

# Getting started with AWS Lambda and Serverless

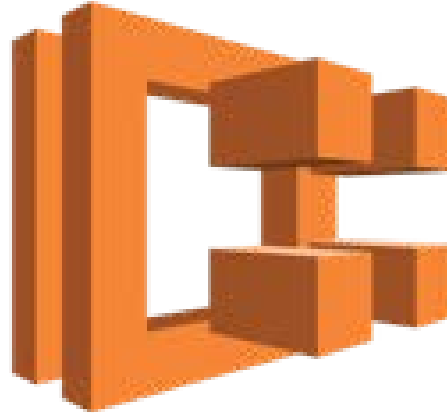
Abby Fuller, AWS  
@abbyfuller



# AWS Compute offerings



**Amazon EC2**  
Virtual servers in the  
Cloud



**Amazon ECS**  
Container management service  
for running Docker containers  
on EC2



**Amazon Lambda**  
Serverless compute platform for  
stateless code execution in  
response to events

# Let's talk about Lambda!



Serverless compute platform for stateless  
code execution in response to events

# Benefits of AWS Lambda



## AWS Lambda

526 points | cpenner461 | 3 years ago | 163 comments | (<http://aws.amazon.com/lambda/>)

## Show HN: Chromeless – Headless Chrome Automation on AWS Lambda

385 points | schickling | 2 months ago | 90 comments | (<https://github.com/graphcool/chromeless>)

## AWS Lambda Is Not Ready for Prime Time

315 points | sebg | a year ago | 137 comments | ([https://www.datawire.io/3-reasons-aws-lambda-not-ready-prime-time/?utm\\_content=buffer5d2eb&utm\\_medium=social&utm\\_source=twitter.com](https://www.datawire.io/3-reasons-aws-lambda-not-ready-prime-time/?utm_content=buffer5d2eb&utm_medium=social&utm_source=twitter.com))

## Cheap Recurring Payments with Stripe and AWS Lambda

299 points | sl8r | 4 months ago | 124 comments | (<http://normal-extensions.com/2017/05/05/simple-recurring/>)

## New for AWS Lambda – Environment Variables and Serverless Application Model

264 points | luhn | 10 months ago | 83 comments | (<https://aws.amazon.com/blogs/aws/new-for-aws-lambda-environment-variables-and-serverless-application-model/>)

## Introducing Apex – Serverless Architecture with AWS Lambda

234 points | johanbrook | 2 years ago | 40 comments | (<https://medium.com/@tjholowaychuk/introducing-apex-800824ffaa70>)

## AWS Lambda as a back end for a single-page app

229 points | arange | 2 years ago | 73 comments | (<http://lg.io/2015/05/16/the-future-is-now-and-its-using-aws-lambda.html>)

## Use AWS Lambda to self-host the comments for your blog

226 points | jarmitage | a year ago | 121 comments | (<https://jimpick.com/2016/05/05/introducing-lambda-comments/>)

## Ask HN: How was your experience with AWS Lambda in production?

215 points | chetanmelkani | 3 months ago | 155 comments

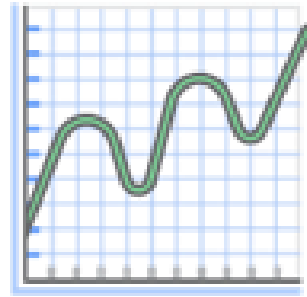
I would like to hear from people who have used AWS Lambda in production, how was there experience with it. It would be great if you have references to project repositories.

## Gordon: a tool to create, wire and deploy AWS Lambdas

# More important benefits



**No servers to  
manage**



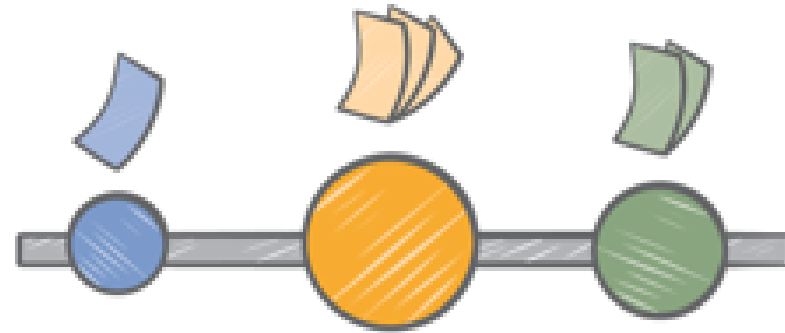
**Continuous  
scaling**



**No idle/cold  
servers**

# Pay per request

- Buy compute time in 100ms increments
- Low request charge
- No hourly, daily or monthly minimums
- No per-device fees
- Never pay for idle



**Free tier: 1 million requests, and 400,000 GBs of compute every month, for every customer.**

# Working with Lambda



## **Bring your own code**

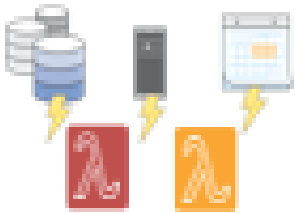
- Node.js, Java, Python, C#
- Bring your own libraries (even native ones)



## **Simple resource model**

- Select power rating from 128MB to 1.5GB
- CPU and network allocated proportionately
- Metrics show usage

# Working with Lambda



## Flexible use

- Call or send events
- Integrated with other AWS services
- Build serverless ecosystems

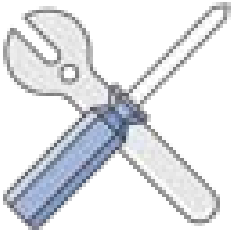


## Flexible authorization

- Securely grant access to resources, including VPCs
- Fine-grained control over what can call your functions

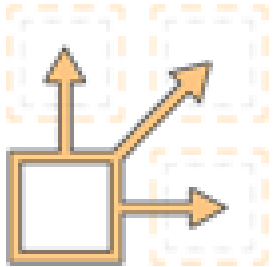


# Working with Lambda



## Programming model

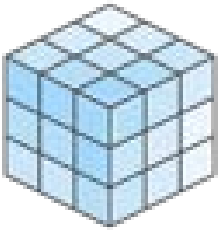
- Built-in AWS SDK
- Front end is Lambda
- Use processes, threads, /tmp and sockets normally



## Authoring functions

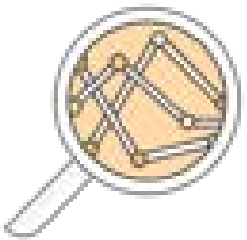
- Author directly with WYSIWYG editor in console
- Package code as .zip and upload to Lambda or S3
- Plugins for Eclipse and Visual Studio
- Command line tools

# Working with Lambda



## Stateless

- Persist data using Amazon S3, RDS, ElastiCache or non-relational database
- No affinity to infrastructure (can't login to host)



## Monitoring and logging

- Built in metrics for requests, latency, errors and throttles
- Built in logging with CloudWatch

# Common use cases



## Data triggers

- Trigger functions on data updates in S3, SNS, etc.



## Big data

- Real time processing of streaming data updates using Kinesis.



## Control systems

- Customize responses and workflows to state changes within AWS.



## Serverless backends

- Execute server-side backend logic

A more specific use case: Lambda + S3



# Dynamic data ingestion with Lambda + S3

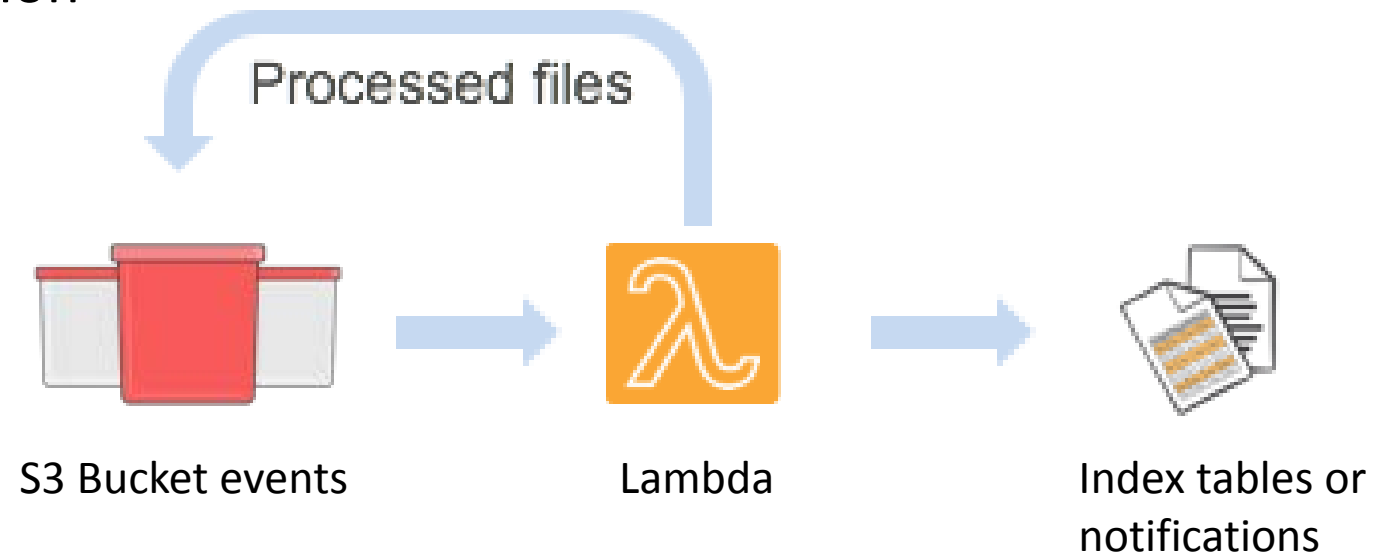


# Customers using S3 and Lambda



## Apply custom logic to process content being uploaded into Amazon S3

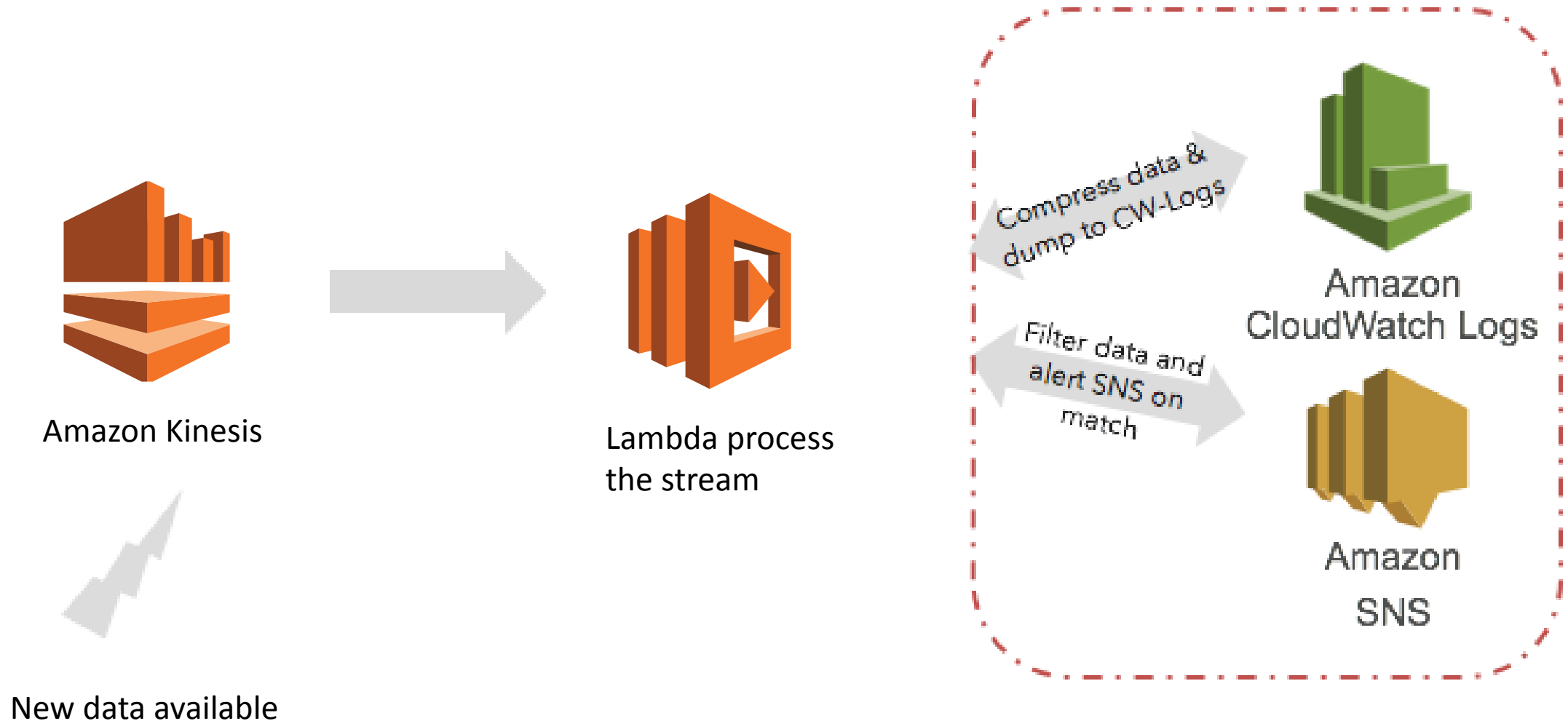
- Watermarking / thumbnail creation
- Transcoding
- Indexing and de-duplication
- Aggregation and filtering
- Pre processing
- Content validation
- WAF updates



# Lambda and Kinesis

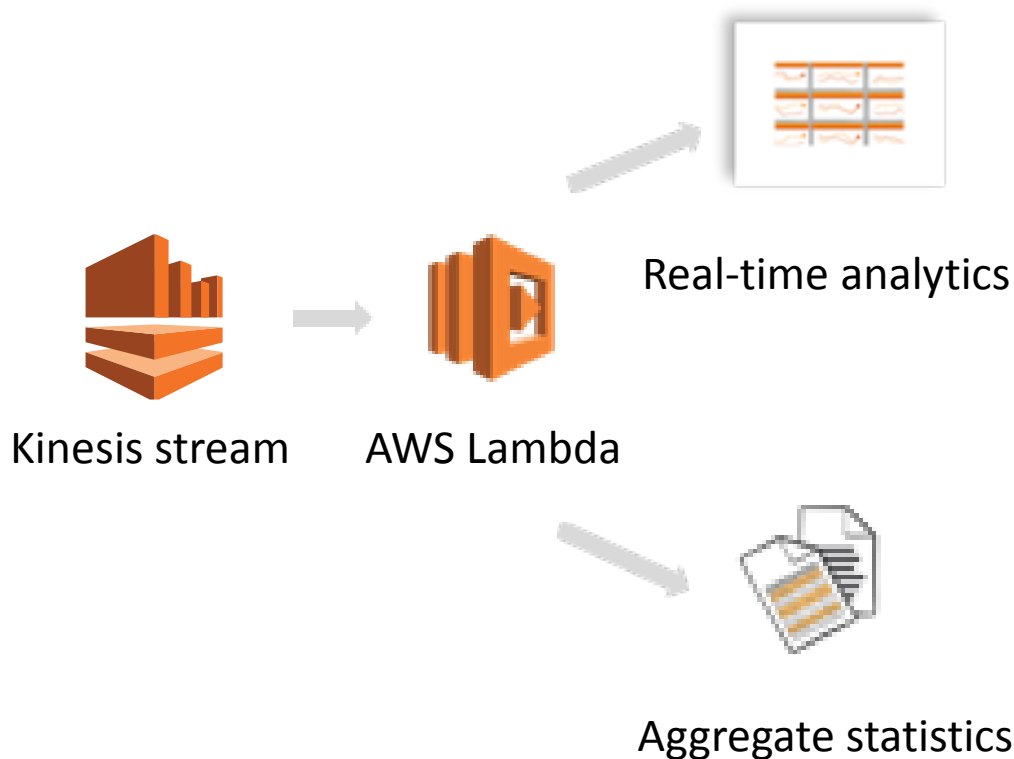


# Dynamic data ingestion with S3 + Kinesis





# Customers using Lambda + Kinesis

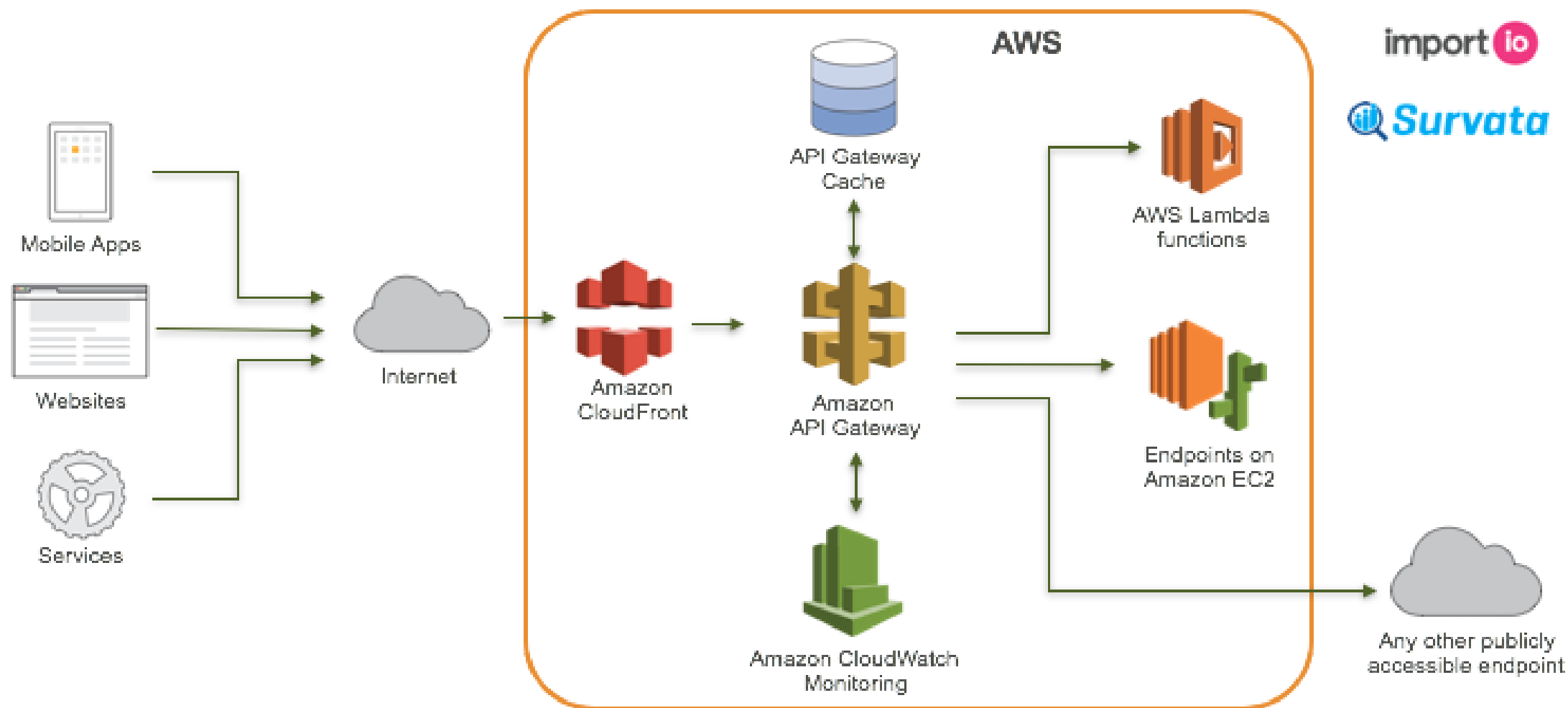


**Apply custom logic to process data being uploaded through Kinesis stream**

- Client activity tracking
- Metrics generation
- Data cleansing
- Log filtering & routing
- Indexing and searching
- Live alarms and notifications

# Lambda powered APIs

# An API call flow

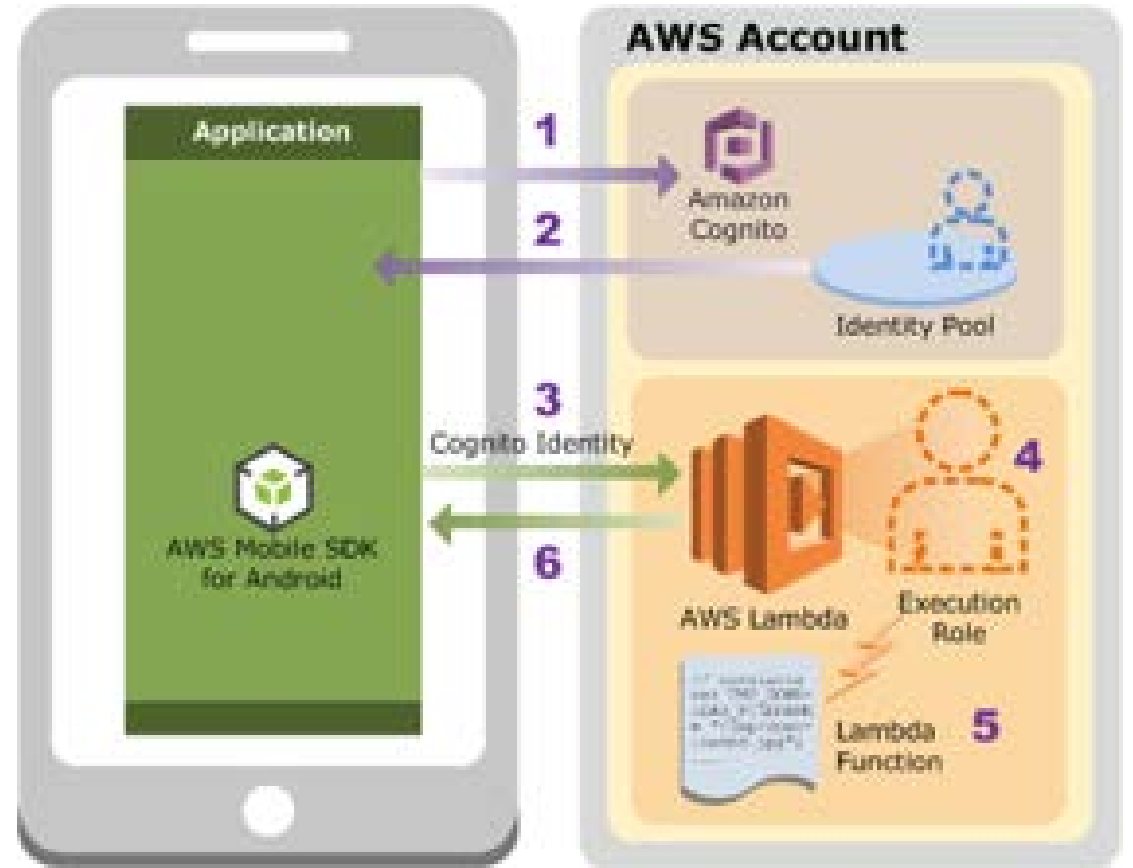


# Lambda + Cognito and mobile apps



# Building mobile backends with Lambda

- No backend experience? No problem.
- You can use Lambda as the backend for mobile apps!
- Easy personalization for users and devices



# Other use cases

# Scheduled events (cron)

- Start or stop an environment at a specific time
- Log cleanup
- Batch data jobs
- Alarm clock
- Infrastructure automation
- Scheduled backups

### Using AWS Lambda with Scheduled Events

You can create a Lambda function and direct AWS Lambda to execute it on a regular schedule. You can specify a fixed rate (for example, execute a Lambda function every hour or 15 minutes), or you can specify a Cron expression. For more information on expressions, schedules, see [Schedule Expressions Using Rate or Cron](#).

This functionality is available when you create a Lambda function using the AWS Lambda console or the AWS CLI. To configure it using the AWS CLI, see [Run an AWS Lambda Function on a Schedule Using the AWS CLI](#). The console provides the **CloudWatch Events - Schedule** as an event source. At the time of creating a Lambda function, you choose this event source and specify a time interval.

If you have made any manual changes to the permissions on your function, you may need to reapply the scheduled event access to your function. You can do that by using the following CLI command.

```
aws lambda add-event-source --function-name my-function --event-source-arn arn:aws:events:us-east-1:123456789012:event-source/lambda/my-function --schedule-expression "cron(0-15 * * * ? *)" --role arn:aws:iam::123456789012:role/lambda-role
```

**Note**

Each AWS account can have up to 100 unique event sources of the **CloudWatch Events - Schedule** source type. Each of these can be the event source for up to five Lambda functions. That is, you can have up to 500 Lambda functions that can be executing on a schedule in your AWS account.

The console also provides a **Blueprint Lambda Library** that adds the **CloudWatch Events - Schedule** source type. Using this blueprint, you can create a sample Lambda function and test this feature. The example code that the blueprint provides checks for the presence of a specific webpage and specific text string on the webpage. If either the webpage or the text string is not found, the Lambda function throws an error.

For a tutorial that walks you through an example setup, see [Tutorial: Using AWS Lambda with Scheduled Events](#).

# Backup and disaster recovery

- Cross-region replication
- Off-site backups
- But! Validation of backups can be hard.
  - Set rules on Lambda to define what needs to be checked and backed up
- Alert on validation failure

AWS Compute Blog

## Synchronizing Amazon S3 Buckets Using AWS Step Functions

by Andy Hume | Jan 25, 2018 | [AWS Lambda](#), [AWS Step Functions](#), [Amazon S3](#) | [Comments](#)



Constantin Ganev is a Principal Solutions Architect at AWS.

In my free time, I run a small blog that uses Amazon S3 to host static content and Amazon CloudFront to distribute it world-wide. I use a home-grown, static website generator to create and upload my blog content onto S3.

My blog uses two S3 buckets: one for staging and testing, and one for production. As a website owner, I want to update the production bucket with all changes from the staging bucket in a reliable and efficient way, without having to create and populate a new bucket from scratch. Therefore, to synchronize files between these two buckets, I use [AWS Lambda](#) and [AWS Step Functions](#).

In this post, I show how you can use Step Functions to build a reliable synchronization engine for S3 buckets and learn some common patterns for designing Step Functions state machines while you do so.

### Step Functions overview

Step Functions makes it easy to coordinate the components of distributed applications and microservices using visual workflows. Building applications from individual components that each perform a discrete function lets you scale and change applications quickly.

While this particular example focuses on synchronizing objects between two S3 buckets, it can be generalized to any other use case that involves coordinated processing of any number of objects in S3 buckets, or other, similar data processing patterns.

### Bucket replication options

Before I dive into the details on how this particular example works, take a look at some alternatives for copying or replicating data between two Amazon S3 buckets:

- The [AWS CLI](#) provides customers with a powerful [aws s3 sync](#) command that can synchronize the contents of one bucket with another.
- [S3DataGP](#) is a powerful tool for users of [Amazon EMR](#) that can efficiently load, save, or copy large amounts of data between S3 buckets and HDFS.
- The [S3 cross-region replication](#) functionality enables automatic, asynchronous copying of objects across buckets in different AWS regions.

In this use case, you are looking for a slightly different bucket synchronization solution that:

- Works within the same region.
- Is more scalable than a CLI approach running on a single machine.
- Doesn't require managing any servers.
- Uses a more finely grained cost model than the hourly based Amazon EMR approach.

You need a scalable, serverless, and customizable bucket synchronization utility.

### Solution architecture

Your solution needs to do three things:

1. Copy all objects from a source bucket into a destination bucket, but leave out objects that are already present, for efficiency.
2. Delete all "orphaned" objects from the destination bucket that aren't present on the source bucket, because you don't want obsolete objects lying around.
3. Keep track of all objects for #1 and #2, regardless of how many objects there are.

In the beginning, you read in the source and destination buckets as parameters and perform basic parameter validation. Then, you operate two separate, independent loops, one for copying missing objects and one for deleting obsolete objects. Each loop is a sequence of Step Functions states that read in chunks of S3 object info and use the continuation token to decide if a choice state whether to continue the loop or not.



# Other resources

- Randall <3s Lambda!
  - @jrhunt on Twitter
  - Tons of examples and projects here: <https://github.com/ranman>
- AWS documentation:  
<http://docs.aws.amazon.com/lambda/latest/dg/welcome.html>
- Tons of compute blog posts:  
<https://aws.amazon.com/blogs/compute/category/aws-lambda/>





# Thanks!

Special thanks to Ian, who originally created the slides

