**ОБЩИЕ ПОЛОЖЕНИЯ**

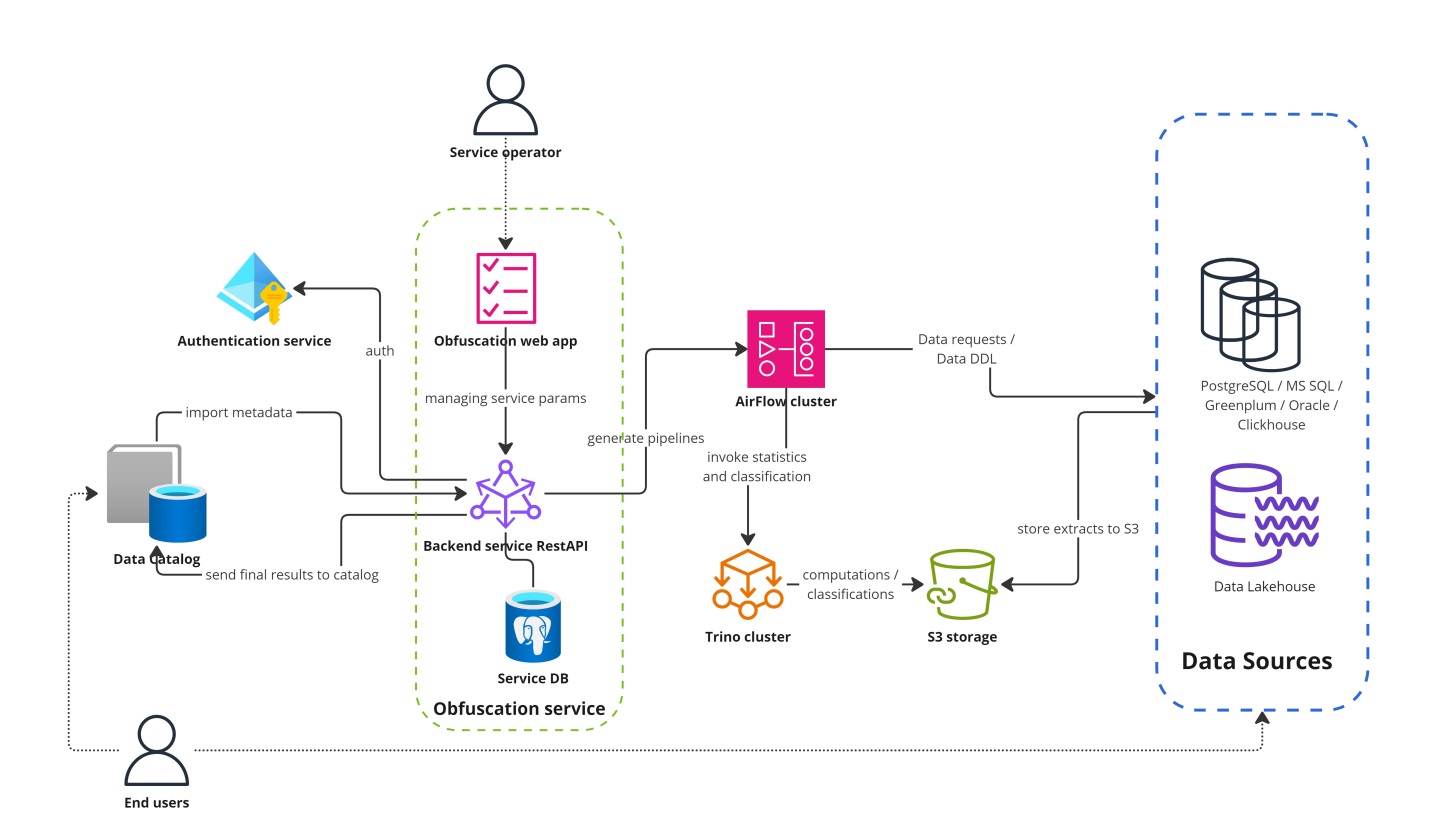
Исходя из требований, решение команды по достижению максимума результата в связи с отсутствием полного комплекта ролей – предложить архитектуру решения с максимально возможной детализацией без самой реализации решения.

В состав архитектуры заложены следующие принципы:

1. Архитектура состоит из open-source проверенных компонентов.
2. Гибкость и прозрачность процесса выполнения.
3. Пригодна для встраивания в любой технический ландшафт предприятия, универсальность к системам - источникам данных.
4. Масштабируется вычислительная мощность
5. Не требуется узкоспециализированных навыков
6. Асинхронное взаимодействие основных шагов, что повышает отказоустойчивость
7. Возможность установки компонентов on-premise.

**ОБЩАЯ АРХИТЕКТУРА РЕШЕНИЯ**

Архитектура решения в ландшафте выглядит следующим образом:



Основной принцип – мастшабируемость и пригодность к работе с большими объемами данных при соблюдении баланса скорости работы алгоритма деперсонификации и нагрузки на системы-источники.

**КОМПОНЕНТЫ АРХИТЕКТУРЫ**

**Data Sources**

Предмет исследования - различные источники данных, составляющие информационно-аналитический ландшафт предприятия.

**Authentication service**

Корпоративный сервис аутентификации пользователей (Keycloak, AD и т.д. )

**Data Catalog**

Корпоративный каталог данных (в качестве opensource версий – DataHub / Openmetadata) – помогает пользователям в поиске необходимой информации. Предполагается, что там же на объектах проставляются признаки конфиденциальности данных, а также индексируются все объекты (таблицы, представления, и тд.) информационно-аналитического ландшафта.

В случае отсутствие каталога данных возможно использовать упрощенный каталог данных (Oracle Apex, Wiki, Spreadsheets) и т.д. Главное требование – наличие интерфейса для чтения и записи структурированных метаданных.

**AirFlow cluster**

Масштабируемое отказоустойчивое решение в качестве средства оркестрации заданий, работающие в асинхронном режиме с основным сервисом. Предполагается, что это исполнитель пайплайнов по извлечению, анализу, классификации и обфускации данных.

**Trino cluster**

Решение для выполнения распределенных запросов для федеративной аналитики данных. Для снижения нагрузки на основные системы предполагается экспорт датасетов для анализа в дешевое хранение с последующей классификацией с помощью Trino.

**S3 Storage**

Объектное хранилище для данных из систем хранения с целью оптимизации производительности. В качестве альтернативы может использоваться Hadoop кластер.

**Obfuscation service**

Собственно сервис по деперсонификации данных. Ниже расписаны детально характеристики сервиса и рекомендуемый стек для имплементации

**OBFUSCATION SERVICE SPECIFICATION**

**Obfuscation web app**

Интерфейс для работы оператора сервиса.

Рекомендуемый стек - ReactJs framework

Предполагаемые роли:

1. Администратор сервиса
2. Владелец источника данных
3. Стюард данных

Основные страницы:

1. Страница аутентификации в сервисе
2. Администрирование системы (для администраторов)
   1. Добавление / удаление пользователей и их роли
   2. Настройки интеграции с каталогом данных (api endpoints на получение состава источников данных, а также на состав их метаданных), принудительная синхронизация метаданных
   3. Общие настройки по префиксам / суффиксам для обезличенных объектов
3. Страница параметров источника данных (для владельцев источника)
   1. Указание кредов подключения для сервисной учетной записи
   2. Max concurrency connections к источнику и технологические окна подключения
   3. Timeout limit
   4. Включение / исключение схем данных из обработки
   5. Просмотр логов обработки источников данных
   6. Принудительная остановка / прерывание процесса
4. Навигация по метаданным источникам (для стюардов данных)
   1. Запуск работы сервиса деперсонификации
   2. Просмотр истории и статусов (логов) работы с объектами
   3. Настройки порогов классификации для решения обфускации
   4. Просмотры результатов классификации объектов
   5. Настройки лимитов сэмплов данных
   6. Возможность установления принудительного признака ПД
   7. Функциональность удаления логов

**Obfuscation service DB**

Это база метаданных работы сервиса.

Рекомендуемый стек – PostgreSQL.

Основные таблицы сервиса:

|  |  |  |
| --- | --- | --- |
| **Table name** | **Предп. Нат. ключ** | **Описание таблицы** |
| user\_r | ldap record | пользователи системы |
| connection\_r | conn string | источники данных и их параметры (креды хранятся в зашифрованном виде) |
| connection\_limit | (connection\_r), time\_from, time\_to | ограничения подключений к источникам данных |
| users\_connection\_role | (user\_r), (connection\_r) | права пользователей по отношению к источникам данных |
| object\_r | (connection\_r), object\_path | объекты в инф. системах и их метаданные |
| column\_r | (object\_r), column\_name | поля (столбцы) в объектах инф. Систем и их метаданные |
| job | Job\_dttm, (connection\_r) | Задача на обработку источника |
| pipeline | (job), (object\_r) | история всех запущенных пайплайнов и их статусов. Один пайплайн – на один объект в задаче на обработку источника |
| pipeline\_step | (pipeline), step\_no | шаги всех пайплайнов и их статусы |
| classification\_result | (pipeline), (column\_r), pd\_class | результаты классификаций |
| pipeline\_log | (pipeline), action\_dttm | Лог работы пайплайна |
| pipeline\_step\_log | (pipeline\_step), action\_dttm | Лог работы пайплайна |
| user\_log | (user), action\_dttm | Лог пользователя |
| object\_obfuscation | (pipeline), (object) | Параметры обфускации – ddl и тд в пайплайне |
| pd\_class\_params | Pd\_class | Параметры для классов ПД |
|  |  |  |
|  |  |  |

**Backend service Rest API**

Это сервис бэкенда по управлению заданиями деперсонализации.

Рекомендуемый стек – Python FastAPI

Основные методы для реализации

* Auth section
  + POST /auth – аутентификация
  + GET /profile – получение профиля текущего пользователя
* Connection management
  + POST /connection – новое подключение
  + PATCH /connection – установка параметров подключения
  + GET /connections – получение списка подключений
  + GET /connection/{id} – информация о заданном подключении
  + DELETE /connection – удаление подключения
* User management
  + POST /role – установка роли пользователя на подключения или систему
  + DELETE /user – удаление пользователя из системы
* Metadata management
  + POST /data\_catalog\_params – установка параметров подключения к каталогу данных
  + POST /data\_catalog\_sync – синхронизация метаданных из каталога данных
  + GET /schemas – получение схем подключения
  + GET /objects – получение объектов в схеме или подключении
  + GET /columns – получение списка колонок
* Job management
  + POST /classification\_params – задание параметров классификации
  + GET /classification\_params – получение параметров классификации
  + GET /job – информация о задаче на источник
  + POST /job – создание новой задачи
  + GET /pipeline – информация о пайплайне
  + GET /pipeline\_log – лог пайплайна
  + GET /pipeline\_steps – информация о шагах пайплайна
  + GET /pipeline\_step – информация о шаге пайплайна
  + GET /pipeline\_step\_log – лог шага пайплайна
  + POST /pipeline – создание нового пайплайна
  + PATCH /pipeline – обновление параметров пайплайна
  + GET /classification\_results – получение результатов классификации объекта
  + GET /obfuscation\_info – информация об обфускации
  + DELETE /obfuscation\_log – удалить лог обфускации

**DEPLOY СЕРВИСА**

Для CI/CD сервиса подойдет любое корпоративное решение развертывания.

Приблизительный стек развертывания и мониторинга для CI/CD (но не обязательный):

* Github - репозиторий код
* Jenkins - ci/cd pipeline management
* Kubernetes – оркестратор контейнеров приложений
* Vault - хранение секретов
* Elasticsearch/Prometheus - сбор логов приложений
* Grafana - визуализация мониторинга

Переменные окружения

* ENV – текущая среда разработки
* DB\_CONN\_STRING – подключение к БД
* AIRFLOW\_URL – host Aiflow
* TRINO\_URL – host Trino
* S3\_BUCKET – URL до S3
* AUTH\_URL – host средства аутентификации

Сенситивные переменные окружения – в Vault):

* Master DB admin account – мастер логин/пароль к базе данных сервиса
* Airflow service account – логин пароль сервисной учетки к AirFlow
* Auth service account – логин пароль сервисной учетки к серверу аутентификации
* Trino service account – логин пароль сервисной учетки к Trino
* S3 service account - key/secret для доступа к S3 на запись и чтение
* Sault – соль для хэширования кредов подключения в БД

**СХЕМА РАБОТЫ СЕРВИСА ПО ДЕПЕРСОНАЛИЗАЦИИ**

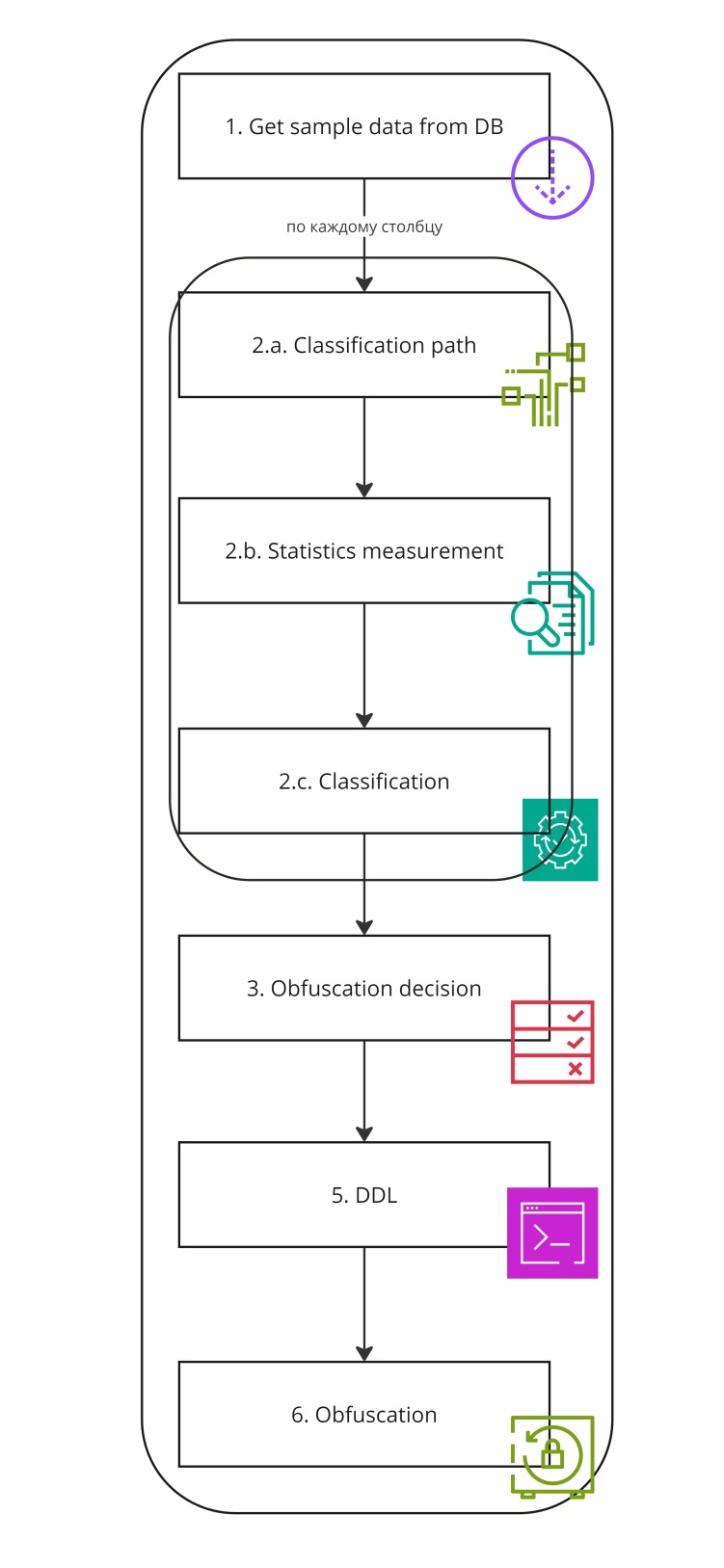
**Создание задачи на обработку источника данных**

Сервис работает по следующему общему алгоритму

1. (в случае неактуальных метаданных) Получение метаданных из каталога данных (или в отсутствие такого – напрямую из источника данных)
2. Создание нового экземпляра Job
3. Фильтрация состава объектов
4. Создание очереди пайплайнов на исполнение по кол-ву объектов источника

**Создание пайплайна**

Сервис работает по следующему алгоритму при поступлении задачи создания пайплайна (т.е. это стадии обработки одного объекта источника), при этом кол-во шагов в пайплайне не фиксировано, а определяется метаданными столбцов, типом источника. Общая схема.



1. Получение сэмпла данных из источника для объекта
2. Для каждого столбца
   1. Решение о шагах для каждого столбца согласно настройкам и метаданным
   2. Сбор статистики и предобработка
   3. Выполнение шагов, определенных в п. 2.а) для классификации
3. Принятие решение об обфускации объекта
4. Выполнение DDL на стороне источника данных – копирование структуры объекта в новой схеме и в той же схеме с префиксом/суффиксом согласно настройкам.
5. Выполнение шага(ов) по копированию данных с обезличиванием столбцов согласно типу источника данных.

**Queue\_management**

Постоянно исполняющаяся задача, которая занимается анализом текущего состояния пайплайнов и их шагов, и переводит пайплайны в следующий статус при соблюдени лимитов подключений, а также при необходимости переподнятие упавших шагов пайплайна, а также записывает все полученные логи в таблицы с логами.

**ШАГИ РАБОТЫ ПАЙПЛАЙНА - ДЕТАЛИЗАЦИЯ**

Ниже детально описываются основные шаги каждого этапа пайплайна. Для каждого этапа ведется лог из соответствующего сервиса.

**Сэмплирование**

Исходя из настроек сервиса, каким образом сэмплировать данные. Извлекается не более n строк, отсортированных либо в random порядке (желательно, при наличии опции такой сортировки у данного типа источника), либо не более n строк, отсортированных, например, по первому столбцу. Используется sqlalchemy.

Данные складываются в виде parquet файлов на хранилище S3.

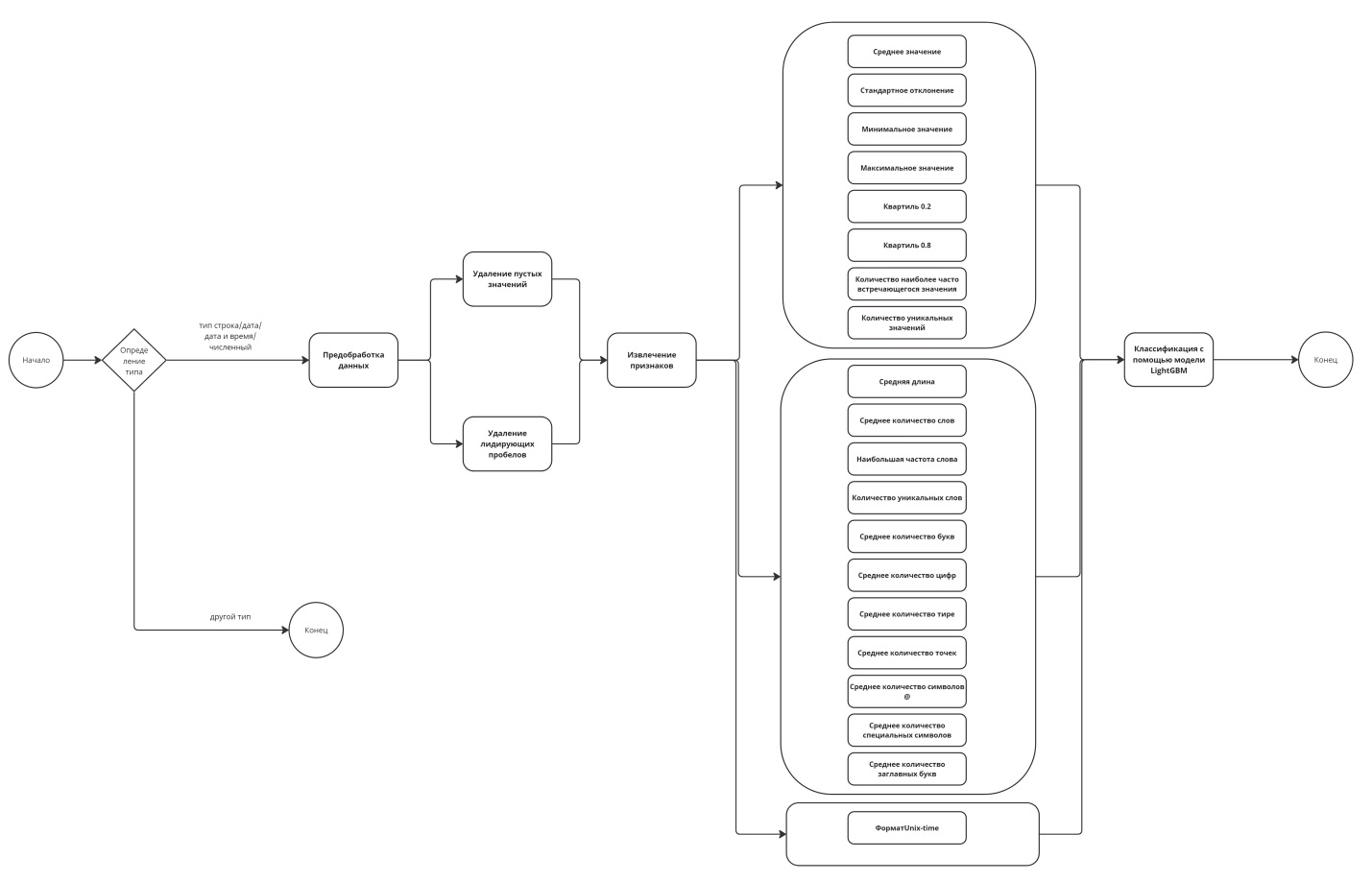
**Решение о составе шагов пайплайна**

Определение состава шагов исходя из метаданных по каждому столбцу.

Например, сразу из состава анализа столбцов исключаются принудительно заданные пользователем значения да/нет по ПД, а также типы столбцов Float, Boolean и их аналоги.

**Сбор статистики и предобработка**

Общая схема сбора статистики и предобработки показана на рисунке:

****

**Классификация**

Классификация ПД происходит из признаков, собранных на основе статистик для сэмпла столбца. Используется метод LightGBM. Библиотека lightgbm.

**Принятие решение о деперсонализации**

На данном этапе произведен анализ всех столбцов с ПД. Далее на основе настроек и результатов обработки данных принимается решение - относить ли объект к ПД или нет, и выделение ведущего класса ПД в случае наличия. Используются простые операторы сравнения. Возможно использовать библиотеку SODA Python.

**Выполнение DDL инструкций**

На данном шаге, используя тот же SqlAlchemy, выполняется экстракт метаданных объекта (DDL). И далее согласно настройкам сервиса этот ddl применяется на стороне БД с модифицированной схемой или именем. При этом внешние ключи не применяются, они остаются на исходных объектах.

**Копирование и обфускация персонализированной информации**

На данном шаге происходит копирование датасета стандартными средствами БД при использовании библиотеки SqlAlchemy и генерация синтетических данных согласно собранной статистики для столбцов ПД, согласно их ведущим классам ПД в созданных объектах.

Для различных классов используется собственный метод генерации при макс. сохранении стат. свойств распределения значений:

* date / datetime / integer: норм. распределение
  + % null значений
  + среднее, минимальное, медиана, максимальное значение, среднекв. отклонение
* text / string / varchar:
  + % null значений
  + средняя длина, медианная длина, среднекв. отклонение длины
  + частотность символов
  + в случае сбора статистики по регулярным выражением - частотность и шаблон

**ОСНОВНЫЕ ПОЛЬЗОВАТЕЛЬСКИЕ СЦЕНАРИИ РАБОТЫ В СИСТЕМЕ**

**Администратор системы**

* Первичная настройка сервиса деперсонализации данных
* Добавление пользователей системы по запросу и указание их роли

**Владелец источника данных**

* По запросу стюарда данных устанавливает настройки источника, добавляет его в каталог данных при отсутствии
* Выдает сервисную учетку для работы системы
* Выставляет технические настройки подключений к источнику для целей деперсонализации

**Стюард данных**

* Настраивает пороги и правила для целей решения о персонализации данных
* Запускает работу над источником
* Проверяет результат работы сервиса
* Проверяет, что каталог данных показывает верные метаданные
* Оповещает пользователей объектов (например, через каталог) о необходимости пользования деперсонализированными данными

**ПЛАН ПО РЕСУРСАМ ДЛЯ ДОСТИЖЕНИЯ РЕЗУЛЬТАТА**

**Разработка решения**

Общая длительность разработки при наличии необходимых ролей оценивается в 3 месяца.

Роли в разработке и оценка часов на выполнение задач в разработке согласно вышеописанной архитектуре - в проект входит:

* подготовка сред разработки для сервиса деперсонализации
* разработка сервиса (ui, frontend, backend)
* создание необходимой инфраструктуры для сервиса, trino, airflow
* установка Trino, Airflow

|  |  |  |
| --- | --- | --- |
| **Роль** | **Оценка ч-ч** | **Рекомендуемый уровень** |
| project manager | 100 | middle |
| solution architect (data engineering) | 100 | senior |
| ui / ux | 40 | middle |
| data analyst | 200 | senior |
| systems analyst | 200 | middle |
| tech lead (python) | 80 | senior |
| backend developer (python) | 300 | middle |
| frontend developer (react) | 120 | middle |
| q & a | 70 | junior |
| devops engineer | 36 | middle |
| infrastructure engineer | 40 | middle |

**ОГРАНИЧЕНИЯ АРХИТЕКТУРЫ**

Имеют место быть следующие ограничения:

1. При наличии объектов кода над объектами ПД (например, view, function, procedure и тд.) - их код не заменяется (обфусцируется), и не создается их деперсонализированная копия.
2. Код не анализирует сложные форматы столбцов типа JSON / Struct / Clob итд