Event-Driven Computing in AWS

Objectives

Key objectives of this chapter

- Event-Driven computing in the cloud
- Lambda Functions
  - Use cases
  - The programming model
  - Lambda blueprints

1.1 Defining Event-Driven Computing

- Event-driven computing is a processing paradigm in which the flow of computing is determined by events raised / generated in response to a change in a resource or system
- Event-driven computing is asynchronous in nature and decouples event producers from their consumers
- When events are published, some sort of notification (message) is sent to a configured destination where that message is retrieved by the consumer
  - Message retrieval happens on demand through some sort of polling or made available to the message consumer through a call-back mechanism
- Event-driven computing with asynchronous communication models allow you to create loosely coupled, independently scalable, and reusable services (this architecture style is sometimes referred to as Microservices Architecture)

Notes:

Event-Driven Computing has its roots in Event-Driven Programming model which, according to Wikipedia, " *... is a programming paradigm in which the flow of the program is determined by events such as user actions (mouse clicks, key presses), sensor outputs, or messages from other programs/threads. ... In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected. In embedded systems, the same may be achieved using hardware interrupts instead of a constantly running main loop.*"

1.2 The IoT

- The Internet of Things (IoT) is a network of interconnected devices that might be in the form of actuators or signal producers, like sensors, embedded devices, on-site applications, etc., working in concert in a particular problem domain to produce some business value
- AWS IoT Platform is a managed cloud service that enables the flow of events from actuators to cloud applications and other devices
- AWS IoT Platform allows users to combine AWS services like Lambda, Kinesis, S3, and DynamoDB to create applications that respond to, gather, process, analyze, and act on data generated by connected devices without user involvement

1.3 AWS IoT SDKs

- AWS IoT Device SDKs enable your devices to connect, authenticate, and exchange messages with AWS IoT Core using a number of protocols:
  - MQTT, HTTP, or WebSockets
- The AWS IoT Device SDK supports C, JavaScript, and Arduino, and includes the client libraries, the developer guide, and other resources to help clients get up to speed with the IoT services quickly and efficiently

1.4 Infrastructure Events

- AWS exposes a wide range of infrastructure events that client applications can subscribe and react to
- Examples:
  - Object upload to your S3 bucket
  - Object deleted in your S3 bucket
  - An items (record) inserted into your DynamoDB table
  - Assorted CloudWatch events
  - RDS DB Instance “Low Storage” event (see the slide's notes for details of how it can be used)
  - EC2 instances launched and terminated by the Auto Scaling policy in response to a change in system loads

Notes:

“When the 'Low Storage' RDS DB Instance event is published to an SNS topic, SNS pushes this event to a subscribing Lambda function, which in turn leverages the RDS API to increase the storage capacity

allocated to your DB instance. The provisioning itself takes place within the specified DB maintenance window."

1.5 AWS Event-Driven Computing Enabling Services

- AWS Services Event-Driven computing is enabled though these managed services:
  - Simple Queue Service (SQS)
  - Simple Notification Service (SNS)
  - Simple Email Service (SES)
- **Note 1:** These are PaaS-type of capabilities
- **Note 2:** Some of these services may have already been covered previously and here we only give the necessary digest of the same

1.6 Simple Queue Service (SQS)

- A fully managed cloud-grade distributed queue-based messaging platform
- Using SQS, you can send, receive, and retain messages in a fully decoupled asynchronous fashion without requiring message consumers to be always up and running
- You have a choice between two message delivery options:
  - **SQS standard queues**
    - ✔ Use it for fire-hose throughput, best-effort (not guaranteed) message ordering, and at-least-once delivery guarantee
  - **SQS FIFO queues**
    - ✔ Guarantee that messages are processed exactly once, in the exact

order that they are sent,

- ✔ Less overall throughput compared with SQS standard queues

Notes:

The Amazon SQS Messaging API for Java supports sending and receiving messages to a queue (the JMS point-to-point model), as specified in the JMS 1.1 specification. The SQS Java Messaging Library uses a JMS interface to Amazon SQS that you can use in your JMS-enabled Java applications with minimum changes.

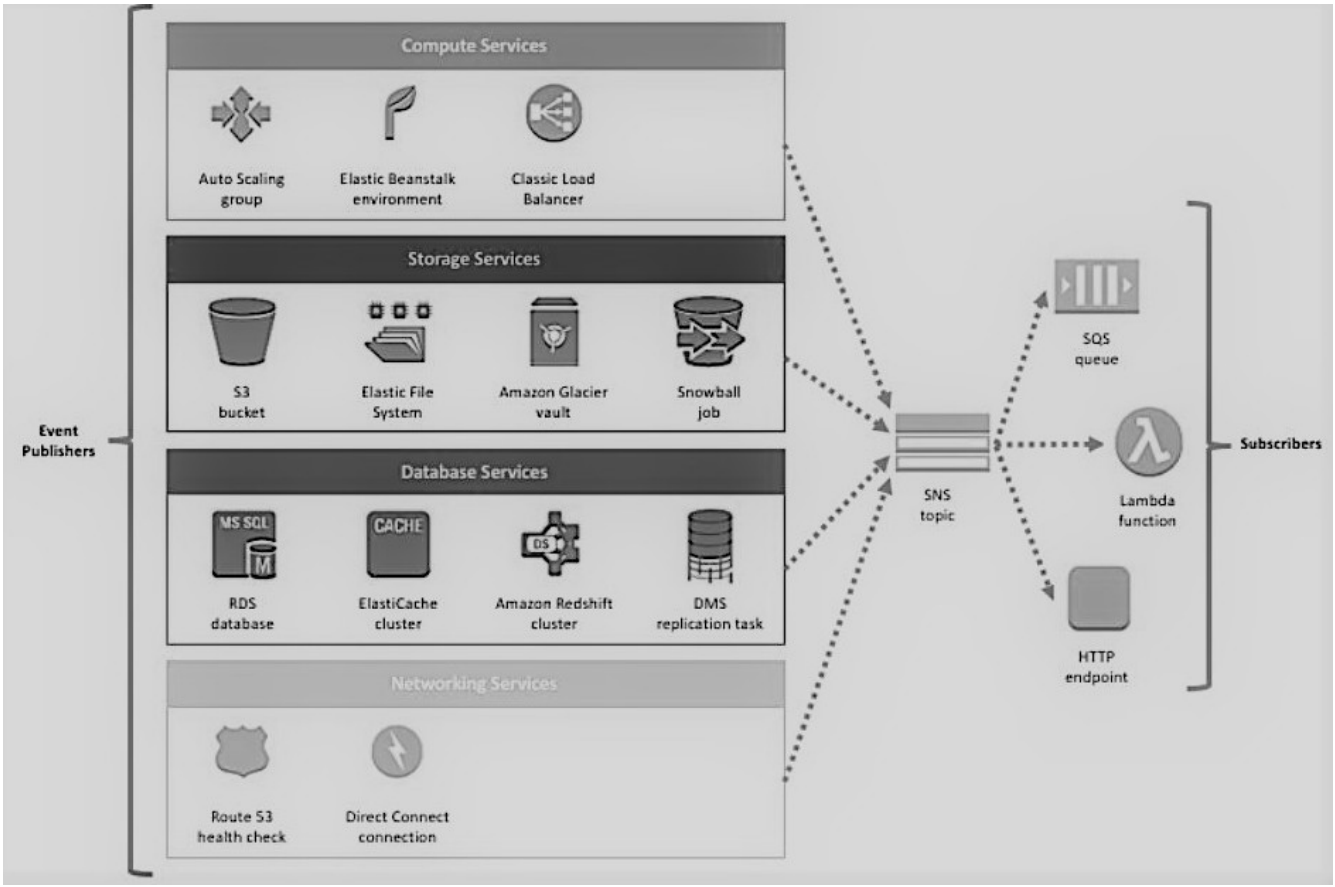
SQS offers clients Delay Queues for delayed message delivery. The delay (during which a message is invisible for consumers) may be configured for as long as 15 minutes. Hiding messages on a queue may help with synchronization of application sub-systems.

A single-threaded SQS client can fetch within a second no more than about 50 messages. To increase the throughput of your application, add multiple clients, or create a multi-threaded SQS client.

1.7 Simple Notification Service (SNS)

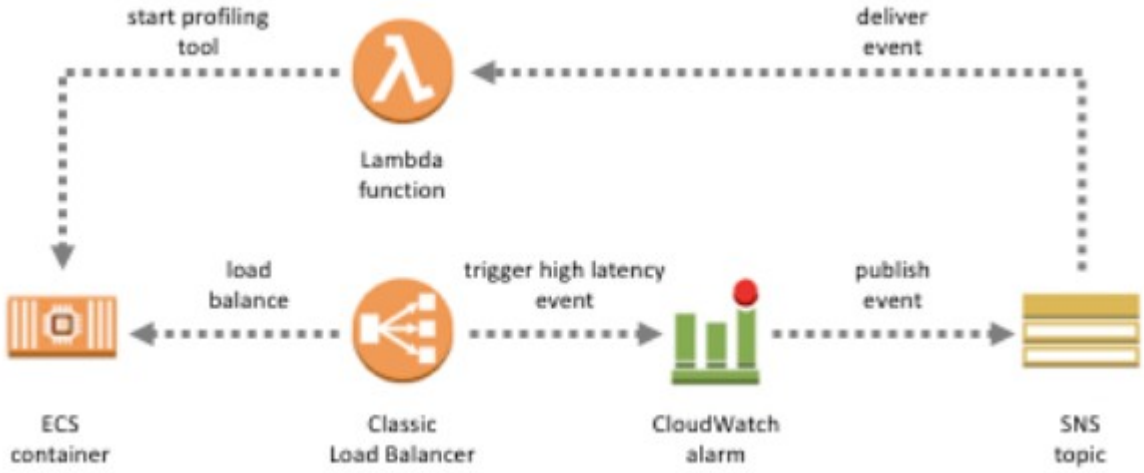
- Simple Notification Service (SNS) is a fully managed cloud-grade push messaging platform
- SNS allows you build applications based on the pub-sub (topic-based) protocol
- You can push individual or fan-out messages to the interested subscribers, such as mobile devices, email recipients, SMS clients, etc.
- Integrates with HTTP end-points, AWS Lambda functions, and SQS

1.8 AWS Services Native Integration with SNS



- Adapted from: https://amzn.to/2ScPGVr

1.9 Example of Using ELB Events to React to ECS High Latency



Source: AWS Documentation

Notes:

"*In this example, whenever your ECS cluster breaches your load balancer latency threshold, an event is posted by CloudWatch to an SNS topic, which then triggers a subscribing Lambda function. This function runs a task on your ECS cluster to trigger a latency profiling tool, hosted on the cluster itself. This can enhance your latency troubleshooting exercise by making it timely.*"

1.10 Simple Email Service (SES)

- Amazon Simple Email Service enables you to send and receive email using a scalable email platform
- Tightly integrates with S3, SNS, and Lambda
- Verifies email addresses and domains
  - Maintains email metrics, such as the number of emails sent and the email bounce rates

1.11 AWS Monitoring with CloudWatch

- Amazon CloudWatch is a monitoring service for AWS-deployed cloud resources
- CloudWatch is capable of monitoring and collecting some predefined resource metrics, or custom-defined resource artifacts (e.g. log files and alarms)
- Through Amazon CloudWatch Events you can respond to state changes in your AWS resources or when a particular threshold is exceeded
- For collecting custom log data, you need to install and configure CloudWatch agents that will be sending your logs to the CloudWatch Logs service and then create your specific metric filter there

1.12 Example of CloudWatch Integration with EC2

- CloudWatch monitors EC2 instances with respect of operational performance, including such metrics as CPU utilization, I/O operations, instance state change, network traffic (in/out), and response latencies
  - For example, CloudWatch generates an event when the state of an EC2 instance changes from pending to running or when Auto Scaling launches an instance
- CloudWatch can also be configured to monitor HTTP status codes in Apache logs, etc.

1.13 Amazon CloudWatch Infrastructure Events

- There are three main components of this capability:
  - **Events**
    - ✔ These are notifications / messages encoded as JSON documents
  - **Rules**
    - ✔ Used to match incoming events and route them to one or more targets for processing
  - **Targets**
    - ✔ Responsible for processing events specified in rules
    - ✔ Currently supported are: built-in targets, Lambda functions, Kinesis streams, and SNS topics, with more types to come
- For more information, visit https://amzn.to/1TLY0XF

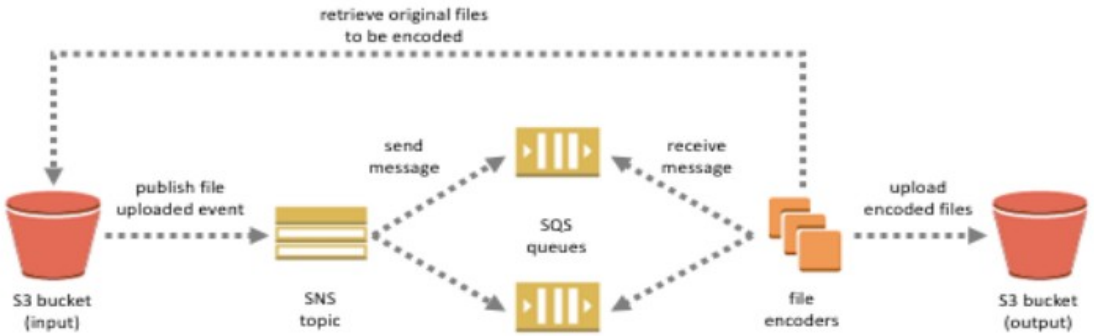
1.14 Events Generated from Resources

- AWS services have been augmented with capabilities to generate events in response to state changes
- Essentially, events trigger notifications that clients can subscribe to for further action
- In most cases, you need to enable notifications at the resource level to have AWS start forwarding notifications about published events to the destinations of your choice, including:
  - SNS topic
  - SQS queue
  - AWS Lambda

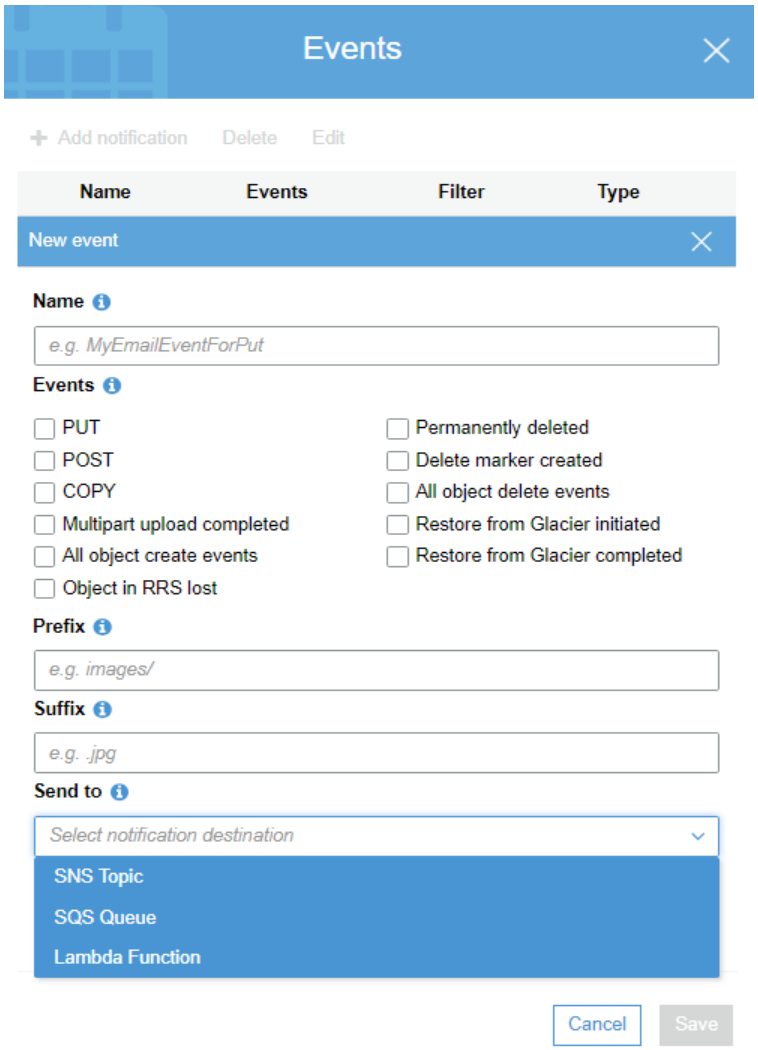
1.15 S3 Events

- S3 allows users receive notifications about certain events that happen at the bucket level
  - You need to enable this notification feature
- With this feature enabled, you can select S3 events you are interested in and the destinations for those notifications
- Currently, Amazon S3 can publish notifications for the following events:
  - A new object created
  - An object removed
  - An object restored (from Glacier)
  - An object lost (may happen in case of Reduce Redundancy Storage)
- For a complete list of event notifications types, visit https://amzn.to/2McvXmt

1.16 Example of Using S3 Object Upload Event



1.17 S3 Events Configuration in the AWS Management Console



- The Events page is available in the **Properties** tab of your S3 bucket

1.18 S3 Event Message Structure

- Notification messages published by S3 are in the JSON format
- The message has the following main keys:
  - **responseElements**
    - ✔ Useful if you want to trace a request by following up with Amazon support
  - **s3**
    - ✔ Provides information about the bucket and object involved in the event
  - **sequencer**
    - ✔ Provides a way to determine the sequence of events
- A sample JSON-encoded event is shown in the slide's notes
- For more information, visit https://amzn.to/2AM0ItO

Notes:

The following example shows version 2.1 of the event message JSON structure that is the version currently being used by Amazon S3.

```
{
   "Records":[
      {
         "eventVersion":"2.1",
         "eventSource":"aws:s3",
         "awsRegion":"us-west-2",
         "eventTime":The time, in ISO-8601 format, for example, 1970-01-
01T00:00:00.000Z, when Amazon S3 finished processing the request,
         "eventName":"event-type",
         "userIdentity":{
            "principalId":"Amazon-customer-ID-of-the-user-who-caused-the-event"
         },
         "requestParameters":{
            "sourceIPAddress":"ip-address-where-request-came-from"
```

```
         },
         "responseElements":{
            "x-amz-request-id":"Amazon S3 generated request ID",
            "x-amz-id-2":"Amazon S3 host that processed the request"
         },
         "s3":{
            "s3SchemaVersion":"1.0",
            "configurationId":"ID found in the bucket notification configuration",
            "bucket":{
               "name":"bucket-name",
               "ownerIdentity":{
                  "principalId":"Amazon-customer-ID-of-the-bucket-owner"
               },
               "arn":"bucket-ARN"
            },
            "object":{
               "key":"object-key",
               "size":object-size,
               "eTag":"object eTag",
               "versionId":"object version if bucket is versioning-enabled,
otherwise null",
               "sequencer": "a string representation of a hexadecimal value used to
determine event sequence,
                   only used with PUTs and DELETEs"
            }
         },
         "glacierEventData": {
            "restoreEventData": {
               "lifecycleRestorationExpiryTime": "The time, in ISO-8601 format, for
example, 1970-01-01T00:00:00.000Z, of Restore Expiry",
               "lifecycleRestoreStorageClass": "Source storage class for restore"
            }
         }
      }
   ]
}
```

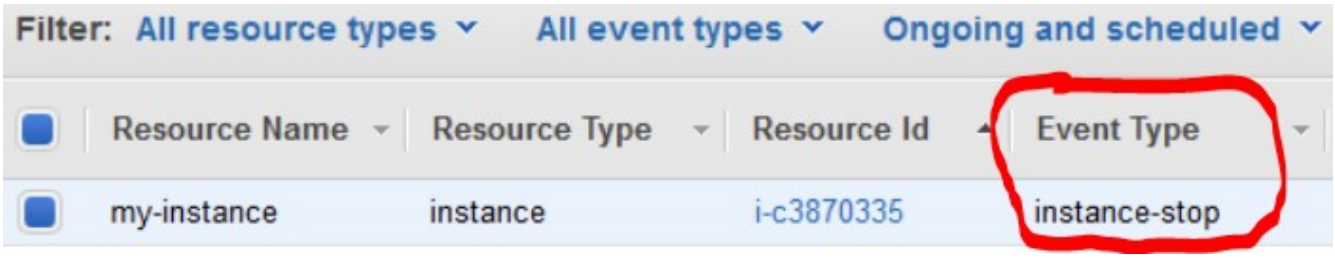
Source: AWS documentation [https://docs.aws.amazon.com/AmazonS3/latest/dev/notification-content-structure.html]

1.19 DynamoDB Stream Events

- DynamoDB Streams is an added-value service that, when enabled, captures a time-ordered sequence of item-level modifications in a DynamoDB table and durably stores the information for up to one day (24 hours)
- A stream, essentially, consists of a sequence of change events each of which is identified by a sequence id
- Applications can access captured records (that are stored in the DynamoDB Streams database) in near-real time
- To read and process DynamoDB Streams records, user applications must access a DynamoDB Streams endpoint in the same Region
- DynamoDB Streams can be combined together with AWS Lambda to create a trigger that would be automatically executed in response to an event of interest that appears in a stream
- You can enable Streams using AWS Management console, CLI, or the SDK of choice

1.20 Scheduled Events in EC2

- You can schedule certain events to happen to perform certain actions on your resources, e.g.:
  - EC2 instances can be configured to respond to these scheduled events: Stop, Reboot, Retire, Maintenance



Notes:

Certain events apply only to instances backed by Amazon EBS.

1.21 Scheduled Events in CloudWatch

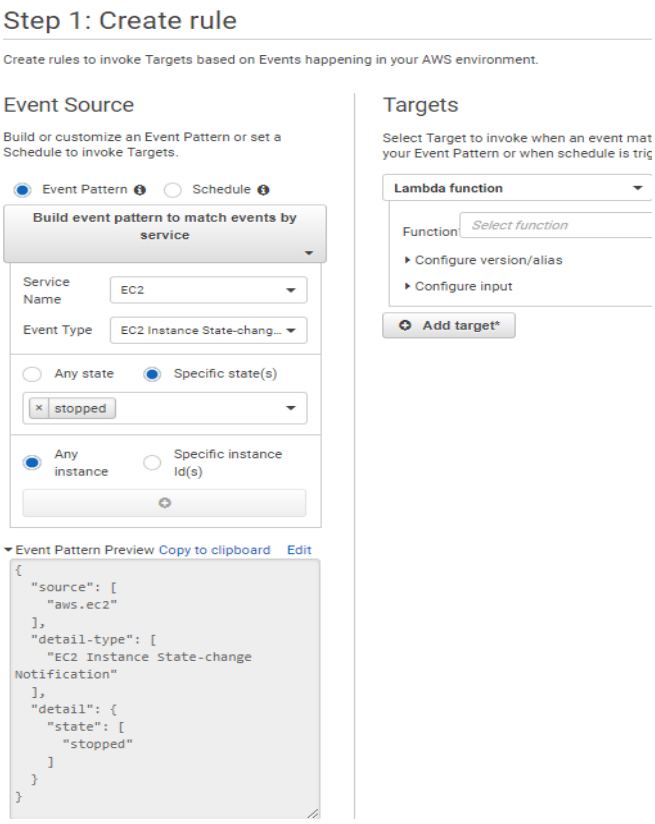
- CloudWatch offers users self-trigger rules that can be scheduled, with a minute precision, using *cron* or *rate* expressions
- The following AWS CLI command will run MyRule rule every day at 12:00pm UTC:

```
aws events put-rule \
     --schedule-expression "cron(0 12 * * ? *)" --name
MyRule
```

  - This is the self-explanatory *rate* rule example:

```
aws events put-rule \
    --schedule-expression "rate(15 minutes)" --name MyRule
```

1.22 CloudWatch Rules Creation in the AWS Management Console



- The rule configuration page is available on the CloudWatch dashboard → Events → Rules

1.23 What is AWS Lambda?

- AWS Lambda lets you run your code written in a number of supported languages in a PaaS-like environment with zero administration on your part
  - This arrangement is referred to as Serverless Computing or NoOps
- AWS provisions, scales, and manages servers needed to run your code, referred to as the *Lambda function*
  - You pay only for the actual time your Lambda function runs
    - You are charged in 0.1 second increments multiplied by the number of Lambda function invocations
  - You are only required to provide the amount of computing memory (the default is 128MB)
    - CPU is allocated in proportion to the requested memory

Notes:

Requesting 256MB of memory for your Lambda function allocates approximately twice as much computing power as in case of the default 128MB of memory.

1.24 What can I do with Lambda Functions

- It may help to think of Lambda functions as UPDATE / INSERT / DELETE triggers in RDBMSes, where a relational table where a record is updated, inserted, etc., and a trigger is attached to, acts as a source of events
- A Lambda Function runs in response to changes (modeled as events) in other AWS services (sources of those events), for example:
  - An object is uploaded to S3

- A DynamoDB item has been updated
- See more examples in a few slides …

1.25 Supported Languages

- As of 2019, Lambda gives you a choice of the following languages:
  - Node.js
  - Python
  - Java 8
    - Lambda provides the Amazon Linux build of openjdk 1.8
  - Go
  - C# (.NET Core)
    - Distributed as a NuGet package with the “dotnetcore1.0” runtime parameter
  - Ruby
- Lambda functions can used dependent libraries, including native ones
  - There is no white-list APIs

1.26 Lambda Constraints and Security

- There are a few activities that are disabled in Lambda functions:
  - Inbound network connections are blocked
  - Outbound connections only support TCP/IP sockets
  - ptrace (debugging) system calls are restricted
  - TCP traffic on port 25 is restricted as an anti-spam measure

1.27 Getting Your Code up and Running in Lambda

- You have three options:
  1. Create your Lambda function code inside the AWS Management Console (the in-line option)
  2. Develop code locally, build a ZIP or JAR bundle with all the dependencies, and upload it to AWS
  3. Upload your ZIP or JAR to S3

Notes:

According to Lambda documentation:"

Uploads must be no larger than 50MB (compressed). You can use the AWS Eclipse plugin to author and deploy Lambda functions in Java. You can use the Visual Studio plugin to author and deploy Lambda functions in C#, and Node.js.

...

You can define Environment Variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code. Learn more. For storing sensitive information, we recommend encrypting values using KMS and the console's encryption helpers."

1.28 Examples of the Base Lambda Function

Target Runtime	Lambda Function Code

Node.js (ECMAScript 2015)	`exports.handler = (event, context, callback) => {`
	`  callback(null, 'Hello from Lambda');`
	`};`

Python	`def lambda_handler(event, context):` `  return 'Hello from Lambda'`

1.29 Lambda Functions Use Cases

- High throughput real-time workflows handling billions of events per day
- Back-ends for you apps
- Media objects transcoding
- Real-time tracking of calls made to any Amazon Web Service from your app
- Front-end HTTP service
  - You can invoke your Lambda function via an HTTP endpoint using Amazon API Gateway

1.30 How It Works

- AWS Lambda functions run inside a default AWS-managed VPC on the computing infrastructure provided by AWS
  - Optionally, you can configure Lambda to run within your custom VPC
- It is highly recommended you write your code in a “stateless” style
  - Any state you may want to retain beyond the lifetime of the request

should be externalized to an AWS persistent store, e.g. Amazon S3, Amazon DynamoDB, etc.

- You can configure triggers for the Lambda, such as uploading a file on S3 that will cause the Lambda to be executed
- You can also invoke lambda functions directly using the AWS SDKs or AWS Mobile SDKs, such as the AWS Mobile SDK for Android
- Lambda provides a great foundation for building microservices

Notes:

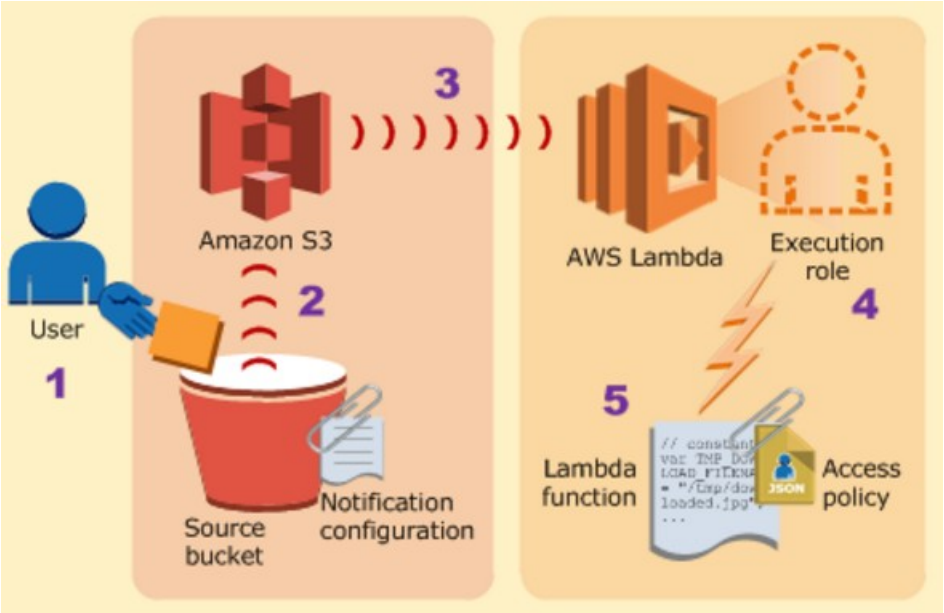
According to Lambda documentation:

"

To improve performance, AWS Lambda may choose to retain an instance of your function and reuse it to serve a subsequent request, rather than creating a new copy. Your code should not assume that this will always happen.

"

1.31 Example: Processing S3 Source Events with Lambda



- Source: AWS Documentation

1.32 The Programming Model

- The following core concepts apply to Lambda functions created in any of the supported languages:
  - **Handler** - The user call-back function invoked by AWS Lambda in response to a registered event; this function is passed the event object and the context object
  - **The context object** - The AWS Lambda object that provides information of the call context
  - **Logging** - Any user Lambda function can contain language-specific logging statements output of which AWS redirects to CloudWatch Logs

Notes:

According to Lambda documentation:

"

AWS Lambda provides this information via the context object:

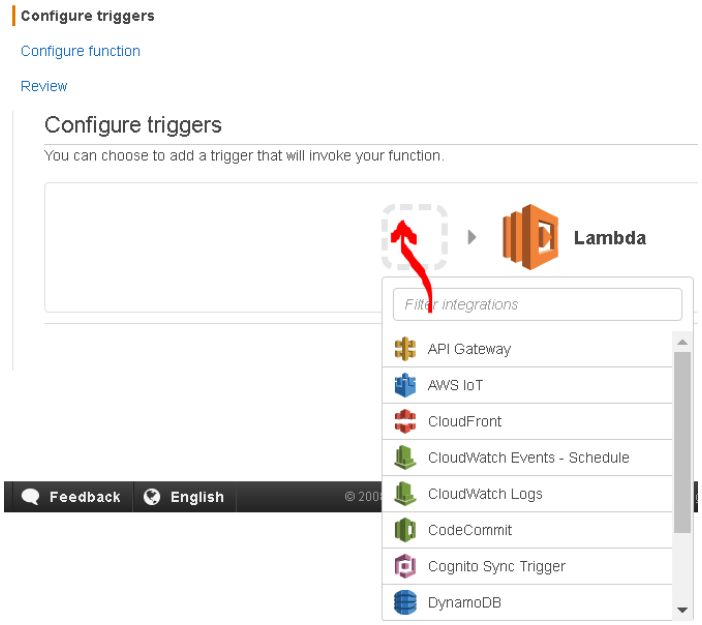
- *How much time is remaining before AWS Lambda terminates your Lambda function (timeout is one of the Lambda function configuration properties).*
- *The CloudWatch log group and log stream associated with the Lambda function that is executing.*
- *The AWS request ID returned to the client that invoked the Lambda function. You can use the request ID for any follow up inquiry with AWS support.*
- *If the Lambda function is invoked through AWS Mobile SDK, you can learn more about the mobile application calling the Lambda function.*

"

1.33 Configuring Lambda Functions

- The Lambda Dashboard of the AWS Management Console offers you a wizard-like Lambda function creation flow, where you need:
  - Specify the source of events to which your lambda function will be triggered to respond
  - Provide the function body written in the language of your choice
    - Write code in-line or upload the ZIP file containing your code
  - Specify the role under which your code will be running
  - Optionally, provide memory size (default is 128 MB), the call timeout (a value between the default 3 seconds and the maximum of 5 minutes), and configuration key-value pairs

1.34 Configure Triggers Page



1.35 Lambda Function Blueprints

- You have an option to either create a Lambda function from scratch, or use Lambda blueprints
- A Lambda blueprint is a sample configuration of an event source and a skeletal Lambda function for handling such events
- As of 2017, Lambda offers 78 blueprints, such as
  - ***dynamodb-process-stream***
    - ✔ "*An Amazon DynamoDB trigger that logs the updates made to a table*"
  - ***lex-book-trip-python***

- ✔ "*Book details of a visit, using Amazon Lex to perform natural language understanding*"
- ◇ ***microservice-http-endpoint***
  - ✔ "*A simple backend (read/write to DynamoDB) with a RESTful API endpoint using Amazon API Gateway*"

1.36 How Do I Troubleshoot and Monitor My Lambda Functions?

- AWS Lambda automatically monitors your Lambda functions reporting the following real-time metrics through Amazon CloudWatch:
  - ◇ Invocation count
  - ◇ Invocation duration
  - ◇ Invocation errors
- You can use these metrics to set CloudWatch custom alarms
- The CloudWatch Logs group associated with your Lambda function can be accessed by this logical path: */aws/lambda/<your function name>*

1.37 Developing Lambda in Java

- You can create a Lambda in a plain Maven project or use the AWS Toolkit plugin for Eclipse. If you are using Maven, add dependency for *com.amazonaws:aws-lambda-java-core*
- Develop POJO classes that will carry request and response data
- Develop the Lambda class that implements

```
public class MyLambda
  implements RequestHandler<MyRequest, MyResponse> {

  public MyResponse handleRequest(MyRequest request,
Context context) {
    //...
  }

}
```

- Build a JAR file that includes all the dependency classes and upload it in for your Lambda

1.38 Summary

- In this chapter, we reviewed the elements of Event-Driven computing in AWS and AWS Lambda service that provides an excellent platform for building microservices in the Amazon Cloud