

ARC401

# AWS re:INVENT

## Serverless Architectural Patterns and Best Practices

Drew Dennis, AWS Solution Architect  
Maitreya Ranganath, AWS Solution Architect

November 28, 2017

AWS  
re:Invent

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

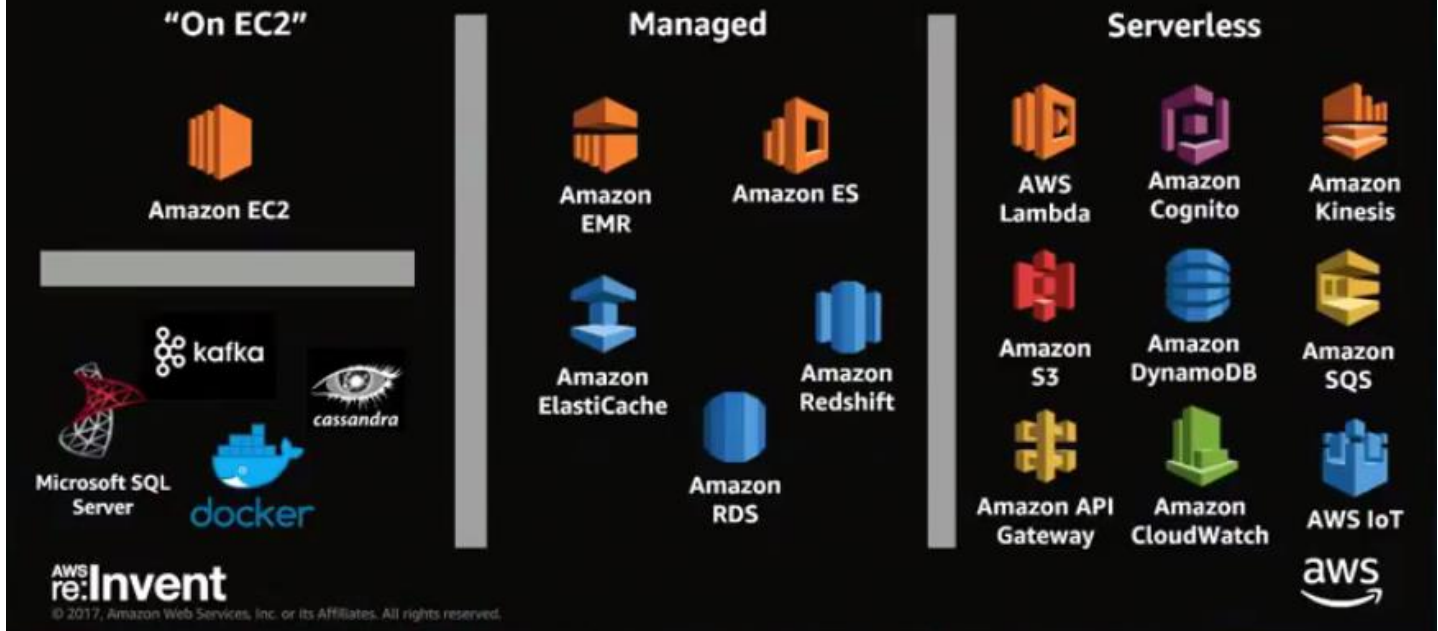


## Agenda

- Serverless Foundations
- Web application
- Data Lake
- Stream processing
- Operations automation

We are going to be focusing on 4 primary Serverless patterns. A **web application pattern** also applicable to mobile backends, microservices, or API deployments. A **data lake pattern** with sub-patterns related to cataloguing and analytical processing of your data in the data lake. We also have the **stream processing pattern** and the **operations automation pattern**.

# Spectrum of AWS offerings



The Managed services still requires that you use servers, right-size and scale them accordingly.

## Serverless means...

- No servers to provision or manage
- Scales with usage
- Never pay for idle
- Built-in High-Availability and Disaster Recovery



AWS re:Invent

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

aws

There are 4 common tenets for serverless applications, you pay for what you use only. Lambda functions are now the unit of scaling for serverless apps

# Lambda considerations and best practices

## Can your Lambda functions survive the cold?

- Instantiate AWS clients and database clients outside the scope of the handler to take advantage of container re-use.
- Schedule with CloudWatch Events for warmth
- ENIs for VPC support are attached during cold start

Executes during cold start

```
import sys
import logging
import rds_config
import pymysql

rds_host = "rds-instance"
db_name = rds_config.db_name
try:
    conn = pymysql.connect(
except:
    logger.error("ERROR:
def handler(event, context):
    with conn.cursor() as cur:
```

Executes with each invocation

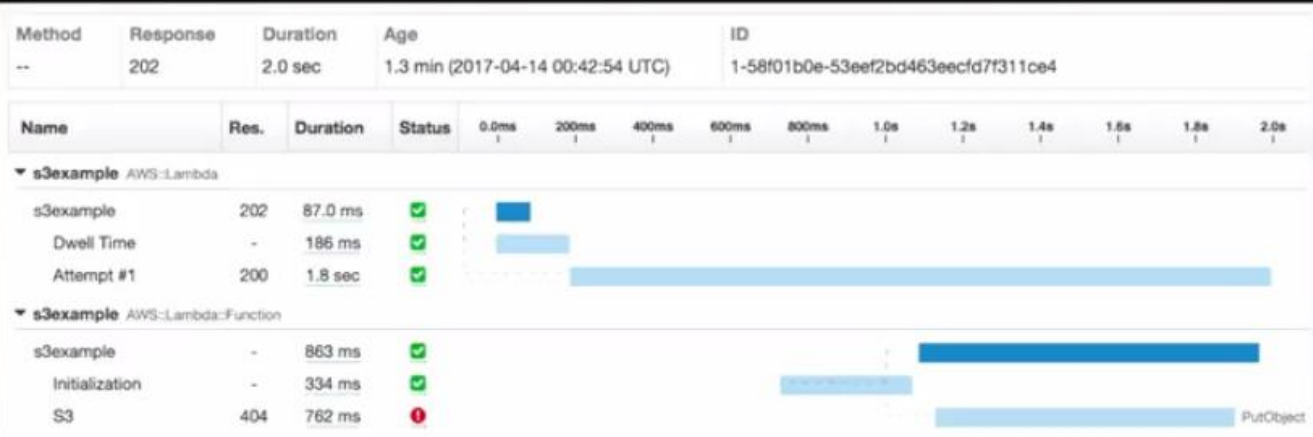
**AWS re:Invent**

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



## 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 Serverless Application Model (SAM)

- CloudFormation extension optimized for serverless
- New serverless resource types: functions, APIs, and tables
- Supports anything CloudFormation supports
- Open specification (Apache 2.0)



<https://github.com/awslabs/serverless-application-model>

**AWS re:Invent**

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



## SAM Local

- Develop and test Lambda locally
- Invoke functions with mock serverless events
- Local template validation
- Local API Gateway with hot reloading

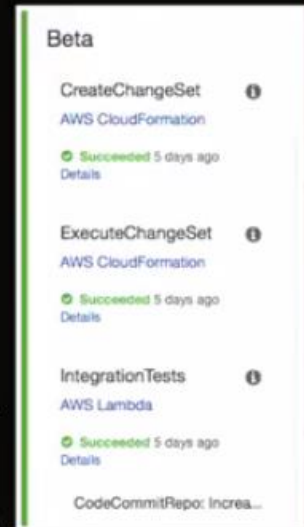


<https://github.com/awslabs/aws-sam-local>

# Delivery via CodePipeline

## Pipeline flow:

1. Commit your code to a source code repository
2. Package/test in CodeBuild
3. Use CloudFormation actions in CodePipeline to create or update stacks via SAM templates  
**Optional:** Make use of ChangeSets
4. Make use of specific stage/environment parameter files to pass in Lambda variables
5. Test our application between stages/environments  
**Optional:** Make use of manual approvals



**AWS re:Invent**

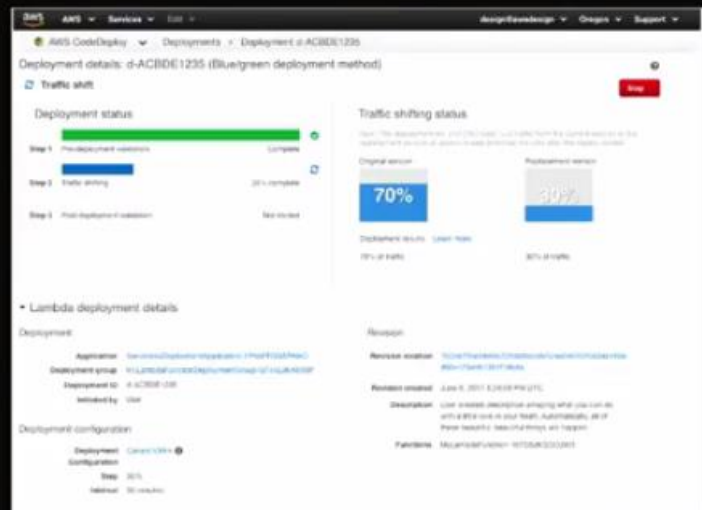
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This is our CI service

# AWS CodeDeploy and Lambda Canary Deployments

- Direct a portion of traffic to a new version
- Monitor stability with CloudWatch
- Initiate rollback if needed
- Incorporate into your SAM templates



**AWS re:Invent**

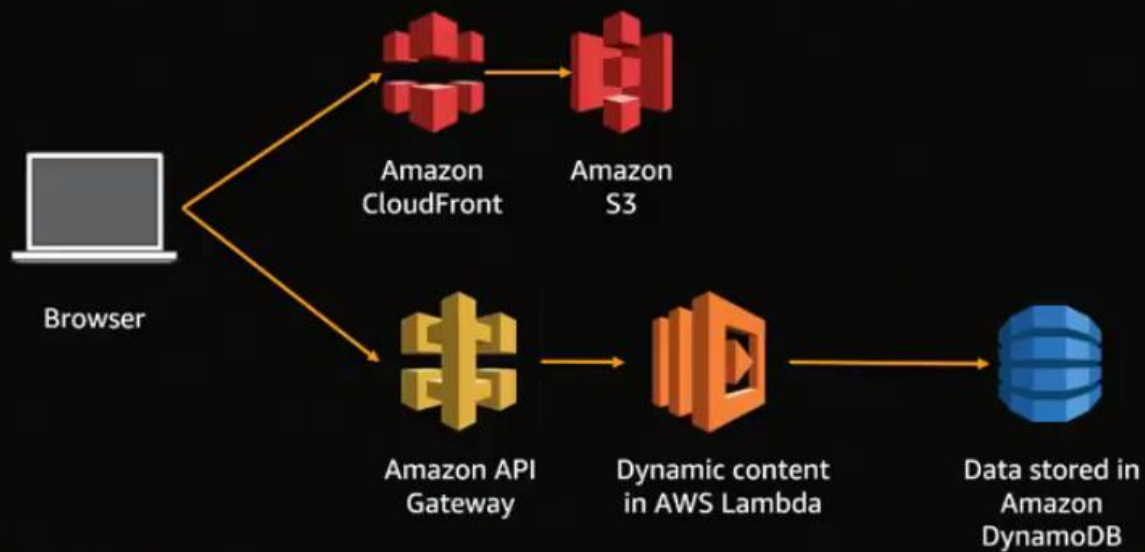
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



This allows a lambda alias mapped to multiple lambda versions using your SAM templates with canary deployments

# Pattern 1: Web App/Microservice/API

## Web application

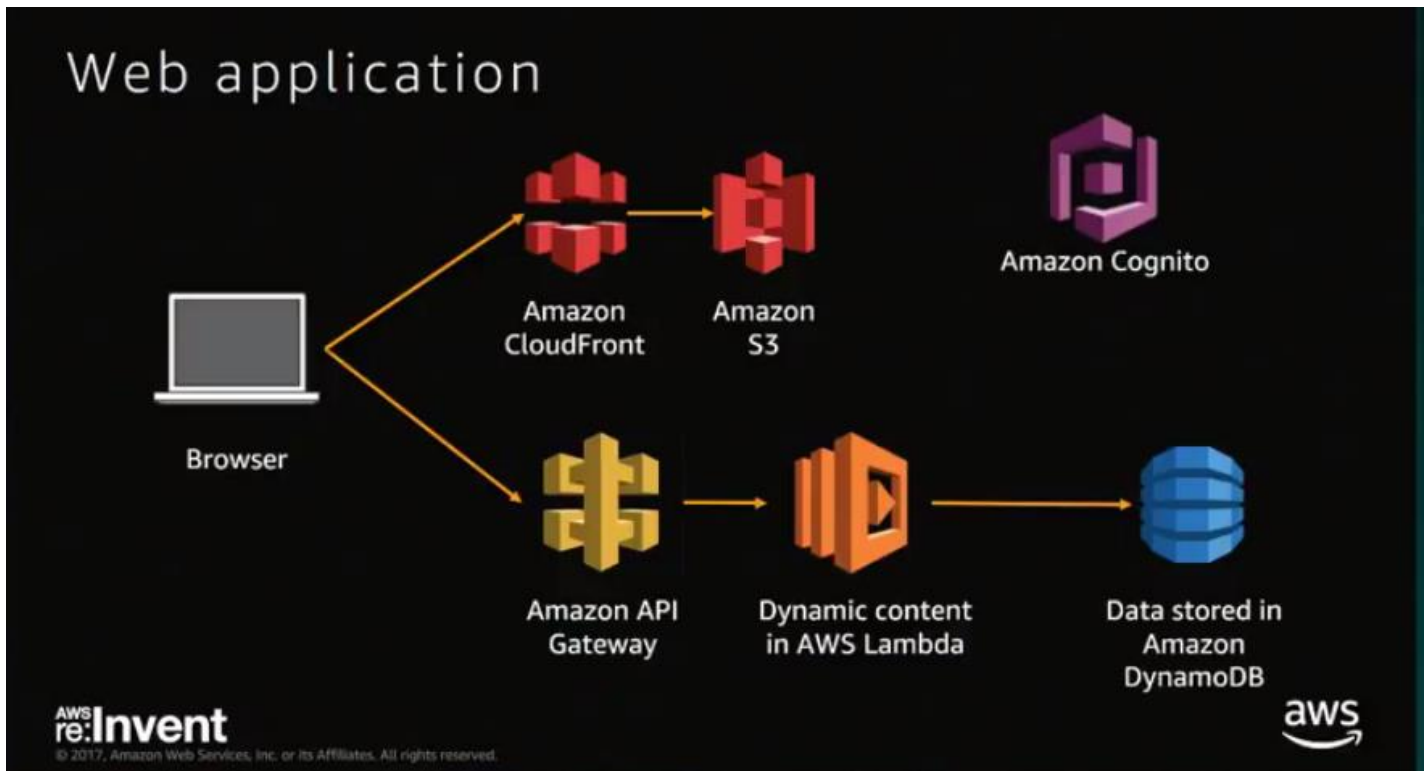


AWS  
re:Invent

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

aws

This is the **common pattern for static content web apps**, static resources in **S3** that are delivered using a content delivery service like **CloudFront**. At the bottom, we have dynamic calls and contents going through the **API Gateway** and **lambdas** that calls other downstream services or databases like **DynamoDB**.




It is important not to leave out the **AWS Cognito** service in this pattern to provide Sign Up and Sign In as well as identity **federation** both internally and across several **web identity providers**. This pattern is well adopted and used across a lot of websites today like Bustle described below

## Bustle Achieves 84% Cost Savings with AWS Lambda

**“**

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

Tyler Love  
CTO, Bustle



**”**

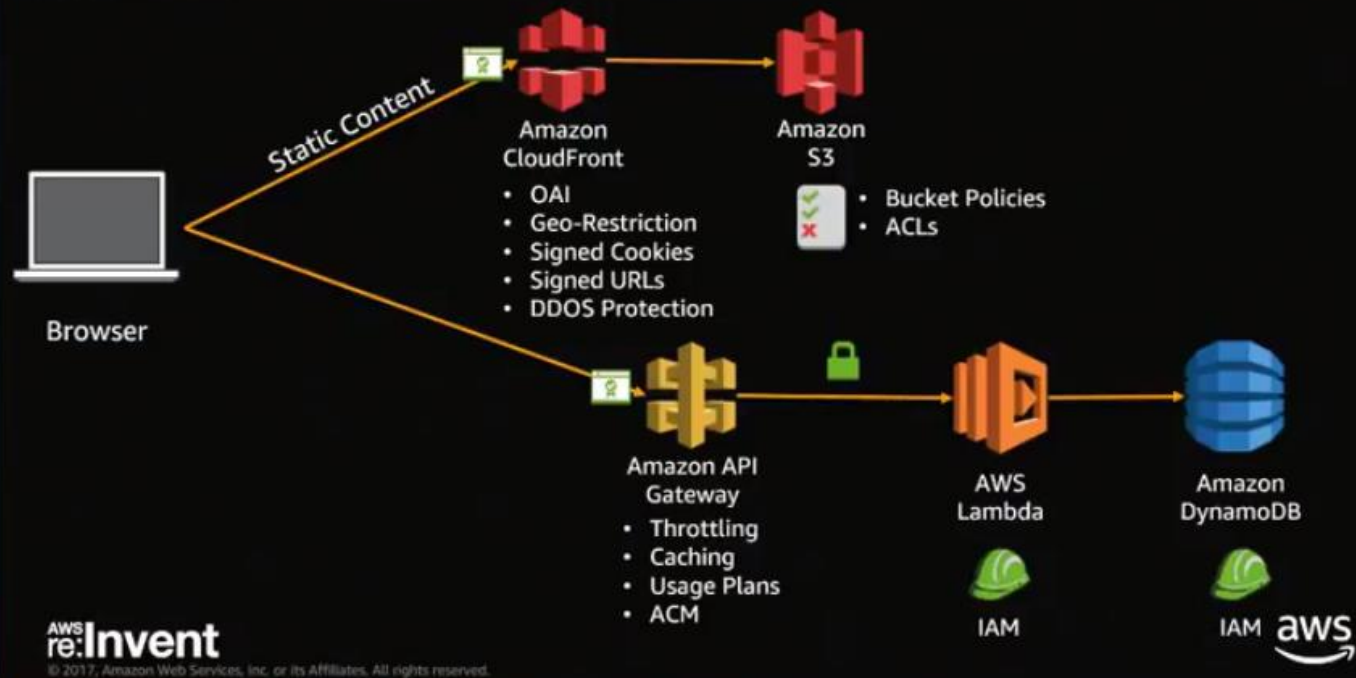
Bustle is a news, entertainment, lifestyle, and fashion website targeted towards women.

- 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 re:Invent**  
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.



# Serverless web app security

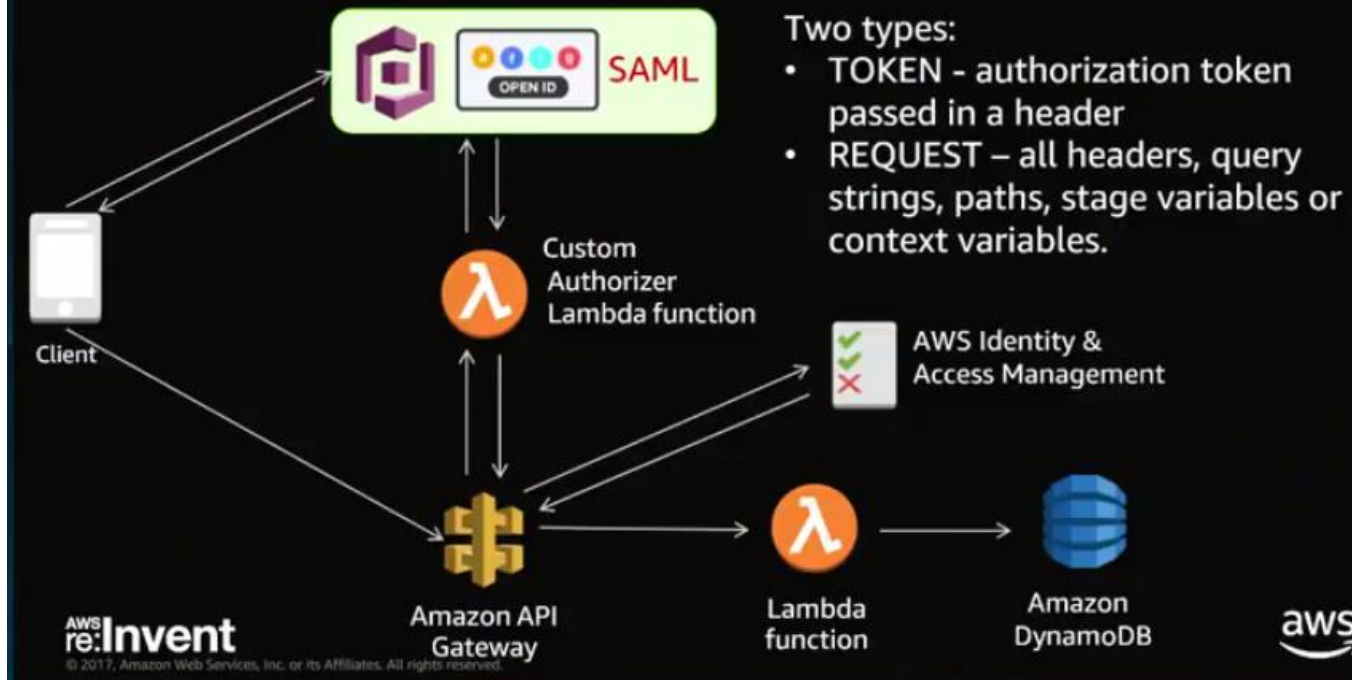


Serverless apps security is a little different from the traditional approach. There is an origin-access-identity capability in CloudFront that ensures that only CloudFront can access the assets/resources stored in your S3 bucket, both CloudFront and API Gateway now support AWS Certificate Manager (ACM) TLS certificates, this makes it easy to create custom domain names and manage the TLS certificates for those domain names for your API Gateway and CloudFront endpoints.

We use IAMs to make sure the API Gateway has the privileges to call in to lambda functions, and that lambda functions have the privileges/execution roles to call other downstream services.

IAM authorizations allow access to methods on the API Gateway, we can leverage Cognito to provide those IAM credentials.

# Custom Authorizers

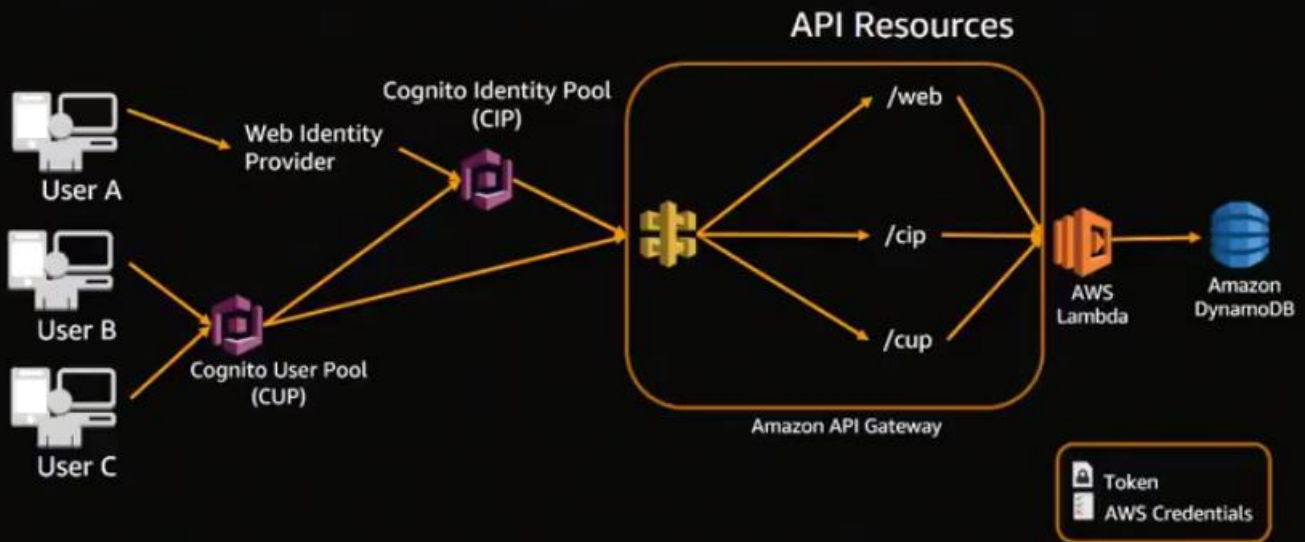


In addition to IAM roles authorization, there is also custom authorizations capabilities with API Gateway, this involves a Lambda function that you are responsible for and the sole purpose of that lambda function is to return an IAM policy that it can validate with the IAM service to provide access to that particular method to go ahead and execute the lambda function behind that API Gateway method. There are 2 types of custom authorizers with the API Gateway:

The Token request type authorization allows you to include a token in the request authorization header for that request, this enables you to implement strategies like JWT validation or use OAuth validation using 3<sup>rd</sup> party providers to validate the token in the requests.

The Request type allows you to use all values in the request headers as well as query strings, stage variables, context variables, etc. you can use this to create different authentication scheme strategies for different stages like Dev, Prod, etc.

# Authorization with Amazon Cognito



aws re:Invent

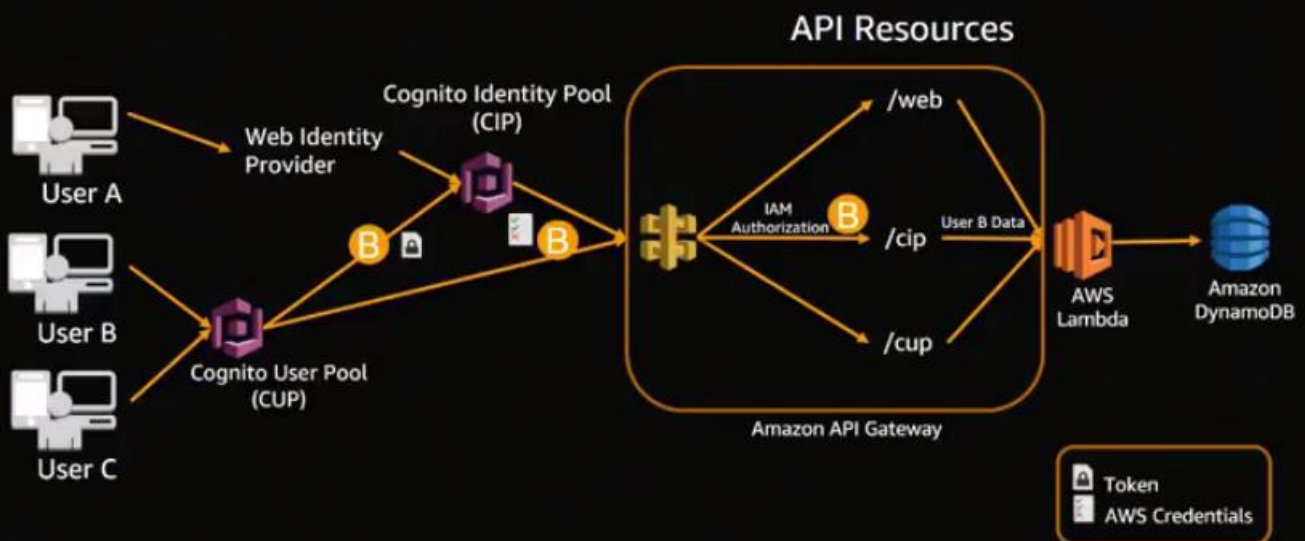
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.

aws

Let us see how AWS Cognito enables some of the authorization scenarios, above we have 3 users on the left side and:

User A authenticates through a web identity provider like Google, Facebook, etc. that web IDP provides a token back to the user that it uses to send to Cognito and exchange that for an IAM credential through a role. This is a very common pattern to get IAM authorization based on web IDPs to get access to some resource like /web that is defined inside of your API.

# Authorization with Amazon Cognito



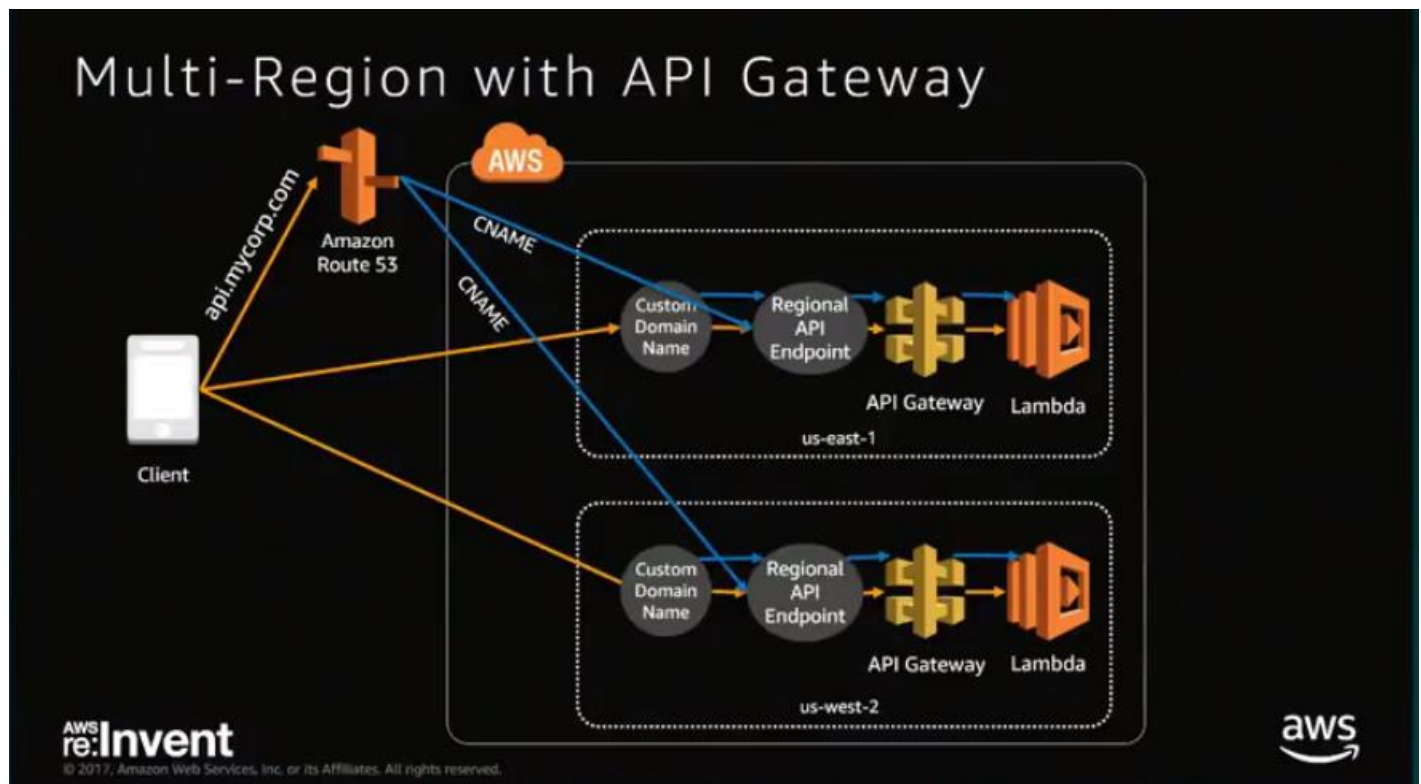
aws re:Invent

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

aws

Another scenario is User B is actually defined inside of the Cognito User Pool, this is a directory of users and groups that you can maintain inside of AWS. The flow is similar to where the user exchanges a token for authentication for a IAM credential and then leverage Cognito IDP to do that. This also lets us leverage group memberships and attributes to get a role for my IAM to get me access to the API Gateway calls. E.g. if I am a member of a group like the engineering group, then I can get access to those particular IAM roles for calling certain API Gateway methods.

Lastly, there is built-in support in the API Gateway for a Cognito User Pools authorizer where the JWT that you receive from the authentication can be validated directly to give you access to the resource. This is separate from the custom or IAM authorizers earlier.



We can now have regional endpoints with API Gateway, this helps to decouple the API Gateway from CloudFront. **This now enables multi-region serverless applications.** This allows you to create regional endpoints for API Gateway as above, each of the regional endpoints can be associated with the same custom domain name. you can now use weighted routing to direct requests to one region, while also be able to send traffic to the other region if there is a failure. You can also implement an active-active strategy where both regions are serving a portion of the requests.



# Useful Frameworks for Serverless Web Apps

- **AWS Chalice**

Python Serverless Framework

<https://github.com/aws/chalice>

Familiar decorator-based API similar to Flask/Bottle

Similar to third-party frameworks, Zappa or Claudia.js

- **AWS Serverless Express**

Run Node.js Express apps

<https://github.com/aws-labs/aws-serverless-express>

- **Java - HttpServlet, Spring, Spark and Jersey**

<https://github.com/aws-labs/aws-serverless-java-container>

**AWS re:Invent**

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



The above frameworks are available to help you adopt serverless applications. Python developers familiar with a decorator-based API can use AWS Chalice, it allows you to model APIs and API paths with lambda functions and deploy an application from python code into a serverless environment.

Existing NodeJS express apps or Java apps written in different frameworks, there are built-in libraries available in GitHub to convert those code over into the AWS Serverless platform.

## Pattern 2: Data Lake

Data Lake Patterns.

# 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 re:Invent**

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



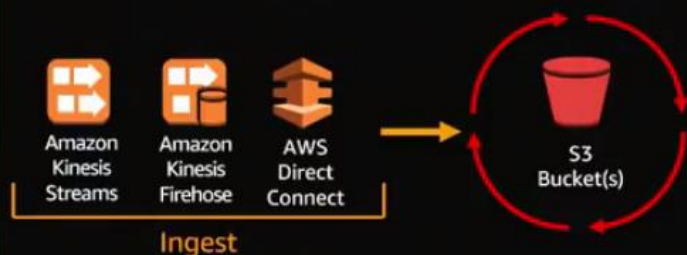
When you think about data analytics and processing data, they are not static. A data lake sets the foundational components for you to be more agile as these changes in the future. A common approach is to get all of your organizational data into that data lake at the start. A data lake should implement a schema-on-read strategy, so that as you can specify a schema when you request to read the data from the data lake.

## AWS Serverless Data Lake



**AWS S3** sits at the center of your data lake strategy

# AWS Serverless Data Lake



A data lake needs to be able to *accept/ingest data in a lot of different forms in great volumes*

# AWS Serverless Data Lake

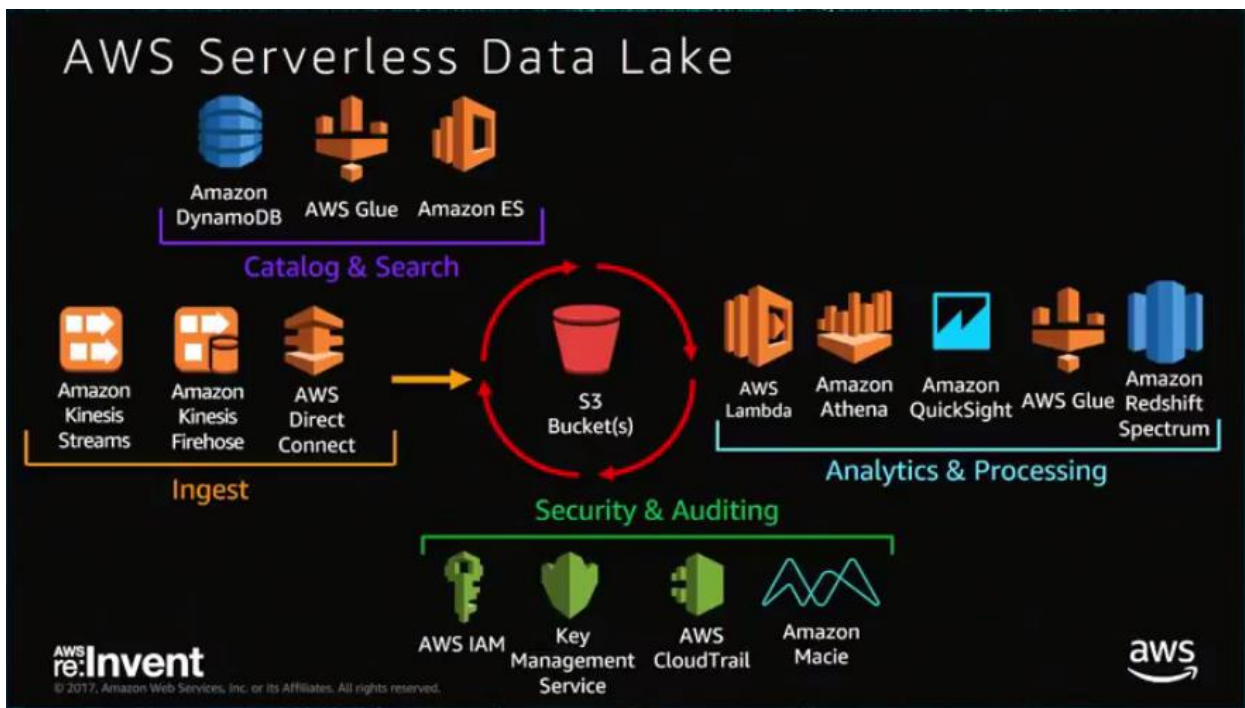


To do *Cataloging and Search*, we have a couple of patterns that leverage *DynamoDB*, *Glue* and *Elastic Search*

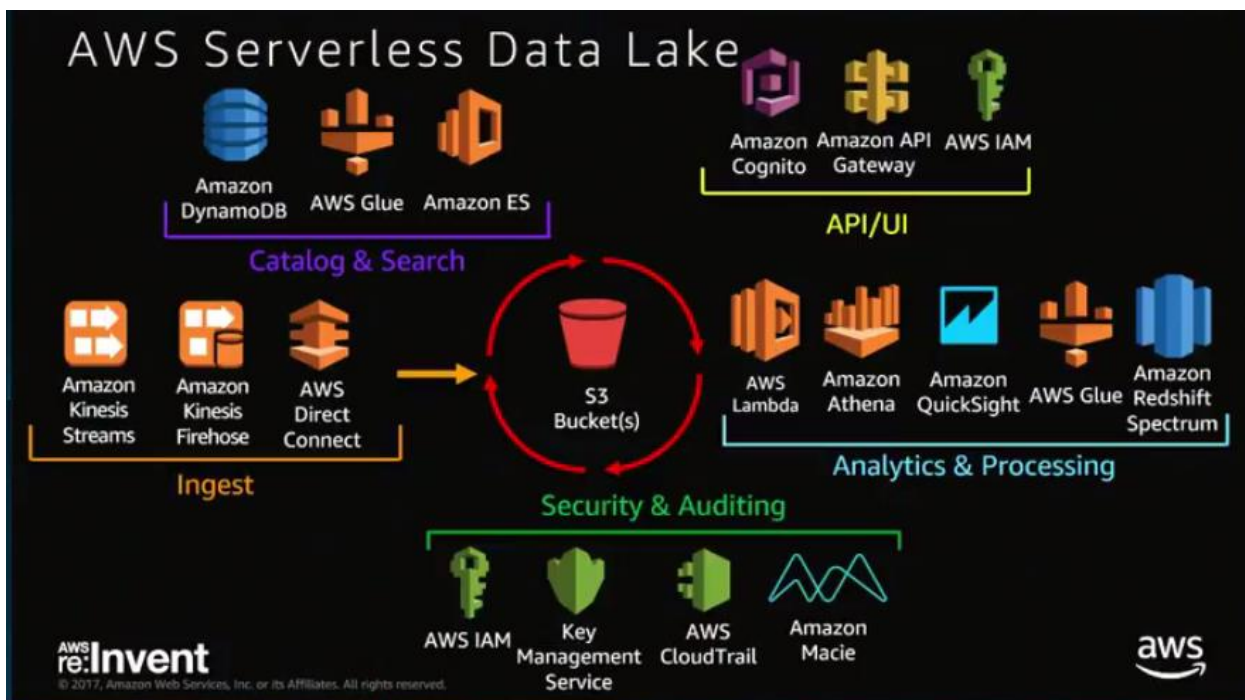
# AWS Serverless Data Lake



For *Analytics and Processing*, there are many different serverless services that we can use to do this as above.



The **Security** component to the data lake allows you to manage entitlements and provide access to data collections for different groups. You want to be able to encrypt your data using **S3** natively or **KMS**, you can audit access to your data using **CloudTrail** and its support for S3. **Amazon Macie** is a data classification service for S3, it will automatically infer the types of the data based on the contents of the files that is stored in S3 using machine learning algorithms.



Finally, you need a **UI** for your data lake to be able to visualize and interact with it. Also, a way for your users to log in to the portal and discover and search for things. You can also have a programmatic API for interacting with your data lake.



# The Foundation...Amazon S3

- No need to run compute clusters for storage
- Virtually unlimited number of objects and volume
- Very high bandwidth – no aggregate throughput limit
- Multiple storage classes
- Versioning
- Encryption
- AWS CloudTrail Data Events
- S3 Analytics and Inventory
- AWS Config automated checks
- S3 Object Tagging

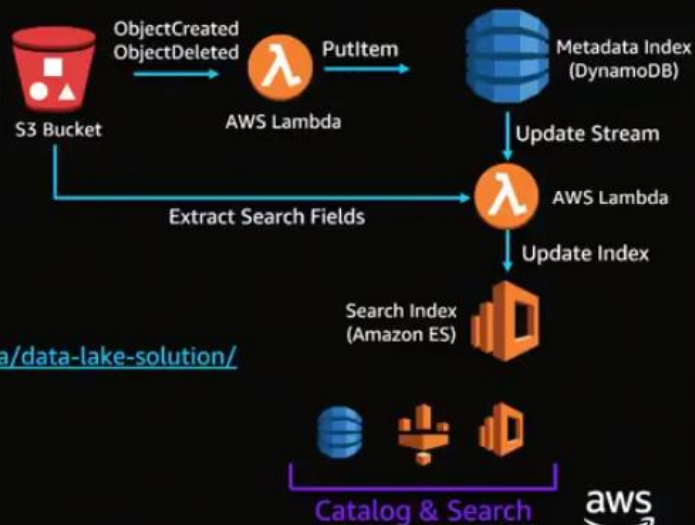


AWS  
re:Invent

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

## Search and Data Catalog

- DynamoDB as Metadata repository
- Amazon Elasticsearch



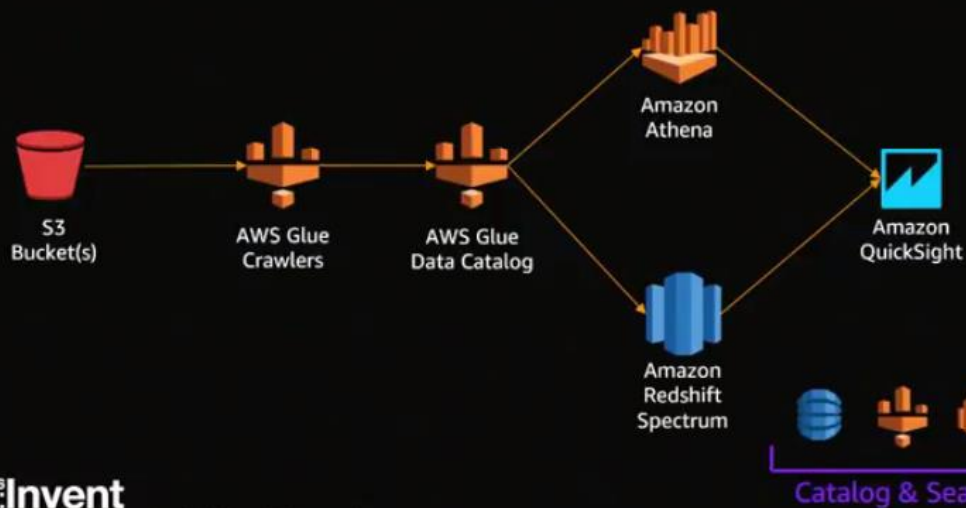
<https://aws.amazon.com/answers/big-data/data-lake-solution/>

AWS  
re:Invent

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

The first pattern is an event-based pattern, as data arrives in S3 we can run a lambda function to store metadata about that file into DynamoDB. This metadata can include data about the data group its from, if its an IoT use case or device, it could be version number. We can then have another lambda function that runs whenever something is saved in DynamoDB and stores some information about that file into Elastic Search to create a searchable index.

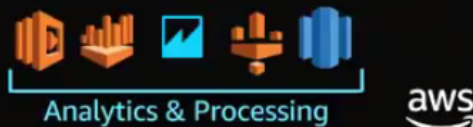
# Instantly query your data lake on Amazon S3



AWS Glue plays a vital role since it can dispatch crawlers on a scheduled basis or on-demand. It can look into your data and build a schema for you for cataloguing purpose.

## Analytics and Processing

- Amazon QuickSight
- Amazon Athena
- AWS Lambda
- Predictive Analytics
- Amazon EMR
- AWS Glue (ETL)



# Athena – Serverless Interactive Query Service

```
SELECT gram, year, sum(count) FROM ngram
WHERE gram = 'just say no'
GROUP BY gram,year ORDER BY year ASC;
```

44.66 seconds...Data scanned: 169.53GB

Cost: \$5/TB or \$0.005/GB = \$0.85



AWS  
re:Invent

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

## Athena – Best Practices

- Partition data  
s3://bucket/flight/parquet/year=1991/month=1/day=2/
- Columnar formats – Apache Parquet, AVRO, ORC
- Compress files with splittable compression (bzip2)
- Optimize file sizes

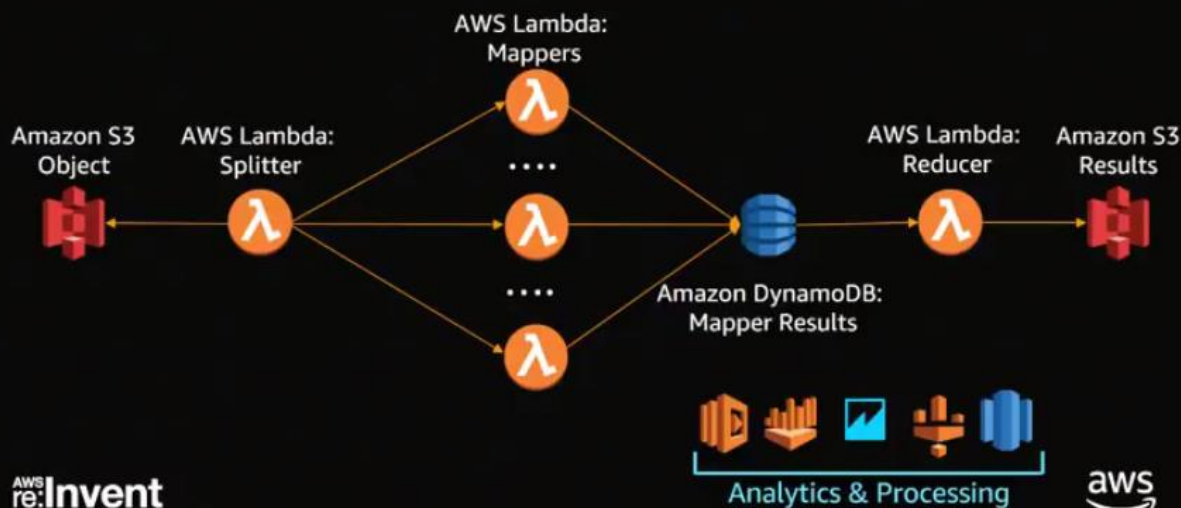
<https://aws.amazon.com/blogs/big-data/top-10-performance-tuning-tips-for-amazon-athena/>



AWS  
re:Invent

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

# Serverless batch processing



The above **serverless batch processing pattern** is suitable for analytical processing needs. This is a DIY approach where you have a lambda function that takes in your source data and splits it into smaller sizes (by lines or by size), then hands them off to a bunch of mapper functions that run in parallel to process the data and write their results to a persistence store like DynamoDB. Then we have a reducer lambda function that reduces the results in the required S3 bucket.

## Fannie Mae Serverless Financial Modeling

Financial Modeling is a Monte-Carlo simulation process to project future cash flows , which is used for managing the mortgage risk on daily basis:

- Underwriting and valuation
  - Risk management
  - Financial reporting
  - Loss mitigation and loan removal
- 
- ~10 Quadrillion ( $10 \times 10^{15}$ ) of cash flow projections each month in hundreds of economic scenarios.
  - One simulation run of ~ 20 million mortgages takes 1.4 hours, >4 times faster than the existing process.



**AWS re:Invent**

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





## Pywren

- Pywren Python library provides 10 TFLOPS of peak compute power with new default – 1,000 concurrent functions
- Achieve over 60 GB/sec of read and 50 GB/sec of write performance using Amazon S3

<https://pywren.io>

**AWS re:Invent**

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



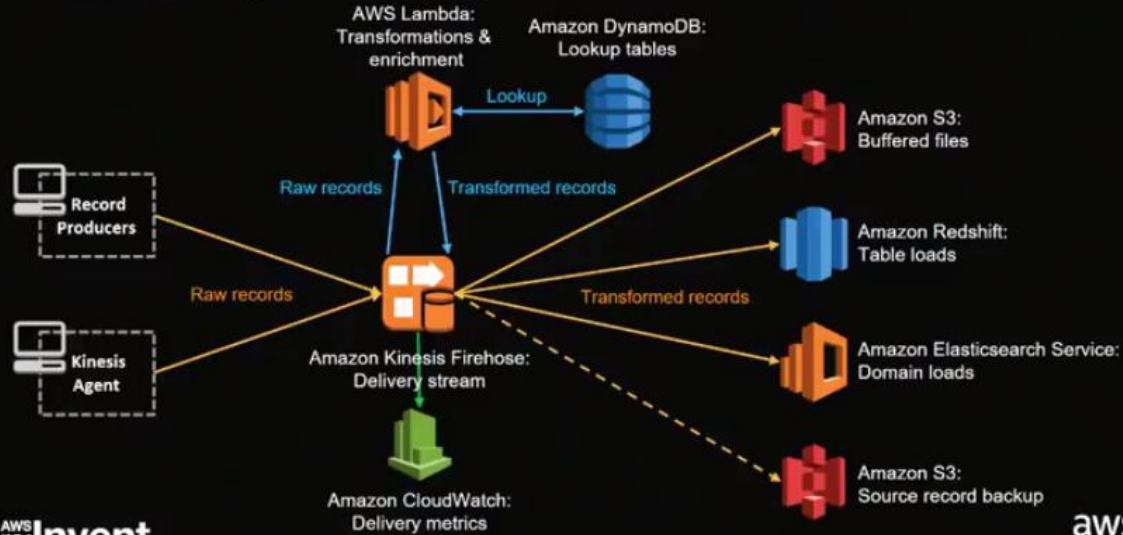
## Pattern 3: Stream Processing

Stream Processing pattern. Assuming you have been collecting click-stream event data off your apps but have been processing them on a daily basis, can you reduce the time to 3 minutes? This is an example of a stream processing problem.

## 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



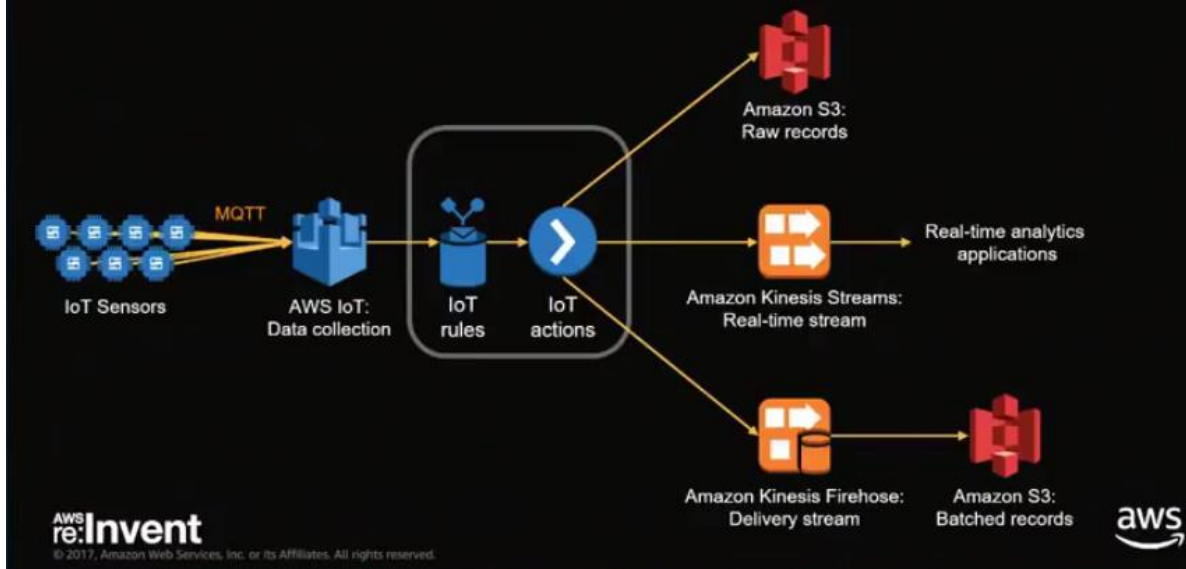
This is an example of an architecture that can solve the stream data ingestion and analytics processing problem for us. The core of this architecture is around **Amazon Kinesis Firehose** in the middle, it lets you ingest large amount of data up, buffers up the data, then delivers them in micro-batches to destination services. This decouples the high rate of data ingestion from the destination services that need the data in smaller batch sizes to act on.

You can send data in raw form directly to the Kinesis firehose API, or you can install a kinesis firehose log agent that will look at your log files and ships new messages to the kinesis firehose messaging stream for you as they arrive. In the firehose delivery stream, you can do transformations on your records using a lambda function that does that. The destination services are listed on the right side like S3, Kibana, etc.

## Best practices

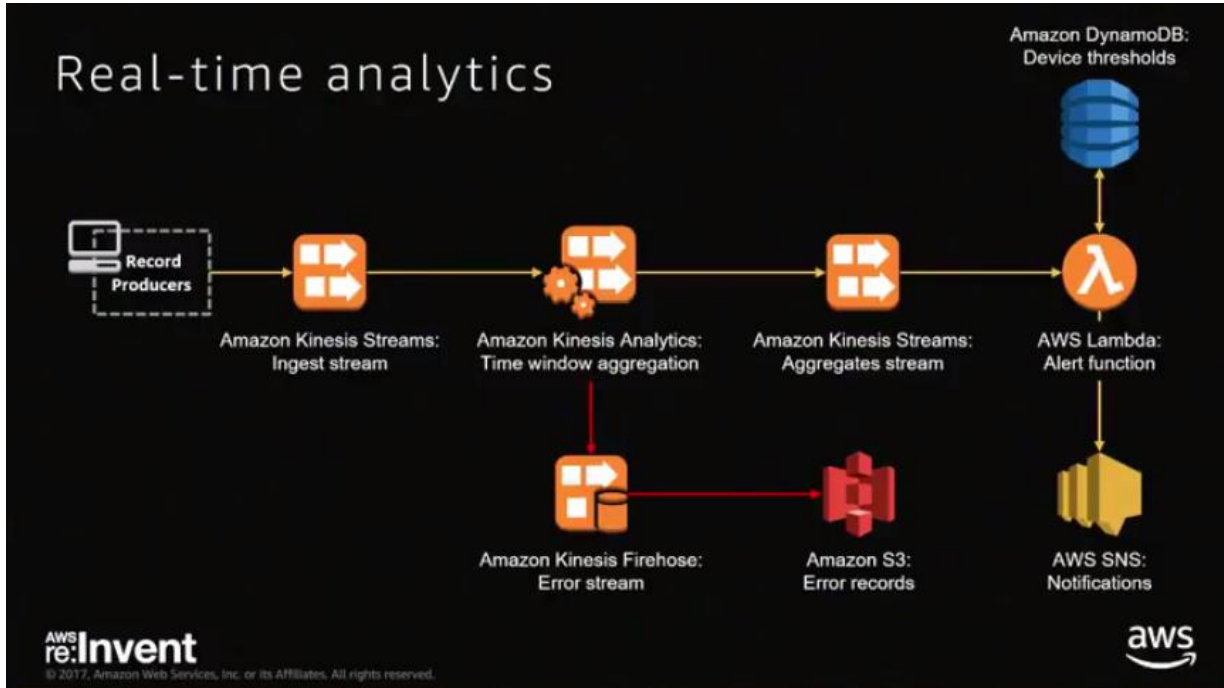
- Tune Firehose **buffer size** and **buffer interval**
  - Larger objects = fewer Lambda invocations, fewer S3 PUTs
- Enable **compression** to reduce storage costs
- Enable **Source Record Backup** for transformations
  - Recover from transformation errors
- Follow [Amazon Redshift Best Practices for Loading Data](#)

## Sensor data collection



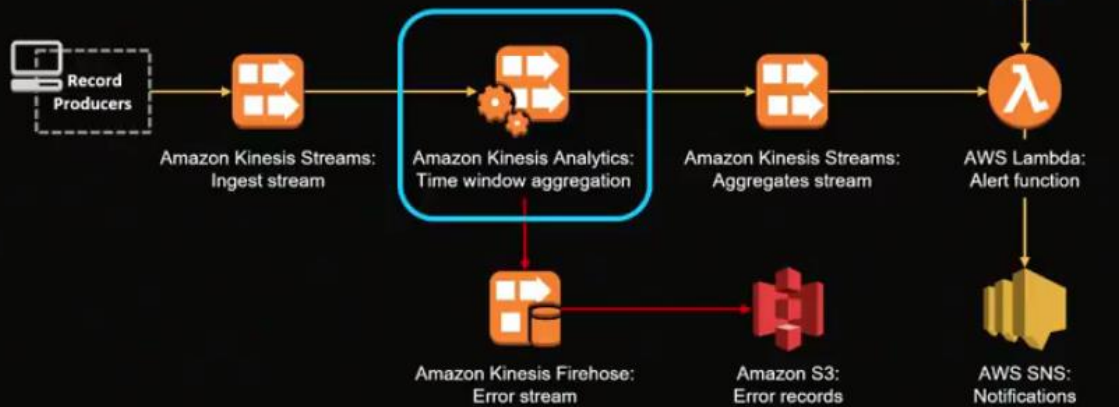
This is an IoT stream processing use case for things like online thermostats. AWS IoT serves as the server side that receives the messages from the IoT devices in the field as they arrive over MQTT protocols, the IoT rules then are used to define filtering rules over the ingested IoT messages. You can create a rule to filter out say temperature readings only from a log containing pressure, temperature, velocity, image data and react to the temperature readings only. The IoT rules deliver the filtered data to an IoT action which will deliver the filtered data to the intended destination like S3, firehose for creating batched data, etc.

## Real-time analytics



An example of a real-time application pipeline is the case for collecting noisy temperature sensor data is to take messages in, smooth them out over a 60 seconds window, then compare that against a threshold that is set like 90 degrees by the user. We want to set an alarm if the average temperature reading exceeds the set threshold.

# Real-time analytics



aws re:Invent

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



## Amazon Kinesis Analytics



```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE) as "window_ts",
SUM("measurement") as "sample_sum",
COUNT(*) AS "sample_count"
FROM "SOURCE_SQL_STREAM_001"
GROUP BY "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
```

aws re:Invent

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



The Amazon Kinesis Analytics service allows you to perform near real-time analytics on data coming into the stream, you represent your logic as a SQL code.



# Amazon Kinesis Analytics



```
CREATE OR REPLACE PUMP "STREAM_PUMP" AS INSERT INTO "DESTINATION_SQL_STREAM"
SELECT STREAM "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE) as "window_ts",
SUM("measurement") as "sample_sum",
COUNT(*) AS "sample_count"
FROM "SOURCE_SQL_STREAM_001"
GROUP BY "device_id",
STEP("SOURCE_SQL_STREAM_001".ROWTIME BY INTERVAL '1' MINUTE);
```

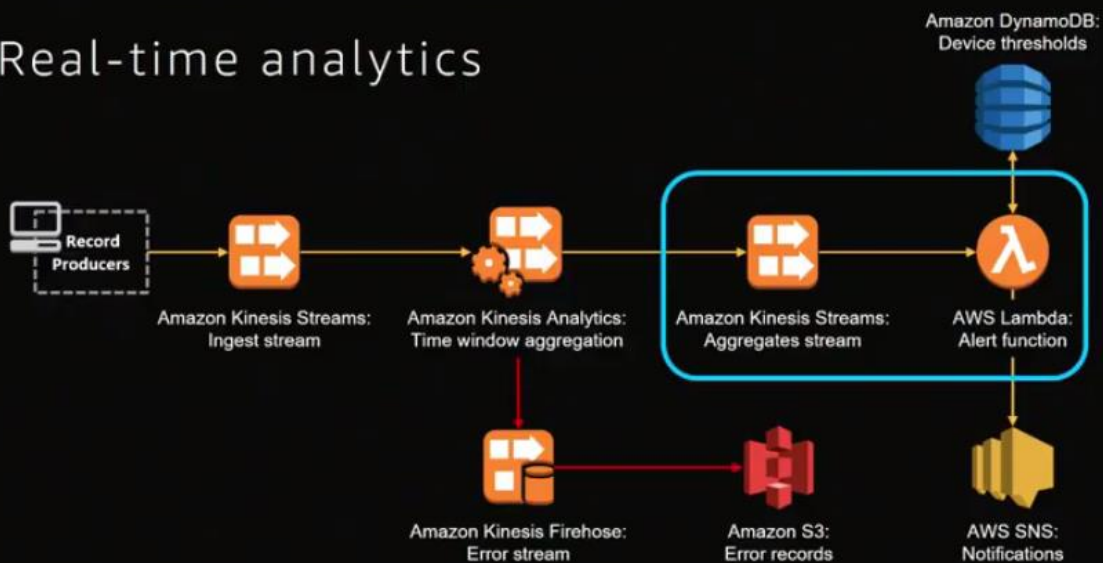
[Aggregation] → 1-minute tumbling window

aws  
re:Invent

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

aws

## Real-time analytics



aws  
re:Invent

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

aws

We can now perform a custom thresholding logic in the downstream Amazon Kinesis stream that aggregates data, use a Lambda function to compare them with the thresholds set in DynamoDB, and then decide if we need to send an alert using SNS. Note the error stream that can be stored in a destination S3 bucket.

# Amazon Kinesis Streams and AWS Lambda



- Number of Amazon Kinesis Streams **shards** corresponds to **concurrent invocations** of Lambda function
- **Batch size** sets maximum number of records per Lambda function invocation

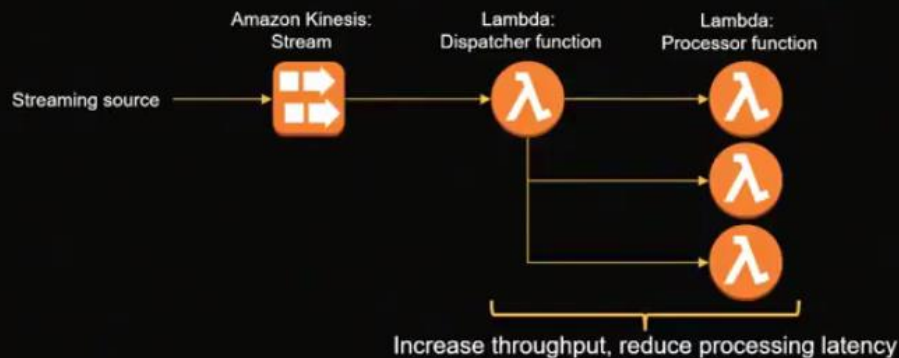
aws  
re:Invent

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



## Fan-out pattern

Fan-out pattern trades strict message ordering vs higher throughput & lower latency



aws  
re:Invent

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



# Thomson Reuters – Product Insight

Solution for **usage analysis tracking**:

Capture, analyze, and visualize analytics data generated by offerings, providing insights to help product teams continuously improve the user experience

Throughput: Tested **4,000** requests / second

Growing to **10,000** requests / second or **25 Billion** requests / month

Latency: new events to user dashboards in less than **10 seconds**

Durable: **no data loss** since inception

**AWS re:Invent**

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



## Best practices

- Tune **batch size** when Lambda is triggered by Amazon Kinesis Streams
  - Higher batch size = fewer Lambda invocations
- Tune **memory** setting for your Lambda function
  - Higher memory = shorter execution time
- Use Kinesis Producer Library (**KPL**) to batch messages and saturate Amazon Kinesis Stream capacity

If you have control over the client, you should look at implementing the **Kinesis Producer Library KPL** locally batches up multiple messages into one kinesis record. This lets you fully saturate your kinesis stream capacity as well as reduce your API invocations to the Kinesis stream service and the stream processing pipeline.

## Compare related services

	Amazon Kinesis Streams	Amazon SQS	Amazon SNS
Maximum record size	1 MB	256 KB	256 KB or 140 Bytes (SMS)
Message durability	Up to retention period	Up to retention period	Retry delivery (depends on destination type)
Maximum retention period	7 days	14 days	Up to retry delivery limit
Message ordering	Strict within Shard	Standard – Best effort FIFO – Strict within Message Group	None
Delivery semantics	At-least-once	Standard – At-least-once FIFO – Exactly-once	At-least-once
Parallel consumption	Multiple Consumers per Shard with independent iterators	Multiple Readers per Queue (but one message is only handled by one Reader at a time)	Multiple Subscribers per Topic
Scaling	By throughput using Shards	Automatic	Automatic
Iterate over messages	Shard iterators	No	No
Delivery destination types	Kinesis Consumers	SQS Readers	HTTP/S, Mobile Push, SMS, Email, SQS, Lambda

**AWS re:Invent**

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



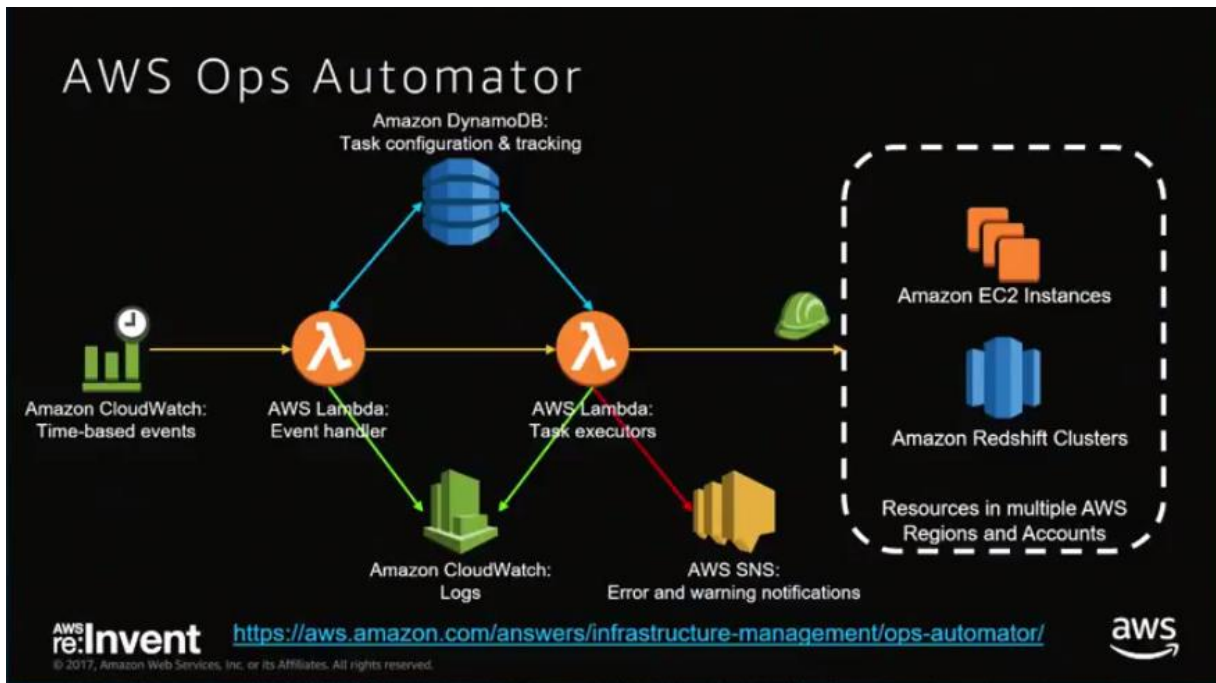
## Pattern 4: Operations Automation

### Automation characteristics

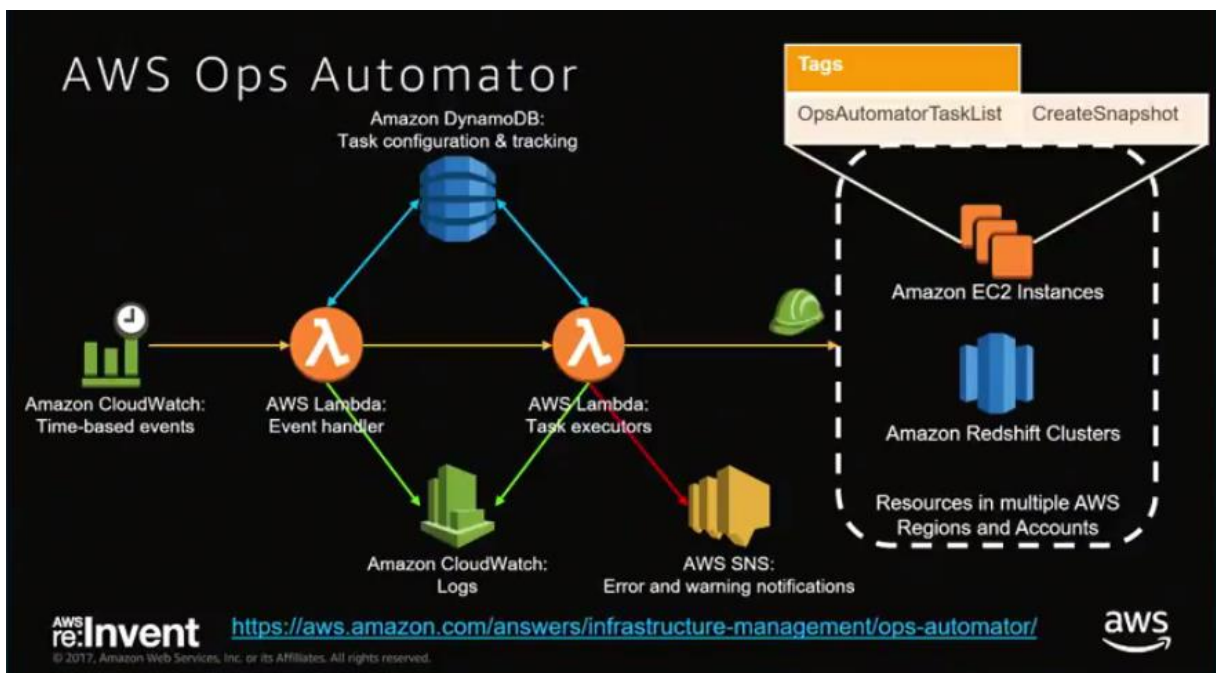
- Periodic jobs
- Event triggered workflows
- Enforce security policies
- Audit and notification
- Respond to alarms
- Extend AWS functionality

... All while being Highly Available, Scalable and Auditable



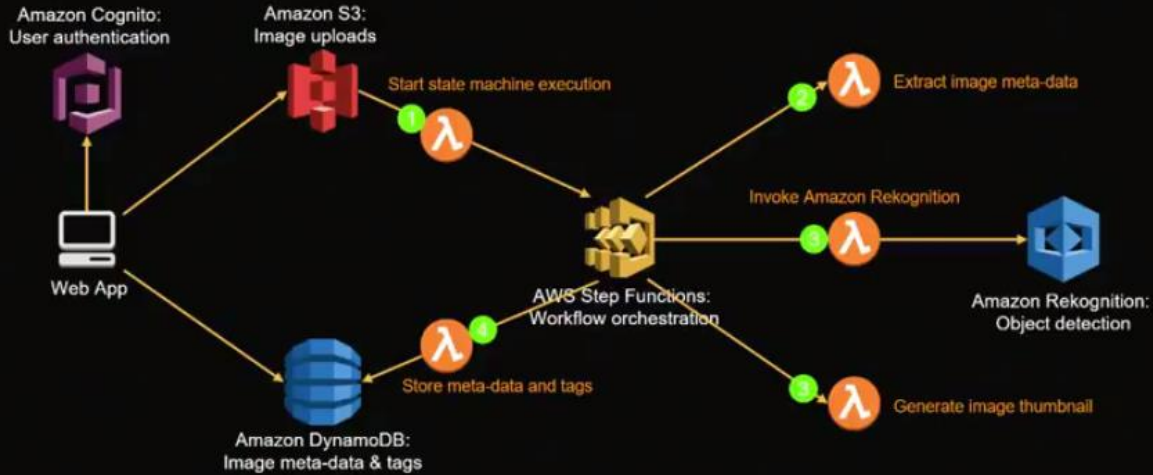


For periodic task where we need to schedule things like taking EBS snapshots and scheduling things like how long we want to retain an EBS snapshot. The **AWS Ops Automator service** can be deployed in the form of 2 templates, the first one has to be deployed in a master account where all the logic and tracking will happen. The second templates are what you deploy in the slave accounts where you want to trigger the actions.



You need to put the tag on the EBS volumes that you want to snapshot.

# Image recognition and processing



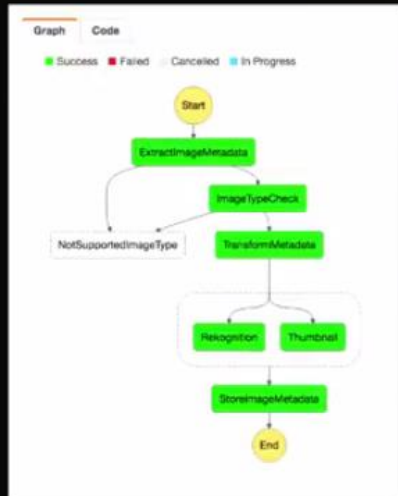
AWS re:Invent

<https://github.com/aws-labs/lambdarefarch-imagercognition>

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



# Step Functions state machine



AWS re:Invent

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



# Enforce security policies



## Autodesk - Tailor

Serverless AWS **Account Provisioning and Management** Service:

- Automates AWS Account creation,
- Configures IAM, CloudTrail, AWS Config, Direct Connect, and VPC
- Enforces corporate standards
- Audit for compliance

Provisions new Accounts in **10 minutes** vs 10 hours in earlier manual process

Open source and extensible: <https://github.com/alanwill/aws-tailor>

**aws re:Invent**

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



## Best practices

- Gracefully handle **API throttling** by retrying with an exponential back-off algorithm (AWS SDKs do this for you)
- Publish **custom metrics** from your Lambda function that are meaningful for operations (e.g. number of EBS volumes snapshotted)
- Enable **X-Ray** tracing for your Lambda functions
- Document how to **disable** event triggers for your automation when troubleshooting

# Summary

Use DevOps tools to **automate** your serverless deployments

Apply serverless patterns for common use-cases:

- Web application
- Data Lake Foundation
- Stream processing
- Operations automation

What will **you** build with Serverless?

**AWS**  
**re:Invent**

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



## Related Sessions

SRV212 - Build a Scalable Serverless Web Application on AWS That Can Span Millions of Concurrent Users

SRV213 - Thirty Serverless Architectures in 30 Minutes

SRV317 - Unlocking High Performance Computing for Financial Services with Serverless Compute

SRV319 - How Nextdoor Built a Scalable, Serverless Data Pipeline for Billions of Events per Day

SRV322 - Migration to Serverless: Design Patterns and Best Practices

SRV403 - Serverless Authentication and Authorization: Identity Management for Serverless Applications

ARC316 - Getting from Here to There: A Journey from On-premises to Serverless Architecture

ABD202 - Best Practices for Building Serverless Big Data Applications

SID205 - Building the Largest Repo for Serverless Compliance-as-Code

SID307 - Serverless for Security Officers: Paradigm Walkthrough and Comprehensive Security Best Practices

**AWS**  
**re:Invent**

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

