

SRV401

Become a Serverless Black Belt

Optimizing Your Serverless Applications

Ajay Nair, Principal Product Manager, AWS Serverless Applications
Peter Sbarski, VP, A Cloud Guru

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This is an advanced session for Serverless apps using lambdas and API Gateway in production

SERVERLESS CUSTOMERS



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Serverless is about maximizing **elasticity**,
cost savings, and **agility**.

There are a few core principles when building a serverless infrastructure

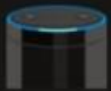
MULTIPLE POINTS TO OPTIMIZE



Amazon
API
Gateway



AWS
IoT



Amazon
Alexa

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



A typical serverless architecture has multiple layers, you have the Access and Authentication layers across your multiple sets of services responsible for deciding the ingress, access and authentication to your backend layers.

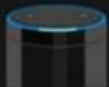
MULTIPLE POINTS TO OPTIMIZE



Amazon
API
Gateway



AWS
IoT



Amazon
Alexa



Amazon
Kinesis



Amazon
SES



AWS Step
Functions



Amazon
SNS

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



There often is also a messaging and orchestration layer responsible for coordinating calls to your backend with things like buffering, state management, collation of records, batching, etc.

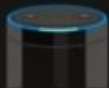
MULTIPLE POINTS TO OPTIMIZE



Amazon
API
Gateway



AWS
IoT



Amazon
Alexa



Amazon
Kinesis



Amazon
SES



AWS Step
Functions



Amazon
SNS



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



There is also a compute layer where all the business logic resides as lambda functions that gets executed against your backend

MULTIPLE POINTS TO OPTIMIZE



Amazon
API
Gateway



AWS
IoT



Amazon
Alexa



Amazon
Kinesis



Amazon
SES



AWS Step
Functions



Amazon
SNS



Amazon
S3



Amazon
DynamoDB



Amazon
Elasticsearch



Amazon
CloudWatch



EC2
instance



Custom
endpoints

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Finally, there is the data layer responsible for state, durable information that your applications need or third-party APIs that you access via available APIs.

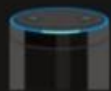
MULTIPLE POINTS TO OPTIMIZE



Amazon
API
Gateway



AWS
IoT



Amazon
Alexa



Amazon
Kinesis



Amazon
SES



AWS Step
Functions



Amazon
SNS



Amazon
S3



Amazon
DynamoDB



Amazon
Elasticsearch



Amazon
CloudWatch



EC2
instance



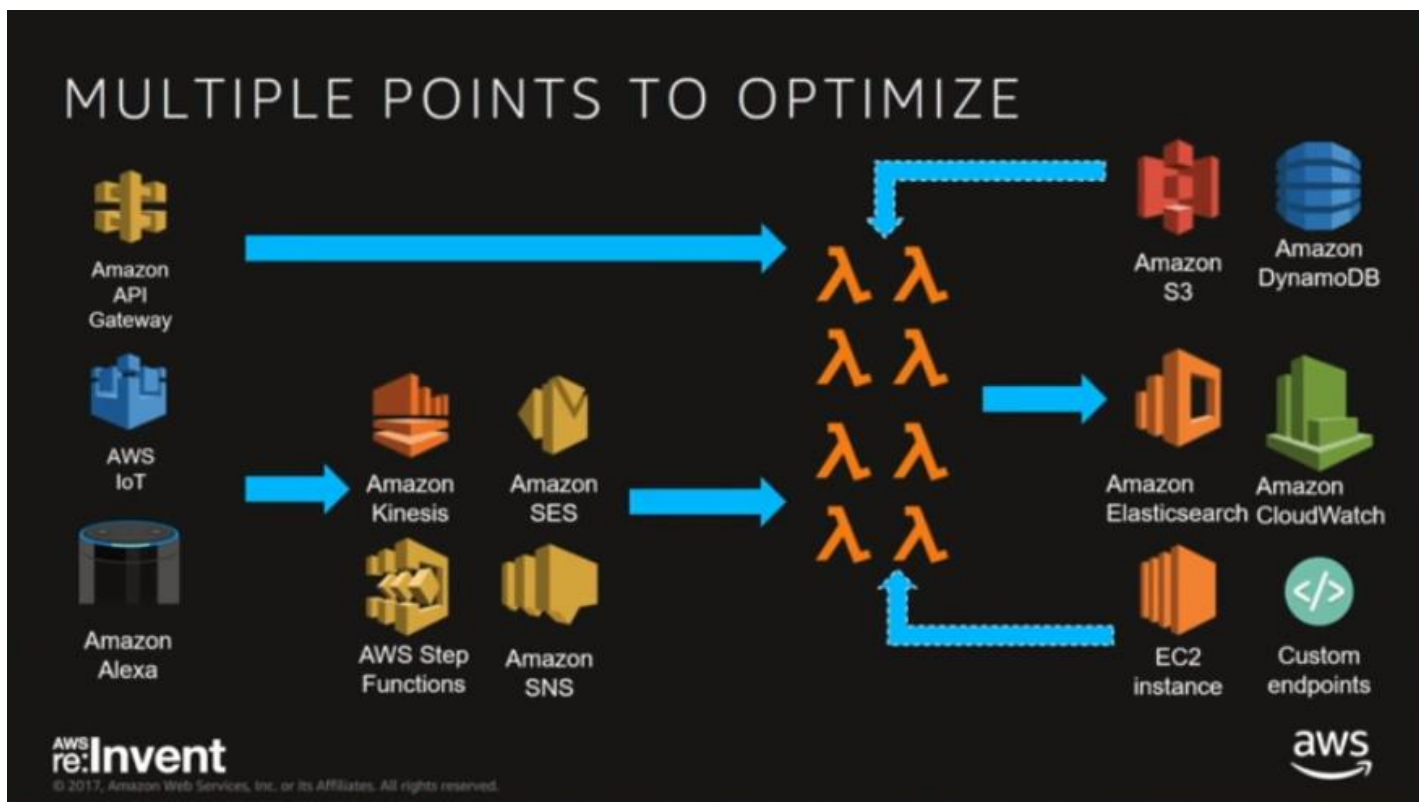
Custom
endpoints

AWS
re:Invent

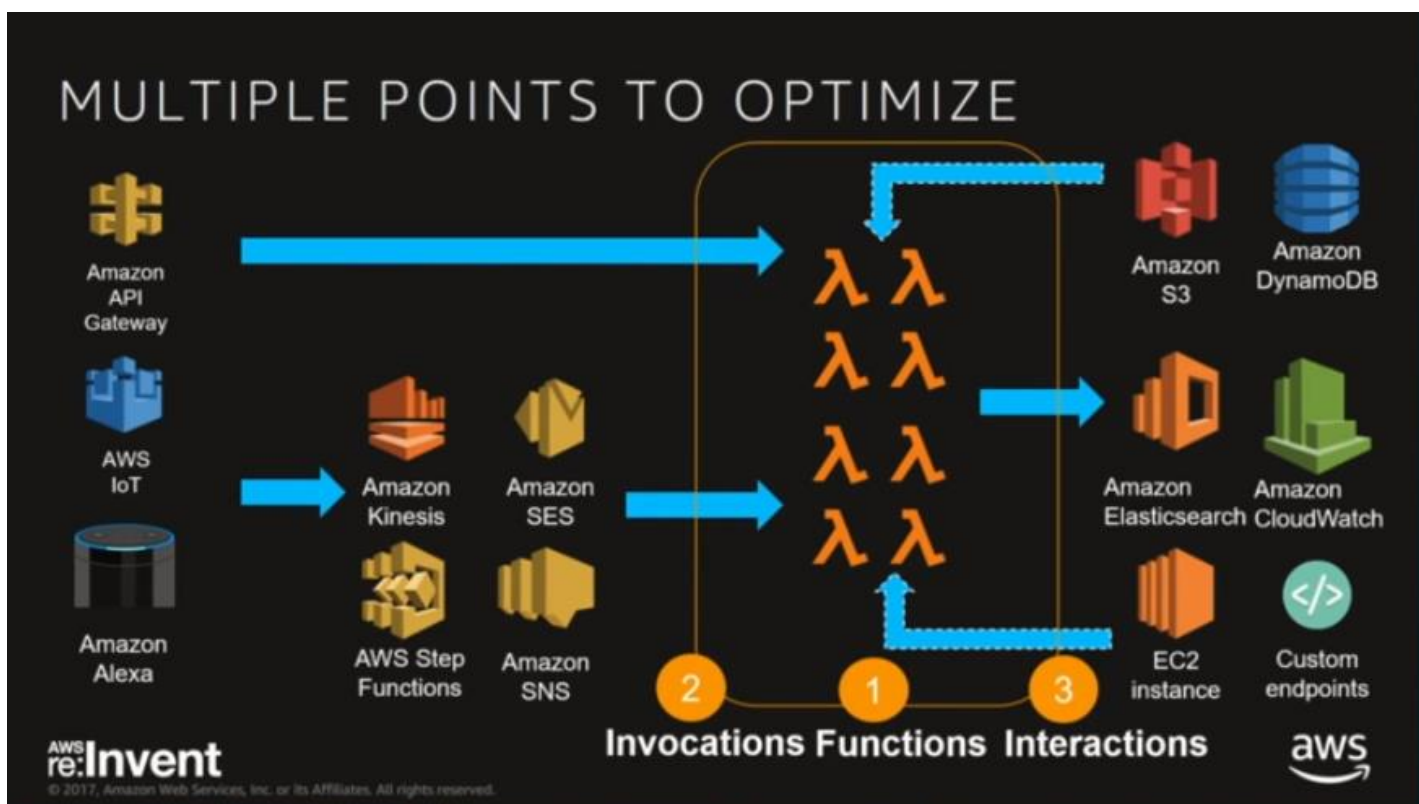
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



These same layers apply whether you are building end user apps which has certain flows



Or when building backend apps where only a couple of the layers are needed



We will be focusing on one specific area for optimization here, looking at your lambda functions, the invocations of these functions, and the interactions these lambda functions have with downstream services.

OPTIMIZATION KATAS

1. THE LEAN FUNCTION

2. EVENTFUL INVOCATIONS

3. COORDINATED CALLS

4. SERVICEFUL OPERATIONS

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We have distilled these practices into 4 main katas

GOAL TODAY

Repeatable regimen for building **highly resilient, high-performance** serverless applications.

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



KATA # 1

THE LEAN FUNCTION

Concise logic, efficient/single purpose code, ephemeral environment

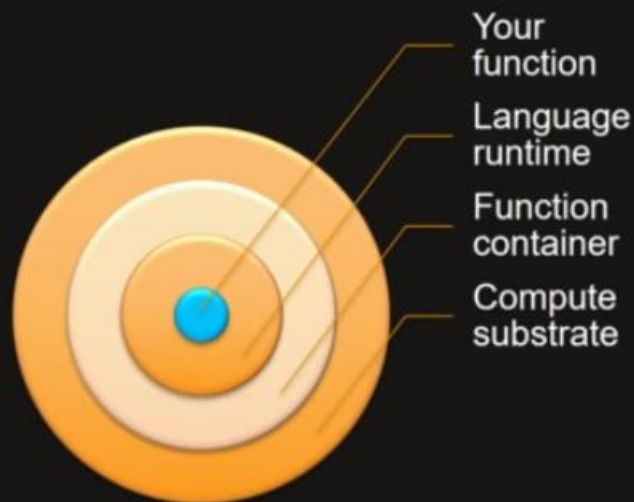
AWS
re:Invent

© 2017, Amazon Web Services, Inc., or its Affiliates. All rights reserved.



The lambda function itself is the most critical part of your architecture and needs to be optimized, it is the glue for all the other different components to talk to each other, controls your business logic, and it has significant implications on how the overall architecture behaves.

ANATOMY OF A FUNCTION



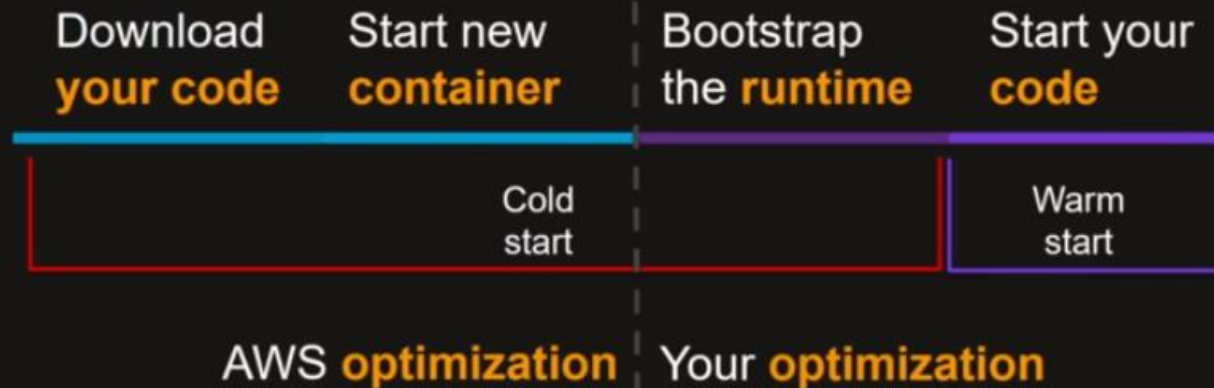
AWS
re:Invent

© 2017, Amazon Web Services, Inc., or its Affiliates. All rights reserved.



Your function is the code that is executing inside your lambda function's handler function.

THE REQUEST LIFECYCLE



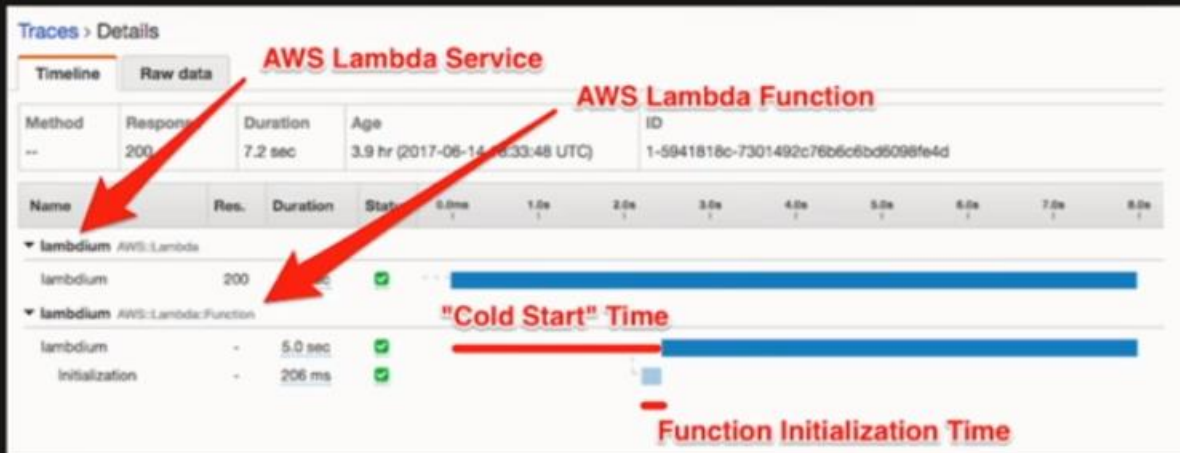
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



The application you are working on is going to be predominantly in the warm start section and will save you cost when optimized and reduce end to end latency for your calls.

SAME VIEW IN X-RAY



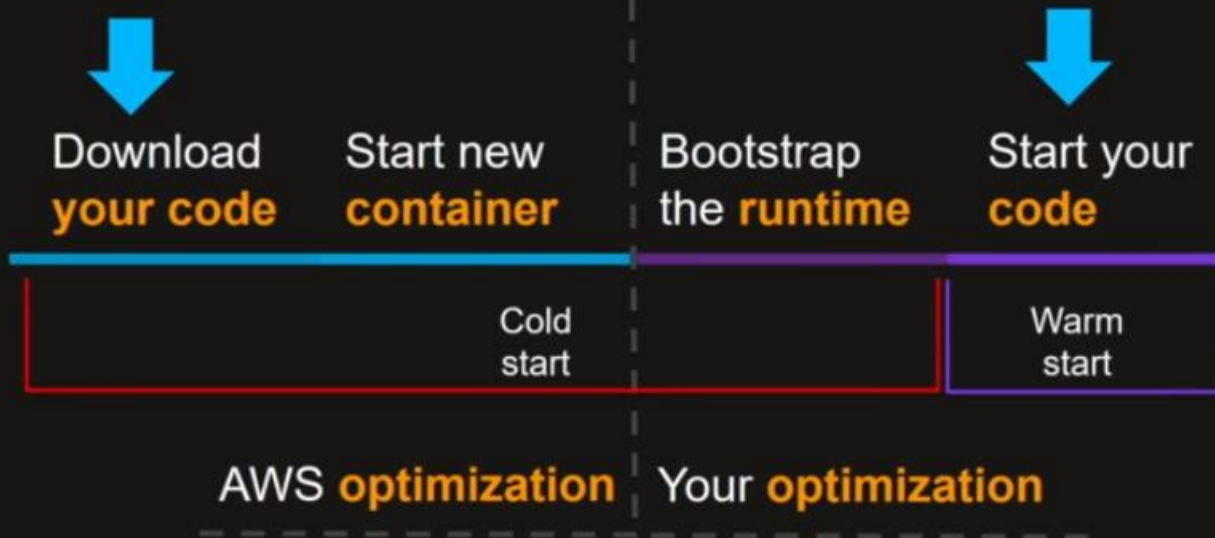
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You have visibility into what goes on at runtime

THE FUNCTION LIFECYCLE



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



EFFICIENT FUNCTION CODE

- Avoid **"fat"/monolithic** functions
- Control the **dependencies** in your function's deployment package
- Optimize for your **language**
 - Node – Browserfy, Minify

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Don't write a bunch of conditional and routing functions, it is always better to write them out as separate and orchestrated individual functions for better performance. You need to compress/shrink the archive that you upload into your lambda function.

JAVA – SCOPE YOUR POM FILE

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk-bom</artifactId>
      <version>2.10.10</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```

Maven Bill Of Materials (BOM) module for AWS SDK

```
<dependencies>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-s3</artifactId>
    <version>1.10.5</version>
  </dependency>
  <dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-dynamodb</artifactId>
    <version>1.10.10</version>
  </dependency>
</dependencies>
```

Select service dependencies only

Avoid **aws-java-sdk** directly!

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You need to pick the exact dependency that you are needing in your POM.xml file instead of the default of using the entire aws-java-sdk itself. Always strip out unneeded dependencies.

EPHEMERAL FUNCTION ENVIRONMENT

- Lambda processes a single event per-container
- No need for non-blocking execution on the frontend
- REMEMBER – containers are reused
 - Lazily load variables in the global scope
 - **Don't** load it if you don't need it – cold starts are affected

```
import boto3

client = None

def my_handler(event, context):
    global client
    if not client:
        client =
        boto3.client("s3")

# process
```

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



CONCISE FUNCTION LOGIC

- Separate Lambda handler (entry point) from core logic
- Use functions to **TRANSFORM**, not **TRANSPORT**
- Read only what you need
 - Query filters in Amazon Aurora
 - Use Amazon S3 select (**new!**)

If all your lambda function is doing is I/O then there is probably a better way to do it instead of waiting for data to come in, the more important thing is to make sure your lambda is doing **TRANSFORM** things instead of just **TRANSPORTing** data back and forth between components.

SMALL CHANGES, BIG DIFFERENCE

Before

200 seconds and 11.2 cents

```
# Download and process all keys
for key in src_keys:
    response =
s3_client.get_object(Bucket=src_bucket,
Key=key)
    contents = response['Body'].read()
    for line in contents.split('\n')[:-1]:
        line_count +=1
        try:
            data = line.split(',')
            srcIp = data[0][:8]
        ....
```

(<https://github.com/aws-labs/lambda-refarch-mapreduce>)

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This is part of the AWS Reference Architecture for running MapReduce using lambda, including code samples on GitHub. This particular example is processing about 770 million rows in 200 seconds for 11.2 cents. This specific example is looking at network log data and looking for specific IP addresses and doing an aggregation of IP ranges.

SMALL CHANGES, BIG DIFFERENCE

Before


200 seconds and 11.2 cents

```
# Download and process all keys
for key in src_keys:
    response =
s3_client.get_object(Bucket=src_bucket,
Key=key)
    contents = response['Body'].read()
    for line in contents.split('\n')[:-1]:
        line_count +=1
    try:
        data = line.split(',')
        srcIp = data[0][:8]
    ....
```

After

95 seconds and costs 2.8 cents

```
# Select IP Address and Keys
for key in src_keys:
    response =
s3_client.select_object_content
        (Bucket=src_bucket, Key=key,
expression =
        SELECT SUBSTR(obj._1, 1, 8),
obj._2 FROM s3object as obj)
    contents = response['Body'].read()
    for line in contents:
        line_count +=1
    try:
```

 (https://github.com/aws-labs/lambdarefarch-mapreduce)



We took the highlighted sections and replaced it with the new **S3 select function** which now only looks at the IP from the entire row and then pulls it out. This reduced about 50% of the cost and time to run.

SMART RESOURCE ALLOCATION

Match resource allocation (up to **3 GB!**) to logic

Stats for Lambda function that calculates **1000 times** all prime numbers **<= 1000000**

128 MB	11.722965sec	\$0.024628
256 MB	6.678945sec	\$0.028035
512 MB	3.194954sec	\$0.026830
1024 MB	1.465984sec	\$0.024638



© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You can trim your logic, your dependencies, your packages, but you also need to make sure that you are tuning your compute logic or compute resources available to match the code that is actually executing. Lambda now allows using up to 3GB of memory use

IMPACT OF MEMORY CHANGE



AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

<https://blog.newrelic.com/2017/06/20/lambda-functions-xray-traces-custom-serverless-metrics/>

aws

There are many tools for seeing the analysis of what dial settings you need to choose for your compute needs like **CloudWatch**, **X-Ray**, etc.

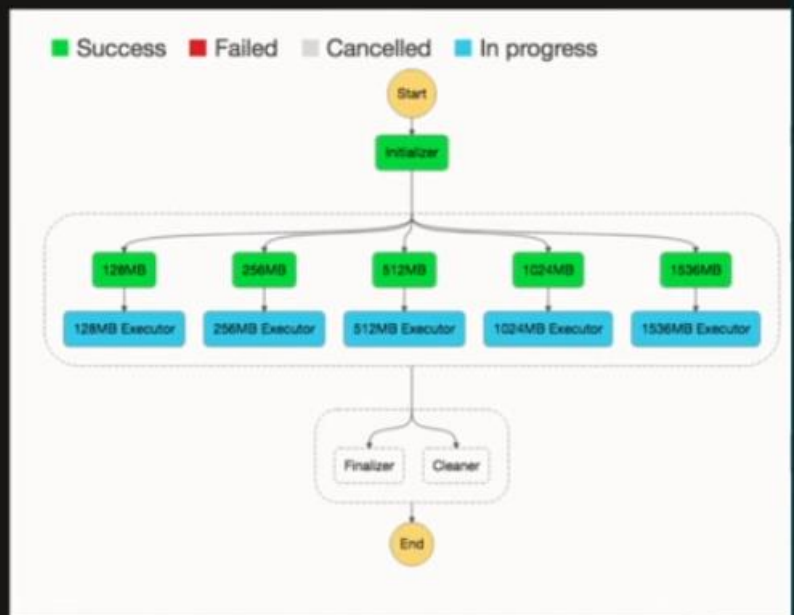
DON'T GUESSTIMATE!



alexcasalboni
aws-lambda-power-tuning

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This **Lambda Power Tuning tool** is available for free on **GitHub** for you to do compute needs analysis, it uses step functions to automatically switch the memory use of your function, run it in parallel for multiple runs, aggregates the output data from **CloudWatch** of the execution time and memory settings, and tells you what the appropriate setting should be for that function. This is an important optimization step that you ALWAYS need to do, make this part of your automation and testing regime as you may have changes based on payload, architecture, infrastructure, etc.

MULTITHREADING? MAYBE

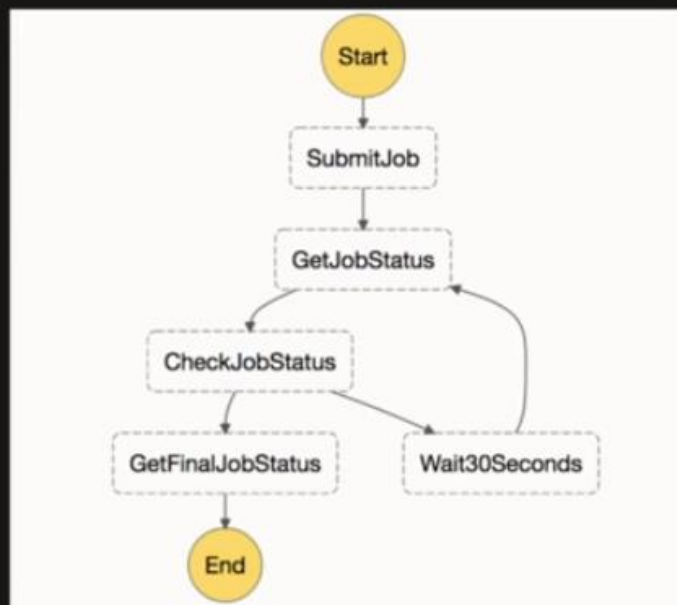
- <1.8GB is still single core
 - CPU bound workloads won't see gains – processes share same resources
- >1.8GB is multi-core
 - CPU bound workloads will see gains, but need to multi thread
- I/O bound workloads WILL likely see gains
 - e.g. parallel calculations to return

If you have a CPU function, you might be better not worrying about multithreading your lambdas since the function will get access to all the resources allocated to it within the 1 core container. But if you are doing I/O bound workloads, you can do optimizations by running the I/O on a separate thread doing a download and calculation separately, and doing your core computation on another thread. But for applications with lambdas using more than 1.8GB memory, we do have multi-thread functionality available. This is for cases like ML models that lend themselves well to multithreading, you can use lambdas for this to see some performance optimizations.

NO ORCHESTRATION IN CODE



NO ORCHESTRATION IN CODE



AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

You can just split it up into 2 lambdas, one for preprocessing and then another for the post-processing functions. You can then use a step function or some external mechanism to sequence between the 2 lambdas.

KATA # 2

EVENTFUL INVOCATIONS

Succinct payloads, resilient routing, concurrent execution

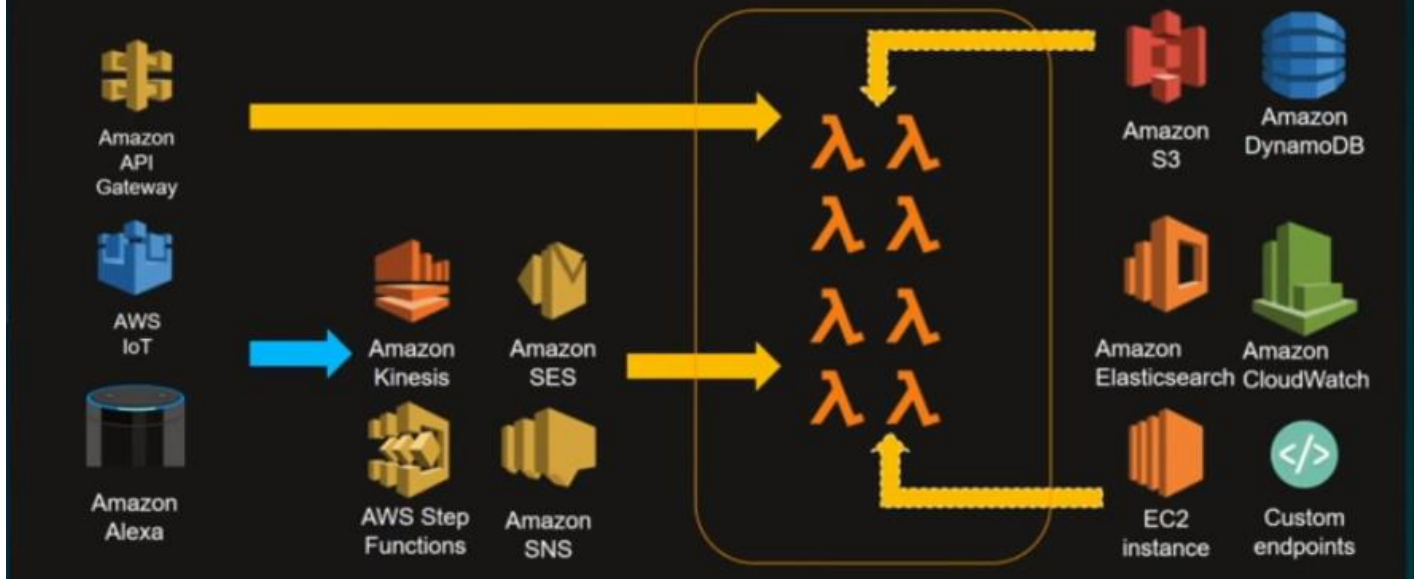
AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

This talks about the actual invocation layer for stuff coming in, how we choose the invocation mechanism makes a big difference.

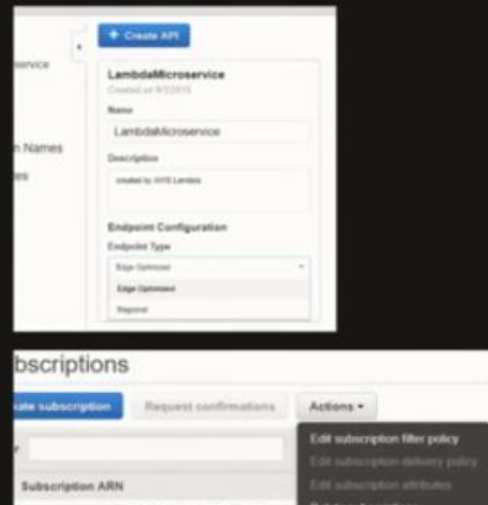
INVOCATION PATHS



When you are using AWS services to invoke the services, AWS controls the invocation paths and the optimizations in play like building in retries, compacting the payload, efficient mechanisms between the wire.

GATEWAYS AND ROUTERS

- Choose suitable entry point for client applications
 - Single, custom client? Use the **AWS SDK**
 - Not end user facing? use **regional endpoints on API Gateway**
- Discard uninteresting events ASAP
 - S3 – Event prefix
 - SNS – Message filtering (**new!**)



For use cases where you control the payload like generating data from an IoT device and funneling it in through kinesis, generating payloads that are coming in via the API Gateway. In this case, you need to make choices about how you are going to do the routing and staging invocations to your lambda functions. **Lambda exposes a direct invoke API that is an HTTP endpoint, you can send a signed request to it and get a response back immediately.** Most services taking data from SNS queues offer a way for you to discard unneeded events data before they get to your lambdas, this will save unneeded processing times and cost. S3 as a way for you to do prefix and suffix filters, this allows you a way to lock down lambda functions to trigger only on updates to certain paths inside your S3 buckets. SNS recently launched a capability for you to do message property based filtering, for example in your SNS payload if you have a property that said 'this is coming from originator app A', you can say only process the payload if it is from originator app B.

SUCCINT INVOCATIONS

- Scrutinize the event
 - Must have provenance i.e. "What happened for this notification to occur?"
 - Additional content – identifier or payload
- Remember payload constraints
 - Async invocation is only 128K
 - Avoid large responses like an image

If you use a custom event and you control it, there is a bare minimum that should be in the event itself – this is what is called **Provenance**. You should be able to say where the event came from, who the identifier is, what timestamp was involved during the event. Everything else is optional, you should avoid too many payloads coming into your function. If you are an image processing system, put the image into an S3 bucket first, then you can have your lambda go and read the image from that bucket if needed.

EXAMPLE - SWITCH TO BINARY

```
'use strict';
const co = require('co');
const Promise = require('bluebird');
const protobuf = Promise.promisifyAll(require("protobufjs"));
const lib = require('./lib');
const fs = require('fs');

module.exports.handler = co.wrap(function* (event, context, callback) {
  console.log(JSON.stringify(event));

  let players = lib.genPlayers();
  let root = yield protobuf.loadAsync("functions/player.proto");
  let Players = root.lookupType("protodemo.Players");
  let message = Players.create(players);
  let buffer = Players.encode(message).finish();

  const response = {
    statusCode: 200,
    headers: { 'Content-Type': 'application/x-protobuf' },
    body: buffer.toString('base64'),
    isBase64Encoded: true
  };
};
```

The same response in Protocol Buffers is nearly 40% smaller compared to default JSON

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

<http://theburningmonk.com/2017/09/using-protocol-buffers-with-api-gateway-and-aws-lambda/>



For API Gateway, you can control the protocol that gets passed on into lambda for tuning the responses back, JSON is great but there are other options available. Above is an example of swapping out the JSON responses coming into lambda for processing with Protocol Buffers. This uses the API Gateway capability of supporting binary requests and responses, you can set the response type to binary and change the header to protoBuf and just pass any service that can emit protocol buffers out there to consume the protobuf response.

RESILIENT: USE AN EVENT STORE



VS.



There is a construct of the Event Store which should or not play a part in your architecture depending on your choice. For many applications, we look at lambda comes in or a request comes in through a router like IoT or API Gateway, turns around and invokes your lambda function. But you may have cases where you need to say 'these events need to be stable even if the one emitting it goes down' or 'this needs to be persisted in some durable fashion if lambda goes down' or 'it needs to be retry if lambda fails'. Kinesis tends to be the favorite for many data processing applications where it is used as a bridge between your ingress point and your lambda, you use Kinesis as the streaming mechanism for turning around and processing those applications. SQS is another viable choice to replace Kinesis depending on your use case of queues or streaming based event sources.

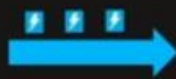
DynamoDB streams is an event store for DynamoDB updates, it gives you a persistent mechanism for seeing what the change log looks like. Lambdas Async API has a queue built in that gives you an event store end to end so that you can do retry policies under the covers.

THINK CONCURRENT, NOT TPS

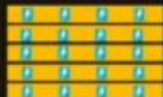
Simple
No event store



Queue based



Stream based



Lambda thinks of its scale in terms of concurrency, it is the unit of the request rate and depends on how you are calling your lambdas. It is a little different if it is stream based for ordering guarantees

CONCURRENCY vs LATENCY

Streams

- Maximum theoretical throughput:
 $\# \text{ shards} * 2 \text{ MB} / (\text{s})$
- Effective theoretical throughput:
 $(\# \text{ shards} * \text{batch size (MB)}) / (\text{function duration (s)} * \text{retries until expiry})$
- If put / ingestion rate is greater than the theoretical throughput, consider increasing number of shards while optimizing function duration to increase throughput

Everything else

- Maximum Processing rate :
 $\text{Maximum concurrency} / \text{average duration (events per second)}$
- Effective Processing rate :
 $\text{Effective concurrency} / \text{average duration (events per second)}$
- Use concurrency metric (**new!**) and duration metric to estimate processing time

Make sure that you are looking at the new concurrency metric called **Concurrency invocations** in your console that tells you what is the effective concurrency available for your account level as well as your individual functions.

RESILENT: RETRY POLICIES

- Understand retry policies
 - Sync never retried
 - Async retried 2 times
 - Streams retried all the time
- Leverage Dead Letter Queues
 - SQS or SNS for replays
- **REMEMBER:** Retries count as invokes

BUILD YOUR OWN



<http://theburningmonk.com/2017/04/aws-lambda-3-pro-tips-for-working-with-kinesis-streams/>

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



SLS has its own Dead Letter DL queue built in that is the same as the Lambda DL queue, you can route the DL queue of failed attempts from Kinesis to an SNS queue instead of sending an error to Kinesis that will trigger a retry.

KATA #3

COORDINATED CALLS

Decoupled via APIs, **scale-matched** downstream, **secured**

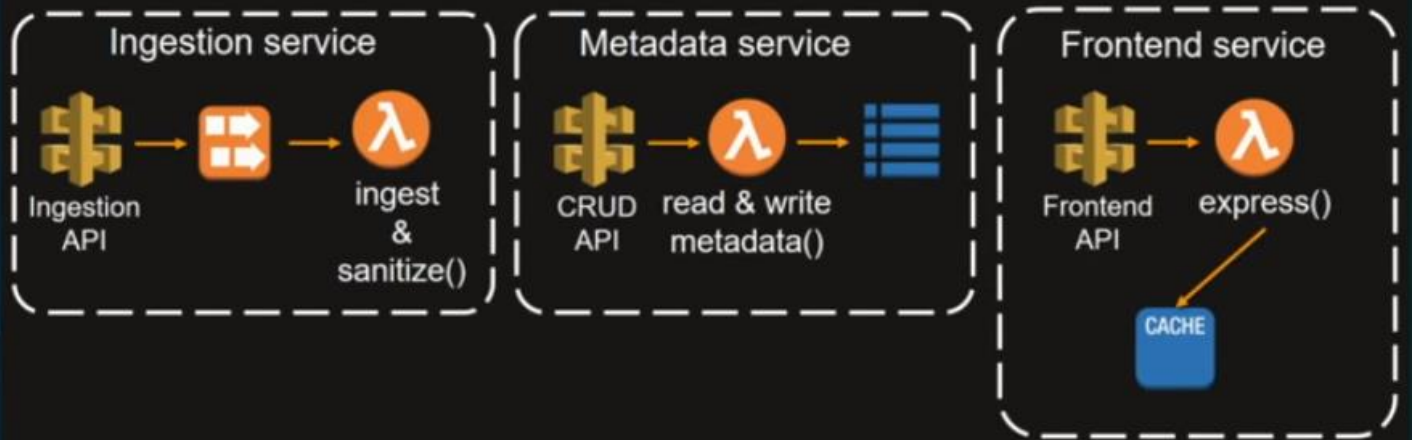
AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This is about how lambdas talk to downstream services.

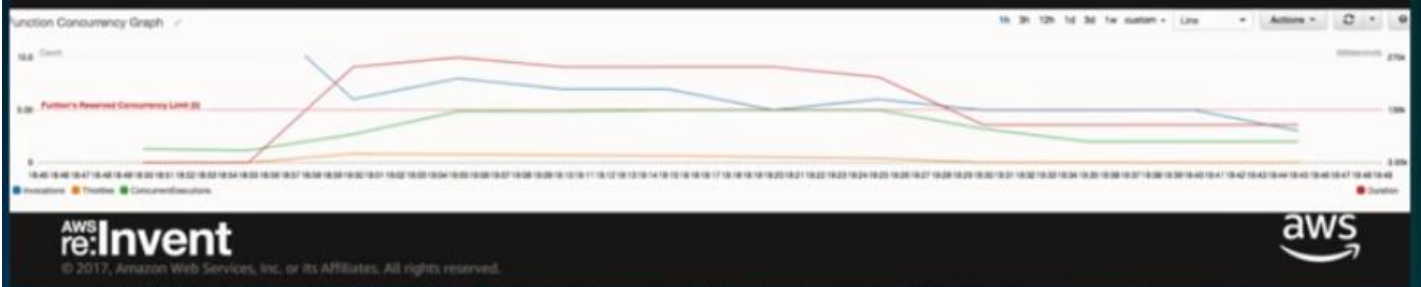
DECOUPLED: APIs AS CONTRACTS



Make sure the way you set up your applications has APIs as Contracts, this is important in the lambda implementation because having APIs between components of your system allows you to have separation of concerns. In this example, we have 3 separate services, an ingestion service that gets processed by a metadata extraction service that is frontend by a frontend service with 3 separate APIs for Ingestion, CRID, and Frontend. **A DDOS attack on the Ingestion API can be rate limited by the CRUD API to prevent both the Metadata and Frontend services.**

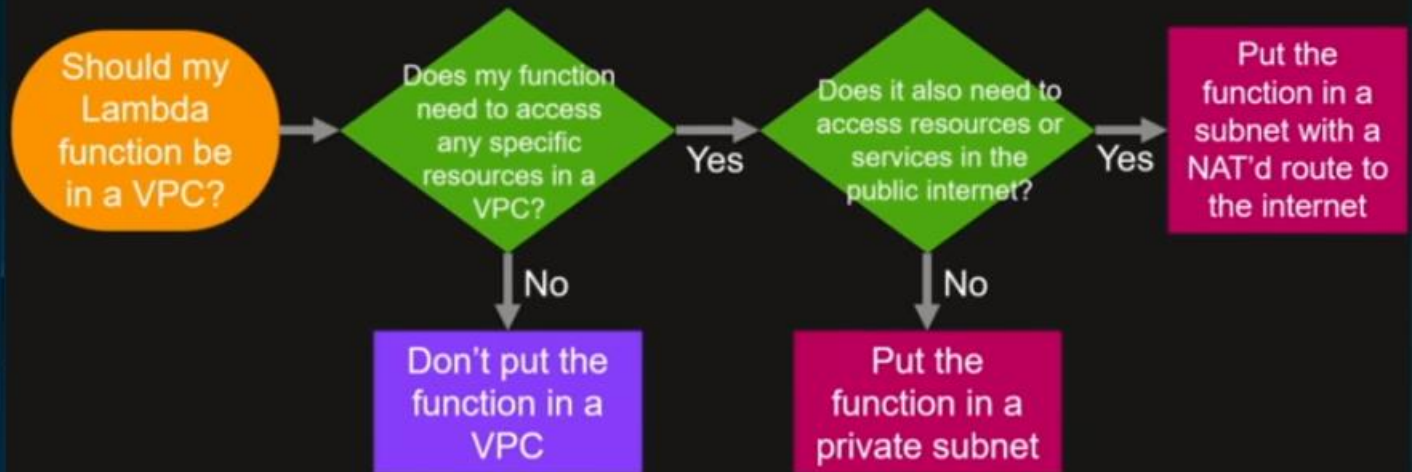
SCALE-MATCHED: CONCURRENCY CONTROLS

- Concurrency a shared pool by default
- Separate using per function concurrency settings
 - Acts as reservation
- Also acts as max concurrency per function
 - Especially critical for data sources like RDS
- "Kill switch" – set per function concurrency to zero

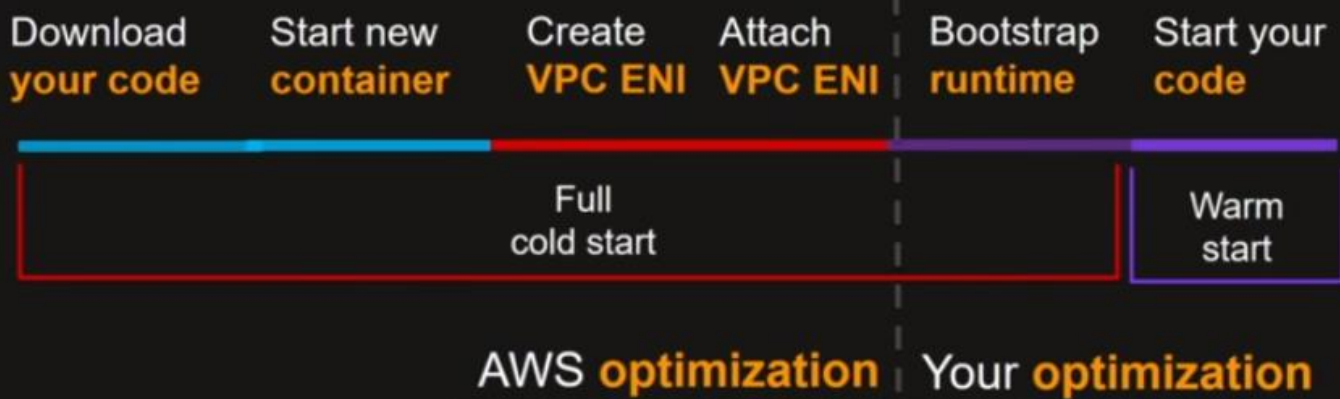


For non-API calls to databases and other service, you need to have Scale Matching for resources that have scale constraints like databases. Make sure that lambdas are calling out to these databases, you are matching the scale limitation for lambda to the end data store. You can now specify the max concurrency for individual lambda functions, this allows you to restrict an individual function from fanning out beyond a specific limit.

SECURED: DO I NEED A VPC?



SECURED: VPC vs LATENCY



SECURED: VPC vs RESILIENCE

- ALWAYS configure a minimum of 2 Availability Zones
- Give your Lambda functions their own subnets
- Give your Lambda subnets a large IP range to handle potential scale
- If your functions need to talk to a resource on the internet, you need a NAT!

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



KATA #4

SERVICEFUL OPERATIONS

Automated operations, Monitored applications, Innovation mindset

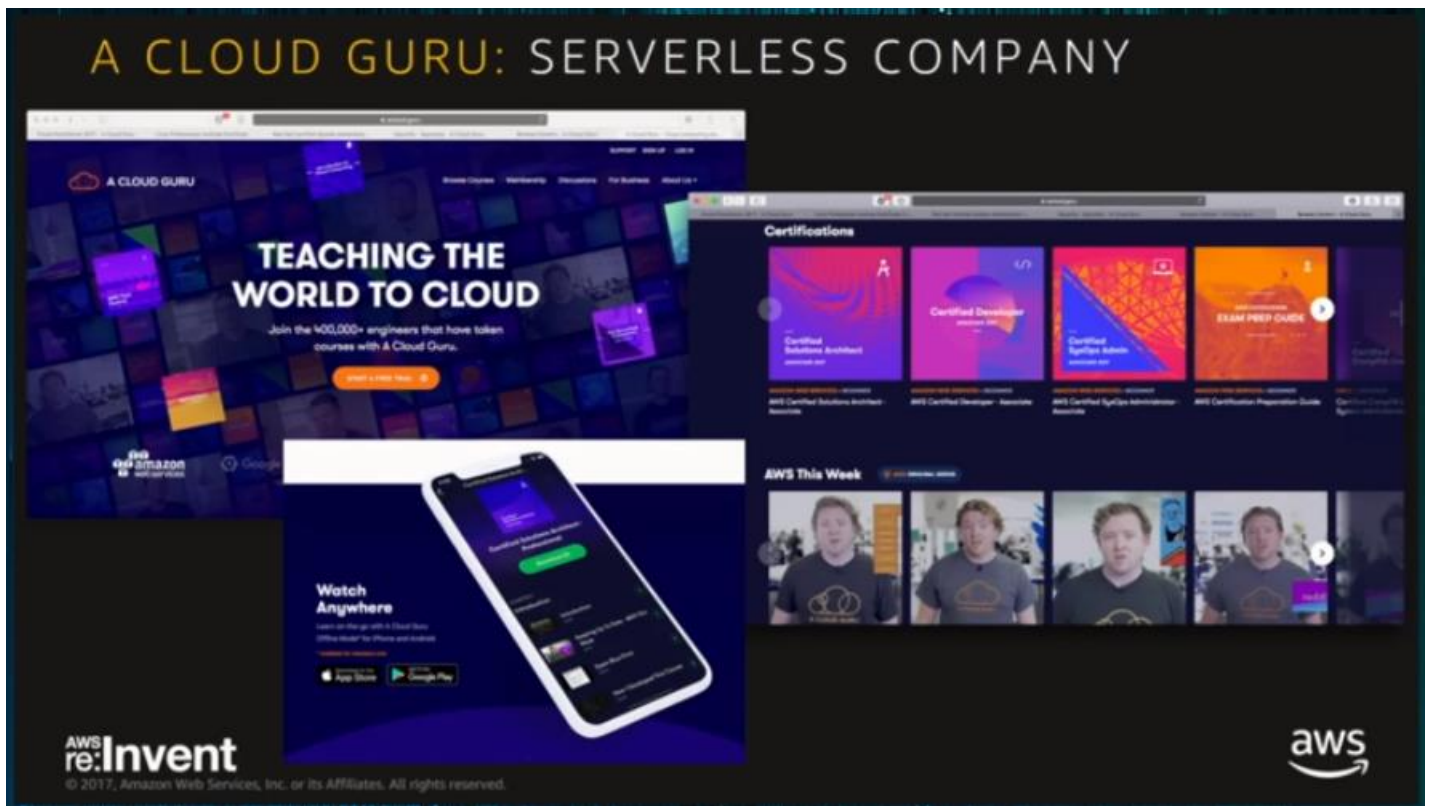
In the end, it's about the people

AWS
re:Invent

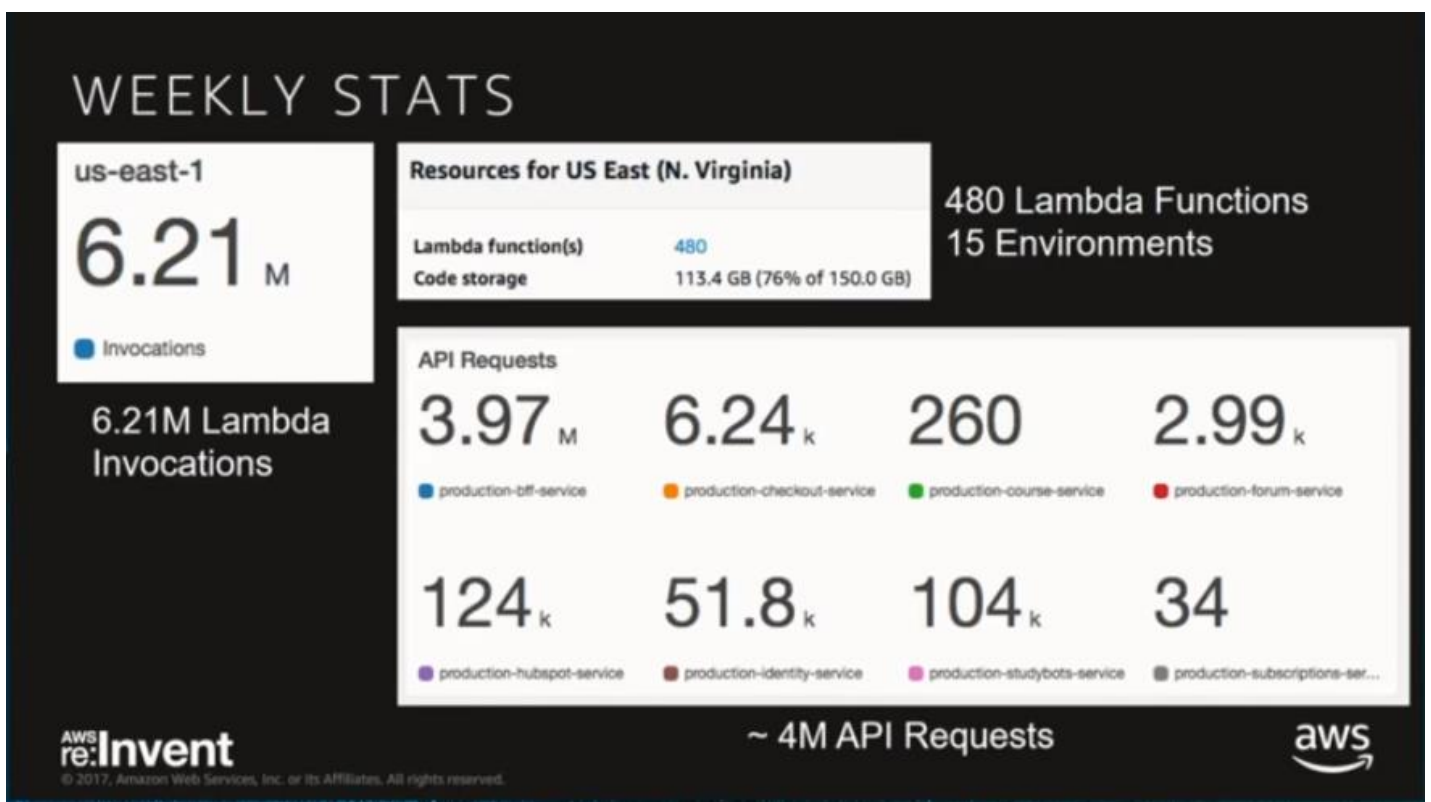
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Let us see how to put a lot of these recommendations into practice at A Cloud Guru



Started 2 years ago and built an entirely Serverless company using lambdas. We also have rich web apps and mobile apps without any servers



There are **3.97 million API Requests** going through the **production-bff-service** service which is the GraphQL endpoint. There is a new **AWS AppSync GraphQL managed service** that AWS just released

WEEKLY STATS

142 Buckets

16

Public

3 Regions



2+ TB of data in S3



7+ TB of data served weekly via CloudFront

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This means we need to optimize the functions in our applications to reduce the overall cost

AUTOMATE

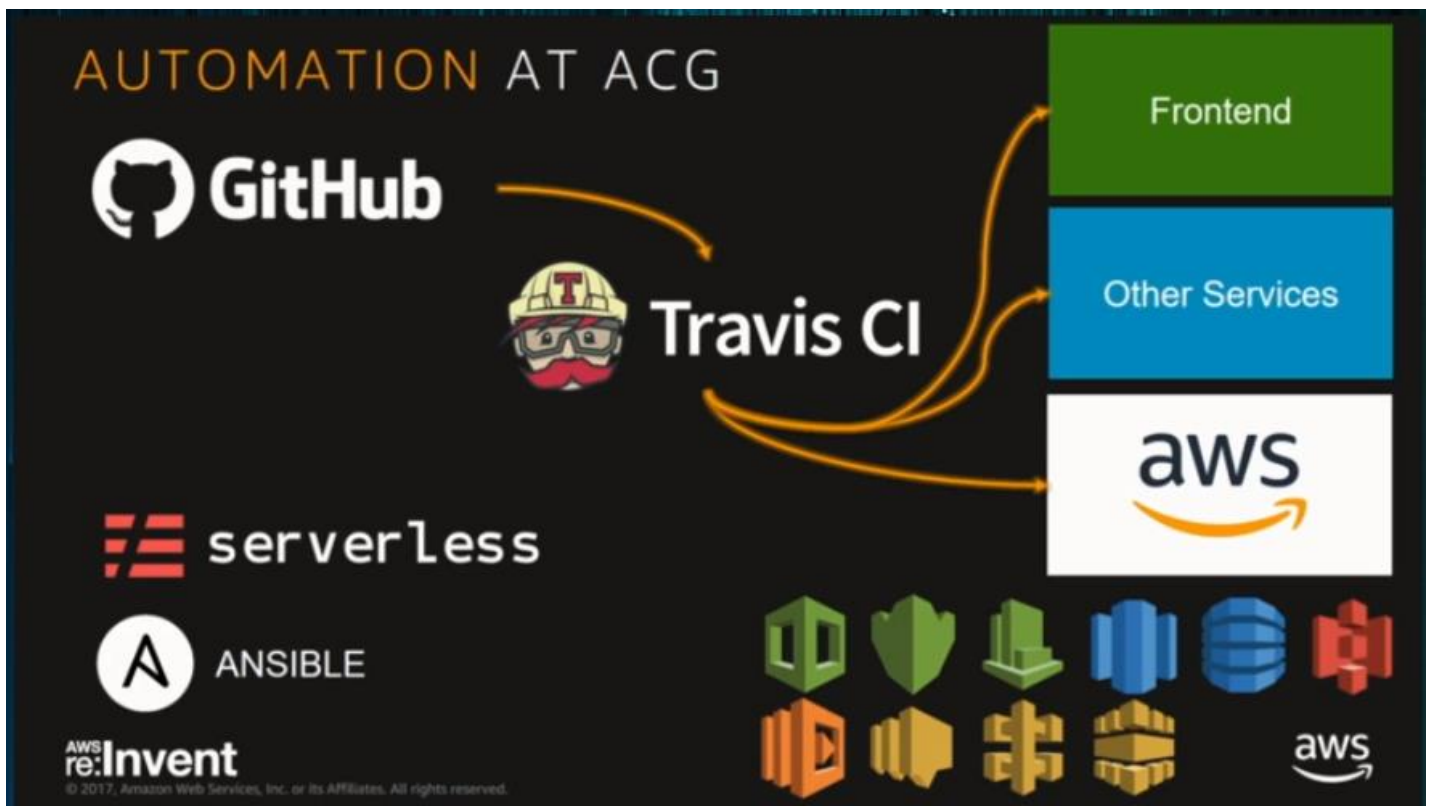
MONITOR

INNOVATE

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.





Serverless applications are distributed by nature, so we have several lambda functions, our SPA, various services to updates, we need to make changes to Firebase which is our real-time streaming database. We do this all via automation and testing. We also use the **Serverless Framework** for **orchestration** and **deployment**, and use Ansible for our Dev environments so that we can deploy some additional CloudFormation scripts. **Don't go serverless without automating the provision and deployments of your applications first, you need to setup your deployment pipeline before starting to work on your applications.**



For testing, we use Selenium, Jester and Travis CI automates the testing process for us.

AUTOMATE

MONITOR

INNOVATE

AWS
re:Invent

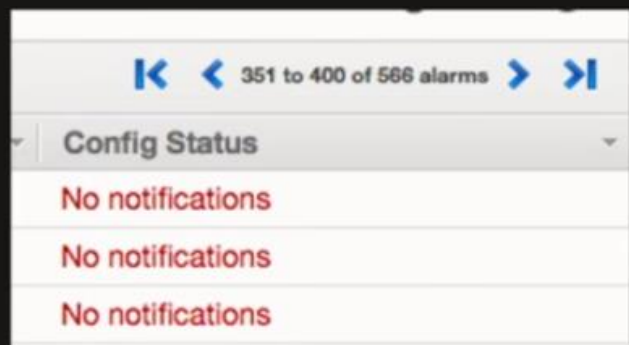
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



MONITORING - ALERTS



Dashboards covering services we use



566 CloudWatch alarms

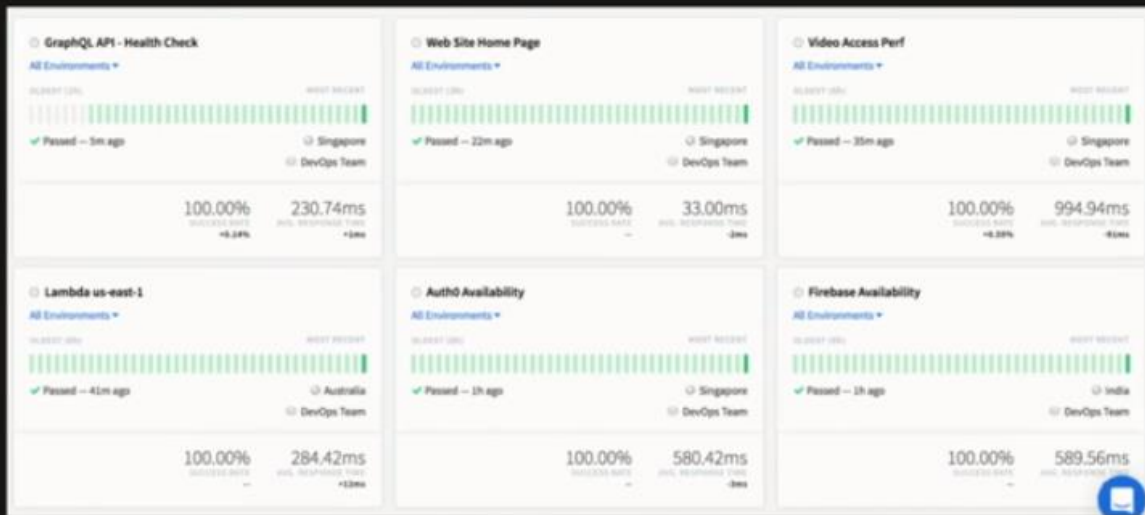
AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



You need to be aware of what is going on in your system, use alerts while also calibrating the signal to noise ratio. You probably should have an alarm for every lambda function you create

MONITORING - DASHBOARDS



Runscope Monitoring

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



The **Runscope tool** helps us monitor the responsiveness and latency within our system. It continuously makes requests to **CloudFront** to see whether we can access our files for all our applications like the GraphQL API, Website, Auth) service, Firebase service, etc.

MONITORING - SECURITY

A1:2017 - Injection	7
A2:2017 - Broken Authentication	8
A3:2017 - Sensitive Data Exposure	9
A4:2017 - XML External Entities (XXE)	10
A5:2017 - Broken Access Control	11
A6:2017 - Security Misconfiguration	12
A7:2017 - Cross-Site Scripting (XSS)	13
A8:2017 - Insecure Deserialization	14
A9:2017 - Using Components with Known Vulnerabilities	15
A10:2017 - Insufficient Logging & Monitoring	16

AWS re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Security in serverless is a joint responsibility between you and AWS. Make sure that your IAM roles, users, and policies are as secured as possible. Have a Role per lambda function. Use things like AWS KMS where possible.

AUTOMATE

MIGRATE

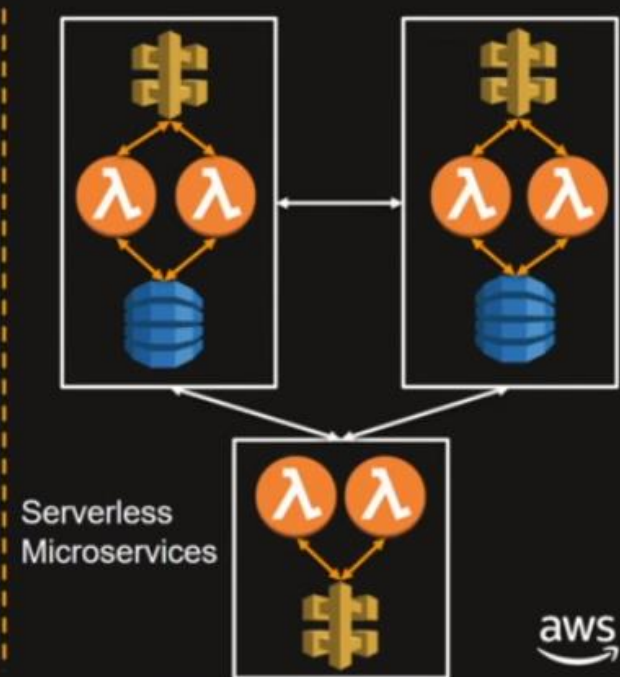
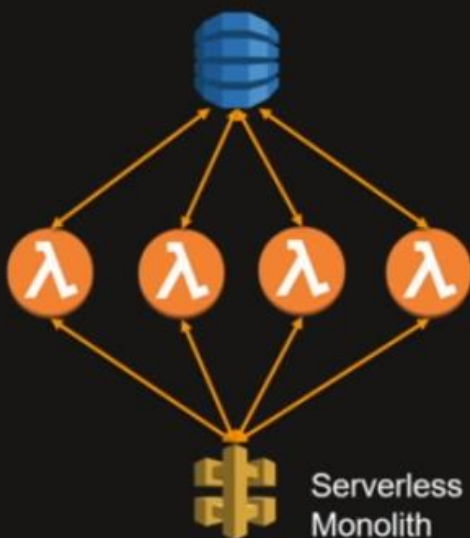
INNOVATE

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



INCREMENTAL ARCHITECTURE



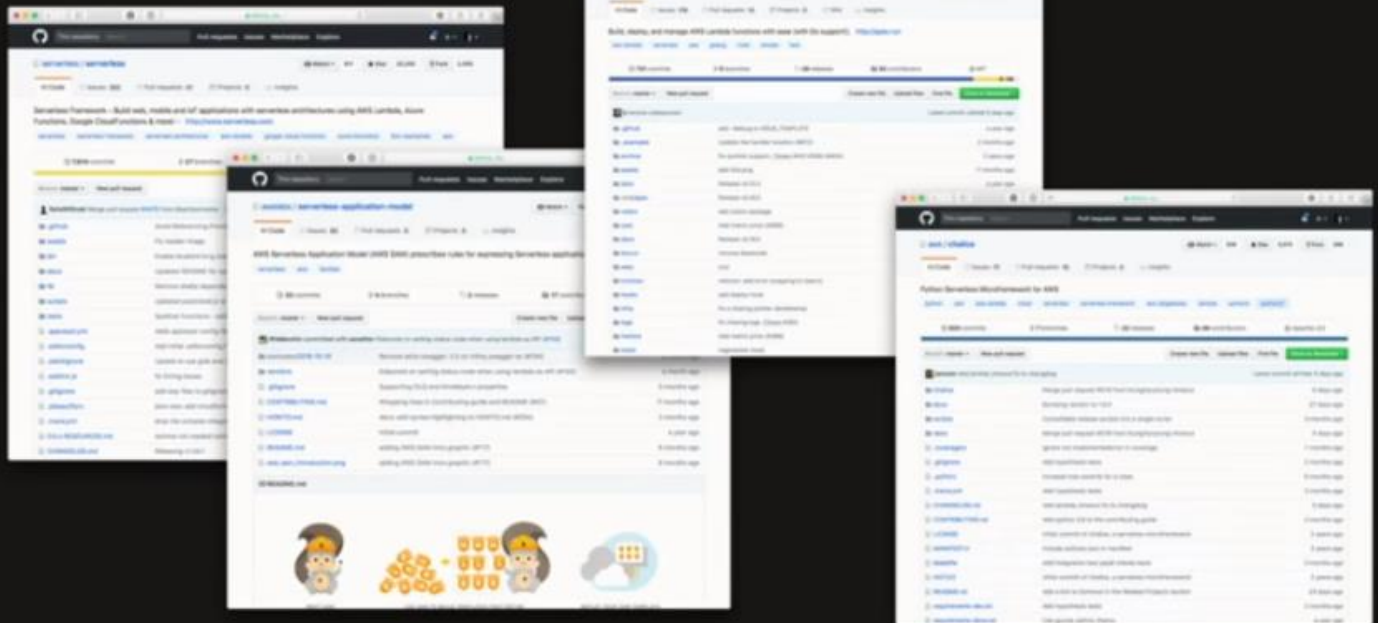
AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Apply the tips and tips about your architecture incrementally, you can then scale out and transition from a serverless monolith to a **serverless microservices** approach just by changing some of the code and changing the **serverless.yml** file without provisioning any server at all.

INNOVATION



aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



As you evolve, best practices will change

INNOVATION

archive	Added backend with lambda
config	Fix to issue where videos and text lessons weren't showing again
grunt	Fixing issues with text sources
node_modules	Added backend with lambda
src	Fixing issues with text sources
utils	Added backend with lambda
Gruntfile.js	lots of build changes to support a second set of production lambdas
Initial Setup Notes.txt	Added backend with lambda

Executable File | 10 lines (9 sloc) | 436 Bytes

```
1 #!/bin/bash
2 grunt create:answer-vote --target=$1 --build=$2
3 grunt create:charge-stripe --target=$1 --build=$2
4 grunt create:get-paypal-token --target=$1 --build=$2
5 grunt create:question-vote --target=$1 --build=$2
6 grunt create:request-restricted-file --target=$1 --build=$2
7 grunt create:submit-answer --target=$1 --build=$2
8 grunt create:submit-question --target=$1 --build=$2
9 grunt create:verify-external-purchase --target=$1 --build=$2
```

```
module.exports = function(grunt) {
  var gtx = require('gruntfile-gtx').wrap(grunt);

  gtx.loadAuto();

  var gruntConfig = require('./grunt');
  gruntConfig.package = require('./package.json');

  gtx.config(gruntConfig);

  var target = grunt.option('target') || 'staging';
  var buildSuffix = grunt.option('build') || null;
  console.log("Build suffix: " + buildSuffix);

  gtx.config({
    globalConfig: {
      target: target,
      awsCLIProfile: 'acloudgurulive'
    }
  });

  console.log("Running for environment " + target);

  // We need our base components in order to develop
  gtx.alias('build:function', ['clean:function', 'copy:function', 'compress:function']);
  gtx.alias('create:function', ['build:function', 'exec:create-lambda-function']);
  gtx.alias('deploy:function', ['build:function', 'exec:upload-lambda-function']);

  function getLambdaFunctionName(baseName) {
    return baseName + '-' + target + (buildSuffix ? '-' + buildSuffix : '');
  }
}
```

aws
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



We used to roll out our own deployment scripts using Grunt in the past but this is not needed anymore once we migrated to the Serverless Framework

A FEW PUBLIC PROJECTS FROM ACG

serverless-docker

This is a proof of concept to see if we can replicate Amazon API Gateway using Docker images to run Lambda.

Features:

- A runtime **supported** by docker lambda.
- CORS
- AuthHeader
- Custom Authorizer **supported**
- Cognito Authorizer **not implemented yet**
- Lambda Integration
- Velocity template support **supported**
- Lambda Proxy Integration **supported**

serverless-plugin-cloudwatch-sumologic

Plugin which auto-subscribes a log delivery lambda function to lambda log groups created by Serverless.

Installation

```
npm install --save-dev serverless-plugin-cloudwatch-sumologic
```

Configuration

1. First follow [this guide](#) to create a new collector and http source on Sumologic.
2. Add the following custom variables to your serverless.yml file.

ACloudGuru / **serverless-plugin-aws-alerts**

Code Issues (6) Pull requests (2) Projects

A Serverless Framework plugin that creates CloudWatch alarms

serverless serverless-framework aws aws-cloudwatch

45 commits 1 branch

Branch: master New pull request

ACloudGuru / **serverless-plugin-crypt**

Code Issues (1) Pull requests (0)

No description, website, or topics provided.

Add topics

10 commits 1 branch

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



Serverless is a **mindset** change toward automation, agility, and innovation.

OPTIMIZATION KATAS

1. THE LEAN FUNCTION

CONCISE. EFFICIENT. EPHEMERAL.

2. EVENTFUL INVOCATIONS

SUCCINT. RESILIENT. CONCURRENT.

3. COORDINATED CALLS

DECOUPLED. SCALE MATCHED. SECURED.

4. SERVICEFUL OPERATIONS

AUTOMATE. MONITOR. INNOVATE.

AWS
re:Invent

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



MORE OPTIMIZATION SESSIONS

SRV303 - Monitoring and Troubleshooting in a Serverless World

SRV322-R2 - Migration to Serverless: Design Patterns and Best Practices

SRV311 - Authoring and Deploying Serverless Applications with AWS SAM

SRV320-R - Best Practices for Using AWS Lambda with RDS/RDBMS Solutions

**AWS
re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



**AWS
re:Invent**

THANK YOU!

**AWS
re:Invent**

© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



CTD309
AWS re:INVENT
Building Serverless Websites with Lambda@Edge
George Joss
Product Manager
Amazon CloudFront/Lambda@Edge
Siddharth Kothandaraman
Senior Software Development Engineer
Amazon CloudFront/Lambda@Edge
December 1, 2017

AWS re:INVENT
What's New in AWS Serverless
Dr. Tim Wagner, General Manager Amazon API Gateway and AWS Lambda
November 30, 2017

AWS re:INVENT
Running Oracle Databases on Amazon RDS
Michael Barras, Sr. Database Engineer, Amazon RDS
Siddharth Ram, Chief Architect—Small Business Division, Intuit
November 30, 2017

SRV302
Building CI/CD Pipelines for Serverless Applications
Chris Munns - Serverless Senior Developer Advocate - AWS
Ben Kehoe - Cloud Robotics Research Scientist - iRobot
December 1, 2017

AWS re:Invent
CON215: Intro to Amazon Elastic Container Service for Kubernetes
Brandon Chavis - Sr. Product Manager

GPSTEC314
AWS re:INVENT
GPS: From Monolithic to Serverless—Why and How to Move
Ian Scafield | Partner Solutions Architect
Paras Bhavs | Partner Solutions Architect
November 28, 2017

SID322
AWS re:Invent
The AWS Philosophy of Security
Eric Brandwine, AWS VP and Distinguished Engineer
November 30, 2017

CON214
INTRODUCTION TO AWS FARGATE
Anthony Suarez - GM, Amazon ECS & ECR
Deepak Dayama - Senior Product Manager, Amazon ECS
November 28, 2017

MCL 365
AWS re:INVENT
INTRODUCTION TO AMAZON SAGEMAKER
KUMAR VENKATESWAR, SR. PRODUCT MANAGER, AWS
MONICA HSU, GROUP MANAGER, DATA SCIENCE, INTUIT
November 29, 2017

NET404
AWS re:INVENT
Connecting Many VPCs
Shared and Transit Architectures
Nick Matthews, Partner Solutions Architect
Vicente De Luca, Staff Software Engineer, Zendesk

DAT403-R
AWS re:INVENT
Advanced Design Patterns for Amazon DynamoDB
Rick Houlahan - Senior Practice Manager, NoSQL
AWS Professional Services

AWS re:Invent
Local Serverless Development using SAM Local
Sam Bragler