Massively Scaled High Performance Web Services with PHP

Demin Yin

slides available at http://deminy.in

About Me

- Staff Engineer and Technical Lead on Glu Mobile's Design Home team.
- Playing with PHP since 2000.
- Worked for a diverse group of companies in China and US, including eBay, Visa and Glu.
- Focus on building high performance, secure web services.
- Born in China, worked in U.S., live in Canada.
- On twitter: @deminy
- On Github: https://github.com/deminy

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

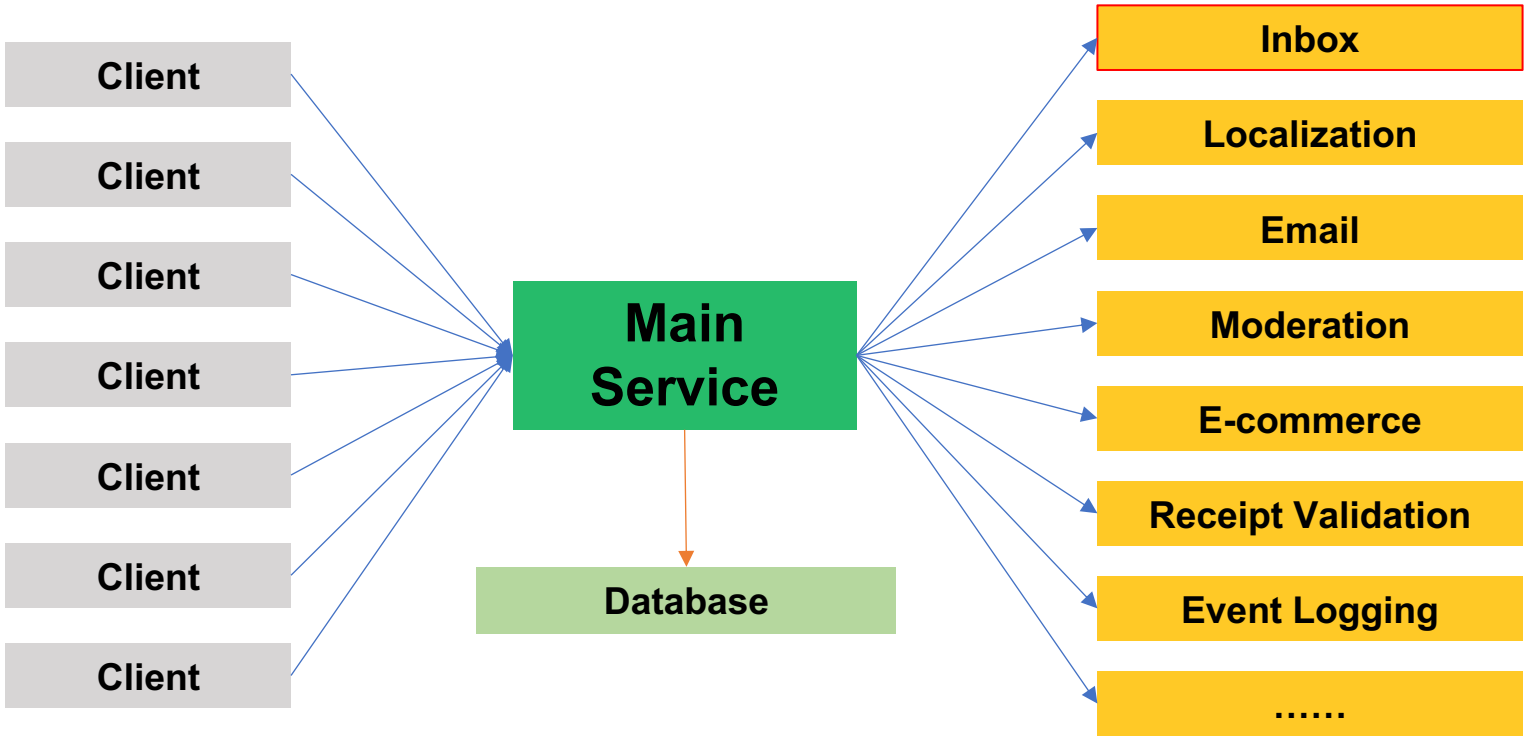
Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

Design Home:
Top-ranked Mobile Game with PHP Backend

- Launched in Q4 2016.
- #1 Top Free and #7 Top Grossing game in the U.S. App Store for iPhone.
- #1 Top Free and #23 Top Grossing game in the U.S. Google Play Store.
- 1M+ daily active users; 40M votes daily; 1 billion designs to date.
- 100,000 API calls per minute; ~~60~~ 56 ms response time on average.

Design Home: List of Backend Services



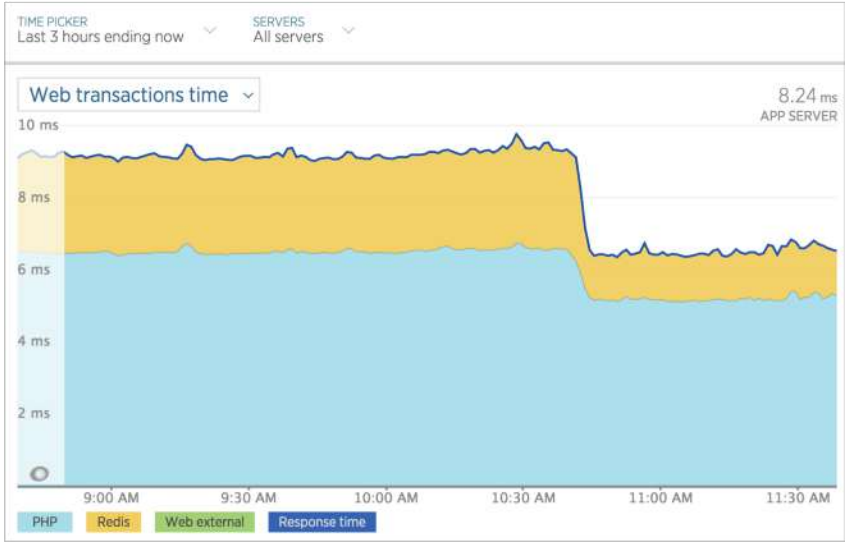
The Inbox Service: Technical Overview

- REST APIs with different CRUD endpoints to manage inbox messages.
- PHP 7, Nginx, Composer, NoSQL, Docker, AWS, Development Tools, ……
- NoSQL: Redis + Couchbase.
- One of the busiest microservices.

  Average Requests: 14k rpm

  Peak Requests: 34k rpm

  Response Time: less than ~~20~~ ~~11~~ ~~9~~ 6 ms



The Inbox Service: Technical Overview (cont.)

- PHP 7.
- Composer.
- Opcache.
- Data cache: APCu, distributed cache and chained cache.
- Tests in place.
    - Unit tests.
    - Blackbox tests, or feature tests, or functional tests.
- PHP The Right Way: Follow good coding practices including design patterns.

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

The Inbox Service: Key Improvements Made

Web Server Instances

- Decouple microservices.
- Dockerize the microservices.
- Development Tools Integration.

HTTP Processing

- Use background processing.
- Use HTTP compression (gzip).
- Dynamic content caching.

Data Storage

- NoSQL solutions.
- Fight against IO and disk space.
- Faster PHP drivers.

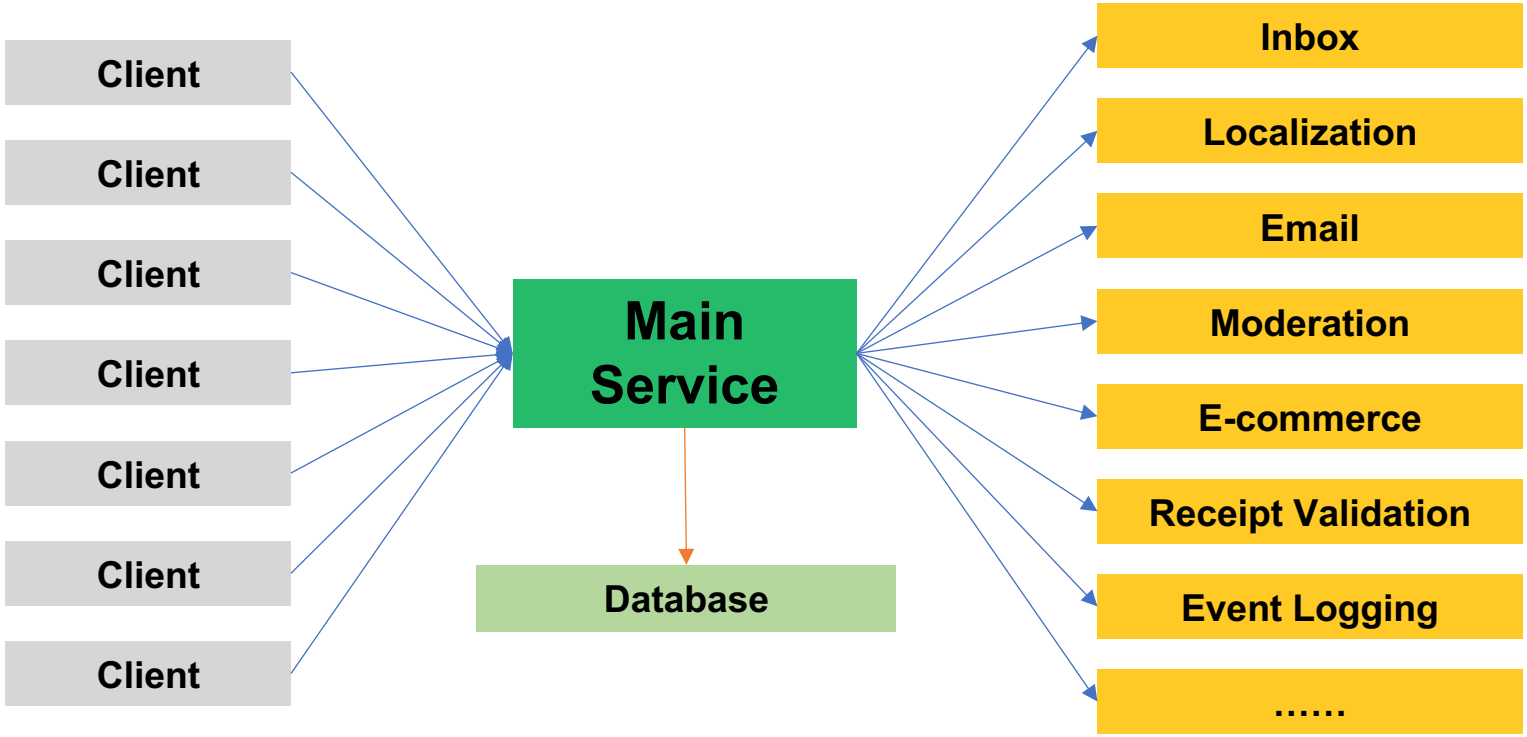
Hardware & Network

- Migrate to Amazon ECS.
- Hardware upgrade.
- Move everything into VPC.

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

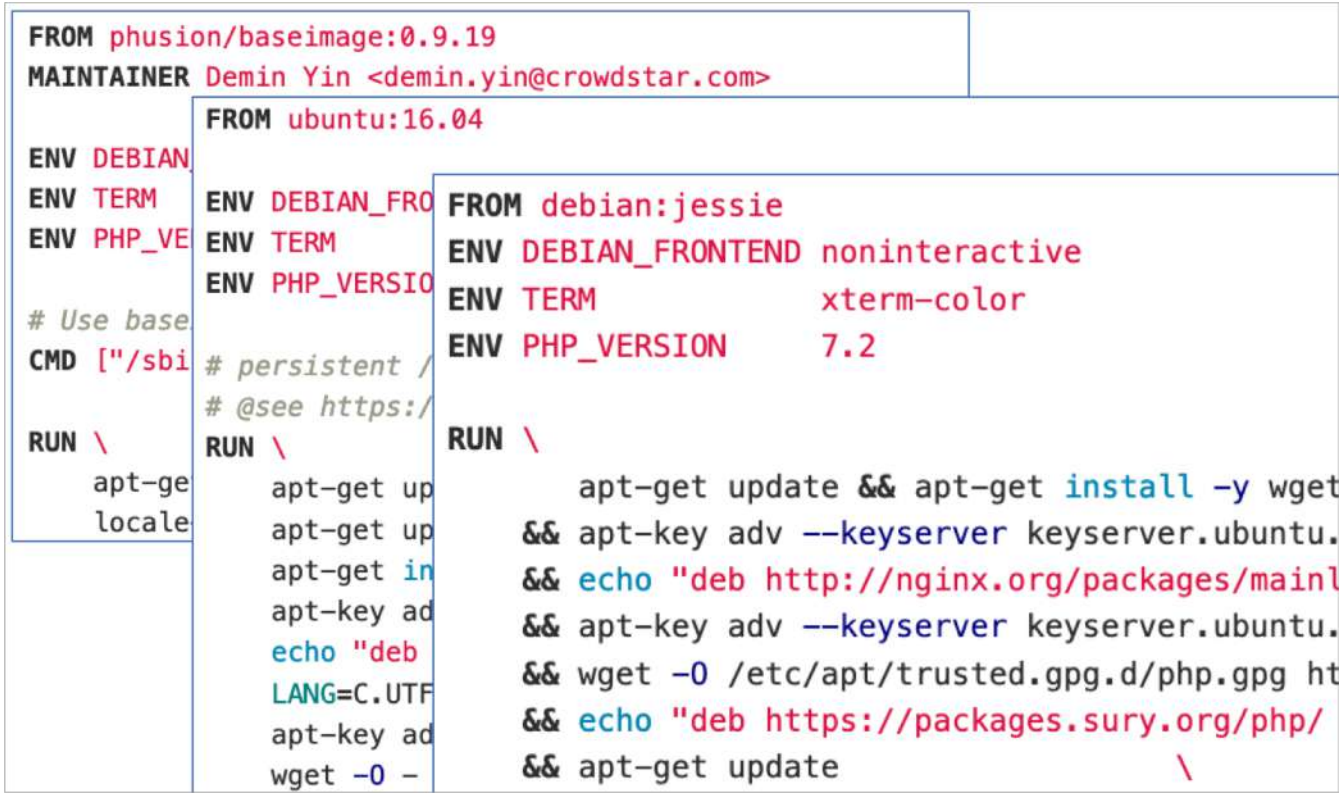
1.1. Decouple Microservices



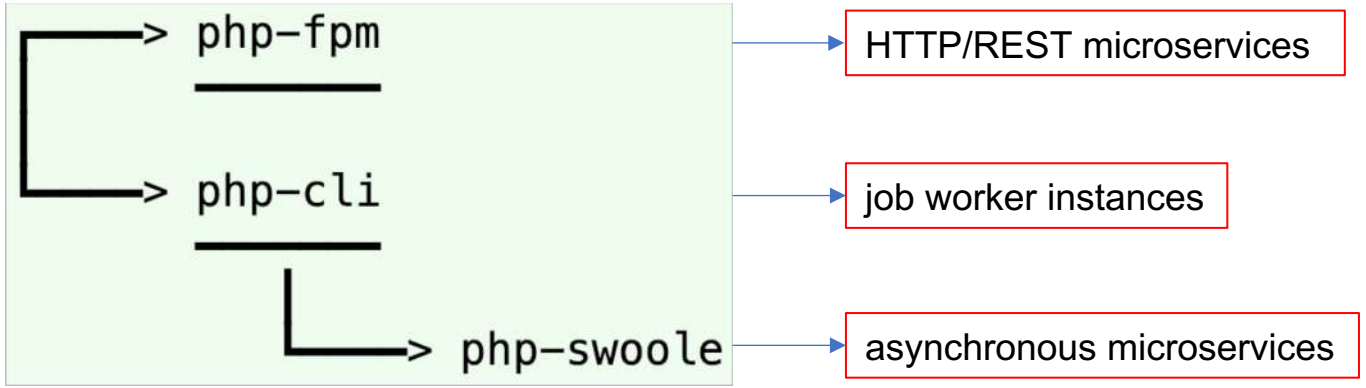
1.2. Dockerized Microservices

- Improved and simplified development environments.
- Easier to upgrade web server and PHP environments.
- Easier to troubleshoot and debug issues.

1.2. Dockerized Microservices (cont.)

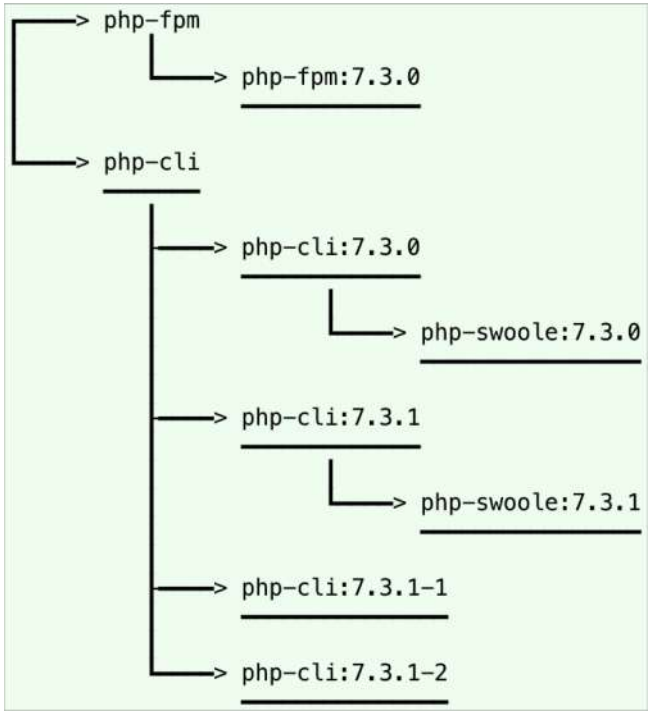


1.2. Dockerized Microservices (cont.)



Base Images

1.2. Dockerized Microservices (cont.)



Base Images

When creating base images:

1. Tags match with PHP versions.
2. Tag *latest* is never used.
3. Freeze tagged base images once used on production.
4. Base images are built and tagged manually.

Benefits:

1. Less efforts to build images for different microservices.
2. Easy and safe to upgrade to new versions of PHP.
3. Easier to fix security vulnerabilities.
4. Easier to make improvements across different microservices.

1.2. Dockerized Microservices (cont.)

```
php-fpm:
  7.3.2:
    status: active
    timezone: America/Los_Angeles
    nginx: "1.10.3-1+deb9u2"
    composer:
      version: 1.8.4
      tokens:
        github: 2fe27eb6137e9a18f0f5121d6298
    newrelic: https://download.newrelic.com/
    extensions:
      apcu:
        version: 5.1.17
        enabled: true
      couchbase:
        version: 2.6.0
        enabled: false
      redis:
        version: 4.2.0
        enabled: false
      zip:
        version: 1.15.4
        enabled: true
    packages:
      "hirak/prestissimo": "~0.3.0"
      "nesbot/carbon": "~2.11"
      "sensiolabs/security-checker": "~5.0"
```

1.2. Dockerized Microservices (cont.)

```dockerfile
FROM crowdstar/php-fpm:7.3.2

COPY ./docker/rootfilesystem /
COPY ./ /var/www/

RUN composer install --no-dev -n
```

```dockerfile
FROM crowdstar/php-fpm:7.3.2

COPY ./docker/rootfilesystem /
COPY ./ /var/www/

# @see https://github.com/nodesource/distributions NodeSource No
RUN \
    curl -sL https://deb.nodesource.com/setup_11.x | bash - && \
    apt-get install -y npm                                  && \
    enable-ldap.sh                                          && \
    docker-php-ext-install bcmath pdo_mysql                 && \
    composer install --no-dev -n
```

1.3. Development Tools Integration

- Error Handling, Error Reporting and System Monitoring
  - New Relic
  - Bugsnag
  - Amazon CloudWatch
- Security Check
  - SonarQube
  - SensioLabs Security Checker
- Debugging and Profiling: Blackfire, Xdebug, etc.

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

2.1. Background Processing

- Use cases: error handling, data reporting, outgoing emails, push notifications, etc.
- Common approaches:
  - Execute external program. `exec("nohup ./http.sh > /dev/null &");`
  - Register shutdown functions.
  - Job server (Gearman, Resque, RabbitMQ, etc.).
  - Function *fastcgi_finish_request()* under PHP-FPM.
  - ……

2.1. Background Processing (cont.)

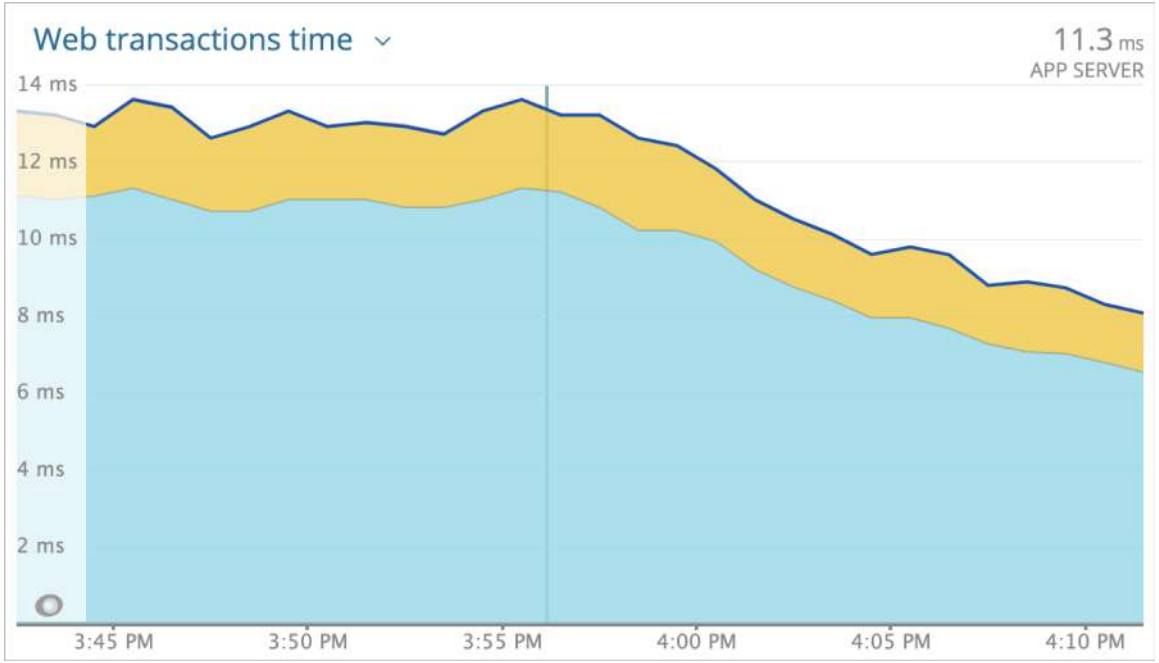
```php
<?php
echo 1;

register_shutdown_function(
    function () {
        echo 3;
    }
);
$a = new class {
    public function __destruct() {
        echo 4;
    }
};

// fastcgi_finish_request();

echo 2;
exit();
echo 5; // Unreachable code.
```

Output: 1234

```php
<?php
echo 1;

register_shutdown_function(
    function () {
        echo 3;
    }
);
$a = new class {
    public function __destruct() {
        echo 4;
    }
};

fastcgi_finish_request();

echo 2;
exit();
echo 5; // Unreachable code.
```

Output: 1

2.1. Background Processing (cont.)



Average Response Time

Without background processing: **13 ms**

With background processing: **9 ms**

2.1. Background Processing (cont.)

How to prevent and address failed operations properly?

- Basic data validation first.
- Exponential backoff.
    - https://github.com/Crowdstar/exponential-backoff
- Proper error reporting.

2.1. Background Processing (cont.)

For more details, please check

- Background Processing: https://github.com/Crowdstar/background-processing
- Technical Discussion: https://github.com/deminy/background-processing-in-php

2.2. HTTP Compression

2.2. HTTP Compression

```
gzip on;
```

2.2. HTTP Compression (cont.)

```
gzip on;
gzip_types text/plain text/css application/json text/xml application/xml application/xml+rss
text/javascript;
```

2.2. HTTP Compression (cont.)

```
gzip on;
gzip_types text/plain text/css application/json text/xml application/xml application/xml+rss
text/javascript;
gzip_proxied any;
```

2.2. HTTP Compression (cont.)

```
gzip on;
gzip_types text/plain text/css application/json text/xml application/xml application/xml+rss
text/javascript;
gzip_proxied any;
gzip_comp_level 5;
gzip_min_length 1280;
```

2.2. HTTP Compression (cont.)

If HTTP header ‘Content-Length’ is not set in a PHP response, Nginx will always compress the response even its length is less than the minimum length.

- Slim 3 always has HTTP header ‘Content-Length’ set by default.
- Laravel and Lumen don’t have HTTP header ‘Content-Length’ set.

2.2. HTTP Compression (cont.)

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Illuminate\Http\Response;

/**
 * By default Lumen won't inject the HTTP response header 'Content-Length'. However, to support HTTP compression (with
 * gzip), HTTP response header 'Content-Length' must be presented. This middleware is to inject the header
 * 'Content-Length'.
 */
class ContentLength
{
    /**
     * Inject HTTP response header 'Content-Length'.
     */
    public function handle(Request $request, Closure $next, $guard = null)
    {
        /** @var Response $response */
        $response = $next($request);

        return $response->header('Content-Length', strlen($response->getContent()));
    }
}
```

A Lumen Middleware to Make HTTP Compression Work as Should

2.3. Dynamic Content Caching

Challenge: feed new data to the client.

Possible Solutions:

- Server push, but it's not supported in HTTP 1 (It's supported under HTTP 2 and the WebSocket API).
- HTTP Clients built on top of HTTP 1 have to hit servers again and again to check and fetch new data.

Our Approach:

- Data cache on the server side if OK.
- Use the ***Last-Modified*** and ***If-Modified-Since*** headers, and return HTTP 304 "***Not Modified***" if no need to retransmit the requested resources.

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

3.1. NoSQL

- Lots of choices: Redis, Couchbase, Aerospike, MongoDB, …
- Our selection and implementation: Redis and Couchbase.
    - Redis.
    - Couchbase.
- Concerns with NoSQL
    - limitations on queries.
    - indexes.
    - serialization and compression.

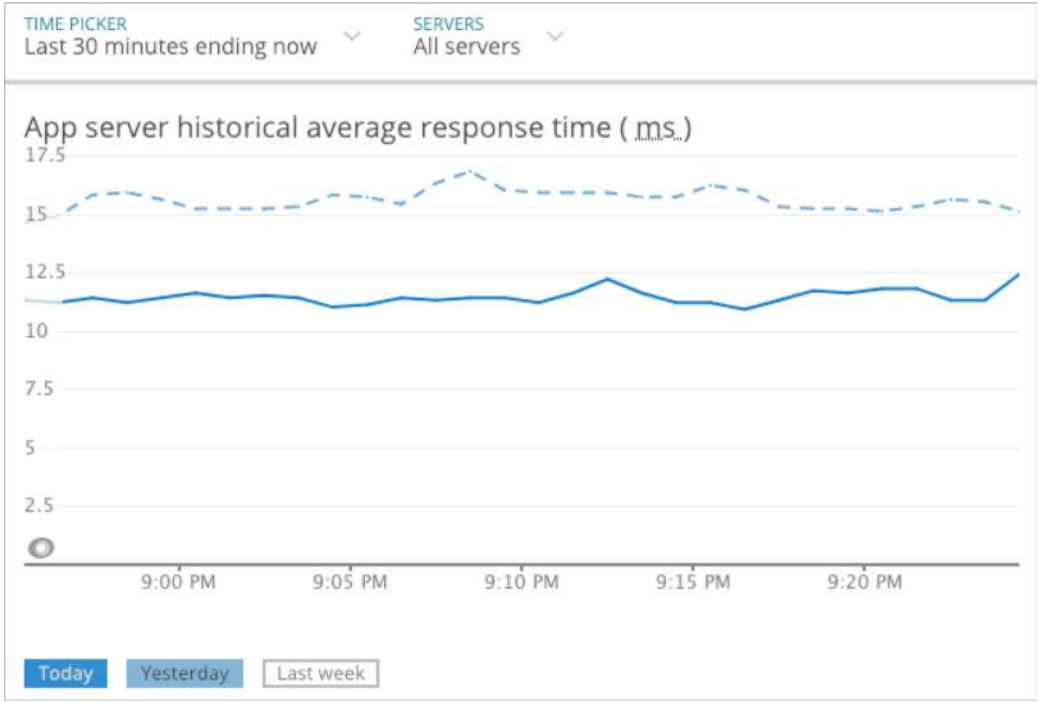
3.2. Fight Against IO and Disk Space

- Data compression to reduce network IO and save disk space.
  - Compress larger messages before saving.
  - Couchbase encoder (serialization).
- Use message templates.
- Shorten field names.
- Add *ttl* (time to live) on certain messages (to expire messages).

3.3. Faster PHP Drivers (Redis)

- **PhpRedis**: the PHP extension for Redis.
- **Predis**: the Redis client written in PHP.
- Other drivers
  - Package **predis/predis-async**: asynchronous (non-blocking) version of Predis built on top of *ReactPHP*.
  - Class **\Swoole\Coroutine\Redis**: the asynchronous Redis Client from *Swoole*.

3.3. Faster PHP Drivers (Redis)



Before and after switching from *Predis* to *PhpRedis*
Average request time reduced from 16 ms to 11 ms

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

4.1. Deployment with Amazon ECS

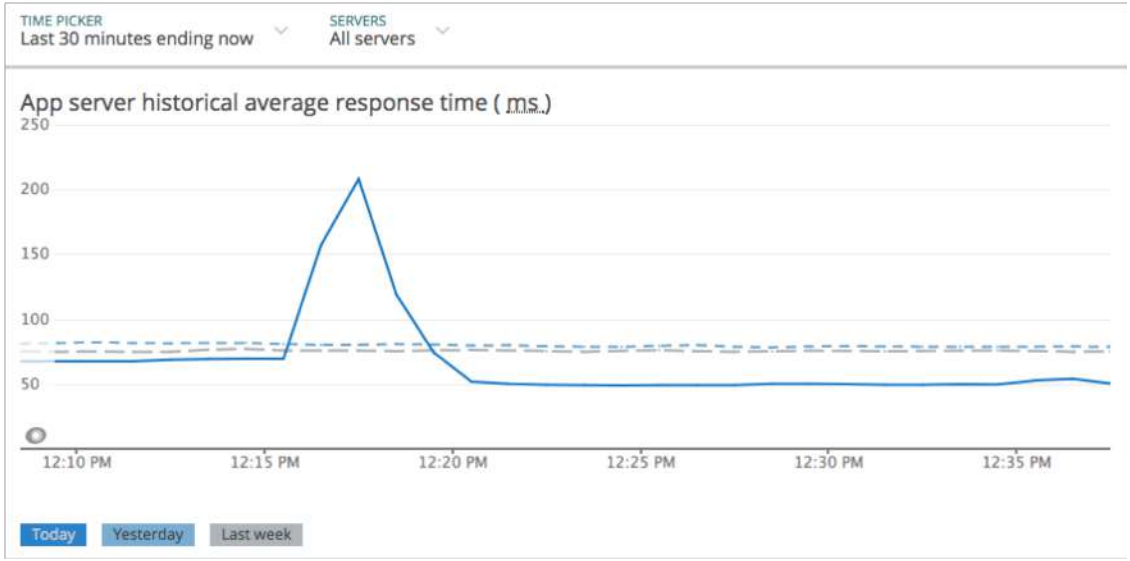
How did we do code deployment before?

4.1. Deployment with Amazon ECS (cont.)

Benefits:

- Zero downtime deployments.
- Quick rollback.
- ELB health checks.
- Autoscaling.

4.2. Hardware Upgrade

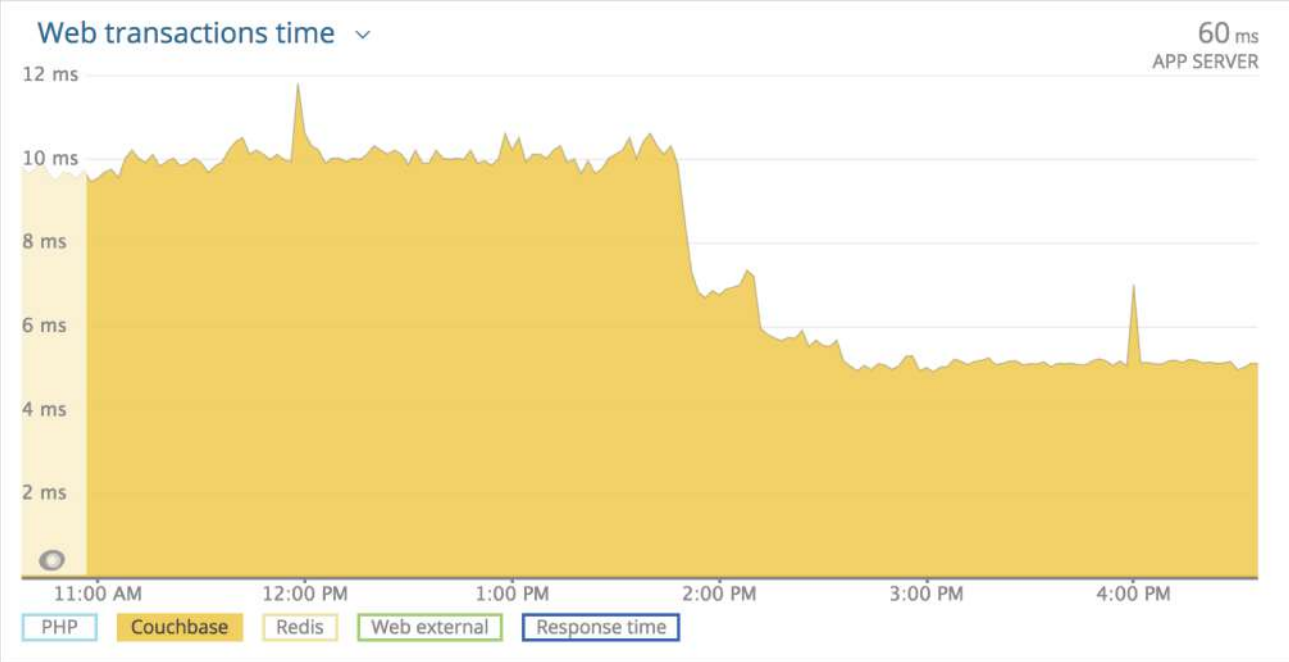


Feb 2018: upgraded API servers with faster ones
(c5.xlarge: 4 vCPU, 8GB memory; Enhanced networking enabled)

4.3. Move Everything into VPC

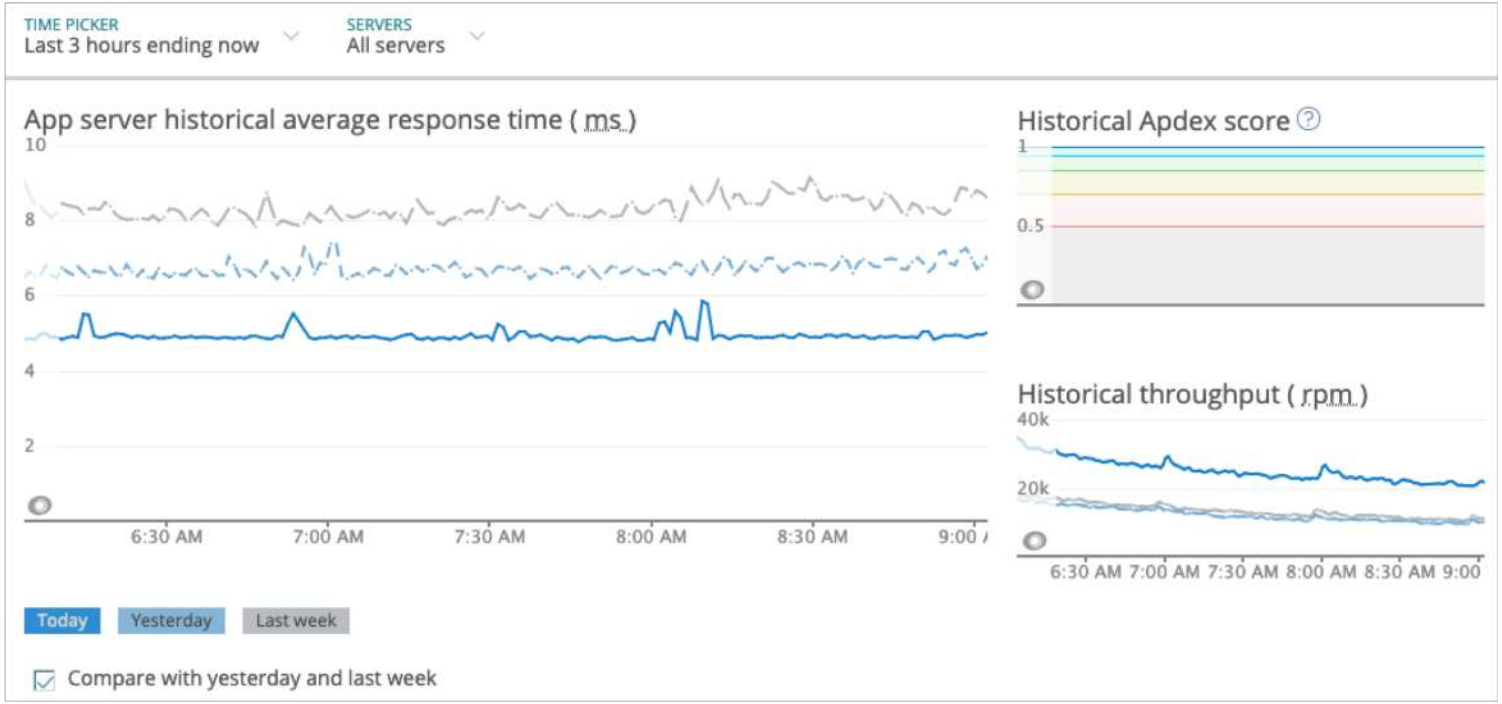
- Place instances closer to each other.
- Move instances to same locations (Availability Zones).
- Use internal network instead of Internet to communicate.
- Use IP addresses directly or use *dnsmasq* (local DNS server).

4.3. Move Everything into VPC (cont.)



Jan 2019: upgraded API servers with faster ones that were closer to the database nodes (same zone)
In the main service, Couchbase operation time decreased to 5 ms

4.3. Move Everything into VPC (cont.)



use IP addresses instead of some domains: response time decreased from 8+ ms to 6+ ms
use *dnsmasq*: *response time decreased from 6+ ms to 5 ms even when traffic doubled*
(Mar 7th, 2019, the inbox microservice)

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

There are always rooms to push the limit of PHP

Next

- REST APIs
  - simdjson (https://github.com/lemire/simdjson)
- Network Protocols
  - HTTP/2
  - Tars (https://github.com/TarsCloud/Tars)
- Asynchronous I/O
  - Swoole (https://github.com/swoole/swoole-src)

Agenda

- **An Overview on Design Home and the Inbox Service**
- **The Inbox Service: Key Improvements Made**
    1. **Web Server**: Dockerized PHP 7 containers
    2. **HTTP Processing**: Background processing, HTTP compression, and more
    3. **Data Storage**: Doing NoSQL with Redis and Couchbase
    4. **Hardware and Network**: Migrating to Amazon Web Services
- **Next**
- **Q&A**

Questions?

slides available at http://deminy.in

Thanks