



Технически университет - София  
Факултет по компютърни системи и технологии Катедра  
“Компютърни системи”  
Специалност “Компютърни системи и технологии”

# ДИПЛОМНА РАБОТА

на тема

Проектиране и разработка на система за одитиране на софтуер

Мирослав Николаев Цветанов

Фак. номер: 121212260

Група: 59

Научен ръководител: д-р инж. Десислав Андреев

Дата: 11.08.2024  
гр. София

# Съдържание

Съдържание .....	2
Введение.....	4
Първа глава .....	6
1.1. Постановка.....	6
1.2. Цел и задачи на разработката .....	7
1.3. Анализ на съществуващи решения.....	8
1.3.1 The Splunk Platform .....	8
1.2.2. Auditd (Linux Auditing System) .....	10
1.2.3. Sentry .....	13
1.2.3. Съпоставка на разгледаните решения .....	18
1.3. Заключение .....	19
Втора глава.....	20
2.1. Основна цел .....	20
2.2. Анализ на налични хранилища за обекти .....	21
2.2.1. 2Amazon S3 .....	21
2.2.2. MinIO.....	22
2.2.3. Google Cloud Storage .....	23
2.2.4. Сравнителен анализ .....	24
2.3 Анализ на налични Message Queue Системи .....	26
2.3.1 Apache Kafka.....	27
2.3.2 RabbitMQ .....	28
2.3.3 Amazon SQS.....	29
2.3.4 Сравнителен анализ .....	30
2.4 Сравнителен анализ на програмни езици .....	31
2.5 Docker контейнери .....	34
2.6 Kubernetes .....	35
2.7 Системни изисквания.....	36
2.7.1 Функционални изисквания.....	36
2.7.2 Нефункционални изисквания.....	37
2.8 Обобщение .....	37
Трета глава .....	39
3.1. Основна архитектура .....	39
3.2. Docker контейнери .....	41
3.2.1. Nginx.....	41
3.2.2. Ingestion API.....	42
3.2.3. Apache Kafka.....	43
3.2.4. Forwarder .....	43
3.2.5. MinIO.....	43

3.2.6. Retrieval API.....	44
3.2.7. Обобщение .....	44
3.3. Предпоставки за бъдещо развитие.....	46
3.4. Заключение .....	47
Четвърта глава .....	49
4.1. Потребителско ръководство за стартиране на системата.....	49
4.1.1 Изисквания и инсталация на необходимите инструменти: .....	49
4.1.2 Конфигурация и стартиране на системата: .....	49
4.1.3 Предварителна конфигурация на OAuth сървъра.....	50
4.1.4 Използване на Swagger документация и OpenAPI Specification .....	50
4.1.5 Достъп и работа със системата .....	52
4.2. Заключение .....	52
Пета глава .....	53
5.1. Тестова постановка.....	53
5.2. Тестване на системата.....	53
5.2.1. Функционално тестване.....	54
5.2.2. Модулно тестване .....	55
5.2.3. Интеграционно тестване.....	57
5.2.4. Тестване на производителността на системата.....	60
5.3. Заключение .....	62
Заключение .....	63
Списък с фигури.....	64
Списък с таблици .....	64
Източници .....	65
Приложение .....	66

# Въведение

Системата за одитиране на софтуер, насочена към одитиране на събития, представлява съществен елемент от съвременните софтуерни решения, особено в контекста на информационната сигурност и съответствието с регуляторните изисквания. Одитирането на събития играе ключова роля в осигуряването на прозрачност и проследимост на действията в системата, като регистрира всяко значимо събитие, което се случва в рамките на софтуера. Това включва информация за това кой потребител е извършил конкретно действие, какво е било направено и в какъв момент е било изпълнено.

Основната цел на тази система е да гарантира целостта и автентичността на записаните данни, като не позволява тяхната неоторизирана промяна или изтриване. За да се постигне това, се прилагат различни методи за защита, като цифрово подписване и хеширане на съобщенията. Тези технологии гарантират, че всяка промяна в данните би била лесно засечена, което от своя страна повишава доверието в системата. Непроменяемостта на записите е от съществено значение, тъй като те често служат като доказателство в случаи на одити, разследвания или спорове.

В допълнение към защитата от неоторизирана промяна, сигурността на данните в системата за одитиране на софтуер се гарантира и чрез криптиране. Всички записани събития се криптират, което осигурява защита срещу неоторизиран достъп и гарантира, че дори в случай на компрометиране на системата, данните ще останат неприкосновени. Криптирането е особено важно в контекста на обработката на чувствителна информация, където опазването на конфиденциалността е приоритет.

Системата за одитиране на софтуер е не само средство за защита на данни, но и инструмент, който улеснява съответствието с различни регуляторни и индустриални стандарти. В много отрасли, като финансови услуги, здравеопазване и публичния сектор, съществуват строги изисквания за отчетност и проследимост на действията. Чрез предоставянето на неизменен и подробен запис на всички важни събития, системата за одитиране на софтуер позволява на организациите да отговорят на тези изисквания и да избегнат сериозни санкции.

Не на последно място, системата за одитиране на софтуер подпомага и управлението на инциденти и анализирането на проблеми, като осигурява детайлна информация за всички ключови събития. Това улеснява откриването на аномалии, неправомерни действия и потенциални заплахи, което е от критично значение за поддържането на сигурността и стабилността на софтуерните системи.

В заключение, системата за одитиране на софтуер е основополагащ компонент за всеки софтуер, където сигурността, прозрачността и съответствието с регулатиците са приоритет. Тя не само осигурява защита на данните чрез неприкосновеност и криптиране, но и предоставя необходимата отчетност, която е от съществено значение за успешното функциониране на съвременните информационни системи.

# Първа глава

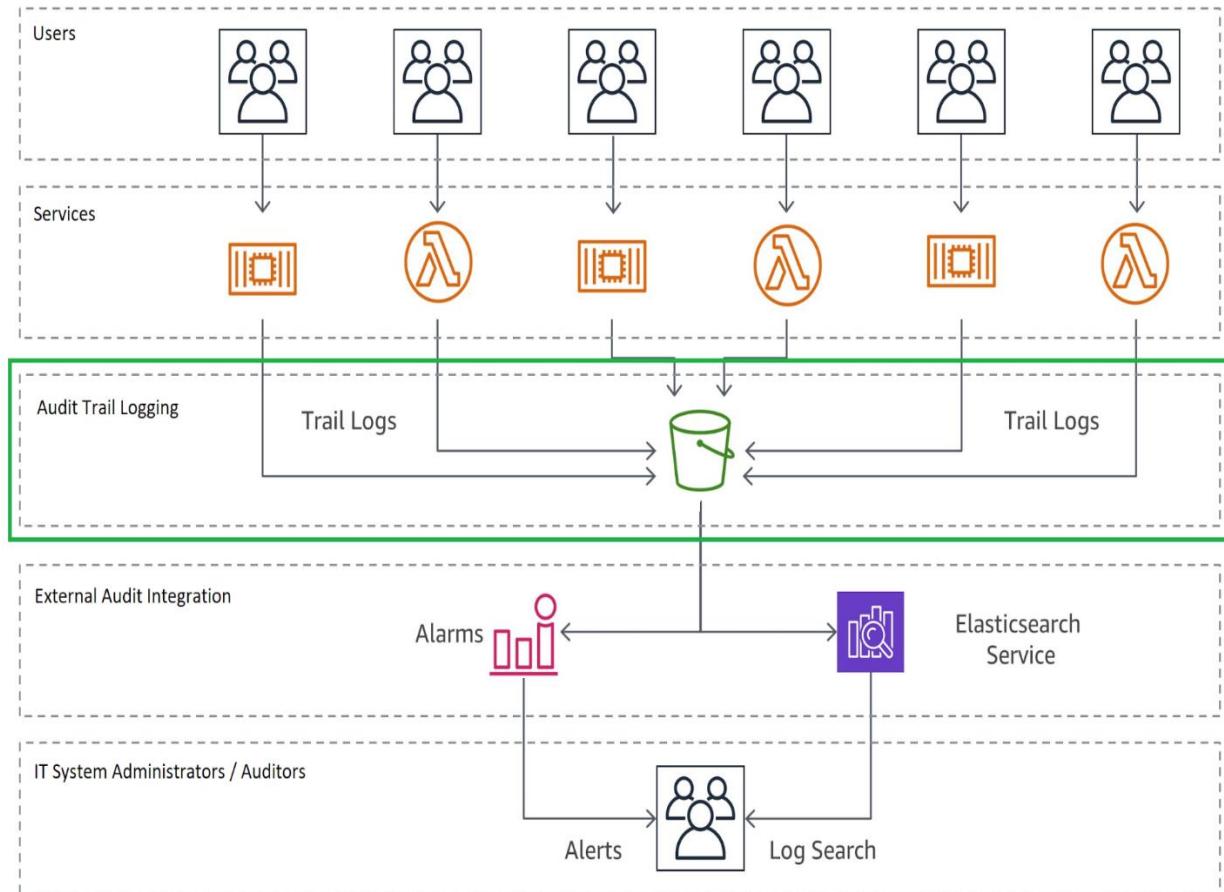
## 1.1. Постановка

С нарастващата нужда от прозрачност, сигурност и в съответствие с регуляторните изисквания, разработването на системи за одитиране на софтуер става все по-важно в съвременните ИТ инфраструктури. Технологичният напредък предоставя нови възможности за създаване на иновативни решения, които не само отговарят на тези нужди, но и предлагат лесна интеграция и използване, без допълнителни разходи. Целта на настоящото решение е да предостави на потребителите ефективна, гъвкава и бесплатна система за одитиране на софтуер, която може да се внедрява в различни софтуерни разработки, осигурявайки същото ниво на качество, както платените решения на пазара.

Разработеното решение ще предлага необходимите функционалности за събиране, криптиране, съхранение, и изчитане на данни от различни източници. Възможността за лесна конфигурация и имплементация ще бъде основен приоритет, за да се гарантира, че системата може да бъде интегрирана бързо и ефективно в различни инфраструктури, без да изисква задълбочени познания за настройка и управление.

Много от съществуващите на пазара решения за системи за одитиране на софтуер са насочени към специфични нужди и изискват значителни усилия за настройка и управление. Освен това, повечето от тях са платени, което ограничава достъпа до тях за малки и средни предприятия или индивидуални разработчици. В отговор на тези предизвикателства, предложената система ще бъде изградена с отворен код и ще използва съвременни технологии, за да предостави решение, което може да бъде адаптирано към разнообразни сценарии на употреба.

Крайната цел на разработката е създаването на модул за одит система, която може да бъде интегрирана в различни системи и платформи, предоставяйки на потребителя възможността да изчита, съхранява и анализира лог данни в подходящ формат. Потребителите ще трябва само да предоставят данните в съвместим формат, за да могат те да бъдат правилно обработени и анализирани от системата (Фиг. 1).



Фиг. 1 – Стандартна система за одитиране на софтуер.

## 1.2. Цел и задачи на разработката

В съвременната дигитална ера данните и тяхната сигурност са от съществено значение за организациите. Одит системите играят ключова роля в осигуряването на прозрачност и сигурност, като позволяват проследяване и анализиране на действията в системите. Съществуващи на пазара решения са разнообразни, но често са свързани с високи разходи и сложни настройки, за да отговорят на специфичните изисквания на потребителя.

Целта на настоящата разработка е да създаде решение за одит система, което да отговори на нуждите на потребителя, използвайки безплатни инструменти с отворен код. Крайната цел е да се предложи решение, което да бъде използвано безвъзмездно и същевременно да предоставя всички необходими функционалности като събиране, съхранение, криптиране и изчитане на данни.

Като обобщение, разработката има три основни задачи:

- 1 Да осигури бесплатно решение за одит система, конкурентно на съществуващите такива на пазара.
- 2 Решението да използва най-иновативните технологии с отворен код.
- 3 Да предостави на потребителя решение, което изисква минимални настройки и лесна интеграция.

### 1.3. Анализ на съществуващи решения

Съществуващите решения предлагат еквивалентни функционалности, като някой от тях са платени са и изискват много настройки от страна на потребителя, за да удовлетворят неговите нужди.

#### 1.3.1 The Splunk Platform

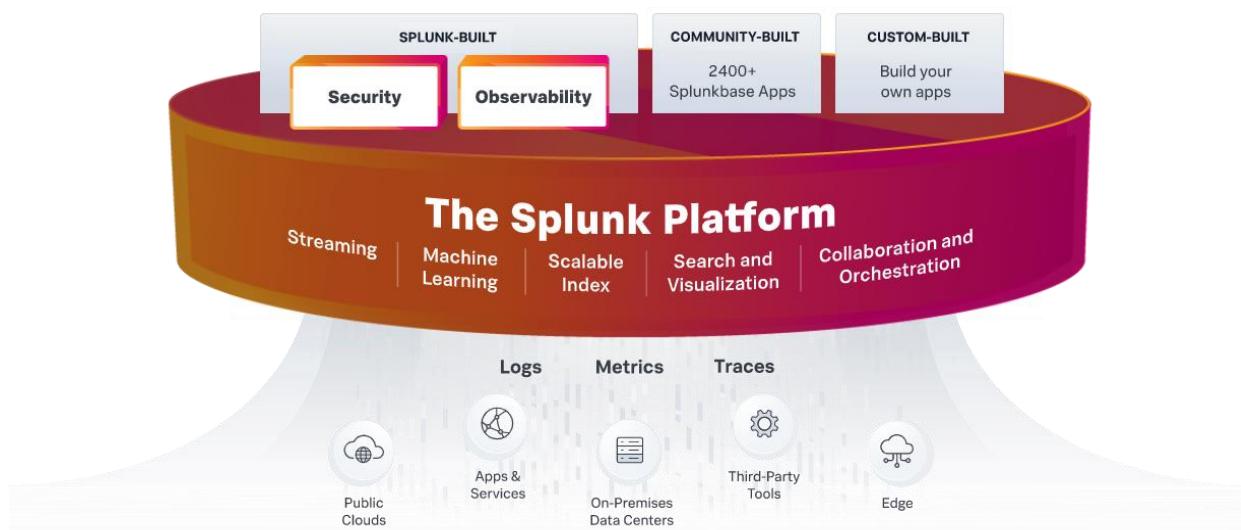
Splunk (Фиг. 2) предоставя гъвкава и мащабируема платформа, която поддържа широк набор от случаи на използване. Могат да бъдат използвани вградените в Splunk решения за търсене и отчитане, сигурност и наблюдение, 2400+ приложения, налични в Splunkbase, и за създаване на персонализирани приложения, настроени към специфични нужди. Splunk също така предоставя водещи на пазара специално създадени решения, които осигуряват пълна точност и унифицирана сигурност. Той е агностичен за източника на данни, покриващ целия технологичен пейзаж - мултиоблачен, хибриден и периферен. Splunk погъща метрики, регистрационни файлове и следи с пълна прецизност, осигурявайки контекстна видимост от край до край.



Фиг. 2 - Лого на Splunk

Splunk е едно от най-използваните решения на пазара поради множеството възможности, които предлага на крайния потребител. Характеристиките, които Splunk предлага са следните:

- Облачно, хибридно или корпоративно внедряване;
- Събира и индексира регистрационни и машинни данни от всеки източник;
- Мощните възможности за търсене, анализ и визуализация дават възможност на потребителите;
- Анализ за откриване на измами и киберзаплахи;
- Анализ в реално време за оперативно разузнаване и бизнес отчитане;
- Осигуряване на информация и анализ на сигурността;
- Наблюдаване и осигуряване на проблеми със съответствието;
- Наблюдаване на машинните данни за логистични RFID и логистични бази данни
- Наблюдаване и управляване на IoT, включително SCADA данни;
- Big Data Analytics, машинни данни от интернет/вътрешна мрежа.



Фиг. 3 - Диаграма на The Splunk Platform

Продуктите, които Splunk платформата предлага са:

- *Splunk Enterprise* е софтуерна платформа, която позволява да търсене, анализиране и визуализиране на данните, събрани от компонентите на съответна инфраструктура. Splunk Enterprise приема данни от приложения, сензори, устройства и т.н. След като бъдат настроени източниците на данни, Splunk Enterprise индексира потока от данни и го анализира в поредица от отделни събития и показатели, които могат да бъдат

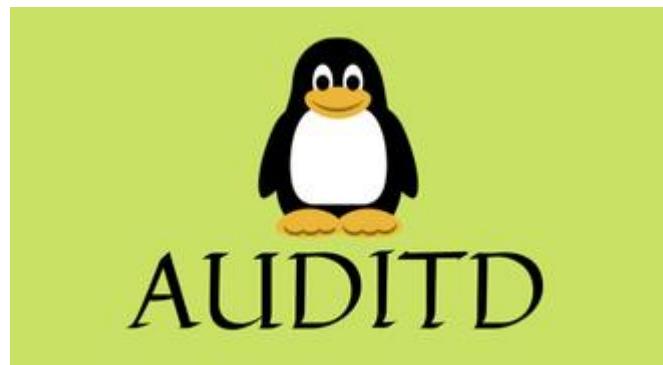
преглеждани и търсени. Потребителите могат да се свързват по два начина към Splunk Enterprise - посредством уеб браузър, като

използват Splunk Web, както и посредством интерфейса на командния ред, който Splunk Enterprise предлага на потребителите.

- *Splunk Cloud* предоставя предимствата на наградения Splunk® Enterprise като облачна услуга. Използвайки Splunk Cloud Platform, потребителят получава функционалността на Splunk Enterprise за събиране, търсене, наблюдение, отчитане и анализиране на всички машинни данни в реално време и исторически данни, използвайки облачна услуга, която се предоставя централно и еднакво от Splunk на голям брой от облачни клиенти, от компании от Fortune 100 до малки и средни предприятия.

### 1.2.2. Auditd (Linux Auditing System)

Auditd (Фиг. 4) е мощна и гъвкава система за одитиране на Linux, която се използва за проследяване на различни събития в операционната система. Тя предоставя детайлни съобщения за действията на потребителите и процесите в системата, като по този начин осигурява висока степен на сигурност и контрол.



Фиг. 4 - Лого на Auditd

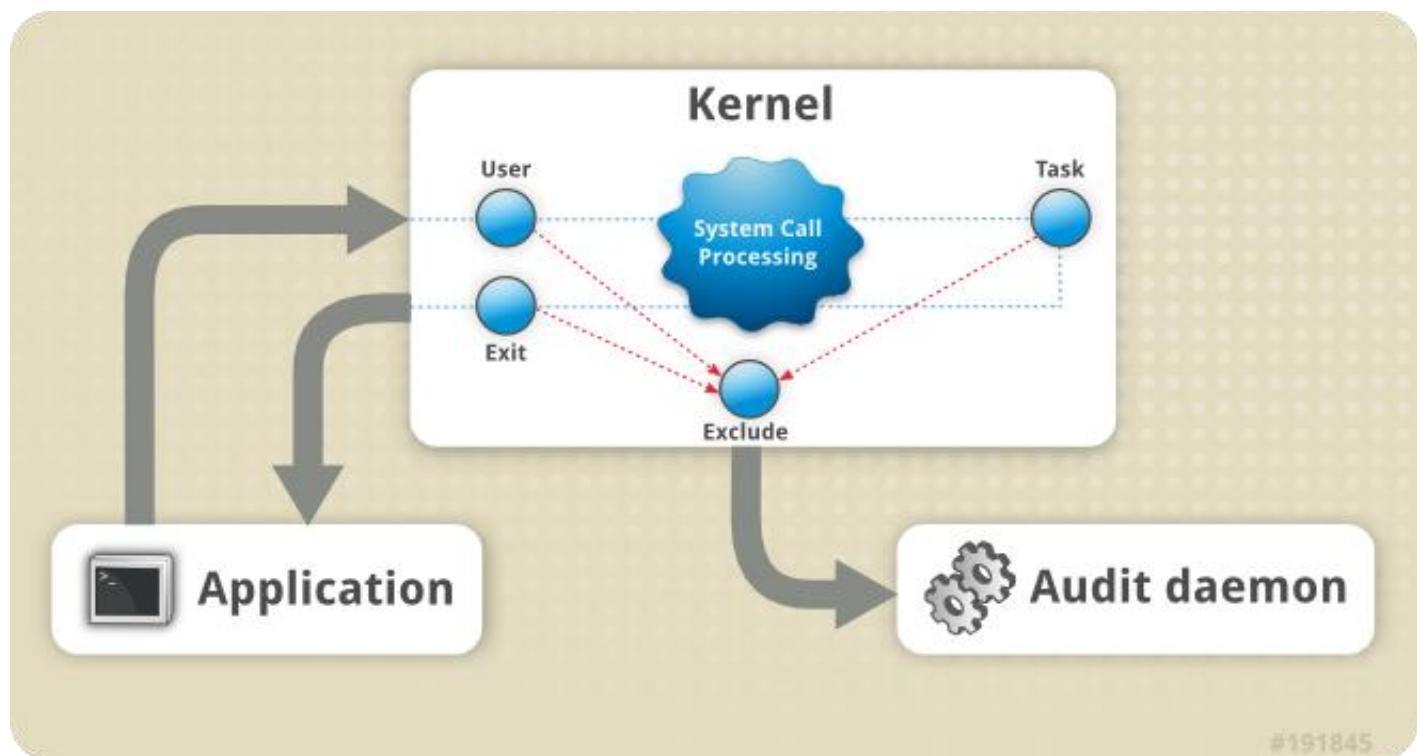
Основни характеристики на Auditd:

**Вграден в Linux ядрото:** Auditd е част от Linux ядрото и е създадена с цел да предоставя детайлна информация за сигурността и събитията в системата. Тя е вградена в операционната система и е достъпна на всички съвременни Linux дистрибуции.

Проследяване на системни събития: Audited позволява проследяване на разнообразни системни събития, включително:

- Достъп до файлове и директории.
- Промени в конфигурационни файлове.
- Стапиране и спиране на процеси.
- Промени в потребителски акаунти и групи.
- Действия, свързани с мрежовите настройки и достъпа до мрежови ресурси.
- Команди, изпълнявани от потребители, и промени в права на достъп.

Гъвкава конфигурация: Audited предоставя възможност за създаване на персонализирани правила, които определят какви събития да бъдат логвани. Тези правила могат да бъдат дефинирани на базата на различни критерии, като потребителски действия, файлове или мрежови събития.



Фиг. 5 – Audited Архитектура

## Основни компоненти на Auditd:

- Audit Daemon (auditd): Това е основният демон, който отговаря за събиране и записване на одит съобщенията в системата. Демонът работи във фонов режим и записва всички събития, които отговарят на зададените правила.
- Audit Rules: Правилата за одитиране са конфигурационни инструкции, които определят какви събития да бъдат наблюдавани и записвани от системата. Те се дефинират чрез команда `auditctl` или чрез конфигурационния файл `/etc/audit/audit.rules`.
- Audit Logs: Съобщенията, генериирани от Auditd, обикновено се записват в директорията `/var/log/audit/` и могат да се разглеждат с команда `ausearch` или `auditctl`. Записите съдържат подробна информация за всяко събитие, включително време, потребител, процес и засегнатите ресурси.
- Audit Reports: Инструментът `aureport` позволява генериране на отчети от одит съобщения, като предоставя обобщена информация за събитията в системата. Това може да включва отчети за достъп до файлове, неуспешни опити за логване и други.

## Приложение и използване на Auditd:

- Сигурност и съответствие: Auditd е критичен инструмент за осигуряване на сигурността на Linux системите. Чрез проследяване на събитията в системата, администраторите могат да идентифицират подозрителни дейности и да гарантират, че системата отговаря на изискванията за съответствие с различни регулатации (например GDPR, HIPAA).
- Анализ на инциденти: В случай на нарушение на сигурността, одит съобщенията предоставят важна информация за анализ на инцидентите и възстановяване на действията, довели до пробива. Те могат да бъдат използвани като доказателство в съдебни разследвания.
- Мониторинг на критични ресурси: Auditd позволява мониторинг на критични файлове и системни ресурси, като генерира аларми при промени или опити за

достъп до тези ресурси. Това може да помогне за предотвратяване на атаки и неупълномощени действия.

Заключение:

Auditd е изключително мощен инструмент за сигурност, който предоставя подробен контрол и мониторинг върху действията в Linux системите. Той е особено полезен в среди, където сигурността и съответствието с регулатиците са критични. Въпреки че изиска известно време за конфигуриране и поддръжка, ползите от използването му са значителни, особено за организации, които се стремят към висока степен на сигурност и прозрачност в управлението на своите системи.

### 1.2.3. Sentry

Sentry е модерна платформа за мониторинг и записване на грешки и изключения, която помага на разработчиците да откриват, диагностицират и поправят проблеми в софтуерните приложения. Основният фокус на Sentry е върху проследяването на грешки в приложението, но тя предлага и възможности за записване на одит съобщения и мониторинг на потребителската активност, които са полезни в контекста на сигурността и съответствието.



*Фиг. 6 – Лого на Sentry*

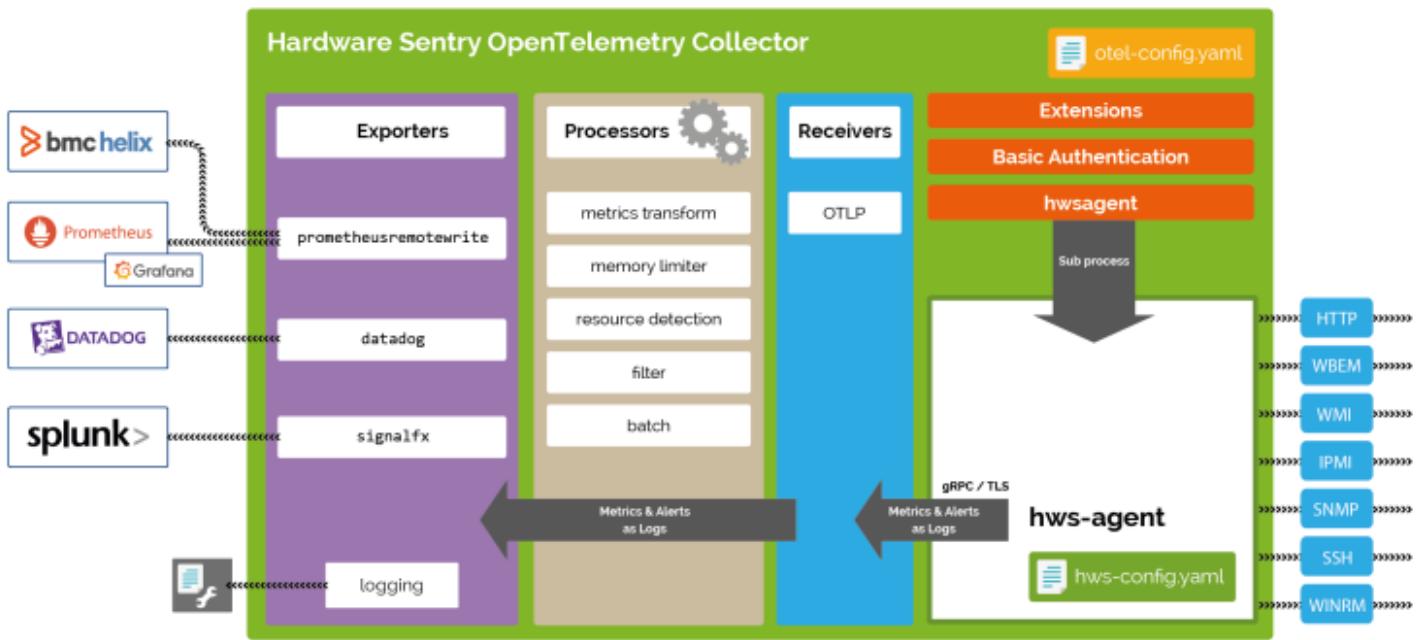
Основни функционалности на Sentry:

- Проследяване на грешки и изключения: Sentry автоматично открива и записва грешки, възникнали в приложението, като предоставя детайлна информация за контекста на грешката, включително стек трейсове, състояние на системата и действията на потребителите непосредствено преди възникването на проблема.
- Информация относно контекста: Освен стек трейсове, Sentry събира информация за средата, в която е възникнала грешката, като операционна система, версия на приложението, конфигурации и други. Това помага на разработчиците да разберат по-добре условията, при които е възникнал проблемът.
- Мониторинг на производителността: Sentry предлага инструменти за мониторинг на производителността, които позволяват на екипите да следят и анализират производителността на своите приложения, идентифицирайки бавни заявки и проблеми със скоростта на отговор.
- Потребителска активност: Платформата проследява активността на потребителите, като позволява на разработчиците да видят какви действия са довели до дадена грешка. Това е особено полезно за разследване на инциденти и анализ на поведението на потребителите.
- Известяване и аларми: Sentry предлага гъвкави опции за известяване, които могат да бъдат изпращани по различни канали, включително имейл, Slack, PagerDuty и други инструменти за комуникация и управление на инциденти. Това гарантира, че екипите могат бързо да реагират на възникнали проблеми.
- Интеграция с други инструменти: Sentry лесно се интегрира с различни инструменти за разработка и DevOps, като GitHub, Jira, Trello, CI/CD системи и други. Това улеснява управлението на грешки и инциденти в рамките на цялостния процес на разработка и поддръжка на софтуера.

## Одит функционалности на Sentry:

Въпреки че Sentry е основно насочена към мониторинг на грешки, тя предлага и функции за одит записи, които са полезни за проследяване на събития, свързани с дейността на потребителите и сигурността:

- История на събитията: Sentry записва история на събитията в приложението, като например потребителски влизания, промени в конфигурациите, стартиране на процеси и други. Това позволява на администраторите да преглеждат и анализират минали събития за целите на сигурността и съответствието.
- Потребителски следи: Платформата проследява действията на конкретни потребители, което е особено полезно за разследване на подозрителни дейности или за диагностика на проблеми, които се появяват само при определени условия.
- Регистриране на грешки в контекста на потребителска активност: Sentry не само записва грешките, но и свързва всяка грешка с конкретни потребителски действия, предоставяйки по-добра представа за причините за възникване на грешката.
- Одит съобщенията на конфигурационни промени: Sentry записва и регистрира промени в конфигурацията на самото приложение или на платформата Sentry, което помага за проследяване на това кой е направил промяната и кога е била извършена.



Фиг. 7 – Sentry OTEL collector

Примери за употреба на Sentry в контекста на одит записи:

- Мониторинг на достъпа до чувствителни части на приложението: Системните администратори могат да използват Sentry за проследяване на опити за достъп до критични функции на приложението, като административни панели или конфигурационни настройки.
- Проследяване на промени в потребителските права: Sentry може да записва всяка промяна в правата на потребителите, като например добавяне или премахване на административни привилегии, което е важно за сигурността и съответствието.
- Докладване на подозрителни активности: При откриване на подозрителни действия, като многократни неуспешни опити за логин или опити за достъп до забранени ресурси, Sentry може да изпраща нотификации на отговорните лица за предприемане на мерки.

Предимства на Sentry като одит система:

- Лесна интеграция: Sentry се интегрира бързо и лесно с различни приложения и платформи, което я прави удобна за употреба без големи усилия от страна на разработчиците.
- Контекстната информация: Детайлната информация, която Sentry предоставя относно контекста за всяко събитие или грешка, улеснява анализирането и отстраняването на проблеми.
- Гъвкави нотификации: Възможността за конфигуриране на нотификации според специфичните нужди на екипа помага за бързото реагиране на инциденти и проблеми.
- Широка поддръжка на платформи и езици: Sentry поддържа множество програмни езици и платформи, което я прави универсално решение за различни типове приложения.

Недостатъци на Sentry като одит система:

- Ограничена фокус: Основният фокус на Sentry е върху грешки и изключения, а не върху цялостното управление на събитията и сигурността, което може да не отговаря на нуждите на организации, които се нуждаят от по-широка одит система.
- Ограничения в бесплатната версия: В бесплатната версия на Sentry има ограничения по отношение на броя на грешките и събитията, които могат да бъдат проследявани, което може да бъде пречка за по-големи организации.
- Липса на специфични функционалности за сигурност: Въпреки че Sentry предлага някои одит възможности, тя не е специализирана платформа за управление на сигурността, което може да ограничи нейната употреба в силно регламентирани среди.

Заключение:

Sentry е мощен инструмент за проследяване на грешки и мониторинг на приложения, който предлага и някои основни аудит лог функционалности. Тя е особено подходяща за разработчици, които търсят лесен начин за идентифициране и отстраняване на грешки в своите приложения, като същевременно следят за сигурността и активността

на потребителите. Въпреки че не е пълноценна аудит лог система, Sentry предлага достатъчно функционалности, които могат да бъдат полезни в контекста на проследяване на потребителска активност и осигуряване на сигурност.

### 1.2.3. Съпоставка на разгledаните решения

Разгledаните по-горе решения са най-известните и използвани за събиране, визуализиране, анализиране на данни и алармиране. Тези решения са разработвани от големи екипи в продължение на много години. В Таблица 1 ще направим базова съпоставка на тези решения. Таблица 2 показва платформите, които решенията поддържат.

*Таблица 1 - Съпоставка на The Splunk Platform, Auditd и Sentry*

Критерии	The Splunk Platform	Auditd	Sentry
Компания	Splunk Founded: 2003 United States	Linux Foundation Part of the Linux kernel	Datadog Founded: 2010 United States
Решение с отворен код	✗	✓	✓
Съхраняване на данните в публично хранилище	✓	✗	✗
Поддръжка	✓	✗	✓
Лесна конфигурация на одит системата	✓	✗	✓
Платено решение	✓	✗	Lite: ✗ Standard: ✓ Pro: ✓ Enterprise: ✓

Таблица 2 - Поддържани платформи

Платформи	The Splunk Platform	Auditd	Sentry
Windows	✓	✗	✓
Mac	✓	✗	✓
Linux	✓	✓	✓
SaaS	✓	✗	✓
Iphone	✓	✗	✓
Android	✓	✗	✓

### 1.3. Заключение

В заключение може да се каже, че поставените цели са амбициозни, но напълно постижими. Разгледаните съществуващи решения предоставят на потребителите необходимите инструменти за одит система. Те предлагат различни подходи за събиране, съхранение, анализ и известяване на събития, свързани със сигурността и дейността на потребителите. Въпреки че всяко от тези решения има своите предимства и недостатъци, нашата цел е да разработим система, която да е лесна за конфигуриране, безплатна и достатъчно гъвкава, за да отговори на нуждите на потребителя, без да изиска сложни настройки или допълнителни разходи.

## Втора глава

### 2.1. Основна цел

Основната цел на тази дипломна работа е да разработи решение за одит система, която да бъде напълно безплатна, лесна за използване и независима от външни доставчици. Това решение има за цел да улесни софтуерните разработчици, като им предостави гъвкава и сигурна платформа за съхранение, криптиране и изчитане съобщения, без нуждата от скъпи абонаменти или сложни настройки. Подобен инструмент би могъл да бъде полезен и за студенти и ученици в техните проекти.

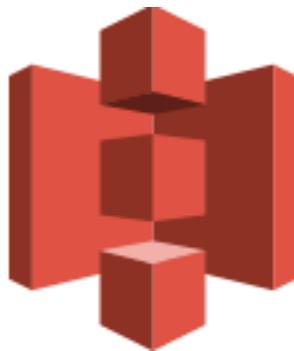
- Напълно безплатно решение – Докато решения като Splunk и Sentry изискват плащане за пълния набор от функционалности, разработваното от мен решение ще бъде напълно безплатно за използване. Това се постига чрез използването на инструменти с отворен код, които могат да се инсталират и стартират на локална машина или сървър, без да е необходим скъп хардуер или абонаменти.
- Гъвкавост в съхранението на данни – Повечето от съществуващите решения съхраняват данните в публични хранилища, което може да не се хареса на потребители, които държат на поверителността и контрола върху своите данни. Моят подход предоставя на потребителя пълна свобода в избора на място за съхранение на данните – било то локално, в частен облак, или на платена виртуална машина, като същевременно избягва ограниченията и рисковете, свързани с публичните хранилища.
- Лесна настройка без нужда от акаунт – Настройката и регистрацията често са сложни и времеемки процеси при решения като Splunk и Sentry. В контраст, моето решение е проектирано така, че да изисква минимални усилия от страна на потребителя. Няма нужда от регистрация или сложни настройки на акаунт. Потребителят трябва само да предостави базова конфигурация и да стартира необходимите докер контейнери, след което може да започне да подава данните в предварително дефиниран формат. Този подход значително намалява времето за внедряване и улеснява интеграцията в съществуващите системи.

## 2.2. Анализ на налични хранилища за обекти

Правилният подбор на хранилище за данните, които ще получаваме от устройствата, е от голямо значение. По-долу ще бъдат разгледани различни хранилища за обекти, които биха могли да се използват за текущия продукт.

### 2.2.1. 2Amazon S3

Amazon Simple Storage Service (Amazon S3) е услуга за съхранение на обекти, която предлага мащабируемост, достъпност на данни, сигурност и производителност. Това означава, че всякакви клиенти могат да го използват за съхраняване и защита на различно количество данни, като например потоци от данни, уебсайтове, мобилни приложения, архивиране и възстановяване, корпоративни приложения, IoT устройства и големи анализи на данни. Amazon S3 предлага функции за управление, така че да може данните да се организират и да се конфигурират различно настроени контроли за достъп, за да може напълно да отговори на нуждите на специфични бизнес, организационни и изисквания за съответствие. Amazon S3 е проектиран за 99,999999999% издръжливост и съхранява данни на милиони приложения за компании от цял свят.

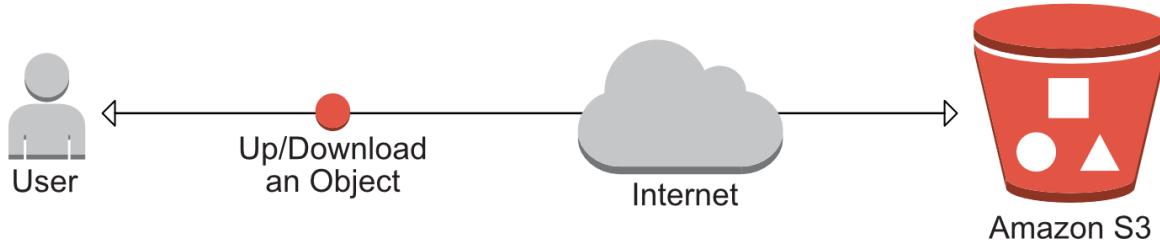


Фиг. 8 - Лого на Amazon S3

Обектите са основните единици, съхранявани в Amazon S3. Обектите се състоят от обектни данни и метаданни. Метаданните са набор от двойки име-стойност, които описват обекта.

Обектите в Amazon S3 се съхраняват в bucket-и (кофи) (Фиг. 9). Bucket-а представлява регион от данни. Неговата роля е да организира namespace-ите на най-високо ниво и да идентифицира акаунта, отговорен за таксите за съхранение и пренос на данни.

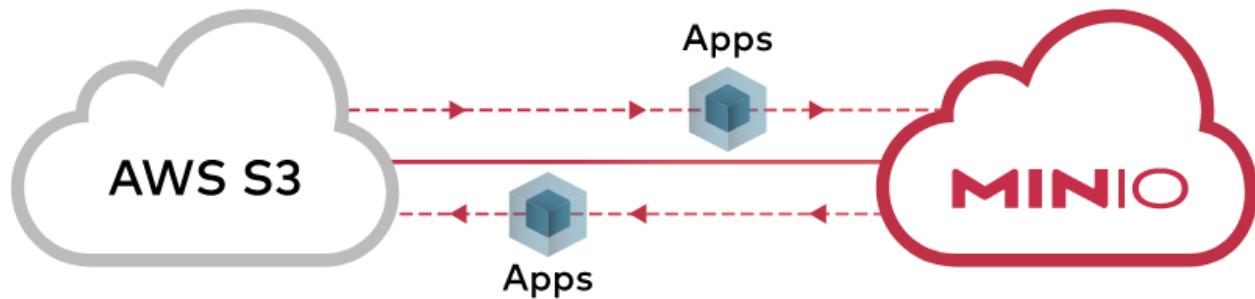
Той играе основна роля в контрола на достъпа и служи за агрегатна единица за отчитана на употребата.



Фиг. 9 - Пример за качване и изтегляне на обект от Amazon S3 Bucket

### 2.2.2. MinIO

MinIO е услуга с отворен код, която служи за съхраняване на обекти. MinIO представлява server-side софтуерно хранилище, което е съвместимо с Amazon S3 (Фиг. 10). Приложенията, които са конфигурирани да си говорят с Amazon S3, също могат да бъдат конфигурирани да си говорят с Minio, това позволява на Minio да бъде добра алтернатива на S3, в случай че клиентът иска повече контрол над сървъра за съхранение на обекти.



Фиг. 10 - Постигане на съвместимост между AWS S3 и MinIO

MinIO се различава по това, че е създаден от самото начало като стандарт за частно/хибридно облачно хранилище за обекти. Услугата съхранява неструктурирани данни като снимки, видеоклипове, регистрационни файлове, резервни копия, изображения и дори може да осигури един сървър за съхранение на обекти, който обединява множество устройства, разпределени в много сървъри. Minio е написан на Go, идва с command line client плюс интерфейс в браузъра. MinIO стекът съдържа три основни компонента: MinIO Server, MinIO Client и MinIO Client SDK.

*MinIO Server* е облачно хранилище, проектирано да бъде мащабирано. Той е достатъчно лек и правен за висока производителност, предлага набор от функции, специфични за големи корпоративни внедрявания. *MinIO Server* е хардуерно базиран, той може да бъде инсталиран на физически или виртуални машини, както и да бъде стартиран като Docker контейнер и разположен на платформи за оркестрация на контейнери като Kubernetes, Mesosphere и Docker Swarm.

*MinIO Client* е известен още като mc, който е настолен клиент за управление на файлове, съхранявани в съвместими с Amazon S3 сървъри. *MinIO Client* (mc) предлага модерна алтернатива на UNIX команди като ls, cat, cp, mirror, diff, find и т.н.

*MinIO Client SDK* може да се използва от разработчиците на приложения за взаимодействие с Amazon S3 съвместими сървъри. *MinIO Client SDK* предоставя API-та за достъп до всеки сървър за съхранение на обекти, съвместим с Amazon S3. Съществуват *MinIO Client SDK* за Java, Python, .NET, JavaScript и Go.

### 2.2.3. Google Cloud Storage

Google Cloud Storage (GCS) позволява съхранение и извлечане на различно количество данни по целия свят по всяко време. GCS може да се използва за редица сценарии, включително обслужване на съдържание на уебсайт, съхраняване на данни за архивиране и възстановяване при бедствия или разпространение на големи обекти от данни на потребители чрез директно изтегляне.

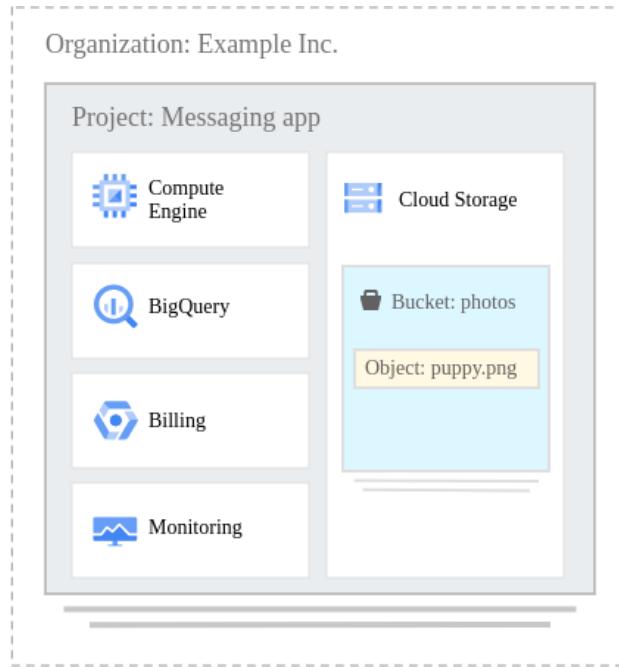


Фиг. 11 - Лого на Google Cloud Storage

Google Cloud Storage е RESTful онлайн услуга, която съчетава производителността и мащабируемостта на облака на Google с усъвършенствани възможности за сигурност и

споделяне. GCS е Infrastructure as a Service (IaaS), сравнима с онлайн услугата за съхранение на Amazon S3.

Посредством GCS съхраняваме обекти в Google Cloud. Обектът е част от данни, които не се изменят, и представлява файл от всякакъв формат. Обектите се съхраняват в контейнери, наречени buckets (кофи). Всички buckets са свързани с проект и това позволява групиране на проектите в организация (Фиг. 12).



Фиг. 12 - Примерна структура на Google Cloud Storage

#### 2.2.4. Сравнителен анализ

За реализацията на решението към текущия научен труд ще трябва да бъдат подбрано достатъчно надеждно хранилище за обекти, където да можем да съхраняваме данните, които ще бъдат получавани. Сравнителен анализ на гореспоменатите хранилища за обекти е представен в Таблица 3.

Таблица 3 - Сравнителен анализ на Amazon S3, MinIO и GCS

Критерии	Amazon S3	MinIO	GCS
Надеждност (availability)	99.99%	Зависи от машината, на която работи	99.99%
Издръжливост (durability)	99.99999999%	Зависи от машината, на която работи	99.99999999%
Максимален брой buckets	По подразбиране всеки акаунт може да създава до 100 buckets, ако се нуждае от повече, може да ги увеличи до 1000	Няма лимит	Няма лимит
Максимален брой обекти в bucket	Няма лимит	Няма лимит	Няма лимит
Максимален размер на обекта	5TiB	5TiB	5TiB
Максимален размер на обекта при операция за добавяне	5GB	5TiB	5TiB
Максимален брой части при операция добавяне	10 000	10 000	10 000
Максимален брой части, върнати при заявка за части от списък	1000	10 000	1000
Максимален брой обекти, върнати при заявка за обекти от списък	1000	4500	1000
Цена	<i>S3 Standard:</i> \$0.022/GB/month to \$0.024/GB/month <i>S3 Intelligent-Tiering:</i> \$0.0018/GB/month to \$0.024/GB/month <i>S3 Standard-Infrequent Access:</i> \$0.0131/GB/month <i>S3 One Zone-Infrequent Access:</i> \$0.01048/GB/month <i>S3 Glacier:</i> \$0.0045/GB/month <i>S3 Glacier Deep Archive:</i> \$0.0018/GB/month	<i>Community:</i> Безлатно <i>Standard:</i> \$10/TB/month <i>Enterprise:</i> \$20/TB/month	<i>Standard Storage:</i> \$0.020/GB/month <i>Nearline Storage:</i> \$0.010/GB/month <i>Coldline Storage:</i> \$0.004/GB/month <i>Archive Storage:</i> \$0.012/GB/month
Услуга с отворен код	✓	✗	✗

Трите разгледани и анализирани по-горе услуги са достатъчно еднакви за целите на разработката. Изборът на технология се опира до цената, лиценза и възможността услугата да се използва локална машина. В случая услугата MinIO ще бъде най-доброят избор, тъй като е безплатна и всеки потребител може да я инсталира на своята локална машина, което означава, че клиентите не са задължени да държат своите данни в публични хранилища, за които да плащат.

## 2.3 Анализ на налични Message Queue Системи

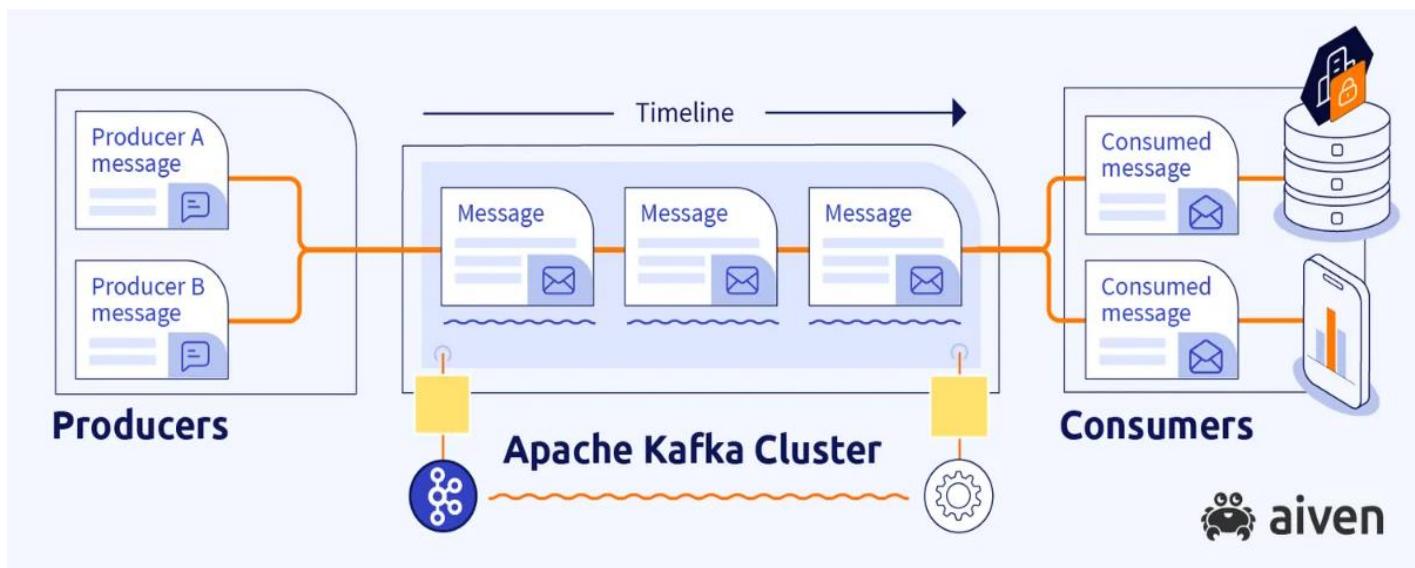
Message Queue (MQ) системите са софтуерни механизми, които позволяват обмен на съобщения между различни части на приложение или между различни приложения. Те поддържат асинхронна комуникация, като съобщенията се поставят в опашка и чакат да бъдат обработени. Основни качества на MQ системите:

1. **Надеждност:** Гарантират доставка на съобщения, дори при сривове.
2. **Устойчивост:** Съхраняват съобщения на диск, за да предотвратят загуба при срив.
3. **Скалируемост:** Обработват увеличаващ се обем съобщения, без да губят производителност.
4. **Производителност:** Бърза обработка на съобщения, дори при големи обеми.
5. **Асинхронност:** Позволяват изпращане на съобщения без изчакване за обработка.
6. **Гъвкавост:** Поддържат различни комуникационни модели (напр. точка-точка, публикуване/абониране).
7. **Транзакционна поддръжка:** Осигуряват консистентност чрез групова обработка на съобщения.
8. **Лесна интеграция:** Поддържат различни езици и платформи за програмиране.

MQ системите са ключови за създаването на надеждни и мащабируеми разпределени системи.

### 2.3.1 Apache Kafka

Apache Kafka е разпределена стрийминг платформа, която първоначално е разработена от LinkedIn и по-късно стана част от Apache Software Foundation. Kafka е проектирана за висока производителност, скалируемост и постоянство на съобщенията, като поддържа както обработка на данни в реално време, така и съхранение на историята на съобщенията.



Фиг. 13 – Apache Kafka диаграма

Силни страни:

- Изключително висока производителност и мащабируемост.
- Поддържа съхранение на съобщения за дълъг период.
- Подходяща за обработка на данни в реално време.
- Разпределена архитектура с висока надеждност.

Слабости:

- Комплексна настройка и управление.
- Изиска значителни ресурси за мащабиране.

## 2.3.2 RabbitMQ

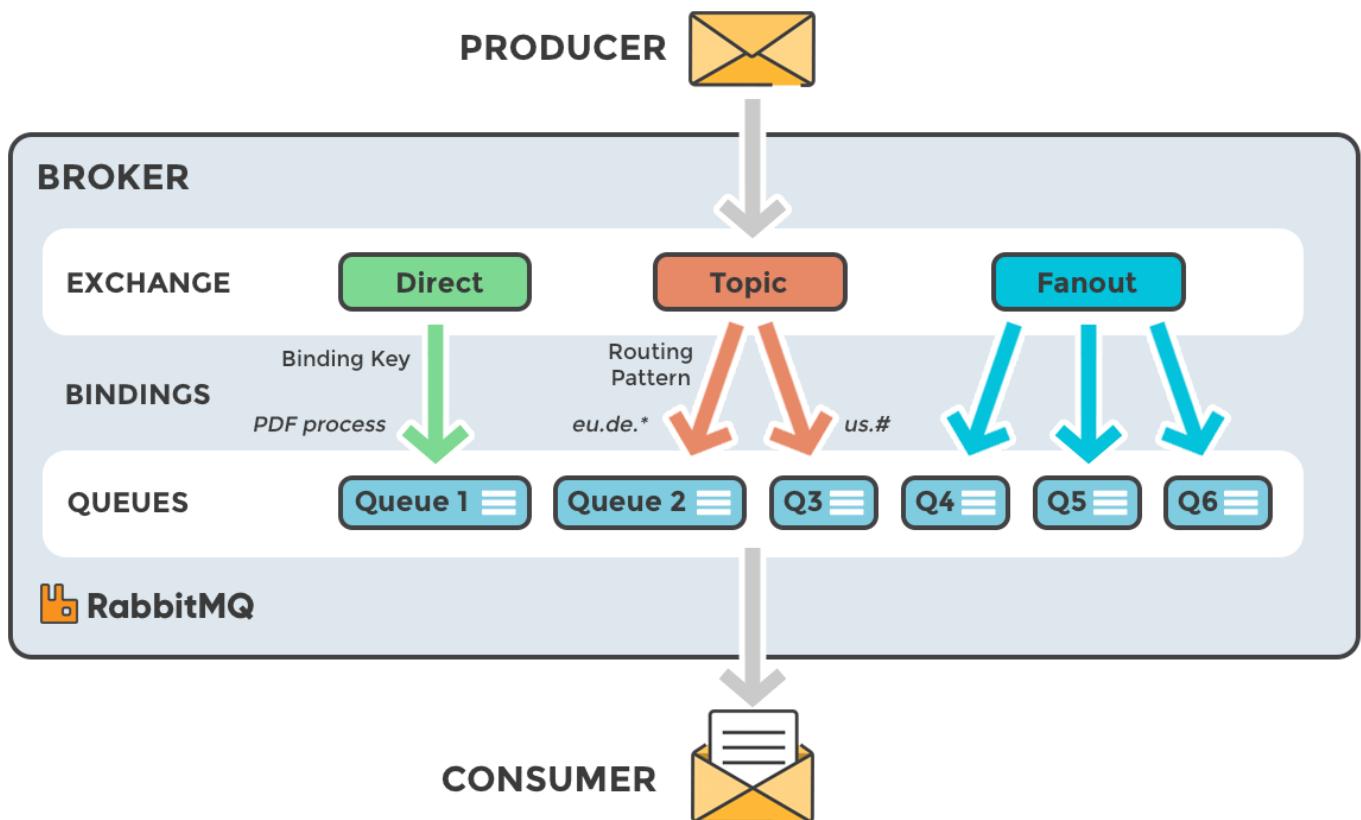
RabbitMQ е една от най-старите и популярни message broker системи, базирана на AMQP (Advanced Message Queuing Protocol). Тя е разработена от Pivotal Software и предлага лесен за използване интерфейс и гъвкави възможности за обработка на съобщения.

Силни страни:

- Поддържа различни протоколи (AMQP, MQTT, STOMP).
- Лесен за настройка и интеграция с различни системи.
- Отлична документация и активна общност.
- Подходящ за по-малки и средни натоварвания.

Слабости:

- Ограничена производителност в сравнение с Kafka.
- По-трудно мащабиране в сравнение с разпределени системи като Kafka.



Фиг. 14 – RabbitMQ диаграма

### 2.3.3 Amazon SQS

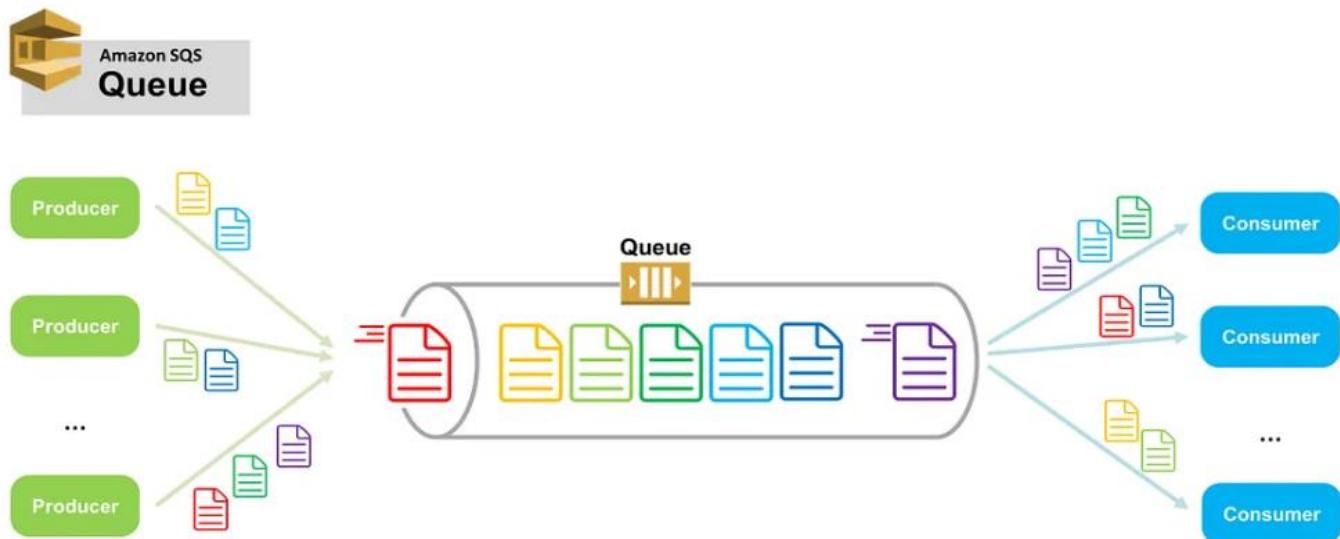
Amazon SQS е напълно управлявана услуга за опашка от съобщения, предлагана от AWS (Amazon Web Services). SQS е лесен за използване и не изиска управление на сървъри, тъй като се поддържа напълно от AWS.

Силни страни:

- Напълно управлявана услуга, която не изиска администриране.
- Лесна интеграция с други услуги на AWS.
- Автоматично мащабиране според нуждите.
- Надеждност и висока достъпност благодарение на инфраструктурата на AWS.

Слабости:

- По-ниска производителност в сравнение с Kafka и RabbitMQ.
- Ограничена функционалност в сравнение с самостоятелно хоствани решения.
- Разходи, които нарастват при по-високи натоварвания.



Фиг. 15 – Amazon SQS диаграма

### 2.3.4 Сравнителен анализ

За реализацията на решението към дадения научен труд тряба да се избере message queue, което да предлага надеждност, висока производителност, скалируемост и да разполага с безплатна версия. Сравнителен анализ на горепосочените message queues е представен в Таблица 4.

*Таблица 4 - Сравнителен анализ на Apache Kafka, RabbitMQ и Amazon SQS*

Критерии	Apache Kafka	RabbitMQ	Amazon SQS
Подходящи за	Стриминг на данни в реално време, големи обеми данни	Обща цел, по-малки натоварвания	Лесна интеграция с AWS, опростено използване
Протоколи	Kafka протокол	AMQP, MQTT, STOMP	Притежаван от AWS протокол
Скалируемост	Изключително висока	Добра, но не като Kafka	Автоматично мащабиране, но с ограничена функционалност
Надеждност	Много висока (разпределена архитектура)	Висока, но изиска ръчно настройване	Много висока (управляван от AWS)
Производителност	Много висока	Висока	Средна
Хостинг	Локален или облачен	Локален или облачен	Облачен (AWS)
Разходи	Ниска цена за мащабиране, но изиска ресурси	По-ниска в сравнение с Kafka	Разходите зависят от употребата

От направения сравнителен анализ става ясно, че разгледаните message queue системи предлагат много сходни функционалности, което означава, че всяка от тях би могла да бъде подходяща за реализиране на нашето решение. Въпреки това, Apache Kafka се откроява със своята мащабируемост, висока производителност и активна поддръжка от общността с отворен код. Освен това, Kafka е напълно безплатна, което е от съществено значение, тъй като една от основните цели на текущата дипломна работа е да предложи на потребителя напълно безплатно и ефективно решение. В този контекст Apache Kafka напълно отговаря на нашите изисквания и ще бъде идеалният избор за нашето решение.

## 2.4 Сравнителен анализ на програмни езици

Изборът на програмен език за реализация на текущия научен труд се основава на изискванията за използваните технологии и интеграция с Apache Kafka. Подборът на език ще се ръководи от няколко ключови критерия: наличието на клиентски SDK и библиотеки за комуникация с Kafka, производителност, продуктивност и поддръжка на frameworks за разработка на RESTful приложения.

- Java:

Java е универсален, обектно-ориентиран език за програмиране, разработен от Sun Microsystems. Той е предпочитан избор за работа с Apache Kafka, тъй като Kafka е разработена на Java и предлага пълна поддръжка на Java клиентски библиотеки. Предимството на Java е лесната преносимост на различни платформи чрез Java Virtual Machine (JVM), което осигурява съвместимост и надеждност. Java разполага с мощни frameworks за разработка на RESTful приложения като Spring Boot, Micronaut и Quarkus, които значително улесняват създаването на скалируеми и производителни решения. Въпреки тези предимства, един от недостатъците на Java е необходимостта от писане на повече код в сравнение с по-нови езици, което може да увеличи сложността и времето за разработка.

- Python:

Python е интерпретиран, обектно-ориентиран език за програмиране на високо ниво с динамична семантика, създаден от Guido van Rossum и издаден през 1991 г. Той е изключително популярен за бърза разработка на приложения, благодарение на прости си и четлив синтаксис. Python поддържа Kafka чрез официалната библиотека `confluent-kafka-python`, която позволява лесна интеграция с Kafka. Освен това, Python има мощни frameworks за разработка на RESTful приложения като Django и Flask, които улесняват създаването на уеб услуги. Основните предимства на Python са неговата гъвкавост и скорост на разработка, което го прави отличен избор за бързо прототипиране и изграждане на сложни системи. Един недостатък на Python е, че в сравнение с Java и Go, може да има по-ниска производителност при работа с много големи обеми от данни.

- Go:

Go, или Golang, е създаден от Google и е компилиран език, известен със своята висока производителност, лекота на използване и вградена поддръжка за конкурентност. Go е отличен избор за работа с Kafka, особено за разработка на високопроизводителни и мащабируеми приложения. Kafka има поддръжка за Go чрез клиентската библиотека `sarama`, която е широко използвана в производствени среди. Go е предпочитан език за разработка на микросервизи и високонадеждни системи, благодарение на своята ефективност и прост синтаксис. Основните предимства на Go са лесното писане на конкурентен код и бързото изпълнение, което го прави идеален за изграждане на производителни Kafka-базирани системи. Недостатък на Go е, че може да изисква повече усилия за имплементация на сложни функционалности в сравнение с Python, но предимствата му в производителността често компенсират това.

Таблица 5 - Сравнителен анализ на Java, Python и Go

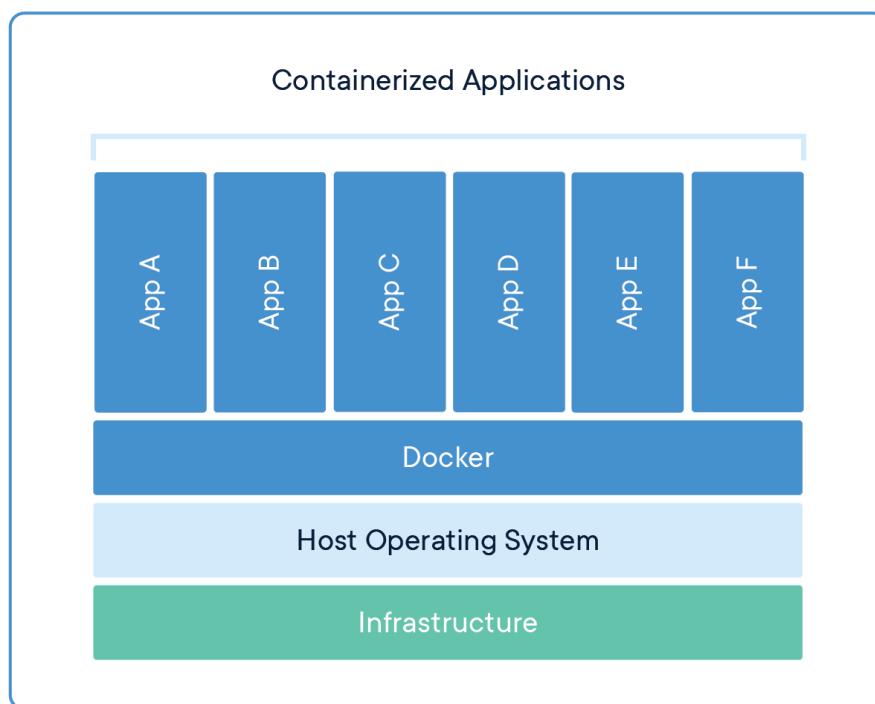
Критерии	Java	Python	Go
Интеграция с Kafka	Пълна поддръжка чрез официални я Kafka клиент	Поддържа се чрез confluent-kafka-python	Поддържа се чрез sarama
Производителност	Висока	Средна	Висока
Синтаксис	Строг и типизиран	Прост и динамичен	Прост и типизиран
Разработка на RESTful API	Spring Boot, Micronaut, Quarkus	Django, Flask	Gin, Echo
Гъвкавост и мащабируемост	Много добра	Много добра	Много добра
Кривата на обучение	Средна	Ниска	Средна

Java е предпочтеният избор за този проект поради своята стабилност, мощни инструменти за разработка и дълбока интеграция с Apache Kafka. Тези характеристики ще осигурят надеждност и производителност, необходими за успешната реализация на нашето решение.

## 2.5 Docker контейнери

Контейнерът е стандартна единица софтуер, която пакетира кода и всички негови зависимости, така че приложението да работи бързо и надеждно от една компютърна среда в друга. Docker е отворена платформа за разработване, доставка и стартиране на приложения. Той позволява отделянето на приложението от инфраструктурата, за да може да се постигне бързо доставяне на софтуер. С него може да се управлява инфраструктурата по същия начин, по който може да се управляват приложението. Docker контейнерът е лек, самостоятелен, изпълним софтуерен пакет, който включва всичко необходимо за стартиране на приложение - код, системни инструменти, системни библиотеки и настройки.

Контейнерите са изолирани един от друг и обединяват собствен софтуер, библиотеки и конфигурационни файлове, комуникацията помежду им става посредством точно определени канали. Множество контейнери могат да се изпълняват на една и съща машина и да споделят ядрото на ОС с други контейнери, всеки от които се изпълнява като изолиран процес в потребителското пространство. (Фиг. 16)



Фиг. 16 - Пример за организацията на Docker контейнери

Docker Hub е най-голямото хранилище в света за изображения (images) на контейнери с множество източници на съдържание, включително разработчици на проекти с отворен код и независими доставчици на софтуер, изграждащи и разпространяващи техния код в контейнери.

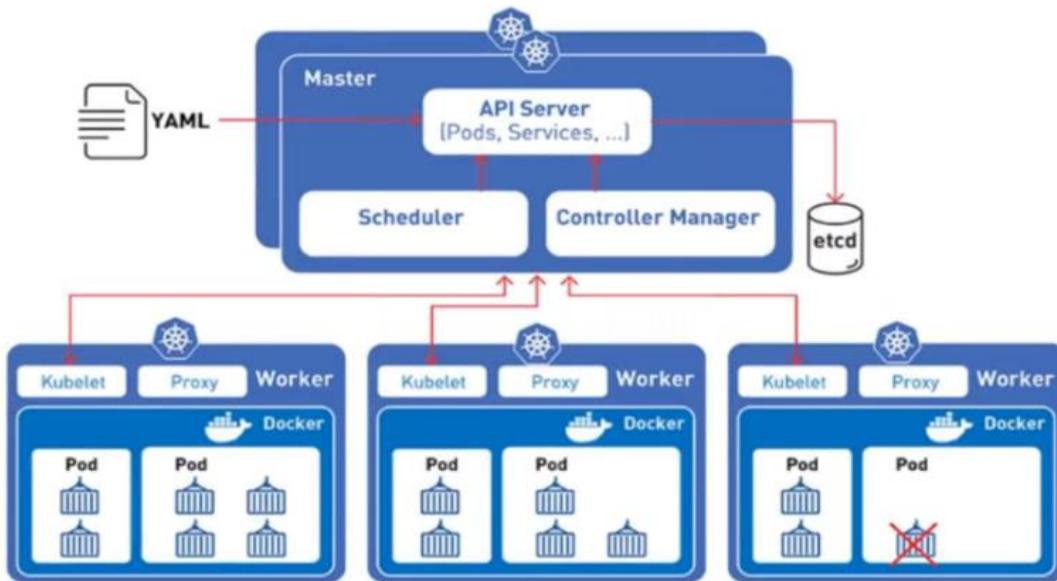
В текущия научен труд Docker контейнерите ще се използват с цел решението да бъде изградено и разпространено по-лесно. По този начин разработката ще бъде независима от операционната система, върху която ще работи.

## 2.6 Kubernetes

Kubernetes (K8s) е платформа с отворен код, която автоматизира разполагането, управлението и мащабирането на контейнеризирани приложения. Тя улеснява управлението на приложения, съставени от множество контейнери, като автоматично разпределя и управлява ресурсите им, осигурява мащабируемост и възстановяване след грешки. Kubernetes предлага функции като автоматично възстановяване на неработещи контейнери, управление на конфигурации, както и балансиране на натоварването.

Разработен е от Google и първоначално представен през 2014 г. Днес Kubernetes се поддържа от Cloud Native Computing Foundation (CNCF) и се използва широко от компании от всякакъв мащаб за управление на контейнеризирани приложения в облака и на локални сървъри. Той е идеален за изграждане на сложни и мащабируеми системи, особено в среда с микросервизи.

# Kubernetes Architecture



Фиг. 17 – Kubernetes архитектура

## 2.7 Системни изисквания

В тази глава ще бъдат разгледани функционалните и нефункционалните изисквания за системата към този научен труд.

### 2.7.1 Функционални изисквания

- Системата трябва да съхранява данните, които потребителят предоставя.
- Системата трябва да гарантира, че съобщението не може да бъде загубено.
- Системата трябва да позволява потребителя възможно най-лесно да достъпва данните.
- Системата трябва да позволява потребителя да може да филтрира по различни полета от съобщението.
- Системата трябва да се грижи за сигурността на данните посредством криптиране.
- Системата трябва да предоставя данните само на оторизирани потребители.
- Системата трябва да валидира дали съобщенията които потребителят изпраща следват определената структура.
- Системата трябва да записва данните изключително бързо и мигновено да връща отговор на потребителят.

- Системата трябва да има възможност за лесна скалируемост.
- Системата трябва да може да се инсталира на всички операционни системи.
- Системата трябва да предоставя RESTful приложения, които да могат лесно да се използват от потребителите без значение от програмният език.

### 2.7.2 Нефункционални изисквания

- Всички отделни модули на системата трябва да живеят в Docker контейнери.
- Инсталрирането на системата да е бързо и лесно.
- Да има документация как се използва системата.
- Всички модули трябва да се стартират бързо и лесно.
- За създаване на RESTful приложението трябва да се използва език, който е лесен за четене и за поддръжка.
- Системата е предназначена за софтуерни разработчици и ползватели.
- Системата ще е напълно бесплатна.

## 2.8 Обобщение

При разработването на сложни системи е от критично значение да се извърши подробно проучване на всички компоненти и тяхната интеграция. Всеки компонент трябва да бъде внимателно подран, за да осигури оптимална производителност и ефективност на цялата система.

В тази разработка, MinIO е избран за основно хранилище на обекти, тъй като предлага напълно бесплатно решение и позволява съхранение на данни в локални или частни облаци, без да се налага използването на публични хранилища.

Kubernetes е избран за управление на контейнеризирани приложения, осигурявайки автоматизация на разполагането, управлението и мащабирането на приложениета. Това ще улесни управлението на всички необходими на системата компоненти и ще осигури надеждност и мащабируемост.

Java & Spring Boot са избрани за разработка на RESTful приложенията. Spring Boot предоставя мощни инструменти за бързо и ефективно разработване на приложения, което ще ускори процеса на създаване на необходимите услуги.

Apache Kafka ще бъде използван за система за обмен на съобщения, предоставящи висока производителност и надеждност при обработката на данни в реално време. Kafka ще събира и управлява потока от данни, но няма да предоставя информацията директно на потребителите.

Всички компоненти на системата ще бъдат разпространявани и управлявани чрез Docker контейнери, което ще улесни тяхната имплементация и поддръжка. Kubernetes ще се грижи за оркестрацията на контейнерите, осигурявайки стабилност и мащабируемост на решението.

## Трета глава

Основната задача на разработваната одит система е да реши проблемите, разгледани в предходната глава. Един от основните приоритети е системата да бъде предоставена безвъзмездно, да не обременява потребителя със заплащане за използването на хранилища и да изиска минимални настройки от негова страна.

Архитектурата на системата ще бъде изградена така, че да позволява лесното ѝ разпространение. Това е важно, тъй като няма да задължава потребителя да разполага със специфичен хардуер или операционна система, върху които да бъде стартирана системата. Използването на контейнери и платформи като Docker и Kubernetes ще осигури съвместимост и гъвкавост, като същевременно ще улесни внедряването на решението в различни среди.

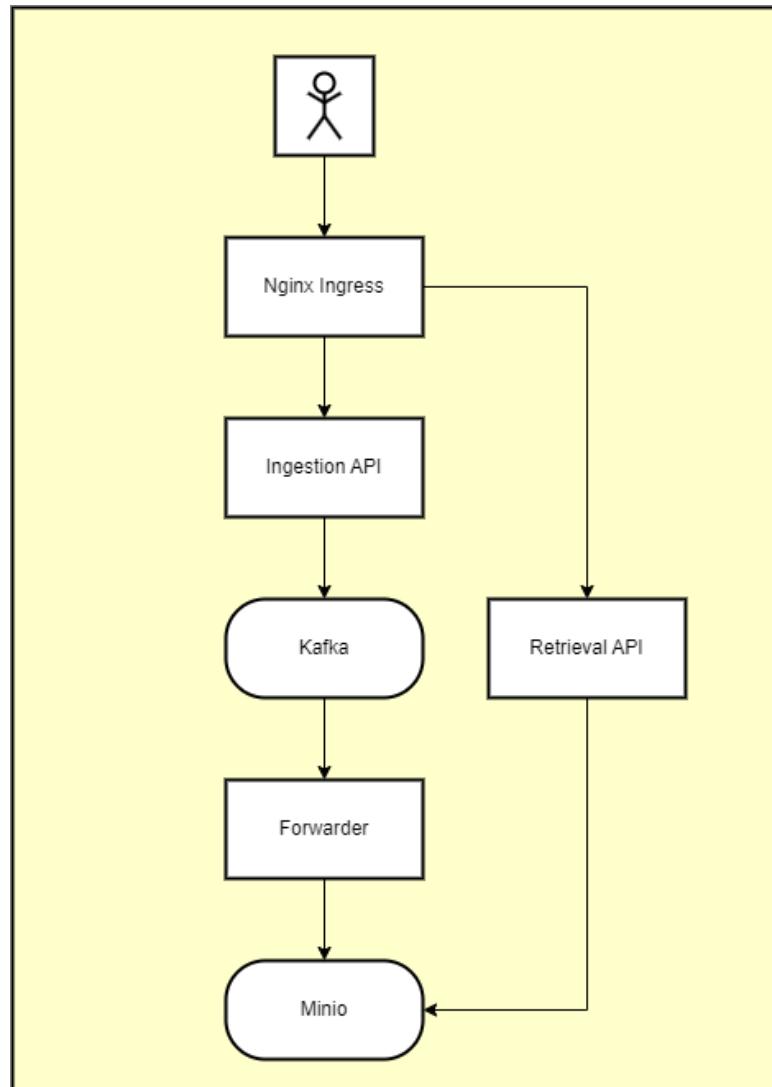
В тази глава ще бъде описан процесът на проектиране и дизайн на одит системата според поставените по-горе изисквания и задачи. Също така ще бъдат разгледани използваните библиотеки, инструменти и връзките между отделните услуги, които съставляват системата. Ще се обърне внимание на основните компоненти като записване на данни, тяхното съхранение и защита, изчитане и филтрация, както и на сигурността и мащабируемостта на системата.

### 3.1. Основна архитектура

На приложената диаграма е представена цялостната архитектура на текущата одит система. Когато клиентът се опитва да запише одит съобщение, заявката първо преминава през нашия Nginx контролер, който въз основа на зададения URL път „/ingestion/send“ я пренасочва към нашето Ingestion API. След като Ingestion API провери дали съответният клиент има права за запис и валидира структурата на съобщението, то записва съобщението в нашия Apache Kafka клъстер и връща отговор на клиента. След това друг компонент от системата отговаря за изчитането на съобщението от Apache Kafka, криптирането му и успешното записване в дългосрочното хранилище MinIO.

След като съобщението е изпратено и успешно записано, клиентът има възможност да го изчете, като изпрати заявка отново чрез нашия Nginx контролер, използвайки пътя

“/retrieval/read” и добавяйки необходимите филтри. Когато заявката достигне нашето Retrieval API, тя проверява дали клиентът има права за достъп и дали заявката е във валиден формат. Ако всичко е наред, Retrieval API изчита съобщенията от MinIO, декриптира ги и връща желания резултат на клиента.



Фиг. 18 – Одит система

## 3.2. Docker контейнери

За по-лесното разпространяване на системата всеки от нейните компоненти ще бъде стартиран в отделен Docker контейнер. За реализацията на системата ще е нужно да използваме 6 контейнера:

1. Nginx
2. Ingestion API
3. Apache Kafka
4. Forwarder
5. MinIO
6. Retrieval API

### 3.2.1. Nginx

Nginx е мощен и широко използван уеб сървър, който също така изпълнява ролята на обратен прокси, load balancer, и API gateway. Основната му функция е да разпределя входящите HTTP заявки към съответните бекенд услуги въз основа на конфигурации, като предоставя възможност за лесна мащабируемост и повищена производителност на уеб приложенията.

В контекста на тази дипломна работа, Nginx се използва като Ingress контролер, който разпределя входящите HTTP заявки към различни микросървиси, базирани на пътя в URL адреса. Това означава, че Nginx анализира пътя на заявката (например /ingestion/send или /retrieval/read) и пренасочва трафика към съответния микросървис, който е конфигуриран да обработи тази пътешка.

Тази функционалност е изключително полезна в архитектури, базирани на микросървиси, където множество малки услуги работят независимо една от друга. Nginx осигурява централизирана точка за управление на тези заявки, като позволява гъвкаво разпределение на трафика, баланс на натоварването между различните инстанции на микросървиси и подобрена сигурност чрез SSL терминализация и други механизми за защита.

Използването на Nginx като Ingress контролер в текущата система гарантира, че заявките ще бъдат ефективно разпределени и обработени от правилните микросървиси, което допринася за стабилността, мащабируемостта и сигурността на цялостното решение.

### 3.2.2. Ingestion API

Ingestion API компонентът е критична част от нашата одит система, реализиран като Java Spring приложение. Този компонент отговаря за приемането на съобщения от клиентите и осигурява няколко слоя на валидация и сигурност, преди съобщенията да бъдат записани в Apache Kafka.

Когато клиент изпрати съобщение към Ingestion API, първата стъпка е проверката на предоставения OAuth токен. За нашата разработка използваме Auth0 като сървър за управление на автентикацията. Ingestion API валидира токена срещу Auth0, за да гарантира, че клиентът е легитимен и има необходимите права за достъп. Ако токенът е невалиден или липсва, API връща HTTP статус код 401 Unauthorized, като информира клиента, че достъпът е отказан.

След успешна валидация на OAuth токена, Ingestion API преминава към следващия етап, а именно валидирането на структурата на самото съобщение. Това се осъществява посредством JSON Schema Validator, който проверява дали изпратеното съобщение отговаря на предварително дефинираната JSON схема. Ако структурата на съобщението не е валидна или съдържа грешки, Ingestion API връща HTTP статус код 400 Bad Request, което указва на клиента, че изпратените данни не отговарят на изискванията.

Ако съобщението премине успешно всички проверки, Ingestion API се опитва да го запише в Apache Kafka. В нашата архитектура Apache Kafka е конфигурирана с replication factor 3, което означава, че всяко съобщение трябва да бъде репликирано на три различни worker nodes. За да се счита, че съобщението е успешно записано, поне две от трите ноди трябва да потвърдят успешното записване. Едва след това Ingestion API връща успешен отговор към клиента, потвърждавайки, че съобщението е правилно съхранено и обработено.

### 3.2.3. Apache Kafka

Apache Kafka играе ключова роля в нашата одит система като основната платформа за съхранение и предаване на съобщения. Тя действа като високопроизводителен, разпределен message queue, който осигурява надеждно и мащабирамо предаване на данни между различните компоненти на системата. В нашата архитектура Kafka служи като централен буфер, в който валидираните съобщения се записват, преди да бъдат криптирани и съхранени дългосрочно в MinIO. Чрез използването на replication factor 3, Kafka гарантира, че съобщенията са защитени и достъпни дори при отказ на една или повече реплики, осигурявайки висока устойчивост и сигурност на данните в системата.

### 3.2.4. Forwarder

Другият ключов компонент в нашата одит система е Forwarder, който играе критична роля в осигуряването на сигурността и надеждността на съхраняваните данни. Forwarder-ът е Java Spring приложение, което се грижи за криптирането на съобщенията и тяхното безопасно прехвърляне към дългосрочното хранилище MinIO. Той функционира като Kafka консуматор, което означава, че постоянно следи за новопостъпили съобщения в Apache Kafka. Веднага щом бъде засечено ново съобщение, Forwarder-ът го изчита незабавно, криптира го с предварително дефиниран алгоритъм и го записва в MinIO. Този процес гарантира, че данните са не само съхранени сигурно, но и че това става с възможно най-малко забавяне, осигурявайки надеждността на системата.

### 3.2.5. MinIO

За дългосрочното съхранение на данни в нашата дипломна работа сме избрали MinIO, разпределена система за съхранение на обекти, съвместима с Amazon S3. MinIO е проектиран да управлява големи обеми данни по сигурен и мащабирам начин, което го прави подходящ за съхранение на одит съобщения. В нашата система, MinIO служи като крайната точка, където криптираните съобщения, обработени от компонента Forwarder, се съхраняват.

MinIO осигурява висока сигурност и надеждност, както и лесна интеграция с останалите компоненти. Освен това, неговата производителност и мащабируемост позволяват ефективно управление на нарастващия обем данни, без значителни разходи за допълнителни ресурси. Този подход е ключов за осигуряване на дългосрочна достъпност и интегритет на данните в системата.

### 3.2.6. Retrieval API

Retrieval API е Java Spring компонент който е създаден с цел да предостави на потребителите и техните одитори възможността да извлекат записаните одит съобщения. При получаване на заявка, Retrieval API първо извършва проверка на валидността на OAuth токена чрез Auth0 сървъра, за да се увери, че клиентът има необходимите права за достъп.

След успешна валидация на токена, компонентът проверява дали заявката съдържа валидни аргументи. Задължителен параметър е времето, което определя периода, за който се търсят съобщения. Освен това, потребителят има възможност да добави допълнителни филтри, като тип на съобщението, потребителско име или тенант, което позволява по-прецизно търсене и извлечане на конкретни данни.

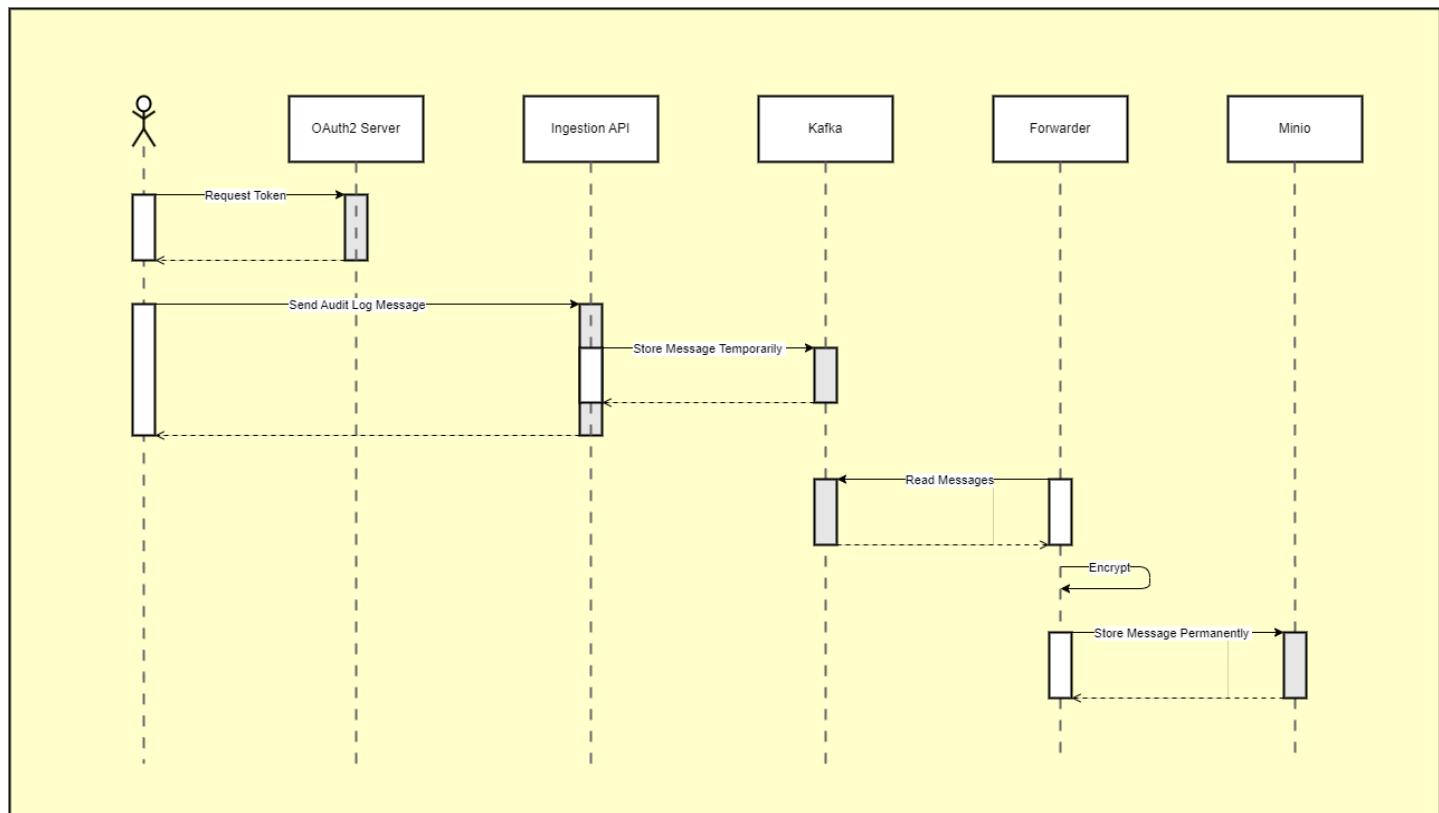
След като заявката бъде валидирана и параметрите проверени, Retrieval API изчita съответните съобщения от MinIO. Извлечените съобщения се декриптират и се прилагат зададените от клиента филтри. Накрая, компонентът връща резултатите под формата на JSON, което осигурява структурирана и лесно достъпна информация на потребителя. Тази конфигурация гарантира както сигурност, така и гъвкавост при управлението и достъпа до одит логовете.

### 3.2.7. Обобщение

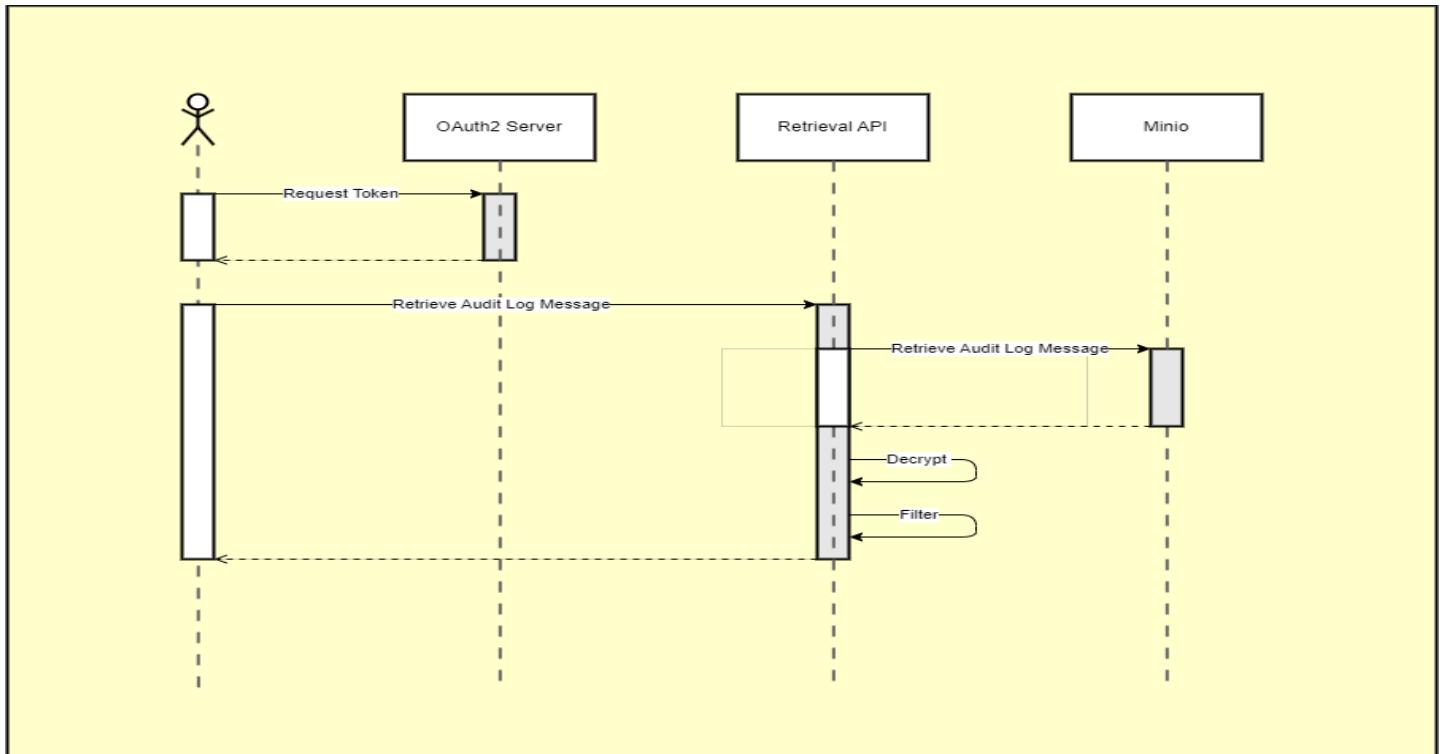
В текущата дипломна работа е разработена цялостна одит система, която отговаря на специфичните нужди на потребителите за сигурно и ефективно управление на одит данни. Системата се състои от няколко ключови компонента: Ingestion API, Forwarder, MinIO и Retrieval API.

Ingestion API е отговорен за приемането и валидирането на одит съобщения, като проверява валидността на OAuth токена и структурата на съобщението. Ако всичко е в ред, съобщението се записва в Apache Kafka. Forwarder компонентът следи Kafka за нови съобщения, криптира ги и ги записва в дългосрочното хранилище MinIO. Retrieval API позволява на потребителите да извлекат съобщения от MinIO чрез валидиране на токен и филтриране на данните по зададени критерии.

Системата е проектирана да бъде бесплатна и лесна за конфигурация, без да изиска специален хардуер или допълнителни настройки от страна на потребителя. За допълнителна яснота, приложените sequence diagrams ще илюстрират взаимодействията между различните компоненти на системата и процесите по обработка на одит съобщенията.



Фиг. 19 – Записване на данни одит система



Фиг. 20 – Изчитане на данни одит система

### 3.3. Предпоставки за бъдещо развитие

Създадената до момента одит система успешно изпълнява основните си функционалности. Въпреки това, съществуват възможности за разширяване и подобряване на текущата система.

**Интеграция със SIEM системи:** В бъдеще, системата може да бъде разширена с интеграция към SIEM (Security Information and Event Management) системи. Това ще позволи анализиране на съобщенията в контекста на сигурността и откриване на потенциални заплахи или аномалии. Чрез интеграцията със SIEM системи, като Splunk или Elastic SIEM, одит данните могат да бъдат по-добре корелирани с други източници на информация и използвани за по-подробен анализ и реакция на инциденти.

**Потребителски интерфейс (UI):** Добавянето на интуитивен и функционален потребителски интерфейс ще улесни взаимодействието на потребителите със системата. UI може да предостави лесен достъп до извлечане на одит съобщения, задаване на филтри, както и визуализация на данни и отчети. Такъв интерфейс ще направи системата по-достъпна и лесна за използване от различни типове потребители, без технически опит.

**Разширени аналитични функции:** Могат да бъдат внедрени допълнителни аналитични функции за дълбочинно разглеждане на одит данните. Например, системата може да включва модули за анализ на тенденции, идентификация на необичайни активности и автоматизирано създаване на отчети. Такива функции ще подобрят способността на системата да предоставя ценни инсайти и препоръки на потребителите.

**Модули за прогноза:** В бъдеще, могат да бъдат добавени модули за прогнозиране на данни, базирани на съществуващите одит съобщения. Това може да включва предсказване на потенциални проблеми или рискове на базата на исторически данни, което ще подобри проактивното управление на системата.

Тези допълнения и подобрения ще разширят функционалността на системата и ще я направят още по-ценна за потребителите, осигурявайки по-добро управление и анализ на одит данните.

### 3.4. Заключение

Разработената одит система комбинира ключови компоненти за осигуряване на ефективно и разширяемо решение за управление на одит съобщения. Архитектурата на системата е проектирана с модулен подход, който позволява лесно разпространение и интеграция, благодарение на Docker контейнери.

Основните компоненти на системата включват:

- **Ingestion API:** Реализирано с Java Spring, това приложение приема одит съобщения, извършва необходимите проверки и записва в Apache Kafka.
- **Apache Kafka:** Осигурява мащабируемо и устойчиво разпределение на съобщенията, използвайки механизъм за репликация за повишаване на надеждността.
- **Forwarder:** Java Spring приложение, което изчита съобщенията от Kafka, криптира ги и ги записва в дългосрочното хранилище MinIO. Той гарантира сигурността и надеждността на съхранението на данните.
- **MinIO:** Използва се като дългосрочно хранилище за одит съобщения, предоставяйки висока достъпност и съвместимост с Amazon S3 API.

- Retrieval API: Реализирано с Java Spring, това приложение позволява на потребителите да извлекат съобщения от MinIO. След извършване на необходимите проверки декриптира съобщенията и ги връща на потребителя.
- Nginx: Използва се като Ingress Controller, който насочва заявките към съответните микроуслуги въз основа на пътя в URL-а, осигурявайки ефективно управление на трафика.

Комбинацията от тези компоненти, съчетана с Docker и Kubernetes, осигурява надеждност и гъвкавост на системата. Която от своя страна предлага цялостно решение за управление на одит съобщения, демонстрирайки успешната интеграция на различни технологии. В бъдеще, може да се обмислят допълнителни функционалности, като интеграция с SIEM системи и добавяне на потребителски интерфейс, за подобряване на функционалността и удобството за потребителите.

# Четвърта глава

Тази глава ще служи като практическо ръководство за използване на системата в реални проекти, включително конкретни стъпки за стартиране и конфигуриране на компонентите. Ще разгледаме подробно необходимите инструменти и процеса на инсталация, също така ще бъде засегната и интеграцията със Swagger документацията и възможностите за генериране на клиенти чрез OpenAPI Specification.

## 4.1. Потребителско ръководство за стартиране на системата

За да стартирате и използвате нашата одит система, ще ви е необходим Kubernetes кълъстер, който може да бъде настроен с помощта на Minikube. Системата се състои от няколко ключови компонента, включително Ingestion API, Apache Kafka, Forwarder, MinIO, Retrieval API, Nginx и OAuth сървър. Тези компоненти работят заедно, за да осигурят сигурно записване и извлечане на одит съобщения.

### 4.1.1 Изисквания и инсталация на необходимите инструменти:

Преди да стартирате системата, уверете се, че на вашата машина са инсталирани следните инструменти:

- Java Development Kit (JDK): Необходим за изпълнение на Java Spring приложенията като Ingestion API, Forwarder и Retrieval API.
- Maven: Използва се за изграждане на Java проектите и управление на зависимостите.
- Minikube: Инструмент за стартиране на локален Kubernetes кълъстер, където ще бъдат разположени компонентите.
- Helm: Инструмент за управление на Kubernetes пакети, който се използва за разгръщане на приложението в кълъстера.

### 4.1.2 Конфигурация и стартиране на системата:

След като сте инсталирали необходимите инструменти, можете да стартирате системата чрез `start.sh` скрипта, който ще извърши следните действия:

- Билдване на компонентите: Скриптът ще използва Maven за изграждане на Ingestion API, Forwarder и Retrieval API.

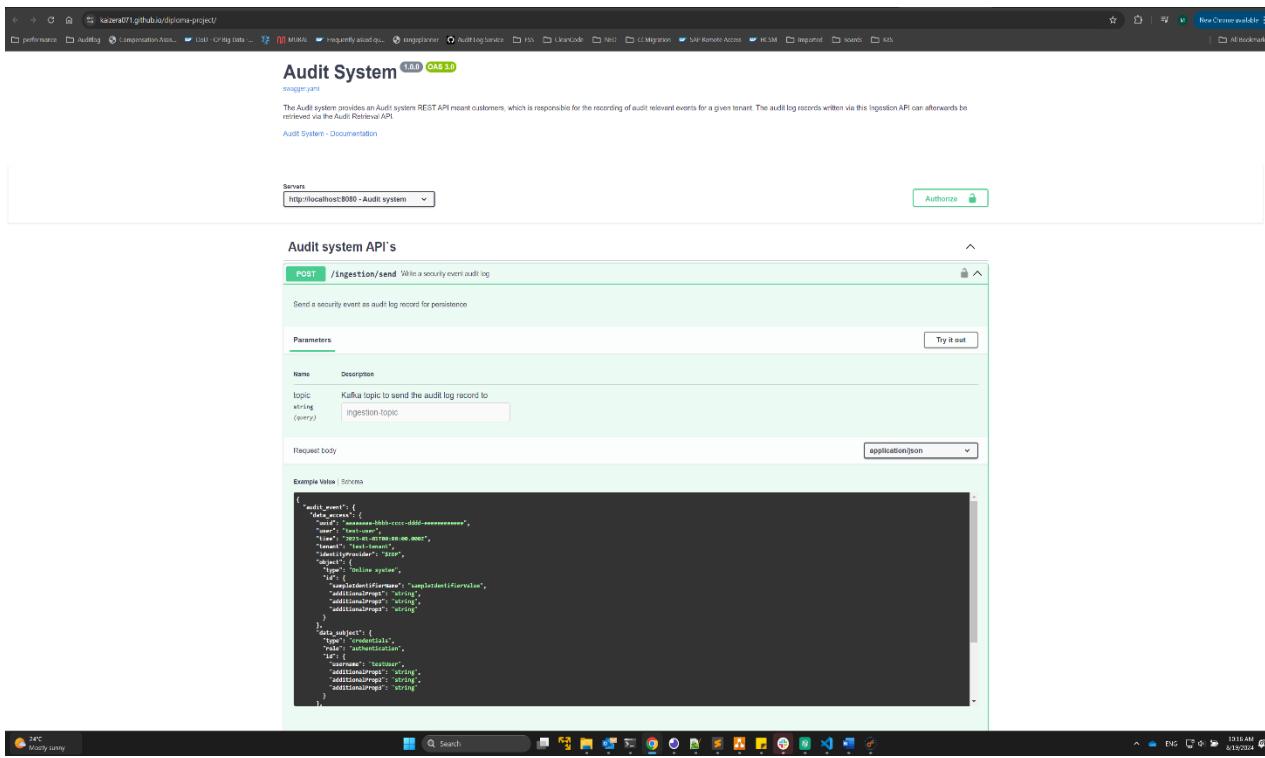
- Инсталация и конфигурация: Компонентите ще бъдат инсталирани и конфигурирани в Minikube кълстера. Това включва:
  - Разгръщане на Apache Kafka за обработка и съхранение на съобщенията.
  - Настройване на Forwarder, който ще изчита съобщенията от Kafka, ще ги криптира и записва в MinIO.
  - Конфигуриране на MinIO като дългосрочно хранилище за данните.
- Генериране на криптографски ключове: При стартирането се генерира уникален ключ за криптиране и декриптиране на съобщенията, който се използва от Forwarder и Retrieval API.
- Nginx конфигурация: Nginx се използва като Ingress контролер, който разпределя заявките към съответните микросървии въз основа на пътя в URL адреса.

#### 4.1.3 Предварителна конфигурация на OAuth сървъра

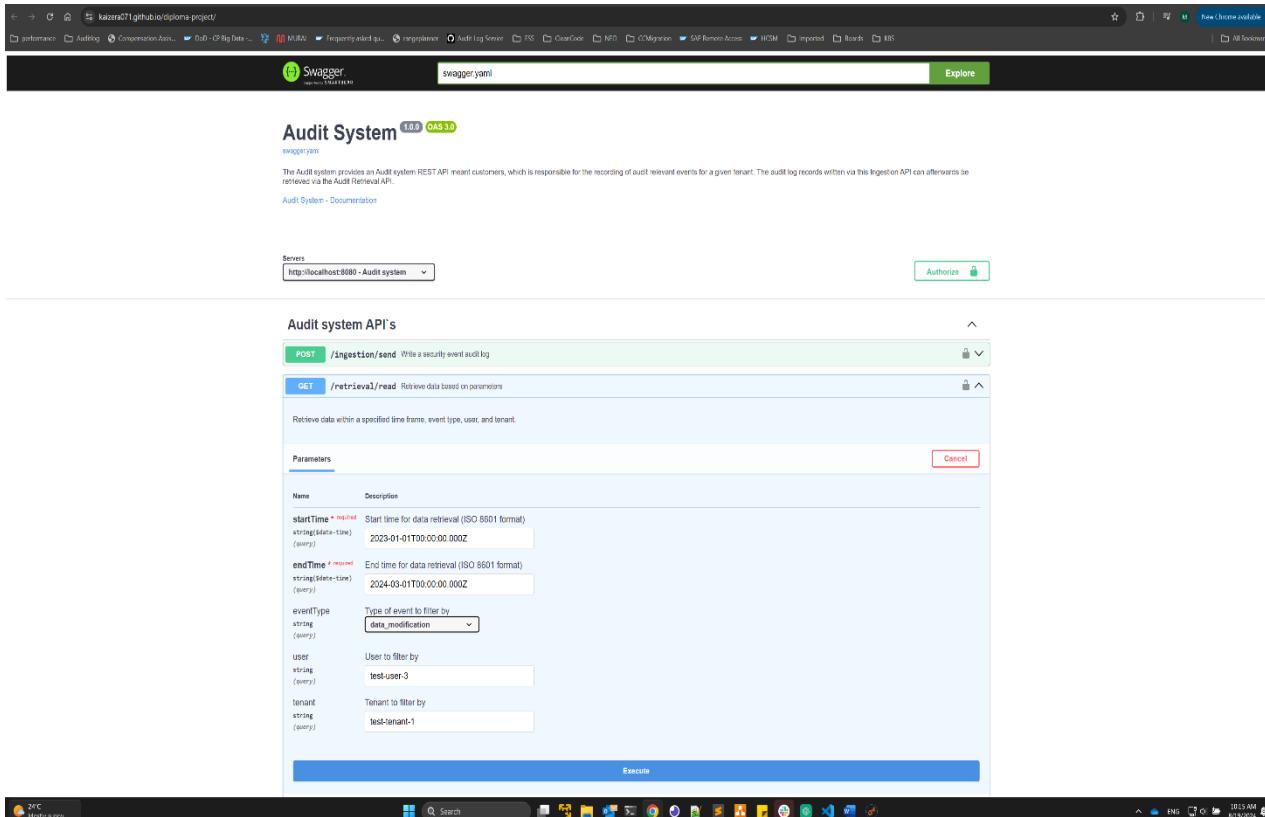
Важно е да се отбележи, че OAuth (Auth0) сървърът трябва да бъде предварително конфигуриран и готов за употреба. Системата разчита на този сървър за валидиране на OAuth токените, които осигуряват сигурен достъп до различните API точки.

#### 4.1.4 Използване на Swagger документация и OpenAPI Specification

Swagger документацията за системата е предварително написана и достъпна за потребителите. Тази документация описва всички API точки и може да бъде използвана за генериране на клиенти на различни програмни езици чрез OpenAPI Specification, което улеснява интеграцията с външни приложения. Документацията е хостната с помощта на GitHub Pages и може да бъде намерена в следното репозитори (<https://kaizera071.github.io/diploma-project/>). Малко по-долу в тази глава ще представим изображения, демонстриращи основните функционалности на системата.



Фиг. 21 – Swagger записване на данни – одит система



Фиг. 22 – Swagger изчитане на данни – одит система

#### 4.1.5 Достъп и работа със системата

След успешно стартиране на системата, можете да започнете да използвате различните API точки, предоставени чрез Nginx. Swagger интерфейсът ви позволява лесно да се ориентирате в наличните функции и да тествате различни заявки. Тази система е гъвкава и лесно може да бъде адаптирана към специфичните нужди на потребителя, благодарение на своята модулна архитектура и използването на Docker контейнери.

## 4.2. Заключение

Системата, разработена в този научен труд, е проектирана така, че да осигури на потребителите лесен и интуитивен начин за работа с одит данни. Чрез използването на надеждни и утвърдени технологии като Java, Spring Boot, Apache Kafka, MinIO и Kubernetes, системата позволява ефективно съхранение, криптиране, и извлечане на одит съобщения. В допълнение, интеграцията с OAuth сървър и наличието на подробно документирани API интерфейси чрез Swagger улесняват взаимодействието с компонентите на системата.

В бъдеще може да се добави интеграция със SIEM система, което ще позволи по-ефективно наблюдение и управление на сигурността на данните. Освен това, разработването на потребителски интерфейс (UI) ще улесни още повече взаимодействието със системата, предоставяйки на потребителите интуитивен начин за управление и визуализация на одит съобщенията.

# Пета глава

## 5.1. Тестова постановка

Тестовата постановка представлява реализираната софтуерна система в състояние на експлоатация. За целта ще бъде стартирана системата на дадена машина и ще бъде демонстрирана пълната функционалност на разработката към текущия научен труд.

Тестовата постановка включва използване на системата на машина със следните характеристики:

- *Лаптоп* - Lenovo T15g Gen2
- *Процесор* - 11th Gen Intel(R) Core(TM) i9-11950H @ 2.60GHz 2.61 GHz
- *RAM памет* - 64.0 GB (63.7 GB usable)
- *Операционна система* - Windows 11 Enterprise
- *Тип на системата* - 64-битова операционна система

## 5.2. Тестване на системата

Тестването на одит системата е ключов етап от процеса на осигуряване на качеството и надеждността на разработеното решение. Този процес е неразделна част от жизнения цикъл на системата, като в него участват различни заинтересовани страни, включително програмисти, софтуерни инженери, ръководители на проекта и крайни потребители. За разработчиците, основната цел на тестването е да потвърдят, че всеки компонент на одит системата функционира в съответствие с предварително зададените изисквания. След като всички компоненти са проверени и валидирани от техническия екип, следващата стъпка е да се уверим, че системата удовлетворява очакванията и изискванията на крайните потребители, осигурявайки лесно и сигурно управление на данните.

### 5.2.1. Функционално тестване

Функционалното тестване на системата е основен аспект от този процес. То включва проверка на всяка функция на софтуера чрез подаване на конкретни входни данни и проверка на изходните резултати спрямо зададените функционални изисквания. Тестването се извършва с цел да се установи, че системата работи правилно и не проявява неочеквани поведенчески или функционални проблеми. В следващият абзац са описани основните тестови сценарии за функционалното тестване на одит системата. По време на тестовете не са наблюдавани сривове или некоректно поведение, което потвърждава стабилността и надеждността на системата.

Запис на валидни данни:

- Описание: Тества дали валидните одит съобщения се записват успешно в системата.
- Тестови данни: Коректно форматирани одит съобщения с валидни OAuth токени.
- Стъпки: Подайте валидни данни към Ingestion API, което трябва да ги запише в Apache Kafka.
- Очакван резултат: Данните трябва да бъдат записани успешно в Kafka и да са налични за последваща обработка.
- Статус: Успешен

Обработка на невалидни данни:

- Описание: Тества дали системата правилно обработва невалидни одит съобщения и връща подходящ код за грешка.
- Тестови данни: Невалидни одит съобщения или съобщения с невалидни OAuth токени.
- Стъпки: Подайте невалидни данни към Ingestion API и проверете реакцията.
- Очакван резултат: Системата трябва да върне код за грешка (например 400 Bad Request) и да отхвърли съобщението.
- Статус: Успешен

Криптиране и запис на съобщения от Forwarder:

- Описание: Тества дали Forwarder компонентът правилно криптира съобщенията и ги записва в MinIO.
- Тестови данни: Одит съобщения, които са записани в Apache Kafka.
- Стъпки: Провери дали Forwarder слуша нови съобщения в Kafka, криптира ги и ги записва в MinIO.
- Очакван резултат: Съобщенията трябва да бъдат успешно криптирани и записани в MinIO без загуба на данни.
- Статус: Успешен

Четене и декриптиране на данни от Retrieval API:

- Описание: Тества дали Retrieval API успешно чете и декриптира съобщения от MinIO.
- Тестови данни: Криптирани одит съобщения, записани в MinIO.
- Стъпки: Подайте заявка за четене на съобщения към Retrieval API, проверете дали API декриптира съобщенията и ги връща в правилния формат.
- Очакван резултат: Retrieval API трябва да декриптира съобщенията правилно и да ги предостави на потребителя в JSON формат.
- Статус: Успешен

Обработка на заявки с невалиден токен от Ingestion/Retrieval API's

- Описание: Тества дали Ingestion/Retrieval API's правилно обработва заявки с невалидни OAuth токени.
- Тестови данни: Заявки със невалиден или изтекъл OAuth токен.
- Стъпки: Подайте заявки към Ingestion/Retrieval API's с невалиден токен и наблюдавайте реакцията.
- Очакван резултат: API трябва да върне код за грешка (например 401 Unauthorized) и да отхвърли заявките.
- Статус: Успешен

### 5.2.2. Модулно тестване

Модулното тестване е ключов етап от процеса на разработка на софтуер, който се фокусира върху проверката на отделни компоненти или единици от кода, за да се увери, че те функционират според зададените изисквания. Това тестване се извършва по време на разработката и има за цел да потвърди, че всяка единица, като функция, метод, процедура, модул или обект, работи правилно и изолирано от останалата част на системата. Модулното тестване е изключително важно за ранното откриване на грешки и дефекти.

За одит системата, модулното тестване е проведено на ключови компоненти, включително Nginx, Ingestion API, Apache Kafka, Forwarder, Retrieval API и MinIO. В следващите точки са представени тестовите сценарии за всеки от тези компоненти. Тези модулни тестове са извършени по време на разработката на системата и са насочени към осигуряване на коректността на отделните компоненти. По този начин се гарантира, че всеки компонент функционира правилно, което е критично за цялостната стабилност и надеждност на одит системата.

## Модулно тестване на Ingestion API:

- **Описание:** Тества функционалността на Ingestion API за правилна обработка на валидни и невалидни одит съобщения.
- **Тестови данни:** Валидни и невалидни одит съобщения, както и валидни и невалидни OAuth токени.
- **Стъпки:** Подайте валидни и невалидни съобщения и токени към Ingestion API. Проверете дали API коректно валидира токените и съобщенията.
- **Очакван резултат:** API трябва да приеме валидни съобщения и да отхвърли невалидните с подходящи кодове за грешки (например 401 или 400).
- **Статус:** Успешен

## Модулно тестване на Forwarder:

- **Описание:** Тества функционалността на Forwarder за криптиране и запис на съобщения в MinIO.
- **Тестови данни:** Одит съобщения, записани в Apache Kafka.
- **Стъпки:** Симулирайте нови съобщения в Kafka и проверете дали Forwarder ги криптира правилно и ги записва в MinIO.
- **Очакван резултат:** Съобщенията трябва да бъдат криптирани коректно и записани в MinIO, без загуба на данни.
- **Статус:** Успешен

## Модулно тестване на Retrieval API:

- **Описание:** Тества функционалността на Retrieval API за правилно четене и декриптиране на съобщения от MinIO.
- **Тестови данни:** Криптирани одит съобщения, записани в MinIO.
- **Стъпки:** Подайте заявки към Retrieval API за четене на съобщения. Проверете дали API декриптира съобщенията и ги връща в правилния формат.
- **Очакван резултат:** API трябва да декриптира съобщенията успешно и да предостави данните в JSON формат, както е описано.
- **Статус:** Успешен

## Модулно тестване на MinIO:

- **Описание:** Тества функционалността на MinIO за правилно съхранение и извлечане на данни.
- **Тестови данни:** Криптирани одит съобщения.
- **Стъпки:** Запишете и изтеглете съобщения от MinIO, за да се уверите, че данните са съхранени и извлечени правилно.
- **Очакван резултат:** Данните трябва да бъдат съхранени и извлечени без грешки и с точност.
- **Статус:** Успешен

Модулно тестване на Nginx:

- **Описание:** Тества конфигурацията на Nginx за правилно разпределение на заявките между различните компоненти на системата.
- **Тестови данни:** Заявки за различни пътища в URL.
- **Стъпки:** Подайте различни заявки към Nginx и проверете дали заявките се пренасочват към съответните API (Ingestion, Retrieval и т.н.) правилно.
- **Очакван резултат:** Nginx трябва да разпределя заявките към правилните компоненти според конфигурацията.
- **Статус:** Успешен

Модулно тестване на OAuth сървър:

- **Описание:** Тества функционалността на OAuth сървъра за валидиране на токени.
- **Тестови данни:** Валидни и невалидни OAuth токени.
- **Стъпки:** Подайте валидни и невалидни токени към OAuth сървъра и проверете реакцията на сървъра.
- **Очакван резултат:** Сървърът трябва да валидира валидните токени и да отхвърли невалидните с подходящи кодове за грешка (например 401 Unauthorized).
- **Статус:** Успешен

Модулно тестване на Apache Kafka:

- **Описание:** Тества функционалността на Apache Kafka за правилно съхранение и разпределение на одит съобщенията.
- **Тестови данни:** Одит съобщения, публикувани в Kafka топик.
- **Стъпки:** Подайте съобщения към Kafka топик и проверете дали съобщенията са правилно съхранени и разпределени. Проверете репликацията и консуматорите.
- **Очакван резултат:** Съобщенията трябва да бъдат правилно публикувани в съответните топики и успешно консумирани от приложенията, осигурявайки коректност и надеждност на разпределителната система.
- **Статус:** Успешен

### 5.2.3. Интеграционно тестване

Интеграционното тестване е критичен етап от процеса на осигуряване на качеството на софтуера, при който отделните софтуерни модули се интегрират и тестват като група. Целта на този вид тестване е да идентифицира дефекти, които могат да възникнат при взаимодействието между различните модули на системата. В типичен софтуерен проект, който често включва множество модули, създадени от различни програмисти или екипи, е изключително важно да се уверим, че тези модули работят добре заедно и осигуряват очакваната функционалност.

По време на интеграционното тестване, се извършва проверка на взаимодействията и комуникацията между компонентите на системата. Това включва тестване на интерфейси, проверки на предаване на данни и обработка на заявки между различните услуги и модули. В контекста на текущата система, интеграционното тестване ще се фокусира върху различните ключови компоненти, включително Ingestion API, Forwarder, MinIO, Apache Kafka, Retrieval API, Nginx и OAuth сървър. Целта е да се потвърди, че тези компоненти работят правилно заедно и че данните се предават и обработват коректно между тях.

Интеграционното тестване е проведено както по време на разработката на системата, така и след завършването ѝ, за да се уверим, че всички компоненти функционират заедно, както се очаква. В следващите точки са представени резултатите от извършените интеграционни тестове, които показват как различните части на системата се справят при взаимодействие помежду си. Този подход осигурява увереност, че интеграцията на различните модули и компоненти на системата не само функционира правилно, но също така отговаря на изискванията за производителност и надеждност.

### Интеграционно тестване на Ingestion API и Forwarder

- **Описание:** Тества взаимодействието между Ingestion API и Forwarder, за да се увери, че съобщенията, получени от API, се криптират и записват правилно от Forwarder.
- **Тестови данни:** Валидни одит съобщения и валидни OAuth токени.
- **Стъпки:** Подайте валидни съобщения към Ingestion API. След това проверете дали тези съобщения са успешно криптирани и записани в MinIO от Forwarder.
- **Очакван резултат:** Съобщенията, получени от Ingestion API, трябва да бъдат криптирани правилно и записани в MinIO, без загуба на данни и с коректна криптография.
- **Статус:** Успешен

## Интеграционно тестване на Forwarder и MinIO

- **Описание:** Тества връзката между Forwarder и MinIO, за да се увери, че криптираните съобщения са правилно съхранени в MinIO и могат да бъдат извлечени без грешка.
- **Тестови данни:** Одит съобщения, криптираны от Forwarder.
- **Стъпки:** Симулирайте нови съобщения в Kafka, криптирайте ги с Forwarder и ги запишете в MinIO. След това извлечете съобщенията от MinIO и проверете дали са криптираны правилно.
- **Очакван резултат:** Криптираните съобщения трябва да бъдат съхранени и извлечени без грешки и с точност.
- **Статус:** Успешен

## Интеграционно тестване на Retrieval API и MinIO

- **Описание:** Тества взаимодействието между Retrieval API и MinIO, за да се увери, че API може да чете и декриптира съобщения от MinIO.
- **Тестови данни:** Криптиран одит съобщения, съхранени в MinIO.
- **Стъпки:** Изпратете заявки към Retrieval API за четене на съобщения от MinIO. Проверете дали API успешно декриптира съобщенията и ги връща в правилния формат.
- **Очакван резултат:** Retrieval API трябва да декриптира съобщенията правилно и да предостави данните в JSON формат, както е описано.
- **Статус:** Успешен

## Интеграционно тестване на Nginx и всички API

- **Описание:** Тества конфигурацията на Nginx за правилно маршрутизиране на заявките към съответните API компоненти (Ingestion API, Retrieval API и т.н.).
- **Тестови данни:** Заявки за различни пътища в URL.
- **Стъпки:** Подайте различни заявки към Nginx и проверете дали заявките се пренасочват към съответните API компоненти (Ingestion, Retrieval и т.н.) правилно.
- **Очакван резултат:** Nginx трябва да разпределя заявките към правилните компоненти според конфигурацията, осигурявайки коректен достъп до всички API.
- **Статус:** Успешен

## Интеграционно тестване на OAuth сървър и API

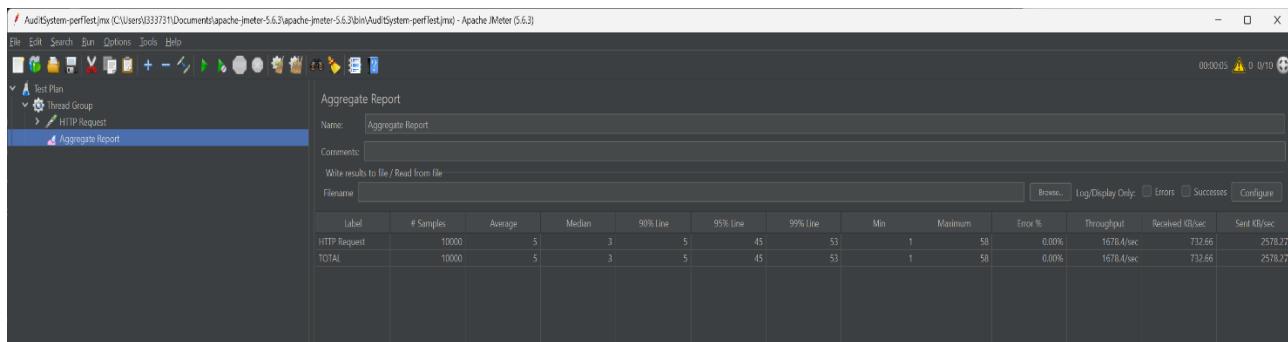
- Описание: Тества интеграцията на OAuth сървъра с различните API компоненти на системата, за да се увери, че токените са правилно валидирани при достъп до API.
- Тестови данни: Валидни и невалидни OAuth токени.
- Стъпки: Подайте валидни и невалидни токени към различни API и проверете дали OAuth сървърът коректно валидира токените и осигурява достъп или отказ на API.
- Очакван резултат: OAuth сървърът трябва да валидира валидните токени и да отхвърли невалидните с подходящи кодове за грешка (например 401 Unauthorized).
- Статус: Успешен

## Интеграционно тестване на Apache Kafka и Forwarder

- Описание: Тества връзката между Apache Kafka и Forwarder, за да се увери, че съобщенията, публикувани в Kafka, се консумират и обработват правилно от Forwarder.
- Тестови данни: Одит съобщения, публикувани в Kafka топик.
- Стъпки: Публикувайте съобщения в Kafka топик и проверете дали Forwarder ги консумира, криптира и записва в MinIO правилно.
- Очакван резултат: Съобщенията трябва да бъдат коректно консумирани от Forwarder и записани в MinIO без загуба на данни и с точност.
- Статус: Успешен

### 5.2.4. Тестване на производителността на системата

Тестването на производителността е критичен аспект от оценката на софтуерната система, който се фокусира върху измерването на ефективността и устойчивостта на системата под различни натоварвания. Целта на това тестване е да се определи как системата се справя с увеличаващ се обем на работа и да се идентифицират потенциални уязвими места или проблеми с производителността.



Фиг. 23 - Резултати от теста със 10000 заявки

В контекста на одит системата, проведеното тестване на производителността включва:

#### Тестови Сценарии и Резултати:

- Тестови Настройки: Тестовете са проведени с 10 активни нишки, при които са записвани общо 10 000 одитни съобщения.
- Средно Време за Отговор: Както се вижда от приложената картичка, системата демонстрира средно време за отговор от около 5 милисекунди при тези условия. Това показва, че системата е в състояние бързо да обработва и отговаря на заявките.
- Пропусклivост (Throughput): Системата успява да поеме около 1678 съобщения в секунда, което подсказва, че може да се справи с висок обем на натоварване и да осигури добра производителност при изпълнение на заявките.

#### Анализ на Производителността:

- Оптимизация: Въз основа на резултатите от тестовете, е възможно да се идентифицират области за оптимизация и подобрения, ако е необходимо, за постигане на още по-високи нива на ефективност.
- Стабилност: Проведените тестове демонстрират стабилността на системата при съществуващи условия, без да се наблюдават значителни забавяния или загуба на данни.

#### Докладване и Резултати:

- Докладване на Данни: Събранныте данни от тестовете предоставят ценна информация за текущото състояние на системата и позволяват идентифицирането на потенциални проблеми.
- Визуализация: Резултатите са представени визуално, което улеснява разбирането на производителността и демонстрира как системата се справя при натоварване.

Тестването на производителността предоставя важна информация за работата на системата под различни условия и помага да се уверите, че системата е способна да отговори на нуждите на потребителите при високи натоварвания. Получените резултати показват, че системата е в състояние да осигури бързо и ефективно обслужване на одитните съобщения.

### 5.3. Заключение

Резултатите от тестването на одит системата показват, че интеграцията на всички компоненти е успешна и системата отговаря на предварително дефинираните функционални изисквания. Всеки модул, включително Ingestion API, Forwarder, Apache Kafka, Retrieval API, MinIO, Nginx, и OAuth сървъра, функционира коректно и изпълнява зададените задачи, съгласно очакванията.

По отношение на производителността, системата демонстрира отлични резултати. Според проведените тестове, средното време за отговор остава стабилно при увеличаване на натоварването, а системата успява да поддържа висока пропускателна способност. Графиката от тестовете показва, че времето за отговор не нараства линейно с нарастващото натоварване, което подсказва добра скалируемост и ефективност на системата при обработка на високи обеми от данни.

В заключение, одит системата е проектирана и тествана успешно, осигурявайки надеждност и производителност, отговаряща на съвременните изисквания за обработка на одит съобщения. Системата е готова за внедряване и може да бъде използвана в реална среда с увереност в нейната ефективност и стабилност.

## **Заключение**

В настоящия научен труд е разгледан широк спектър от технологии и решения, които са събрани и интегрирани, за да се предостави една ефективна и функционална одит система. Разработката е насочена към събиране, криптиране, сигурно записване, както и възможност за декриптиране и изчитане на одит данни. В първа глава са определени целите и задачите на проекта, а втора глава е посветена на детайлен анализ на използваните технологии, включително хранилища за обекти, message queue системи и контейнери, които са избрани въз основа на техните предимства за реализиране на зададените цели.

Трета глава е фокусирана върху проектирането и реализацијата на одит системата, като всяка част от системата е описана в контекста на нейната роля и интеграция с другите компоненти. Особено внимание е отделено на конфигурирането на услугите и разработването на основните функционални модули. Също така, разгледани са перспективите за бъдещо развитие на системата. Докато в четвърта глава е предоставено подробно потребителско ръководство, което включва стъпките за стартиране, конфигуриране и интегриране със системата.

В пета глава са анализирани резултатите от експерименталните тестове, включително функционални, модулни, интеграционни и тестове за производителност. Резултатите показват, че системата успешно изпълнява функционалните изисквания и демонстрира висока производителност и надеждност. Интеграцията на компонентите е успешна и системата показва добра скалируемост при нарастващи натоварвания.

Крайният резултат от проекта постига основната цел на предоставяне на напълно безплатна одит система, която е конкурентоспособна спрямо съществуващите търговски решения. Тази система предлага иновативни технологии с отворен код, което дава възможност на всеки потребител да я имплементира в своите проекти без финансови разходи. Системата е проектирана да отговори на съвременните изисквания и да бъде надеждно решение за обработка на одит данни, предоставящи пълна функционалност и висока производителност.

# Списък с фигури

Фиг. 1 Стандартна система за одитиране на софтуер.....	6
Фиг. 2 Лого на Splunk.....	8
Фиг. 3 Диаграма на The Splunk Platform .....	9
Фиг. 4 Лого на Auditd.....	10
Фиг. 5 Auditd Архитектура.....	11
Фиг. 6 Лого на Sentry .....	14
Фиг. 7 Sentry OTEL collector .....	16
Фиг. 8 Лого на Amazon S3.....	22
Фиг. 9 Пример за качване и изтегляне на обект от Amazon S3 Bucket.....	23
Фиг. 10 Постигане на съвместимост между AWS S3 и MinIO .....	23
Фиг. 11 Лого на Google Cloud Storage.....	24
Фиг. 12 Примерна структура на Google Cloud Storage [21] .....	25
Фиг. 13 Apache Kafka диаграма .....	28
Фиг. 14 RabbitMQ диаграма .....	29
Фиг. 15 Amazon SQS диаграма .....	30
Фиг. 16 Пример за организацията на Docker контейнери .....	35
Фиг. 17 Kubernetes архитектура.....	37
Фиг. 18 Одит система .....	41
Фиг. 19 Записване на данни одит система.....	47
Фиг. 20 Изчитане на данни одит система .....	47
Фиг. 21 Swagger записване на данни - одит система.....	52
Фиг. 22 Swagger изчитане на данни - одит система.....	52
Фиг. 23 Резултати от теста със 10000 заявки.....	61

# Списък с таблици

Таблица 1 Съпоставка на The Splunk Platform, Auditd и Sentry .....	19
Таблица 2 Поддържани платформи.....	19
Таблица 3 Сравнителен анализ на Amazon S3, MinIO и GCS.....	26
Таблица 4 Сравнителен анализ на Apache Kafka, RabbitMQ и Amazon SQS .....	31
Таблица 5 Сравнителен анализ на Java, Python и Go .....	34

## Източници

- [1] Splunk, [https://www.splunk.com/en\\_us/about-us/why-splunk.html](https://www.splunk.com/en_us/about-us/why-splunk.html) - последно посетен на 11.08.2024
- [2] Splunk, <https://www.splunk.com/> - последно посетен на 11.08.2024
- [3] Splunk,  
<https://docs.splunk.com/Documentation/Splunk/8.2.0/Overview/AboutSplunkEnterprise> - последно посетен на 11.08.2024
- [4] Splunk, <https://docs.splunk.com/Documentation/SplunkCloud> - последно посетен на 11.08.2024
- [5] Auditd, <https://izyknows.medium.com/linux-auditd-for-threat-detection-d06c8b941505> - последно посетен на 13.08.2024
- [6] Auditd,  
[https://docs.redhat.com/en/documentation/red\\_hat\\_enterprise\\_linux/8/html/security\\_hardening/auditing-the-system\\_security-hardening#pre-configured-audit-rules-files-for-compliance-with-standards\\_auditing-the-system](https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/8/html/security_hardening/auditing-the-system_security-hardening#pre-configured-audit-rules-files-for-compliance-with-standards_auditing-the-system) - последно посетен на 13.08.2024
- [7] Auditd, <https://man7.org/linux/man-pages/man8/auditd.8.html> - последно посетен на 13.08.2024
- [8] Sentry, <https://www.linkedin.com/pulse/hardware-sentry-opentelemetry-collector-architecture-simply-> - последно посетен на 13.08.2024
- [9] Sentry, <https://sentry.io/security/> - последно посетен на 13.08.2024
- [10] Sentry, <https://sentry.io/branding/> - последно посетен на 13.08.2024
- [11] Apache Kafka, <https://aiven.io/blog/kafka-simply-explained> - последно посетен на 14.08.2024
- [12] Apache Kafka, <https://kafka.apache.org/documentation/> - последно посетен на 14.08.2024
- [13] Apache Kafka, <https://kafka.apache.org/intro> - последно посетен на 14.08.2024
- [14] RabbitMQ, <https://www.rabbitmq.com/> - последно посетен на 14.08.2024
- [15] RabbitMQ <https://www.rabbitmq.com/tutorials> - последно посетен на 14.08.2024
- [16] RabbitMQ <https://www.rabbitmq.com/docs> - последно посетен на 14.08.2024
- [17] Amazon S3, <https://aws.amazon.com/s3/> - последно посетен на 14.08.2024
- [18] Amazon S3, <https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingBucket.html> - последно посетен на 14.08.2024
- [19] MinIO, <https://github.com/minio/minio/blob/master/README.md> - последно посетен на 14.08.2024
- [20] MinIO, <https://min.io/> - последно посетен на 14.08.2024
- [21] What is Cloud Storage? , <https://cloud.google.com/storage/docs/introduction> - последно посетен на 14.08.2024
- [22] Amazon SQS, -  
<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDriverGuide/welcome.html> последно посетен на 14.08.2024
- [21] Amazon SQS, -  
<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDriverGuide/sqs->

[setting-up.html](#) последно посетен на 15.08.2024

[22] Amazon SQS,

<https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDriverGuide/sqs-getting-started.html> - последно посетен на 15.08.2024

[23] Amazon SQS, <https://rishabh27sharma.medium.com/amazon-sqs-a-case-study-2c8f550a2fa>

- последно посетен на 15.08.2024

[24] Kubernetes <https://kubernetes.io/> - последно посетен на 15.08.2024

[25] Kubernetes <https://kubernetes.io/docs/concepts/> - последно посетен на 15.08.2024

[26] Kubernetes , <https://www.linkedin.com/pulse/kubernetes-architecture-installation-roli-singh>

- последно посетен на 15.08.2024

[27] MinIO , <https://docs.min.io/docs/java-client-quickstart-guide.html> - последно посетен на

16.08.2024

[28] Python, <https://www.python.org/> - последно посетен на 16.08.2024

[29] What is a Container? , <https://www.docker.com/resources/what-container> - последно посетен на 16.08.2024

[30] Docker Hub, <https://www.docker.com/products/docker-hub> - последно посетен на 16.08.2024

[31] MinIO, <https://hub.docker.com/r/minio/minio> - последно посетен на 16.08.2024

[32] MinIO, <https://hub.docker.com/r/minio/mc> - последно посетен на 16.08.2024

[33] Java, [https://en.wikipedia.org/wiki/Java\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Java_(programming_language)) - последно посетен на 16.08.2024

[34] Spring Boot, <https://spring.io/projects/spring-boot> - последно посетен на 16.08.2024

[35] Python, [https://en.wikipedia.org/wiki/Python\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) - последно посетен на 16.08.2024

[36] Go, [https://en.wikipedia.org/wiki/Go\\_\(programming\\_language\)](https://en.wikipedia.org/wiki/Go_(programming_language)) - последно посетен на 16.08.2024

## Приложение

Изходният код на системата може да бъде намерен на:

<https://github.com/kaizera071/diploma-project>