Using PHP to Build a Mobile Internet Platform

Florian Hoenig <florian@kargo.com>
Overview

1. About Kargo
2. PHP beyond “Hypertext Preprocessing”
3. Kargo’s Mobile Web Architecture
4. Conclusions
5. Ideas & Discussion
Who is Kargo?

Kargo is a mobile media company that conceives, builds and manages leading mobile products for media properties.

1. Mobile Web
2. Web Applications with Mobile Integration
3. Messaging Applications
4. Native Applications (J2ME, iPhone, Blackberry, etc.)
Web/Messaging
Panoramic Applications

Web Apps

Mobile

Widget

Messaging

Text “BOB12” to 85555 to get content
Technical Challenges

• Many thousands of variations of devices from over 40 vendors to target
• Devices vary strongly in terms of capabilities (and bugs!)
• Network speeds very strongly
• Significant amount of engineering and operational time spent on compensating for the above.
How to address?

- Need for a flexible architecture
- Loosely coupled (web-) technology helps
- Single choice of technology
- PHP!
PHP beyond Hypertext
Preprocessing
building enterprise architectures with PHP5 and Zend Framework
Enterprise Architecture

• From Microsoft.com: “...a methodology to encompass all of the various IT aspects and processes into a single practice.”

• From MIT: “...IT infrastructure reflecting the integration and standardization requirements of the firm’s operating model.”

In simple words: A single practice to reflect what the company does
Qualitative Requirements

- flexibility - platform’s potential to address new business requirements
- rapid development - reuse, simplicity, short learning curves
- performance - write efficient apps should be straight forward
- reliability - robust platforms, solid testing, security
- interoperability - offer simple ways to talk to interface with third-party systems
- simple deployment - application installation and update
- scalability - architectures must scale on demand
- high availability - be able to guarantee uptime
- manageable costs - ability to budget
Qualitative Requirements

- flexibility - platform’s potential to address new business requirements
- rapid development - reuse, simplicity, short learning curves
- performance - write efficient apps should be straightforward
- reliability - robust platforms, solid testing, security
- interoperability - offer simple ways to talk to interface with third-party systems
- simple deployment - application installation and update
- scalability - architectures must scale on demand
- high availability - be able to guarantee uptime
- manageable costs - ability to budget

How to address with PHP?
Motivation and Claim

- Scripting languages (e.g. Python, Ruby) continue to grow in popularity for many applications where PHP is usually not considered.

- PHP is ready to do the same
Legacy and Evolution

Two random code snippets. PHP looks quite different nowadays.
But, is it ready for Enterprise environments?
Zend Framework

- Already offers several enterprise grade features, principles and practices for web development
- Has roadmap for much more. Highly active community that you can plan with.
- Could be foundation to take PHP on the market for general purpose application development à la Java, .Net, C++.

Read more at: http://framework.zend.com/about/overview
Zend Framework

- Already offers several enterprise grade features, principles and practices for web development
- Has roadmap for much more. Highly active community that you can plan with.
- Could be foundation to take PHP on the market for general purpose application development à la Java, .Net, C++.

Read more at: http://framework.zend.com/about/overview
Considerations for building your architecture using ZF/PHP

1. Identify architectural layers of your environment
   - Use PHP “traditionally” on the front end
   - Consider using PHP-CLI for back-end services

2. Maintain your own class framework
   - Adapt from ZF: best practices, coding standards, testing standards, etc.
   - Really solid - no need to re-invent
   - Be ready to patch ZF components, without waiting for next release

3. Manage your PHP configuration
   - choose your extensions carefully
   - optionally write your own extensions when needed
More Considerations for building your architecture using ZF/PHP

4. Avoid using PEAR
   - lack of consistency
   - not as enterprise-friendly as ZF (license)

5. Avoid tying yourself to a particular system environment
   - don’t let the environment dictate your application design

6. Avoid using tightly-coupled PHP frameworks (“CMSs”)
   - Often very restrictive and hard to extend or integrate with other systems
What’s missing?

- Multi-threading
- A more concrete life-cycle model for long running processes
- However: not a huge sacrifice
Kargo’s Mobile Application Platform
Kargo’s Architecture: Logical Overview

Application Layer
- Mobile Web Applications
- Web Applications
- Public Web Services

Foundation Classes
- Messaging Service
- Billing Subsystem
- Device Management
- Reporting Engine
- Digital Asset Manager

Persistence Layer
- Database Nodes
- Shared FS
Kargo’s Architecture: Product Overview

**Kargo Framework**
Zend Framework plus mobile specific Kargo class library. Basis for all other products.

**Karpet**
Feed-based mobile magazine application with community features. Abstracts mobile specifics.

**Fever**
Mobile downloadable content and application storefront application, with cross-carrier billing.

**Kamp**
Web-based mobile messaging application. Allows for editorial content and costumer interaction.

**Kellner**
Mobile Messaging Application Server. Unified programming model, across all messaging protocols.

**Kargo DAM**

**Kargo Platform**
The core business logic application server. Handles subscriber management, device detection, reporting, ad serving, etc. All front-end applications use this as their back-end.
Zend Framework offers a lot already, but here is what we had to do ourselves:

- **Kargo_Daemon** - toolkit for forking, shared objects, IPC, etc.
- **Kargo_Dam** - wrapper classes for our digital asset management system
- **Kargo_Db** - additions to Zend_Db_Adapter, supporting fail-over and load-balancing.
- **Kargo_Event** - wrappers for our libevent extension for asynchronous socket programming.
- **Kargo_Mime** - a full and flexible MIME implementation to use beyond Mail.
- **Kargo_Mm7** - a full MM7 6.8.0 protocol implementation.
- **Kargo_Platform** - controller plug-ins for interaction with platform services
- **Kargo_Service** - a set of Zend_Service style third-party web-service wrappers for our partners.
- **Kargo_Smpp** - implementation for OpenMarket, SinglePoint, mBlox and Standard SMPP v3.4.
- **Kargo_WebDav** - a webdav implementation
Example: Approach for the Mobile Web Architecture

- Problem: Classical web server models are sub-optimal for a lot of new applications.

- Usually solved with lots of “workarounds”, mostly incorporating caching techniques on many levels.

- Solution: Cleanly decouple front-end logic from core services and let the back-end be a stateful application.

- This is common in Java-based architectures, but never seen in pure PHP.
Platform server is a stateful application using asynchronous http handling
What’s going on (simplified)

1: first request

2: start session (headers)

return: device, user, carrier, etc.

3: write session

4: another request

5: sign up [sends text message to user]

6: sendSMS()
How it works

1. Setup Db, logger, etc. only on startup

2. Start Http server with it’s socket being monitored by the underlying event notification system

3. On an incoming http request, the handleRequest callback is triggered, serving the xml rpc request

Now we can keep our back-end services running stateful
Performance

• Baseline: Apache/mod_php5, pre-fork, no opcode cache

• Setup: PHP 5.2.6 on Macbook Pro/OS X (KQueue) with libevent 1.4.5 and Zend Framework 1.5

• Concurrency level: 50 requests

• Apache: Mean requests per second: 8.76

• Kargo (one process): Mean requests per second: 450.82
Pros

- Incredible speedup
- Low CPU and memory utilization
- Surprisingly stable (runs for months now under heavy loads)
- No need for memcached, opcode cache, db connections pooling, or similar techniques.
- Back-end applications are now stateful. Allows for lazy loading, better task scheduling, etc.
- Lot’s of database queries can be kept in memory.
Cons

- Developers have to be more aware of memory management
- Some people call it “abuse of PHP”, since it ignores the request based PHP execution cycle.
- Garbage collection a request tear-down is not called.
- Doesn’t work well with Zend_Controller_Front, because of extensive cyclic referencing (memory leaks)
However

- Current server does not leak memory.
- Near Future: PHP 5.3 adds control over the garbage collector, which adds additional support for this scenario.
Conclusions

- Use of Zend Framework and its principles opens paths into solid software architectures.

- Using PHP as a general purpose programming language has shown to be surprisingly fast and robust in a production environment.

- Kargo has successfully replaced Java-typical application servers with PHP, leaving a pure single choice of technology across all tiers.

- Kargo has build an enterprise-grade architecture, simple and flexible enough to react to the complex requirement in the mobile domain.
How about building an app server which passes Zend_Controller_Request objects to a pool of workers?

How about a restricted ZF app server similar to Google App Engine, running on Amazon EC2?
Thank You!

florian@kargo.com

text “FH” to 99799