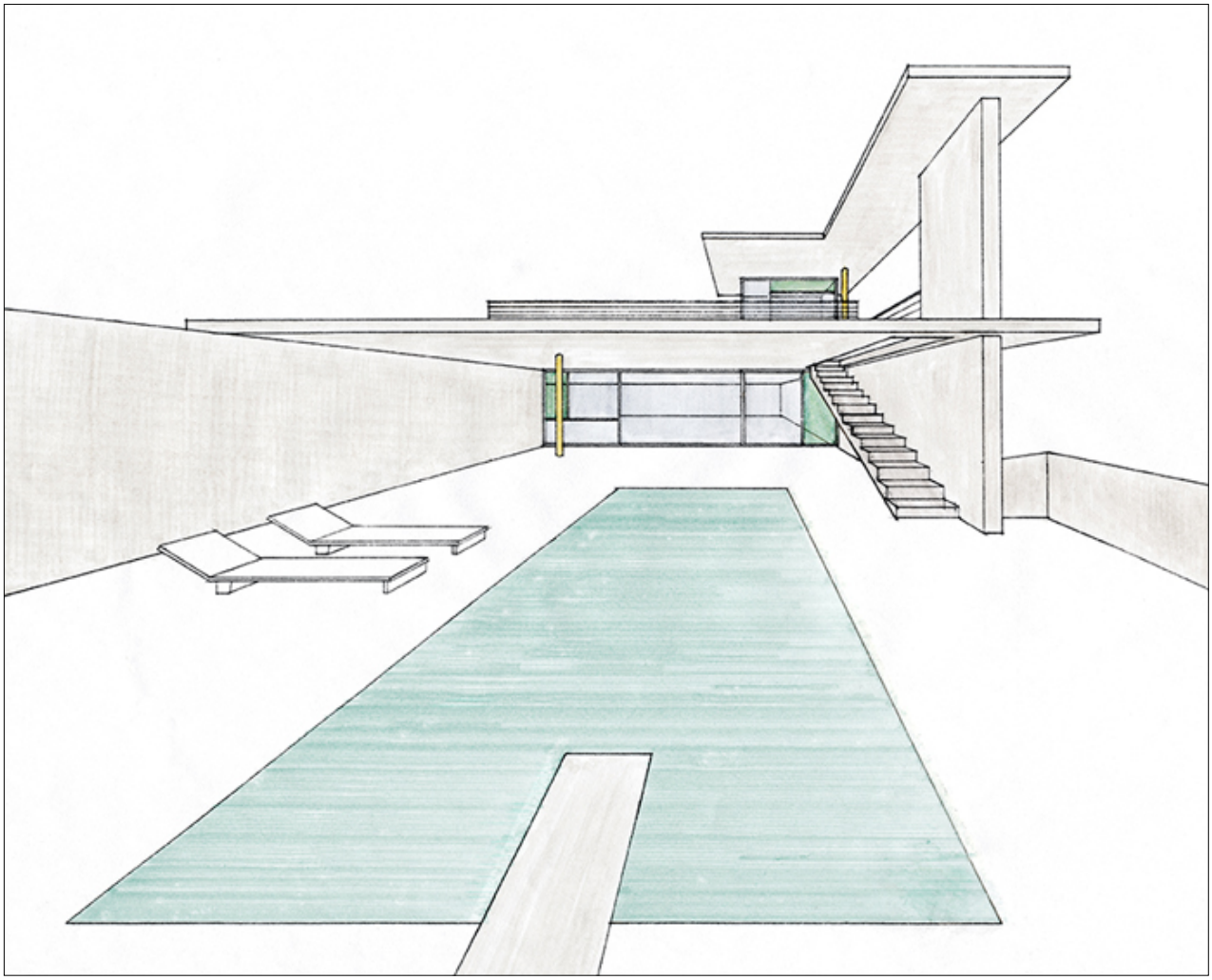
SHVYREV SERGEY



Приложение для конкурса FunCorp

Prepared by: Shvyrev Sergey
18 January 2020
Proposal number: +7 977 302 1878

ОСНОВНЫЕ ПОЛОЖЕНИЯ

Задание

Написать приложение на Java или/и KotlIn, которое с максимальной скоростью распознает и соберёт все мемы на немецком языке и вернёт их в качестве ленты контента, сопровождаемого метаданными, отсортированной по времени публикации от более свежих к более старым.

Цели

- Контент в приложении должен быть уникальным
- Гибкая настройка источников мемов
- Приложение должно работать в Docker-контейнере
- Сопровождаться файлом README с описанием сборки и запуска

Решение

В качестве реализации выбрал использование нативного приложения для клиентских запросов, Jvm приложения для парсера.
Обоснование :

- Никто не любит JDK в Docker.
- Нативные приложения пока имеют весомые ограничения, ограничивающие их использование.

Для реализации поставленных задач разработал два приложения :

- приложение для получения уникального контента (далее Parser)
- приложение для обработки клиентских запросов (далее Core)

Разделение функционала необходимо :

- для обеспечения бесперебойной работы приложения. Резкий рост capacity со стороны пользовательских запросов должен обрабатываться за счет максимально быстрого разворачивания дополнительных инстансов приложения. За счет уменьшения функционала Core и использования нативных билдов получим необходимый гандикап по времени запуска контейнера.
- приложение для получения контента от источников траффика не имеет жесткой зависимости от capacity роста клиентских запросов и времени разворачивания, но имеет более сложный функционал. Поэтому в такой реализации можно отказаться от использования нативных приложений и связанных с их работой ограничений.
- конфигурация приложений типа Core будет идентичной, с другой стороны конфигурация приложений типа Parser будет совершенно разной в зависимости от источников траффика, которые будут использоваться. Разделение на два приложения позволит проще контролировать инстансы через Service Discovery и конфигурацию каждого билда.

- использование нативной сборки с одной стороны ускорит клиентское приложение, но с другой, принесет с собой ограничения, которых нет в Jvm приложениях. Поэтому Core будет нативным билдом, Parser, будет инстанцироваться через JDK.

Таким образом разделим необходимый функционал между двух приложений.

В задачи приложения Core будут входить:
- обработка пользовательских запросов

В задачи приложения Parser будут входить :
- парсинг страниц источников, на наличие мемов
- определение языка
- получение контента (текстовой и медийной информации) от источников
- проверка на наличие ранне сохраненного контента, для предотвращения дубликатов
- сохранение полученного контента в формате данных приложения

ОПИСАНИЕ

Доменная модель приложения Core

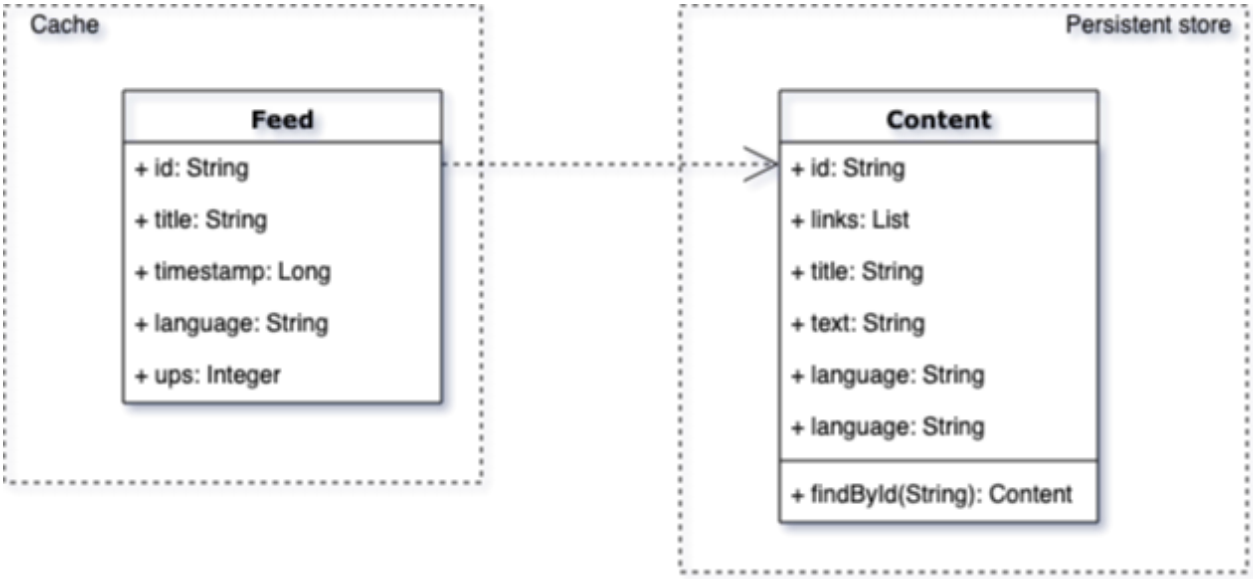
Для реализации доменной модели выбрал разделение только на две сущности Feed и Content. Сущность MetaData объеденил с Feed, т.к. из всех перечисленных в задании полей (дата/время публикации, источник, автор, язык, число пользовательских реакций, число комментариев и т.п.) на сайтах с немецкими мемами, которые нашел, было только поле “лайков”.
По запросу в гугле https://www.google.com/search?client=firefox-b-d&q=germany+mem+sites первая ссылка не открылась, на второй и третьей нет ссылки на сайты, и четвертая ведет на сайт Quora https://www.quora.com/Is-there-any-German-version-website-of-9GAG-or-FMyLife. Самый популярный комментарий предлагает три сайта на выбор. Первые два были однотипными и как раз не содержали никакой мета информации кроме лайков, а в третьем я не смог разобрался. Первый сайт - http://www.orschlurch.net/, был не только на немецком языке, но и на английском, это позволило протестировать определение языка текста на двух языках и при короткой длине текстового сообщения (редкое описание достигает 140 символов). Так же этот сайт содержал серию изображений для некоторых мемов, это позволило реализовать последовательную загрузку для таких мемов.

Для модели Feed кастомизировал спецификацию RSS.

Модель ***Feed***:
id - идентификатор
title - заголовок
timestamp - дата публикации
language - язык
ups - количество пользовательских лайков

Модель ***Content***:
id - идентификатор
links - ссылки для медиа контента
title - заголовок
text - описание
ups - количество пользовательских лайков

Модель Feed будет сохраняться только в кластере кэша приложения, модель Content будет сохраняться только в кластере персистентного хранилища.

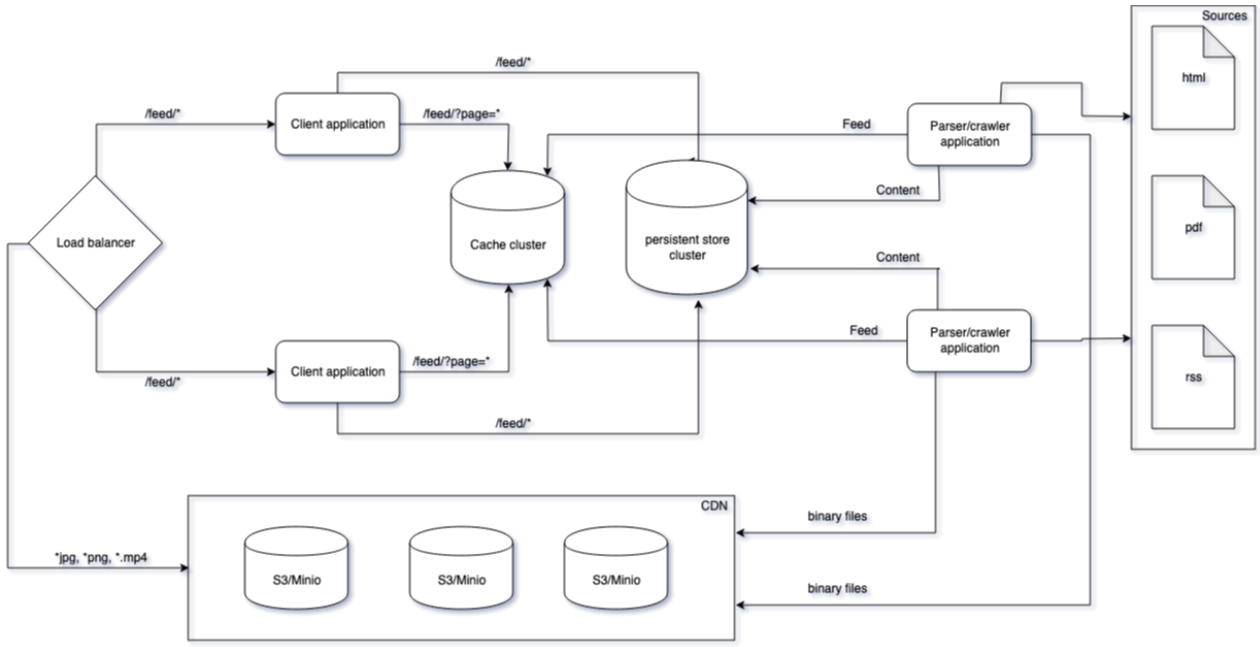


Доменная модель приложения Core

Parser получает информацию от источников, проверяет наличие совпадений, проверяет содержится ли язык мема в списке разрешенных языков, сохраняет Content в кластере персистентного хранилища, сохраняет Feed в кластер кэша, сохраняет медиа контент в S3/Minio кластер.

Core обрабатывает запросы ***/feed/**** получая данные из персистентного хранилища, запросы ***/feed/?[page=*]*** получая данные из кэша, запросы для получения медийного контента через ссылку на CDN. Это позволит сократить latency при загрузке ленты мемов.

Для исключения расхождений при обработке запросов (например, когда ноды кластера кэша синхронизировались раньше нод персистентного хранилища) использовал возможность повторных запросов продолжительностью около 500 миллисекунд (более подробно в разделе FaultTolerance).



Функциональная схема работы приложения

Выбор программных средств хранения

В качестве кластерного решения для реализации кэша выбран Infinispan.

Обоснование :

- бинарный протокол
- нативная реализация
- в двое меньший размер памяти

Все это позволит быстрее обрабатывать пользовательские запросы.

В качестве кластерного решения для реализации персистентного хранилища выбран MongoDB.

В качестве хранилища для медиа контента рассматривал S3 и Minio. Выбор сделан в пользу клиента Minio. В конфигурации приложения предусмотрены возможность использовать любое хранилище. Достаточно изменить доступы с Minio на S3 и приложение будет использовать AWS S3 и обратно.

Выбор фреймворка и JDK

В качестве фреймворка для реализации приложения Parser выбран фреймворк Vertx.

Vertx Http Client имеет идеальную реализацию для этих целей. Прежде всего не блокирует поток. Это важно так как буду использовать несколько воркеров для парсинга и загрузки контента одновременно.

Большим ограничением для всех парсеров и краулеров является работа с DNS именами.

Vertx использует собственную реализацию DNS Resolver. Она работает быстрее нативной. Использует для взаимодействия сервера Google (8.8.8.8, …) из коробки, либо настраивается любой другой DNS сервер по желанию. Для текущей реализации Google DNS вполне достаточно.

Vertx использует неблокирующий доступ к файловой системе. Это очень выгодно при загрузке медиа контента.

В качестве JDK используется GraalVM. Позволяет создавать нативные приложения и сокращает время инстанцирования java приложений.

Метрики

Метрики собираются в формате Prometeus. Кроме метрик клиентских запросов, добавил несколько метрик для различных методов. Измеряющих частоту и продолжительность вызова методов.

Описание настройки Prometeus доступно в README.MD.

Fault Tolerance

С целью исключения расхождений в процессе синхронизации, между Infinispan и MongoDb. Добавил повторные попытки. Каждый раз когда запрос завершается неудачно, 4 раза приложение пытается завершить успешно. Этого времени должно быть достаточно, чтобы синхронизация MongoDB догнала InfiniSpan.

Добавил Circuit Breaker для предотвращения ситуаций, когда доступ к кластерам Mongo или Infinispan будет ограничен.

Error Notifications

Мониторинг ошибок осуществляется при помощи подключения облачного сервиса Sentry. Он не бесплатный, но имеет достаточный функционал в бесплатной версии.

ОГРАНИЧЕНИЯ

На момент реализации вышедший релиз GraalVM 19.3 имел много критических багов, которые не гарантировали стабильной работы приложения. Вышедший пару дней назад хотфикс 19.3.1 не проверял. Использовал предыдущий стабильный релиз 19.2.1. К сожалению в этом релизе для комьюнити версии не было возможности использовать Java выше версии 1.8. Пришлось кое что переписать. Теперь выглядит не очень современно.
Кроме этого при использовании Kotlin в нативных приложениях есть существенные ограничения при работе с корутинами. Пришлось для реализации выбрать vanila Java.

При компиляции приложения с GraalVM столкнулся с ограничением на использование нативной реализации в Vertx паттерна Service Proxy. Связано с дублированием аннотаций при кодогенерации. Переписал решение с использованием различных Verticle в качестве воркеров. Количество воркеров зависит от доступных процессоров по формуле : N + 1, либо задается в конфигурации.