Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 20.Б10-мм

Петров Владимир Сергеевич

Обеспечение доступности данных и добавление единого хранилища логов в сервисе CMDB

Отчёт по учебной практике

в форме «Производственное задание»

Научный руководитель:
доц. кафедры СП, к. т. н. Ю.В. Литвинов

Консультант:
руководитель группы, ООО «Яндекс.Технологии», К.С. Сазонов

Санкт-Петербург
2023

Оглавление

1. Введение

В современном мире информационных технологий разработка приложений становится все более важной и сложной задачей, требующей комплексного подхода и использования современных методов и технологий. В то же время, успешный результат разработки и реализации приложения зависит не только от его функциональных возможностей и исполнения поставленных бизнес-задач, но и от обеспечения надлежащих условий для его легкой эксплуатации, поддержки и безопасности. К одним из таких условий относятся компоненты по обеспечению доступа к данным, включающую в себя процессы аутентификации и авторизации пользователя, и системы логирования.

Аутентификация – это процесс верификации личности пользователя, позволяющий приложению получить информацию о том, кто именно взаимодействует с его элементами и данными. На основе различных секретных данных таких как, пароли, сертификаты, ключи, приложение идентифицирует пользователя, который к нему обращается.

Вслед за аутентификацией идет авторизация – процесс, который определяет, какие действия пользователь с успешно прошедшей аутентификацией может выполнять в рамках приложения. Спроектированная система авторизации контролирует доступ к различным разделам, функциям и информации.

Логирование – еще одна ключевая компонента успешного приложения, способствующая его легкой эксплуатации и поддержке. Система логирования предусматривает сбор, хранение и анализ данных о взаимодействии пользователей с приложением, а также информации обо всех возникающих ошибках и событиях. Правильно настроенное логирование помогает обнаруживать и устранять проблемы, оценивать состояние и работу приложения.

Проблемы с перечисленными компонентами были выявлены в CMDB – недавно выпущенным сервисом компании Яндекс для внутреннего пользования, который хранит данные о контрактах, физических и логических сетевых подключениях с операторами связи. Основ-

ными пользователями данного приложения являются другие сервисы, для работы которых нужна информация о сети (об внешних стыках с операторами), и сетевые инженеры.

Изначально CMDB использовал внутреннюю систему для межсервисной аутентификации TVM. Она вместе с реализованной логикой на стороне сервиса обязывала клиента иметь два ключа (один хранящий информацию о сервисе-клиенте, второй о конечном пользователе), чтобы получать ресурсы CMDB. После внедрения сервиса в эксплуатацию две команды выявили следующие недостатки такого подхода.

- Не всегда есть возможность иметь ключ пользователя, так как для его получения нужно иметь идентификатор сеанса[1] или SSH-ключ[2]. Но есть процессы, в которых обращения к CMDB не инициируется пользователем, например, роботизированные системы или демон-приложения, поэтому неоткуда такую информацию брать.

- Данные ключи имеют маленький срок жизни, что затрудняет или делает невозможным реализацию некоторых сценариев, так как данные ключи нужно постоянно получать.

- Такой подход хорошо работает, если пользователи использует CMDB через фронтенд. Но есть потребность использования только API[3] приложения, в частности через CURL[4]. Данный подход усложняет такие сценарии, так как надо вручную через стороннее приложение выписывать эти ключи и потом вручную прописывать их в заголовки формируемого HTTP запроса.

После анализа данных проблем команды составили следующий список требований к доработке аутентификации и авторизации.

[1]Идентификатор сеанса — это фрагмент данных, который используется в сетевых коммуникациях для идентификации сеанса, серии связанных обменов сообщениями.

[2]SSH-ключ — это учетные данные для доступа по сетевому протоколу Secure Shell (SSH)

[3]API (application programming interface) — описание способов взаимодействия одной компьютерной программы с другими.

[4]CURL — служебная программа командной строки, позволяющая взаимодействовать с множеством различных серверов по множеству различных протоколов

- Реализовать аутентификацию и авторизацию через OAuth (решает все перечисленные проблемы).

- Доработать текущий способ авторизации, в котором read-only операции можно будет производить только с ключом сервиса (решает первую проблему).

Также с логированием была выявлена следующая проблема.

Логи хранятся на серверах, где работает сервис. Сейчас у приложения два сервера, и, чтобы найти нужный лог, иногда приходится заходить на каждый сервер. Если серверов будет становиться больше, эта проблема станет более очевидной. Поэтому нужно иметь единое централизованное хранилище для логов.

Резюмируя всё вышесказанное, в данной работе рассматривается доработка систем логирования, аутентификации и авторизации в проекте CMDB.

2. Постановка задачи

Целью работы является доработка систем логирования, аутентификации и авторизации в проекте CMDB.
Для её достижения были поставлены следующие задачи:

- Для аутентификации и авторизации:
    1. Добавить аутентификацию и авторизацию через OAuth.
    2. Доработать авторизацию через TVM на read-only операции.
    3. Выпустить релиз с пользовательской документацией.
- Для логирования:
    1. Выбрать систему для централизованного хранения логов.
    2. Настроить поставку логов с серверов в централизованную систему хранения.

3. Обзор

3.1. Технические детали проекта CMDB

Этот раздел является обзором технической части CMDB для лучшего понимания принятых решений и проделанной работы.

CMDB – это REST API[5] приложение, которое хранит некоторые данные о сети Яндекса. Оно написано на языке golang.

Развернуто на двух серверах в linux контейнерах[6]. Управлением конфигурацией данных контейнеров занимается система salt[7], конфиги которой находятся в едином репозитории Яндекса.

Логи являются структурированными (в виде json), и пишутся в стандартный вывод. Это означает, что они находятся в журнале приложения.

3.2. OAuth

OAuth — протокол авторизации, позволяющий выдать одному сервису (приложению) права на доступ к ресурсам пользователя на другом сервисе.

Если обратится к документу RFC 6749 [2], который описывает принципы работы OAuth, то можно увидеть иллюстрацию его работы, представленной на рисунке 1.

Для того чтобы понять эту схему, разберем представленную в RFC терминологию:

- **Resource Owner** – конечный пользователь, который предоставляет доступ к своим защищенным ресурсам.
- **Resource Server** – сервер, на котором размещены защищенные ресурсы пользователя.

[5]REST API – архитектурный стиль разработки API веб-приложений или компонентов распределённого приложения, используя протокол HTTP.

[6]Linux container – подсистема контейнеризации, позволяющая запускать нескольких изолированных экземпляров операционной системы Linux на одном узле.

[7]Salt – система управления конфигурациями и удалённого выполнения операций.

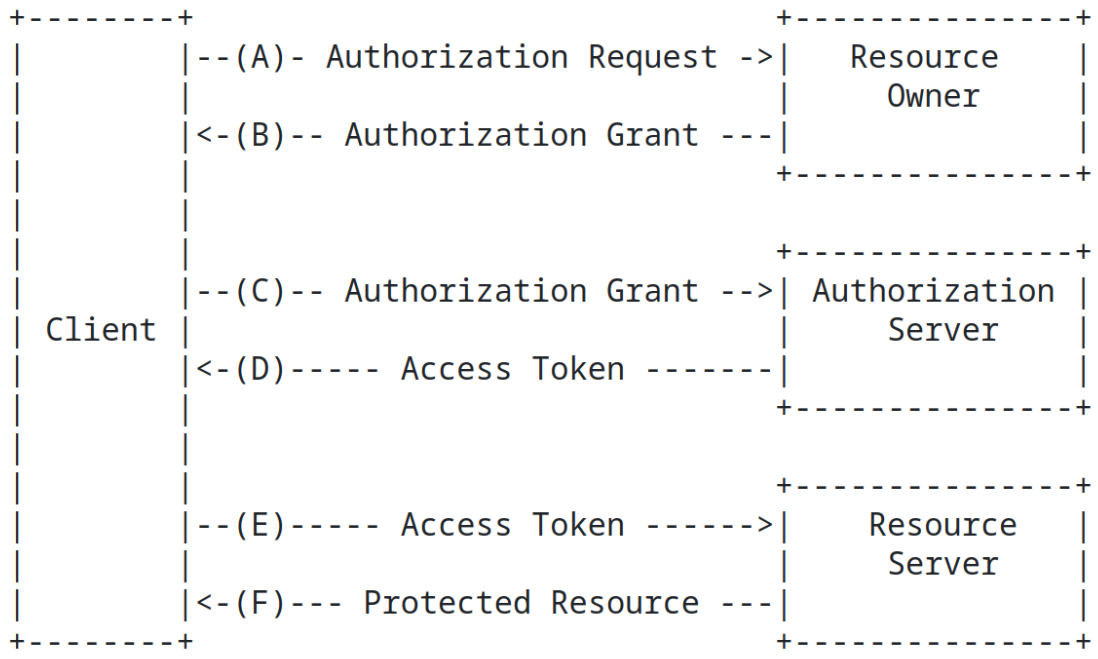

Рис. 1: Иллюстрация работы протокола OAuth.

- **Client** – приложение, которое хочет получить доступ к защищенным ресурсам от имени пользователя и с его разрешением.
- **Authorization Server** – сервер, выдающий токены доступа к защищенным ресурсам клиенту после успешной аутентификации владельца ресурса.

Теперь проясним саму схему работы протокола.

Client хочет получить ресурсы пользователя (Resource Owner) со стороннего сервиса (Resource Server). Этот процесс проходит через следующие этапы:

a) Клиент запрашивает у пользователя разрешение (Authorization Request) на получение доступа к ресурсам от его имени.

b) Пользователь дает разрешение клиенту (Authorization Grant).

c) Клиент предоставляет серверу авторизации разрешение пользователя.

d) Сервер авторизации даёт клиенту токен доступа (Access Token), с помощью которого клиент может получать защищенные ресурсы на сервере ресурсов.

e) Клиент запрашивает защищенные ресурсы у сервера ресурсов, прикладывая токен доступа.

f) Сервер ресурсов проверяет валидность токена и то, что этот токен позволяет получать запрашиваемые ресурсы. Если проверки прошли успешно, сервер отдает клиенту защищенные ресурсы.

Стоит отметить, что в OAuth существует несколько потоков работы протокола. Выше описан стандартный поток работы OAuth. В случае CMDB клиент будет обращаться напрямую к серверу авторизации, предоставляя свои секретные данные и получая токен. На рисунке 2 показано, как это будет выглядеть.

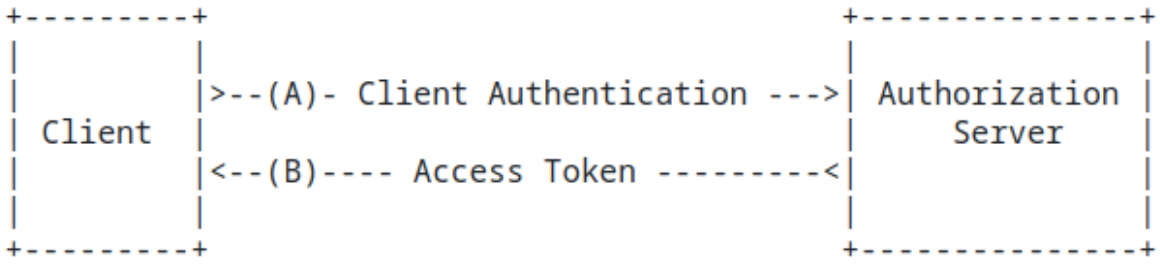

Рис. 2: Поток OAuth, основанный на получении токена с помощью секретов клиента.

3.3. OAuth в Яндексе

Для работы приложения через OAuth в Яндексе существует своя инфраструктура, которая позволяет выстроить различные процессы авторизации, доступные через этот протокол.

В частности для CMDB нужна реализация, которая позволяет пользователю напрямую обращаться к Authorization Server и получать

Access Token. На рисунке 3 изображен данный способ использования OAuth:

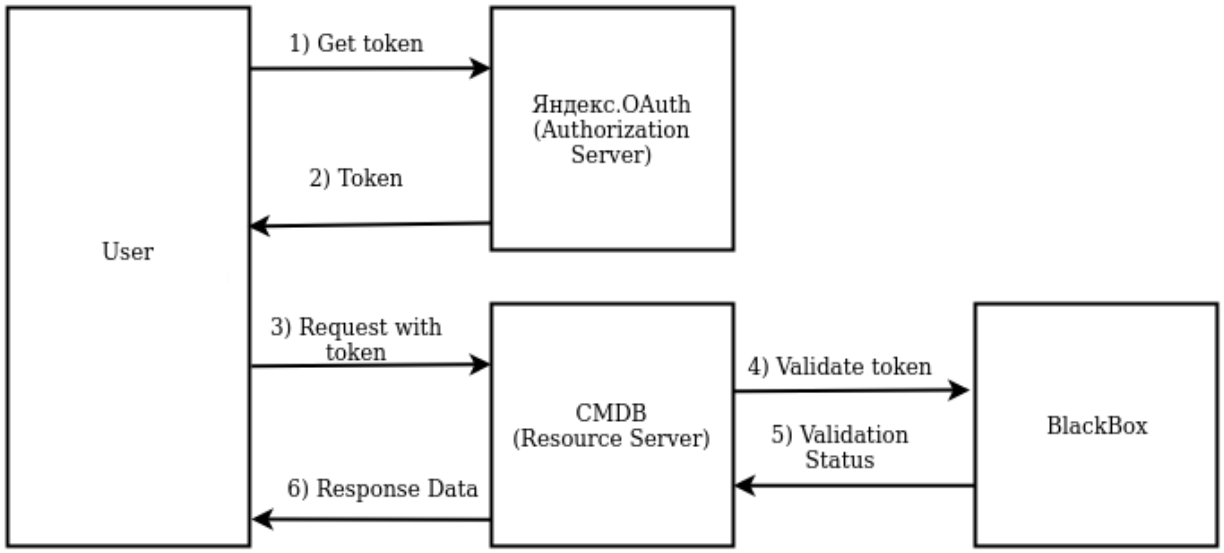

Рис. 3: OAuth в CMDB.

Здесь были введены две новые сущности:

Яндекс.OAuth – сервис для создания и настройки OAuth-приложений внутри Яндекса. Исполняет роль Authorization Server – с помощью него пользователь получает Access token.

BlackBox – это внутренний HTTP-сервис, являющийся частью системы авторизации Яндекса. Основная задача Черного ящика – проверка авторизационных данных пользователей сервисов Яндекса. В данном случае используется как сервис для валидации токена и получения информации о его владельце.

Обращение к CMDB через OAuth выглядит следующим образом:

1. Пользователь переходит по специальной ссылке в Яндекс.OAuth для получения access token.

2. Яндекс.OAuth передает пользователю токен.

3. Пользователь обращается к ресурсам CMDB, прикладывая к запросу полученный токен.

4. CMDB отправляет данный токен в BlackBox.

5. BlackBox отправляет данные о валидности токена и информацию об его владельце (username, uid), с помощью которой сервис проводит аутентификацию и авторизацию.

6. Далее внутренняя логика CMDB проводит аутентификацию и авторизацию. Если всё прошло успешно, CMDB отдает запрашиваемые данные пользователю.

Стоит отметить, что обычно конкретные права, которые предоставляет токен, описываются в scope токена. В данном случае он не используется, так как в CMDB есть внутренняя база ролей. По роли можно точно определить, к каким данным и операциям пользователь имеет доступ. Именно поэтому с BlackBox запрашиваются еще данные об username. По этим данных можно понять, какой пользователь обращается к сервису и какие у него роли. Сейчас в CMDB только две роли:

- Admin – дает доступ к read-write операциям.
- Reader – дает доступ к read-only операциям.

3.4. TVM

TVM [3] – система межсервисной аутентификации и авторизации, использующаяся внутри компании Яндекс, в частности в проекте CMDB. В этой системе фигурирует два ключа:

- Service Ticket – ключ, который идентифицирует сервис. Этот ключ выписывается каждый раз, когда один сервис хочет обратиться к другому сервису. В нем закладывается информация о том, кто к кому обращается.
- User Ticket – ключ пользователя. Выписывается вместе с Service Ticket в обмен на session_id пользователя.

Приложение, к которому обращаются через данную систему, должно обратиться к TVM серверу для валидации данных тикетов и получения информации, связанной с ними. По этой информации приложение должно проверить, что тикет выписан на обращение к нему, проверить, что обращающийся сервис есть в списке разрешенных, получить информацию о пользователе и по внутренней логике провести авторизацию.

3.5. Старая реализация авторизации и аутентификации

На рисунке 4 представлена старая реализация авторизации и аутентификации.

Она состояла из трех промежуточных слоев:

- TVMCheckServiceTicketMiddleware
- TVMCheckUserTicketMiddleware
- TVMCheckPermissionsMiddleware

Первые два занимаются аутентификацией пользователя, третий его авторизацией.

TVMCheckServiceTicketMiddleware достает из запроса ключ сервиса, валидирует его, смотрит, что обращающийся сервис находится в списке разрешенных. Если всё прошло успешно, то вызывается следующий слой TVMCheckUserTicketMiddleware, иначе возвращается ошибка аутентификации пользователю.

TVMCheckUserTicketMiddleware достает из запроса ключ пользователя, валидирует и парсит его, получая тем самым информацию о пользователе, смотрит, что пользователь находится во внутренней базе ролей. Если в этом процессе возникли какие-то ошибки, то возвращается ошибка аутентификации пользователю, иначе у запроса может быть два пути. Если, обращение идет к read-only обработчику, то запрос передается в этот обработчик, иначе запрос отправляется в TVMCheckPermissionsMiddleware.

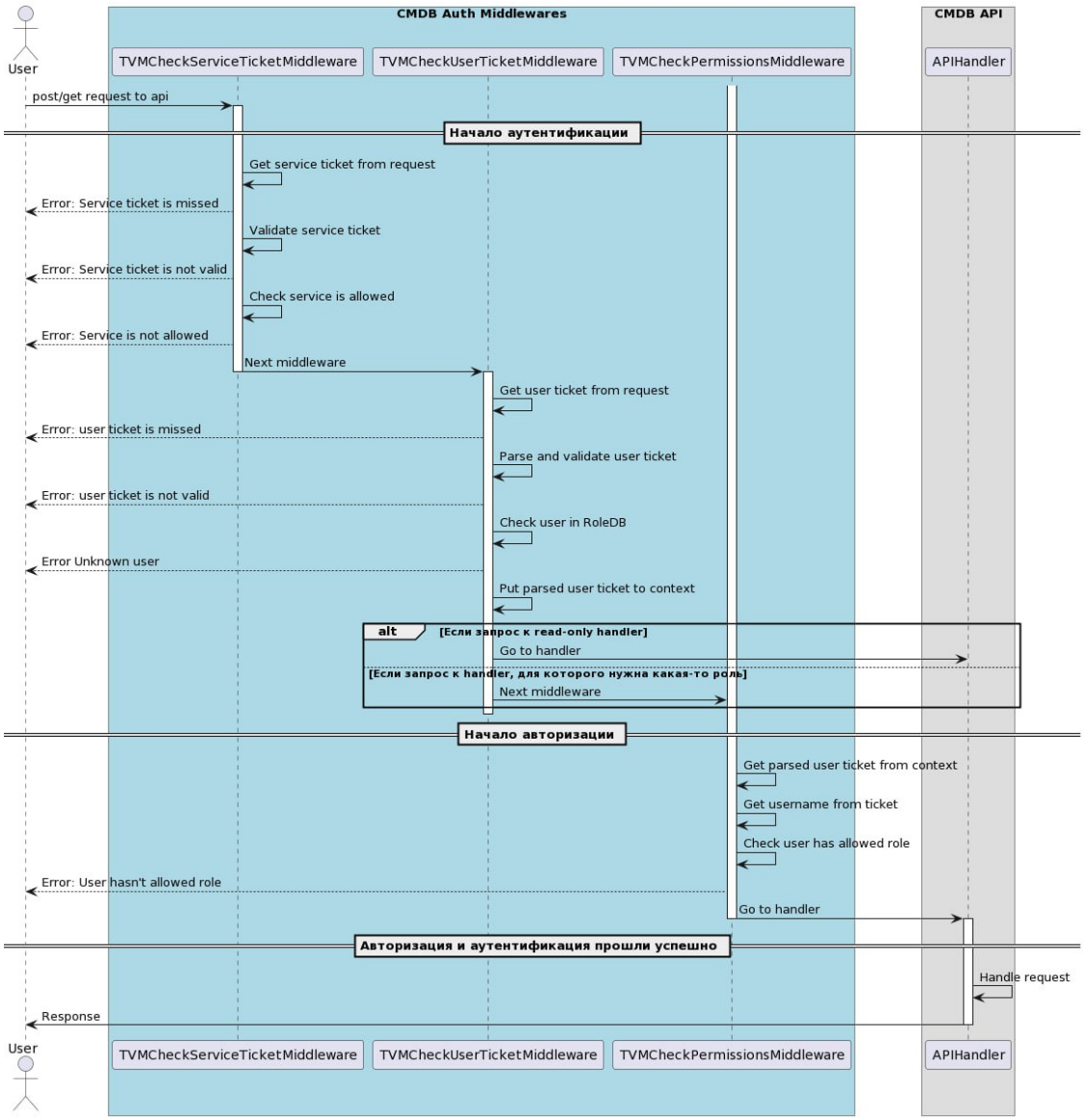

Рис. 4: Авторизация и аутентификация CMDB.

TVMCheckPermissionsMiddleware устроен, так, что он создается перед каждым обработчиком и в нем указывается, каким ролям разрешен доступ к данному обработчику. Данный слой достает из контекста запроса тикет, достает из него имя пользователя, и смотрит в базу ролей, проверяя, есть ли у пользователя роль, разрешенная для доступа в данный обработчик. Если есть, то запрос отправляется в обработчик, иначе отправляется ошибка авторизации пользователю.

3.6. Обзор систем для централизованного хранения логов

Критерии для выбора хранилища логов были следующие:

- Умение хранить структурированные логи.
- Умение фильтровать логи по их содержимому.
- Простое в установке и использовании.

3.6.1. ELK

ELK [9] – явный лидер в сфере хранения, анализа и мониторинга логов. Он основан на стеке программного обеспечения, состоящего из Elasticsearch, Logstash, Kibana. Он действительно хорошо решает свои задачи:

Logstash позволяет принимать логи от разных отправителей и из разных мест, агрегируя их перед записью в базу данных. Elasticsearch – мощный инструмент для хранения и поиска информации, позволяет хранить логи и выполнять к ним различные запросы. Kibana – интерфейс для построения дашбордов различных уровней сложности, работающий поверх Elasticsearch.

Проблемой является установка и настройка всего этого стека. В книге 'Руководство системного администратора Unix' [9] указывается эта проблема и говорится, что даже у опытных администраторов это вызывает трудности.

3.6.2. Graylog

Graylog [1] – это платформа для централизованного сбора, хранения, анализа и мониторинга логов. Он написан на Java и построен поверх другого программного обеспечения с открытым исходным кодом, такого как MongoDB и Elasticsearch. Она позволяет хранить структурированные логи и проводить поиск и фильтрацию по ним. Установка не выглядит слишком сложной, так как основными шагами является

выделение сервера под хранилище логов, настройка nginx, установка и запуск Elasticsearch, MongoDB и Graylog, настройка rsyslog на серверах CMDB для отправки логов в graylog.

3.6.3. ClickHouse

ClickHouse [7] – это колоночная аналитическая СУБД с открытым кодом, позволяющая выполнять аналитические запросы в режиме реального времени на структурированных больших данных. Данное решение дает возможность хранить логи структурировано в виде таблиц и выполнять различные фильтрации и поиски по логам в виде sql запросов.

3.6.4. YT

YTsaurus [6] – еще одна система для хранения и обработки больших данных, разработанная компанией Яндекс. Как и в ClickHouse, в YT можно хранить логи в таблицах и выполнять поиск и фильтрацию с помощью sql запросов.

3.6.5. Итоги

Как итог были выбраны ClickHouse и YT, так как для компании это стандартные инструменты для хранения логов. Этот выбор становится очевидным из-за следующих факторов:

- Так как это является стандартными инструментами, много сотрудников умеет с ними работать.
- Не нужно выстраивать всю инфраструктуру по отправке и хранению логов с нуля (пользуемся готовыми и отлаженными инструментами).
- Множество внутренних сервисов умеют интегрироваться с этими системами, это не приведет к проблемам, когда какие-то определенные агрегированные логи нужно будет выкачать и отобразить в другом сервисе.

Связка этих двух систем выбрана из-за того, что ClickHouse в Яндексе умеет быстро отображать отправленные в него данные (задержка около 30 секунд), но не хранит их длительное время (хранение около двух недель). В YT ситуация обратная, отправленные в него данные появляются медленно, но хранятся долго. Может показаться, что вместо одного инструмента придется тратить много времени, чтобы настроить два. Но инфраструктура так устроена, что, если настроить поставку в YT, настройка поставки в ClickHouse занимает несколько минут.

3.7. Logbroker

Logbroker [5] – это сервис для передачи упорядоченных потоков данных. Logbroker позволяет обмениваться сообщениями через очереди сообщений по модели publish-subscriber [8]. В нем используется понятие ‘топик‘ – это очередь однотипных сообщений, это позволяет помечать и группировать сообщения. Logbroker используется в данной работе как очередь сообщений для поставки логов в YT и ClickHouse.

3.8. Unified Agent

Unified Agent — это агент, специализирующийся на передаче потоков данных. Примерами таких потоков данных могут быть логи, метрики, данные трассировок и любые другие данные, которые можно представить в виде потока сообщений. Это внутренний сервис компании, для внешнего пользования он представлен в Yandex.Cloud [4]. В данный работе он используется, как поставщик логов с серверов, где запущен CMDB, в logbroker.

4. Реализация системы авторизации и аутентификации

На рисунке 5 представлена реализация авторизации и аутентификации.

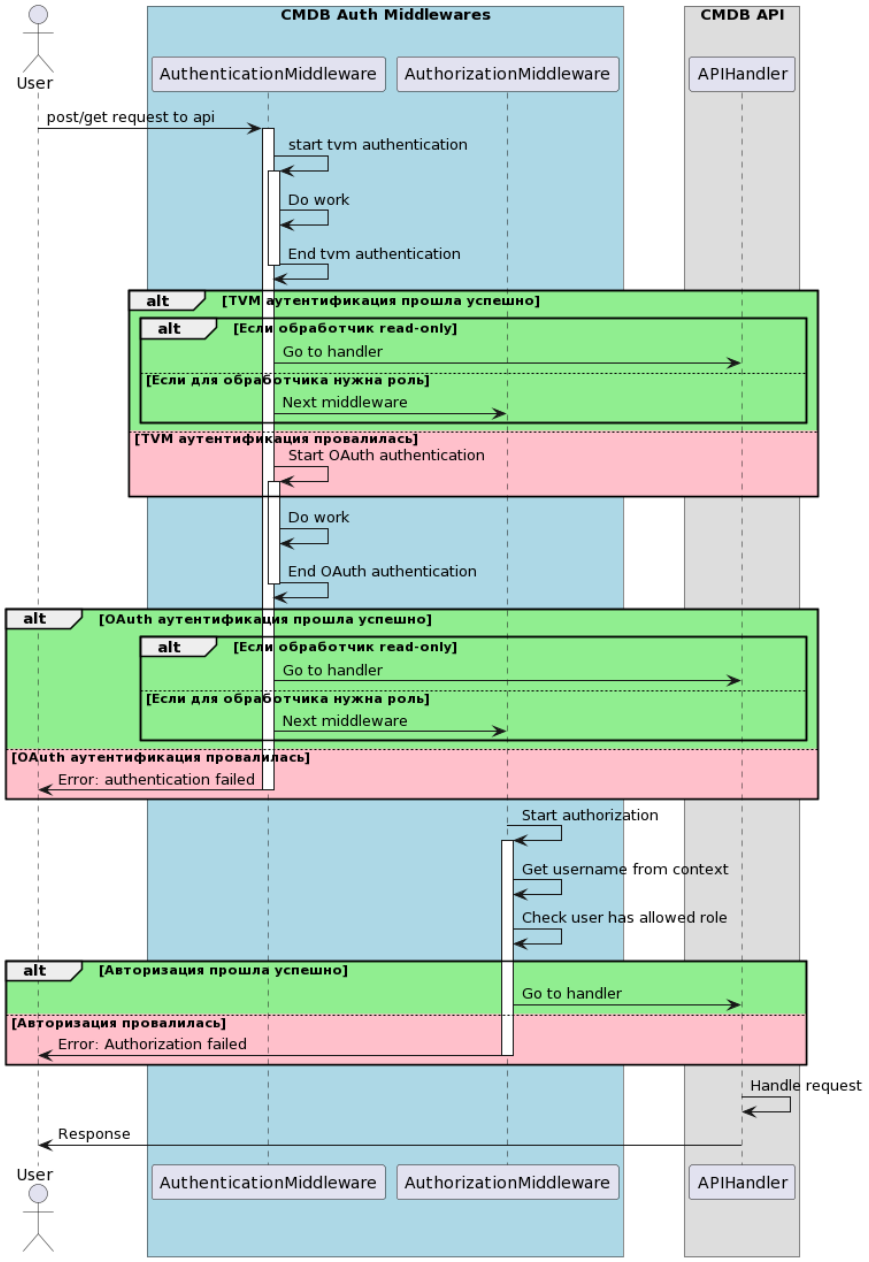

Рис. 5: Авторизация и аутентификация CMDB.

Вместо трех слоёв, предназначенных для TVM, были разработаны два слоя AuthenticationMiddleware и AuthorizationMiddleware, первый

занимается аутентификацией, второй авторизацией.

AuthenticationMiddleware вначале проводит аутентификацию через TVM, если она провалилась, запускает аутентификацию через OAuth. Для этого была разработана структура AuthenticationMethodStorage – хранилище методов аутентификации. С методами Add(authMethod AuthenticationMethod) и BuildAuthenticationMiddleware() func(http.Handler) http.HandleFunc.

Первый метод добавляет в хранилище метод аутентификации, удовлетворяющий интерфейсу AuthenticationMethod. В этом интерфейсе описывается единственный метод StartAuthentication, который проводит аутентификацию. Для этого интерфейса были созданы две реализации: для TVM и OAuth.

Второй метод собирает обработчик промежуточного слоя, который в цикле достает из хранилища методы аутентификации и запускает их. Данный обработчик добавляется в промежуточный слой приложения.

Таким образом видно, что данная реализация позволяет легко расширяться. Если появится новый метод аутентификации, то для него нужно будет написать структуру, удовлетворяющую интерфейсу AuthenticationMethod, и добавить её в AuthenticationMethodStorage.

AuthorizationMiddleware – слой авторизации, он не зависит от метода аутентификации, который мы используем. Так как для проведения авторизации достаточно знать имя пользователя, а дальше из внутренней базы ролей можно узнать, какие роли есть у пользователя и провести авторизацию. Перед каждым обработчиком из API создается этот слой, в нем указывается, каким ролям разрешен доступ к данному обработчику. Каждый AuthenticationMethod в конце аутентификации добавляет имя пользователя в контекст запроса. Далее AuthorizationMiddleware достает это имя и проводит авторизацию.

Ранее было сказано, что были созданы реализации AuthenticationMethod для OAuth и TVM. Рассмотрим их реализацию.

На рисунке 6 представлена реализация AuthenticationMethod для TVM:

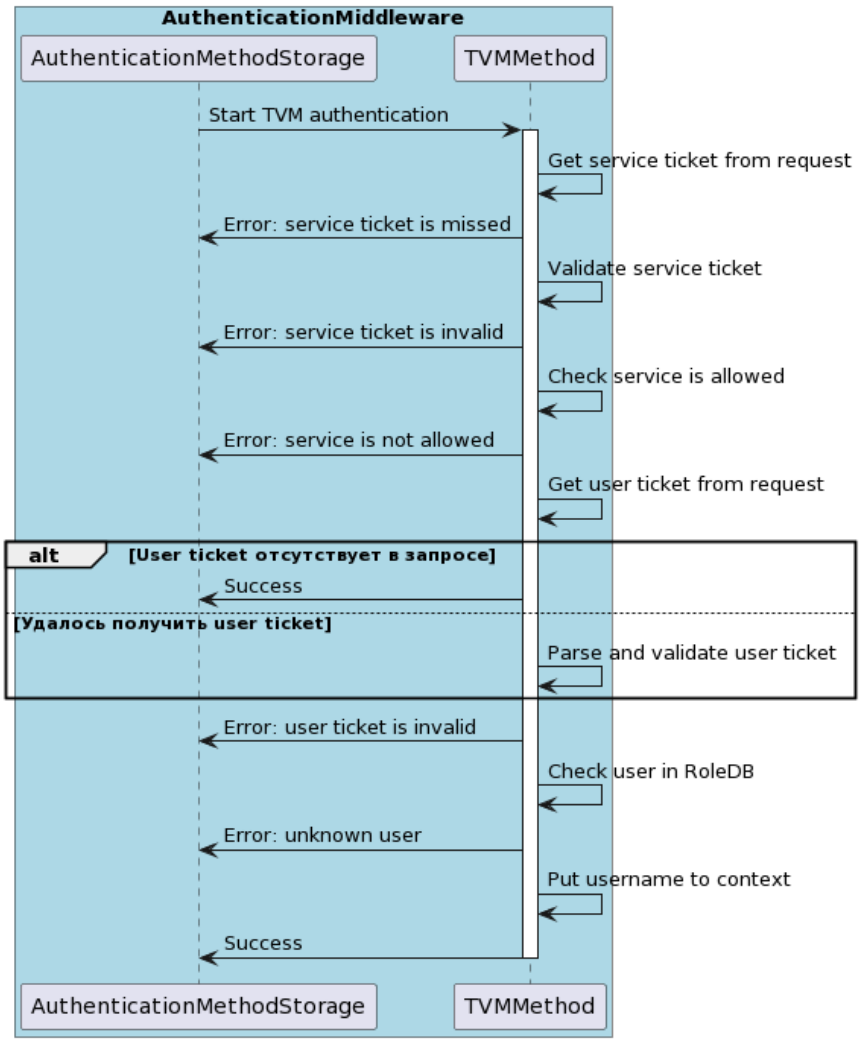

Рис. 6: Реализация TVM аутентификации.

Здесь была помещена логика двух старых слоев TVMCheckServiceTicketMiddleware и TVMCheckUserTicketMiddleware. Отличие состоит в том, что, если метод смог достать service ticket из запроса и провалидировать его, но в запросе не было user ticket, то аутентификация считается пройденной. Сделано это для того, чтобы для read-only операций было достаточно ключа сервиса.

На рисунке 7 представлена реализация AuthenticationMethod для OAuth метода.

Здесь из запроса достается oauth токен, отсылается запрос в blackbox для валидации токена и получение информации о его пользователе, проверяется наличие пользователя в базе ролей и кладется

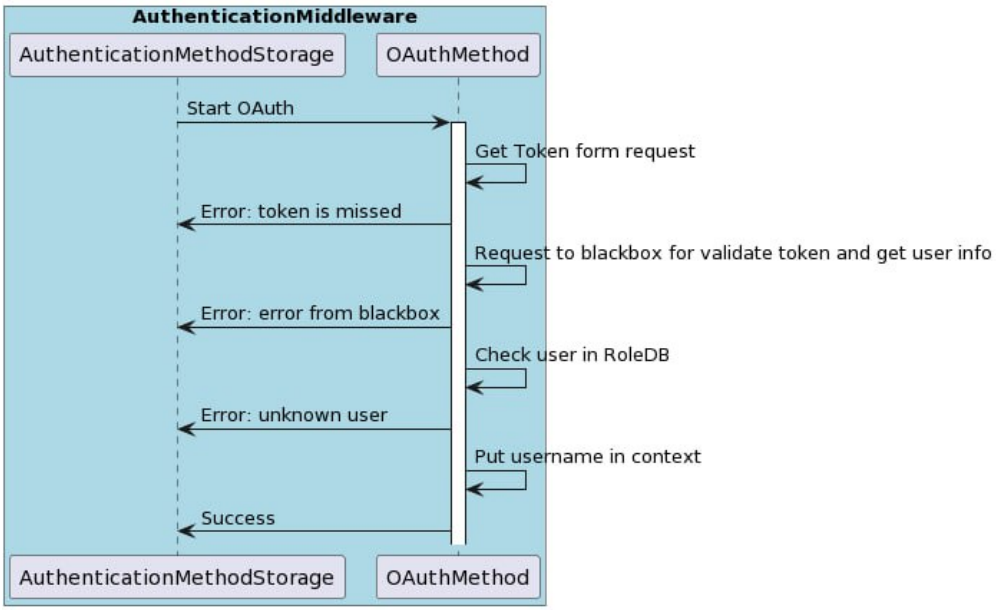

Рис. 7: Реализация OAuth аутентификации.

его имя в контекст для дальнейшей авторизации.

5. Реализация отправки логов в YT и ClickHouse

На рисунке 8 представлена схема отправки логов в YT и ClickHouse.

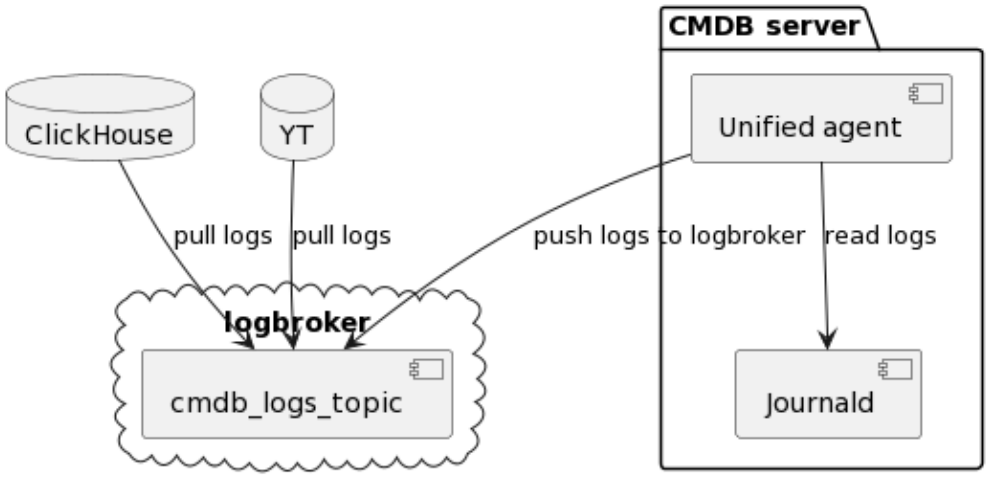

Рис. 8: Схема отправки логов в YT и ClickHouse.

Unified agent забирает с журнала приложения логи и отправляет их в logbroker. Дальше YT и ClickHouse забирают данные логи и раскладывают их по своим таблицам.

В logbroker был создан топик cmdb_logs_topic. В дальнейшем в этот топик записывались и из него читались логи. Были прописаны права, которые позволяли писать CMDB логи и считывать их системам YT и ClickHouse. Всё это было сделано в UI приложении logbroker.

В salt конфигурацию были импортированы готовые конфиги, которые ставили unified agent. Также там была прописана конфигурация, которая делала следующее:

- Создавала пользователя и группу unified-agent.
- Монтировала файл с его конфигурацией.
- В конфигурации прописывалось, что логи нужно читать с журнала CMDB, и отправлять их в logbroker в топик cmdb_logs_topic.

В едином репозитории компании есть место, куда нужно прописывать конфигурации для новых YT и ClickHouse таблиц. Там были прописаны название создаваемой таблицы, из какого топика logbroker читать логи, как их парсить в таблицы (по сути проставление соответствия между полями структурированного лога и колонками таблиц).

6. Результаты работы

Как результат разработки был выпущен релиз с новыми системами авторизации и аутентификации. Была выпущена пользовательская документация про интегрирование с приложением. Данные системы разблокировали две задачи, завязанные на интеграции с CMDB, и привели к их успешному выполнению. Также по логам приложения видно, что приложением активно пользуются, как через TVM, так и через OAuth.

Также в результате работы над централизованным хранилищем логов для CMDB появились таблицы в YT и ClickHouse, хранящие логи приложения. Данное решение сделало просмотр и поиск по логам доступным. Теперь не нужно знать, на каких серверах находятся логи и в каком месте их искать. В документации были оставлены ссылки на таблицы в YT и ClickHouse, что превратило поиск логов в переход по ссылке и отправки sql запросов.

7. Заключение

В рамках учебной практики были выполнены следующие задачи:

- Для аутентификации и авторизации:
    1. Добавлена аутентификация и авторизация через OAuth.
    2. Доработана авторизация через TVM на read-only операции.
    3. Выпущен релиз с пользовательской документацией.
- Для логирования:
    1. Выбраны системы для централизованного хранения логов.
    2. Настроена поставка логов с серверов в централизованную систему хранения.

Код работы находится в закрытом репозитории.

Список литературы

[1] Graylog. — URL: https://graylog.org/ (online; accessed: 2023-08-25).

[2] RFC 6749 описывающий протокол OAuth. — URL: https://datatracker.ietf.org/doc/html/rfc6749 (online; accessed: 2023-08-25).

[3] Доклад от Яндекса про TVM. — URL: https://www.youtube.com/watch?v=0Zvi6_94r5E (online; accessed: 2023-08-25).

[4] Документация для Unified Agent в Yandex.Cloud. — URL: https://cloud.yandex.ru/docs/monitoring/concepts/data-collection/unified-agent/ (online; accessed: 2023-08-25).

[5] Статья Яндекса о Logbroker. — URL: https://habr.com/en/companies/yandex/articles/239823/ (online; accessed: 2023-08-25).

[6] Статья Яндекса об YTSaurus. — URL: https://habr.com/en/companies/yandex/articles/721526/ (online; accessed: 2023-08-25).

[7] Статья о ClickHouse. — URL: https://ru.wikipedia.org/wiki/ClickHouse (online; accessed: 2023-08-25).

[8] Статья о шаблоне publish-subscribe. — URL: https://en.wikipedia.org/wiki/Publish-subscribe_pattern (online; accessed: 2023-08-25).

[9] Эви Немет Гарт Снайдер Трент Хейн Бэн Уэйли Дэн Макин. UNIX И LINUX РУКОВОДСТВО СИСТЕМНОГО АДМИНИСТРАТОРА 5-е издание. — P. 347 – 348.