Санкт-Петербургский государственный университет

Мирзазянов Глеб Романович

Выпускная квалификационная работа

Реализация платформы для сбора необработанных данных в реальном времени и мониторинга состояния производственных линий

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2018 «Программная инженерия»*

Научный руководитель:
к.ф.-м.н., доцент кафедры системного программирования Луцив Д. В.

Рецензент:
ведущий биг дата инженер ООО «ГРИД ДИНАМИКС» Гершун Т. О.

Санкт-Петербург
2022

Saint Petersburg State University

Mirzazyanov Gleb Romanovich

Bachelor's Thesis

Implementation of a platform for collecting raw data in real time and monitoring the status of production lines

Education level: bachelor

Speciality *09.03.04 «Software Engineering»*

Programme *CB.5080.2018 «Software Engineering»*

Scientific supervisor:
C.Sc., docent Luciv D. V.

Reviewer:
Senior Big Data Engineer at "Grid Dynamics" Gershun T. O.

Saint Petersburg
2022

Оглавление

1. Введение **4**

2. Постановка задачи **6**

3. Обзор **7**

3.1. Существующее приложение внутри компании 7

3.2. Amazon QuickSight 8

3.3. SAP BusinessObjects 9

3.4. Tableau 10

3.5. Вывод 10

4. Архитектура **12**

4.1. Основные компоненты 12

4.2. Навигация пользователя 13

5. Особенности реализации **15**

5.1. Взаимодействие классов 15

5.2. Сбор данных 16

5.3. Отказоустойчивость при общении с сервисами 17

5.4. Мониторинг состояния приложения 18

6. Тестирование и апробация приложения **19**

6.1. Модульное, интеграционное тестирование 19

6.2. Нагрузочное тестирование 20

6.3. Апробация приложения 22

Заключение **24**

Список литературы **25**

1. Введение

В наше время аббревиатура IoT (Internet of Things), или «интернет вещей», уже не кажется чем-то инновационным. Впервые это определение было дано в 1999 году и претерпело энное количество значительных изменений. Сегодня под интернетом вещей подразумевается концепция сети передачи данных между физическими объектами («вещами»), которые оснащены встроенными средствами и технологиями для взаимодействия друг с другом или с внешней средой [15]. Более простыми словами — это технология, которая позволяет автоматизировать процессы. Существуют различные области применения IoT: от торговли до энергетики; в данной работе будет рассматриваться сфера производства.

Современное оборудование на крупных фабриках оснащается специальными контроллерами, сенсорами, программным обеспечением для сбора и обмена информацией. Каждой произведенной детали назначается уникальный идентификатор. Применение IoT на фабриках позволяет производителям проводить подробный анализ продукции, предотвращать множество ведущих к простою и убыткам ошибок, повышать качество и максимизировать прибыль.

Тесно связан с IoT и набирающий популярность подход принятия решений, основанный на данных — так называемый *data-driven approach* [38]. Это, думается, легко объяснить развитием новых технологий, увеличением вычислительных мощностей [6] и возможностью хранения больших объемов данных. Современные приложения генерируют огромное количество информации, например: операционные метрики, навигация пользователя по странице, история транзакций etc. Полученные данные используют в машинном обучении, а бизнес аналитики получают возможность анализировать поведение пользователя и создавать новые показатели оценки продукта.

У менеджеров крупных корпораций, руководящих фабриками по производству как целых продуктов, так и отдельных деталей, возникает потребность в использовании приложения с возможностью сбора

и анализа статистики для повышения качества и уменьшения гарантийных расходов. Получать данные необходимо в реальном времени, то есть сразу после регистрации продукта, производственной линии и целой фабрики. Учитывая масштаб некоторых корпораций, фабрики, производящие детали, могут находится в разных странах или не принадлежать компании вовсе, что добавляет новые задачи по стандартизации процесса сбора, хранения и внедрения инфраструктуры корпорации.

Данная работа посвящена созданию нового приложения, помогающего автоматизировать процессы, связанные со сбором сырой информации с различных производственных линий на фабриках и визуализации полученных данных.

2. Постановка задачи

Цель работы — реализация приложения, которое позволило бы собирать данные в реальном времени, обрабатывать их и выдавать статистику о состоянии фабричных конвейерных лент. Для достижения цели были поставлены следующие задачи.

1. Обзор аналогов и существующих решений.
2. Проектирование архитектуры приложения.
3. Разработка основной функциональности приложения.
4. Покрытие приложения модульными, интеграционными и нагрузочными тестами.
5. Апробация созданного приложения.

3. Обзор

3.1. Существующее приложение внутри компании

Первым делом рассмотрим существующее решение внутри нашей компании. Для каждой фабрики устанавливаются локальные реляционные базы данных Oracle [28] с другими различными локальными сервисами. Такой подход был популярен в начале 2000 годов и имел ряд особенностей, продиктованных временем:

- Ограничение необходимой пропускной способности глобальной сети. Перемещение большого количества данных по глобальной сети было процессом небыстрым и недешевым
- Ограничение баз данных по масштабируемости, что вынуждало иметь свою базу данных для каждой фабрики

Рассмотрим проблемы такого решения:

- Отсутствие общей инфраструктуры компании на фабриках - поставщиках для унифицированного сбора данных
- Несоответствие формата данных отдельных фабрик единому стандарту
- Нарушена ассоциация между деталями и продуктом: нет возможности определить продукт по его комплектующим
- Отсутствие функциональности анализа данных по определенным временным рамкам
- Невозможно сохранение изображений
- Поддержка отдельного соединения для каждой базы данных

Исходя из вышенаписанного, следует, думается, перейти к обзору существующих аналогов, которые — теоретически — способны решить указанные проблемы.

3.2. Amazon QuickSight

Amazon QuickSight [1] — разработка, основанная на облачных вычислениях, которая предоставляет клиенту полноценную виртуальную платформу с различными инструментами и сервисами. Рассмотрим ее основные особенности:

Рис. 1: Amazon QuickSight. Источник

- Автомасштабирование до 100 тыс. пользователей при помощи технологии Spice [3]
- Извлечение информации из различных видов хранения данных: озера данных(Data lake) [36], хранилища данных (Data warehouse) [37] и баз данных: как локальных, так и развернутых в облаке
- Встроенные методы машинного обучения позволяют не только подсказывать тренд и находить шаблоны, но и указывать на аномалии
- Доступ к различным видам таблиц с интерактивной статистикой
- Фильтрация данных при помощи обычной письменной речи

- Логическое разделение на группы — авторы и читатели
- Оплата происходит только за фактическое использование

3.3. SAP BusinessObjects

SAP Business Objects [30] — программное обеспечение для бизнес-аналитики, которое предлагает комплексную отчетность, анализ и интерактивную визуализацию данных. Особенности его следующие:

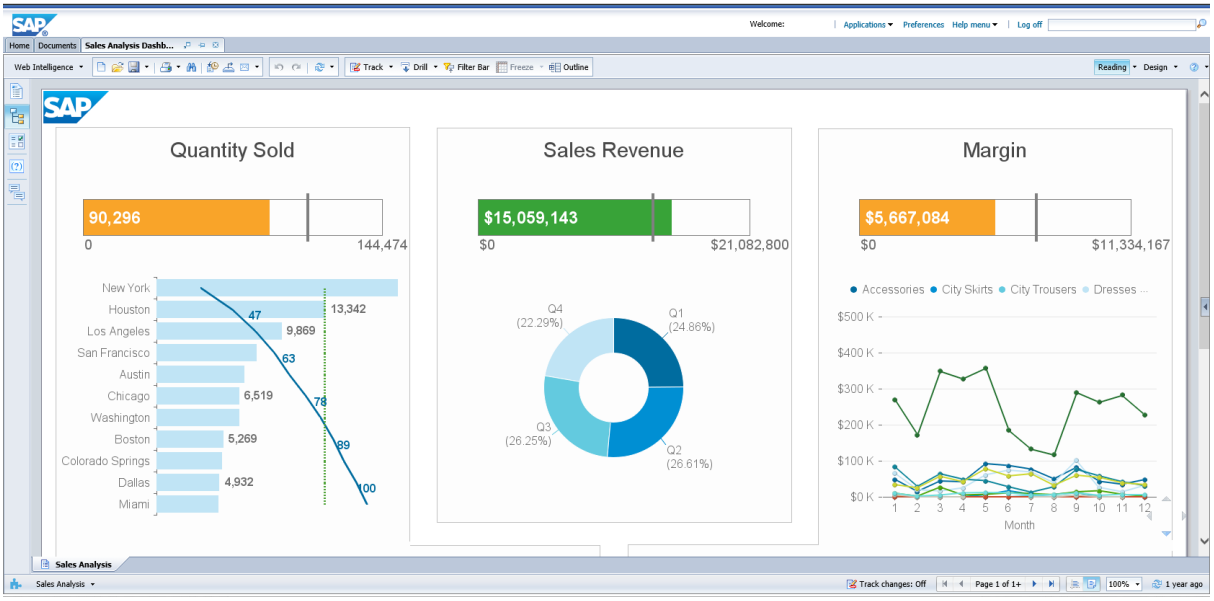


Рис. 2: Sap Business Objects. Источник

- Комбинирование данных из разных видов баз: как, например, реляционной, пространственной [40], графовой [39], так и нереляционных документо-ориентированных баз [25]
- Извлечение данных при помощи запросов из веб-сервисов, файлов, облаков, Apache Hadoop [13] и Apache Spark [14]
- Использование как локального приложения, так и веб-сервиса
- Большой порог вхождения из-за насыщенной функциональности приложения

3.4. Tableau

Tableau [7] — еще один инструмент бизнес-аналитики, который специализируется исключительно на визуализации данных:

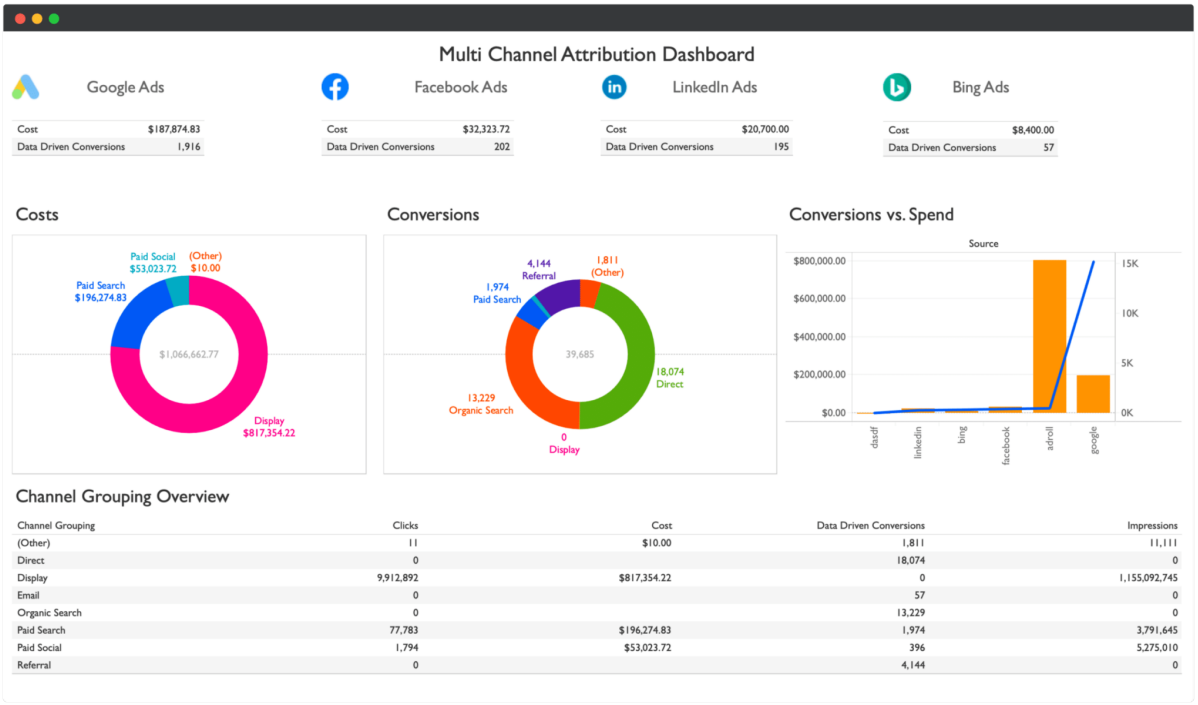


Рис. 3: Tableau пример визуализации. Источник

- Интеграция с несколькими источниками данных. Например, MS Excel [23], Oracle, MS SQL [24], Google Analytics [16] и SalesForce [31]
- Простой интуитивно понятный интерфейс
- Предоставление API [12]: программного интерфейса приложения для интеграции со сторонними приложениями

3.5. Вывод

Такие платформы-аналоги как Amazon QuickSight и SAP Business Objects являются устоявшимися цельными продуктами, которые способны решить поставленные задачи. Существует, впрочем, несколько причин, по которым ни один из них не подходит для использования:

- Внутренняя политика компании запрещает хранить данные в сторонних приложениях
- Исследование, проведенное в рамках компании, показывает, что разработка собственного приложения оказывается куда выгоднее, чем использование уже существующих
- Аналоги характеризуются избыточной функциональностью и сложным интерфейсом

Исходя из перечисленного было принято решение разработать собственный продукт для сбора и визуализации данных. В качестве инструмента визуализации было выбрано Tableau. Во-первых, оно обладает понятным интерфейсом, что сильно упрощает работу пользователя. Во-вторых, — что важнее всего — корпоративная версия этого приложения решает все проблемы с конфиденциальностью.

4. Архитектура

В связи с соглашением о неразглашении, описание некоторой части архитектуры и спецификации приходится пропустить в рамках данной выпускной квалификационной работы.

4.1. Основные компоненты

Система делится на 4 основных логических уровня.

Рис. 4: Основные компоненты

Первый — хранилище данных. Данные со стороны пользователя фабрики выгружаются на предоставленный приложением Amazon s3 бакет[2]. Они представляют собой архив с таблицами, в которых может хранится информация как о продукте в целом, так и об отдельной составляющей. Реализация s3 бакета включает в себя следующую полезную функциональность:

- автомасштабирование — возможность увеличения или уменьшения объема хранилища в зависимости от размера данных
- разделение доступа — отдельные пути для каждого пользователя
- мониторинг ошибок — наличие системного хранилища, сохраняющего ошибки
- универсальность объектов — хранение данных любого типа и размера

Заметим, что из-за конфиденциальности данных s3 бакет развернут в приватной сети компании.

Следующий уровень — моделирование данных. На этом этапе они подготавливаются к сериализации и трансформации. Также здесь содержатся различные настройки: выбор определенного продукта, создание схемы или объединение данных. Схема представляет собой метаданные, с которыми пользователь хочет работать: названия столбцов и их типы данных. Агрегирующие операции *JOIN* и *UNION* позволяют пользователю взаимодействовать объединенными данными.

Третий уровень — перенос данных в OLAP[8] базу данных Druid[4] и создание автоматической системы уведомлений и удаления неиспользованных данных. Создается пайплайн, который сканирует заданный путь для хранилища данных и при появлении новой информации переносит ее в Druid, которая быстро агрегирует данные и синхронизируется с инструментами визуализации.

На финальном уровне инструменты Tableau [7], Superset [5] визуализируют данные в виде интерактивных таблиц. Помимо этого — предоставляется интерфейс и появляется возможность отслеживать успех или неудачи текущего пайплайна.

4.2. Навигация пользователя

Теперь обратимся к рисунку 5 и рассмотрим процесс навигации пользователя. Первым делом стоит учесть, что существует несколько

ролей: пользователи данных и их владелец. Первые — это аналитики и менеджеры компании, которые должны иметь доступ к мониторингу данных.Владелец же имеет особые права, ведь ему предстоит настроить сбор и трансформацию данных. Все пользователи внутри сети компании имеют доступ к стартовой странице с краткой информацией о приложении.

Далее ознакомимся с каждой страницей по отдельности:

- Домашняя страница — обзор ранее совершенных активностей
- Список проектов — список проектов, доступных для пользователя. Один проект соответствует одному конкретному продукту и фабрике
- Детали проекта — страница для конфигурации самого проекта. Помимо ассоциации с продуктом и фабрикой включает в себя способ хранения данных, которые связанны с проектом после его удаления
- Конфигурация данных — страница с настройками схемы данных для загружаемых файлов. Она включает в себя настройку станции (производственной линии, или фабричного конвейера), на которой производят деталь, и саму деталь внутри станции
- Страница с загрузкой данных предоставляет интерфейс для их загрузки через приложение
- Внедрение группы — страница, на которой при помощи агрегирующих функций UNION и JOIN детали объединяются в группы. Это необходимо для визуализации нескольких деталей в совокупности
- Исполнение внедрения — страница, которая помогает отслеживать перенос данных в реальном времени
- Визуализация — страница для перехода на сторонние сервисы, которые предоставляют визуализацию для загруженных данных

Рис. 5: Навигация пользователя

5. Особенности реализации

Приложение разработано на языке Java одиннадцатой версии с использованием следующих основных фреймворков и библиотек:

- Spring Boot — расширение Spring, которое автоматизирует управление зависимостями и конфигурацию приложения [32]
- Spring MVC — MVC фреймворк для удобной разработки RESTful веб-сервисов [33]
- WebFlux для асинхронного и неблокирующего общения между сервисами [34].
- Hibernate [17] как реализация JPA [29] для объектно-реляционного отображения между Java классами и данными в PostgreSQL

5.1. Взаимодействие классов

Реализация обработки запросов была вынесена в отдельные классы-контроллеры. В них данные проходят валидацию, происходит десе-

риализация из JSON формата в Java объект и сериализация обратно из Java объекта в JSON; возврат кодов состояния HTTP. Классы-контроллеры делегируют исполнение бизнес-логики классам-сервисам, которые взаимодействуют с классами-репозиториями для обработки данных и с классами-клиентами для общения с другими сервисами. Классы репозиториев помимо простых CRUD-операций также реализуют интерфейс разбиения по страницами Pageable и предоставляют более сложные запросы, а также позволяют проверить целостность данных. Заметим, что вместо того, чтобы удалять данные, была выбрана мягкая стратегия деактивации сущностей. Причина этого — в важности хранимой информации. Большая роль в разработке была уделена классам-клиентам, которые взаимодействуют со сторонними сервисами. Аутентификация происходит через HTTP-заголовок по схеме *Базовая (Basic) аутентификация*. Аутентификации реализуется так просто потому, что приложение находится внутри приватной корпоративной сети. Это снимает с разработчика часть обязанностей. Для каждого класса-клиента разработан метод проверки доступности сервиса — *isHealthy*. Классы сервисов, репозиториев и клиентов используются через интерфейсы и технологию внедрения зависимостей (DI).

5.2. Сбор данных

Для начала введем несколько терминов. DAG (Directed Acyclic Graph) Airflow — это цепочка задач для запланированного запуска в виде направленного ациклического графа, которая представлена в виде кода на языке Python. Spec — это конфигурационный файл приложения, в котором хранится вся метаинформация о конкретной детали продукта: название строк, тип данных и время их хранения после удаления.

Для создания DAG с задачами переноса данных из s3-бакета пользователь с помощью интерфейса приложения создает и публикует Spec. Используя метаинформацию, предоставленную в Spec, приложение генерирует Python файл, который загружается на внутреннюю корпоративную версию GitHub [19]. Для этого были реализованы функции pull,

add, commit, push в классе *GitService*. GitHub в данном случае используется как способ передачи и хранения Airflow DAG.

Класс *AirflowClient* позволяет пользователю взаимодействовать с DAG напрямую: получать его статус в реальном времени, запускать, останавливать или удалять. Для этого приложение интегрируется с предоставленным API Airflow. Например, при использовании $GET/api/v1/dags/\{dag\_id\}/dagRuns$, можно получить статус текущего DAG по его ID.

Агрегация реализована в классе-сервисе, в котором обрабатываются объединенные колонки и происходит их валидация с помощью операции $JOIN$. После чего создается агрегирующий DAG.

5.3. Отказоустойчивость при общении с сервисами

При межсервисном взаимодействии всегда необходимо учитывать вероятность того, что один из сервисов окажется недоступен. Поэтому в классах-клиентах возможен повторный запрос в случае ошибки. Параметры количества повторений и времени между запросами выставляются при запуске приложения. Если после повторных запросов вновь возникает ошибка, рассматриваются два случая.

1. Сервер возвращается к состоянию до начала запроса.

2. Запрос выполняется с уведомлением о частичном успехе.

Одна из часто встречающихся ошибок при параллельном использовании *GitService* — неконсистентность коммитов в используемой ветке между локальным и удаленным репозиторием. Для решением проблемы можно использовать стратегию backpressure [26]. Сделать это помогает Redis Stream [22]. Такое подход позволит уменьшить количество вызовов Git push/pull, накапливая промежуточные изменения Git в пакетах.

Пакеты перемещаются по мере накопления достаточного количества результатов в течение 60 секунд или — если за последние 10 секунд не наблюдалось никаких изменений.

5.4. Мониторинг состояния приложения

Любое приложение генерирует множество полезных системных метрик, которые необходимо отслеживать. Мониторинг происходит с помощью связки Prometheus [42], Grafana [21] и специального Exporter. За время работы развернутого приложения можно привести в пример следующие метрики, которые демонстрируют успешную работоспособность приложения:

- Скорость передачи данных из s3-бакета: 10 мегабайт в секунду
- 1 миллион успешных DAG в неделю
- Более 1 терабайта данных, которые показаны в Tableau

6. Тестирование и апробация приложения

Качественное тестирование повышает качество кода и снижает количество получаемых ошибок. Это экономит время на отладке и уменьшает время разработки, поэтому важно выделять достаточное количество времени на создание тестов.

Рассмотрим основные виды тестирования, которые были использованы в нашей работе.

6.1. Модульное, интеграционное тестирование

Модульное тестирование предназначено для проверки корректности отдельных модулей программы. При такого вида тестировании используется популярный для языка Java фреймворк — JUnit5 [20]. Возможности этого фреймворка позволяют, например, используя JSON-файл, формировать набор входных данных. Большинство методов, связанных с бизнес логикой, взаимодействуют с другими сторонними сервисами, поэтому для модульных тестов используют фрейморк Mockito [35], посредством которого создаются специальные заглушки. Они позволяют получить нужный результат при вызове сторонней функции. Во время исполнения одного теста помимо ожидаемого результата проверяется и количество взаимодействий со сторонними сервисами, что значительно повышает качество тестирования.

В результате модульного тестирования было покрыто 75% строчек кода.

Во время интеграционных тестов проверяется группа модулей. Spring MVC Test фреймворк дает возможность тестировать MVC-контроллеры без развертывания специального сервера и проверять доступ к данным с использованием JDBC [11] или инструмента ORM [41], а в нашем случае — запросы Hibernate. Для корректной работы интеграционных тестов нужно развернуть базу данных. Для локального тестирования подходит библиотека Testcontainers [27] и наличие Docker [10]. Ограничение библиотеки — отсутствие возможности автоматизации при помощи внутреннего специального CI/CD инструмента. Его название опу-

щено из-за NDA. Поэтому было предложено другое решение — автоматическое развертывание базы данных PostgreSQL внутри внутреннего CI пайплайна, который исполняет интеграционные тесты.

По итогам интеграционного тестирования было покрыто 15 основных запросов с различными сценариями.

6.2. Нагрузочное тестирование

Чтобы измерить время работы основных запросов в приложении и определить границы приемлемой производительности, было решено провести нагрузочное тестирование.

Пользовательские сценарии были написаны на языке Scala с использованием специального фреймворка для нагрузочного тестирования Gatling [9]. Каждый сценарий начинается с 3-х пользователей, которые посылают на сервер параллельные запросы. Каждые 2 минуты добавляется новый пользователь. Время каждого сценария — 10 минут. Приведем основные сценарии:

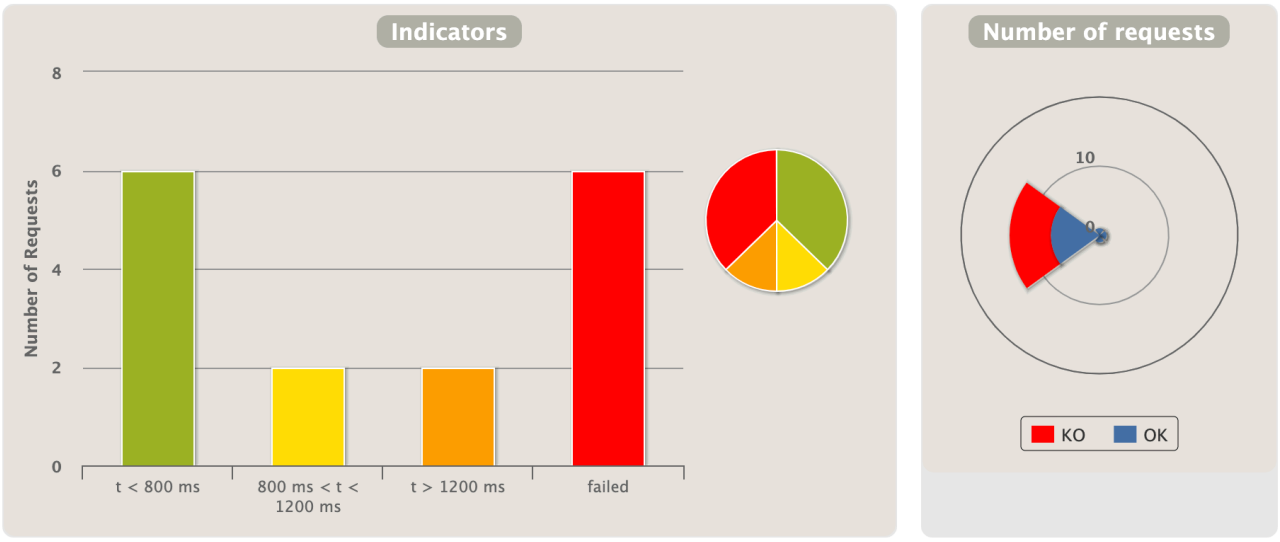
- Получение списка проектов
- Создание и удаление проекта с двумя конфигурациями
- Создание проекта с двумя конфигурациями и последующая агрегация, удаление проекта
- Удаление проекта

После выполнения формируется отчет в виде html-страницы (рисунки 6,7) с результатами.

Каждый отдельный сценарий был развернут внутри CI/CD-пайплайна для удаленного исполнения. Файл с результатами добавляется к другим развернутым на GitHub Pages [18] файлам.

В ходе тестирования было найдено узкое место при удалении проекта. Старая реализация предполагала удаление каждой Spec по отдельности и вызов (каждый раз) операций Git commit, pull, push. Помимо

этого параллельные запросы вызывали несогласованность истории коммитов. Так как запрос завершался ошибкой, это вынуждало исполнять запрос заново. Обратившись к замерам (рис 6), заметим, что среднее время удаления — 16.3 секунды, в количество неудачных запросов — 33% . Отметим, что неудачный запрос — это запрос, который завершился с ошибкой или исполнялся более 30 секунд.



STATISTICS — Expand all groups | Collapse all groups

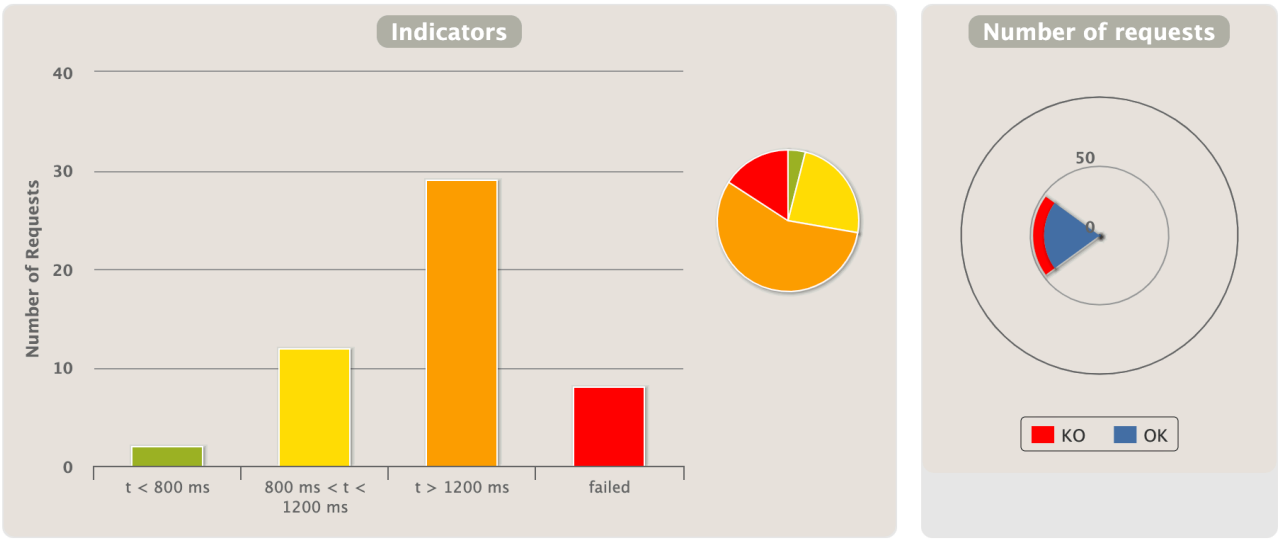
Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	16	10	6	38%	0.074	176	1383	30041	30046	30046	30046	13385	14265
Authentication	1	1	0	0%	0.005	1500	1500	1500	1500	1500	1500	1500	0
Get Projects	1	1	0	0%	0.005	176	176	176	176	176	176	176	0
Delete Project	13	7	6	46%	0.06	745	26437	30042	30046	30046	30046	16345	14272

Рис. 6: Удаление проекта до оптимизаций

Новая реализация предусматривает удаление всех Spec одним пакетом, что позволяет избавиться от повторных вызовов Git операций. При параллельных запросах используется Redis Stream и стратегия backpressure, которая была описана раннее. Обратимся к замерам после оптимизаций на рисунке 7.

Все так же существует 17% проектов, время удаления которых превышает 30 секунд. С другой стороны, среднее время удаления уменьшилось почти в 4 раза и составляет теперь 4.9 секунды.

Помимо узких мест найдены ошибки, которые возникают при одно-



STATISTICS — Expand all groups | Collapse all groups

Requests	Executions					Response Time (ms)							
	Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
Global Information	51	43	8	16%	0.214	571	1399	1930	29394	30038	30038	4650	8064
Authentication	1	1	0	0%	0.004	1493	1493	1493	1493	1493	1493	1493	0
Get Projects	1	1	0	0%	0.004	575	575	575	575	575	575	575	0
Delete Project	48	40	8	17%	0.202	895	1406	2042	29586	30038	30038	4897	8249

Рис. 7: Удаление проекта после оптимизаций

временных запросах: например, состояние гонки данных при удалении проекта. В команде сложилась практика запуска нагрузочных тестов перед большим обновлением на двух версиях приложения, что позволяет сравнить результаты разрабатываемой версии (dev) и уже существующей (prod).

6.3. Апробация приложения

Приложение развернуто и апробировано среди 30 уникальных пользователей. Пользователи — высшие менеджеры крупных производственных корпораций, которые хотят следить за своими производственными линиями. Анализ отзывов выявил следующее:

- Получены позитивные комментарии о простоте использования приложения
- Найдены пограничные случаи, которые не были учтены при те-

стировании, и при которых приложение работало некорректно. Чаще всего они связаны с созданием, редактированием и удалением некоторых сущностей

- Исследованы возможности для следующих улучшений:
  - копирование существующих объектов конфигураций приложений и возможности их копирования
  - фильтрация данных при переносе по определенным правилам
  - разработка интерфейса командной строки

Заключение

При выполнении дипломной работы были получены следующие результаты:

1. Проведен обзор решений-аналогов Sap Business Objects и Amazon QuickSight, которые не могут быть использованы для решения поставленной задачи из-за наличия конфиденциальных данных. Найдены недостатки в существующем решении: поддержка отдельного соединения для каждой фабрики и отсутствие корпоративного стандарта.

2. Спроектирована архитектура приложения. Приложение разбито на несколько подсистем, что существенно снизило сложность разработки.

3. Реализована основная функциональность приложения, которая позволяет пользователю проводить агрегацию данных и их мониторинг в виде таблиц в реальном времени.

4. Сформированы наборы тестовых данных для модульного и интеграционного тестирования. Разработано приложение для нагрузочного тестирования. С его помощью найдены и оптимизированы узкие места в приложении.

5. Приложение развернуто для 30 менеджеров производственных корпораций. Получены позитивные отзывы, исправлены ошибки и собраны пожелания о новой функциональности.

Отдельно хочу выразить благодарность специалисту компании Grid Dynamics — Черногорову Владиславу за неоценимую помощь в подготовке к выпускной квалификационной работе.

Список литературы

[1] Amazon. Amazon QuickSight Official Site. — URL: https://aws.amazon.com/ru/quicksight/.

[2] Amazon. Amazon S3. — URL: https://aws.amazon.com/s3/.

[3] Amazon. What is Spice. — URL: https://docs.aws.amazon.com/quicksight/latest/user/managing-spice-capacity.html.

[4] Apache Druid. — URL: https://druid.apache.org/.

[5] Apache Superset. — URL: https://superset.apache.org/.

[6] CPU benchmark. — URL: https://www.cpubenchmark.net/year-on-year.html.

[7] Christian Chabot Chris Stolte Andrew Beers. Tableau official website. — URL: https://www.tableau.com/.

[8] Codd E.F.; Codd S.B. Salley C.T. Providing OLAP to User-Analysts: An IT Mandate. — URL: http://www.estgv.ipv.pt/PaginasPessoais/jloureiro/ESI_AID2007_2008/fichas/codd.pdf.

[9] Corp Gatling. Gatling framework official web site. — URL: https://gatling.io/.

[10] Docker. Docker official web site. — URL: https://www.docker.com/.

[11] Documentation IBM. What is JDBC. — URL: https://www.ibm.com/docs/en/informix-servers/12.10?topic=started-what-is-jdbc.

[12] Education IBM Cloud. API definition. — URL: https://www.ibm.com/cloud/learn/api.

[13] Foundation Apache Software. Apache Hadoop Official Website. — URL: https://hadoop.apache.org/.

[14] Foundation Apache Software. Apache Spark Official Website. — URL: https://spark.apache.org/.

[15] Gartner. IoT definition. — URL: https://www.gartner.com/en/information-technology/glossary/internet-of-things.

[16] Google. Google Analytics official web site. — URL: https://analytics.google.com.

[17] Hat Red. Hibernate official web site. — URL: https://hibernate.org/.

[18] Inc. Github. Github Pages official web site. — URL: https://pages.github.com/.

[19] Inc. Github. Github official web site. — URL: https://github.com/.

[20] JUnit.org. JUnit official web site. — URL: https://junit.org/junit5/docs/current/user-guide/.

[21] LABs Grafana. Grafana LABs official website. — URL: https://grafana.com/.

[22] Labs. Redis. Redis official website. — URL: https://redis.io/.

[23] Microsoft. MS Excel official web site. — URL: https://www.microsoft.com/en-us/microsoft-365/excel.

[24] Microsoft. Microsoft SQL Server official web site. — URL: https://www.microsoft.com/en-us/sql-server/sql-server-downloads.

[25] MongoDB. What is a Document Database. — URL: https://www.mongodb.com/document-databases.

[26] Nasirloo M. Reza. RxJava Backpressure and why should you care? — URL: https://proandroiddev.com/rxjava-backpressure-and-why-you-should-care-369c5242c9e6.

[27] North Richard, other authors. Testcontainers official web site. — URL: https://www.testcontainers.org/.

[28] Oracle. Oracle Database Official Website. — URL: https://www.oracle.com/database/.

[29] Process Java Community. Jakarta Persistence 2.2. — URL: https://jakarta.ee/specifications/persistence/2.2/.

[30] SAP. SAP Business Objects Technology. — URL: https://www.sap.com/products/bi-platform.html.

[31] Salesforce. Salesforce official web site. — URL: https://www.salesforce.com/.

[32] Spring. Spring Boot official web site. — URL: https://spring.io/projects/spring-boot.

[33] Spring. Spring MVC official web site. — URL: https://docs.spring.io/spring-framework/docs/3.2.x/spring-framework-reference/html/mvc.html.

[34] Spring. Spring WebFlux official web site. — URL: https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html.

[35] Szczepan Faber Brice Dutheil et al. Mockito official web site. — URL: https://site.mockito.org/.

[36] What is Data Lake. — URL: https://aws.amazon.com/big-data/datalakes-and-analytics/what-is-a-data-lake/.

[37] What is Data Warehouse. — URL: https://www.oracle.com/database/what-is-a-data-warehouse/.

[38] What is Data-driven. — URL: https://en.wikipedia.org/wiki/Data-driven.

[39] What is Graph DB. — URL: https://en.wikipedia.org/wiki/Graph_database.

[40] What is Spatial DB. — URL: https://en.wikipedia.org/wiki/Spatial_database.

[41] Wikipedia. Object–relational mapping. — URL: https://en.wikipedia.org/wiki/Object%E2%80%93relational_mapping.

[42] dev. Prometheus. Prometheus official website. — URL: https://prometheus.io/.