ПРОЕКТИРОВАНИЕ МИКРОСЕРВИСНОЙ АРХИТЕКТУРЫ В ПРОМЫШЛЕННОЙ СИСТЕМЕ МОНИТОРИНГА ТЕХНИЧЕСКИХ УСТРОЙСТВ

Дмитрийчук Д.И.[1], Дёмин А.Ю[2,3]
[1] НИ ТПУ, ИШИТР, гр. 8К11, did7@tpu.ru
[2]НИ ТПУ, ИШИТР, к.т.н., доцент ОИТ, ad@tpu.ru
[3] НИ ТГУ, ИПМКН, к.т.н. доцент каф. ТОИ

Аннотация

В статье рассматривается процесс проектирования микросервисной архитектуры для промышленной системы мониторинга технических устройств. Анализируются преимущества данного подхода по сравнению с традиционными монолитными. Описываются ключевые технологии, такие как Docker, Kubernetes, Python, Prometheus и Grafana, используемые в реализации системы. Приводится описание основных сервисов, их взаимодействий и методов обеспечения безопасности и мониторинга.

Ключевые слова: микросервисная архитектура, мониторинг, технические устройства, промышленная автоматизация, Python, Kubernetes, Prometheus, Grafana, PostgreSQL.

Введение

Современные опасные производственные объекты (ОПО) требуют постоянного контроля технического состояния оборудования [1]. В условиях высокой нагрузки на технические устройства важно оперативно выявлять неисправности и возможные отказы. Использование традиционных методов мониторинга, основанных на формировании документации и периодическом осмотре оборудования, становится недостаточно эффективным. Эти методы не позволяют обеспечить своевременное реагирование и обладают высоким риском человеческого фактора.

Монолитные программные решения, которые применяются в ряде промышленных систем, также имеют ряд ограничений [2]. Они сложны как в использовании, так и в сопровождении, плохо масштабируются и требуют значительных ресурсов при обновлении. В связи с этим возникает необходимость применения более гибких архитектурных решений, обеспечивающих удобство развертывания, надежность и легкость интеграции с другими системами. Микросервисная архитектура является перспективным направлением развития промышленных систем мониторинга. Она позволяет разбить систему на небольшие, автономные сервисы, которые могут функционировать независимо друг от друга [3]. Это повышает отказоустойчивость, упрощает масштабирование и ускоряет процесс внедрения новых функций.

В последние годы микросервисный подход получил широкое распространение в различных отраслях, включая промышленную автоматизацию [4]. Микросервисы значительно повышают гибкость систем, позволяют быстрее адаптироваться к изменениям в технологических процессах и упрощают поддержку программного обеспечения. В то же время разработчикам необходимо решать вопросы, связанные с обеспечением согласованности данных, организацией взаимодействия сервисов и их мониторингом. Применение инструментов контейнеризации, таких как Docker и docker-compose, а также инструментов оркестрации для развертывания в локальном кластере Kubernetes позволяет эффективно управлять распределенными системами и обеспечивать надежность передачи данных.

В данной работе рассматриваются основные аспекты проектирования системы мониторинга на основе микросервисной архитектуры, а также механизмы организации взаимодействия между сервисами, управления нагрузкой и обеспечения безопасности данных.

Целью работы является проектирование микросервисной архитектуры для промышленной системы мониторинга технических устройств. Основные задачи включают:

• Разработку системы, обеспечивающей отказоустойчивость и масштабируемость.
• Оптимизацию взаимодействия между микросервисами.
• Интеграцию системы с внешними информационными потоками и промышленными платформами.

Основная часть

Для реализации системы выбран язык программирования Python, что обусловлено его развитой экосистемой, поддержкой асинхронного программирования и широким набором инструментов для разработки микросервисных приложений. Основные фреймворки разработки системы – Django и FastAPI, т.к. Django обеспечивает удобную работу с базой данных и систему авторизации, а FastAPI позволяет работать с асинхронными API. Одним из значимых нововведений, которое окажет влияние на производительность микросервисных систем, является отключение GIL (Global Interpreter Lock) в Python 3.13 [5]. Этот механизм исторически ограничивал многопоточное выполнение кода, создавая препятствия для эффективного использования многопроцессорных систем. Их снятие позволит полноценно использовать многопоточность без необходимости прибегать к многопроцессной обработке, что особенно важно для высоконагруженных систем мониторинга.

Асинхронное программирование играет ключевую роль в разработке микросервисных систем [6], обеспечивая эффективную обработку большого количества входящих запросов без блокировки потоков выполнения. В данном проекте используются такие инструменты, как FastAPI и asyncio, которые позволяют реализовать высокопроизводительные API с низкой задержкой. Асинхронные вызовы снижают нагрузку на сервер за счет параллельной обработки задач, таких как взаимодействие с базами данных, передача сообщений через брокеры сообщений.

Микросервисная архитектура, используемая в проекте (рис. 1), предполагает разделение системы на несколько независимых сервисов, каждый из которых выполняет строго определенные функции с возможностью масштабирования отдельных сервисов в зависимости от нагрузки. В рамках данной системы были выделены следующие основные сервисы:

• Сервис отображения данных – обеспечивает агрегирование информации о технических устройствах, обработку данных из различных источников и формирование сводных таблиц.

• Сервис формирования отчетов – собирает и анализирует данные, формирует отчеты и предоставляет их пользователям в удобном формате.

• Сервис уведомлений – отвечает за отправку предупреждений и критически важных сообщений при обнаружении неисправностей или выходе оборудования за допустимые параметры, а также о приближении окончания сроков безопасной эксплуатации.

• Сервис загрузки данных – реализует импорт данных из внешних источников, старых версий системы, CSV и JSON-файлов.

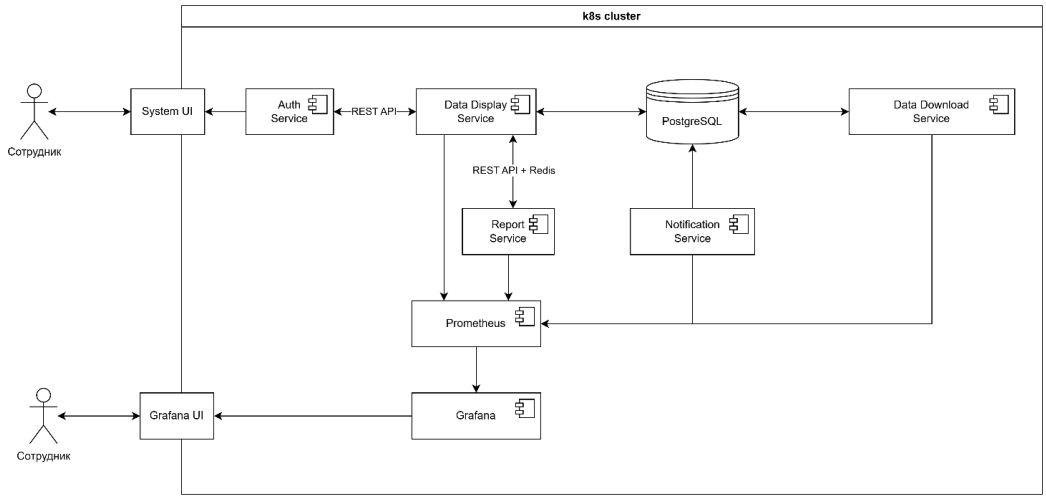

Рис. 1. Архитектура системы в локальном кластере

Для обеспечения эффективного обмена данными между микросервисами используется REST API, позволяющий сервисам взаимодействовать через стандартизированные HTTP-запросы. Взаимодействие между сервисами также дополняется использованием брокера сообщений, кэширования и очереди, что позволяет организовать асинхронную обработку событий и повысить отказоустойчивость системы. Так, например, сервис отображения данных кеширует частые запросы к базе данных, что уменьшает время получения ответа сервисом формирования отчетов.

Важным аспектом проектирования является организация хранения данных. В данном проекте используется единая база данных PostgreSQL, которая обеспечивает централизованное хранение информации и обеспечивает один из механизмов взаимодействия микросервисов: добавления объектов посредством одного сервиса и получение данных из базы данных другим сервисом для дальнейшей обработки (рис. 2). Это позволяет поддерживать целостность данных и упрощает управление транзакциями.

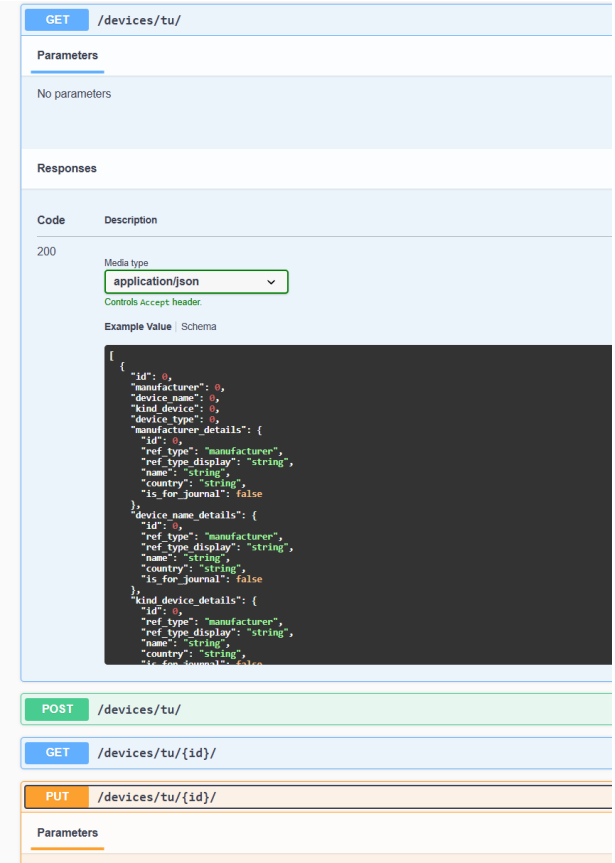

Рис. 2. Эндпоинты API для управления тех. устройствами

Безопасность системы обеспечивается использованием механизмов аутентификации и авторизации. Внедрение протоколов OAuth 2.0 и OpenID Connect позволит контролировать доступ пользователей и защищать передаваемые данные. Для защиты данных также применяются механизмы шифрования и управления доступом на уровне API.

Для автоматического подтягивания учетных данных пользователей, работающих в локальной сети предприятия, без необходимости ручного ввода логинов и паролей, предполагается интеграция системы с существующей корпоративной системой авторизации, использующей учетные записи сотрудников.

Мониторинг состояния системы осуществляется с помощью инструментов Prometheus и Grafana. Prometheus собирает метрики со всех сервисов, Grafana подключается к Prometheus для визуализации и отображает метрики и оповещения, что позволяет оперативно отслеживать работоспособность сервисов, анализировать нагрузку и прогнозировать возможные сбои.

Результаты

Спроектирована микросервисная архитектура системы, определены ключевые компоненты и их функциональные роли. Предложенные механизмы взаимодействия, использование инструментов контейнеризации позволят обеспечить отказоустойчивость, масштабируемость и удобство интеграции системы с внешними источниками данных. Был реализован демонстрационный стенд с сервисом отображения данных, подключённым к Prometheus и Grafana (рис. 3).

Рис. 3. Дэшборд Grafana основного сервиса

Были получены конкретные метрики, такие как количество запросов в секунду, время отклика, количество и длительность SQL-запросов, потребление CPU и оперативной памяти Система сбора метрик и их визуализации позволила оценить производительность сервиса и выявить возможные проблемы.

Заключение

В данной работе рассмотрены основные аспекты проектирования микросервисной архитектуры для промышленной системы мониторинга технических устройств. Применение такого подхода позволяет достичь высокой отказоустойчивости, гибкости масштабирования и упрощения сопровождения системы. Использование асинхронного программирования, контейнеризации и инструментов мониторинга обеспечивает надежную и эффективную работу системы в реальном времени. Перспективными направлениями развития являются интеграция с промышленными IoT-платформами, внедрение машинного обучения для предсказания отказов оборудования и автоматизация обновления сервисов.

Список использованных источников

1. Российская Федерация. Законы. О промышленной безопасности опасных производственных объектов: федер. закон от 21.07.1997 № 116-ФЗ // КонсультантПлюс. – [Электронный ресурс]. – URL: consultant.ru/document/cons_doc_LAW_15234/3b668215163e7 5b4b35ed3e0c0286007fd4ddfbd/ (дата обращения: 27.03.2025).

2. Монолитная и микросервисная архитектура. Сравнение // Хабр: сайт. – 2023. – [Электронный ресурс]. – URL: habr.com/ru/companies/haulmont/articles/758780/ (дата обращения: 27.03.2025).

3. Ричардсон К. Микросервисы. Паттерны разработки и рефакторинга. – Санкт-Петербург: Питер, 2021. – 480 с.

4. Лютягин Д.В., Зюков В.А. Цифровизация производственных процессов в рамках концепции «Индустрия 4.0» // Экономика и управление. – 2021. – № 9. – С. 25-32. –[Электронный ресурс]. – URL: publishing-vak.ru/file/archive-economy-2021-9/25-lyutyagin-zyukov.pdf (дата обращения: 28.03.2025).

5. What's new in Python 3.13 // Python Documentation. – 2024. – URL: docs.python.org/3/whatsnew/3.13.html (дата обращения: 28.03.2025).

6. Gupta P. Patterns for Microservices – Sync vs. Async // DZone: сайт. – 2023. – URL: dzone.com/articles/patterns-for-microservices-sync-vs-async (дата обращения: 28.03.2025).