|  |  |  |
| --- | --- | --- |
|  | Изображение выглядит как зарисовка, рисунок, символ, корона  Автоматически созданное описание |  |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |

Институт Информационных технологий

Кафедра Инструментального и прикладного программного обеспечения

**ПРАКТИЧЕСКАЯ РАБОТА №8**

по дисциплине «Технологии виртуализации клиент-серверных приложений»

**Тема практической работы: «Взаимодействие с minikube»**

**Студент группы** ИКБО-20-21 Сидоров С.Д.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись студента)

**Руководитель практической работы** старший преподаватель Волков М.Ю.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись руководителя)

Работа представлена «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Допущен к работе «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Москва 2024

**СОДЕРЖАНИЕ**

[ЦЕЛЬ РАБОТЫ 3](#_Toc184384787)

[ВОПРОСЫ 40](#_Toc184384788)

[ВЫВОДЫ 42](#_Toc184384789)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 43](#_Toc184384790)

ЦЕЛЬ РАБОТЫ

Развернуть в кластере Kubernetes с помощью minikube 3 сервиса. Один из которых взаимодействует с redis, два оставшихся должны взаимодействовать друг с другом через kafka. Для логирования применить graylog, для мониторинга использовать jaeger, grafana, prometheus. В качестве базы данных должна использоваться PostgreSQL, развернутая отдельно для каждого сервиса.

ХОД РАБОТЫ

Конфигурация подключения Redis для AuthService представлена на рисунке 1.



Рисунок 1 – Конфигурация подключения Redis Для AuthService

Конфигурация работы с OpenTracing представлена на рисунке 2.



Рисунок 2 – Конфигурация работы с OpenTracing

AuthService представлен на рисунках 3-5.

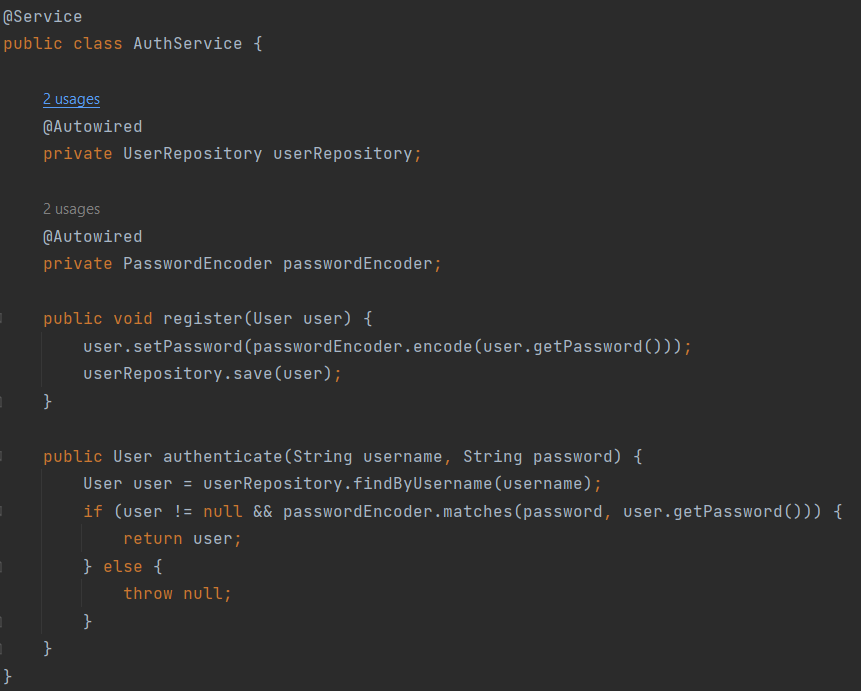


Рисунок 3 – AuthService

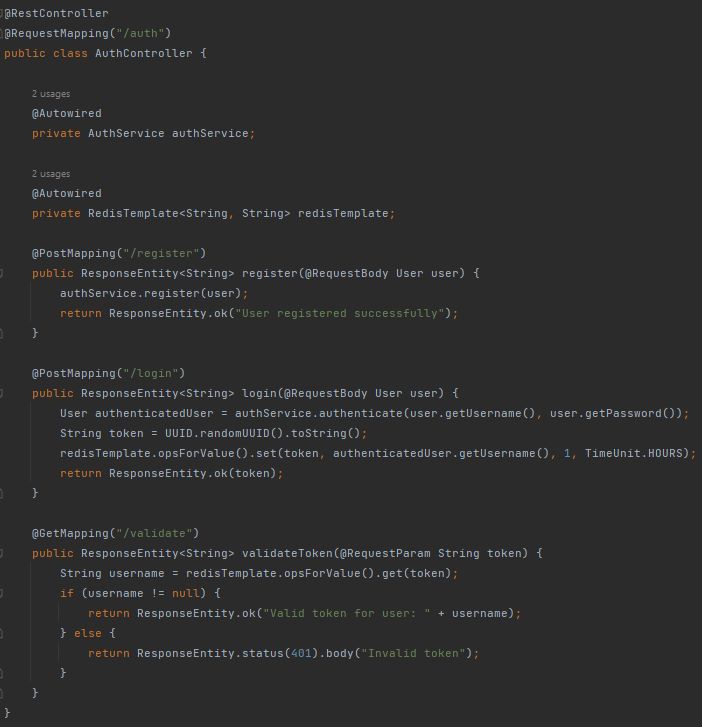


Рисунок 4 – Контроллеры AuthService

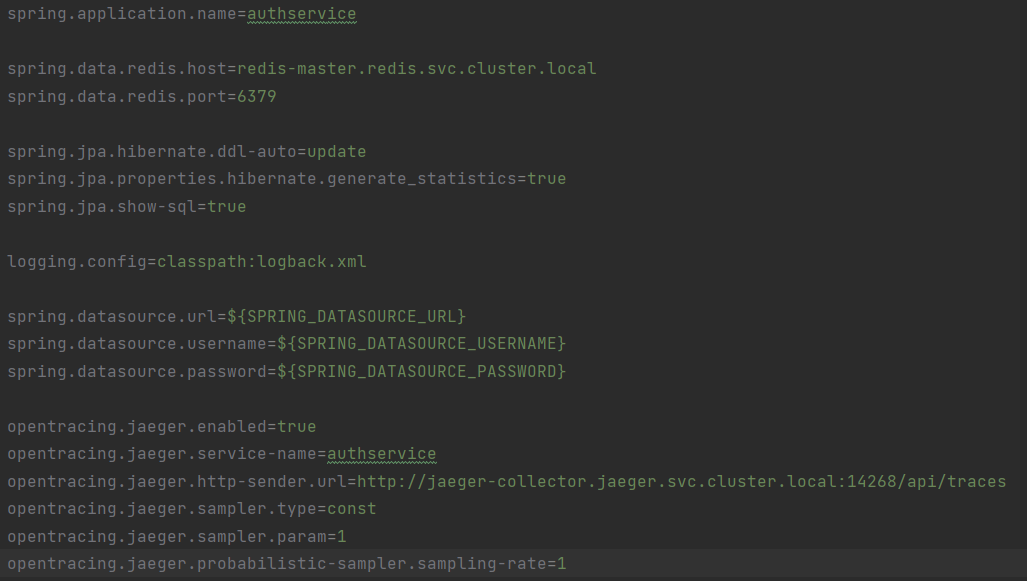


Рисунок 5 – properties AuthService

NotificationService представлен на рисунках 6-9.

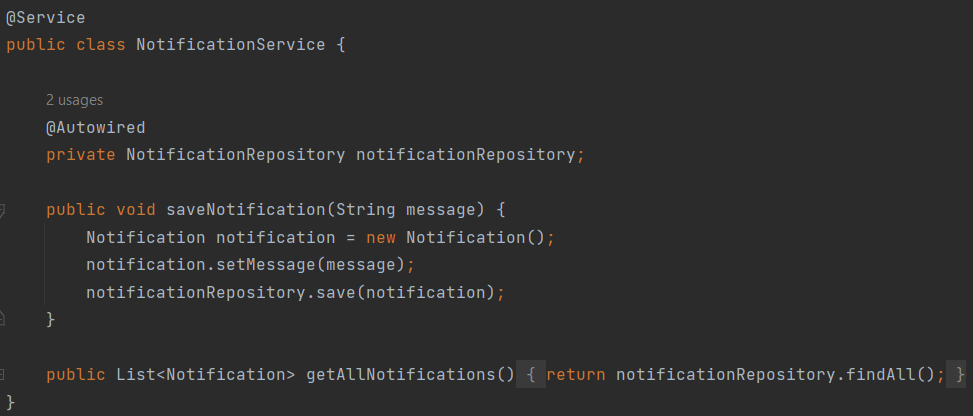


Рисунок 6 – NotificationService



Рисунок 7 – Контроллеры NotificationService

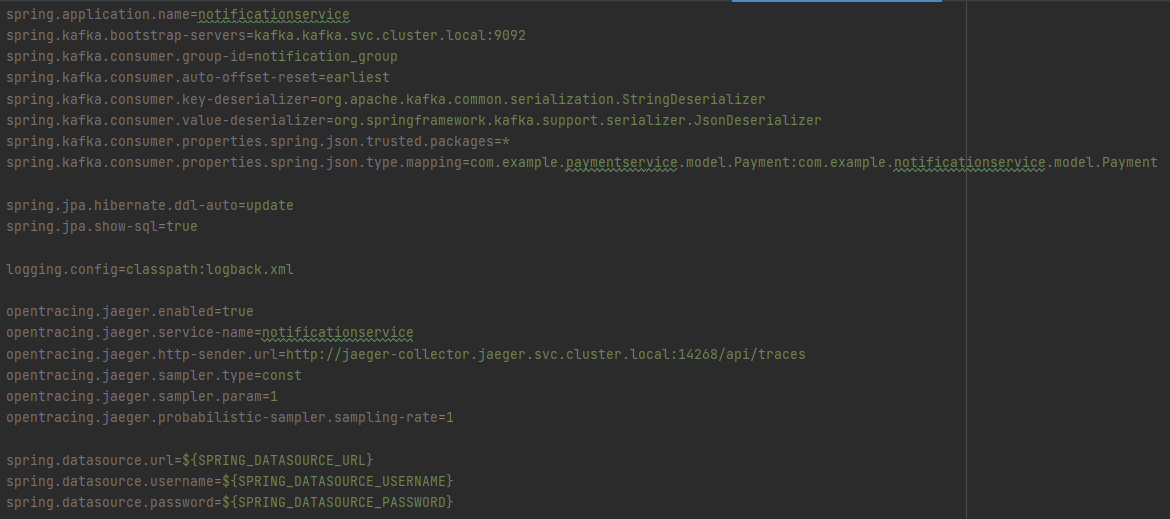


Рисунок 8 – properties NotificationService



Рисунок 9 – KafkaConsumer NotificationService

PaymentService представлен на рисунках 10-12.



Рисунок 10 – PaymentService

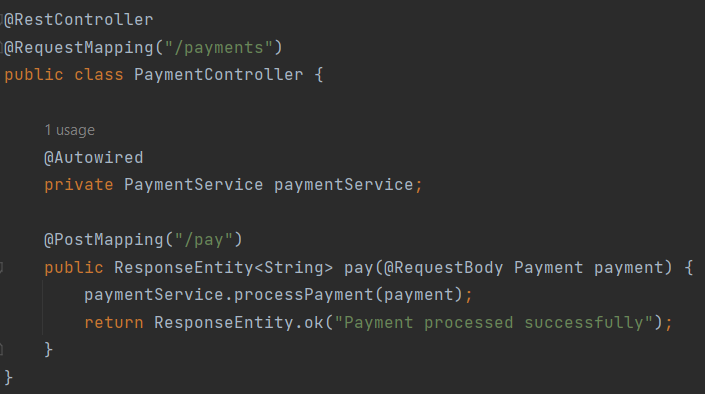


Рисунок 11 – Контроллеры PaymentService

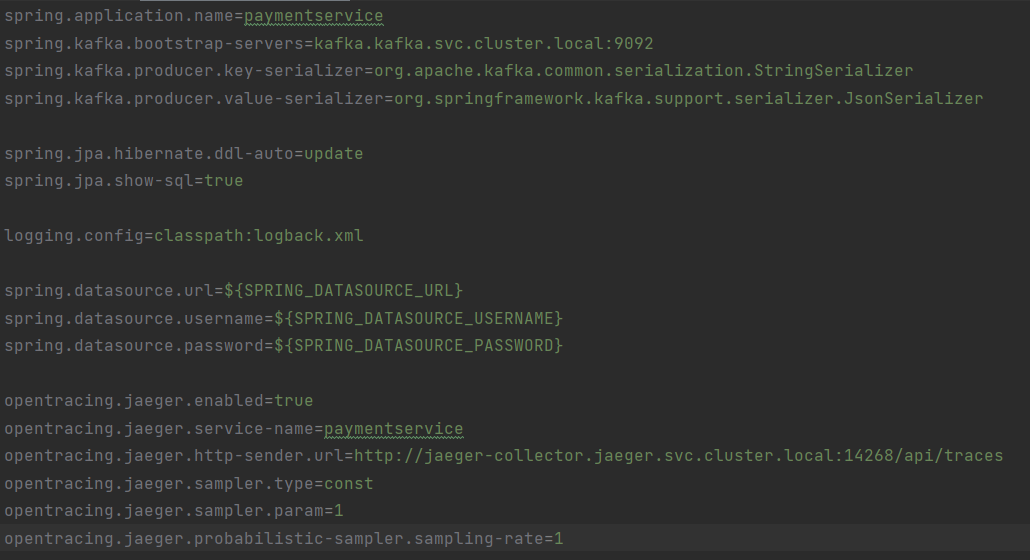


Рисунок 12 – properties PaymentService

Logback представлен на рисунке 13.



Рисунок 13 – logback

Grafna представлен на рисунках 14-15.

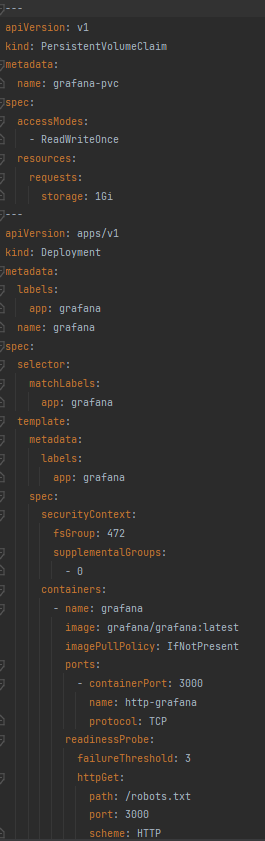


Рисунок 14 – Grafana



Рисунок 15 – Grafana

Поды grafna представлены на рисунке 17.

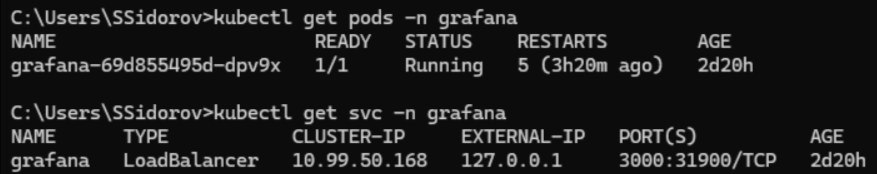


Рисунок 16 – Grafana pods

Jaeger представлен на рисунках 17-20.

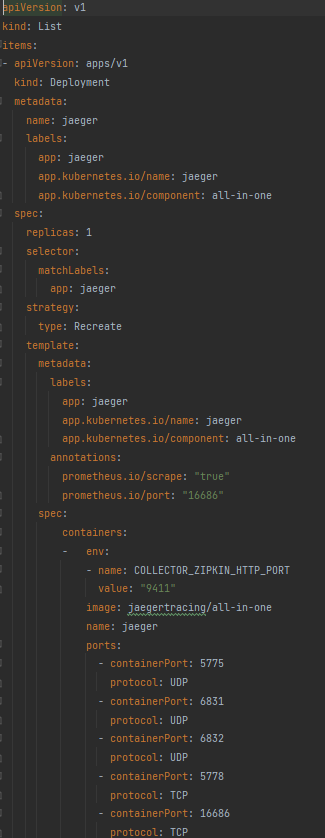


Рисунок 17 – Jaeger

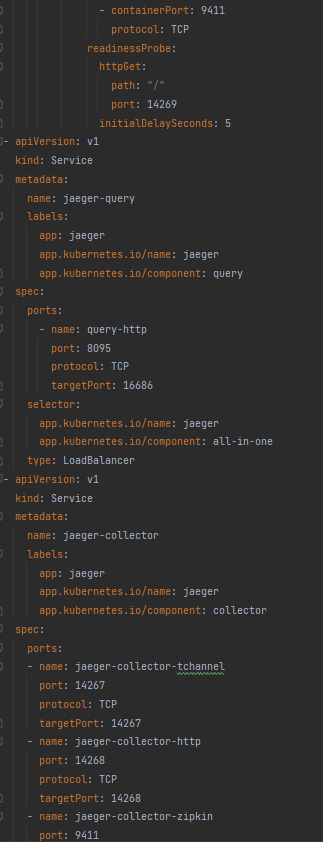


Рисунок 18 – Jaeger

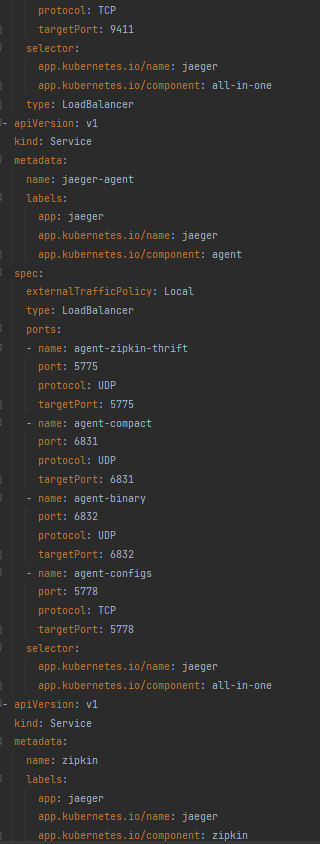


Рисунок 19 – Jaeger

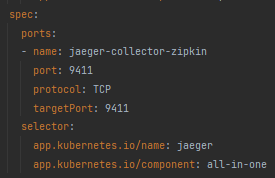


Рисунок 20 – Jaeger

Поды jaeger представлены на рисунке 21.

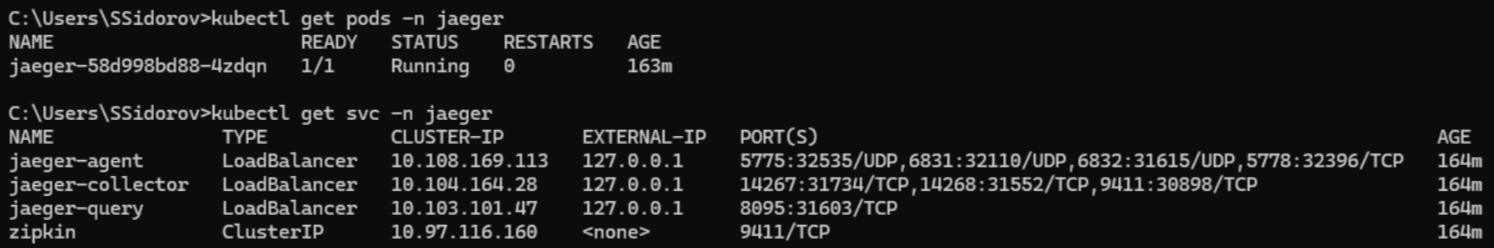


Рисунок 21 – Jaeger pods

Krakend представлен на рисунках 22-25.

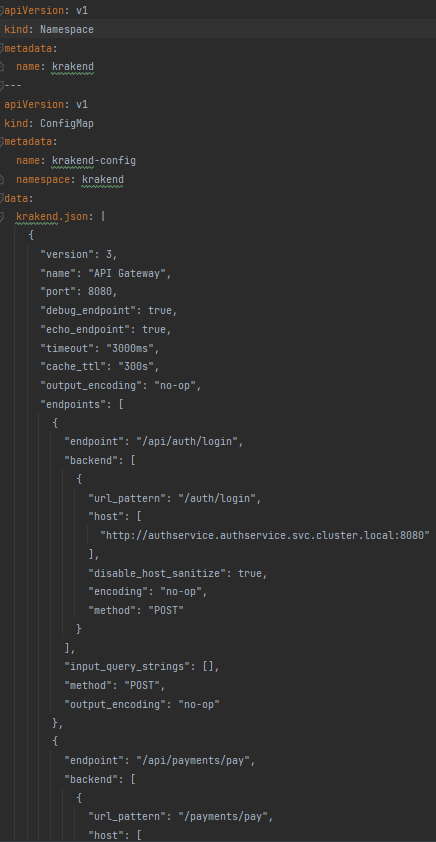


Рисунок 22 – krakend



Рисунок 23 – krakend

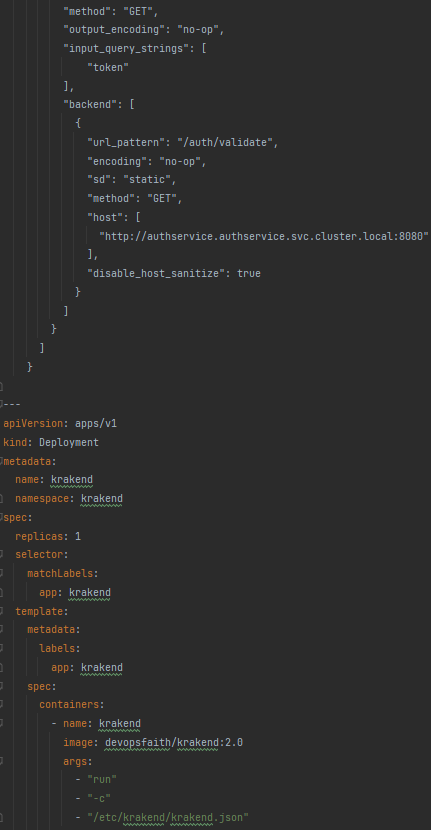


Рисунок 24 – Krakend

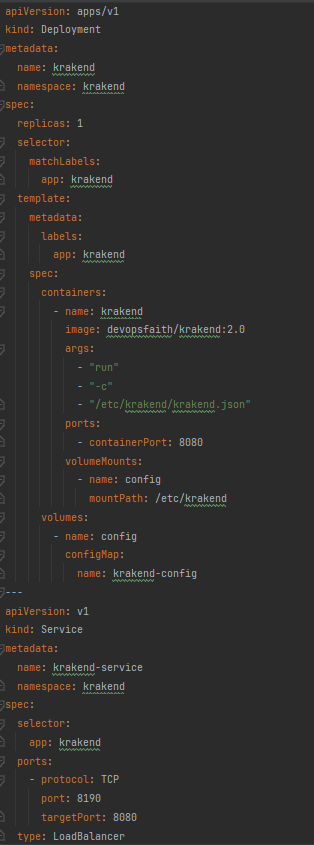


Рисунок 25 – krakend

Поды krakend представлены на рисунке 26.

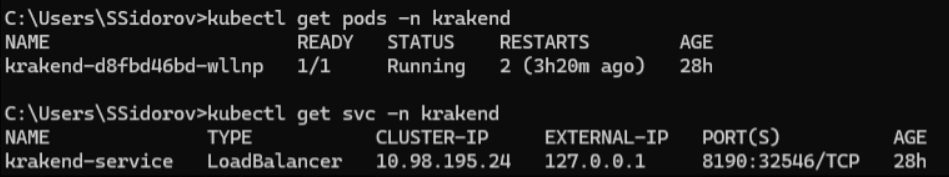


Рисунок 26 – Krakend pods

Kafka представлен на рисунках 27-29.

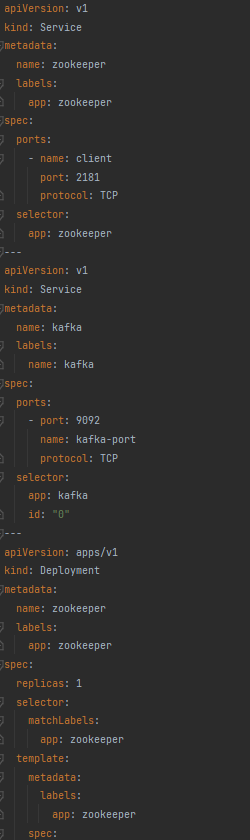


Рисунок 27 – kafka

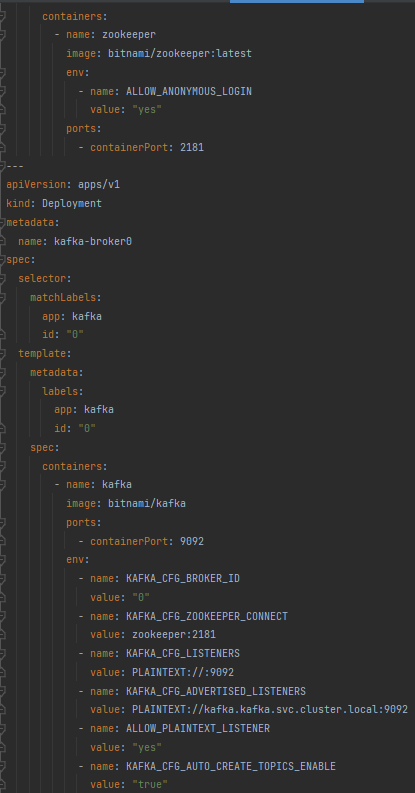


Рисунок 28 – kafka

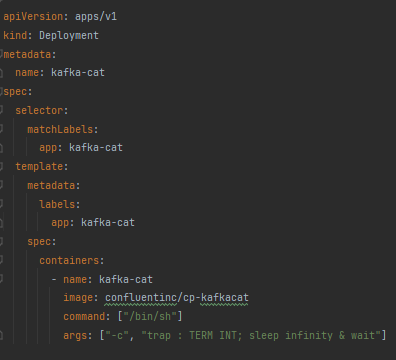


Рисунок 29 – kafka

Поды kafka представлены на рисунике 30.



Рисунок 30 – kafka pods

Поды AuthService представлены на рисунке 31.

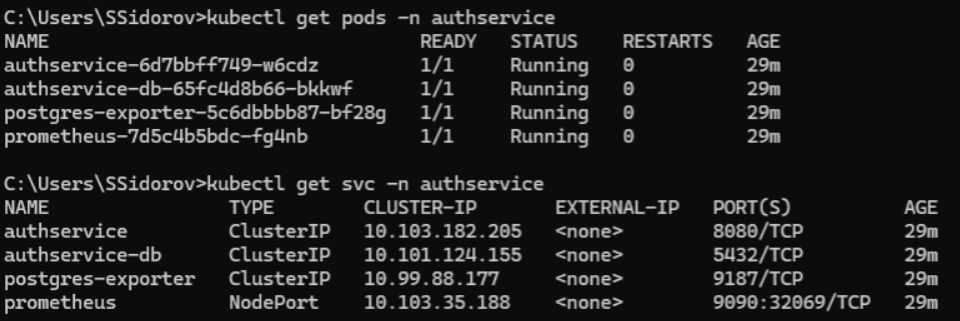


Рисунок 31 – AuthService pods

Поды PaymentService представлены на рисунке 32.

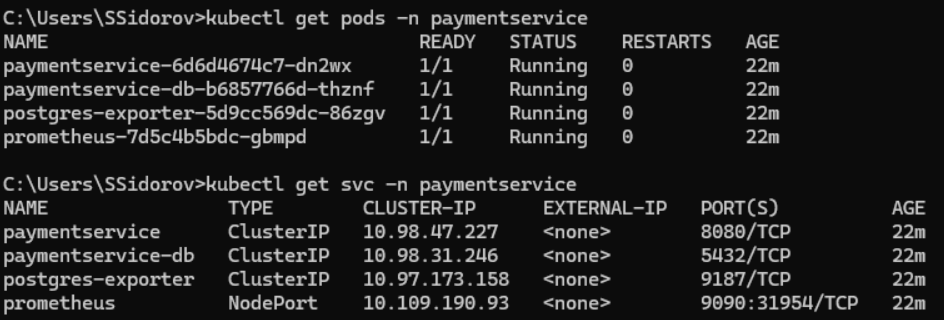


Рисунок 32 – PaymentService pods

Поды NotificationService представлены на рисунке 33.



Рисунок 33 – NotificationService pods

Поды redis представлены на рисунке 35.

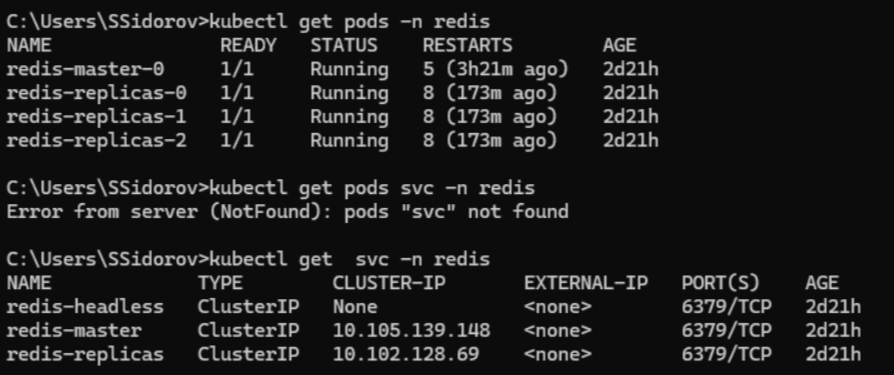


Рисунок 34 – redis pods

Содержимое service представлено на рисунке 35.

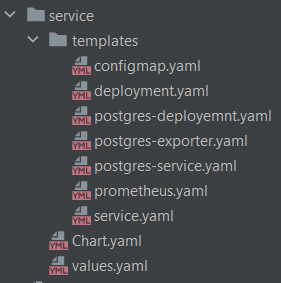


Рисунок 35 – Содержимое service

Grafna AuthService представлен на рисунке 36.

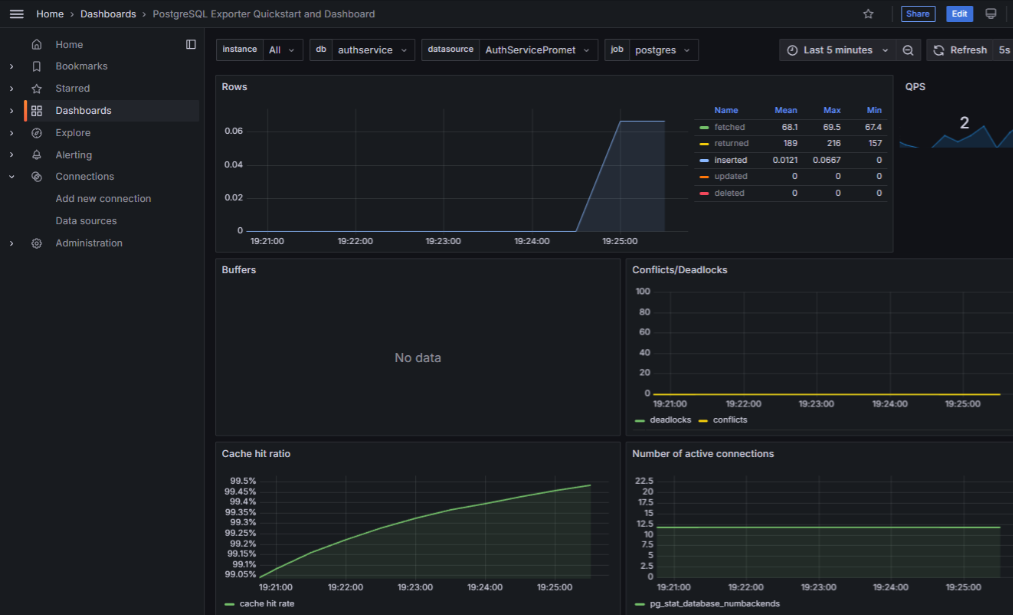


Рисунок 36 – Grafana AuthService

Установка PaymentService представлена на рисунке 37.

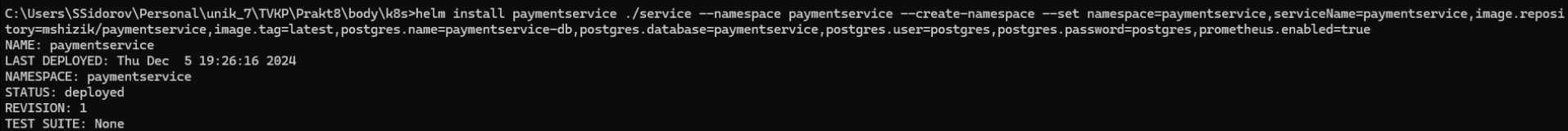


Рисунок 37 – Установка PaymentService

Grafna PaymentService представлен на рисунке 38.

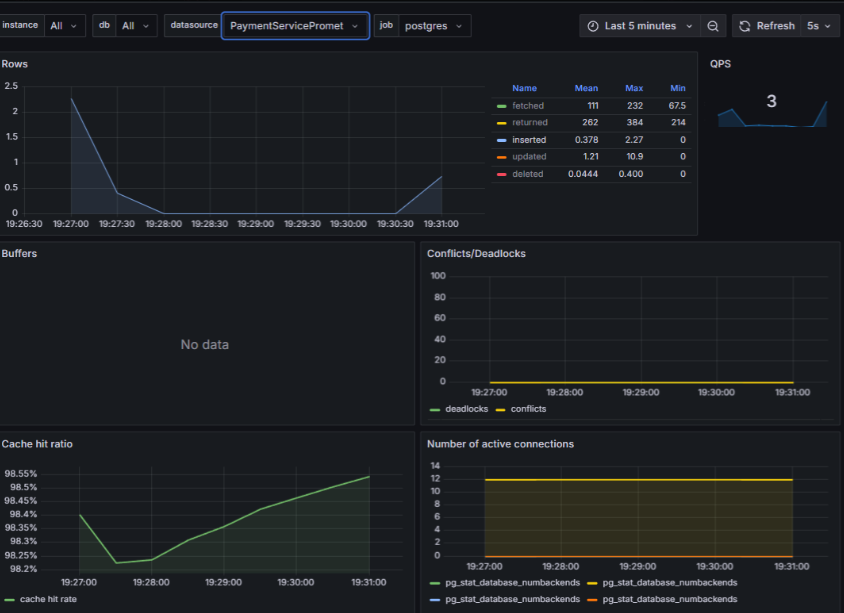


Рисунок 38 – Grafana PaymentService

Установка NotificationService представлена на рисунке 39.

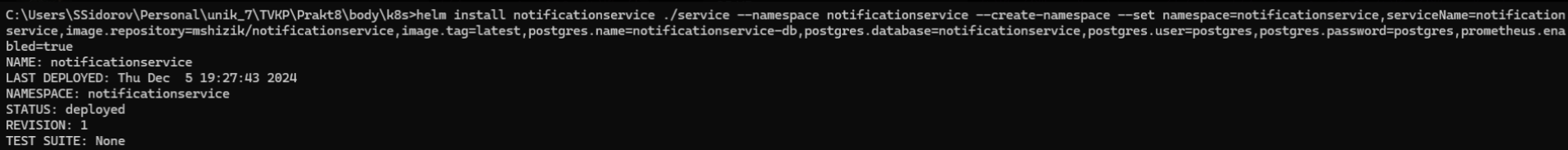


Рисунок 39 – Установка NotificationService

Grafna NotificationService представлен на рисунке 40.

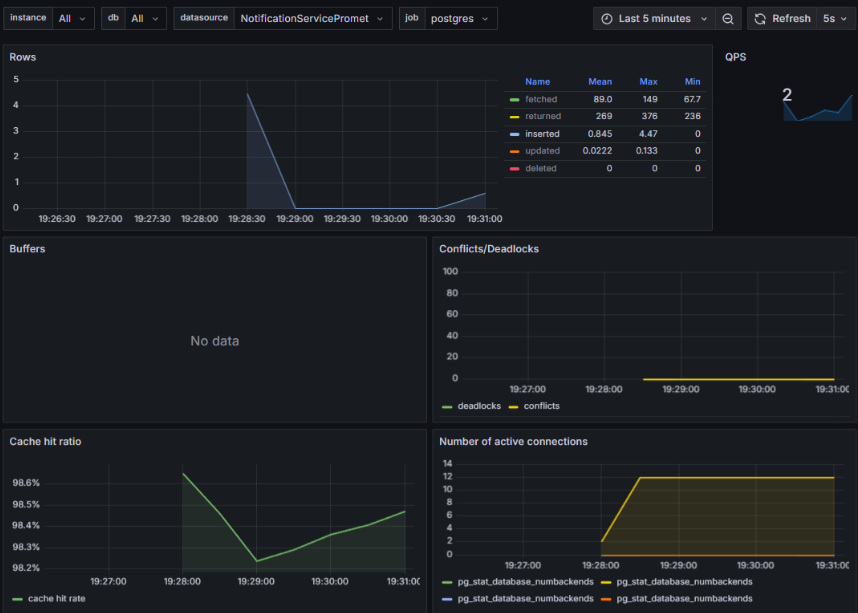


Рисунок 40 – Grafana NotificationService

Отправка запроса PaymentService представлена на рисунке 41.

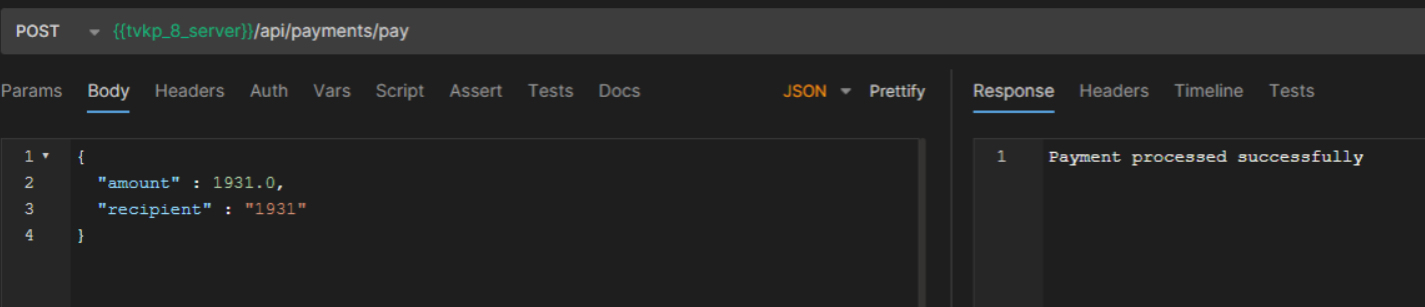


Рисунок 41 – Отправка запроса PaymentService

GrayLog PaymentService представлен на рисунке 42.

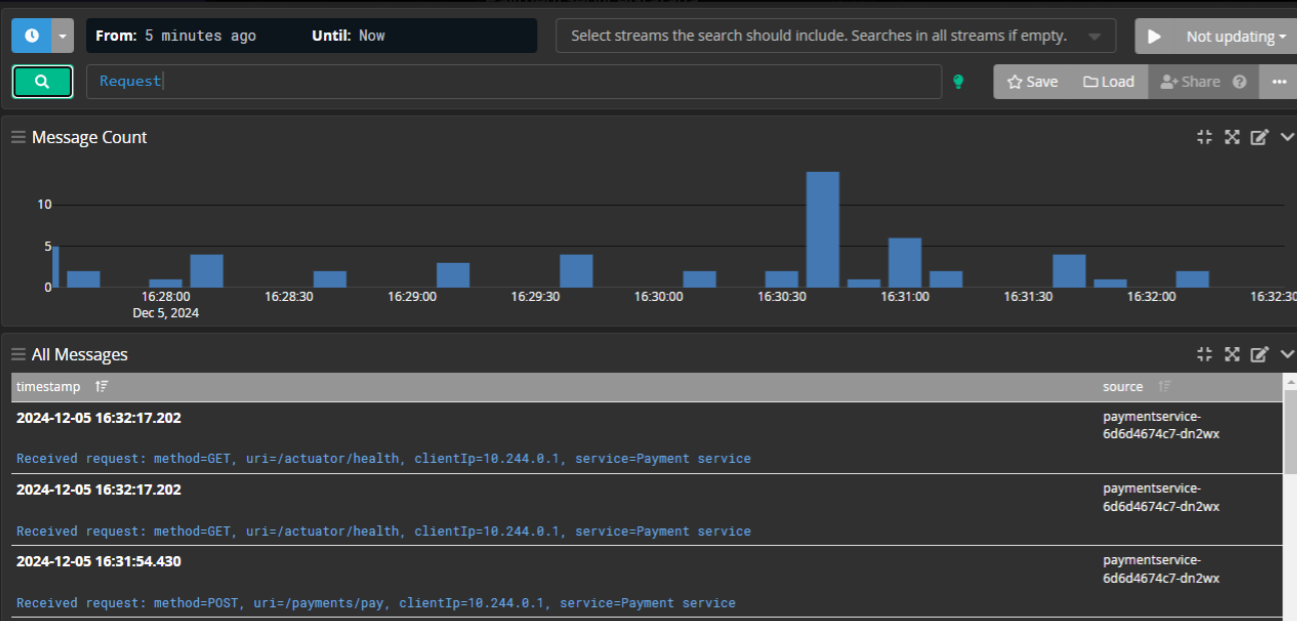


Рисунок 42 – GrayLog PaymentService

Очередь jaeger PaymentService представлена на рисунке 44.

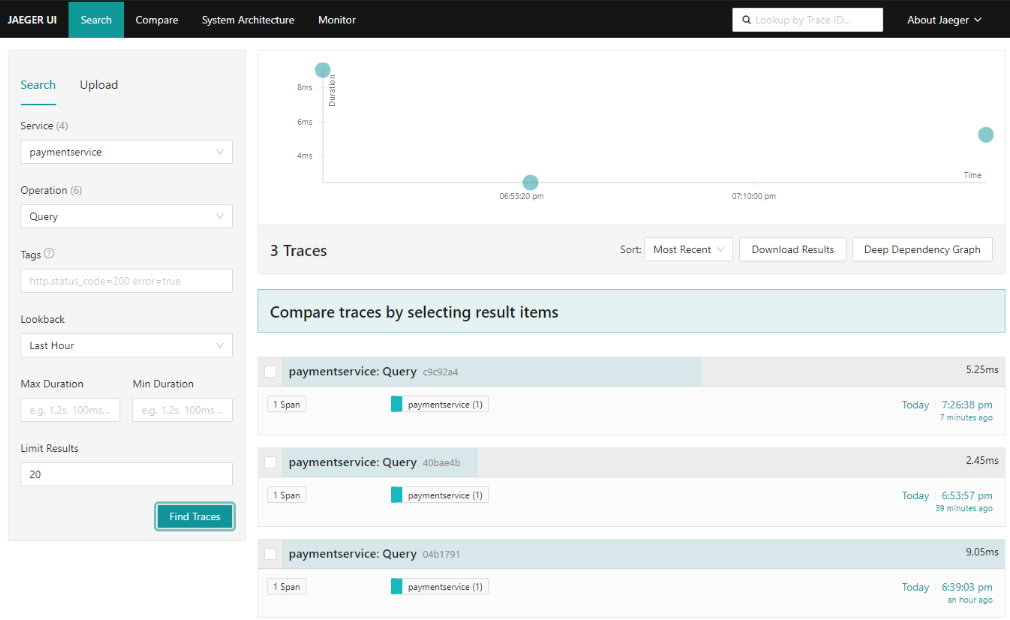


Рисунок 43 – Очередь jaeger PaymentService

Jaeger PaymentService представлен на рисунке 44.

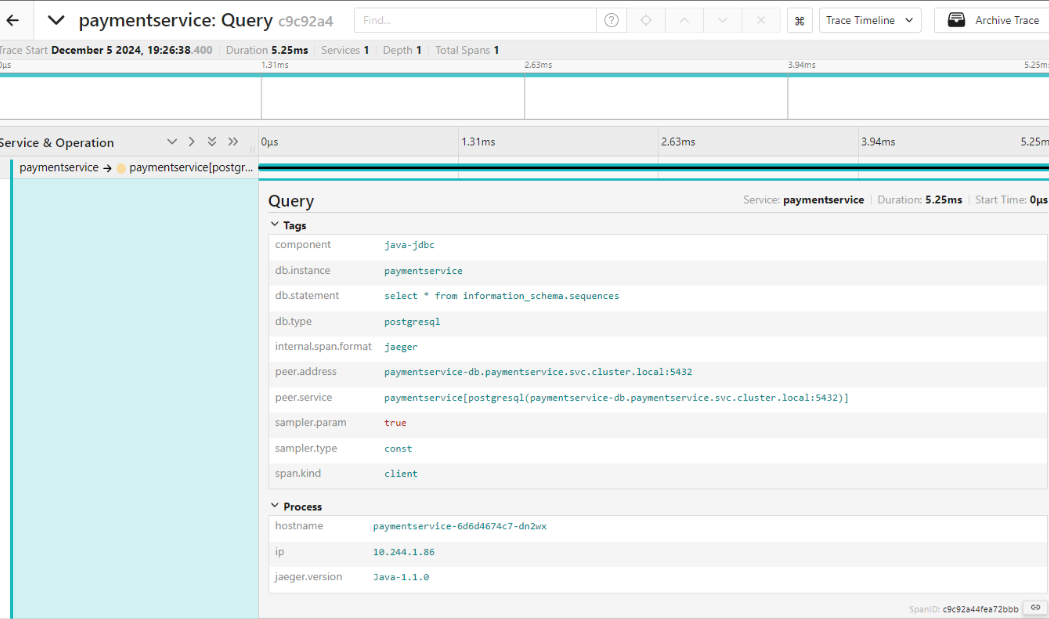


Рисунок 44 – Jaeger PaymentService

Запрос NotificationService представлен на рисунке 45.

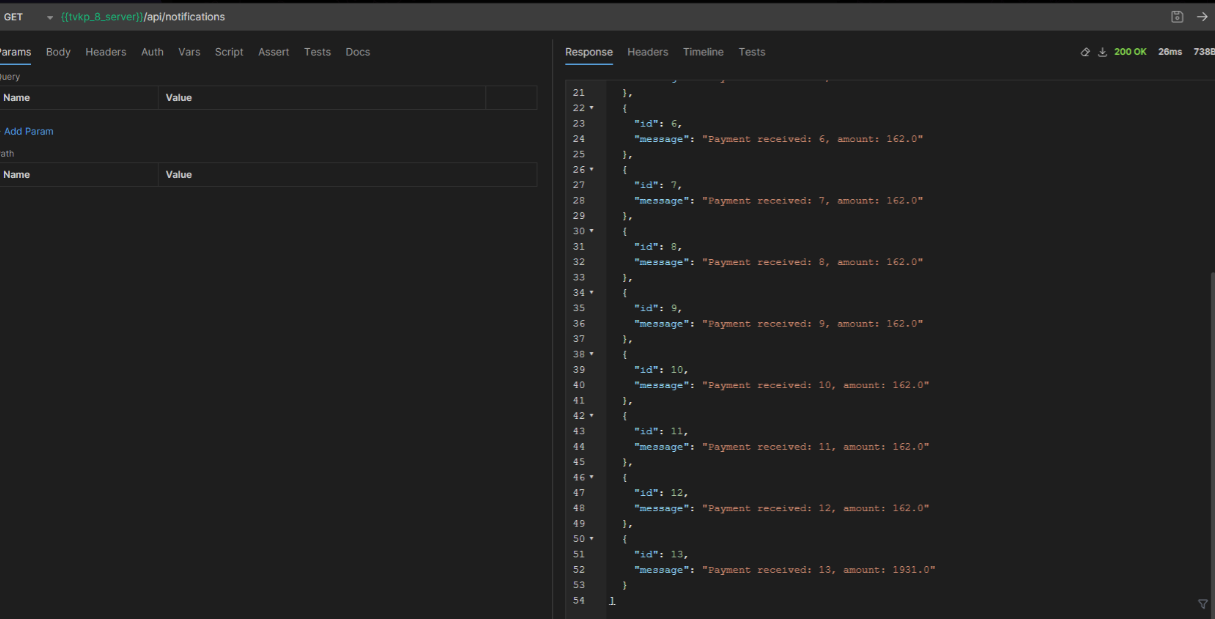


Рисунок 45 – Запрос NotificationService

Graylog NotificationService представлен на рисунке 46.

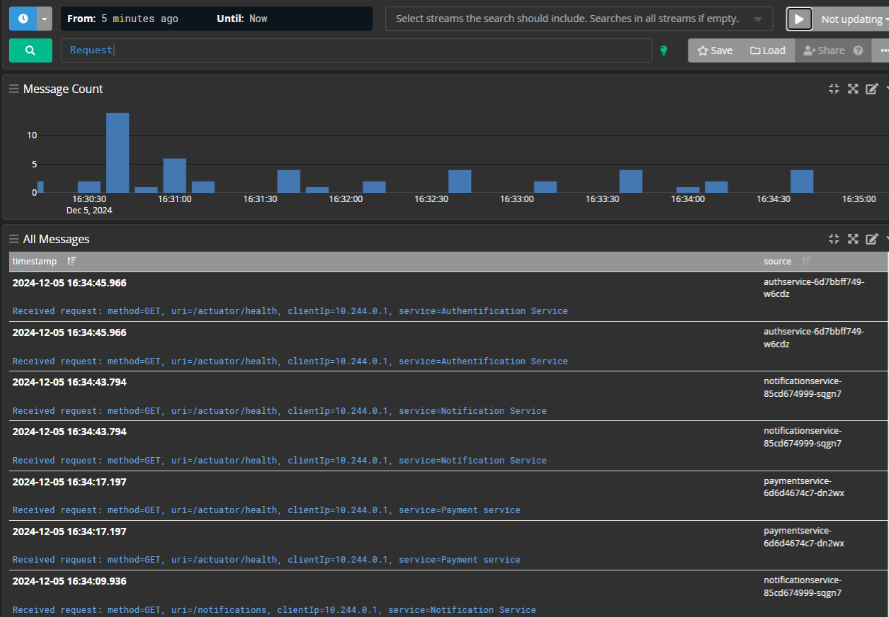


Рисунок 46 – Graylog NotificationService

Очередь jaeger NotificationService представлена на рисунке 47.

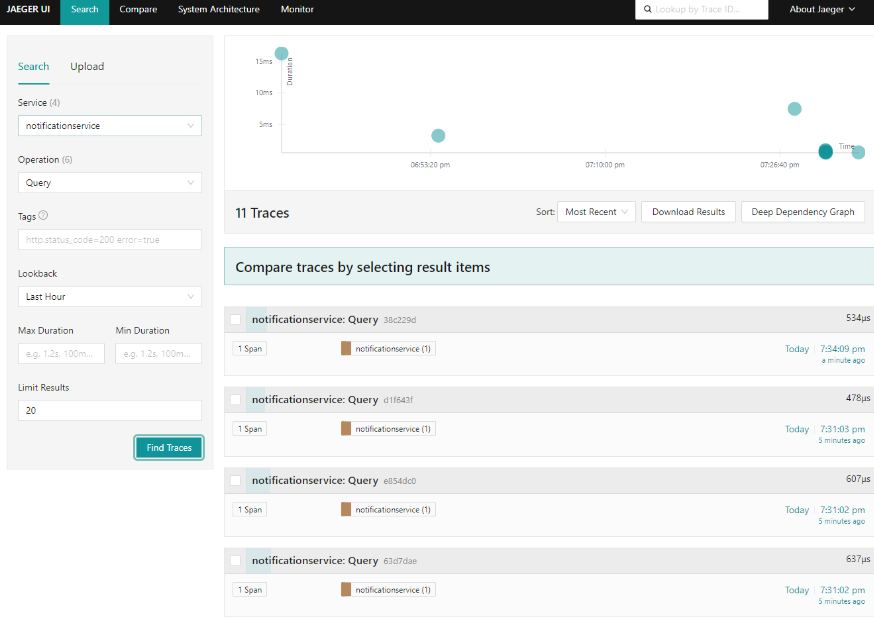


Рисунок 47 – Очередь jaeger NotificationService

Jaeger NotificationService представлен на рисунке 48.

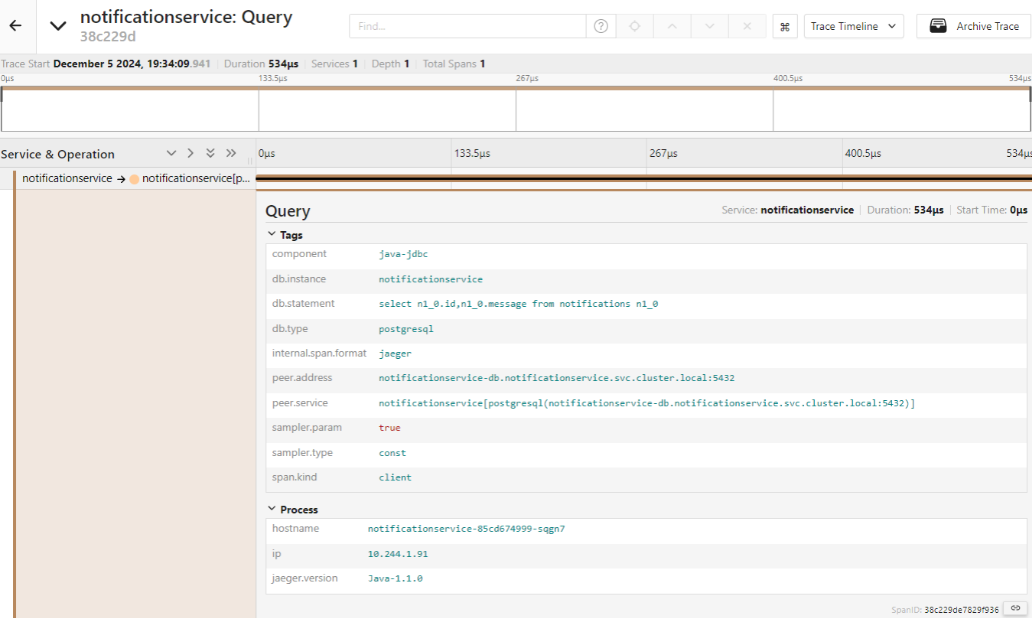


Рисунок 48 – Jaeger NotificationService

Запросы AuthService представлены на рисунках 49-51.

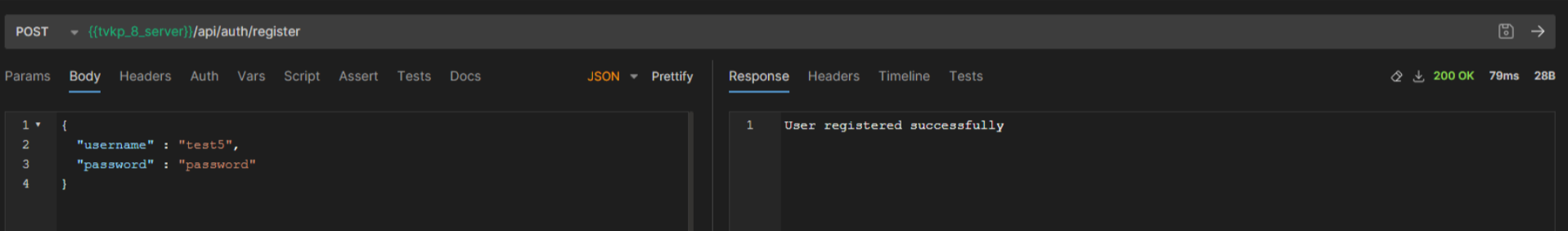


Рисунок 49 – Регистрация AuthService

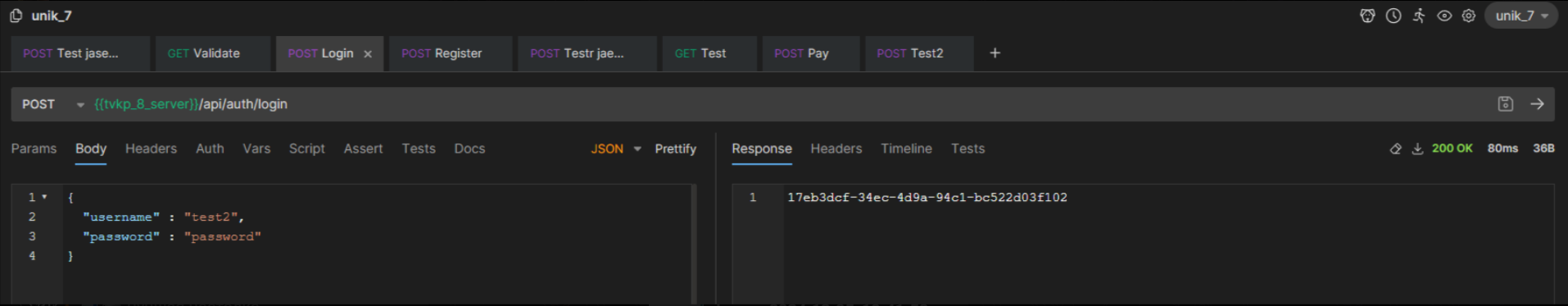


Рисунок 50 – Логин AuthService

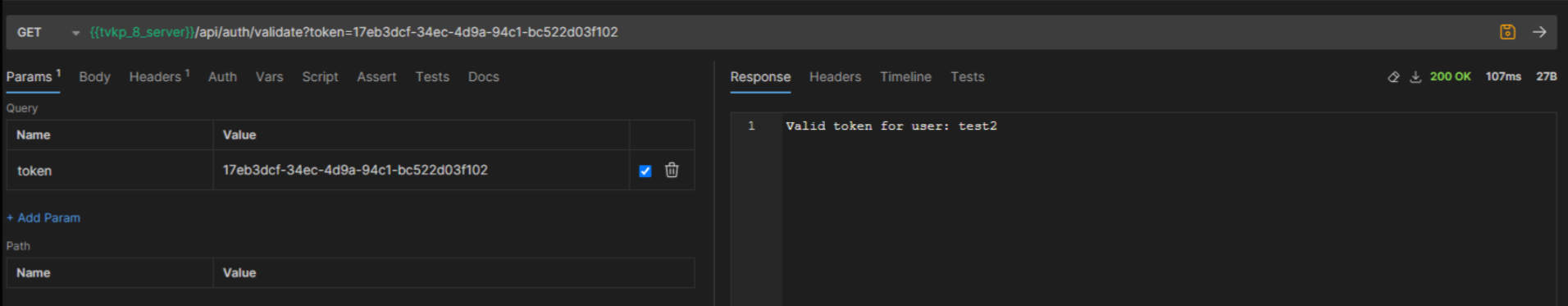


Рисунок 51 – Валидация токена AuthService

GrayLog AuthService представлен на рисунке 52.

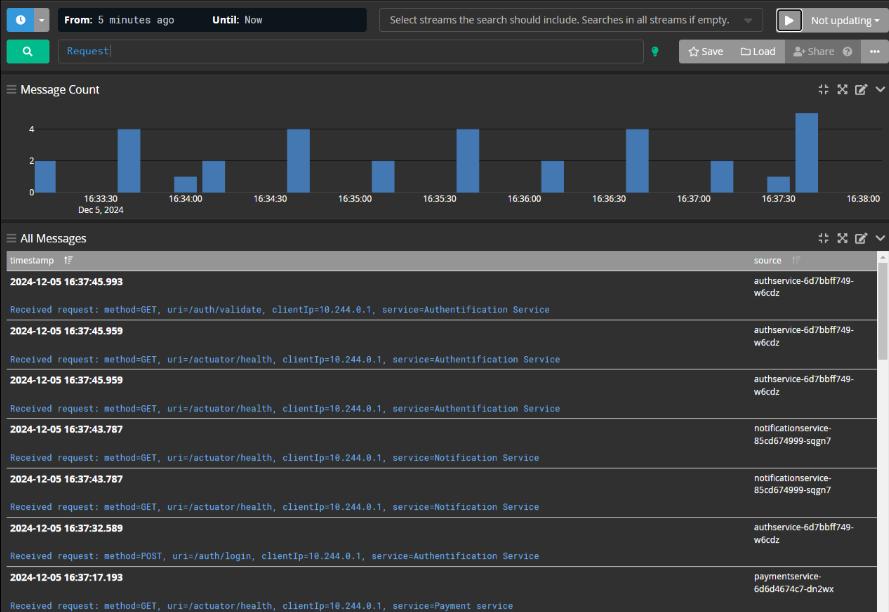


Рисунок 52 – GrayLog AuthService

Очередь jaeger AuthService представлена на рисунке 53.

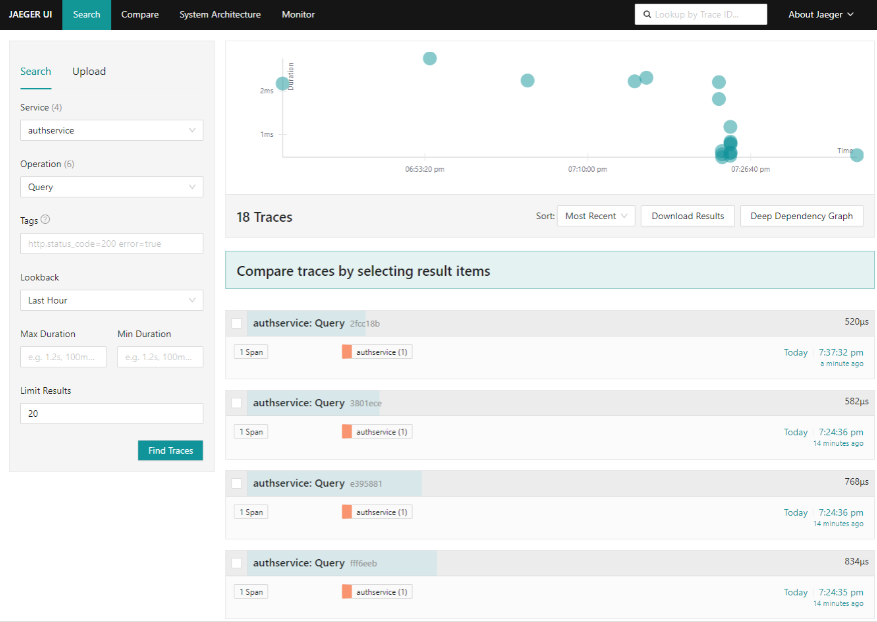
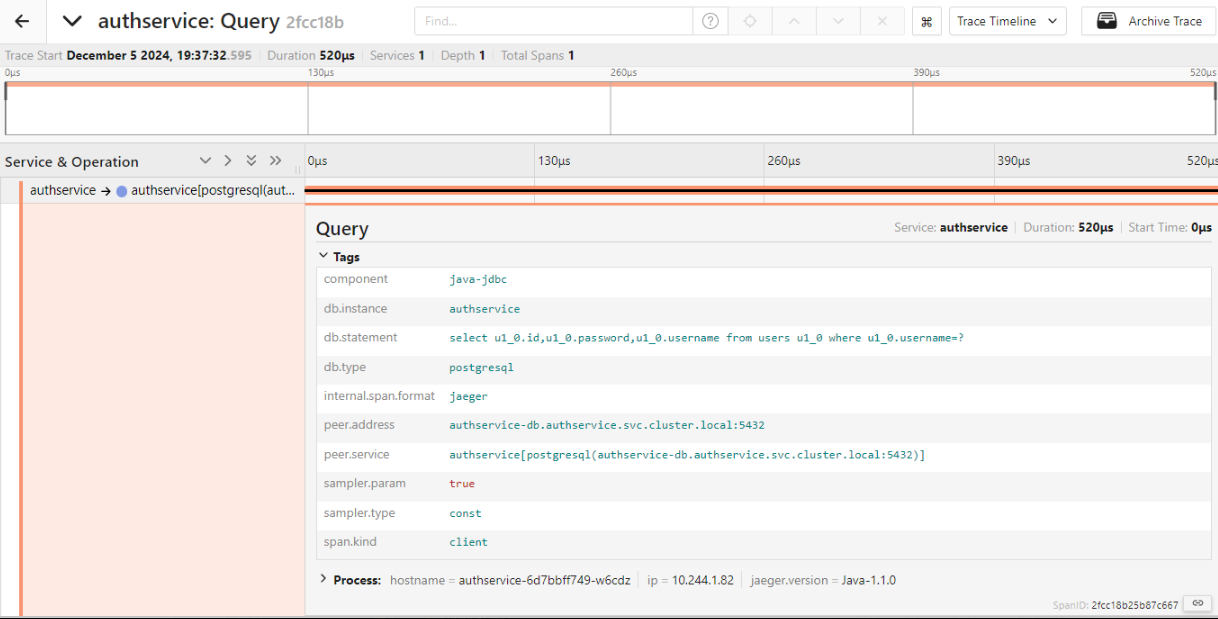


Рисунок 53 – Очередь jaeger AuthService

Jaeger AuthService представлен на рисунке 54.

Рисунок 54 – Jaeger AuthService

**ВОПРОСЫ**

1. Назовите плюсы и минусы использования helm. Какие

форматы конфигурационных файлов он поддерживает? Какие существуют

аналоги?

Плюсы:

* Упрощение управления: Позволяет описывать Kubernetes-ресурсы как единое целое, упрощая деплой.
* Шаблоны: Позволяет создавать шаблоны конфигурационных файлов, снижая дублирование кода.
* Управление версиями: Легко откатываться на предыдущую версию.
* Большое сообщество: Доступно множество готовых чартов.
* Пакетирование приложений: Удобно для разработки, тестирования и распространения Kubernetes-приложений.

Минусы:

* Кривая обучения: Понимание структуры и работы чартов требует времени.
* Сложность поддержки: Большие чарты могут быть трудно читаемыми и поддерживаемыми.
* Потенциальные проблемы безопасности: Устаревшие или плохо поддерживаемые чарты могут содержать уязвимости.

Поддерживаемые форматы конфигурационных файлов:

* YAML (используется для описания Kubernetes-ресурсов и значений переменных в чартах).

Аналоги:

* Kustomize: Альтернатива с акцентом на декларативное управление конфигурацией без использования шаблонов.
* Jsonnet: Используется для генерации Kubernetes-манифестов с гибкой логикой.
* Terraform: Хотя в основном предназначен для инфраструктуры, может использоваться для управления ресурсами Kubernetes.
* Carvel: Инструменты для управления Kubernetes-конфигурацией.

2. Почему для микросервисного взаимодействия используется

шина сообщений? Какие аналоги для микросервисного взаимодействия

существуют? Каковы их преимущества и недостатки?

Причины использования:

* Асинхронное взаимодействие: Уменьшает зависимость от времени ответа сервисов.
* Гибкость: Позволяет сервисам взаимодействовать без жестких связей.
* Масштабируемость: Шина сообщений легко масштабируется для большого количества сообщений.
* Надежность: Обеспечивает гарантированную доставку сообщений.
* Декуплинг: Упрощает добавление новых сервисов без изменений в существующих.

Аналоги и их преимущества/недостатки:

1. Шины сообщений:
   * RabbitMQ, Kafka, NATS, ActiveMQ.
   * Преимущества: Высокая производительность, надежность, поддержка различных паттернов коммуникации (например, publish/subscribe).
   * Недостатки: Сложность настройки и администрирования.
2. REST API:
   * Простота и знакомство для большинства разработчиков.
   * Недостатки: Синхронное взаимодействие, высокие задержки.
3. gRPC:
   * Быстрое двоичное взаимодействие.
   * Недостатки: Менее удобен для интеграции с внешними системами.
4. GraphQL:
   * Гибкость запросов.
   * Недостатки: Сложнее реализовать асинхронные операции.

3. Для чего используется persistent volume в БД? Какие

преимущества и недостатки у паттерна “1 БД на 1 сервис”?

Persistent Volume (PV): Используется для сохранения данных в Kubernetes-кластере, чтобы данные не терялись при перезапуске или удалении подов.

Преимущества PV:

* Надежное хранение данных.
* Поддержка различных провайдеров (NFS, AWS EBS, GCP Persistent Disk и т. д.).
* Разделение данных и приложений.

Паттерн “1 БД на 1 сервис”:

* Преимущества:
  + Независимость сервисов.
  + Упрощенная схема данных для каждого сервиса.
  + Меньший риск взаимных конфликтов.
* Недостатки:
  + Усложнение общей архитектуры.
  + Увеличение затрат на управление несколькими БД.
  + Трудности с интеграцией и анализом данных между сервисами.

4. Почему для чтения данных из БД системами мониторинга

создается отдельный пользователь? К чему может привести использование

аккаунта с лишними правами

Причины создания отдельного пользователя:

* Разделение прав: У пользователя для мониторинга минимальный набор прав (например, только на чтение метрик).
* Повышение безопасности: Ограничивает риск случайных или злонамеренных изменений данных.

Риски использования аккаунта с лишними правами:

* Утечка данных.
* Непреднамеренное изменение или удаление данных.
* Возможность использования злоумышленником для атаки.

5. Какой функционал предоставляет KrakenD? Какие

существуют аналоги?

Функционал KrakenD:

* API-шлюз с поддержкой высокопроизводительных операций.
* Агрегация запросов: Объединяет данные из нескольких источников.
* Авторизация и аутентификация.
* Лимитирование запросов и кэширование.
* Преобразование данных (например, JSON).
* Расширяемость за счет плагинов.

Аналоги:

* Kong Gateway: Популярен за счет расширяемости и множества плагинов.
* NGINX: Используется как API-шлюз или обратный прокси.
* Apigee: Коммерческое решение от Google.
* Traefik: Легкий API-шлюз с поддержкой Kubernetes.
* AWS API Gateway: Решение для API в облаке AWS.

Преимущества KrakenD:

* Высокая производительность.
* Простота настройки и управления.

Недостатки KrakenD:

* Ограниченные возможности расширения по сравнению с некоторыми конкурентами.

вопросы

1. 3 QoS-класса (Quality of Service) в Kubernetes:

Guaranteed: Все запросы и лимиты ресурсов указаны, контейнеры гарантированно получают ресурсы.

Burstable: Заданы запросы ресурсов (requests), но не все лимиты или запросы меньше лимитов.

BestEffort: нет указанных запросов или лимитов, используется оставшаяся производительность узла.

1. Основные ресурсы системы и их единицы измерения в Kubernetes:

**CPU:** измеряется в миллиядрах (mCPU) или ядрах (1 CPU = 1000mCPU).

**Memory**: измеряется в байтах, но обычно используют Mi или Gi (гибибайты).

Storage: измеряется в Gb.

1. Для чего нужен HPA (Horizontal Pod Autoscaler):

Автоматически масштабирует количество подов (горизонтальное масштабирование) в зависимости от нагрузки, основываясь на метриках, таких как использование CPU, памяти или пользовательские метрики.

1. Для чего необходимо устанавливать ограничения в Kubernetes:

Предотвращает перерасход ресурсов контейнерами.

Обеспечивает стабильность системы, избегая ситуаций, когда один контейнер потребляет все ресурсы узла.

Позволяет эффективно распределять ресурсы между контейнерами и узлами.

1. Что будет с узлом при превышении ограничений:

**Превышение лимитов контейнера:**

CPU: Контейнер ограничивается и не может использовать больше лимита.

Memory: Контейнер будет завершён (OOMKilled) при превышении лимита памяти.

**Превышение ресурсов узла:**

Kubernetes начнёт эвакуировать поды с наименьшим QoS-классом (сначала BestEffort, затем Burstable).

Узел может выйти из строя, если ресурсы полностью исчерпаны.

Выводы

В рамках данной практической работы были получены навыки разработки распределенной системы на базе Spring Boot с использованием микросервисной архитектуры. Реализован сервис авторизации с хранением токенов в RedisDB, что позволило ознакомиться с подходами к управлению сессиями и обеспечению безопасности приложений.

Освоена интеграция микросервисов через шину сообщений Kafka, что продемонстрировало преимущества асинхронного взаимодействия и декуплинга между компонентами системы. В качестве базы данных использовалась PostgreSQL с настройкой persistent volume, что позволило изучить особенности хранения данных в Kubernetes и обеспечения их сохранности при перезапуске подов.

Были получены навыки настройки API-шлюза на базе KrakenD, что способствовало пониманию маршрутизации и централизованного управления доступом к микросервисам. Логирование обращений к сервисам было реализовано с использованием Graylog, что помогло освоить работу с централизованной системой сбора и анализа логов.

Настройка мониторинга с использованием Prometheus и Grafana, а также трассировки транзакций с Jaeger, дала возможность изучить инструменты наблюдаемости за системой, анализ метрик и поиск узких мест. Конфигурация автоскейлинга и probes в Kubernetes позволила понять принципы обеспечения отказоустойчивости и масштабируемости приложений.

Практическая работа позволила закрепить теоретические знания о микросервисах и Kubernetes, а также получить ценный опыт их применения в реальных условиях.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Docker Documentation | Docker Documentation – Текст: электронный [сайт]. – URL: <https://docs.docker.com/>

2. Kubernetes – Текст: электронный [сайт]. – URL: <https://kubernetes.io/>

3. Docker Compose Documentation | Docker Documentation – Текст: электронный [сайт]. – URL: <https://docs.docker.com/compose/>

4. 50 вопросов по Docker, которые задают на собеседованиях, и ответы на них | Хабр. – Текст: электронный [сайт]. – URL: <https://habr.com/ru/company/southbridge/blog/528206/>