Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Научно-образовательный центр «Высшая ИТ школа»

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК

Руководитель ООП

д-р. физ.-мат. наук, профессор

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ О.А.Змеев

*подпись*

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

АВТОМАТИЗАЦИЯ ПРОЦЕССА ЛОГ-МЕНЕДЖМЕНТА В ЦЕНТРЕ

МОНИТОРИНГА ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

по направлению подготовки 09.03.04 Программная инженерия

Направленность (профиль) «Программная инженерия»

Руководитель ВКР

кандидат физ-мат. наук, доцент

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. С. Ким

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Научный консультант

ведущий инженер, ООО "ТЦР"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.С. Зоркин

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

Автор работы

студент группы № 972103

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Е. С. Шамов

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 2025 г.

АННОТАЦИЯ

Выпускная квалификационная работа: 51 страниц, 22 рисунка, 7 листингов, 7 источников.

АВТОМАТИЗАЦИЯ, ЛОГ-МЕНЕДЖМЕНТ, DEVOPS, SIEM, APACHE KAFKA, ANSIBLE, CI/CD, ЦЕНТРАЛИЗОВАННОЕ УПРАВЛЕНИЕ.

Объект исследования: процесс подключения источников логов и управления их конфигурациями в инфраструктуре SOC.

Цель работы: разработка и внедрение автоматизированного процесса подключения источников логов, включающего централизованное управление конфигурациями, автоматическое развертывание коллекторов и интеграцию с существующими компонентами SOC.

Результат работы: выполнен анализ текущего состояния процесса лог-менеджмента, выявлены основные недостатки существующей процедуры, спроектирован и реализован целевой процесс, основанный на хранении состояния в Git-репозитории и автоматизации операций с использованием CI/CD, Ansible и контейнеризации.

Оглавление

Глоссарий 3

Введение 4

1 Введение в предметную область 6

1.1 Процесс обработки событий 7

1.2 Акторы процессов SOC 11

2 Анализ процесса настройки сбора логов 12

2.1 Описание типового процесса настройки 12

2.2 Нетиповые процессы настройки 16

2.3 Оценка существующего процесса настройки сбора логов. 17

2.4 Выводы по анализу 19

3 Изменения во взаимодействии с Kafka 20

3.1 Использование привилегированной сервисной учетной записи. 21

3.2 Миграция в Kafka as a Service 22

3.3 Вывод 24

4 Изменение процесса конфигурации балансировщиков 26

4.1 Варианты решения проблемы 27

5 Проектирование автоматизации 29

5.1 Единый «источник правды» 29

5.2 Организация пайплайна автоматизации 31

5.3 Оптимизация процесса выбора свободного порта для сетевых коллекторов 34

5.4 Описание выбранных инструментов 35

6 Реализация автоматизации 38

6.1 Структура Git-репозитория. 38

6.2 Алгоритм применения изменений 40

6.3 Предоставление данных о развернутом экземпляре заинтересованным лицам 47

6.4 Итоговый процесс 48

Заключение 49

Список использованных источников и литературы 50

Глоссарий

**Инцидент информационной безопасности** – это подтвержденное событие или группа событий, указывающие на нарушение политики безопасности, компрометацию системы или другую угрозу, требующую расследования и реагирования [1].

**Аудит безопасности** – это процесс систематической (часто автоматизированной) проверки и оценки состояния информационной безопасности системы или ее компонента с целью выявления уязвимостей, несоответствий требованиям и рисков, а также выработки рекомендаций по их устранению [2].

**CI/CD-пайплайн** – это автоматизированная последовательность этапов, предназначенная для непрерывной интеграции, тестирования и доставки изменений в программном обеспечении, обеспечивающая ускоренную и безопасную поставку новых версий продукта[3]. Организацией запуска пайплайнов занимается **CI/CD-система**.

Введение

Security Operations Center (**SOC**) – специализированное подразделение, обеспечивающее круглосуточный мониторинг и реагирование на инциденты информационной безопасности (ИБ) [4]. Работа SOC основана на анализе данных, поступающих от разнообразных компонентов контролируемых информационных систем. Каждую фиксируемую запись о значимом событии SOC рассматривает как лог (событие). **Лог** представляет собой краткую структурированную запись, содержащую временную метку, идентификатор источника и описание действия (например, аутентификация пользователя, возникновение ошибки или установление сетевого соединения). Анализ таких записей позволяет выявлять отклонения в поведении систем и потенциальные инциденты ИБ.

Централизованный прием, хранение и обработка логов выполняются системой управления событиями и информацией безопасности (Security Information and Event Management, SIEM). Помимо агрегации данных SIEM осуществляет их обогащение сведениями из внешних источников и корреляционный анализ, формируя сложные цепочки взаимосвязанных событий, указывающих на инциденты.

При нагрузках, достигающих сотен тысяч событий в секунду, в архитектуре SIEM используется промежуточная шина сообщений; в данной инфраструктуре эту функцию выполняет распределенная платформа Apache Kafka. Kafka обеспечивает балансировку потоков между компонентами, временное хранение сообщений, высокую отказоустойчивость и горизонтальное масштабирование, позволяя обрабатывать данные асинхронно с минимальными задержками [5].

Ряд устаревших или специализированных источников логов не поддерживает прямую передачу данных в Kafka. Для обеспечения совместимости применяется промежуточный сервис – коллектор. Коллектор принимает логи по доступным источнику протоколам и перенаправляет их в Kafka без изменения содержимого сообщения, сохраняя целостность данных и снижая трудоемкость интеграции.

Быстрый рост инфраструктуры привел к регулярному появлению новых источников (один-два ежемесячно); их общее количество уже исчисляется десятками. Анализ текущего состояния выявил две основные проблемы.

1. Подготовка нового коллектора занимает до двух рабочих дней; в течение этого времени инженер должен контролировать процесс, что не позволяет ему решать другие задачи.
2. Учет подключенных источников ведется вручную и выполняется параллельно настройке приема событий.

Указанные трудности обусловлены недостаточной автоматизацией существующих процедур. **Цель работы** – разработать и внедрить автоматизированный процесс подключения источников логов, включающий развертывание и конфигурирование коллекторов и сопутствующих компонентов.

Для достижения этой цели в работе необходимо решить следующие задачи.

1. Проанализировать существующий процесс развертывания коллекторов; выявить операции, подлежащие автоматизации.
2. Спроектировать и разработать инструменты автоматизации, которые позволят исключить или минимизировать ручные операции
3. Внедрить разработанное решение в инфраструктуру SOC и провести оценку его эффективности.

# Введение в предметную область

Эксплуатация SOC опирается на связку двух ключевых платформ – SIEM и SOAR.

* **SIEM** (Safety Information and Event Management) агрегирует события из разнородных источников, выполняет корреляционный анализ и формирует **алерты** – уведомления о потенциальных инцидентах информационной безопасности, требующих проверки [6].
* **SOAR** (Security Orchestration, Automation and Response) обеспечивает оркестрацию и автоматизацию реагирования на инциденты. В рамках SOAR функционирует команда реагирования (**CSIRT**, Computer Security Incident Response Team), задачами которой являются анализ алертов, эскалация в инциденты, расследование и подготовка отчетов [7].

На рисунке 1 представлены основные сущности, задействованные в процессах SIEM, и схема их взаимодействия с платформой SOAR.

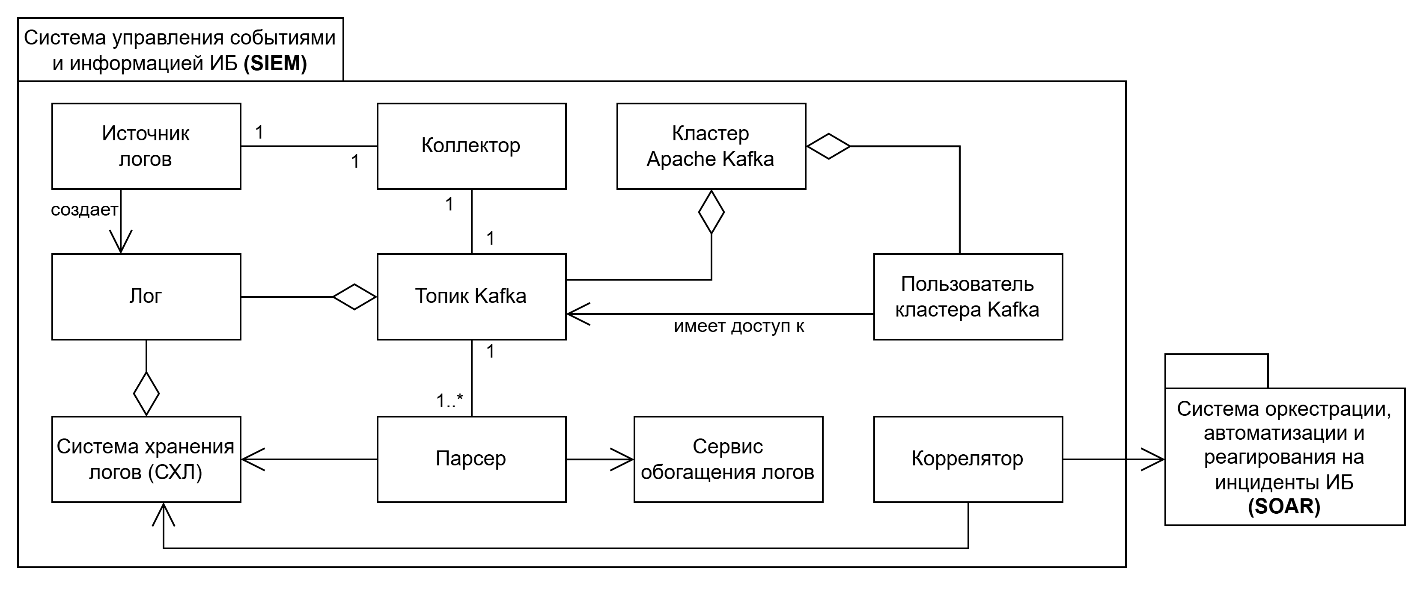


Рисунок 1 – Предметная область SIEM

Каждому источнику логов соответствует отдельный экземпляр коллектора, поскольку такой подход обеспечивает изоляцию потоков данных и поддержку принципа минимальных привилегий. Коллектор, назначенный для конкретного источника, принимает события только от него и осуществляет их передачу строго в один топик Apache Kafka, также выделенный для данного источника. Это позволяет однозначно сопоставлять события с их источником, облегчает управление правами доступа, а также минимизирует риски, связанные с ошибками в маршрутизации или обработке данных. Следовательно, на схеме связи между сущностями изображены как «один к одному»: один источник – один коллектор – один топик.

## Процесс обработки событий

### Источники логов

Источниками событий может выступать сетевое оборудование, программное обеспечение для защиты от вирусов, средства для аудита безопасности операционных систем, системы управления учетными записями и доступом (IdM/IAM – Identity Management System, Identity and Access Management System) и иные корпоративные сервисы, генерирующие технические или аудиторские сообщения.

Источники логов могут передавать сообщения по различным протоколам взаимодействия, таким как TCP (англ. Transmission Control Protocol – протокол управления передачей), UDP (англ. User Datagram Protocol – протокол пользовательских датаграмм), HTTP (англ. Hypertext Transfer Protocol – протокол передачи гипертекста) и другим протоколам более высокого уровня[[1]](#footnote-1).

Некоторые источники логов не способны передавать данные по сети, но занимаются их хранением. Например, система управления проектами Jira[[2]](#footnote-2) сохраняет логи авторизации и других действий пользователей в базе данных приложения, так же делает и корпоративная база знаний Confluence [8]. Данные из таких источников необходимо собирать внешними средствами.

### Коллекторы

**Коллектор** – это сервис, который принимает события от различных источников по сетевым протоколам, а также собирает события из источников, которые не способны их отправлять, и после чего передает их в систему Apache Kafka. То есть основная задача коллектора заключается в перенаправлении данных из источника событий в шину данных. При этом коллектор не изменяет смысл или содержимое события, а только адаптирует транспортный уровень передачи данных, что позволяет централизованно и унифицировано обрабатывать логи в SIEM-системе.

Коллектор должен уметь пропускать через себя большой поток данных в единицу времени, оставаясь при этом легковесным и нересурсозатратным.

### Apache Kafka

**Apache Kafka** представляет собой распределенную платформу потоковой передачи данных, предназначенную для обработки и передачи больших объемов сообщений в режиме реального времени [9]. Как компонент SIEM, Kafka обеспечивает надежную и масштабируемую доставку логов от коллекторов к следующим компонентам системы – парсерам.

Основной логической единицей хранения и передачи данных в Kafka является топик – канал, в который публикуются события. Для оптимизации производительности и масштабируемости каждый топик может быть разбит на несколько партиций, позволяя осуществлять параллельную обработку данных.

В системе Kafka различают два основных типа пользователей: **продьюсеры** (англ. producers), осуществляющие запись данных в топики, и **консьюмеры** (англ. consumers), считывающие сообщения из топиков для дальнейшей обработки. При необходимости одна и та же учетная запись может обладать правами как на запись, так и на чтение данных в одном или нескольких топиках.

Благодаря внедрению Apache Kafka архитектура SIEM получает устойчивость к временным сбоям и перегрузкам, так как события могут накапливаться в Kafka до момента их обработки следующими компонентами.

### Парсеры (нормализаторы)

**Парсер,** или **нормализатор логов,** представляет собой компонент SIEM-системы, осуществляющий преобразование необработанных логов из Apache Kafka в стандартизированный вид, пригодный для дальнейшего анализа и корреляции событий. Поскольку источники логов существенно различаются по формату и структуре данных, задачей парсера является устранение этих различий путем приведения логов к унифицированной структуре с обязательным набором полей.

Например, сетевое оборудование генерирует сообщения с информацией о сетевых портах и IP-адресах в одной форме, тогда как приложения фиксируют данные о действиях пользователей или ошибках в другой. Парсер нормализует такие сообщения, стандартизируя поля, метки времени, уровни критичности и иные атрибуты, необходимые для эффективного анализа и корреляции.

Также парсер может выполнять следующие функции:

* фильтрация логов, направленная на исключение событий, не представляющих интереса для анализа;
* обогащение логов дополнительной технической информацией (например, определение геолокации по IP-адресу источника, получение информации о руководителе сотрудника);
* маркировка событий тегами, облегчающими последующую обработку и корреляционный анализ.

Для реализации процесса обогащения логов в инфраструктуре SIEM применяется специальный компонент – сервис обогащения логов (англ. enrichment service). Данный сервис хранит и предоставляет дополнительную контекстную информацию (например, базы данных геолокации IP-адресов, связи между сотрудниками организации). Парсер при обработке событий осуществляет запросы к сервису обогащения, получая от него необходимые дополнительные сведения и добавляя их в обработанные логи. Таким образом, обогащенные события поступают на вход коррелятора в максимально подготовленном виде, что значительно повышает качество и эффективность последующего анализа инцидентов.

### Система хранения логов

**Система хранения логов (СХЛ)** – это компонент SIEM, предназначенный для долговременного хранения нормализованных, обогащенных и подвергнутых предварительному анализу логов. Основной задачей СХЛ является сохранение и систематизация больших объемов данных с последующей возможностью эффективного поиска, фильтрации и углубленного аналитического исследования.

В качестве СХЛ могут использоваться как проприетарные решения, так и системы с открытым исходным кодом. Одной из самых известных и популярных СХЛ является Elasticsearch[[3]](#footnote-3), предоставляющая возможности полнотекстового поиска, высокую скорость обработки запросов, гибкую фильтрацию данных и горизонтальное масштабирование.

События, поступающие в СХЛ, хранятся в расширенном формате, содержащем дополнительную контекстную информацию, полученную на этапах нормализации и обогащения. Подобный подход позволяет специалистам по информационной безопасности, включая команду реагирования и других сотрудников SOC, оперативно и качественно выполнять расследование инцидентов, осуществлять комплексный анализ событий и принимать обоснованные решения по реагированию.

### Коррелятор

**Коррелятор** – это система, выполняющая анализ нормализованных и обогащенных событий с целью выявления цепочек взаимосвязанных событий, которые в совокупности могут указывать на инцидент информационной безопасности.

В отличие от парсера, который работает с отдельными логами, коррелятор обрабатывает события в совокупности, используя заданные правила или сценарии. Эти правила могут описывать, например:

* последовательность событий от одного пользователя за короткий промежуток времени (вход, изменение прав, попытка подключения по SSH);
* множественные ошибки входа с разных IP-адресов, указывающие на подбор пароля;
* аномальные действия, выходящие за пределы типичной активности для конкретного пользователя или системы.

Корреляторы могут быть основаны как на простых правилах, так и на более сложных алгоритмах машинного обучения или поведенческого анализа. Результатом работы коррелятора являются алерты, которые поступают в SOAR-систему для последующего автоматического реагирования или передачи команде CSIRT для обработки вручную.

Таким образом, коррелятор играет ключевую роль в выявлении инцидентов, трансформируя поток отдельных логов в осмысленные и значимые сценарии потенциальных угроз.

### Процесс обработки событий



Рисунок 2 – Маршрут из систем, которые проходит каждое событие

На рисунке 2 представлен процесс обработки логов. Источники событий направляют данные через коллекторы в Apache Kafka, парсеры приводят логи к единому формату, после чего они сохраняются в системе хранения логов. Коррелятор анализирует события, формируя алерты, а SOAR принимает решение о легитимности события, инициируя реагирование при необходимости.

## Акторы процессов SOC

В описываемых далее процессах участвуют несколько групп специалистов, каждая из которых выполняет определенные функции и взаимодействует с компонентами системы управления информацией и событиями безопасности (SIEM). Ниже описаны ключевые акторы, их роли и задачи в контексте существующего процесса.

* + - * **Системные инженеры** **SOC** отвечают за настройку и эксплуатацию инфраструктуры SIEM и SOAR, включая развертывание и конфигурацию коллекторов, создание топиков в Apache Kafka, настройку балансировщиков и обеспечение отказоустойчивости системы. Одна из их задач – обеспечить бесперебойный прием и передачу логов от источников к системе хранения. Системные инженеры также взаимодействуют с внешними командами, например, для согласования изменений в используемых системах вне зоны их ответственности (к таким системам относятся кластера Kafka, балансировщики нагрузки).
      * **TH-аналитики** (англ. Threat Hunting Analysts, специалисты по поиску угроз) занимаются анализом логов и разработкой сценариев для коррелятора, который выявляет потенциальные инциденты информационной безопасности. В процессе настройки сбора логов они определяют требования к параметрам топиков Kafka (например, название топика и его предназначения, примерный объем данных в топике). Также TH-аналитики разрабатывают конфигурации парсеров.
      * **Команда обслуживания Kafka**, (инженеры Kafka) – команда, управляющая кластерами Apache Kafka, используемыми в SOC. Они отвечают за автоматизацию создания топиков, согласование изменений параметров топиков (например, времени хранения сообщений), а также предоставление учетных записей для доступа к кластеру. В текущем процессе системные инженеры согласовывают с этой командой любые неавтоматические изменения в кластере.
      * **Команда обслуживания балансировщиков** обслуживает платформенное решение для балансировки сетевого трафика, используемое для распределения логов от источников к коллекторам. Члены данной команды согласовывают и применяют изменения в конфигурационных файлах балансировщиков, которые вносят системные инженеры.
      * **Команды, ответственные за источники логов,** управляют системами, генерирующими события. Они участвуют в определении протоколов передачи данных и настраивают источники для отправки логов в коллекторы SOC.

# Анализ процесса настройки сбора логов

Настройка сбора логов предполагает создание цепочки передачи данных от источника логов к СХЛ. Процесс включает получение данных от источника, их нормализацию и передачу в СХЛ для последующего анализа. Настоящий раздел описывает текущий процесс, его этапы, участников и особенности, включая краевые случаи.

## Описание типового процесса настройки

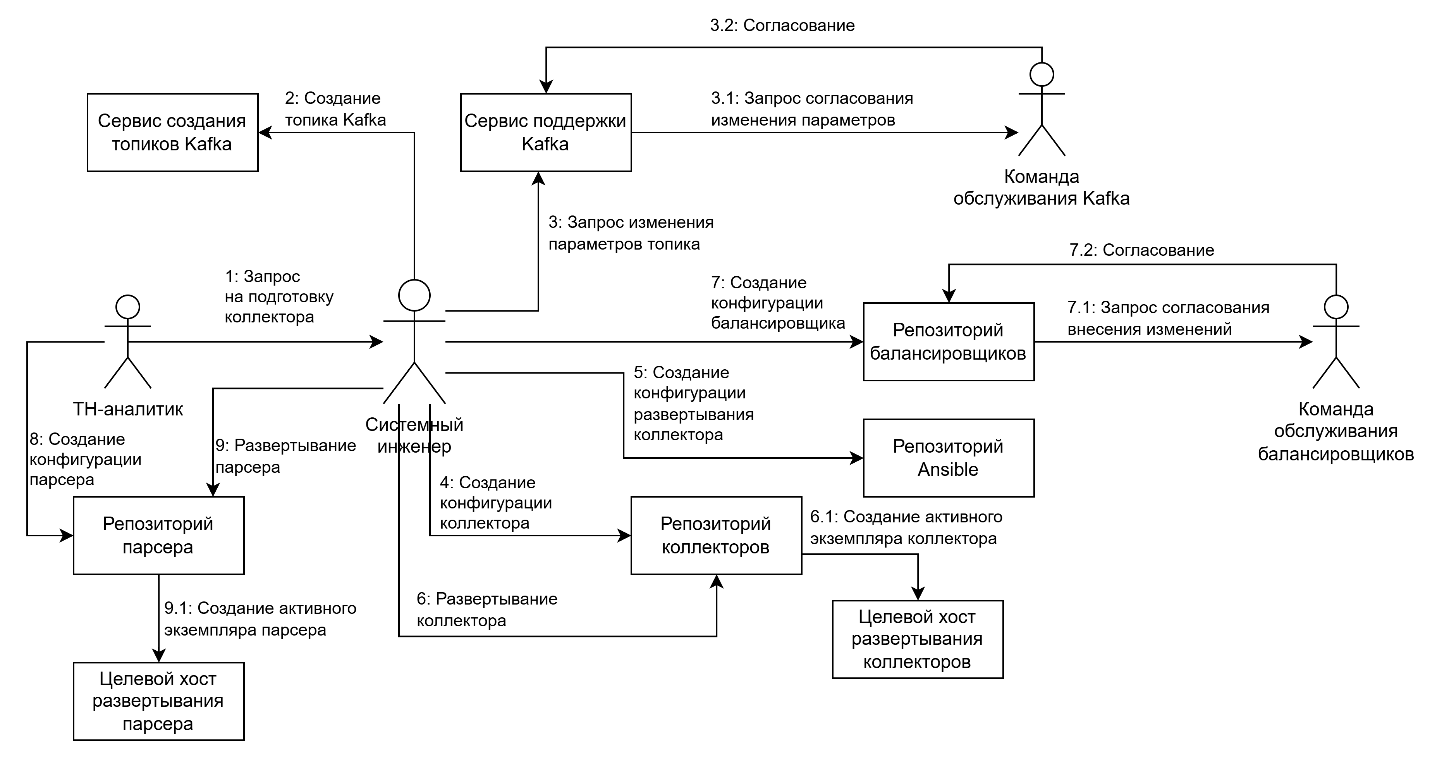


Рисунок 3 – Процесс настройки сбора логов с источника с использованием сетевого коллектора

Типовой процесс настройки сбора логов представлен на рисунке 3. Данный процесс охватывает наиболее распространенный сценарий, при котором источник передает логи в коллектор, используя сетевые протоколы, такие как TCP или основанные на TCP (например, Syslog, HTTP).

Для реализации типового процесса необходимо выполнить следующие этапы: подготовить топик в Apache Kafka (этапы 2 – 3), настроить коллектор (этапы 4 – 7), реализовать логику нормализации событий в парсере и развернуть его (этапы 8 – 9). Эти этапы выполняются с участием различных акторов, описанных в разделе 1.1.2, и включают как ручные, так и частично автоматизированные операции.

### Создание коллектора

Настройка коллектора предполагает создание его конфигурационного файла, который определяет источник входящих данных (поле «входа») и адрес передачи данных (поле «выхода»). На листинге 1 изображен псевдо-конфигурационный файл с настройками, типичными для сетевого коллектора. В секции input указаны настройки интерфейса, принимающего данные, в секции output – настройки адреса назначения событий.

Листинг 1 – Пример конфигурации коллектора, принимающего данные по протоколу TCP



В типовом сценарии коллектор функционирует как сервер, принимающий данные по сети через открытый сетевой порт. Однако данный сценарий применим не ко всем источникам. Процесс настройки сбора логов для источников с нетиповыми протоколами взаимодействия будет рассмотрен в разделе Нетиповые процессы настройки 2.2.

Все сетевые коллекторы развернуты на нескольких виртуальных машинах, размещенных в различных дата-центрах. Для повышения отказоустойчивости и упрощения процессов управления каждый коллектор присутствует на всех выделенных виртуальных машинах с одинаковой конфигурацией. Такой подход обеспечивает гибкость в распределении сетевой нагрузки между узлами, вне зависимости от выбранного механизма маршрутизации.

Поскольку на одной виртуальной машине одновременно функционирует несколько коллекторов, занимаемые сетевые порты среди них должны быть уникальны. На текущий момент выбор порта выполняется вручную: Системный инженер анализирует репозиторий конфигураций коллекторов и определяет минимальный свободный порт, который затем фиксируется в конфигурационном файле. Такой подход обеспечивает упорядоченное распределение портов, однако требует определенных временных затрат.

На «выходе» коллектор всегда передает данные в Apache Kafka. Для передачи данных в топик коллектору требуется учетная запись с правами на запись. В текущем процессе используется единая учетная запись с доступом на запись во все топики, что упрощает настройку, но нарушает принцип наименьших привилегий [10], создавая потенциальные риски безопасности.

Также в конфигурационном файле коллектора необходимо указывать название топика и адреса брокеров Kafka. Название топика определяется TH-аналитиками и указывается в запросе (этап 1 на рисунке 3).

#### Конфигурация балансировщика

Для источников событий не сообщаются адреса виртуальных машин с активными экземплярами коллекторов. Им предоставляется доменное имя, указывающее на адреса балансировщиков нагрузки, которые управляют распределением трафика от источника событий до виртуальных машин коллекторов.

Используемые балансировщики нагрузки в SOC – это платформенное решение, обслуживаемое внешней командой внутри компании. Системные инженеры только описывают конфигурацию балансировщика в соответствующем репозитории, а ее применением на активных экземплярах балансировщиков занимается команда обслуживания.

Листинг 2 – Пример конфигурации балансировщика



На листинге 2 изображен пример псевдо-конфигурации балансировщика с доменным именем *collector.example.com*. В типовом процессе настройки сбора логов при конфигурации балансировщика системный инженер SOC дублирует существующие записи в разделах frontends и backends, заменяя только сетевой порт, а также имена сущностей (например, collector\_antivirus). Для всех источников используется одно и то же доменное имя, поэтому все «фронтенды» и «бэкенды» отличаются только номером порта и названием.

Любые изменения в конфигурации балансировщика подлежат обязательному согласованию с командой обслуживания балансировщиков. Среднее время согласования составляет несколько часов, но в некоторых случаях может достигать суток.

### Создание топика

Используемый кластер Apache Kafka в SOC также управляется внешней командой с экспертизой в Kafka. Системные инженеры SOC имеют учетные записи с правами на чтение из любых топиков, запись в любые топики, а также с правами на создание топиков, однако настройка параметров топиков, создание новых учетных записей и другие административные действия выполняются только через команду обслуживания Kafka. Создание топика осуществляется с использованием веб-сервиса, разработанного данной командой. Процесс включает авторизацию системного инженера в веб-сервисе и заполнение формы с параметрами топика: названием, описанием, размером, количеством партиций и временем хранения сообщений.

На этапе создания топика существуют ограничения на значения некоторых параметров: например, максимальное время хранения сообщений в топике составляет 24 часа. Если требуемые параметры превышают установленные лимиты (например, для долговременного хранения логов необходимо увеличить *retention.ms* – время хранения сообщений), Системный инженер должен сначала создать топик с базовыми параметрами, а затем направить запрос на увеличение значений через сервис поддержки. Данный запрос также требует согласования с командой обслуживания Kafka. Среднее время обработки такого запроса составляет около одного рабочего дня (8 часов), что в контексте выполнения задачи эквивалентно суткам.

### Разработка парсера

В отличие от коллекторов, код парсеров организован в нескольких репозиториях, сгруппированных по типу обрабатываемых логов, например: парсер логов сетевых устройств, парсеров логов операционной системы Windows или парсер логов VPN (англ. Virtual Private Network). Каждый репозиторий может содержать несколько пайплайнов. **Пайплайн парсера** – это сценарий обработки логов, содержащий правила переименования, удаления или добавления полей в лог [11].

Разработка парсера для нового источника предполагает либо создание нового пайплайна в существующем репозитории, либо формирование нового репозитория с соответствующим пайплайном. Данный процесс выполняется TH-аналитиками, которые обладают экспертизой в области требований к данным для корреляционного анализа. Системные инженеры, в свою очередь, отвечают за развертывание парсеров, выделение ресурсов и выполнение операционных задач, таких как мониторинг и устранение сбоев.

## Нетиповые процессы настройки

Типовой процесс, описанный в разделе 2.1, применим не ко всем источникам логов. В данном разделе рассматриваются процессы настройки сбора логов, требующие отклонения от типового сценария.

### Источники, осуществляющие прямую запись в Apache Kafka

Некоторые источники логов поддерживают прямую передачу данных в Apache Kafka, что является предпочтительным вариантом с точки зрения отказоустойчивости и минимизации точек отказа. Kafka представляет собой распределенную систему, обеспечивающую надежную буферизацию данных, и введение дополнительных промежуточных компонентов, таких как коллекторы, может увеличить риски сбоев. В данном случае процесс настройки упрощается. Системные инженеры создают топик в Kafka, запрашивают через сервис поддержки учетную запись с правами на запись в этот топик, после чего передают учетные данные команде, ответственной за источник. В данном сценарии упразднено использование коллектора.

### Источники, не способные отправлять данные через сетевые протоколы.

В разделе 2.1.1 были упомянуты системы Jira и Confluence, которые сохраняют логи исключительно в собственных базах данных. В таких случаях сетевой коллектор, функционирующий как сервер, не применим – необходимо использовать специализированный коллектор, способный считывать события из базы данных и передавать их в Kafka.

Одним из используемых решений является Logstash[[4]](#footnote-4), который поддерживает чтение данных из реляционных баз данных и их последующую передачу в Kafka. Для NoSQL баз данных могут применяться приложения собственной разработки на языках программирования, таких как Python или Ruby.

Такие коллекторы запускаются в одном экземпляре для избежания дублирования данных, а настройкой конфигурации подобных коллекторов занимается команда TH-аналитиков. В данном процессе системный инженер занимается только настройкой топика.

### Процесс настройки сбора логов с источника, отправляющего данные по UDP

Значимая часть сетевого оборудования для отправки логов использует протокол UDP. Для сбора данных с таких источников используются сетевые коллекторы с открытым сетевым портом. Основная проблема данного процесса заключается в сложности балансировки UDP трафика. Протокол UDP не ориентирован на установление соединения и передачу пакетов с подтверждением доставки, что затрудняет или делает невозможным эффективное распределение пакетов между несколькими коллекторами. В отличие от TCP, где балансировка может осуществляться посредством установленных соединений, UDP-трафик не имеет встроенных механизмов отслеживания сессий или потоков данных, поэтому стандартные программные балансировщики трафика (например, HAProxy, Nginx) не поддерживают полноценную балансировку UDP. На текущий момент в компании не существует программных решений для активной балансировки UDP трафика. Поэтому при настройке коллекторов создание балансировщиков не производится.

## Оценка существующего процесса настройки сбора логов.

Текущий процесс настройки сбора логов, описанный в разделах 2.1 и 2.2, характеризуется рядом недостатков, которые снижают его эффективность, увеличивают временные затраты и создают риски для работы Security Operations Center.

### Количественная оценка временных затрат

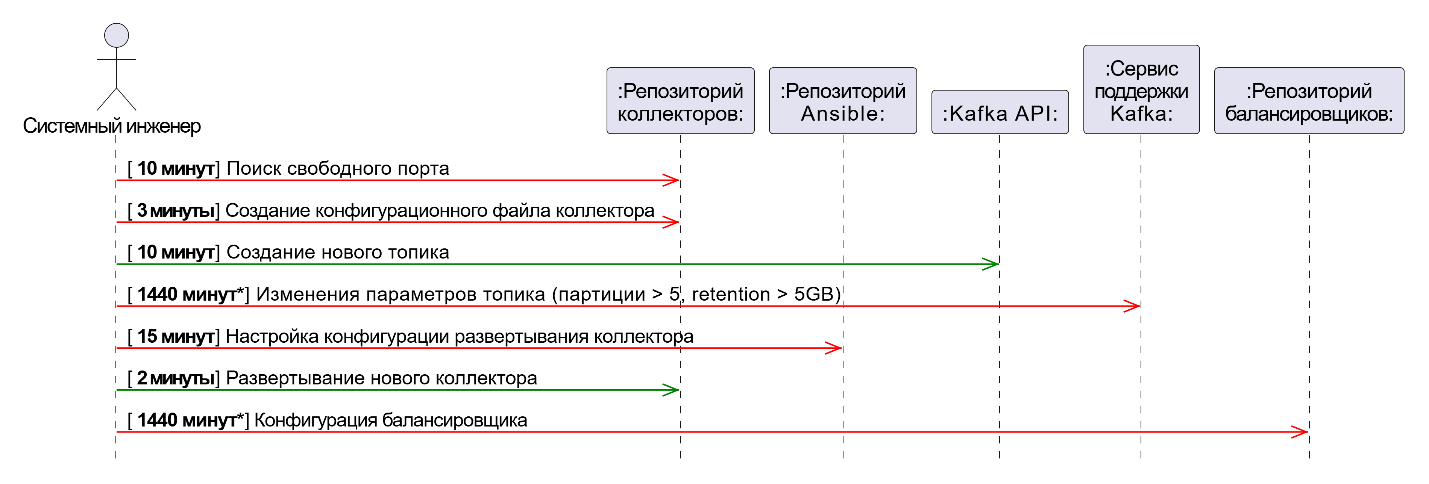


Рисунок 4 – Описание времязатрат для создания коллектора и топика в типовом сценарии настройки сбора логов с источника

Процесс настройки сбора логов из нового источника в типовом сценарии (раздел 2.1) занимает значительное время, что обусловлено большим количеством ручных операций и необходимостью согласования с внешними командами. На рисунке 4 указаны времязатраты системного инженера на процесс сбора логов. Красным цветом выделены этапы, выполняемые вручную, зеленым – частично автоматизированные.

Общие временные затраты системного инженера на настройку сбора логов с одного источника в типовом сценарии составляют от 24 до 72 часов. В среднем, 48 часов занимают этапы, связанные с согласованием с внешними командами. Это подтверждает, что процесс является трудоемким и требует вовлеченности системных инженеров в процесс в течение длительного времени.

### Качественная оценка недостатков

Помимо высоких временных затрат, текущий процесс обладает рядом качественных недостатков, которые влияют на его эффективность, безопасность и масштабируемость. Ниже приведены основные проблемы, выявленные в ходе анализа.

#### Высокая доля ручных операций

Многие этапы процесса, такие как поиск свободного порта, создание конфигурационных файлов коллектора и внесение изменений в различные репозитории выполняются вручную. Это приводит к следующим проблемам.

* Снижение производственной эффективности системного инженера. Выполнение рутинных операций, требующих постоянного участия, отвлекает специалиста от более приоритетных задач, таких как разработка автоматизированных решений, улучшение уровня технической поддержки и наставничество для менее опытных сотрудников [12].
* Повышенная вероятность ошибок, обусловленных человеческим фактором. В частности, некорректный выбор порта, уже используемого другим коллектором, может привести к конфигурационным конфликтам, сбоям в логировании или нерациональному увеличению времени на выполнение задачи.

#### Нарушение принципа наименьших привилегий

В текущем процессе для всех коллекторов используется единая учетная запись (УЗ) с правами на запись во все топики Apache Kafka. Такой подход упрощает настройку, однако создает значительный риск безопасности – риск утечки учетной записи.

Этот риск влечет за собой предоставление злоумышленнику возможности записывать произвольные данные во все топики, что может привести к перегрузке парсеров, потенциальной потере значимых событий и нарушению целостности данных.

Данный недостаток особенно критичен в контексте SOC. Потеря событий или их искажение может привести к незамеченной нелегитимной активности, угрожающей безопасности инфраструктуры, а также к репутационным и юридическим рискам, связанным с невыполнением требований внутренней политики ИБ или внешнего законодательства

#### Отсутствие централизованного управления конфигурациями

Конфигурации коллекторов, балансировщиков и топиков производится в нескольких инструментах, что усложняет их управление и инвентаризацию. Отсутствие единого «источника правды» приводит к следующим проблемам.

* Инвентаризация источников логов ведется вручную параллельно процессу настройки, что увеличивает вероятность ее несоответствий или недосказанности.
* Скорость реагирования на проблемы, связанными с коллекторами снижается из-за необходимости поиска информации о конфигурации коллектора в различных источниках.
* Скорость настройки сбора логов из источника может быть снижена из-за человеческого фактора – инженер может забыть выполнить вовремя один из этапов процесса.

#### Зависимость от внешних команд

Процесс настройки включает этапы, требующие согласования с командами обслуживания Kafka и балансировщиков. Данная зависимость приводит к неконтролируемым временным задержкам, что увеличивает общую продолжительность процесса и затрудняет срочную настройку новых источников логов при необходимости.

#### Отсутствие процесса удаления неактуальных сущностей

Процесс настройки сбора логов должен сопровождаться процессом выведения из эксплуатации неиспользуемых сущностей. На текущий момент в случае ненадобности удаляются только активные экземпляры коллекторов, в то время как конфигурации балансировщиков, топики и пользователи остаются нетронутыми. Такой подход порождает избыточный технический долг.

## Выводы по анализу

Проведенный анализ выявил, что текущий процесс настройки приема логов характеризуется высокими временными затратами, значительной долей ручных операций и зависимостью от внешних команд. Качественные недостатки, такие как нарушение принципа наименьших привилегий и отсутствие централизованного управления, создают риски для безопасности и отказоустойчивости системы. В совокупности эти проблемы снижают эффективность работы SOC, особенно в условиях роста числа источников логов. Процесс может быть упрощен и автоматизирован.

# Изменения во взаимодействии с Kafka

В разделе 2.1 говорится о том, что существующий кластер Kafka администрируется инженерами команды обслуживания, поэтому ряд действий с сущностями Kafka требует согласования данной команды. На рисунке 5 изображен процесс взаимодействия с существующим кластером. В качестве примера рассматривается одна из типовых задач при работе с Kafka – создание пользователей в кластере, но такой же процесс характерен для следующих задач:

* управления существующими топиками в кластере;
* изменения прав доступа пользователей;
* удаления топиков и пользователей.

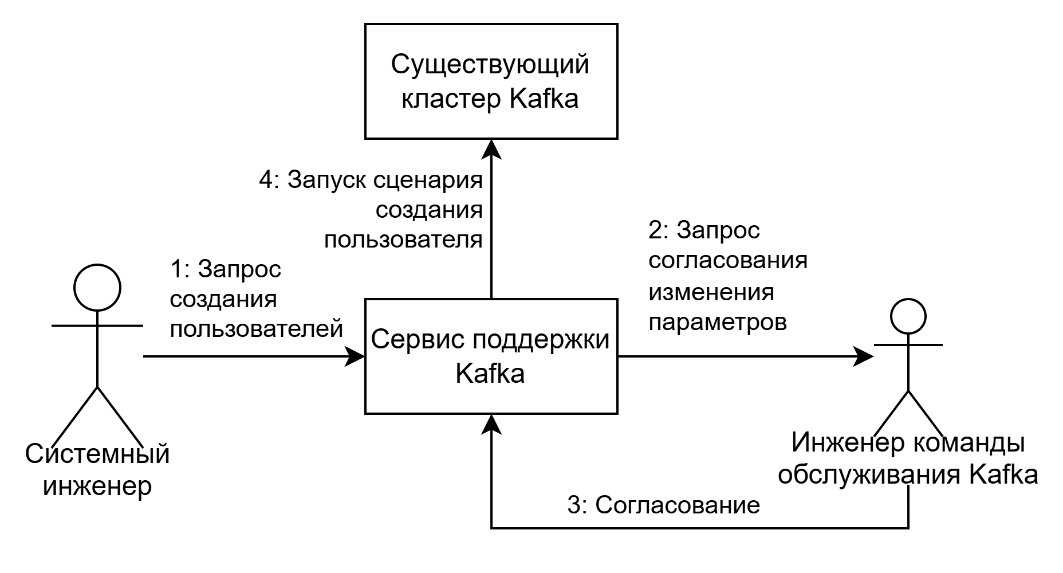


Рисунок 5 – Процесс создания пользователей в существующем кластере Kafka

Существенным недостатком текущего процесса является необходимость получения предварительного согласования с внешней командой для выполнения типовых операций в кластере. Это приводит к увеличению общего времени исполнения процедур и снижает оперативность реагирования.

Действия в кластере Kafka могут выполняться исключительно от имени пользователя, обладающего соответствующими привилегиями. К примеру, выполнение сценария создания нового пользователя (действие 4 на рисунке 5) требует наличия прав на администрирование. Следовательно, при создании отдельного сервисного пользователя с ограниченными, но достаточными правами для выполнения типовых задач, становится возможным исключить необходимость постоянного взаимодействия с командой сопровождения Kafka. Это позволит ускорить выполнение операций и повысить автономность процесса.

## Использование привилегированной сервисной учетной записи.

По результатам анализа рисков и ограничений, связанных с эксплуатацией такого подхода, было принято решение отказаться от его реализации. Основные причины отказа заключались в следующем.

### Невозможность документирования сущностей

Помимо управления сущностями, сервис поддержки Kafka выполняет их автоматическое документирование (рис. 6). Для каждого топика и пользователя кластера Kafka в базе знаний компании создается отдельная страница, на которой отражаются параметры, права доступа (разрешения), описание сущности и другие полезные данные.

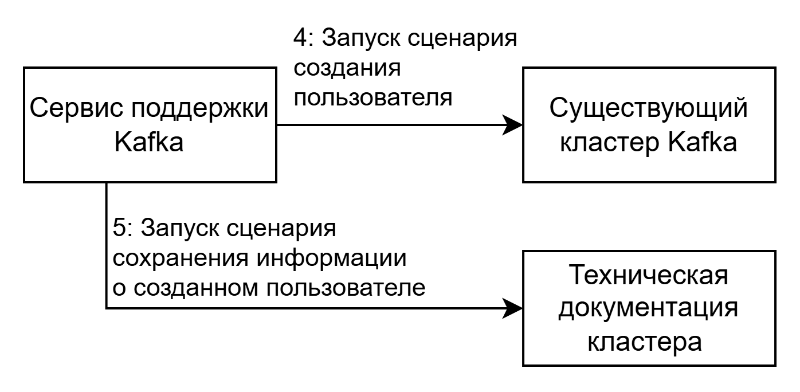


Рисунок 6 – Действия, выполняемые сервисом поддержки

Управление сущностями в обход сервиса поддержки Kafka влечет за собой устаревание технической документации, что является недопустимым.

### Статус устаревшего кластера

Более критичным основанием для отказа от дальнейшей реализации является текущий статус кластера — End-of-Life (EoL, «конец жизненного цикла»). Присвоение данного статуса свидетельствует о том, что кластер Kafka признан устаревшим, и любые действия, связанные с внедрением автоматизации, обновлением программного обеспечения или повышением уровня обслуживания, нецелесообразны с точки зрения долгосрочной эксплуатации.

Несмотря на сохраняющуюся техническую возможность создания новых топиков и пользователей, предполагается, что в обозримом будущем все объекты и сервисы, работающие в данном кластере, будут полностью мигрированы на более современную и поддерживаемую платформу.

## Миграция в Kafka as a Service

**Kafka as a Service (KaaS)** – это управляемое облачное решение на базе Apache Kafka, предоставляемое внутри компании. На текущий момент KaaS является единственным рекомендуемым сервисом для использования Kafka в компании. Принципиальная разница между существующим кластером и KaaS заключается в модели управления.

* Существующий кластер Kafka развернут на физических мощностях отдела SOC, за управление ресурсами в кластере отвечает команда системных инженеров SOC, но управлением и конфигурацией кластера занимаются инженеры команды обслуживания Kafka. Для задачи создания топика без согласований командой обслуживания настроен веб-сервис, остальные действия выполняются только через сервис поддержки Kafka.
* В модели KaaS клиентам (командам внутри компании) предоставляются квоты на различные сущности, например, количество топиков и пользователей, общий размер кластера (в байтах), общая сетевая пропускная способность (в байтах/сек). Распределение данных квот выполняется клиентом при создании различных сущностей через веб-интерфейс.

Ключевым преимуществом KaaS является возможность управления топиками, пользователями и их разрешениями без необходимости согласований через веб-интерфейс или API. Это позволяет автоматизировать процессы создания и настройки данных сущностей.

### Описание процесса настройки топика и пользователей Kafka в KaaS

Процесс создания сущностей Kafka в KaaS осуществляется через веб-интерфейс и состоит из нескольких шагов:

1. **Выбор кластера** (рисунок 7). На данном этапе пользователь выбирает кластер Kafka и рабочее окружение, в рамках которого будет создан новый топик.
2. **Заполнение основной информации** (рисунок 8). Пользователь вводит название и описание топика. Название должно содержать обязательный префикс, определяемый политикой KaaS, а описание должно кратко характеризовать назначение создаваемого топика. На этом этапе также задаётся политика очистки сообщений (например, *delete*).
3. **Настройка дополнительных параметров** (рисунок 9). Здесь задаются параметры производительности и хранения данных: количество партиций (*partitions*), время хранения сообщений (*retention.ms*), максимально допустимый размер топика (*retention.bytes*), тип временных меток сообщений (*message.timestamp.type*) и фактор репликации (*replication.factor*).

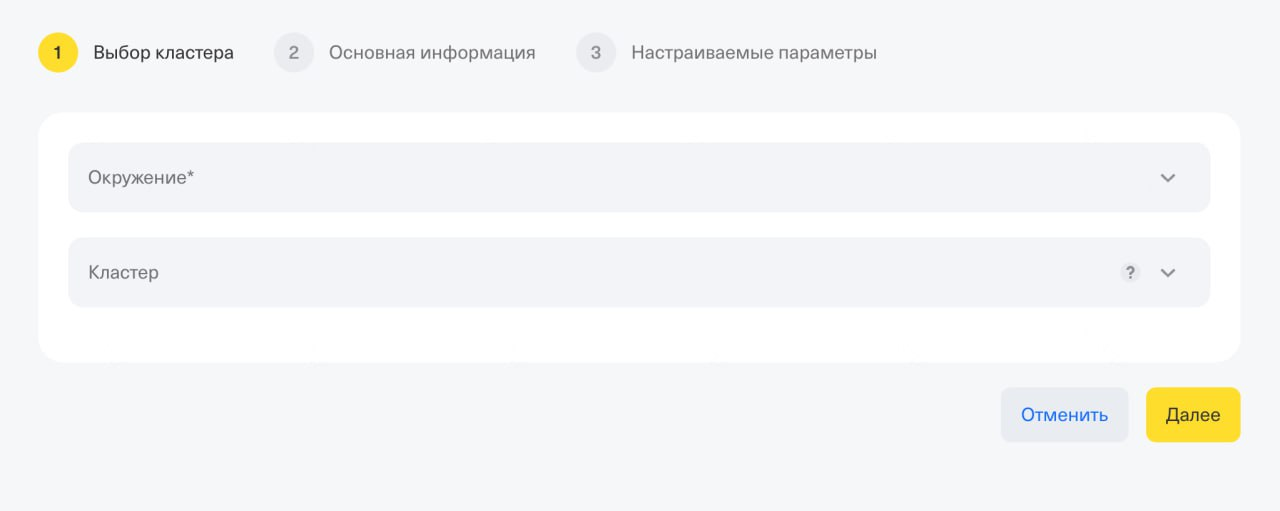


Рисунок 7 – Процесс создания топика в KaaS: выбор окружения и кластера

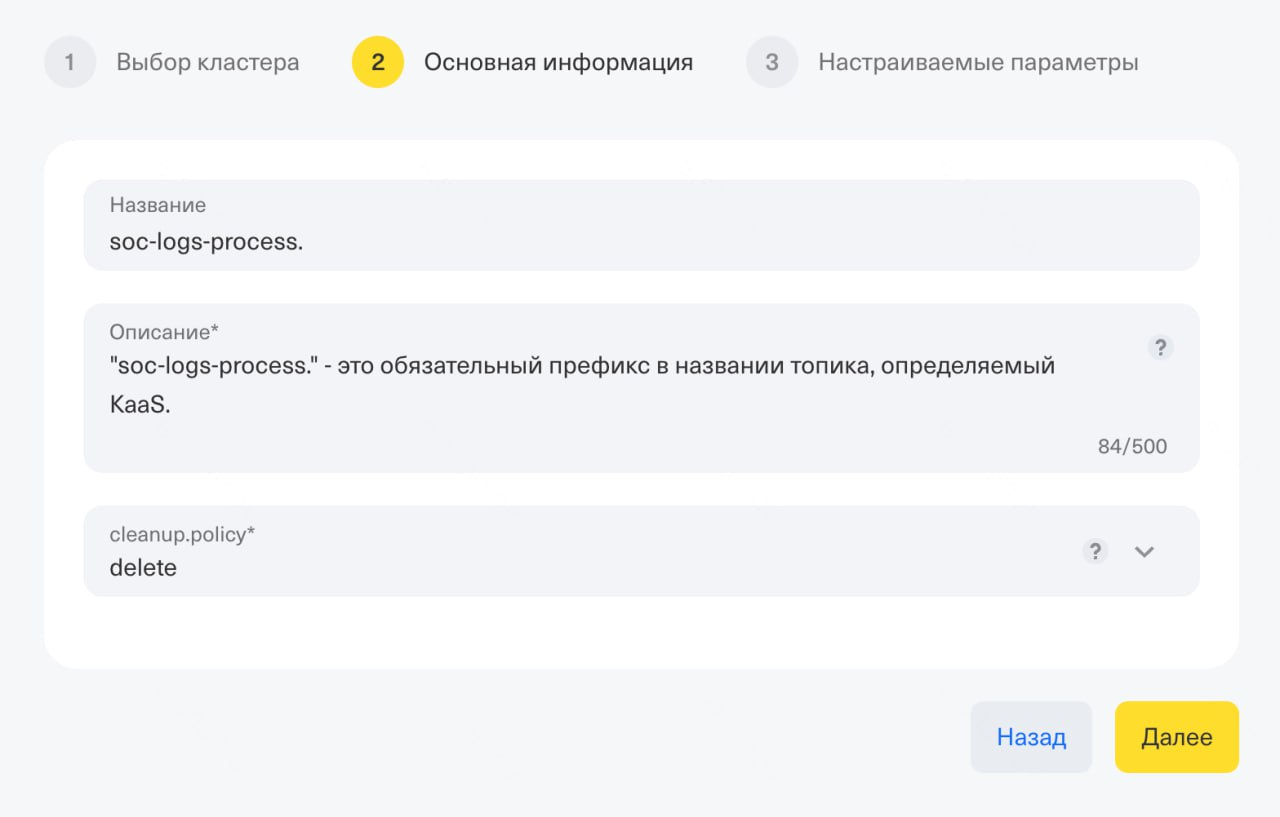


Рисунок 8 – Процесс создания топика в KaaS: заполнение названия и описания

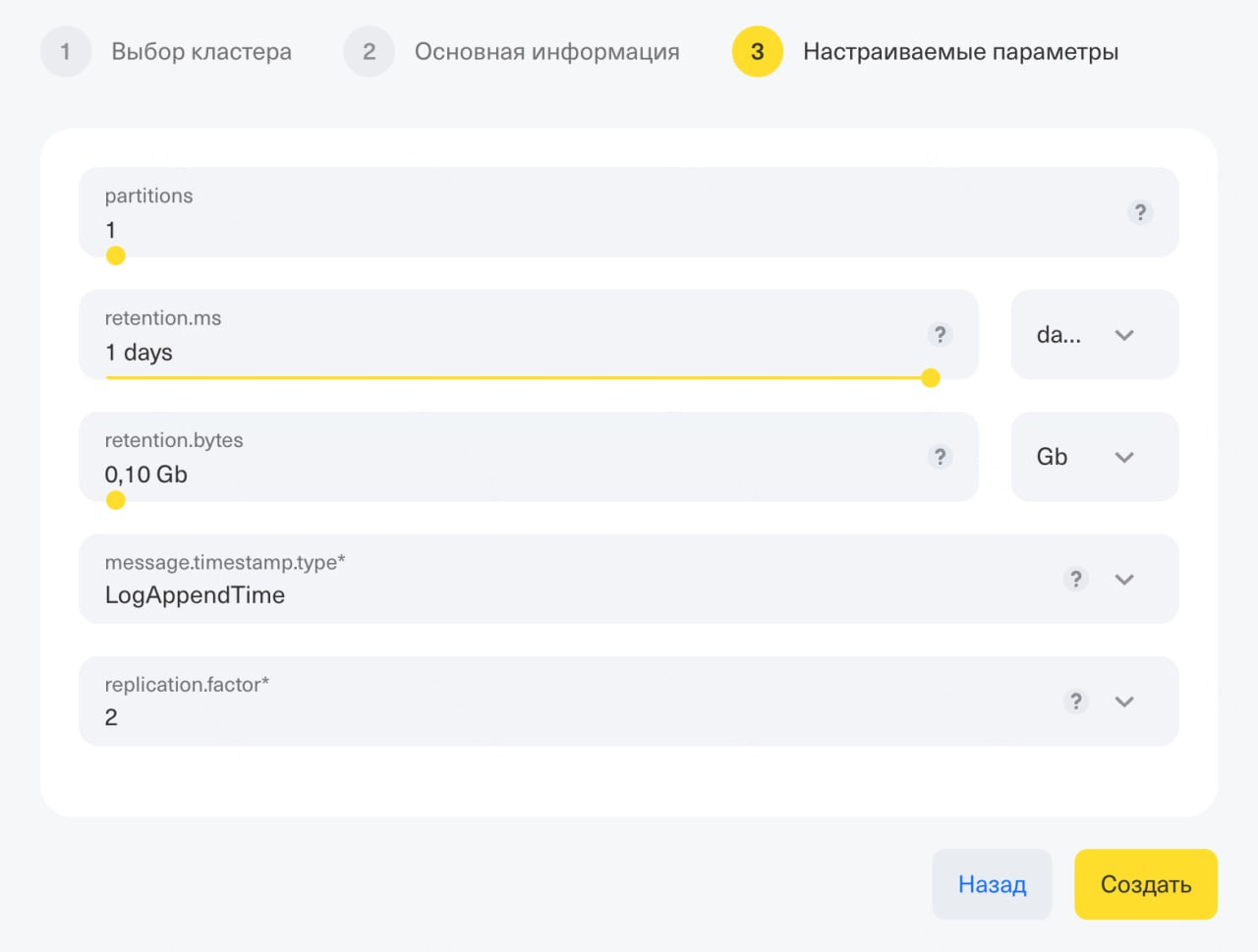


Рисунок 9 – Процесс создания топика в KaaS: настройка параметров топика

Подобный подход с использованием веб-интерфейса удобен при разовом создании топиков или настройке небольшого числа сущностей. Однако такой метод не подходит для массового или частого создания новых топиков, поскольку ограничивает возможности полной автоматизации процесса. В связи с этим для реализации предложенной автоматизации предпочтительнее использовать API KaaS, который позволяет динамически создавать и конфигурировать сущности Kafka на основе декларативных описаний в конфигурационных файлах

## Вывод

На основе полученной информации можно сделать вывод о необходимости использования KaaS для решения упомянутой проблемы. Это значит, что управление сущностями Kafka будет автоматизировано через проприетарный API KaaS. Использование KaaS позволяет решить проблемы, обозначенные в разделах 2.3.2.4 (Зависимость от внешних команд) и 2.3.2.2 (Нарушение принципа наименьших привилегий).

Переход на KaaS будет осуществлен по принципу «естественной миграции» (англ. rolling migration): новая инфраструктура и все вновь подключаемые источники логов сразу настраиваются с использованием KaaS и автоматизированного сценария, в то время как уже существующие источники продолжают функционировать в прежнем кластере до момента их планового перевода. Такой подход позволяет избежать необходимости организации отдельного проекта по миграции, снизить операционные риски и обеспечить постепенное снижение нагрузки на устаревший кластер.

Со временем, по мере вывода из эксплуатации устаревших сущностей и появления новых требований, все процессы и объекты инфраструктуры будут переведены на автоматизированное управление с использованием KaaS, что обеспечит унификацию, снижение трудозатрат и повышение безопасности операций.

# Изменение процесса конфигурации балансировщиков

Процесс подготовки балансировщика (рисунок 10) требует согласований, поскольку подразумевает внесение изменений в репозиторий, находящийся в зоне ответственности внешней команды.

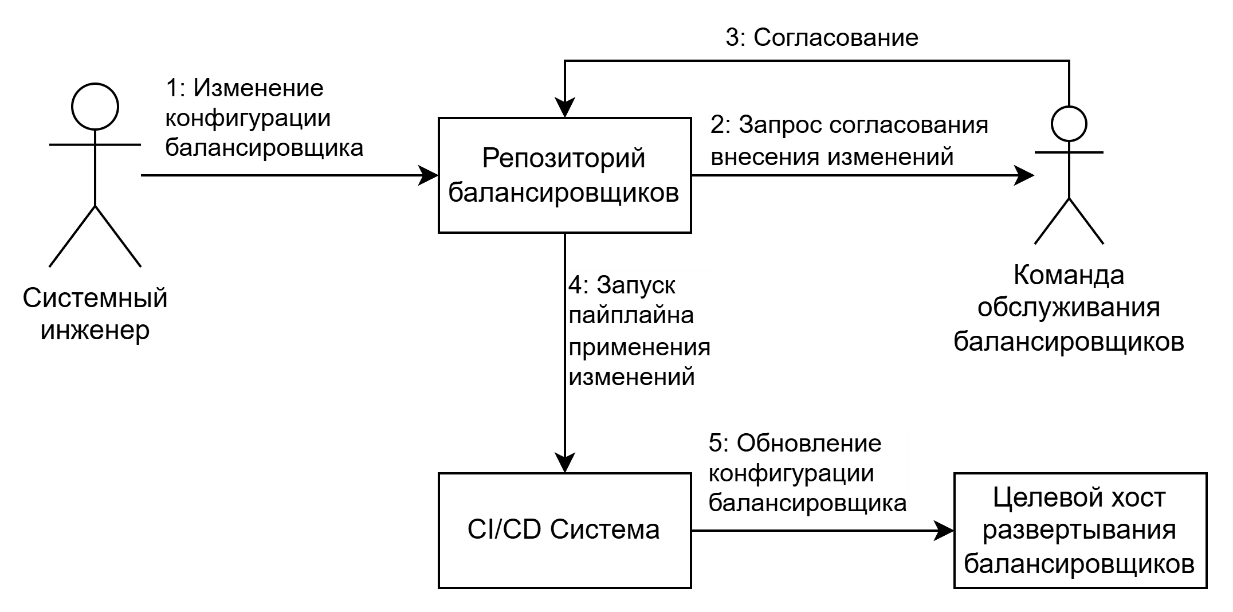


Рисунок 10 – Процесс изменения конфигурации балансировщика

После внесения изменений в основную ветку репозитория, пайплайн в CI/CD системе автоматически разворачивает балансировщики с необходимыми конфигурациями. За активное состояние балансировщиков, их процесс развертывания и другие автоматизации в этом репозитории ответственна команда обслуживания балансировщиков.

Прямое внесение изменений (то есть внесение изменений без согласования) доступно только команде обслуживания балансировщиков и сервисным учетным записям данной команды. Все остальные изменения вносятся через механизм форков[[5]](#footnote-5) (англ. fork, ответвление) репозитория. Данный механизм подразумевает, что репозиторий будет скопирован в личное пространство пользователя в системе контроля версий, но пользователь будет иметь возможность предлагать внесение изменений в исходный репозиторий. Любое такое предложение требует согласования.

Полная автоматизация данного процесса, исключающая подпроцесс согласования, требует использования сервисной учетной записи с возможностью внесения прямых изменений в репозиторий. Однако, автоматизированные сценарии, выполняемые от имени такой учетной записи, должны быть тщательно протестированы и одобрены инженером с квалификацией в управлении балансировщиками. Получение подобной квалификации для решения данной проблемы – это дорогой подход, так как требует времени и человеческого ресурса.

## Варианты решения проблемы

По правилам системы контроля версий, используемой в компании, создание форков возможно только от личной учетной записи, сервисные учетные записи не могут иметь личных репозиториев. В таких условиях, полная автоматизация задачи внесения изменений возможна только в случае владения сервисной учетной записью для внесения изменений в репозиторий без необходимости создания форка.

Один из вариантов решения проблемы – использовать учетную запись пользователя, запускающего механизм автоматизации. То есть создание форка репозитория и запроса на внесение изменений в основной репозиторий может выполняться от имени реального пользователя. Однако такой подход не способен обеспечить полную автоматизацию процесса и потребует ручных действий на одном из шагов (например, передача токена доступа для авторизации или запуск автоматизации от своего имени на локальной машине).

Как еще одну альтернативу можно рассмотреть создание собственной выделенной инфраструктуры балансировщиков, что даст полный контроль над процессом конфигурации балансировщика, но также повлечет за собой необходимость поддержки, развития экспертизы команды в данной области, а также потребление дополнительных ресурсов, на которых будут размещены данные балансировщики. Данный вариант экономически невыгоден для решения обозначенной проблемы.

Выбранный вариант решения проблемы – создать необходимые конфигурации балансировщиков вручную заранее. Это значит, что вместо автоматической настройки балансировщика при создании или изменении коллектора несколько балансировщиков будут заранее сконфигурированы как готовые к использованию.

На листинге 3 представлен пример заранее сгенерированной конфигурации. Каждое название «фронтенда» и «бэкенда» создано на основе номера управляемого сетевого порта. Возможно создать несколько десятков или сотен «фронтендов» и «бэкендов», подобных представленным, и исключить необходимость согласования изменений на долгое время. Недостаток такого подхода в том, что в конфигурации балансировщика не будет известно, трафик какого источника обрабатывает балансировщик. Например, вместо ранее понятного обозначения, такого как collector\_antivirus, теперь используется абстрактное имя вида collector\_tcp5001, что затрудняет последующий анализ и сопровождение. Также важно учесть, что подход подразумевает создание объемного артефакта (конфигурации балансировщика) в ручном или полуавтоматическом режиме.

Листинг 3 – Пример файла конфигурации балансировщика со сгенерированными полями.



### Вывод

Генерация конфигурации балансировщика с обезличенными сущностями позволяет избежать необходимости согласования при развертывании очередного коллектора, что позволяет сократить время выполнения всего процесса.

# Проектирование автоматизации

Настоящая глава описывает переход от перечня требований к конкретной архитектуре подсистемы автоматического подключения источников журналов. Цель проектирования – сформировать решение, удовлетворяющее четырем ключевым критериям:

* **воспроизводимость** –  состояние инфраструктуры однозначно детерминируется содержимым репозитория;
* **минимизация ручных действий**  –  участие инженера ограничено коммитом декларативного описания;
* **безопасность** –  отсутствие привилегированных учетных записей общего назначения, хранение секретов исключительно в Vault[[6]](#footnote-6);
* **масштабируемость** –  линейный рост времени развертывания при увеличении числа источников.

## Единый «источник правды»

От выбираемого инструмента, используемого в качестве единого «источника правды», будет зависеть последующий выбор инструментов автоматизации. На рисунке 11 представлены возможные варианты связок сервиса автоматизации и источника для хранения состояния.

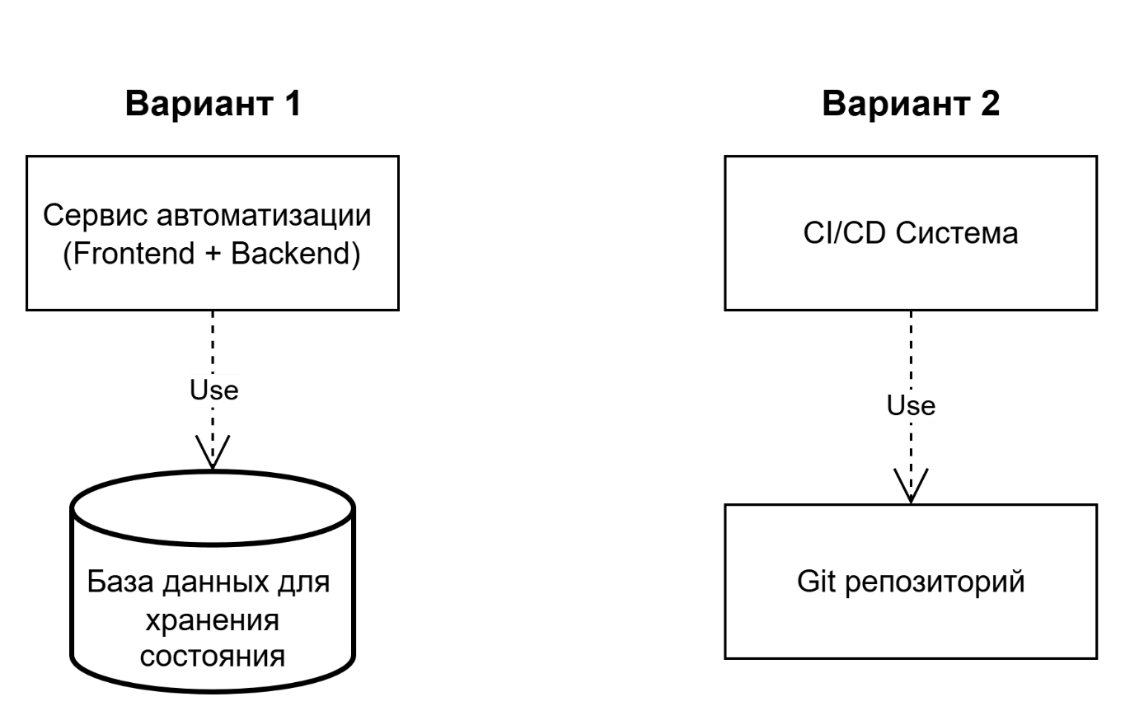


Рисунок 11 – Варианты подходов автоматизации

Первый вариант предусматривает создание сервиса автоматизации с графическим интерфейсом и базой данных для хранения состояния. Второй – предлагает использовать Git репозиторий для хранения состояния и CI/CD систему в качестве инструмента автоматизации.

Стоимость реализации и поддержки второго варианта значительно ниже, поскольку он:

* не требует создания графического интерфейса;
* предоставляет высокие гарантии сохранности данных по умолчанию;
* требует для реализации именно те навыки, которыми обладают системные инженеры.

Так как одно из ключевых требований – возможность аварийного восстановления, описание состояния желаемых сущностей является критически важным элементом системы. Хранение состояния в децентрализованной системе контроля версий не только предоставляет высокие гарантии сохранности, но еще и не требует вычислительных ресурсов, в отличие от активных экземпляров баз данных.

### Структура хранилища состояния

Использование Git репозитория в качестве хранилища состояния подразумевает, что описания необходимых сущностей будут описаны в файлах конфигурации внутри репозитория [13]. К структуризации этих файлов можно подойти различными способами.

Исходя из требований, проектируемая автоматизация должна управлять такими сущностями как коллектор, топик, пользователь и разрешение пользователя. На основе этого можно предположить следующие варианты группировки конфигурационных файлов.

#### По типу сущности

Данный метод подразумевает создание независимых подкаталогов для каждой сущности (листинг 4).

Листинг 4 – Группирование конфигурационных файлов по типу сущности



Такой подход удобен, когда существуют независимые процессы управления сущностями, или когда сущности могут быть не связаны между собой. Однако в такой структуре сложнее ориентироваться, если необходимо управлять конфигурациями нескольких сущностей, относящихся к одному источнику. Также стоит отметить, что создание или удаление всех сущностей, связанных с конкретным источником, потребует внесения изменений в трех разных директориях, что может быть неудобным при большом количестве сущностей в каталогах.

#### По источнику данных

Группировка по источнику данных (листинг 5) позволяет удобно описывать все сущности, связанные с конкретным источником, в одном месте.

Листинг 5 – Группирование конфигурационных файлов по источнику данных



Например, создание необходимых сущностей для нового источника производится через создание новой директории (или копирование и переименование уже существующей). Удаление сущностей ненужного источника происходит также через одно изменение – удаление директории. Данный подход может быть неудобен при наличии зависимости источников друг от друга, но в данном случае это неприменимо, так как источники данных для SOC независимы друг от друга.

Так как данная группировка требует меньшего количества ручных действий в сравнении с группировкой по типу сущности, предлагается использовать этот вариант для хранения состояния в репозитории.

## Организация пайплайна автоматизации

В качестве механизма автоматизации предлагается использовать пайплайн в CI/CD системе, который будет выполнять идемпотентные[[7]](#footnote-7) действия, ориентируясь на ожидаемое состояние системы, описанное в Git репозитории.

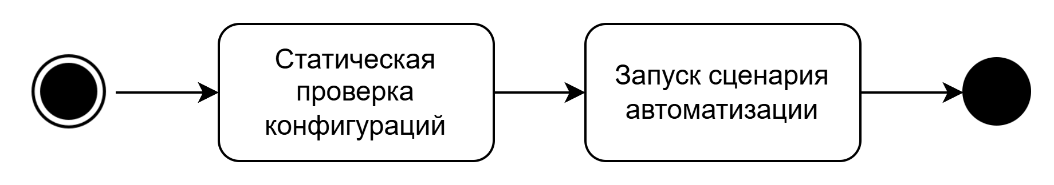


Рисунок 12 – Предлагаемый пайплайн автоматизации

На рисунке 12 изображен предлагаемый вариант CI/CD пайплайна, подразумевающий два шага: статическую проверку конфигураций и запуск сценария автоматизации. Первый шаг занимается проверкой синтаксиса конфигурационных файлов и валидности значений указываемых параметров. А второй – выполняет «бизнес-логику» используя специально разработанную программу (сценарий автоматизации).

### Архитектура сценария автоматизации

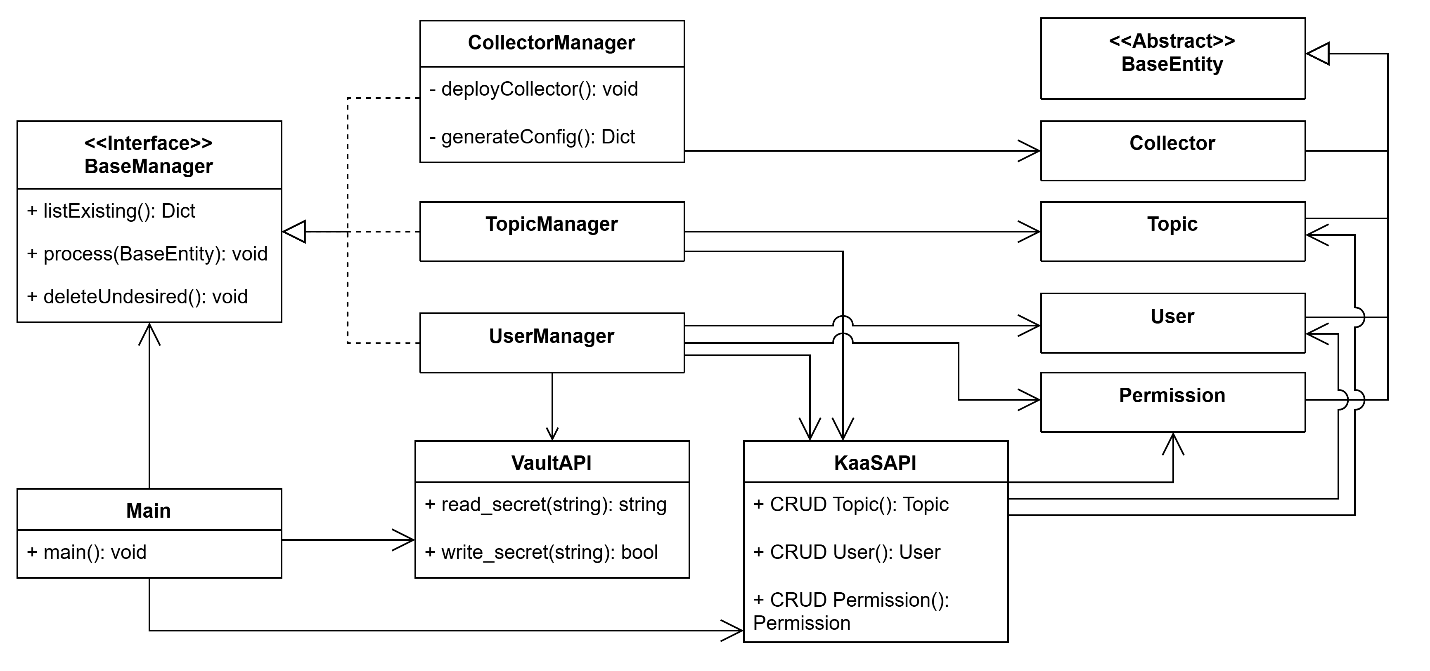


Рисунок 13 – Диаграмма классов сценария автоматизации

На рисунке 13 отражены основные элементы сценария автоматизации. Он реализует бизнес-логику на языке Python, обеспечивая обработку декларативно описанных конфигураций для управления следующими сущностями инфраструктуры: коллекторы, топики, пользователи и разрешения для пользователей. Структура сценария спроектирована с учетом принципов модульности и расширяемости: каждая сущность управляется собственным менеджером (CollectorManager, TopicManager, UserManager), что позволяет легко добавлять и изменять поведение системы в будущем.

Взаимодействие с внешними системами, такими как HashiCorp Vault и KaaS будет реализовано через создание интеграционных классов KaaSAPI и VaultAPI, что позволит переиспользовать функционал данных классов в разных модулях (например, в UserManager и TopicManager).

Важно отметить, что на диаграмме классов представлена условная архитектура классов и интерфейсов, характерная для строго типизированных языков. Реализация проекта на языке Python, обладающем динамической типизацией, вносит некоторые особенности в использование данного проектного решения. В частности, интерфейс BaseManager не реализуется явно через классический механизм интерфейсов и наследования, а используется неявно, посредством соблюдения структуры классов (так называемая "утиная типизация", от англ. duck typing) [14]. Это означает, что соблюдение формальных контрактов определяется наличием методов, а не явным указанием реализации интерфейса. Также динамическая типизация языка формировать сущности (например, классов Topic или Collector) непосредственно из словарей данных (структуры Dict), получаемых в результате парсинга файлов конфигураций.

### Описание работы сценария автоматизации

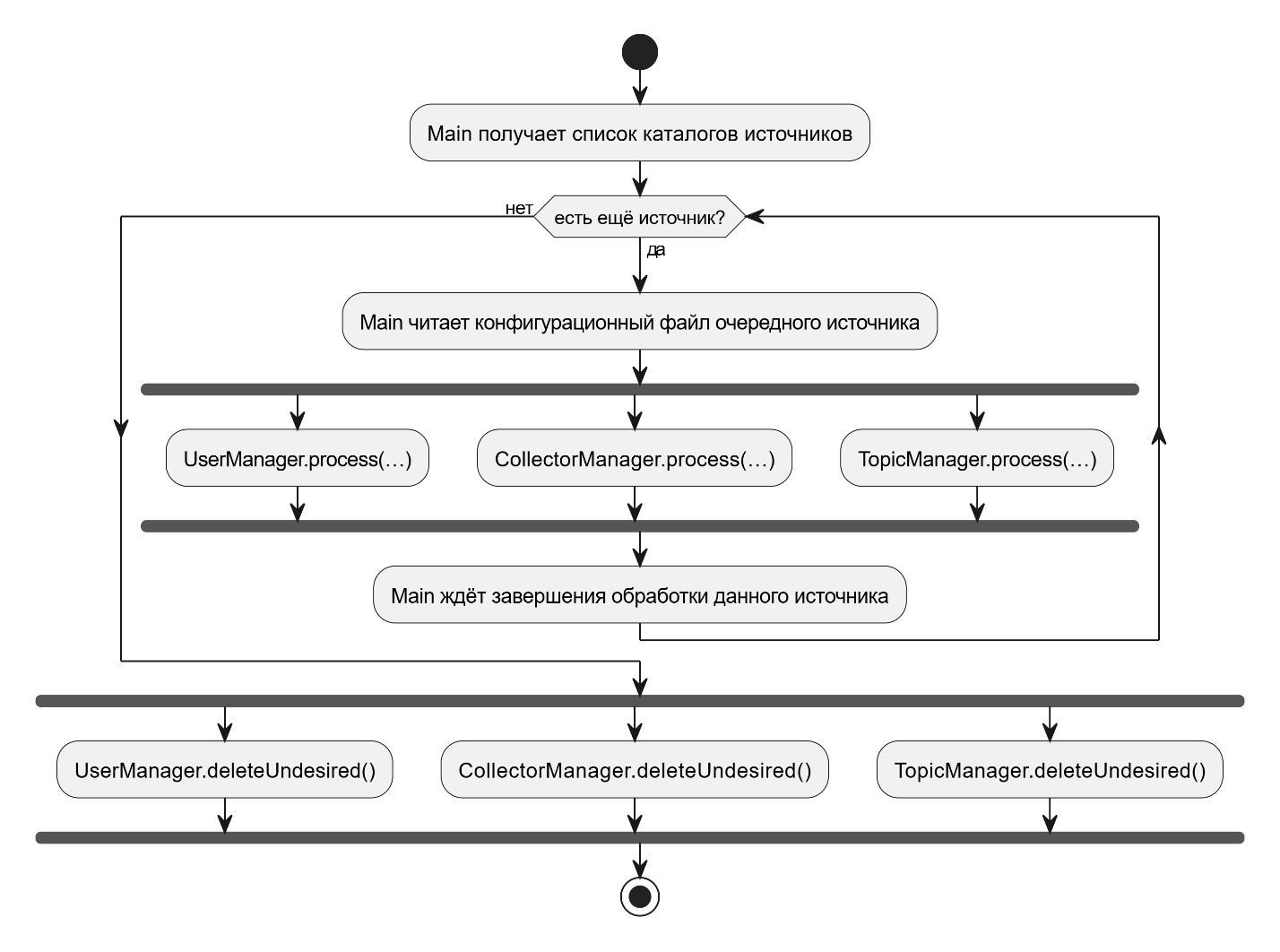


Рисунок 14 – Логика работы сценария автоматизации

На рисунке 14 представлен высокоуровневое описание алгоритма работы сценария автоматизации. Предполагается, что для каждой сущности, описанной в каталоге источника будет вызван метод process() у соответствующего менеджера, который определит, существует ли сущность, и если существует, соответствует ли она описанию в файле. В случае несоответствия состояния «менеджер» создаст или изменит требуемую сущность. После завершения итерации по каталогам источников «менеджерам» будет дан сигнал о необходимости удалить сущности из фактического состояния, если их они не описаны в репозитории.

## Оптимизация процесса выбора свободного порта для сетевых коллекторов

Одной из технических задач, возникающих при создании нового сетевого коллектора, является выбор уникального свободного порта, на котором данный коллектор будет принимать входящий трафик. Корректное назначение порта необходимо для предотвращения конфликтов между сервисами на одном и том же хосте и обеспечения стабильной маршрутизации данных от источника.

В рамках проектирования целевого процесса были рассмотрены следующие варианты решения задачи автоматизации поиска свободного порта.

### Автоматическое назначение порта на этапе создания коллектора

Данный подход предполагает, что система конфигурации самостоятельно определяет минимально свободный порт из допустимого диапазона и вставляет его в конфигурацию коллектора при генерации. Преимущества такого подхода:

* полностью устраняет необходимость участия инженера в выборе порта;
* минимизирует вероятность ошибок, связанных с ручным назначением занятых портов.

Недостатки:

* при пересоздании инфраструктуры (например, при восстановлении из Git-репозитория) невозможно гарантировать повторное назначение того же самого порта, что нарушает воспроизводимость конфигурации;
* возможны рассогласования между настройками источника и конфигурацией коллектора, если порт будет изменен без уведомления заинтересованных сторон.

С учетом критичности сохранения идентичного состояния инфраструктуры при повторном развертывании, данный подход был признан неприемлемым.

### Предоставление рекомендаций по свободному порту

В рамках этого варианта предполагается, что где-нибудь указывается подсказка об актуальном свободном порту из допустимого диапазона. Например, в документации сервиса, файле README.md (титульная страница репозитория) или на дашборде с метриками коллекторов.

Преимущества:

* сохраняется контроль инженера над выбором порта;
* достигается частичная автоматизация, не нарушающая воспроизводимость.

При этом, главный недостаток данного решения в том, что это не автоматизация ручного действия, а лишь ускорение его выполнения. При этом данный подход можно считать допустимым, так как он позволяет достичь баланса между стабильностью конфигураций и снижением временных затрат на ручной поиск.

Таким образом, для решения задачи оптимизации выбора свободного порта в целевом процессе рекомендуется использовать этот вариант, так как он сохраняет прозрачность и воспроизводимость конфигураций, а также ускоряет процесс поиска свободного порта.

## Описание выбранных инструментов

В процессе проектирования сценария автоматизации были приняты архитектурные решения относительно формата конфигурационных файлов, инструментов автоматизации и способов развертывания приложений. Данные решения принимались исходя из удобства использования, опыта команды и интеграции с существующей инфраструктурой SOC.

### Выбор формата конфигураций (YAML)

Для описания декларативных конфигураций инфраструктуры был выбран формат YAML [15]. Данное решение обусловлено его преимуществами.

* Читаемость и простота: YAML обеспечивает человекочитаемый формат, который позволяет легко интерпретировать содержимое конфигураций без дополнительной обработки.
* Легкость интерпретации: YAML файлы легко и однозначно преобразуются в стандартные структуры данных (словари и списки), что удобно для дальнейшей обработки в сценариях автоматизации на Python.
* Совместимость с существующими решениями: Большинство конфигураций других систем в инфраструктуре SOC уже использует YAML, что позволяет сохранить единообразие подходов.

В качестве альтернативных решений рассматривались форматы JSON и TOML, однако JSON менее удобен для человека из-за отсутствия комментариев и строгого синтаксиса, а TOML, хотя и прост в понимании, не распространен в текущей инфраструктуре, что создало бы дополнительную сложность при интеграции с уже существующими решениями.

### Инструмент оркестрации и автоматизации (Ansible)

В качестве инструмента для оркестрации и автоматизации процесса развертывания коллекторов был выбран Ansible. Причинами этого выбора стали следующие факторы.

* Опыт команды: все члены команды системных инженеров уже обладают достаточной квалификацией и практическим опытом работы с Ansible.
* Интеграция с существующими процессами: Ansible уже используется в инфраструктуре SOC для развертывания других сервисов, что минимизирует необходимость внедрения новых инструментов и дополнительного обучения персонала.
* Простота и эффективность: Ansible предоставляет декларативный подход к описанию инфраструктуры, что удобно и прозрачно для поддержки и внесения изменений.

Другие инструменты оркестрации, такие как Puppet, Chef или SaltStack, могли бы быть рассмотрены как альтернативы. Однако их внедрение не рассматривалось ввиду отсутствия у команды опыта работы с этими инструментами и наличия уже готовых Ansible-скриптов и плейбуков, используемых в инфраструктуре.

### Технология контейнеризации (Docker)

Каждый коллектор разворачивается в виде контейнера Docker. Такое решение обосновано следующими преимуществами контейнеризации.

* Стандартизация среды: использование Docker позволяет стандартизировать управление коллекторами на основе разных приложений независимо от специфики каждого отдельного приложения.
* Безопасность: Docker обеспечивает изоляцию среды приложений, тем самым повышая уровень безопасности инфраструктуры и снижая риски, связанные с компрометацией отдельных компонентов системы [16].
* Удобство сопровождения и масштабирования: Docker-контейнеры легко масштабировать, обновлять и сопровождать, что существенно упрощает работу системных инженеров.

В качестве альтернативы могла быть рассмотрена технология Podman, обеспечивающая аналогичный функционал, но с дополнительными возможностями безопасности (отсутствие демона Docker и возможность работы в rootless-режиме). Однако использование Podman не было выбрано в связи с отсутствием опыта и квалификации команды системных инженеров с данным инструментом. В свою очередь, команда уже обладает значительным опытом работы с Docker, что обеспечило быструю интеграцию решения в существующие процессы SOC.

### Итоги

В результате проектирования была сформирована архитектура подсистемы автоматического подключения источников логов, которая удовлетворяет как функциональным, так и нефункциональным требованиям.

Итоговая архитектура использует **Git-репозиторий в качестве централизованного хранилища состояния сущностей и управления изменениями в них, CI/CD систему как инструмент для запуска автоматизированных действий, выполняемых спроектированным сценарием автоматизации. Такой подход обеспечивает воспроизводимость, безопасность, масштабируемость и минимизацию ручных действий.**

# Реализация автоматизации

## Структура Git-репозитория.

Для реализации предложенного подхода был разработан репозиторий, содержащий необходимые файлы и директории, обеспечивающие централизованное управление конфигурациями сущностей автоматизируемой инфраструктуры.

Структура репозитория представлена на рисунке 15.



Рисунок 15 – Файловая структура репозитория с конфигурациями источников данных

Репозиторий содержит следующие компоненты:

* файл .gitlab-ci.yml, содержащий описание CI/CD пайплайна, автоматизирующего процессы проверки и применения конфигураций;
* директория src, содержащая исходный код сценария автоматизации, реализованного на языке Python;
* директория configs, включающая описание состояния сущностей инфраструктуры в декларативной форме.

Поддиректории в директории configs названы по уникальному идентификатору каждого источника данных. Эти идентификаторы используются в качестве подстроки для имен создаваемых сущностей, что обеспечивает удобство идентификации и управления конфигурациями. Например, топик Kafka для источника antivirus будет называться в формате <<cluster-name>>.<<environment>>.antivirus, где значения переменных cluster-name и environment задаются сервисом KaaS при создании сущностей.

### Структура файла «main.yml»

Каждая директория в configs содержит файл main.yml, в котором декларируются метаданные источника данных, параметры развертывания, а также входные и выходные каналы передачи информации. Содержимое файла main.yml служит основой для автоматической генерации конфигураций и последующего создания соответствующих сущностей в инфраструктуре.

Пример содержания файла main.yml представлен на листинге 6.

Листинг 6 – Пример файла с описанием коллектора



Согласно приведенному примеру, развертывание коллектора предполагает запуск трех экземпляров в зоне доступности us-east-1a. Каждый экземпляр получает параметры окружения, один из которых (SOME\_VAR) является секретом и извлекается из внешнего хранилища HashiCorp Vault. Коллектор открывает TCP-порт 6100 для приема данных и отправляет полученные данные в соответствующий топик Apache Kafka, описанный в файле конфигурации kafka.yml.

### Структура файла «kafka.yml»

Файл конфигурации kafka.yml содержит параметры для создания и настройки топика Kafka и пользователей, которым необходим доступ к данному топику. Пример файла приведен на листинге 7.

Листинг 7 – Пример файла конфигурации топика и пользователей Kafka



В конфигурации топика обязательны два параметра: количество партиций и объем хранения (retention\_bytes). Остальные параметры (например, retention\_ms или replication\_factor) имеют предопределенные значения по умолчанию и могут быть опущены. Для пользователей обязательными параметрами являются ограничения по скорости записи и чтения данных (produce\_byte\_rate и consume\_byte\_rate соответственно).

## Алгоритм применения изменений

На рисунке 16 представлен алгоритм применения изменений конфигураций посредством с использованием CI/CD-системы.

Системный инженер инициирует процесс изменения конфигурации путем создания коммита и последующего Merge Request в основную ветку репозитория. CI/CD-система реагирует на эти события запуском пайплайнов, выполняющих статические проверки с использованием линтеров (как сторонних, таких как Yaml Lint[[8]](#footnote-8), так и собственной разработки). После успешного прохождения проверок и подтверждения изменений инженером, изменения сливаются в основную ветку, что запускает пайплайн применения изменений.

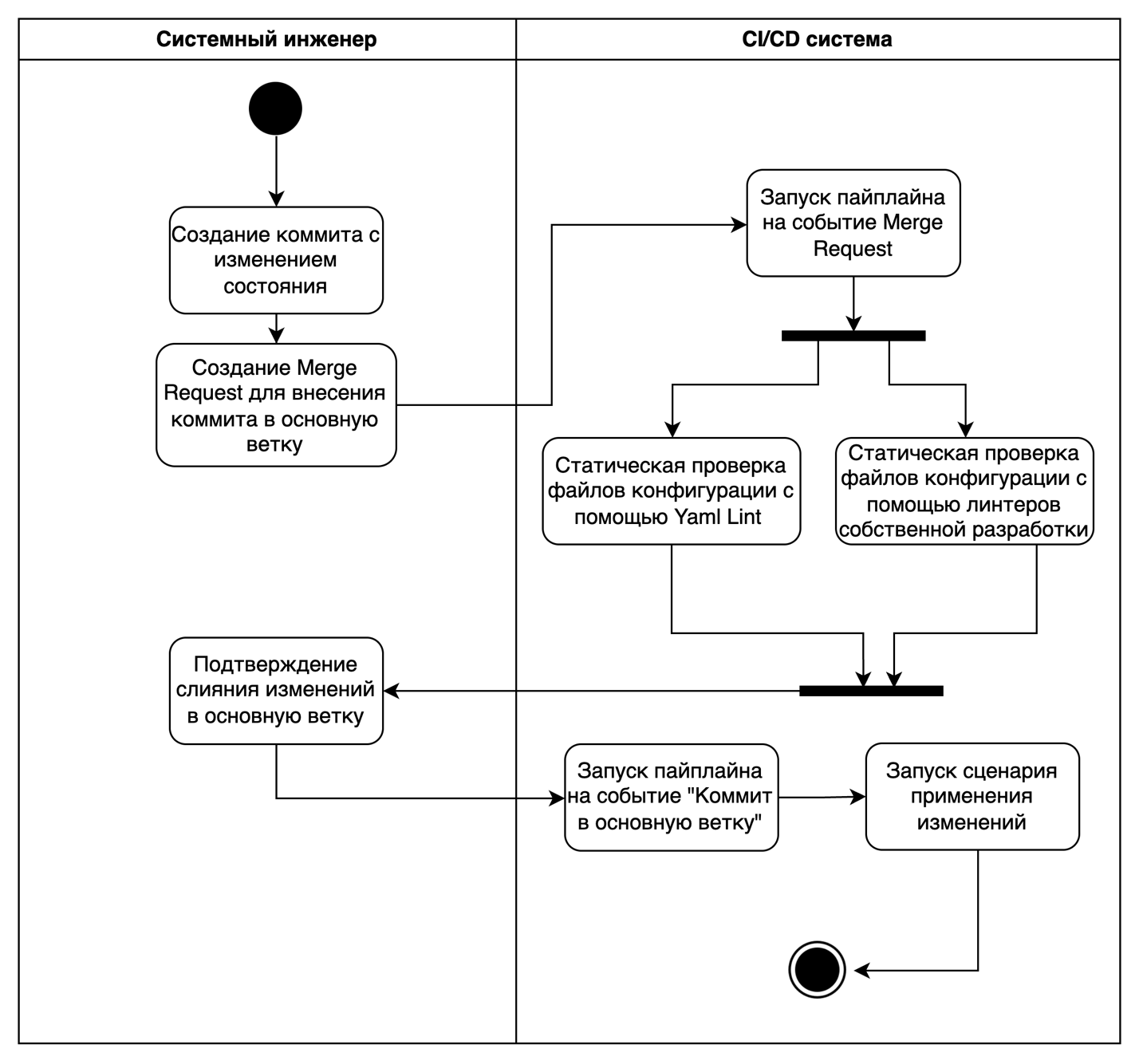


Рисунок 16 – Алгоритм применения изменений

### Общая логика сценария автоматизации

На рисунке 17 приведена последовательность выполнения сценария автоматизации, реализованного на языке Python.

После инициализации необходимых объектов менеджеров (TopicManager, UserManager, CollectorManager) вызывается метод retrieveExistingStates(), который считывает текущее состояние всех сущностей, находящихся под управлением системы. Далее происходит обход всех каталогов с конфигурациями, парсинг их содержимого и вызов соответствующих методов обработки (process) для каждой сущности (топики, пользователи, коллекторы). После завершения обработки всех сущностей вызывается метод deleteUndesiredEntities(), который удаляет сущности, не задействованные в текущем цикле автоматизации.

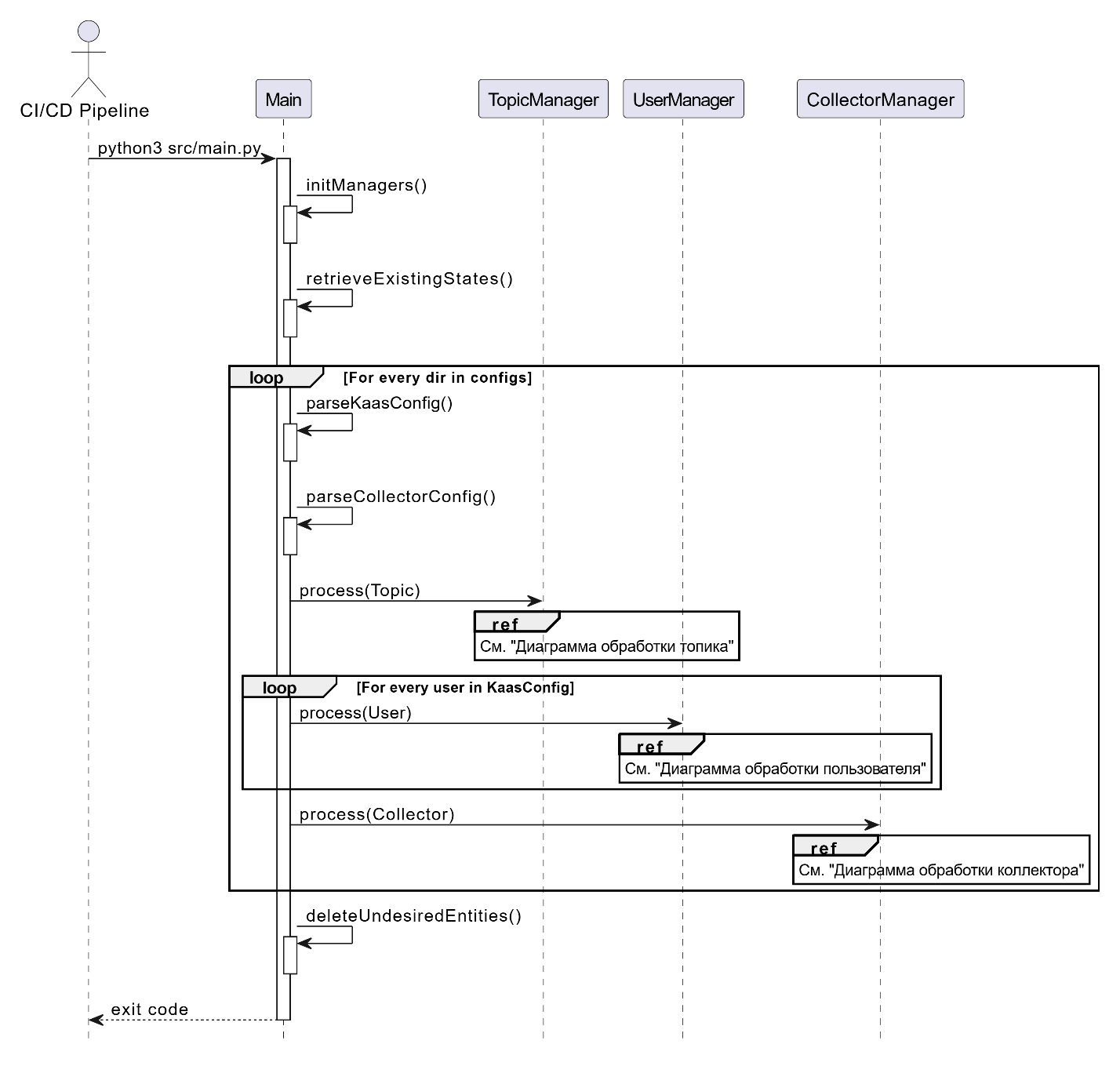


Рисунок 17 – Логика работы сценария автоматизации

Для наглядности работы методов обработки сущностей в разделах 6.2.2 – 6.2.4 приведен детальный разбор выполняемых операций.

### Автоматизация управления топиками

На рисунке 18 приведен детальный алгоритм работы с топиками Apache Kafka в рамках разработанного сценария автоматизации.

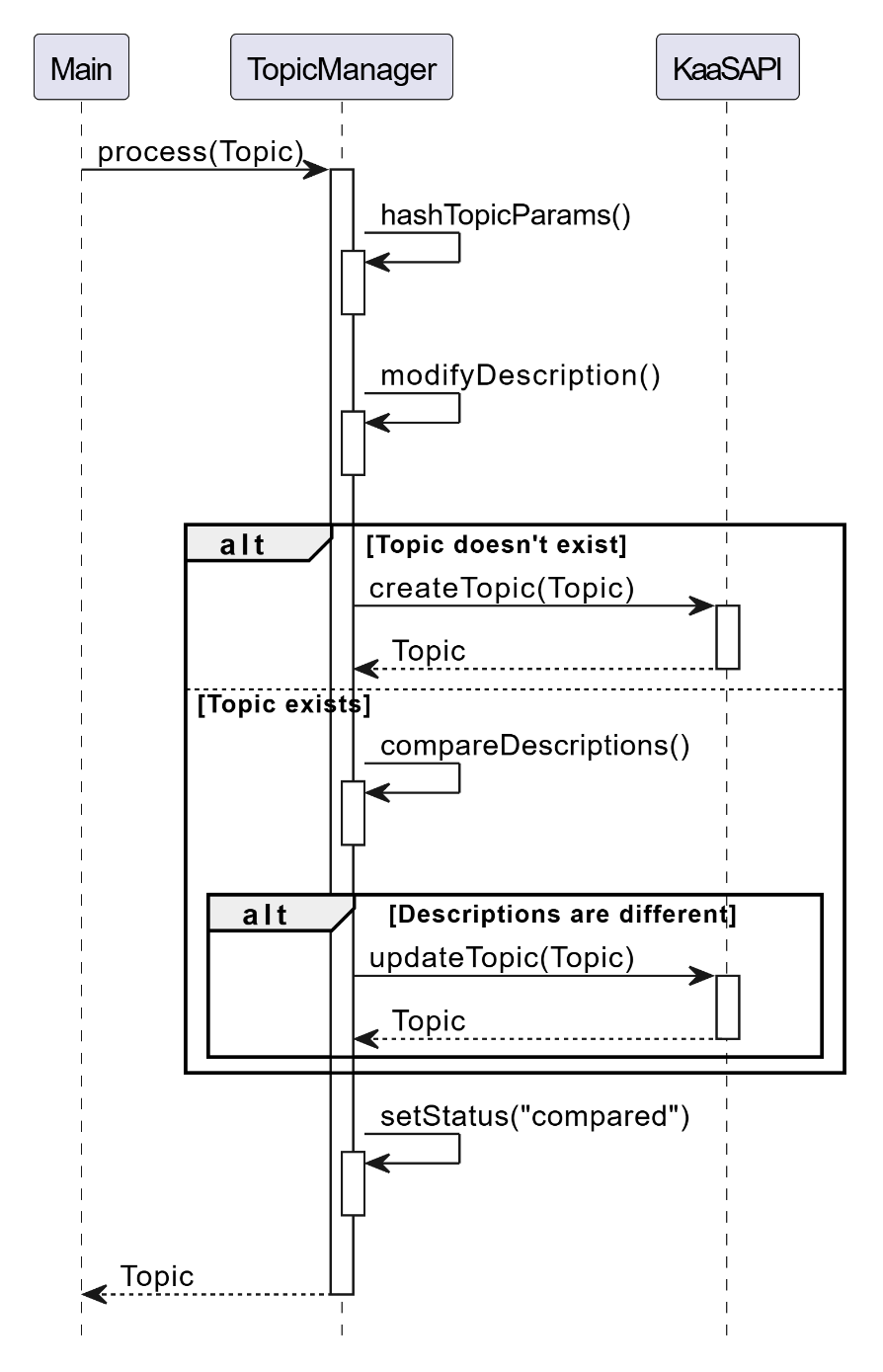


Рисунок 18 – Процесс обработки топика

При обработке каждого топика TopicManager выполняет следующие действия.

1. Вычисляет хэш технических параметров топика и модифицирует описание, добавляя этот хэш и необходимую метаинформацию.
2. Выполняет проверку существования топика в текущем состоянии инфраструктуры.
   * Если топик не существует, отправляет POST-запрос к API сервиса KaaS для создания топика. Запрос содержит обязательные параметры, такие как срок хранения сообщений, количество партиций и максимальный объем хранилища.
   * Если топик уже существует, производится сравнение текущего описания с ожидаемым по ранее вычисленному хэшу параметров:
     + в случае несовпадения описаний отправляется PATCH-запрос в KaaS API для приведения параметров топика в соответствие с конфигурацией;
     + в случае совпадения описаний действие не требуется;
3. В конце обработки устанавливается статус топика, обозначающий, что сущность была проверена и актуальна.

Использование механизма хэширования обусловлено ускорением процесса проверки актуальности параметров и минимизацией обращений к API KaaS.

### Автоматизация управления пользователями

На рисунке 19 представлена детальная логика управления учетными записями пользователей в сервисе KaaS.

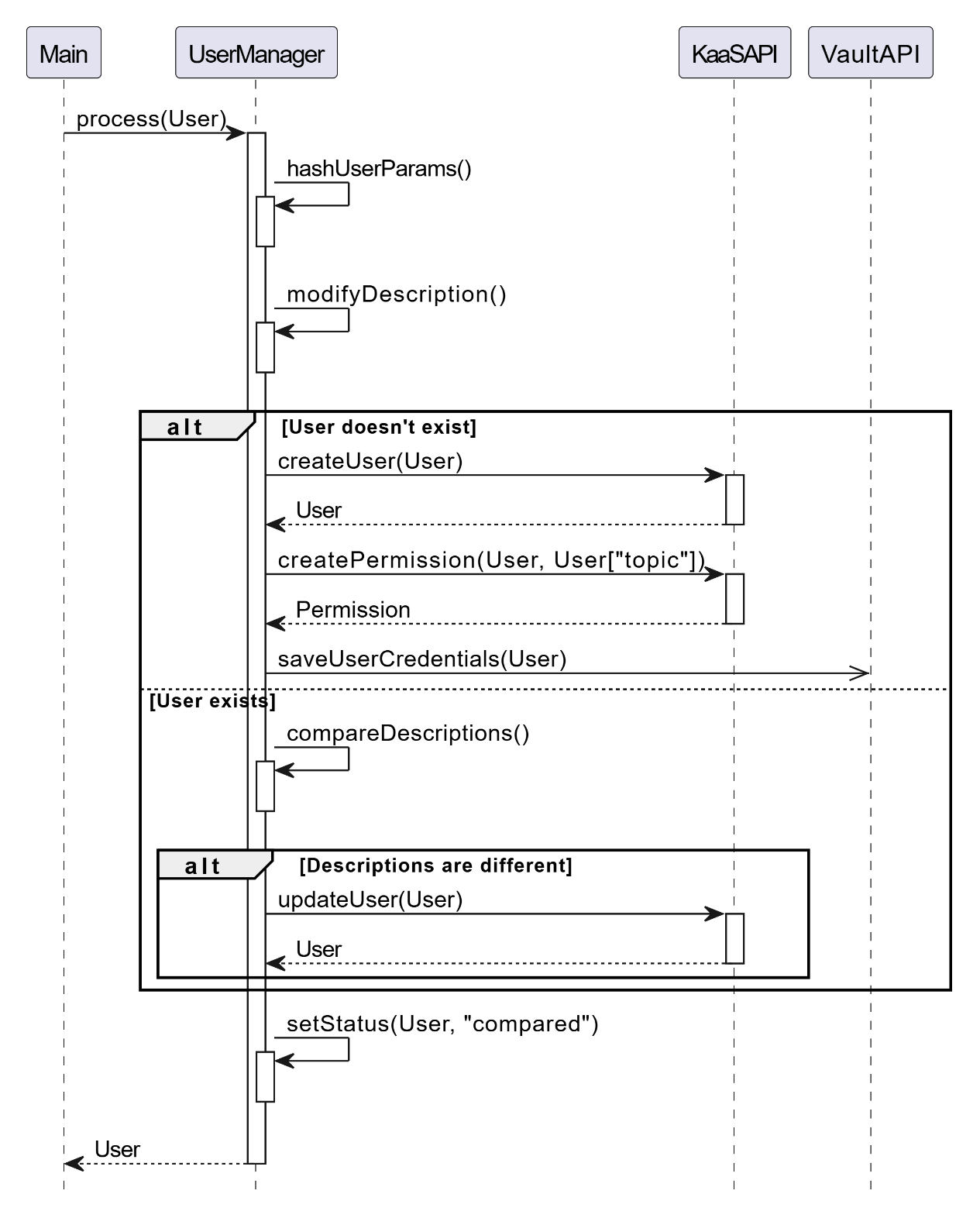


Рисунок 19 – Процесс обработки пользователя

Алгоритм работы менеджера пользователей (UserManager) состоит из следующих этапов.

1. Проверка наличия учетной записи пользователя в словаре existing\_entities.
   * Если учетная запись уже существует, сравнивается ее текущее описание с ожидаемым, и, в случае несовпадения, отправляется PATCH-запрос к KaaS API для обновления конфигурации учетной записи.
   * Если учетная запись не существует, создается новая запись посредством POST-запроса к API KaaS.
2. После создания или подтверждения актуальности учетной записи выполняется настройка разрешений (permissions), определяющих права доступа пользователей к соответствующим топикам.
3. Учетные данные пользователя отправляются в систему безопасного хранения секретов HashiCorp Vault с целью обеспечения безопасности и защиты конфиденциальности информации.

Для оптимизации количества обращении к KaaS API при настройке разрешений система автоматизации всегда посылает запрос на создание необходимого разрешения и обрабатывает ответ с кодом 409 как штатную ситуацию и не требующую дополнительных действий. Такой подход позволяет сократить количество сетевых запросов примерно в 2 раза.

### Автоматизация управления коллекторами

В процесс обработки коллектора (рисунок 20) CollectorManager выполняет следующие шаги.

1. Менеджер получает информацию о текущем состоянии активных коллекторов путем вызова Ansible-модуля shell. Модуль выполняет команду Docker, собирая информацию о контейнерах с меткой управления.
2. Результат выполнения команды преобразуется в «словарь» (hash-map, dict в языке Python) активных коллекторов (existing\_collectors), используемый для дальнейших проверок.
3. Далее менеджер проверяет наличие каждого коллектора в словаре существующих:
   * в случае отсутствия активного экземпляра коллектора запускается Ansible-плейбук (deploy.yml) для его развертывания;
   * Если активный экземпляр уже существует, обновляется его статус в словаре.
4. Перед непосредственным развертыванием происходит генерация конфигурационного файла на основе Jinja2-шаблонов, использующих параметры из файла конфигурации источника данных. Генерация учитывает специфику источника данных, будь то сетевой протокол или SQL-запрос к базе данных.
5. Сформированная конфигурация передается на целевой хост с помощью Ansible-плейбука.

Использование шаблонизатора Jinja2 обеспечивает универсальность и масштабируемость процесса, позволяя легко адаптировать конфигурацию под различные типы источников данных[].

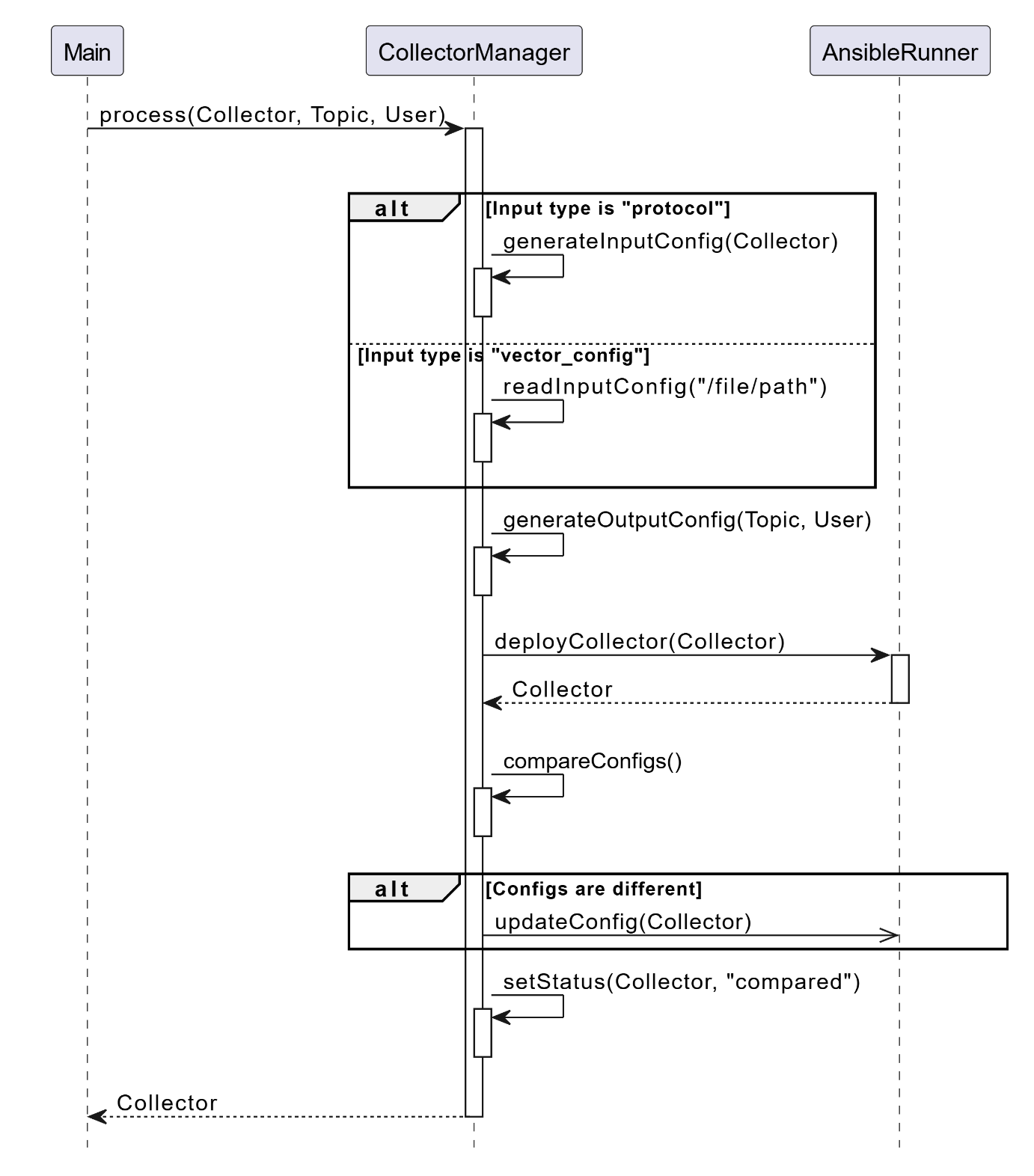


Рисунок 20 – Процесс обработки коллектора

## Предоставление данных о развернутом экземпляре заинтересованным лицам

Основной потребностью заинтересованных лиц (команды TH-аналитиков и ответственных за источники логов) является получение информации об адресе коллектора, настроенного на прием данных с определенного источника, и статусе функционирования развернутого экземпляра данного коллектора. Первый аспект удовлетворяется за счет использования единого источника достоверной информации, реализованного в виде Git-репозитория, где фиксируется адрес, по которому коллектор принимает входящие данные. Второй аспект требует применения систем мониторинга, обеспечивающих оценку эксплуатационных характеристик.

Ключевыми метриками для определения работоспособности коллектора выступают скорость приема данных коллектором, скорость записи данных в кластер Apache Kafka и количество сообщений, накопленных в Kafka. Ненулевые значения указанных метрик при активной отправке данных в коллектор свидетельствуют о корректной работе системы. Оптимальным решением для повышения доступности информации является разработка интерактивного дашборда (рисунок 21), агрегирующего указанные метрики, что позволит заинтересованным сторонам получать оперативные ответы на запросы без вовлечения системных инженеров.

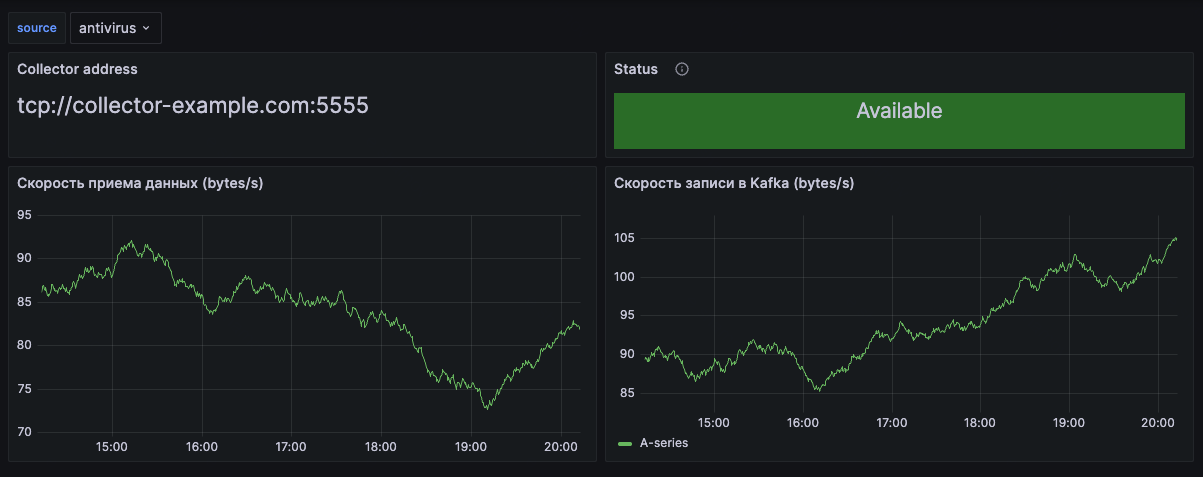


Рисунок 21 – Пример интерактивного дашборда с основными метриками работоспособности коллектора

## Итоговый процесс



Рисунок 22 – Процесс настройки сбора логов с источника

Итоговый процесс настройки сбора логов представлен на рисунке 22. В результате реализации предложенной автоматизации количество ручных операций, выполняемых системным инженером, сократилось до одной – описания конфигурации необходимых сущностей (ранее их было шесть, рис. 3). Учитывая высокую повторяемость конфигураций для различных коллекторов, данная операция, как правило, сводится к копированию уже существующей конфигурации в новую директорию с последующим переименованием этой директории и незначительными изменениями описаний сущностей в конфигурационных файлах.

Сокращено количество задействованных команд – миграция в KaaS и упрощение взаимодействия с конфигурациями балансировщиков позволила упростить типовой процесс, избавив его от согласований внешних команд.

Среднее время настройки сбора логов с источника было сокращено до ≈24 часов (ранее 72 часа), а среднее время, в которое задействован системный инженер сокращено с 48 часов до 10 минут, которые необходимы на настройку конфигурации.

Выполнение задач, связанных с конфигурацией парсера в данном процессе, остается ручным, поскольку требует экспертного анализа, неэффективного при автоматизации.

Заключение

В рамках данной работы была выполнена задача автоматизации процесса лог-менеджмента. Проведенный анализ существующего процесса настройки приема логов выявил ключевые недостатки, включая высокие временные затраты, значительную долю ручных операций, нарушение принципа наименьших привилегий и зависимость от согласований с внешними командами. Эти проблемы снижали эффективность работы SOC и создавали риски для безопасности и масштабируемости системы.

Для устранения выявленных недостатков был спроектирован и реализован целевой процесс, основанный на централизованном управлении конфигурациями через Git-репозиторий и автоматизации ключевых операций с использованием CI/CD-пайплайнов. Разработанное решение включает автоматизацию создания топиков и учетных записей, настройки коллекторов, а также предварительную конфигурацию балансировщиков, что позволило сократить время настройки нового источника логов до 2–3 часов.

Внедрение решения обеспечило выполнение функциональных и нефункциональных требований, включая повышение безопасности за счет соблюдения принципа наименьших привилегий, улучшение масштабируемости и отказоустойчивости системы, а также совместимость с существующей инфраструктурой SOC. Реализация мониторинга и предоставление информации заинтересованным лицам через дашборд дополнительно повысили прозрачность и управляемость процесса.

Сравнение прежнего и обновленного процессов показало значительное сокращение временных затрат и устранение большинства качественных недостатков, таких как ручные операции и зависимости от внешних команд. Разработанное решение создает основу для дальнейшего развития подсистемы сбора логов SOC, обеспечивая ее адаптивность к росту числа источников и повышению сложности инфраструктуры.

В перспективе возможно дальнейшее совершенствование процесса за счет автоматизации верификации парсеров, интеграции с системами мониторинга для более глубокого анализа метрик и разработки механизмов автоматического удаления неактуальных сущностей, что позволит минимизировать технический долг и повысить общую эффективность работы SOC.

Список использованных источников и литературы

1. Инцидент информационной безопасности // WikiSec – [Б.м.]., 2025 – URL: <https://wikisec.ru/index.php?title=Инцидент_информационной_безопасности>
2. Аудит информационной безопасности информационных систем // CFO Russia – [Б.м.]., 2025 – URL: <https://www.cfo-russia.ru/glossariy/229/80048>
3. What is CI/CD Pipeline // RedHat – URL: <https://www.redhat.com/en/topics/devops/what-cicd-pipeline>
4. Bhadauria, S. S., & Sahu, R. (2020). Cybersecurity Operations Handbook. Elsevier
5. **Клеппман М.** «Потоковая обработка данных с помощью Kafka» (Designing Data-Intensive Applications), 2019.
6. What is SIEM? // Microsoft – [Б.м.]., 2025 – URL: <https://www.microsoft.com/en-us/security/business/security-101/what-is-siem>
7. What is SOAR? // Palo Alto Networks – [Б.м.]., 2025 – URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-soar>
8. Accessing Jira Audit Information through the Database // Atlassian Support – [Б.м.]., 2025 – URL: <https://support.atlassian.com/jira/kb/accessing-jira-audit-information-through-the-database>
9. Apache Kafka Documentation // Apache Kafka – [Б.м.]., 2025 – URL: <https://kafka.apache.org/documentation/>
10. What Is the Principle of Least Privilege? // Palo Alto Networks – [Б.м.]., 2025 – URL: <https://www.paloaltonetworks.com/cyberpedia/what-is-the-principle-of-least-privilege>
11. Gormley, C., & Tong, Z. (2015). Elasticsearch: The Definitive Guide. O’Reilly Media
12. Site Reliability Engineering // Google – [Б.м.], 2025. – URL: <https://sre.google>
13. Infrastructure as Code: Managing Servers in the Cloud / K. Morris – O’Reilly Media, 2016.
14. Duck Typing in Python: Writing Flexible and Decoupled Code URL: <https://realpython.com/duck-typing-python>
15. YAML 1.2 Specification. https://yaml.org/spec/1.2/spec.html
16. Программные механизмы изоляции контейнеров Docker // ОКБ САПР – [Б.м.]., 2025 – URL: <https://www.okbsapr.ru/library/publications/programmnye-mekhanizmy-izolyatsii-konteynerov-docker>
17. Jinja2 Templating Engine Documentation URL: <https://jinja.palletsprojects.com/en/stable>

1. Протоколы более высокого уровня по модели OSI. Подробнее про модель OSI: <https://ru.wikipedia.org/wiki/Сетевая_модель_OSI> [↑](#footnote-ref-1)
2. Jira – коммерческая система отслеживания ошибок, предназначена для организации взаимодействия с пользователями, хотя в некоторых случаях используется и для управления проектами. Источник – Википедия (<https://ru.wikipedia.org/wiki/Jira>) [↑](#footnote-ref-2)
3. Elasticsearch — тиражируемая программная поисковая система, написана на Java. По состоянию на середину 2010-х годов — самая популярная в своей категории. Источник – Википедия (<https://ru.wikipedia.org/wiki/Elasticsearch>). [↑](#footnote-ref-3)
4. Logstash – конвейер обработки данных на стороне сервера с открытым исходным кодом. Подробнее: <https://www.elastic.co/logstash> [↑](#footnote-ref-4)
5. Подробнее о внесении изменений через форки: <https://git-scm.com/book/ru/v2/GitHub-Внесение-собственного-вклада-в-проекты> [↑](#footnote-ref-5)
6. HashiCorp Vault - система управления секретами с открытым исходным кодом, предназначенная для безопасного хранения, распределения и контроля доступа к чувствительным данным, таким как токены, пароли, сертификаты и ключи шифрования. Vault является единственным рекомендуемым решением для хранения секретов в компании. Подробнее о Vault: <https://developer.hashicorp.com/vault> [↑](#footnote-ref-6)
7. Идемпотентность – свойство операции, означающее, что операция будет выполнять одни и те же действия, даже если будет выполнена несколько раз (Источник: <https://blog.dreamfactory.com/what-is-idempotency>). [↑](#footnote-ref-7)
8. Yaml Lint – инструмент для статической проверки синтаксиса файлов в формате YAML. Источник: <https://trunk.io/linters/yaml/yamllint> [↑](#footnote-ref-8)