# ПОСТРОЕНИЕ СОБЫТИЙНО-ОРИЕНТИРОВАННОЙ АРХИТЕКТУРЫ В СИСТЕМЕ КРЕДИТНОГО СКОРИНГА

*Зарубин В.А*[1]
*Научный руководитель: Демин А.Ю.*[2,3]
[1]*НИ ТПУ, ИШИТР, гр. 8К13, студент, vaz30@tpu.ru*
[2]*НИ ТПУ, ИШИТР, к.т.н., доцент ОИТ, ad@tpu.ru*
[3]*НИ ТГУ, ИПМКН, к.т.н., доцент каф. ТОИ, adyomin@mail.tsu.ru*

**Аннотация**

В статье рассматривается построение событийно-ориентированной архитектуры для системы кредитного скоринга с использованием брокера сообщений Apache Kafka. Описаны преимущества этой архитектуры в контексте повышения масштабируемости, отказоустойчивости и минимизации задержек в передаче данных между микросервисами и моделями машинного обучения.

**Ключевые слова:** событийно-ориентированная архитектура, микросервисы, брокер сообщений, высоконагруженные системы.

## Введение

Кредитный скоринг представляет собой один из ключевых элементов современной финансовой системы, обеспечивая возможность объективной оценки кредитоспособности заемщиков на основе множества факторов. В процессе принятия решений широко используются модели машинного обучения, анализирующие разнообразные параметры, включая кредитную историю, транзакционную активность, демографические характеристики и иные данные. Эффективность подобных моделей во многом зависит от качества и актуальности поступающей информации, что требует надежных механизмов сбора, обработки и передачи данных [1].

Традиционные подходы к организации потоков данных в скоринговых системах зачастую опираются на архитектуры с жестко заданными каналами взаимодействия, в которых компоненты напрямую обмениваются информацией. Такой подход ограничивает гибкость системы, усложняет масштабирование и увеличивает задержки при передаче данных. В условиях динамичной финансовой среды, где требуется оперативное обновление информации и адаптация моделей к изменяющимся условиям, подобные ограничения становятся существенным препятствием [2].

В данной статье будет рассмотрен процесс построения событийно-ориентированной архитектуры в системе кредитного скоринга, а также принципы интеграции Apache Kafka в механизм передачи данных для машинного обучения.

## Основная часть

Рассматриваемая система кредитного скоринга построена на микросервисной архитектуре, где каждый сервис выполняет строго определенные задачи, имеет собственное API и использует реляционные или нереляционные базы данных. В данной архитектуре имеются разные варианты для передачи данных в сервис с ML моделью.

Один из распространенных вариантов заключается в синхронной передаче данных через REST API. В этом случае, при запросе на скоринг, данные с помощью API передаются по HTTP в сервис с ML-моделью, который в ответ возвращает предсказанный скоринговый балл. Такой метод обеспечивает мгновенную обработку запроса, однако он накладывает жесткие требования к доступности сервиса с моделью. При высоких нагрузках возможны задержки и сбои, что делает этот вариант неустойчивым при масштабировании системы [4].

Другой вариант – сохранение скоринговых запросов в реляционной или NoSQL базе данных, откуда сервис ML-модели периодически извлекает новые записи для обработки. Это

позволяет разгрузить основную систему и выполнять предсказания в асинхронном режиме, но такой подход увеличивает задержки в обработке данных, поскольку модель не получает информацию в момент запроса, а работает с задержкой, зависящей от частоты выборки данных. Кроме того, использование базы данных в качестве промежуточного буфера требует дополнительных ресурсов и усложняет архитектуру за счет необходимости управления процессами записи и чтения.

В рассматриваемой системе ранее использовался централизованный парсер, который объединял логи из различных микросервисов и сохранял их в кластер Hadoop Distributed File System. Данный подход позволял агрегировать данные из разных источников, однако сбор данных представлял из себя объёмный по времени и затратам процесс, что создавало узкое место в системе. Кроме того, сложность логики парсинга затрудняла масштабирование сервисов, так как с изменением сервисов требовалось изменять парсер.

Хорошим решением проблемы в данном контексте стало использование событийно-ориентированной архитектуры с брокером сообщений. В этом случае каждый микросервис на своём уровне формирует лог в JSON-формате и публикует его в определенный топик. Сервис с ML-моделью подписывается на этот топик и получает данные в режиме реального времени, что обеспечивает минимальные задержки в обработке.


*Рис. 1. Передача сообщений через брокер*

В качестве брокера сообщений выбрана Apache Kafka, поскольку она лучше подходит для обработки потоков событий в реальном времени по сравнению с RabbitMQ. Kafka обеспечивает более высокую пропускную способность за счет лог-ориентированной архитектуры, эффективного хранения сообщений на диске и возможности горизонтального масштабирования за счет партиционирования топиков. В отличие от RabbitMQ, который ориентирован на обработку отдельных сообщений и требует подтверждений доставки, Kafka поддерживает механизм репликации и ретенцию сообщений, что позволяет не только передавать данные, но и воспроизводить их в случае необходимости. Эти свойства делают Kafka одним из наиболее подходящих решений для высоконагруженных систем, работающих с потоками данных в режиме реального времени [5].

Однако системе важно не только передавать данные в модель, но и сохранять события, так как они представляют ценность для дальнейшего анализа и обучения модели. Хранение логов обеспечивает возможность детального мониторинга системы, помогает в отладке и ускоряет выявление потенциальных проблем. Для этого используется связка Apache Kafka и ClickHouse, где Kafka выполняет роль надежного транспорта событий, а ClickHouse служит основным хранилищем для анализа данных.

После публикации события в топик Kafka потребитель, отвечающий за обработку данных, получает его и записывает в ClickHouse. Такой подход гарантирует, что ни одно событие не будет потеряно, даже если база данных временно недоступна, так как Kafka может хранить сообщения в течение определенного времени. ClickHouse, в свою очередь, обеспечивает высокую скорость обработки и хранения больших объемов данных благодаря колонночной структуре и оптимизированным аналитическим запросам. Это позволяет эффективно агрегировать информацию, строить отчеты по качеству модели и адаптировать систему к изменяющимся требованиям [6].

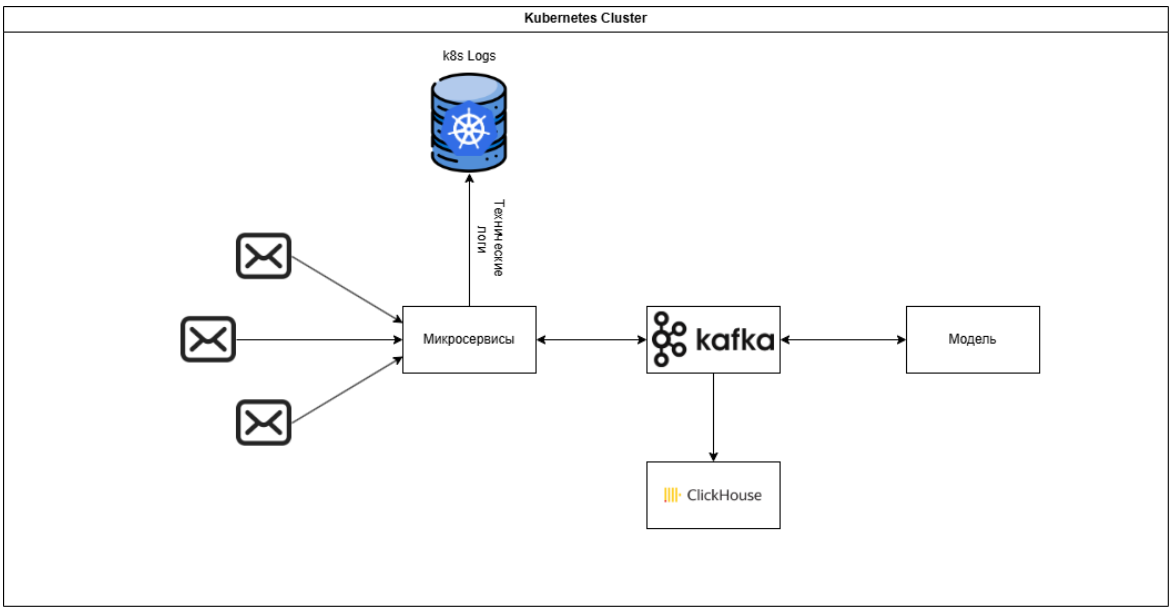


*Рис. 2. Архитектура системы с использованием Kubernetes*

Для обеспечения отказоустойчивости и масштабируемости система разворачивается в кластере Kubernetes, который управляет всеми компонентами, включая Kafka, ClickHouse и микросервисы скоринга. В событийно-ориентированной архитектуре это критически важно, так как Kubernetes автоматически балансирует нагрузку, перезапускает упавшие сервисы и масштабирует их в зависимости от потока событий. Kafka обеспечивает надежную доставку данных, ClickHouse эффективно обрабатывает большие объемы информации, а микросервисы динамически адаптируются к изменяющимся требованиям [7].

Было проведено нагрузочное тестирование обоих систем, которое показало следующие результаты.

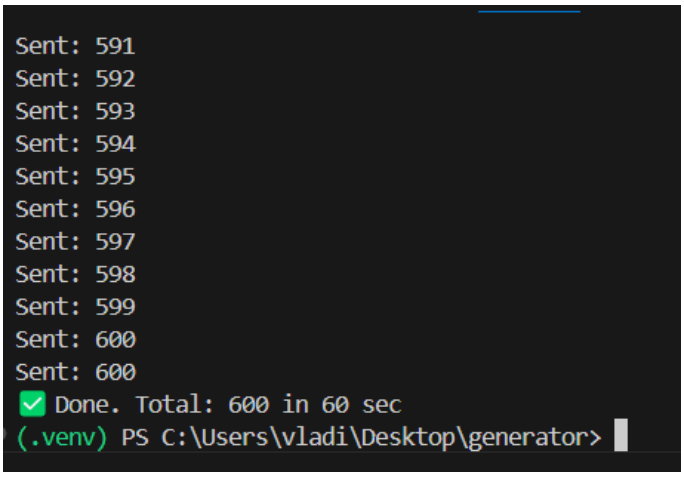


*Рис. 3. Отправка запросов по REST в течение минуты*

Первоначальная система использующая централизованный парсер смогла принять только 600 запросов за 60 секунуд. Связано это с тем, что REST API работает синхронно, а каждый вызов обрабатывается последовательно. Сначала происходит ожидание ответа от сервиса, а затем запись логов в HDFS, что создаёт блокирующую цепочку действий и увеличивает время ответа на каждую операцию.

```
2025-04-10 16:29:28 Sent: 4800
2025-04-10 16:29:28 Sent: 5000
2025-04-10 16:29:28 Sent: 5200
2025-04-10 16:29:28 Sent: 5400
2025-04-10 16:29:28 Sent: 5600
2025-04-10 16:29:28 Sent: 5800
2025-04-10 16:29:28 Sent: 6000
2025-04-10 16:29:28 Sent: 6200
2025-04-10 16:29:28 Sent: 6400
2025-04-10 16:29:28 Sent: 6600
2025-04-10 16:29:28 Sent: 6800
2025-04-10 16:29:28 Sent: 7000
2025-04-10 16:29:28 Sent: 7200
2025-04-10 16:29:28 Sent: 7400
2025-04-10 16:29:28 Sent: 7600
2025-04-10 16:29:28 Sent: 7800
2025-04-10 16:29:28 Sent: 8000
2025-04-10 16:29:28 Sent: 8200
2025-04-10 16:29:28 Sent: 8400
2025-04-10 16:29:28 Sent: 8600
2025-04-10 16:29:28 Sent: 8800
2025-04-10 16:29:28 Sent: 9000
2025-04-10 16:29:28 Sent: 9200
2025-04-10 16:29:28 Done: 9200 in 60 sec
```

*Рис. 4. Отправка событий через Kafka в течение минуты*

Использование событийно-ориентированной архитектуры же позволило добиться принципиально иной производительности. За то же самое время новая система приняла 9200 сообщений, что 15 раз больше, чем в системе с использованием парсера. Такой прирост обусловлен тем, что сервисы производители работают асинхронно и не зависят от скорости обработки сообщений потребителем. Каждое событие мгновенно публикуется в брокер сообщений Kafka без ожидания ответа, что исключает задержки, характерные для REST-вызовов. Кроме того, постоянные соединения и бинарный протокол передачи данных позволяют снизить накладные расходы.

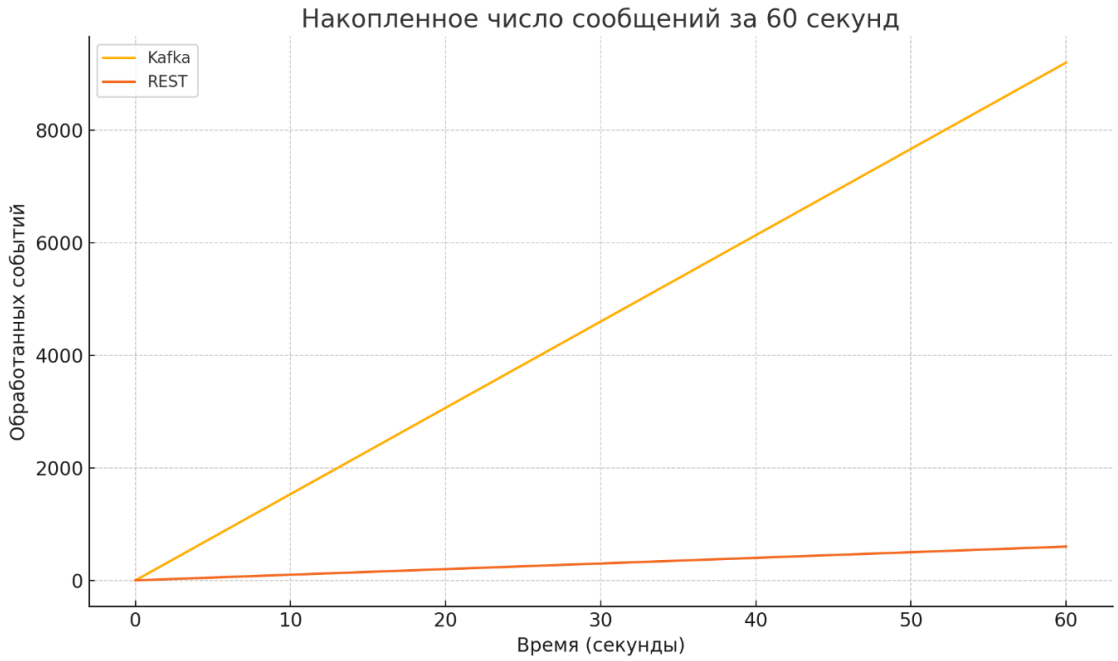


*Рис. 4. График сравнения обработки запросов с использованием REST и Kafka*

## Результаты

Внедрение событийно-ориентированной архитектуры повысило производительность и устойчивость системы за счет отказа от централизованного парсера и перехода к унифицированному формату логов. Теперь каждый микросервис публикует события напрямую в Kafka, исключая сложную логику парсинга и предварительную обработку данных. Это позволило снизить нагрузку на систему логирования, ускорило передачу данных в хранилище и упростило процесс масштабирования. Передача данных через Kafka происходит намного быстрее за счёт асинхронной модели обмена сообщениями.

Ранее парсер связывал логи различных сервисов и загружал их в Hadoop, что приводило к задержкам по времени. Теперь события хранятся в ClickHouse в структурированном виде, что уменьшило объем избыточных данных на кластере HDFS. Также снизился сбор метрик в Prometheus, так как все данные поступают в единый поток событий без дополнительной обработки.

Дополнительным преимуществом стала ослабленная связанность сервисов. Каждый компонент системы работает независимо, а обмен данными происходит через брокер сообщений. Это упростило интеграцию новых сервисов, сделало систему более гибкой и отказоустойчивой. Даже при временной недоступности отдельных компонентов данные не теряются, так как Kafka сохраняет их до момента обработки.

Однако следует учитывать, что событийно-ориентированная архитектура может быть избыточной для небольших систем с невысокой нагрузкой, где традиционные REST-запросы или прямая работа с базами данных обеспечивают достаточную производительность. В таких случаях дополнительная инфраструктура, связанная с развертыванием и поддержкой брокера сообщений, может привести к неоправданным затратам на администрирование и усложнению архитектуры без значительных выгод.

## Заключение

Событийно-ориентированная архитектура позволила существенно сократить задержки в передаче данных, упростить интеграцию сервисов и повысить надежность обработки запросов. Использование логов как основного формата данных унифицирует взаимодействие компонентов и облегчает их масштабирование. Однако такой подход может быть избыточным для небольших систем с низкой нагрузкой, где его внедрение не оправдает дополнительных затрат. В высоконагруженных же системах событийная модель обеспечивает гибкость, отказоустойчивость и возможность оперативной адаптации к изменяющимся условиям.

## Список использованных источников

1. Кредитный скоринг: что это и как работает. [Электронный ресурс]. – URL: alfabank.ru/help/articles/credit-cards/kreditnyj-skoring/#part_7 (дата обращения 27.03.2025).

2. Извлечение данных при машинном обучении. [Электронный ресурс]. – URL: habr.com/ru/companies/plarium/articles/460675/ (дата обращения 27.03.2025).

3. А может событийно-ориентированная архитектура? [Электронный ресурс]. – URL: habr.com/ru/companies/otus/articles/596107/ (дата обращения 27.03.2025).

4. Cagaty Catal. et al. RESTful API Testing Methodologies: Rationale, Challenges, and Solution Directions // 2022 Applied Sciences 12.9. – IEEE, 2022. – С. 1-16.

5. P. Dobbelaere and K. S. Esmaili. Kafka versus rabbitmq // A comparative study of two industry reference publish/subscribe implementations. Inthe 11th ACM DEBS, pages 227-238, 2017

6. ClickHouse: Передовой инструмент для оперативной обработки данных [Электронный ресурс]. – URL: habr.com/ru/companies/otus/articles/773174/ (дата обращения 29.03.2025).

7. Event-driven архитектура в Kubernetes [Электронный ресурс]. – URL: habr.com/ru/companies/otus/articles/682466/ (дата обращения 29.03.2025).