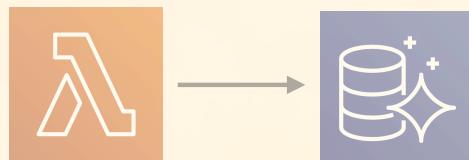




BUILDING SERVERLESS COLDFUSION APPLICATIONS WITH CFLAMBDA

Brian Klaas – @brian_klaas



WOULDN'T IT BE NICE?



SERVERLESS != FUNCTIONS



SERVERLESS = SCALABLE, PAY-AS-YOU-GO, FULLY MANAGED SERVICES



“If the first era of the cloud is defined by primitives, its days are coming to an end. The next is likely to be defined by, as the computing industry has since its inception, the abstractions we build on top of those primitives.”

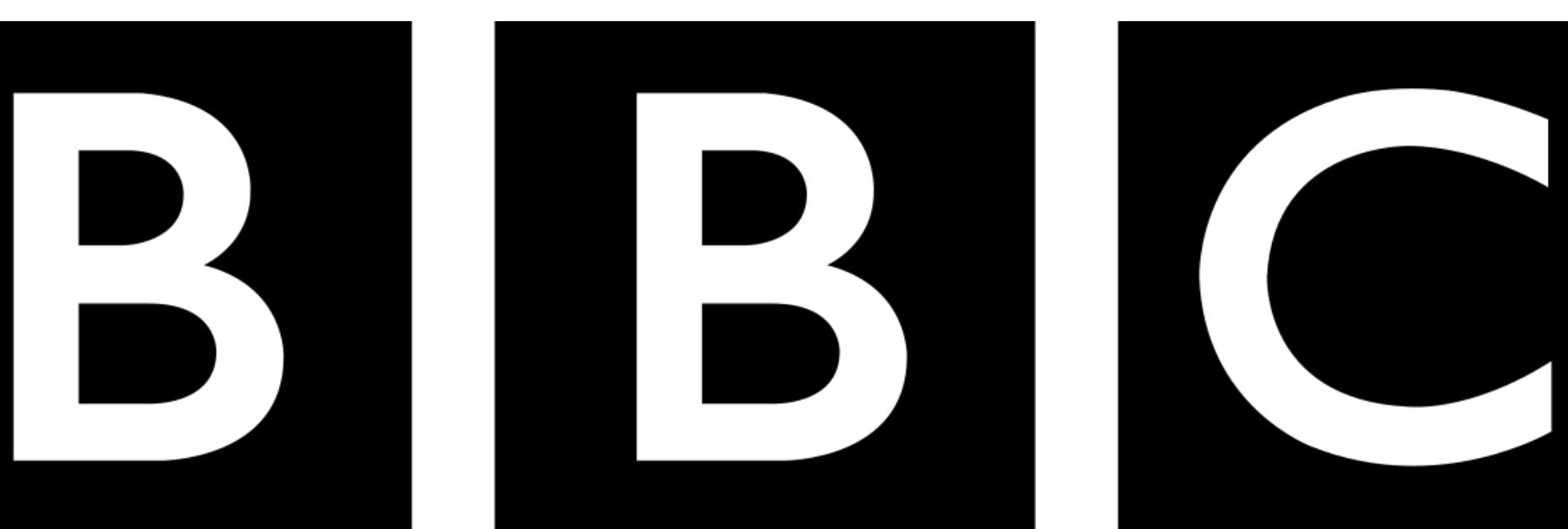
- STEPHEN O'GRADY, [REDMONK](#)

SKATE TO WHERE
THE PUCK WILL BE.





LEGO SHOP + CRM



BBC + BBC WORLD



AMAZON: SHOPPING CART + PRIME DAYS

Yes, we know it's not all unicorns and rainbows.



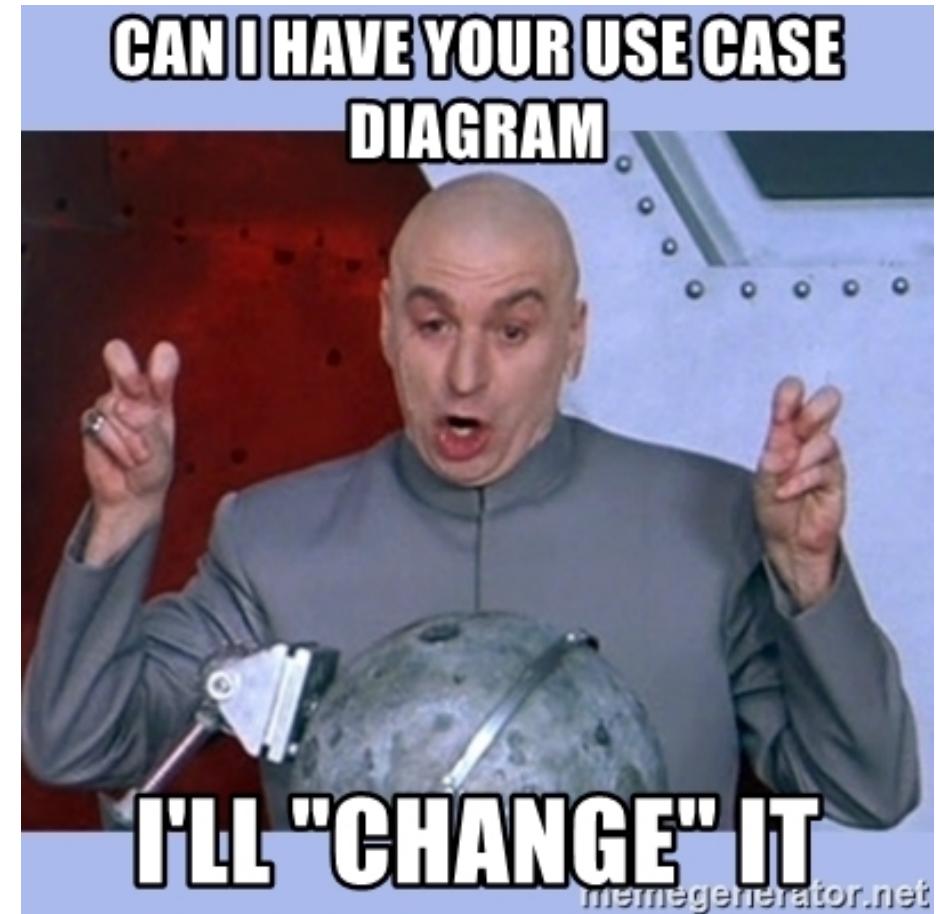
Yes, we know.



Yes, we know.



Yes, we know.



SKATE TO WHERE THE PUCK
WILL BE WITH COLDFUSION.



SKATE TO WHERE THE PUCK
WILL BE WITH CFLAMBDA.





Cloud services are a core part of CF2021

AWS:

- Simple Storage Service (S3)
- Simple Queue Service (SQS)
- Simple Notification Service (SNS)
- DynamoDB

Azure:

- Azure Blob Storage
- Azure Service Bus



cflambda is part of CF2021

A mini ColdFusion runtime that lets you create AWS Lambda services with ColdFusion

The 15 Step cflambda

- 
1. Get a copy of the cflambda ZIP
 2. Put your code inside <cflambda>/cfusion/wwwroot
 3. Update permissions on all files in <cflambda>/cfusion/bin to allow execute privileges
 4. Run cfpm to install needed modules
 5. Run cfsetup to configure the server settings (if needed)
 6. Run the lambdazip utility
 7. Get the resulting .zip file from <cflambda>/cfusion/bin/dist
 8. Upload the .zip file to S3
 9. Create a new Java 11 Lambda function in the AWS Console
 10. Link to the .zip file on S3
 11. Edit Lambda configuration to increase RAM
 12. Specify the appropriate Java class handler
 13. Add an environment variable to point to the handler function name
 14. Configure a test event
 15. Run your function for fun and profit!

TOOLING AND DEBUGGING ARE
TERRIBLE ACROSS THE ENTIRE
SERVERLESS SPACE



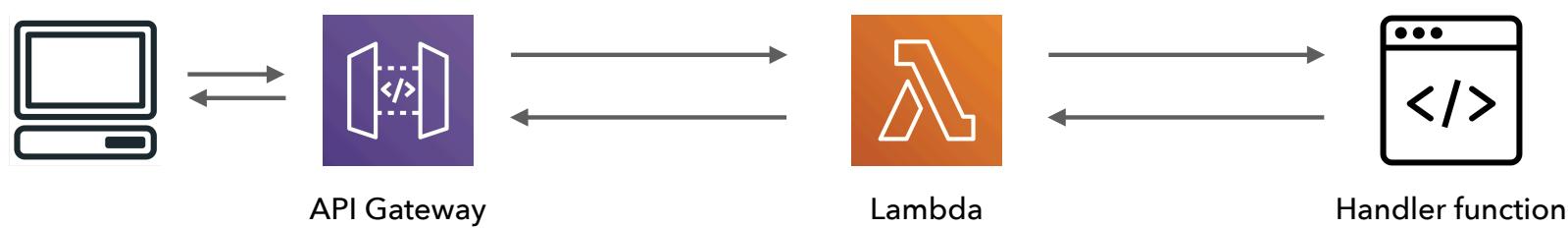
TWO WAYS TO INVOKE A LAMBDA FUNCTION

Sync vs. Async

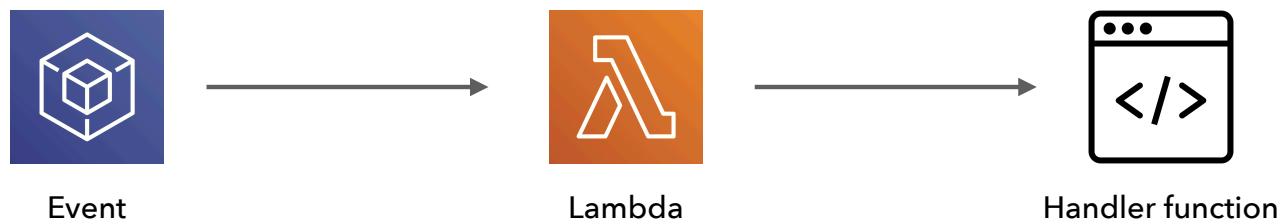
Async Invocation



Synchronous Invocation



Invoking a Lambda function



I'm going to focus on async invocation today.

GETTING UP AND RUNNING
IN AWS IS COMPLICATED



Running Your cflambda Function

- 
1. Get a copy of the cflambda ZIP
 2. Put your code inside <cflambda>/cfusion/wwwroot
 3. Update permissions on all files in <cflambda>/cfusion/bin to allow execute privileges
 4. Run cfpm to install needed modules
 5. Run cfsetup to configure the server settings (if needed)
 6. Run the lambdazip utility
 7. Get the resulting .zip file from <cflambda>/cfusion/bin/dist
 8. Upload the .zip file to S3
 9. Create a new Java 11 Lambda function in the AWS Console
 10. Link to the .zip file on S3
 11. Edit Lambda configuration to increase RAM
 12. Specify the appropriate Java class handler
 13. Add an environment variable to point to the handler function name
 14. Configure a test event
 15. Run your function for fun and profit!

The “Complex” Calculator CFC

```
component output="false" hint="A complex, time-consuming calculator" {

    public numeric function performComplexCalculation(any event, any context) {

        var returnValue = 0;

        var multiplier = Val(event.multiplier);

        cfhttp(method="GET", charset="utf-8", url="https://finnhub.io/api/v1/quote?symbol=AMZN", result="currentAmazonStockInfo") {};

        if (currentAmazonStockInfo.statusCode eq "200 OK") {
            var stockInfo = deserializeJSON(currentAmazonStockInfo.fileContent);
            var currentPrice = stockInfo.c;
            returnValue = multiplier * stockInfo.c;
        }

        return returnValue;
    }
}
```

The “Complex” Calculator CFC

```
component output="false" hint="A complex, time-consuming calculator" {  
    public numeric function performComplexCalculation(any event, any context) {  
  
        var returnValue = 0;  
  
        var multiplier = Val(event.multiplier);  
  
        cfhttp(method="GET", charset="utf-8", url="https://finnhub.io/api/v1/quote?symbol=AMZN", result="currentAmazonStockInfo") {};  
  
        if (currentAmazonStockInfo.statusCode eq "200 OK") {  
            var stockInfo = deserializeJSON(currentAmazonStockInfo.fileContent);  
            var currentPrice = stockInfo.c;  
            returnValue = multiplier * stockInfo.c;  
        }  
  
        return returnValue;  
    }  
}
```

All Lambda functions require an event and context argument.

The “Complex” Calculator CFC

```
component output="false" hint="A complex, time-consuming calculator" {  
    public numeric function performComplexCalculation(any event, any context) {  
  
        var returnValue = 0;  
  
        var multiplier = Val(event.multiplier);  
  
        cfhttp(method="GET", charset="utf-8", url="https://finnhub.io/api/v1/quote?symbol=AMZN", result="currentAmazonStockInfo") {};  
  
        if (currentAmazonStockInfo.statusCode eq "200 OK") {  
            var stockInfo = deserializeJSON(currentAmazonStockInfo.fileContent);  
            var currentPrice = stockInfo.c;  
            returnValue = multiplier * stockInfo.c;  
        }  
  
        return returnValue;  
    }  
}
```

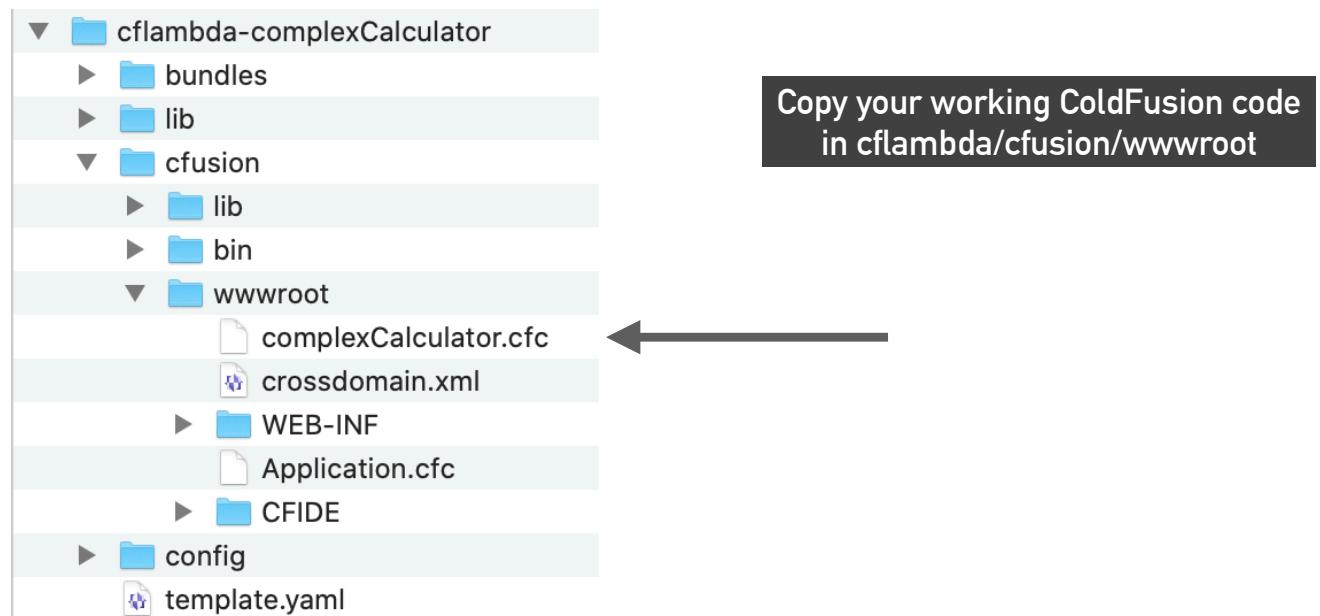
The event argument contains all incoming data to the function.

The “Complex” Calculator CFC

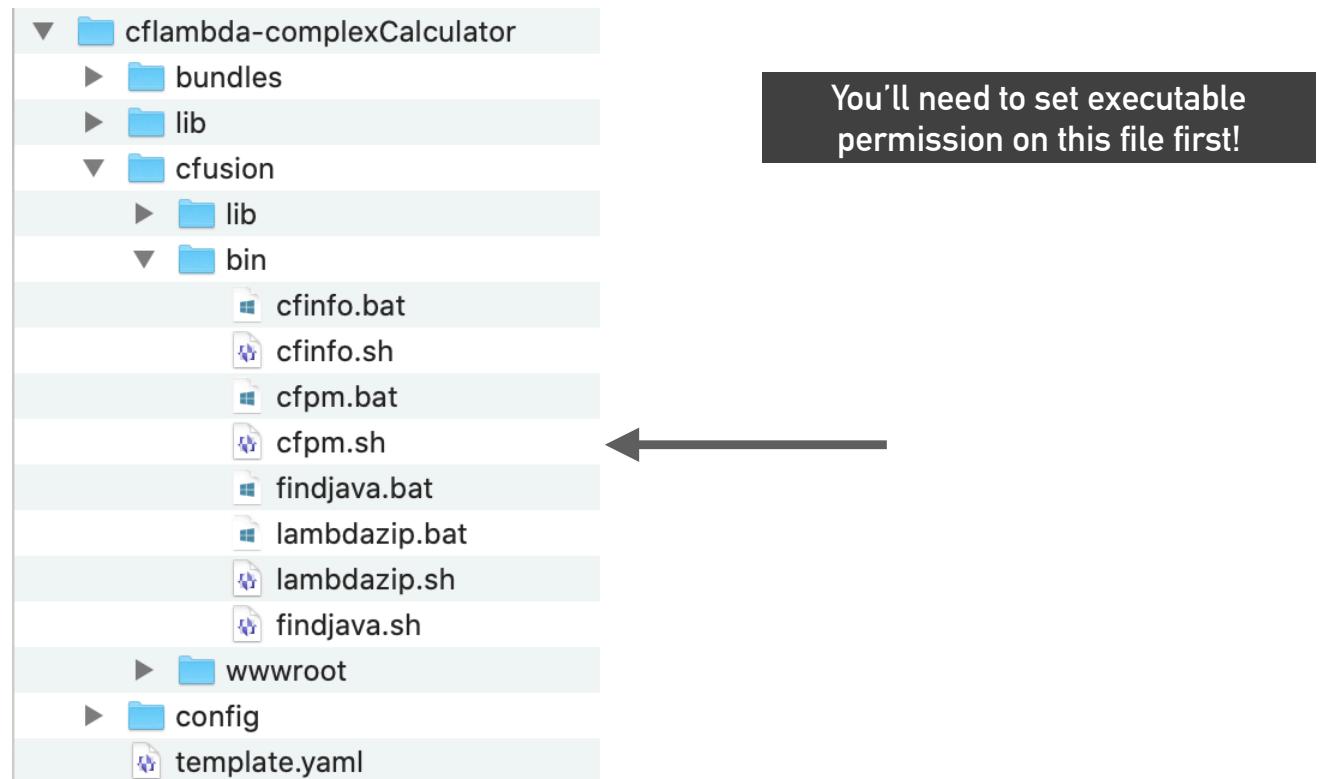
```
component output="false" hint="A complex, time-consuming calculator" {  
    public numeric function performComplexCalculation(any event, any context) {  
        var returnValue = 0;  
  
        var multiplier = Val(event.multiplier);  
  
        cfhttp(method="GET", charset="utf-8", url="https://finnhub.io/api/v1/quote?symbol=AMZN", result="currentAmazonStockInfo") {};  
  
        if (currentAmazonStockInfo.statusCode eq "200 OK") {  
            var stockInfo = deserializeJSON(currentAmazonStockInfo.fileContent);  
            var currentPrice = stockInfo.c;  
            returnValue = multiplier * stockInfo.c;  
        }  
  
        return returnValue;  
    }  
}
```

The context argument contains information about the Lambda runtime. Example: context.getAwsRequestId()

The cflambda File Bundle



Run cfpm (ColdFusion Package Manager)



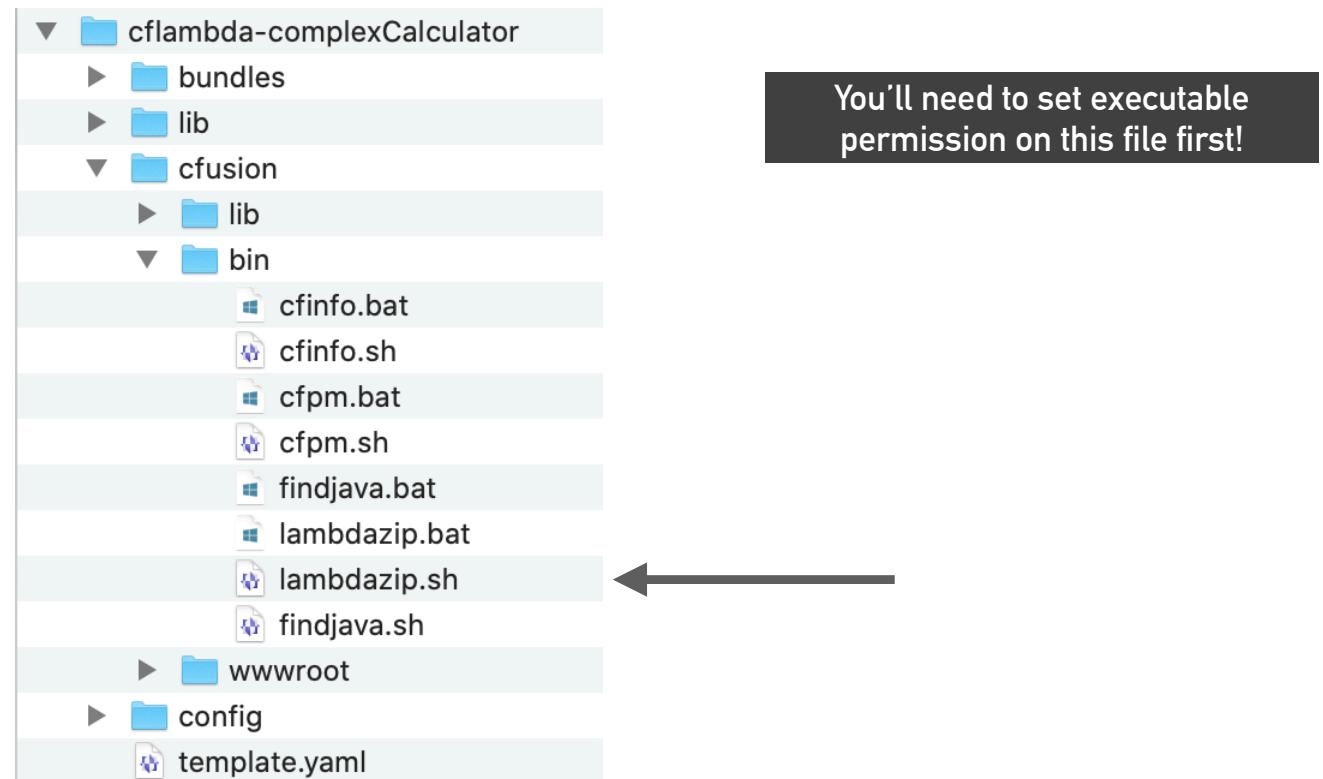
Run cfpm (ColdFusion Package Manager)

scanandinstall path/to/your/cflambda/cfusion/wwwroot
http://your.local.CF2021.server/

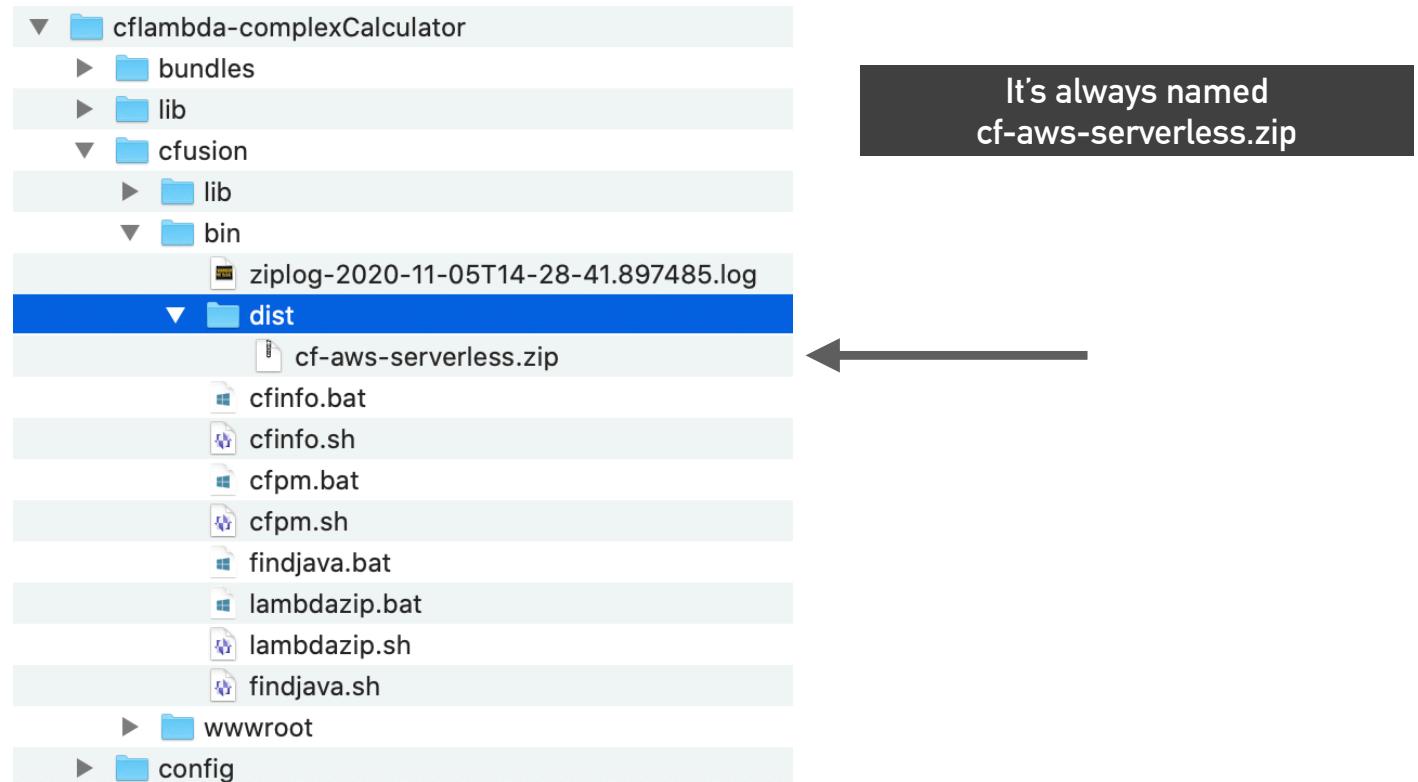
```
[BK-Home-iMac-2019:bin brianklaas$ ./cfpm.sh
cfpm>scanandinstall /Users/brianklaas/Downloads/cflambda-complexCalculator/cfusion/wwwroot
  http://127.0.0.1:8502/
The following packages will be installed : debugger,derby
debugger (2021.0.0.321466) package is already installed.
derby (2021.0.0.321466) package is already installed.

Scanning and installation completed.
cfpm> ]
```

Run lambdazip



Get the resulting ZIP file



Upload the ZIP file to S3

The Lambda console has a max file size limit of 50MB. The smallest (current) lambdazip-generated bundle is 90MB.

Name	Type	Last modified	Size	Storage class
Brian_Klaas_Headshot.jpg	jpg	October 26, 2020, 13:58 (UTC-04:00)	1.5 MB	Standard
complexCalc.zip	zip	October 28, 2020, 11:51 (UTC-04:00)	92.2 MB	Standard
defaultACFBundle.zip	zip	October 26, 2020, 13:32 (UTC-04:00)	86.2 MB	Standard
lions.jpeg	jpeg	October 28, 2020, 14:06 (UTC-04:00)	4.0 MB	Standard
resizerWithXRay.zip	zip	November 2, 2020, 15:44 (UTC-05:00)	102.1 MB	Standard
simpleResizer.zip	zip	October 29, 2020, 14:58 (UTC-04:00)	102.1 MB	Standard

The Lambda console has a max file size limit of 50MB. The smallest (current) lambdazip-generated bundle is 90MB.

Create a New Lambda Function

Create function [Info](#)

Choose one of the following options to create your function.

Author from scratch Start with a simple Hello World example.
A icon showing a computer monitor and gear.

Use a blueprint Build a Lambda application from sample code and configuration presets for common use cases.
A icon showing two checkmarks in boxes.

Browse serverless app reposi Deploy a sample Lambda application Serverless Application Repository.
A icon showing a cloud and a folder.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function.
 ←

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ [Change default execution role](#)

cflambda functions use the Java 11 (Corretto) runtime

Are You Using the Right IAM Permissions?

Create function Info

Choose one of the following options to create your function.

Author from scratch Start with a simple Hello World example.


Use a blueprint Build a Lambda application from sample code and configuration presets for common use cases.


Browse serverless app reposi... Deploy a sample Lambda application Serverless Application Repository.


Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime Info
Choose the language to use to write your function.

Permissions Info
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

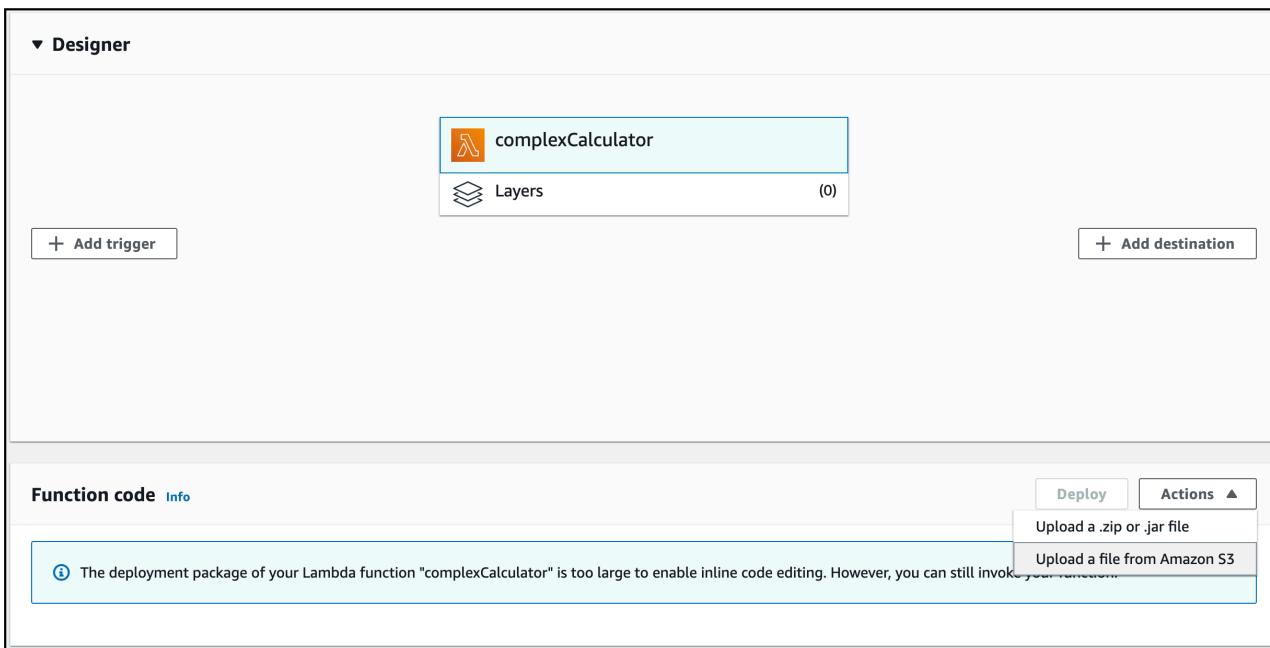
► Change default execution role 

Make sure the Lambda function has permission to access all services required by the function!

Lambda functions run under IAM **roles**, not user accounts.

The minimum permission needed is logging to CloudWatch.

Link Your Function to the ZIP on S3



You'll need the full http path to your ZIP bundle in S3.

YOU HAVE TO RE-LINK TO THE ZIP
IN S3 **EVERY TIME** YOU MAKE A
CODE CHANGE



Edit “Basic Settings”: Memory

Basic settings [Info](#)

Description - optional

Runtime

Java 11 (Corretto)

Handler [Info](#)

coldfusion.serverless.handlers.cf.CFLambdaRequestStreamHandler::handleEventRequir

Memory (MB)

Your function is allocated CPU proportional to the memory configured.

512 MB

Timeout

0 min 20 sec

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role

Create a new role from AWS policy templates

Existing role

Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

Lambda_ServerlessAgentPolicy

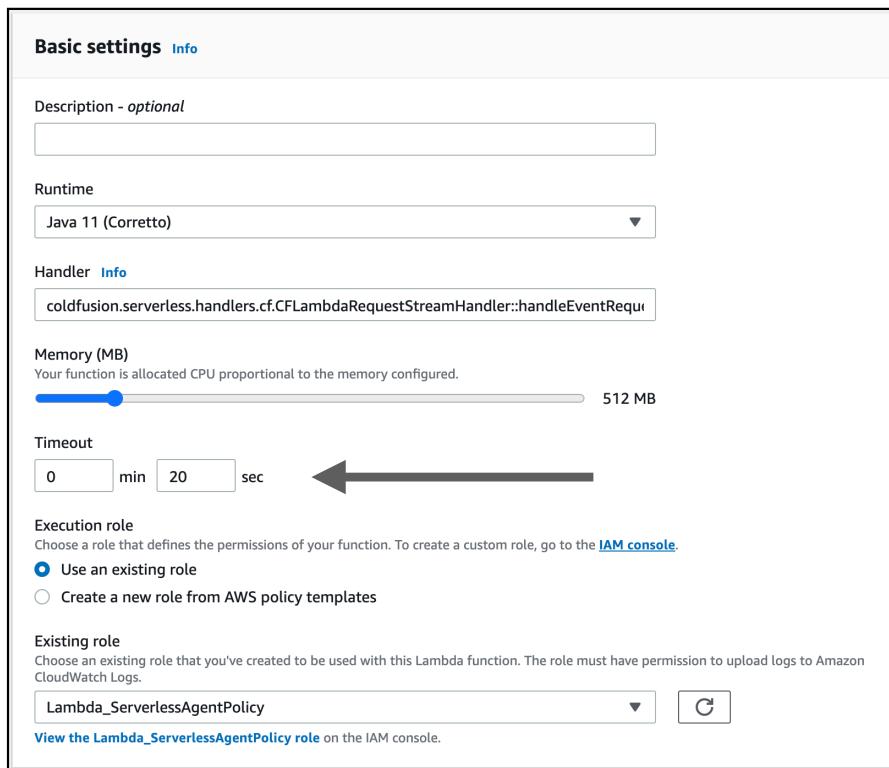
[View the Lambda_ServerlessAgentPolicy role](#) on the IAM console.

Lambda allocates CPU power linearly in proportion to the amount of memory configured.

Don't over-allocate. This costs you money.

Use **aws-lambda-power-tuning**:
<https://github.com/alexcasalboni/aws-lambda-power-tuning>

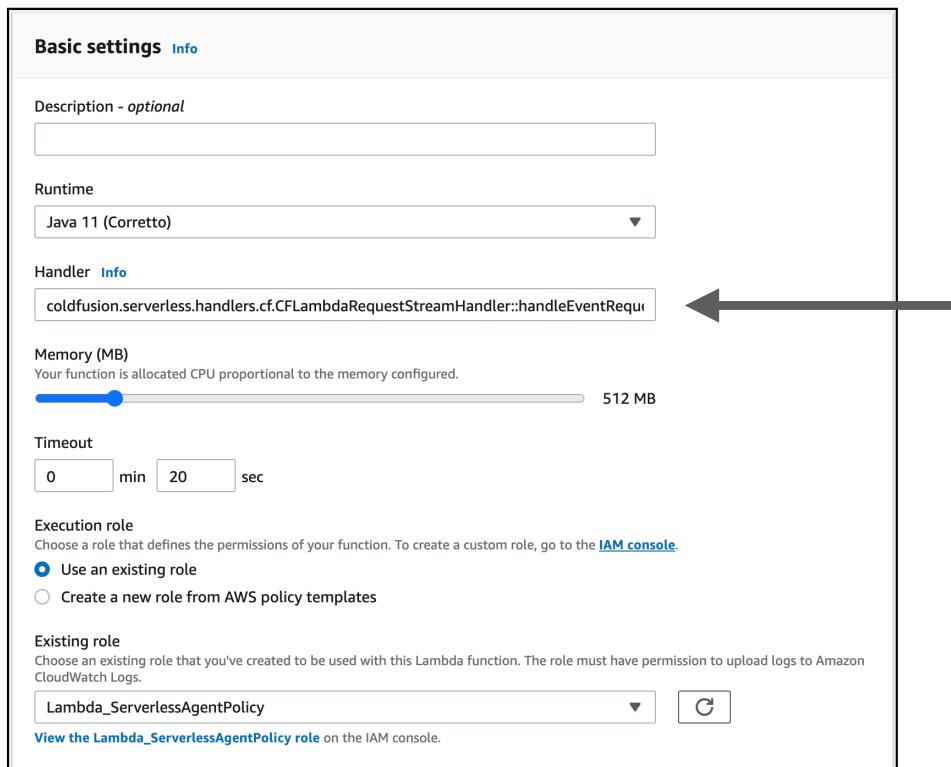
Edit “Basic Settings”: Timeout



The screenshot shows the 'Basic settings' section of the AWS Lambda function configuration. It includes fields for Description, Runtime (Java 11 (Corretto)), Handler (coldfusion.serverless.handlers.cf.CFLambdaRequestStreamHandler::handleEventRequ), Memory (512 MB), and Timeout (20 sec). The 'Execution role' section shows 'Use an existing role' selected, with 'Lambda_ServerlessAgentPolicy' chosen. A note on the right says: 'Be generous with the timeout. Cold starts can take a long time.'

Be generous with the timeout.
Cold starts can take a long time.

Edit “Basic Settings”: Handler



Basic settings [Info](#)

Description - optional

Runtime

Java 11 (Corretto)

Handler [Info](#)

coldfusion.serverless.handlers.cf.CFLambdaRequestStreamHandler::handleEventRequest

Memory (MB)
Your function is allocated CPU proportional to the memory configured.

512 MB

Timeout

0 min 20 sec

Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Use an existing role

Create a new role from AWS policy templates

Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.

Lambda_ServerlessAgentPolicy

[View the Lambda_ServerlessAgentPolicy role](#) on the IAM console.

The “handler” is the java servlet,
not your CFC.

coldfusion.serverless.handlers.cf.
CFLambdaRequestStreamHandler::handleEventRequest

Add the Function as an Environment Variable

The eventHandler is the function in your CFC.

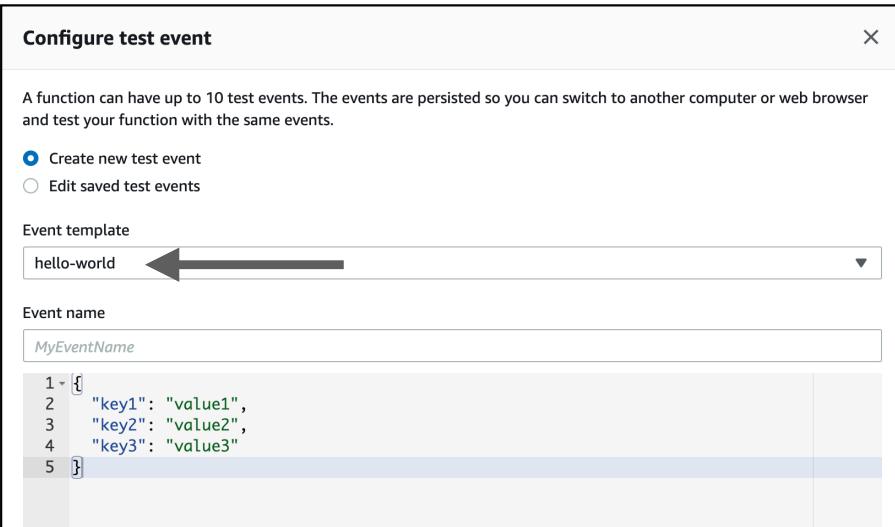
Environment variables (1)

The environment variables below are encrypted at rest with the default Lambda service key.

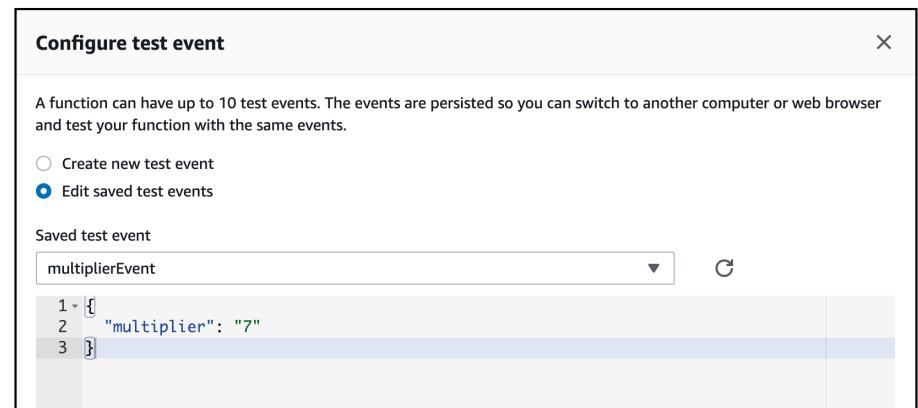
Key	Value
eventHandler	complexCalculator::performComplexCalculation

Not providing an eventHandler environment variable for asynchronous execution defaults to executing the serverlessEvent() method present in the default Application.cfc provided as part of the cflambda bundle.

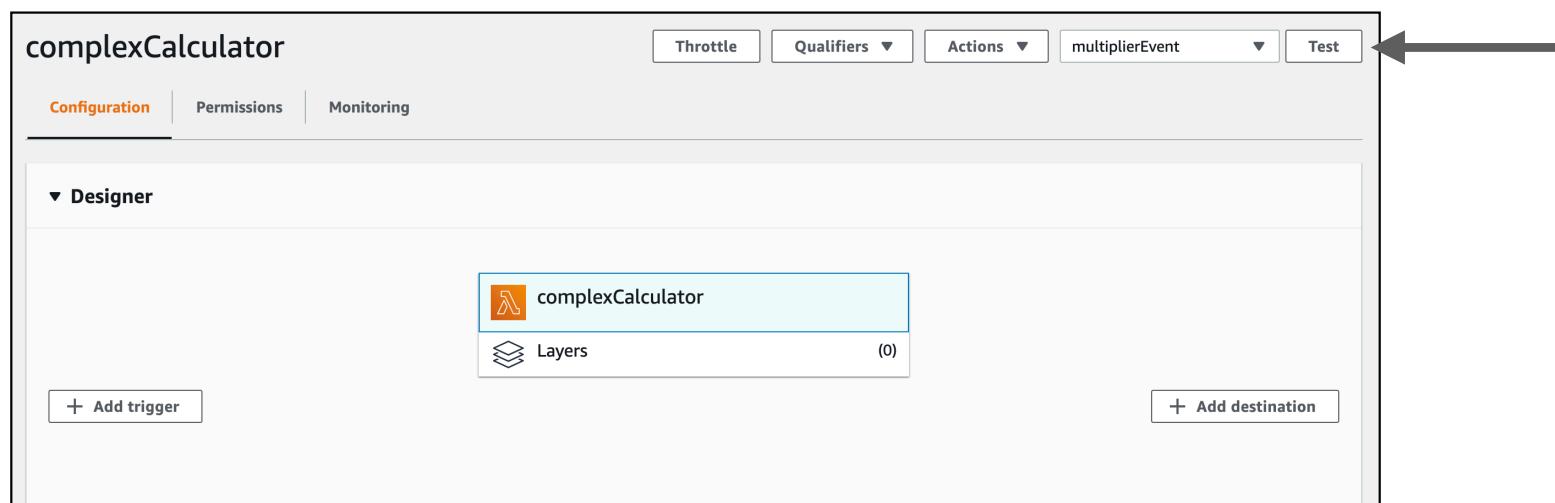
Configure a Test Event



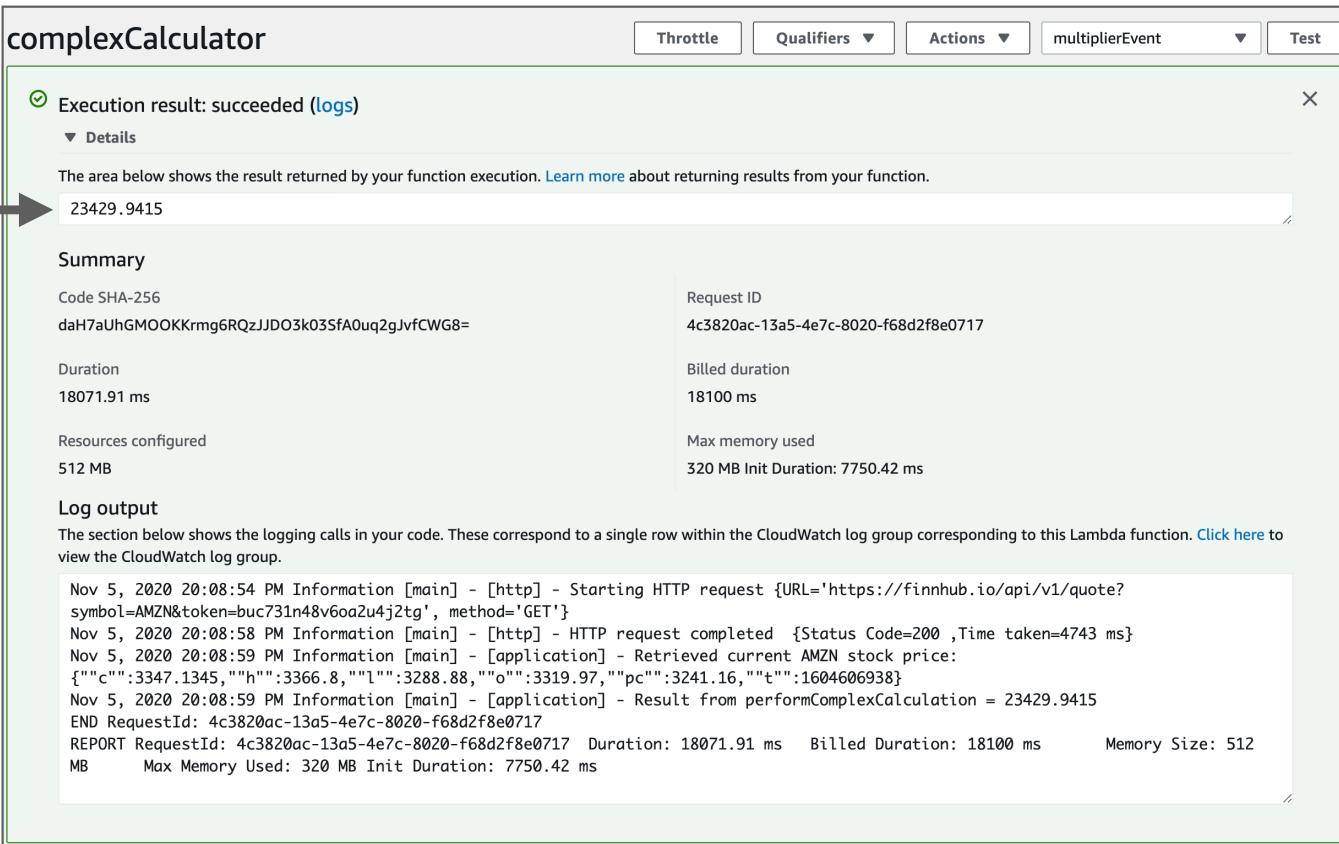
All arguments required by your function need to be in the test event.



Run Your cflambda Function!



See the Results



The screenshot shows the AWS Lambda function details page for 'complexCalculator'. At the top, there are tabs for Throttle, Qualifiers, Actions, multiplierEvent, and Test. Below the tabs, a green checkmark indicates 'Execution result: succeeded (logs)'. A large text area displays the result value '23429.9415'. To the left of this value is a black arrow pointing from the left side of the slide towards it. Below the result value, there's a 'Summary' section with various metrics like Request ID, Duration, and Max memory used. At the bottom, there's a 'Log output' section showing CloudWatch logs.

complexCalculator

Throttle Qualifiers Actions multiplierEvent Test

Execution result: succeeded (logs)

Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

23429.9415

Summary

Code SHA-256
daH7aUhGMOOKKrmg6RQzJJDO3k03SfA0uq2gJvfCWG8=

Duration
18071.91 ms

Resources configured
512 MB

Request ID
4c3820ac-13a5-4e7c-8020-f68d2f8e0717

Billed duration
18100 ms

Max memory used
320 MB Init Duration: 7750.42 ms

Log output

The section below shows the logging calls in your code. These correspond to a single row within the CloudWatch log group corresponding to this Lambda function. [Click here](#) to view the CloudWatch log group.

```
Nov 5, 2020 20:08:54 PM Information [main] - [http] - Starting HTTP request {URL='https://finnhub.io/api/v1/quote?symbol=AMZN&token=buc731n48v6oa2u4j2tg', method='GET'}
Nov 5, 2020 20:08:58 PM Information [main] - [http] - HTTP request completed {Status Code=200 ,Time taken=4743 ms}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Retrieved current AMZN stock price: {"c":3347.1345,"h":3366.8,"l":3288.88,"o":3319.97,"pc":3241.16,"t":1604606938}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Result from performComplexCalculation = 23429.9415
END RequestId: 4c3820ac-13a5-4e7c-8020-f68d2f8e0717
REPORT RequestId: 4c3820ac-13a5-4e7c-8020-f68d2f8e0717 Duration: 18071.91 ms Billed Duration: 18100 ms Memory Size: 512
MB Max Memory Used: 320 MB Init Duration: 7750.42 ms
```

Where is the Log Info Coming From?

The screenshot shows the AWS Lambda execution history interface for a function named "complexCalculator". The execution has succeeded, returning the result "23429.9415". The "Logs" section displays CloudWatch logs, which are highlighted with a red box. The logs show the following sequence of events:

```
Nov 5, 2020 20:08:54 PM Information [main] - [http] - Starting HTTP request {URL='https://finnhub.io/api/v1/quote?symbol=AMZN&token=buc731n48v6oa2u4j2tg', method='GET'}
Nov 5, 2020 20:08:58 PM Information [main] - [http] - HTTP request completed {Status Code=200 ,Time taken=4743 ms}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Retrieved current AMZN stock price: {"c":3347.1345,"h":3366.8,"l":3288.88,"o":3319.97,"pc":3241.16,"t":1604606938}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Result from performComplexCalculation = 23429.9415
END RequestId: 4c3820ac-13a5-4e7c-8020-f68d2f8e0717
REPORT RequestId: 4c3820ac-13a5-4e7c-8020-f68d2f8e0717 Duration: 18071.91 ms Billed Duration: 18100 ms Memory Size: 512 MB Max Memory Used: 320 MB Init Duration: 7750.42 ms
```

Add to CloudWatch Logs with the APPLICATION_LOG

```
var cflogObject = CreateObject("java","coldfusion.log.CFLogs");
var logger = cflogObject.APPLICATION_LOG;

logger.info("Starting Request to performComplexCalculation: " & context.getAwsRequestId());

logger.info("Result from performComplexCalculation = " & returnValue);
```

Your Log Entries in the CloudWatch Logs

The screenshot shows the AWS Lambda execution results for a function named "complexCalculator". The execution was successful, returning the value 23429.9415. The log output section highlights the following application-layer logs:

```
Nov 5, 2020 20:08:54 PM Information [main] - [http] - Starting HTTP request {URL='https://finnhub.io/api/v1/quote?symbol=AMZN&token=buc731n48v6oa2u4j2tg', method='GET'}
Nov 5, 2020 20:08:58 PM Information [main] - [http] - HTTP request completed {Status Code=200 Time taken=4743 ms}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Retrieved current AMZN stock price: {"c":3347.1345,"h":3366.8,"l":3288.88,"o":3319.97,"pc":3241.16,"t":1604606938}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Result from performComplexCalculation = 23429.9415
```

What's With That Super-Long Duration?

The screenshot shows the AWS Lambda execution details page for a function named "complexCalculator". The execution was successful, returning the value 23429.9415. The summary shows a duration of 18071.91 ms. The log output section contains several log entries, with the duration field highlighted by a red box.

Execution result: succeeded (logs)

23429.9415

Summary

Code SHA-256	Request ID
daH7aUhGMOOKKrmg6RQzJJDO3k03SfA0uq2gJvfCWG8=	4c3820ac-13a5-4e7c-8020-f68d2f8e0717
Duration	Billed duration
18071.91 ms	18100 ms
Resources configured	Max memory used
512 MB	320 MB Init Duration: 7750.42 ms

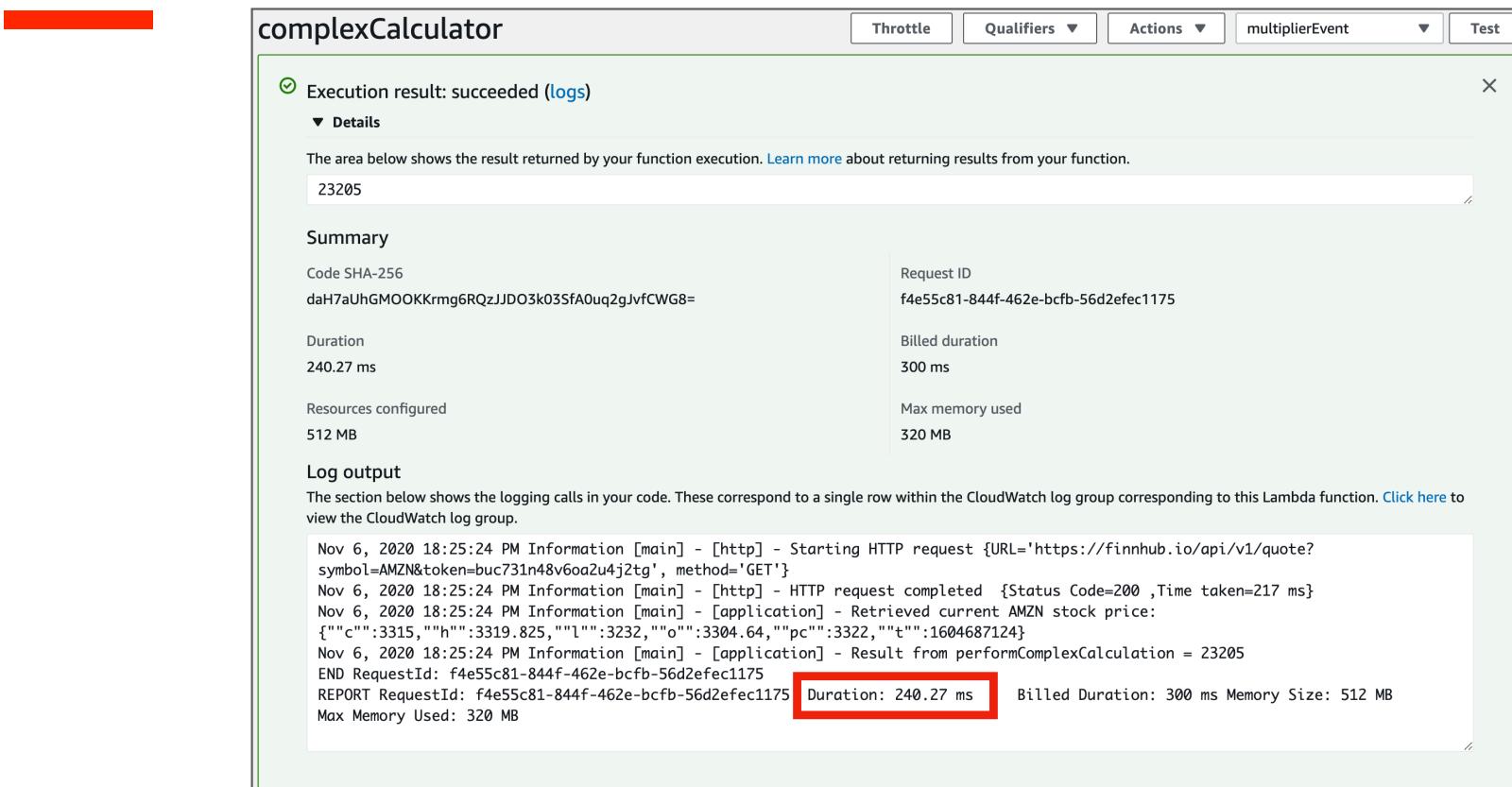
Log output

```
Nov 5, 2020 20:08:54 PM Information [main] - [http] - Starting HTTP request {URL='https://finnhub.io/api/v1/quote?symbol=AMZN&token=buc731n48v6oa2u4j2tg', method='GET'}
Nov 5, 2020 20:08:58 PM Information [main] - [http] - HTTP request completed {Status Code=200 ,Time taken=4743 ms}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Retrieved current AMZN stock price: {"c":3347.1345,"h":3366.8,"l":3288.88,"o":3319.97,"pc":3241.16,"t":1604606938}
Nov 5, 2020 20:08:59 PM Information [main] - [application] - Result from performComplexCalculation = 23429.9415
END RequestId: 4c3820ac-13a5-4e7c-8020-f68d2f8e0717
REPORT RequestId: 4c3820ac-13a5-4e7c-8020-f68d2f8e0717 Duration: 18071.91 ms Billed Duration: 18100 ms Memory Size: 512
MB Max Memory Used: 320 MB Init Duration: 7750.42 ms
```

Why So Long, Lambda?

1. Fire up the container
2. Download your source ZIP file
3. Unpack your source ZIP file
4. Start Java
5. Start the ColdFusion server
6. Execute your code

Subsequent Invocations: Fast



The screenshot shows the AWS Lambda execution history for a function named "complexCalculator". The execution was successful, returning the value 23205. The summary details show a duration of 240.27 ms and a billed duration of 300 ms. The log output displays the request for a stock quote and the resulting calculation.

Execution result: succeeded (logs)

Summary

Code SHA-256	Request ID
daH7aUhGMOOKKrmg6RQzJJDO3k03SfA0uq2gJvfCWG8=	f4e55c81-844f-462e-bcfb-56d2efec1175

Duration	Billed duration
240.27 ms	300 ms

Resources configured	Max memory used
512 MB	320 MB

Log output

```
Nov 6, 2020 18:25:24 PM Information [main] - [http] - Starting HTTP request {URL='https://finnhub.io/api/v1/quote?symbol=AMZN&token=buc731n48v6oa2u4j2tg', method='GET'}
Nov 6, 2020 18:25:24 PM Information [main] - [http] - HTTP request completed {Status Code=200 ,Time taken=217 ms}
Nov 6, 2020 18:25:24 PM Information [main] - [application] - Retrieved current AMZN stock price: {"c":3315,"h":3319.825,"l":3232,"o":3304.64,"pc":3322,"t":1604687124}
Nov 6, 2020 18:25:24 PM Information [main] - [application] - Result from performComplexCalculation = 23205
END RequestId: f4e55c81-844f-462e-bcfb-56d2efec1175
REPORT RequestId: f4e55c81-844f-462e-bcfb-56d2efec1175 Duration: 240.27 ms Billed Duration: 300 ms Memory Size: 512 MB
Max Memory Used: 320 MB
```

COLD STARTS ARE STILL
A PROBLEM IN JAVA



Avoiding Long Cold Starts

1. Use provisioned concurrency
2. Reduce the size of the deployment bundle
3. Use CloudWatch triggers
4. Don't worry – it's async!

How Would You Invoke This from CF?

```
lambda = awsServiceFactory.createServiceObject('lambda');

invokeRequest = CreateObject('java', 'com.amazonaws.services.lambda.model.InvokeRequest').init();

invokeRequest.setFunctionName(lambdaFunctionARN);

invokeRequest.setPayload(jsonPayload);

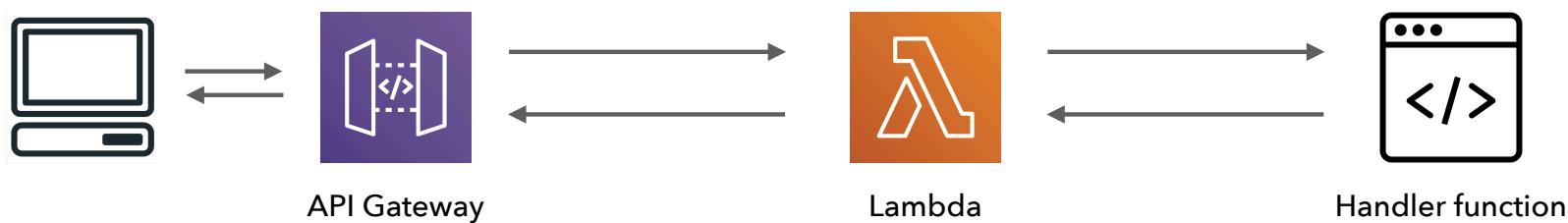
result = variables.lambda.invoke(invokeRequest);
```

Full example in my AWS Playbox repo:
<https://github.com/brianklaas/awsPlaybox>

Async Invocation



What about synchronous invocation from a Web app?



Paul Kukiel's sample API Gateway Lambda repo:
<https://github.com/kukielp/acf-lambda>

AWS Serverless Application Model (SAM)

```
AWSTemplateFormatVersion: '2010-09-09'  
Transform: AWS::Serverless-2016-10-31  
  
Globals:  
  
Function:  
  
    Timeout: 60  
  
Resources:  
  
AWSLambda:  
  
    Type: AWS::Serverless::Function  
  
Properties:  
  
    CodeUri: ./cfusion/bin/dist/cf-aws-serverless.zip  
  
    Handler: coldfusion.serverless.handlers.cf.CFLambdaRequestStreamHandler::handleEventRequest  
  
    Runtime: java11  
  
    MemorySize: 1024
```



Use AWS SAM to Manage Your cflambda Deployment

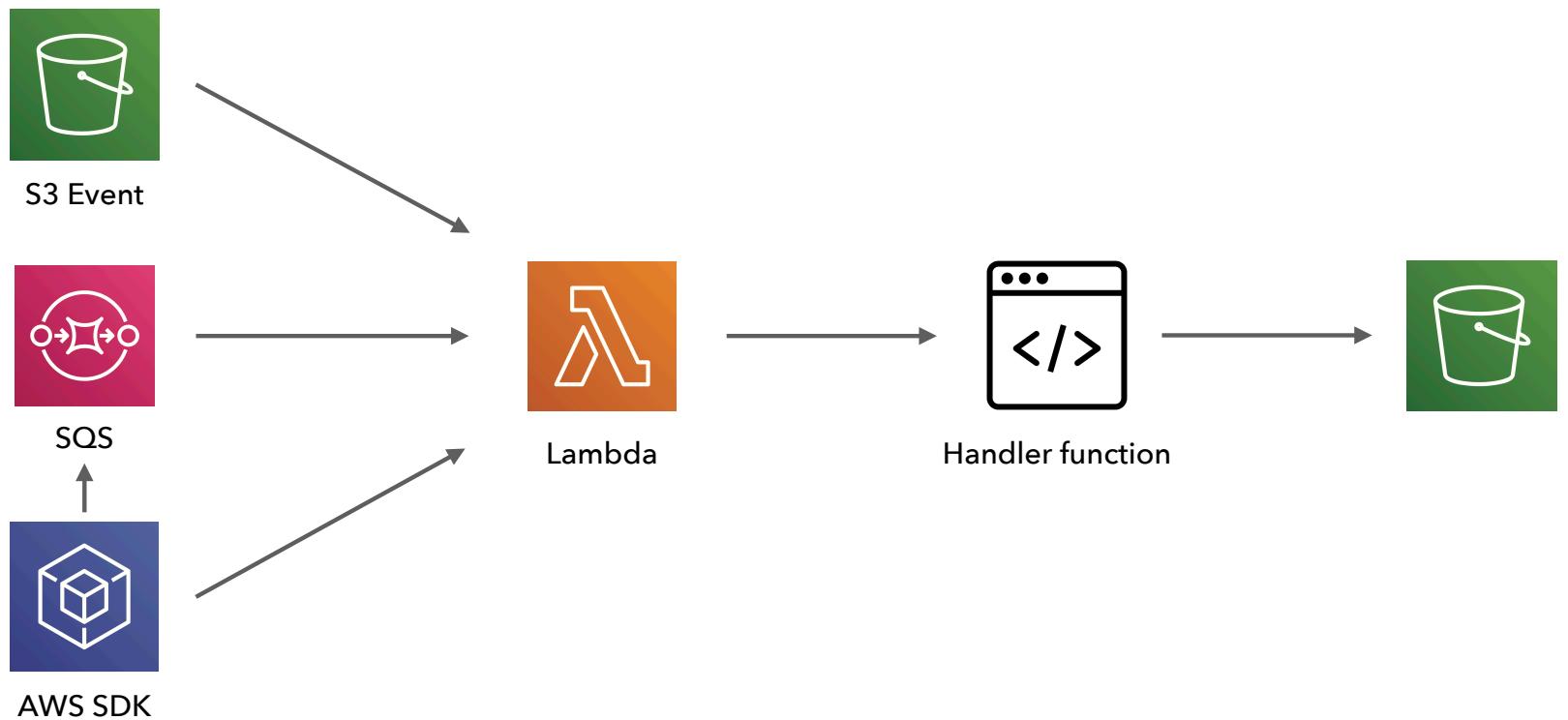
Built and supported by AWS

Infrastructure as code – no fiddling in the console

Handles upload and runtime config for you in a single command

Can run Lambda functions locally for testing!

The Real Power: Serverless Applications



The Canonical Image Resizer CFC

```
component output="false" hint="A utility for resizing images" {

    public string function resizelImage(any event, any context) {

        init();
        variables.logger.info("Executing main body of resizelImage - RequestID: " & context.getAwsRequestId());

        // Read event values
        if (structKeyExists(event, "urlOfImageOnS3")){
            variables.logger.info("This request came in via a direct invocation");
            var pathToImageOnS3 = event.urlOfImageOnS3;

        } else {
            variables.logger.info("This request came in via SQS");
            variables.logger.info("SQS message body:" & event.records[1].body);
            var sqsBodyAsJSON = deserializeJSON(event.records[1].body);
            var pathToImageOnS3 = sqsBodyAsJSON.urlOfImageOnS3;
        }
    }
}
```

The Canonical Image Resizer CFC

```
var origFileExtension = ListLast(pathToImageOnS3, "");  
var origFileName = ListLast(Left(pathToImageOnS3, (Len(pathToImageOnS3) - Len(origFileExtension) - 1)), "/");  
var pathToFileOnS3 = Left(pathToImageOnS3, (Len(pathToImageOnS3) - Len(origFileName) - Len(origFileExtension) - 2));  
var fileNameToWrite = "templImage." & origFileExtension;  
  
var sourceFileInTmp = "/tmp/" & fileNameToWrite; ←  
  
variables.s3Service = cloudService(this.s3Cred, this.s3Conf);  
variables.bucketObj = s3Service.bucket("nameOfMyBucket");
```

Lambda has a read-only file system!

500MB of temp storage in /tmp

The Canonical Image Resizer CFC

```
var origFileExtension = ListLast(pathToImageOnS3, "");  
var origFileName = ListLast(Left(pathToImageOnS3, (Len(pathToImageOnS3) - Len(origFileExtension) - 1)), "/");  
var pathToFileOnS3 = Left(pathToImageOnS3, (Len(pathToImageOnS3) - Len(origFileName) - Len(origFileExtension) - 2));  
var fileNameToWrite = "templImage." & origFileExtension;  
  
var sourceFileInTmp = "/tmp/" & fileNameToWrite;  
  
variables.s3Service = cloudService(this.s3Cred, this.s3Conf);  
variables.bucketObj = s3Service.bucket("nameOfMyBucket");
```

Yes, you can use CF2021's
cloudService() functionality in cflambda!



The Canonical Image Resizer CFC

```
beginAWSXraySubsegment("downloadFileFromS3");
var downloadRequest = {
    "destinationFile": sourceFileInTmp,
    "key": pathToImageOnS3
};
variables.logger.info("Getting file: " & pathToImageOnS3);
try {
    var downloadResponse = variables.bucketObj.downloadToFile(downloadRequest);
} catch (any e) {
    variables.logger.info("Error while getting file from S3:" & serializeJSON(e));
    return false;
}
endAWSXraySubsegment();
```

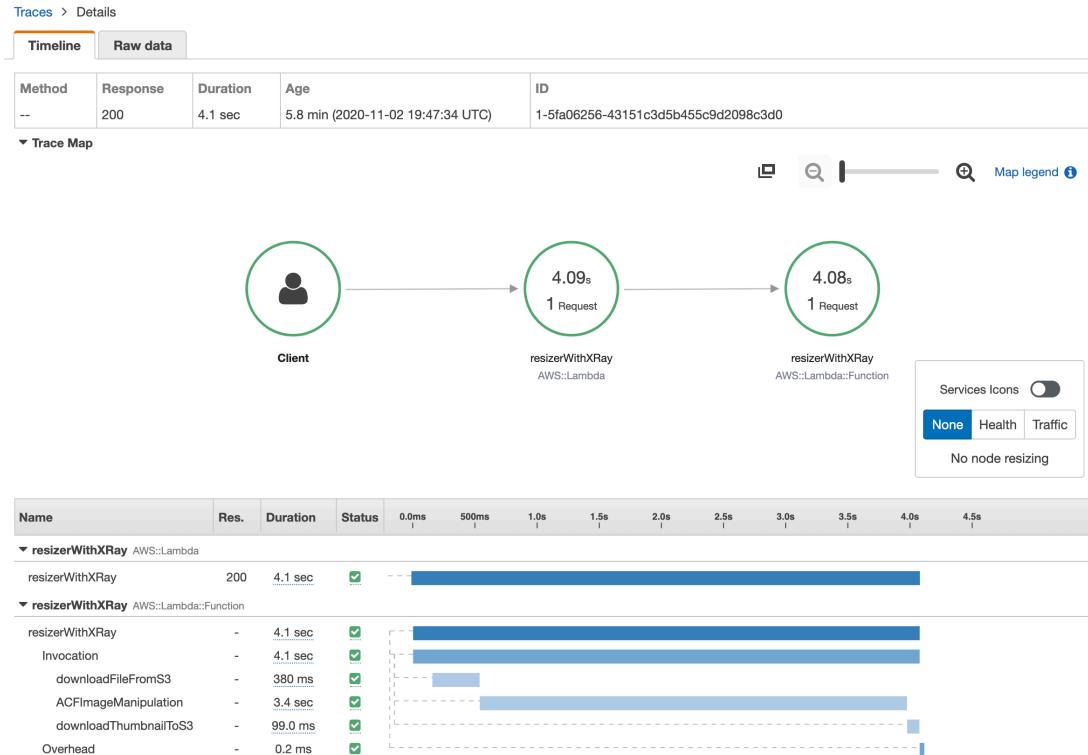
The Canonical Image Resizer CFC

```
beginAWSXraySubsegment("downloadFileFromS3");
var downloadRequest = {
    "destinationFile": sourceFileInTmp,
    "key": pathToImageOnS3
};
variables.logger.info("Getting file: " & pathToImageOnS3);
try {
    var downloadResponse = variables.bucketObj.downloadToFile(downloadRequest);
} catch (any e) {
    variables.logger.info("Error while getting file from S3:" & serializeJSON(e));
    return false;
}
endAWSXraySubsegment();
```

What's this X-Ray stuff?



X-Ray: Distributed Tracing and Timing



X-Ray shows you request times across AWS services or in your own code.

X-Ray: Distributed Tracing and Timing



The screenshot shows a CloudWatch Metrics Insights timeline for the `resizerWithXRay` Lambda function. The timeline starts at 0.0ms and ends at 4.5s. The main function execution is labeled `resizerWithXRay` with a duration of 4.1 seconds. This execution is represented by a large blue horizontal bar. Within this bar, several sub-tasks are visible, each with a green checkmark icon. Three specific sub-tasks are highlighted with a red box: `downloadFileFromS3` (duration 380 ms), `ACFImageManipulation` (duration 3.4 sec), and `downloadThumbnailToS3` (duration 99.0 ms). These highlighted tasks also have green checkmark icons next to their names.

Name	Res.	Duration	Status
resizerWithXRay	200	4.1 sec	✓
Invocation	-	4.1 sec	✓
downloadFileFromS3	-	380 ms	✓
ACFImageManipulation	-	3.4 sec	✓
downloadThumbnailToS3	-	99.0 ms	✓
Overhead	-	0.2 ms	✓

The Canonical Image Resizer CFC



```
beginAWSXraySubsegment("downloadFileFromS3");
var downloadRequest = {
    "destinationFile": sourceFileInTmp,
    "key": pathToImageOnS3
};
variables.logger.info("Getting file: " & pathToImageOnS3);
try {
    var downloadResponse = variables.bucketObj.downloadToFile(downloadRequest);
} catch (any e) {
    variables.logger.info("Error while getting file from S3:" & serializeJSON(e));
    return false;
}
endAWSXraySubsegment();
```

You can make as many X-Ray subsegments as you want.

The Canonical Image Resizer CFC

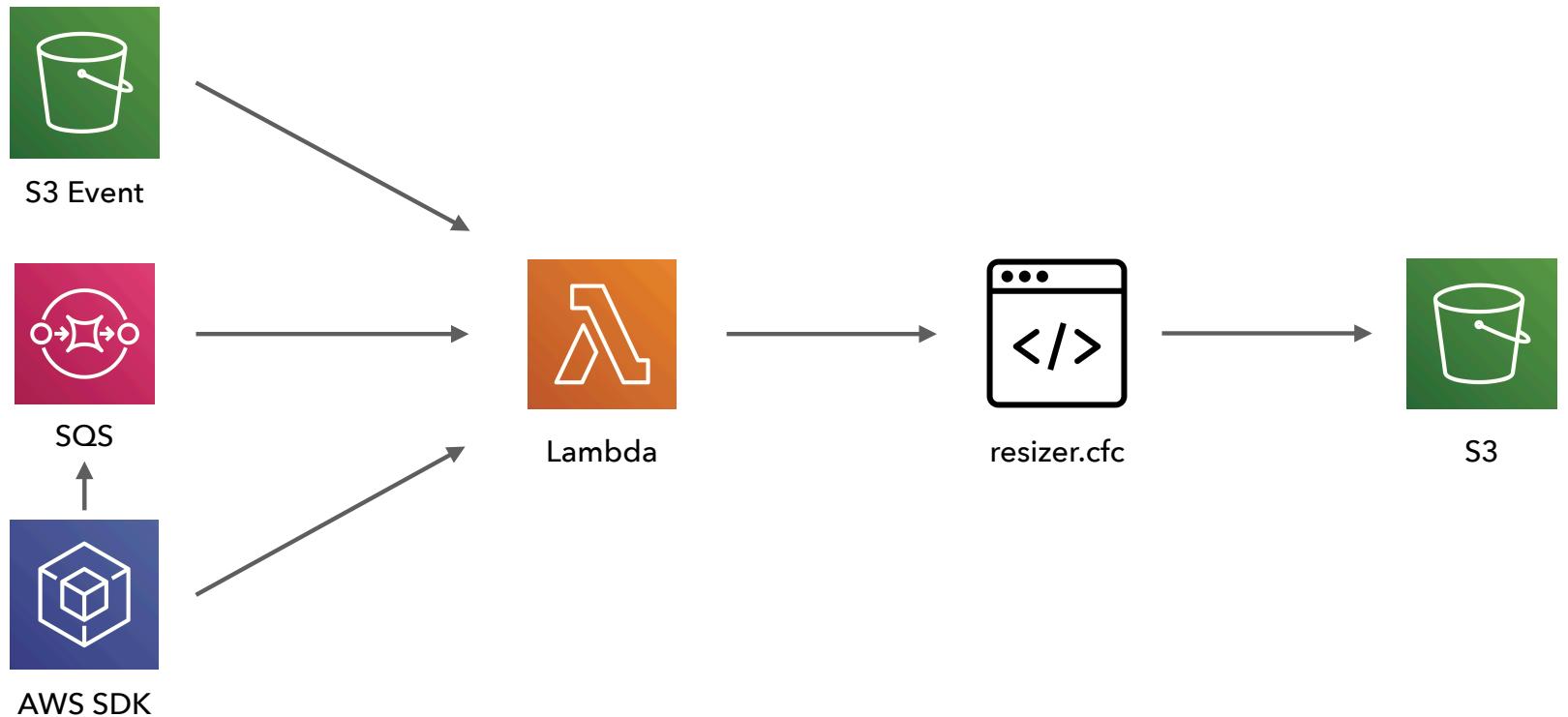
```
beginAWSXraySubsegment("ACFImageManipulation");
var sourceImage = imageRead(sourceFileInTmp);
imageResize(sourceImage, "25%", "");
var resizedImageName = origFileName & "-thumb." & origFileExtension;
var resizedImageInTmp = "/tmp/" & resizedImageName;
imageWrite(sourceImage, resizedImageInTmp);
endAWSXraySubsegment();
```

cflambda can use almost any
ColdFusion language feature/function

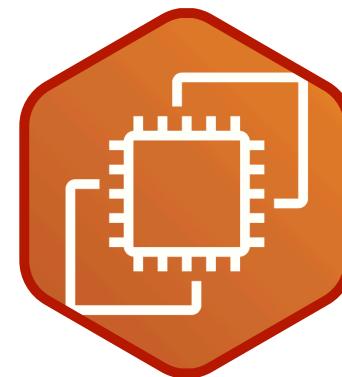
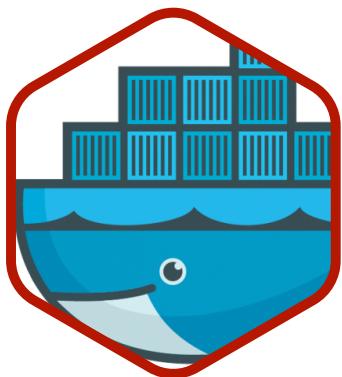
The Canonical Image Resizer CFC

```
beginAWSXraySubsegment("uploadThumbnailToS3");
var destinationPathOnS3 = "";
if (len(pathToFileOnS3)) {
    destinationPathOnS3 &= pathToFileOnS3 & "/";
}
destinationPathOnS3 &= resizedImageName;
var uploadRequest = {
    "srcFile" : resizedImageInTmp,
    "key" : destinationPathOnS3
};
variables.logger.info("Writing thumbnail file: " & destinationPathOnS3);
try {
    var uploadResponse = variables.bucketObj.uploadFile(uploadRequest);
} catch (any e) {
    variables.logger.info("Error while writing resized file to S3:" & serializeJSON(e));
    return false;
}
endAWSXraySubsegment();
```

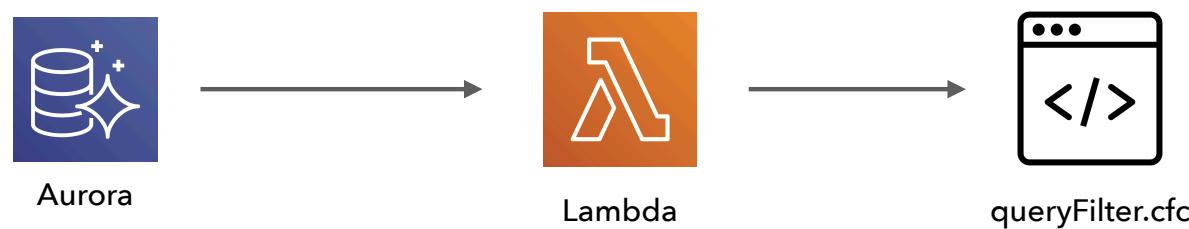
Invoke from S3, SQS or the Java SDK



Microservices without the overhead of:



Write Aurora User-Defined Functions in ColdFusion

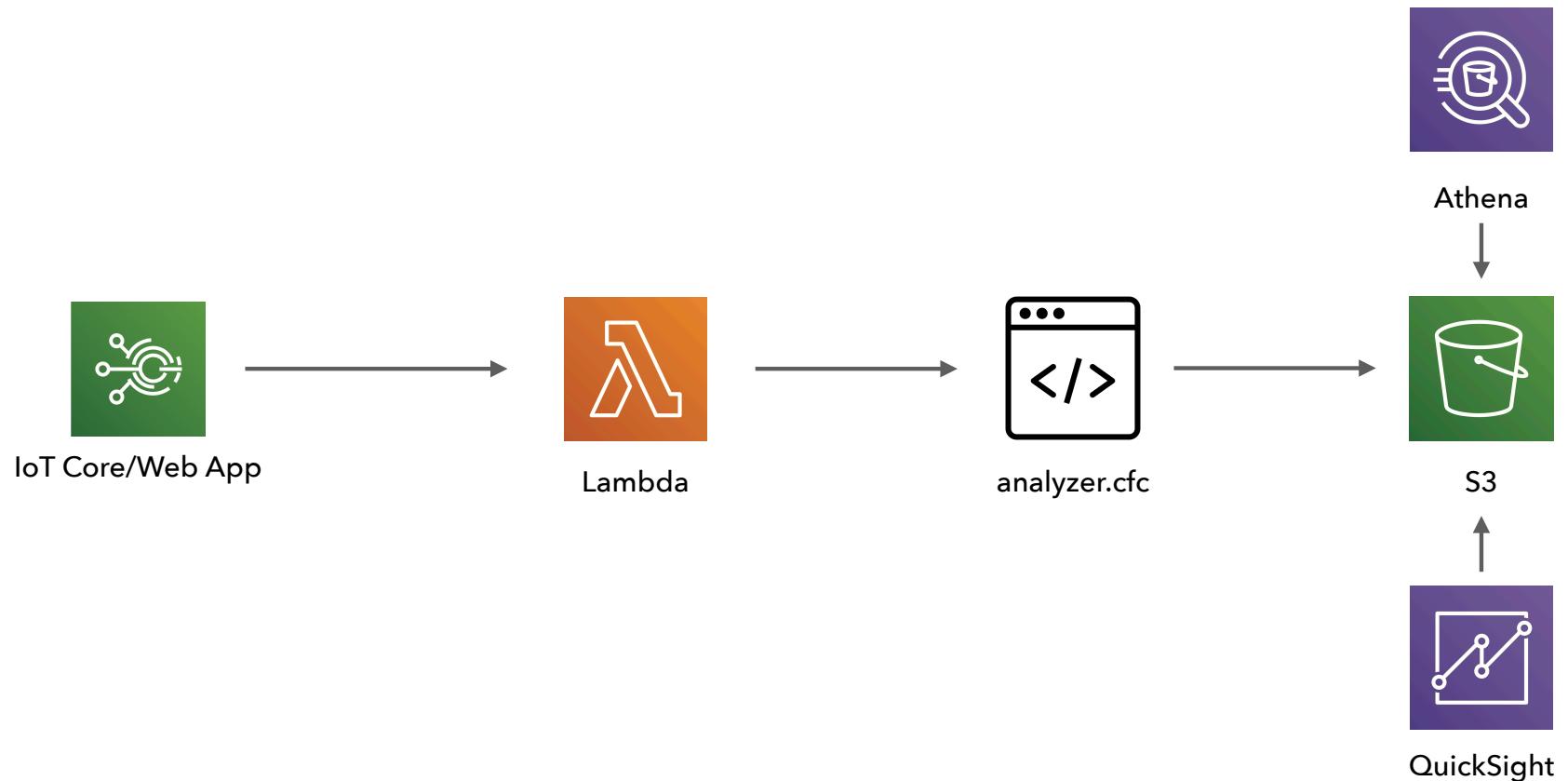


Capture Form Data and Write to Google Sheets in ColdFusion

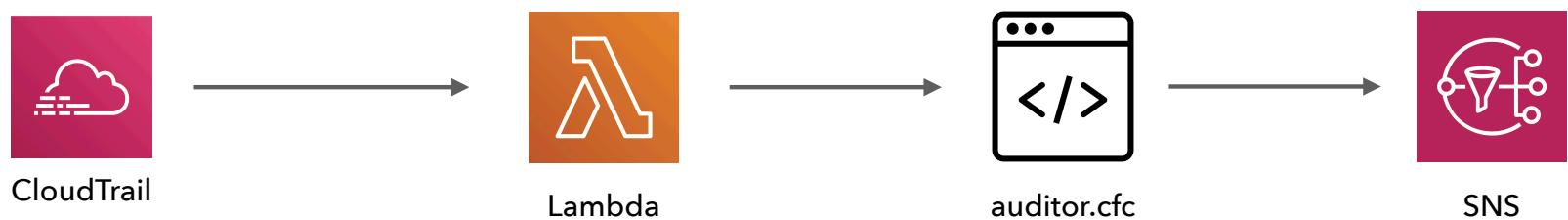


* without running a server!

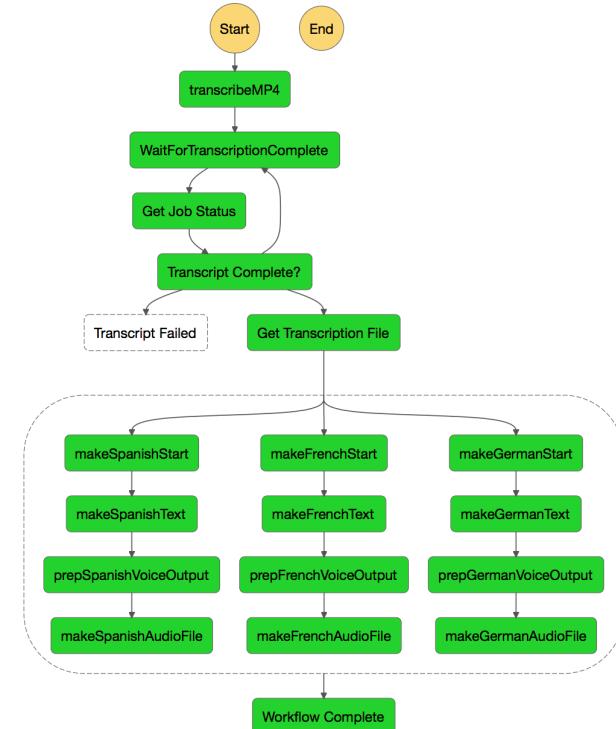
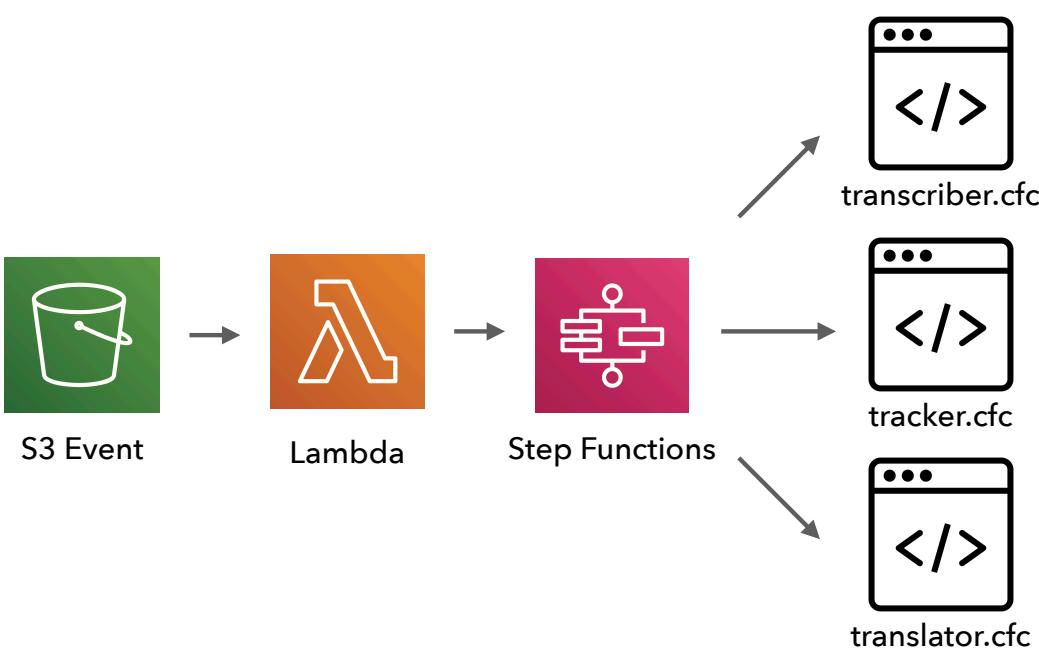
Analyze Clickstream Data in ColdFusion



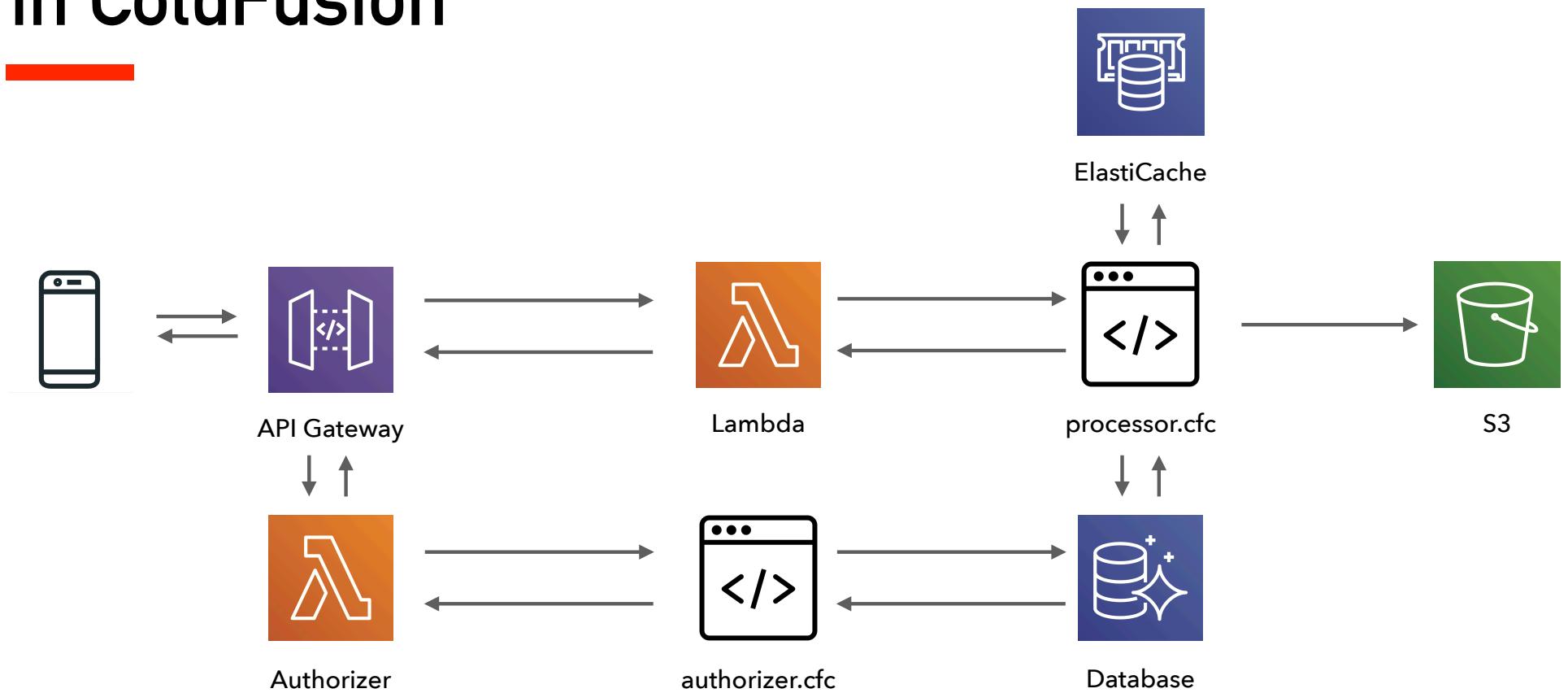
Perform Audits of Your Entire AWS Infrastructure in ColdFusion



Run ColdFusion–Based Lambdas in Your Step Functions Workflows



Build a Full-Stack Serverless Web App in ColdFusion



GO DO!

Resources from this presentation:
brianklaas.net

AWS Playbox:
github.com/brianklaas/awsPlaybox

brian.klaas@gmail.com
@brian_klaas