AWS Lambda:

- AWS lambda is a compute service that lets you run the code without provisioning or managing servers.
- With AWS lambda, you can run code for virtually any type of application or backend service- all with zero administration.
- AWS lambda manages all the administration it manages:
    - Provisioning and capacity of the compute fleet that offers a balance of memory, CPU, network, and other resources.
    - Server and O.S maintenance
    - High availability and automatic scaling
    - Monitoring fleet health
    - Applying security patches
    - Deploying your code
    - Monitoring and logging your lambda functions.
    - AWS lambda runs your code on a high-availability compute infrastructure.
- AWS lambda runs your code on a high availability compute infrastructure.
- AWS lambda executes your code only when needed and scales automatically form a few requests per day to thousands per seconds.
- You pay only for the compute time; you consume no charge when your code is not running.
- All you need to do is supply your code in the form of one or more lambda functions to AWS lambda, in one of the languages that AWS supports (currently Node.js, java, powershell, C#, Ruby, Python and Go) and the service can run the code on your behalf.
- Typically, the lifecycle for an AWS lambda-based application includes authoring code, deploying code to AWS lambda and then monitoring and troubleshooting.
- This is in exchange for flexibility, which means you cannot log into compute instances or customize the operating system of language runtime.
- If you do want to manage your own compute, you can use EC2 of Elastic Beanstalk.

How Lambda Works:

- First you upload your code to lambda in one or more lambda function.
- AWS lambda will then execute the code in your behalf.
- After the code is invoked, lambda automatically take care of provisioning and managing the required servers.

Difference between AWS Lambda and EC2:

AWS Lambda:

- AWS lambda is a platform-as-a-service.
- It supports only limited languages like Node.js, python, java, C#, Ruby, Go and powershell.
- Write your code and push the code into AWS lambda.
- You cannot log into compute instances, choose customized O.S or language platform.

AWS EC2:

- AWS EC2 is an infrastructure—as-a-service.
- No environment restrictions, you can run any code or language.
- For the first time in EC2, you have to choose the O.S and install all the software required and then push your code in EC2.
- You can select variety of O.S, instance types, network and security patches, RAM and CPU etc.

Important Terms Used in Lambda:

a. **Function:** a function is a resource that you can invoke to run your code in AWS lambda. A function has code that processes events and a runtime that passes request and responses between lambda and the function code.

b. **Runtime:** lambda runtime allows functions in different languages to run in the same base execution environment. The runtime sits in between the lambda service and your function code relying invocation events, context information and responses between the two.

c. **Event:** it is a JSON formatted document that contains data for a function to process.

d. **Event Source/ Trigger:** an AWS service such as Amazon SNS or a custom service that triggers your function and executes its logic.

e. **Downstream Resource:** an AWS service, such as Dynamo DB tables or S3 buckets that your lambda function call once it is triggered.

f. **Concurrency:** number of request that your function is serving in any given time.

When Lambda Triggers:

- You can use AWS lambda to run your code in response to:
  - Events such as changes to data in an Amazon S3 or an Amazon Dynamo DB table.
  - To run your code in response to HTTP request using Amazon API gateway.
  - With the capabilities you can use lambda to easily build data processing triggers for AWS services like Amazon S3, and Amazon Dynamo DB process streaming data stored in kinesis or create your own backend that operates at AWS scale, performance, and security.

Example of S3:

- The user creates an object in a bucket.
- Amazon S3 invokes your lambda functions using the permission provided by the execution role.
- Amazon S3 knows which lambda function to invoke based on the event source mapping that is stored in the bucket notification configuration.

AWS Lambda Function Configuration:

- A lambda function consists of code and any associated dependencies.
- In addition, a lambda function also has configuration information associated with it.
- Initially you specify the configuration information when you create a lambda function.
- Lambda provides an API for you to update some of the configuration data.

Lambda function configuration information includes the following key elements:

- Compute resources that you need you only specify the amount of memory you want to allocate from your lambda function.
- AWS lambda allocates CPU power proportional to the memory by

using the same ratio as a general-purpose Amazon EC2 instance type, such as an M3 type.

- You can update the configuration and request additional memory in 64mb increments from 128mb to 3008mb.
- Functions larger than 1536mb are allocated multiple CPU threads.

Maximum Execution Timeout:

- You pay for the AWS resources that are used to run your lambda function.
- To prevent your lambda function from running indefinitely, you specify a timeout.
- When the specified timeout is reached, AWS lambda terminates your lambda function.
- Default is 3sec and maximum is 9000sec (15min).

- **IAM ROLE:** this is the role that AWS lambda assumes when it executes the lambda function on your behalf.

AWS Lambda Function- Services it can access:

- Lambda function can access:
  - AWS services and non-AWS services.
  - AWS services running in AWS VPC (e.g: redshift, elastic cache, RDS instance)
  - Non-AWS services running on instances in an AWS VPC.
  - AWS lambda run your function code securely with in a VPC by default.
  - However to enable your lambda function to access resources inside your private VPC you must provide additional VPC specific configuration information that includes VPC subnet ID and Security group IDs

Different way to invoke Lambda Function:

1. Synchronous invoke (push)
2. Asynchronous invoke (event)
3. Poll-based invoke (pull based)

1. **Synchronous Invoke:**

- Synchronous invoke are the most straight forward way to invoke your

lambda function. In this model your functions execute immediately when you perform the lambda invoke API call.

- Invocation flag specifies a value of "request-response".
- You wait for the function to process the event and

return a response. Here is a list of services that invoke

lambda function synchronously:

- Elastic Load Balancer
- Amazon Cognito
- Cloud front
- API Gateway
- Amazon Lex
- Kinesis data firehose

2. **Asynchronous Invoke:**

   - For asynchronous invocation, lambda places the event in a queue and returns a success response without additional information.
   - Lambda queues the event for processing and returns a response immediately.

   - You can configure lambda to send an invocation record to another service like SQS, SNS, lambda and eventbridge.

Here is a list of services that invoke lambda function asynchronously:

- Amazon S3
- Amazon SNS
- SES
- Cloud watch logs
- Cloud watch events
- AWS code commit
- AWS config

3. **Poll-Based Invoke:**
   The invocation model is designed to allow you to integrate with AWS stream and queue-based service with no code or server management lambda will poll the following service on your behalf, retrieve records and invoke your function. The following are supported service:
      - Amazon Kinesis , SQS, DynamoDB Stream

SIMPLE QUEUE SERVICE (SQS)

- SQS is a fast, reliable, fully managed message queue service.
- It is a web service that gives you access to message queue that stores messages waiting to be processed.
- It offers a reliable highly scalable, hosted queue for storing messages between servers.
- It allows the decoupling of application components such that a failure in one component doesn't cause a bigger problem to application functionality. (like in coupled application)
- Using SQS, you no longer need a highly available message cluster or the burden of running it.
- You can delete all the messages in an SQS queue without deleting the SQS Queue itself.
- You can use applications on EC2 instances to read and process the SQS queue message.
- You can use auto scaling to scale the EC2 fleet processing the SQS messages, as the queue size increases.
- These applications on EC2 instances can process the SQS message/ jobs then post the SQS results to other SQS queues or other AWS service.

Types of Amazon Queue Services:

It is of two types such as:

1. Standard Queue
2. FIFO Queue

1. **Standard Queue:**
   - High Throughput
   - At Least One Delivery
   - Duplicacy is possible
   - Best effort ordering

2. **FIFO Queue:**
   - Limited throughput (300 TPS)
   - Exactly one processing
   - Duplicacy not possible
   - Strict ordering: first-in-first-out
   - FIFO queue are limited to 300 transactions per second (TPS),

but have all the capabilities of standard queue.

SQS Pricing:

- The first 1 million monthly requests are free, after that pricing is according to regions.
- For e.g: in Mumbai region:
  - Standard Queue- $0.40/ million request
  - FIFO Queue- $0.50/ million request

How Amazon SQS charges:

1. API action: every Amazon SQS action count as request.
2. FIFO request: API actions for sending, receiving, deleting, and changing visibility of messages from FIFO queues are charged at FIFO rates.
3. Contents of Request: a single request can have from 1 to 10 messages, up to a maximum total payload of 256kb.
4. Size of Payload: each 64kn chunk of a payload is billed as 1 request. (for e.g: API action with a 256kb payload is billed as 4 request)
5. Interaction with Amazon S3.
6. Interaction with AWS KMS.

Short Polling:

- A request is returned immediately even if the queue is empty.
- It doesn't wait for message to appear in the queue.
- It queries only a subset of the available servers for messages (based on weighted random distribution)
- Default by SQS
- Receive message wait time is set to 0.
- More request is used which implies higher cost.

Long Polling:

- It is preferred to regular/ short polling. It uses fewer requests and request cost by eliminating false empty responses by querying all the servers.
- Reduce the number of empty responses by allowing Amazon SQS to wait until a message is available in the queue before sending a response, Unless the connection timeout (20sec)
- Receive message wait time is set to a non-zero value (max 20sec)

- Billing is same for both pollings.

SQS Retention Period:

- SQS messages can remain in the queue for up to 14 days. (SQS retention period)
- Range is 1 min to 14 days. (default is 4days)
- Once the maximum retention period of a message id reached, it will be deleted automatically from the queue.
- Messages can be sent to the queue and read from the queue simultaneously.
- SQS can be used with Dynamo DB, EC2, ECS, redshift, RDS, lambda, S3 to make distributed/ decoupled application.
- You can have multiple queues with different properties.

SQS Visibility Timeout:

- It is the **duration of time a message is locked for read** by other servers.
- Maximum is 12 hours and default is 30 sec.
- A server that read a message process it, can change the message visibility timeout if it needs more time to process the message.
- After a message is read, there are the following possibilities:
  a. An ACK is received that a message is processed, so it must be deleted from the queue to avoid duplicates.
  b. If a fail is received of the visibility timeout expires, the message will then be locked for read such that it can be read and processed by another servers.

Delivery Delay: AWS SQS provides delivery delay options to postpone the delivery of new messages to a queue. If delivery delay is defined for a queue, any new messages will not be visible to the server for the duration of delay. The default (min) delay for a queue is 0 seconds. The maximum is 15 minutes.

Receive Message Wait Time: the default time is 0 seconds. This is maximum amount of time that a long polling receive call will wait for a message to become available before returning an empty response (maximum value is 20sec).

Dead Letter Queue:

- The main task of a dead letter queue is handling message failure. A dead letter queue lets you to set aside and isolated message that can't be processed correctly to determine why their processing didn't success.
- Don't use a dead letter queue with a FIFO queue, if you don't want to break the exact order of messages or operations.
- DLQ must be of the same type as the source queue. (standard or FIFO)

SIMPLE NOTIFICATION SERVICE (SNS)

- SNS is a fast, flexible, fully managed PUSH notification service.
- It is a web service that delivery or sending of messages to subscribe endpoints or clients.
- It allows for sending individual messages of fan-out messages to many recipients or to other distributed AWS services.
- Messages published to an SNS topics will be delivered to the subscriber immediately.
- Inexpensive, pay-as-you-go model with no upfront cost.
- Reliable: at least three copies of the data are store across multiple AZ in same region.
- It is a way of sending messages. When you are using auto scaling, it triggers an SNS service which will email you that "your EC2 instance is growing".

Publisher: publishers are also known as produce and send the message to the SNS which is a logical access point.

Subscriber: subscribers such as web servers, email addresses, Amazon SQS queues, AWS lambda receive the message of notification from the SNS over one of the supported protocols (Amazon SQS, email, lambda, HTTPS, SMS).

SNS Topic:

- It is a logical access point and communication channel.
- Each topic has a unique name.
- A topic name is limited to 256 alphanumeric character.
- The topic name must be unique with in the AWS account.
- Each topic is assigned an AWS ARN once it gets created.
- A topic can support subscribers and notification deliveries over multiple protocols.
- Message/ request published to a single topic can be

delivered over multiple protocols as configured when creating each subscriber.

- Delivery format/ transport protocols (endpoints.), SMS, e-mail, email: JSON –for applications, HTTP/ HTTPS, SQS, AWS lambda.
- When using Amazon SNS, you (as the owner) create a topic and control access to it by defining access policies that determine which publishers and subscribers can communicate with the topic.
- Instead of including a specific destination address in each message to topics that they have created or to topics they have permission to publish to.
- Amazon SNS matches the topic to a list of subscribers who have subscribed to that topic and delivers the message to each of these subscribers.
- Each topic has a unique name that identifies the Amazon SNS endpoint for publishers to past message and subscribers to register for notification.
- Subscriber receive all messages published to the topics to which they subscribe, and all subscribers to a topic receive the same message.
- By default, only the topic owner (who created it) can publish to the SNS topic.
- The owner can set/ change permission to one of more users (with valid AWS ID) to publish to his topic.
- Only the owner of the topic can grant/ change permission for the topic.
- Subscriber can be those with/ without AWS ID, only subscriber with AWS ID can request subscription.
- Both publishers and subscriber can use SSL to help secure the channel to send and receive message.
- Supported push notification platforms:
    - Amazon Device Messaging
    - Apple Push Notification Service.
    - Google Cloud Messaging
    - Windows Push Notification Service
    - Baidu Cloud Push for Android
- SNS topic can have subscribers from any supported push notification platforms as well as any other endpoint type

such as SMS or Email.

- When you publish a notification to a topic SNS will send identical copies of that, message to each endpoint subscribed to the topic.

Amazon SNS Alternatives:

a. Amazon Kinesis Data Stream
b. Amazon Managed Queue Service (AWS MQS)
c. Apache Kafka
d. Twilio
e. Pusher

Amazon SNS Pricing:

a. Publish action: each 64kb of request payload count as one request. So, 256kb payload will charged as four requests.
b. Mobile push notification: for e.g: $0.50/ million request
c. SMS: price depends on country
d. E-mail: $2/100,000
e. HTTP/s notification: $ 0.60/ million
f. SQS and lambda calls are free. These are charged at SQS and lambda rates.
g. Data Transfer

Amazon EventBridge

Amazon EventBridge:

- A serverless event bus service for Software-as-a-Service (SAAS) and AWS services.
- In simple words, Amazon EventBridge provides an easy solution to integrate SAAS, custom-build applications with more than 17+ AWS services with the delivery of real-time data from different event sources. Users can easily set up the routing rules to determine the target web-service, and multiple target locations (such as AWS Lambda or AWS SNS) can be selected at once.
- Amazon EventBridge is a fully managed service that takes care of event ingestion, delivery, security, authorization, error handling, and required infrastructure management tasks to set up and run a highly scalable serverless event bus.
- EventBridge was formerly called Amazon CloudWatch Events, and it uses the same CloudWatch Event API.

Key Concepts

1. **Event Buses:**

An event bus receives events. When a user creates a rule, which will be associated with a specific event bus, the rule matches only to the event received by the event bus. Each user's account has one default event bus, which receives events from AWS services. We can also create our custom event buses.

2. **Events**:

An event indicates a change in the environment. By creating rules, you can have AWS services that act automatically when changes occur in other AWS services, in SaaS applications, or user's custom applications.

3. **Shema Registry:**

A Schema Registry is a container for schemas. Schemas are available for the events for all AWS services on Amazon EventBridge. Users can always create or update their schemas or automatically infer schemas from events running on event buses. Each schema will have multiple versions. Users can use the latest schema or select earlie**r versions.**

4. **Rules:**

A rule matches incoming events and routes them to targets for processing. A single rule can route an event (JSON format) to multiple targets. All pointed targets will be processed in parallel and no particular order.

5. **Targets:**

A target processes events and receives events in JSON format. A rule's target must be in the same region as a rule.

Features:

- Fully managed, pay-as-you-go.
- Native integration with SaaS providers.
- 90+ AWS services as sources.
- 17 AWS services as targets.
- $1 per million events put into the bus.
- No additional cost for delivery.
- Multiple target locations for delivery.
- Easy to scale and manage.

AWS Step Functions

Step functions:
Step functions allow developers to offload application orchestration into fully managed AWS services. This means you can just modularize your code to “Steps” and let AWS worry about handling partial failure cases, retries, or error handling scenarios.
Types of step functions:
1. Standard workflow: Standard workflow can be used for long-running, durable, and auditable workflows.
2. Express Workflow: Express workflow is designed for high volume, and event processing workloads.
Features:
● Allow to create workflow which follows a fixed or dynamic sequence.
● Inbuilt “Retry” and error handling functionality.
● Support integration with AWS native Lambda, SNS, ECS, AWS Fargate
● Support GUI audit workflow process, input/output, etc., well.
● GUI provides support to analyse the running process and detect the failures immediately.
● High scalable and low cost.
● Manages the states of the application during workflow execution.
● Ability to deliver real-time execution with Amazon cloud watch and cloud train applications.
● High availability.
Best Practices:

- Set time-outs in state machine definitions, which help in better task response when something goes wrong in getting a response from an activity.

Example:
"ActivityState": {
"Type": "Task",
"Resource":
"arn:aws:states:us-east-1:123456789012:activity:abc",
"TimeoutSeconds": 900,
"HeartbeatSeconds": 40,
"Next": "State2"
}

- Always provide the Amazon S3 arn (amazon resource name) instead of large payloads to the state machine when passing input to Lambda function.

Example:
{
"Data": "**arn:aws:s3:::MyBucket/data.json**"
}

- Handle errors in state machines while invoking AWS lambda functions.

Example:
"Retry": [{
"**ErrorEquals": ["Lambda.CreditServiceException"]**
"IntervalSeconds": 2,
"MaxAttempts": 3,
120
"BackoffRate": 2
}]

- AWS step functions have a hard quota of 25K entries during execution history. To avoid this for long-running executions, implement a pattern using the AWS lambda function.

AWS Step function supports below AWS services:

- Lambda
- AWS Batch
- DynamoDB
- ECS/Fargate
- SNS
- SQS
- SageMaker
- EMR

Pricing:
With Step Functions Express Workflows, you pay only for what you use. You are charged based on the number of requests for your workflow and its duration.
○ $0.025 per 1,000 state transitions (For Standard workflows)
○ $1.00 per 1M requests (For Express workflows)
Note: There would be additional charging if your workflow utilizes other services such as data transfer or other AWS services during the operation.

Important Notes:
Step function is based on the concepts of tasks and state machines.

- Tasks can be defined by using an activity or an AWS Lambda function.
- State machines can express an algorithm that contains relations, input/output.
- State machines using JSON based Amazon States Language.
- AWS step functions had 99.9% SLA.
- There are six types of states in AWS Step functions, which are mentioned below:

Task State: work activity in the state machine, for example, AWS Lambda function.
○ Choice state.
○ Fail or succeed State.
○ Pass state.
○ Wait state.
○ Parallel state.

State machine data is represented by JSON text, which depicts the below structure:

- Initial input into the state machine.
- Data passed between states.
- Output passed to the next state.

AWS Kinesis

- AWS Kinesis Scalable streaming service. It is designed to inject data from lots of devices or lots of applications.
- Many producers send data into a Kinesis Stream.
- The stream can scale from low to near infinite data rates.
- Highly available public service by design.
- Streams store a 24-hour moving window of data.
    - Can be increased to 7 days.
    - Data 24 hours + 1s is replaced by new data entering the stream.
- Kinesis includes the storage costs within it for data that can be ingested during a 24-hour period. However, much you ingest for 24 hours, that's included.
- Multiple consumers can access data from that moving window.
    - One might look at data points once per hour
    - Another looks at data 1 per minute.
- Kinesis stream starts with 1 shard and expands as needed.
    - Each shard can have 1MB/s for ingestion and 2MB/s consumption.

Kinesis data records (1MB) are stored across shards and are the blocks of data for a stream.

Kinesis Data Firehose connects to a Kinesis stream. It can move the data from a stream onto S3 or another service.

SQS vs Kinesis:

AWS Kinesis	AWS SQS
1. Large throughput or large numbers of devices	1. 1 thing sending messages to the queue
2. Huge scale ingestion with multiple consumers	2. One consumption group from that tier
3. Rolling window for multiple consumers	3. Allow for async communications.

4. Designed for data ingestion, analytics, monitoring, app clicks	4. Once the message is processed, it is deleted