Общая информация

Аудит представляет собой систему фиксации и хранения событий, происходящих в информационной системе. Компонента аудита обеспечивает регистрацию действий пользователей, изменений данных и системных операций для последующего анализа, расследования инцидентов и соблюдения требований регуляторов.

Назначение аудита

Компонента аудита используется для решения двух основных задач:

- Внутренний контроль;
- Аудит требований безопасности и регулятора.

1. Внутренний контроль обеспечивает:
    1. Мониторинг действий пользователей в системе;
    2. Выявление аномальной активности и потенциальных инцидентов безопасности;
    3. Расследование инцидентов и спорных ситуаций;
    4. Анализ использования системных ресурсов;
    5. Контроль соблюдения внутренних политик и процедур;
    6. Отслеживание изменений критичных бизнес-данных;
    7. Восстановление последовательности событий при технических сбоях.
2. Соответствие требованиям регуляторов включает:
    1. Выполнение требований законодательства;
    2. Предоставление доказательной базы для внешних аудиторов;
    3. Формирование отчетности для контролирующих органов;
    4. Обеспечение юридически значимого подтверждения действий пользователей;
    5. Демонстрация наличия системы контроля и управления доступом.

Объем фиксируемых событий

Компонента аудита фиксирует следующие категории событий:

Аутентификация и авторизация

- Вход пользователя в систему (успешный / не успешный);

- Выход пользователя из системы;
- Изменение прав доступа пользователя;
- Попытки несанкционированного доступа;
- Смена пароля и других учетных данных;

Операции с бизнес данными

- Создание новых документов, записей, сущностей
- Изменение существующих данных (с фиксацией измененных полей)
- Удаление данных
- Просмотр конфиденциальной информации ?
- Экспорт данных из системы ?

Системные операции

- Изменение конфигурации системы
- Выполнение административных операций
- Массовые операции (импорт, пакетные обновления)

Критичные бизнес процессы

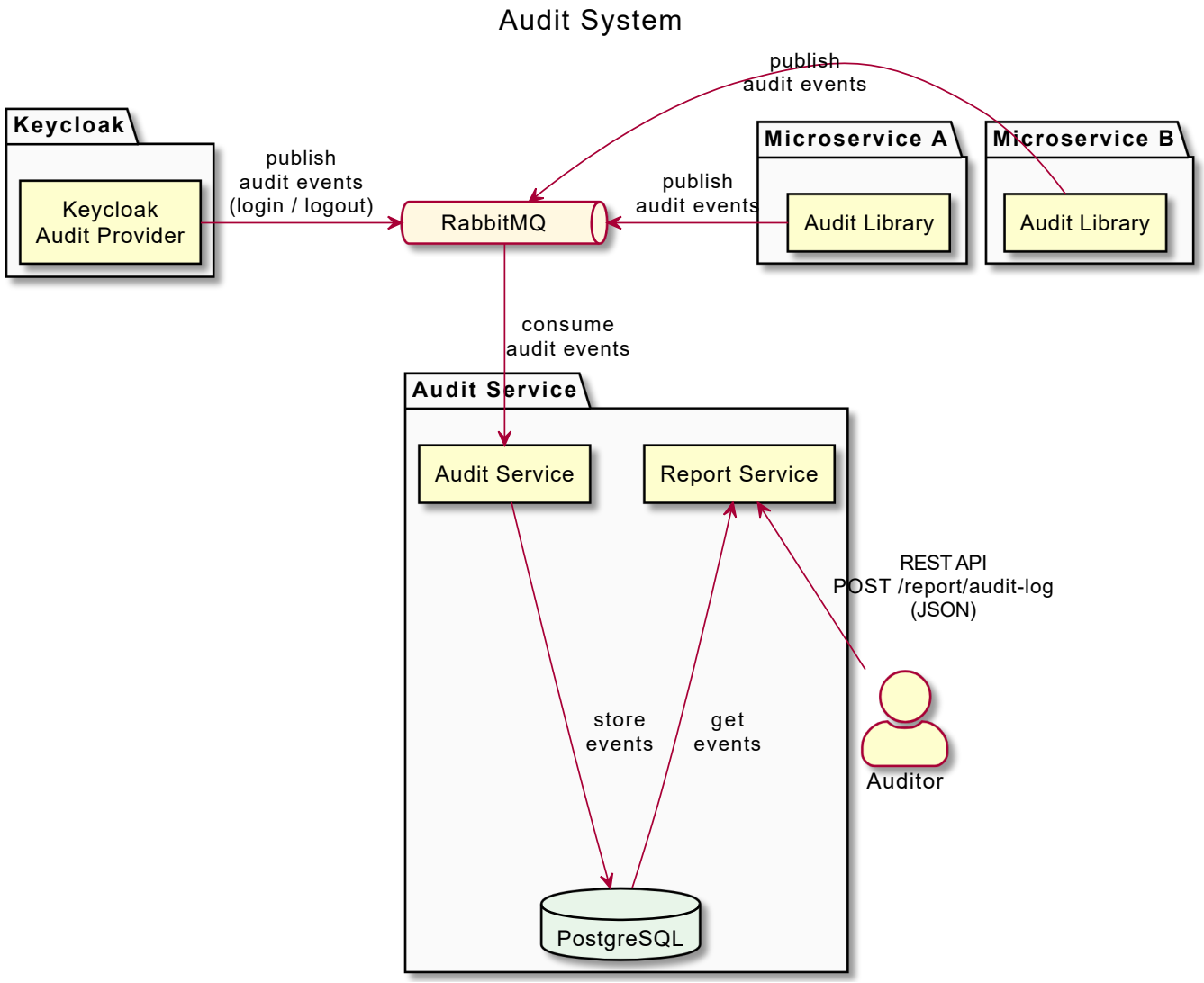
- Операции по сервисам (например, платежи, договоры)
- Изменение статусов документов
- Согласование и утверждение документов
- Формирование и печать отчетов

Хранение данных аудита

PostgreSQL выбран в качестве основного хранилища, поскольку:

- Обеспечивает гарантии доставки и транзакционность (единственная БД из рассмотренных вариантов которая имеет полное соответствие ACID). Полученный клиентом SUCCESS гарантирует сохранение записи в БД, благодаря чему проще отчитываться перед регуляторным аудитом.
- Не требует глубокой аналитики в реальном времени
- Простота внедрения и обслуживания
- При необходимости можно добавить Elasticsearch для полнотекстового поиска и дашбордов (через Debezium или параллельный поток)
- Архитектура: PostgreSQL - надежное хранилище, Elasticsearch -опциональный слой для быстрого поиска и визуализации
- Из рассмотренных вариантов это лидер по стоимости эксплуатации

Способ доставки событий аудита



Бизнес-сервисы при выполнении операций не записывают аудит напрямую в БД или REST-сервис,а публикуют события (audit events) в брокер сообщений - Kafka или RabbitMQ.
Сервис аудита выступает консьюмером и сохраняет сообщения в базу данных.

1. Минимальное влияние на бизнес-логику.

Отправка события в брокер - асинхронная и быстрая операция. Это не увеличивает время отклика бизнес-сервисов и не блокирует выполнение основной логики, в отличие от REST-запросов к внешнему сервису аудита.

Гарантия доставки и надёжность. Kafka и RabbitMQ обеспечивают:

- долговременную очередь с подтверждением доставки (ack);
- возможность повторной обработки при сбое сервиса аудита;
- защиту от потери данных при перезапуске.

2. Масштабируемость

Добавление новых источников аудита (новых микросервисов) не требует изменения логики уже существующих:

- сервис просто публикует сообщение в брокер;
- сервис аудита может быть масштабирован горизонтально и читать данные параллельно.

3. Гибкость архитектуры

Асинхронная интеграция через брокер - это event-driven архитектура, которая:

- упрощает построение централизованной системы аудита;
- позволяет реализовать разные консьюмеры для разных целей (например, сохранение в БД, потоковая аналитика, уведомления);

4. Изоляция отказов

Сбой сервиса аудита не влияет на работу основного приложения:

- сервис продолжает публиковать события в брокер;
- сообщения сохраняются до восстановления аудита.

Формат и структура записей аудита

Минимальный набор полей журнала аудита:

Поле	Описание
id	Уникальный идентификатор записи (ПК)
event_parent_id	Идентификатор родительского события (для цепочек действий)
event_child_id	Идентификатор дочернего события
event_time	Время возникновения события (timestamp with time zone)
event_type	Тип действия (CREATE, UPDATE, DELETE, LOGIN, LOGOUT, VIEW, EXPORT, APPROVE, REJECT и т.д.)
entity_type	Тип сущности (Document, User, Contract, Payment и т.д.)
entity_id	Идентификатор сущности
user_id	Уникальный идентификатор пользователя
user_login	Логин пользователя
user_name	Человекочитаемое имя пользователя (ФИО)

Поле	Описание
user_role	Роль пользователя на момент события (или список ролей/прав)
source_system	Название сервиса, отправившего событие
ip_address	IP-адрес клиента
user_agent	Браузер/клиентское приложение
details	Полное тело события в формате JSONB (payload, измененные поля, дополнительная информация)
result_status	Результат операции (SUCCESS, FAILURE, ERROR)
hash_chain	Криптографический хэш текущей записи для контроля целостности
previous_hash	Криптографический хэш предыдущей записи (участвует в формировании hash_chain)
create_time	Время приема события системой аудита

Обоснование структуры

- Набор полей покрывает основные вопросы аудита: КТО, ЧТО, КОГДА, ГДЕ, С ЧЕМ
- JSONB-поле details обеспечивает гибкость без изменения схемы
- Поля связей (parent/child) позволяют строить цепочки событий
- Хэш обеспечивает защиту от модификации записей
- Разделение event_time и create_time позволяет выявлять задержки доставки

Доступ и использование аудита

Архитектура

Разрабатывается отдельный сервис с собственным бэкендом и фронтендом для работы с журналом аудита.

Компоненты сервиса аудита

BACKEND

- Прием событий от бизнес-сервисов (брокер сообщений)
- Сохранение в хранилище (PostgreSQL)
- API для поиска и фильтрации событий

- Экспорт данных в различных форматах (CSV, Excel, PDF)

FRONTEND (WEB-интерфейс)

- Поиск и фильтрация событий по всем параметрам
- Просмотр детальной информации о событии
- Построение цепочек связанных событий
- Конструктор отчетов
- Шаблоны типовых запросов

Функции

Поиск и фильтрация

- Поиск по пользователю, дате, типу события, сущности
- Полнотекстовый поиск по содержимому (при подключении Elasticsearch)
- Сохранение часто используемых фильтров

Отчёты

- Действия пользователя за период
- История изменений документа
- Аудит операций по сервису
- Отчеты для регуляторов
- Экспорт в различных форматах

Контроль доступа

- Разграничение прав на просмотр аудита
- Аудит доступа к самому журналу аудита ?
- Ролевая модель (администратор безопасности, аудитор, бизнес-пользователь)

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Инфраструктура системы аудита PostgreSQL

Infrastructure of Audit System (PostgreSQL)

Документация описывает пошаговое развёртывание инфраструктуры аудита в PostgreSQL.

Содержание

- Схема, пользователь, права
- Таблица, индексы
- Установка и настройка PG_PARTMAN
- Автоматизация создания партиций
- Защита от изменения данных
- Таблицы и права для генерации цепочки целостности логов аудита

1. Схема, пользователь и права доступа

Создаём изолированную схему `audit` и пользователя `audit_usr` с минимально необходимыми правами.

```
-- Создаём схему audit
CREATE SCHEMA IF NOT EXISTS audit;

-- Создаём пользователя audit_usr
CREATE USER audit_usr WITH PASSWORD 'audit_pwd';

-- Предоставляем доступ только к схеме audit
GRANT USAGE ON SCHEMA audit TO audit_usr;

-- Запрет создания объектов в схеме audit
REVOKE CREATE ON SCHEMA audit FROM audit_usr;

-- Настройка прав для будущих таблиц
ALTER DEFAULT PRIVILEGES IN SCHEMA audit
    GRANT SELECT, INSERT ON TABLES TO audit_usr;

ALTER DEFAULT PRIVILEGES IN SCHEMA audit
    REVOKE UPDATE, DELETE ON TABLES FROM audit_usr;

-- Настройка прав для будущих последовательностей

```

```
22 ALTER DEFAULT PRIVILEGES IN SCHEMA audit
       GRANT USAGE, SELECT ON SEQUENCES TO audit_usr;
```

2. Таблица аудита и индексы

Последовательность

```
CREATE SEQUENCE audit.audit_log_seq
    START WITH 1
    INCREMENT BY 1000
    CACHE 100;
```

Таблица audit.audit_log

```
CREATE TABLE audit.audit_log (
    id              BIGINT NOT NULL DEFAULT nextval('audit.audit_log_seq'),
    event_parent_id UUID        NOT NULL,                   -- Уникальный иден
    event_child_id  UUID        NOT NULL,                   -- Уникальный иден
    event_time      TIMESTAMP   NOT NULL DEFAULT now(),     -- Время события
    event_type      VARCHAR(255) NOT NULL,                  -- Тип действия (CI
    entity_type     VARCHAR(255),                           -- Тип сущности (D
    entity_id       VARCHAR(255),                           -- Идентификатор с
    user_id         VARCHAR(255),                           -- Пользователь, с
    user_name       VARCHAR(255),                           -- Человекочитаемо
    user_role       VARCHAR(500),                           -- Роль
    source_system   VARCHAR(255) NOT NULL,                  -- Откуда пришло с
    ip_address      INET,                                   -- IP адрес пользо
    user_agent      VARCHAR(255),                           -- Б
    result_status   VARCHAR(100),                           -- Результат обраб
    details         JSONB,                                  -- Полное тело соб
    hash            BYTEA,                                  -- Хэш записи (для
    create_time     TIMESTAMP NOT NULL DEFAULT now()        -- Время вставки
) PARTITION BY RANGE (create_time);
```

Индексы

```
CREATE INDEX idx_audit_user_id ON audit.audit_log(user_id);
CREATE INDEX idx_audit_user_name ON audit.audit_log(user_name);
CREATE INDEX idx_audit_event_time ON audit.audit_log(event_time);
CREATE INDEX idx_audit_event_parent_id ON audit.audit_log(event_parent_id);
CREATE INDEX idx_audit_event_type ON audit.audit_log(event_type);
CREATE INDEX idx_audit_entity_type ON audit.audit_log(entity_type, entity_id);
CREATE INDEX idx_audit_source_system ON audit.audit_log(source_system);
CREATE INDEX idx_audit_create_time ON audit.audit_log USING BRIN(create_time);
```

Таблица audit.audit_hb_kind

```
CREATE TABLE audit.audit_hb_kind (
    id              BIGINT PRIMARY KEY,       -- Уникальный идентификатор запи
    sys_name        VARCHAR(255) NOT NULL,    -- Системное имя справочника
    name            VARCHAR(255)              -- Наименование справочника
);
INSERT INTO audit.audit_hb_kind (id, sys_name, name) VALUES (1, 'eventType', '
INSERT INTO audit.audit_hb_kind (id, sys_name, name) VALUES (2, 'resultStatus'
```

Таблица audit.audit_hb

```
CREATE TABLE audit.audit_hb (
    id              BIGSERIAL PRIMARY KEY,    -- Уникальный идентификатор запи
    sys_name        VARCHAR(255) NOT NULL,    -- Системное имя параметра
    name            VARCHAR(255),             -- Наименование параметра
    hb_kind_id      BIGINT NOT NULL           -- Идентификатор справочника (au
);
INSERT INTO audit.audit_hb (sys_name, name, hb_kind_id)
VALUES ('LOGIN', 'Вход пользователя', 1),
       ('LOGIN_ERROR', 'Ошибка входа пользователя', 1),
       ('LOGOUT', 'Выход пользователя', 1),
       ('LOGOUT_ERROR', 'Ошибка выхода пользователя', 1),
       ('REGISTER', 'Регистрация пользователя', 1),
       ('REGISTER_ERROR', 'Ошибка регистрации пользователя', 1),
       ('GRANT_CONSENT', 'Предоставление согласия', 1),
       ('GRANT_CONSENT_ERROR', 'Ошибка предоставления согласия', 1),
       ('UPDATE_CONSENT', 'Обновление согласия', 1),
       ('UPDATE_CONSENT_ERROR', 'Ошибка обновления согласия', 1),
       ('REVOKE_GRANT', 'Отзыв согласия', 1),
       ('REVOKE_GRANT_ERROR', 'Ошибка отзыва согласия', 1),
       ('SEND_VERIFY_EMAIL', 'Отправка подтверждения по эл.почте', 1),
       ('SEND_VERIFY_EMAIL_ERROR', 'Ошибка отправки подтверждения по эл.почте'
       ('SEND_RESET_PASSWORD', 'Отправка запроса на сброс пароля', 1),
       ('SEND_RESET_PASSWORD_ERROR', 'Ошибка отправки запроса на сброс пароля'
       ('RESET_PASSWORD', 'Сброс пароля', 1),
       ('RESET_PASSWORD_ERROR', 'Ошибка сброса пароля', 1),
       ('USER_CREATED', 'Пользователь создан', 1),
       ('USER_UPDATED', 'Пользователь обновлён', 1),
       ('USER_DELETED', 'Пользователь удалён', 1),
       ('DATA_ACCESS', 'Доступ к данным', 1),
       ('DATA_CREATED', 'Создание данных', 1),
       ('DATA_UPDATED', 'Обновление данных', 1),
       ('DATA_DELETED', 'Удаление данных', 1),
       ('DATA_EXPORT', 'Экспорт данных', 1),
       ('DATA_IMPORT', 'Импорт данных', 1),
       ('PROCESS_STARTED', 'Начало процесса', 1),
       ('PROCESS_STEP', 'Шаг процесса', 1),
```

```
        ('PROCESS_COMPLETED', 'Завершение процесса', 1),
        ('PROCESS_FAILED', 'Ошибка во время процесса', 1),
        ('PROCESS_CANCELLED', 'Отмена процесса', 1),
        ('SCHEDULED_JOB_EXECUTED', 'Успешное выполнение регулярного задания', 1
        ('SCHEDULED_JOB_FAILED', 'Ошибка при выполнении регулярного задания', 1
        ('API_CALL', 'Вызов API', 1),
        ('API_CALL_FAILED', 'Ошибка вызова API', 1);

INSERT INTO audit.audit_hb (sys_name, name, hb_kind_id)
VALUES ('SUCCESS', 'Операция выполнена успешно', 2),
       ('FAILURE', 'Операция завершилась с ошибкой', 2),
       ('IN_PROGRESS', 'Операция в процессе выполнения', 2),
       ('CANCELLED', 'Операция отменена пользователем или системой', 2),
       ('SKIPPED', 'Операция пропущена', 2),
       ('ACCESS_DENIED', 'Доступ запрещен', 2),
       ('TIMEOUT', 'Операция завершилась по таймауту', 2);
```

3. Установка и настройка расширения PG_PARTMAN

Создание схемы и установка расширения

```
-- Создаём отдельную схему, чтобы служебные объекты не пересекались с бизнес-об
CREATE SCHEMA IF NOT EXISTS partman;

-- Устанавливаем расширение
CREATE EXTENSION pg_partman WITH SCHEMA partman;
```

> **Если расширение недоступно**, установите его вручную в контейнере:
>
> ```
> docker exec -it <container_name> bash
> apt-get update
> apt-get install -y postgresql-18-partman
> # apt-get install -y postgresql-16-partman  # для PostgreSQL 16.*
> ```

Шаблон таблицы и регистрация партиций

```
-- Создаём шаблон, по которому будут создаваться партиции
CREATE TABLE partman.audit_log_template (LIKE audit.audit_log INCLUDING ALL);

-- Регистрируем таблицу в pg_partman: партиции по неделям, 3 партиции создаются
SELECT partman.create_parent(
    p_parent_table   := 'audit.audit_log',
    p_control        := 'create_time',
```

```
    p_interval       := '1 week',
    p_premake        := 3,
    p_template_table := 'partman.audit_log_template'
);
```

4. Автоматизация создания партиций через BGW

`pg_partman` поддерживает фоновый рабочий процесс (Background Worker) для автоматического обслуживания партиций.

Шаг 1 - Проверить текущую конфигурацию

```
SHOW shared_preload_libraries;
```

Если результат пустой - необходимо настроить BGW.

Шаг 2 - Найти файл конфигурации

```
SHOW config_file;
```

Шаг 3 - Отредактировать `postgresql.conf`

Найдите строку `shared_preload_libraries`, раскомментируйте и добавьте `pg_partman_bgw`:

```
shared_preload_libraries = 'pg_partman_bgw'
# Если уже есть другие библиотеки - добавьте через запятую:
# shared_preload_libraries = 'other_lib,pg_partman_bgw'
```

Ниже добавьте блок конфигурации BGW:

```
# BGW pg_partman
pg_partman_bgw.role     = 'postgres'   # пользователь для запуска обслуживания
pg_partman_bgw.dbname   = 'postgres'   # база данных с партициями
pg_partman_bgw.interval = 3600         # интервал запуска в секундах (раз в ча
```

Шаг 4 - Перезапустить инстанс БД

```
docker restart <container_name>
```

Шаг 5 — Проверить результат

```
SHOW shared_preload_libraries;
-- Ожидаемый вывод: pg_partman_bgw
```

5. Защита от изменений (иммутабельность аудита)

Создаём триггер, который запрещает любые операции `UPDATE` и `DELETE` в таблице аудита.

```
-- Функция-обработчик триггера
CREATE OR REPLACE FUNCTION audit.prevent_audit_modifications()
    RETURNS trigger AS $$
BEGIN
    RAISE EXCEPTION 'Audit log is immutable: % operation is not allowed', TG_OI
END;
$$ LANGUAGE plpgsql;

-- Триггер на таблице аудита
CREATE TRIGGER audit_no_modifications
    BEFORE UPDATE OR DELETE ON audit.audit_log
    FOR EACH ROW EXECUTE FUNCTION audit.prevent_audit_modifications();
```

6. Таблицы, пользователь и права доступа для генерации цепочки целостности логов аудита

Таблицы и индексы

Таблица `audit.audit_block_job` содержит информацию о зарегистриованных джобах и их детализацию

```
CREATE TABLE audit.audit_block_job (
    block_id BIGINT PRIMARY KEY NOT NULL,
    from_time TIMESTAMP NOT NULL,
    to_time TIMESTAMP NOT NULL,
    status VARCHAR(20) NOT NULL,
    worker_id VARCHAR(100),
    row_count BIGINT,
    block_hash BYTEA,
    chain_hash BYTEA,
    previous_chain_hash BYTEA,
    started_at TIMESTAMP,
    completed_at TIMESTAMP,
    created_at TIMESTAMP DEFAULT now()
);
```

Индекс `idx_audit_block_job_status_block_id` для оперативного поиска заданий

```
CREATE INDEX IF NOT EXISTS idx_audit_block_job_status_block_id
    ON audit.audit_block_job (status, block_id);
```

Таблица `audit.audit_chain_state` с детализацией о последнем выполненном задании, чтобы избежать дорого поиска по таблице `audit.audit_block_job`

```
CREATE TABLE audit.audit_chain_state (
    id INT PRIMARY KEY NOT NULL,
    last_finalized_block BIGINT NOT NULL,
    last_chain_hash BYTEA,
    last_allocated_to_time TIMESTAMP,
    last_allocated_block BIGINT NOT NULL
);

INSERT INTO audit.audit_chain_state (id, last_finalized_block, last_chain_hash
VALUES (1, 0, NULL, 0)
ON CONFLICT DO NOTHING;
```

Таблица `audit.shedlock` для атомарности запуска воркером. Воркеры по созданию заданий и их финализации (склеивания прошлого и текущего хэша записи)

```
CREATE TABLE audit.shedlock (
    name VARCHAR(64) PRIMARY KEY NOT NULL,
    lock_until TIMESTAMP NOT NULL,
    locked_at TIMESTAMP NOT NULL,
    locked_by VARCHAR(255) NOT NULL
);
```

Пользователь

Создаём пользователя `audit_chain_usr` с минимально необходимыми правами

```
-- Создаём пользователя audit_chain_usr
CREATE USER audit_chain_usr WITH PASSWORD 'audit_chain_pwd';

-- Предоставляем доступ только к схеме - audit
GRANT USAGE ON SCHEMA audit TO audit_chain_usr;

-- Запрет создания объектов в схеме - audit
REVOKE CREATE ON SCHEMA audit FROM audit_chain_usr;

GRANT SELECT, INSERT, UPDATE ON TABLE audit.audit_block_job TO audit_chain_usr
GRANT SELECT, INSERT, UPDATE ON TABLE audit.audit_chain_state TO audit_chain_u
```

```
13 GRANT SELECT ON TABLE audit.audit_log TO audit_chain_usr;
   GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE audit.shedlock TO audit_chain_usr
```

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Провайдер аудита для Keycloak

Keycloak Audit Event Listener Provider

Кастомный провайдер для Keycloak, который обеспечивает перехват событий аутентификации, авторизации и управления пользователями и публикует их в RabbitMQ.

Провайдер реализует интерфейс `EventListenerProvider` (Keycloak SPI) и выступает посредником между встроенной системой событий Keycloak и брокером сообщений RabbitMQ.

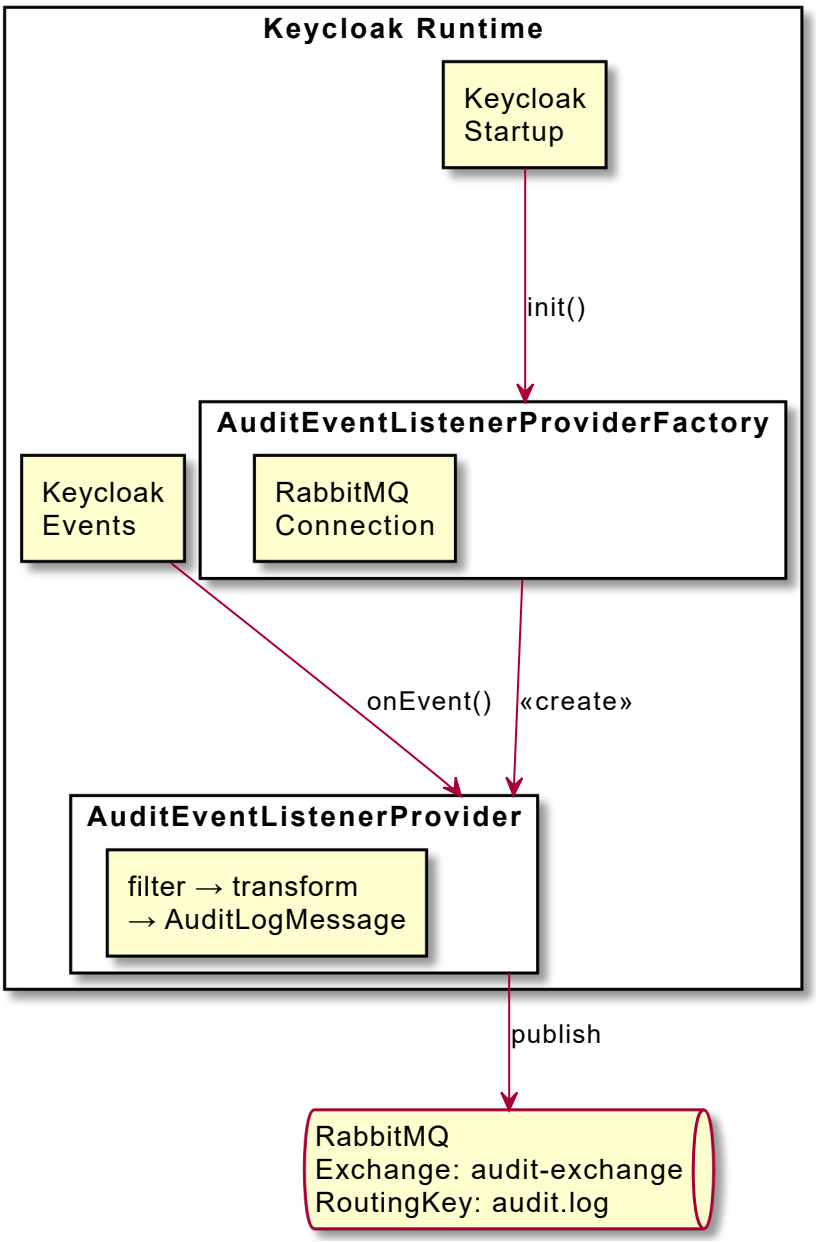
Что делает провайдер:

1. Перехватывает пользовательские события (логин, логаут, регистрация, сброс пароля и др.)
2. Перехватывает административные события (создание, обновление, удаление пользователей)
3. Фильтрует только релевантные для аудита события
4. Трансформирует нативные события Keycloak в унифицированный формат `AuditLogMessage`
5. Публикует сообщения в RabbitMQ

Содержание

- Архитектура
- Требования
- Сборка
- Конфигурация
- Обрабатываемые события
- Формат сообщения
- Структура проекта

Архитектура



Жизненный цикл:

Keycloak запускается

1. Происходит обнаружение AuditEventListenerProviderFactory через SPI
2. Factory.init() -> подключается к RabbitMQ
3. На каждый запрос - factory.create() -> новый Provider
4. provider.onEvent(...) -> публикует событие в RabbitMQ

Keycloak останавливается

1. Factory.close() -> закрывает соединение с RabbitMQ

Требования

- Java 21+

- Keycloak 26.2.3+
- RabbitMQ
- Maven 3.8+

Сборка

```
mvn clean package
```

Результат: `target/audit-keycloak-provider-1.0.0-SNAPSHOT.jar`

Сборка создаёт **shaded JAR** - все зависимости (RabbitMQ AMQP client) упакованы внутрь одного файла. Зависимости Keycloak (`keycloak-core` , `keycloak-server-spi` , `keycloak-services`) предоставляемые самим Keycloak, в JAR не включены.

Конфигурация

Провайдер конфигурируется через переменные окружения.

Переменные окружения

Переменная	Описание	Значение по умолчанию
`RABBITMQ_HOST`	Хост RabbitMQ	
`RABBITMQ_PORT`	Порт RabbitMQ	
`RABBITMQ_USERNAME`	Имя пользователя RabbitMQ	
`RABBITMQ_PASSWORD`	Пароль RabbitMQ	
`RABBITMQ_EXCHANGE`	Наименование exchange	audit-exchange
`RABBITMQ_ROUTING_KEY`	Routing key для публикации	audit.log

Установка и деплой

1. Скопировать JAR в директорию providers Keycloak

```
cp target/audit-keycloak-provider-1.0.0-SNAPSHOT.jar $KEYCLOAK_HOME/providers/
```

2. Задать переменные окружения

Установить переменные из раздела Конфигурация перед запуском Keycloak.

3. Включить провайдер в Keycloak

1. Открыть Keycloak Admin Console
2. Выбрать необходимый Realm
3. Перейти: **Realm Settings** → **Events** → **Event listeners**
4. Добавить `audit-event-listener`
5. Сохранить

4. Запустить Keycloak



NOTE: 1 пункт будет удалён после внедрения в процесс деплоя keycloak стадии загрузки артефактов из nexus

Обрабатываемые события

Пользовательские события (`onEvent`)

Тип события	Описание
`LOGIN`	Успешный вход
`LOGIN_ERROR`	Ошибка входа
`LOGOUT`	Успешный выход
`LOGOUT_ERROR`	Ошибка выхода
`REGISTER`	Успешная регистрация
`REGISTER_ERROR`	Ошибка регистрации
`GRANT_CONSENT`	Предоставление согласия
`GRANT_CONSENT_ERROR`	Ошибка предоставления согласия
`UPDATE_CONSENT`	Обновление согласия
`UPDATE_CONSENT_ERROR`	Ошибка обновления согласия
`REVOKE_GRANT`	Отзыв согласия

Тип события	Описание
`REVOKE_GRANT_ERROR`	Ошибка отзыва согласия
`SEND_VERIFY_EMAIL`	Отправка письма подтверждения email
`SEND_VERIFY_EMAIL_ERROR`	Ошибка отправки письма подтверждения
`SEND_RESET_PASSWORD`	Отправка письма сброса пароля
`SEND_RESET_PASSWORD_ERROR`	Ошибка отправки письма сброса пароля
`RESET_PASSWORD`	Успешный сброс пароля
`RESET_PASSWORD_ERROR`	Ошибка сброса пароля

Административные события (`onAdminEvent`)

Обрабатываются только события с типом ресурса `USER` :

Операция	Тип в аудит-сообщении
`CREATE`	`USER_CREATED`
`UPDATE`	`USER_UPDATED`
`DELETE`	`USER_DELETED`

Формат сообщения

Каждое событие публикуется в RabbitMQ.

Схема

Поле	Тип	Описание
`eventParentId`	UUID	UUID родительского события в цепочке
`eventChildId`	UUID	UUID текущего события
`eventTime`	LocalDateTime	Время события источника (ISO-8601)

Поле	Тип	Описание
`eventType`	String	Тип события (см. таблицу выше)
`entityType`	String	Тип сущности (всегда `"User"`)
`entityId`	String	Идентификатор сущности
`userId`	String	Идентификатор пользователя
`userName`	String	Имя пользователя
`userRole`	String / null	Роли пользователя (не заполняется)
`sourceSystem`	String	Система-источник события (всегда `"keycloak"`)
`ipAddress`	String	IP-адрес клиента
`userAgent`	String / null	User-Agent клиента (не заполняется)
`resultStatus`	String	Результат операции
`details`	Object	Дополнительный контекст события

Пример сообщения (успешный логин)

```
{
  "eventParentId": "550e8400-e29b-41d4-a716-446655440000",
  "eventChildId": "6ba7b810-9dad-11d1-80b4-00c04fd430c8",
  "eventTime": "2025-03-30T10:15:30",
  "eventType": "LOGIN",
  "entityType": "User",
  "entityId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "userId": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "userName": "demo",
  "userRole": null,
  "sourceSystem": "keycloak",
  "ipAddress": "192.168.1.13",
  "userAgent": null,
  "resultStatus": "SUCCESS",
  "details": {
    "auth_method": "openid-connect",
    "auth_type": "code",
    "code_id": "348358f6-f14e-4b84-be04-f0e836341577",
    "consent": "no_consent_required",
```

```
    "realm_name": "distr-demo",
    "realmId": "bf7f7ae8-875e-415e-aed7-01ea3aaf3a32",
    "clientId": "web-app",
    "sessionId": "348358f6-f14e-4b84-be04-f0e836341577",
    "username": "demo"
  },
  "createTime": "2025-03-30T10:15:30"
}
```

Структура проекта

Класс	Описание
`AuditEventListenerProviderFactory`	SPI-фабрика: создаёт провайдер, управляет жизненным циклом соединения
`AuditEventListenerProvider`	Перехватывает события, фильтрует, преобразовывает, вызывает publisher
`RabbitMqAuditPublisher`	Сериализует `AuditLogMessage` и публикует в RabbitMQ
`AuditLogMessage`	DTO - унифицированная структура аудит-сообщения
`AuditEventType`	Константы типов событий аудита

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Библиотека аудита

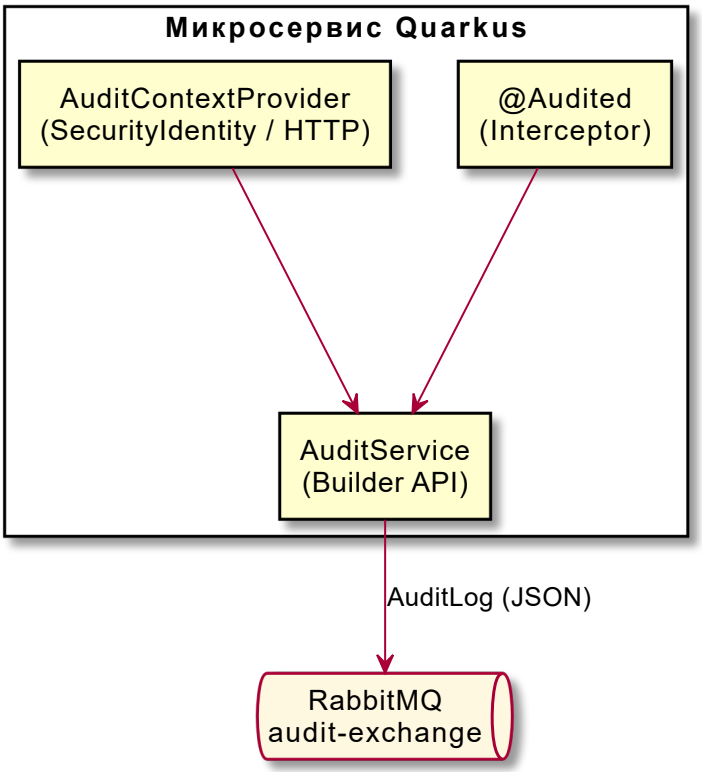
Audit Library

Библиотека для формирования и отправки сообщений аудита в Quarkus микросервисах. Обеспечивает сбор событий безопасности, доступа к данным и бизнес-операций с их асинхронной передачей через RabbitMQ или Kafka.

Содержание

- Архитектура
- Требования
- Подключение
- Использование
- Справочник
- Связывание событий между сервисами

Архитектура



Библиотека автоматически извлекает из текущего запроса:

- контекст безопасности - `userId`, `userName`, `userRole`, `sessionId` из JWT/OIDC;
- сетевые данные клиента - IP-адрес с поддержкой `X-Forwarded-For` и `X-Real-IP`, `User-Agent` из HttpServerRequest;
- идентификатор родительского события - `eventParentId` для трассировки цепочки вызовов между сервисами.

Модули

Модуль	Описание
`audit-base`	Базовые интерфейсы, модели и константы
`audit-rabbit`	Полная реализация с отправкой событий в RabbitMQ

Требования

- Java 21+
- Quarkus 3.19.4+
- RabbitMQ 3.13+
- Maven 3.9.12+

Подключение

1. Конфигурация микросервисов

1.1. Зависимости

Добавьте зависимость в `pom.xml` вашего микросервиса:

```
<dependency>
    <groupId>com.bivgroup.lib</groupId>
    <artifactId>audit-rabbit</artifactId>
    <version>${revision}</version>
</dependency>
```

1.2. Конфигурация

В application.properties вашего микросервиса в параметрах config-maps укажите:

```
quarkus.kubernetes-config.config-maps=\
  common-rabbit-audit-cfg
```

если сервис работает вне кластера k8s тогда в параметрах application.properties необходимо указать:

```
audit.rabbitmq-host=<rabbitmq_hostname>
audit.rabbitmq-port=<port>
audit.rabbitmq-username=<username>
audit.rabbitmq-password=<password>

mp.messaging.outgoing.audit-log.connector=smallrye-rabbitmq
mp.messaging.outgoing.audit-log.host=${audit.rabbitmq-host}
mp.messaging.outgoing.audit-log.port=${audit.rabbitmq-port}
mp.messaging.outgoing.audit-log.username=${audit.rabbitmq-username}
mp.messaging.outgoing.audit-log.password=${audit.rabbitmq-password}
mp.messaging.outgoing.audit-log.exchange.name=audit-exchange
mp.messaging.outgoing.audit-log.exchange.declare=false
mp.messaging.outgoing.audit-log.default-routing-key=audit.log
mp.messaging.outgoing.audit-log.publish-confirms=true
```

По умолчанию ошибки возникающие во время отправки лога аудита в брокер сообщений пробрасываются в вызывающий метод что приводит к прерыванию бизнес логики, если необходимо подавлять ошибки (т.е мы идём на риск потери логов аудита в угоду выполнения бизнес операций) необходимо в конфигурации задать параметр:

```
lib.audit.producer.bypass-mode=true
```

По умолчанию для логирования параметров `UserId` и `SessionID` используются JWT claim `sub` и `sid` соответственно, в случае необходимости можно их переопределить в конфигурации:

```
lib.audit.jwt.user-id=<user id claim>
lib.audit.jwt.session-id=<session id claim>
```

Использование

С использованием построителя (builder) сообщения

```
auditService.audit()
    .eventType(AuditEventType.DATA_UPDATED)
    .entityType("Contract")
    .entityId(contract.getId())
    .resultStatus(AuditResultStatus.SUCCESS)
    .addDetail("previousStatus", contract.getOldStatus())
    .addDetail("newStatus", contract.getNewStatus())
    .send();
```

Метод `.send()` асинхронно публикует событие в RabbitMQ.
Метод `.build()` возвращает объект построителя `AuditLog` без отправки.

Отправка статуса при обработке бизнес логики

```
@ApplicationScoped
public class UserService {

    @Inject
    AuditService auditService;

    public void loginUser(String userId) {
        AuditService.AuditBuilder audit = auditService.audit()
                .eventType(AuditEventType.LOGIN)
                .entityType("User")
                .entityId(userId)
                .resultStatus(AuditResultStatus.SUCCESS);
        try {
            // ... бизнес-логика ...
            audit.resultStatus(AuditResultStatus.SUCCESS).send();
        } catch (Exception e) {
            audit.resultStatus(AuditResultStatus.FAILURE).send();
            throw e;
        }
    }
}
```

Аннотация @Audited

Декларативный способ аудита методов без изменения бизнес-логики.
Перехватчик (Interceptor) автоматически создаёт и отправляет событие при вызове метода.

```
@Audited(
        eventType = AuditEventType.DATA_ACCESS,
        entityType = "Report",
        entityId = "#reportId"
)
public ReportDto getReport(Long reportId) {
    return reportRepository.findById(reportId);
}
```

Динамические значения через выражения

Поддерживается синтаксис `#gridDataRequest.getGridViewId` для извлечения данных из аргументов метода.
Где `#gridDataRequest` соответствует имени аргумента метода, а `getGridViewId` название метода получения параметра.

```
@ApplicationScoped
public class GridDataService {

```

```
4
5       @Audited(
6               eventType = AuditEventType.DATA_ACCESS,
7               entityType = "#gridDataRequest.getEntityBrief",
8               entityId = "#gridDataRequest.getGridViewId",
9               details = {
10                      "userFilter:#gridDataRequest.getUserFilter",
11                      "programFilter:#gridDataRequest.getProgramFilter"
12              },
13              logResponse = true
14      )
15      public GridDataResponse list(GridDataRequest gridDataRequest) {
16          return internalList(gridDataRequest, false);
17      }
    }
```

Параметр logResponse = `true` отвечает за автоматическое сохранение ответа (результата обработки метода) в объект details с ключом - `response` . По умолчанию logResponse = `false` .

> **Note:** Если метод выбрасывает исключение, статус события автоматически устанавливается в `FAILURE` .
> В противном случае - `SUCCESS` .

Валидация параметров указанных при объявлении аннотации @Audited

Для исключения ошибок связанных с некорректным указанием методов вида - `#gridDataRequestt.getGridViewId` был написан процессор `com.bivgroup.lib.audit.processor.AuditedAnnotationProcessor` , который на этапе компиляции производит сверку на корректность наименования объектов и их методов.

> **Note:** Если в библиотеке или сервисе в плагине - `org.apache.maven.plugins:maven-compiler-plugin` присутствует явный список процессоров (тег - `annotationProcessorPaths`) то для запуска процессора для аннотации `@Audited` необходимо также добавить путь к артефакту `com.bivgroup.lib:audit-base` , как это сделано в примере ниже.

```
1   <plugin>
2       <groupId>org.apache.maven.plugins</groupId>
3       <artifactId>maven-compiler-plugin</artifactId>
4       <version>${compiler-plugin.version}</version>
5       <configuration>
6           <source>${maven.compiler.source}</source>
7           <target>${maven.compiler.target}</target>
8           <annotationProcessorPaths>
9               <path>
```

```
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok</artifactId>
                <version>${lombok.version}</version>
            </path>
            <!-- This is needed when using Lombok 1.18.16 and above -->
            <path>
                <groupId>org.projectlombok</groupId>
                <artifactId>lombok-mapstruct-binding</artifactId>
                <version>0.2.0</version>
            </path>
            <!-- Validate annotation - @Audited -->
            <path>
                <groupId>com.bivgroup.lib</groupId>
                <artifactId>audit-base</artifactId>
                <version>${revision}</version>
            </path>
        </annotationProcessorPaths>
        <forceJavacCompilerUse>true</forceJavacCompilerUse>
    </configuration>
</plugin>
```

Ручное управление контекстом

Когда нужно переопределить автоматически извлечённый контекст - например, в фоновых задачах или при вызовах от имени другого пользователя:

```
AuditContext context = AuditContext.builder()
    .userId("system")
    .userName("Scheduler")
    .userRole("SYSTEM")
    .sourceSystem("report-service")
    .build();

auditService.audit(context)
    .eventType(AuditEventType.SCHEDULED_JOB_EXECUTED)
    .entityType("DailyReportJob")
    .resultStatus(AuditResultStatus.SUCCESS)
    .send();
```

Справочник

AuditService

Точка входа для создания аудит-событий. Инжектируется через CDI.

```
@Inject
AuditService auditService;
```

Метод	Описание
`audit()`	Создаёт билдер с контекстом из текущего HTTP-запроса
`audit(AuditContext context)`	Создаёт билдер с явно переданным контекстом

Методы билдера

Метод	Описание
`eventType(String)`	Тип события (см. AuditEventType)
`entityType(String)`	Тип затронутой сущности
`entityId(String)`	Идентификатор сущности
`resultStatus(String)`	Результат операции (см. AuditResultStatus)
`addDetail(String key, Object value)`	Добавить произвольное поле в JSON-детали
`details(JsonNode)`	Установить детали как готовый JSON-узел
`eventTime(LocalDateTime)`	Переопределить время события (по умолчанию - `now()`)
`eventParentId(UUID)`	UUID родительского события
`eventChildId(UUID)`	UUID текущего события (по умолчанию генерируется)
`userId(String)`	Переопределить userId из контекста
`userName(String)`	Переопределить имя пользователя
`userRole(String)`	Переопределить роль пользователя
`sourceSystem(String)`	Переопределить систему-источник
`ipAddress(String)`	Переопределить IP-адрес

Метод	Описание
`userAgent(String)`	Переопределить User-Agent
`send()`	Асинхронно отправить событие в RabbitMQ
`build()`	Собрать объект `AuditLog` без отправки

AuditEventType

Предопределённые константы типов событий.

Категория	Константы
Аутентификация	`LOGIN`, `LOGIN_ERROR`, `LOGOUT`, `LOGOUT_ERROR`, `REGISTER`, `REGISTER_ERROR`
Авторизация	`GRANT_CONSENT`, `GRANT_CONSENT_ERROR`, `UPDATE_CONSENT`, `UPDATE_CONSENT_ERROR`, `REVOKE_GRANT`, `REVOKE_GRANT_ERROR`
Пароли	`SEND_VERIFY_EMAIL`, `SEND_VERIFY_EMAIL_ERROR`, `SEND_RESET_PASSWORD`, `SEND_RESET_PASSWORD_ERROR`, `RESET_PASSWORD`, `RESET_PASSWORD_ERROR`
Пользователи	`USER_CREATED`, `USER_UPDATED`, `USER_DELETED`
Данные	`DATA_ACCESS`, `DATA_CREATED`, `DATA_UPDATED`, `DATA_DELETED`, `DATA_EXPORT`, `DATA_IMPORT`
Процессы	`PROCESS_STARTED`, `PROCESS_STEP`, `PROCESS_COMPLETED`, `PROCESS_FAILED`, `PROCESS_CANCELLED`
Задачи	`SCHEDULED_JOB_EXECUTED`, `SCHEDULED_JOB_FAILED`
API	`API_CALL`, `API_CALL_FAILED`

> **Note:** Допускается передача произвольных строк, если стандартных констант недостаточно.

AuditResultStatus

Константа	Описание
`SUCCESS`	Операция выполнена успешно
`FAILURE`	Операция завершилась ошибкой
`IN_PROGRESS`	Операция выполняется
`CANCELLED`	Операция отменена
`SKIPPED`	Операция пропущена
`ACCESS_DENIED`	Отказано в доступе
`TIMEOUT`	Превышено время ожидания

AuditContext

Контейнер для данных о пользователе и запросе. Используется для переопределения автоматически извлечённых значений.

```
AuditContext context = AuditContext.builder()
    .userId("string")
    .userName("string")
    .userRole("string")        // роли через запятую
    .ipAddress("string")
    .userAgent("string")
    .sourceSystem("string")
    .requestURI("string")
    .sessionId("string")
    .eventParentId("string")   // UUID родительского события
    .build();
```

`AuditContext.empty()` создаёт пустой контекст без каких-либо данных.

AuditLog

Модель аудит-события для отправки в RabbitMQ.

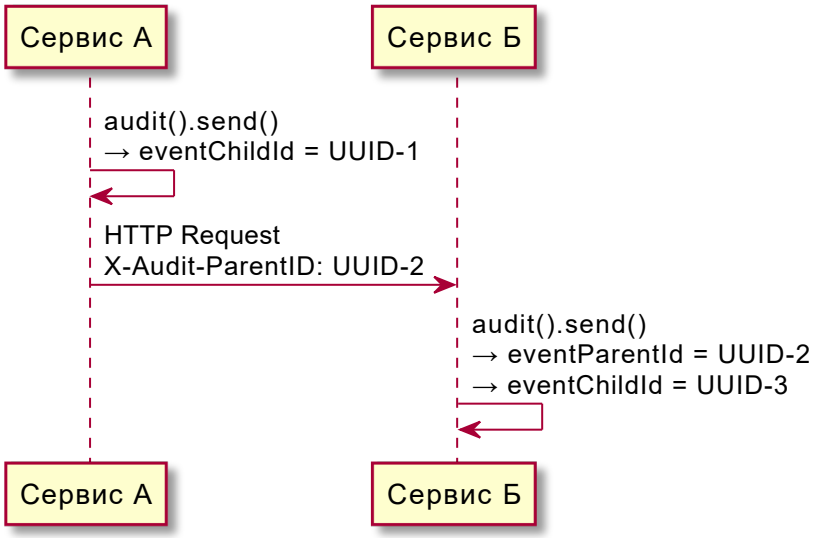
Поле	Тип	Описание
`eventParentId`	`UUID`	UUID родительского события в цепочке

Поле	Тип	Описание
`eventChildId`	`UUID`	UUID текущего события
`eventTime`	`LocalDateTime`	Время события (ISO-8601)
`eventType`	`String`	Тип события
`entityType`	`String`	Тип сущности
`entityId`	`String`	Идентификатор сущности
`userId`	`String`	Идентификатор пользователя
`userName`	`String`	Имя пользователя
`userRole`	`String`	Роли пользователя
`sourceSystem`	`String`	Система-источник события (`product.name` или `quarkus.application.name`)
`ipAddress`	`String`	IP-адрес клиента
`userAgent`	`String`	User-Agent клиента
`resultStatus`	`String`	Результат операции
`details`	`JsonNode`	Дополнительный контекст события

Связывание событий между сервисами

Библиотека поддерживает трассировку цепочки событий при межсервисных вызовах через HTTP-заголовок `X-Audit-ParentID` .

Схема распространения идентификатора родительского события



Все события одной бизнес-операции объединяются в трассируемую цепочку по полю `eventParentId`.

Регистрация фильтра для REST Client

Для автоматической передачи заголовка при исходящих HTTP-вызовах необходимо зарегистрировать `AuditRestClientFilter`:

```
@RegisterRestClient
@RegisterProvider(AuditRestClientFilter.class)
public interface ExternalServiceClient {
    // ...
}
```

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Сервис аудита

Audit Service

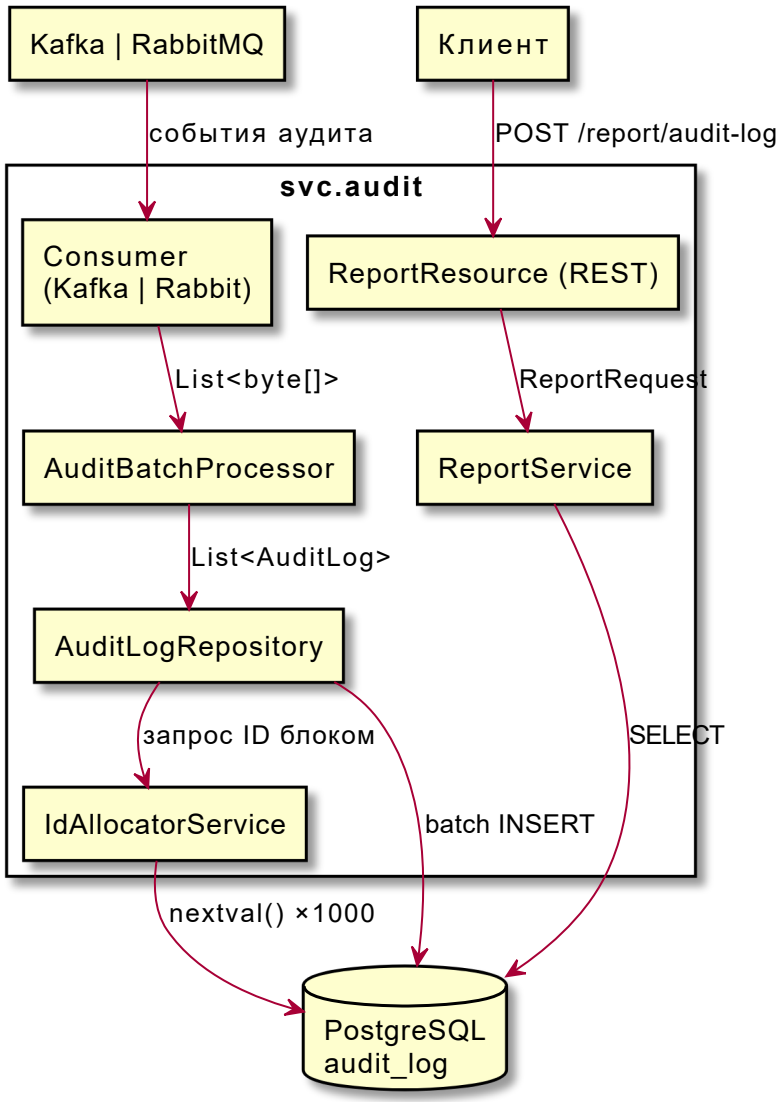
Микросервис обработки и хранения событий аудита фиксирует все значимые действия пользователей и систем: кто, что, когда и с каким результатом выполнил.
Сервис выполняет следующие функции:

- Получение аудит-событий из брокеров сообщений (Apache Kafka или RabbitMQ)
- Пакетная обработка и сохранение событий в PostgreSQL
- Формирование цепочки хэшей для обеспечения целостности журнала (ещё не реализовано)
- REST API для получения отчётов

Содержание

- Архитектура
- Требования
- Конфигурация
- Модель данных
- REST Report API
- Интеграция с брокерами сообщений

Архитектура



1. Брокер сообщений (Kafka или RabbitMQ) доставляет события в виде `byte[]`.
2. Консьюмер аккумулирует события в пакет (batch).
3. `AuditBatchProcessor` десериализует `byte[]` в `AuditLog`, проставляет `create_time`.
4. `IdAllocatorService` занимается аллоакцией Id для минимизации нагрузки на sequence БД.
5. `AuditLogRepository` сохраняет пакет через batch INSERT.
6. Сообщения, которые не удалось обработать, направляются в финальную очередь для ручного разбора.

Требования

- Java 21+
- Quarkus 3.19.4+
- PostgreSQL 16.6+
- RabbitMQ | Kafka
- Maven 3.8+

Конфигурация

Все параметры задаются через `application.properties` с подключением config-maps. Конфигурация для брокеров сообщений описана в config-map - `svc-audit-cfg` .

Ввиду того что в сервисе содержится реализация как для rabbit, так и для kafka то при конфигурировании нужно включить требуемую конфигурацию и выключить все остальные.

```
audit.broker.is-rabbit=true
audit.broker.is-kafka=false
```

По умолчанию при сохранении логов аудита происходит вычисление hash суммы лога с последующим его сохранением в поле - `hash` таблицы - `audit-log` . Поле в дальнейшем участвует в построении цепочки целостности всех записей аудита. Если данная опция не нужна тогда в конфигурации сервиса необходимо задать параметр

```
# hash
audit.compute-hash=false
```

Сервис отчётов позволяет производить экспорт в xlsx файл максимально возможного размера (1 048 576 строк), чтобы не держать их в памяти формирование таблиц производится с помощью временных файлов хранящихся на диске во временной директории операционной системы

```
audit.export.dir=${java.io.tmpdir}/audit-exports
```

Временный файл должен удаляться после завершения операции экспорта, но если в процессе происходит аварийная ситуация то файл будет удалён шедулером, который запускается с интервалом

```
audit.export.cleanup.job-start-interval=6h
```

и производит удаление файлов с маской - `audit-export-*.xlsx` которые старше

```
audit.export.cleanup.file-max-age-minutes=60
```

Модель данных

Таблица `audit_log`

Колонка	Тип	Описание
`id`	`BIGINT` (PK)	Уникальный идентификатор записи
`event_parent_id`	`UUID`	UUID родительского события в цепочке
`event_child_id`	`UUID`	UUID текущего события
`event_time`	`TIMESTAMP`	Время события источника (ISO-8601)
`event_type`	`VARCHAR`	Тип события: `LOGIN`, `LOGOUT`, `DATA_ACCESS` и др.
`entity_type`	`VARCHAR`	Тип сущности
`entity_id`	`VARCHAR`	Идентификатор сущности
`user_id`	`VARCHAR`	Идентификатор пользователя
`user_name`	`VARCHAR`	Имя пользователя
`user_role`	`VARCHAR`	Роли пользователя
`source_system`	`VARCHAR`	Система-источник события
`ip_address`	`INET`	IP-адрес клиента
`user_agent`	`VARCHAR`	User-Agent клиента
`result_status`	`VARCHAR`	Результат операции (`SUCCESS` / `FAILURE` и др.)
`details`	`JSONB`	Дополнительный контекст события
`hash`	`BYTEA`	Хэш записи (цепочка целостности)
`create_time`	`TIMESTAMP`	Время записи в БД (проставляется сервисом)

Sequence

`audit.audit_log_seq` - сиквенс для генерации Id.
`IdAllocatorService` запрашивает блоки по 1000 значений за одну операцию, что

минимизирует нагрузку на БД при высоком потоке событий.

REST Report API

POST /audit/v3/get-log

Возвращает список аудит-записей с поддержкой фильтрации, сортировки и пагинации.

GET /audit/v3/get-hb/

Возвращает справочник

POST /audit/v3/export-log

Возвращает отчёт в виде xlsx файла

Интеграция с брокерами сообщений

Kafka

Топик	Назначение
`audit-log-topic`	Основной топик входящих событий
`audit-final-topic`	Топик для необработанных сообщений (ручной разбор)

Топики создаются автоматически при старте сервиса (`KafkaMQTopologyConfiguration`).

RabbitMQ

Сущность	Назначение
`audit-exchange` (topic)	Основной exchange
`audit-log-queue`	Основная очередь событий
`audit-retry-queue`	Очередь повторных попыток (TTL 360 с, DLX -> основная)
`audit-final-queue`	Очередь необработанных сообщений для ручного разбора

Exchange, очереди и маппинг создаются автоматически при старте сервиса (`RabbitMQTopologyConfiguration`).

Батчинг и пропускная способность

- **Kafka:** группировка сообщений на уровне брокера (`KafkaRecordBatch`).
- **RabbitMQ:** ручной батчинг - сброс при достижении `max-outstanding-messages` (100) или по расписанию каждую 1 секунду.

- Пул потоков обработки: `audit-pool` , максимум 10 одновременных обработчиков.

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js

Сервис построитель цепочки целостности логов аудита

Audit Chain Builder

Воркер построения хэш-цепочки для верификации целостности журнала аудита.
Сервис постфактум обрабатывает записи `audit_log`, разбивая их на временные блоки и вычисляет цепочку хэшей.

Сервис выполняет следующие функции:

- Аллокация временных блоков - нарезка журнала на равные интервалы (окна) и создание заданий на обработку
- Параллельное вычисление `block_hash` для каждого блока (SHA-256 хэшей всех записей окна)
- Последовательная финализация цепочки - вычисление `chain_hash = SHA256(prev_chain_hash || block_hash)`

Содержание

- Архитектура
- Требования
- Установка и настройка
- Конфигурация
- Модель данных
- Алгоритм построения цепочки
- Мониторинг
- Устранение неполадок

Архитектура

Шаги обработки

Этап	Компонент	Тип	Описание
1	`BlockAllocatorJob` / `BlockAllocatorService`	Singleton	Создаёт задания `audit_block_job` (статус `NEW`) для каждого временного окна с лагом

Этап	Компонент	Тип	Описание
2	`BlockHashJob` / `BlockHashService`	Параллельный	Захватывает `NEW` задания через `SKIP LOCKED`, читает хэши записей, вычисляет `block_hash`, переводит в `READY`
3	`ChainFinalizerJob` / `ChainFinalizerService`	Singleton	Финализирует все `READY` блоки строго по порядку, вычисляя `chain_hash`

Ключевые решения

Аллокатор: использует `last_allocated_to_time` в `audit_chain_state` вместо `MAX(to_time)` по всей таблице.

Хэш-воркер: `SELECT ... FOR UPDATE SKIP LOCKED` позволяет нескольким экземплярам сервиса забирать задания конкурентно без координации. Хэши записей читаются через cursor (`setFetchSize(5000)`) для потоковой обработки больших блоков без загрузки всего результата в память.

Финализатор: строгий порядок гарантируется ожиданием - если блок N+1 ещё в `PROCESSING`, финализатор останавливается и ждёт следующего запуска.

Распределённые блокировки: `BlockAllocatorJob` и `ChainFinalizerJob` используют ShedLock (таблица `audit.shedlock`) с `lockAtMostFor=PT5M` . При падении пода блокировка снимается автоматически через 5 минут.

Требования

- Java 21+
- Quarkus 3.33.1+
- PostgreSQL 16.6+
- Maven 3.8+
- Развёрнутый `svc.audit`

Установка и настройка

1. Схема базы данных

Changesets (`src/main/resources/db/changeLog.xml`):

ID	Описание
`audit-table-audit_block_job-1`	Таблица заданий на построение блоков
`audit-index-audit_block_job_status_block_id-2`	Индекс `(status, block_id)` для эффективного SKIP LOCKED
`audit-table-audit_chain_state-3`	Таблица состояния цепочки (одна строка)
`audit-insert-table-audit_chain_state-4`	Начальная строка состояния
`audit-table-shedlock-5`	Таблица блокировок ShedLock

2. Инициализация состояния цепочки

При первом запуске миграция автоматически вставляет начальную строку:

```
INSERT INTO audit.audit_chain_state (id, last_finalized_block, last_chain_hash
VALUES (1, 0, NULL, 0)
ON CONFLICT DO NOTHING;
```

Конфигурация

Все параметры задаются в `application.properties` .

Параметр	Описание	Значение по умолчанию	Обязательный
`audit-chain-builder.block-allocator-job.schedule`	Интервал запуска аллокатора	`60s`	Да
`audit-chain-builder.block.window-seconds`	Размер временного окна одного блока (в секундах)	`60`	Да
`audit-chain-builder.block.lag-seconds`	Временной лаг. Блоки создаются	`120`	Да

Параметр	Описание	Значение по умолчанию	Обязательный
	только для записей старше этого значения (в секундах)		
`audit-chain-builder.block-hash-job.schedule`	Интервал опроса новых заданий воркером хэша	`10s`	Да
`audit-chain-builder.block.processing-timeout-seconds`	Таймаут зависшего `PROCESSING` задания (в секундах). По истечении задание возвращается в `NEW`	`300`	Да
`audit-chain-builder.chain-finalizer-job.schedule`	Интервал запуска финализатора цепочки	`60s`	Да
`audit-chain-builder.block-allocator-job.lock-at-most-for`	Максимальное время удержания ShedLock аллокатора. При падении пода блокировка снимается через этот интервал	`PT5M`	Да
`audit-chain-builder.chain-finalizer-job.lock-at-most-for`	Максимальное время удержания ShedLock финализатора	`PT5M`	Да
`service.hostname`	Идентификатор воркера (проставляется в `worker_id` при захвате задания)	`${HOSTNAME}`	Да

> **Note:** `processing-timeout-seconds` должен быть больше максимально возможного времени обработки одного блока. При тестировании 1 000 000 хэшей обрабатывалось за 1-2 секунды

Модель данных

Таблица `audit.audit_block_job`

Хранит задания на вычисление хэша блока. Одна строка это один временной блок.

Колонка	Тип	Описание
`block_id`	`BIGINT` (PK)	Сквозной идентификатор блока (без пропусков)
`from_time`	`TIMESTAMP`	Начало временного окна (включительно)
`to_time`	`TIMESTAMP`	Конец временного окна (не включительно)
`status`	`VARCHAR(20)`	Статус задания (см. Жизненный цикл задания)
`worker_id`	`VARCHAR(100)`	Hostname воркера, захватившего задание
`row_count`	`BIGINT`	Количество записей `audit_log` в окне
`block_hash`	`BYTEA`	SHA-256 хэш блока
`chain_hash`	`BYTEA`	Хэш цепочки для этого блока
`previous_chain_hash`	`BYTEA`	Хэш цепочки предыдущего блока (для верификации)
`started_at`	`TIMESTAMP`	Время захвата задания воркером
`completed_at`	`TIMESTAMP`	Время завершения вычисления `block_hash`
`created_at`	`TIMESTAMP`	Время создания задания аллокатором

Жизненный цикл задания

```
NEW -> PROCESSING -> READY -> FINALIZED

при зависании на этапен PROCESSING -> таймаут -> NEW
```

Статус	Описание
`NEW`	Задание создано аллокатором, ожидает воркера
`PROCESSING`	Воркер захватил задание, вычисляет `block_hash`
`READY`	`block_hash` вычислен, ожидает финализатора
`FINALIZED`	`chain_hash` вычислен, блок включён в цепочку

Таблица `audit.audit_chain_state`

Единственная строка с вспомогательной информацией.

Колонка	Тип	Описание
`id`	`INT` (PK)	Всегда `1`
`last_finalized_block`	`BIGINT`	`block_id` последнего финализированного блока
`last_chain_hash`	`BYTEA`	`chain_hash` последнего финализированного блока
`last_allocated_to_time`	`TIMESTAMP`	Граница последнего аллоцированного временного окна
`last_allocated_block`	`BIGINT`	`block_id` последнего созданного задания

Таблица `audit.shedlock`

Стандартная таблица ShedLock для распределённых блокировок.

Колонка	Тип	Описание
`name`	`VARCHAR(64)` (PK)	Имя блокировки (`block-allocator-job` , `chain-finalizer-job`)
`lock_until`	`TIMESTAMP`	Время, до которого удерживается блокировка
`locked_at`	`TIMESTAMP`	Время захвата блокировки
`locked_by`	`VARCHAR(255)`	Идентификатор захватившего узла

Алгоритм построения цепочки

Хэш записи (вычисляется в `svc.audit`)

```
record_hash = SHA256(
  id | event_parent_id | event_child_id | event_time | event_type |
  entity_type | entity_id | user_id | user_name | user_role |
  source_system | ip_address | user_agent | result_status |
  details | create_time
)
```

Поля `null` заменяются пустой строкой.
Результат - hex-строка, хранится в `audit_log.hash` как `BYTEA` .

Хэш блока (вычисляется в `BlockHashService`)

```
block_hash = SHA256(record_hash_1 | record_hash_2 | ... | record_hash_n)
```

Записи сортируются по `(create_time, id)` . Хэши читаются побайтово через `rs.getBytes("hash")` .

Хэш цепочки (вычисляется в `ChainFinalizerService`)

```
chain_hash[N] = SHA256(chain_hash[N-1] | block_hash[N])
```

Для первого блока: `chain_hash[1] = SHA256(block_hash[1])` .

Финализация строго последовательна: блок N+1 не финализируется раньше N.
При наличии нескольких готовых блоков все они обрабатываются в одной транзакции.

Верификация целостности

Для проверки цепочки от блока `A` до блока `B`:

```
SELECT block_id, block_hash, chain_hash, previous_chain_hash
FROM audit.audit_block_job
WHERE block_id BETWEEN :A AND :B
  AND status = 'FINALIZED'
ORDER BY block_id;
```

Для каждого блока: `SHA256(previous_chain_hash | block_hash)` должно совпадать с `chain_hash`.

Мониторинг

Ключевые события в логах

Уровень	Сообщение	Значение
`INFO`	`Аллоцировано N блок-джобов`	Аллокатор создал новые задания
`INFO`	`block_id=N is READY, rowCount=K`	Воркер успешно вычислил хэш блока
`INFO`	`Финализировано N блоков, последний block_id=K`	Финализатор обработал цепочку
`WARN`	`Возвращено в статус NEW N зависших джобов`	Обнаружены зависшие задания (признак предыдущего падения пода)
`DEBUG`	`Аллокатор уже запущен на другом инстансе`	ShedLock заблокировал повторный запуск (норма при нескольких подах)
`ERROR`	`Ошибка аллокатора блоков`	Исключение в аллокаторе, требует внимания
`ERROR`	`Ошибка финализатора цепочки`	Исключение в финализаторе, требует внимания

Диагностические запросы

```
-- Текущее состояние построения цепочки целостности
SELECT * FROM audit.audit_chain_state WHERE id = 1;
```

```
4
5  -- Распределение заданий по статусам
6  SELECT status, COUNT(*) FROM audit.audit_block_job GROUP BY status;
7
8  -- Отставание финализатора от аллокатора
9  SELECT
10   last_allocated_block - last_finalized_block AS pending_blocks
   FROM audit.audit_chain_state WHERE id = 1;
```

Устранение неполадок

Задания не создаются (аллокатор не работает)

Симптомы: таблица `audit_block_job` пуста, нет логов `Аллоцировано N блок-джобов`.

Причины и решения:

1. Нет записей в `audit_log` старше `lag-seconds`:

```
1  SELECT MIN(create_time) FROM audit.audit_log;
2  -- Если NULL или время < NOW() - lag_seconds - данных ещё нет
```

2. ShedLock удерживается после падения другого пода:

```
1  SELECT * FROM audit.shedlock WHERE name = 'block-allocator-job';
2  -- Если lock_until в прошлом - блокировка зависла, подождите истечения lock
```

Задания зависают в статусе `PROCESSING`

Симптомы: задания в `PROCESSING` дольше `processing-timeout-seconds`.

Причина: под-воркер упал во время обработки. Задания автоматически возвращаются в `NEW` при следующем тике `BlockHashJob` (по умолчанию через 5 минут).

Ускоренное восстановление вручную:

```
1  UPDATE audit.audit_block_job
2  SET status = 'NEW', worker_id = NULL, started_at = NULL
3  WHERE status = 'PROCESSING'
4    AND started_at < now() - interval '5 minutes';
```

Финализатор не продвигается

Симптомы: `last_finalized_block` не растёт, хотя есть `READY` задания.

Причина: следующий по порядку блок ещё не `READY` (воркер ещё обрабатывает). Финализатор обабатывает задания строго по порядку.

Проверка:

```
SELECT block_id, status, started_at, completed_at
FROM audit.audit_block_job
WHERE block_id = (SELECT last_finalized_block + 1 FROM audit.audit_chain_state
```

Если блок в `PROCESSING` слишком долго - см. раздел «Задания зависают в статусе `PROCESSING`».

2026 ООО "БиАйВи". Все права защищены. | Powered by Wiki.js