Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

Научно-образовательный центр «Высшая ИТ школа»

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА**

АВТОМАТИЗАЦИЯ ПРОЦЕССА ЛОГ-МЕНЕДЖМЕНТА В ПРОЕКТЕ SOC

Шамов Егор Сергеевич

Направление подготовки 09.03.04 Программная инженерия

Направленность (профиль) «Программная инженерия»

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Ким Константин Станиславович, доцент, учебный офис

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

Научный консультант

\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Зоркин Александр Сергеевич, ведущий инженер, ООО "ТЦР"

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

Автор работы

студент группы № 972103

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

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_ 20 \_\_\_ г.

Оглавление

[1 Анализ существующего процесса 7](#_Toc196859327)

[1.1 Введение в предметную область 7](#_Toc196859328)

[1.1.1 Источники логов 7](#_Toc196859329)

[1.1.2 Коллекторы 8](#_Toc196859330)

[1.1.3 Apache Kafka 8](#_Toc196859331)

[1.1.4 Парсеры (нормализаторы) 9](#_Toc196859332)

[1.1.5 Система хранения логов 10](#_Toc196859333)

[1.1.6 Коррелятор 10](#_Toc196859334)

[1.1.7 Введение акторов 11](#_Toc196859335)

[1.2 Описание существующего процесса настройки приема логов 12](#_Toc196859336)

[1.2.1 Описание типового процесса настройки 12](#_Toc196859337)

[1.2.2 Краевые случаи бизнес-процесса 15](#_Toc196859338)

[1.3 Описание недостатков в существующем процессе. 16](#_Toc196859339)

[1.3.1 Времязатраты на создание коллектора и топика 16](#_Toc196859340)

[2 Проектирование целевого процесса 17](#_Toc196859341)

[2.1 Требования к решению 17](#_Toc196859342)

[2.2 Сокращение количества участников процесса 17](#_Toc196859343)

[2.2.1 Автоматизация настройки параметров топика 17](#_Toc196859344)

[2.2.2 Конфигурация балансировщика 18](#_Toc196859345)

[2.2.3 Итог 19](#_Toc196859346)

[2.3 Настройка автоматизации 19](#_Toc196859347)

[2.3.1 Предлагаемая структура проекта для управления коллекторами. 20](#_Toc196859348)

[2.3.2 Структура файла «main.yml» 20](#_Toc196859349)

[2.3.3 Описание процесса развертывания 22](#_Toc196859350)

[2.3.4 Автоматизация поиска свободного порта 22](#_Toc196859351)

[2.3.5 Автоматизация создания топика 22](#_Toc196859352)

[2.3.6 Автоматизация создания пользователей 22](#_Toc196859353)

[2.3.7 Автоматизация создания коллектора 23](#_Toc196859354)

[2.3.8 Предоставление информации о развернутом экземпляре заинтересованным лицам 23](#_Toc196859355)

[2.4 Настройка CI/CD пайплайнов в репозитории 24](#_Toc196859356)

[2.4.1 Проверка корректности конфигурации 24](#_Toc196859357)

[2.4.2 Применение изменений 24](#_Toc196859358)

[3 Реализация целевого процесса 25](#_Toc196859359)

[3.1 Автоматизация 25](#_Toc196859360)

[3.2 Обновленный процесс 25](#_Toc196859361)

[3.3 Сравнение предыдущего процесса и обновленного ( 25](#_Toc196859362)

Глоссарий

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

**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.

Введение

Security Operations Center (**SOC**) — специализированное подразделение, обеспечивающее круглосуточный мониторинг и реагирование на инциденты информационной безопасности (ИБ). 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 приём, 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 распределённая платформа Apache Kafka. Kafka обеспечивает балансировку потоков между компонентами, временное хранение сообщений, высокую отказоустойчивость и горизонтальное масштабирование, позволяя обрабатывать данные асинхронно с минимальными задержками.

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 трудоёмкость интеграции.

Быстрый рост инфраструктуры привёл 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. Учёт подключённых источников ведётся вручную и выполняется параллельно настройке приёма событий.

Указанные трудности обусловлены недостаточной автоматизацией существующих процедур. **Цель работы** — разработать и внедрить автоматизированный процесс подключения источников логов, включающий развёртывание 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.

# Анализ существующего процесса

Прежде чем перейти к анализу процесса, необходимо погрузиться в предметную область.

## 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.
* **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 отчётов.

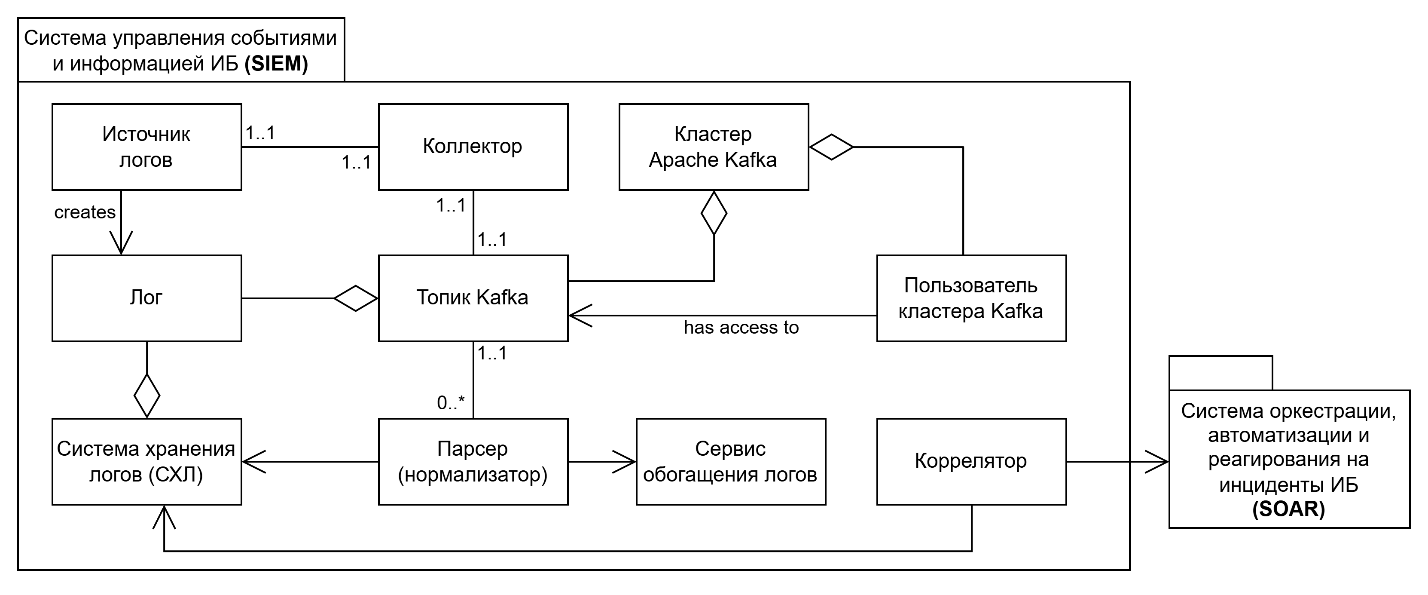
На рисунке 1 представлены основные сущности, задействованные в процессах SIEM, и схема их взаимодействия с платформой SOAR.

Рисунок 1 – Предметная область SIEM

### Источники логов

Источниками событий выступают сетевое оборудование, антивирусные решения, средства аудита операционных систем, системы управления учётными записями (IdM/IAM) и иные корпоративные сервисы, генерирующие технические или аудиторские сообщения.

Некоторые источники логов не способны передавать данные через современные протоколы взаимодействия. Например, большое количество серверного оборудования предлагает экспорт логов только через UDP (англ. User Datagram Protocol, сетевой протокол транспортного уровня). А некоторые приложения (например, система управления проектами Jira) сохраняют данные аудита только в базу данных (БД), в таком случае, единственный путь получения данных – читать их прямо из БД. Для интеграции подобных источников требуется промежуточный компонент – коллектор.

### Коллекторы

**Коллектор** — это сервис, который принимает события от различных источников по разным протоколам и интерфейсам и передаёт их в стандартизированном виде в систему Apache Kafka. Основная задача коллектора заключается в обеспечении совместимости устаревших или ограниченных в плане интеграции систем с современной инфраструктурой SIEM. При этом коллектор не изменяет смысл или содержимое события, а только адаптирует транспортный и форматный 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. Среди инструментов с открытым исходным кодом можно выделить следующие инструменты, подходящие на роль коллектора: Filebeat[[1]](#footnote-1), FluentD[[2]](#footnote-2), Vector.dev[[3]](#footnote-3)

### Apache Kafka

**Apache Kafka** представляет собой распределённую платформу потоковой передачи данных, предназначенную для обработки и передачи больших объёмов сообщений в режиме реального времени. Как компонент SIEM Kafka обеспечивает надёжную 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.

Kafka используется в архитектуре SIEM для обеспечения отказоустойчивости, балансировки нагрузки между компонентами и временного хранения событий при высоких нагрузках или сбоях отдельных сервисов.

### 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 путём 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, обогащённые 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, обогащённых и подвергнутых предварительному анализу логов. Основной задачей СХЛ является сохранение и систематизация больших объёмов данных с последующей возможностью эффективного поиска, фильтрации и углублённого 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[[4]](#footnote-4), 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 обогащённых 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.

### Введение акторов

В процессе настройки приема логов в Security Operations Center (SOC) 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.

* + - * **SRE-инженеры** **SOC** (англ. Site Reliability Engineers, инженеры по обеспечению надежности в отделе SOC, далее – SRE-инженеры). SRE-инженеры 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. SRE-инженеры также взаимодействуют с внешними командами для согласования изменений в конфигурациях Kafka и балансировщиков.
      * **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 рамках настройки приема 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 SRE-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. Они согласовывают и применяют изменения в конфигурационных файлах балансировщиков, которые вносят SRE-инженеры.
      * **Команды, ответственные за источники логов** управляют системами, генерирующими события. Они участвуют в определении протоколов передачи данных (например, UDP, TCP, чтение из базы данных) и настраивают источники для отправки логов в коллекторы SOC.

## Описание существующего процесса настройки приема логов

Настройка приема логов из источника 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

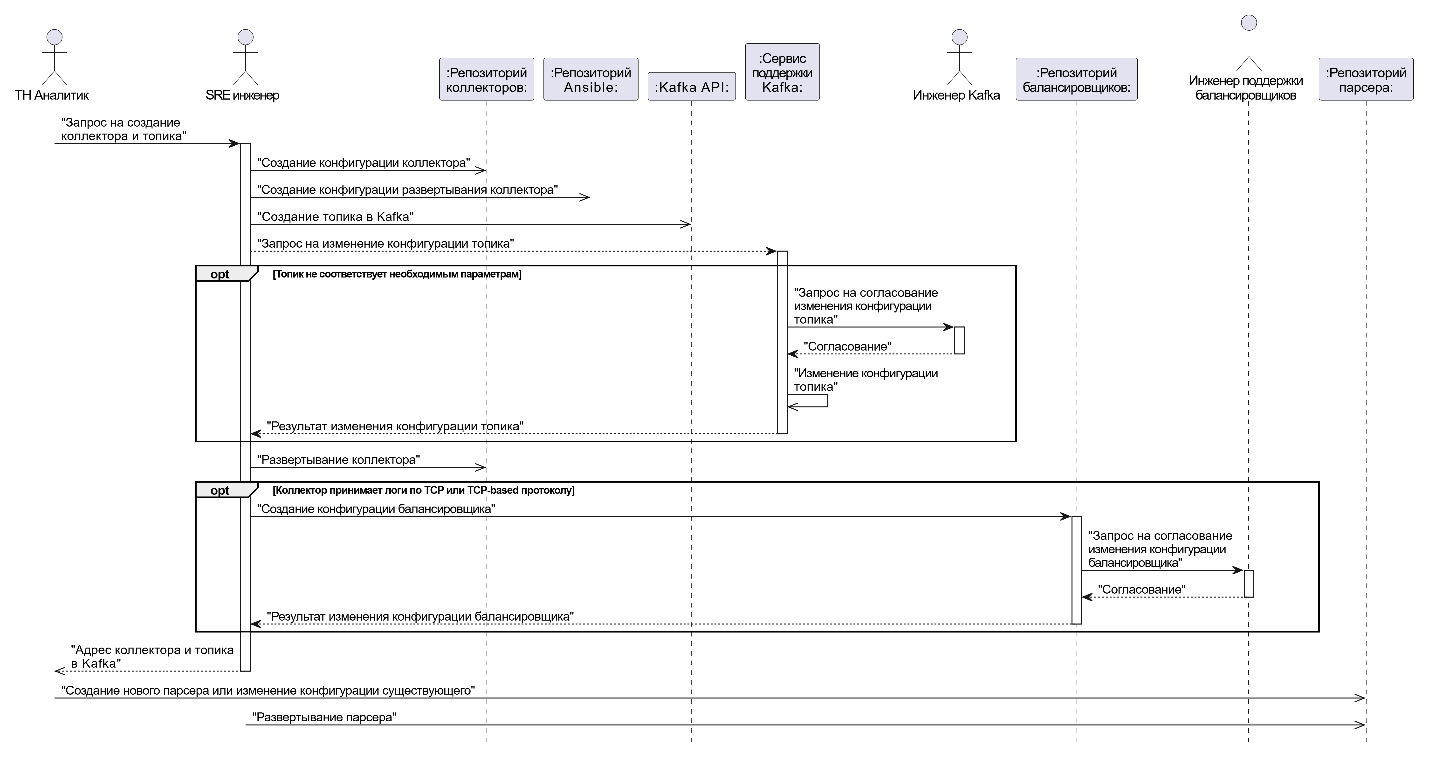


Рисунок 2 – Процесс настройки сбора логов с источника с использованием сетевого коллектора

Типовой процесс настройки сбора логов представлен на рисунке 2. Данный процесс охватывает наиболее распространенный сценарий, при котором источник передает логи через сетевой коллектор, использующий протоколы TCP или TCP-based (например, Syslog). В рамках типового процесса исключаются краевые случаи, такие как прямое взаимодействие источника с Apache Kafka, чтение журналов событий из баз данных или отклонение запросов на согласование со стороны внешних команд[[5]](#footnote-5)

Для реализации типового процесса необходимо выполнить следующие этапы: настройка коллектора, создание и настройка параметров топика в Apache Kafka, настройка балансировщика и разработка парсера для нормализации логов. 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»). В типовом сценарии коллектор функционирует как сервер, принимающий данные по сети через открытый порт, работающий по протоколам TCP или UDP. Однако данный подход применим не ко всем источникам: например, для источников, хранящих логи в базах данных, требуется иная логика обработки, которая будет рассмотрена в разделе 1.2.2.

Все сетевые коллекторы развернуты на виртуальных машинах, размещенных в различных дата-центрах. Для обеспечения отказоустойчивости и упрощения управления каждый коллектор представлен на каждой виртуальной машине, что позволяет балансировать сетевой трафик, особенно для протоколов TCP и TCP-based. Поскольку на одной машине одновременно функционирует несколько коллекторов, каждому из них необходимо назначить уникальный порт. На текущий момент выбор порта выполняется вручную: SRE-инженер анализирует репозиторий конфигураций коллекторов и определяет минимальный свободный порт, который затем фиксируется в конфигурационном файле. Такой подход обеспечивает упорядоченное распределение портов, однако требует значительных временных затрат, что будет рассмотрено в разделе 1.3.

На выходе коллектор всегда передает данные в Apache Kafka. Для передачи данных в топик Kafka коллектору требуется учетная запись (УЗ) 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, создавая потенциальные риски безопасности. Также в конфигурационном файле коллектора необходимо указывать название топика и адреса брокеров Kafka. Название топика определяется TH-аналитиками на этапе согласования требований к данным, что позволяет указать его в конфигурации до фактического создания топика в Kafka

Для обеспечения балансировки трафика от источника к коллектору используются платформенные балансировщики, предоставляемые внутри компании. SRE-инженеры описывают конфигурацию балансировщика в соответствующем репозитории, указывая доменное имя и порт, который должен прослушивать балансировщик, а также адреса виртуальных машин, на которые перенаправляется трафик. Для всех источников используется единый адрес балансировщика, различающийся только портом, что упрощает управление, но требует внесения новой записи в конфигурацию при создании каждого коллектора. 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.

TODO: листинг с примером файла балансировщика.

#### Создание топика

Кластеры Apache Kafka в компании предоставляются по модели Infrastructure as a Service (IaaS). Команда SRE в SOC имеет учетные записи с правами на чтение, запись и создание топиков, однако настройка параметров топиков и другие административные действия выполняются через команду обслуживания Kafka. Создание топика осуществляется с использованием веб-сервиса, разработанного данной командой. Процесс включает авторизацию SRE-инженера в веб-сервисе и заполнение формы с параметрами топика, такими как название, описание, количество партиций и время 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), SRE-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 SOC код парсеров организован в репозиториях, сгруппированных по типу обрабатываемых логов, например, сетевые логи, логи операционной системы Windows или логи VPN. Каждый репозиторий может содержать несколько пайплайнов. Пайплайн – это сценарий обработки логов, содержащий правила переименования, удаления или добавления полей в лог.

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. SRE-инженеры, в свою очередь, отвечают за развертывание парсеров, выделение ресурсов и выполнение операционных задач, таких как мониторинг и устранение сбоев.

### Краевые случаи процесса настройки

Типовой процесс, описанный в разделе 1.2.1, применим не ко всем источникам логов. В данном разделе рассматриваются два основных краевых случая, требующих отклонения от стандартного 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: SRE-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. Коллектор в данном сценарии не используется, однако TH-аналитики по-прежнему разрабатывают парсер для нормализации логов, поступающих в топик.

#### Источники, логи которых можно получить только из базы данных

Некоторые источники, такие как Kaspersky Security Center (консоль удаленного управления системами антивирусной защиты на устройствах предприятия) или Jira (система управления проектами), сохраняют логи аудита исключительно в собственных базах данных. В таких случаях сетевой коллектор, функционирующий как сервер, не применим, и требуется альтернативный подход. Для извлечения логов из баз данных применяются специализированные коллекторы, способные считывать данные непосредственно из базы и передавать их в топик Kafka.

Одним из распространенных решений является использование Logstash, 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. Настройкой таких коллекторов занимается команда TH-аналитиков.

## Анализ существующего процесса настройки приема логов.

Текущий процесс настройки приема логов, описанный в разделе 1.2, характеризуется рядом недостатков, которые снижают его эффективность, увеличивают временные затраты и создают риски для работы Security Operations Center (SOC).

### Количественная оценка временных затрат

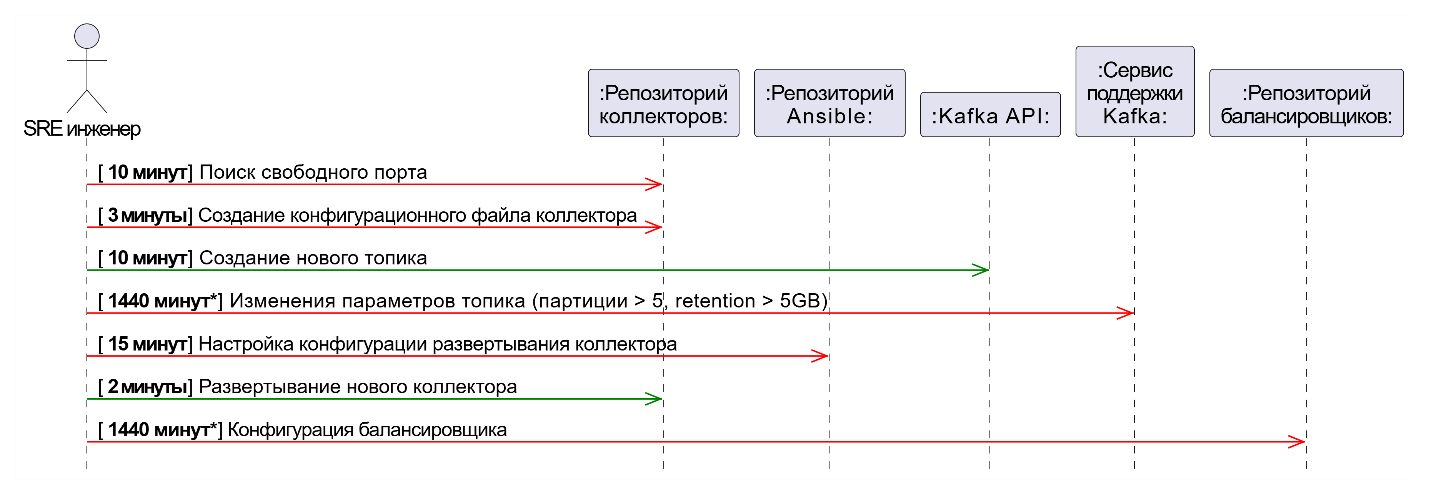


Рисунок 3 – Описание процесса настройки приема данных из источника по TCP или TCP-based протоколу с указанием времязатрат.

Процесс настройки приема логов из нового источника в типовом сценарии (раздел 1.2.1) занимает значительное время, что обусловлено большим количеством ручных операций и необходимостью согласования с внешними командами. На рисунке 3 представлена схема процесса с указанием этапов и их продолжительности. Красным цветом выделены этапы, требующие ручных действий, зеленым - автоматизированные.

Общие временные затраты на настройку приема логов из одного источника в типовом сценарии составляют до 2895 минут (примерно 48 часов, или 2 суток). Из них 2160 минут (36 часов) приходится на этапы, связанные с согласованием с внешними командами (балансировщики и Kafka). Данные значения подтверждают, что процесс является трудоемким и требует значительного участия SRE-инженеров, которые вынуждены держать контекст задачи 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 репозиторий балансировщиков, выполняются вручную. Это приводит к следующим проблемам.

* Понижение эффективности труда SRE-инженера. Рутинные операции, требующие постоянного вовлечения, занимают рабочее время, не позволяя инженеру заниматься более интересными и важными задачами, такими как автоматизация, повышение уровня поддержки или развитие младших коллег.
* Риск человеческих ошибок. Выбор занятого порта при конфигурации может привести к конфликту между коллекторами, что вызовет сбои в передаче логов или вынудит потратить на задачу больше времени, чем необходимо.

#### 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 значительные риски безопасности.

* Риск утечки учетной записи ситуацией, когда злоумышленник может записывать произвольные данные во все топики, что приведет к перегрузке парсеров и потенциальной потере значимых событий.
* Риск нарушения целостности данных. Из-за отсутствия изоляции злоумышленник или неверно сконфигурированный коллектор может перезаписать данные в множестве топиков, что затруднит их дальнейшую обработку и анализ.

Данный недостаток особенно критичен в контексте SOC, где потеря событий или их искажение может привести к незамеченной нелегитимной активности, угрожающей безопасности инфраструктуры.

#### 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 её несоответствий или недосказанности.
* Скорость реагирования на проблемы, связанны 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.

## Выводы по анализу

Проведенный анализ выявил, что текущий процесс настройки приема логов характеризуется высокими временными затратами (до 48 часов на источник), 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, и требуют разработки автоматизированного решения, которое будет рассмотрено в последующих главах.

# Проектирование целевого процесса

На основании анализа недостатков текущего процесса настройки приема логов, проведенного в разделе 1.3, разработан целевой процесс, направленный на устранение выявленных проблем. Основной целью проектирования является минимизация ручного труда SRE-инженеров, сокращение временных затрат и повышение эффективности работы Security Operations Center (SOC). В данной главе описаны требования к целевому процессу, предложены изменения для устранения узких мест, представлена архитектура решения и проанализированы его ограничения.

## Требования к решению

Разработка целевого процесса должна учитывать как функциональные, так и нефункциональные требования, направленные на решение проблем, выявленных в разделе 1.3. Требования также учитывают ограничения существующей инфраструктуры SOC.

### Функциональные требования

#### Централизованное управление конфигурациями

Для устранения проблемы отсутствия единого источника правды (раздел 1.3.2.3) все конфигурации (коллекторов, топиков, учетных записей) должны управляться через единый источник правды, обеспечивающий доступ к информации о состоянии системы (например, адрес коллектора, параметры топика) в одном месте.

#### Автоматизация ручных операций

Для сокращения доли ручного труда (раздел 1.3.2.1) необходимо автоматизировать следующие процессы:

* поиск свободного порта для коллектора;
* создание топиков в Apache Kafka;
* создание учетных записей с минимальными правами;
* настройка балансировщиков;
* развертывание коллекторов.

#### Сокращение времени настройки

Время настройки приема логов из одного источника должно быть сокращено с 48 часов (раздел 1.3.1) до нескольких часов, что позволит SRE-инженерам сосредоточиться на других задачах, таких как поддержка инфраструктуры и помощь другим командам SOC.

#### Мониторинг и уведомления

Для обеспечения прозрачности процесса необходимо предоставить заинтересованным сторонам (TH-аналитикам, командам источников) информацию о состоянии коллекторов (например, адрес для отправки данных, статус работоспособности) без участия SRE-инженеров.

### Нефункциональные требования

#### Безопасность

Для устранения проблемы нарушения принципа наименьших привилегий (раздел 1.3.2.2) каждая учетная запись должна предоставлять доступ только к строго необходимым топикам, минимизируя риски утечки данных и несанкционированной записи.

#### Масштабируемость

Решение должно поддерживать рост числа источников логов (на текущий момент добавляется 1–2 источника ежемесячно, с прогнозируемым увеличением до 5–7 источников в месяц в течение года).

#### Отказоустойчивость

Процесс должен быть устойчив к сбоям отдельных компонентов (например, недоступности кластера Kafka или CI/CD-пайплайна), минимизируя риск потери логов.

#### Совместимость

Решение должно быть совместимо с существующей инфраструктурой SOC, включая использование Apache Kafka, сетевых коллекторов (Filebeat, Vector.dev) и платформенных балансировщиков.

### Ограничения

#### Инфраструктурные ограничения

Необходимо использовать существующие компоненты (Kafka, балансировщики), минимизируя изменения в инфраструктуре.

#### Ресурсные ограничения

Решение должно быть реализовано в рамках доступных ресурсов SOC, без привлечения дополнительных серверов или лицензий.

#### Совместимость с устаревшими источниками

Некоторые источники логов используют устаревшие протоколы (например, UDP), что требует поддержки таких сценариев.

## Описание целевого процесса

Предлагаемые изменения в целевом процессе происходят на этапах настройки получения данных от источника до поставки данных в топик Kafka. На рисунке 4 в формате диаграммы коммуникаций представлено предлагаемое описание целевого процесса.

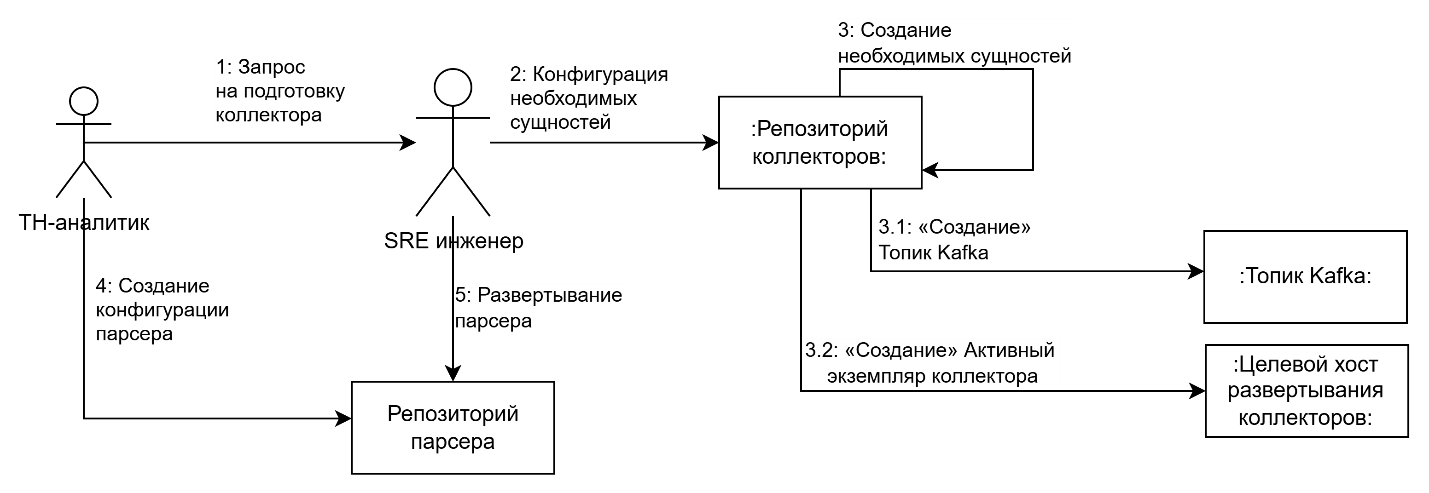


Рисунок 4- Описание целевого процесса настройки сбора логов с источника

Целевой процесс, как и в существующем процессе, активирует TH-аналитик, создавая запрос для SRE-инженера на подготовку коллектора (и сопутствующих сущностей). Также не происходит изменений и в процессе конфигурирования парсеров. Основные изменения касаются деятельности SRE-инженеров. Для настройки получения данных из источника в SOC, необходимо подготовить коллектор и топик для него, эти действия могут быть подвержены автоматизации. Процесс на рисунке 4 предполагает, что SRE-инженер будет вносить изменения в репозиторий коллекторов – единый источник правды, на основе информации из которого системы автоматизации (CI/CD) будут генерировать необходимое состояние системы и применять его на необходимом окружении.

### Автоматизация управления сущностями в Kafka

Для исправления проблемы из раздела 1.3.2.4 (зависимость от внешних команд) необходимо сократить участие внешних команд в целевом процессе до минимального. Также, для исправления проблемы из раздела 1.3.2.2 (нарушение принципа наименьших привилегий), необходимо ввести ограничение на использование учетных записей с широким доступом. Это порождает необходимость создавать учетные записи при создании топика.

Существующий кластер Kafka развернут на инфраструктуре, находящейся в распоряжении отдела SOC, однако его администрированием занимается команда обслуживания Kafka. Известно, что настройка параметров (а также создание новых пользователей) уже осуществляется автоматикой, но её запуск происходит только от учетной записи с правами администратора и только после согласования с ответственными.

В компании существует альтернативное решение – сервис Kafka as a Service (KaaS). **Kafka as a Service (KaaS)** — это управляемое облачное решение на базе Apache Kafka, предоставляемое внутри компании. В модели KaaS клиентам предоставляются квоты на различные сущности — такие как топики, группы потребителей и учетные записи. Принципиальная разница между текущим решением и KaaS заключается в модели управления: сервис остаётся тем же (Kafka), но процессы взаимодействия стандартизированы и унифицированы.

Ключевым преимуществом KaaS является то, что управление топиками, пользователями и их доступами осуществляется не через обращения к команде сопровождения, а напрямую силами клиентов через веб-интерфейс или API. Это позволяет автоматизировать процессы создания и настройки топиков, а также исключить необходимость согласования изменений с командой обслуживания Kafka, что значительно ускоряет работу и повышает гибкость взаимодействия.

Соответственно, для решения упомянутых проблем возможны два подхода:

* получение сервисной УЗ для управления сущностями Kafka в существующем топике;
* использование сервиса KaaS.

В обсуждении данных проблем с командой обслуживания Kafka было получено решение о невозможности предоставления УЗ с требуемыми правами в существующем кластере Kafka, так как он находится в состоянии EoL (англ. End-of-Life), что значит, что он не обновляется, и любые изменения в его конфигурации или политиках доступа не осуществляются. Помимо этого, была получена рекомендация мигрировать на решение Kafka as a Service.

Так, на основе полученной информации можно сделать вывод о необходимости использования KaaS для решения упомянутых проблем. Это значит, что автоматизация управления сущностями Kafka будет автоматизировано через проприетарный API KaaS.

### Конфигурация балансировщиков

Подготовка балансировщика занимает много времени, поскольку также подвержена проблеме, обозначенной в разделе 1.3.2.4 (зависимость от внешних команд). Ниже рассматриваются варианты решения данной проблемы.

Настройка всех балансировщиков происходит через изменения в одном репозитории, CI/CD пайплайн которого автоматически разворачивает балансировщики с необходимыми конфигурациями. За активное состояние балансировщиков, их процесс развертывания и другие автоматизации в этом репозитории ответственна команда обслуживания балансировщиков.

Чтобы создать новый балансировщик или изменить конфигурацию существующего, необходимо внести изменение в репозиторий. Прямое внесение изменений доступно только команде обслуживания балансировщиков, все остальные изменения вносятся через механизм форков (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 от команды обслуживания балансировщиков.

По правилам системы контроля версий, используемой в компании, создание форков возможно только от личной учетной записи, сервисные УЗ не могут иметь личных репозиториев. В таких условиях, данную задачу возможно автоматизировать только в случае владения сервисной УЗ для внесения изменений в репозиторий без необходимости создания форка.

Альтернативный вариант – 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. Минус от такого подхода в том, что в конфигурации балансировщика не будет известно, трафик какого источника обрабатывает балансировщик, так предполагается создание сущностей с динамически генерируемым именем, например, balancer\_for\_port5678 вместо balancer\_for\_windows\_logs\_collector. Также стоит учесть, что подход подразумевает создание объемного артефакта в ручном или полуавтоматическом режиме, но один раз и надолго.

Также как альтернатива могла бы быть рассмотрена собственная ферма балансировщиков, но это влечет за собой необходимость её поддержки, необходимость развития экспертизы команды в данной области, а также потребление дополнительных ресурсов, на которых будут размещены данные балансировщики.

Первый вариант, предполагающий создание сервисной УЗ, не был согласован командой обслуживания балансировщиков, так как они не готовы брать на себя ответственность за управление этой УЗ. В таком случае, самым быстрым и дешевым решением для команды является ручное формирование списка конфигураций балансировщиков.

### Автоматизация управления коллекторами

Создание коллектора подразумевает развертывание активного экземпляра коллектора с конфигурацией, соответствующей той, что описана в репозитории коллекторов. В репозитории коллекторов конфигурация может быть написана как декларативно, с описанием только необходимых сущностей, так и используя синтаксис определенного приложения, выступающего в качестве коллектора. В последнем случае в основном файле конфигурации коллектора должно быть указано, что конфигурацию необходимо использовать из соответствующего файла или списка файлов. В случае декларативного описания коллектора, используемая конфигурация должна быть сгенерирована из шаблона.

### Поиск 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 изменён без уведомления заинтересованных сторон.

С учётом критичности сохранения идентичного состояния инфраструктуры при повторном развёртывании, данный подход был признан неприемлемым.

#### Предоставление рекомендаций по свободному порту в документации

В рамках этого варианта предполагается, что актуальный свободный порт из допустимого диапазона указывается в документе README.md, хранящемся в репозитории конфигураций коллекторов. После каждого изменения в конфигурациях CI/CD-процесс автоматически обновляет этот файл, указывая ближайший незанятый порт, что упрощает выбор при последующих конфигурациях.

Преимущества:

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

Основной недостаток такого подхода в том, что для обновления README.md необходимо создание коммита, что порождает дополнительные сложности в реализации.

Данный подход допустим, так как позволяет достичь баланса между стабильностью конфигураций и снижением временных затрат на ручной поиск.

#### Игнорирование задачи централизованного поиска порта

В этом случае предполагается, что инженер вручную определяет порт, анализируя текущую занятость на рабочих узлах (например, посредством подключения к производственным серверам и выполнения сетевых команд).

Недостатки:

* приводит к росту времени настройки;
* требует прямого доступа к продакшн-средам, что нарушает принципы безопасности и повышает риск ошибок;
* в условиях масштабируемого роста числа источников становится неустойчивым и трудозатратным.

Таким образом, для решения задачи выбора свободного порта в целевом процессе рекомендуется использовать вариант 2, предполагающий публикацию актуального свободного порта в документации. Этот вариант сохраняет прозрачность и воспроизводимость конфигураций, а также ускоряет процесс поиска свободного порта.

### Итоги

В результате проектирования сформирован целевой процесс (рис. 4), позволяющий сократить время интеграции нового источника журналов с ~48 ч до величины, сопоставимой с длительностью единственного CI/CD‑конвейера (не более 2 – 3 ч). Ключевые улучшения достигаются за счёт следующих действий.

* Исключение внешних команд из цепочки согласований благодаря переходу на KaaS и заблаговременной подготовке пула балансировщиков.
* Внедрение централизованного управления конфигурациями коллектора, топика и учётных записей в едином репозитории, что обеспечивает воспроизводимость состояния при повторных развёртываниях.
* Реализация принципа наименьших привилегий (автоматическое создание индивидуальных учётных записей Kafka с гранулярными ACL).

Часть задач (инициация запроса TH‑аналитиком и верификация логической корректности конфигурации парсера) остаётся ручной, поскольку требует экспертного анализа, неэффективного при автоматизации.

Ограничивающими предпосылками целевого процесса являются эксплуатационная доступность KaaS и фиксированный пул заранее сконфигурированных балансировщиков; при изменении этих условий требуются корректировки модулей автоматизации.

Таким образом, спроектированный процесс удовлетворяет как функциональным, так и нефункциональным требованиям (разд. 2.1) и создаёт основу для дальнейшего повышения масштабируемости и надёжности подсистемы сбора событий SOC.

## Проектирование автоматизации

Одним из ключевых требований к целевому процессу является организация централизованного управления состоянием системы, выступающего единым источником достоверной информации. Для реализации данной задачи предлагается использование Git-репозитория с четко определенной структурой файлов и директорий, обеспечивающей предсказуемость и упрощающую идентификацию сущностей.

### Предлагаемая структура проекта для управления коллекторами.



Рисунок 5 - файловая структура инвентаря источников

Создание инвентаря источников логов требует разработки строгой иерархической структуры, обеспечивающей уникальность идентификаторов. Предлагаемая файловая структура представлена на рисунке 5. Основной принцип заключается в использовании названий директорий как уникальных ключей, которые служат идентификаторами для коллекторов, топиков Kafka и других связанных сущностей, обеспечивая четкую привязку к конкретному источнику данных.

### Структура файла «main.yml»

В каждой директории, соответствующей конкретному коллектору, размещается файл «main.yml», содержащий метаинформацию об источнике, данные о развертывании, а также параметры входных и выходных каналов передачи данных. На основе содержимого данного файла осуществляется генерация необходимых конфигураций и создание соответствующих сущностей в инфраструктуре.

Листинг 1 - Пример файла с описанием коллектора



### Описание точки входа в программу

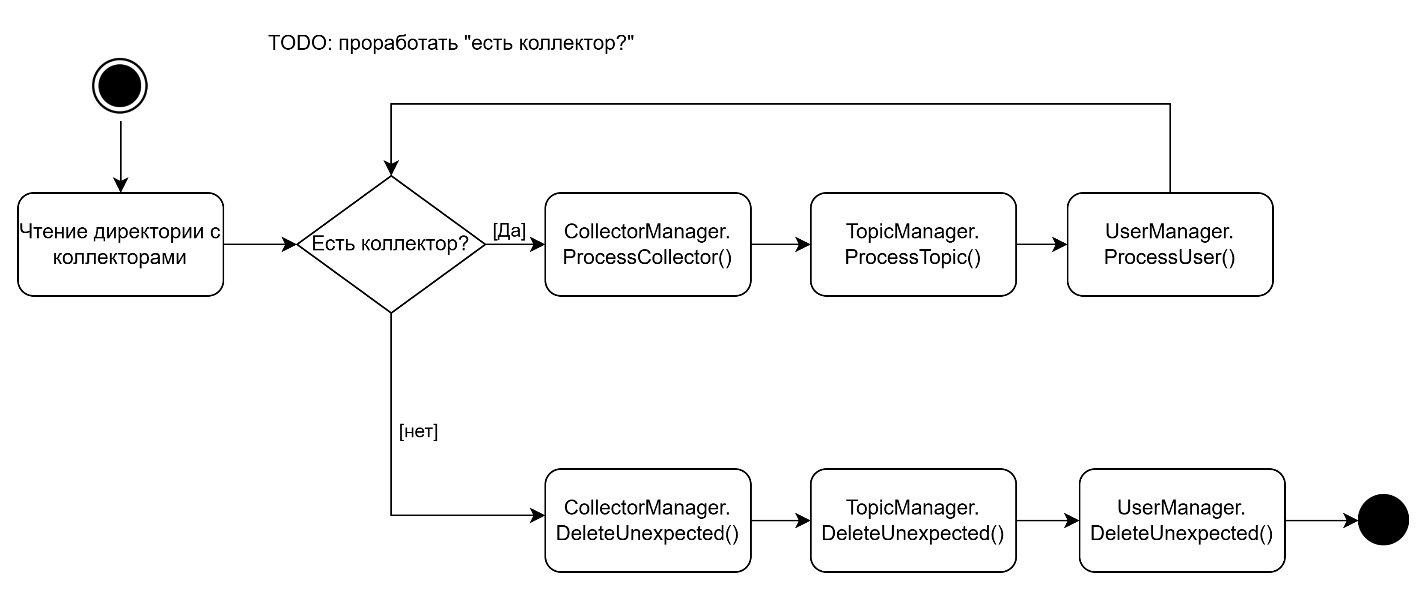


Рисунок 6 – Логика основного цикла программы

Логика работы основной программы, выступающей точкой входа, иллюстрируется на рисунке 6. Программа инициализирует экземпляры классов CollectorManager, TopicManager и UserManager (далее менеджеры), предназначенные для управления коллекторами, топиками Kafka и учетными записями соответственно. При инициализации в каждом объекте вызывается метод list\_existing() для получения текущего состояния сущностей, за которые ответственен конкретный менеджер. Это состояние сохраняется в поле existing\_entities соответствующего менеджера.

После инициализации сущностей запускается основной цикл программы, выполняющий итерацию по директориям с описанием сущностей определенного источника. Во время итераций происходит вызов функций, обрабатывающих файлы конфигурации. Содержимое файлов передается в методы Process<Entity>() (в качестве Entity может быть Topic, User или Collector) соответствующих объектов. По завершении основного цикла у каждого менеджера вызывается метод DeleteUnexpected(), выполняющий работу по удалению сущностей, к которым не происходило обращений во время выполнения основного цикла. Если к объекту не происходило обращений, это значит, что такой объект не был описан в директории с коллекторами или был описан некорректно и подлежит удалению.

### Автоматизация управления топиками

При вызове метода ProcessTopic() происходит проверка существования топика в словаре existing\_topics в объекте TopicManager. В случае, если топик не существует, выполняется его создание с помощью Post запроса в KaaS API c передачей таких параметров, как количество партиций, срок хранения сообщений по времени, максимальный объем памяти, занимаемый топиком, название топика и его описание. Важно отметить, что в описании передается несколько параметров с метаинформацией. Например, в описание попадает сгенерированный хэш от конкатенации значений технических параметров топика для возможности быстрой проверки соответствия параметров. Также в описание попадает информация, используемая для фильтрации топиков, управляемых данной программой автоматизации. Такой подход позволяет не внести неожиданных изменений в топики, созданные вручную или другой автоматикой.

В случае, если топик существует в словаре, происходит сравнение описания существующего топика и желаемого топика. Если описания не совпадают, подразумевается, что не совпадают хэши технических параметров, поэтому происходит отправка Patch запроса в KaaS API для изменения параметров топика на ожидаемые, затем устанавливается пометка в существующих топиках, что к топику было обращение, и он был изменен. Если описания совпадают – в существующих топиках устанавливается пометка, что к топику было обращение.

### Автоматизация управления пользователями

Метод ProcessUser() реализует логику управления учетными записями пользователей, аналогичную механизму, применяемому в методе ProcessTopic(). На первом этапе выполняется проверка наличия учетной записи в словаре existing\_entities объекта UserManager. Если учетная запись существует и ее описание совпадает с ожидаемым, устанавливается флаг обращения, подтверждающий актуальность записи. В случае расхождения описаний инициируется процесс перенастройки: отправляется PATCH-запрос к API сервиса KaaS для обновления параметров учетной записи, после чего выполняется попытка установки прав доступа (Permission) между пользователем и связанным топиком. Если в ответ на запрос возвращается код ответа 409, это интерпретируется как индикация уже установленного разрешения, и дальнейшие действия по настройке прав не предпринимаются. Код ответа 200 программа воспринимает как факт установки прав доступа. После завершения обновления устанавливается флаг обращения.

При отсутствии учетной записи в словаре existing\_entities выполняется ее создание посредством POST-запроса к API KaaS с передачей необходимых параметров. В описание учетной записи включается метаинформация, содержащая хэш технических параметров и данные для фильтрации, позволяющие идентифицировать учетные записи, управляемые данной системой автоматизации. Далее отправляется POST-запрос для создания соответствующего разрешения (Permission) в зависимости от роли пользователя: ConsumePermission для потребителей (consumers) или ProducePermission для производителей (producers).

### Автоматизация управления коллекторами

Для управления коллекторами в CollectorManager используется библиотека ansible\_runner, позволяющая выполнять Ansible-плейбуки из Python-кода. Получение информации о существующих коллекторах выполняется с помощью вызова Ansible-модуля shell с передачей команды docker ps --all --filter “management-label” –format “ID | STATUS | RUNNING\_FOR | NAME”. Такой вызов возвращает информацию о запущенных контейнерах с определенной меткой в формате таблицы, где столбцы разделены знаком «|». Полученные данные преобразуются в словарь existing\_collectors.

Во время вызова метода ProcessCollector() происходит проверка, существует ли коллектор с таким именем в словаре. Если существует – отметка об обращении. Если не существует – вызывается плейбук «deploy.yml» с логикой развертывания коллектора.

Важный элемент коллектора – его конфигурация. Она доставляется на хост во время выполнения метода ProcessCollector() также с помощью Ansible плейбука. До передачи конфигурации выполняется ее генерация в методе generateConfig(), где на основе Jinja2-шаблонов или файлов input/output формируется файл конфигурации для последующей установки на хост. Учитывая разнообразие типов коллекторов — включая сетевые и те, которые осуществляют чтение данных из баз данных через SQL-запросы, — файлы конфигурации адаптируются в зависимости от специфики каждого типа коллектора.

## Предоставление информации о развернутом экземпляре заинтересованным лицам

Чаще всего заинтересованным лицам (TH аналитики, команда, отвечающая за источник логов) нужно знать две вещи: «куда отправлять данные?» и «все ли работает?». На первый вопрос ответ дает единый источник правды, в котором должен быть указан адрес, по которому коллектор принимает данные. На второй вопрос должны отвечать системы мониторинга. Ключевые метрики для понимания, работает ли коллектор – скорость приема данных коллектором, скорость записи данных в кластер Kafka и наличие (количество) сообщений в Kafka. Если эти метрики не нулевые при отправке данных в коллектор, значит все работает хорошо. Удобным решением было бы собрать дашборд с указанными метриками, таким образом, заинтересованные лица смогли бы получать ответы на свои вопросы без участия команды SRE.

## Настройка CI/CD пайплайнов в репозитории

CI/CD пайплайн (англ. Continuous Integration / Continuous Delivery & Deployment Pipeline) - определение пайплайна.

В данном репозитории при внесении изменений мы ожидаем, что будет произведено достаточно много действий. Чтобы не усложнять решение за счет ограничений пайплайнов, все изменения будем производить в рамках одного шага – развертывания. На Merge Request будут пайплайны со статическими проверками кода и конфигураций, которые включают в себя проверки отсутствия коллизий занимаемых портов, проверки корректности существующих конфигураций коллекторов.

### Проверка корректности конфигурации

Перед развертыванием измененных конфигураций необходимо удостовериться, точно ли они корректны. Так как схема конфигурационных файлов была определена нами, необходимо написать линтер(ы) конфигураций. Для этой задачи можно воспользоваться готовыми инструментами или написать его вручную.

Если говорить о готовых инструментах, то есть те, что позволяют задать «схему файла» с определением того, что должно содержаться в том или ином поле конфигурационного файла. Это вариант для самой базовой проверки синтаксиса, она позволяет убедиться, что как минимум типы данных используются корректные.

Если необходимо проверять более сложные условия (например, уникальность значений ключа среди множества файлов или что-то сложнее), то без скриптов собственной разработки не обойтись. Чаще такие типы линтеров создаются на Python из-за простоты разработки.

### Применение изменений

Тут говорю про то, как запускается тот комплексный скрипт

# Реализация целевого процесса

## Автоматизация

Тут рассказываю про то, как реализовал логику применения состояния. Про то, что есть основной цикл, читающий директорию, про то, что он запускается внутри CI/CD или вручную, если необходимо, про то, что он авторизуется в KaaS, а также занимается деплоем и остановкой/удалением сущностей.

## Обновленный процесс

Показываю, как фактически теперь выглядит поставка источника на мониторинг.

## Сравнение предыдущего процесса и обновленного (

Тут показываю графики по времени, диаграммы было/стало, возможно еще какую-то табличку. Это должен быть последний раздел этой главы

Заключение

В рамках данной работы …..

Список использованных источников и литературы

1. What is SIEM? // Microsoft – [Б.м.]., 2025 – URL: <https://www.microsoft.com/en-us/security/business/security-101/what-is-siem>

1. Filebeat - агент из экосистемы Elastic Stack, написан на Go. Ссылка на страницу с описанием инструмента: <https://www.elastic.co/beats/filebeat>. [↑](#footnote-ref-1)
2. FluentD – коллектор данных из проекта CNCF (англ., Cloud Native Computing Foundation). Ссылка на страницу с описанием инструмента: <https://www.fluentd.org>. [↑](#footnote-ref-2)
3. Vector (vector.dev) – современный высокопроизводительный агент, написан на Rust. Ссылка на страницу с описанием инструмента: <https://vector.dev>. [↑](#footnote-ref-3)
4. 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-4)
5. Случай отказа в согласовании в сценариях, указанных на рисунке 2 не будет рассмотрен, так как таких ситуаций еще не возникало и нет предпосылок, чтобы они возникли. Отказ может возникнуть в случае проблем с какими-либо ресурсами, рассмотрение этого сценария не имеет смысла в данной работе. [↑](#footnote-ref-5)