Как Dhall вытеснил Postgres

Артем Канев
GetShopTV

2020

Кто такие?

GETSHOP.TV
TV Commerce Platform

GetShopTV

Стартап про интерактивную рекламу через ТВ-приставки. Haskell под капотом.

http://getshop.tv/

Артем Канев

Fullstack middle в GetShopTV. Предпочитает Haskell другим языкам программирования.

https://github.com/nixorn

Что делают?

Крутим рекламу через телевизионные приставки.
Чтобы понять что, когда и на какой приставке показать серверу нужно знать нехилый контекст:

- локальное время приставки
- временную зону
- канал
- геолокацию приставки
- идентификатор приставки
- разрешение экрана
- контент и его настройки
- прочая, прочая, прочая…

Как было до: Postgres

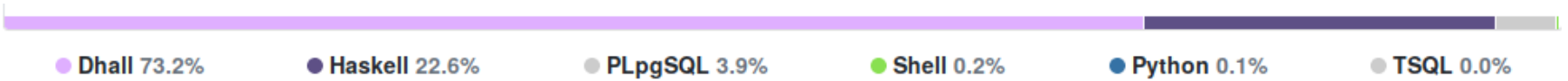
users_devices	reactions	users	extras
phone_calls	channels	accounts	hot_calls
keys	devices	sessions	creatives
sms_parts	smss	events	phones
sess_extras	permissions	redirects	sessions_web

Переход

Порядок событий:

- выход в прод
- команда осознает что решение не держит нагрузку, изначальные прикидки отличаются от факта как “Винни-Пух” от “Пилы”, попытки оптимизации неудачны
- CTO сообщает правлению, что придется писать новый сервер с нуля с той же API но другой архитектурой, посматривает на Dhall
- 2 месяца разработки
- ядро написано, Dhall в процессе внедрения
- я прихожу на проект
- еще 3 месяца разработки
- новое решение в проде

Как стало после: Dhall



phone_calls	channels	accounts	
	devices	sessions	creatives
	smss	events	phones

```
$ find . -name *.dhall -exec cat {} \; | wc -l
44104
```

Про Dhall

Gabriel Gonzalez

Создатель Dhall и много чего еще.
Широко известен в узких кругах.

https://github.com/Gabriel439
http://www.haskellforall.com/

Dhall

Программируемый язык конфигурации,
повернутый на безопасности.

https://dhall-lang.org/

Dhall: Фичи и особенности

- ~~don't repeat yourself~~
- импорты
- безопасный рефакторинг
- программирование
- типы
- не полный по Тьюрингу
- есть только чтение (файлы, окружение), но не запись

Что не так с SQL

- плохо адаптируется к изменениям требований
- дополнительная сложность
- легко повредить данные

SQL и изменения: Миграции

Миграции это ответ SQL на изменение требований. Три вида непредсказуемости миграций:

- применяется дольше, чем предполагалось
- не применяется
- повреждает данные

SQL и изменения: Редуценты

Если в базу не вливать достаточно времени, она мутирует.

- таблицы обрастают редуцентными полями
- логика мутирует вслед за схемой

```haskell
-- | User role.
data UserRole
  = UserRoleAdmin       -- Используется
  | UserRoleManager     -- Не используется
  | UserRoleAdPlatform -- Используется, но имя должно быть другое
  | UserRoleAdvertiser -- Не используется
  | UserRoleClient      -- Не используется
  | UserRoleChannel     -- Используется
  deriving (Eq, Show, Bounded, Enum, Ord, Generic)
```

Дополнительная сложность

База это standalone сервис, и у нее своя игра. Когда она приходит, воссоздание окружения становится дороже.

- данные для тестов
- бекапы
- менеджмент
- мониторинг
- тюнинг

Легко повредить данные

Пути повреждения базы:

- кривая миграция
- кривое приложение
- ошибка в ручном update
- операционная ошибка администратора
- неисправность железки

Изменение требований и SQL

Как происходит разработка софта в реальной жизни:

1) устаканить требования
2) заимплементить
3) провести процедуры контроля качества
4) выкатить
5) узнать, что требования поменялись
6) GOTO 1

SQL заточено на декларативное описание предметной области, и уязвимо к её изменениям. А изменения происходят часто.

Конфиг классический

Содержит данные, которые

- определяют поведение приложения
- не изменяются между запусками приложения

Умеет хранить

- простые значения (числа, строки, булевы)
- структуры (списки, key-value)

Ошибки интерпретации данных

Это когда программа не может понять что значат данные и это ведет к одному из двух исходов:

- исключение
- неправильное поведение и молчаливое продолжение работы

Такие ошибки бывают двух видов:

- логические
- структурные

Логические ошибки

Примеры

- True вместо False
- комбо настроек, образующее формулу
- скопировали для дебага и забыли удалить

Как бороться

- тесты
- хорошее ревью

Структурные ошибки

Примеры

- отсутствие поля
- отсутствие ключа в key-value
- неверный тип значения

Как бороться

- валидировать данные

Киллерфичи Dhall(ИМХО)

- превращает некоторые логические ошибки в структурные
- не допускает структурные ошибки как явление
- экстремально адаптивен к изменениям

Переосмысление роли конфига

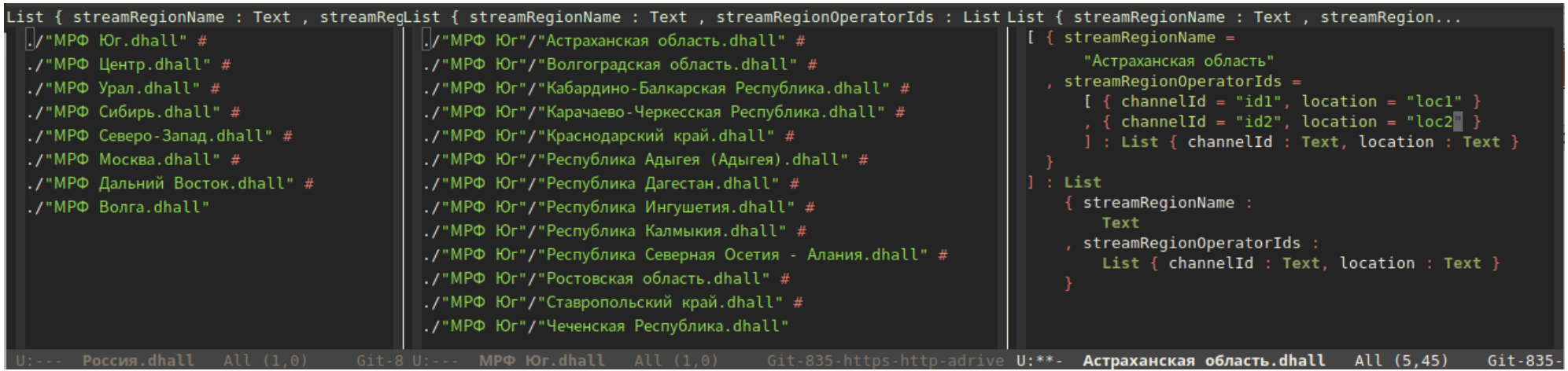
Как много у вас данных которые не изменяются между запусками приложения?

Могут ли эти данные влезть в оперативную память без урона производительности?

Можете ли вы представить для них структуру, в которой удобно их искать и держать в памяти?

Насколько источники этих данных поддаются машинной обработке?

Переосмысление роли конфига



Текстовые файлы, раскиданные по директориям. Они сделаны скриптом, источник – csv от партнера. Тут тысячи значений. Больше похоже на базу чем на конфигурацию.

Источники данных

- скрипты импорта из внешних источников
- самописные системы редактирования контента
- руки и головы разработчиков

Достоинства подхода

- больше безопасности в
  - рефакторинге (миграции)
  - деплое
  - рантайме
- редактируется как конфиг, структурировано как база
- данные это код, он под гитом
  - проще воссоздать окружение
  - не боишься за потерю данных

Недостатки подхода

- бесячий синтаксис Dhall
- на обучение людей нужно время
- применение ограничено с языками без поддержки Dhall
- придется велосипедить туллинг под свою ситуацию

Демо

https://github.com/nixorn/dhall-demo

Благодарности

- команде GetShopTV
- Габриелю Гонсалесу
- сообществу https://ruhaskell.org