

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
Математическое обеспечение и
применение ЭВМ

Направление подготовки 09.04.02 «Информационные системы и технологии»
Магистерская программа Проектирование, разработка и эксплуатация
информационных систем

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему

**Повышение эффективности функционирования систем
поточковой обработки событий и исполнения активных
правил**

Студент	_____	Красномясов Андрей Александрович
	(подпись, дата)	(ФИО полностью)
Научный руководитель	_____	Шибанов С.В.
	(подпись, дата)	(фамилия, инициалы)
Нормоконтролёр	_____	Такташкин Д.В.
	(подпись, дата)	(фамилия, инициалы)
Рецензент	_____	Васин Л.А.
(должность, место работы)	(подпись, дата)	(фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____ № _____)

Заведующий кафедрой	_____	Козлов А.Ю.
	(подпись)	(фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____ № _____)

Секретарь ГЭК	_____	Попова Н.А.
	(подпись)	(фамилия, инициалы)

Пенза, 2023

Содержание

Введение	4
1 Проблемы обеспечения эффективности систем потоковой обработки событий с применением активных правил	8
1.1 Особенности построения и функционирования систем потоковой обработки событий с применением активных правил	8
1.1.1 Обзор современных технологий обработки событий с применением активных правил.....	14
1.1.2 Обобщенная структура системы потоковой обработки событий с применением активных правил	18
1.1.3 Функционирование систем потоковой обработки событий с применением активных правил	20
1.2 Анализ показателей для оценки эффективности систем потоковой обработки событий с применением активных правил	22
1.3 Обзор современных средств сбора и анализа диагностических данных в процессе функционирования систем потоковой обработки событий с применением активных правил	25
Выводы	26
2 Критерии и алгоритмы оценки эффективности систем потоковой обработки событий с применением активных правил	28
2.1 Модельное представление системы потоковой обработки событий с применением активных правил как системы массового обслуживания	28
2.2 Алгоритмы анализа эффективности систем потоковой обработки событий с применением активных правил	29
2.2.1 Алгоритм пороговых значений.....	30
2.2.2 Алгоритм модельного представления системы	31
2.2.3 Алгоритм поиска аномалий	32
2.3 Классификация инцидентов в системах потоковой обработки событий с применением активных правил	36

2.4	Алгоритмы реагирования на инциденты в системах потоковой обработки событий с применением активных правил	37
	Выводы	42
3	Программные средства сбора, анализа и контроля диагностических показателей для повышения эффективности сервиса потоковой обработки событий с применением активных правил	43
3.1	Архитектура сервиса потоковой обработки событий и исполнения активных правил.....	43
3.2	Обоснование выбора технологий и средств разработки программные средства сбора, анализа и контроля диагностических показателей	46
3.3	Архитектура и сценарий функционирования программных средства сбора, анализа и контроля диагностических показателей	49
3.4	Разработка программных средства сбора, анализа и контроля диагностических показателей	51
3.4.1	Создание и настройка окружения	51
3.4.2	Разворачивание сервиса потоковой обработки данных.....	52
3.4.3	Установка и настройка программных средств мониторинга Prometheus, AlertManager, Grafana.....	55
3.4.4	Установка и настройка n8n	56
3.4.5	Настройка правил выявления инцидентов	57
3.4.6	Создание автоматизированных действий.....	60
3.5	Экспериментальная оценка программных средств сбора, анализа и контроля диагностических показателей	66
	Выводы	70
	Заключение	72
	Список использованных источников	74
	Приложение А. Глосарий	77
	Приложение Б. Код конфигурации prometheus.....	80

Введение

Актуальность исследования. В современном информационном обществе, где объемы данных растут в геометрической прогрессии, системы потоковой обработки событий с применением активных правил играют важную роль в обеспечении оперативной обработки и анализа данных в режиме реального времени. Эти системы используются в различных отраслях, таких как финансы, телекоммуникации, интернет-технологии, здравоохранение и другие, где быстрая и точная обработка данных является ключевым фактором успеха.

Актуальность исследования заключается в необходимости повышения эффективности и производительности систем потоковой обработки данных с применением активных правил. При работе с большими объемами данных и высокими скоростями потоков информации, возникают сложности, связанные с обработкой и анализом данных в реальном времени, контролем состояния системы, выявлением и реагированием на инциденты.

Разработка новых методов и алгоритмов, а также применение современных инструментов и технологий позволит повысить эффективность функционирования систем потоковой обработки данных и исполнения активных правил. Такие исследования имеют практическое значение для различных отраслей, помогая организациям принимать оперативные решения, оптимизировать бизнес-процессы и повысить качество обслуживания клиентов.

Таким образом, исследования, направленные на повышение эффективности систем потоковой обработки данных с применением активных правил, имеют высокую актуальность в современной информационной среде и востребованы для обеспечения оперативной обработки и анализа данных в режиме реального времени.

Объектом исследования являются процессы мониторинга и конфигурирования систем потоковой обработки событий с применением активных правил.

Предметом исследования является алгоритмы оценки показателей эффективности и автоматического реагирования на инциденты систем потоковой обработки событий на основе активных правил с учетом инфраструктурных решений и настроек компонентов системы.

Цель и задачи работы. Целью диссертационной работы является исследование и разработка алгоритмов и автоматизированных программных средств оценки и повышения эффективности функционирования систем потоковой обработки событий с применением активных правил.

Для достижения поставленной цели планируется решить следующие задачи:

- 1) определение системы критериев эффективности систем потоковой обработки событий и исполнения активных правил;
- 2) разработка алгоритмов выявления инцидентов в процессе функционирования систем потоковой обработки событий и исполнения активных правил на основе оценки критериев эффективности;
- 3) разработка алгоритмов реагирования на инциденты, возникающие в процессе функционирования систем потоковой обработки событий и исполнения активных правил;
- 4) разработка автоматизированных программных средств сбора диагностической информации, оценки эффективности и применения алгоритмов реагирования на инциденты для повышения эффективности систем потоковой обработки событий и исполнения активных правил.

Научная новизна, оригинальность идеи в работе:

Научная новизна, оригинальность идей в работе состоит в разработке следующих решений:

- 1) система критериев для оценки эффективности системы потоковой обработки событий и исполнения активных правил;

2) алгоритмы выявления инцидентов в процессе функционирования систем потоковой обработки событий и исполнения активных правил на основе оценки критериев эффективности, позволяющие оценить текущее и будущее состояние системы;

3) алгоритмы реагирования на инциденты в процессе функционирования систем потоковой обработки событий и исполнения активных правил, обеспечивающие поддержку заданных параметров исследуемой системы.

Практическая ценность обосновывается следующими предложениями:

1) архитектура и реализация автоматизированных программных средств мониторинга и реагирования на инциденты в системах потоковой обработки событий и исполнения активных правил, обеспечивающих контроль и поддержание приемлемых показателей функционирования;

2) экспериментальная оценка разработанных программных средств на основе реального применения.

Методы проведенных исследований. Методы исследований основаны на теоретических положениях анализа данных, объектно-ориентированного моделирования, реляционной алгебры, механизма регулярных выражений и алгебры событий.

Основные положения, выносимые на защиту:

1) система критериев эффективности систем потоковой обработки событий и исполнения активных правил;

2) алгоритмы выявления инцидентов в процессе функционирования систем потоковой обработки событий и исполнения активных правил на основе оценки критериев эффективности;

3) алгоритмы реагирования на инциденты, возникающие в процессе функционирования систем потоковой обработки событий и исполнения активных правил;

4) автоматизированные программные средства сбора диагностической информации, оценки эффективности и применения алгоритмов реагирования на

инциденты для повышения эффективности систем потоковой обработки событий и исполнения активных правил.

Апробация результатов исследований. Основные положения и результаты исследований докладывались на следующих научно-технических конференциях:

- Научный семинар кафедры МОиПЭВМ (Пенза, 28 октября 2022);
- VI Всероссийская национальная научная конференция молодых учёных «Молодежи и наука: актуальные проблемы фундаментальных и прикладных исследований» (Комсомольск-на-Амуре, 14 апреля 2023);
- VI Всероссийская научно-практическая конференция с международным участием «Вызовы цифровой экономики: технологический суверенитет и экономическая безопасность» (Брянск, 19 мая 2023);
- LXIII Международная научно-практическая конференция «Вопросы технических и физико-математических наук в свете современных исследований» (Новосибирск, 24 мая 2023).

Внедрение результатов исследований. Исследования, разработка алгоритмов и программных средств выполнялись в рамках кафедральной инициативной научно-исследовательской работы «Методы и алгоритмы повышения эффективности оперативной обработки событий в системах активных правил». Результаты разработки были размещены в веб-сервис хостинга проектов Github [1].

Публикации. По теме диссертации опубликовано 4 работы в сборниках трудов конференции, индексированных в РИНЦ.

Структура и объем диссертации. Диссертация состоит из введения, 3 глав, заключения, списка использованных источников, включающего 26 источников, и 2 приложений. Материал диссертации изложен на 82 листах, включает 35 рисунков, 1 таблицу и 6 страниц приложений.

1 Проблемы обеспечения эффективности систем потоковой обработки событий с применением активных правил

1.1 Особенности построения и функционирования систем потоковой обработки событий с применением активных правил

На протяжении многих лет, данные и их обработка являются неотъемлемой частью деятельности предприятий. С течением времени, объемы собираемых и используемых данных постоянно увеличивался, что привело к разработке и внедрению инфраструктуры для эффективного управления этими данными. Для обработки данных существуют два основных подхода: традиционная обработка данных и потоковая обработка данных. Традиционная обработка данных, в свою очередь, подразделяется на транзакционную обработку данных и аналитическую обработку данных.

В настоящее время практически все компании ежедневно используют различные приложения, такие как системы планирования ресурсов предприятия (ERP), системы управления взаимодействием с клиентами (CRM), веб-приложения и другие. Такие приложения обычно имеют отдельные уровни для обработки данных (приложение само по себе) и хранения данных (система транзакционной обработки данных). Структура таких приложений показана на рисунке 1.

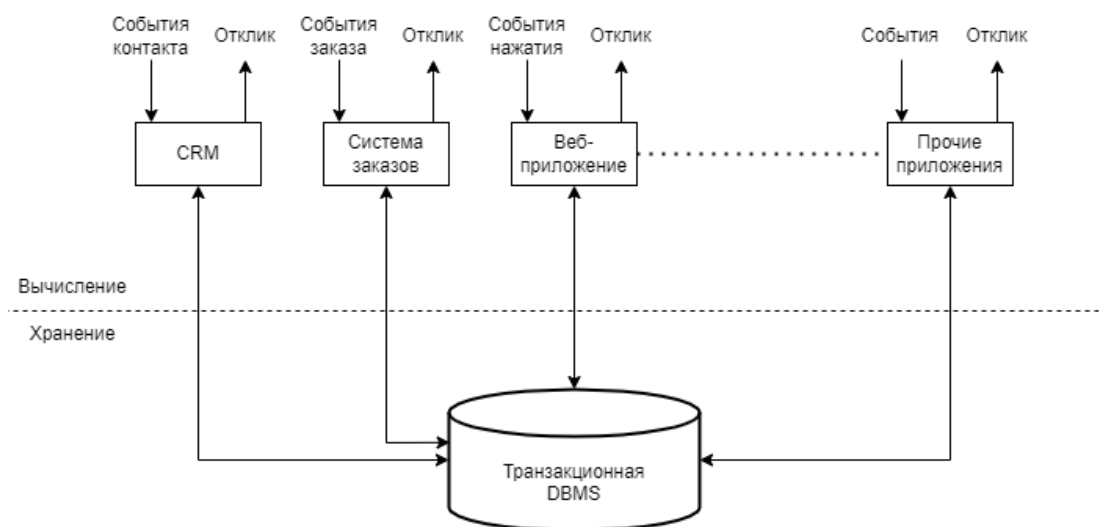


Рисунок 1 – Традиционная структура транзакционных приложений

Приложения непрерывно обрабатывают пользовательские действия, такие как клики на веб-сайте или оформление заказов в интернет-магазине. В процессе обработки приложению может потребоваться доступ к базе данных, которая обслуживает несколько приложений и содержит общую информацию. Однако такая ситуация может вызвать проблемы при разработке и масштабировании приложений, так как изменения в структуре базы данных могут повлиять на все приложения, работающие с ней.

Для анализа определенных данных системы, создания отчетов, прогнозирования и выявления причинно-следственных связей внутри системы применяется аналитическая обработка. Транзакционные данные, которые часто хранятся в разных системах баз данных, становятся более полезными, когда их можно анализировать вместе. Кроме того, для совместного анализа данных часто требуется преобразование данных в единый формат.

Вместо того, чтобы напрямую выполнять аналитические запросы к транзакционным базам данных, «сырые» данные обычно реплицируются в специализированное хранилище данных (data warehouse). Хранилище данных оптимизировано для обработки аналитических запросов. Для заполнения хранилища данных, данные из транзакционных баз данных копируются и преобразуются специальным образом. Этот процесс состоит из трех этапов: извлечение, преобразование и загрузка (extract-transform-load, ETL). Процессы ETL запускаются регулярно для синхронизации данных в хранилище.

После того как данные попадают в хранилище данных, их можно запросить и проанализировать. В хранилище данных обычно выполняются два типа запросов. Первый тип – это периодические отчеты, которые вычисляют статистику, релевантную для бизнеса. Второй тип – это специальные запросы, которые используются для получения ответов на конкретные вопросы и поддержки важных для бизнеса решений [2]. Оба типа запросов выполняются в режиме пакетной обработки (batch processing), как показано на рисунке 2.

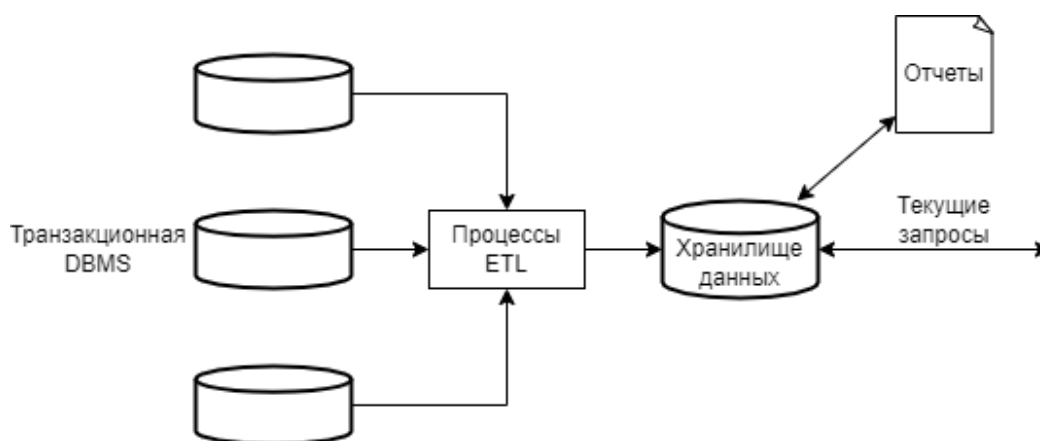


Рисунок 2 – Традиционная форма хранилища данных для транзакционной обработки

Быстрый рост технологий, а также сложность и объем данных, в частности, поставили перед исследовательским сообществом новые задачи. Существует множество источников данных, таких как потоки кликов в приложениях, беспроводные сети, наборы транзакций и другие, которые производят данные непрерывно. Эти источники называются потоками данных. Поток данных представляет собой упорядоченную последовательность экземпляров, которую можно прочитать только один или небольшое количество раз из-за ограниченных вычислительных ресурсов и возможности хранения. Эти источники данных характеризуются тем, что они открыты, передаются с высокой скоростью и генерируются нестационарными распределениями (распределение значений параметров изменяется во времени) в динамических средах.

Приложения, которые обрабатывают потоки данных, отличаются от традиционных систем управления базами данных (СУБД) по скорости поступления данных, частоте обновления, требованиям к обработке, качеству обслуживания (Quality of Service, QoS) и поддержке уведомлений. Запросы, которые обрабатываются в традиционных СУБД, называются Ad hoc (специальные) запросы. Они обычно специфицируются, оптимизируются и оцениваются один раз на основе снимка базы данных. В отличие от этого, запросы в потоковой обработке данных называются непрерывными запросами

(continuous queries, CQ). Непрерывные запросы указываются один раз и оцениваются многократно на основе новых данных в течение заданного периода жизни или до тех пор, пока данные существуют в потоке. Также предполагается, что результатом таких запросов будет поток данных, который может иметь другую скорость и схему по сравнению с исходным входным потоком [3].

Для потоковой обработки данных можно использовать традиционные СУБД, загружая входящие потоковые данные и многократно выполняя одни и те же специальные запросы. Однако это может привести к значительной задержке времени отклика из-за хранения данных, индексации и выполнения запросов, что может быть неприемлемо для многих потоковых систем. Проблема состоит в том, что данные должны быть сохранены на вторичных устройствах с высокой задержкой, прежде чем СУБД сможет получить к ним доступ и обработать их в основной памяти с низкой задержкой. Кроме того, такой подход имеет следующие проблемы:

- метод моментальных снимков не всегда подходит для оценки потоковых данных;
- невозможно задать требования к качеству обслуживания для СУБД, что делает ее менее пригодной для потоковых приложений;
- обычная оптимизация запросов, направленная на минимизацию обращений к диску, может быть неприменимой для потоковой обработки данных.

Таким образом, методы, разработанные для традиционных СУБД, должны быть пересмотрены, чтобы соответствовать требованиям приложений, использующих потоковые данные. Для удовлетворения потребностей потоковых приложений были разработаны системы управления потоками данных (Data Stream Management System – DSMS). Схема парадигмы систем управления базами данных представлена на рисунке 3.

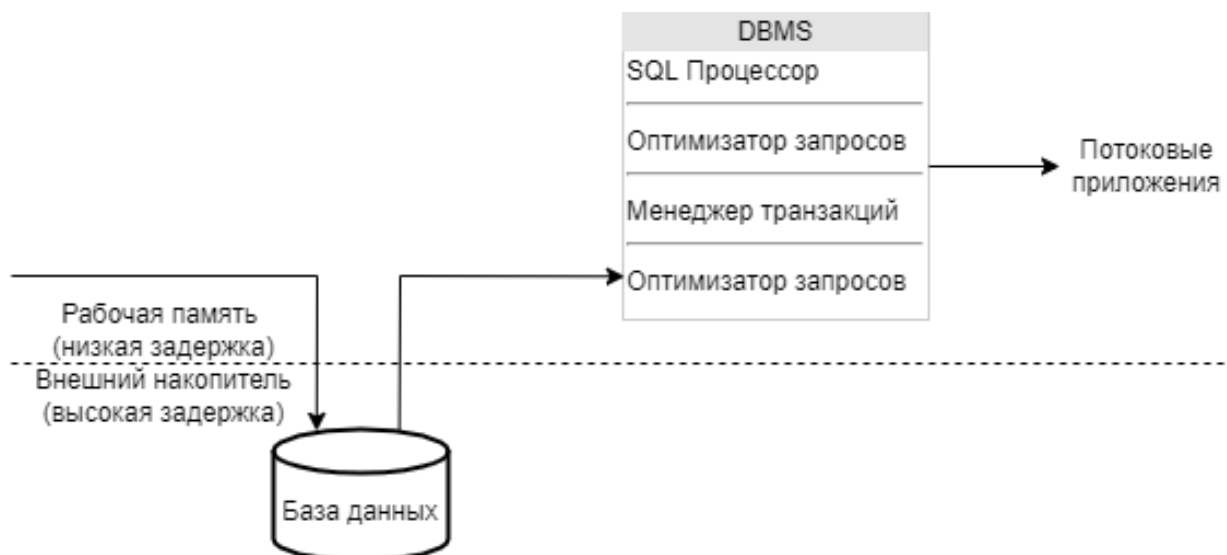


Рисунок 3 – Обработка потока данных с помощью СУБД

Потоковые данные сначала сохраняются в базе данных, и запросы обрабатываются по сохраненным данным. Этот подход требует двух операций ввода-вывода для каждого обрабатываемого элемента данных. Из-за высокой скорости поступления данных и задержки при записи и извлечении, обработка элементов данных может отставать, что приводит к увеличению времени отклика на запросы. В отличие от этого, на рисунке 4 показана архитектура, в которой входящие потоки обрабатываются непосредственно с помощью DSMS.

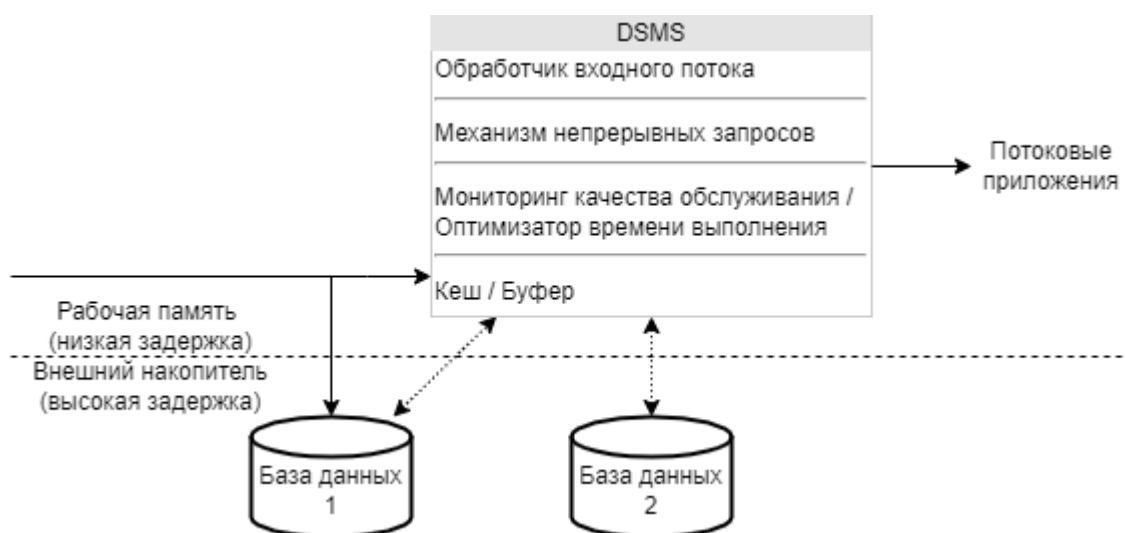


Рисунок 4 – Обработка данных с помощью DSMS

В данном случае операции ввода-вывода не выполняются для каждого элемента данных, что приводит к сокращению времени отклика на запросы.

Данные могут быть сохранены для архивирования или других целей, но это не влияет на вычисления и производительность непрерывных запросов [4].

Также потоковая обработка данных обладает следующими свойствами.

Потоковый слой устойчив к отказам. Он работает как кластер на всех доступных узлах. Если один узел прекращает работу, то другой продолжит его работу с того же самого места. И наоборот, можно масштабировать кластер путем добавления новых обрабатывающих узлов. Задачи и все необходимые состояния будут автоматически перенаправлены на использование этих ресурсов.

Каждый узел потокового обработчика может содержать собственное состояние. Это необходимо для буферизации или хранения целых таблиц, например, чтобы производить улучшение данных. Идея локального хранилища важна, поскольку она даёт обработчику потока возможность производить быстрые запросы, по входящему сообщению, не опрашивая соседние узлы – это необходимая функция для высокоскоростных операций и систем. Но эта способность сохранять состояния в локальных хранилищах также оказывается полезна для целого ряда бизнес-задач.

Каждый потоковый обработчик может записывать и сохранять локальное состояние. Опрашивать соседние узлы для обработки каждого входящего сообщения может быть не слишком хорошей идеей при работе с потоковыми системами большой пропускной способности. Именно поэтому потоковые обработчики записывают данные локально (для быстрой записи и чтения) и резервируют эти записи. Эта возможность локального хранения данных концептуально очень похожа на взаимодействие с базой данных в традиционных приложениях. Но в отличие от традиционного двухуровневого приложения, где взаимодействие с базой данных подразумевает сетевой запрос, в обработке потоков всё состояние локально (в виде кэша), поэтому доступ к нему происходит быстро и сетевые запросы не требуются [5].

1.1.1 Обзор современных технологий обработки событий с применением активных правил

За все время развития области обработки событий, возникло четыре технологических направления, основанных на обработке различных типов событий и разработанных для разных целей.

Самым ранним подходом к обработке событий является дискретное моделирование событий. Дискретное моделирование событий – это методология моделирования и анализа систем, в которых происходят дискретные события. Она используется для описания поведения системы во времени и позволяет изучать взаимодействия и последствия событий на различных уровнях. События в данном подходе используются для управления ходом вычислений, моделирующих систему, и для продвижения вычислений вперед на дискретный шаг (рисунки 5, 6).

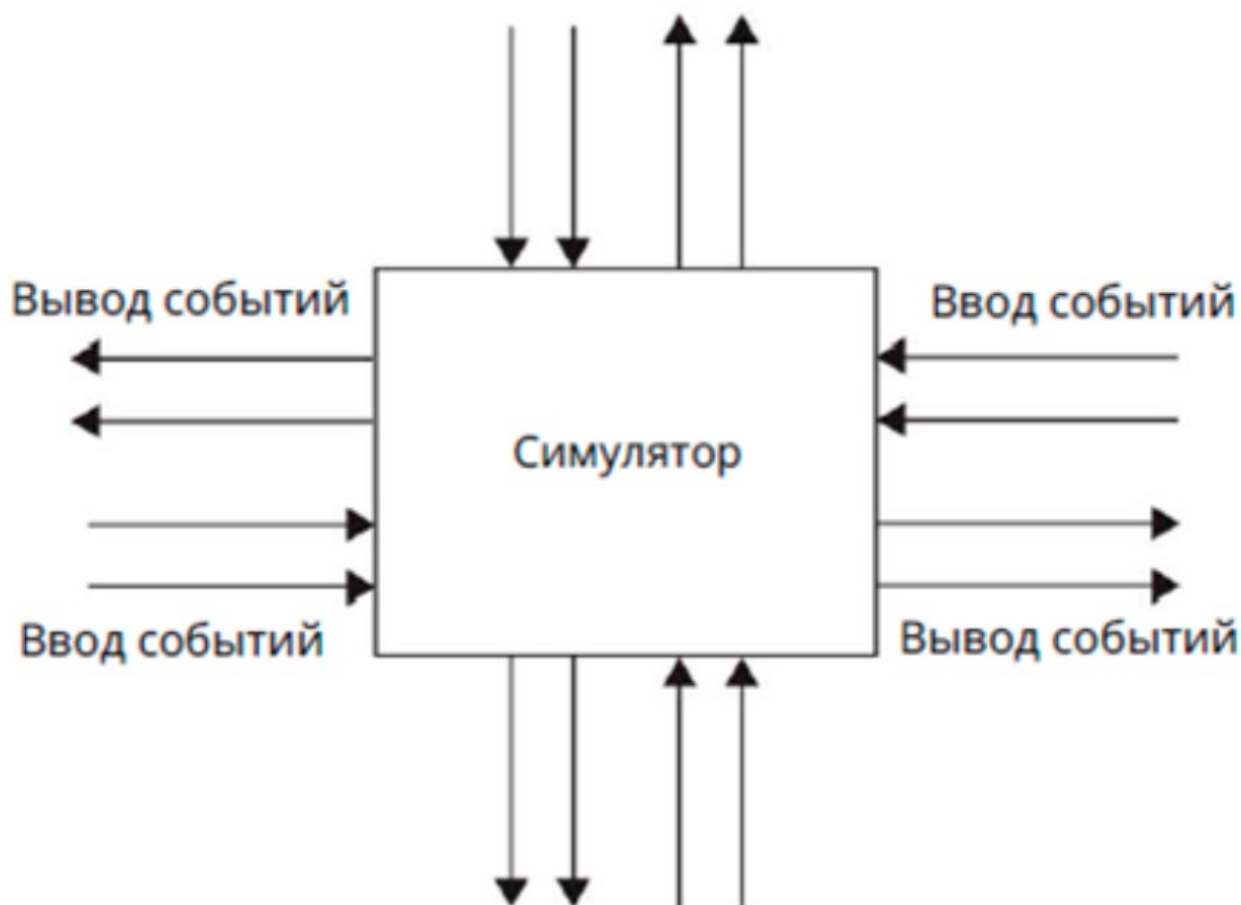


Рисунок 5 – Ячейка моделирования

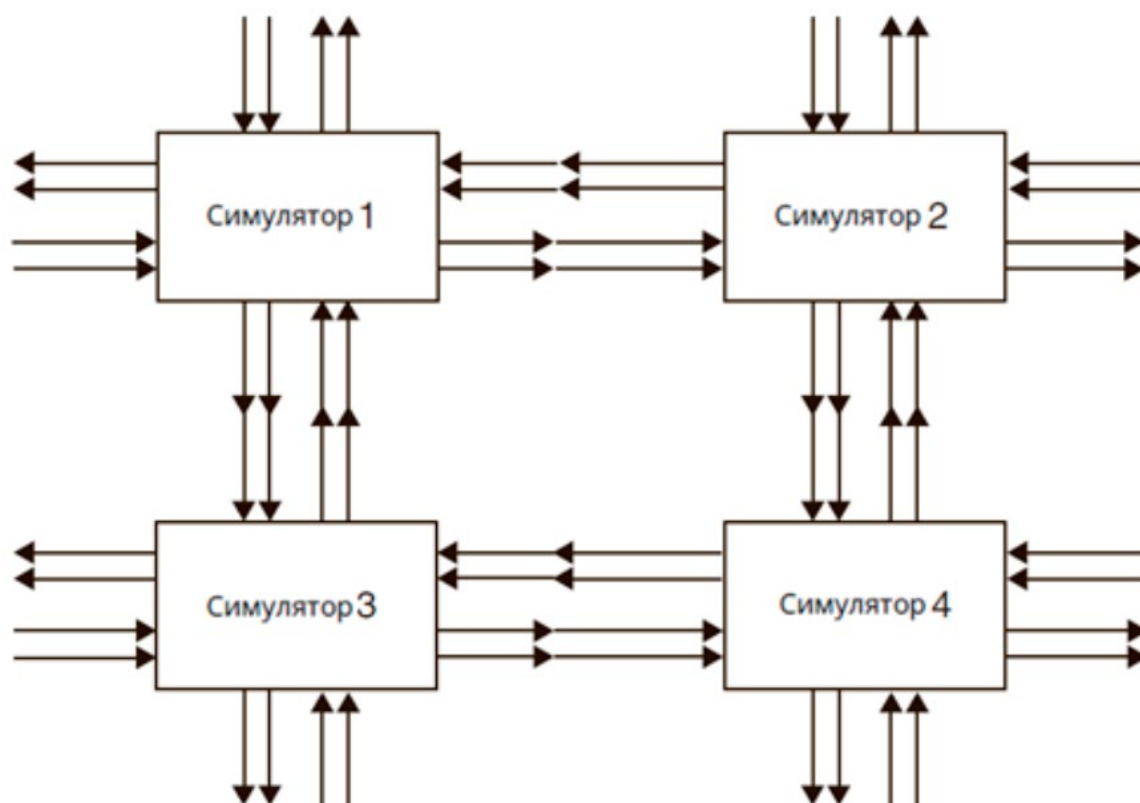


Рисунок 6 – Сетка моделирования

Каждый шаг включает запуск модели на данных, представляющих текущее состояние системы – дискретный шаг. События, содержащие данные, передаются на последующий шаг. В результате модель запускается на этих данных и создает новый набор выходных событий, содержащих данные, определяющие новое состояние. Эти события являются входными данными для следующего шага. Таким образом моделирование, управляемое событиями, создает пошаговую последовательность наборов событий, которые представляют собой состояния моделируемой системы [6].

Дискретное моделирование событий находит применение в различных областях, включая производство, логистику, телекоммуникации, транспорт, финансы и другие. Оно помогает понять динамику системы, прогнозировать ее поведение и принимать обоснованные решения для оптимизации работы системы.

Примеры инструментов и языков дискретного моделирования событий включают SDL (Specification and Description Language), Event-B, Petri Nets,

SimEvents (часть среды моделирования MATLAB/Simulink), AnyLogic, Arena и другие. Эти инструменты предоставляют средства для создания моделей, имитации событий и анализа результатов моделирования [7].

Следующим разработанным подходом является применение активных баз данных. Традиционные базы данных пассивны в том смысле, что они ждут запросов и отвечают на них. Активные базы данных способны вызывать другие приложения в ответ на запрос. Например, активная база данных может инициировать действие с системой управления запасами для повторного заказа товара, если база данных обнаружит, что уровень запасов этого товара упал ниже заданного уровня.

Технология активных баз данных позволяет реализовать реактивность и автоматическую обработку событий в реальном времени в рамках базы данных. Это особенно полезно для систем, требующих оперативного мониторинга и реагирования на изменения данных, таких как системы управления транзакциями, системы мониторинга и системы управления сетевым оборудованием.

Технология обработки событий с использованием корпоративной служебной шины (Enterprise Service Bus, ESB) представляет собой архитектурный подход, который обеспечивает связь и взаимодействие различных приложений и сервисов внутри корпоративной сети. Она основана на концепции шины, которая предоставляет среду для передачи и обмена сообщениями между различными компонентами системы.

Технология ESB позволяет реализовать гибкую и распределенную обработку событий в корпоративной сети. Она упрощает интеграцию различных систем, обеспечивает надежность и гибкость в обработке сообщений, а также позволяет управлять потоком данных и обеспечивать согласованность информации. Это позволяет компаниям эффективно реагировать на события, связанные с бизнес-процессами и операциями, и улучшать общую производительность и эффективность корпоративных систем [8].

Технология обработки событий на основе сервис-ориентированной структуры (Service-Oriented Architecture, SOA) представляет собой подход к разработке и интеграции систем, основанный на представлении приложений в виде сервисов, которые могут взаимодействовать друг с другом через обмен сообщениями.

В сервис-ориентированной структуре обработка событий происходит путем определения и развертывания сервисов, которые представляют конкретные функциональные возможности системы. Эти сервисы могут выполнять различные операции и предоставлять функциональность, которая может быть использована другими компонентами системы.

Технология SOA обеспечивает гибкость и масштабируемость обработки событий, позволяет компаниям создавать модульные и переиспользуемые компоненты, улучшать интеграцию и взаимодействие между различными системами, а также обеспечивать более эффективное использование ресурсов и управление бизнес-процессами.

Архитектура, управляемая событиями (Event-Driven Architecture, EDA) – это подход к разработке и построению систем, где события играют центральную роль в обмене информацией и управлении процессами. Она позволяет создавать гибкие, масштабируемые и отзывчивые системы, которые легко адаптируются к изменяющимся условиям.

В архитектуре, управляемой событиями, события являются основными источниками информации, которые происходят в системе или внешних источниках. Источники событий отслеживают изменения и генерируют события при возникновении определенных условий. События передаются через шину событий, которая служит централизованным механизмом для передачи событий между компонентами системы.

Компоненты системы подписываются на определенные типы событий и реагируют на них с помощью обработчиков событий. Обработчики событий выполняют различные операции или действия в ответ на поступающие события. Они могут обновлять данные, запускать процессы, отправлять

уведомления и выполнять другие действия в зависимости от бизнес-логики системы.

В архитектуре EDA события также могут быть агрегированы и анализированы для получения ценной информации и предоставления решений. Например, события могут быть агрегированы в сложные события, которые представляют собой комбинацию нескольких элементарных событий. Анализ и обработка таких сложных событий позволяют выявлять тенденции, паттерны и аномалии в системе.

Технологии, поддерживающие архитектуру, управляемую событиями, включают различные системы управления событиями, шины сообщений и фреймворки для обработки событий. Эти технологии обеспечивают надежную и эффективную передачу событий, управление подписками на события и обработку событий в реальном времени [9].

Архитектура, управляемая событиями, находит широкое применение во многих областях, таких как финансы, телекоммуникации, интернет вещей (IoT), системы мониторинга и управления, электронная коммерция и другие. Она позволяет создавать гибкие и реактивные системы, способные адаптироваться к изменяющимся требованиям и обеспечивать эффективное взаимодействие между компонентами.

1.1.2 Обобщенная структура системы потоковой обработки событий с применением активных правил

Цель системы потоковой обработки событий с применением активных правил заключается в эффективной обработке и анализе непрерывного потока событий для выявления значимых паттернов (сложных событий), выработки решений и управления объектами на основе определенных правил [10].

Обобщенная структура системы потоковой обработки событий с применением активных правил состоит из следующих компонентов: объекты мониторинга, источник, обработчик входных событий, обработчик сложных

событий, планировщик объектов управления, отправитель, объекты управления, правила и команды (рисунок 7).

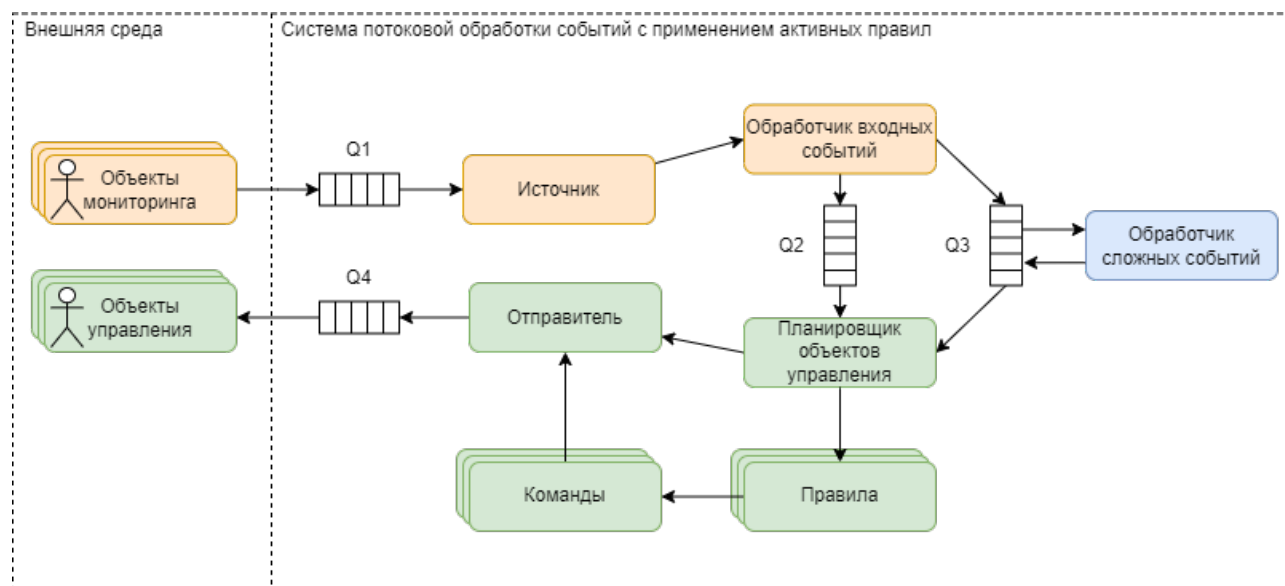


Рисунок 7 – Общая структура системы потоковой обработки событий с применением активных правил

Объекты мониторинга генерируют события, которые попадают в очередь Q1. Эта очередь служит буфером, который хранит входные события, ожидающие обработки. Очередь Q1 позволяет изолировать и стабилизировать поток событий, обеспечивая более предсказуемую обработку и распределение нагрузки на систему.

Далее, события из очереди Q1 направляются в обработчик входных событий. Здесь события проходят предварительную обработку и разделяются на две очереди: очередь простых и агрегированных событий Q2 и очередь сложных событий Q3. Разделение на очереди позволяет эффективно управлять разными типами событий и обеспечивает более гибкую обработку в зависимости от их характеристик и требований.

Очередь сложных событий Q3 является ключевым элементом структуры системы. События, находящиеся в этой очереди, взаимодействуют с обработчиком сложных событий. Обработчик анализирует элементарные события из очереди Q3 и определяет сложные события.

После этапа обработки и анализа, события из очередей Q2 и Q3 передаются в планировщик объектов управления. Планировщик основывается на заданных активных правилах и использует содержащуюся информацию для формирования команд, которые определяют необходимые действия для управляемых объектов. Команды отправляются в очередь команд (Q4), где ожидают своей обработки и выполнения.

Такая структура позволяет системе обрабатывать входные события в режиме реального времени, анализировать данные, применять активные правила для выработки решений и управления объектами. В результате достигается более эффективное и точное реагирование на события, а также возможность принятия действий на основе заранее определенных правил.

1.1.3 Функционирование систем потоковой обработки событий с применением активных правил

Описание функционирования систем потоковой обработки событий с применением активных правил представлена в виде диаграммы деятельности.

Диаграмма деятельности – это графическое представление процессов и задач, выполняемых в рамках определенного проекта или деятельности. Эта диаграмма задает последовательность выполнения действий в рамках определенного процесса или проекта. Она позволяет визуализировать последовательность задач и операций, которые нужно выполнить, чтобы достичь желаемого результата. Схемы деятельности могут быть использованы для моделирования бизнес-требований, создания высокоуровневого представления о функциональных возможностях системы, анализа сценариев использования и для различных других целей.

Все эти характеристики делают диаграмму деятельности хорошим выбором для анализа функционирования систем. Диаграмма, построенная для системы потоковой обработки событий с применением активных правил представлена на рисунке 8.



Рисунок 8 – Функционирование системы потоковой обработки событий с применением активных правил

События генерируются источником элементарных событий и отправляются на обработку входных событий. Далее происходит фильтрация поступающих элементарных событий, после чего возможно 2 ситуации: элементарное событие является частью сложного или элементарное событие активных правил. Если событие является частью сложного, то из него формируется сложное событие. Далее элементарные события активных правил и сложные события, сформированные на прошлом этапе, передаются для выполнения действий активных правил над объектами взаимодействия активных правил. В завершении, обработанные события журналируются и процесс завершается.

1.2 Анализ показателей для оценки эффективности систем потоковой обработки событий с применением активных правил

Показателем эффективности вычислительных систем обычно выступает их производительность — количество эталонных задач (алгоритмов), выполняемых в единицу времени. Оценка производительности систем потоковой обработки должна учитывать случайный характер поступления задач и время их обработки.

На основе проведенных исследований были определены и классифицированы следующие метрики эффективности систем. Эти метрики позволяют оценить сильные и слабые стороны работы рассматриваемых систем в режиме реального времени.

Метрики памяти. Данный класс метрик отражает количество памяти, потребляемое системой при функционировании. Для оценки можно использовать процент доступной кучи, вычисляемой путем деления общего объема свободной памяти кучи на общий объем доступной памяти кучи в мегабайтах:

$$\text{Доступная куча (\%, Мб)} = \frac{\text{Свободная куча}}{\text{Общий размер кучи}} \times 100.$$

Также, важным параметром является время сборки мусора *GCTime*:

$$GCTime = \alpha + \beta + \gamma,$$

где α — общее накопленное время, затраченное на определение количества достижимых объектов;

β — количество недостижимых объектов;

γ — время освобождения памяти в миллисекундном кадре окна.

Время сборки мусора отражает общее накопленное время в миллисекундах, затрачиваемое управляемым компонентом сборщика мусора на поиск и восстановление недоступных объектов для освобождения места в памяти [11].

N-Вертикальные метрики. В системах потоковой обработки данных, вертикальные метрики связаны с отслеживанием характеристик отдельных компонентов системы. Они предоставляют информацию о производительности конкретных узлов, приложений или запросов в системе. Примеры таких метрик включают загрузку ЦПУ, количество ошибок, количество запросов и т. д.

Коммуникационные показатели. Обратное давление (Back pressure) – это механизм, используемый в системах потоковой обработки данных для управления производительностью и предотвращения переполнения буфера в случае, когда производительность обработки данных не соответствует их объему. Когда производительность системы недостаточна для обработки всего потока данных, буфер заполняется, и система отправляет обратные сигналы потребителям данных, чтобы они замедляли скорость своих запросов. Таким образом, производительность потребления данных снижается до тех пор, пока система не сможет вернуться в нормальный режим работы.

Для расчета обратного давления в системах потоковой обработки данных используются различные методы, такие как оценка загрузки буфера или сравнение пропускной способности потока данных с производительностью системы.

Показатели вычислений. Основными показателями вычислений являются пропускная способность и задержка.

Пропускная способность измеряет вычислительную мощность системы и указывает, сколько событий система способна обработать за единицу времени. Задержка, или временной интервал, показывает время, необходимое для обработки события, то есть период между получением события и появлением его результатов в выходных данных [12].

Внешние показатели. Важным аспектом характера входных потоков является их объем и частота. Чем больше данных поступает в систему, тем больше ресурсов требуется для их обработки, и тем выше требования к производительности системы. Кроме того, если данные поступают в систему

слишком быстро, то система может столкнуться с проблемами сбоев и неравномерной обработки данных [13].

Специализированные метрики. Данный класс описывает метрики, характерные для отдельных, более узконаправленных классов систем. К ним относятся метрики, оценивающие качество интеллектуального анализа данных.

Качество интеллектуального анализа данных измеряет эффективность алгоритмов и методов анализа данных в обнаружении скрытых закономерностей, моделей и трендов в больших объемах данных. Эти метрики могут включать точность, полноту, F1-меру, ROC-кривую и площадь под ней (AUC), а также другие метрики, связанные с конкретной задачей.

Выбор критериев эффективности для конкретной системы является важным шагом, который должен быть основан на требованиях приложения и бизнес-целях. Различные системы имеют разные потребности и цели, поэтому необходимо определить наиболее значимые критерии для каждой из них. Например, система, используемая для обработки финансовых транзакций, должна обладать высокой точностью обработки событий, чтобы минимизировать возможность ошибок и потерь данных. Надежность такой системы является также ключевым фактором, поскольку любые сбои или сбои в работе могут привести к финансовым потерям или недовольству клиентов. Кроме того, система должна быть масштабируемой, чтобы эффективно обрабатывать большое количество транзакций и соответствовать растущим потребностям бизнеса.

При выборе критериев эффективности также необходимо учитывать возможности аппаратного обеспечения и требования к вычислительным ресурсам. Например, ограничения по памяти, процессорной мощности или доступной сетевой пропускной способности могут оказывать влияние на выбор критериев и определение оптимальной конфигурации системы.

1.3 Обзор современных средств сбора и анализа диагностических данных в процессе функционирования систем потоковой обработки событий с применением активных правил

Инструментальные средства мониторинга играют ключевую роль в обеспечении бесперебойной работы IT-инфраструктуры любого предприятия. Особое внимание уделяется мониторингу систем потоковой обработки событий с применением активных правил, поскольку эти системы обрабатывают огромное количество событий и имеют высокую степень значимости.

Существует множество инструментальных средств мониторинга, таких как Zabbix, Icinga и Prometheus, которые позволяют мониторить различные параметры IT-инфраструктуры и применять различные методы мониторинга.

Zabbix – это масштабируемый инструмент мониторинга, разработанный для отслеживания разнообразных параметров сети и состояния служб, которые влияют на производительность сети. Он обладает функциями мониторинга, визуализации, уведомлений, автоматического обнаружения и предсказания на основе данных из разных источников. С его помощью можно контролировать множество сетевых параметров, включая производительность серверов, использование сети и целостность данных.

Однако, Zabbix может быть сложным для настройки и управления, особенно для новых пользователей. Несмотря на большой набор функций, этот инструмент может требовать значительных усилий для настройки, поддержки и интеграции с другими системами [14].

Icinga – это система мониторинга с открытым исходным кодом, которая позволяет отслеживать все аспекты исследуемой инфраструктуры. Она предоставляет широкий набор возможностей, включая проверку состояния, уведомления, обработку инцидентов, создание дашбордов и отчетов.

Icinga обладает гибкой архитектурой и поддерживает большое количество плагинов, что позволяет пользователям настроить систему под свои конкретные

потребности. Она также поддерживает мониторинг в реальном времени, что позволяет оперативно реагировать на любые проблемы или отклонения.

Однако, как и Zabbix, настройка и управление Icinga могут быть сложными. Гибкость является одним из его основных преимуществ, но она также может добавить дополнительную сложность, особенно для новых пользователей или команд с ограниченными ресурсами [15].

Prometheus – это относительно новый инструмент мониторинга и оповещения с открытым исходным кодом, разработанный специально для мониторинга микросервисов и контейнеров.

Prometheus имеет мощные возможности для сбора и анализа метрик в реальном времени, включая поддержку множества типов метрик и гибкую систему запросов. Он использует свой собственный язык запросов PromQL, что позволяет пользователям создавать сложные запросы для анализа данных.

Однако, для функционирования уведомлений в Prometheus, необходимо использовать дополнительные инструменты, такие как Alertmanager. Кроме того, Prometheus может являться не лучшим выбором для обработки очень больших объемов данных из-за своей модели хранения, которая оптимизирована для быстрого чтения, а не для долгосрочного хранения [16].

Все три инструмента предлагают мощные возможности для мониторинга системы потоковой обработки событий и выполнения активных правил. Однако они отличаются по своей функциональности, гибкости, сложности настройки и управления, а также способности обрабатывать большие объемы данных.

Выводы

В данной главе были рассмотрены основные аспекты, связанные с построением и функционированием систем потоковой обработки событий с применением активных правил, а также анализ показателей для оценки их эффективности. Также проведен обзор современных средств сбора и анализа диагностических данных в процессе функционирования таких систем.

Был представлен обзор современных технологий обработки событий с применением активных правил. Было отмечено, что такие системы находят широкое применение в различных областях, таких как финансовые транзакции, мониторинг сетей, анализ социальных сетей и т.д. Описана обобщенная структура системы потоковой обработки событий с применением активных правил, которая включает компоненты для сбора событий, обработки правил и принятия решений. Рассмотрены основные особенности функционирования таких систем.

Проведен анализ основных показателей, которые могут быть использованы для оценки эффективности таких систем. В ходе анализа было выяснено, что выбор показателей должен основываться на требованиях приложения и бизнес-целях. В качестве примера были рассмотрены критерии, такие как точность обработки событий, надежность, пропускная способность, скорость обработки и масштабируемость. Было подчеркнуто, что важно учитывать также возможности аппаратного обеспечения и требования к вычислительным ресурсам при выборе показателей эффективности.

Также, был проведен обзор современных средств, которые могут быть использованы для сбора и анализа диагностических данных в процессе функционирования таких систем. Было отмечено, что эффективный мониторинг и анализ диагностических данных позволяют обнаружить проблемы и улучшить производительность системы.

Полученные в данной главе результаты позволяют лучше понять особенности построения и функционирования систем потоковой обработки событий с применением активных правил, а также предоставляет базу для оценки и улучшения их эффективности с помощью анализа соответствующих показателей и использования современных средств сбора и анализа диагностических данных. Также результаты главы позволили определить направление исследований и разработок, представленных в последующих главах диссертации.

2 Критерии и алгоритмы оценки эффективности систем потоковой обработки событий с применением активных правил

2.1 Модельное представление системы потоковой обработки событий с применением активных правил как системы массового обслуживания

Рассматриваемая система потоковой обработки событий с применением активных правил может быть представлена в виде системы массового обслуживания. В данной системе предполагается использование пуассоновского потока событий, где события происходят случайным образом и не зависят от предыдущих событий.

В системе присутствуют элементы, которые играют роль каналов обслуживания. Они могут быть представлены обработчиками событий. События поступают в систему с некоторой интенсивностью, соответствующей пуассоновскому потоку. Это означает, что между поступлениями событий существует случайное время, которое распределено по экспоненциальному закону.

Каждое поступившее событие проходит через обработку в системе. В марковском процессе предполагается, что обработка события зависит только от его текущего состояния и не зависит от предыдущих событий или состояний системы. Это обеспечивает свойство отсутствия памяти у марковского процесса [17].

Система массового обслуживания характеризуется следующими основными характеристиками:

- 1) интенсивность потока поступления событий (λ) – это параметр, определяющий среднее количество событий, поступающих в систему за единицу времени;
- 2) интенсивность обслуживания (μ) – это параметр, определяющий среднее количество событий, которые система может обработать за единицу времени;

- 3) абсолютная пропускная способность (A) – среднее число заявок, которое может обслужить система за единицу времени;
- 4) относительная пропускная способность (Q) – средняя доля поступивших заявок, обслуживаемая системой, т. е. отношение среднего обслуживаемых за единицу времени заявок к среднему числу заявок, поступивших за это время в систему;
- 5) вероятность отказа ($P_{\text{отк}}$) – это вероятность того, что заявка покинет СМО необслуженной;
- 6) среднее число заявок в системе ($L_{\text{сист}}$);
- 7) среднее число заявок в очереди ($L_{\text{оч}}$);
- 8) среднее время пребывания заявок в системе ($T_{\text{сист}}$);
- 9) среднее время пребывания заявок в очереди ($T_{\text{оч}}$);
- 10) вероятность того, что канал занят ($P_{\text{зан}}$).

Таким образом, рассматриваемая система потоковой обработки событий с применением активных правил может быть описана и рассмотрена как система массового обслуживания, использующая пуассоновский поток событий и работающая в рамках марковского процесса. Следовательно, для анализа эффективности можно использовать характеристики, характерные для систем массового обслуживания [18].

2.2 Алгоритмы анализа эффективности систем потоковой обработки событий с применением активных правил

Алгоритмы поиска инцидентов в системах потоковой обработки данных играют важную роль в обнаружении аномалий и неполадок, позволяя оперативно реагировать на возникающие проблемы. Выделяют следующие алгоритмы, используемые для поиска инцидентов:

- 1) пороговые значения;
- 2) модельное представление системы;
- 3) поиск аномалий.

Каждый из этих методов обладает своими преимуществами и применяется в зависимости от требований системы.

2.2.1 Алгоритм пороговых значений

Алгоритм поиска инцидентов с использованием пороговых значений является простым, но эффективным методом, который основывается на определении предельных значений для определенных параметров или метрик системы. Если значение параметра превышает или падает ниже установленного порога, считается, что произошел инцидент, и необходимы дополнительные действия или реакция.

Плюсы использования алгоритма пороговых значений:

- 1) простота и прозрачность: этот подход легко понять и реализовать;
- 2) гибкость и настраиваемость: пороговые значения могут быть легко настроены под конкретные требования и характеристики системы;
- 3) быстрота реакции: при превышении пороговых значений система может моментально реагировать и генерировать оповещения или автоматические действия;
- 4) применимость к простым сценариям: алгоритм поиска инцидентов с использованием пороговых значений хорошо подходит для обнаружения простых аномалий и инцидентов, где предельные значения являются надежными индикаторами проблем.

Несмотря на свои преимущества, данный подход имеет некоторые ограничения и минусы:

- 1) ограниченность точностью: алгоритм на основе пороговых значений может быть ограничен в точности обнаружения инцидентов;
- 2) чувствительность к настройкам: правильная настройка пороговых значений является ключевым фактором для эффективного функционирования алгоритма;

3) сложность адаптации к изменениям: если окружение или характеристики системы изменяются со временем, необходимо перенастраивать пороговые значения.

Алгоритм поиска инцидентов с использованием пороговых значений предпочтителен в следующих случаях:

- когда требуется быстрый и простой способ обнаружения инцидентов в системе;
- когда имеются четкие и измеримые критерии для определения аномалий и установления пороговых значений;
- когда система имеет стабильные и предсказуемые паттерны работы, где пороговые значения могут быть эффективными индикаторами проблем;
- в случаях, когда другие более сложные алгоритмы и методы не являются необходимыми или применимыми для решения конкретных задач.

Важно отметить, что алгоритм поиска инцидентов с использованием пороговых значений является основным методом и может быть дополнен другими алгоритмами и методами анализа данных для более точного и комплексного обнаружения инцидентов [19].

2.2.2 Алгоритм модельного представления системы

В этом подходе используется модель системы, которая описывает ее нормальное функционирование. Эта модель строится на основе знаний о системе и ее поведении. Путем сравнения текущего состояния системы с модельным представлением можно выявить отклонения и потенциальные инциденты. Например, можно построить модель, которая описывает типичные временные задержки в системе. Если задержки значительно отклоняются от ожидаемых значений, это может указывать на проблему или инцидент.

Плюсы использования алгоритма модельного представления системы:

- 1) способность обнаруживать сложные и неочевидные аномалии, которые могут быть упущены другими методами;

- 2) гибкость и настраиваемость модели в соответствии с уникальными характеристиками системы;
- 3) возможность предоставления дополнительных деталей о характере и контексте инцидента.

Минусы алгоритма поиска инцидентов на основе модельного представления системы:

- 1) сложность построения и поддержки модели системы;
- 2) возможность упустить некоторые аспекты системы на этапе проектирования модели, из-за чего её характеристики могут быть не точными;
- 3) необходимость иметь достаточное количество исторических данных для построения корректной модели;
- 4) возможность ложных срабатываний, особенно в сложных и динамических системах.

Алгоритм поиска инцидентов на основе модельного представления системы предпочтителен в случаях, когда необходимо обнаружить сложные или нестандартные аномалии, имеются достаточные исторические данные для обучения модели, требуется более детальная информация о характере и контексте инцидента, а также когда другие методы поиска инцидентов неэффективны или неприменимы для конкретной системы, но он является достаточно сложным в плане разработки и последующей поддержки.

2.2.3 Алгоритм поиска аномалий

Алгоритм поиска инцидентов на основе поиска аномалий является методом обнаружения отклонений от нормального поведения в системе. Этот алгоритм основан на предположении, что аномалии представляют собой необычные и редкие события, которые отличаются от типичных паттернов.

Можно использовать различные методы статистического анализа и машинного обучения, чтобы обнаружить необычные или непредсказуемые паттерны в данных. Например, можно анализировать изменения в потоке

событий или распределение нагрузки на различные компоненты системы. Обнаружение аномалий может указывать на возникновение инцидентов.

Поиск аномалий подразделяется на следующие подкатегории:

- 1) метод скользящего окна;
- 2) кластеризация;
- 3) нейронные сети;
- 4) статистические методы.

Метод скользящего окна является эффективным подходом для поиска аномалий во временных рядах. В этом методе временной ряд разбивается на несколько подпоследовательности (окон), и аномалии ищутся путем скольжения окна по всему ряду с заданным шагом. Однако у этого метода есть недостатки, такие как необходимость тщательного выбора размера окна, чтобы захватить аномалию, и увеличение вычислительной сложности с увеличением числа окон и длины временного ряда.

Кластеризация является еще одним универсальным методом поиска аномалий. Этот метод основан на группировке похожих значений в кластеры, и не требует знаний о свойствах возможных отклонений. Выявление аномалий строится на следующих предположениях:

- нормальные экземпляры данных относятся к какому-то кластеру;
- нормальные данные ближе к центру кластера, аномальные – дальше;
- нормальные данные образуют большие плотные кластеры, а аномальные – маленькие и разрозненные.

Однако она имеет свои ограничения, такие как способность характеризовать аномалии в целом, но не давать точных границ аномальных подпоследовательностей. Кроме того, незначительные аномалии могут быть игнорированы данными методами.

Использование нейронных сетей для поиска аномалий – это метод, который позволяет автоматически обнаруживать аномальные паттерны или поведение в данных. Нейронные сети могут быть обучены на нормальном

поведении и использоваться для определения отклонений от этого нормального шаблона. Для поиска аномалий с помощью нейронных сетей, данные обычно подаются на вход нейронной сети, которая обучается на эталонных данных. Обучение может проводиться различными методами, включая обучение с учителем или без учителя. После обучения, нейронная сеть может применяться для анализа новых данных и определения, являются ли они аномальными или соответствуют нормальному поведению. Это может быть полезно для обнаружения необычных событий, ошибок или атак на систему [20].

Статистические методы широко используются для проверки гипотез о сходстве или различии двух выборок. Корреляционный анализ оценивает статистическую связь между идеальным и реальным временными рядами, и изменение этой связи может указывать на аномальность. Эти методы успешно применяются в решении задач поиска аномалий в физических процессах, где доступна история наблюдений.

К статистическим методам, направленным на поиск аномалий, относят:

- правило трех сигм;
- z-оценка;
- межквартильный размах.

Практически все статистические методы используют вычисление среднего $\bar{x}$, стандартного отклонения S и медианы M – среднее значение, получаемое при упорядочивании всех чисел по возрастанию и взятии того, что находится в середине. Расчет вычисления среднего и стандартного отклонения производится по формулам:

$$\bar{x} = (x_1 + \dots + x_n) / n,$$

$$S = \sqrt{\frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})^2},$$

где x – значение временного ряда,

n – количество элементов в выборке.

Правило трех сигм заключается в том, что при нормальном распределении практически все значения величины с вероятностью 0,9973 лежат не далее трех сигм в любую сторону от математического ожидания.

Значение является аномальным (выбросом) при выполнении условий уравнения:

$$|x_i - \bar{x}| > 3 \times S.$$

Z-оценка – это мера относительного разброса наблюдаемого или измеренного значения, которая показывает, сколько стандартных отклонений составляет его разброс относительно среднего значения. Данная величина рассчитывается по формуле:

$$Z = |x_i - \bar{x}|/S.$$

Если значение Z превышает задаваемый порог (чаще всего равный 3), то значение считается аномальным.

Межквартильный размах – это разница между третьим и первым квартилем – значением, которое больше 75% значений в выборке, и значением, которое больше 25% значений в выборке. Наблюдение можно считать аномальным, если оно в полтора раза превышает межквартильный размах, превышающий третий квартиль, или в полтора раза превышает межквартильный размах, меньше первого квартиля.

Скрытые модели Марковских процессов (СММ) представляют собой конечные автоматы, которые описывают систему по наблюдаемым параметрам. Одна из методик обнаружения аномалий основана на предположении о существовании скрытого Марковского процесса, который генерирует нормальные временные ряды. Если такой процесс отсутствует, то методика не способна обнаружить аномалии.

Нечеткие методы представляют класс математических моделей, основанных на нечетких временных рядах (НВР), которые описывают динамику процессов в условиях неопределенности с помощью нечетких меток. Применение нечетких методов для обнаружения аномалий в НВР связано с задачей диагностики процессов, описываемых этими рядами. В НВР

аномалиями могут быть значения исходного временного ряда, уровни нечеткого временного ряда, нечеткие элементарные тенденции, нечеткие локальные тенденции, синтаксические правила и весь НВР в целом. Для поиска нетипичных показателей в НВР можно использовать методы частотного или синтаксического анализа. Последний подразумевает введение нечеткого языка (нечеткой грамматики) и его распознавание с помощью нечетких конечных автоматов. Кроме того, обнаружение аномалий в НВР можно свести к задачам, решаемым с использованием ранее упомянутых методов, таких как скользящее окно и кластеризация, с последующим проведением частотного анализа результатов.

Алгоритм поиска аномалий предпочтителен в случаях, когда необходимо обнаружить разнообразные типы аномалий, система не имеет заранее известных моделей или представлений, и доступны исторические данные для обучения и анализа. Он может использоваться в различных областях, таких как финансы, кибербезопасность, производство и другие, где важно обнаружить необычное или потенциально вредное поведение.

2.3 Классификация инцидентов в системах потоковой обработки событий с применением активных правил

Инцидентом считается любое отклонение системы от нормального поведения или предпосылки к ее отклонению. Инциденты, возникающие в системах потоковой обработки данных, можно классифицировать следующим образом:

- 1) информационные (Info);
- 2) предупреждающие (Warning);
- 3) критические (Critical).

Информационные инциденты – это события, которые указывают на возможное нарушение или отклонение от нормы, но еще не считаются инцидентом. Примером может быть сигнал о том, что объем входных данных

увеличился на 20% за последний час. Обнаружение таких событий происходит автоматически при помощи мониторинговых систем и анализа метрик.

Предупреждающие инциденты – это отклонения от нормы, которые уже требуют внимания и реакции. Они могут быть как незначительными, так и серьезными, и включают в себя такие проблемы, как сбои в системе, ошибки обработки данных, перегрузка системы и т. д. Обнаружение таких инцидентов может происходить как автоматически при помощи мониторинговых систем и анализа метрик, так и при помощи пользовательских отчетов и жалоб.

Критические инциденты – это наиболее серьезные инциденты, которые требуют мгновенной реакции, так как могут привести к серьезным последствиям для бизнеса или пользователей. Эти инциденты могут включать в себя такие проблемы, как потеря данных, нарушение безопасности, отказ в работе ключевых сервисов и т. д. Обнаружение критических инцидентов может происходить как автоматически при помощи мониторинговых систем и анализа метрик, так и при помощи пользовательских отчетов и жалоб.

В зависимости от вида инцидента, необходимо применять автоматические действия для его устранения. Так, для информационных инцидентов зачастую достаточно уведомления SRE инженера компании, а для критических необходимо предусмотреть автоматизированные действия и широковебательные рассылки для наиболее быстрого исправления ситуации.

2.4 Алгоритмы реагирования на инциденты в системах потоковой обработки событий с применением активных правил

Автоматизированные действия по исправлению инцидентов – это набор predetermined действий и процедур, которые выполняются автоматически системой или инструментами управления, с целью решить возникший инцидент или проблему. Они разработаны для быстрого и эффективного реагирования на инциденты и восстановления нормальной работы системы.

Назначение автоматизированных действий по исправлению инцидентов заключается в минимизации влияния возникших проблем на работу системы и

максимальном сокращении времени простоя. Они предназначены для автоматического обнаружения и устранения проблем, без необходимости вмешательства человека. Автоматизированные действия могут включать в себя такие процессы, как перезапуск сервисов, восстановление поврежденных данных, масштабирование ресурсов, перенаправление трафика и другие действия, которые способны восстановить нормальное функционирование системы.

Использование автоматизированных действий по исправлению инцидентов имеет ряд преимуществ. Во-первых, они позволяют сократить время реакции на проблему и устранить ее быстрее, чем при ручном вмешательстве. Во-вторых, автоматизированные действия повышают эффективность и надежность системы, так как их выполнение осуществляется по заранее определенным правилам и процедурам. В-третьих, они позволяют снизить человеческий фактор и вероятность ошибок при решении проблем.

Назначение автоматизированных действий по исправлению инцидентов включает в себя обеспечение оперативности восстановления системы, минимизацию воздействия на пользователей и бизнес-процессы, а также повышение общей доступности и надежности системы. Они также позволяют оптимизировать использование ресурсов и улучшить производительность системы путем автоматической оптимизации и настройки.

Предлагаемая классификация автоматизированных действий по исправлению инцидентов включает следующие категории:

- изменение технических параметров сервисов;
- изменение инструментальных параметров сервисов;
- действия, специфичные для предметной области.

Изменение технических параметров сервисов является важным аспектом автоматизированных действий по исправлению инцидентов. Эти параметры включают количество запущенных реплик, количество выделенных ресурсов и другие технические характеристики, которые имеют прямое влияние на производительность и доступность сервисов.

В случае возникновения инцидентов или проблем в системе, автоматизированные действия могут включать следующие изменения технических параметров:

- количество запущенных реплик: если система испытывает нагрузку, которая превышает возможности текущего числа запущенных реплик сервиса, автоматизированная система может автоматически масштабировать количество реплик, увеличивая их количество для более эффективной обработки запросов;
- количество выделенных ресурсов: в зависимости от изменяющихся требований и нагрузки системы, автоматизированные действия могут включать изменение количества выделенных ресурсов, таких как CPU, память и сетевые ресурсы;
- настройка конфигурации: дополнительно к изменению количества реплик и выделенных ресурсов, автоматизированные действия также могут включать настройку других параметров конфигурации, таких как таймауты, размеры буферов и другие параметры, которые могут быть оптимизированы для решения конкретных проблем или улучшения производительности.

Изменение инструментальных параметров сервисов, в частности параметров Kafka, имеет важное значение в рамках автоматизированных действий по исправлению инцидентов. Как часть архитектуры приложения, Kafka является критическим компонентом системы потоковой обработки данных и событий, и его параметры могут быть оптимизированы для улучшения эффективности и надежности работы системы.

Автоматизированные действия по изменению инструментальных параметров Kafka могут включать следующие аспекты: настройка пропускной способности и задержек, настройка репликации и сохранности данных, настройка партиционирования и балансировки нагрузки.

Изменение параметров, связанных с пропускной способностью и задержками Kafka, может быть полезным при решении проблем с производительностью и латентностью системы. Например, увеличение максимального размера пакета или настройка максимальной задержки передачи

сообщений может помочь улучшить пропускную способность и снизить задержку передачи данных.

Изменение параметров, связанных с репликацией и сохранностью данных в Kafka, может быть необходимым для обеспечения достаточной отказоустойчивости и сохранности сообщений. Это может включать настройку количества реплик, режима сохранности данных (например, синхронный или асинхронный режим) и других параметров, связанных с репликацией и сохранностью.

Изменение параметров, связанных с партиционированием и балансировкой нагрузки в Kafka, может быть полезным при оптимизации распределения данных и равномерного использования ресурсов. Это может включать настройку количества партиций, стратегии балансировки и других параметров, связанных с партиционированием и балансировкой.

В рамках автоматизированных действий по исправлению инцидентов в системах потоковой обработки данных и событий могут быть применены специфичные для предметной области действия. Некоторые из них включают: сброс нагрузки, выставление приоритетов, упрощение алгоритмов.

При возникновении инцидентов, связанных с высокой нагрузкой на систему, можно автоматически выполнить сброс некритической нагрузки или перераспределить ее на другие ресурсы. Это помогает предотвратить перегрузку системы и поддерживать ее стабильность.

В зависимости от характера инцидента и его влияния на систему, можно автоматически изменить приоритет обработки определенных задач или событий. Это позволяет уделить больше ресурсов и внимания наиболее критическим задачам или событиям, способствуя более эффективному функционированию системы.

В случае возникновения инцидентов, связанных с сложными или ресурсоемкими алгоритмами обработки данных, можно применить автоматическое упрощение этих алгоритмов. Это может включать изменение параметров алгоритма, использование более эффективных алгоритмических

подходов или временное отключение определенных функций для обеспечения более быстрой и стабильной обработки данных.

Обобщенный список инцидентов и реакции на них представлен в таблице 1.

Таблица 1 – Автоматические воздействия на инциденты

Инцидент	Автоматизированные действия
Высокая нагрузка на систему	Сброс не критической нагрузки; Перераспределение нагрузки; Динамическое масштабирование.
Пониженная производительность	Увеличение выделенных ресурсов; Перераспределение задач; Упрощение алгоритмов.
Высокий уровень ошибок	Отключение определенных функций; Изменение параметров алгоритмов; Перераспределение задач.
Инцидент, требующий мгновенного реагирования	Выполнение предопределенных аварийных процедур; Мгновенная рассылка диагностической информации персоналу.
Инцидент, специфичный для предметной области	Специализированное действие, зависящее от конкретной области.

Классификация автоматизированных действий по исправлению инцидентов, основанная на технических параметрах сервисов, инструментальных параметрах и действиях, специфичных для предметной области, позволяет системе эффективно реагировать на проблемы и инциденты, применяя соответствующие действия в зависимости от характера и требований системы. Это позволяет достичь оптимального функционирования и максимальной производительности системы потоковой обработки данных и событий.

Выводы

В главе были рассмотрены различные аспекты, связанные с оценкой эффективности систем потоковой обработки событий с применением активных правил, включая модельное представление системы, алгоритмы анализа эффективности, классификацию инцидентов и алгоритмы реагирования на них.

Было предложено модельное представление системы потоковой обработки событий с применением активных правил в виде системы массового обслуживания (СМО). Это позволяет применить существующие методы анализа и оценки эффективности СМО для оценки системы потоковой обработки событий.

Также были рассмотрены алгоритмы пороговых значений, модельного представления и поиска аномалий, позволяющие выявлять инциденты в ходе работы системы, которые в свою очередь служат для оценки и повышения эффективности анализируемых систем.

Рассмотрены различные стратегии реагирования на инциденты, включая автоматическое восстановление после сбоя, динамическое масштабирование ресурсов, изменение правил обработки и др. Каждая стратегия реагирования была описана с указанием ее преимуществ и ограничений. Были рассмотрены как общие алгоритмы, применимые для большого класса систем, так и более специализированные, характерные для систем потоковой обработки с определенной структурой.

Таким образом, глава представляет комплексный подход к оценке и анализу эффективности систем потоковой обработки событий с применением активных правил, а также предлагает алгоритмы для классификации инцидентов и оптимального реагирования на них. Результаты этой главы служат основой для программной реализации системы мониторинга и реагирования на инциденты.

3 Программные средства сбора, анализа и контроля диагностических показателей для повышения эффективности сервиса поточковой обработки событий с применением активных правил

3.1 Архитектура сервиса поточковой обработки событий и исполнения активных правил

Сервис поточковой обработки событий и исполнения активных правил предназначен для эффективной обработки данных и событий, поступающих из различных систем, таких как базы данных и технические объекты. Он основан на событийно-ориентированной обработке, где система реагирует на события, соответствующие заданным шаблонам, и выполняет определенные активные правила для управляемых систем.

В контексте баз данных, механизмы исполнения активных правил могут использоваться для поддержания целостности данных, создания материализованных представлений, производных данных, а также для координирования распределенных вычислений, поддержки транзакций и принятия решений на основе изменений данных.

В отношении технических объектов, механизм исполнения активных правил может быть применен в системах управления и контроля, медицинских приложениях для мониторинга состояния пациентов и предупреждения врачей, транспортных приложениях для прогнозирования пробок, а также в системах контроля воздушного движения для обнаружения потенциально опасных маршрутов и предупреждения диспетчера о возможных столкновениях самолетов.

Таким образом, сервис поточковой обработки событий и исполнения активных правил является мощным инструментом для эффективного управления и контроля различных систем и может быть применен в различных отраслях и приложениях.

Основными функциями сервиса являются:

- создание и редактирование правил, проверка корректности правила, подключение и отключение правила;
- выявление и устранение коллизий исполнения активных правил (статический анализ на этапе конструирования и динамический анализ на этапе исполнения);
- обработка потока событий от наблюдаемых систем, первичная обработка входных событий и формирование событий на основе агрегации входных данных, выявление сложных событий на основе заданных шаблонов, доставка событий к соответствующим компонентам сервиса для их обработки;
- исполнение активных правил;
- журнализация и анализ данных журналов событий;
- аудит обрабатываемых событий и активных правил в режимах онлайн и офлайн, выявление «проблемных» событий;
- настройка параметров функционирования отдельных компонент с учетом результатов аудита, масштабирование отдельных компонент и сервиса в целом, синхронизация глобальных и локальных метаданных, описывающих конфигурацию системы и отдельных компонент.

Архитектура сервиса потоковой обработки событий и исполнения активных правил построена на принципах интеграции микросервисного и событийно-ориентированных подходов. Для доставки событий, поступающих от наблюдаемых систем, используется событийно-ориентированный подход на основе механизма очередей и платформы потоковой обработки данных Kafka с применением фреймворка Kafka Stream. Для обращения к компонентам сервиса в процессе их настройки, запуск и остановка, синхронизации метаданных используется REST [21].

Архитектура сервиса потоковой обработки событий и исполнения активных правил представлена на рисунке 9.

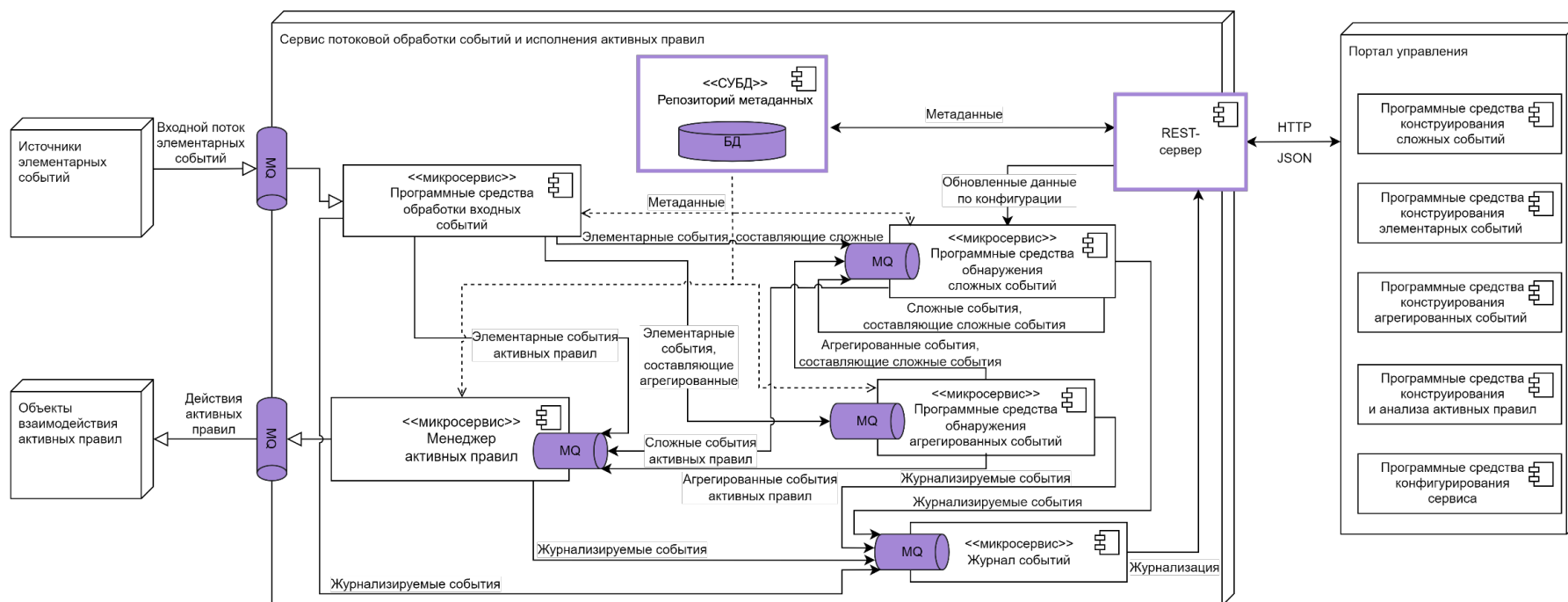


Рисунок 9 – Архитектура сервиса потоковой обработки событий с применением активных правил

Для разработки системы мониторинга и реагирования на инциденты, вышеописанная система, будет реализованная в частичном виде. Этого будет достаточно для проверки функционирования разрабатываемой системы и в тоже время не мешает её применению на более сложных программных комплексах.

3.2 Обоснование выбора технологий и средств разработки программные средства сбора, анализа и контроля диагностических показателей

Для разработки сервиса мониторинга и реагирования на инциденты были выбраны следующие программные средства:

- Kubernetes;
- Prometheus;
- Grafana;
- N8n.

Kubernetes (или K8s) – это открытая платформа для автоматизации развертывания, масштабирования и управления контейнеризированными приложениями. Он предоставляет набор инструментов и механизмов, которые облегчают управление контейнерами и их оркестрацию в распределенной среде.

Основные принципы и концепции Kubernetes включают следующие свойства:

1) контейнеризация: приложения и их зависимости упаковываются в контейнеры, что обеспечивает изоляцию, портативность и повторяемость в различных средах;

2) микросервисная архитектура: приложение разбивается на набор независимых и масштабируемых сервисов, где каждый сервис может быть упакован в отдельный контейнер и масштабирован независимо от других сервисов;

3) оркестрация – механизм для автоматического развертывания, масштабирования и управления контейнеризированными приложениями, который управляет распределением контейнеров по физическим или виртуальным узлам, обеспечивая балансировку нагрузки, отказоустойчивость и эффективное использование ресурсов;

4) декларативная конфигурация – описание желаемого состояния системы, которое будет стремиться поддерживать Kubernetes, с помощью конфигурационных файлов;

5) самовосстановление – механизм для обнаружения и автоматического восстановления от сбоев, выполняющий перезапуск контейнеров, масштабирование приложений, переключение трафика и обеспечение высокой доступности приложений;

6) мониторинг и логирование обеспечивает механизмы для мониторинга и сбора логов из контейнеров и узлов для отслеживания состояния приложений и ресурсов, а также обнаружения и решения проблемы в режиме реального времени.

Основными преимуществами для выбора данной платформы являются мониторинг, который позволяет в реальном времени собирать и анализировать состояние компонентов системы, и механизм оркестрации, который позволяет производить масштабирование микросервисов в зависимости от параметров системы. Кроме того, Kubernetes обладает широким сообществом и экосистемой инструментов, что облегчает разработку и поддержку системы [22].

Prometheus и Grafana стоит рассматривать вместе, так как Prometheus используется для сбора метрик, а Grafana для их последующей визуализации. Для выбора системы мониторинга было рассмотрено три варианта: Zabbix, Icinga и Prometheus.

Zabbix и Icinga предлагают более широкий набор функций по умолчанию, включая оповещение и создание отчетов. Они также поддерживают большое количество плагинов и интеграций, что делает их гибкими решениями для

различных сценариев мониторинга. Однако, эти инструменты могут быть сложными в настройке и управлении, особенно для новых пользователей.

С другой стороны, Prometheus предоставляет мощную систему мониторинга событий, позволяющую быстро реагировать на изменения в системе. Хранилище данных Prometheus основано на NoSQL (TSDB), обеспечивая быстрый доступ к данным и высокую производительность. Кроме того, Prometheus поддерживает сторонний Node Exporter, который позволяет собирать различные метрики, связанные с аппаратной частью и ядром.

Prometheus также предоставляет простой интерфейс для расширения и интеграции с другими инструментами, в том числе Grafana для визуализации данных. Большое сообщество разработчиков активно поддерживает и развивает Prometheus, обеспечивая его актуальность и совершенствование.

n8n – это бесплатный, открытый и расширяемый инструмент для автоматизации различных сценариев работы с данными, интеграции сервисов друг с другом и реализации серверной логики приложений. Этот инструмент позволит добавлять и редактировать управляющие воздействия в режиме реального времени.

n8n имеет широкий набор встроенных узлов (nodes), которые позволяют интегрировать различные сервисы, приложения и инструменты, такие как базы данных, облачные сервисы, электронная почта, Slack и другие. Пользователи могут создавать собственные рабочие процессы, выбирая нужные узлы и определяя их последовательность и параметры.

Преимущества n8n включают гибкость, расширяемость и возможность интеграции с различными сервисами. Он позволяет быстро создавать и изменять рабочие процессы визуально, без необходимости программирования. Это делает его полезным инструментом для разработки системы мониторинга и реагирования на инциденты, где требуется автоматизация операций для эффективного управления и реагирования на инциденты.

3.3 Архитектура и сценарий функционирования программных средства сбора, анализа и контроля диагностических показателей

Система мониторинга и диагностирования показателей является набором компонентов, осуществляющих сбор и анализ показателей наблюдаемой системы, а также набор автоматизированных действий, направленных на поддержание показателей системы при выявлении инцидентов. Система включает в себя (рисунок 10):

- система мониторинга (Prometheus);
- визуализация данных (Grafana);
- менеджер уведомлений (AlertManager);
- конструктор автоматизированных действий (n8n).

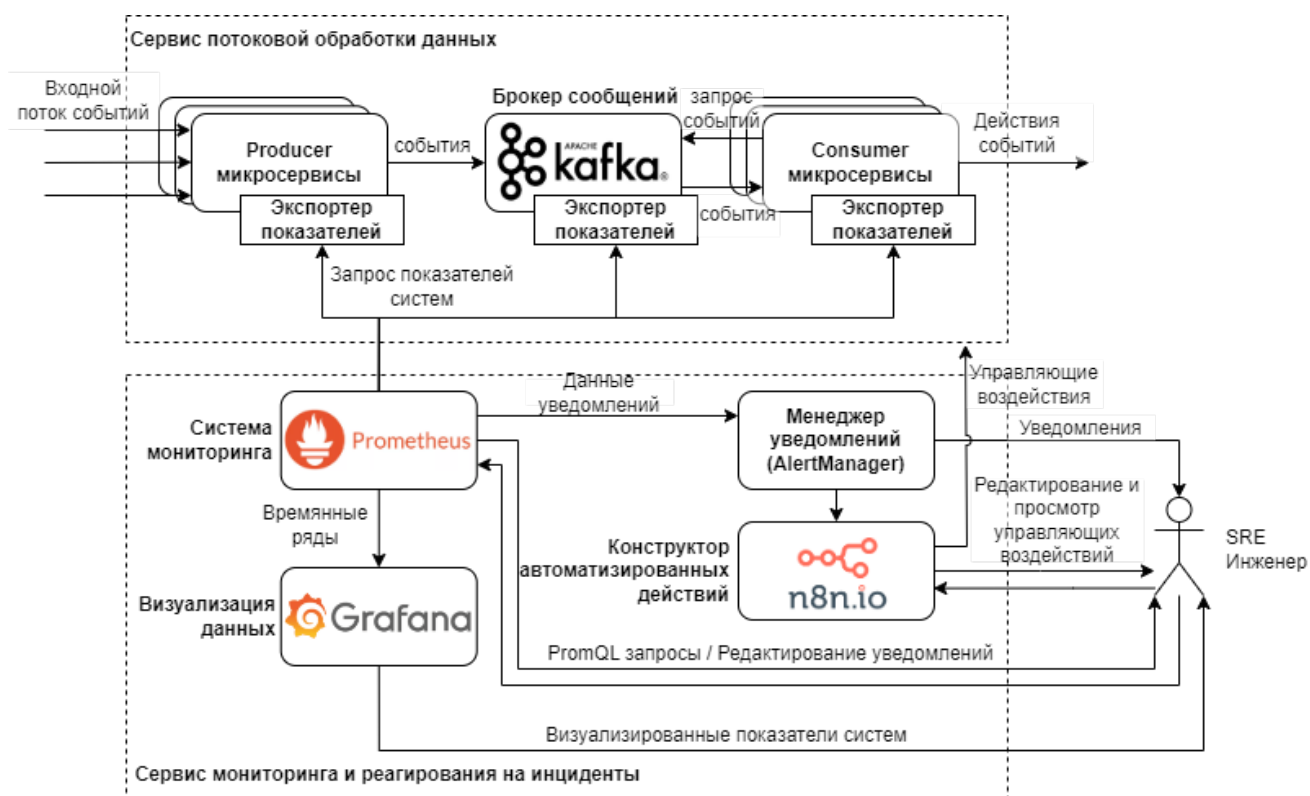


Рисунок 10 – Архитектура сервиса

Также для проверки и демонстрации работы системы мониторинга, был разработан упрощенный прототип системы потоковой обработки данных, состоящей из следующих компонентов:

- 1) producer микросервисы (масштабируемые микросервисы для доставки сообщений а брокер);
- 2) брокер сообщений (Kafka);
- 3) consumer микросервисы (масштабируемые микросервисы для обработки сообщений).

Базовый сценарий функционирования сервиса заключается в выявлении инцидентов и соответствующей реакции на них (рисунок 11) [23].

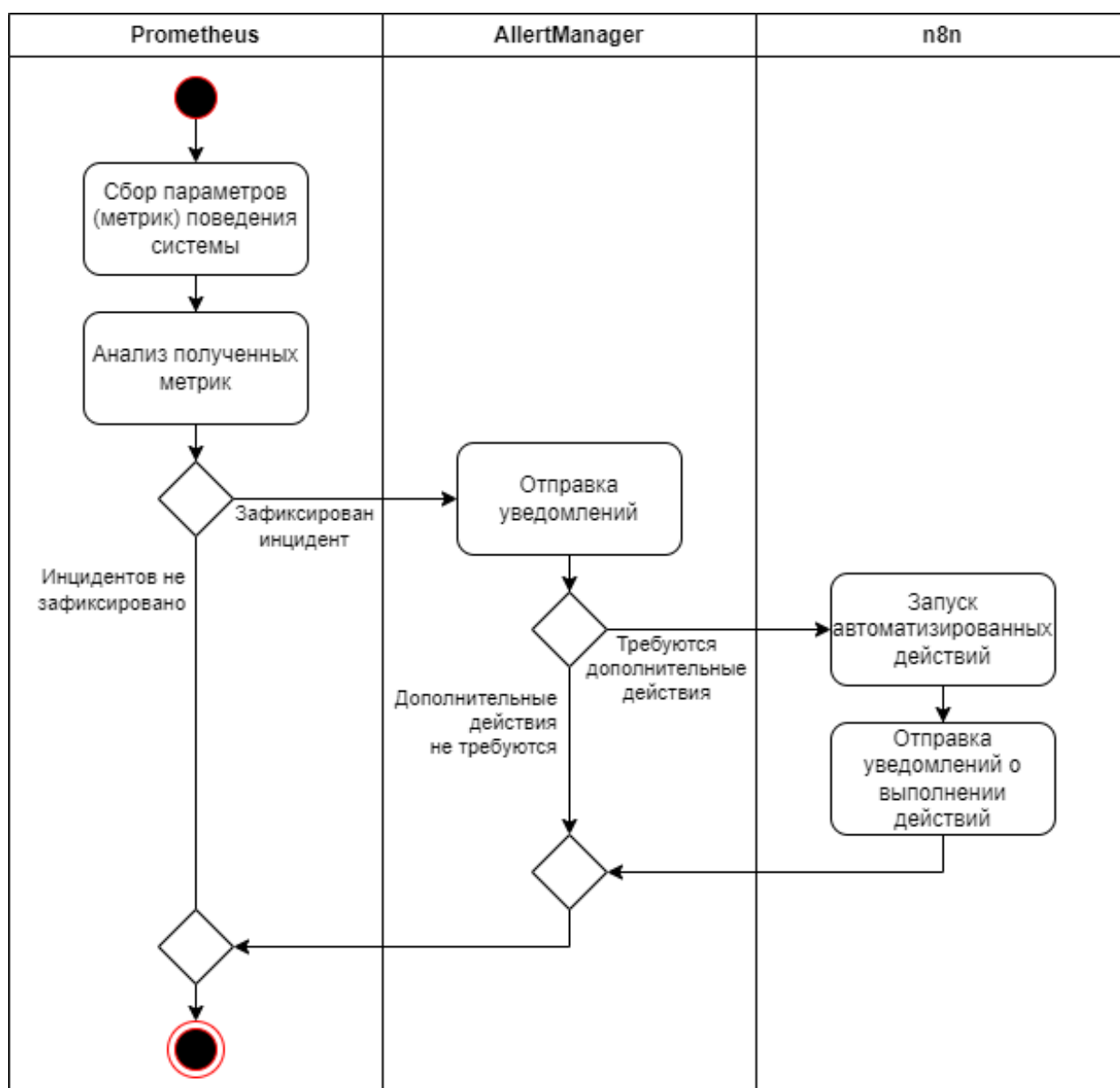


Рисунок 11 – Сценарий функционирования сервиса

В зависимости от сложности отслеживаемой системы сценарий функционирования может быть усложнен и расширен дополнительными

шагами, таким как запрос дополнительным метрик для уточнения неисправности, анализ показателей с помощью модельного представления системы и т. д. Текущий вариант системы функционирования является наиболее универсальным и покрывающим большую часть потребности мониторинга и реагирования на инциденты [24].

3.4 Разработка программных средства сбора, анализа и контроля диагностических показателей

В данной главе рассматриваются основные этапы разработки сервиса мониторинга и реагирования на инциденты. Данные этапы включают в себя установку, настройку и разработку компонентов сервиса, а также конфигурацию правил выявления инцидентов и создание автоматизированных действий.

3.4.1 Создание и настройка окружения

В качестве платформы для разработки был выбран персональный компьютер со следующими техническими характеристиками:

- оперативная память: 16 Гб. DDR4 SO-DIMM;
- встроенная память: SSD 128 Гб;
- процессор: Intel Core i5-7200U;
- оперативная система: Debian 11.7.0 XFCE.

Для разворачивания системы оркестровки контейнеризированных приложений Kubernetes было использовано открытое программное обеспечение minikube. Minikube – инструмент для запуска одноузлового кластера Kubernetes на виртуальной машине в персональном компьютере. В качестве виртуальной машины был использован docker контейнер с предварительно настроенным лимитом ресурсов (рисунки 12, 13).

```
dockdrew@dockdrew:~$ minikube start
minikube v1.30.1 на Debian 11.7
Используется драйвер docker на основе существующего профиля
Запускается control plane узел minikube в кластере minikube
Скачивается базовый образ ...
Перезагружается существующий docker container для "minikube" ...
Подготавливается Kubernetes v1.26.3 на Docker 23.0.2 ...
Configuring bridge CNI (Container Networking Interface) ...
Компоненты Kubernetes проверяются ...
  ■ Используется образ registry.k8s.io/metrics-server/metrics-server:v0.6.3
  ■ Используется образ gcr.io/k8s-minikube/storage-provisioner:v5
  ■ Используется образ docker.io/kubernetesui/dashboard:v2.7.0
  ■ Используется образ docker.io/kubernetesui/metrics-scraper:v1.0.8
Some dashboard features require the metrics-server addon. To enable all features please run:

    minikube addons enable metrics-server

Включенные дополнения: storage-provisioner, default-storageclass, metrics-server, dashboard
Готово! kubectl настроен для использования кластера "minikube" и "default" пространства имён по умолчанию
```

Рисунок 12 – Запуск minikube

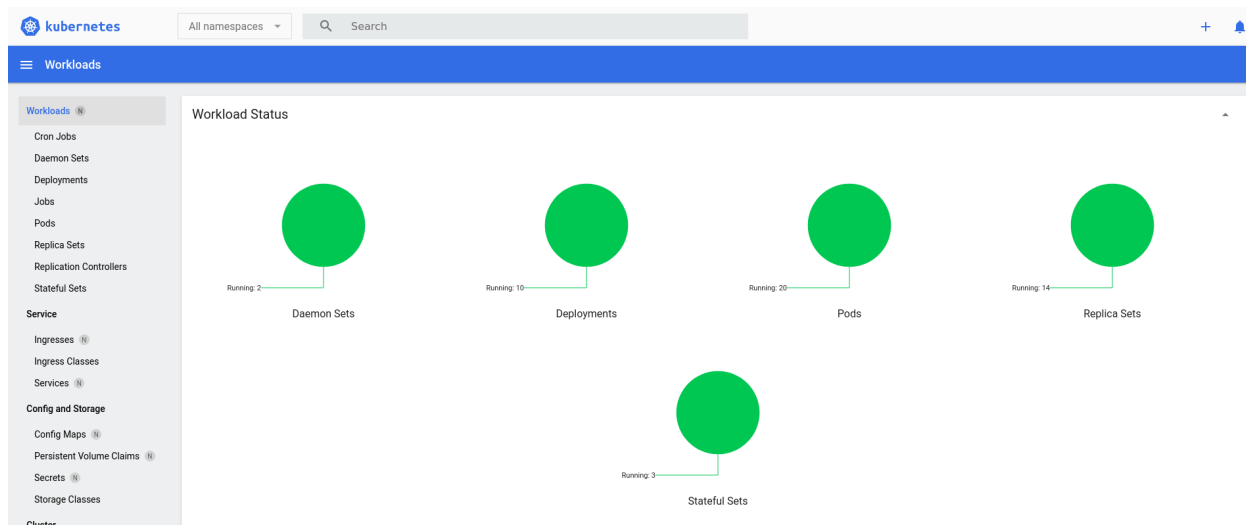


Рисунок 13 – Дашборд запущенного kubernetes

Для разворачивания остальных компонентов будут использованы пакетный менеджер helm, содержащий в себе пакеты для разворачивания уже готовых программных решений, а также собственные конфигурации, написанные в формате YAML для написания собственных сервисов [25].

3.4.2 Разворачивание сервиса потоковой обработки данных

Для разворачивания сервиса потоковой обработки данных, параметры которого будут анализироваться в сервисе мониторинга, необходимо развернуть 3 компонента: consumer, producer, kafka, а также экспортер для предоставления параметров kafka.

Consumer и producer были написаны на языке Java с использованием фреймворка Spring Boot. Для предоставления метрик была использована библиотека micrometer. Она была настроена на предоставление основных параметров JVM, а также времени обработки данных. Образы данных приложений были собраны и добавлены в dockerHub по следующим путям: «dockdrew/consumer:1.0.6» и «dockdrew/producer:1.0.2». Для добавления микросервисов были написаны конфигурационные файлы для Kubernetes в формате YAML. Пример конфигурационного файла для микросервиса consumer представлен в листинге 1.

Листинг 1 – Конфигурация микросервиса consumer

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: consumer
spec:
  selector:
    matchLabels:
      app: consumer
  replicas: 1
  template:
    metadata:
      labels:
        app: consumer
    annotations:
      prometheus.io/scrape: "true"
      prometheus.io/port: "8080"
      prometheus.io/path: "/actuator/prometheus"
  spec:
    containers:
      - name: consumer
        image: dockdrew/consumer:1.0.6
        imagePullPolicy: Always
        ports:
          - containerPort: 8080
        resources:
          limits:
            memory: "1024Mi"
            cpu: "500m"
```

Особое внимание здесь стоит обратить на разделы annotations и containers. В annotations указаны параметры Prometheus для автоматического сбора метрик.

Раздел `containers` содержит основные параметры запускаемого контейнера: образ для развертывания, порт, лимиты ресурсов.

Kafka была развернута с помощью пакетного менеджера следующей командой «`helm install kafka oci://registry-1.docker.io/bitnamicharts/kafka -n kafka`». По умолчанию Kafka не предоставляет метрики для Prometheus, поэтому необходимо установить дополнительный экспортер, также с помощью пакетного менеджера «`helm install kminion kminion/kminion -n monitoring -f kminion-values.yaml`». В ходе проведенных операций были развернуты микросервисы в 3 пространствах имен: `kafka`, `monitoring` и `processing` (рисунки 14 - 16).

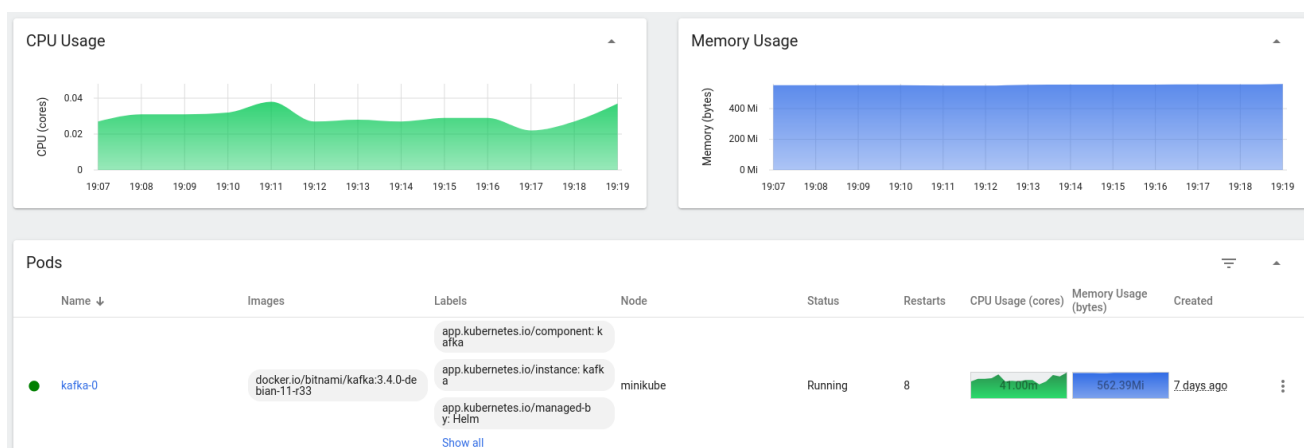


Рисунок 14 – Запущенные поды в пространстве `kafka`

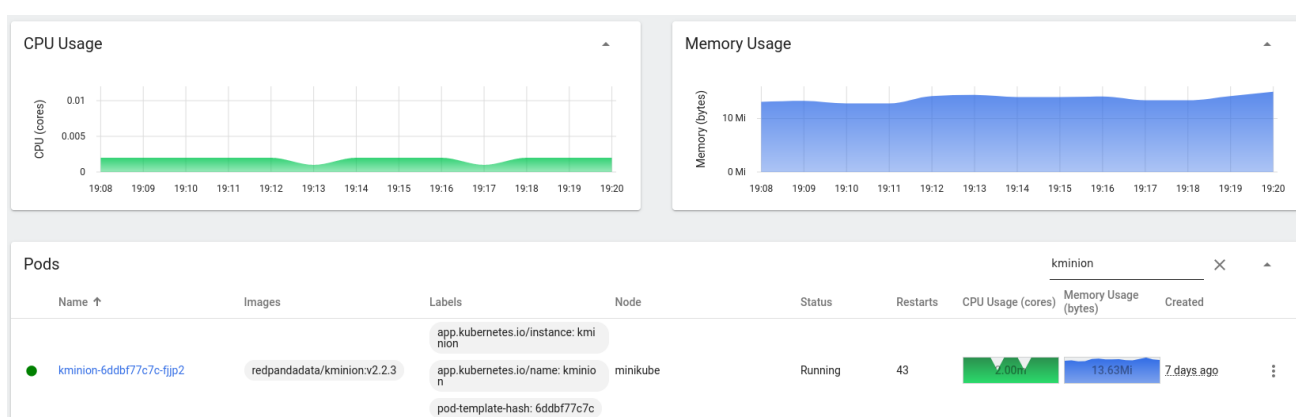


Рисунок 15 – Запущенные поды в пространстве `monitoring`

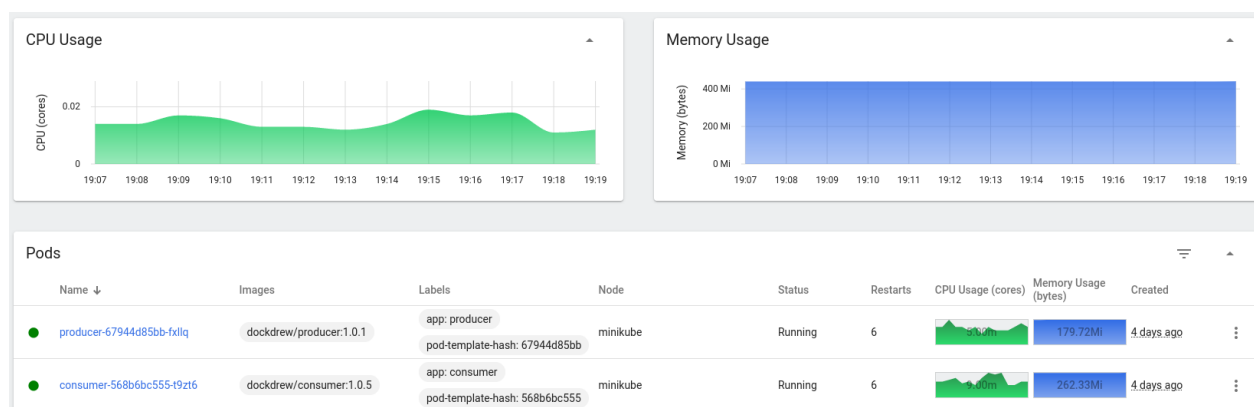


Рисунок 16 – Запущенные поды в пространстве processing

Далее необходимо установить непосредственно сами средства мониторинга.

3.4.3 Установка и настройка программных средств мониторинга Prometheus, AlertManager, Grafana

Данные средства, как и ранее рассмотренная Apache Kafka, входят в пакетный менеджер helm. Установка производится командой «helm install kube-prom-stack prometheus-community/kube-prometheus-stack -n monitoring -f prometheus-values.yaml» (рисунок 17).

```
dockdrew@dockdrew:~$ kubectl get svc -n monitoring
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
alertmanager-operated	ClusterIP	None	<none>	9093/TCP, 9094/TCP, 9094/UDP	6d5h
kminion	LoadBalancer	10.101.243.240	10.101.243.240	8080:30526/TCP	7d19h
kube-prom-stack-grafana	LoadBalancer	10.105.234.117	10.105.234.117	80:31813/TCP	7d19h
kube-prom-stack-kube-prometheus-alertmanager	LoadBalancer	10.108.224.145	10.108.224.145	9093:32543/TCP	7d19h
kube-prom-stack-kube-prometheus-operator	ClusterIP	10.103.23.17	<none>	443/TCP	7d19h
kube-prom-stack-kube-prometheus	LoadBalancer	10.108.43.148	10.108.43.148	9090:31880/TCP	7d19h
kube-prom-stack-kube-state-metrics	ClusterIP	10.100.61.249	<none>	8080/TCP	7d19h
kube-prom-stack-prometheus-node-exporter	ClusterIP	10.102.3.61	<none>	9100/TCP	7d19h
prometheus-operated	ClusterIP	None	<none>	9090/TCP	7d19h

Рисунок 17 – Список установленных программных средств

В данный пакет уже входят все основные компоненты, такие как Prometheus, AlertManager и Grafana. Также в пакет уже входит экспортер базовых показателей всего кластера, но Prometheus пока что не может собирать метрики с ранее внедренных микросервисов. Для того, чтобы реализовать автоматическое добавление мониторинга подов с параметром «prometheus.io/scrape: "true"» необходимо добавить конфигурацию Prometheus (листинг 2).

Листинг 2 – Конфигурация Prometheus

```
prometheus:
  prometheusSpec:
    additionalScrapeConfigs:
      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
          - role: pod
        relabel_configs:
          - source_labels:
            [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
            action: keep
            regex: true
          - source_labels:
            [__meta_kubernetes_pod_annotation_prometheus_io_path]
            action: replace
            target_label: __metrics_path__
            regex: (.+)
          - source_labels: [__address__,
            __meta_kubernetes_pod_annotation_prometheus_io_port]
            action: replace
            regex: ([^:]+)(?::\d+)?;(\d+)
            replacement: $1:$2
            target_label: __address__
          - action: labelmap
            regex: __meta_kubernetes_pod_label_(.+)
          - source_labels: [__meta_kubernetes_namespace]
            action: replace
            target_label: kubernetes_namespace
          - source_labels: [__meta_kubernetes_pod_name]
            action: replace
            target_label: kubernetes_pod_name
```

При применении данной конфигурации, Prometheus автоматически будет собирать метрики с новых подключаемых подов в кластере, если у них заданы соответствующие параметры.

3.4.4 Установка и настройка n8n

N8n необходимо установить вне кластера, в котором расположены все остальные компоненты системы. Это необходимо для того, чтобы n8n имел доступ ко всему кластеру и к каждому из его компонентов.

Данный компонент написан на платформе Node.js и входит в состав менеджера пакетов npm, что позволяет произвести установку с помощью одной команды «`sudo npm install n8n -g`». Последующий запуск производится

командой «n8n start –tunnel», где аргумент «tunnel» необходим для приема webhook запросов от внешних сервисов (рисунок 18).

```
dockdrew@dockdrew:~$ n8n start --tunnel
License manager not initialized
n8n ready on 0.0.0.0, port 5678
Initializing n8n process

Waiting for tunnel ...
Tunnel URL: https://m3freuu51s5154ayz3493uf7.hooks.n8n.cloud/

IMPORTANT! Do not share with anybody as it would give people access to your n8n instance!
Version: 0.230.3

Editor is now accessible via:
http://localhost:5678/

Press "o" to open in Browser.
```

Рисунок 18 – Запуск n8n

Таким образом модуль n8n был развернут и подготовлен к работе.

3.4.5 Настройка правил выявления инцидентов

Правила выявления инцидентов настраиваются в конфигурационном файле Prometheus. Общая структура правила состоит из следующих параметров:

- alert – название правила;
- expr – условие, необходимое для исполнения правила;
- for – время, в течении которого должно выполняться условие;
- labels – секция с пользовательскими параметрами, передаваемыми в уведомлении;
- annotation – текстовая формулировка заголовка и описания уведомления.

Для демонстрации было внедрено 2 правила для параметра kminion-kafka-consumer-group-topic-lag, отражающий в себе количество сообщений, который не были обработаны. Первое правило применяется при превышении параметра критического значение равного 100. Второе правило рассчитывается на основании z-оценки (листинг 3). Z-оценка – это мера относительного разброса наблюдаемого или измеренного значения, которая показывает, сколько

стандартных отклонений составляет его разброс относительно среднего значения [26].

Листинг 3 – Конфигурация правил Prometheus

```
additionalPrometheusRulesMap:
  rule-name:
    groups:
      - name: kafka-group
        rules:
          - record: kafka_topic_lag_avg_over_time_1w
            expr:
avg_over_time(kminion_kafka_consumer_group_topic_lag[1w])
          - record: kafka_topic_lag_stddev_over_time_1w
            expr:
stddev_over_time(kminion_kafka_consumer_group_topic_lag[1w])

          - alert: ConsumerGroupTopicLag
            expr: kminion_kafka_consumer_group_topic_lag > 100
            for: 1m
            labels:
              severity: critical
              group_name: kafka-group
            annotations:
              summary: "Consumer group {{ $labels.group_id }} is
lagging"
              description: "Consumer group {{ $labels.group_id }}
is lagging for topic {{ $labels.topic_name }}"

          - alert: ConsumerGroupTopicLag_Z-score
            expr: abs((kminion_kafka_consumer_group_topic_lag -
kafka_topic_lag_avg_over_time_1w) /
kafka_topic_lag_stddev_over_time_1w) > 3
            for: 1m
            labels:
              severity: critical
              group_name: kafka-group
            annotations:
              summary: "Consumer group {{ $labels.group_id }} is
lagging"
              description: "Consumer group {{ $labels.group_id }}
is lagging for topic {{ $labels.topic_name }}"
```

При симуляции повышенной нагрузки, отслеживаемый параметр превышает допустимые значения и оба правила успешно срабатывают. Уведомления приходят в мессенджер Telegram, а также в систему

автоматизации действий n8n для последующего применения управляющих воздействий (рисунки 19 - 21).

▼ ConsumerGroupTopicLag (1 active)

name: ConsumerGroupTopicLag

expr: kminion_kafka_consumer_group_topic_lag > 100

for: 1m

Labels:

group_name: kafka-group

severity: critical

annotations:

description: Consumer group {{ \$labels.group_id }} is lagging for topic {{ \$labels.topic_name }}

summary: Consumer group {{ \$labels.group_id }} is lagging

Labels

alertname=ConsumerGroupTopicLag

app_kubernetes_io_instance=kminion

app_kubernetes_io_name=kminion

group_id=default

group_name=kafka-group

instance=10.244.1.87:8080

job=kubernetes-pods

kubernetes_namespace=monitoring

kubernetes_pod_name=kminion-6d9d99d6fd-525cd

pod_template_hash=6d9d99d6fd

severity=critical

topic_name=t.default

▼ ConsumerGroupTopicLag_Z-score (1 active)

name: ConsumerGroupTopicLag_Z-score

expr: abs((kminion_kafka_consumer_group_topic_lag - kafka_topic_lag_avg_over_time_1w) / kafka_topic_lag_stddev_over_time_1w) > 3

for: 1m

Labels:

group_name: kafka-group

severity: critical

annotations:

description: Consumer group {{ \$labels.group_id }} is lagging for topic {{ \$labels.topic_name }}

summary: Consumer group {{ \$labels.group_id }} is lagging

Labels

alertname=ConsumerGroupTopicLag_Z-score

app_kubernetes_io_instance=kminion

app_kubernetes_io_name=kminion

group_id=default

group_name=kafka-group

instance=10.244.1.87:8080

job=kubernetes-pods

kubernetes_namespace=monitoring

kubernetes_pod_name=kminion-6d9d99d6fd-525cd

pod_template_hash=6d9d99d6fd

severity=critical

topic_name=t.default

Рисунок 19 – Уведомления в Prometheus

Webhook

Listen For Test Event

Parameters

Docs

Webhook URLs

Test URL

Production URL

POST

http://localhost:5678/webhook-test/kafka-workflow

Authentication

None

HTTP Method

POST

Path

kafka-workflow

Respond

Immediately

Response Code

200

Options

No properties

Add Option

OUTPUT

1 Item

headers	params	query	body
<div><div>x-real-ip: 37.113.55.143</div><div>x-forwarded-for: 37.113.55.143</div><div>host: m3freuu51s5154ayz3493uf7.hooks.n8n.cloud</div><div>x-forwarded-proto: https</div><div>x-nginx-proxy: true</div><div>connection: close</div><div>content-length: 1781</div><div>user-agent: Alertmanager/0.25.0</div><div>content-type: application/json</div></div>	<div>[empty object]</div>	<div>[empty object]</div>	<div><div>receiver: webhook-kafka</div><div>status: firing</div><div>alerts</div><div>0</div><div><div>status: firing</div><div>labels</div><div><div>alertname: ConsumerGroupTopicLag</div><div>app_kubernetes_io_instance: kminion</div><div>app_kubernetes_io_name: kminion</div><div>group_id: default</div><div>group_name: kafka-group</div><div>instance: 10.244.1.87:8080</div><div>job: kubernetes-pods</div><div>kubernetes_namespace: monitoring</div><div>kubernetes_pod_name: kminion-6d9d99d6fd-525cd</div><div>pod_template_hash: 6d9d99d6fd</div><div>prometheus: monitoring/kube-prom-stack-kube-prometheus</div><div>severity: critical</div><div>topic_name: t.default</div></div><div>annotations</div><div><div>description: Consumer group default is lagging for topic t.default</div><div>summary: Consumer group default is lagging</div></div><div>startsAt: 2023-06-11T10:42:41.986Z</div></div></div>

Рисунок 20 – Уведомление в n8n

```

Alerts Firing:
Labels:
- alertname = ConsumerGroupTopicLag_Z-score
- app_kubernetes_io_instance = kminion
- app_kubernetes_io_name = kminion
- group_id = default
- group_name = kafka-group
- instance = 10.244.1.87:8080
- job = kubernetes-pods
- kubernetes_namespace = monitoring
- kubernetes_pod_name = kminion-6d9d99d6fd-525cd
- pod_template_hash = 6d9d99d6fd
- prometheus = monitoring/kube-prom-stack-kube-prometheus
- severity = critical
- topic_name = t.default
Annotations:
- description = Consumer group default is lagging for topic t.default
- summary = Consumer group default is lagging
Source: http://kube-prom-stack-kube-prometheus.monitoring:9090/graph?g0.expr=abs%28%28kminion\_kafka\_consumer\_group\_topic\_lag%2Fkafka\_topic\_lag\_avg\_over\_time\_1w%29+%2Fkafka\_topic\_lag\_stddev\_over\_time\_1w%29+%3E+3&g0.tab=1
13:14

```

Рисунок 21 – Уведомление в telegram

Полная конфигурация средств мониторинга находится в приложении Б. В ней содержатся настройки prometheus, настройки отправки уведомлений и сами правила, для формирования уведомлений. Дальнейшим шагом будет настройка автоматизированных действий на полученные уведомления.

3.4.6 Создание автоматизированных действий

Ранее разработанные правила свидетельствуют о том, что вычислительной мощностью микросервиса consumer не хватает для своевременной обработки поступающих сообщений. Для исправления этой ситуации необходимо реализовать автоматическое масштабирование микросервиса при наступлении данной ситуации. В случае со 2-м правилом, основывающимся на поиске аномалий, требуется уточнить, требуется ли масштабирование микросервиса в

данной ситуации. Для этого необходимо запросить дополнительные параметры, такие как загруженность процессора данным микросервисом.

Для инициации управляющих действий необходимо добавить компонент Webhook (рисунок 22).

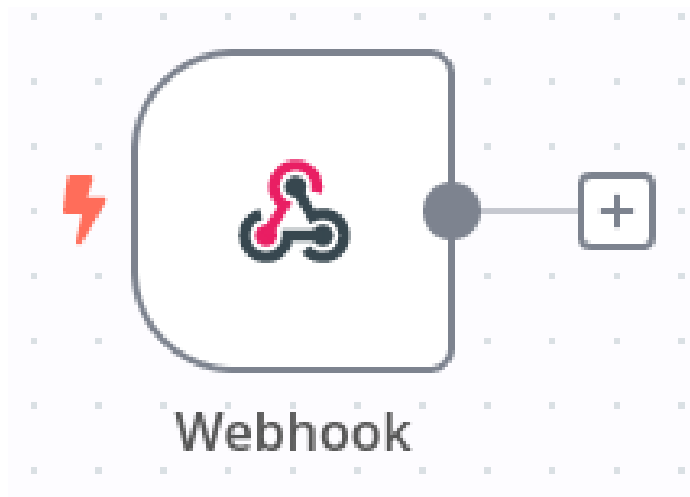


Рисунок 22 – Компонент webhook

AlertManager был предварительно настроен на отправку уведомления на данный webhook. Данный компонент будет обрабатывать входные запросы и передавать полученные данные на последующие этапы обработки. Далее необходимо произвести классификацию получаемых уведомлений. Это можно сделать с помощью компонента switch, используя для сравнения alertname (рисунки 23, 24).

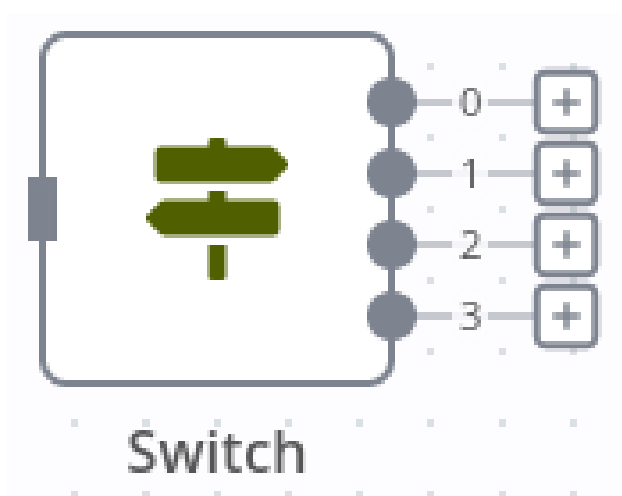


Рисунок 23 – Компонент switch

Switch Execute node

Parameters Docs

Mode
Rules

Data Type
String

Value 1
fx {{ \$json.body.commonLabels.alertname }}

Routing Rules

Operation
Equal

Value 2
ConsumerGroupTopicLag

Output
0

Operation
Equal

Value 2
ConsumerGroupTopicLag_Z-score

Output
1

Add Routing Rule

Fallback Output
3

Рисунок 24 – Настройки компонента switch

После этого необходимо реализовать само автоматизированное действие, а именно масштабирование микросервиса. Это реализуется консольной командой «`kubectrl scale --current-replicas=1 --replicas=2 deployment/consumer -n processing`», которая обозначает, что требуется повысить количество реплик микросервиса `consumer` до 2, если текущее количество реплик равно 1. Для выполнения данной команды необходимо добавить компонент `Execute Command` и вписать в него описанную команду (рисунки 25, 26).

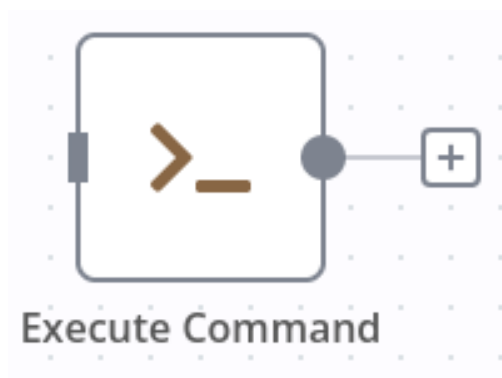


Рисунок 25 – Компонент Execute Command

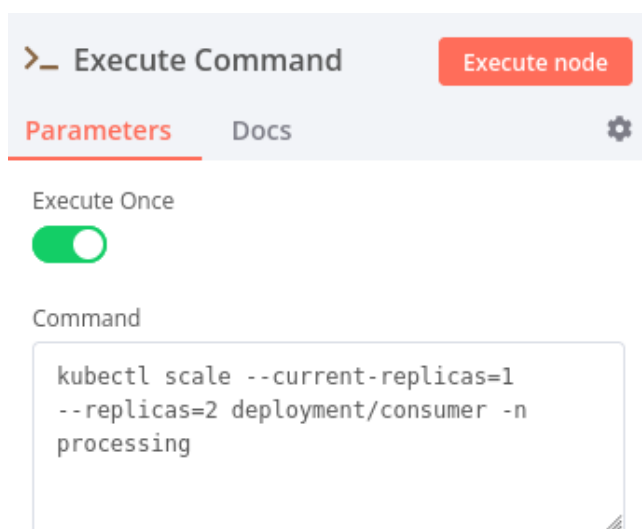


Рисунок 26 – Настройки компонента Execute Command

Завершающим действием будет отправка уведомления о том, что автоматическое действие было выполнено. Как и для уведомлений от Prometheus здесь использовался чат telegram. Для этого использовался компонент telegram с соответствующими параметрами, в которых задается id чата и текст сообщения (рисунки 27, 28).

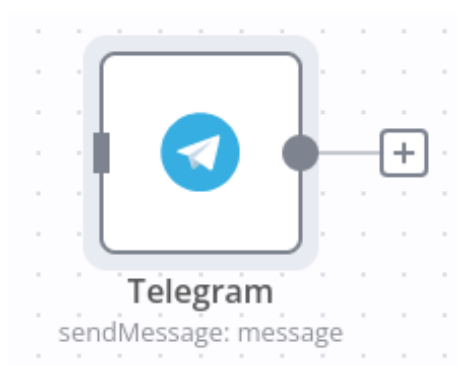


Рисунок 27– Компонент Telegram

Рисунок 28 – Настройки компонента Telegram

Для второго типа инцидентов, помимо уже созданных компонентов, необходимо реализовать дополнительную проверку. Данная проверка реализуется с помощью компонентов HTTP Request и IF (рисунок 29).

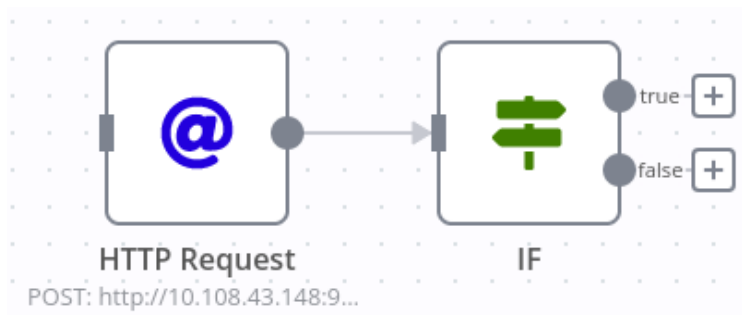


Рисунок 29 – Компоненты HTTP Request и IF

Компонент HTTP Request делает запрос к api сервера Prometheus, чтобы получить загрузку процессора микросервисом consumer

(«`http://10.108.43.148:9090/api/v1/query?query=process_cpu_usage{app="consumer"}`»)). Компонент IF проверяет, если текущее значение загрузки превышает 70%, то происходит автоматическое масштабирование микросервисов, иначе масштабирование не требуется. Полная схема выполнения, с сопутствующими уведомлениями в telegram, представлены на рисунках 30, 31.

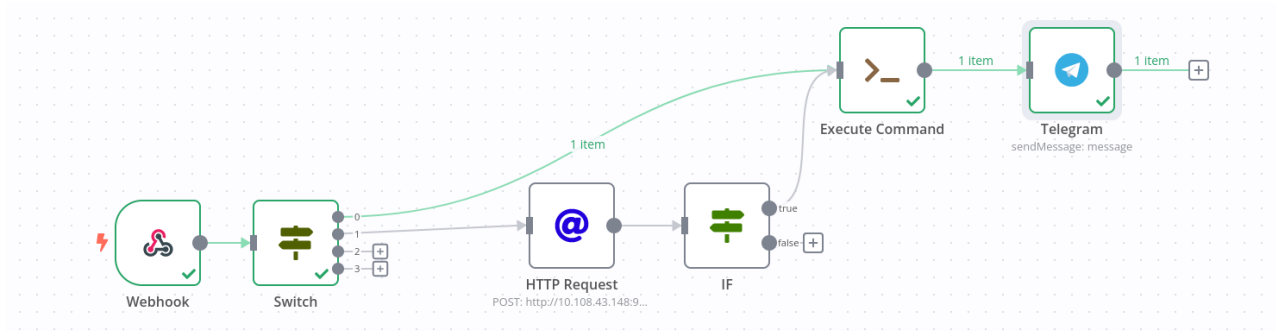


Рисунок 30 – Схема выполнения автоматизированного действия

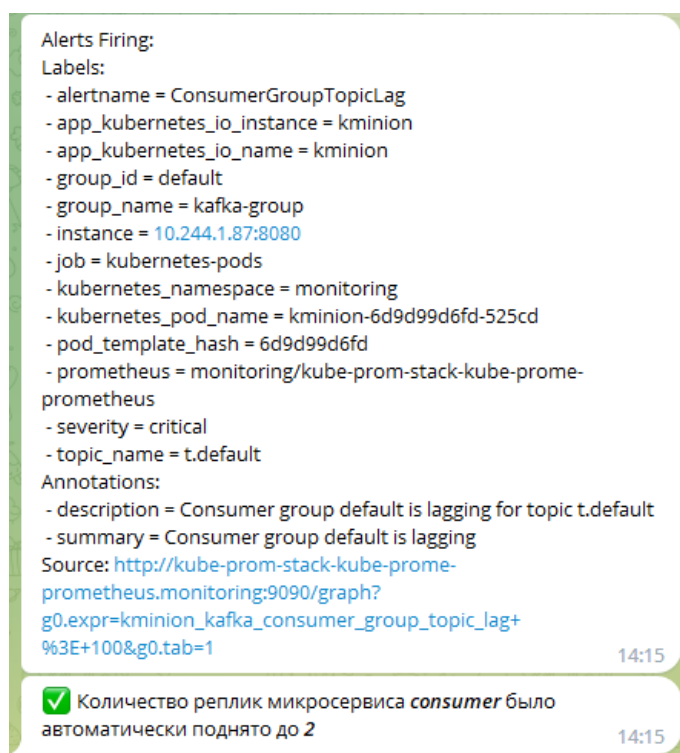


Рисунок 31 – Уведомления в мессенджере telegram

Таким образом, при наступлении критического инцидента, SRE инженер своевременно получит уведомление, а п8п попытается автоматически исправить ситуацию. Система автоматизированных действий позволяет моментально реагировать на сложившуюся ситуацию, на основе заложенных в

нее алгоритмов, что позволяет повысить общую эффективность анализируемой системы. Результаты разработки были размещены в веб-сервис хостинга проектов Github [1].

3.5 Экспериментальная оценка программных средств сбора, анализа и контроля диагностических показателей

Для экспериментальной оценки разработанных программных средств необходимо выделить параметры для оценки эффективности и методы реагирования на инциденты. Из-за того, что системы обработки событий с применением активных правил является прототипом и не имеет большого количества записанных показателей нормального функционирования системы, для анализа эффективности будет использован алгоритм пороговых значений.

Для оценки эффективности будут использованы следующие показатели:

- отношение обрабатываемых событий к поступающим;
- количество сообщений, находящихся в очереди на обработку;
- показатели микросервисов обработки событий (показатели загрузки центрального процессора).

Для реагирования на инциденты также невозможно применить все ранее описанные методы ввиду урезанной архитектуры системы. Поэтому будут применены такие методы:

- масштабирование микросервиса обработки событий;
- изменение логики обработки событий в микросервисе обработки, а именно упрощение обработки наиболее затратных источников;
- ограничение размера очереди kafka (парметр «retention.bytes» топика kafka).

Методы реагирования на инциденты представлены в порядке их влияния на логику работы системы. Первый метод производит масштабирование микросервиса обработки, в следствии чего нагрузка равномерно распределяется и микросервисы могут обработать большее количество событий.

Если при применении предыдущего метода метрики все еще превышают критические значения, то логика обработки наиболее затратных событий изменяется для ускорения их обработки. Структура тестового события выглядит так: «{"data":"9ad2dc6d-54c4-4161-9710-96e77109815c","source":6}». Поле «data» является случайно сгенерированной строкой, а поле «source» содержит в себе номер источника события. Для проведения эксперимента, алгоритм обработки событий был реализован таким способом, что чем больше порядковый номер source, тем ресурсозатратнее выполняется обработка события. Данный метод может сильно влиять на логику обработки событий, так как упрощение алгоритма обработки может привести к неверным расчетам.

Последний метод ограничивает очередь kafka и применяется в том случае, когда предыдущих 2 методов не хватило для исправления ситуации. В следствии применения этого метода на обработку будут поступать только новые события, старые будут стираться, что может привести к сильному нарушению логики работы определенных систем.

Для проведения эксперимента была произведена симуляция входящего потока из 4000000 событий, равномерно распределённом на отрезке времени в 20 минут с выключенной системой реагирования и с включенной. На основе анализируемых параметров были построены графики в системе мониторинга Grafana (рисунки 32, 33).

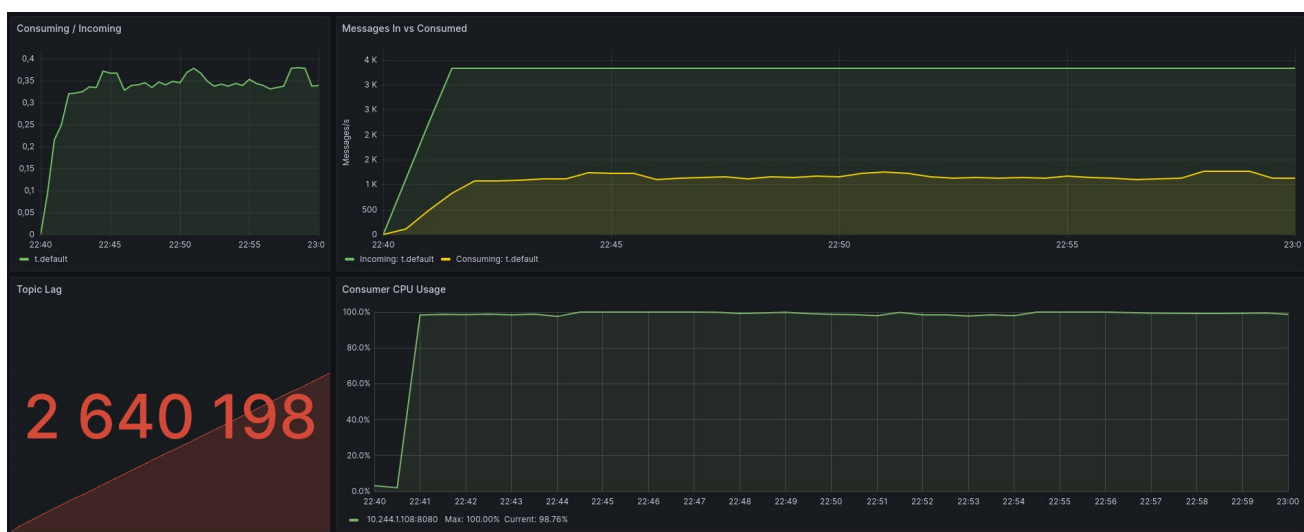


Рисунок 32 – Поведение системы без средств реагирования на инциденты

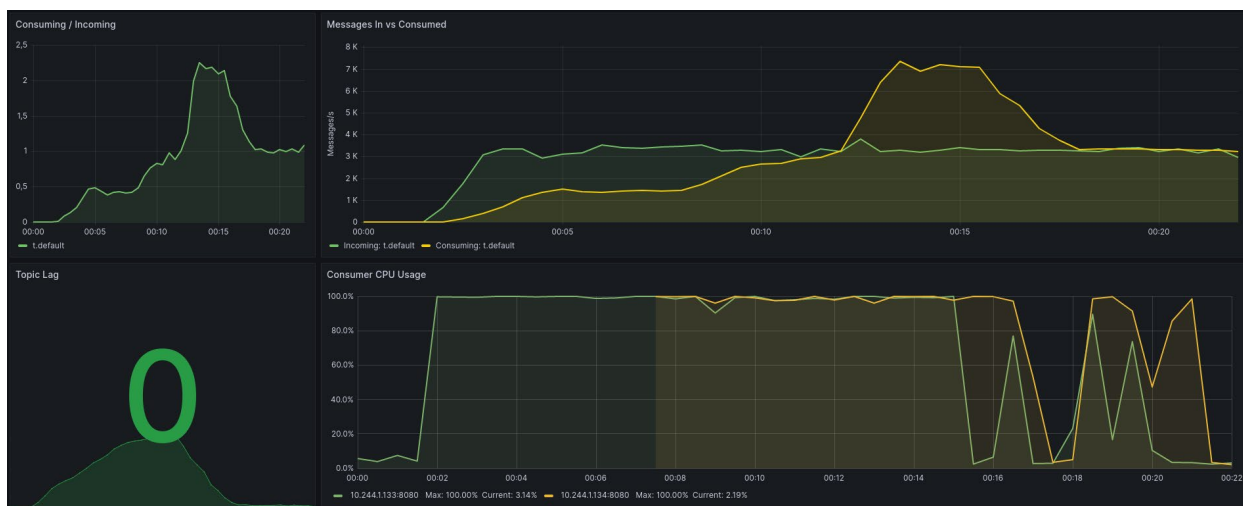


Рисунок 33 – Поведение системы со средствами реагирования на инциденты

На рисунках расположены 4 графика:

- Consuming / Incoming – отношение обрабатываемых событий к поступающим;
- Messages In vs Consumed – график, отражающий количество поступающих и обрабатываемых событий;
- Topic lag – события, которые находятся в очереди на обработку;
- Consumer CPU Usage – использование процессора микросервисами обработки событий.

В 1 эксперименте система не справилась с объемом входящих данных. Очередь необработанных сообщений линейно росла, в следствии чего было обработано всего лишь 1359802 из 4000000. Средняя скорость обработки составила 1155 событий в секунду, что в 3 раза меньше необходимой скорости для непрерывной обработки потока сгенерированного потока (3333 сообщений в секунду). При дальнейшем функционировании системы ситуация бы только ухудшалась, что привело к увеличению задержки между поступлением и обработки событий, некорректной работы обработчика из-за постоянной повышенной нагрузки, переполнению очереди необработанных событий и т. д.

Для анализа 2 эксперимента, необходимо разбить графики на 4 временных отрезка (рисунок 34).



Рисунок 34 – Сегментированные график 2 эксперимента

На 1 этапе функционирование системы аналогично 1 эксперименту: система не справляется с входящим потоком и события накапливаются в очереди.

Через 5 минут происходит автоматическое действие, в ходе чего происходит масштабирование микросервиса обработки событий. Скорость обработки событий увеличивается в 2 раза, но этого было недостаточно для обработки накопившихся сообщений, поэтому через еще 5 минут запустилось 2 автоматизированное действие.

На 3 этапе произошло изменения алгоритма обработки наиболее загруженного типа событий. Скорость обработки событий значительно возросла и достигла 7000 событий в секунду, что позволило обработать сообщения, накопившиеся в очереди. Система начала приходить в стабильное состояние, поэтому 3 управляющее действие не потребовалось.

На 4 этапе произошла стабилизация системы. Очередь из накопившихся запросов была обработана, а нагрузка на процессор снизилась до приемлемых значений.

Сравнение скорости обработки событий в 2-х экспериментах представлено на рисунке 35.

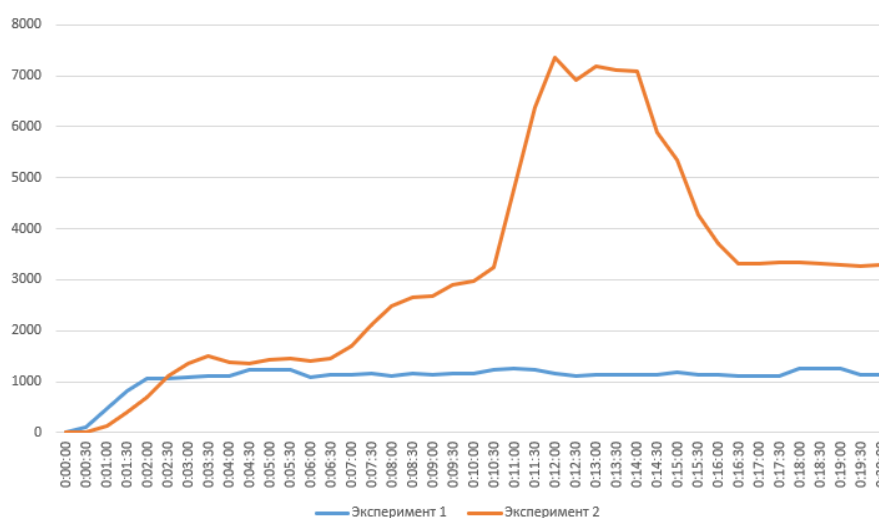


Рисунок 35 – Сравнение скорости обработки инцидентов

Таким образом, эксперимент продемонстрировал факт того, что система мониторинга и реагирования на инциденты может значительно повысить эффективность работы системы потоковой обработки событий с использованием активных правил.

Выводы

Глава посвящена разработке программных средств сбора, анализа и контроля диагностических показателей для повышения эффективности функционирования систем потоковой обработки событий с применением активных правил. Работа программных средств заключается в постоянном

отслеживании метрик системы, с целью выявления инцидентов, и поддержанию их в заданном диапазоне.

На основе произведенного анализа, был обоснован выбор конкретны технологий и средств разработки для реализации программных средств сбора, анализа и контроля диагностических показателей. Описаны основные требования к таким средствам и приведены примеры существующих технологий, таких как Prometheus, AlertManager, Grafana и n8n.

Была представлена архитектура программных средств сбора, анализа и контроля диагностических показателей. Описаны основные компоненты архитектуры, а также был представлен сценарий функционирования этих программных средств, описывающий процесс сбора, анализа и контроля диагностических показателей в реальном времени.

Шаги и процессы разработки программных средств были подробно описаны. Рассмотрены различные этапы, включая создание и настройку окружения, развертывание сервиса потоковой обработки данных, установку и настройку программных средств мониторинга, установку и настройку n8n, настройку правил выявления инцидентов и создание автоматизированных действий.

В завершении была произведена экспериментальная оценка разработанных программных средств. Для этого было произведено сравнение 2-х ситуаций: обработка событий без участия разработанной системы и обработка событий с использованием разработанных средств. В результате было выявлено, что разработанные средства помогают значительно повысить эффективность обработки событий, используя алгоритмы выявления инцидентов и реагирования на них.

Таким образом, глава представляет важные аспекты разработки и использования программных средств для сбора, анализа и контроля диагностических показателей. Эти выводы позволяют разрабатывать и применять эффективные инструменты для повышения эффективности систем потоковой обработки событий с применением активных правил.

Заключение

В данной магистерской работе были исследованы и предложены подходы, направленные на оптимизацию процесса функционирования систем потоковой обработки событий и исполнения активных правил.

В ходе исследования были достигнуты следующие цели:

- разработана система критериев оценки эффективности систем потоковой обработки событий с применением активных правил;
- классифицированы и разработаны алгоритмы оценки эффективности систем потоковой обработки событий с применением активных правил;
- предложен ряд алгоритмов для реагирования на инциденты, часть из которых была реализована в рамках эксперимента;
- реализованы программные средства для сбора и мониторинга показателей систем потоковой обработки событий с применением активных правил, а также средства для автоматического реагирования на инциденты.

В ходе работы были рассмотрены и описаны основные аспекты систем потоковой обработки событий с применением активных правил. Были построены соответствующие диаграммы, для описания работы системы, а также была приведена спецификация системы, как система массового обслуживания.

Были рассмотрены возможные алгоритмы для анализа показателей эффективности системы, реагирования на инциденты, а также выделены параметры эффективности, которые можно использовать для анализа. Все параметры и алгоритмы были описаны и представлены ситуации, в которых их предпочтительно использовать.

В разделе разработки был разработан прототип системы потоковой обработки событий, а также средства для мониторинга и реагирования на инциденты. Также были внедрены некоторые из рассмотренных алгоритмов анализа показателей эффективности и реагирования на инциденты.

В конце был сделан эксперимент, заключающийся в симуляции нагрузки на систему потоковой обработки событий с выключенными средствами мониторинга и реагирования на инциденты и с включенными. В ходе эксперимента было выяснено, что разработанные средства могут значительно повысить эффективность обработки событий.

Таким образом, результаты данной магистерской работы представляют собой важный вклад в область повышения эффективности систем потоковой обработки событий с применением активных правил, что открывает новые возможности для улучшения эффективности и надежности таких систем.

Список использованных источников

1. Красномясов А.А. Программный код магистерской работы «Повышение эффективности функционирования систем потоковой обработки событий и исполнения активных правил» // Веб-сервис хостинга проектов Github [Электронный ресурс]. URL: <https://github.com/D0ckDrew/PSU-masters-thesis> (дата обращения: 09.06.2023).
2. Данилина, Е. А. Системы обработки сложных событий или СЕР-системы / Е. А. Данилина, Д. А. Королева, С. В. Шибанов // Информационные технологии в науке и образовании. Проблемы и перспективы: Сборник научных статей Всероссийской межвузовской научно-практической конференции, Пенза, 14 марта 2018 года / Под редакцией Л.Р. Фионовой. – Пенза: Пензенский государственный университет, 2018. – С. 149-152.
3. Самарев, Р. С. Обзор состояния области потоковой обработки данных / Р. С. Самарев // Труды Института системного программирования РАН. – 2017. – Т. 29, № 1. – С. 231-260.
4. Когаловский М.Р. Энциклопедия технологий баз данных / М.Р. Когаловский. – Москва: Финансы и статистика, 2005. – 800 с.
5. Стопфорд Бен Проектирование событийно-ориентированных систем: Концепции и шаблоны проектирования сервисов потоковой обработки данных с использованием Apache Kafka / Бен Стопфорд. – Иркутск: ITSumma Press, 2019. – 175 с.
6. Яблонский, Д. А. Механизм обработки потоковых данных / Д. А. Яблонский // Научный аспект. – 2021. – Т. 8, № 2. – С. 865-869.
7. Пселтис, Э. Потоковая обработка данных. Конвейер реального времени / Э. Вселтис – М.:ДМК Пресс, 2018. – 218 с.
8. Уэске, Ф. Потоковая обработка данных с Apache Flink / Ф. Уэске, В. Калаври – пер. с англ. В. С. Яценкова. М.: ДМК Пресс, 2021. – 298 с.
9. Нархид, Н. Apache Kafka. Потоковая обработка и анализ данных / Н. Нархид, Г. Шапира, Т. Палино – СПб.: Питер, 2019. – 320 с.

10. Шибанов, С. В. Сервис потоковой обработки событий и исполнения активных правил / С. В. Шибанов, Я. С. Шлепнев // Математическое моделирование и суперкомпьютерные технологии : Труды XXI Международной конференции, Нижний Новгород, 22–26 ноября 2021 года. – Нижний Новгород: Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского, 2021. – С. 403-407.
11. Chakravarthy S. Stream Data Processing: A Quality of Service Perspective / Sharma Chakravarthy, Qingchun Jiang – Springer Science+Business Media, LLC, 2009. – 340 с.
12. Gautam, B., Performance prediction of data streams on high-performance architecture / B. Gautam, A. Basava // Human-centric Computing and Information Sciences volume 9, Article number: 2, 2019. – 23 с.
13. Королев, Е. Н. Использование асинхронного обмена сообщениями между сервисами с помощью брокера Apache Kafka / Е. Н. Королев, М. И. Петриков, М. Е. Королева // Интеллектуальные информационные системы: труды Международной научно-практической конференции: Воронеж, 02–04 декабря 2020 года. Том Часть 2. – Воронеж: Воронежский государственный технический университет, 2021. – С. 45-48.
14. Olups, R. Zabbix Network Monitoring - Second Edition / R. Olups – Packt Publishing, 2016. – 754 с.
15. Mehta, V. Icinga Network Monitoring / V. Mehta – Packt Publishing, 2013. – 118 с.
16. Brazil, B. Prometheus: Up & Running / B. Brazil – O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, 2018. – 386 с.
17. Романенко, В. А. Системы и сети массового обслуживания: учебное пособие / В. А. Романенко – Самара: Издательство Самарского университета, 2021. – 68 с.
18. Солнышкина, И. В. Теория массового обслуживания: учебное пособие / И. В. Солнышкина – Комсомольск-на-Амуре: ФГБОУ ВПО «КНАГТУ», 2015. – 76 с.

19. Yingmin, Li. Diagnosis of Large Software Systems Based on Colored Petri Nets. Software Engineering / Li. Yingmin – Universit'e Paris Sud – Paris XI, 2010. – 213 с.
20. Королева, Д. А. Сравнительный анализ систем обработки сложных событий с открытым исходным кодом / Д. А. Королева, Е. А. Данилина, С. В. Шибанов // Информационные технологии в науке и образовании. Проблемы и перспективы: Сборник научных статей Всероссийской межвузовской научно-практической конференции, Пенза, 14 марта 2018 года – Пенза: Пензенский государственный университет, 2018. – С. 163-164.
21. Самарев, Р. С. Обзор состояния области потоковой обработки данных / Р. С. Самарев // Труды Института системного программирования РАН. – 2017. – Т. 29, № 1. – С. 231-260.
22. Telenyk, S. A Comparison of Kubernetes and Kubernetes-Compatible Platforms / S. Telenyk, O. Sopov, E. Zharikov, G. Nowakowski // Proceedings of the 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, IDAACS 2021: 11, Cracow, 22–25 сентября 2021 года. Vol. 1. – Cracow, 2021. – С. 313-317.
23. Корабейников, Д. Н. Особенности построения систем потоковой обработки данных / Д. Н. Корабейников, И. И. Киселев // Инновации. Наука. Образование. – 2020. – № 24. – С. 1469-1474.
24. Толпинская, Н. Б. Алгоритмы обработки потоковых данных / Н. Б. Толпинская, А. Сычев // Проблемы автоматизации и управления. – 2019. – № 1(36). – С. 110-117.
25. Мазманян, В. Г. Алгоритм запуска многоузлового кластера Kubernetes и работа с ним в операционной системе centos Linux / В. Г. Мазманян, Э. М. Вихтенко // Вестник Российского нового университета. Серия: Сложные системы: модели, анализ и управление. – 2022. – № 2. – С. 149-156.
26. Lucknam, D. Event Processing for Business: Organizing the Real-Time Enterprise / David C. Luckham – Published by John Wiley & Sons, Inc., Hoboken, New Jersey., 2012. – 288 с.

Приложение А
ГЛОССАРИЙ
(справочное)

Активное правило: объект специального вида, задающий реакцию на событие.

Обработка сложных событий (Complex Event Processing, CEP): это подход к анализу событий в реальном времени, с выявлением их взаимосвязей, причин и следствий.

Предметная область: часть реального мира, рассматриваемая в пределах данного контекста. Под контекстом здесь может пониматься, например, область исследования или область, которая является объектом некоторой деятельности.

Сложное событие: событие, полученное из одного или нескольких других событий.

Событие: любой факт или действие, которое происходит с поименованным объектом в определенный интервал времени (идентифицируется объектом и временем).

Журнализация: это процесс записи и хранения событий, происходящих в системе или приложении, в форме журналов или лог-файлов.

Кластер: это группа связанных компьютеров или серверов, которые работают вместе как единая система для выполнения задач или обработки данных. Кластер обычно состоит из нескольких узлов, которые работают параллельно и координируют свою работу для достижения общей цели.

Партиция: это логическое и физическое разделение данных в теме. Каждая тема в Kafka может быть разделена на несколько партиций, и каждая партиция является упорядоченной и независимой последовательностью сообщений.

Инцидент: любое отклонение системы от нормального поведения или предпосылки к ее отклонению

Аномалия: это отклонение, необычное или неожиданное поведение, явление или событие, которое отличается от ожидаемого или нормального состояния.

Оркестрация: это процесс управления и координации различными компонентами или сервисами в компьютерных системах или средах

выполнения. Он включает в себя управление жизненным циклом приложений, развертывание, масштабирование, управление ресурсами и координацию взаимодействия между различными компонентами системы.

Микросервис: это архитектурный подход, при котором приложение разбивается на небольшие, независимые и слабо связанные сервисы, каждый из которых выполняет конкретную функциональность.

SRE инженер: это IT-специалист, чья основная задача заключается в обеспечении надежности, масштабируемости и эффективности операционных систем и сервисов компании.

Метрики: различные показатели программной системы, среды её исполнения или физической вычислительной машины, под которой запущена система с меткой времени, того момента, когда метрики были получены.

Приложение Б
КОД КОНФИГУРАЦИИ PROMETHEUS
(рекомендуемое)

Листинг Б.1 – Код конфигурации Prometheus

```

prometheus:
  prometheusSpec:
    additionalScrapeConfigs:
      - job_name: 'kubernetes-pods'
        kubernetes_sd_configs:
          - role: pod
        relabel_configs:
          - source_labels:
              [__meta_kubernetes_pod_annotation_prometheus_io_scrape]
            action: keep
            regex: true
          - source_labels: [__meta_kubernetes_pod_annotation_prometheus_io_path]
            action: replace
            target_label: __metrics_path__
            regex: (.+)
          - source_labels: [__address__,
              __meta_kubernetes_pod_annotation_prometheus_io_port]
            action: replace
            regex: ([^:]+)(?::\d+)?;(\d+)
            replacement: $1:$2
            target_label: __address__
          - action: labelmap
            regex: __meta_kubernetes_pod_label_(.+)
          - source_labels: [__meta_kubernetes_namespace]
            action: replace
            target_label: kubernetes_namespace
          - source_labels: [__meta_kubernetes_pod_name]
            action: replace
            target_label: kubernetes_pod_name

alertmanager:
  config:
    global:
      telegram_api_url: "https://api.telegram.org"
    route:
      group_by: ['alertname']
      repeat_interval: 1m
      routes:
        - match_re:
            group_name: ^(kafka-group|producer-group|consumer-group)$
            receiver: telegram-alert
            continue: true
        - matchers: [ '{severity!="info",group_name="producer-group"}' ]
            receiver: webhook-producer
            continue: true
        - matchers: [ '{severity!="info",group_name="consumer-group"}' ]
            receiver: 'webhook-consumer'
            continue: true
        - matchers: [ '{severity!="info",group_name="kafka-group"}' ]
            receiver: webhook-kafka
            continue: true
    receivers:
      - name: 'telegram-alert'
        telegram_configs:
          - bot_token: 5426146274:AAHLUaDXLSjqn8cN_Y8oQCkyFjvWc2YcY-E
            api_url: https://api.telegram.org
            chat_id: 382030266
            parse_mode: ''
      - name: 'webhook-producer'
        webhook_configs:

```

```

    - url: 'https://m3freuu51s5154ayz3493uf7.hooks.n8n.cloud/webhook-
test/producer-workflow'
    - name: 'webhook-consumer'
      webhook_configs:
        - url: 'https://m3freuu51s5154ayz3493uf7.hooks.n8n.cloud/webhook-
test/consumer-workflow'
        - name: 'webhook-kafka'
          webhook_configs:
            - url: 'https://m3freuu51s5154ayz3493uf7.hooks.n8n.cloud/webhook-
test/kafka-workflow'

additionalPrometheusRulesMap:
  rule-name:
    groups:
      - name: kafka-group
        rules:
          - record: kafka_topic_lag_avg_over_time_1w
            expr: avg_over_time(kminion_kafka_consumer_group_topic_lag[1w])
          - record: kafka_topic_lag_stddev_over_time_1w
            expr: stddev_over_time(kminion_kafka_consumer_group_topic_lag[1w])

          - alert: ConsumerGroupTopicLag
            expr: kminion_kafka_consumer_group_topic_lag > 100
            for: 1m
            labels:
              severity: critical
              group_name: kafka-group
            annotations:
              summary: "Consumer group {{ $labels.group_id }} is lagging"
              description: "Consumer group {{ $labels.group_id }} is lagging
for topic {{ $labels.topic_name }}"

          - alert: ConsumerGroupTopicLag_Z-score
            expr: abs((kminion_kafka_consumer_group_topic_lag -
kafka_topic_lag_avg_over_time_1w) / kafka_topic_lag_stddev_over_time_1w) > 3
            for: 1m
            labels:
              severity: critical
              group_name: kafka-group
            annotations:
              summary: "Consumer group {{ $labels.group_id }} is lagging"
              description: "Consumer group {{ $labels.group_id }} is lagging
for topic {{ $labels.topic_name }}"

```