Prague AWS User Group

Serverless on AWS: Architectural Patterns and Best Practices

Vladimir Simek, Sr. Solutions Architect

17th September 2019

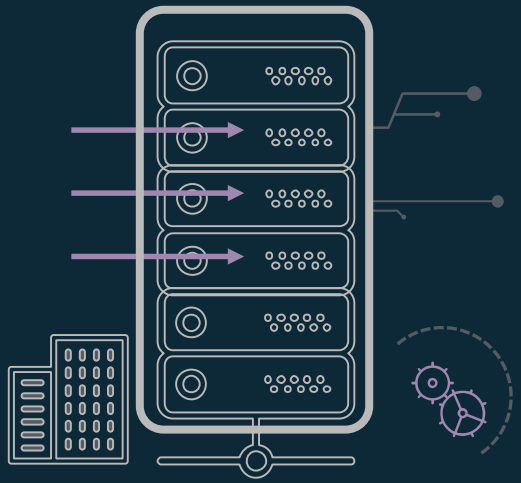
Prague AWS User Grooup

aws

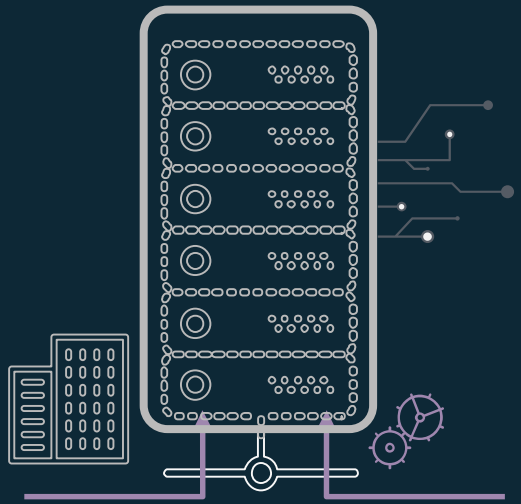
Evolution of computing

Let’s take a look at the evolution of computing

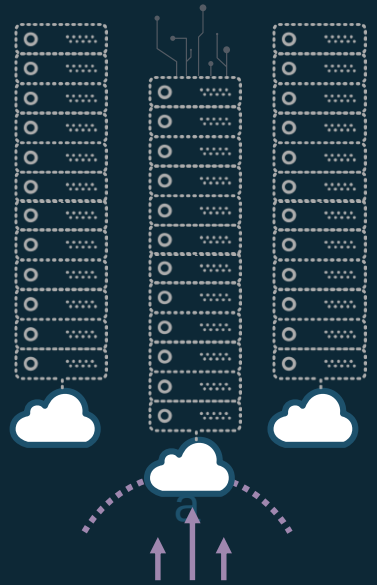
Physical Servers in Datacenters



Virtual Servers in Datacenters



Virtual Servers in the Cloud



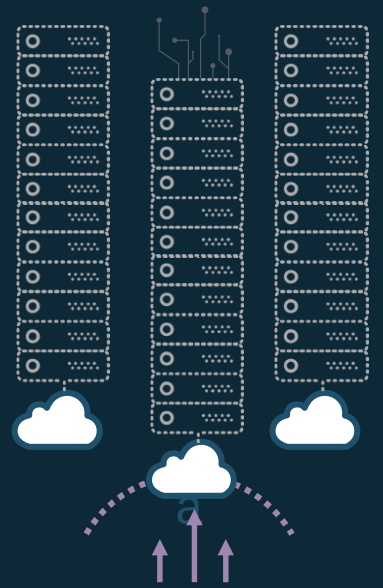
aws

Each progressive step was better

- Higher utilization
- Faster provisioning speed
- Improved uptime
- Disaster recovery
- Hardware independence

- Trade CAPEX for OPEX
- More scale
- Elastic resources
- Faster speed and agility
- Reduced maintenance
- Better availability and fault tolerance
- Better Automation

Virtual Servers in the Cloud



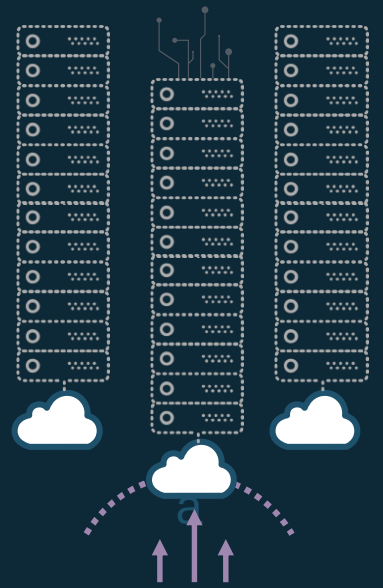
aws

But there are still limitations

- Still need to administer virtual servers
- Still need to manage capacity and utilization
- Still need to size workloads
- Still need to manage availability, fault tolerance
- Still expensive to run intermittent jobs

- Trade CAPEX for OPEX
- More scale
- Elastic resources
- Faster speed and agility
- Reduced maintenance
- Better availability and fault tolerance
- Better Automation

Virtual Servers in the Cloud



aws

The Next Step of Evolution - Serverless

Serverless means …

Serverless means…

No servers to provision or manage

Scales with usage

Never pay for idle

Availability and fault tolerance built in

aws

No server is easier to manage than "no server.”

Werner Vogels—Amazon CTO

Serverless means:

Serverless means:

Greater agility

Less overhead

Better focus

Increased scale

More flexibility

Faster time to market

aws

Serverless applications

Event source

Changes in data state

Requests to endpoints

Changes in resource state

Function

Node.js
Python
Java
C#
Go
Ruby
BYOR (Bring your own runtime)

Services (anything)

aws

Anatomy of a Lambda function

Handler() function

Function to be executed upon invocation

Event object

Data sent during Lambda Function Invocation

Context object

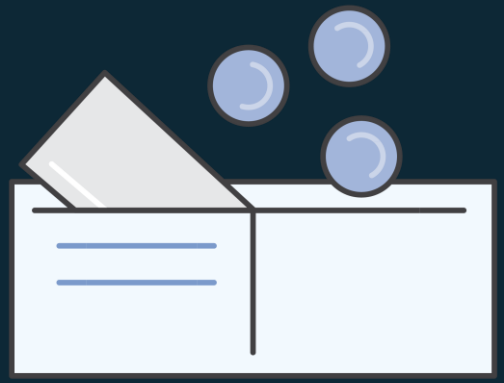
Methods available to interact with runtime information (request ID, log group, etc.)

```
public String handleRequest(Book book, Context context) {
    saveBook(book);

    return book.getName() + " saved!";
}
```


aws

Fine-Grained Pricing



Free Tier

1M requests and 400,000 GB-s of compute.

Every month, every customer.

Buy compute time in 100ms increments

Low request charge

No hourly, daily, or monthly minimums

No per-device fees

Never pay for idle

aws

AWS Lambda release history

*As of October 2018, does not include region launches

Lambda permissions model

Fine grained security controls for both execution and invocation:

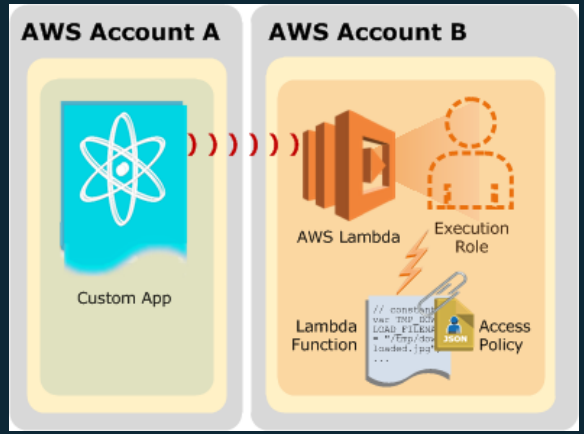
Execution policies:

- Define what AWS resources/API calls can this function access via IAM
- Used in streaming invocations
- E.g. “Lambda function A can read from DynamoDB table users”

Function policies:

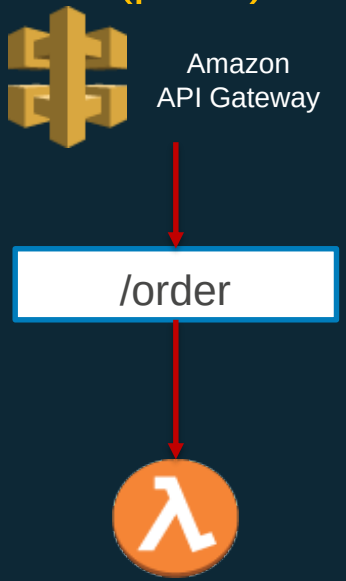
- Used for sync and async invocations
- E.g. “Actions on bucket X can invoke Lambda function Z"
- Resource policies allow for cross account access

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*"
        }
    ]
}
```



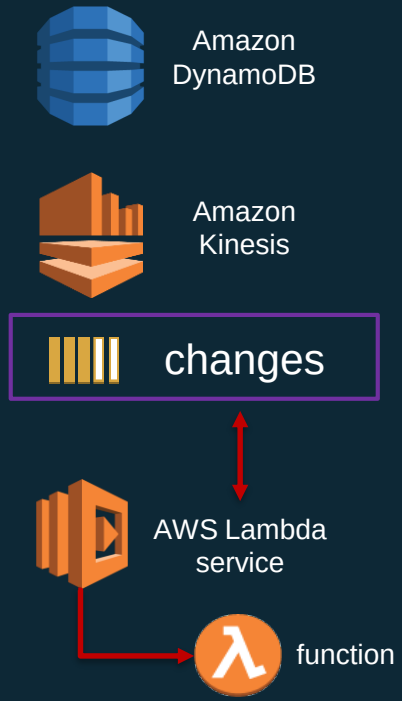
Lambda execution models

Synchronous (push)



Asynchronous (event)

Poll-based



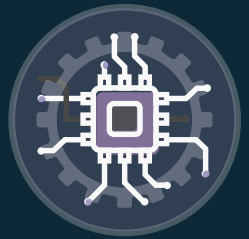
Common Lambda use cases

Web Applications

- Static websites
- Complex web apps
- Packages for Flask and Express

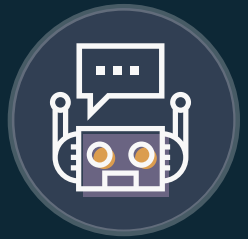
Backends

- Apps & services
- Mobile
- IoT



Data Processing

- Real time
- MapReduce
- Batch



Chatbots

- Powering chatbot logic

Amazon Alexa

- Powering voice-enabled apps
- Alexa Skills Kit

IT Automation

- Policy engines
- Extending AWS services
- Infrastructure management

aws

aws

Home > Applications

AWS Serverless Application Repository

Discover, deploy, and publish serverless applications

alexa

1 2 3

alexa-anagram

Alexa responds with the count and anagrams for a requested word

evanchiu 3 deployments

anagram alexa nodejs

alexa-random-restaurant

A basic python based back-end for an Alexa skill that randomly gives you an open restaurant in a specified city using the Yelp API.

Harsha Warrdhan Shar... 3 deployme...

alexa-smart-home-skill-adapter

Provides the basic framework for a skill adapter for a smart home skill.

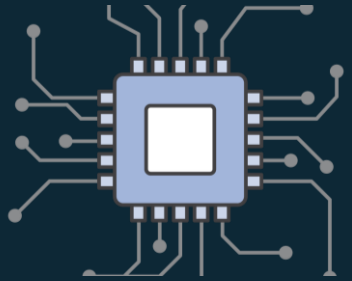
AWS 6 deployments

alexa-skills-kit-color-expert

Demonstrates a basic skill built with the Amazon Alexa Skills Kit.

AWS 26 deployments

Amazon API Gateway



Create a unified API frontend for multiple micro-services

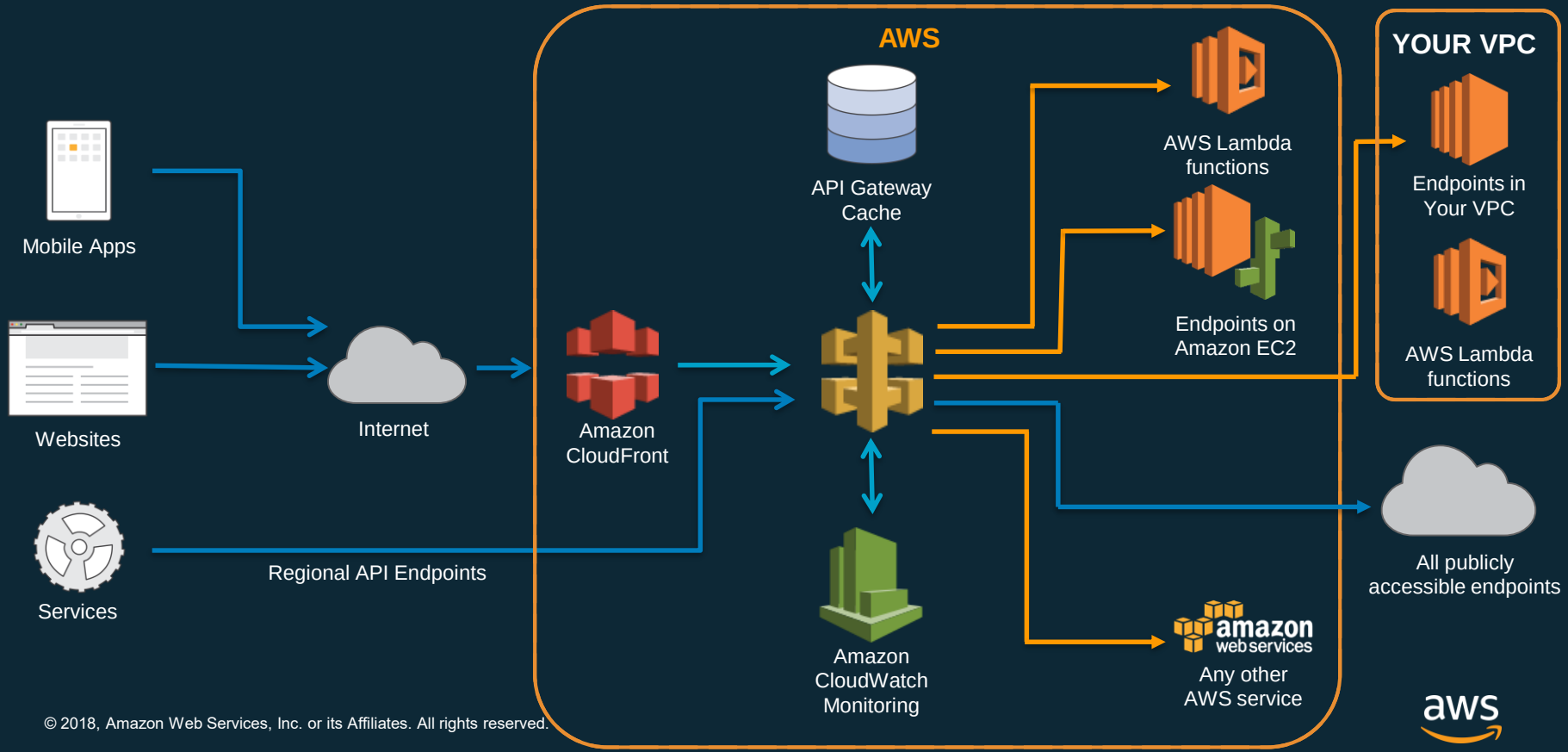
DDoS protection and throttling for your backend

Authenticate and authorize requests to a backend

Throttle, meter, and monetize API usage by 3rd party developers

aws

API Gateway integrations



Demo – Lambda function(s)

Lambda Best Practices

- Minimize package size to necessities
- Use Environment Variables to modify operational behavior
- Self-contain dependencies in your function package
- Delete large unused functions (75GB limit)
- Leverage “Max Memory Used” to right-size your functions

aws

Tweak your function’s computer power

Lambda exposes only a memory control, with the **% of CPU core and network capacity** allocated to a function proportionally

Is your code CPU, Network or memory-bound? If so, it could be **cheaper** to choose more memory.

aws

Smart resource allocation

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

Stats for Lambda function that calculates **1.000 times** all prime numbers **<= 1.000.000**

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

Green==Best **Red**==Worst

aws

Smart resource allocation

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

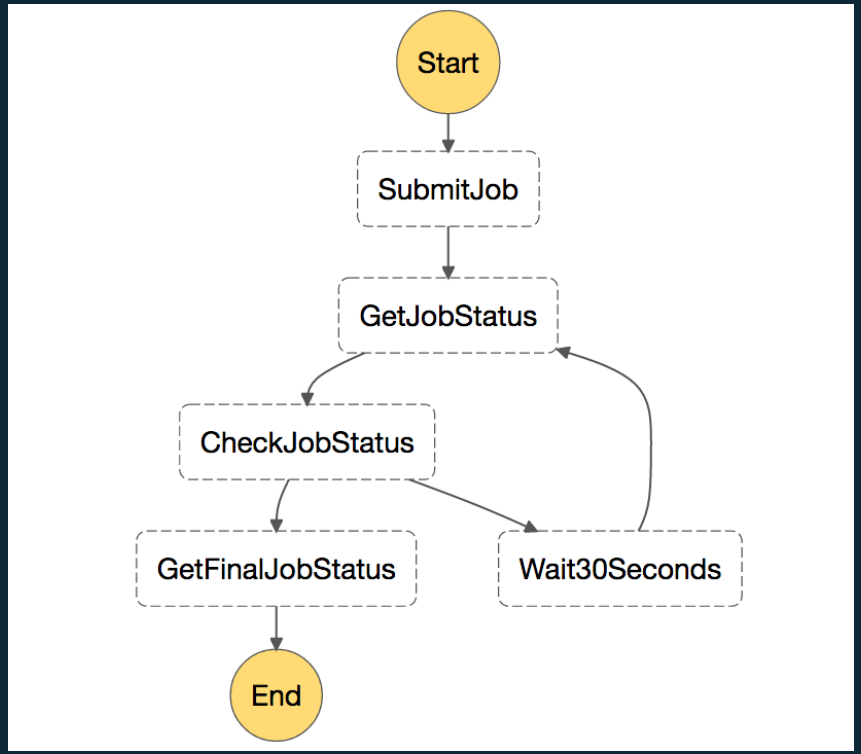
Stats for Lambda function that calculates **1.000 times** all prime numbers **<= 1.000.000**

128 MB	11.722965sec	$0.024628
256 MB	6.678945sec	$0.028035
512 MB	3.194954sec	$0.026830
1024 MB	1.465984sec	$0.024638
	-10.256981sec	+$0.00001

Green==Best **Red**==Worst

aws

Best Practice: Keep orchestration out of code.

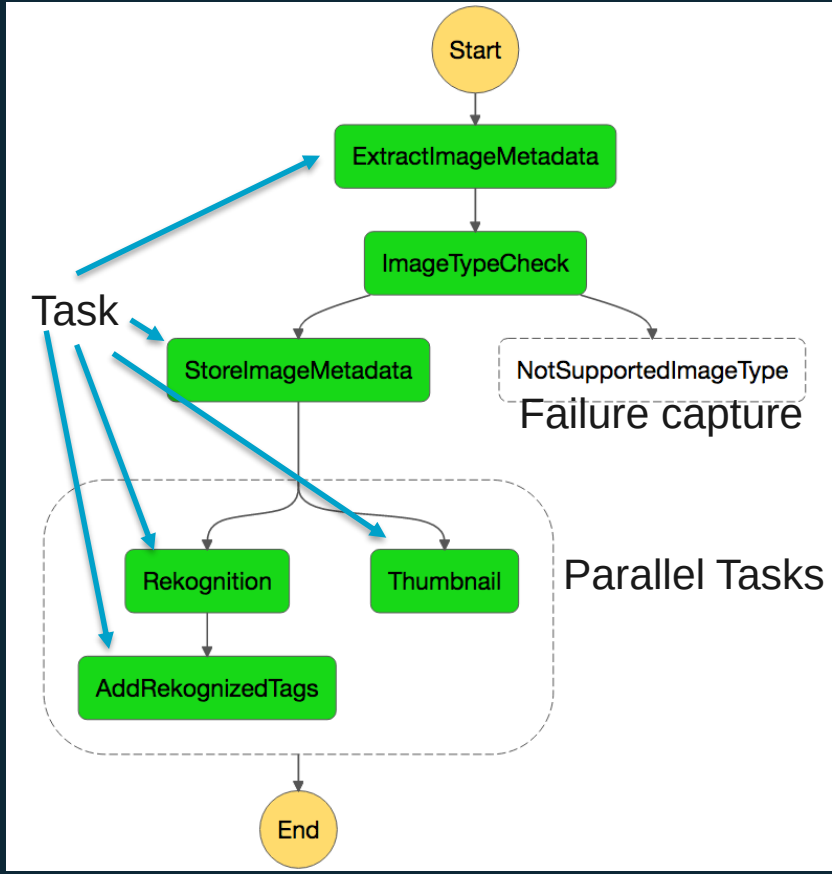


Keep orchestration out of code.

AWS Step Functions

“Serverless” workflow management with zero administration

- Makes it easy to coordinate the components of distributed applications and microservices using visual workflows
- Automatically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected
- Logs the state of each step, so when things do go wrong, you can diagnose and debug problems quickly




Best Practice: Monitor!
https://secure.flickr.com/photos/jasoneppink/499531891

Metrics and logging are a universal right

CloudWatch Metrics:

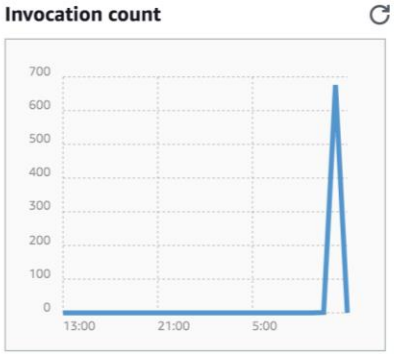
- 7 Built in metrics for Lambda
  - Invocation Count, Invocation duration, Invocation errors, Throttled Invocation, Iterator Age, DLQ Errors, Concurrency
  - Can call “put-metric-data” from your function code for custom metrics
- 7 Built in metrics for API-Gateway
  - API Calls Count, Latency, 4XXs, 5XXs, Integration Latency, Cache Hit Count, Cache Miss Count
  - Error and Cache metrics support *averages and percentiles*

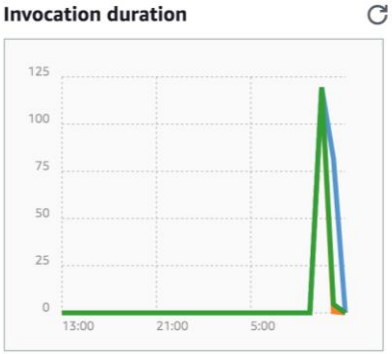
aws

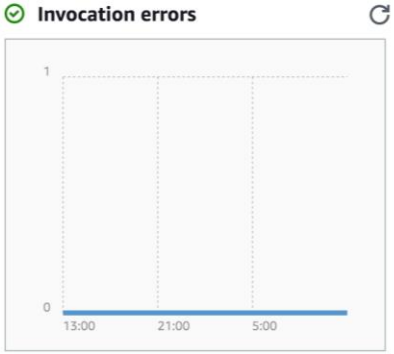
CloudWatch metrics at a glance (last 24 hours)

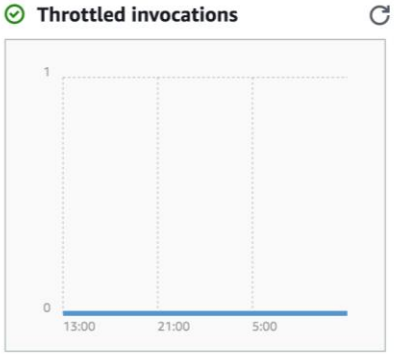
View traces in X-Ray

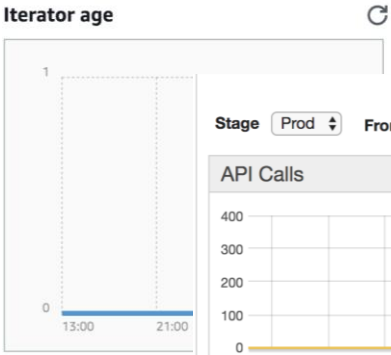
View logs in CloudWatch



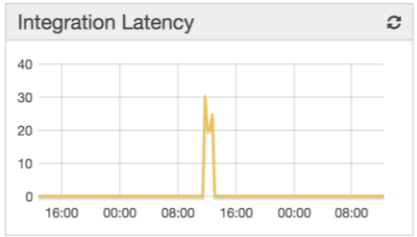


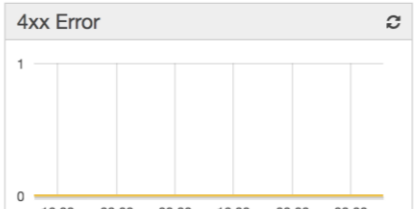


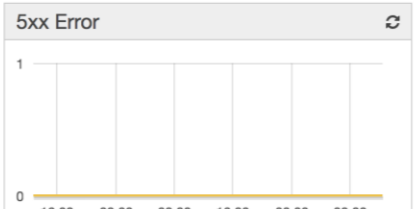




Stage Prod From 11/23/17 To 11/24/17

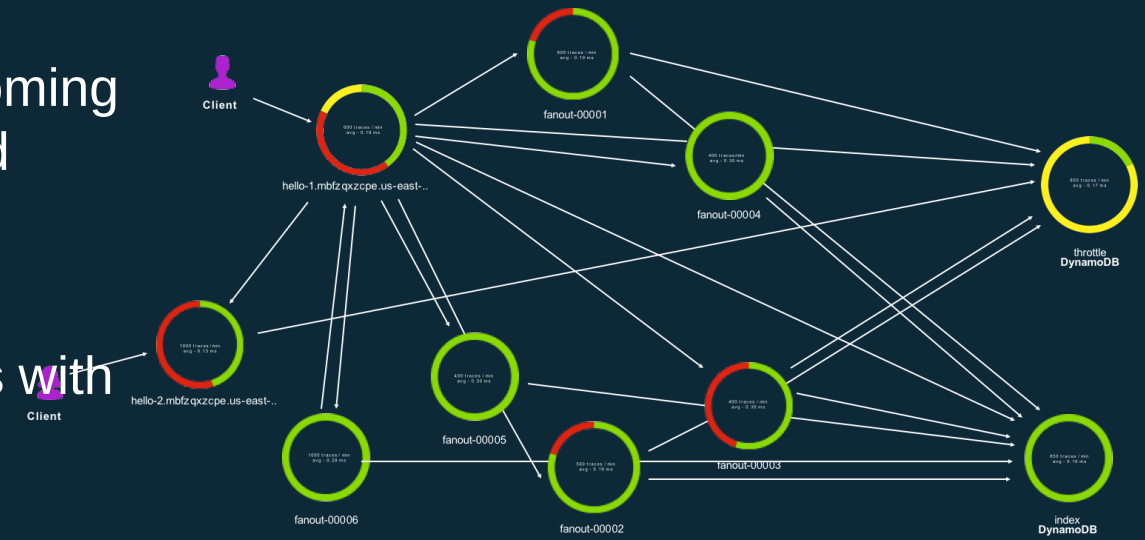




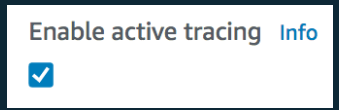


AWS X-Ray Integration with Serverless

- Lambda instruments incoming requests for all supported languages
- Lambda runs the X-Ray daemon on all languages with an SDK

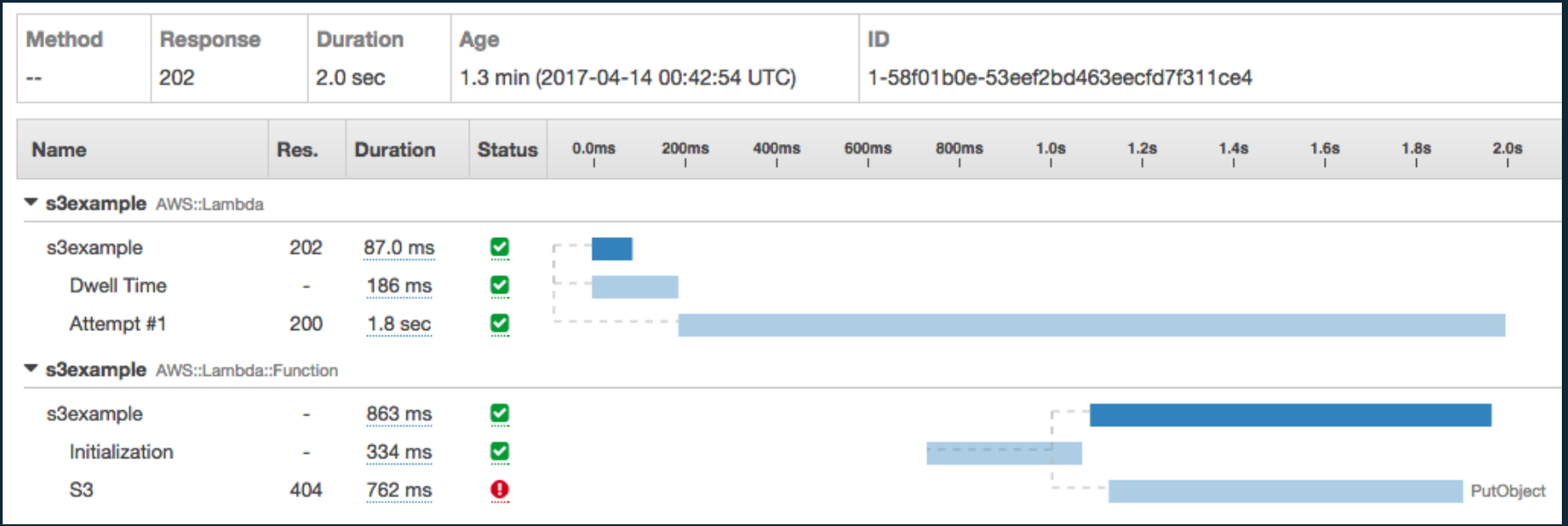



```
var AWSXRay = require('aws-xray-sdk-core');
AWSXRay.middleware.setSamplingRules('sampling-rules.json');
var AWS = AWSXRay.captureAWS(require('aws-sdk'));
S3Client = AWS.S3();
```



aws

X-Ray Trace Example



Best Practice: Leverage other Serverless services

"On EC2"

Amazon EC2

Microsoft SQL Server

kafka

cassandra

docker

Managed

Amazon EMR

Amazon ES

Amazon ElastiCache

Amazon Redshift

Amazon RDS

Serverless

AWS Lambda

Amazon Cognito

Amazon Kinesis

Amazon S3

Amazon DynamoDB

Amazon SQS

Amazon API Gateway

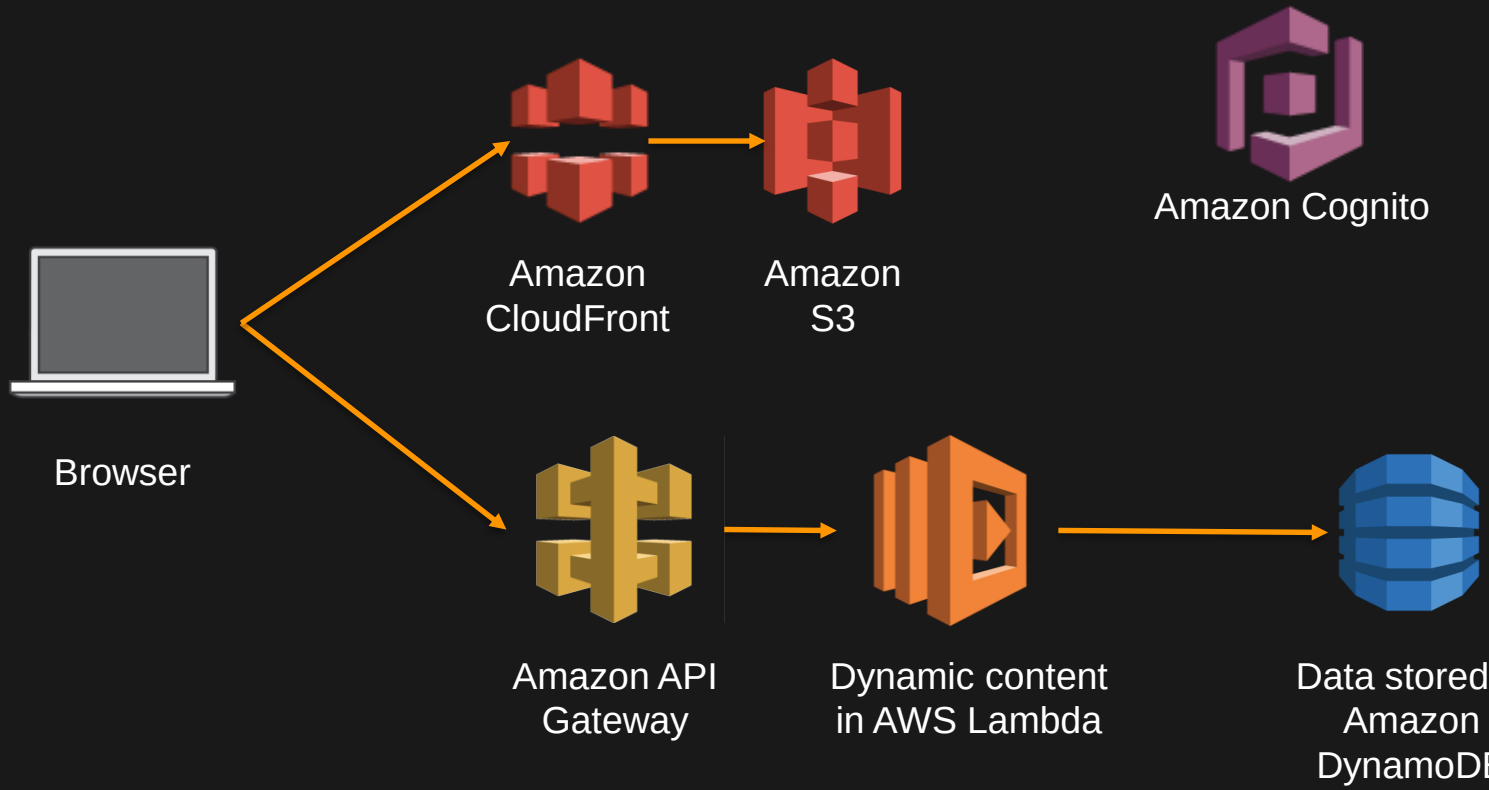
Amazon CloudWatch

AWS IoT

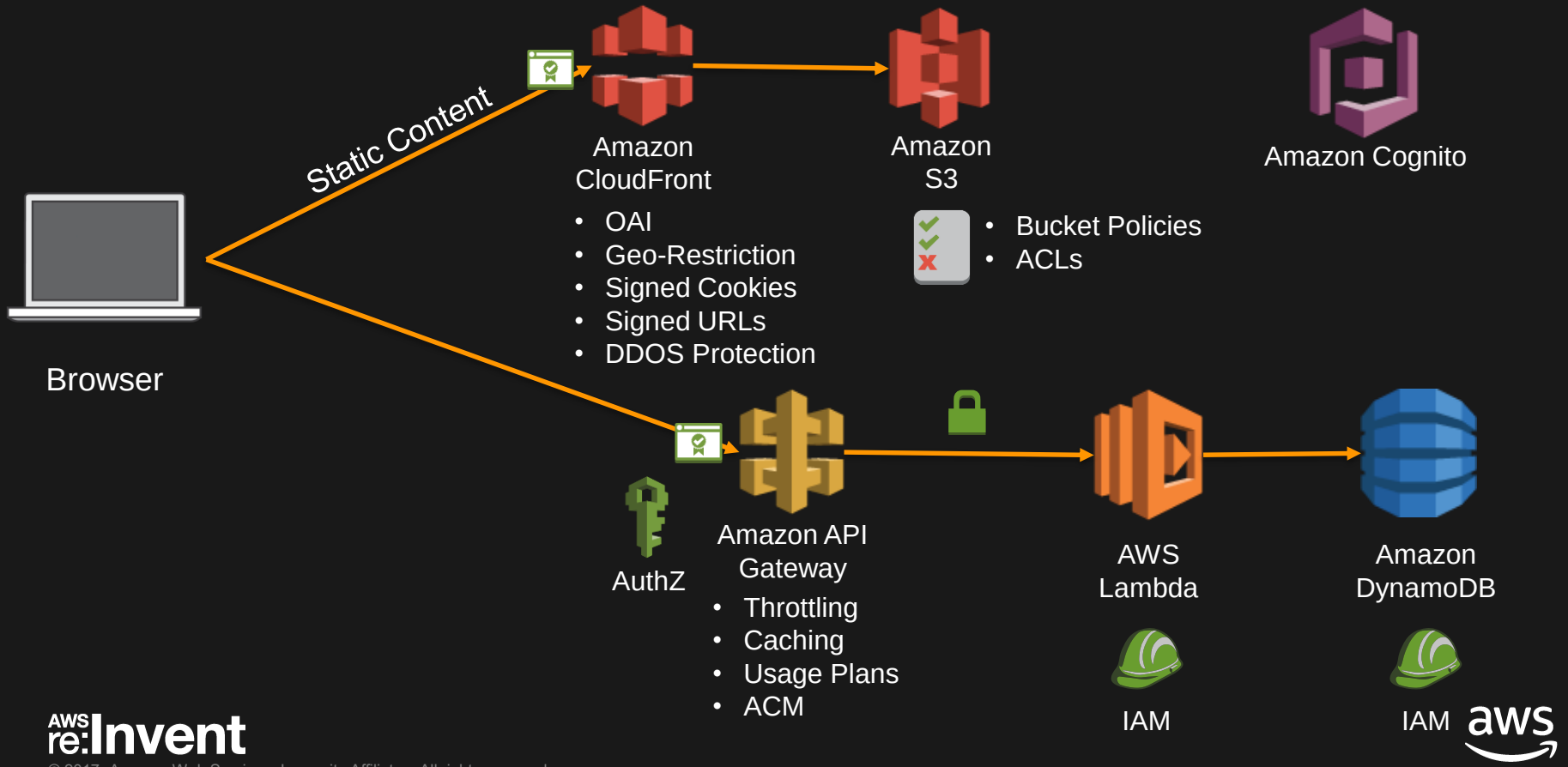
aws

Pattern 1: Serverless Web Apps

Web application

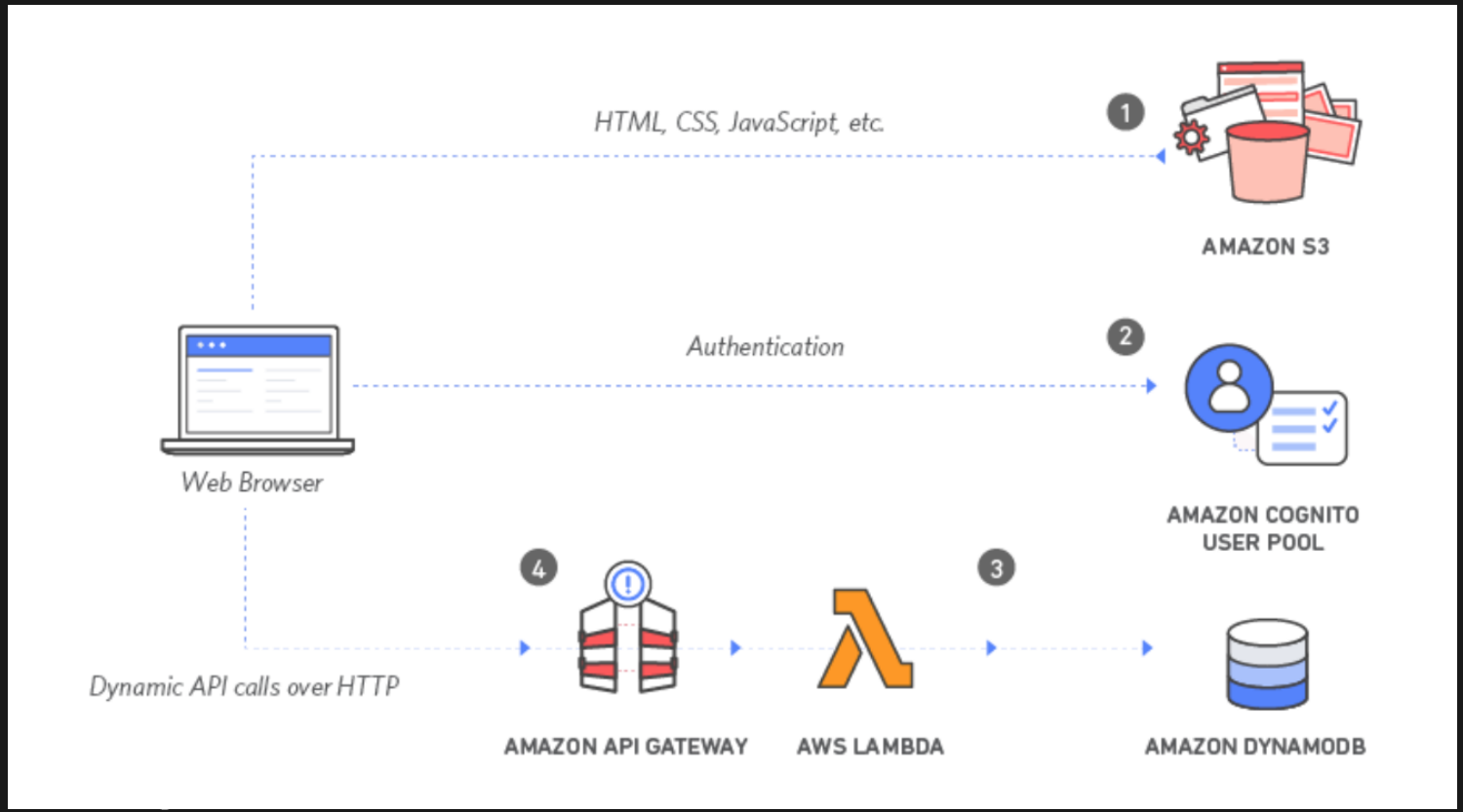


Serverless web app security



Demo – Wild Rydes

Wild Rydes Architecture



Pattern 2: Serverless Data Lake

Serverless Data Lake Characteristics

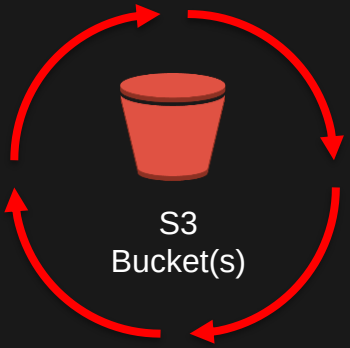
- Collect/Store/Process/Consume and Analyze all organizational data
- Structured/Semi-Structured/Unstructured data
- AI/ML and BI/Analytical use cases
- Fast automated ingestion
- Schema on Read
- Complementary to EDW
- Decoupled Compute and Storage

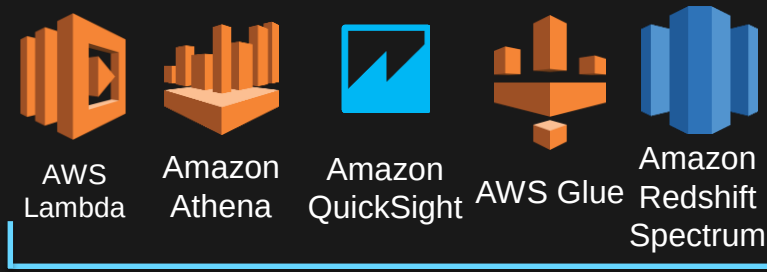
AWS Serverless Data Lake

Catalog & Search

API/UI

Ingest





Analytics & Processing

Security & Auditing

AWS re:Invent

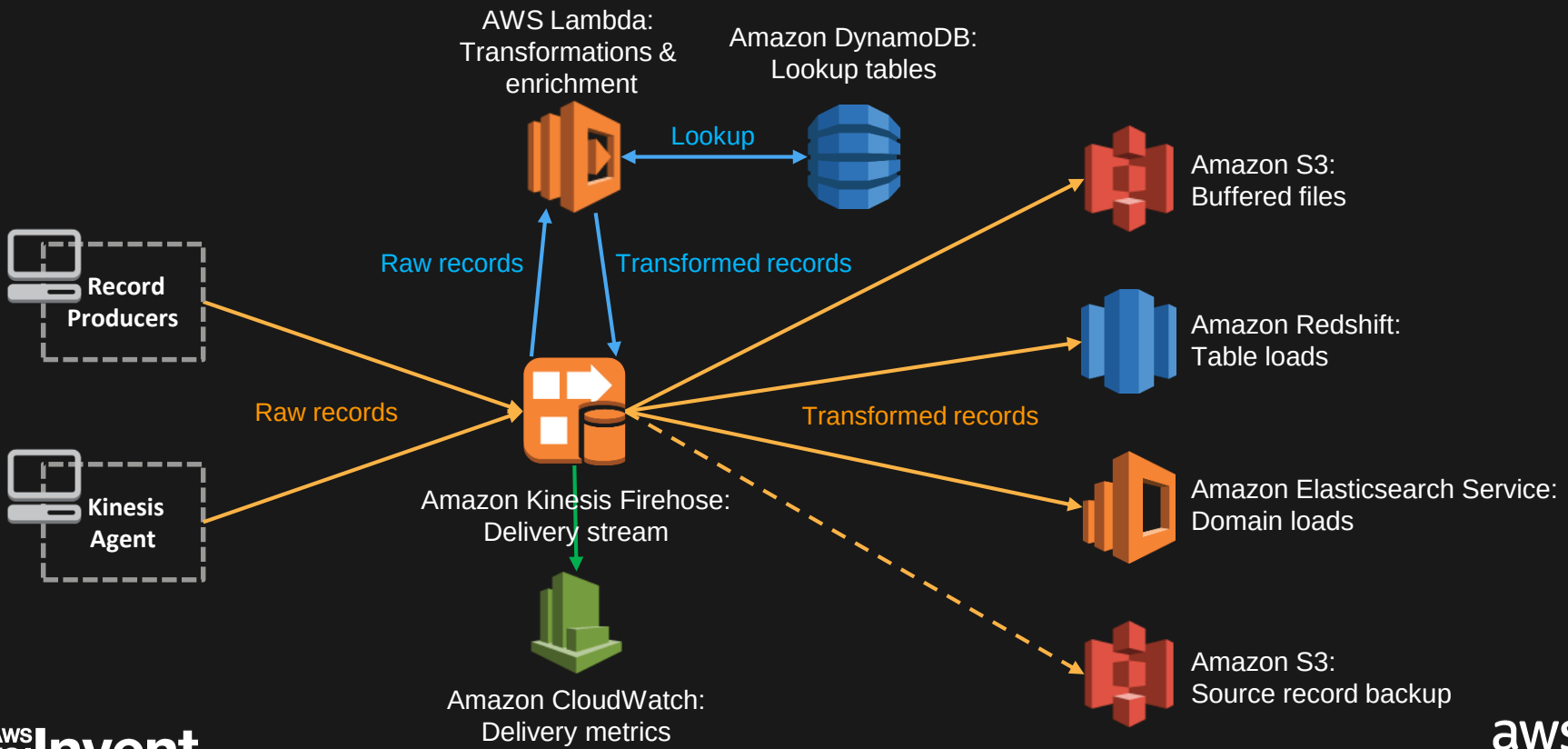
aws

Pattern 3: Stream Processing

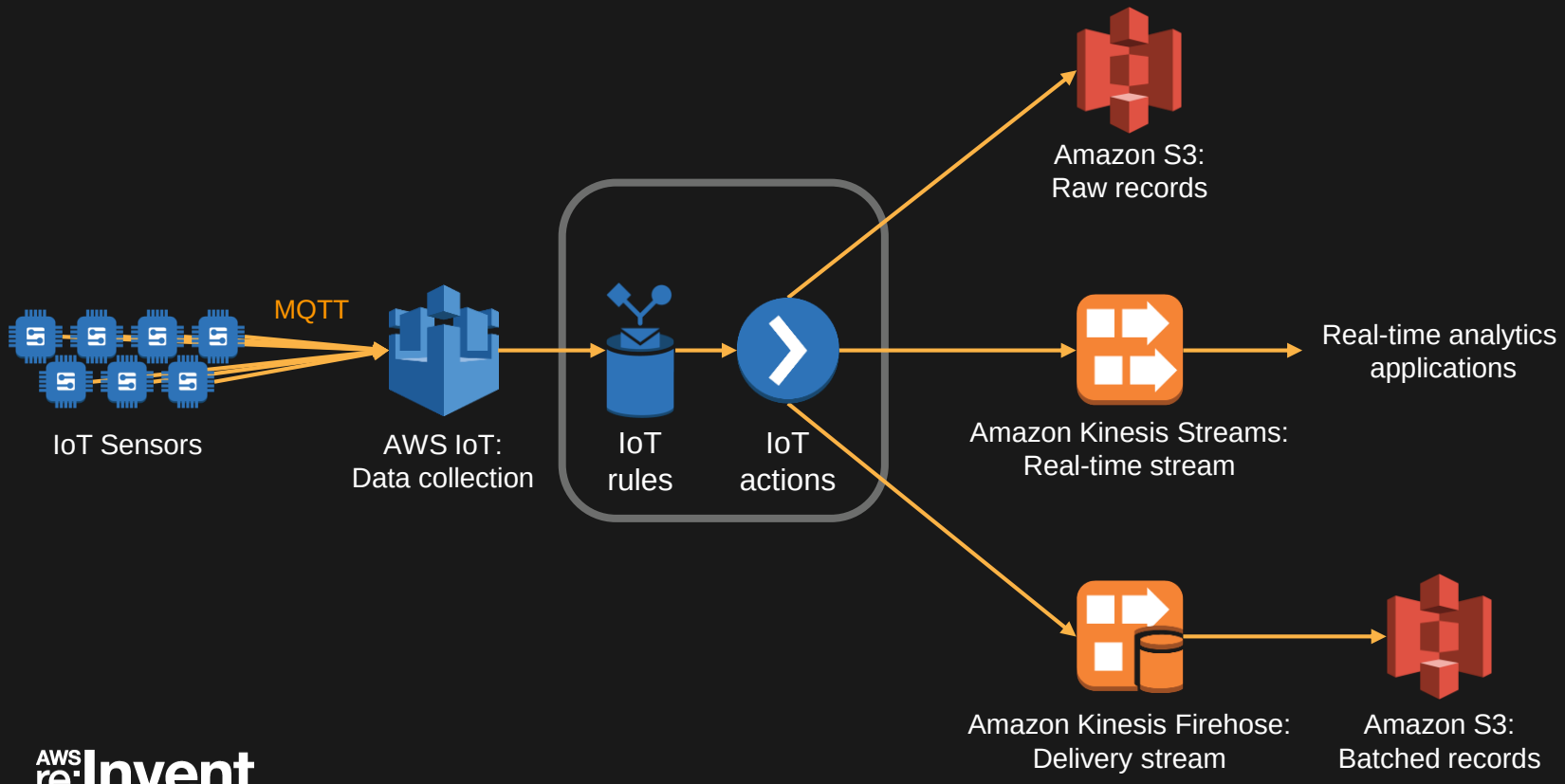
Stream processing characteristics

- High ingest rate
- Near real-time processing (low latency from ingest to process)
- Spiky traffic (lots of devices with intermittent network connections)
- Message durability
- Message ordering

Streaming data ingestion



Sensor data collection



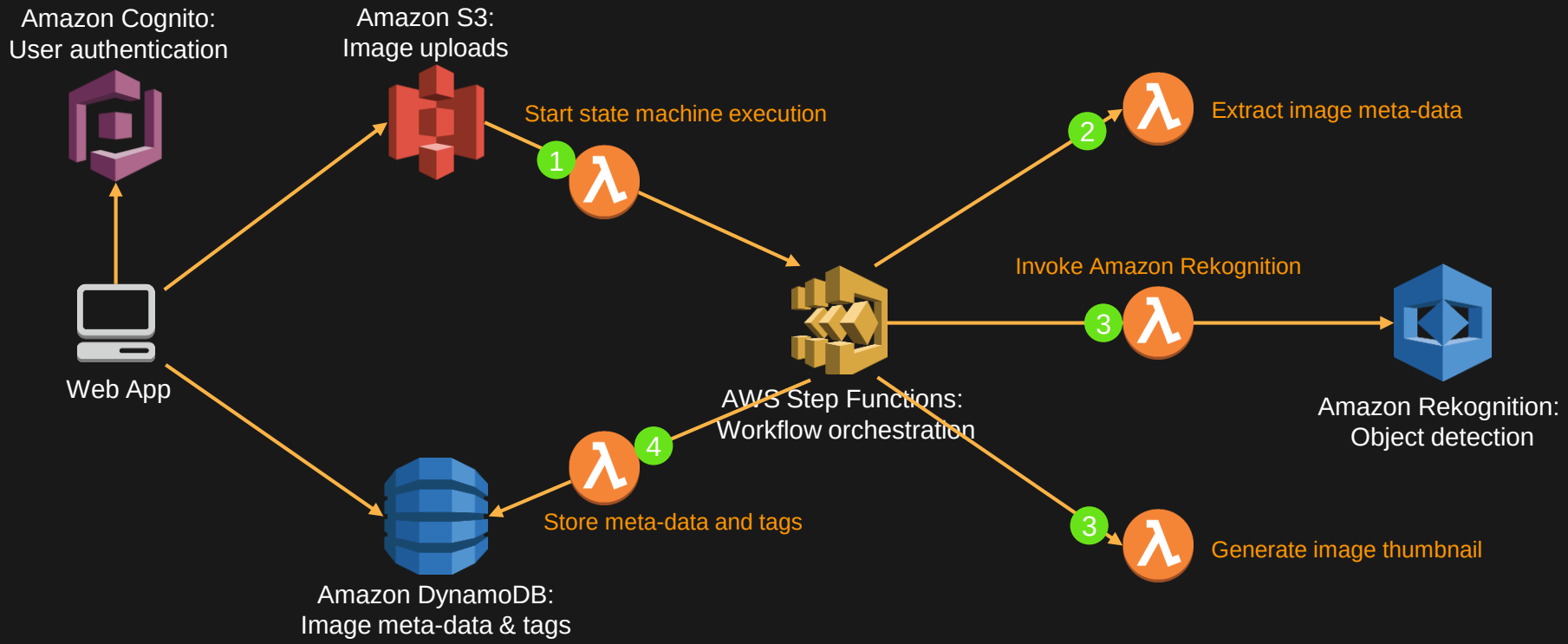
Pattern 4: Operations Automation

Automation characteristics

- Periodic jobs
- Event triggered workflows
- Enforce security policies
- Audit and notification
- Respond to alarms
- Extend AWS functionality

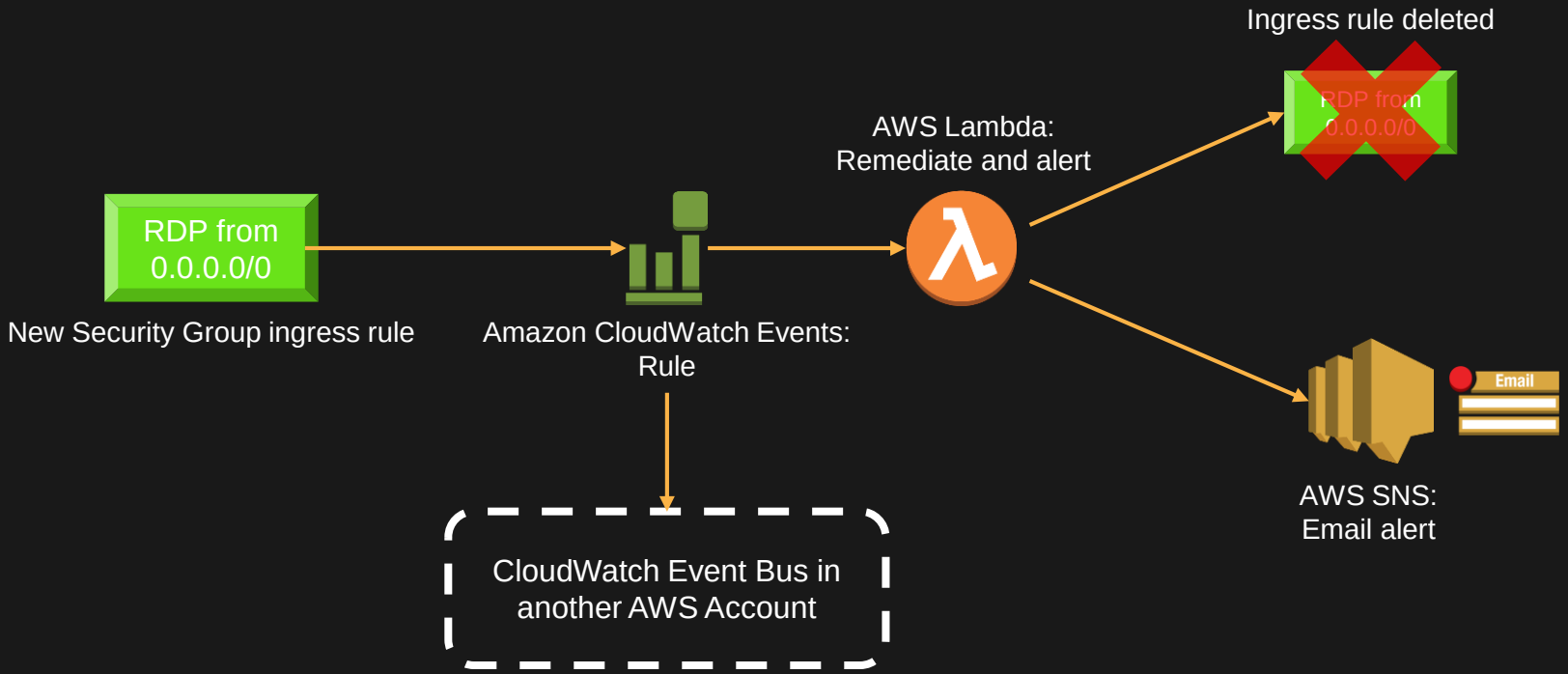
… All while being Highly Available, Scalable and Auditable

Image recognition and processing



https://github.com/awslabs/lambda-refarch-imagerecognition

Enforce security policies



https://secure.flickr.com/photos/stevendepolo/5749192025/

Start with a framework

AWS:

AWS SAM

AWS Amplify

AWS Chalice

Third-party:

APEX

architect

Claudia.js

Serverless Framework

SPARTA

AWS Serverless Application Model (SAM)

AWS CloudFormation extension optimized for serverless

New serverless resource types: functions, APIs, and tables

Supports anything AWS CloudFormation supports

Open specification (Apache 2.0)

https://aws.amazon.com/serverless/sam

AWS SAM Command Line Interface (AWS CLI)

CLI tool for local development, debugging, testing, deploying, and monitoring of serverless applications

Supports API Gateway “proxy-style” and Lambda service API testing

Response object and function logs available on your local machine

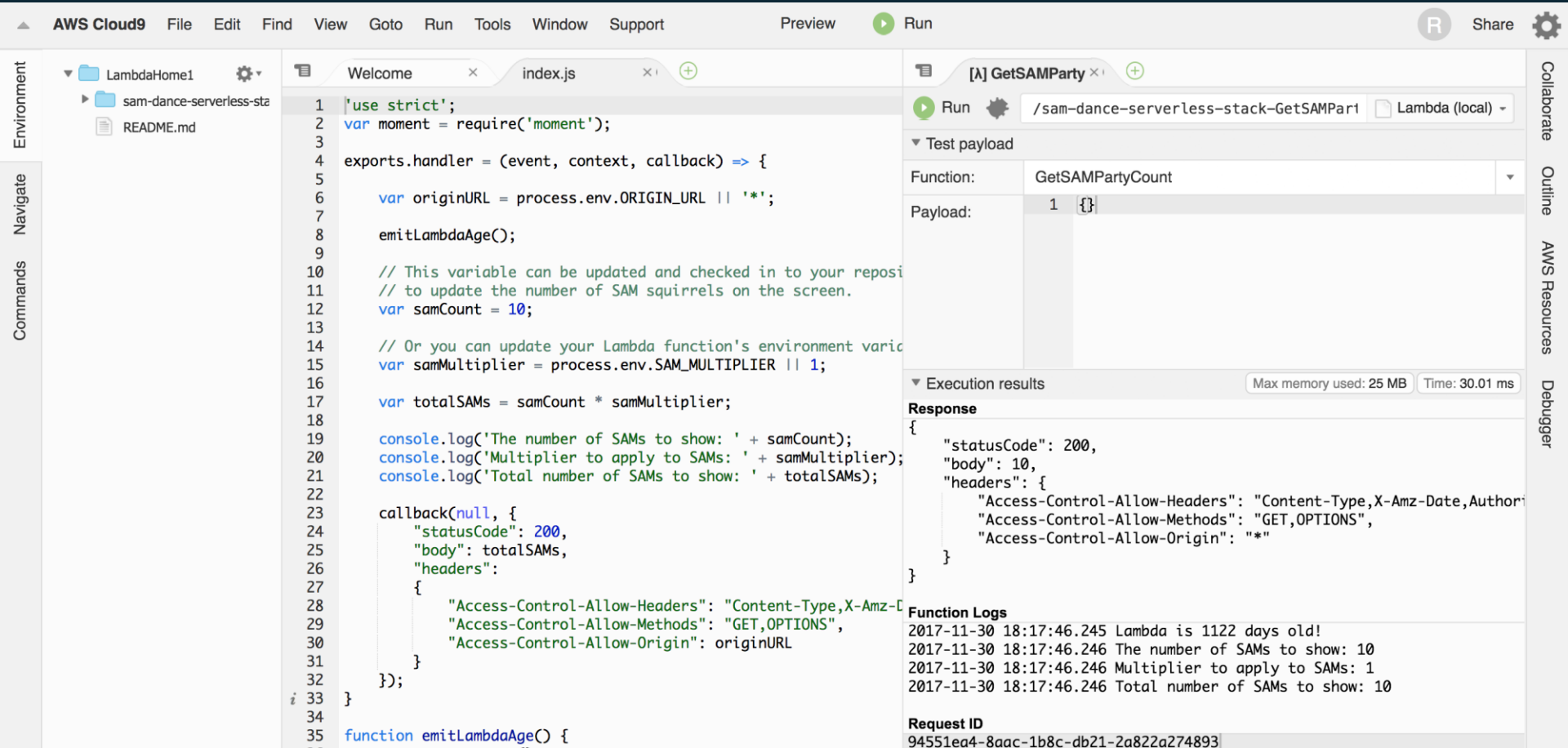
Uses open source docker-lambda images to mimic Lambda’s execution environment such as timeout, memory limits, runtimes

Can tail production logs from CloudWatch logs

https://aws.amazon.com/serverless/sam

aws

AWS Cloud9


AWS Cloud9 File Edit Find View Goto Run Tools Window Support Preview Run Share
Environment
Navigate
Commands
LambdaHome1
sam-dance-serverless-sta
README.md
Welcome
index.js
'use strict';
var moment = require('moment');
exports.handler = (event, context, callback) => {
var originURL = process.env.ORIGIN_URL || '*';
emitLambdaAge();
// This variable can be updated and checked in to your reposi
// to update the number of SAM squirrels on the screen.
var samCount = 10;
// Or you can update your Lambda function's environment varic
var samMultiplier = process.env.SAM_MULTIPLIER || 1;
var totalSAMs = samCount * samMultiplier;
console.log('The number of SAMs to show: ' + samCount);
console.log('Multiplier to apply to SAMs: ' + samMultiplier);
console.log('Total number of SAMs to show: ' + totalSAMs);
callback(null, {
"statusCode": 200,
"body": totalSAMs,
"headers":
{
"Access-Control-Allow-Headers": "Content-Type,X-Amz-D
"Access-Control-Allow-Methods": "GET,OPTIONS",
"Access-Control-Allow-Origin": originURL
}
});
}
function emitLambdaAge() {
[λ] GetSAMParty
Run
/sam-dance-serverless-stack-GetSAMPart
Lambda (local)
Test payload
Function:
GetSAMPartyCount
Payload:
{}
Execution results
Max memory used: 25 MB
Time: 30.01 ms
Response
{
"statusCode": 200,
"body": 10,
"headers": {
"Access-Control-Allow-Headers": "Content-Type,X-Amz-Date,Author
"Access-Control-Allow-Methods": "GET,OPTIONS",
"Access-Control-Allow-Origin": "*"
}
}
Function Logs
2017-11-30 18:17:46.245 Lambda is 1122 days old!
2017-11-30 18:17:46.246 The number of SAMs to show: 10
2017-11-30 18:17:46.246 Multiplier to apply to SAMs: 1
2017-11-30 18:17:46.246 Total number of SAMs to show: 10
Request ID
94551ea4-8aac-1b8c-db21-2a822a274893
Collaborate
Outline
AWS Resources
Debugger

SERVERLESS AT SCALE IS THE NEW NORM

THOMSON REUTERS

processes **4,000 requests** per second

FINRA

processes **half a trillion** validations of stock trades daily

Zillow

executes **16 million** requests a month

mlbam

ingests, analyzes and stores **17+ petabytes of data** per season

Driver & Vehicle Licensing Agency

API traffic to register and license more than **47 million driver records** in Great Britain,

Localytics

processes **tens of billions** of data points monthly

aws

SERVERLESS CUSTOMERS

aws

Further Resources

AWS Serverless https://aws.amazon.com/serverless/

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

AWS SAM CLI (Beta) https://github.com/awslabs/aws-sam-cli

Wild Rydes with Unicorns (serverless web app)

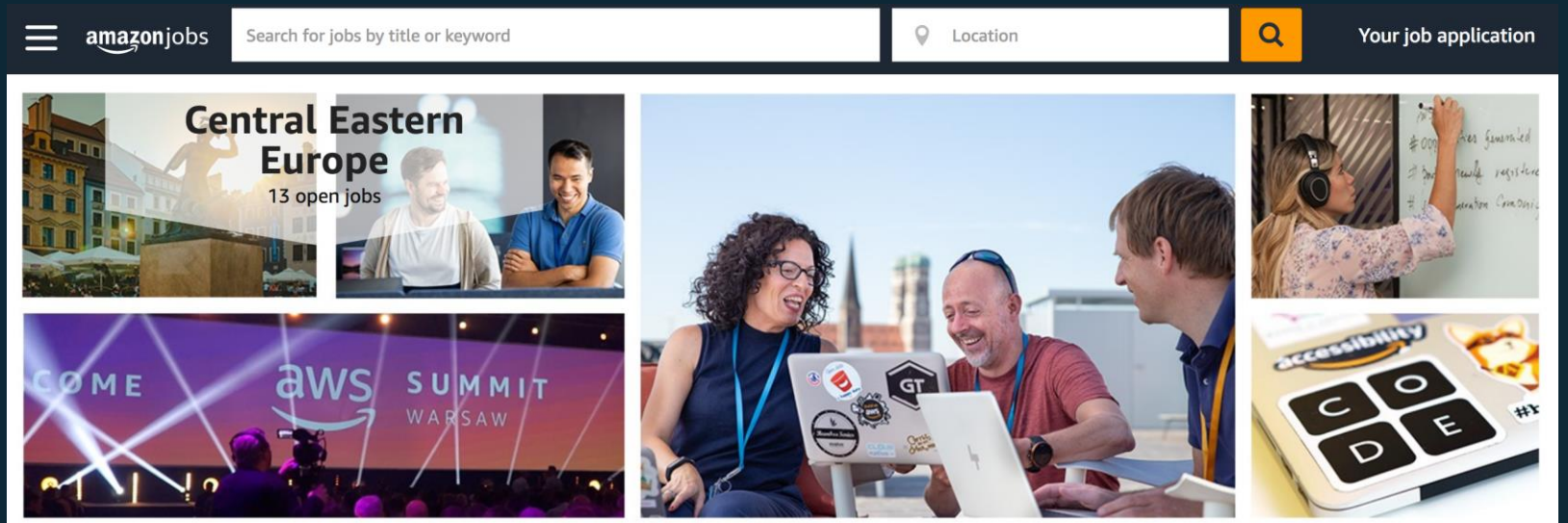
https://aws.amazon.com/getting-started/projects/build-serverless-web-app-lambda-apigateway-s3-dynamodb-cognito/

aws

Q & A

We’re hiring – AWS in CEE

https://www.amazon.jobs/en/landing_pages/aws-central-eastern-europe



Building the future of cloud in Central Eastern Europe

Would you like to work with some of the most innovative AWS customers in CEE, helping them to scale and improve their capability?

AWS is growing, and as we expand into new territories we need local experts who want to build the future of cloud from the ground up. Our Account Managers and Solutions Architects deliver on a strategy to build awareness and adoption of AWS, while working closely with our customers to enable continued success on the platform.

If you have a passion for technology, knowledge of the CEE market, and want to be a part of the most comprehensive and broadly adopted cloud platform in the world, view our open positions below.

Thank you

vladsim@amazon.com