A 60-minute tour of AWS Compute

EC2 | Elastic Beanstalk | EC2 Container Service | Lambda

Julien Simon, Principal Technical Evangelist
@julsimon

Meetup AWS User Group Rennes #1
29/03/2016

What to expect from this talk

- An introduction to all four AWS Compute technologies
- A good understanding on when to use them best
- Demos
    - Launching an EC2 instance from the CLI
    - Deploying a Ruby on Rails web app with Elastic Beanstalk
    - Deploying a PHP web app with ECS
    - Implementing an API with API Gateway and a Lambda function in Python
    - Reacting to S3 events with a Lambda function in Java
- Answers to your questions ☺

AWS Compute technologies

EC2

Amazon Elastic Compute Cloud (EC2) provides resizable compute capacity in the cloud.

Elastic Beanstalk

AWS Elastic Beanstalk is an application container for deploying and managing applications.

Lambda

AWS Lambda is a compute service that runs your code in response to events and automatically manages the compute resources for you.

EC2 Container Service

Amazon ECS allows you to easily run and manage Docker containers across a cluster of Amazon EC2 instances.

Amazon EC2

- Infrastructure as a Service, launched in 2006
- Based on virtual machines (“EC2 instances”) and images (“Amazon Machine Image”, “AMI”)
- Many instance types for different needs: general purpose, compute, memory, GPU, etc.
- Users can pick from Amazon-supported AMIs, vendor-supported AMIs (“EC2 Marketplace”) or they can build their own

- All-inclusive: networking (“Virtual Private Cloud”), storage (“Elastic Block Storage”), firewalling (“Security Group”), load balancing (“Elastic Load Balancing”), high availability (“Availability Zones”), automatic scaling (“Auto-scaling groups”), monitoring (“Cloudwatch”)
- Pay on an hourly basis

The best option if you need full control over your instances

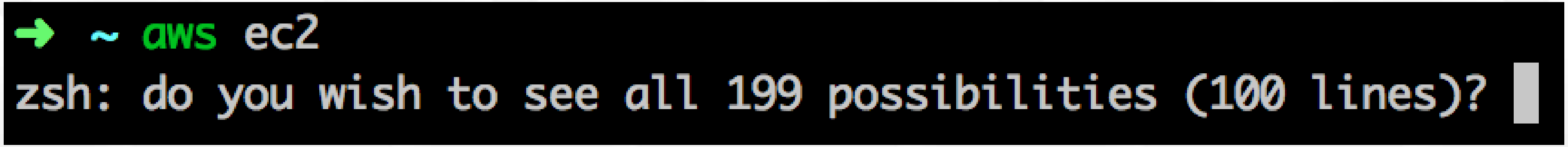
Use Reserved Instances and Spot instances for massive savings

Amazon EC2 demo

Launch an Amazon Linux instance
in the default VPC with the default security group

```
$ aws ec2 run-instances --image-id ami-e1398992
--instance-type t2.micro --key-name lab2
--security-group-ids sg-09238e6d --region eu-west-1
```

This is the most important command ;)
Take some time to experiment with the '*aws ec2*' command line



```
➜  ~ aws ec2
zsh: do you wish to see all 199 possibilities (100 lines)?
```

Amazon Elastic Beanstalk

- Platform as a Service, launched in 2011
- Supports PHP, Java, .NET, Node.js, Python, Go, Ruby IIS, Tomcat and Docker containers

- Developer-friendly CLI : ‘*eb*’
- Uses AWS Cloudformation to build all required resources

- Built-in monitoring (Amazon Cloudwatch), networking (Amazon VPC), load balancing (Amazon ELB) and scaling (Auto-scaling)
- Relational data tier is available through Amazon Relational Data Service (RDS)

- No charge for the service itself

The simplest and most intuitive way to deploy your applications

This should really be your default option for deployment

Amazon Elastic Beanstalk demo

1. Create a new Rails application

2. Add a resource to the application

3. Declare a new Rails application in Amazon Elastic Beanstalk

4. Create an environment and launch the application

Create a new Rails application

```
$ rails new blog

$ cd blog

$ git init

$ git add .

$ git commit -m "Initial version"
```

Add a ‘post’ resource to the application

```
$ rails generate scaffold post title:string body:text

$ bundle exec rake db:migrate

$ git add .

$ git commit -m "Add post resource"

$ rails server

$ open http://localhost:3000/posts
```

Initialize a Ruby application

```
$ eb init blog --platform Ruby --region eu-west-1

$ git add .gitignore

$ git commit -m "Ignore .elasticbeantalk directory"
```

Create a ‘blog-dev’ environment

Single instance (no auto scaling, no load balancing), t2.micro instance size (default value)

```
$ eb create blog-dev --single --keyname aws-eb \
  --envvars SECRET_KEY_BASE=`rake secret`
$ eb deploy
$ eb terminate blog-dev –force
```

For more information on Elastic Beanstalk (load balancing, high availability, RDS with Postgres)
http://www.slideshare.net/JulienSIMON5/deploying-a-simple-rails-application-with-aws-elastic-beanstalk

Amazon EC2 Container Service

- Container as a Service, launched in 2015
- Built-in clustering, state management, scheduling and high availability
- EC2 Container Registry (ECR): private Docker registry hosted in AWS

- Developer-friendly CLI : ‘*ecs-cli*’
- Uses AWS Cloudformation to build all required resources
- Supports Docker 1.9.1, including Docker Compose files

- No charge for the service itself

A simple and scalable way to manage your Dockerized applications

Amazon ECS demo

1. Build a Docker image for a simple PHP web app
2. Push it to an ECR repository
3. Create an ECS cluster
4. Deploy and scale the containerized web app

Amazon ECR demo: build and push container

```
$ git clone https://github.com/awslabs/ecs-demo-php-simple-app.git
$ cd ecs-demo-php-simple-app
$ docker build -t php-simple-app .
$ docker tag php-simple-app:latest \
  ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/php-simple-app:latest

$ aws ecr get-login --region us-east-1
<run docker login command provided as output>
$ docker push \
  ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com php-simple-app:latest
```

Amazon ECS demo: write a Compose file

```
php-demo:
  image: ACCOUNT_ID.dkr.ecr.us-east-1.amazonaws.com/php-simple-app:latest
  cpu_shares: 100
  mem_limit: 134217728
  ports:
    - "80:80"
  entrypoint:
    - "/usr/sbin/apache2"
    - "-D"
    - "FOREGROUND"
```

Amazon ECS demo: launch cluster & service

```
$ ecs-cli configure --cluster myCluster --region eu-west-1
$ ecs-cli up --keypair lab2 --capability-iam --size 1 \
  --instance-type t2.micro
$ ecs-cli compose service up

$ ecs-cli scale --size 3 --capability-iam
$ ecs-cli compose service scale 3

$ ecs-cli compose service delete
$ ecs-cli down myCluster --force
```

For more information on ECS: http://www.slideshare.net/JulienSIMON5/amazon-ecs-january-2016

AWS Lambda

- Code as a Service, launched in 2014
- Supports Java 8, Python 2.7 and Node.js v0.10.36

- Build event-driven applications
- Build APIs in conjunction with Amazon API Gateway
- Interact with other AWS services (S3, DynamoDB, etc)
- Log automatically to CloudWatch Logs

- Pay as you go: number of requests + execution time (100ms slots)

The future: serverless applications and NoOps ☺

AWS Lambda demo (Python)

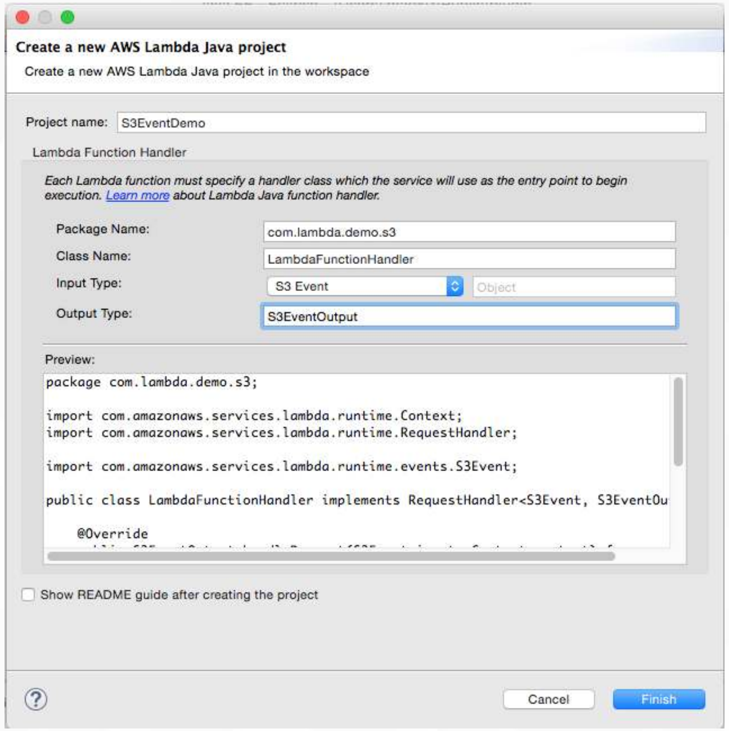
1. Write a simple Lambda function adding two integers
2. Create a REST API with API Gateway (resource + method)
3. Create a new stage
4. Deploy our API to the stage
5. Invoke the API with ‘*curl*’

A simple Lambda function in Python

```
def lambda_handler(event,context):
    result = event['value1'] + event['value2']
    return result
```

```
$ curl -H "Content-Type: application/json"
-X POST -d "{\"value1\":5, \"value2\":7}" https://API_ENDPOINT/STAGE/RESOURCE
12%
```

AWS Lambda in Java with Eclipse



Create a new AWS Lambda Java project

Create a new AWS Lambda Java project in the workspace

Project name: S3EventDemo

Lambda Function Handler

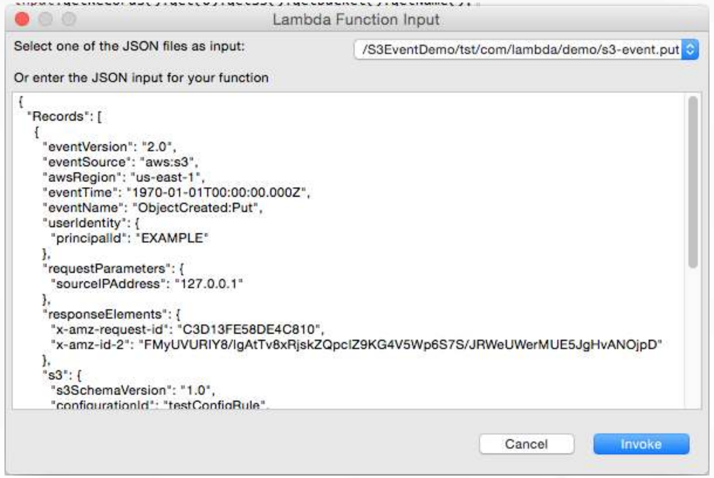
Each Lambda function must specify a handler class which the service will use as the entry point to begin execution. Learn more about Lambda Java function handler.

Package Name:	com.lambda.demo.s3
Class Name:	LambdaFunctionHandler
Input Type:	S3 Event
Output Type:	S3EventOutput

Preview:

```
package com.lambda.demo.s3;

import com.amazonaws.services.lambda.runtime.Context;
import com.amazonaws.services.lambda.runtime.RequestHandler;

import com.amazonaws.services.lambda.runtime.events.S3Event;

public class LambdaFunctionHandler implements RequestHandler<S3Event, S3EventOu

    @Override
```

Show README guide after creating the project

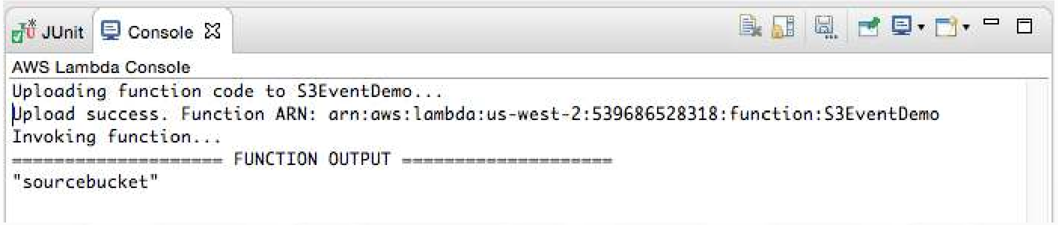


Lambda Function Input

Select one of the JSON files as input: /S3EventDemo/tst/com/lambda/demo/s3-event.put

Or enter the JSON input for your function

```
{
  "Records": [
   {
    "eventVersion": "2.0",
    "eventSource": "aws:s3",
    "awsRegion": "us-east-1",
    "eventTime": "1970-01-01T00:00:00.000Z",
    "eventName": "ObjectCreated:Put",
    "userIdentity": {
      "principalId": "EXAMPLE"
    },
    "requestParameters": {
      "sourceIPAddress": "127.0.0.1"
    },
    "responseElements": {
      "x-amz-request-id": "C3D13FE58DE4C810",
      "x-amz-id-2": "FMyUVURIY8/IgAtTv8xRjskZQpcIZ9KG4V5Wp6S7S/JRWeUWerMUE5JgHvANOjpD"
    },
    "s3": {
      "s3SchemaVersion": "1.0",
      "configurationId": "testConfigRule",
```



AWS Lambda Console

```
Uploading function code to S3EventDemo...
Upload success. Function ARN: arn:aws:lambda:us-west-2:539686528318:function:S3EventDemo
Invoking function...
==================== FUNCTION OUTPUT ====================
"sourcebucket"
```

https://java.awsblog.com/post/TxWZES6J1RSQ2Z/Testing-Lambda-functions-using-the-AWS-Toolkit-for-Eclipse

AWS Lambda demo (Java)

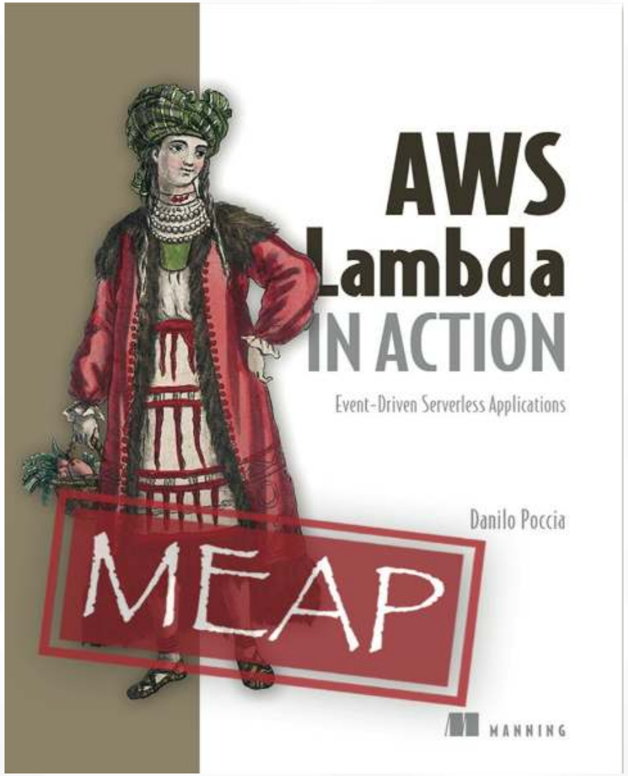
1. In Eclipse, write a simple Lambda function triggered by an S3 event
2. Unit-test the function with Junit
3. Using the AWS Eclipse plug-in, upload and run the function in AWS
4. Run the function again in the AWS Console

AWS Lambda in Node.js with Serverless framework

- Run/test AWS Lambda functions locally, or remotely
- Auto-deploys & versions your Lambda functions
- Auto-deploys your REST API to AWS API Gateway
- Auto-deploys your Lambda events
- Support for multiple stages
- Support for multiple regions within stages
- Manage & deploy AWS CloudFormation resources

https://github.com/serverless/serverless

Upcoming book on AWS Lambda


AWS Lambda IN ACTION

Event-Driven Serverless Applications

Danilo Poccia

MEAP

MANNING

Written by AWS Technical Evangelist Danilo Poccia

Early release available at:

https://www.manning.com/books/aws-lambda-in-action

And now the trip begins. Time to explore!

https://aws.amazon.com/fr/documentation/gettingstarted/
https://docs.aws.amazon.com
https://aws.amazon.com/fr/blogs/compute/

Next events

sido
The Connected Business

April 6-7 (Lyon)

DEVOXX France

April 20-22

dotScale

April 25

May 31st

AWSome Day

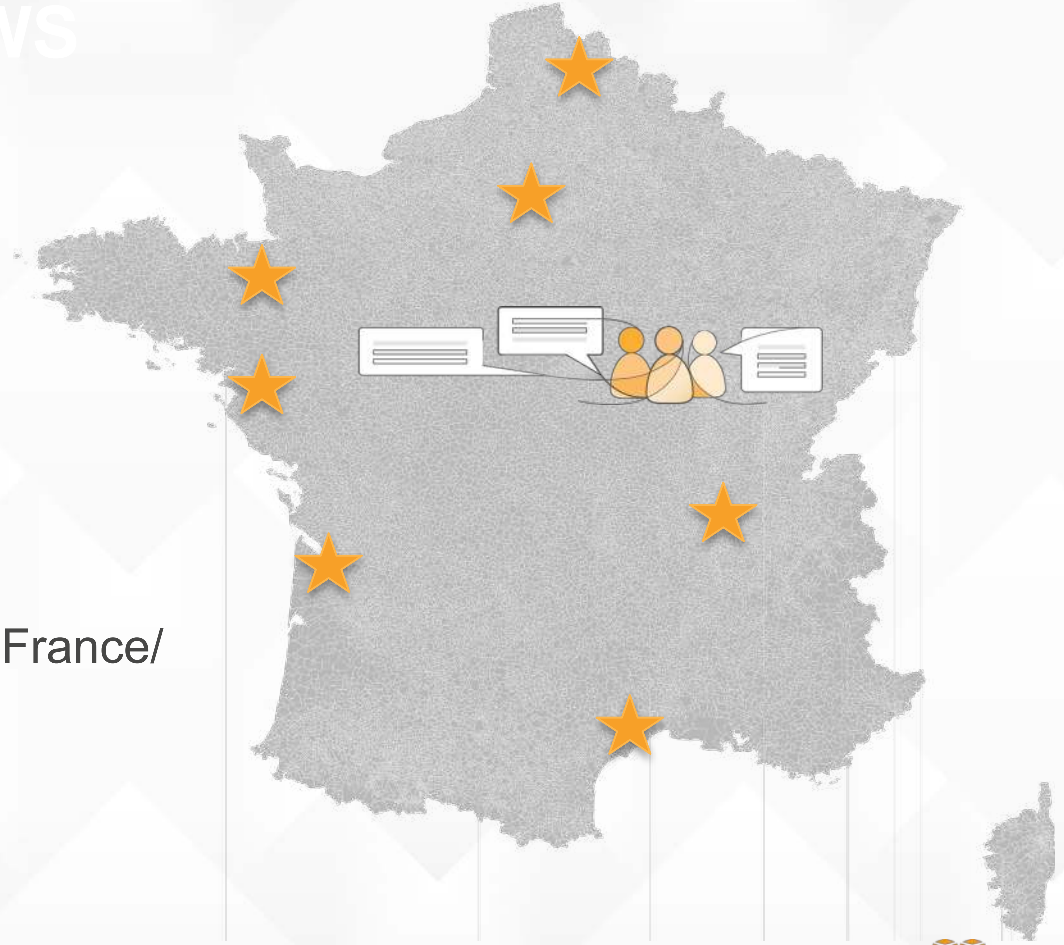
June 28
September 27
December 6

amazon web services

AWS User Groups

Meetup

Lille
Paris
Rennes
Nantes
Bordeaux
Lyon
Montpellier



facebook.com/groups/AWSFrance/

@aws_actus

Merci !

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