Рефакторинг приложения с использованием Go

Виталий Левченко
системный архитектор PropellerAds
19 июня 2015 года

Кто я

- PHP разработчик
- Архитектор
- Организатор Go митапов в СПб
- Проповедую bleeding edge технологии

Кто я

- PHP разработчик
- Архитектор
- Организатор Go митапов в СПб
- Проповедую bleeding edge технологии

Исходное состояние

Исходное состояние

- Монолитный рекламный сервер на PHP (OpenX).
- Костыли из-за быстрого роста нагрузки.
- Критическая масса технического долга.

Проблемы

- Производительность (rps/node, latency)
- 15+Gb tmpfs cache
- Проблемная поддержка и доработка
- Полное отсутствие тестов

Цели

- Производительность
- Разгрузить БД
- Упростить работу с кодом
- Тесты

Варианты

- Точечный рефакторинг
- Масштабный рефакторинг
- Выделение компонент

Точечный рефакторинг

- Профилируем и исправляем.
- Архитектурные проблемы не видны и не решаются.

Точечный рефакторинг

- У нас:
  - элементы кеша до 10Мб;
  - сериализация и десериализация очень дорогая;
  - много вычислений в памяти;

Точечный рефакторинг

- У нас:
  - элементы кеша до 10Мб;
  - сериализация и десериализация очень дорогая;
  - много вычислений в памяти;

Массовый рефакторинг

- Можно, но…
- Нужно разгрести технический долг.
- Часть проблем плохо решается с PHP.

Массовый рефакторинг

- Можно, но…
- Нужно разгрести технический долг.
- Часть проблем плохо решается с PHP.

Выделение компонент

- Можно сразу сделать хорошо
- Любой язык программирования
- Микросервисная архитектура
- 12 factor application

Выделение компонент

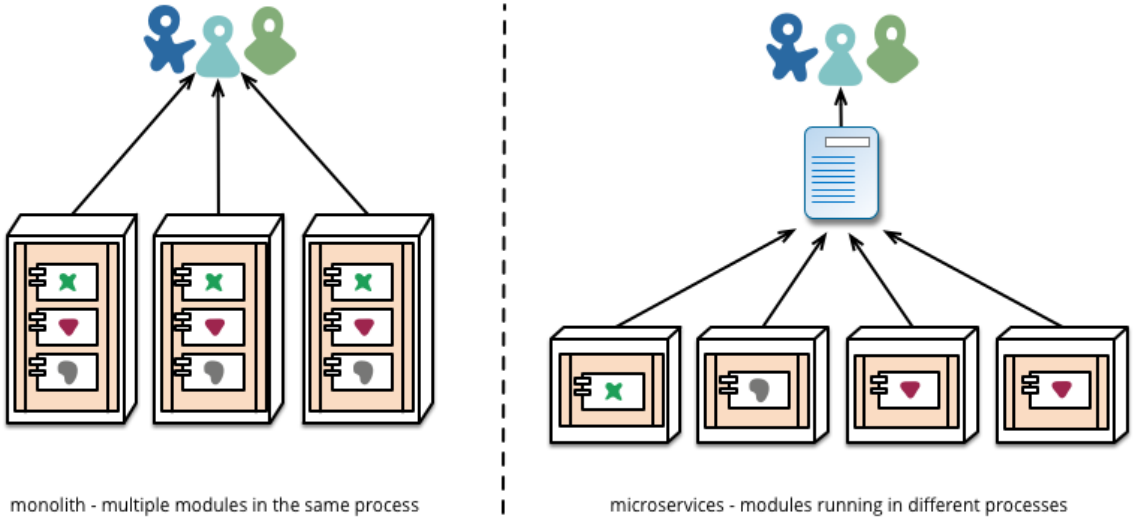
- Можно сразу сделать хорошо
- Любой язык программирования
- Микросервисная архитектура
- 12 factor application

Микросервисная архитектура

- Shared nothing
- No persistence state
- HTTP interface

Микросервисная архитектура

- Позволяет прозрачно масштабироваться
- Изолирует компоненты
- Упрощает разработку



monolith - multiple modules in the same process

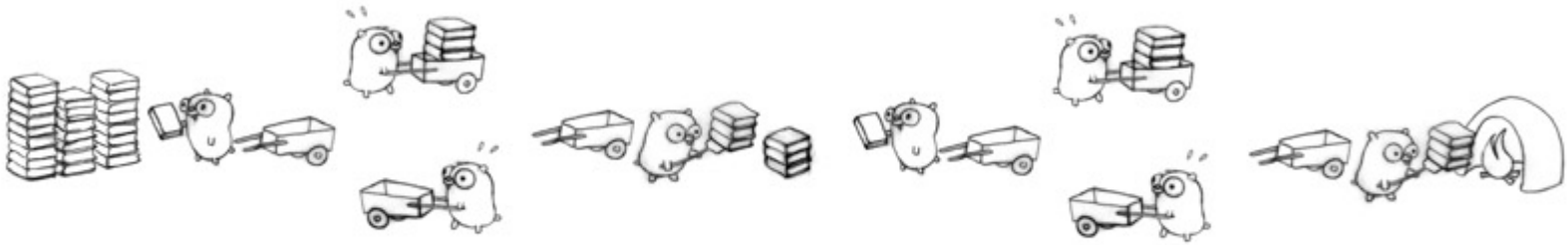
microservices - modules running in different processes

Языки программирования

- Не скриптовые: слишком неэффективно
- Не C++: слишком долго
- Не Java/C#: too verbose, большие требования к экспертизе

Go

- Эффективная работа с памятью
- Эффективная работа с CPU (как Java)
- Простая и эффективная параллелизация

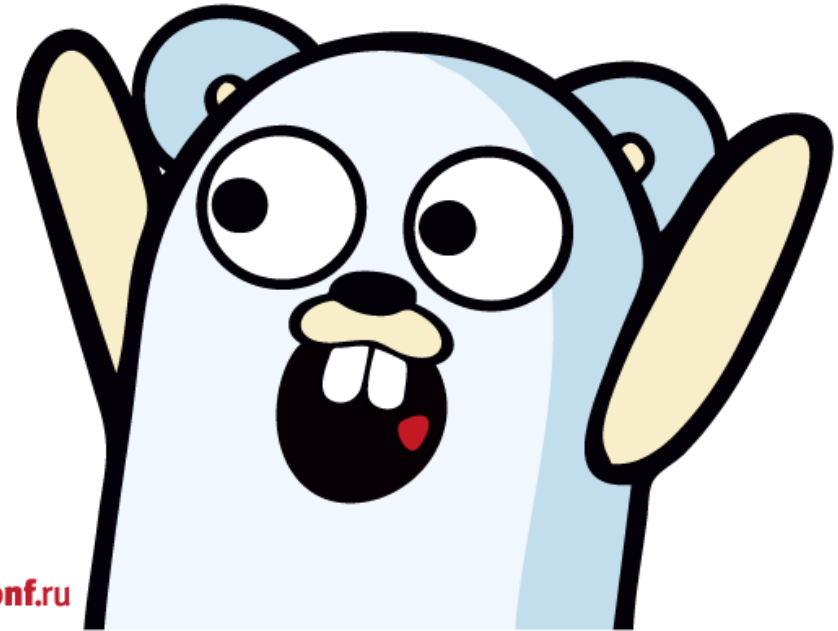


Go

- В разработке похож на скриптовые языки
- Обучение за 1 день!
- Новые инфраструктурные проекты выбирают Go

Go

- В разработке похож на скриптовые языки
- Обучение за 1 день!
- Новые инфраструктурные проекты выбирают Go



Реализация

- Выделяем самую тяжёлую и изолированную логику
- Заменяем вызов функции на вызов сервиса
- Здесь: ротация рекламы (выбор баннера)

Реализация

- Разделяемый кеш в памяти
- Кеш максимально близко к БД
- Оптимизация алгоритмов

Проблемы

- GC
- NUMA
- Обработка проблем с БД
- Таймауты и зависающие коннекты

Производительность

- Latency: 100ms -> 20ms
- Rotation: 80ms -> 1.5ms
- PHP memory: 300Mb -> 10Mb
- RPS/node: 300 -> 1500
- Сервис: 10k rps/instance, 1.8k rps/core

Инфраструктурные проблемы

- Service discovery
- Протокол взаимодействия
- Деплой и оркестрация
- Сбор логов и мониторинг

Профит

- Доработка несравнимо удобнее
- Устранено узкое место
- Фундамент для новых микросервисов

Бонус

Как сделать микросервисы?

- Минимальные компоненты
- Docker контейнеры
- Consul.io: service discovery + health check
- HTTP интерфейсы
- Опционально: Kubernates/Nginx+ для виртуализации IP
- Splunk логи
- Prometheus мониторинг

Вопросы?

Спасибо

Виталий Левченко,
системный архитектор PropellerAds