СПОСОБЫ ПОДДЕРЖАНИЯ ЦЕЛОСТНОСТИ ДАННЫХ В ПРИЛОЖЕНИЯХ С МИКРОСЕРВИСНОЙ АРХИТЕКТУРОЙ

Кнюх А. И., Кононов А. А. - студенты гр.051002

Белорусский государственный университет информатики и радиоэлектроникиг. Минск, Республика Беларусь

Шостак Е. В. – ассистент каф. ПОИТ

Аннотация. В настоящее время большой популярностью пользуется при создании приложений микросервисная архитектура. Она позволяет комфортно горизонтально масштабировать разработку проекта. Поскольку микросервисы работают автономно в переделах приложения, достаточно часто возникают вопросы о поддержании целостности данных, их актуальности в реальном времени и избегании потерь данных. Сетевая связь позволяет решить данные проблемы, однако она вносит нюансы в организацию ее работы, которые могут решиться различными стратегиями, которые перечислены в данной статье.

Ключевые слова. Микросервисная архитектура, микросервис, масштабирование, интерфейс, база данных, асинхронность, брокер сообщений, транзакция, шаблон проектирования, шаблон Inbox, шаблон Outbox.

В настоящее время одной из наиболее популярных архитектур программных средств является микросервисная архитектура, рассматривающая программное средство как совокупность множества взаимодействуйщих сервисов. В свою очередь каждый микросервис выполняет одну конкретную функцию и может быть разработан и развернут независимо от других. Данный подход неоптимален в разработке небольших приложений, где обеспечение возможностей гибкого горизонтального масштабирования не является приоритетной задачей проектирования. Однако в больших проектах микросервисная архитектура позволяет удобно развивать проект, концентрироваться на отдельных функциях программного средства, распределять задачи между множеством команд разработчиков, проще обеспечить непрерывную поставку программного средства.

В канонической микросервисной архитектуре предполагается зависимость микросервисов другот друга только по предоставляемому ими интерфейсу. Из этого следует, что каждый отдельный микросервис должен инкапсулировать данные, с которыми он непосредственно взаимодействует. Так,для нескольких экземпляров микросервиса, каждый из которых использует для своей работы одну базу данных, создается несколько экземпляров этой базы данных. Кроме того, возникает необходимость разделить одни данные между несколькими базами данных, что усложняет поддержание их целостности по сравнению с единым централизованным хранилищем.

Сетевой характер связи между микросервисами и разное время выполнения функций микросервисами вносит в архитектуру программного средства точки потенциального отказа. Из-за задержек в операциях, требующих взаимодействия нескольких микросервисов, появляется опасность потери согласованности данных. Может возникнуть необходимость реализации механизмов балансировки нагрузки и повторных попыток.

Одним из решений проблемы задержек в микросервисной архитектуре является применение брокера сообщений, за счет чего достигается асинхронность в работе программного средства: микросервис-источник передает свой запрос как сообщение брокеру. Тот поддерживает несколько очередей сообщений, в некоторые из них направляются полученные брокером сообщения. Затем по некоторой логике сообщения передаются микросервисам-получателям. По завершению выполнения запрашиваемой функции микросервис-источник получает результат выполнения функции.

Существует несколько стратегий обеспечения согласованности данных:

1. Сильная согласованность: Сильная согласованность означает, что все узлы системы в любой момент времени имеют одинаковую информацию о состоянии данных. Как только данные изменяются в одной части системы, эти изменения сразу же становятся доступными для всех остальных частей системы. Чтобы достичь сильной согласованности, часто применяют блокирующие операции, которыегарантируют, что изменение данных будет завершено полностью перед тем, как оно станет видимым для других операций. Системы, обеспечивающие сильную согласованность, обычно следуют принципам ACID-транзакций, где соблюдается атомарность, согласованность, изоляция идолговечность изменений.

2. Слабая согласованность: В системах со слабой согласованностью изменения данных распространяются по всем узлам, но без определенной временной гарантии. Системы с таким типом согласованности часто оперируют асинхронными методами репликации и обновления данных. Отсутствие немедленного обновления данных во всей системе может привести к временным расхождениям.

3. Событийная согласованность: Событийная согласованность обеспечивает, что если изменения в данных прекратятся, то после некоторого времени все узлы системы сойдутся к одному итому же состоянию данных. Часто реализуется через асинхронные процессы репликации

исинхронизации данных. Обеспечивает баланс между согласованностью и доступностью, принимая временное расхождение данных.

В микросервисной архитектуре часто используется публикация событий для обмена информацией между сервисами. Однако, может возникнуть проблема из-за того, что сохранение состояния и публикация события не происходят одновременно, что может привести к несогласованности данных в системе. Для решения этой проблемы предлагается сначала сохранять изменения и создавать событие в рамках одной транзакции, а затем публиковать это событие после успешного сохранения данных.

Существует два шаблона проектирования, обеспечивающие гарантированность доставки:

1. В рамках шаблона Inbox все сообщения, поступающие в систему, сначала сохраняются в специальном "ящике" (inbox), а затем обрабатываются по мере возможности.

2. В рамках шаблона Outbox все сообщения, которые требуется отправить или обработать, сначала сохраняются в специальном "ящике", а затем поочередно извлекаются и отправляются внешнему сервису или компоненту.

Для управления транзакциями в распределенной среде используются следующие шаблоны проектирования:

1. Двухфазный коммит – шаблон проектирования, который используется для обеспечения атомарности транзакций в распределенных системах. Он представляет собой протокол для координации выполнения транзакций между несколькими участниками системы, гарантируя либо успешное завершение всех операций, либо их откат при возникновении ошибки. Однако, двухфазныйкоммит имеет недостатки, такие как длительное время блокировки ресурсов во время выполнения транзакций и возможность блокировки всей системы при отказе координатора. Поэтому его использование требует внимательного проектирования и учета особенностей конкретной системы.

2. Шаблон SAGA – шаблон проектирования, который используется для управления транзакциями в распределенной системе, где операции могут влиять на несколько сервисов или компонентов. Сага представляет собой последовательность локальных транзакций, которые выполняются в каждом участнике системы, чтобы достичь глобальной целостности операции.

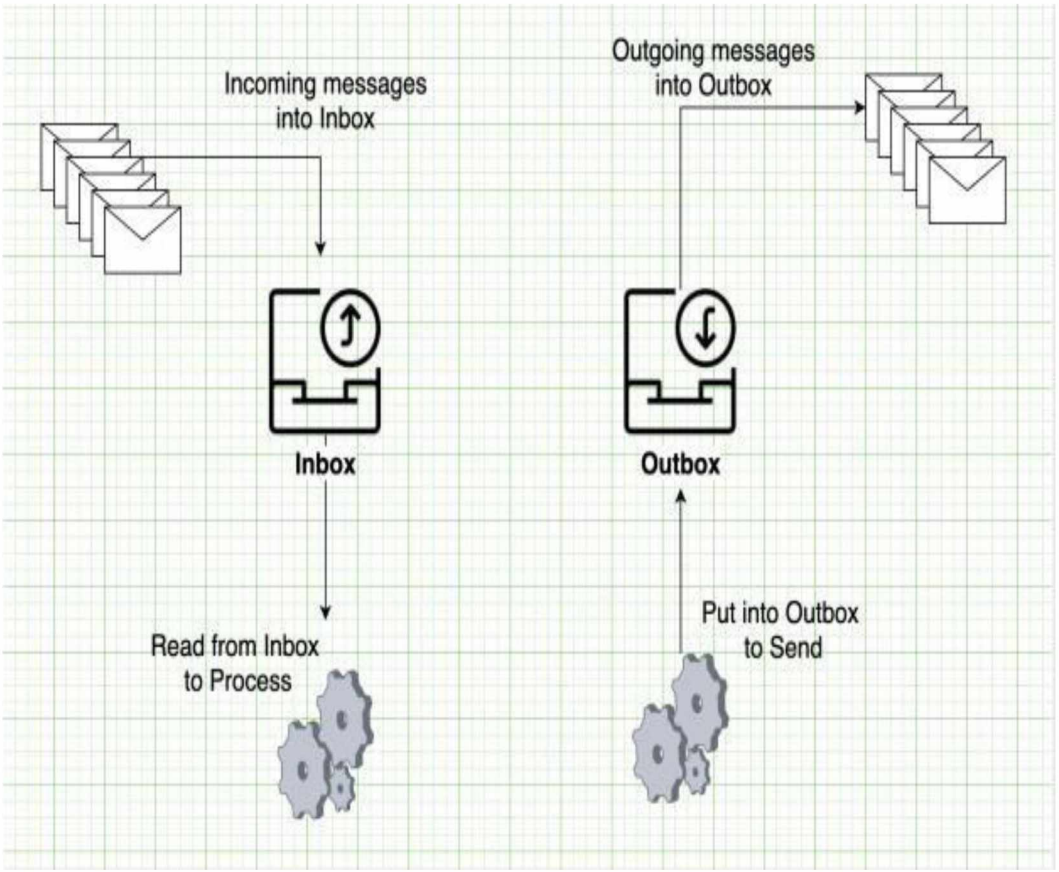

Рисунок 1. Схематическое изображение шаблонов Inbox и Outbox

Список использованных источников:

1. Balalaie, A.; Heydarnoori, A.; Jamshidi, P. Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture / IEEE Software 2016.— P. 42-52.

2. Микросервисы. Паттерны разработки и рефакторинга / Крис Ричардсон. — СПб.: Питер, 2019. — С. 544.

3. What are Microservices? [Электронный ресурс]. – Режим доступа https://aws.amazon.com/ru/microservices/. – Дата доступа: 15.04.2024.