AWS Lambda

* AWS Lambda offers Serverless computing
* Serverless computing allows applications and services to be built and run without thinking about servers. With serverless computing, application still runs on servers, but all the server management is done by AWS.
* Lambda lets you run code without provisioning or managing servers, where you pay only for the compute time when the code is running.
* Lambda is priced on a pay per use basis and there are no charges when the code is not running
* Lambda allows running of code for any type of application or backend service with zero administration
* Lambda performs all the operational and administrative activities on your behalf, including capacity provisioning, monitoring fleet health, applying security patches to the underlying compute resources, deploying code, running a web service front end, and monitoring and logging the code.
* Lambda does not provide access to the underlying compute infrastructure
* Scalability and availability
  + Lambda provides easy scaling and high availability to the code without additional effort on your part.
  + Lambda is designed to process events within milliseconds.
  + Latency will be higher immediately after a Lambda function is created, updated, or if it has not been used recently.
  + Lambda is designed to run many instances of the functions in parallel
  + Lambda is designed to use replication and redundancy to provide high availability for both the service and the Lambda functions it operates.
  + There are no maintenance windows or scheduled downtimes for either
  + For any Lambda function updates, there is a brief window of time, less then a minute, when requests would be served by both the versions
  + Lambda has a default safety throttle for number of concurrent executions per account per region
* Security
  + Lambda stores code in S3 and encrypts it at rest and performs additional integrity checks while the code is in use.
  + Each AWS Lambda function runs in its own isolated environment, with its own resources and file system view
* All calls made to AWS Lambda must complete execution within  900 seconds. Default timeout is 3 seconds. Timeout can be set the timeout to any value between 1 and  900 seconds.
* AWS Step Functions can help coordinate a series of Lambda functions in a specific order. Multiple Lambda functions can be invoked sequentially, passing the output of one to the other, and/or in parallel, while the state is being maintain by Step Functions.
* AWS X-Ray helps tracing for Lambda functions, which provides insights such as Lambda service overhead, function init time, and function execution time

## Lambda Functions & Event Sources

Core components of Lambda are Lambda functions and event sources.

* Event source is an AWS service or custom application that publishes events
* Lambda function is the custom code that processes the events

### Lambda Functions

* Each Lambda function has associated configuration information, such as its name, description, entry point, and resource requirements
* Lambda may choose to retain an instance of the function and reuse it to serve a subsequent request, rather than creating a new copy
* **Each Lambda function receives 500MB of non-persistent disk space in its own /tmp directory.**
* Design Lambda function as **stateless**
  + Lambda functions should be stateless, to allow AWS Lambda launch as many copies of the function as needed as per the demand
  + Local file system access, child processes, and similar artifacts may not extend beyond the lifetime of the request
  + State can be maintained externally in DynamoDB or S3
* Lambda function can be granted permissions to access other resources using an IAM role
* Lambda functions have the following restrictions
  + Inbound network connections are blocked by AWS Lambda
  + Outbound connections only TCP/IP sockets are supported
  + ptrace (debugging) system calls are blocked
  + TCP port 25 traffic is also blocked as an anti-spam measure.
* Lambda automatically monitors functions, reporting real-time metrics through CloudWatch, including total requests, latency, error rates, and throttled requests
* Lambda automatically integrates with CloudWatch logs, creating a log group for each function and providing basic application lifecycle event log entries, including logging the resources consumed for each use of that function
* Lambda functions supports code written in
  + Node.js (JavaScript)
  + Python
  + Java (Java 8 compatible)
  + C# (.NET Core)
  + Go
* Security
  + For sensitive information, for e.g. passwords, AWS recommends using client-side encryption using AWS Key Management Service and store the resulting values as ciphertext in your environment variable.
  + Lambda function code should include the logic to decrypt these values
* Versioning
  + Each AWS Lambda function has a single, current version of the code and there is no versioning of the same function.
  + **Each Lambda function version has a unique ARN and after it is published it is immutable**
  + **Versioning can be implemented using**[**Aliases**](http://docs.aws.amazon.com/lambda/latest/dg/versioning-aliases.html)**.**
  + Lambda supports creating aliases, which are mutable, for each Lambda function versions
  + **Alias is a pointer to a specific function version, with unique ARN**
  + **Each alias maintains an ARN for a function version to which it points**
  + **An alias can only point to a function version, not to another alias**
  + **Alias helps in rolling out new changes or rolling back to old versions**

A picture containing graphical user interface

Description automatically generated

Graphical user interface, website

Description automatically generated

Text

Description automatically generated

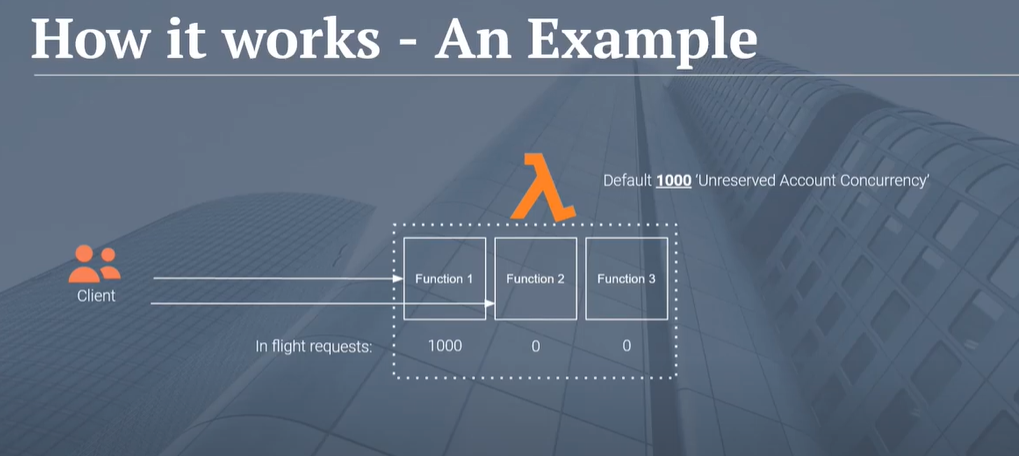
Graphical user interface

Description automatically generated with medium confidence

* Failure Handling
  + For S3 bucket notifications and custom events, Lambda will attempt execution of the function three times in the event of an error condition in the code or if a service or resource limit is exceeded
  + For ordered event sources, for e.g. DynamoDB Streams and Kinesis streams, that Lambda polls, it will continue attempting execution in the event of a developer code error until the data expires.
  + Kinesis and DynamoDB Streams retain data for a minimum of 24 hours
  + Dead Letter Queues can be configured for events to be placed, once the retry policy for asynchronous invocations is exceeded

### Lambda Throttling

* Intentional rejection of request by lambda.
* Throttling occurs when your concurrent execution count increases your concurrency limit.
* Lambda has a default concurrency limit of 1000 request/sec per region across account.
* Example : Since function-1 is having 1000 request/second, other functions (like function-2 & function-3) will get “too many request (status code - 429)” exception unless function-1 traffic goes down.



### 

Text

Description automatically generated

### Lambda Event Sources

# Lambda Execution Model

* When AWS Lambda executes the Lambda function, it takes care of provisioning and managing resources needed to run the Lambda function.
* When a Lambda function is invoked for the first time or after it has been updated there is a latency for bootstrapping as Lambda tries to reuse the Execution Context for subsequent invocations of the Lambda function
* When a Lambda function is invoked, Lambda launches an Execution Context based on the provided configuration settings i.e. memory and execution time
* After a Lambda function is executed, Lambda maintains the Execution Context for some time in anticipation subsequent function invocation
* Execution Context is a temporary runtime environment that initializes any external dependencies of the Lambda function code, for e.g. database connections or HTTP endpoints.
* Subsequent invocations perform better performance as there is no need to “cold-start” or initialize those external dependencies
* Lambda manages Execution Context creations and deletion, there is no AWS Lambda API to manage Execution Context.

## Lambda in VPC

* Lambda function can be configured to connect to private subnets in a virtual private cloud (VPC) in the AWS account.
* Lambda runs the function code securely within a VPC by default.
* Lambda function connected to VPC can access private resources databases, cache instances, or internal services during execution
* To enable the Lambda function to access resources inside the private VPC, additional VPC-specific configuration information that includes private subnet IDs and security group IDs must be provided.
* AWS Lambda uses this information to set up elastic network interfaces (ENIs) that enable the function to connect securely to other resources within your private VPC.
* Lambda functions connected to VPC need a NAT Gateway to access any external resources outside of AWS.
* Lambda functions cannot connect directly to a VPC with dedicated instance tenancy, instead peer it to a second VPC with default tenancy.

## Lambda@Edge

* Lambda@Edge allows running of code across AWS locations globally without provisioning or managing servers, responding to end users at the lowest network latency
* Lambda function can be configured to be triggered in response to CloudFront requests, which includes
  + Viewer Request – event occurs when an end user or a device on the Internet makes an HTTP(S) request to CloudFront, and the request arrives at the edge location closest to that user.
  + Viewer Response – event occurs when the CloudFront server at the edge is ready to respond to the end user or the device that made the request
  + Origin Request – event occurs when the CloudFront edge server does not already have the requested object cached, and the viewer request is ready to be sent to backend origin webserver
  + Origin Response – event occurs when the CloudFront server at the edge receives a response from your backend origin webserver.
* Lambda function executes across AWS locations globally when a request for content is received, and scales with the volume of CloudFront requests globally
* Lambda@Edge only supports Node.js for global invocation by CloudFront events at this time

## Lambda Best Practices

* **Lambda function code should be stateless, and ensure there is no affinity between the code and the underlying compute infrastructure.**
* Instantiate AWS clients outside the scope of the handler to take advantage of connection re-use.
* Make sure you have set +rx permissions on your files in the uploaded ZIP to ensure Lambda can execute code on your behalf.
* Lower costs and improve performance by minimizing the use of startup code not directly related to processing the current event.
* Use the built-in CloudWatch monitoring of your Lambda functions to view and optimize request latencies.
* Delete old Lambda functions that you are no longer using.

## AWS Lambda Event Source

* Core components of Lambda are Lambda functions and event sources.
  + An AWS Lambda event source is the AWS service or custom application that publishes events
  + Lambda function is the custom code that processes the events
* An event source is an AWS service or developer-created application that produces events that trigger an AWS Lambda function to run
* Event sources can be either AWS Services or Custom applications
* Event sources can be both push and pull sources
  + Services like S3, SNS publish events to Lambda by invoking the cloud function directly
  + Lambda can also poll resources in services like Kinesis streams that do not publish events to Lambda
* Events are passed to a Lambda function as an event input parameter. For batch event sources, such as Kinesis Streams, the event parameter may contain multiple events in a single call, based on the requested batch size

## Lambda Event Source Mapping

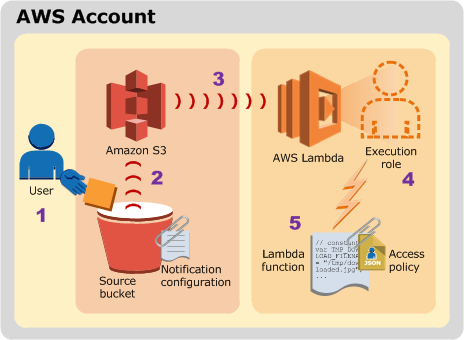
* Lambda Event source mapping refers to the configuration which maps an event source to a Lambda function.
* Event source mapping enables automatic invocation of the Lambda function when events occur.
* Each event source mapping identifies the type of events to publish and the Lambda function to invoke when events occur
* AWS supported event sources can grouped into
  + Regular AWS services
    - also referred to as Push model
    - includes services like S3, SNS, SES etc.
    - event source mapping maintained on their side
    - as the event sources invoke the Lambda function, resource-based policy should be used to grant the event source necessary permissions
  + Stream-based event sources
    - also referred to as Pull model
    - includes services like DynamoDB & Kinesis streams
    - need to have the event source mapping maintained on the Lambda side

## Lambda Supported Event Sources

AWS Lambda can be configured as an event source for multiple AWS services

### ****Amazon S3****

* S3 bucket events, such as the object-created or object-deleted events can be processed using Lambda functions for e.g., Lambda function can be invoke when a user uploads a photo to a bucket to read the image and create a thumbnail
* S3 bucket notification configuration feature can be configured for the event source mapping, to identify the S3 bucket events and the Lambda function to invoke.
* Error handling for a event source depends on how Lambda is invoked
* S3 invokes your Lambda function **asynchronously**.

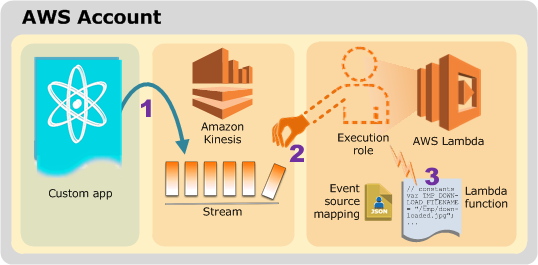


### ****DynamoDB****

* Lambda functions can be used as triggers for DynamoDB table to take custom actions in response to updates made to the DynamoDB table.
* **Trigger can be created by**
  + **First enabling DynamoDB Streams for the table.**
  + **Lambda polls the stream and processes any updates published to the stream**
* DynamoDB is a stream-based event source and with stream based service, the event source mapping is created in Lambda, identifying the stream to poll and which Lambda function to invoke.
* Error handling for a event source depends on how Lambda is invoked

**Kinesis Streams**

* AWS Lambda can be configured to automatically poll the Kinesis stream periodically (once per second) for new records
* Lambda can process any new records such as social media feeds, IT logs, website click streams, financial transactions, and location-tracking events
* Kinesis Streams is a stream-based event source and with stream based service, the event source mapping is created in Lambda, identifying the stream to poll and which Lambda function to invoke.
* Error handling for a event source depends on how Lambda is invoked



### ****Simple Notification Service****

* SNS notifications can be process using Lambda
* When a message is published to an SNS topic, the service can invoke Lambda function by passing the message payload as parameter, which can then process the event
* Lambda function can be triggered in response to CloudWatch alarms and other AWS services that use SNS.
* SNS via topic subscription configuration feature can be used for the event source mapping, to identify the SNS topic and the Lambda function to invoke
* Error handling for a event source depends on how Lambda is invoked
* SNS invokes your Lambda function **asynchronously**.

### ****Simple Email Service****

* SES can be used to receive messages and can be configured to invoke Lambda function when messages arrive, by passing in the incoming email event as parameter
* SES using the rule configuration feature can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* SES invokes your Lambda function **asynchronously**.

### ****Amazon Cognito****

* Cognito Events feature enables Lambda function to run in response to events in Cognito for e.g. Lambda function can be invoked for the Sync Trigger events, that is published each time a dataset is synchronized.
* Cognito event subscription configuration feature can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* Cognito is configured to invoke a Lambda function **synchronously**

### ****CloudFormation****

* Lambda function can be specified as a custom resource to execute any custom commands as a part of deploying CloudFormation stacks and can be invoked whenever the stacks are created, updated or deleted.
* CloudFormation using stack definition can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* CloudFormation invokes the Lambda function **asynchronously**

### ****CloudWatch Logs****

* Lambda functions can be used to perform custom analysis on CloudWatch Logs using CloudWatch Logs subscriptions.
* CloudWatch Logs subscriptions provide access to a real-time feed of log events from CloudWatch Logs and deliver it to the AWS Lambda function for custom processing, analysis, or loading to other systems.
* CloudWatch Logs using the log subscription configuration can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* CloudWatch Logs invokes the Lambda function **asynchronously**

### ****CloudWatch Events****

* CloudWatch Events help respond to state changes in the AWS resources. When the resources change state, they automatically send events into an event stream.
* Rules that match selected events in the stream can be created to route them to the Lambda function to take action for e.g., Lambda function can be invoked to log the state of an EC2 instance or AutoScaling Group
* CloudWatch Events by using a rule target definition can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* CloudWatch Events invokes the Lambda function **asynchronously**

### ****CodeCommit****

* Trigger can be created for an CodeCommit repository so that events in the repository will invoke a Lambda function for e.g., Lambda function can be invoked when a branch or tag is created or when a push is made to an existing branch.
* CodeCommit by using a repository trigger can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* CodeCommit Events invokes the Lambda function **asynchronously**

### ****Scheduled Events (powered by CloudWatch Events)****

* AWS Lambda can be invoke regularly on a scheduled basis using the schedule event capability in CloudWatch Events.
* CloudWatch Events by using a rule target definition can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* CloudWatch Events invokes the Lambda function **asynchronously**

### ****AWS Config****

* Lambda functions can be used to evaluate whether the AWS resource configurations comply with custom Config rules.
* As resources are created, deleted, or changed, AWS Config records these changes and sends the information to the Lambda functions, which can then evaluate the changes and report results to AWS Config. AWS Config can be used to assess overall resource compliance
* AWS Config by using a rule target definition can be used for the event source mapping
* Error handling for a event source depends on how Lambda is invoked
* AWS Config invokes the Lambda function **asynchronously**

### ****Amazon API Gateway****

* Lambda function can be invoked over HTTPS by defining a custom REST API and endpoint using Amazon API Gateway.
* Individual API operations, such as GET and PUT, can be mapped to specific Lambda functions. When a HTTPS request to the API endpoint is received, the API Gateway service invokes the corresponding Lambda function.
* Error handling for a event source depends on how Lambda is invoked
* API Gateway is configured to invoke a Lambda function **synchronously**.

### ****Other Event Sources: Invoking a Lambda Function On Demand****

* Lambda functions can be invoked on demand without the need to preconfigure any event source mapping in this case.

## AWS Certification Exam Practice Questions

1. Your serverless architecture using AWS API Gateway, AWS Lambda, and AWS DynamoDB experienced a large increase in traffic to a sustained 400 requests per second, and dramatically increased in failure rates. Your requests, during normal operation, last 500 milliseconds on average. Your DynamoDB table did not exceed 50% of provisioned throughput, and Table primary keys are designed correctly. What is the most likely issue?
   1. Your API Gateway deployment is throttling your requests.
   2. Your AWS API Gateway Deployment is bottlenecking on request (de)serialization.
   3. **You did not request a limit increase on concurrent Lambda function executions.**(Refer [link](http://docs.aws.amazon.com/general/latest/gr/aws_service_limits.html#limits_lambda) – AWS API Gateway by default throttles at 500 requests per second steady-state, and 1000 requests per second at spike. Lambda, by default, throttles at 100 concurrent requests for safety. At 500 milliseconds (half of a second) per request, you can expect to support 200 requests per second at 100 concurrency. This is less than the 400 requests per second your system now requires. Make a limit increase request via the AWS Support Console.)
   4. You used Consistent Read requests on DynamoDB and are experiencing semaphore lock.