The Design of The C++ Runtime For AWS Lambda

CppCon 2019

Marco Magdy

Senior Software Engineer

Disclaimer

- This talk is mini deep dive on how C++ runs on Lambda
- This talk is NOT a tutorial or a walkthrough on using the C++ runtime for Lambda
- This talk is NOT a sales pitch for AWS Lambda

Overview

1. **What is AWS Lambda?**
2. A bit of history
3. What is the C++ runtime for Lambda?
4. What was particularly hard about running C++ in Lambda?
5. Advantages of using C++ with AWS Lambda

What is AWS Lambda?

- A service that lets you run code without provisioning or managing servers.
- You pay only for the compute time you consume.
- Lambda takes care of everything required to run and scale your code with high availability.
- You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.
- The code runs on x86 64-bit Amazon Linux machine

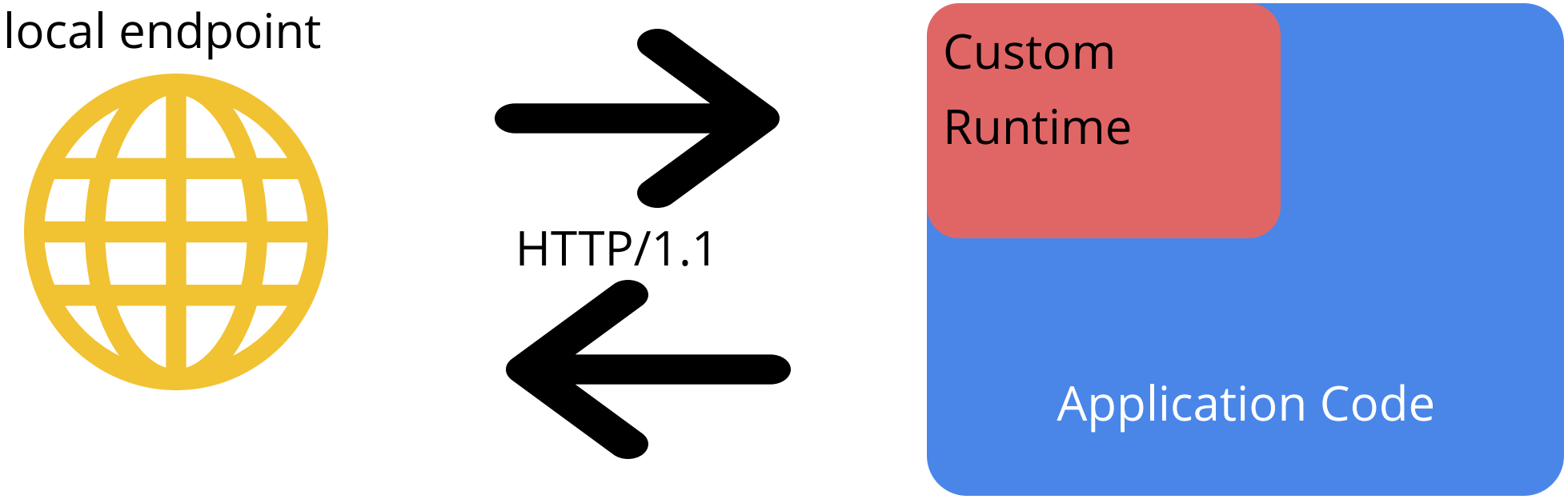
Overview

1. What is AWS Lambda?
2. **A bit of history**
3. What is the C++ runtime for Lambda?
4. What was particularly hard about running C++ in Lambda?
5. Advantages of using C++ with AWS Lambda

A Bit of History

- Lambda supports running code written in JavaScript, .NET, Go, Python, Java, and *custom-runtimes*
- Custom runtimes support was added in 2018, which made it possible to write Lambda functions in any language.
- If your language speaks HTTP, then you can interface with Lambda.
- A GET HTTP request to get the details of an incoming invocation
- A POST HTTP request to submit the results back to Lambda

Lambda Host



Overview

1. What is AWS Lambda?
2. A bit of history
3. **What is the C++ runtime for Lambda?**
4. What was particularly hard about running C++ in Lambda?
5. Advantages of using C++ with AWS Lambda

Hello World

```
1  #include <aws/lambda-runtime/runtime.h>
2
3  using namespace aws;
4
5  lambda_runtime::invocation_response my_handler(lambda_runtime::invocation_request const& req)
6  {
7    if (req.payload == "Hi") {
8      return lambda_runtime::invocation_response::success("Hello World"      /*payload*/,
9                                                          "application/text" /*content-type*/);
10   }
11
12   return lambda_runtime::invocation_response::failure("Something Went wrong" /*error message*/,
13                                                       "BadRequest"           /*error type*/);
14 }
15
16
17 int main()
18 {
19   lambda_runtime::run_handler(my_handler);
20   return 0;
21 }
```

Hello World

```
1  #include <aws/lambda-runtime/runtime.h>
2
3  using namespace aws;
4
5  lambda_runtime::invocation_response my_handler(lambda_runtime::invocation_request const& req)
6  {
7    if (req.payload == "Hi") {
8      return lambda_runtime::invocation_response::success("Hello World"      /*payload*/,
9                                                          "application/text" /*content-type*/);
10   }
11
12   return lambda_runtime::invocation_response::failure("Something Went wrong" /*error message*/,
13                                                       "BadRequest"           /*error type*/);
14 }
15
16
17 int main()
18 {
19   lambda_runtime::run_handler(my_handler);
20   return 0;
21 }
```

Hello World

```
1  #include <aws/lambda-runtime/runtime.h>
2
3  using namespace aws;
4
5  lambda_runtime::invocation_response my_handler(lambda_runtime::invocation_request const& req)
6  {
7    if (req.payload == "Hi") {
8      return lambda_runtime::invocation_response::success("Hello World"      /*payload*/,
9                                                          "application/text" /*content-type*/);
10   }
11
12   return lambda_runtime::invocation_response::failure("Something Went wrong" /*error message*/,
13                                                       "BadRequest"           /*error type*/);
14 }
15
16
17 int main()
18 {
19   lambda_runtime::run_handler(my_handler);
20   return 0;
21 }
```

Hello World

```
1  #include <aws/lambda-runtime/runtime.h>
2
3  using namespace aws;
4
5  lambda_runtime::invocation_response my_handler(lambda_runtime::invocation_request const& req)
6  {
7    if (req.payload == "Hi") {
8      return lambda_runtime::invocation_response::success("Hello World"      /*payload*/,
9                                                          "application/text" /*content-type*/);
10   }
11
12   return lambda_runtime::invocation_response::failure("Something Went wrong" /*error message*/,
13                                                       "BadRequest"           /*error type*/);
14 }
15
16
17 int main()
18 {
19   lambda_runtime::run_handler(my_handler);
20   return 0;
21 }
```

Hello World

```
1  #include <aws/lambda-runtime/runtime.h>
2
3  using namespace aws;
4
5  lambda_runtime::invocation_response my_handler(lambda_runtime::invocation_request const& req)
6  {
7    if (req.payload == "Hi") {
8      return lambda_runtime::invocation_response::success("Hello World"      /*payload*/,
9                                                          "application/text" /*content-type*/);
10   }
11
12   return lambda_runtime::invocation_response::failure("Something Went wrong" /*error message*/,
13                                                       "BadRequest"           /*error type*/);
14 }
15
16
17 int main()
18 {
19   lambda_runtime::run_handler(my_handler);
20   return 0;
21 }
```

- The user code package is a zip file that looks like this

```
$ tree my-package

|-- bootstrap
|-- bin
|   -- demo-app
|-- lib
|   |-- liby.so
|   |-- libz.so
 ...
```

- Lambda executes the "bootstrap" file

Overview

1. What is AWS Lambda?
2. A bit of history
3. What is the C++ runtime for Lambda?
4. **What was particularly hard about running C++ in Lambda?**
5. Advantages of using C++ with AWS Lambda

Dependencies

Any C++ application has the following dependencies:

Dependencies

Any C++ application has the following dependencies:

- Application specific dependencies

Dependencies

Any C++ application has the following dependencies:

- Application specific dependencies
- The C++ runtime (libstdc++ or libc++)

Dependencies

Any C++ application has the following dependencies:

- Application specific dependencies
- The C++ runtime (libstdc++ or libc++)
- The C runtime (GNU libc, must libc, etc.)

System Dependencies

Development Machine	Production Machine
Ubuntu 18.04	Amazon Linux 2017.03
GNU Lib C v2.27	GNU Lib C v2.17

System Dependencies

Development Machine	Production Machine
Ubuntu 18.04	Amazon Linux 2017.03
GNU Lib C v2.27	GNU Lib C v2.17

MyApp: /lib64/libc.so.6: version `GLIBC_2.18' not found (required by libstdc++.so.6)

MyApp: /lib64/libc.so.6: version `GLIBC_2.25' not found (required by libcrypto.so.1.1)

Application Dependencies

Application Dependencies

```
-Wl, rpath /usr/lib/x86_64-linux-gnu/libxyz.so
```

If they don't match between the build and host machine:

Application Dependencies

```
-Wl, rpath /usr/lib/x86_64-linux-gnu/libxyz.so
```

If they don't match between the build and host machine:

error while loading shared libraries: libxyz.so: cannot open shared object file: No such file or directory

Two Problems To Solve

1. Package the dependencies reliably
2. Resolve the conflict between the build machine system libraries and the host machine system libraries.

Solutions That Did NOT Work

Solution #1

1. Run `ldd`

2. Copy the files with their respective paths in the zip file

3. Unzip them on the Lambda host and overlay the files on the existing directories

```
$ tree my-package

|-- demo-app
├── usr
│   └── lib
│       ├── libxyz.so
│       ├── libabc.so
 ...
```

Solutions That Did NOT Work

Solution #2

1. Run `ldd`
2. Add the files to the zip file under a single directory
3. Unzip the files on the Lambda host
4. Set `LD_LIBRARY_PATH` to that directory
5. Execute the binary

Solutions That Did NOT Work

Solution #3

1. Run `ldd`
2. Add the files with their respective paths in the zip file
3. Unzip the files in a directory on the Lambda host
4. `chroot` that directory
5. Execute the binary

Solutions That Did NOT Work

Solution #3

Name

chroot - run command or interactive shell with special root directory

Synopsis

```
chroot [OPTION] NEWROOT [COMMAND [ARG]...]
```

Solutions That Did NOT Work

Solution #3

```
$ tree my-package

|-- demo-app
├── usr
|   └── lib
|       ├── libxyz.so
|       ├── libabc.so
...

$ sudo chroot my-package my-package/demo-app
```

Solutions That Did NOT Work

Solution #4

1. Run `ldd`
2. Add the files to the zip file under a single directory
3. Unzip the files in a directory on the Lambda host
4. Set `LD_LIBRARY_PATH` to that directory
5. Set `LD_PRELOAD` to the packaged libc and libstdc++
6. Execute the binary

Solution That ALMOST Worked

Solution #5

1. Run `ldd`
2. Add the files to the zip file under a single directory
3. Unzip the files in a directory on the Lambda host
4. Execute the binary using `ld-linux` from the user's package and constrain its search path with `--library-path`

```
MyPackage/lib/ld-linux.so --library-path MyPackage/lib MyPackage/bin/MyApp
```

Solution That ALMOST Worked

Solution #5

Name

ld-linux.so - dynamic linker/loader

Synopsis

`/lib/ld-linux.so` *`[OPTIONS] [PROGRAM [ARGUMENTS]`*

Options

`--library-path` *`PATH`*

`Use` *`PATH`* `instead of LD_LIBRARY_PATH environment variable setting`

Solutions That DID Work

Solution #6 - Winner

1. Run ldd
2. **Query the Linux distro package manager for libc's list of files**
3. Add both sets of files to the zip file under a single directory
4. Unzip the files in a directory on the Lambda host
5. Execute the binary using ld-linux

Solutions That DID Work

Solution #6 - Winner

```
#!/bin/bash

set -euo pipefail

export AWS_EXECUTION_ENV=lambda-cpp

exec $LAMBDA_TASK_ROOT/lib/ld-linux-x86-64.so.2 --library-path $LAMBDA_TASK_ROOT/lib $LAMBDA_TASK_ROOT/bin/MyApp
```

3 Working Solutions

1. Query the package manager for libc's list of files and execute the binary via the dynamic loader
2. Build on Amazon Linux and set `LD_LIBRARY_PATH`
3. Use musl libc and set `LD_LIBRARY_PATH`

Known Problems & Limitations

- Dynamically loaded dependencies via dlopen require aren't packaged
- TLS certificates live under different paths between Linux distributions
- Data files are not automatically packaged
- Packaging the entirety of GNU Lib C creates large zip files
- Debugging is challenging

Why would I use this?

- $$$ - Time is money on AWS Lambda & Faster == Cheaper
- SIMD is easy(ier) in C/C++
- Brings C++ to the web in a secure, safe way

The C++ Lambda Runtime

- Open Source - github.com/awslabs/aws-lambda-cpp
- It uses libcurl to speak HTTP to Lambda's host (only HTTP is required)
- It requires a C++11 (or later) to build
- The user code can be in any version of C++ or C
- You can use your favorite Linux distro to build your code

Questions?