

# Perl on Lambda

Exploration of AWS Lambda custom runtimes  
@zjt

# Why Perl on Lambda?

## Perla

- Legacy code hard to replicate in other languages (time)
- Monolith → Microservices
- Because I still love Perl

## Lambda

- [https://en.wikipedia.org/wiki/AWS\\_Lambda](https://en.wikipedia.org/wiki/AWS_Lambda)
- I don't want to bother with running servers or Docker containers
- I have access to AWS and familiar with their PaaS services

**TIMTOWTDI is a universal law of nature,  
ergo it must be possible to run Perl on Lambda  
(Logic!)**

# Lambda example: Battlesnake (as a service)

<https://play.battlesnake.io/>

The players (snakes) are each HTTP servers.

No reason to keep a server running.

Python (naturally)

<https://play.battlesnake.io/g/700d84ae-4cd2-43cc-9fda-824ce8fc5a69/>

<https://us-east-2.console.aws.amazon.com/lambda/home?region=us-east-2#/functions/cloud9-battlesnake-battlesnake-5U9V0NRXHE6R?tab=graph>

<https://us-east-2.console.aws.amazon.com/cloudwatch/home?region=us-east-2#logStream:group=/aws/lambda/cloud9-battlesnake-battlesnake-5U9V0NRXHE6R;streamFilter=typeLogStreamPrefix>

<https://us-east-2.console.aws.amazon.com/cloud9/ide/fe59d87faa444e0eb692038c9274d807>

# Research

I could build my own FaaS (not interested)

- <https://github.com/openfaas/faas>
- <https://openwhisk.apache.org/>

Found this blog:

<https://medium.com/@avijitsarkar123/aws-lambda-custom-runtime-really-works-how-i-developed-a-lambda-in-perl-9a481a7ab465>

Lambda custom runtimes:

<https://docs.aws.amazon.com/lambda/latest/dg/runtimes-walkthrough.html>

# AWS IAM setup

1. Open the [roles page](#) in the IAM console.
2. Choose **Create role**.
3. Create a role with the following properties.
  - **Trusted entity** – **Lambda**.
  - **Permissions** – **AWSLambdaBasicExecutionRole**.
  - **Role name** – **lambda-role**.
4. The **AWSLambdaBasicExecutionRole** policy has the permissions that the function needs to write logs to CloudWatch Logs.

# bootstrap

```
#!/bin/sh
set -euo pipefail
# Initialization - load function handler
source $LAMBDA_TASK_ROOT/"$(echo $_HANDLER | cut -d. -f1).sh"
# Processing
while true
do
    HEADERS="$(mktemp)"
    # Get an event
    EVENT_DATA=$(curl -sS -LD "$HEADERS" -X GET "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invoke/next")
    REQUEST_ID=$(grep -Fi Lambda-Request-Id "$HEADERS" | tr -d '[:space:]' | cut -d: -f2)
    # Execute the handler function from the script
    RESPONSE=$(echo "$_HANDLER" | cut -d. -f2) "$EVENT_DATA")
    # Send the response
    curl -X POST "http://${AWS_LAMBDA_RUNTIME_API}/2018-06-01/runtime/invoke/$REQUEST_ID/response" -d "$RESPONSE"
done
```

# function.sh

```
function handler () {  
    EVENT_DATA=$1  
    echo "$EVENT_DATA" 1>&2;  
    RESPONSE=$(perl -e 'print "hello, perl!. " .join(", ",@ARGV)' "$EVENT_DATA")  
    echo $RESPONSE  
}
```

# Create and run “hello, perl!”

## Create

```
chmod 755 function.sh bootstrap
```

```
zip function.zip function.sh bootstrap
```

```
aws lambda create-function \  
  --function-name bash-runtime \  
  --zip-file fileb://function.zip \  
  --handler function.handler \  
  --runtime provided \  
  --role arn:aws:iam::123456789012:role/lambda-role
```

## Run

```
aws lambda invoke \  
  --function-name bash-runtime \  
  --payload '{"text":"Hello"}' response.txt
```

```
cat response.txt
```



# Update the function only - using layers

```
zip runtime.zip bootstrap
```

```
aws lambda publish-layer-version \  
  --layer-name bash-runtime \  
  --zip-file fileb://runtime.zip
```

```
aws lambda update-function-configuration \  
  --function-name bash-runtime \  
  --layers arn:aws:lambda:us-east-2:123456789012:layer:bash-runtime:1
```

```
zip function-only.zip function.sh
```

```
aws lambda update-function-code \  
  --function-name bash-runtime \  
  --zip-file fileb://function-only.zip
```

# Cleaning up

Delete the layers

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 1
```

```
aws lambda delete-layer-version --layer-name bash-runtime --version-number 2
```

Delete the function

```
aws lambda delete-function --function-name bash-runtime
```

# Parsing the event - Needs JSON.pm

Need to install JSON perl library to parse request/response.

First, we need to download the dependencies

<https://metacpan.org/pod/Carton>

Install Carton to **local lib**.

```
perl -MCPAN -Mlocal::lib -e 'CPAN::install(Carton)'
```

Define the dependencies in **cpanfile**

```
requires 'JSON';
```

Use **carton** to grab the dependencies.

Using local lib here because carton was installed in local lib.

```
perl -Mlocal::lib ../perl5/bin/carton install
```

On my build environment...

CPAN isn't even installed.

Perl on AWS Linux is bare bones.

```
sudo yum install perl-CPAN
```

Why I use Cloud9?

- Seems logical to build on AWS Linux so that Lambda runtime server is most similar
- Cloud9 is convenient/awesome, since I use ChromeOS

# Update function to use bundled module

Let's update the function to use JSON from the local lib. Also included perl version in output.

```
RESPONSE=$(perl -Ilocal/lib/perl5 -MJSON -MData::Dumper  
-e 'print "hello, perl!. $]\n".Dumper(decode_json($ARGV[0]))' "$EVENT_DATA")
```

Now all of JSON dependencies are in the local dir, and the function is updated to load a module from it..

First, let's test our code so that we can catch errors before redeploying. **test.sh**

```
#!/bin/bash  
source function.sh  
handler $1  
  
$ ./test.sh '{"text":"Hello"}'  
{"text":"Hello"}  
hello, perl!. 5.016003 $VAR1 = { 'text' => 'Hello' };
```

In my testing I determined that I needed to also use Data::Dumper so that I could confirm that the JSON decoded into a perl hash

# Bundle and invoke again

Update the zip to bundle both, and invoke again.

This time deploy and invoke as a script: `deploy_and_invoke.sh`

```
zip -r function-only.zip function.sh local
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://function-only.zip
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}' response.txt
cat response.txt
```

Fail. But why?

```
{
  "FunctionError": "Unhandled",
  "ExecutedVersion": "$LATEST",
  "StatusCode": 200
}
```

```
{"errorType": "Runtime.ExitError",
"errorMessage": "RequestId: f24c1edc-c2ae-48b6-8dcb-2a4cbd4ae9a4
Error: Runtime exited with error: exit status 2"}
```

▶	00:40:25	START RequestId: f24c1edc-c2ae-48b6-8dcb-2a4cbd4ae9a4 Ver
▶	00:40:25	{"text":"Hello"}
▶	00:40:25	Can't locate Data/Dumper.pm in @INC (@INC contains: local/lib/
▶	00:40:25	BEGIN failed--compilation aborted.
▶	00:40:25	END RequestId: f24c1edc-c2ae-48b6-8dcb-2a4cbd4ae9a4
▶	00:40:25	REPORT RequestId: f24c1edc-c2ae-48b6-8dcb-2a4cbd4ae9a4 Ver

Oops! I included `Data::Dumper` without bundling it with Carton. Cloudwatch screenshot.

# Problem! Core Perl modules aren't installed

1. Update cpanfile with Data::Dumper
2. Carton install
3. Doesn't download core modules



miyagawa commented on Jul 25, 2013

In general, Carton doesn't install core modules.

Is the perl on Cloud9 EC2 different than what's on the lambda instance?

```
ec2-user:~/environment $ perl -v  
This is perl 5, version 16, subversion 3
```



miyagawa commented on Jul 26, 2013

Are you using the same version of perl on your development and build server? These are not dependencies of cpanm (they are core modules) and should be available on both machines.

Currently carton doesn't support different versions of perl (and architecture specific modules) for local and deployment.

Inside the lambda

```
hello, perl!. 5.016003{"text":"Hello"
```

Same version.

Carton only works with perl installation with the complete set of **core** modules. If you use perl installed by a vendor package with modules stripped from **core**, Carton is not expected to work correctly.

# So why not just avoid core modules?

Switch to Data::Printer

```
$ cat cpanfile
requires 'Data::Printer';
requires 'JSON';
```

```
$ perl -Mlocal::lib ../perl5/bin/carton install
```

```
$ cat function.sh
...
RESPONSE=$(perl -Ilocal/lib/perl5 -MJSON -MData::Printer -e 'p(decode_json($ARGV[0]))' "$EVENT_DATA")
...
```

But, I don't think I can always avoid core module dependencies!

# How do I get core modules down to local lib?

cpanm installs it OK but Carton still doesn't pick up core modules

```
$ perl -MCPAN -Mlocal::lib -e 'CPAN::install(Data::Dumper)'
```

```
$ find .. | grep Data/Dumper.pm | grep -v .cpan  
../perl5/lib/perl5/x86_64-linux-thread-multi/Data/Dumper.pm
```

```
$ perl -Mlocal::lib ../perl5/bin/carton install  
Installing modules using /home/ec2-user/environment/cpanfile  
Complete! Modules were installed into /home/ec2-user/environment/local
```

```
$ find .. | grep Data/Dumper.pm | grep -v .cpan  
../perl5/lib/perl5/x86_64-linux-thread-multi/Data/Dumper.pm
```

```
$ find .. | grep Data/Printer.pm | grep -v .cpan  
../environment/local/lib/perl5/Data/Printer.pm
```



# What about App::FatPacker?

```
$ cat hello.pl
```

```
#!/usr/bin/env perl
```

```
use Data::Dumper;
```

```
use Data::Printer;
```

```
$ perl -MCPAN -Mlocal::lib -e 'CPAN::install(App::FatPacker)'
```

```
$ perl -Mlocal::lib -Ilocal/lib/perl5 ../perl5/bin/fatpack trace hello.pl
```

```
Can't locate Data/Printer.pm in @INC (@INC contains: /home/ec2-user/perl5/lib/perl5/5.16.3/x86_64-linux-thread-multi /home/ec2-user/perl5/lib/perl5/5.16.3
```

```
/home/ec2-user/perl5/lib/perl5/x86_64-linux-thread-multi /home/ec2-user/perl5/lib/perl5 /usr/local/lib64/perl5 /usr/local/share/perl5
```

```
/usr/lib64/perl5/vendor_perl /usr/share/perl5/vendor_perl /usr/lib64/perl5 /usr/share/perl5 .) at hello.pl line 3.
```

```
BEGIN failed--compilation aborted at hello.pl line 3.
```

```
$ export PERL5OPT="-Ilocal/lib/perl5"
```

```
ec2-user:~/environment $ perl -Mlocal::lib ../perl5/bin/fatpack packlists-for `cat fatpacker.trace`
```

```
...
```

```
local/lib/perl5/x86_64-linux-thread-multi/auto/Data/Printer/.packlist
```

```
...
```

```
(No Data::Dumper packlist)
```

# The fatpacker docs solved the cpanm issue

<https://metacpan.org/pod/App::FatPacker::Simple::Tutorial>

```
# Oh, HTTP::Tiny is not core module for old perls, so we have to fatpack it too!
```

```
$ cpanm --reinstall -Llocal -nq HTTP::Tiny
```

```
$ perl -Mlocal::lib ../perl5/bin/cpanm --reinstall -Llocal -nq Data::Dumper
```

```
Successfully installed Data-Dumper-2.173 (upgraded from 2.135_06)
```

```
1 distribution installed
```

Not sure if the PERL5OPT env var is still necessary, but it doesn't hurt

Fatpacker still might be a good way to package Perl on Lambda though...

# Now we're cookin'

```
$ perl -Mlocal::lib ../perl5/bin/fatpack pack hello.pl >hello.packed.pl
```

```
hello.pl syntax OK
```

```
File /home/ec2-user/environment/fatlib/x86_64-linux-thread-multi/auto/Data/Dumper/Dumper.so isn't a .pm file - can't pack this -- if you hoped we were going to, things may not be what you expected later
```

```
File /home/ec2-user/environment/fatlib/x86_64-linux-thread-multi/auto/Data/Dumper/Dumper.bs isn't a .pm file - can't pack this -- if you hoped we were going to, things may not be what you expected later
```

```
$ grep -c Data::Dumper hello.packed.pl
```

```
75
```

# Putting it all together - carton, cpanm, fatpacker

```
$ cat hello.pl
#!/usr/bin/env perl
use Data::Dumper;
use JSON;
print Dumper(decode_json($ARGV[0])
```

```
$ cat function.sh
...
RESPONSE=$(perl -Ilocal/lib/perl5 hello.packed.pl "$EVENT_DATA")
...
```

```
$ cat deploy_and_invoke.sh
#!/bin/sh
export PERL5OPT="-Ilocal/lib/perl5"
perl -Mlocal::lib ../perl5/bin/carton install
perl -Mlocal::lib ../perl5/bin/cpanm --reinstall -Llocal -nq Data::Dumper # TODO move to cpanfile?
perl -Mlocal::lib ../perl5/bin/fatpack pack hello.pl >hello.packed.pl
zip -r function-only.zip function.sh hello.packed.pl local
aws lambda update-function-code --function-name bash-runtime --zip-file fileb://function-only.zip
aws lambda invoke --function-name bash-runtime --payload '{"text":"Hello"}' response.txt
cat response.txt
```

# Pushing it to the limit

Moose (add to cpanfile and hello.pl)

```
{"errorMessage":"2019-05-14T03:34:44.940Z 11921be2-dd0f-45e7-a0b0-c35006be6398 Task timed out after 3.00 seconds"}
```

- Dancer doesn't time out (for comparison)

XS modules don't work because the libraries aren't installed

- You could bundle the C libraries locally in the runtime, in theory. But my years of struggling with Perl on Solaris taught me that there are a ton of Perl modules that make assumptions about where libraries are installed

▼ 03:43:17

Can't load 'local/lib/perl5/x86\_64-linux-thread-multi/auto/Net/SSLeay/SSLeay.so' for module Net::SSLeay: /lib64/libcrypto.so.1

Can't load 'local/lib/perl5/x86\_64-linux-thread-multi/auto/Net/SSLeay/SSLeay.so' for module Net::SSLeay: /lib64/libcrypto.so.10: version linux-thread-multi/auto/Net/SSLeay/SSLeay.so) at /usr/lib64/perl5/DynaLoader.pm line 190.

# What else?

## Ship Perl in the Lambda runtime?

- Yes, this would work, and it would make the dependency issue easier as well as giving you better versions of Perl to use.
- The aforementioned blog actually did this using Activeperl.
- Would perlbrew work better?
- How does this affect the startup performance and cost (CPU/memory overhead)?
- Future research TBD

## Docker?

- I tend to dockerize all my apps nowadays, so this would be super awesome
- In theory...

On the build server:

```
docker pull perl
```

```
docker save perl > docker-perl.tar
```

In the Lambda runtime

```
docker load docker-perl.tar
```

```
docker run ...
```

- But... It turns out that docker isn't installed on the Lambda server (boo)

# What's next

A PSGI shim so that all of the pure-Perl Plack modules on CPAN work - would be awesome

Experiment with bundling perlbrew in the Lambda runtime - determine if there is a penalty

Rewrite battlesnake in Perl - because why not

# Perlbrew

```
$ perl -MCPAN -Mlocal::lib -e 'CPAN::install(App::perlbrew)'
```

(or)

```
$ cpanm -llocal -nq App::perlbrew
```

```
$ perl -Mlocal::lib ../perl5/bin/perlbrew init
```

```
$ perl -Mlocal::lib ../perl5/bin/perlbrew install-patchperl
```

```
$ perl -Mlocal::lib ../perl5/bin/perlbrew install perl-5.28.2
```