

PHP ON LAMBDA WITH CUSTOM RUNTIMES CONFOO MONTREAL 2020

Ian Littman / [@iansltx](https://twitter.com/iansltx)

Follow along at <https://ian.im/lambfoo20>

QUESTIONS WE'LL ANSWER

- Why you'd use Lambda, and what caveats come with using it
- How Lambda works with Custom Runtimes (including PHP)
- How to build an email reply bot with Lambda, SES, S3, and PHP
- How to build a Lambda-powered API via API Gateway
- Bonus: How to build API Gateway endpoints with [Bref](#)

TOPICS WE WON'T COVER

- A deep dive into Bref or Serverless Framework
- Building Lambda applications via AWS's CLI
- AWS SAM
- Laravel Vapor

WHY LAMBDA?

- No servers to manage
- No worrying about concurrency within an app instance
- Quick automatic scaling + high levels of concurrency
- Very granular billing + fair-sized free tier
- Stateless per-request, shared-nothing-ish
- Integrates with other AWS components

WHAT KINDS OF INTEGRATIONS?

- Trigger events from **S3, SNS, SQS**
- **Application Load Balancer** (billed for capacity + time)
- **API Gateway** (billed per request)

SHARED-NOTHING-ISH?

- AWS keeps Lambda containers around for a bit to avoid "cold start" penalties
- 500 MB in /tmp is available per-instance; can use as a cache
- Instances != invocations
- Instances == concurrency

A NOTE ON PERFORMANCE

- CPU speed scales with RAM allotment
 - Minimum: 128 MB
 - Maximum: 3008 MB
 - Increment size: 64 MB
 - At 1792 MB you have one full core (stop there)
- Using Serverless (e.g. via Bref)? Default is 1024 MB
- There are a few other limits to keep in mind

VIRTUAL PRIVATE CAVEATS

- No internet connectivity by default
- NAT Gateway is \$\$\$ (5¢/hr + 5¢/GB in Canadian region)
- Workaround: split functionality between functions
 - Internet-connected, outside VPC (external APIs)
 - Internet-disconnected, inside VPC (internal resources)
- Cold start times No longer an issue as of late 2019

SO, WHAT ABOUT PHP ON LAMBDA?

- Custom runtime support has been available since 2018
- Build the file system needed to run your function code
- Store the file system as one or more (up to 5) Layers
- Worker-based, long-polling-esque model
- **NOT** HTTP request based without additional abstractions

LAYERS

- Extracted to /opt (function code is in /var/task)
- Versioned
- Can include library dependencies (e.g. /vendor)
- Zipped + submitted to AWS API (like functions)
- Function specifies inclusion order

CUSTOM RUNTIME LIFECYCLE

- Bootstrap (cold start)
- Task execution (warm start)
- Lambda decides when to kill the instance
 - Inactivity timeout (~10 minutes)
 - Task deadline (configurable, up to 15 minutes)

BOOTSTRAP (COLD START) PHASE

- Initialize runtime
- Prep function for execution
- Pass env vars including file/method for handler function
- If there's an error init-ing, [make an API call](#)

TASK EXECUTION (WARM START) PHASE

- Call APIs to process work
- **Next invocation** (GET)
 - Request ID and more in headers
 - Task data in response body, JSON-encoded
- **Invocation response** (POST)
 - Request ID in URL
 - Response in body
 - Consuming services may require specific format

LOGGING & ERRORS

- [Invocation error](#) (POST)
- STDOUT -> CloudWatch Logs
- Metrics available via CloudWatch (or in function UI)

WHAT MIGHT THIS LOOK LIKE FROM AWS'S SIDE?

1. Are any workers for target function waiting on work?
2. If so, skip to step 3; if not...
 1. Build a new instance (layers + function code)
 2. Run `/opt/bootstrap` (or `/var/task/bootstrap`)
 3. Wait for it to do one of the following
 1. Return nonzero from the bootstrap (init failure)
 2. Call the init error API endpoint
 3. Call the task request API endpoint
3. Respond to the next invocation endpoint with event
4. Wait for API calls to invocation success/failure or timeout, whichever comes first



EXAMPLE 1: EMAIL SPONGEBOT

1. SES rule pushes email to S3
2. S3 triggers Lambda
3. Lambda function executes...
 1. Grabs email from S3
 2. Parses using MIME parser
 3. RaNdOMlY CaPiTaLiZeS LEtTerS
 4. Sends email reply via SES

EXAMPLE 2: SPONGEBOT API

1. HTTP request hits API Gateway
2. Lambda function executes...
 1. Grabs query string parameter if there is one
 2. RaNdOMiY CaPiTaLiZeS LEtTerS
 3. Sends a JSON response
3. API Gateway turns the Lambda payload into an HTTP response

...BUT YOU SHOULD PROBABLY USE **BREF** IN PROD

...OR **VAPOR** FOR LARAVEL-ITES

returntrue.win uses Bref

BONUS: EXAMPLE 2 (API GATEWAY) VIA BREF

FURTHER READING

- [An early post on PHP-on-Lambda by Michael Moussa](#)
- [Bref docs](#)
- [Firecracker](#) (underlying "microVM" layer for Lambda)

THANKS! QUESTIONS?

- <https://ian.im/lambfoo20> - this presentation
- <https://github.com/iansltx/spongebot> - sample code
- <https://twitter.com/iansltx> - me
- <https://github.com/iansltx> - my code