



Building a serverless data pipeline

Julien Simon, Principal Technical Evangelist, AWS
julsimon@amazon.fr
[@julsimon](https://twitter.com/julsimon)



Requirements For Internet-scale Apps



Simplicity



Scalability



Low Cost



Reliability



Low Latency

Building Blocks For Internet-scale Apps

Storage



Amazon
S3

Data Store



Amazon
DynamoDB

Compute



Evolution of Computing

Weeks



On-premises

Minutes



Amazon EC2

Seconds



Amazon EC2
Container Service

A photograph of a stage presentation. A person is standing in the center of a stage, facing away from the camera towards a large screen. The screen displays the text "No Server Is Easier To Manage Than No Server". The stage is lit with warm, golden light, and the background screen has a subtle pattern of diagonal lines. Two podiums with the Amazon logo are visible on either side of the speaker.

No Server Is Easier To Manage Than No Server

Werner Vogels, CTO, Amazon.com
AWS re:Invent 2015

AWS Lambda

- No infrastructure to manage: deploy only **functions** in Java, Python and Node.js
- Built-in **scalability** and **high-availability**
- Works nicely with other AWS managed services
- Build **event-driven applications**
- Build RESTful **APIs** in conjunction with Amazon API Gateway
- **Pay as you go:**
number of requests + execution time (100ms slots)

Managed services
+
AWS Lambda
=
Serverless architecture

Another way to put it...

Tim Wagner,
General Manager,
AWS Lambda

Serverless conference,
NYC, May 2016



Selected serverless customers



THREAT INTELLIGENCE
AND ANALYTICS



MOBILE CHAT
APP



AD DATA ANALYTICS
AND ROUTING



MOBILE APP
ANALYTICS



IMAGE CONTENT
FILTERING



WEB
APPLICATIONS



WEB APPLICATIONS



DATA
PROCESSING



CLOUD
TELEPHONY



REAL-TIME VIDEO
AD BIDDING

NORDSTROM

PRODUCT
RECOMMENDATION



THOMSON REUTERS

NEWS CONTENT
PROCESSING

BUSTLE

NEWS CONTENT
PROCESSING



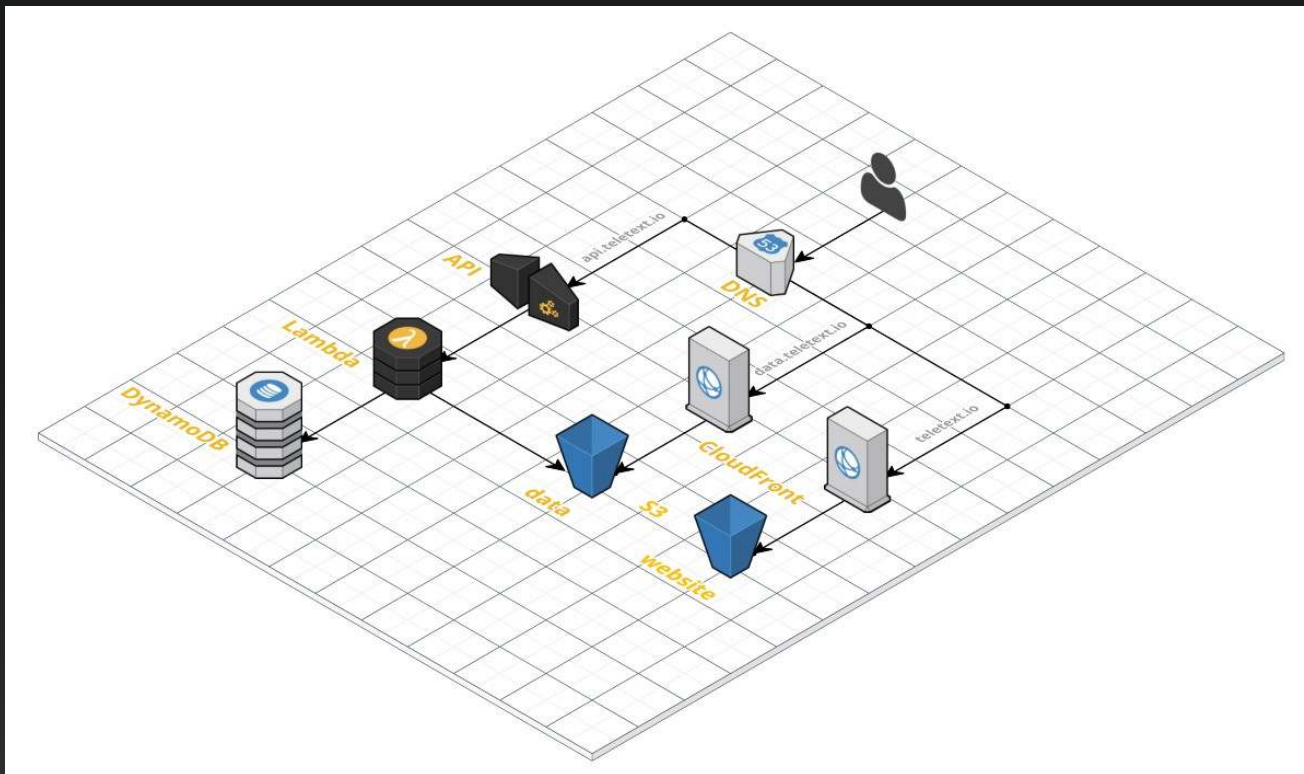
Benchling

GENE SEQUENCE
SEARCH



GAME METRICS ANALYTICS

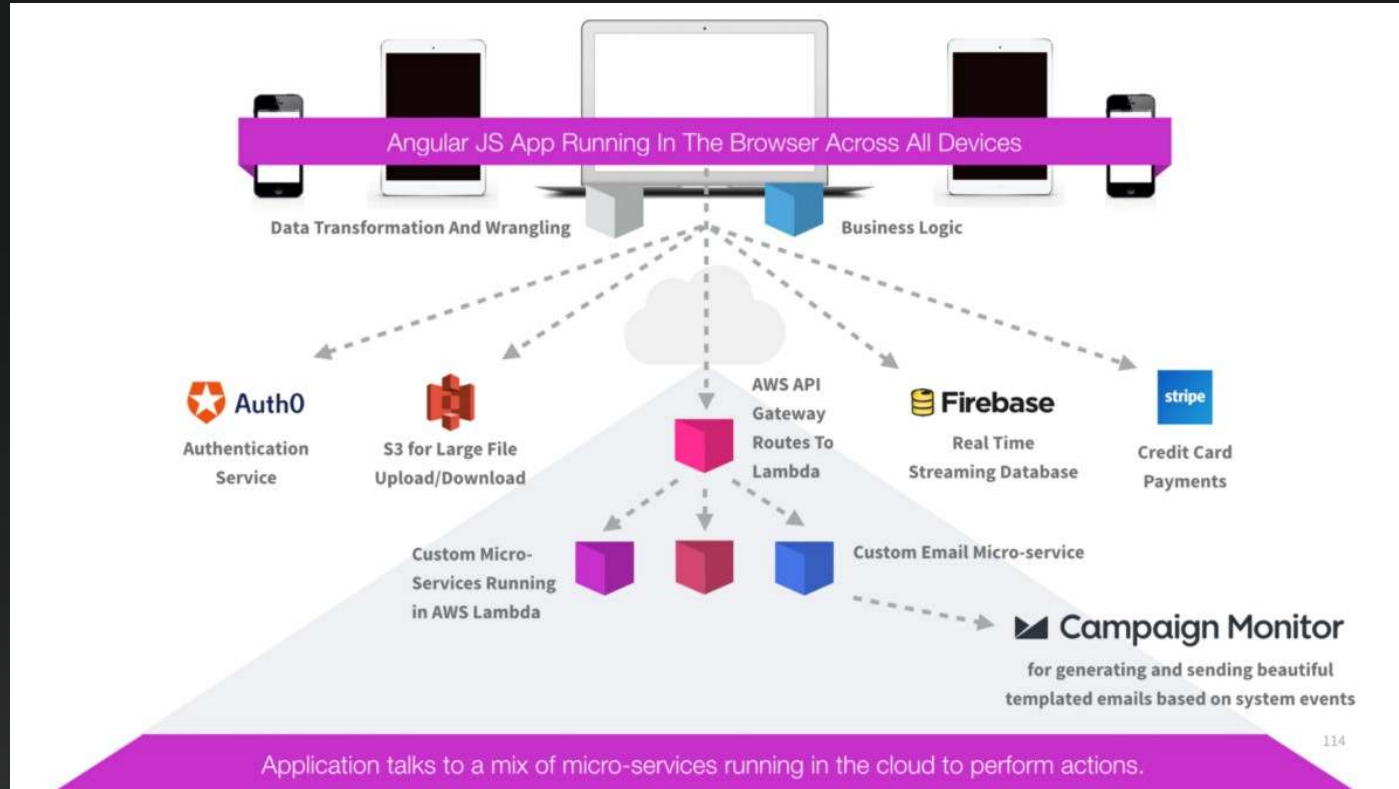
Instant.cm: 100% Serverless



<https://blog.instant.cm/a-serverless-architecture-with-zero-maintenance-and-infinite-scalability-b00c2ceb4c2b>

<http://highscalability.com/blog/2015/12/7/the-serverless-start-up-down-with-servers.html>

A Cloud Guru: 100% Serverless



114

AWS Lambda 'Hello World' (Python)

1. Write a simple Lambda function in Python
2. Create a REST API with API Gateway (resource + POST method)
3. Deploy the API
4. Invoke the API with '*curl*'

A simple Lambda function in Python

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

```
aws lambda create-function --function-name myFunc \  
--handler myFunc.lambda_handler --runtime python2.7 \  
--zip-file fileb://myFunc.zip --memory-size 128 \  
--role arn:aws:iam::ACCOUNT_NUMBER:role/lambda_basic_execution
```

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

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:

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:

Class Name:

Input Type:

Output Type:

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, S3EventOutput> {

    @Override
```

☐ Show README guide after creating the project

Lambda Function Input

Select one of the JSON files as input:

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": "FMMyUVURIY8/IgAtTv8xRjskZQpciZ9KG4V5Wp6S7/JRWELUwerMUE5JgHVANOjpD"
      },
      "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 with the Serverless framework

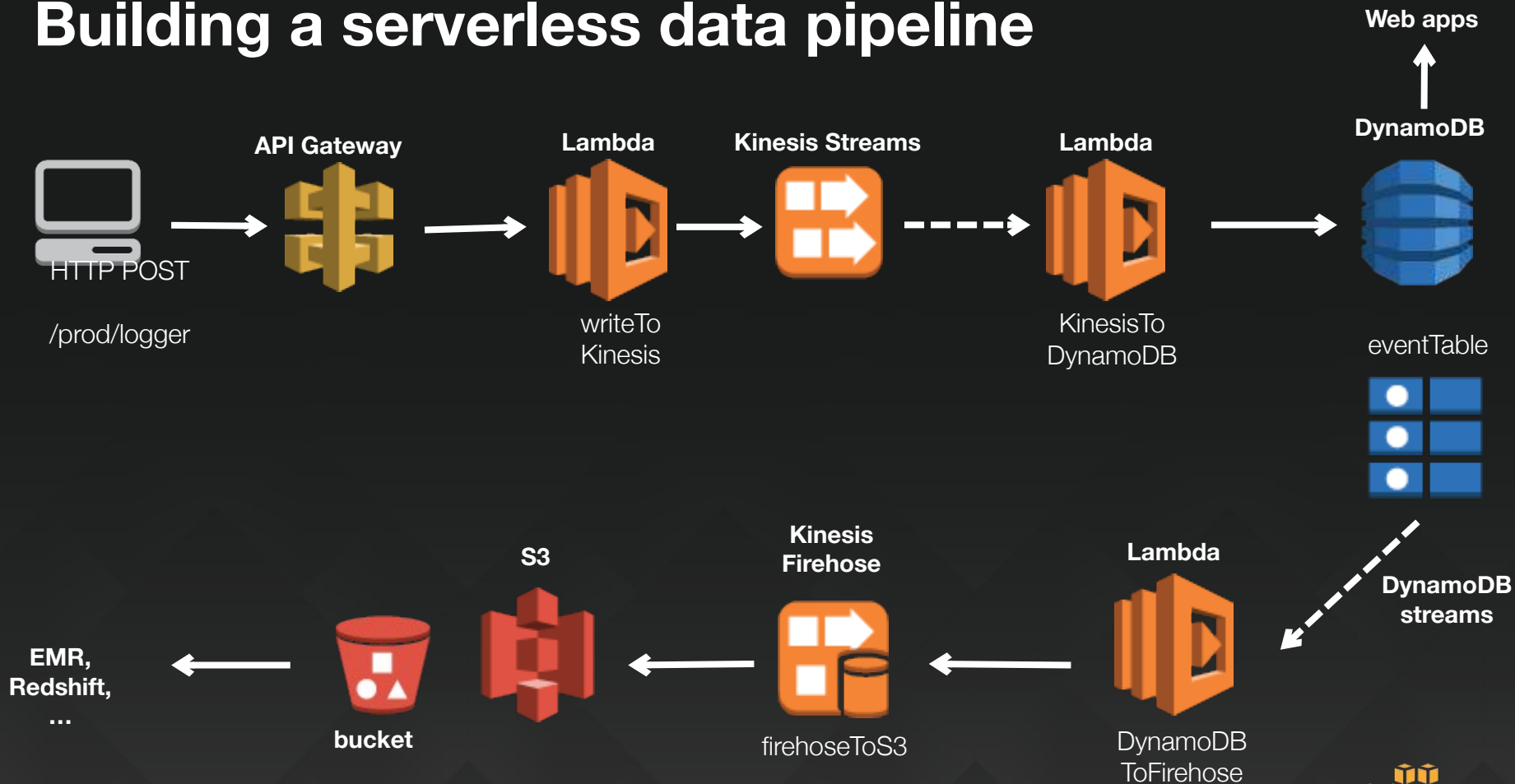


<http://github.com/serverless/serverless>

- 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

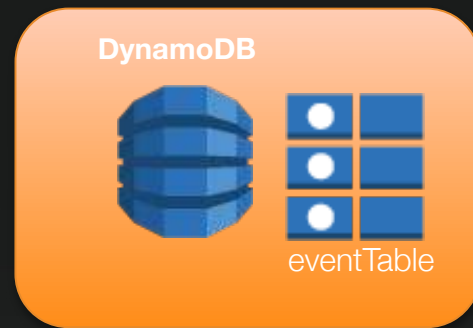
Other notable frameworks: Chalice (Python), Zappa (Python), Apex (golang)

Building a serverless data pipeline



Step 1: create DynamoDB table

```
aws dynamodb create-table \  
--table-name eventTable \  
--attribute-definitions \  
AttributeName=userId,AttributeType=N \  
AttributeName=timestamp,AttributeType=N \  
--key-schema \  
AttributeName=userId,KeyType=HASH \  
AttributeName=timestamp,KeyType=RANGE \  
--provisioned-throughput ReadCapacityUnits=5,WriteCapacityUnits=5 \  
--stream-specification StreamEnabled=true,StreamViewType=NEW_IMAGE
```



Step 2: IAM role for Lambda function

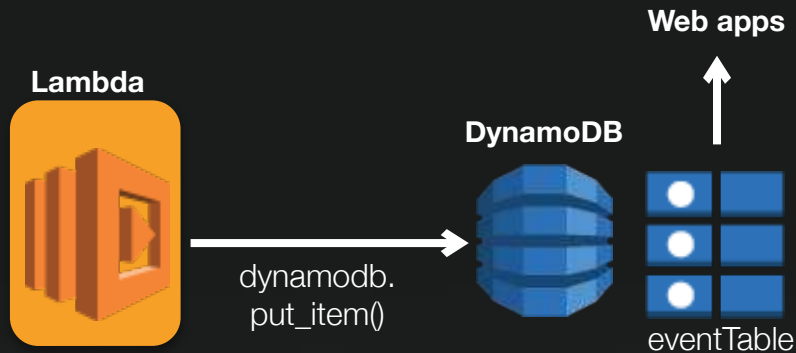
```
aws iam create-role \  
--role-name writeToDynamoDB_role \  
--assume-role-policy-document file://lambda_trust_policy.json
```

```
aws iam create-policy \  
--policy-name writeToDynamoDB_policy \  
--policy-document file://writeToDynamoDB_policy.json
```

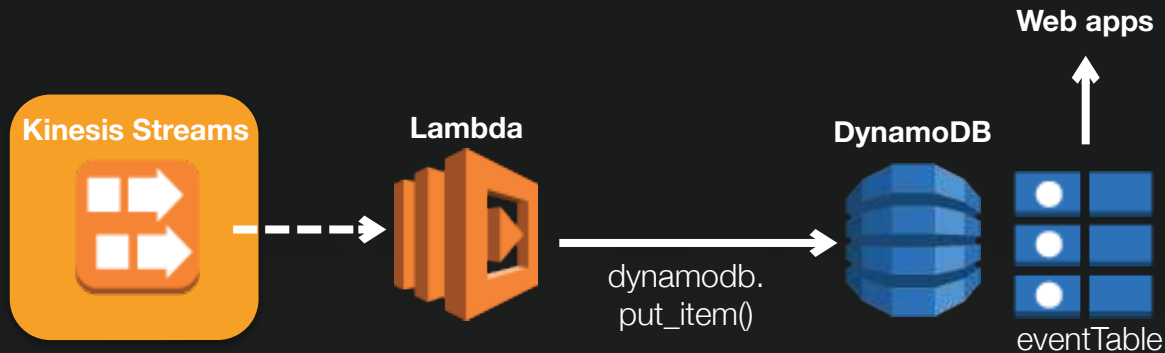
```
aws iam attach-role-policy \  
--role-name writeToDynamoDB_role \  
--policy-arn WRITETODYNAMODB_POLICY_ARN
```

Step 3: create Lambda function

```
aws lambda create-function \  
--function-name writeToDynamoDB \  
--role WRITETODYNAMO_DB_ROLE \  
--zip-file fileb://writeToDynamoDB.zip \  
--handler writeToDynamoDB.lambda_handler \  
--runtime python2.7 \  
--memory-size 128 \  
--description "Write events to DynamoDB"
```



Step 4: create Kinesis Stream



```
aws kinesis create-stream --stream-name APItoDynamoDB --shard-count 1
```

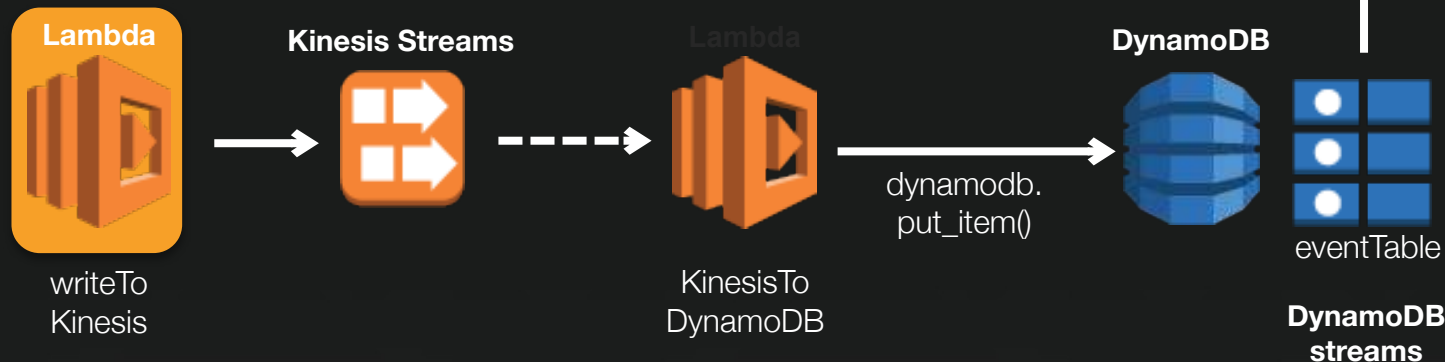
Step 5: IAM role for Lambda function

```
aws iam create-role \  
--role-name writeToKinesis_role \  
--assume-role-policy-document file://lambda_trust_policy.json
```

```
aws iam create-policy \  
--policy-name writeToKinesis_policy \  
--policy-document file://writeToKinesis_policy.json
```

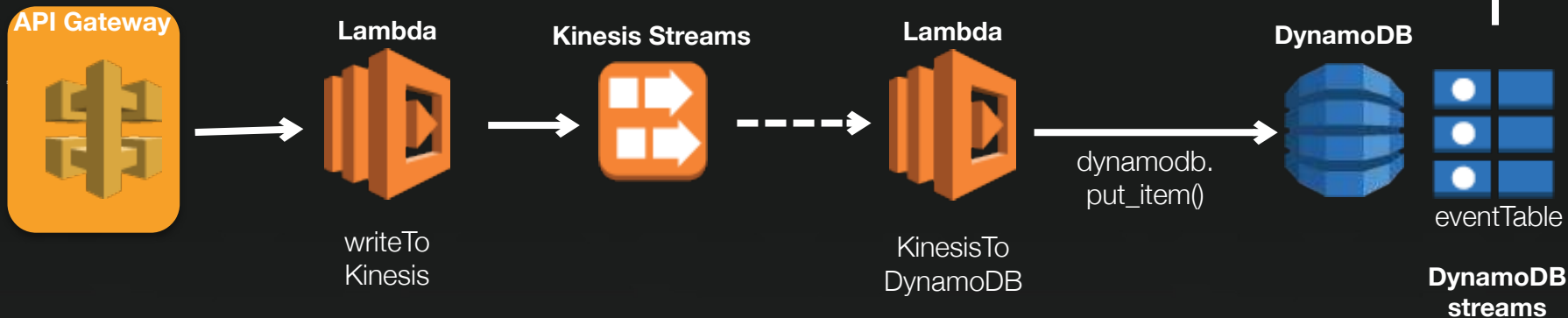
```
aws iam attach-role-policy \  
--role-name writeToKinesis_role \  
--policy-arn WRITETOKINESIS_POLICY_ARN
```

Step 6: create Lambda function



```
aws lambda create-function \  
--function-name writeToKinesis\  
--role WRITETOKINESIS_ROLE \  
--zip-file fileb://writeToKinesis.zip \  
--handler writeToKinesis.lambda_handler \  
--runtime python2.7 \  
--memory-size 128 \  
--description "Write events to Kinesis"
```

Step 7: create API



Painful to do with the CLI: 9 “aws apigateway” calls :-/

- Use the **console**
- Use a **Swagger** File
<http://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-import-api.html>
- Use a **framework** (Serverless, Chalice, etc.)

Step 8: create IAM role

```
aws iam create-role \  
--role-name DynamoDBToFirehose_role \  
--assume-role-policy-document file://lambda_trust_policy.json
```

```
aws iam create-policy \  
--policy-name DynamoDBToFirehose_policy \  
--policy-document file://DynamoDBToFirehose_policy.json
```

```
aws iam attach-role-policy \  
--role-name DynamoDBToFirehose_role \  
--policy-arn DYNAMODBTOFIREHOSE_POLICY_ARN
```


Step 9: create Lambda function and DynamoDB trigger

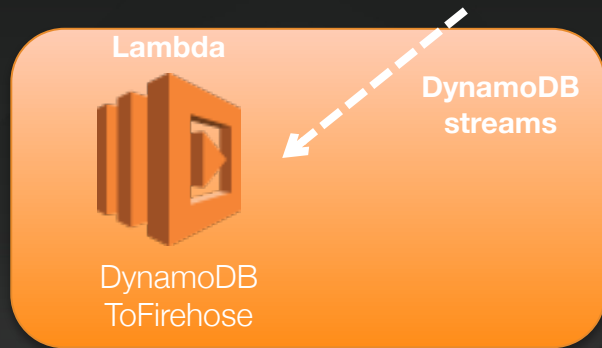
```
aws lambda create-function \  
--function-name DynamoDBToFirehose \  
--role DYNAMODBTOFIREHOSE_ROLE_ARN \  
--zip-file fileb://DynamoDBToFirehose.zip \  
--handler DynamoDBToFirehose.lambda_handler \  
--runtime python2.7 \  
--memory-size 128 \  
--description "Write DynamoDB stream to Kinesis Firehose"
```

```
aws lambda create-event-source-mapping \  
--function-name DynamoDBToFirehose \  
--event-source DYNAMODB_STREAM_ARN \  
--batch-size 10 \  
--starting-position TRIM_HORIZON
```

DynamoDB



eventTable



Step 10: create IAM role

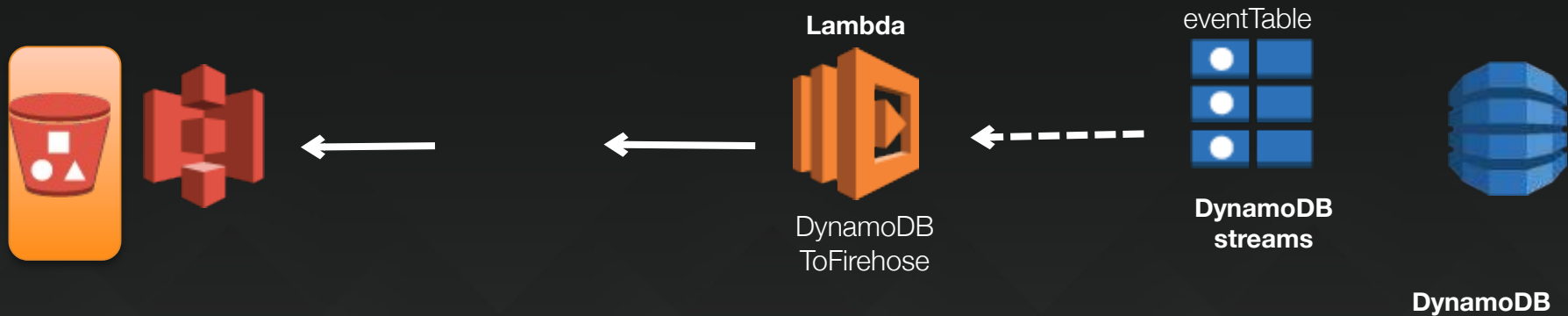
```
aws iam create-role \  
--role-name firehoseToS3_role \  
--assume-role-policy-document file://firehose_trust_policy.json
```

```
aws iam create-policy \  
--policy-name firehoseToS3_policy \  
--policy-document file://firehoseToS3_policy.json
```

```
aws iam attach-role-policy \  
--role-name firehoseToS3_role \  
--policy-arn FIREHOSETOS3_POLICY_ARN
```

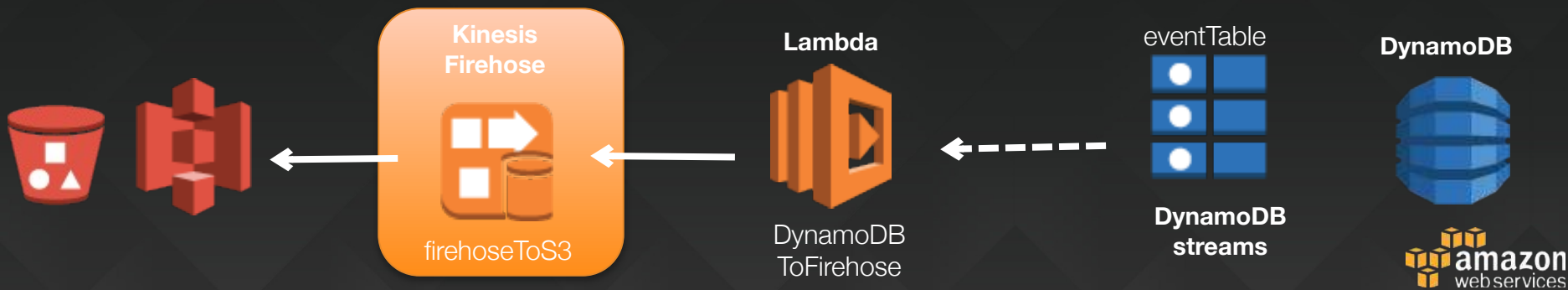
Step 11: create S3 bucket

```
aws s3 mb s3://jsimon-public
```



Step 12: create Kinesis Firehose stream

```
aws firehose create-delivery-stream \  
--delivery-stream-name firehoseToS3 \  
--s3-destination-configuration \  
RoleARN=FIREHOSETOS3_ROLE_ARN, \  
BucketARN="arn:aws:s3:::jsimon-public", \  
Prefix="firehose", \  
BufferingHints=\{SizeInMBs=1,IntervalInSeconds=60\}, \  
CompressionFormat="GZIP", \  
EncryptionConfiguration={NoEncryptionConfig="NoEncryption"}
```



Building a serverless data pipeline



Lines of code: 16

Number of servers: zero

Performance & scalability: maximum

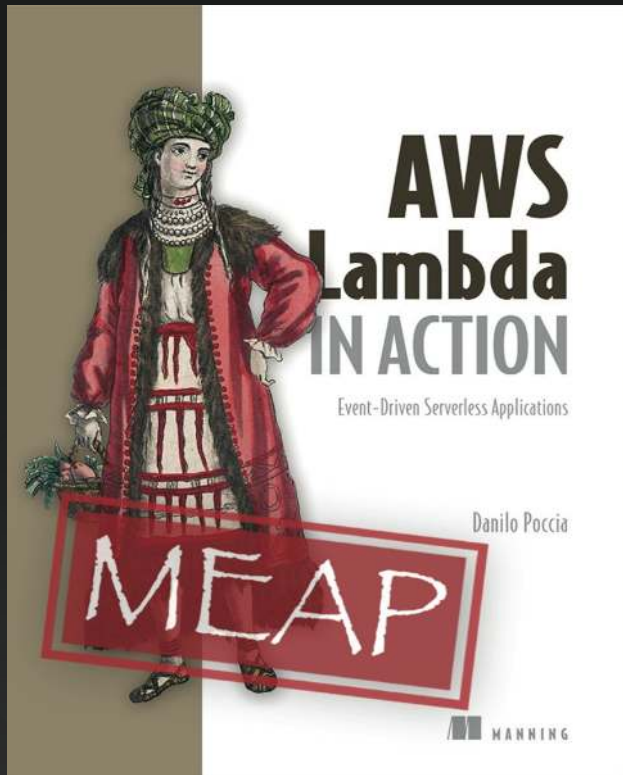


Ready for some testing?

<http://api.julien.org>



Upcoming book on AWS Lambda



Written by AWS Technical Evangelist Danilo Poccia

Early release available at:

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

Going further

AWS re:Invent 2014 | (MBL202) NEW LAUNCH: Getting Started with AWS Lambda

<https://www.youtube.com/watch?v=UFj27laTWQA>

AWS re:Invent 2015 | (DEV203) Amazon API Gateway & AWS Lambda to Build Secure and Scalable APIs

<https://www.youtube.com/watch?v=ZBxWZ9bgd44>

AWS re:Invent 2015 | (DVO209) JAWS: The Monstrously Scalable Serverless Framework

https://www.youtube.com/watch?v=D_U6luQ6l90

<https://github.com/serverless/serverless>

AWS re:Invent 2015 | (ARC308) The Serverless Company Using AWS Lambda

<https://www.youtube.com/watch?v=U8ODkSCJpJU>

AWS re:Invent 2015 | (CMP407) Lambda as Cron: Scheduling Invocations in AWS Lambda

<https://www.youtube.com/watch?v=FhJxTlq81AU>

Reference architectures

<http://www.allthingsdistributed.com/2016/06/aws-lambda-serverless-reference-architectures.html>



Dank u wel!

Julien Simon, Principal Technical Evangelist, AWS

julsimon@amazon.fr

[@julsimon](#)

