

Getting started with PHP on AWS Lambda

Thomas Bley, June 2019 @ ipc

About me

- Senior PHP Developer
- Linux, PHP, MySQL since 2001
- studied at TU München
- working for Bringmeister in Berlin



What is AWS Lambda?

- event-driven, serverless computing platform
- a managed service that runs code in response to events
- computing resources are automatically managed (scaling)
- purpose: build smaller, on-demand applications that are responsive to events
- starting a Lambda instance (VM) within 100ms of an event
- now also available for PHP!

sources:

https://en.wikipedia.org/wiki/AWS_Lambda

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>

Serverless $\stackrel{?}{=}$ proprietary CGI



AWS Firecracker (microVM)

- lightweight virtual machine for serverless computing
- higher security level and isolation (uses own kernel, KVM)
- decreases startup time (100ms)
- decreases memory overhead (5MB)
- increases hardware utilization
- designed for short-lived workloads

sources:

<https://firecracker-microvm.github.io/>

https://fosdem.org/2019/schedule/event/containers_firecracker/attachments/slides/3188/export/events/attachments/containers_firecracker/slides/3188/Firecracker_as_a_container_runtime_FOSDEM2019_4_3.pdf

No longer required

- building, shrinking, uploading containers
- nginx, php-fpm
- host or kernel updates
- load balancers
- auto-scaling groups
- blue green deployments
- over-provisioning
- service discovery

Getting started

- compile PHP binary (**bootstrap layer**)
- composer.json, populate vendor folder (**vendor layer**)
- create php files (**application layer**)
- create zip files for php binary, php application, vendor folder
- set up a IAM role in AWS
- setup Lambda layers in AWS (upload **zip files**)
- create Lambda function in AWS (upload **zip file**)
- update Lambda function in AWS (upload **zip file**)

feels like how we deployed 10 years ago?

Compile php binary

```
# Build PHP for Lambda
FROM amazonlinux:2018.03.0.20190514 as builder

RUN sed -i 's;^releasever.*;releasever=2018.03;;' /etc/yum.conf && \
    yum clean all && \
    yum install -y autoconf bison gcc gcc-c++ libcurl-devel libxml2-devel openssl-devel

RUN curl -sL https://github.com/php/php-src/archive/php-7.3.6.tar.gz | tar -xz && \
    cd php-src-php-7.3.6 && \
    ./buildconf --force && \
    ./configure --prefix=/opt/php/ --with-openssl --with-curl --with-mysqli --with-pdo-mysql \
    --with-zlib --enable-mbstring --without-sqlite3 --without-pdo-sqlite --without-pear && \
    make install

# Create runtime container for use with lambdaci
FROM lambci/lambda:provided as runtime

COPY --from=builder /opt/php/bin/php /opt/bin/php

# sources:
# https://hub.docker.com/r/lambci/lambda/
# https://github.com/akrabat/lambda-php/tree/2019-01-31-article/hello-world
```


Compile PHP binary #2

Compile php binary:

```
docker build -t lambda-php-runtime .
```

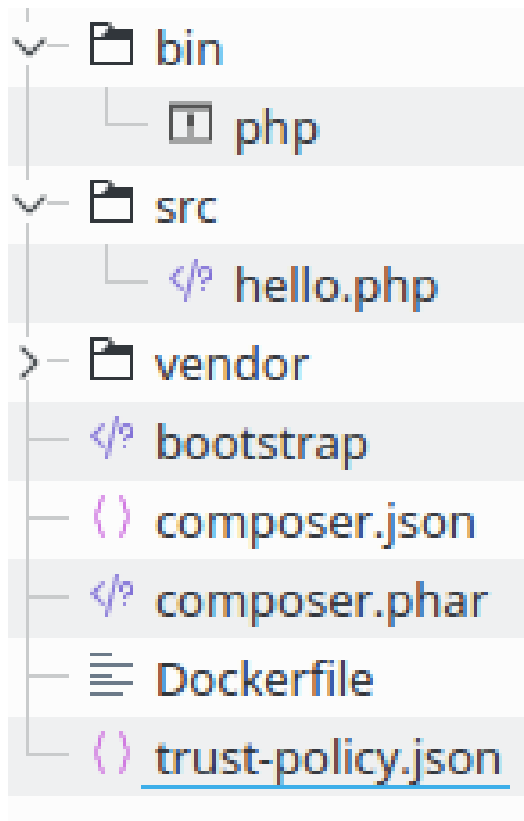
Test php binary:

```
docker run --rm --entrypoint /opt/bin/php lambda-php-runtime -v  
docker run --rm --entrypoint /opt/bin/php lambda-php-runtime -m  
docker run --rm --entrypoint /opt/bin/php lambda-php-runtime -r 'echo 0.7+0.1;'  
docker run --rm --entrypoint /opt/bin/php lambda-php-runtime -r 'echo  
json_encode(0.7+0.1);'
```

Copy php binary from container to host:

```
docker run --rm --entrypoint bash lambda-php-runtime -c "cat /opt/bin/php" >bin/php
```

PHP Lambda Project



hello.php:

```
<?php
function hello($eventData)
{
    print_r($eventData);
    return 'Hello all!';
}
```

PHP Lambda Project #2

bootstrap:

```
#!/opt/bin/php
```

```
<?php
```

```
error_reporting(E_ALL);
```

```
require __DIR__ . '/vendor/autoload.php';
```

```
do {
```

```
    $request = getNextRequest();
```

```
    $handlerFunction = preg_replace('/[^a-zA-Z0-9_]+/', '', $_ENV['_HANDLER']);
```

```
    require_once $_ENV['LAMBDA_TASK_ROOT'] . '/src/' . $handlerFunction . '.php';
```

```
    $response = $handlerFunction($request);
```

```
    sendResponse($request['invocationId'], $response);
```

```
} while (true);
```

PHP Lambda Project #3

Local testing:

```
docker run --rm -v "$PWD":/var/task lambda-php-runtime hello '{"Hello": "ipc}"'
```

START RequestId: 52fdcf07-2182-154f-163f-5f0f9a621d72 Version: \$LATEST

Array(

[invocationId] => 52fdcf07-2182-154f-163f-5f0f9a621d72

[payload] => Array([Hello] => ipc)

)

END RequestId: 52fdcf07-2182-154f-163f-5f0f9a621d72

REPORT RequestId: 52fdcf07-2182-154f-163f-5f0f9a621d72 Init Duration: 9.52 ms
Duration: 2.04 ms **Billed Duration: 100 ms** Memory Size: 1536 MB **Max Memory
Used: 24 MB**

"Hello all!"

Setup AWS Lambda

Create zip files (layers):

```
zip -r runtime.zip bootstrap bin
```

```
zip -r vendor.zip vendor/
```

```
zip hello.zip src/hello.php
```

source:

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>

Setup AWS Lambda: IAM

trust-policy.json:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

source:

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>

Setup AWS Lambda: create layers

```
aws iam create-role \  
  --role-name LambdaPhp --path "/service-role/" \  
  --assume-role-policy-document file:///trust-policy.json
```

```
aws lambda publish-layer-version \  
  --layer-name php-runtime --region eu-central-1 \  
  --zip-file fileb:///runtime.zip
```

```
aws lambda publish-layer-version \  
  --layer-name php-vendor --region eu-central-1 \  
  --zip-file fileb:///vendor.zip
```

source:

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>

Setup AWS Lambda: create function

```
aws lambda create-function --function-name php-hello --handler hello \  
  --runtime provided --region eu-central-1 \  
  --zip-file fileb://hello.zip \  
  --role "arn:aws:iam::<given output from trust-policy upload>" \  
  --layers "arn:aws:lambda:<given output from runtime zip upload>" \  
    "arn:aws:lambda:<given output from vendor zip upload>" \  
    "arn:aws:lambda::awslayer:AmazonLinux1803" \  
  --memory-size 128 --timeout 5
```

source:

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>
<https://aws.amazon.com/blogs/compute/upcoming-updates-to-the-aws-lambda-execution-environment/>

Setup AWS Lambda: deploy function

```
zip hello.zip src/hello.php
```

```
aws lambda update-function-code --function-name php-hello \  
  --zip-file fileb://hello.zip \  
  --region eu-central-1
```

source:

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>

Setup AWS Lambda: execute function

```
aws lambda invoke --function-name php-hello \  
  --region eu-central-1 --log-type Tail --query 'LogResult' \  
  --output text --payload '{"Hello": "ipc"}' \  
  output.txt | base64 --decode
```

```
cat output.txt
```

```
"Hello all!"
```

source:

<https://aws.amazon.com/de/blogs/apn/aws-lambda-custom-runtime-for-php-a-practical-example/>

Problems

- Event loop
 - you need to close your resources manually (!!)
 - you need to take care for memory leaks (!)
- Error handling
- Logging (read-only file system, only /tmp is writable)
- Caching
- Memory limit, maximum execution time
- Globals are not pre-populated (\$_GET, \$_POST, \$_REQUEST, etc.)
- Scaling PHP \neq scaling databases

Error handling

AWS Trigger	Invocation	Failure Behavior
API Gateway	Synchronous	API Gateway returns error to client
Kinesis Streams	Synchronous	Retry until success, blocks stream
Simple Notification Service, S3 Events, Simple Email Service	Asynchronous	Up to 3 retries, possible to configure a dead letter queue (SQS, SNS)
Simple Queue Service	Asynchronous	Event returned to queue, no blocking

source: <https://docs.aws.amazon.com/lambda/latest/dg/retries-on-errors.html>

Problems: Memory Limit

bootstrap:

```
#!/opt/bin/php
```

```
<?php
```

```
error_reporting(E_ALL);
```

```
require __DIR__ . '/vendor/autoload.php';
```

```
do {
```

```
    $request = getNextRequest();
```

```
    $handlerFunction = preg_replace('/[^\^a-zA-Z0-9_]+/', '', $_ENV['_HANDLER']);
```

```
    require_once $_ENV['LAMBDA_TASK_ROOT'] . '/src/' . $handlerFunction . '.php';
```

```
    $response = $handlerFunction($request);
```

```
    sendResponse($request['invocationId'], $response);
```

```
} while (memory_get_peak_usage(true) < 83886080); // 80M
```

Problems: Memory Limit

bootstrap:

```
#!/opt/bin/php
```

```
<?php
```

```
error_reporting(E_ALL);
```

```
require __DIR__ . '/vendor/autoload.php';
```

```
do {
```

```
    $request = getNextRequest();
```

```
    $handlerFunction = preg_replace('/[^\^a-zA-Z0-9_]+/', '', $_ENV['_HANDLER']);
```

```
    require_once $_ENV['LAMBDA_TASK_ROOT'] . '/src/' . $handlerFunction . '.php';
```

```
    $response = $handlerFunction($request);
```

```
    sendResponse($request['invocationId'], $response);
```

```
} while (memory_get_peak_usage(true) < 83886080); // 80M
```

API Gateway handling

```
function hello($eventData)
{
    $responseData = ...;
    $response = [];
    $response["isBase64Encoded"] = false;
    $response["statusCode"] = 200;
    $response["headers"] = ['Content-type' => 'application/json'];
    $response["body"] = json_encode($responseData);
    return $response;
}
```

Limitations

- max. 3 GB RAM
- max. execution time 15 min.
- max. layers: 5
- zip package: 50 MB (zipped), 250 MB (unzipped, incl. layers)
- /tmp storage: 512 MB
- Concurrent executions: 1000 (can be increased)

source: https://docs.aws.amazon.com/en_en/lambda/latest/dg/limits.html

Pricing

- Number of requests

0.20 USD per 1M additional requests (1M requests for free per month)

- Duration of requests (billing rounded to 100ms)

0.00001667 USD per GB-second (400k GB-seconds for free per month)

- Example comparison (duration)

Lambda: 8 x 1 GB = 9.6 USD per day (if continuously running)

EC2: T3 Large 8 GB, 2 vCPU = 2.3 USD per day (on demand)

- Other costs: CloudWatch, API Gateway, Data transfer

sources: <https://www.ec2instances.info> <https://aws.amazon.com/lambda/pricing/>

Lambda Breakeven Analysis for an m4.large Instance

Function Execution Memory & Time	Requests per Hour Required for Lambda Cost to Equal EC2 Cost	Requests per Second
100 ms @ 128 MB	295,000	81.9
200 ms @ 512 MB	64,000	17.8
200 ms @ 1 GB	34,000	9.4
1 sec @ 1 GB	7,100	2.0

So what does this mean? If a typical transaction in your application takes 100 milliseconds to run and uses 128 MB of RAM, your m4.large instance (with 2 vCPU and 8 GB RAM) would need to be running 82 requests per second, every second of every day, before it is more cost effective than running the same workload on Lambda. Could a single m4.large even handle 82 requests per second? Well of course that depends greatly on the workload, but typically that would be quite a lot.

source: <https://www.trek10.com/blog/lambda-cost/>

Use cases

- cronjobs (don't pay for idle)
- websites with unknown traffic spikes
- event based workers (Kinesis, SQS, S3, etc.)
- event based backend tasks (image resizing, creating pdfs, sending emails, etc.)
- integration with database events, e.g. RDS Aurora:

```
CALL lambda_async('arn:aws:lambda:...', '{"operation": "ping"}');
```

```
SELECT lambda_sync('arn:aws:lambda:...', '{"operation": "ping"}');
```

source:

<https://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/AuroraMySQL.Integrating.Lambda.html>

Thanks for listening!

Questions?

slides and sources:
github.com/thomasbley/php-lambda



Further reading

- How to operate reliable AWS Lambda applications
<https://www.concurrencylabs.com/blog/how-to-operate-aws-lambda/>
- PHP JIT
<https://wiki.php.net/rfc/jit>
- Using opcache for php-cli
<https://tideways.com/profiler/blog/dodge-the-thundering-herd-with-file-based-opcache-in-php7>
- Loop mode for php-cli
<https://github.com/mnapoli/php-src/pull/2>