
AWS
re:Invent

SVS205

Serverless, the Swift way

Nicki Stone

Senior Software Development Engineer, AWS Amplify

AWS

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Agenda

What is the new Swift AWS Lambda Runtime?

How do I use it?

One cool demo!



AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is it?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Swift AWS Lambda Runtime

Distributed as a Swift package, Swift AWS Lambda Runtime is designed to help Swift developers build serverless functions for the Lambda platform

Custom Lambda runtimes

```
~/Projects/Reinvent2020/SwiftLambdaDemo/.build/lambda/SwiftLambdaDemo
$ zipinfo lambda.zip
Archive:  lambda.zip
Zip file size: 21762576 bytes, number of entries: 11
-rwxr-xr-x  3.0 unx  9610216 bx defN 20-Oct-30 11:11 SwiftLambdaDemo
lrwxr-xr-x  3.0 unx       15 bx stor 20-Oct-30 11:11 bootstrap
-rwxr-xr-x  3.0 unx    13360 bx defN 20-Oct-30 11:11 libBlocksRuntime.so
-rwxr-xr-x  3.0 unx 11464040 bx defN 20-Oct-30 11:11 libFoundation.so
-rwxr-xr-x  3.0 unx   442064 bx defN 20-Oct-30 11:11 libdispatch.so
-rwxr-xr-x  3.0 unx 27980992 bx defN 20-Oct-30 11:11 libicudataswift.so.65
-rwxr-xr-x  3.0 unx  4187840 bx defN 20-Oct-30 11:11 libicui18nswift.so.65
-rwxr-xr-x  3.0 unx  2480824 bx defN 20-Oct-30 11:11 libicuucswift.so.65
-rwxr-xr-x  3.0 unx  7834064 bx defN 20-Oct-30 11:11 libswiftCore.so
-rwxr-xr-x  3.0 unx   338232 bx defN 20-Oct-30 11:11 libswiftDispatch.so
-rwxr-xr-x  3.0 unx   110136 bx defN 20-Oct-30 11:11 libswiftGlibc.so
11 files, 64461783 bytes uncompressed, 21760778 bytes compressed:  66.2%
```

How does it work?



© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.

How does it work?

```
1   import AWSLambdaRuntime
2
3   struct Input: Codable {
4     let number: Double
5   }
6
7   struct Output: Codable {
8     let result: Double
9   }
10
11  Lambda.run { (context, input: Input, callback) in
12    callback(.success(Output(result: input.number * input.number)))
13  }
14
```

What about events?

```
// Import the modules
import AWSLambdaRuntime
import AWSLambdaEvents

// In this example we are receiving a SQS Message, with no response (Void)
Lambda.run { (context, message: SQS.Message, callback) in
  ...
  callback(.success(Void()))
}
```

Yep, even API Gateway events

```
// Import the modules
import AWSLambdaRuntime
import AWSLambdaEvents

// In this example we are receiving an APIGateway.V2.Request,
// and respoding with APIGateway.V2.Response
Lambda.run { (context, request: APIGateway.V2.Request, callback) in
    ...
    callback(.success(APIGateway.V2.Response(statusCode: .accepted)))
}
```

What about performance?

```
// Import the modules
import AWSLambdaRuntime
import AWSLambdaEvents
import NIO

// Our Lambda handler, conforms to EventLoopLambdaHandler
struct Handler: EventLoopLambdaHandler {
    typealias In = SNS.Message // Request type
    typealias Out = Void // Response type, or Void

    // In this example we are receiving a SNS Message, with no response (Void)
    func handle(context: Lambda.Context, payload: In) -> EventLoopFuture<Out> {
        ...
        context.eventLoop.makeSucceededFuture(Void())
    }
}

Lambda.run(Handler())
```

What is under the hood?

```
public protocol LambdaHandler: EventLoopLambdaHandler {
    /// Defines to which `DispatchQueue` the Lambda execution is offloaded to.
    var offloadQueue: DispatchQueue { get }

    /// The Lambda handling method
    /// Concrete Lambda handlers implement this method to provide the Lambda functionality.
    ///
    /// - parameters:
    ///   - context: Runtime `Context`.
    ///   - event: Event of type `In` representing the event or request.
    ///   - callback: Completion handler to report the result of the Lambda back to the runtime engine.
    ///   The completion handler expects a `Result` with either a response of type `Out` or an `Error`
    func handle(context: Lambda.Context, event: In, callback: @escaping (Result<Out,
        Error>) -> Void)
}
```

Event loop Lambda handler

```
public protocol EventLoopLambdaHandler: ByteBufferLambdaHandler {
    associatedtype In
    associatedtype Out

    func handle(context: Lambda.Context, event: In) -> EventLoopFuture<Out>

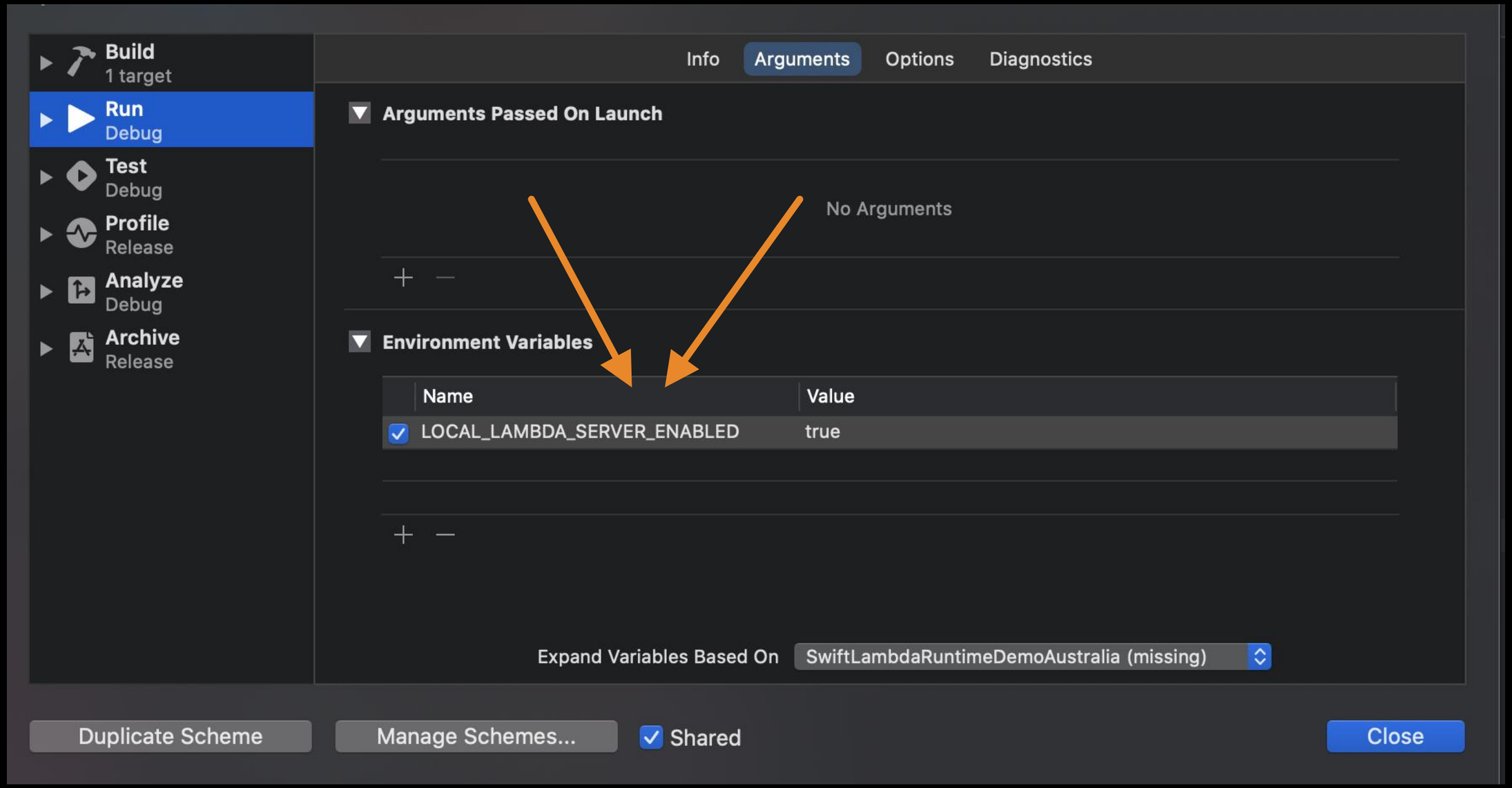
    /// Encode a response of type `Out` to `ByteBuffer`
    /// Concrete Lambda handlers implement this method to provide coding functionality.
    /// - parameters:
    ///  - allocator: A `ByteBufferAllocator` to help allocate the `ByteBuffer`.
    ///  - value: Response of type `Out`.
    ///
    /// - Returns: A `ByteBuffer` with the encoded version of the `value`.
    func encode(allocator: ByteBufferAllocator, value: Out) throws -> ByteBuffer?

    /// Decode a`ByteBuffer` to a request or event of type `In`
    /// Concrete Lambda handlers implement this method to provide coding functionality.
    ///
    /// - parameters:
    ///  - buffer: The `ByteBuffer` to decode.
    ///
    /// - Returns: A request or event of type `In`.
    func decode(buffer: ByteBuffer) throws -> In
}
```

Byte buffer Lambda handler

```
public protocol ByteBufferLambdaHandler {
    /// The Lambda handling method
    /// Concrete Lambda handlers implement this method to provide the Lambda functionality.
    ///
    /// - parameters:
    ///   - context: Runtime `Context`.
    ///   - event: The event or request payload encoded as `ByteBuffer`.
    ///
    /// - Returns: An `EventLoopFuture` to report the result of the Lambda back to the runtime engine.
    /// The `EventLoopFuture` should be completed with either a response encoded as `ByteBuffer` or an `Error`
    func handle(context: Lambda.Context, event: ByteBuffer) -> EventLoopFuture<ByteBuffer?>
}
```

What about local testing?



Fine-tuning with environment variables

LOG_LEVEL – define logging level as defined by SwiftLog; set to **INFO** by default

MAX_REQUESTS – maximum cycles library should handle before exiting; set to **none** by default

STOP_SIGNAL – signal to capture for termination; set to **TERM** by default

REQUEST_TIMEOUT – maximum time to wait for responses to come back from AWS runtime engine; set to **none** by default

How do I package it and send it to AWS?

1. SAM
2. Manually create zip and upload it
3. Container Image *new*
4. Amplify CLI integration; RFC is out – vote now!

All resources for each method and links will be provided at the end

Performance tweaks

1. Remove use of foundation



Instead, use

```
import AWSLambdaRuntimeCore
```

2. Use static linking in Swift 5.3.1

TAG

nightly-5.3-amazonlinux2

Last pushed **25 days ago** by swiftlang

DIGEST

e75a22a4242e

Swift vs. itself

Runtime	Cold start (ms)	Warm start (ms)
Swift – w/ static linking (5.3.1)	114	~0.9
Swift – w/ dynamic linking	151	~0.9

Swift vs. the world

Runtime	Cold start (ms)	Warm start (ms)
Python	53.51	0.75
Rust	67.36	0.6
Node	91.28	0.71
Swift – w/ static linking (5.3.1)	114	~0.9
Go	129.49	0.72
Swift – w/ dynamic linking	151	~0.9
Ruby	212.65	1.36
Java	350.42	0.79
.NET	400.22	0.59

Resources

SESSION CONTENT AND DEMO LINK

https://serverlessland.com/reinvent2020/svs205



Thank you!

Nicki Stone

Twitter: @kneekey23
GitHub: @kneekey23

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.



Please complete the session survey

AWS re:Invent

© 2020, Amazon Web Services, Inc. or its affiliates. All rights reserved.