Продуманная архитектура и другие преждевременные оптимизации

Федор Лаврентьев

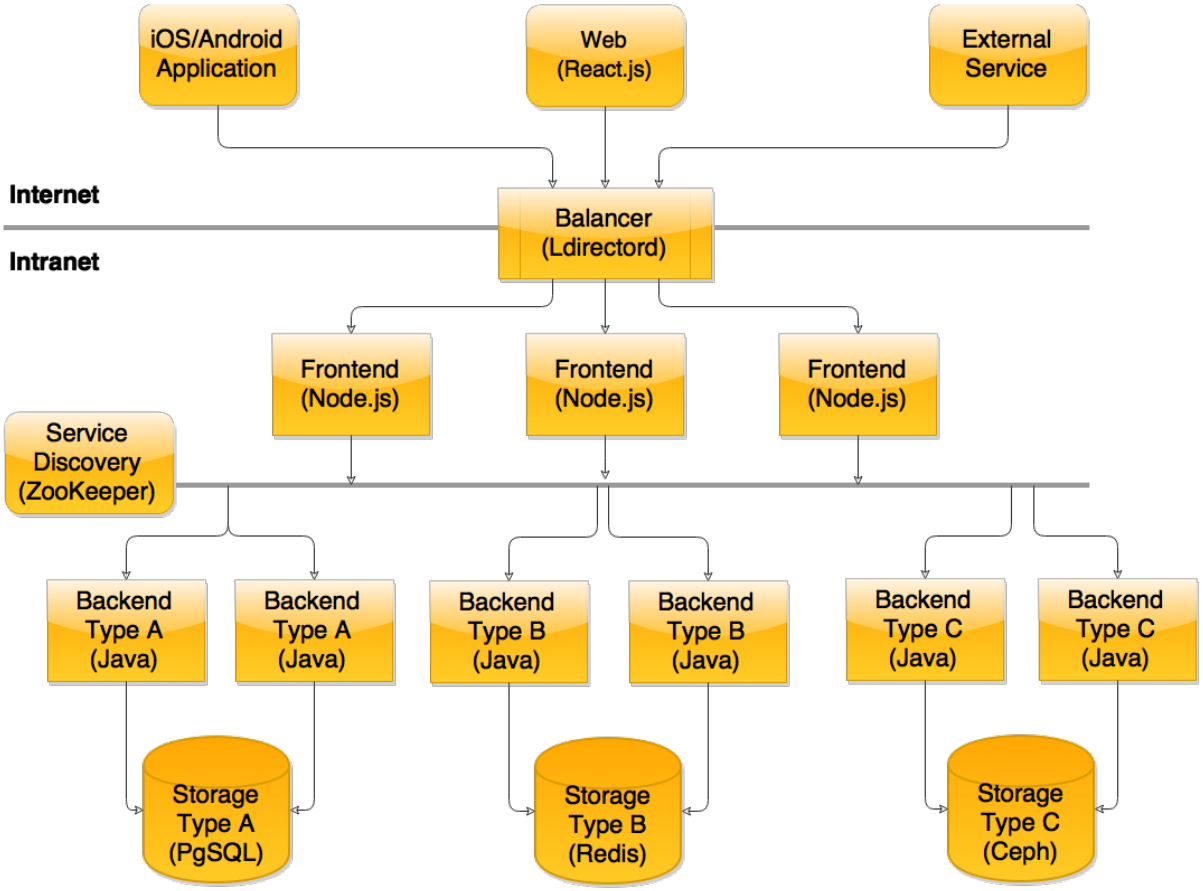
www.oorraa.net

19 июня 2015

С чего началось

- Оптовый B2B Marketplace
- Клиенты - оптовые поставщики и покупатели
- На момент начала разработки платформы компания существовала полгода
- Оценка аудитории - 1 млн MAU, 100к DAU
- Оценка пиковой нагрузки - 10к RPS

Архитектура приложения: как сделали



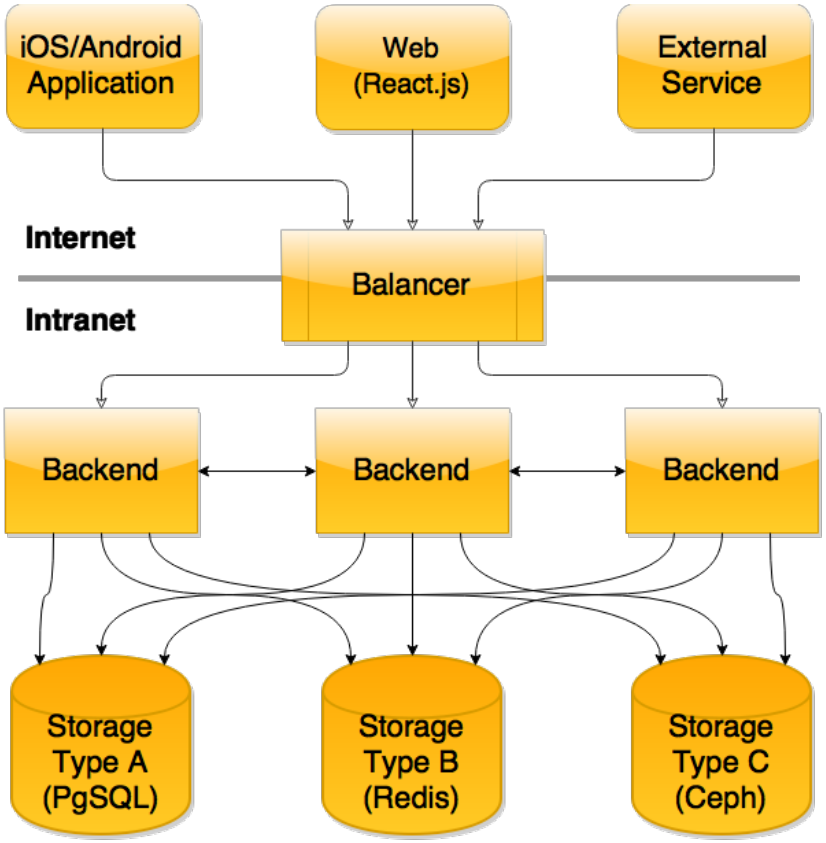
Архитектура приложения: очевидные плюсы

- Линейное масштабирование
- Отсутствие единой точки отказа
- Полная изоляция кодовой базы сервисов
- Защищенность хранилищ сервисов

Архитектура приложения: (не)очевидные минусы

- Сложно дублировать (сделать копию продакшена)
- Неподвластно волшебство Service Discovery
- Система может разваливаться по частям
- Приходится эксплуатировать 10+ процессов бекенда вместо одного

Архитектура приложения: как можно было сделать



Связь внутри кластера: что хотели

- Легковесный RPC, готовый к употреблению
- Реализация на Java, Node.js, Python
- Схемы API - валидация, документация и пр.
- Возможность эволюции API (версионирование и пр.)
- Поддержка Service Discovery
- Встроенная балансировка нагрузки

Связь внутри кластера: что сделали

- Apache Thrift как протокол
- Apache ZooKeeper как хранилище конфигурации
- Twitter Finagle на бекенде (Java)
- Vanilla Thrift RPC на фронтенде (Node.js, Python)

Связь внутри кластера: очевидные плюсы

- Полностью готовая реализация на бекенде
- ZooKeeper - это надежно
- Бинарный протокол - это быстро и решительно
- В Thrift нативное версионирование
- Строгие схемы, документация в .thrift-файлах
- Twitter Finagle реактивен

Связь внутри кластера: (не)очевидные минусы

- Все должны знать, где живет ZooKeeper
- Реализация Thrift для Node.js и Python ужасна
- Дебажить проблемы бинарного протокола - боль
- Админы не влияют на Service Discovery
- Приходится патчить ~~кривой~~ код на Scala
- Twitter Finagle реактивен o_O

Связь внутри кластера: как переделали

- Выпилили Thrift, Finagle и ZooKeeper
- Только JSON
- HTTP REST как API сервисов
- Nginx upstream proxy вокруг каждого сервиса
- Адресация на уровне DNS
- Валидация и версионирование - молитвами

Быстрое хранилище: что хотели

- Хотим чат на платформе
- На чат придется львиная доля запросов
- Для чатов надо хранить историю переписки, списки собеседников, предмет беседы
- Бизнес-логика тесно завязана на сортировку
- Постоянно дергаются различные счетчики
- Требования к чатам неоднократно изменялись

Быстрое хранилище: что сделали

- Все данные о чатах хранятся в Redis
- Много индексов (zset)
- Много счетчиков (incrby + get + getset)
- Метаданные лежат в хешах (hset)
- Тела сообщений лежат там же в виде JSON

Быстрое хранилище: очевидные плюсы

- Дьявольски быстро работает
- Относительно просто масштабируется
- Сортировки, пагинации и счетчики из коробки

Быстрое хранилище: (не)очевидные минусы

- Собственные Client-Side Join
- Собственные ~~транзакции~~ корректировщики
- Надо не забывать обновлять индексы
- Надо держать в голове всю структуру БД
- При изменении требований нужны новые индексы
- Дьявольски сложно рефакторить
- Нереально анализировать данные in-place

Быстрое хранилище: что получилось

- Бекенд переусложнен
- Структура БД избыточна
- Данные окаменели
- Рефакторинг взрывоопасен
- Самое медленное место в системе
- Для анализа копируем все в SQL

Быстрое хранилище: как переделали

- Положили все данные в PgSQL
- Построили удобные индексы
- Счетчики считаются агрегатами
- Оставили возможность шардировать
- Для тяжелых счетчиков и индексов - кеш в Redis

Выводы

- Чем круче система, тем сильнее выпирают ее преимущества и недостатки
- При проектировании недостатки кажутся не такими критичными
- Они и вправду не критичны, но иногда мешают
- Плохо знакомые технологии несут больше скрытых недостатков и имеют меньше надежных костылей

Спасибо за внимание!

Федор Лаврентьев
f.lavrentyev@oorraa.net

www.oorraa.com
www.oorraa.net