Developing and deploying serverless applications

Julien Simon
Principal Technical Evangelist, AWS
julsimon@amazon.fr
@julsimon

Agenda

- What's new on AWS Lambda?
- Simplifying development
  - Demo: The Serverless framework
  - Demo: Gordon
  - Demo: Chalice
  - Other tools
- Simplifying deployment
  - Demo: AWS Serverless Application Model
- Additional resources
- Q&A

What’s new on AWS Lambda?

Environment variables for Lambda functions New

You can define Environment Variables as key/value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more.

Environment variables		
var1	value1	✖
var2	value2	✖
Key	*Value*	

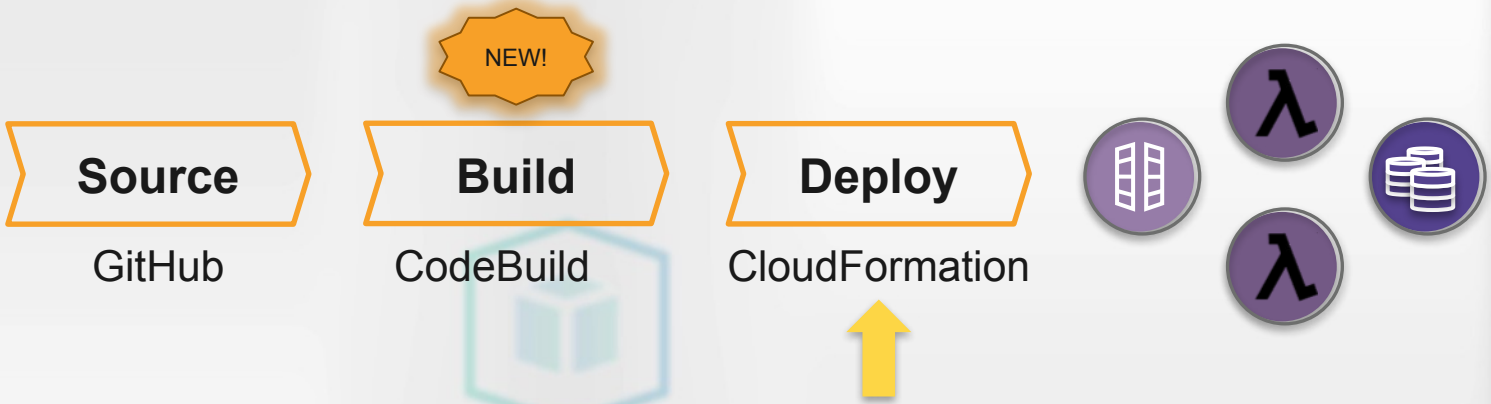
```
var AWS = require('aws-sdk');

    exports.handler = function(event, context, callback) {

        var bucketName = process.env.S3_BUCKET;
        callback(null, bucketName);
        });
    }
```

https://aws.amazon.com/fr/blogs/aws/new-for-aws-lambda-environment-variables-and-serverless-application-model/

Serverless CI/CD pipeline



- Pull source directly from GitHub or AWS CodeCommit using AWS CodePipeline
- Build and package serverless apps with AWS CodeBuild
- Deploy your completed Lambda app with AWS CloudFormation

https://docs.aws.amazon.com/lambda/latest/dg/automating-deployment.html

Dead-letter queue for events

New

Easily create reliable end-to-end event processing solutions

- Sends all unprocessed events to your SQS queue or SNS topic: 3-strike rule
- Preserves events even if your code has an issue or the call was throttled
- Per-function
- Works for all async invokes, including S3 and SNS events

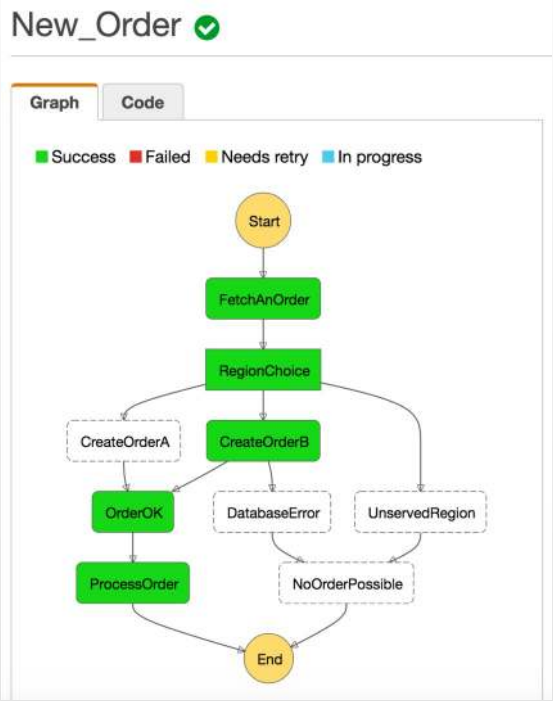
https://docs.aws.amazon.com/lambda/latest/dg/dlq.html

AWS Step Functions

New

Reliably orchestrate multiple Lambda functions

- Attempt a function more than 3X
- Add callbacks to asynchronous functions
- Handle situations that require waiting
- Chain function execution (A→B→C)
- Supports long-running workflows



https://aws.amazon.com/fr/blogs/aws/new-aws-step-functions-build-distributed-applications-using-visual-workflows/

Simplifying Development

Code samples available at https://github.com/juliensimon/aws/tree/master/lambda_frameworks

Typical development workflow

1. Write and deploy a Lambda function
2. Create a REST API with API Gateway
3. Connect the API to the Lambda function
4. Invoke the API
5. Test, debug and repeat ;)

The Serverless framework

formerly known as JAWS: Just AWS Without Servers

- Announced at re:Invent 2015 by Austen Collins and Ryan Pendergast
- Supports Node.js, as well as Python and Java
- Auto-deploys and runs Lambda functions, locally or remotely
- Auto-deploys your Lambda event sources: API Gateway, S3, DynamoDB, etc.
- Creates all required infrastructure with CloudFormation
- Simple configuration in YML

http://github.com/serverless/serverless
https://serverless.com
AWS re:Invent 2015 | (DVO209) https://www.youtube.com/watch?v=D_U6luQ6l90 & https://vimeo.com/141132756

Serverless: “Hello World” API

```
$ serverless create

Edit handler.js, serverless.yml and event.json

$ serverless deploy [--stage stage_name]

$ serverless invoke [local] --function function_name

$ serverless info

$ http $URL

$ serverless remove
```

Demo:
Serverless Framework

Gordon

- Released in Oct'15 by Jorge Batista
- Supports Python, Javascript, Golang, Java, Scala, Kotlin (including in the same project)
- Auto-deploys and runs Lambda functions, locally or remotely
- Auto-deploys your Lambda event sources: API Gateway, CloudWatch Events, DynamoDB Streams, Kinesis Streams, S3
- Creates all required infrastructure with CloudFormation
- Simple configuration in YML

https://github.com/jorgebastida/gordon
https://news.ycombinator.com/item?id=11821295

Gordon: “Hello World” API

```
$ gordon startproject helloworld

$ gordon startapp helloapp

Write hellofunc() function

$ gordon build

$ echo '{"name":"Julien"}' | gordon run helloapp.hellofunc

$ gordon apply [--stage stage_name]

$ http post $URL name=Julien
```

Demo:
Gordon

AWS Chalice

Think of it as a serverless framework for Flask apps

- Released in Jul'16, still in beta
- Just add your Python code
  - Deploy with a single call and zero config
  - The API is created automatically, the IAM policy is auto-generated
- Run APIs locally on port 8000 (similar to Flask)
- Fast & lightweight framework
  - 100% *boto3* calls (AWS SDK for Python) → fast
  - No integration with CloudFormation → no creation of event sources

https://github.com/awslabs/chalice
https://aws.amazon.com/blogs/developer/preview-the-python-serverless-microframework-for-aws/

AWS Chalice: “Hello World” API

```
$ chalice new-project helloworld

Write your function in app.py

$ chalice local

$ chalice deploy

$ export URL=`chalice url`

$ http $URL

$ http put $URL/hello/julien

$ chalice logs [ --include-lambda-messages ]
```

AWS Chalice: PUT/GET in S3 bucket

```
$ chalice new-project s3test

Write your function in app.py

$ chalice local

$ http put http://localhost:8000/objects/doc.json value1=5 value2=8

$ http get http://localhost:8000/objects/doc.json

$ chalice deploy [stage_name]

$ export URL=`chalice url`

$ http put $URL/objects/doc.json value1=5 value2=8

$ http get $URL/objects/doc.json
```

Demo:
Chalice

Summing things up

Serverless

The most popular serverless framework

Built with and for Node.js. Python and Java: YMMV

Rich features, many event sources

Not a web framework

Gordon

Great challenger!

Node.js, Python, Java, Scala, Golang

Comparable to Serverless feature-wise

Not a web framework

Chalice

AWS project, in beta

Python only

Does only one thing, but does it great

Dead simple, zero config

Flask web framework

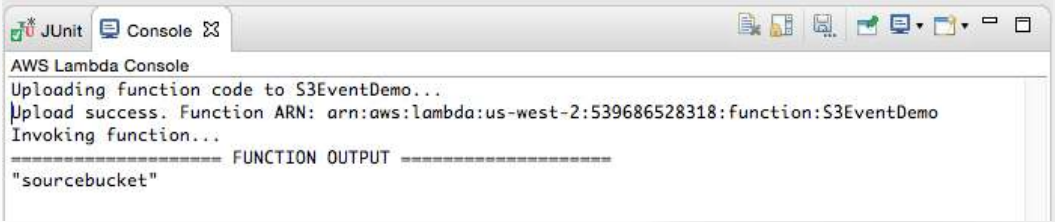
More Lambda frameworks

- Kappa https://github.com/garnaat/kappa
  - Released Dec'14 by Mitch Garnaat, author of boto and the AWS CLI (still maintained?)
  - Python only, multiple event sources
- Apex https://github.com/**apex**/**apex**
  - Released in Dec'15 by TJ Holowaychuk
  - Python, Javascript, Java, Golang
  - Terraform integration to manage infrastructure for event sources
- Zappa https://github.com/Miserlou/Zappa
  - Released in Feb'16 by Rich Jones
  - Python web applications on AWS Lambda + API Gateway
- Docker-lambda https://github.com/lambci/docker-lambda
  - Released in May'16 by Michael Hart
  - Run functions in Docker images that "replicate" the live Lambda environment

2 Java tools for AWS Lambda

Eclipse plug-in

- Code, test and deploy Lambda from Eclipse
- Run your functions locally and remotely
- Test with local events and Junit4
- Deploy standalone functions, or with the AWS Serverless Application Model (Dec'16)



Serverless Java Container

- Run Java RESTful APIs as-is
- Default implementation of the Java servlet `HttpServletRequest` `HttpServletResponse`
- Support for Java frameworks such as Jersey or Spark

https://java.awsblog.com/post/TxWZES6J1RSQ2Z/Testing-Lambda-functions-using-the-AWS-Toolkit-for-Eclipse
https://aws.amazon.com/blogs/developer/aws-toolkit-for-eclipse-serverless-application
https://github.com/awslabs/aws-serverless-java-container

Simplifying Deployment

AWS Serverless Application Model (SAM)

formerly known as Project Flourish

New

- CloudFormation extension released in Nov'16 to bundle Lambda functions, APIs & events
- 3 new CloudFormation resource types
  - *AWS::Serverless::Function*
  - *AWS::Serverless::Api*
  - *AWS::Serverless::SimpleTable*
- 2 new CloudFormation CLI commands
  - *'aws cloudformation package'*
  - *'aws cloudformation deploy'*
- Integration with CodeBuild and CodePipeline for CI/CD
- Expect SAM to be integrated in most / all frameworks

https://aws.amazon.com/fr/blogs/compute/introducing-simplified-serverless-application-deployment-and-management
https://github.com/awslabs/serverless-application-model/

AWS Serverless Application Model

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources: GetHtmlFunction:
Type: AWS::Serverless::Function
Properties:
CodeUri: s3://flourish-demo-bucket/todo_list.zip
Handler: index.gethtml
Runtime: nodejs4.3
Policies: AmazonDynamoDBReadOnlyAccess
Events:
GetHtml: Type: Api
Properties: Path: /{proxy+} Method: ANY
ListTable: Type: AWS::Serverless::SimpleTable
```

Functions

APIs

Storage

AWS Serverless Application Model

```
AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
Resources: GetHtmlFunction:
Type: AWS::Serverless::Function
Properties:
CodeUri: s3://flourish-demo-bucket/todo_list.zip
Handler: index.gethtml
Runtime: nodejs4.3
Policies: AmazonDynamoDBReadOnlyAccess
Events:
GetHtml: Type: Api
Properties: Path: /{proxy+} Method: ANY
ListTable: Type: AWS::Serverless::SimpleTable
```

REPLACES:

```
AWSTemplateFormatVersion: '2010-09-09'
Resources:
GetHtmlFunctionGetHtmlPermissionProd:
Type: AWS::Lambda::Permission
Properties:
Action: lambda:invokeFunction
Principal: apigateway.amazonaws.com
FunctionName:
Ref: GetHtmlFunction
SourceArn:
Fn::Sub: arn:aws:execute-api:$
{AWS::Region}:${AWS::AccountId}:$
{ServerlessRestApi}/Prod/ANY/*
ServerlessRestApiProdStage:
Type: AWS::ApiGateway::Stage
Properties:
DeploymentId:
Ref: ServerlessRestApiDeployment
RestApiId:
Ref: ServerlessRestApi
StageName: Prod
ListTable:
Type: AWS::DynamoDB::Table
Properties:
ProvisionedThroughput:
WriteCapacityUnits: 5
ReadCapacityUnits: 5
AttributeDefinitions:
- AttributeName: id
AttributeType: S
KeySchema:
- KeyType: HASH
AttributeName: id
GetHtmlFunction:
Type: AWS::Lambda::Function
Properties:
Handler: index.gethtml
Code:
S3Bucket: flourish-demo-bucket
S3Key: todo_list.zip

Role:
Fn::GetAtt:
- GetHtmlFunctionRole
- Arn
Runtime: nodejs4.3
GetHtmlFunctionRole:
Type: AWS::IAM::Role
Properties:
ManagedPolicyArns:
-
arn:aws:iam::aws:policy/AmazonDynamoDBR
eadOnlyAccess
-
arn:aws:iam::aws:policy/service-role/AWSLam
bdaBasicExecutionRole
AssumeRolePolicyDocument:
Version: '2012-10-17'
Statement:
- Action:
- sts:AssumeRole
Effect: Allow
Principal:
Service:
- lambda.amazonaws.com
ServerlessRestApiDeployment:
Type: AWS::ApiGateway::Deployment
Properties:
RestApiId:
Ref: ServerlessRestApi
Description: 'RestApi deployment id:
127e3fb91142ab1ddc5f5446adb094442581a
90d'
StageName: Stage
GetHtmlFunctionGetHtmlPermissionTest:
Type: AWS::Lambda::Permission
Properties:
Action: lambda:invokeFunction
Principal: apigateway.amazonaws.com
FunctionName:
Ref: GetHtmlFunction
SourceArn:
Fn::Sub: arn:aws:execute-api:$
{AWS::Region}:${AWS::AccountId}:$
{ServerlessRestApi}/*/ANY/*
ServerlessRestApi:
Type: AWS::ApiGateway::RestApi
Properties:
Body:
info:
version: '1.0'
title:
Ref: AWS::StackName
paths:
"/{proxy+}":
x-amazon-apigateway-any-method:
x-amazon-apigateway-integration:
httpMethod: ANY
type: aws_proxy
uri:
Fn::Sub: arn:aws:apigateway:$
{AWS::Region}:lambda:path/2015-03-
31/functions/${GetHtmlFunction.Arn}/
invocations
responses: {}
swagger: '2.0'
```

SAM: Open Specification

New

Apache 2.0 licensed

GitHub project

https://github.com/awslabs/serverless-application-model

AWS Serverless Application Model (SAM)

Version 2016-10-31

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD
"MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The AWS Serverless Application Model (SAM) is licensed under The Apache License, Version 2.0

Introduction

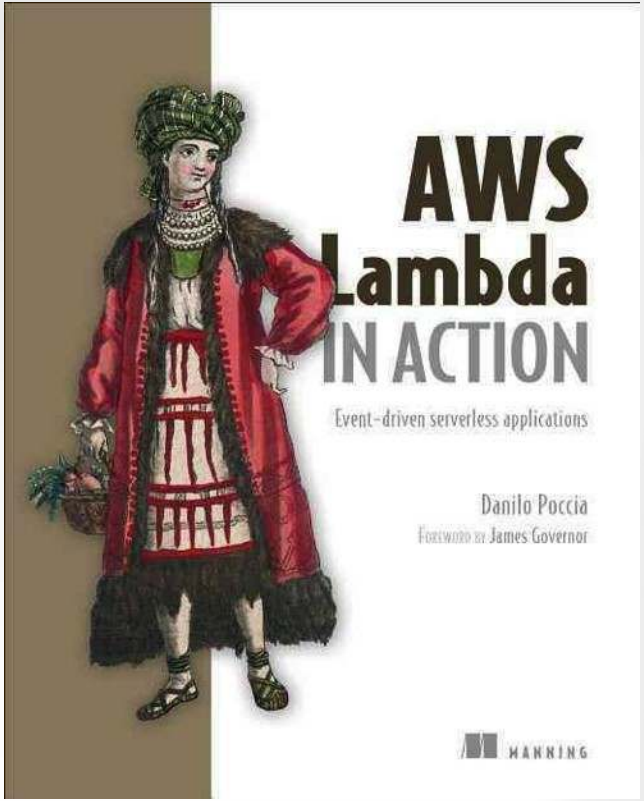
AWS SAM is a model used to define serverless applications on AWS.

Serverless applications are applications composed of functions triggered by events. A typical se
one or more AWS Lambda functions triggered by events such as object uploads to Amazon S3
and API actions. Those functions can stand alone or leverage other resources such as Amazon D
buckets. The most basic serverless application is simply a function.

Demo:
Serverless Application Model

Additional resources

The only Lambda book you need to read



Written by AWS Technical Evangelist Danilo Poccia

Just released!

https://www.amazon.com/Aws-Lambda-Action-Event-driven-Applications/dp/1617293717/

New Lambda videos from re:Invent 2016

AWS re:Invent 2016: What's New with AWS Lambda (SVR202)https://www.youtube.com/watch?v=CwxWhyGteNc

AWS re:Invent 2016: Serverless Apps with AWS Step Functions (SVR201)
https://www.youtube.com/watch?v=75MRve4nv8s

AWS re:Invent 2016: Real-time Data Processing Using AWS Lambda (SVR301)
https://www.youtube.com/watch?v=VFLKOy4GKXQ

AWS re:Invent 2016: Serverless Architectural Patterns and Best Practices (ARC402)
https://www.youtube.com/watch?v=b7UMoc1iUYw

AWS re:Invent 2016: Bringing AWS Lambda to the Edge (CTD206)
https://www.youtube.com/watch?v=j26novaqF6M

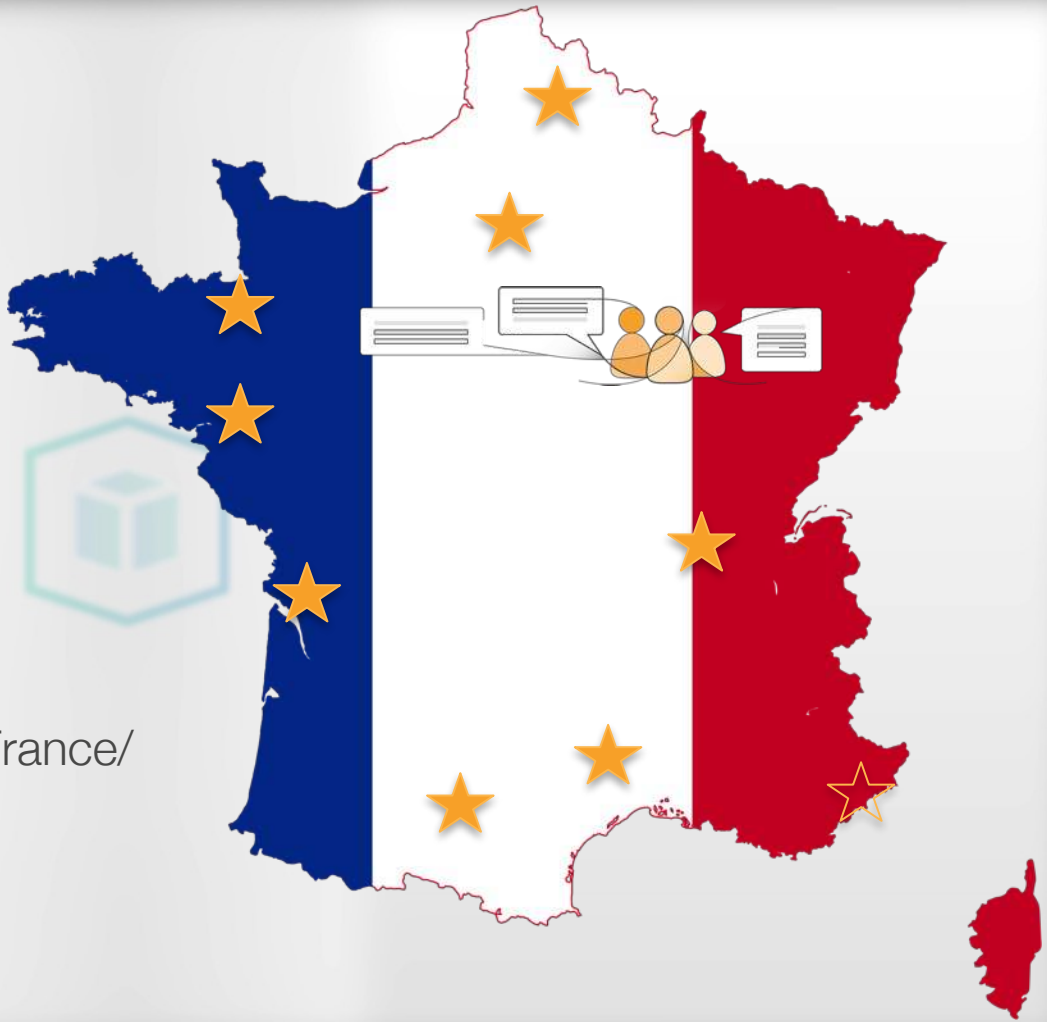
AWS re:Invent 2016: Ubiquitous Computing with Greengrass (IOT201)
https://www.youtube.com/watch?v=XQQjX8GTEko

AWS User Groups

Lille
Paris
Rennes
Nantes
Bordeaux
Lyon
Montpellier
Toulouse
Côte d'Azur (soon!)

facebook.com/groups/AWSFrance/

@aws_actus



Merci !

Julien Simon
Principal Technical Evangelist, AWS
julsimon@amazon.fr
@julsimon