aws

INNOVATE 2018

ONLINE CONFERENCE

DEVELOPER EDITION

Serverless Architectural Patterns

Julio Faerman @faermanj

@awscloud

#AWSInnovate

NDC { Oslo }

5

Motivation or Hype? Time & Money

$9 Million vs. $ 500

A spectrum of managed services

"On EC2"

Amazon EC2

Microsoft SQL Server

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

Let’s talk about…

Event Processing Architecture
Operations Automation Architecture
Web Application Architecture
Stream Processing Architecture
Telemetry Processing Architecture

Serverless means…

No servers to provision or manage

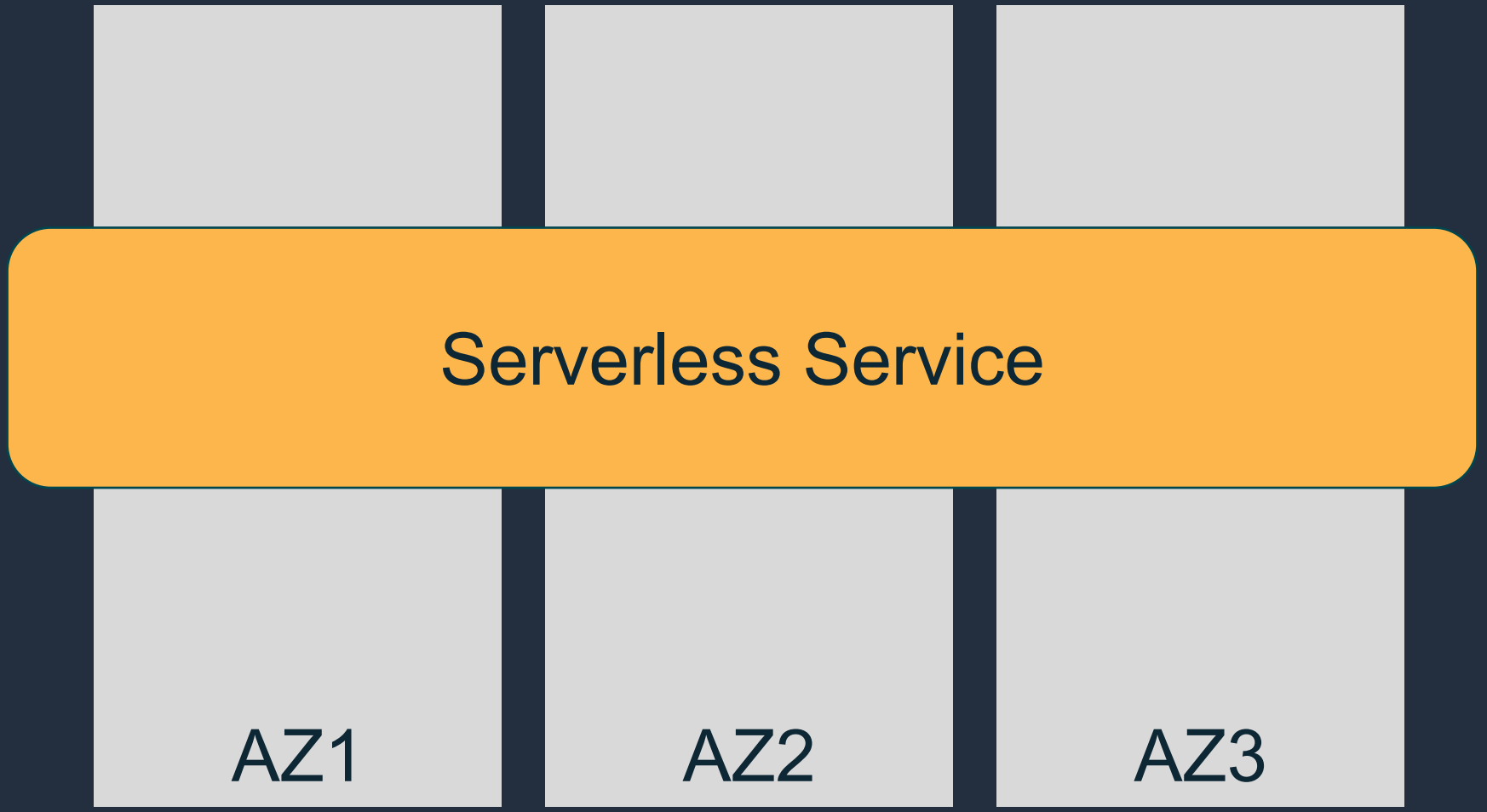
Scales with usage

Never pay for idle

Availability and fault tolerance built in

aws

Regional services



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.)

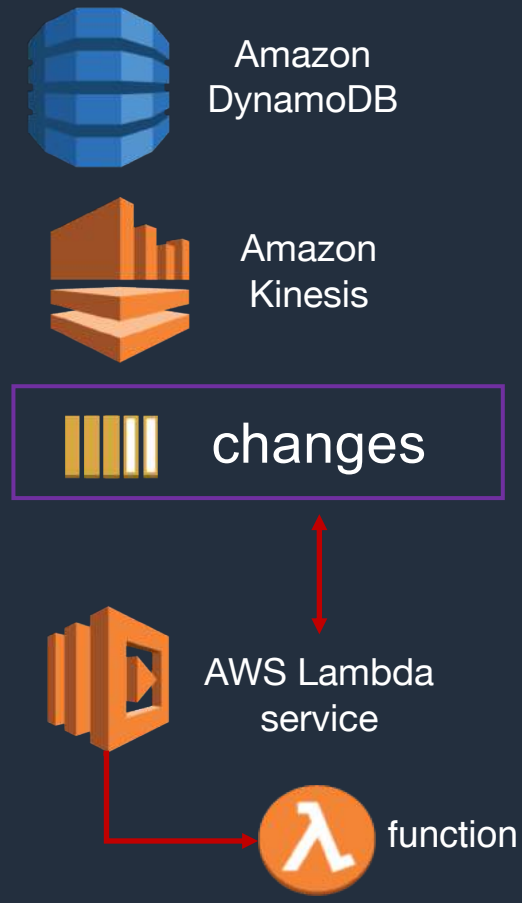
```
def handler(event, context):
    return {
        "message": ”Hello World!",
        "event": event
}
```

Lambda execution model

Synchronous (push)

Asynchronous (event)

Stream-based



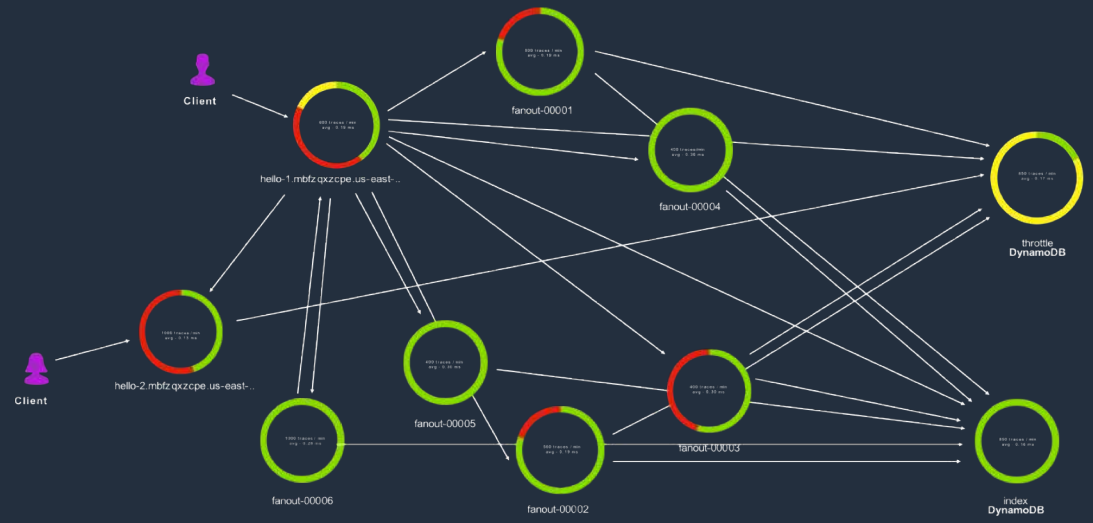
aws

Lambda Best Practices

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

AWS X-Ray Integration with Serverless

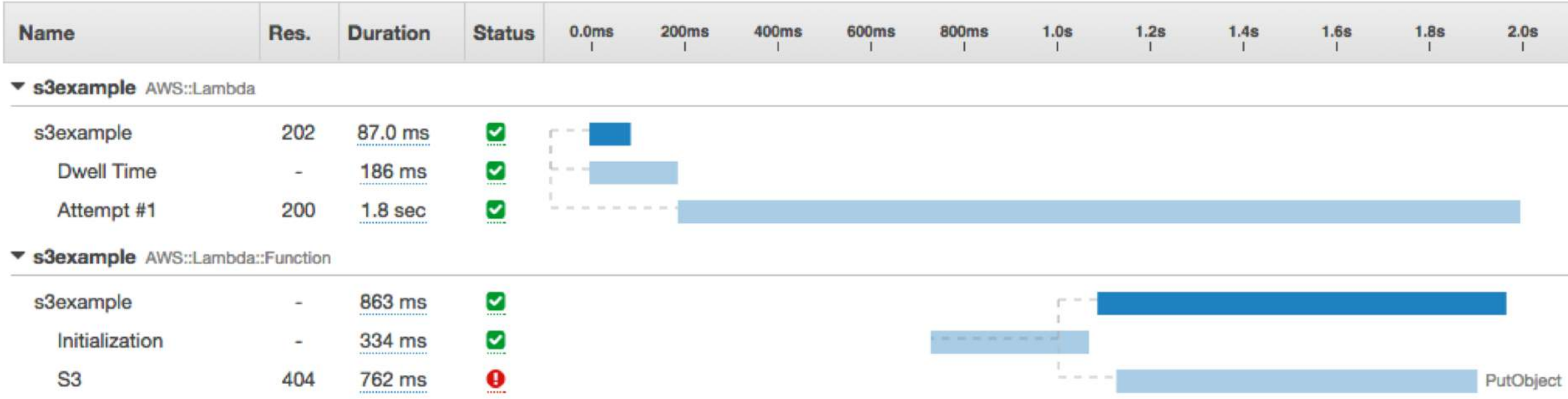
- Lambda instruments incoming requests for all supported languages
- Lambda runs the X-Ray daemon on all languages with an SDK




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

X-Ray Trace Example

Method	Response	Duration	Age	ID
--	202	2.0 sec	1.3 min (2017-04-14 00:42:54 UTC)	1-58f01b0e-53eef2bd463eecfd7f311ce4



Serverless Frameworks

Chalice

awslabs/aws-serverless-express

awslabs/aws-serverless-java-container

aws

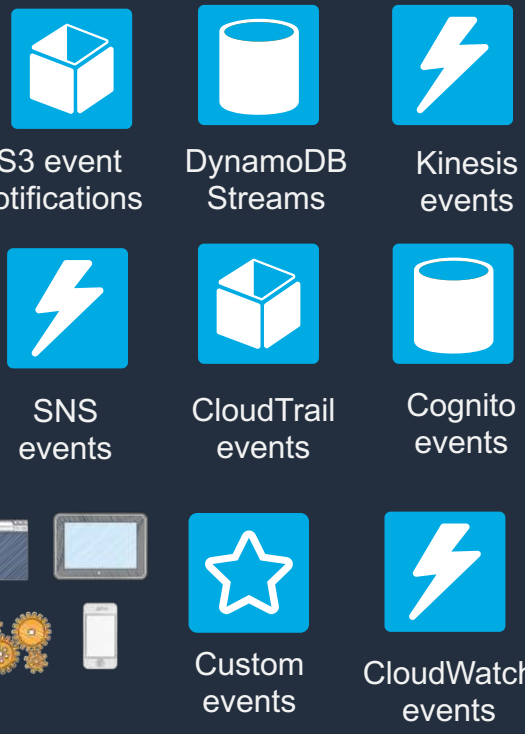
Event Processing Architecture

Event driven

Event A on B triggers C

Event-driven platform

Invoked in response to events
- **Changes in data**
- **Changes in state**



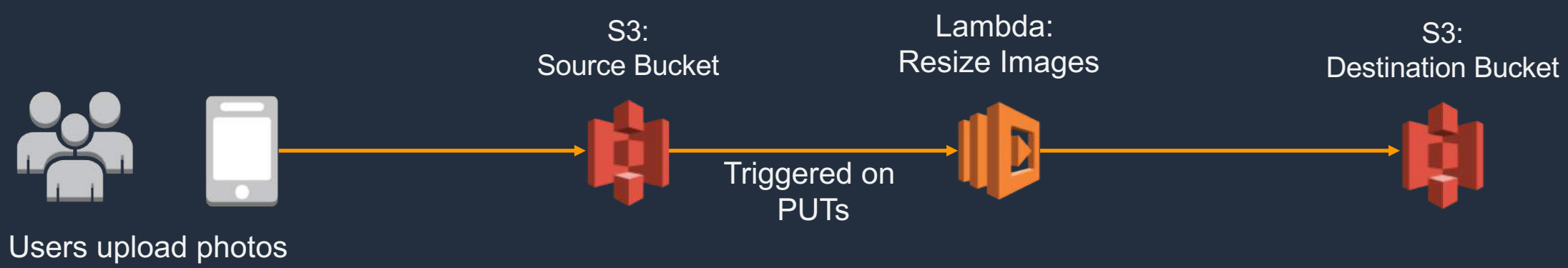
Lambda functions

Access any service, including your own

Any custom

Such as…

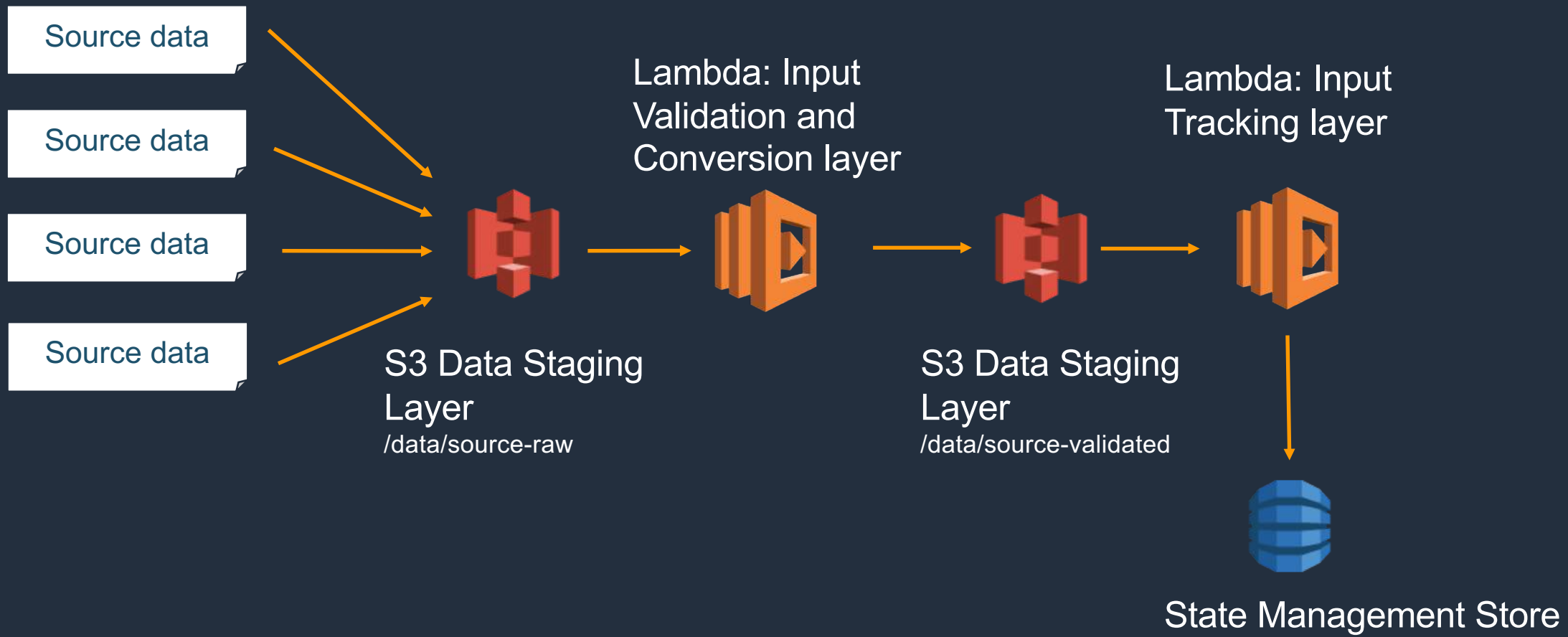
Event-driven actions



THOMSON REUTERS™ CMP.LY The Seattle Times NETFLIX FireEye

aws

Event-driven controls

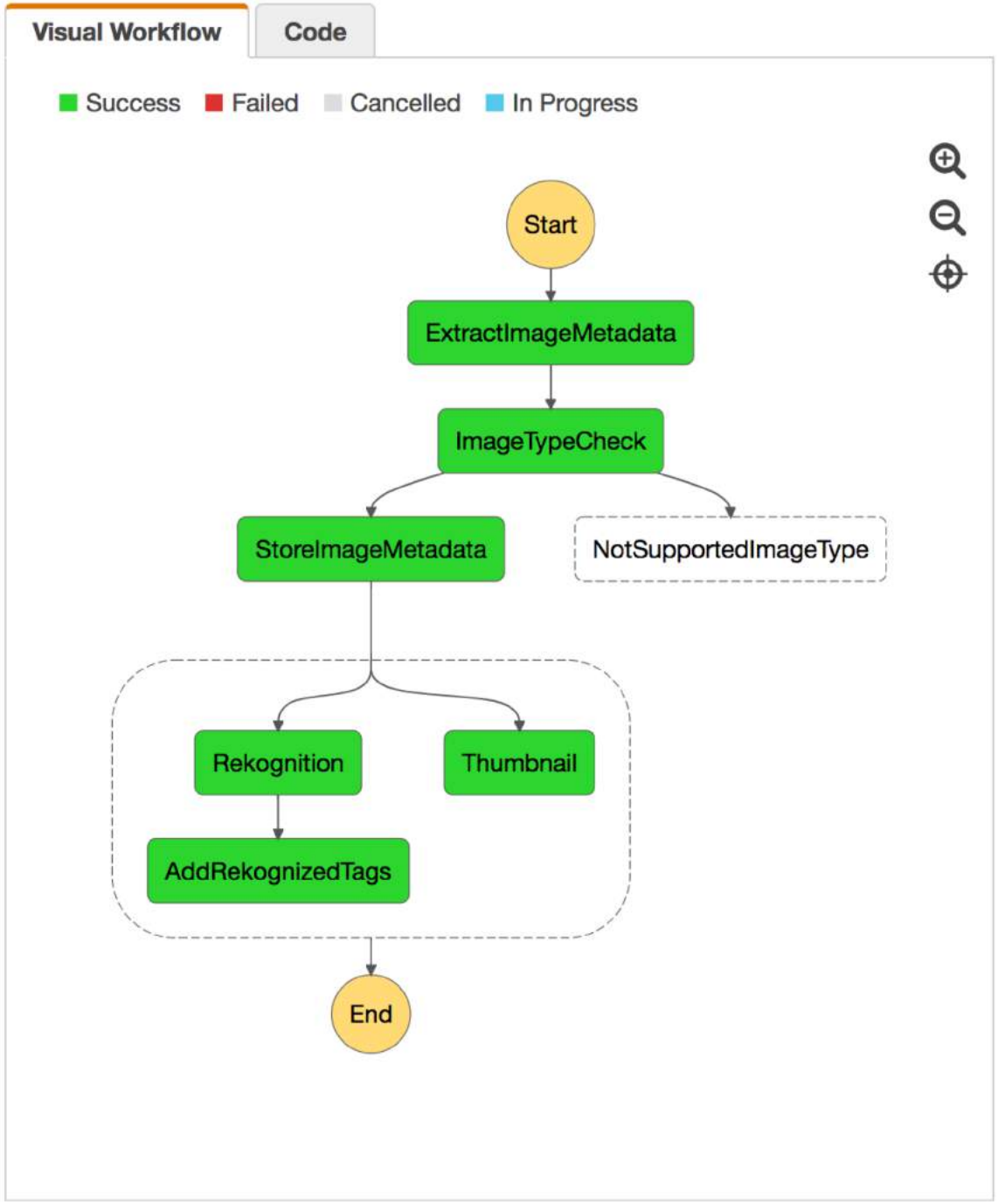


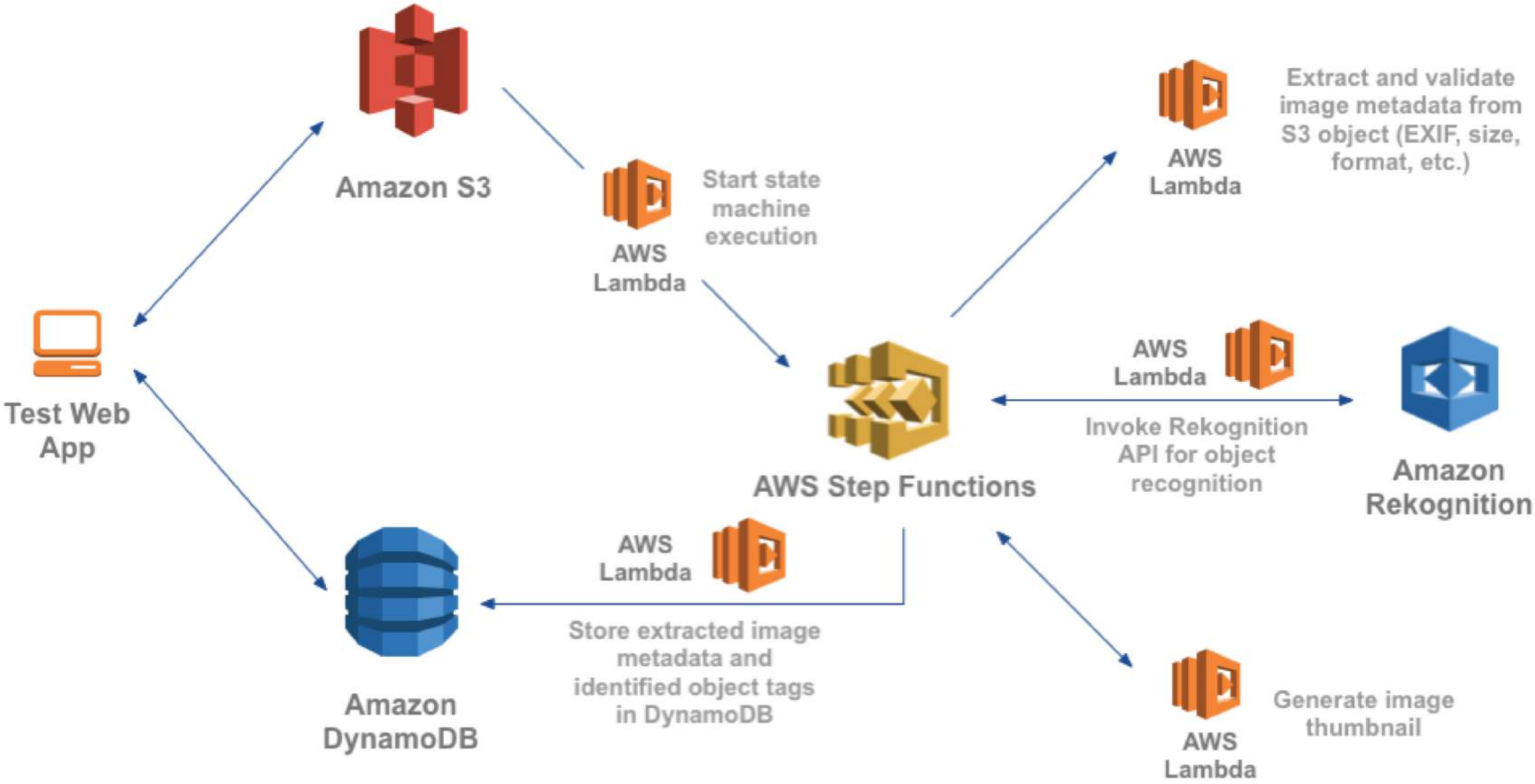
AWS Step Functions:

Orchestrate a Serverless processing
workflow using AWS Lambda

Manages state, checkpoints and restarts for you to make sure that your application executes in order and as expected

Build visual workflows that enable fast translation of business requirements





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

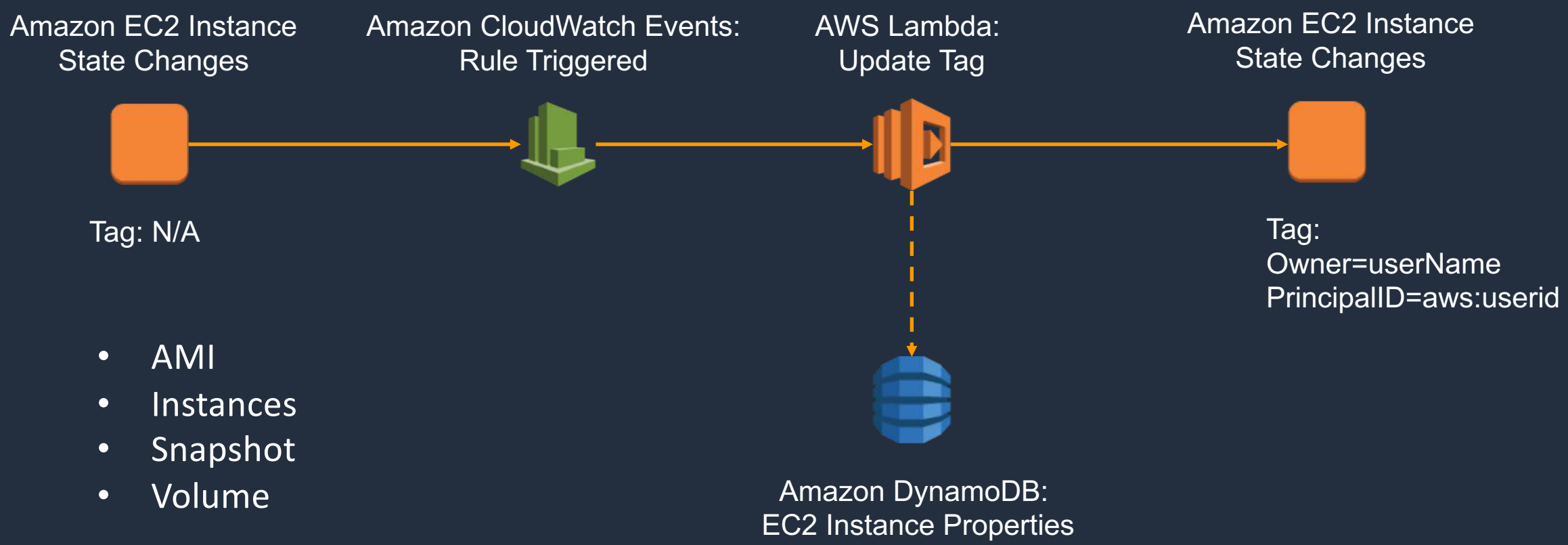
Operations Automation Architecture

Automation characteristics

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

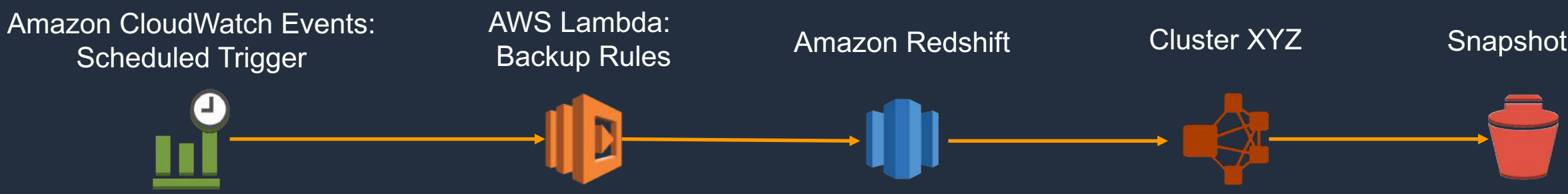
… All while being Highly Available, Scalable and Auditable

Auto tagging resources as they start

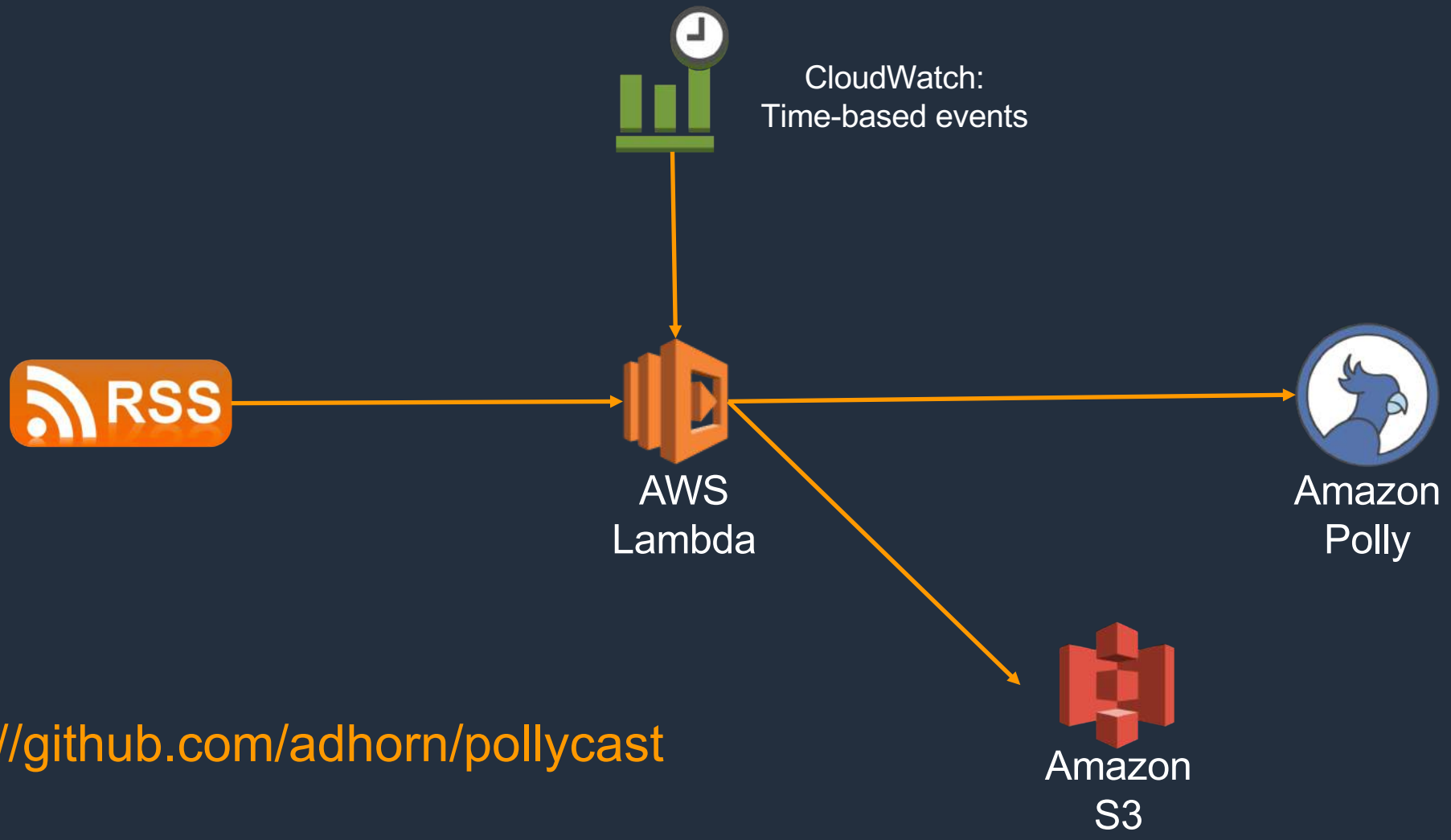


- AMI
- Instances
- Snapshot
- Volume

Scheduled backup operation

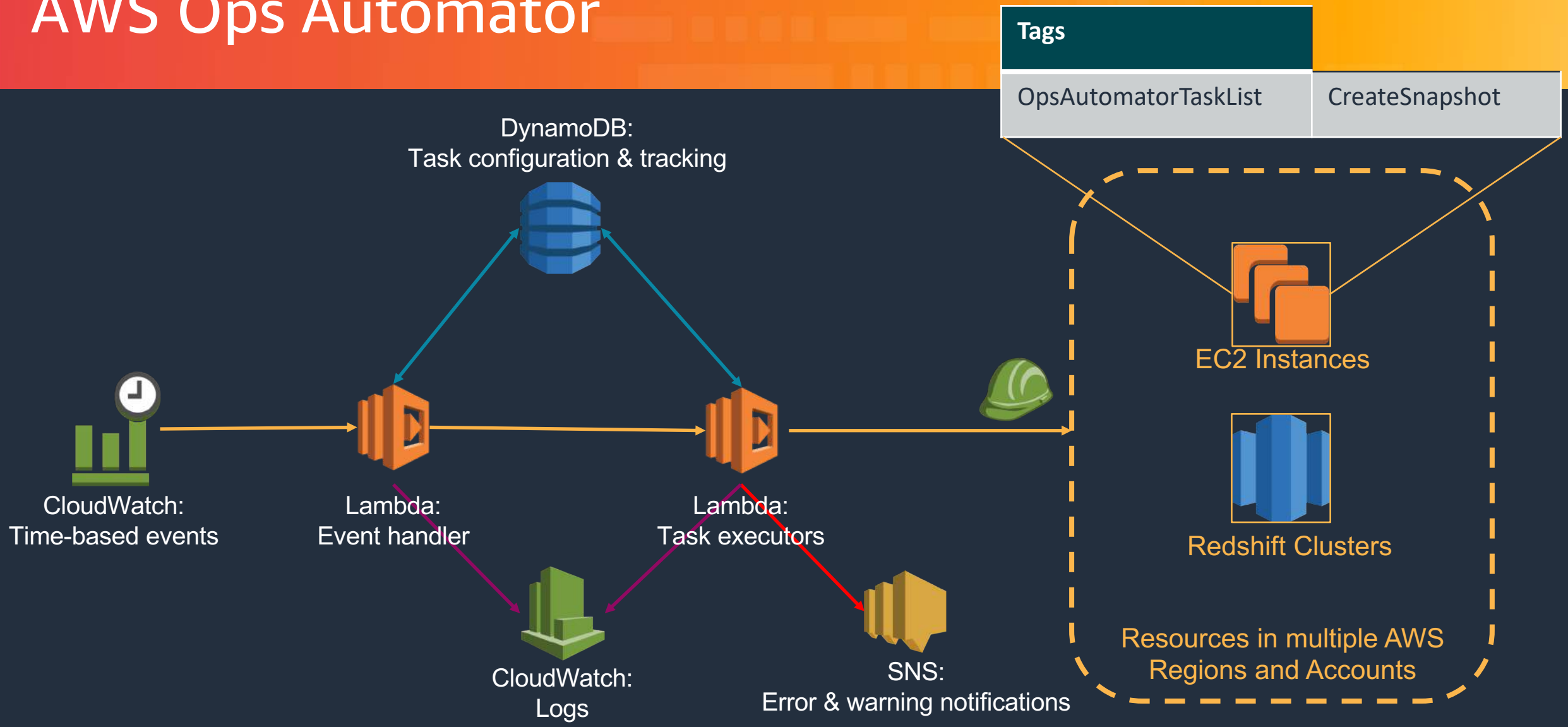


Scheduled AI jobs



https://github.com/adhorn/pollycast

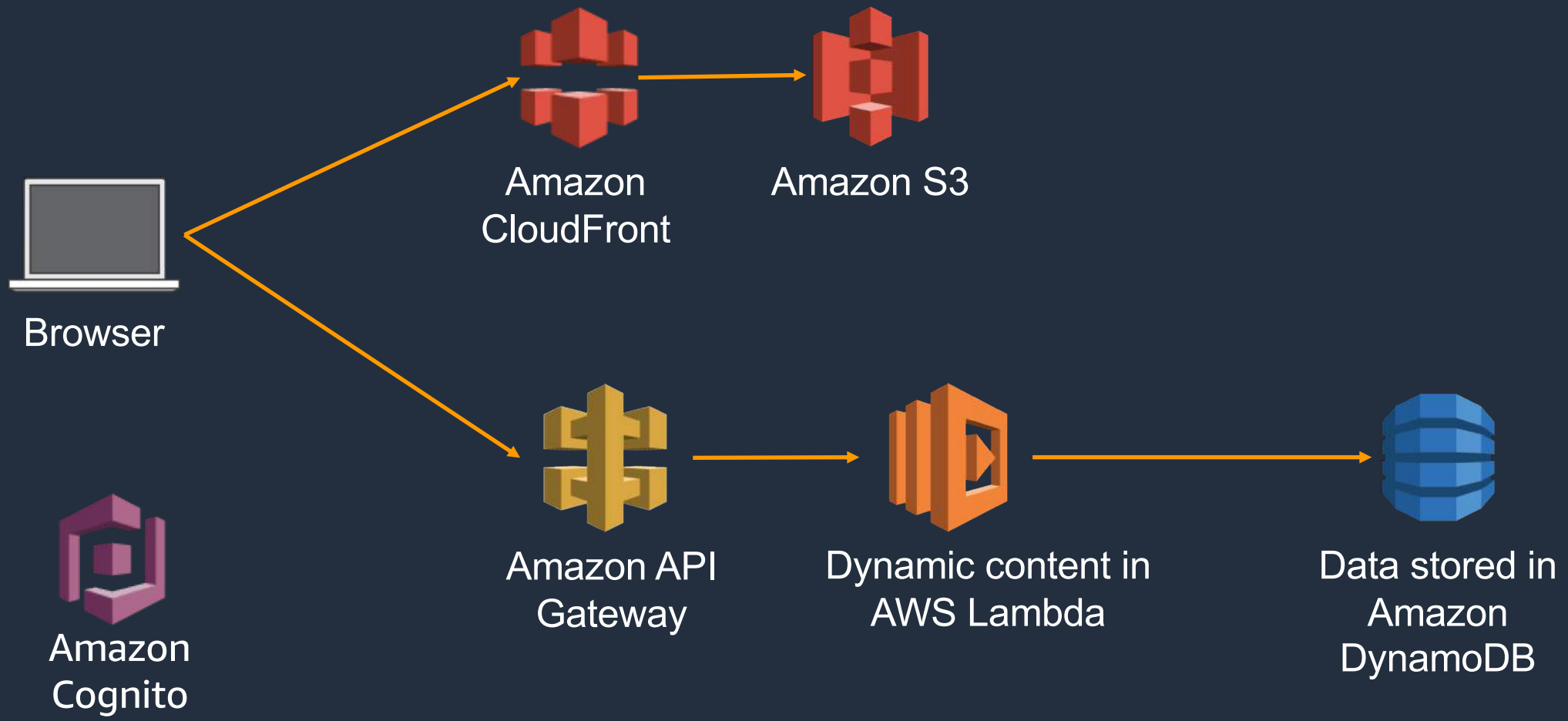
AWS Ops Automator



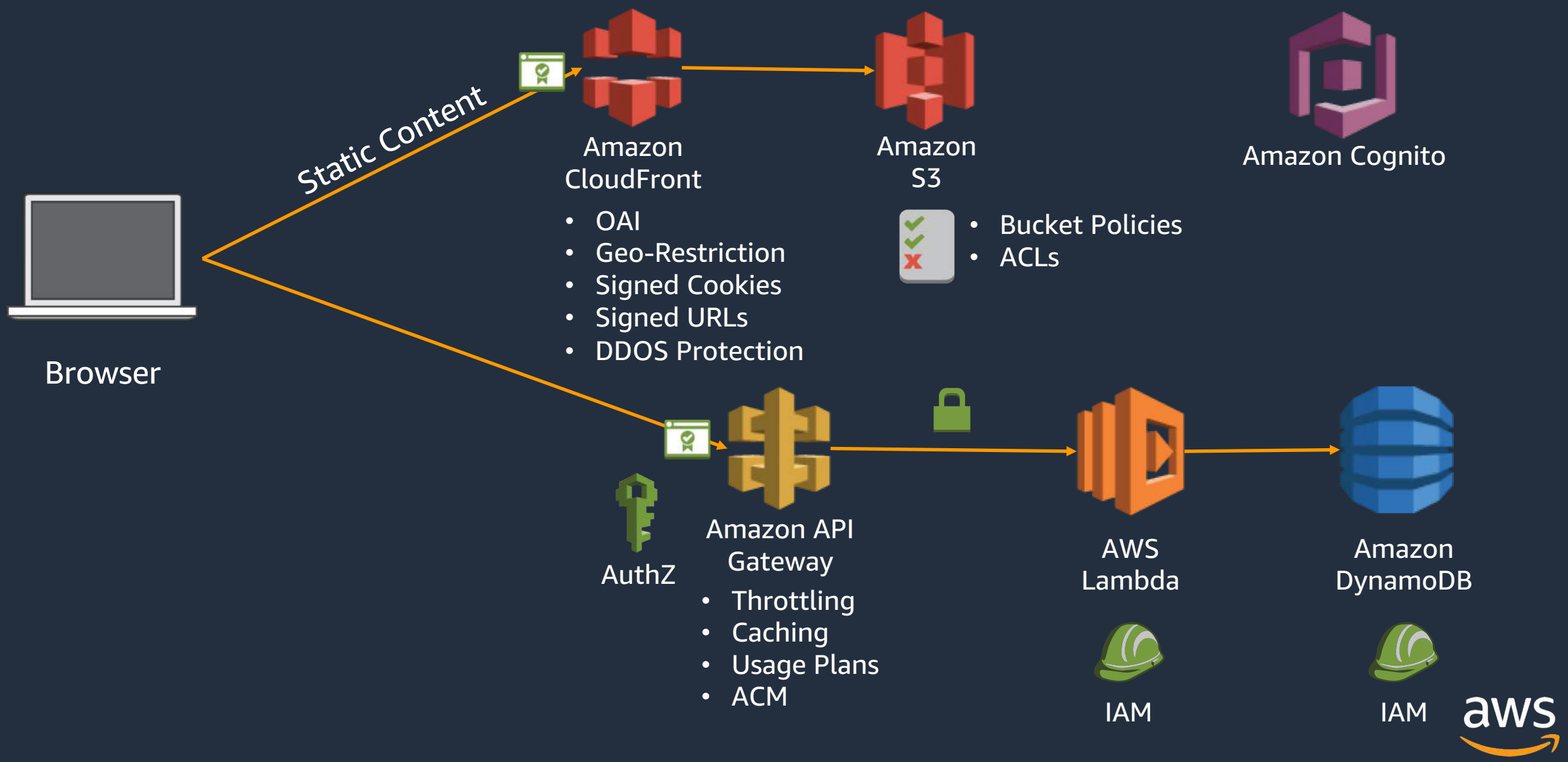
https://aws.amazon.com/answers/infrastructure-management/ops-automator/

Web Application Architecture

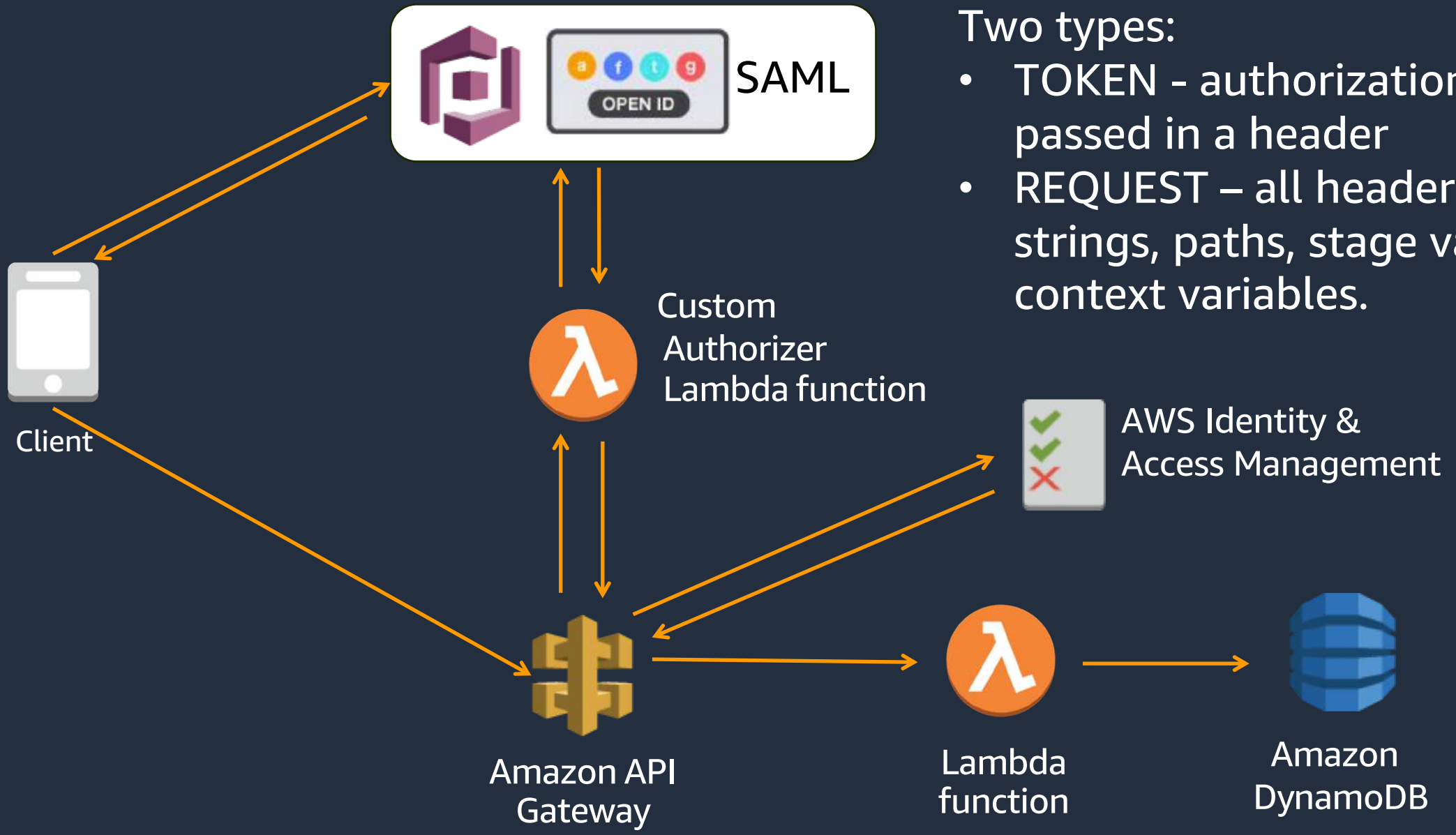
Web application



Serverless web app security



Custom Authorizers



Two types:

- TOKEN - authorization token passed in a header
- REQUEST – all headers, query strings, paths, stage variables or context variables.

Bustle Achieves 84% Cost Savings with AWS Lambda

"With AWS Lambda, we eliminate the need to worry about operations"

Tyler Love
CTO, Bustle

Bustle is a news, entertainment, lifestyle, and fashion website targeted towards women.

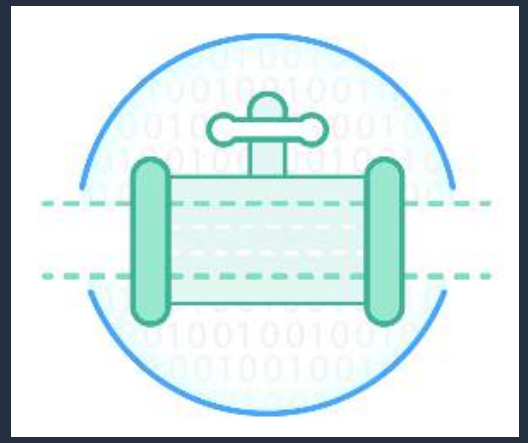
- Bustle had trouble scaling and maintaining high availability for its website without heavy management
- Moved to serverless architecture using AWS Lambda and Amazon API Gateway
- Experienced approximately 84% in cost savings
- Engineers are now focused on innovation

aws

Stream Processing Architecture

Kinesis-based apps

Amazon Kinesis makes it easy to work with real-time streaming data

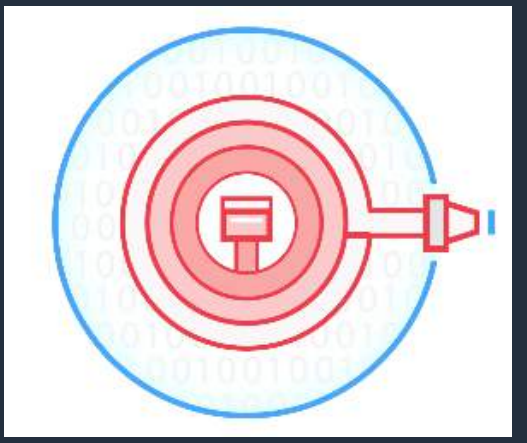


Amazon Kinesis Streams

- For Technical Developers
- Collect and stream data for ordered, replay-able, real-time processing

Amazon Kinesis Analytics

- For all developers, data scientists
- Easily analyze data streams using standard SQL queries

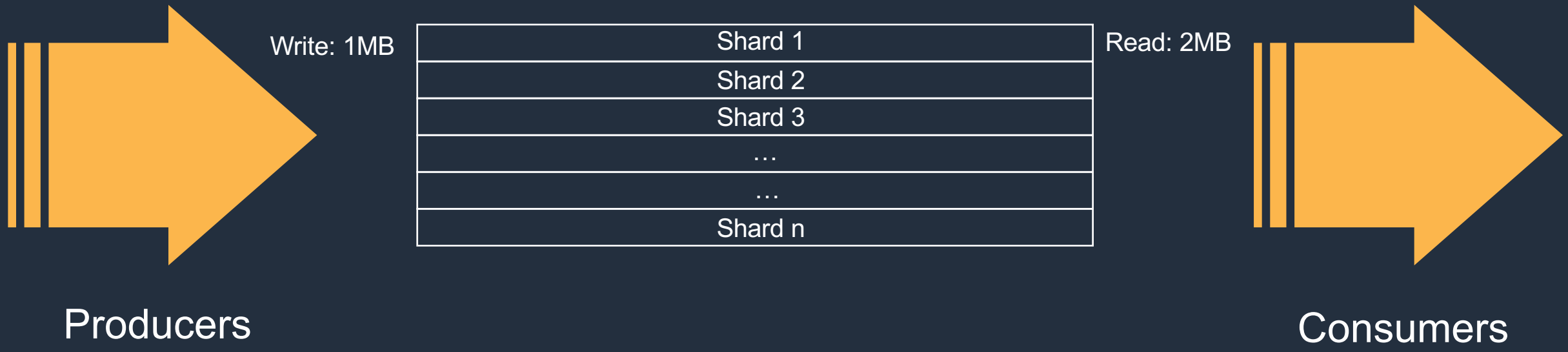


Amazon Kinesis Firehose

- For all developers, data scientists
- Easily load massive volumes of streaming data into Amazon S3, Redshift, ElasticSearch

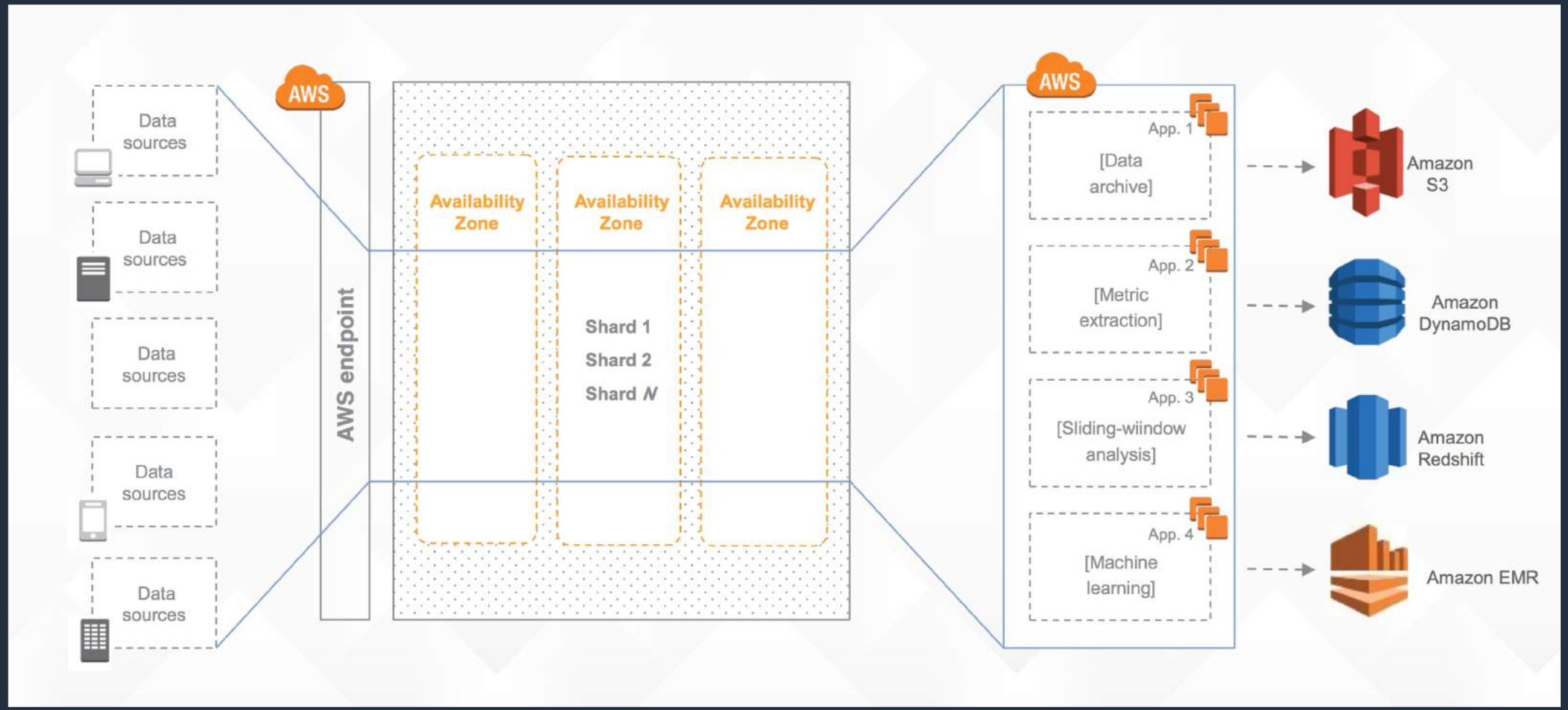
aws

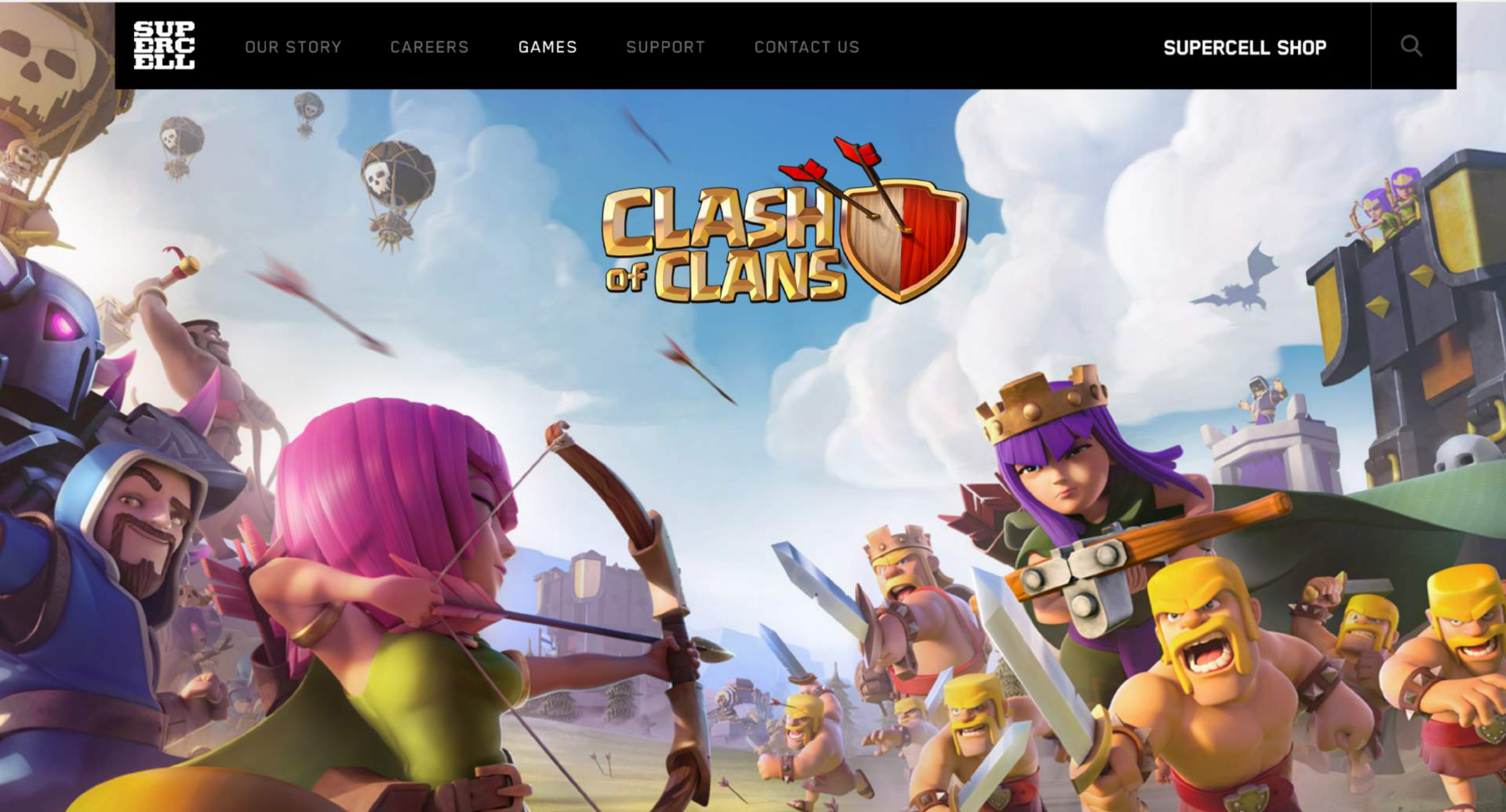
Amazon Kinesis



** A *shard* is a group of data records in a stream

Amazon Kinesis under the hood




CLASH of CLANS

Netflix & Amazon Kinesis Streams Case Study

NETFLIX

> Amazon Kinesis Streams processes multiple terabytes of log data each day, yet events show up in our analytics in seconds. We can discover and respond to issues in real time, ensuring high availability and a great customer experience.
>
> **John Bennett**
>
> Senior Software Engineer, Netflix

About Netflix

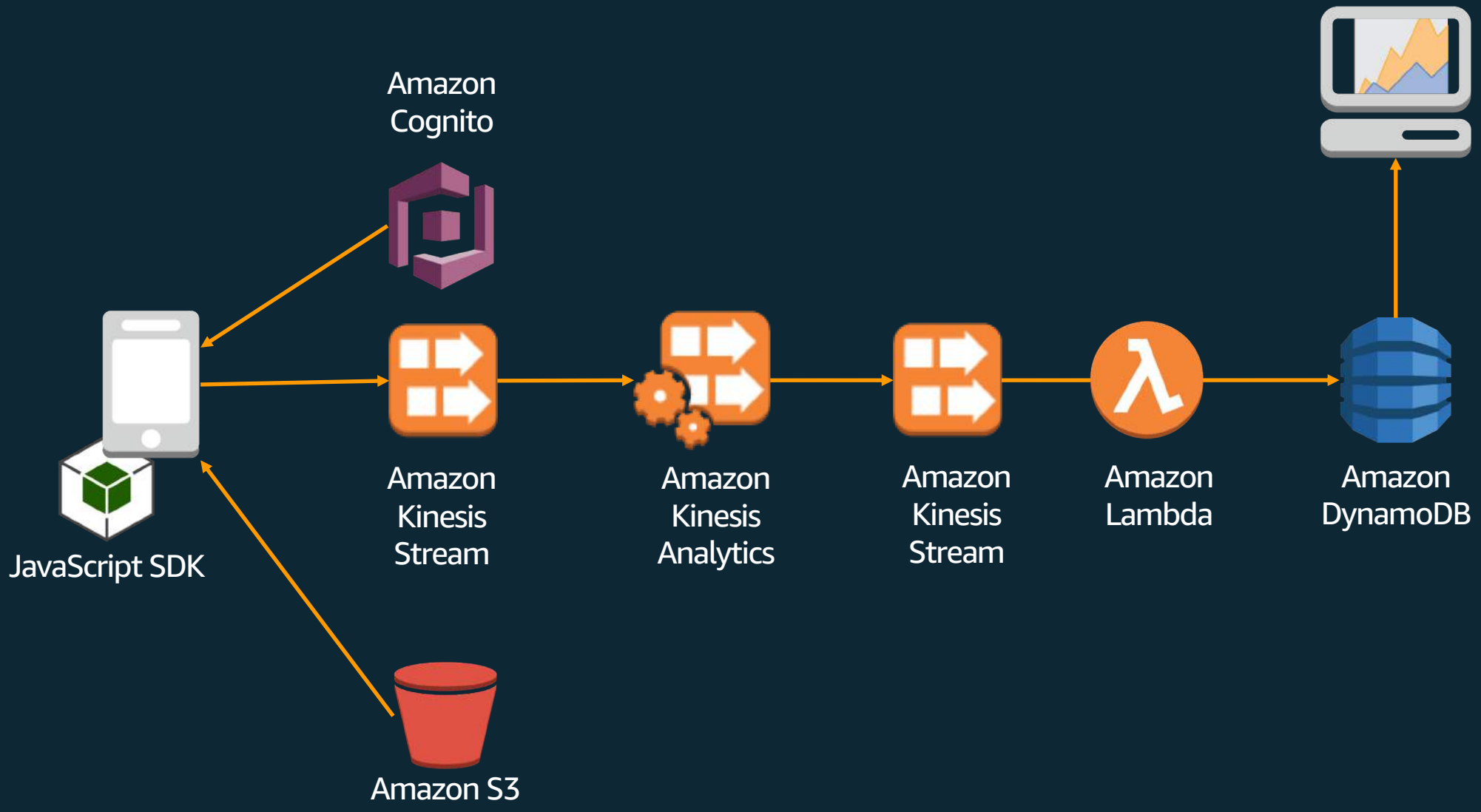
Netflix is the world's leading internet television network, with more than 100 million members worldwide enjoying 125 million hours of TV shows and movies each day, including original series, documentaries, and feature films. Members can watch as much as they want, anytime, anywhere, on nearly any Internet-connected screen.

Application Monitoring on a Massive Scale

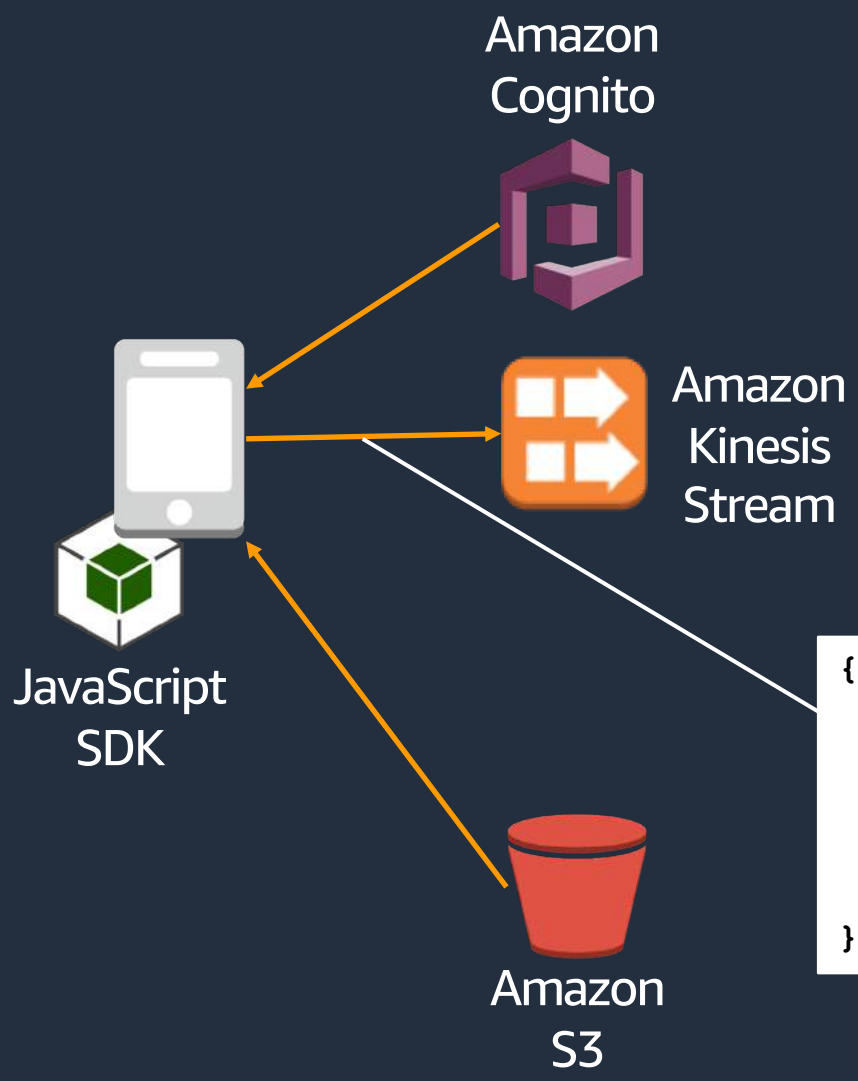
Netflix uses Amazon Web Services (AWS) for nearly all its computing and storage needs, including databases, analytics, recommendation engines, video transcoding, and more—hundreds of functions that in total use more than 100,000 server instances on AWS.

This results in an extremely complex and dynamic networking environment where applications are constantly communicating inside AWS and across the Internet. Monitoring and optimizing its network is critical for Netflix to continue improving customer experience, increasing efficiency, and reducing costs. In particular, Netflix needed a solution for ingesting, augmenting, and analyzing the multiple terabytes of data its network generates daily in the form of virtual private cloud (VPC) flow logs. This would enable Netflix to identify performance-improvement opportunities, such as identifying apps that are communicating across regions and collocating them. The company would also be able to increase uptime by quickly detecting and mitigating application downtime.

Real-time analytics



Data Input



Source JSON Data

- Once per second, using JavaScript SDK:
  - Unique Cognito ID (anonymous user)
  - OS
  - Quadrant
  - Data sent to Kinesis Stream

```
{
  "recordTime": 1486505943.204,
  "cognitoId": "us-east-1:3626e211-d2a3-447b-8231-e1f4e0486f44",
  "os": "Android",
  "quadrant": "A”,
  …
}
```

aws

How is raw data mapped to a schema?

Amazon Kinesis stream

Amazon Kinesis Analytics

```
{
  "recordTime": 1486505943.204,
  "cognitoId": "us-east-1:<guid>",
  "os": "Android",
  "quadrant": "A"
}
```

cognitoID	os	quadrant
<guid1>	Android	A
<guid2>	iOS	B

Source Data for Kinesis Analytics

aws

How is streaming data accessed with SQL?

STREAM

- Analogous to a TABLE
- Represents continuous data flow

```
CREATE OR REPLACE STREAM DISTINCT_USER_STREAM(
    COGNITO_ID VARCHAR(64),
    DEVICE VARCHAR(32),
    OS VARCHAR(32),
    QUADRANT char(1),
    DT TIMESTAMP);
```

aws

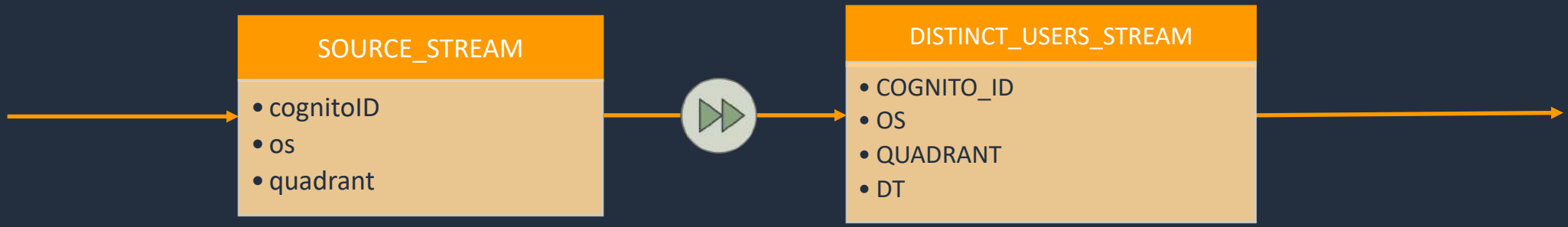
How is streaming data accessed with SQL?

PUMP

- Continuous INSERT query
- Inserts data from one in-application stream to another

```
CREATE OR REPLACE PUMP "DISTINCT_USER_PUMP" AS
    INSERT INTO "DISTINCT_USER_STREAM"
        SELECT STREAM DISTINCT
            "cognitoId",
              ...
```

How do we get distinct user records?



Use PUMP to insert distinct records into in-app STREAM

```
CREATE OR REPLACE PUMP "DISTINCT_USER_PUMP" AS
    INSERT INTO "DISTINCT_USER_STREAM"
        SELECT STREAM DISTINCT
            "cognitoId",
            "device",
            "os",
            "quadrant",
            FLOOR(s.ROWTIME TO SECOND)
        FROM "SOURCE_SQL_STREAM_001" s;
```

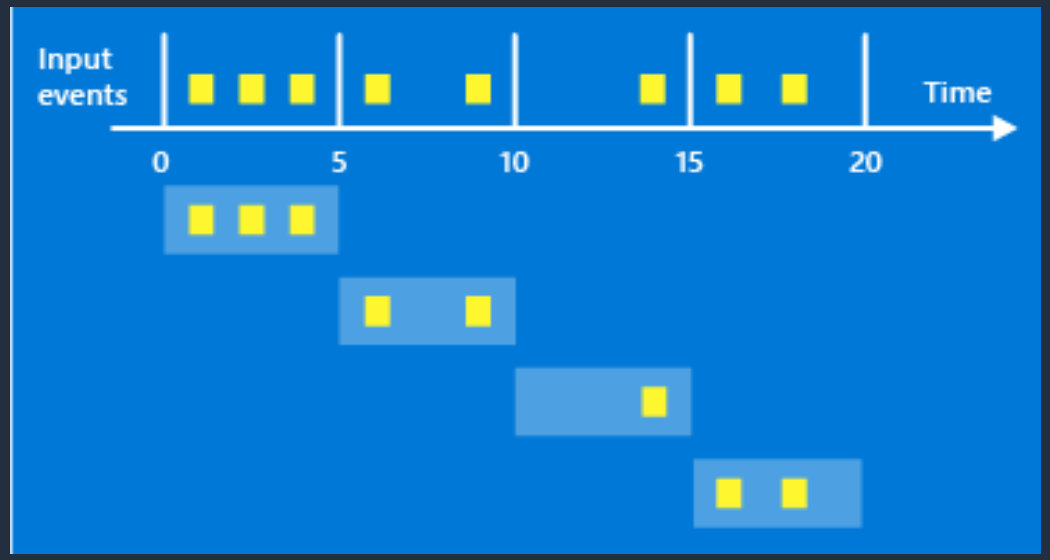
How do we aggregate per second?

- Tumbling window, group by time period

```
CREATE OR REPLACE PUMP "OUTPUT_PUMP" AS
    INSERT INTO "DESTINATION_SQL_STREAM"
        SELECT STREAM
            COUNT(dus.COGNITO_ID) AS UNIQUE_USER_COUNT,
            COUNT((CASE WHEN dus.OS = 'Android' THEN COGNITO_ID ELSE null END)) AS ANDROID_COUNT,
            COUNT((CASE WHEN dus.OS = 'iOS' THEN COGNITO_ID ELSE null END)) AS IOS_COUNT,
            COUNT((CASE WHEN dus.OS = 'Windows Phone' THEN COGNITO_ID ELSE null END)) AS WINDOWS_PHONE_COUNT,
            COUNT((CASE WHEN dus.OS = 'other' THEN COGNITO_ID ELSE null END)) AS OTHER_OS_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'A' THEN COGNITO_ID ELSE null END)) AS QUADRANT_A_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'B' THEN COGNITO_ID ELSE null END)) AS QUADRANT_B_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'C' THEN COGNITO_ID ELSE null END)) AS QUADRANT_C_COUNT,
            COUNT((CASE WHEN dus.QUADRANT = 'D' THEN COGNITO_ID ELSE null END)) AS QUADRANT_D_COUNT,
            ROWTIME
        FROM "DISTINCT_USER_STREAM" dus
        GROUP BY
            FLOOR(dus.ROWTIME TO SECOND);
```

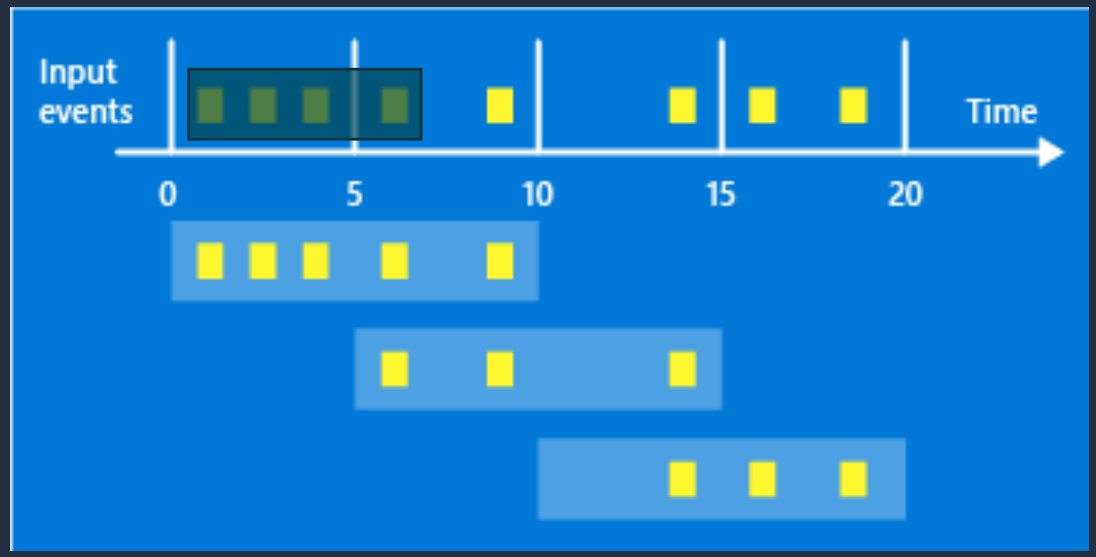
Comparing Types of Windows

- Output created at the end of the window
- The output of the window will be single event based on the aggregate function used



Tumbling window

Aggregate per time interval

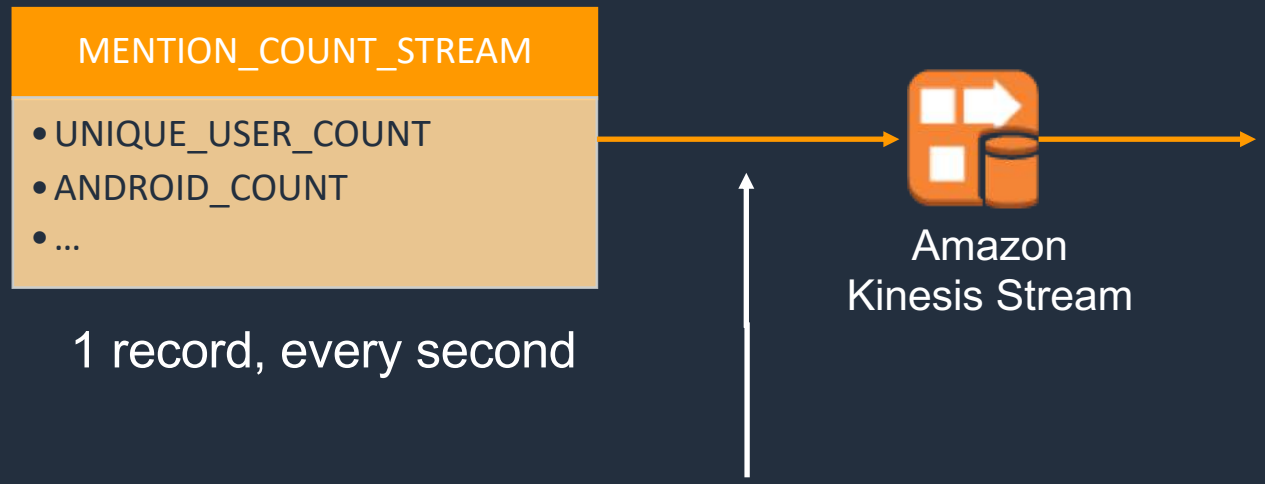


Sliding window

Windows constantly re-evaluated

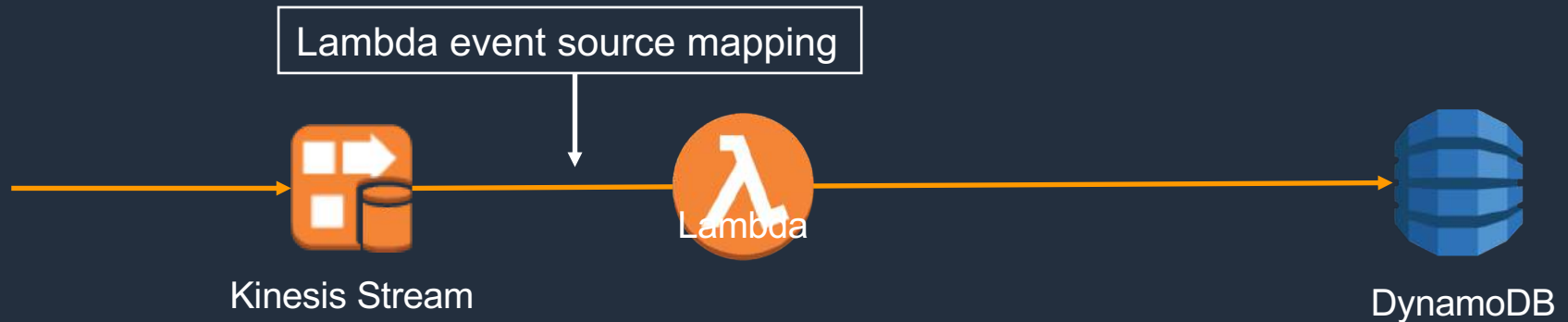
aws

Output to Kinesis Stream



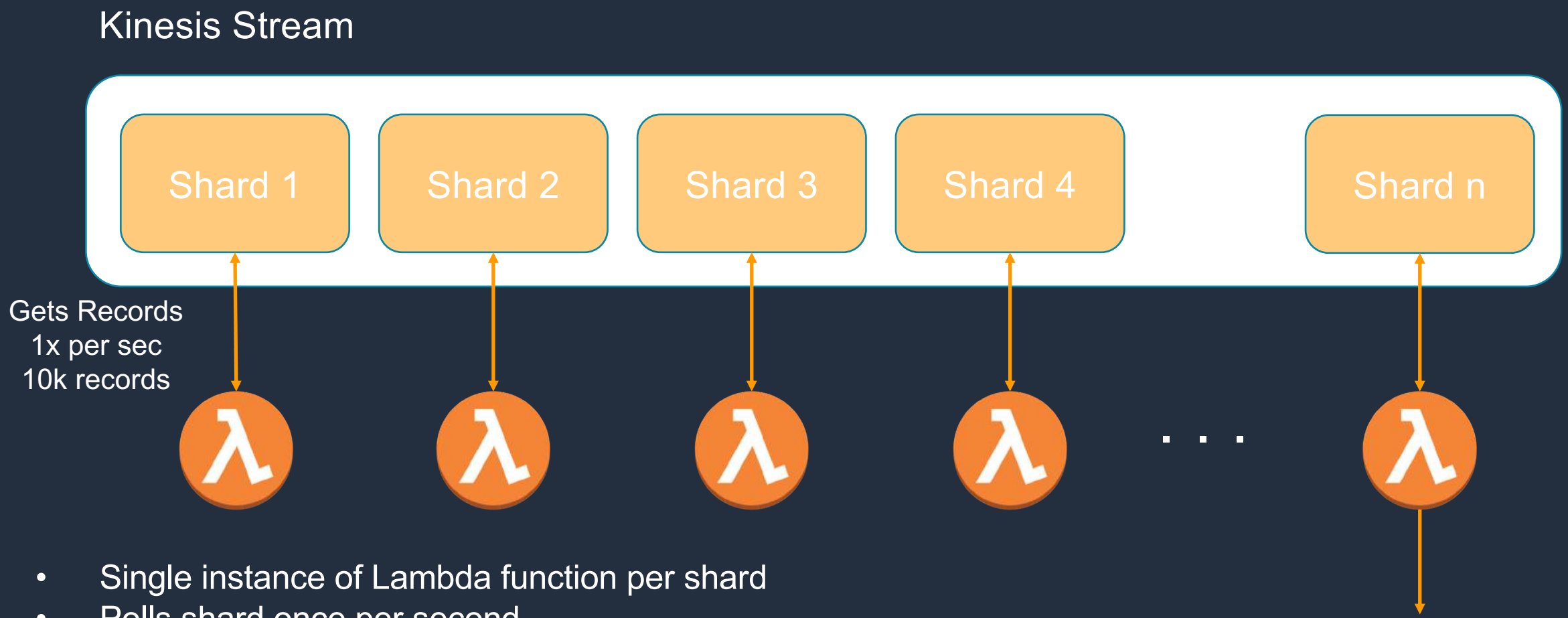

```
{
      "unique_user_count": 96,
      "android_count": 50,
      "ios_count": 46,
      "android_count": 50,
      "quadrant_a_count": 80,
      "quadrant_b_count ": 10,
      "quadrant_c_count ": 3,
      "quadrant_d_count ": 3
}
```

Persist aggregated data in DynamoDB




```
event.Records.forEach((record) => {
      const payload = new Buffer(record.kinesis.data, 'base64').toString('ascii');
      var docClient = new AWS.DynamoDB.DocumentClient();
      var table = "user-quadrant-data";
      var data = JSON.parse(payload);
      var params = {
          TableName: table,
          Item:{
              "dataType": "quadrantRollup",
              "windowtime": (new Date(data.WINDOW_TIME)).getTime(),
              "userCount": data.UNIQUE_USER_COUNT,
              "quadrantA": data.QUADRANT_A_COUNT,
              "quadrantB": data.QUADRANT_B_COUNT, ...
          }
      };
     docClient.put(params, function(err, data) {   ...
```

Processing a Kinesis Streams with AWS Lambda

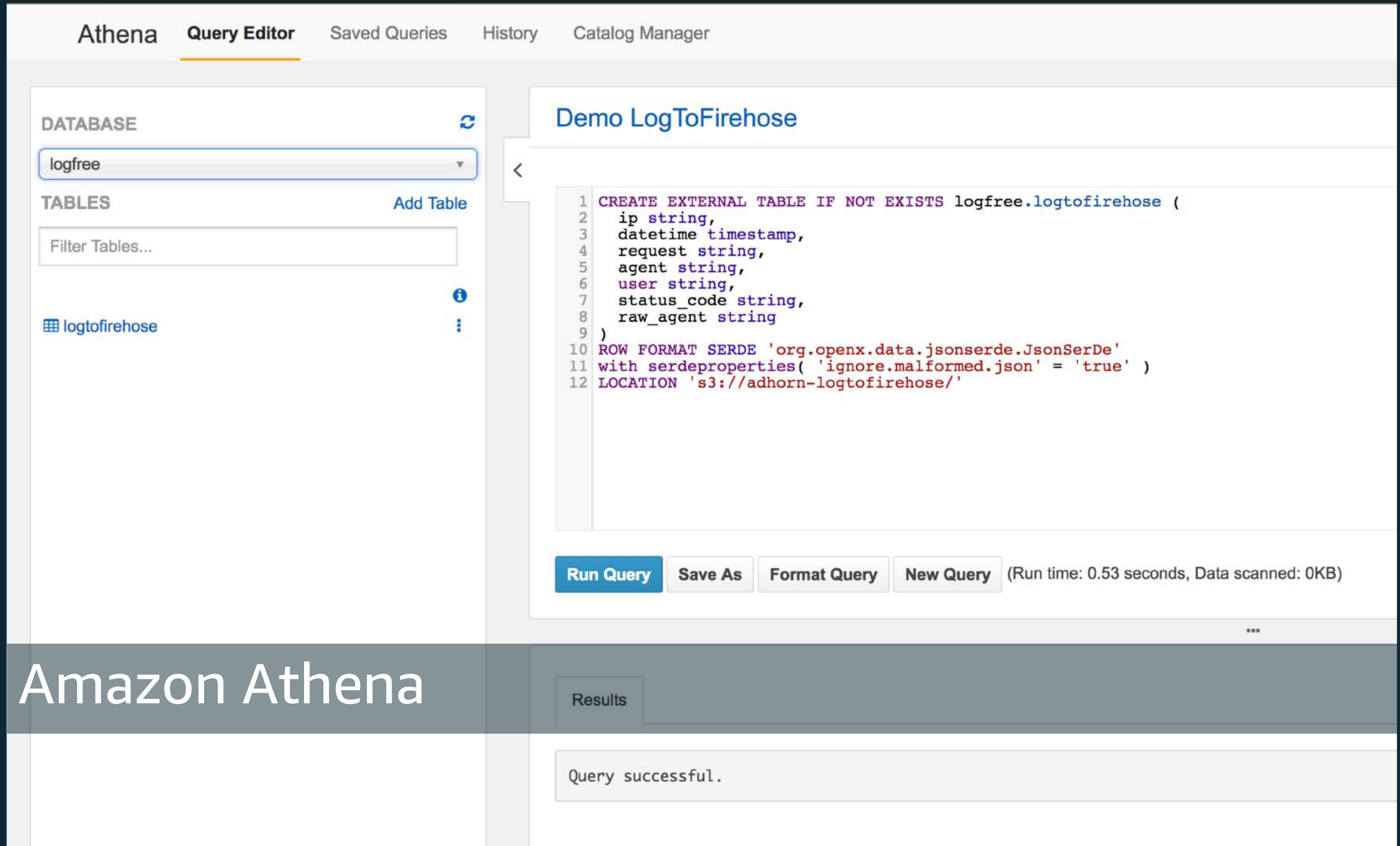


- Single instance of Lambda function per shard
- Polls shard once per second
- Lambda function instances created and removed automatically as stream is scaled

aws

Logs Analytics

https://github.com/adhorn/logtoes


Athena
Query Editor
Saved Queries
History
Catalog Manager
DATABASE
logfree
TABLES
Add Table
Filter Tables...
logtofirehose
Demo LogToFirehose
1 CREATE EXTERNAL TABLE IF NOT EXISTS logfree.logtofirehose (
2 ip string,
3 datetime timestamp,
4 request string,
5 agent string,
6 user string,
7 status_code string,
8 raw_agent string
9)
10 ROW FORMAT SERDE 'org.openx.data.jsonserde.JsonSerDe'
11 with serdeproperties('ignore.malformed.json' = 'true')
12 LOCATION 's3://adhorn-logtofirehose/'
Run Query
Save As
Format Query
New Query
(Run time: 0.53 seconds, Data scanned: 0KB)
Amazon Athena
Results
Query successful.

Create Demo LogToFirehose

```
1 SELECT * FROM logfree."logtofirehose" order by datetime desc limit 100;
```

Use Ctrl + Enter to run query, Ctrl + Space to autocomple

Run Query | Save As | Format Query | New Query (Run time: 1.11 seconds, Data scanned: 20.24KB) ...

Results

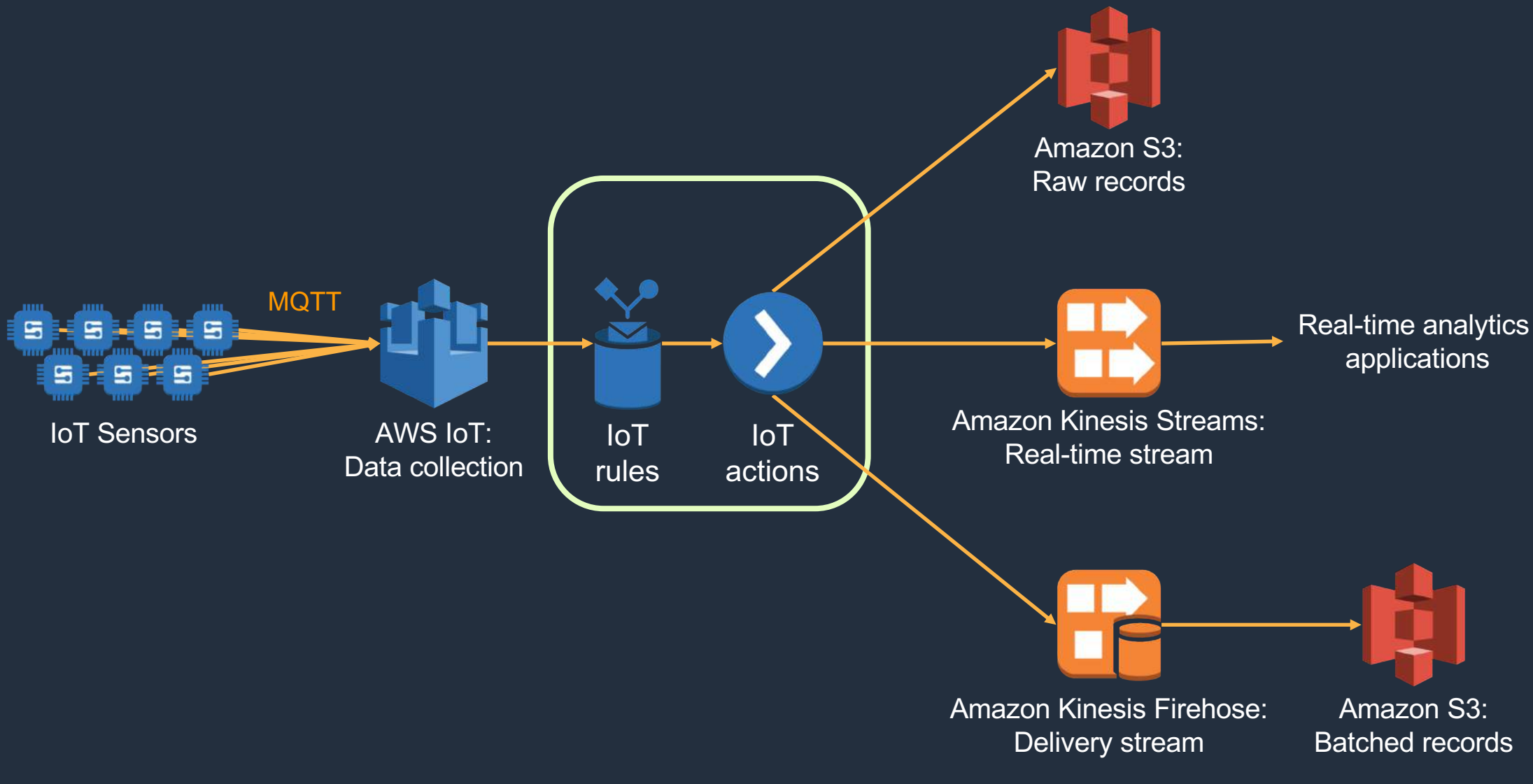
	ip	datetime	request	agent	user	status_code	raw_agent
1	127.0.0.1	2017-09-13 03:07:16.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
2	127.0.0.1	2017-09-13 03:07:14.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
3	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
4	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
5	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
6	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
7	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
8	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
9	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
10	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
11	127.0.0.1	2017-09-13 03:07:13.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
12	127.0.0.1	2017-09-13 03:07:12.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
13	127.0.0.1	2017-09-13 03:07:12.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36
14	127.0.0.1	2017-09-13 03:06:45.000	GET /api/echo	macos | chrome 60.0.3112.113	guest	200	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/60.0.3112.113 Safari/537.36

Amazon Athena

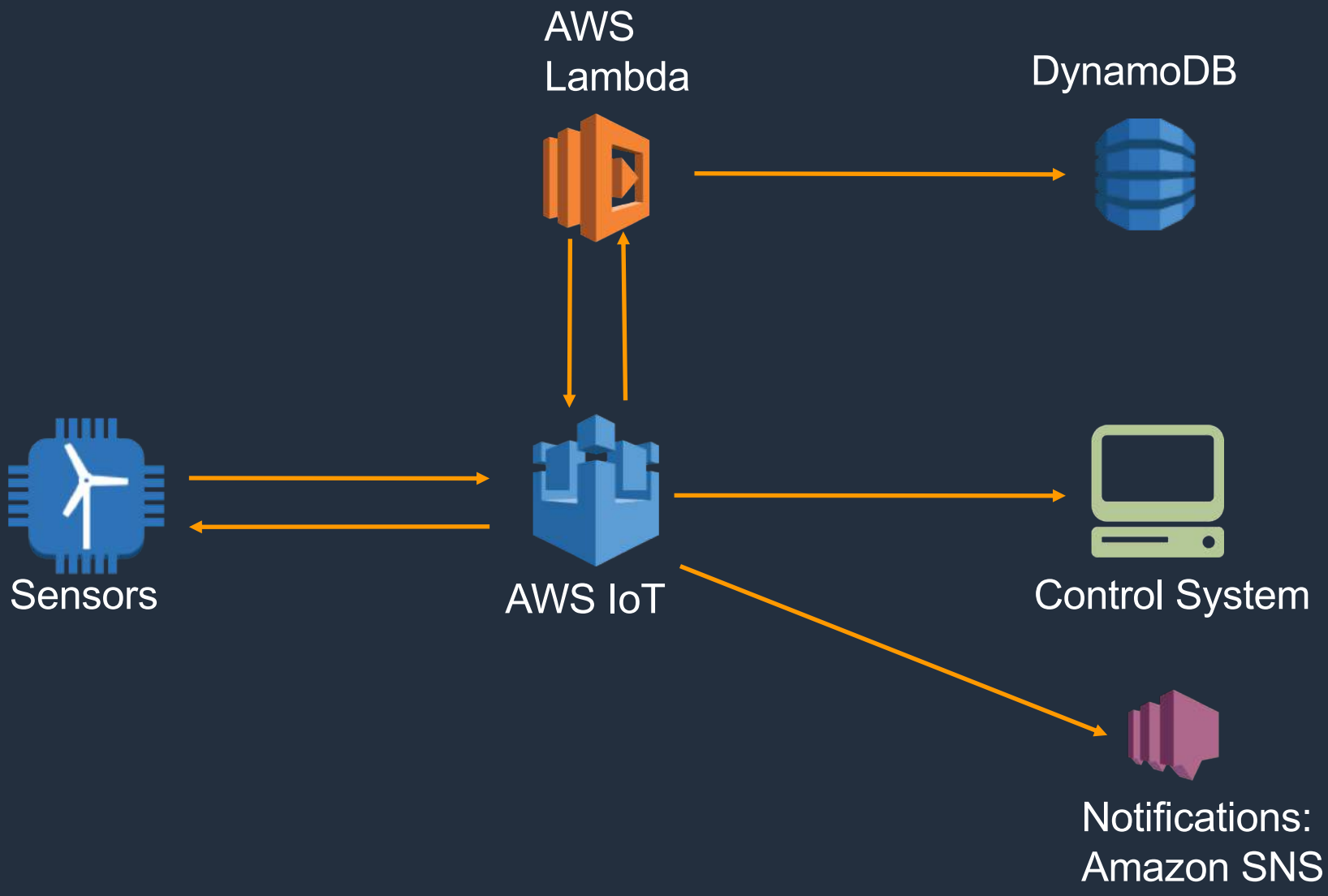
Telemetry Processing Architecture

IoT-based apps

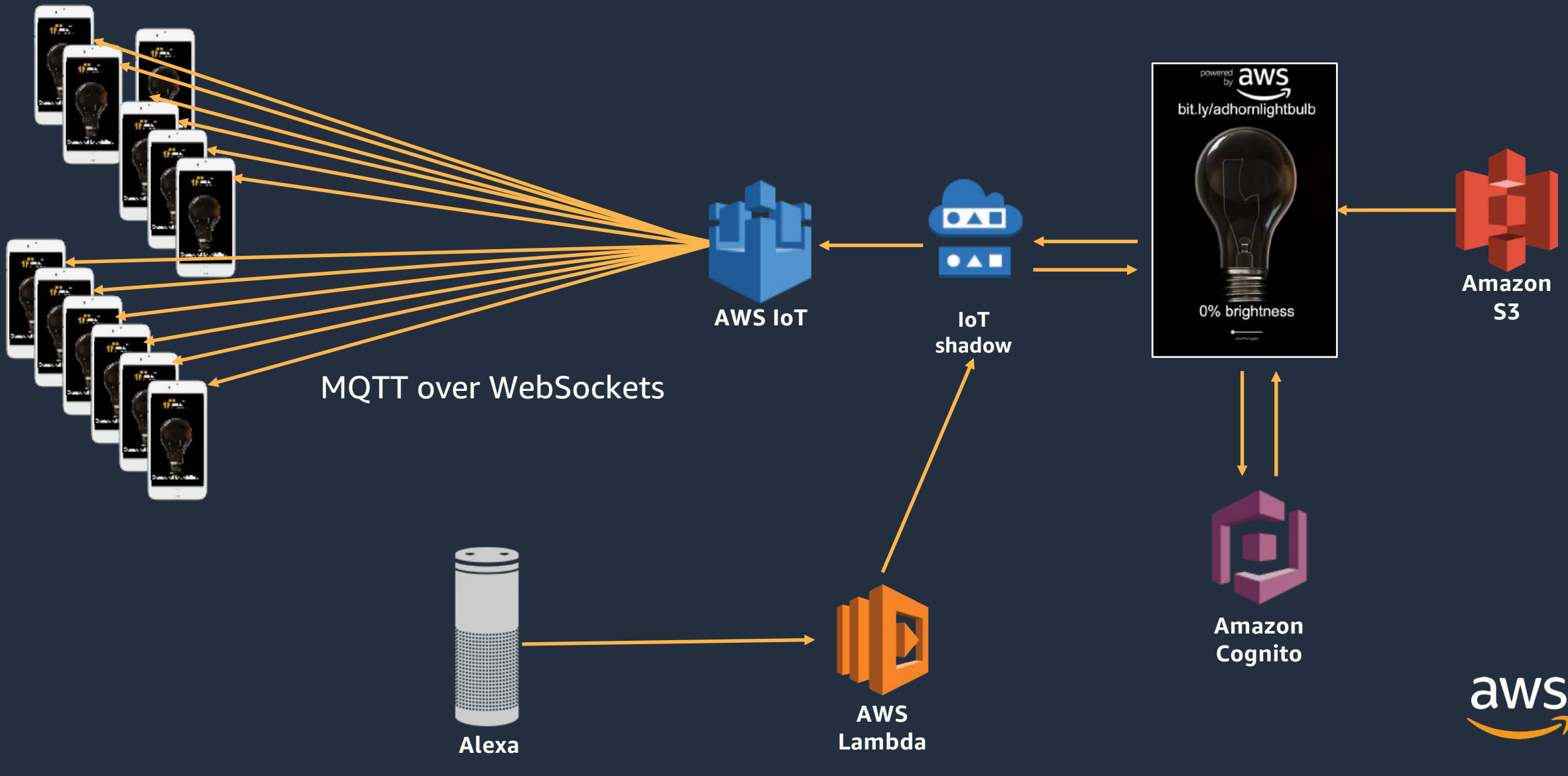
Sensor data collection



Anomaly Detection Using AWS Lambda



Interactive Serverless applications



Further Reading

Optimizing Enterprise Economics with Serverless Architectures
https://d0.awsstatic.com/whitepapers/optimizing-enterprise-economics-serverless-architectures.pdf

Serverless Architectures with AWS Lambda
https://d1.awsstatic.com/whitepapers/serverless-architectures-with-aws-lambda.pdf

Serverless Applications Lens - AWS Well-Architected Framework
https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf

Streaming Data Solutions on AWS with Amazon Kinesis
https://d1.awsstatic.com/whitepapers/whitepaper-streaming-data-solutions-on-aws-with-amazon-kinesis.pdf

AWS Serverless Multi-Tier Architectures
https://d1.awsstatic.com/whitepapers/AWS_Serverless_Multi-Tier_Archiectures.pdf

Key Takeaways

Event Processing Architecture
Operations Automation Architecture
Web Application Architecture
Stream Processing Architecture
Telemetry Processing Architecture

INNOVATE2018

ONLINE CONFERENCE

DEVELOPER EDITION

Thank you!

@awscloud

#AWSInnovate