue(jsonConverter.convertToString(task));

        logger.info("Sending a message: {}", message);
```

```
        queueProducer.send(message);
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/scanner/PartitionScannerFactory.java

```
package ua.ivan909020.scheduler.core.service.worker.scanner;

import java.util.List;

public interface PartitionScannerFactory {

    PartitionScanner create(List<Integer> partitions);

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/scanner/PartitionScannerFactoryDefault.java

```
package ua.ivan909020.scheduler.core.service.worker.scanner;

import java.util.List;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.queue.QueueProducer;
import ua.ivan909020.scheduler.core.repository.TaskRepository;
import ua.ivan909020.scheduler.core.service.converter.JsonConverter;

public class PartitionScannerFactoryDefault implements PartitionScannerFactory {

    private final SchedulerProperties schedulerProperties;

    private final TaskRepository taskRepository;

    private final QueueProducer queueProducer;

    private final JsonConverter jsonConverter;

    public PartitionScannerFactoryDefault(
            SchedulerProperties schedulerProperties,
```

```
            TaskRepository taskRepository,
            QueueProducer queueProducer,
            JsonConverter jsonConverter) {

        this.schedulerProperties = schedulerProperties;
        this.taskRepository = taskRepository;
        this.queueProducer = queueProducer;
        this.jsonConverter = jsonConverter;
    }

    @Override
    public PartitionScanner create(List<Integer> partitions) {
        return new PartitionScannerDefault(schedulerProperties, taskRepository,
queueProducer,
                jsonConverter, partitions);
    }

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/scanner/handler/QueueMessageHandler.java

```
package ua.ivan909020.scheduler.core.service.worker.scanner.handler;

public interface QueueMessageHandler {

    void start();

    void stop();

}
```

scheduler-core/src/main/java/ua/ivan909020/scheduler/core/service/scanner/handler/QueueMessageHandlerDefault.java

```
package ua.ivan909020.scheduler.core.service.worker.scanner.handler;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor;

import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
```

```
import ua.ivan909020.scheduler.core.queue.QueueConsumer;
import ua.ivan909020.scheduler.core.repository.TaskRepository;
import ua.ivan909020.scheduler.core.service.converter.JsonConverter;
import ua.ivan909020.scheduler.core.service.handler.TaskHandler;
import ua.ivan909020.scheduler.core.service.handler.TaskHandlerRegistry;

public class QueueMessageHandlerDefault implements QueueMessageHandler {

    private final Logger logger = LoggerFactory.getLogger(getClass());

    private final TaskRepository taskRepository;

    private final QueueConsumer queueConsumer;

    private final JsonConverter jsonConverter;

    private final TaskHandlerRegistry taskHandlerRegistry;

    private final ThreadPoolTaskExecutor taskHandlerExecutor;

    public QueueMessageHandlerDefault(
            TaskRepository taskRepository,
            QueueConsumer queueConsumer,
            JsonConverter jsonConverter,
            TaskHandlerRegistry taskHandlerRegistry,
            ThreadPoolTaskExecutor taskHandlerExecutor) {

        this.taskRepository = taskRepository;
        this.queueConsumer = queueConsumer;
        this.jsonConverter = jsonConverter;
        this.taskHandlerRegistry = taskHandlerRegistry;
        this.taskHandlerExecutor = taskHandlerExecutor;
    }

    @Override
    public void start() {
        logger.info("Starting");

        queueConsumer.subscribe(message -> {
            logger.info("Received a message: {}", message);

            taskHandlerExecutor.execute(() -> {
                try {
                    Task task = jsonConverter.convertToObject(message.getValue(),
Task.class);
```

```
                handle(task);
            } catch (Exception e) {
                logger.warn("Failed to handle a message: {}", message);
            }
        });
    });
    queueConsumer.start();
}

private void handle(Task task) {
    TaskHandler taskHandler = taskHandlerRegistry.findTaskHandler(task);

    Exception exception = null;
    try {
        taskHandler.handle(task);

        logger.info("Task successfully handled: {}", task);
    } catch (Exception e) {
        exception = e;
        logger.warn("Failed to handle a task: {}", task, e);
    }

    if (exception == null) {
        task.setStatus(TaskStatus.SUCCEEDED);
        taskRepository.updateStatus(task);
    } else {
        task.setStatus(TaskStatus.FAILED);
        taskRepository.updateStatus(task);
    }
}

@Override
public void stop() {
    logger.info("Stopping");

    queueConsumer.stop();
    taskHandlerExecutor.shutdown();
}

}
```

scheduler-core/src/main/resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

ua.ivan909020.scheduler.core.configuration.TaskAutoConfiguration
ua.ivan909020.scheduler.core.configuration.SchedulerAutoConfiguration
ua.ivan909020.scheduler.core.configuration.WorkerAutoConfiguration

scheduler-provider-database/postgresql/src/main/java/ua/ivan909020/scheduler/database/postgres/configuration/PostgresDatabaseAutoConfiguration.java

```
package ua.ivan909020.scheduler.database.postgres.configuration;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import ua.ivan909020.scheduler.core.repository.TaskRepository;
import
ua.ivan909020.scheduler.database.postgres.repository.TaskRepositoryExtendedPostgres;

@Configuration
public class PostgresDatabaseAutoConfiguration {

   @Bean
   @ConditionalOnMissingBean
   public TaskRepository taskRepository(NamedParameterJdbcTemplate jdbcTemplate)
{
      return new TaskRepositoryExtendedPostgres(jdbcTemplate);
   }

}
```

scheduler-provider-database/postgresql/src/main/java/ua/ivan909020/scheduler/database/postgres/repository/TaskRepositoryPostgres.java

```
package ua.ivan909020.scheduler.database.postgres.repository;

import java.sql.Timestamp;
import java.sql.Types;
import java.time.Instant;
import java.util.List;
import java.util.UUID;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
```

```
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.core.repository.TaskRepository;

public class TaskRepositoryPostgres implements TaskRepository {

    private NamedParameterJdbcTemplate jdbcTemplate;

    public TaskRepositoryPostgres(NamedParameterJdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    @Override
    public void create(Task task) {
        String sql = """
                INSERT INTO task(
                    partition, id, version, status, execute_at, name, data
                )
                VALUES (
                    :partition, :id, :version, :status, :executeAt, :name, :data
                )
                """;

        MapSqlParameterSource parameters = new MapSqlParameterSource();
        parameters.addValue("partition", task.getPartition());
        parameters.addValue("id", UUID.fromString(task.getId()));
        parameters.addValue("version", task.getVersion());
        parameters.addValue("status", task.getStatus(), Types.OTHER);
        parameters.addValue("executeAt", Timestamp.from(task.getExecuteAt()),
Types.TIMESTAMP);
        parameters.addValue("name", task.getName());
        parameters.addValue("data", task.getData());

        try {
            int insertedCount = jdbcTemplate.update(sql, parameters);
            if (insertedCount != 1) {
                throw new IllegalStateException("Task with id " + task.getId() + " was not
inserted");
            }
        } catch (Exception e) {
            throw new IllegalStateException("Task with id " + task.getId() + " was not
inserted", e);
        }
```

```
    }

    @Override
    public void updateStatus(Task task) {
        String sql = """
                UPDATE task SET
                    version = :version + 1,
                    status = :status
                WHERE
                    partition = :partition AND
                    id = :id AND
                    version = :version
                """;

        MapSqlParameterSource parameters = new MapSqlParameterSource();
        parameters.addValue("partition", task.getPartition());
        parameters.addValue("id", UUID.fromString(task.getId()));
        parameters.addValue("version", task.getVersion());
        parameters.addValue("status", task.getStatus(), Types.OTHER);

        int updatedCount = jdbcTemplate.update(sql, parameters);
        if (updatedCount != 1) {
            throw new IllegalStateException("Task with id " + task.getId() + " was not
updated");
        }

        task.setVersion(task.getVersion() + 1);
    }

    @Override
    public List<Task> findAllOverdue(
            List<Integer> partitions, List<TaskStatus> statuses, Instant timestamp, int limit)
{

        String sql = """
                SELECT
                  partition, id, version, status, execute_at AS executeAt, name, data
                FROM task
                WHERE
                  partition IN (:partitions) AND
                  status IN (:statuses) AND
                  execute_at <= :executeAt
                ORDER BY execute_at ASC
                LIMIT :limit
                """;
```

```
        MapSqlParameterSource parameters = new MapSqlParameterSource();
        parameters.addValue("partitions", partitions);
        parameters.addValue("statuses", statuses.stream().map(TaskStatus::name).toList(),
Types.OTHER);
        parameters.addValue("executeAt", Timestamp.from(timestamp),
Types.TIMESTAMP);
        parameters.addValue("limit", limit);

        return jdbcTemplate.query(sql, parameters, new
BeanPropertyRowMapper<>(Task.class));
    }

}
```

scheduler-provider-database/postgresql/src/main/java/ua/ivan909020/scheduler/database/postgres/repository/TaskRepositoryExtendedPostgres.java

```
package ua.ivan909020.scheduler.database.postgres.repository;

import java.sql.Types;
import java.util.List;

import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.namedparam.MapSqlParameterSource;
import org.springframework.jdbc.core.namedparam.NamedParameterJdbcTemplate;

import ua.ivan909020.scheduler.core.model.domain.page.KeysetPage;
import ua.ivan909020.scheduler.core.model.domain.page.PagedList;
import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.core.repository.TaskRepositoryExtended;

public class TaskRepositoryExtendedPostgres extends TaskRepositoryPostgres
implements TaskRepositoryExtended {

    private NamedParameterJdbcTemplate jdbcTemplate;

    public TaskRepositoryExtendedPostgres(NamedParameterJdbcTemplate
jdbcTemplate) {
        super(jdbcTemplate);
        this.jdbcTemplate = jdbcTemplate;
    }
```

```
    @Override
    public PagedList<Task> findAll(List<Integer> partitions, List<TaskStatus> statuses,
KeysetPage page) {
        String sql = """
                SELECT
                  partition, id, version, status, execute_at AS executeAt, name, data
                FROM task
                WHERE
                  partition IN (:partitions) AND
                  status IN (:statuses)
                ORDER BY execute_at ASC
                LIMIT :limit
                """;

        MapSqlParameterSource parameters = new MapSqlParameterSource();
        parameters.addValue("partitions", partitions);
        parameters.addValue("statuses", statuses.stream().map(TaskStatus::name).toList(),
Types.OTHER);
        parameters.addValue("limit", page.size());

        List<Task> result = jdbcTemplate.query(sql, parameters, new
BeanPropertyRowMapper<>(Task.class));

        String nextCursor = result.size() > 1 ? result.get(result.size() - 1).getId() : null;
        return new PagedList<>(result, nextCursor);
    }

}
```

scheduler-provider-database/postgresql/src/main/resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
ua.ivan909020.scheduler.database.postgres.configuration.PostgresDatabaseAutoConfig
uration
```

scheduler-provider-database/postgresql/src/main/resources/migration/changelog.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<databaseChangeLog
    xmlns="http://www.liquibase.org/xml/ns/dbchangelog"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ext="http://www.liquibase.org/xml/ns/dbchangelog-ext"
    xmlns:pro="http://www.liquibase.org/xml/ns/pro"
    xsi:schemaLocation="http://www.liquibase.org/xml/ns/dbchangelog
        http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-latest.xsd
```

```
        http://www.liquibase.org/xml/ns/dbchangelog-ext
http://www.liquibase.org/xml/ns/dbchangelog/dbchangelog-ext.xsd
        http://www.liquibase.org/xml/ns/pro
http://www.liquibase.org/xml/ns/pro/liquibase-pro-latest.xsd">

    <include
        file="classpath:migration/sql/1.0.0__create_task_table.sql" />

</databaseChangeLog>
```

scheduler-provider-database/postgresql/src/main/resources/migration/sql/1.0.0__create_task_table.sql

```
CREATE TABLE task(
    partition INT NOT NULL,
    id UUID NOT NULL,
    version BIGINT NOT NULL,
    status VARCHAR NOT NULL,
    execute_at TIMESTAMP NOT NULL,
    name VARCHAR NOT NULL,
    data TEXT NOT NULL,

    PRIMARY KEY (id),
    UNIQUE (id, partition)
);

CREATE INDEX ON task(status, execute_at);
```

scheduler-provider-discovery/zookeeper/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>ua.ivan909020.scheduler</groupId>
        <artifactId>distributed-scheduler-parent</artifactId>
        <version>1.0.0</version>
        <relativePath>../../pom.xml</relativePath>
    </parent>
```

```
    <artifactId>distributed-scheduler-discovery-provider-zookeeper</artifactId>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>ua.ivan909020.scheduler</groupId>
            <artifactId>distributed-scheduler-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-starter-zookeeper-discovery</artifactId>
        </dependency>
    </dependencies>

</project>
```

scheduler-provider-discovery/zookeeper/src/main/java/ua/ivan909020/scheduler/discovery/zookeeper/configuration/ZookeeperDiscoveryAutoConfiguration.java

```
package ua.ivan909020.scheduler.discovery.zookeeper.configuration;

import org.apache.curator.x.discovery.ServiceDiscovery;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.cloud.zookeeper.discovery.ZookeeperDiscoveryClient;
import org.springframework.cloud.zookeeper.discovery.ZookeeperInstance;
import
org.springframework.cloud.zookeeper.serviceregistry.ServiceInstanceRegistration;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;
import
ua.ivan909020.scheduler.discovery.zookeeper.service.InstanceRegistryZookeeper;

@Configuration
public class ZookeeperDiscoveryAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public InstanceRegistry instanceRegistry(
            SchedulerProperties schedulerProperties,
```

```
            ServiceInstanceRegistration instanceRegistration,
            ZookeeperDiscoveryClient discoveryClient,
            ServiceDiscovery<ZookeeperInstance> serviceDiscovery) {

        return new InstanceRegistryZookeeper(schedulerProperties, instanceRegistration,
                discoveryClient, serviceDiscovery);
    }

}
```

scheduler-provider-discovery/zookeeper/src/main/java/ua/ivan909020/scheduler/discovery/zookeeper/service/InstanceRegistryZookeeper.java

```
package ua.ivan909020.scheduler.discovery.zookeeper.service;

import static
org.springframework.cloud.zookeeper.support.StatusConstants.INSTANCE_STATUS_
KEY;
import static
ua.ivan909020.scheduler.core.model.domain.instance.Instance.PARTITIONS;
import static
ua.ivan909020.scheduler.core.model.domain.instance.Instance.UPDATED_AT;

import java.time.Instant;
import java.util.List;
import java.util.Map;

import org.apache.curator.x.discovery.ServiceDiscovery;
import org.apache.curator.x.discovery.ServiceInstance;
import org.springframework.cloud.zookeeper.discovery.ZookeeperDiscoveryClient;
import org.springframework.cloud.zookeeper.discovery.ZookeeperInstance;
import org.springframework.cloud.zookeeper.discovery.ZookeeperServiceInstance;
import
org.springframework.cloud.zookeeper.serviceregistry.ServiceInstanceRegistration;
import org.springframework.util.ReflectionUtils;

import ua.ivan909020.scheduler.core.configuration.properties.SchedulerProperties;
import ua.ivan909020.scheduler.core.model.domain.instance.Instance;
import ua.ivan909020.scheduler.core.model.domain.instance.InstanceStatus;
import ua.ivan909020.scheduler.core.service.converter.PartitionConverter;
import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;

public class InstanceRegistryZookeeper implements InstanceRegistry {
```

```
    private final SchedulerProperties schedulerProperties;

    private final ServiceInstanceRegistration instanceRegistration;

    private final ZookeeperDiscoveryClient discoveryClient;

    private final ServiceDiscovery<ZookeeperInstance> serviceDiscovery;

    public InstanceRegistryZookeeper(
            SchedulerProperties schedulerProperties,
            ServiceInstanceRegistration instanceRegistration,
            ZookeeperDiscoveryClient discoveryClient,
            ServiceDiscovery<ZookeeperInstance> serviceDiscovery) {

        this.schedulerProperties = schedulerProperties;
        this.instanceRegistration = instanceRegistration;
        this.discoveryClient = discoveryClient;
        this.serviceDiscovery = serviceDiscovery;
    }

    @Override
    public Instance getCurrentInstance() {
        setPortIfNeeded();
        return buildInstance(new ZookeeperServiceInstance(
                schedulerProperties.getGroupId(), instanceRegistration.getServiceInstance()));
    }

    private void setPortIfNeeded() {
        if (instanceRegistration.getServiceInstance().getPort() == null) {
            instanceRegistration.setPort(0);
        }
    }

    @Override
    public void updateCurrentInstanceMetadata(Map<String, String> metadata) {
        ServiceInstance<ZookeeperInstance> serviceInstance =
instanceRegistration.getServiceInstance();

        Map<String, String> currentMetadata =
serviceInstance.getPayload().getMetadata();
        currentMetadata.put(Instance.UPDATED_AT, Instant.now().toString());
        currentMetadata.putAll(metadata);

        try {
            serviceDiscovery.updateService(serviceInstance);
```

```
        } catch (Exception e) {
            ReflectionUtils.rethrowRuntimeException(e);
        }
    }

    @Override
    public List<Instance> getAllInstances() {
        return discoveryClient.getInstances(schedulerProperties.getGroupId()).stream()
                .map(ZookeeperServiceInstance.class::cast)
                .map(this::buildInstance)
                .toList();
    }

    private Instance buildInstance(ZookeeperServiceInstance serviceInstance) {
        Instance instance = new Instance();
        instance.setServiceInstance(serviceInstance);

instance.setRegisteredAt(Instant.ofEpochMilli(serviceInstance.getServiceInstance().get
RegistrationTimeUTC()));

        if (serviceInstance.getMetadata().containsKey(UPDATED_AT)) {

instance.setUpdatedAt(Instant.parse(serviceInstance.getMetadata().get(UPDATED_AT))
);
        }
        if (serviceInstance.getMetadata().containsKey(INSTANCE_STATUS_KEY)) {

instance.setStatus(InstanceStatus.valueOf(serviceInstance.getMetadata().get(INSTANC
E_STATUS_KEY)));
        }
        if (serviceInstance.getMetadata().containsKey(PARTITIONS)) {

instance.setPartitions(PartitionConverter.toList(serviceInstance.getMetadata().get(PAR
TITIONS)));
        }
        return instance;
    }

}
```

scheduler-provider-discovery/zookeeper/src/main/resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
ua.ivan909020.scheduler.discovery.zookeeper.configuration.ZookeeperDiscoveryAutoC
onfiguration
```

scheduler-provider-queue/kafka/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>ua.ivan909020.scheduler</groupId>
        <artifactId>distributed-scheduler-parent</artifactId>
        <version>1.0.0</version>
        <relativePath>../../pom.xml</relativePath>
    </parent>

    <artifactId>distributed-scheduler-queue-provider-kafka</artifactId>
    <packaging>jar</packaging>

    <dependencies>
        <dependency>
            <groupId>ua.ivan909020.scheduler</groupId>
            <artifactId>distributed-scheduler-core</artifactId>
            <version>${project.version}</version>
        </dependency>
        <dependency>
          <groupId>org.springframework.kafka</groupId>
          <artifactId>spring-kafka</artifactId>
        </dependency>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
        </dependency>

        <dependency>
            <groupId>org.testcontainers</groupId>
            <artifactId>kafka</artifactId>
            <version>${testcontainers.version}</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.awaitility</groupId>
            <artifactId>awaitility</artifactId>
```

```
        <scope>test</scope>
      </dependency>
    </dependencies>

    <build>
      <plugins>
        <plugin>
          <groupId>org.codehaus.mojo</groupId>
          <artifactId>build-helper-maven-plugin</artifactId>
          <executions>
            <execution>
              <id>add-integration-test-source</id>
              <phase>generate-test-sources</phase>
              <goals>
                <goal>add-test-source</goal>
              </goals>
              <configuration>
                <sources>
                  <source>src/test-integration/java</source>
                </sources>
              </configuration>
            </execution>
            <execution>
              <id>add-integration-test-resource</id>
              <phase>generate-test-resources</phase>
              <goals>
                <goal>add-test-resource</goal>
              </goals>
              <configuration>
                <resources>
                  <resource>
                    <directory>src/test-integration/resources</directory>
                  </resource>
                </resources>
              </configuration>
            </execution>
          </executions>
        </plugin>
      </plugins>
    </build>

</project>
```

scheduler-provider-queue/kafka/src/main/java/ua/ivan909020/scheduler/queue/kafka/configuration/KafkaQueueAutoConfiguration.java

```
package ua.ivan909020.scheduler.queue.kafka.configuration;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.core.ProducerFactory;

import ua.ivan909020.scheduler.queue.kafka.configuration.properties.KafkaQueueProperties;
import ua.ivan909020.scheduler.queue.kafka.service.KafkaQueueConsumer;
import ua.ivan909020.scheduler.queue.kafka.service.KafkaQueueProducer;

@Configuration
@EnableConfigurationProperties
public class KafkaQueueAutoConfiguration {

    @Bean
    @ConditionalOnMissingBean
    public KafkaQueueProperties kafkaQueueProperties() {
        return new KafkaQueueProperties();
    }

    @Bean
    @ConditionalOnMissingBean
    public KafkaQueueProducer kafkaQueueProducer(
            KafkaQueueProperties kafkaQueueProperties,
            ProducerFactory<String, String> kafkaProducerFactory) {

        return new KafkaQueueProducer(kafkaQueueProperties, kafkaProducerFactory);
    }

    @Bean
    @ConditionalOnMissingBean
    public KafkaQueueConsumer kafkaQueueConsumer(
            KafkaQueueProperties kafkaQueueProperties,
            ConsumerFactory<String, String> kafkaConsumerFactory) {

        return new KafkaQueueConsumer(kafkaQueueProperties, kafkaConsumerFactory);
    }
```

```
}
```

scheduler-provider-queue/kafka/src/main/java/ua/ivan909020/scheduler/queue/kafka/configuration/properties/KafkaQueueProperties.java

```
package ua.ivan909020.scheduler.queue.kafka.configuration.properties;

import org.springframework.boot.context.properties.ConfigurationProperties;

@ConfigurationProperties("scheduler")
public class KafkaQueueProperties {

    private String queueTopic;

    public String getQueueTopic() {
        return queueTopic;
    }

    public void setQueueTopic(String queueTopic) {
        this.queueTopic = queueTopic;
    }

}
```

scheduler-provider-queue/kafka/src/main/java/ua/ivan909020/scheduler/queue/kafka/service/KafkaQueueConsumer.java

```
package ua.ivan909020.scheduler.queue.kafka.service;

import java.util.HashSet;
import java.util.Set;
import java.util.function.Consumer;

import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.kafka.core.ConsumerFactory;
import org.springframework.kafka.listener.ContainerProperties;
import org.springframework.kafka.listener.KafkaMessageListenerContainer;
import org.springframework.kafka.listener.MessageListener;

import ua.ivan909020.scheduler.core.model.domain.queue.QueueMessage;
```

```
import ua.ivan909020.scheduler.core.queue.QueueConsumer;
import
ua.ivan909020.scheduler.queue.kafka.configuration.properties.KafkaQueueProperties;

public class KafkaQueueConsumer implements QueueConsumer {

   private final Logger logger = LoggerFactory.getLogger(getClass());

   private final KafkaQueueProperties kafkaQueueProperties;

   private final ConsumerFactory<String, String> kafkaConsumerFactory;

   private KafkaMessageListenerContainer<String, String>
kafkaMessageListenerContainer;

   private Set<Consumer<QueueMessage>> subsribers;

   public KafkaQueueConsumer(
         KafkaQueueProperties kafkaQueueProperties,
         ConsumerFactory<String, String> kafkaConsumerFactory) {

      this.kafkaQueueProperties = kafkaQueueProperties;
      this.kafkaConsumerFactory = kafkaConsumerFactory;
      this.subsribers = new HashSet<>();
   }

   @Override
   public void start() {
      if (kafkaMessageListenerContainer != null) {
         return;
      }
      kafkaMessageListenerContainer =
createMessageListenerContainer(kafkaQueueProperties, kafkaConsumerFactory);
      kafkaMessageListenerContainer.start();
   }

   @Override
   public void stop() {
      if (kafkaMessageListenerContainer == null) {
         return;
      }
      kafkaMessageListenerContainer.stop();
   }

   @Override
```

```
    public void subscribe(Consumer<QueueMessage> handler) {
        subsribers.add(handler);
    }

    private KafkaMessageListenerContainer<String, String>
createMessageListenerContainer(
            KafkaQueueProperties kafkaQueueProperties, ConsumerFactory<String, String>
kafkaConsumerFactory) {

        ContainerProperties containerProperties = new
ContainerProperties(kafkaQueueProperties.getQueueTopic());
        containerProperties.setMessageListener(new MessageListener<String, String>() {

            @Override
            public void onMessage(ConsumerRecord<String, String> data) {
                for (Consumer<QueueMessage> subscriber : subsribers) {
                    try {
                        QueueMessage message = new QueueMessage();
                        message.setKey(data.key());
                        message.setValue(data.value());

                        subscriber.accept(message);
                    } catch (Exception e) {
                        logger.warn("Failed to consume a message: {}", data, e);
                    }
                }
            }

        });
        return new KafkaMessageListenerContainer<>(kafkaConsumerFactory,
containerProperties);
    }

}
```

scheduler-provider-queue/kafka/src/main/java/ua/ivan909020/scheduler/queue/kafka/service/KafkaQueueProducer.java

```
package ua.ivan909020.scheduler.queue.kafka.service;

import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.kafka.core.ProducerFactory;

import ua.ivan909020.scheduler.core.model.domain.queue.QueueMessage;
```

```
import ua.ivan909020.scheduler.core.queue.QueueProducer;
import
ua.ivan909020.scheduler.queue.kafka.configuration.properties.KafkaQueueProperties;

public class KafkaQueueProducer implements QueueProducer {

   private final KafkaQueueProperties kafkaQueueProperties;

   private final KafkaTemplate<String, String> kafkaTemplate;

   public KafkaQueueProducer(
         KafkaQueueProperties kafkaQueueProperties,
         ProducerFactory<String, String> kafkaProducerFactory) {

      this.kafkaQueueProperties = kafkaQueueProperties;
      this.kafkaTemplate = new KafkaTemplate<>(kafkaProducerFactory);
   }

   @Override
   public void send(QueueMessage message) {
      kafkaTemplate.send(kafkaQueueProperties.getQueueTopic(), message.getKey(),
message.getValue());
   }

}
```

scheduler-provider-queue/kafka/src/main/resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
ua.ivan909020.scheduler.queue.kafka.configuration.KafkaQueueAutoConfiguration
```

scheduler-rest-api/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

   <modelVersion>4.0.0</modelVersion>

   <parent>
      <groupId>ua.ivan909020.scheduler</groupId>
      <artifactId>distributed-scheduler-parent</artifactId>
      <version>1.0.0</version>
```

```
    <relativePath>../pom.xml</relativePath>
</parent>

<artifactId>distributed-scheduler-rest-api</artifactId>
<packaging>jar</packaging>

<properties>
    <openapi.version>2.1.0</openapi.version>
</properties>

<dependencies>
    <dependency>
        <groupId>ua.ivan909020.scheduler</groupId>
        <artifactId>distributed-scheduler-core</artifactId>
        <version>${project.version}</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>${openapi.version}</version>
    </dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>build-helper-maven-plugin</artifactId>
            <executions>
                <execution>
                    <id>add-integration-test-source</id>
                    <phase>generate-test-sources</phase>
                    <goals>
                        <goal>add-test-source</goal>
                    </goals>
                    <configuration>
                        <sources>
                            <source>src/test-integration/java</source>
                        </sources>
                    </configuration>
                </execution>
```

```
                <execution>
                    <id>add-integration-test-resource</id>
                    <phase>generate-test-resources</phase>
                    <goals>
                        <goal>add-test-resource</goal>
                    </goals>
                    <configuration>
                        <resources>
                            <resource>
                                <directory>src/test-integration/resources</directory>
                            </resource>
                        </resources>
                    </configuration>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>

</project>
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/configuration/InstanceAutoConfuguration.java

```
package ua.ivan909020.scheduler.rest.configuration;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;
import
ua.ivan909020.scheduler.rest.controller.endpoint.instance.InstanceRestController;
import ua.ivan909020.scheduler.rest.service.instance.InstanceService;
import ua.ivan909020.scheduler.rest.service.instance.InstanceServiceDefault;

@Configuration
public class InstanceAutoConfuguration {

    @Bean
    @ConditionalOnMissingBean
    public InstanceService instanceService(InstanceRegistry instanceRegistry) {
        return new InstanceServiceDefault(instanceRegistry);
    }
```

```
    @Bean
    @ConditionalOnMissingBean
    public InstanceRestController instanceRestController(InstanceService
instanceService) {
        return new InstanceRestController(instanceService);
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/configuration/TaskAutoConfuguration.java

```
package ua.ivan909020.scheduler.rest.configuration;

import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import ua.ivan909020.scheduler.core.repository.TaskRepositoryExtended;
import ua.ivan909020.scheduler.core.service.core.PartitionService;
import ua.ivan909020.scheduler.rest.controller.endpoint.task.TaskRestController;
import ua.ivan909020.scheduler.rest.service.task.TaskService;
import ua.ivan909020.scheduler.rest.service.task.TaskServiceDefault;

@Configuration
public class TaskAutoConfuguration {

    @Bean
    @ConditionalOnMissingBean
    public TaskService taskService(TaskRepositoryExtended taskRepository,
PartitionService partitionService) {
        return new TaskServiceDefault(taskRepository, partitionService);
    }

    @Bean
    @ConditionalOnMissingBean
    public TaskRestController taskRestController(TaskService taskService) {
        return new TaskRestController(taskService);
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/controller/endpoint/instance/InstanceRestController.java

```
package ua.ivan909020.scheduler.rest.controller.endpoint.instance;

import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import ua.ivan909020.scheduler.rest.model.dto.instance.InstanceDto;
import ua.ivan909020.scheduler.rest.service.instance.InstanceService;

@RestController
@RequestMapping("/api")
public class InstanceRestController {

    private final InstanceService instanceService;

    public InstanceRestController(InstanceService instanceService) {
        this.instanceService = instanceService;
    }

    @GetMapping("/v1/instances")
    public List<InstanceDto> findAll() {
        return instanceService.findAll();
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/controller/endpoint/task/TaskRestController.java

```
package ua.ivan909020.scheduler.rest.controller.endpoint.task;

import java.util.List;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```
import ua.ivan909020.scheduler.core.model.domain.page.KeysetPage;
import ua.ivan909020.scheduler.core.model.domain.page.PagedList;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.rest.exception.EntityValidationException;
import ua.ivan909020.scheduler.rest.model.dto.task.TaskDto;
import ua.ivan909020.scheduler.rest.service.task.TaskService;

@RestController
@RequestMapping("/api")
public class TaskRestController {

   private final TaskService taskService;

   public TaskRestController(TaskService taskService) {
      this.taskService = taskService;
   }

   @GetMapping("/v1/tasks")
   public PagedList<TaskDto> findAll(
         @RequestParam List<TaskStatus> statuses,
         @RequestParam Integer pageSize,
         @RequestParam(required = false) String pageCursor) {

      if (pageSize < 1 || pageSize > 1000) {
         throw new EntityValidationException("Parameter 'pageSize' must be >= 1 and <
1000");
      }

      return taskService.findAll(statuses, new KeysetPage(pageSize, pageCursor));
   }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/controller/handler/ControllerExceptionHandler.java

```
package ua.ivan909020.scheduler.rest.controller.handler;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
```

```
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHan
dler;

import ua.ivan909020.scheduler.rest.exception.EntityValidationException;
import ua.ivan909020.scheduler.rest.model.dto.ErrorCode;
import ua.ivan909020.scheduler.rest.model.dto.ErrorResponseDto;

@RestControllerAdvice
public class ControllerExceptionHandler extends ResponseEntityExceptionHandler {

   @ExceptionHandler(EntityValidationException.class)
   public ResponseEntity<ErrorResponseDto>
handleEntityValidationException(EntityValidationException ex) {
      return ResponseEntity.status(HttpStatus.BAD_REQUEST)
            .body(new
ErrorResponseDto(ErrorCode.REQUEST_PARAMETERS_NOT_VALID,
ex.getMessage()));
   }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/excrption/EntityValidationException.java

```
package ua.ivan909020.scheduler.rest.exception;

public class EntityValidationException extends RuntimeException {

   private static final long serialVersionUID = 1L;

   public EntityValidationException(String message, Throwable cause) {
      super(message, cause);
   }

   public EntityValidationException(String message) {
      super(message);
   }

   public EntityValidationException(Throwable cause) {
      super(cause);
   }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/model/dto/ErrorCode.java

```
package ua.ivan909020.scheduler.rest.model.dto;

public enum ErrorCode {

    REQUEST_BODY_NOT_VALID,
    REQUEST_PARAMETERS_NOT_VALID

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/model/dto/ErrorResponseDto.java

```
package ua.ivan909020.scheduler.rest.model.dto;

public class ErrorResponseDto {

    private ErrorCode errorCode;
    private String message;

    public ErrorResponseDto() {
    }

    public ErrorResponseDto(ErrorCode errorCode, String message) {
        this.errorCode = errorCode;
        this.message = message;
    }

    public ErrorCode getErrorCode() {
        return errorCode;
    }

    public void setErrorCode(ErrorCode errorCode) {
        this.errorCode = errorCode;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
```

```
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/model/dto/instance/InstanceDto.java

```
package ua.ivan909020.scheduler.rest.model.dto.instance;

import java.net.URI;
import java.time.Instant;
import java.util.List;
import java.util.Objects;

import com.fasterxml.jackson.annotation.JsonInclude;
import com.fasterxml.jackson.annotation.JsonInclude.Include;

import ua.ivan909020.scheduler.core.model.domain.instance.InstanceStatus;

@JsonInclude(Include.NON_NULL)
public class InstanceDto {

    private String id;

    private URI uri;

    private Instant registeredAt;

    private Instant updatedAt;

    private InstanceStatus status;

    private List<Integer> partitions;

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public URI getUri() {
        return uri;
    }
```

```
public void setUri(URI uri) {
    this.uri = uri;
}

public Instant getUpdatedAt() {
    return updatedAt;
}

public void setUpdatedAt(Instant updatedAt) {
    this.updatedAt = updatedAt;
}

public Instant getRegisteredAt() {
    return registeredAt;
}

public void setRegisteredAt(Instant registeredAt) {
    this.registeredAt = registeredAt;
}

public InstanceStatus getStatus() {
    return status;
}

public void setStatus(InstanceStatus status) {
    this.status = status;
}

public List<Integer> getPartitions() {
    return partitions;
}

public void setPartitions(List<Integer> partitions) {
    this.partitions = partitions;
}

@Override
public int hashCode() {
    return Objects.hash(id, partitions, registeredAt, status, updatedAt, uri);
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
```

```
            return true;
        }
        if (obj == null || getClass() != obj.getClass()) {
            return false;
        }
        InstanceDto other = (InstanceDto) obj;
        return Objects.equals(id, other.id) &&
                Objects.equals(partitions, other.partitions) &&
                Objects.equals(registeredAt, other.registeredAt) &&
                status == other.status &&
                Objects.equals(updatedAt, other.updatedAt) &&
                Objects.equals(uri, other.uri);
    }

    @Override
    public String toString() {
        return "InstanceDto [" +
                "id=" + id +
                ", uri=" + uri +
                ", registeredAt=" + registeredAt +
                ", updatedAt=" + updatedAt +
                ", status=" + status +
                ", partitions=" + partitions + "]";
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/model/dto/task/TaskDto.java

```
package ua.ivan909020.scheduler.rest.model.dto.task;

import java.time.Instant;
import java.util.Objects;

import ua.ivan909020.scheduler.core.model.entity.TaskStatus;

public class TaskDto {

    private String id;

    private TaskStatus status;

    private Instant executeAt;
```

```
private String name;

public String getId() {
    return id;
}

public void setId(String id) {
    this.id = id;
}

public TaskStatus getStatus() {
    return status;
}

public void setStatus(TaskStatus status) {
    this.status = status;
}

public Instant getExecuteAt() {
    return executeAt;
}

public void setExecuteAt(Instant executeAt) {
    this.executeAt = executeAt;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

@Override
public int hashCode() {
    return Objects.hash(executeAt, id, name, status);
}

@Override
public boolean equals(Object obj) {
    if (this == obj) {
        return true;
    }
    if (obj == null || getClass() != obj.getClass()) {
```

```
            return false;
        }
        TaskDto other = (TaskDto) obj;
        return Objects.equals(executeAt, other.executeAt) &&
                Objects.equals(id, other.id) &&
                Objects.equals(name, other.name) &&
                status == other.status;
    }

    @Override
    public String toString() {
        return "TaskDto [" +
                "id=" + id +
                ", status=" + status +
                ", executeAt=" + executeAt +
                ", name=" + name + "]";
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/service/instance/InstanceService.java

```
package ua.ivan909020.scheduler.rest.service.instance;

import java.util.List;

import ua.ivan909020.scheduler.rest.model.dto.instance.InstanceDto;

public interface InstanceService {

    List<InstanceDto> findAll();

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/service/instance/InstanceServiceDefault.java

```
package ua.ivan909020.scheduler.rest.service.instance;

import java.util.Comparator;
import java.util.List;

import org.springframework.cloud.client.ServiceInstance;
```

```
import org.springframework.stereotype.Service;

import ua.ivan909020.scheduler.core.model.domain.instance.Instance;
import ua.ivan909020.scheduler.core.service.discovery.InstanceRegistry;
import ua.ivan909020.scheduler.rest.model.dto.instance.InstanceDto;

@Service
public class InstanceServiceDefault implements InstanceService {

    private final InstanceRegistry instanceRegistry;

    public InstanceServiceDefault(InstanceRegistry instanceRegistry) {
        this.instanceRegistry = instanceRegistry;
    }

    @Override
    public List<InstanceDto> findAll() {
        return instanceRegistry.getAllInstances().stream()
                .map(this::buildInstanceDto)
                .sorted(Comparator.comparing(InstanceDto::getRegisteredAt))
                .toList();
    }

    private InstanceDto buildInstanceDto(Instance instance) {
        ServiceInstance serviceInstance = instance.getServiceInstance();

        InstanceDto instanceDto = new InstanceDto();
        instanceDto.setId(serviceInstance.getInstanceId());
        instanceDto.setUri(serviceInstance.getUri());
        instanceDto.setRegisteredAt(instance.getRegisteredAt());
        instanceDto.setUpdatedAt(instance.getUpdatedAt());
        instanceDto.setStatus(instance.getStatus());
        instanceDto.setPartitions(instance.getPartitions());
        return instanceDto;
    }

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/service/task/TaskService.java

```
package ua.ivan909020.scheduler.rest.service.task;

import java.util.List;
```

```
import ua.ivan909020.scheduler.core.model.domain.page.KeysetPage;
import ua.ivan909020.scheduler.core.model.domain.page.PagedList;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.rest.model.dto.task.TaskDto;

public interface TaskService {

   PagedList<TaskDto> findAll(List<TaskStatus> statuses, KeysetPage page);

}
```

scheduler-rest-api/src/main/java/ua/ivan909020/scheduler/rest/service/task/TaskServiceDefault.java

```
package ua.ivan909020.scheduler.rest.service.task;

import java.util.List;

import org.springframework.stereotype.Service;

import ua.ivan909020.scheduler.core.model.domain.page.KeysetPage;
import ua.ivan909020.scheduler.core.model.domain.page.PagedList;
import ua.ivan909020.scheduler.core.model.entity.Task;
import ua.ivan909020.scheduler.core.model.entity.TaskStatus;
import ua.ivan909020.scheduler.core.repository.TaskRepositoryExtended;
import ua.ivan909020.scheduler.core.service.core.PartitionService;
import ua.ivan909020.scheduler.rest.model.dto.task.TaskDto;

@Service
public class TaskServiceDefault implements TaskService {

   private final TaskRepositoryExtended taskRepository;

   private final PartitionService partitionService;

   public TaskServiceDefault(TaskRepositoryExtended taskRepository, PartitionService
partitionService) {
      this.taskRepository = taskRepository;
      this.partitionService = partitionService;
   }

   @Override
   public PagedList<TaskDto> findAll(List<TaskStatus> statuses, KeysetPage page) {
      PagedList<Task> tasks = taskRepository.findAll(partitionService.getAll(), statuses,
page);
```

```
        List<TaskDto> result = tasks.content().stream().map(this::convert).toList();
        return new PagedList<>(result, tasks.nextCursor());
    }

    private TaskDto convert(Task task) {
        TaskDto taskDto = new TaskDto();
        taskDto.setId(task.getId());
        taskDto.setStatus(task.getStatus());
        taskDto.setExecuteAt(task.getExecuteAt());
        taskDto.setName(task.getName());
        return taskDto;
    }

}
```

scheduler-rest-api/src/main/resources/META-INF/spring/org.springframework.boot.autoconfigure.AutoConfiguration.imports

```
ua.ivan909020.scheduler.rest.configuration.InstanceAutoConfuguration
ua.ivan909020.scheduler.rest.configuration.TaskAutoConfuguration
```

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">

    <modelVersion>4.0.0</modelVersion>
    <groupId>ua.ivan909020.scheduler</groupId>
    <artifactId>distributed-scheduler-parent</artifactId>
    <version>1.0.0</version>
    <packaging>pom</packaging>

    <modules>
        <module>scheduler-core</module>
        <module>scheduler-rest-api</module>
        <module>scheduler-provider-discovery/zookeeper</module>
        <module>scheduler-provider-queue/kafka</module>
        <module>scheduler-provider-database/mongodb</module>
        <module>scheduler-provider-database/postgresql</module>
    </modules>

    <properties>
```

```
        <java.version>17</java.version>
        <maven.compiler.source>${java.version}</maven.compiler.source>
        <maven.compiler.target>${java.version}</maven.compiler.target>

        <spring.boot.version>3.0.5</spring.boot.version>
        <spring.cloud.version>2022.0.1</spring.cloud.version>
        <testcontainers.version>1.18.0</testcontainers.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-inline</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <dependencyManagement>
        <dependencies>
            <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-dependencies</artifactId>
                <version>${spring.boot.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
            <dependency>
                <groupId>org.springframework.cloud</groupId>
                <artifactId>spring-cloud-dependencies</artifactId>
                <version>${spring.cloud.version}</version>
                <type>pom</type>
                <scope>import</scope>
            </dependency>
        </dependencies>
    </dependencyManagement>

</project>
```