

Метод построения распределённой системы сбора, обработки и хранения климатических данных на основе микросервисной архитектуры

Вихляев Дмитрий Романович

Приамурский государственный университет имени Шолом-Алейхема

Студент

Аннотация

В данном исследовании содержится описание метода разработки распределённой системы, основанной на микросервисной архитектуре с участием программного обеспечения способствующего организовать обработку климатических, погодных и пространственных данных. Основными инструментами, способствующими организации системы, являются контейнеризация и балансировка нагрузки и очередь и межсервисных сообщений. В результате работы будет описана архитектура системы и связывающие компоненты, а также представлены программные обеспечения и преимущества их использования в описываемой системе.

Ключевые слова: геообработка, контейнеризация, брокер сообщений, микросервисы.

A method for building a distributed system for collecting, processing and storing climate data based on a micro-service architecture

Vikhlyaev Dmitry Romanovich

Sholom-Aleichem Priamursky State University

Student

Abstract

This study describes a method for developing a distributed system based on a micro-service architecture with the participation of software that helps to organize the processing of climatic, weather and spatial data. The main tools contributing to the organization of the system are containerization and load balancing and queues of inter-service messages. As a result of the work, the architecture of the system and the connecting components will be described, as well as software and the advantages of using them in the described system will be presented.

Keywords: geoprocessing, containerization, message broker, microservices.

1 Введение

1.1 Актуальность

В современных условиях мониторинг и анализ климатических данных имеют ключевое значение для решения задач в сферах сельского хозяйства, управления стихийными бедствиями, городской инфраструктуры и экологии. Большие объемы данных, разнообразие источников информации и

необходимость оперативной обработки ставят вопрос о новых методах построения дистрибутивных систем. Распределенная система сбора, обработки и хранения климатических данных на базе микросервисной архитектуры представляет собой современный подход к решению этих задач. Микросервисная архитектура обеспечивает модульность, масштабируемость и независимость развития компонентов, что особенно важно при работе с динамичными нагрузками и объемами данных. Микросервисы выполняют узкоспециализированные функции (сбор данных, обработка, хранение) и работают независимо друг от друга. Каждый микросервис можно масштабировать независимо в зависимости от нагрузки на конкретную функцию (например, увеличить количество сервисов для обработки данных в часы пиковой нагрузки). Ошибка в одном микросервисе не оказывает критического влияния на всю систему. Это облегчает отладку и улучшает стабильность. Каждый микросервис можно разрабатывать, тестировать и обновлять отдельно от других, что упрощает поддержку системы.

1.2 Обзор исследований

В статье Н.А.А. Chaudhary, Т. Margaria предлагают метод интеграции микросервисов в качестве компонентов в средах моделирования для разработки с малым объемом кода. Расширения (прикладных) доменно-специфичных языков, поддерживаемых двумя средами разработки с малым кодом на основе формальных моделей, а именно DIME и Pyro, для включения функций, размещенных на гетерогенных технологиях и платформах [1].

Авторами статьи V.A. Lokhvitsky, V.A. Goncharenko, E.S. Levchik предложена масштабируемая модель микросервиса на основе системы очереди с «охлаждением». Формализация функционирования масштабируемого обслуживания с учетом затрат времени на настройку вычислительных узлов на базе многоканальной неэкспоненциальной системы массового обслуживания с «нагревом» и «охлаждением» каналов [2].

V. Dolzhenko описала разработку микросервисов с использованием фреймворка nestjs. В статье подробно анализируются ключевые аспекты архитектуры микросервисов, включая их независимое развертывание, масштабируемость и легкое обслуживание [3].

Работа Е.О. Гуляева, М.С. Костенко, М.И. Чердынцева посвящена основным принципам миграции из монолитов в микросервисы и её сетевой реализации посредством применения Reverse Proxy для работы микросервисов интерактивного расписания ИММиКН ЮФУ [4].

В статье А. Нуржанкызы рассмотрела лучшие практики контейнеризации для эффективного развертывания микросервисов в системах высокой нагрузки. Представленные в статье рекомендации и примеры позволят разработчикам и инженерам создавать стабильные, масштабируемые и отказоустойчивые архитектуры микросервисов, способные эффективно работать в условиях высокой нагрузки [5].

1.3 Цель исследования

Цель исследования – описать способ организации меж сервисного и клиент-сервисного взаимодействия в микросервисной архитектуре.

2 Материалы и методы

Для описания процессов работы применяются схемы организации компонентов и диаграммы последовательности функций.

3 Результаты и обсуждения

Информационная система, разбитая на сервисы, где каждый сервис изолирован и выполняет одну конкретную задачу, относится к микросервисной архитектуре. В описываемой системе можно сгруппировать некоторые сервисы по методам использования.

Сервисы, реализующие сбор данных, представляют собой отдельные приложения, которые получают показатели с удалённого источника. К получаемой информации относятся метеорологические показатели с метеостанций, спутниковые снимки, картографические данные. Также существует необходимость собирать аналогичные погодные данные с альтернативных источников для сравнительного анализа, также в отдельном сервисе.

Сервис обработки данных выполняет основную задачу, перевести информацию в хранилище данных. При этом данные поступают от сервисов сбора. В хранилище данных информация может перенестись как в сыром виде, так и предварительной переработкой, в которую входят функции заполнения полей присущей структуре хранилища данных, и создание собственных структур. Новые структуры содержат результаты определённого анализа или расчётные значения, такие как индексы, интерполяция и восстановление показателей.

Сервис хранения данных, представляет реляционную базу данных способную хранить картографическую (точки, полигоны) и растровую (изображения) информацию.

К сервисам представления данных относятся приложения, работающие на стороне клиента (frontend) и сервера (backend). Первое, отображает для пользователя начальное представление данных в виде статических файлов (HTML, CSS, JS). Второе, использует интерфейс первого для передачи пользователю динамических данных из хранилища данных.

Сервис, связывающий компоненты системы между собой, выступает в качестве брокера сообщений. Этот компонент системы реализует очередь сообщений, гарантирует доставку сообщений, а также применяется в качестве балансировки нагрузки для сервисов двойников (рис.1).

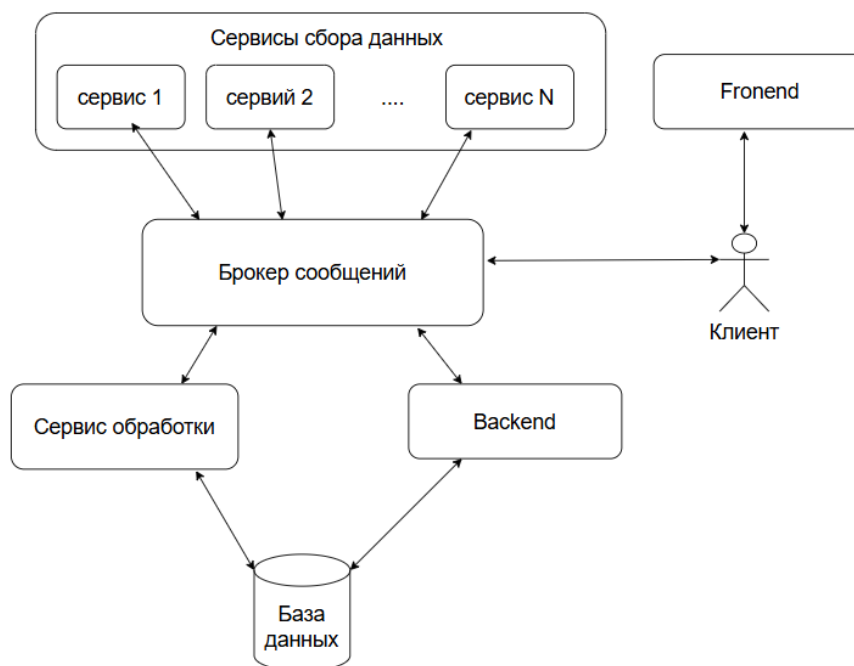


Рис. 1. Схема информационной системы

Конкретное программное обеспечение для каждого сервиса выбрано, как одно из широко применимых, эффективных и легкодоступных инструментов.

Для клиента и сервера выбран язык программирования JavaScript, где на стороне сервера используется интерпретатор Node.js, на стороне клиента фреймворк React.js. Такой подход позволяет реализовать Fullstack разработку, используя только один язык программирования. На данный момент на стороне клиента в браузере можно использовать только JavaScript. Связка Node.js и React.js, является одной из самых популярных для реализации клиент-серверного взаимодействия.

Сервисы сбора и обработки данных реализованы на языке Python, обладающим большим количеством модулей для проведения анализа данных, в том числе инструменты компьютерного зрения и парсеры различных форматов данных.

В качестве брокера сообщений, связующего все сервисы, используется Nats, один из популярных инструментов с большим набором функций. Основанный на протоколе TCP, Nats клиент доступен для всех популярных языков программирования.

Система управления базой данных Postgresql, открытое и надёжное программное обеспечение с богатой документацией. Одним из преимуществ данной системы является расширение postGis, содержащее большинство необходимых функций для работы с непространственными данными.

Основой для создания микросервисной архитектуры является Docker. С помощью данного программного обеспечения можно полностью изолировать каждый сервис, в том числе файловую систему, друг от друга. При этом использовать только один физический сервер. Далее представлено описание

хранения и работа сервисов, взаимодействующих с пользователями и администраторами системы внутри платформы Docker.

В Docker одновременно работают 5 сервисов, а именно frontend, backend, база данных, сервис для управления базой данных и брокер сообщений. Все контейнеры в Docker изолированы друг от друга и изолированы от внешнего мира. Если не добавлять проброс портов, то контейнеры будут полностью работать внутри Docker без возможности подключения к ним снаружи. Клиенты находятся снаружи. Внешний клиент должен подключиться вовнутрь контейнера, который находится в Docker.

Когда клиенты подключаются к frontend, в качестве ответа возвращаются файлы html, css и javascript. Далее на стороне клиента, например, веб-браузере будет производить рендер страниц на основе полученных файлов. При совершении пользователем определённого действия, к которому привязано событие, например, нажатие кнопки. В результате, которого должны быть выведены динамические данные, хранящиеся на сервере или базе данных frontend приложение с клиента должно послать запрос на backend. При этом frontend-сервис в данном процессе уже не принимает участия, веб-браузер, либо мобильное приложение на стороне клиента реализует запрос по инструкции, полученной в статических javascript файлах. В представленной архитектуре обращение к backend-сервису осуществляется не напрямую, а посредством брокера сообщений. После каждого такого запроса от клиента, backend будет обращаться к базе данных. Ответ будет идти в обратном направлении.

Для управления базой данных администратор использует веб-браузер. По HTTP необходимо иметь возможность подключиться к сервису управления базой данных. А уже этот сервис будет подключаться к базе данных.

Чтобы взаимодействовать с системой, нужно открыть доступ к Docker сервису, к сервису брокера сообщений и к сервису управления базой данных. Снаружи к базе данных подключения нет. То есть с базой могут взаимодействовать только сервисы внутри Docker.

У каждого сервиса есть свой порт, на котором определенный сервис работает у frontend порт 3000, у backend порт 5000. Эти порты открыты внутри контейнера, определены программистом на этапе разработки. База данных Postgresql, сервис управления базой данных pgadmin и брокер сообщений Nats используют порт по умолчанию 5432, 80 и 4222 соответственно.

Для того, чтобы была возможность подключения к этим сервисам снаружи, необходимо опубликовать порты для фронт-энда, для бэк-энда и для сервиса управления базой.

Проброс портов создается за счёт пары внешнего и внутреннего портов. На сервере Docker пробрасывает все соединения к внешнему порту сервера на внутренний порт контейнера.

У Frontend приложения, которое работает на порту 3000, сопоставится соответствующий внешний порт 3333. Брокер сообщений Nats получил внешний порт 4222 соответствующий внутреннему порту. Сервис управления базой данных открыт снаружи по порту 8080.

При этом внешнего доступа к базе данных и backend-сервису нет, так как отсутствует проброс портов. Поэтому снаружи к этим контейнерам подключиться нельзя. Это дополнительный уровень защиты базы данных и так обычно поступают в любых реальных приложениях. База данных должна быть скрыта от внешних пользователей. Такой подход даёт возможность в будущем при увеличении нагрузки создать двойник backend-сервиса на другой машине или контейнере и использовать функцию балансировщика нагрузки внутри Nats для равномерного распределения запросов от клиента. Тогда не нужно будет менять архитектуру системы, и увеличивать аппаратные возможности сервера (рис. 2).

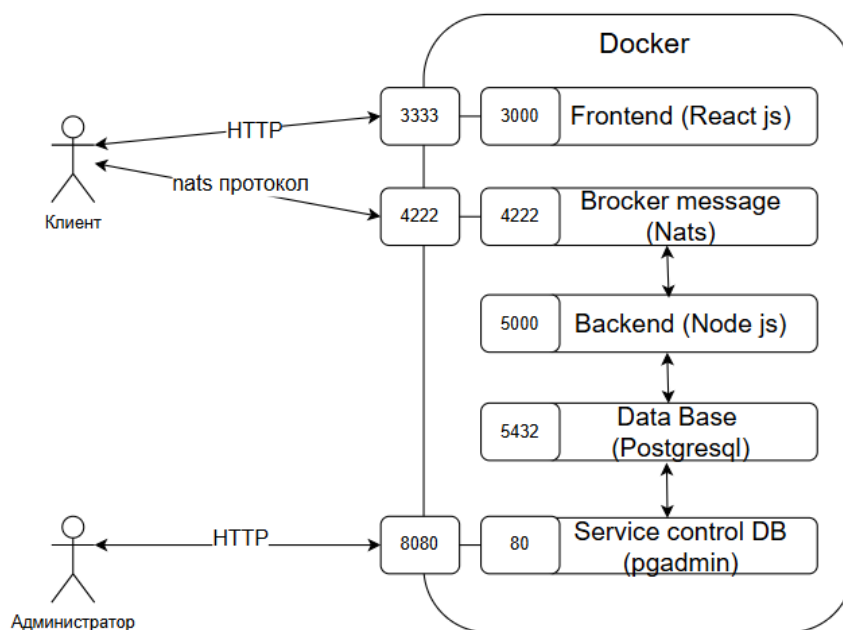


Рис. 2. Схема взаимодействия пользователей с сервисами внутри Docker

Сервисы сбора и обработки данных не взаимодействуют с пользователем. Группа сервисов сбора данных имеют идентичную функциональную последовательность. Это роботы, постоянно работающие на сервере, ожидающие определённого момента срабатывает функция планировщик, по времени вызывая метод сбора данных. Из полученных данных, будь то текстовые графические или бинарные, извлекаются нужные показатели, при необходимости декодируются и помещаются в структуру данных и приводятся к шаблонному виду для дальнейшей передачи другому сервису (рис.3). Далее сервис обработки, принимает информацию в понятном для него виде и реализует собственные методы.

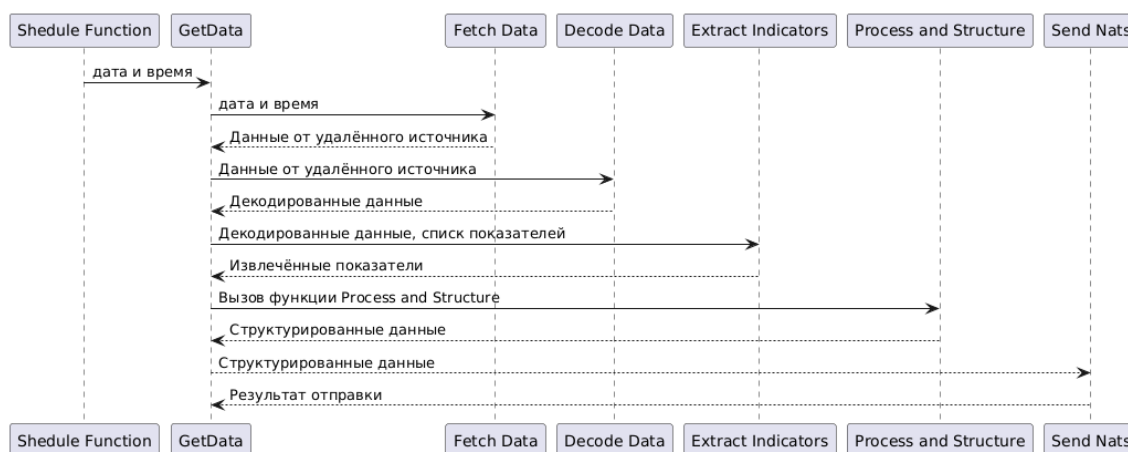


Рис. 3. Внутренняя организация работы сервисов сбора данных

В контексте брокера сообщений клиент, который отправляет сообщения, называется издателем, а клиент получатель подписчиком. Подписчики подписываются на определённую тему (topic), тем самым регистрируясь, как получатели сообщений в NATS сервере. Издатель, отправляя сообщения, не имеет понятия, где находится получатель и сколько их, ему нужно только отправить сообщение и указать тему. Брокер сообщений при получении нового сообщения для конкретной темы организует очередь сообщений для доставки содержания всем подписчикам. Сервис NATS содержит набор функций JetStream, данная система хранит сообщения в NATS, включая подтверждения доставки сообщений. Отправитель может автоматически получить подтверждение о том, что сообщение было успешно сохранено в хранилище или доставлено подписчику. Подписчик вручную отправляет подтверждение о получении сообщения. Если подтверждение не отправлено в течение определённого времени, сообщение повторно доставляется. JetStream автоматически подтверждает получение сообщения, если подписчик не отправляет отрицательного подтверждения. После публикации сообщения брокер возвращает отправителю PubAck, указывающий, что сообщение успешно записано в хранилище (рис.4).

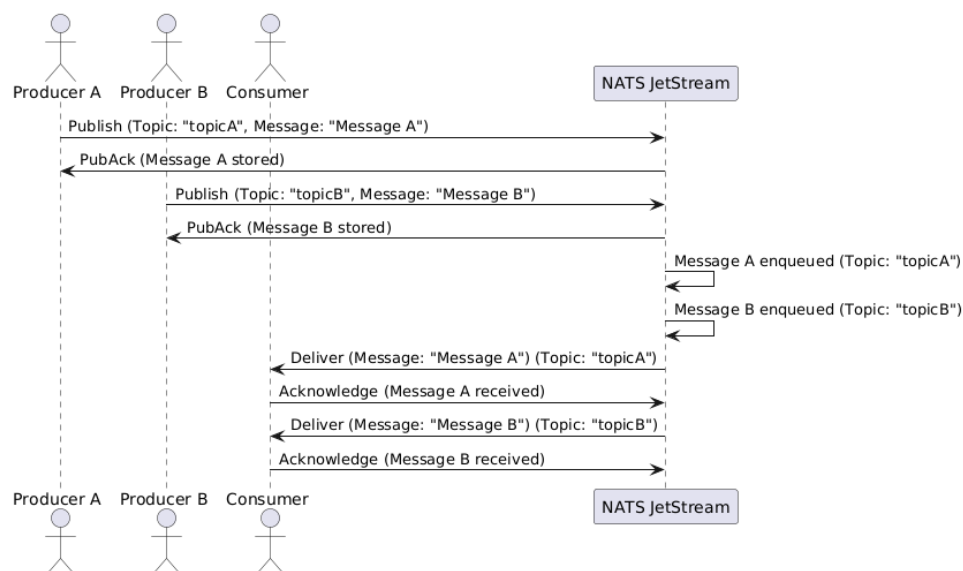


Рис. 4. Диаграмма организации доставки сообщений

Описанный подход организации информационной системы широко применяется в современном мире в частности для реализации распределённых систем. Обеспечивается широкая масштабируемость и отказоустойчивость за счёт использования двух систем.

Контейнеризация Docker позволяет не только изолировать каждый сервис, но и содержать инструкции для пересоздания как одного, так и группы контейнеров. С помощью файла `docker-compose.yml` реализуется декларативный подход описания контейнеров, наличие определённых версий образов, инструкция создания контейнеров, первичная настройка. Достаточно написать инструкции один раз, чтобы получить возможность пересоздать систему на любой машине с установленным движком Docker.

Организация меж сервисного взаимодействия, основанного на брокере сообщений, повышает отказоустойчивость системы, даёт возможность распределять нагрузку в нагруженных местах и предоставляет ряд готовых инструментов для мониторинга работы системы.

В результате данной статьи, приведено описание метода построения распределённой системы сбора, обработки и хранения климатических данных на основе микросервисной архитектуры. Описаны все компоненты системы и организация взаимодействия. Представлены названия программного обеспечения и преимущества и использования для каждого компонента в описываемом контексте.

Библиографический список

1. Chaudhary H.A.A., Margaria T. Integration of micro-services as components in modeling environments for low code development // Proceedings of the Institute for System Programming of the RAS. 2021. Т. 33. № 4. С. 19-30.
2. Lokhvitsky V.A., Goncharenko V.A., Levchik E.S. A scalable microservice model based on a queuing system with "cooling" // Intellectual Technologies on

- Transport. 2022. № 4 (32). С. 46-51.
3. Dolzhenko V. Development of microservices using nestjs: architecture and practical examples // Universum: технические науки. 2024. № 6-6 (123). С. 7-11.
 4. Гуляева Е.О., Костенко М.С., Чердынцева М.И. Разработка микросервисов для интерактивного расписания ИММиКН ЮФУ // В сборнике: Современные информационные технологии: тенденции и перспективы развития. Материалы XXX научной конференции. Редакционная коллегия – Г. В. Муратова, Е. Е. Блинова, Я. М. Демяненко, В. А. Нестеренко, А. Н. Соловьев, В. Ю. Тополов. 2023. С. 147-148.
 5. Нуржанкызы А. Лучшие практики контейнеризации для развертывания микросервисов в системах высокой нагрузки // Научный аспект. 2024. Т. 44. № 4. С. 5858-5862.
 6. Овакянян Л.К. Архитектура микросервисов как современный инструмент коммуникаций в экосистеме // В сборнике: Экономика, управление и политика: проблемы теории практики и взаимодействия. Материалы ежегодной Всероссийской студенческой научно-практической конференции. Тверь, 2024. С. 241-246.
 7. Корниенко Д.В., Никулин А.В. Визуализация архитектур информационных систем, основанных на микросервисах, с использованием данных opentelemetry // Computational Nanotechnology. 2024. Т. 11. № 1. С. 94-103.
 8. Остапенко Д.А. Применение java-микросервисов для построения эффективных и гибких распределенных систем // В сборнике: Технологическое предпринимательство: проблемы проектирования, применения и безопасности информационных систем в условиях цифровой экономики и реализации инновационного и технологического предпринимательства. Материалы международной студенческой научно-практической конференции. Ростов-на-Дону, 2023. С. 319-322.
 9. Глумов К.С. Безопасность микросервисов: управление секретами и безопасная аутентификация // Актуальные исследования. 2024. № 45-1 (227). С. 23-29.
 10. Меркулов А.С. Роль service discovery в динамическом масштабировании микросервисов // Актуальные исследования. 2024. № 45-1 (227). С. 30-32.