Санкт-Петербургский государственный университет

Милосердова Любовь Михайловна

Выпускная квалификационная работа

Разработка сервиса для работы с porto-слоями

Уровень образования: бакалавриат

Направление *09.03.04 «Программная инженерия»*

Основная образовательная программа *СВ.5080.2019 «Программная инженерия»*

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

Рецензент:
ведущий разработчик, ООО «Яндекс.Технологии»
Тихонов Д. А.

Санкт-Петербург
2023

Saint Petersburg State University

Miloserdova Liubov

Bachelor's Thesis

Development of a service for working with porto layers

Education level: bachelor

Speciality *09.03.04 «Software Engineering»*

Programme *CB.5080.2019 «Software Engineering»*

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

Reviewer:
Senior Software Engineer at LLC «Yandex.Technologies»
Tikhonov D. A.

Saint Petersburg
2023

Оглавление

Введение

Каждому сервису, предоставляющему вычислительные ресурсы, требуется обеспечить изоляцию пользовательских пространств как друг от друга, так и от возможного негативного воздействия на операционную систему, в которой они исполняются. Пользователям этих сервисов для запуска вычислительных процессов, кроме изоляции, необходимо настроенное определённым образом окружение — установленные пакеты и библиотеки. В компании Яндекс окружения и их изоляция в основном реализуются с помощью Porto [1] и Docker [2]. Porto — система контейнеризации, разработанная внутри компании.

Porto–слой является отдельным набором директорий с файлами, по структуре повторяющий Unix–систему и служащий для запуска задач в предсказуемом окружении. Porto–слои — де–факто стандарт внутри компании Яндекс для запуска задач в предсказуемом окружении. Они используются в вычислительных сервисах различного назначения, например, в YTsaurus [3] — в сервисе распределённого хранения больших объёмов данных и распределённых MapReduce [4] вычислений. Кроме того, porto–слои используются для организации непрерывной интеграции (continuous integration, CI) и непрерывной доставки (continuous delivery, CD) кода во внутренней системе контроля версий.

Однако работа со слоями далека от того, чтобы быть удобной и прозрачной. Во многих сервисах можно лишь указать идентификатор слоя, который нужно использовать для запуска задачи, но нельзя посмотреть никакую информацию о слое и тем более найти подходящий слой. Лишь в сервисе Nirvana [5] (неспециализированная облачная платформа для визуализации, управления и запуска вычислительных процессов) уже реализован просмотр списка слоёв с различными фильтрами и сортировками, просмотр метаинформации слоя и сборка нового слоя.

В рамках данной работы предлагается вынести эту функциональность в отдельный сервис — реестр porto–слоёв внутри компании, а также реализовать настройку прав доступа и просмотр истории версий слоя и их списаний. Ожидается, что новый сервис повысит не только

удобство работы с porto–слоями, но и переиспользуемость слоёв внутри компании. Это в свою очередь повысит эффективность выполнения задач в сервисах YTsaurus и Nirvana за счёт того, что популярные слои закэшируются на узлах вычислительных кластеров.

1. Обзор предметной области

1.1. Porto: предпосылки создания

Porto — система контейнеризации, разработанная внутри Яндекса. Разработка Porto началась в 2014 году, на заре появления Docker. Одной из причин необходимости разработки собственной технологии была проблема, связанная с обратной совместимостью Docker. После выпуска очередного релиза обновление на новую версию занимало около часа на одном сервере или требовало удаления всех образов перед обновлением, а затем их повторной загрузки. Эта проблема была специфична для компании, так как большинство пользователей Docker использовали его в AWS [6], где могли легко переконфигурировать сервер. Однако в случае собственных вычислительных кластеров требовалось переконфигурировать десятки тысяч серверов, что представлялось невозможным. По этой причине в Porto отдельное внимание было уделено стабильности: обновлению без перезапуска контейнеров, сохранению состояния контейнеров в случае падения.

Собственная разработка позволила заказывать любую необходимую функциональность, а также обращаться в поддержку, если возникают проблемы. При использовании Docker возникали сложности, связанные с уникальностью инфраструктуры компании. Для решения этих проблем приходилось разбираться в исходном коде, создавать расширения (patch) и обсуждать их с разработчиками Docker, цели которых могли отличаться от потребностей компании Яндекс. Также собственная разработка позволила поддерживать расширения ядра Linux, созданные внутри компании.

1.2. Porto–слои

Porto–слой — отдельный набор директорий с файлами, по структуре повторяющий Unix–систему. Обычно слой представлен архивом с образом файловой системы или разницей с подготовленным окружением для запуска.

Некоторыми системами, в том числе системой Nirvana, поддерживаются базовые слои для Ubuntu (Bionic, Trusty, Xenial, Focal, Jammy). Базовый слой — самодостаточный образ, в котором лежит основная часть окружения (/bin/bash, libc6 и т. д.) соответствующей версии операционной системы Ubuntu, а также некоторые пакеты, разработанные внутри компании (например, внутренняя модификация archive–keyring). Пользователи могут создавать дельта–слои, которые содержат дополнительные библиотеки или бинарные артефакты. Последовательность слоёв формирует образ корневой файловой системы. Слои в последовательности перечисляются от самых верхних к нижним. Последний слой в этой последовательности должен быть базовым, остальные — дельта–слоями.

1.3. Системы Nirvana и YTsaurus

Nirvana — это неспециализированная облачная платформа для визуализации, управления и запуска произвольных вычислительных процессов. Nirvana предоставляет вычислительные ресурсы и упрощает изменение конфигурации процессов, позволяет переиспользовать процессы и их компоненты, созданные и поддерживаемые другими пользователями. Согласно закрытой статистике, в неделю сервисом пользуется около 2000 пользователей, а также в нём запускается около 20 миллионов процессов.

Процесс в Nirvana представляет собой граф. Вершинами графа являются операции — программы с заранее обозначенным набором параметров вычисления, входных и выходных данных, выполняющие определённые автором действия. Ребра графа задают последовательность исполнения операций. Для выполнения операции может быть необходимо определённое окружение, например, должны быть установлены некоторые пакеты. Для этого у операций есть опция job–layer, значение опции — список идентификаторов porto–слоёв.

Запуск Nirvana–операций происходит в YTsaurus — в сервисе распределённых MapReduce вычислений. YTsaurus — высоконагруженный

сервис, его крупнейший вычислительный кластер способен хранить и обрабатывать данные 10 000 пользователей, обрабатывать данные на более чем 300 000 Hyper Threading [7] ядрах, а также решать задачи машинного обучения на более чем 2000 графических процессорах (GPU).

YTsaurus — один из тех сервисов, в котором все взаимодействие со слоями сводится к указанию идентификаторов слоя в конфигурации запусков вычислительных задач. В нём нет возможности собрать или найти слой. За счёт создания отдельного сервиса для работы с porto–слоями, а конкретно возможности поиска по слоям вне системы Nirvana, ожидается повышение эффективности выполнения задач в системах YTsaurus и Nirvana, так как популярные слои закешируются в tmpfs [8] узлов вычислительных кластеров. Механизм выбора оптимального набора porto–слоёв для кэширования их в tmpfs был реализован ранее в рамках курсовой работы.

1.4. Ранее реализованная функциональность

На момент начала работы был доступен просмотр метаинформации о слое: название, описание, автор, информация о дистрибутиве Linux, дата сборки, размер, индикатор нахождения в tmpfs вычислительных кластеров YTsaurus, количество использований слоя в сервисе Nirvana. Кроме того, была реализована возможность сборки porto–слоёв через интерфейс.

В предыдущей работе была реализована детекция Debian и Python пакетов, установленных в слое, и механизм выбора оптимального набора porto–слоёв для кэширования их в tmpfs. Также был реализован поиск конкретного слоя по заданным пользователем пакетам, была поддержана фильтрация результата (например, по версии операционной системы или по автору) и сортировка результата (например, по размеру или по дате создания).

2. Постановка задачи

На основе проведенного обзора предметной области была поставлена цель разработать сервис для улучшения взаимодействия с porto-слоями пользователей различных вычислительных платформ, используемых в компании. Для достижения цели были поставлены следующие задачи:

- собрать требования к сервису;
- уточнить архитектуру сервиса;
- реализовать требуемую функциональность;
- покрыть написанный код тестами и выполнить апробацию сервиса.

3. Требования

В процессе сбора требований были проведены интервью с представителями команд, активно пользовавшихся сервисом, а также написан пост во внутрикорпоративной социальной сети, в комментариях к которому сотрудники компании могли поделиться тем, чего им не хватает для использования сервиса.

3.1. Потенциальные пользователи

Было выявлено, что porto–слои, кроме сервиса Nirvana, активно используются в следующих сервисах.

- Sandbox — распределённая система выполнения задач общего назначения, включающая в себя систему хранения ресурсов и возможность создания релизов; в основном система используется для сборки и тестирования компонент поиска.

- Yandex Planner — система для выделения ресурсов во внутреннем облаке, её основными задачами является агрегация информации об имеющихся физических ресурсах (память, CPU, диски, сеть), выделение (scheduling) ресурсов для экземпляров сервисов, хранение и предоставление метаданных для задачи service discovery.

- Arcadia CI — система для организации непрерывной интеграции (CI) и непрерывной доставки (CD) кода в едином репозитории с исходными кодами Arcadia.

- YTsaurus — система распределенного хранения и обработки больших объемов данных с поддержкой MapReduce, распределенной файловой системой и NoSQL key–value базой данных.

3.2. Функциональные требования

В ходе опроса пользователей были выявлены следующие пожелания к функциональности сервиса.

Первым из них было улучшение поиска по слоям, а именно поиск слоёв с частичным соответствием по пакетам (найдено M пакетов из N указанных). Такой поиск позволил бы подбирать родительский слой в случае, если по указанным пакетам слои не были найдены или найденные слои не подходят по каким-либо параметрам (например, слишком много весят). Также пользователи просили добавить фильтр нахождения слоя в tmpfs и возможность указания ограничений на версии пакетов, а в случае Python пакетов и на версию Python.

Следующим пожеланием была возможность просматривать историю версий слоя и списывать версии. История версий нужна в первую очередь для обеспечения наблюдаемости изменений среди версий. Ранее создание новой версии слоя было эквивалентно созданию нового слоя. Таким образом, авторы слоёв не имели удобного инструмента для уведомления пользователей о появлении новой версии, рекомендуемой к использованию. Также пользователям хотелось иметь возможность в конфигурации своих процессов указывать необходимость использования рекомендуемой версии слоя (Main–версии) вместо указания конкретной версии. Списания необходимы для того, чтобы уведомить пользователя, что данную версию слоя не рекомендуется использовать. Например, в пакете, установленном в слое, была обнаружена уязвимость и в новой версии слоя была установлена безопасная версия этого пакета.

Также хотелось бы добавить возможность настройки прав доступа к слою, чтобы создавать новые версии слоя и управлять ими могла определённая группа лиц. Слои практически всегда нужны для командной работы, поэтому и управление ими должно быть доступно всей команде, а не отдельному человеку.

Таким образом, основные случаи использования сервиса можно увидеть на рисунке 1.

3.3. Нефункциональные требования

От сотрудников, которые пока не пользовались сервисом, но он мог бы быть им полезен, мы узнали, что их пугает многообразие функци-

ональности сервиса Nirvana. Таким образом, было выделено нефункциональное требование – вынесение сервиса на отдельный домен, чтобы скрыть от пользователей ненужную им функциональность. Забегая вперёд, хотелось бы отметить, что сервис был вынесен на отдельный домен, но в этой работе мы не будем на этом останавливаться подробнее, так как на новый домен переехал фронтенд сервиса, в то время как данная работа фокусируется на бэкенд–части.

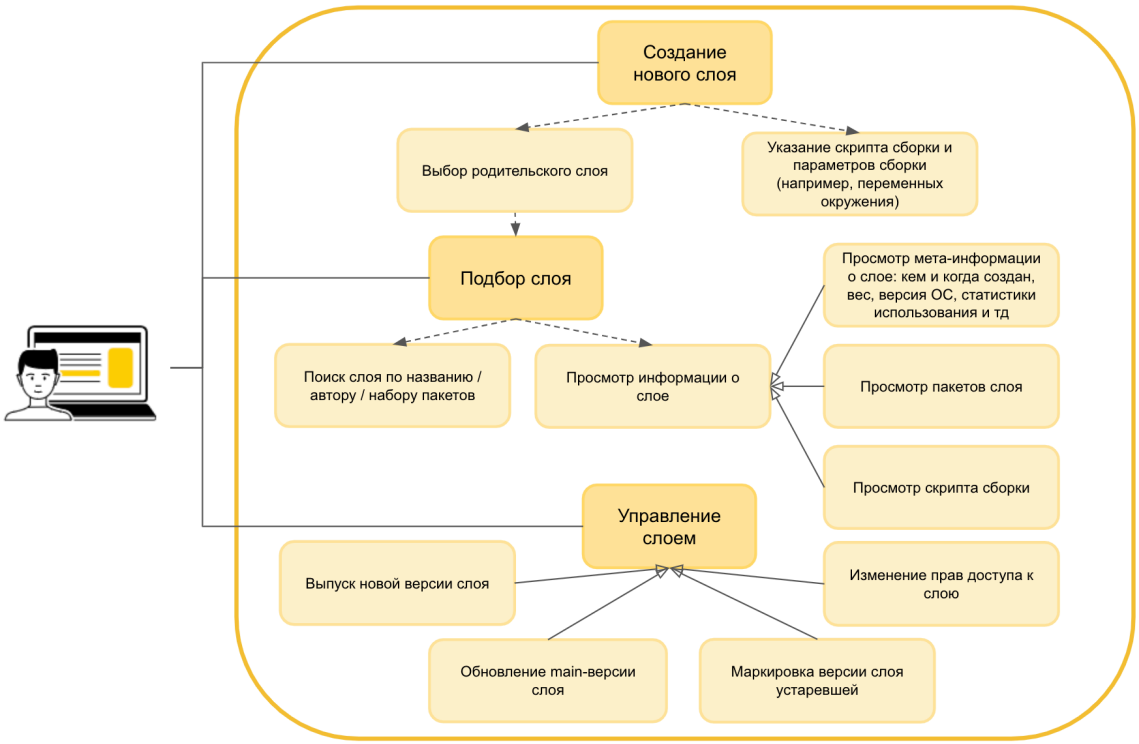

Рис. 1: Диаграмма случаев использования сервиса

4. Архитектура

Рассмотрим общую архитектуру разрабатываемого сервиса — Layer Service. Он состоит из следующих основных компонент.

- Background Workers — классы, выполняющие некоторые (возможно, периодические) фоновые задачи, например, обновление tmpfs вычислительных кластеров или же пересчет метаинформации о слое.
- Data management — хранение и обновление модельных сущностей (слои, пакеты, сборка, скрипты сборки и другие).
- Frontend API — API, используемый фронтендом; к этому API не предъявляется жестких требований поддержания обратной совместимости, любые изменения согласуются с командой фронтенда.
- API — обратно совместимый API для использования другими сервисами.

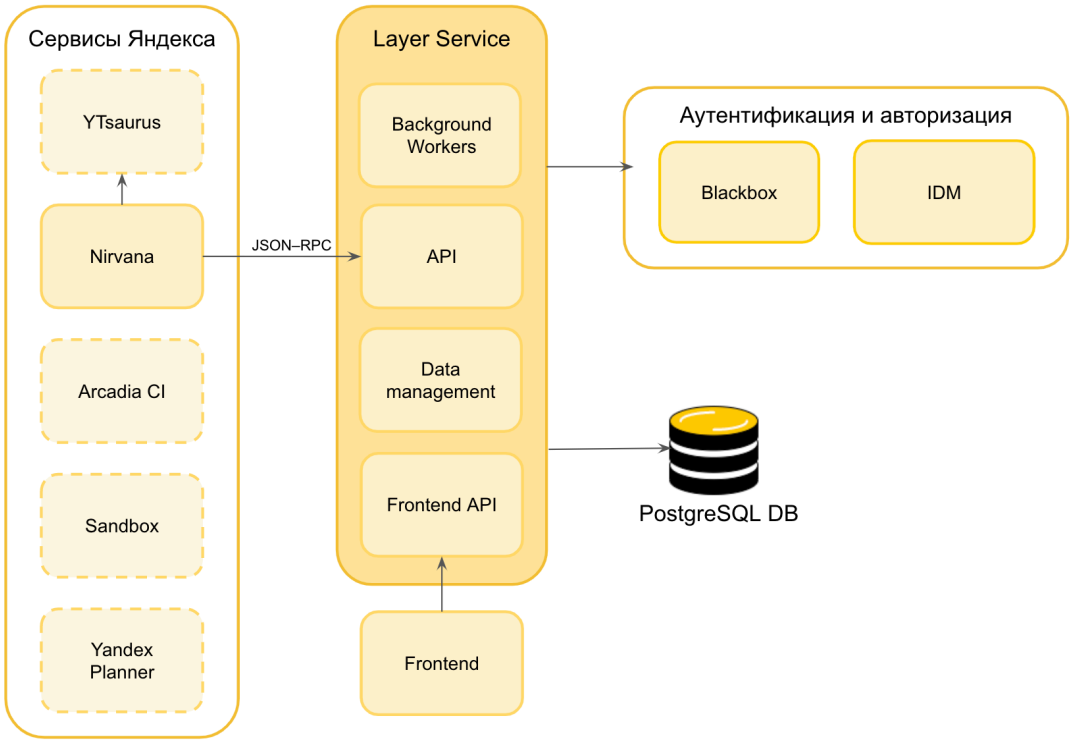

Рис. 2: Архитектура сервиса и его связь с окружающими системами

Для API и Frontend API используется протокол JSON-RPC [9] поверх HTTP [10]. JSON–RPC — это один из протоколов удаленного вызова процедур (remote procedure call, RPC), который использует формат данных JSON для передачи информации между клиентом и сервером.

Сервис интегрирован с IDM — сервисом управления доступами пользователей (позволяет предоставлять и отзывать роли) и Blackbox — внутренний HTTP-сервис, являющийся частью системы авторизации Яндекса (с его помощью происходит аутентификация пользователей по OAUTH–токенам).

На данный момент API сервиса пользуется только Nirvana, работающая поверх YTsaurus, но в ближайшем полугодии мы надеемся на дальнейшую интеграцию как напрямую с YTsaurus, так и с такими сервисами как Arcadia CI, Sandbox и Yandex Planner.

Разработка проходила на языке Java восьмой версии с использованием следующих основных технологий (фреймворков) и библиотек:

- Hibernate [11] — реализация JPA [12] для объектно–реляционного отображения между Java классами и данными в PostgreSQL [13];
- Spring Boot [14] — расширение Spring, которое автоматизирует управление зависимостями и конфигурацию приложения;
- Maven [15] — фреймворк для управления и сборки проектов.

Для автоматической генерации кода в проекте используются инструменты Eclipse Modeling Tools [16] и плагин Acceleo [17]. Ecore — это язык моделирования, являющийся частью Eclipse Modeling Framework (EMF) [18] и используемый для описания моделей. Ecore–модель определяет структуру модели, включая классы и их атрибуты, а также типы данных и различные ограничения (например, необходимость определенному атрибуту быть ненулевым). По ecore–модели генерируется код на языке Java, например, код сериализации и десериализации данных в JSON–формат или шаблон класса JSON–RPC сервиса для API метода. Кроме того, по этой модели генерируются JSON–схемы запросов и ответов API методов.

5. Особенности реализации

5.1. Поиск по porto–слоям

Для поиска по пакетам были реализованны API методы для подсказки существующих в системе названий пакетов и их версий. Также модифицирован API метод самого поиска:

- поддержан фильтр закешированности слоя;
- реализована возможность указать ограничение на версию пакета, а также на версию Python: конкретная версия Python (например, Python 3.6), Python 2 или Python 3.

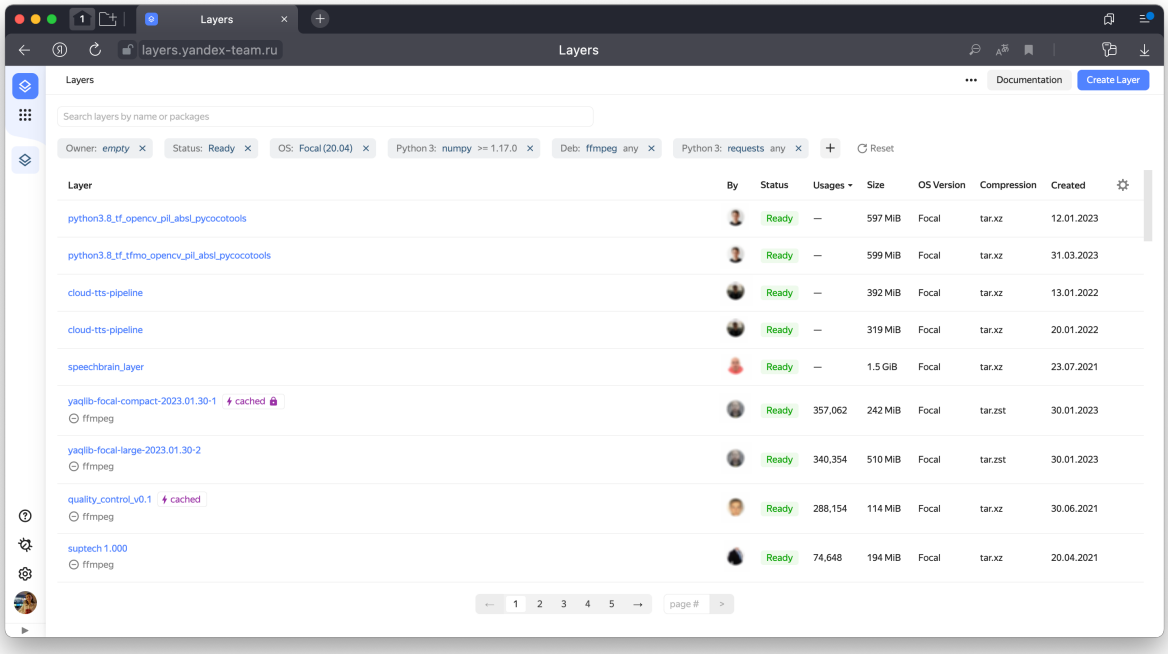

Рис. 3: Интерфейс поиска по porto–слоям

Как можно заметить, в выдаче отображаются слои с частичным соответствием и явно указаны ненайденные пакеты. Благодаря этому, если подходящий слой не был найден, пользователь может выбрать базовый слой, поверх которого он соберет дельта–слой с недостающими

пакетами. Например, первые пять слоёв в выдаче на рисунке 3 за последний месяц не использовались ни разу. Шестой же слой используется другими пользователями более активно и находится в кеше вычислительного кластера, а значит, при запуске вычислительного процесса не придётся тратить время на скачивание слоя и его распаковку. Таким образом, лучше выбрать шестой слой и создать поверх него дельта–слой с установкой недостающего пакета ffmpeg.

5.2. Версионирование porto–слоёв

В ходе реализации версионирования porto–слоёв был проведен рефакторинг схемы данных, обнаружен мёртвый код. Не актуальные более поля таблиц базы данных и код были удалены.

Была выделена новая сущность ”Layer Group”, имеющая такие атрибуты, как имя, описание, тип, автор, идентификатор основной версии слоя. К сущности ”Layer” были добавлены атрибуты версии и идентификатора группы, удалены атрибуты типа и автора. Было наложено ограничение на уникальность среди слоёв пары идентификатор группы и версия.

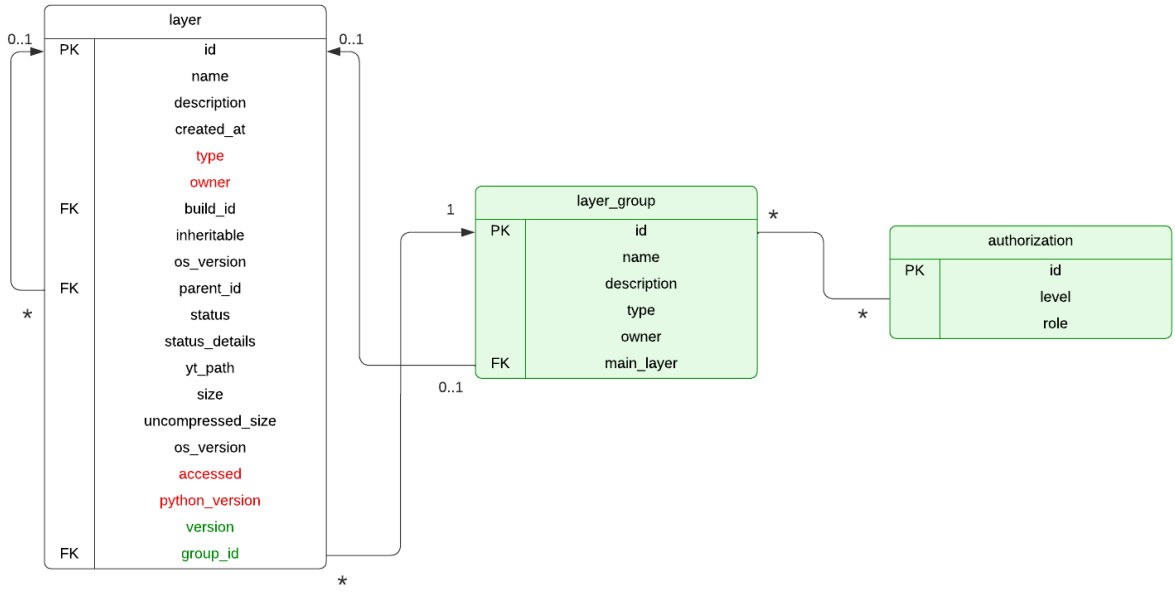

Рис. 4: Фрагмент ER диаграммы

Для дальнейшей работы с группами были реализованы следующие основные API методы:

- создание новой версии слоя;
- изменение основной версии слоя;
- депрекация версии;
- перемещение версии в другой слой;
- изменение названия слоя, его описания и прав доступа к нему;
- получение списка версий слоя.

5.3. Настройка прав доступа к porto-слоям

Для реализации настройки прав доступа к слоям была создана сущность авторизации с атрибутами уровня доступа (READ, MODIFY) и именами ролей. Проведена интеграция с IDM — сервисом управления доступами пользователей (позволяет предоставлять и отзывать роли). Модифицировано порядка 30 API методов для проверки прав доступа на то или иное действие.

5.4. Сложности, возникшие в процессе реализации

Несмотря на то, что изменения схемы данных, описанные выше, могут показаться довольно незначительными, на практике необходимо уделить должное внимание возможным негативным последствиям и постараться их избежать. Неправильно проведенная миграция базы данных может привести к блокировке, что может остановить работу всего сервиса. Более того, если обратная совместимость не будет обеспечена, пользователи могут столкнуться с ошибками и неожиданным поведением сервиса, а при обнаружении критических ошибок можно столкнуться с невозможностью откатить релиз и быстро восстановить корректное поведение сервиса.

5.4.1. Миграции базы данных без долгих блокировок

Некоторые операции с базой данных блокируют таблицы на чтение или на запись, то есть на время их выполнения сервис может быть недоступен или доступен лишь для чтения. В случае если база данных большая, время блокировки может быть критичным. Упомянем способы избежания данных проблем для PostgreSQL баз данных.

При создании индекса блокируются операции записи на все время создания индекса. Однако индекс можно создавать с ключевым словом `CONCURRENTLY` [19]. Такая операция может выполняться дольше, однако во время ее выполнения таблица будет доступна и на чтение, и на запись.

При добавлении ограничений на таблицу происходит сканирование таблицы, чтобы убедиться, что все существующие строки в таблице удовлетворяют новому ограничению. Сканирование большой таблицы для проверки ограничения может занять много времени, на это время другие обновления таблицы блокируются. При использовании ключевого слова `NOT VALID` сканирования таблицы не происходит, только новые строки проверяются на соответствие ограничению. С помощью команды `VALIDATE CONSTRAINT` можно проверить все существующие строки. Эта команда не создаёт блокировок ни на чтение, ни на запись [20].

При добавлении новой колонки со значением по умолчанию произойдет перезапись всей таблицы и всех её индексов, на это время она будет заблокирована и на чтение, и на запись [20]. Чтобы этого избежать нужно сначала в одной транзакции отдельным запросом добавить новую колонку без значения по умолчанию и отдельным запросом добавить значение по умолчанию. В этом случае блокировки не будет, так как добавление значения по умолчанию не приводит к изменению существующих данных в таблице. После закрытия описанной транзакции необходимо обновить все данные — делать это необходимо группами по несколько тысяч строк, чтобы избежать долгой блокировки. Далее, если необходимо, чтобы колонка имела ограничение `NOT NULL`, нужно

выполнить соответствующую команду.

5.4.2. Обеспечение обратной совместимости

Поскольку невозможно одновременно сделать релиз и фронтенда, и бекенда, необходимо поддержать обратную совместимость и разделить релиз новой версии на несколько этапов. На первом этапе создается релиз бекенда, сохраняющий предыдущее поведение сервиса при добавлении новых функций и атрибутов сущностей. На данном этапе необходимо дублировать атрибуты, перемещенные из одной сущности в другую, а также удаленные атрибуты. Важно не забывать об обеспечении консистентности данных. На следующем этапе происходит релиз фронтенда, в котором происходит переключение со старой схемы данных на новую. Если на данном этапе не обнаружено критических ошибок и в откат релиза не требуется, то можно делать релиз бекенда, в котором уже отсутствует дублирование атрибутов. Если после этого этапа также не возникло критических ошибок, можно проводить финальную миграцию базы данных с удалением полей таблиц.

6. Тестирование и апробация

6.1. Тестирование

Важной частью процесса разработки любого программного продукта является тестирование, которое помогает выявить ошибки и проблемы в работе программы. Рассмотрим основные виды тестирования, которые были использованы в данной работе.

Библиотека JUnit [21] является одной из наиболее популярных библиотек для тестирования Java–приложений. Она предоставляет удобный интерфейс для написания модульных тестов, которые позволяют проверять работу отдельных компонентов приложения. Например, можно проверить, что метод возвращает ожидаемое значение или что выбрасывается исключение в нужной ситуации. С использованием данной библиотеки были написаны модульные тесты, в частности, для всех методов классов–репозиториев.

Библиотека Mockito [22], в свою очередь, является библиотекой для создания и использования mock-объектов. Mock–объекты позволяют заменять реальные объекты в тестах на объекты, которые имитируют их поведение, но не имеют реальной реализации. Таким образом, mock–объекты помогают создать тестовые сценарии, которые не зависят от внешних ресурсов, таких как базы данных или внешние сервисы. С использованием данной библиотеки были написаны тесты для некоторых API методов для проверки корректности обеспечения прав доступа к тем или иным действиям. В целях экономии времени было проведено ручное тестирование некоторых API методов, которые состоят из проверки права доступа и вызова методов класса–репозитория (на которые написаны отдельные тесты). Для этого использовался инструмент для тестирования API — Postman [23], позволяющий отправлять HTTP–запросы и получать ответы от сервера. Таким образом, удалось значительно сократить время, затрачиваемое на написание тестов, а также проверить корректность схемы возвращаемых данных.

Модульные тесты позволяют выявлять проблемы на ранних этапах

разработки и облегчают процесс отладки кода. Однако важно помнить, что модульные тесты не являются заменой для интеграционного тестирования, которое проверяет, как отдельные компоненты системы работают вместе, обнаруживая ошибки, которые могут возникнуть при взаимодействии между ними. Поэтому также были написаны интеграционные тесты, которые использовали тестовую базу данных, которая была идентична рабочей (production) базе данных, но не содержала реальных данных. Были протестированы сценарии взаимодействия с другими сервисами.

6.2. Апробация

Приложение развернуто и апробировано среди более чем 450 уникальных пользователей. В ходе выпуска релиза, а конкретно проведения миграции базы данных, удалось избежать недоступности сервиса. Проблем с работой пользовательских бизнес–процессов не было обнаружено.

О запуске написан пост во внутрикорпоративной социальной сети, а также проведено выступление на встрече команды внутренней инфраструктуры (присутствовало более 200 слушателей). Получена следующая обратная связь.

- Пользователи отметили удобство поиска по слоям и наглядность страницы с метаинформацией о слое.
- Пользователи выразили заинтересованность в интеграции с другими сервисами.
- Пользователи, которые ранее не пользовались сервисом, столкнулись с некоторыми проблемами, связанными в основном с документацией сервиса, например, для некоторых опций сборки слоя был неясен правильный формат их ввода. Было заведено порядка 5 задач, которые впоследствии были решены.

Заключение

В ходе данной работы были получены следующие результаты.

- Выявлены потенциальные пользователи сервиса, проведен сбор их пожеланий к функциональности. Сформулированы функциональные и нефункциональные требования.

- Уточнена архитектура сервиса. Проведена интеграция с сервисом управления доступами пользователей. Сервис реализован на языке Java с использованием фреймворков Spring Boot, Maven и библиотеки Hibernate.

- Улучшен поиск по porto–слоям и добавлена следующая функциональность: версионирование porto-слоев и возможность настройки прав доступа к porto-слоям.

- Написаны unit–, mock– и интеграционные тесты для проверки корректности работы сервиса. В ходе выпуска релиза, а конкретно проведения миграции базы данных, удалось избежать недоступности сервиса. Проведена апробация сервиса пользователями, никакой из бизнес-процессов не был сломан. Пользователи отметили повышение удобства работы с porto–слоями.

Список литературы

[1] Yandex. Yet another Linux container management system. — Access mode: https://github.com/yandex/porto (online; accessed: 2022-12-20).

[2] Docker Inc. Docker documentation. — Access mode: https://docs.docker.com/ (online; accessed: 2022-12-20).

[3] YTsaurus. Documentation. — Access mode: https://ytsaurus.tech/docs/en/ (online; accessed: 2022-12-20).

[4] Wikipedia. MapReduce. — Access mode: https://en.wikipedia.org/wiki/MapReduce (online; accessed: 2022-12-20).

[5] Яндекс. Познаём Нирвану — универсальную вычислительную платформу Яндекса. — 2018. — Access mode: https://habr.com/ru/company/yandex/blog/351016/ (online; accessed: 2022-12-20).

[6] AWS. Cloud Computing Services — Amazon Web Services. — Access mode: https://aws.amazon.com/ (online; accessed: 2022-12-20).

[7] Intel. What Is Hyper-Threading? — Access mode: https://www.intel.co.uk/content/www/uk/en/gaming/resources/hyper-threading.html (online; accessed: 2022-12-20).

[8] The Kernel Development Community. The Linux Kernel documentation. — Access mode: https://www.kernel.org/doc/html/latest/filesystems/tmpfs.html (online; accessed: 2022-12-20).

[9] JSON-RPC Working Group. JSON–RPC 2.0. — Access mode: https://www.jsonrpc.org/specification (online; accessed: 2023-05-01).

[10] M. Thomson, C. Benfield. HTTP/2. — Access mode: https://datatracker.ietf.org/doc/html/rfc9113 (online; accessed: 2023-05-01).

[11] Red Hat. Hibernate. — Access mode: https://hibernate.org (online; accessed: 2022-12-20).

[12] Jakarta Persistence Team. Jakarta Persistence 2.2. — Access mode: https://jakarta.ee/specifications/persistence/2.2/ (online; accessed: 2022-12-20).

[13] The PostgreSQL Global Development Group. PostgreSQL: About. — Access mode: https://www.postgresql.org/about/ (online; accessed: 2023-05-01).

[14] Spring. Spring Boot official web site. — Access mode: https://spring.io/projects/spring-boot (online; accessed: 2022-12-20).

[15] Apache. Maven. — Access mode: https://maven.apache.org/ (online; accessed: 2022-12-20).

[16] Eclipse Foundation. Eclipse Modeling Tools. — Access mode: https://www.eclipse.org/downloads/packages/release/2023-03/r/eclipse-modeling-tools (online; accessed: 2023-05-01).

[17] Eclipse Foundation. Acceleo Documentation. — Access mode: https://help.eclipse.org/latest/index.jsp?topic=/org.eclipse.acceleo.doc/pages/index.html (online; accessed: 2023-05-01).

[18] Eclipse Foundation. Eclipse Modeling Framework (EMF). — Access mode: https://www.eclipse.org/modeling/emf/ (online; accessed: 2023-05-01).

[19] The PostgreSQL Global Development Group. PostgeSQL Documentation: Building Indexes Concurrently. — Access mode: https://www.postgresql.org/docs/current/sql-createindex.html#SQL-CREATEINDEX-CONCURRENTLY (online; accessed: 2022-12-20).

[20] The PostgreSQL Global Development Group. PostgeSQL Documentation: ALTER TABLE. — Access mode: https://www.postgresql.org/docs/15/sql-altertable.html (online; accessed: 2022-12-20).

[21] JUnit.org. JUnit official web site. — Access mode: https://junit.org/junit5/docs/current/user-guide/ (online; accessed: 2023-05-01).

[22] Szczepan Faber. Mockito official web site. — Access mode: https://site.mockito.org/ (online; accessed: 2023-05-01).

[23] Postman Inc. Postman. — Access mode: https://www.postman.com/product/what-is-postman/ (online; accessed: 2023-05-01).