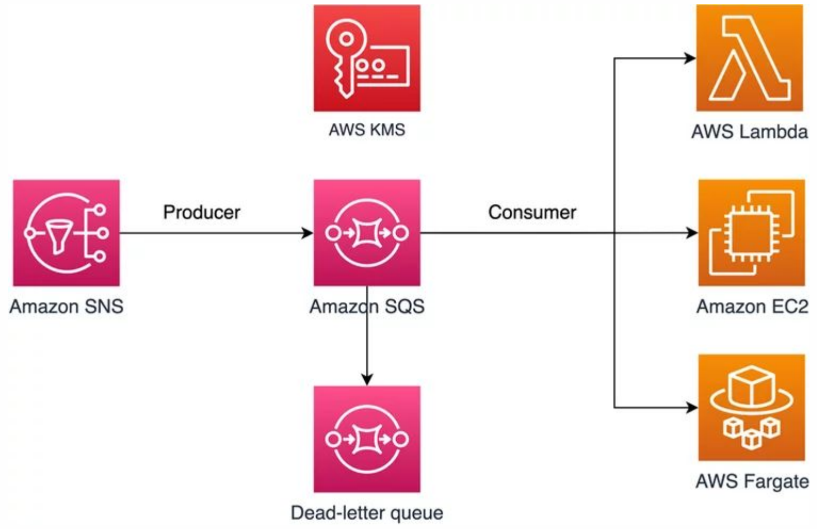
Amazon SQS

Amazon SQS (Simple Queue Service) is a fully managed message queuing service that enables decoupling of microservices, distributed systems, and serverless applications. Decoupling means that the producer of the message (e.g., an application or service) does not need to wait or depend on the consumer to be available. Messages are sent to a queue and stored until the consumer retrieves them. This increases fault tolerance and allows systems to scale independently.

Key Points:

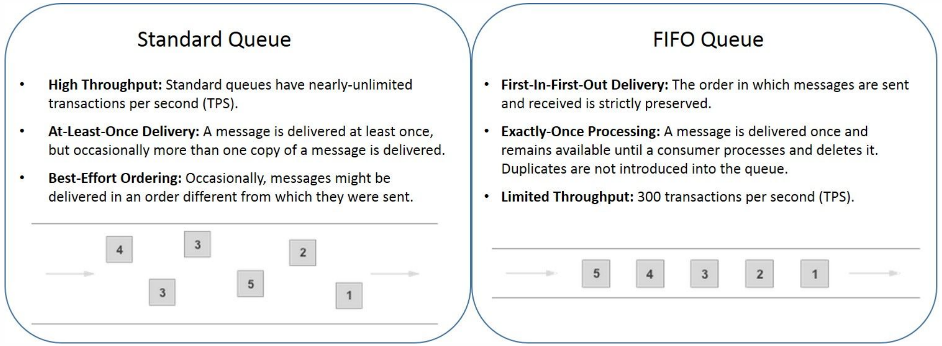
- Fully managed, serverless, and scalable.
- Improves resilience by decoupling components.
- Temporarily stores messages until processing.



SQS Standard + FIFO

1. **Standard Queue**
   - Best-effort ordering (messages may be delivered more than once and out of order).
   - High throughput – nearly unlimited number of transactions per second.
   - Use case: General-purpose apps that tolerate occasional duplicates (e.g., order processing, background tasks).

2. **FIFO Queue (First-In-First-Out)**
   - Strict ordering – messages are processed exactly once, in the order sent.
   - Lower throughput – 300 messages per second (or 3000 with batching).
   - Use case: Systems that require order and exactly-once processing (e.g., financial transactions, inventory updates).



Standard Queue

- **High Throughput:** Standard queues have nearly-unlimited transactions per second (TPS).
- **At-Least-Once Delivery:** A message is delivered at least once, but occasionally more than one copy of a message is delivered.
- **Best-Effort Ordering:** Occasionally, messages might be delivered in an order different from which they were sent.

FIFO Queue

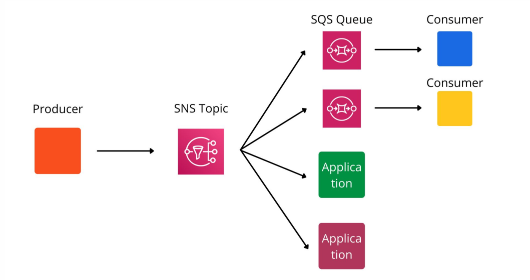
- **First-In-First-Out Delivery:** The order in which messages are sent and received is strictly preserved.
- **Exactly-Once Processing:** A message is delivered once and remains available until a consumer processes and deletes it. Duplicates are not introduced into the queue.
- **Limited Throughput:** 300 transactions per second (TPS).

Producing + Consuming from SQS

SQS is based on a polling model:

- Producing (Sending Messages):
  - Producers (applications or services) send messages to an SQS queue using the AWS SDK, CLI, or console.
  - Each message can be up to 256 KB in size.
  - Messages are retained for up to 14 days.

- Consuming (Receiving Messages):
  - Consumers poll the queue to receive messages (short or long polling).
  - Once a message is received, it's hidden temporarily (visibility timeout) to avoid being processed twice.
  - After processing, the consumer deletes the message from the queue.
  - If the message is not deleted before the visibility timeout expires, it becomes visible again for other consumers.



SQS Security

Amazon SQS provides multiple layers of security to ensure safe message handling:

1. Encryption
   - At rest: Messages are encrypted using AWS KMS (SSE-SQS).
   - In transit: Uses HTTPS to protect messages as they travel between producers, SQS, and consumers.

2. Access Control
   - IAM policies define who can send, receive, or delete messages.
   - You can create resource-based policies on queues to allow cross-account access.
   - Supports condition-based permissions (e.g., source IP, encryption, MFA).

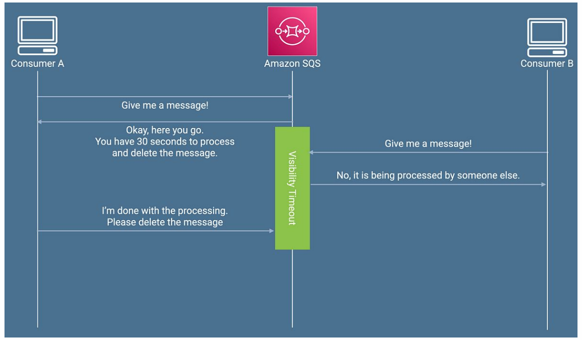
3. Private Access
   - Use VPC endpoints (interface type) to keep SQS traffic within your VPC.
   - Prevents exposure to the public internet.

Message Visibility Timeout

The Visibility Timeout is the period of time during which a message is invisible to other consumers after being read from the queue. It helps prevent duplicate processing.

1. When a consumer receives a message, it becomes hidden from other consumers for a set duration (default: 30 seconds, max: 12 hours).
2. If the consumer successfully processes and deletes the message within that window, all is good.
3. If it fails to delete it in time, the message becomes visible again and can be received by another consumer, potentially leading to reprocessing.

✅ Best practice: Tune the visibility timeout based on your app's average message processing time.

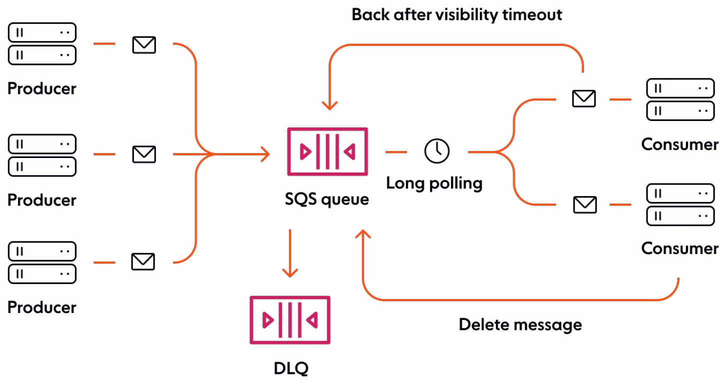

Consumer A
Amazon SQS
Consumer B
Give me a message!
Okay, here you go. You have 30 seconds to process and delete the message.
Visibility Timeout
Give me a message!
No, it is being processed by someone else.
I'm done with the processing. Please delete the message

FlipTheScript

Long Polling

Long polling reduces the number of empty responses and lowers costs by waiting until a message is available in the queue (or the timeout expires).

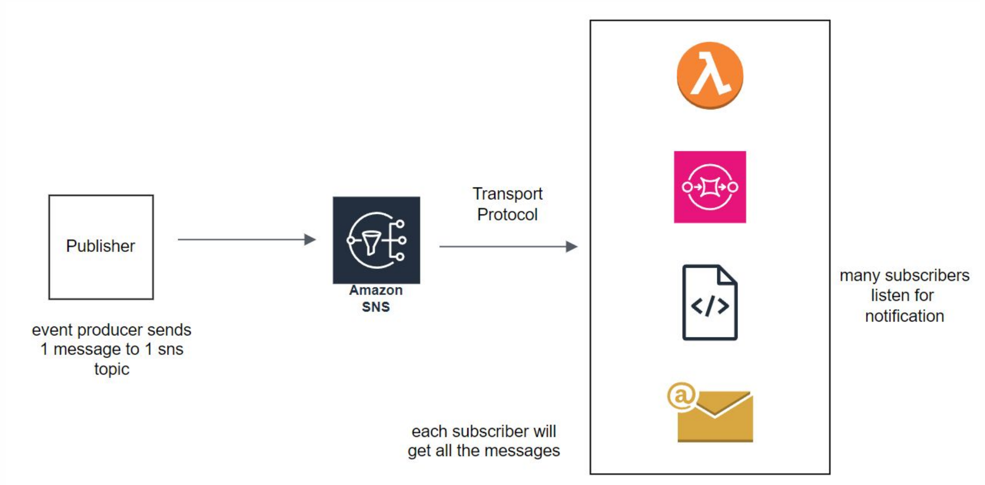
- By default, short polling immediately returns, possibly with no messages.
- Long polling waits up to 20 seconds for a message to arrive before returning.
- More efficient and cost-effective than constant short polling.
- Set using the WaitTimeSeconds parameter (max: 20 seconds) when calling ReceiveMessage.



Amazon SNS

Amazon SNS (Simple Notification Service) is a fully managed pub/sub messaging service that allows applications and AWS services to publish messages to a topic, which then fan out to multiple subscribers like SQS, Lambda, HTTP endpoints, email, or SMS.

- Use Case: When an image is uploaded to S3, S3 triggers SNS, which sends notifications to multiple systems (e.g., Lambda for processing, email alert to admins, etc.).


Publisher
event producer sends 1 message to 1 sns topic
Amazon SNS
Transport Protocol
each subscriber will get all the messages
many subscribers listen for notification

FlipTheScript

SNS Security

Amazon SNS secures message publishing and delivery with multiple mechanisms:

- IAM Policies: Control who can Publish, Subscribe, or Receive messages. You can apply these at the user/role or topic level.
- Resource-Based Policies: Allow cross-account access by explicitly defining which principals (users/accounts) can interact with a topic.
- Encryption:
  - In Transit – messages are secured with HTTPS.
  - At Rest – topics can use AWS KMS to encrypt stored messages.

- Private Communication: You can use VPC Endpoints to ensure SNS traffic stays inside your private AWS network and never touches the public internet.

Use Case:
A fintech company uses SNS to broadcast payment events. To meet compliance (e.g., PCI-DSS), they encrypt topics with KMS and restrict publishing to only approved services using IAM roles. Communication is kept within the VPC using interface endpoints to prevent exposure.

SNS + SQS Fan-out

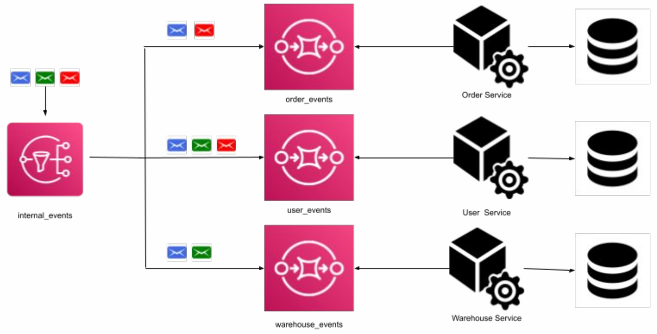
The Fan-out pattern in AWS SNS enables a single message published to an SNS topic to be delivered to multiple subscribers simultaneously. These subscribers can be:

- SQS queues (for message buffering)
- AWS Lambda functions (for processing)
- HTTP endpoints, email, or SMS (for alerts)

This design increases system scalability and decouples processing logic across different services.

Use Case:

- An e-commerce platform sends an order confirmation message to an SNS topic. The topic fans out the message to:
    - An SQS queue for the shipping service,
    - A Lambda function for billing,
    - An SMS endpoint to notify the customer.



SNS FIFO + Message Filtering

Amazon SNS now supports FIFO topics, which ensure:

- Exactly-once message delivery
- Strict message order (first-in, first-out)
- In addition, SNS supports Message Filtering, allowing subscribers to receive only specific messages based on message attributes. This reduces unnecessary processing and improves system efficiency.
- Filtering is configured using a subscription filter policy, where each subscriber defines which types of messages it wants to receive.

Use Case:

- A mobile app sends various notification events (INFO, WARNING, ERROR) to a FIFO SNS topic.
- One SQS FIFO queue subscribes with a filter policy to receive only "ERROR" messages.
- Another queue receives all message types for logging purposes.

Kinesis Overview

Amazon Kinesis is a platform for real-time data streaming at scale. It enables the collection, processing, and analysis of streaming data in real time, supporting use cases such as log analytics, fraud detection, real-time dashboards, and IoT applications.

Kinesis includes four core services:

- Kinesis Data Streams – for custom, real-time processing of data.
- Kinesis Data Firehose – for automatic delivery of data to destinations like S3, Redshift, or OpenSearch.
- Kinesis Data Analytics – for querying and analyzing streaming data using SQL.
- Kinesis Video Streams – for ingesting and processing video data.

Use Case:

- A ride-sharing company streams GPS location data from drivers to Kinesis. The data is processed in real time to calculate estimated arrival times and to update user apps with live locations.

Kinesis Data Streams

Kinesis Data Streams (KDS) is a core service in the Kinesis suite that allows real-time ingestion of large volumes of data, broken into shards (the basic unit of capacity).

There are two capacity modes:

- Provisioned Mode (default):
  - You manually specify the number of shards.
  - Each shard provides:
  - 1 MB/sec write throughput (or 1000 records/sec)
  - 2 MB/sec read throughput
  - You pay per shard-hour.

- On-Demand Mode:
  - Automatically scales based on usage.
  - Ideal for unpredictable or spiky workloads.
  - No need to manage shard count.
  - Each record in a stream has a retention window (default 24 hours, extendable to 7 days) and a partition key to control ordering.

Use Case:

- A gaming platform streams player activity data to Kinesis Data Streams. Using Provisioned Mode, they configure the shards based on average traffic during peak gaming hours. This enables near real-time analytics for player behavior and in-game events.

Kinesis Data Streams Security

Kinesis Data Streams offers several layers of security to protect real-time data:

- Encryption at Rest:
  - Uses AWS Key Management Service (KMS) to encrypt stored data in shards.

- Encryption in Transit:
  - Data is secured using HTTPS when moving between producers, the stream, and consumers.

- Access Control:
  - IAM policies control who can PutRecord, GetRecords, DescribeStream, etc.
  - Fine-grained permissions can be applied to specific streams or actions.

- VPC Endpoint Support:
  - You can configure private access to Kinesis using VPC interface endpoints to keep traffic inside your AWS network.

Use Case:

- A healthcare analytics system ingests patient monitoring data using Kinesis Data Streams. To comply with HIPAA, data is encrypted using customer-managed KMS keys, access is tightly controlled with IAM roles, and all traffic flows through VPC endpoints to avoid public exposure.

Kinesis Data Firehose

Amazon Kinesis Data Firehose is a fully managed service for delivering real-time streaming data to destinations like:
Amazon S3, Amazon Redshift, Amazon OpenSearch, Custom HTTP endpoints, Third-party services via API etc..

Key characteristics:

- No need to manage shards or infrastructure — fully serverless.
- Automatic batching, compression, transformation, and encryption.
- Can use Lambda for data transformation before delivery.
- Firehose buffers data before sending, using size or time thresholds, and ensures reliable delivery (supports retries).

Use Case:

- A security monitoring system collects logs from multiple AWS accounts. Firehose buffers, compresses, and transforms the data before storing it in S3 and indexing in OpenSearch, enabling near-real-time search and visualization.

Kinesis Data Streams vs Kinesis Firehose

Kinesis Data Streams gives you full control over how data is ingested and processed in near real-time using custom consumers like Lambda or EC2. You manage the scaling and processing logic manually.

Kinesis Firehose, on the other hand, is fully managed and ideal for automatically delivering data to destinations like S3, Redshift, or OpenSearch, with built-in buffering, retries, and optional transformations using Lambda.

Use Case:
A ride-hailing app uses Kinesis Data Streams to track driver locations with sub-second latency.
Meanwhile, a financial firm uses Kinesis Firehose to automatically load trading logs into S3 and Redshift for analytics and compliance.

SQS vs SNS vs Kinesis

- SQS is a message queue used to decouple producers and consumers. Messages are stored until a consumer retrieves and processes them, ensuring reliable delivery and allowing asynchronous communication.
- SNS is a pub/sub service used to push messages to multiple subscribers at once (e.g., SQS queues, Lambda, HTTP). It enables real-time broadcasting of events.
- Kinesis is designed for real-time streaming of large volumes of ordered data, enabling analytics and custom processing in near real-time.

Use Case:

1. An e-commerce platform uses SNS to notify multiple systems when a new order is placed.
2. SQS to queue the order for background processing.
3. And Kinesis to stream click data from users for behavioral analytics.

Amazon MQ

Amazon MQ is a managed message broker service that supports industry-standard protocols like AMQP, MQTT, and STOMP. It is based on Apache ActiveMQ and RabbitMQ, making it ideal for migrating existing applications that already use traditional messaging systems.

Unlike SQS or SNS, which are cloud-native and integrate deeply with AWS services, Amazon MQ is designed for compatibility with existing enterprise systems. It provides features like message durability, ordering, transactions, and support for message acknowledgments across multiple protocols.

Use Case:

- A legacy enterprise system using RabbitMQ on-premises migrates to AWS without changing application code by switching to Amazon MQ, preserving protocol compatibility and integration behavior.

Exam Questions

A company is building a serverless order processing system. When an order is placed, the system must notify several downstream services (inventory, billing, and customer notifications) in parallel. Some of these services need to buffer the messages before processing. Which design best meets these requirements using AWS services?

A. Use Amazon SQS with multiple queues and configure each service to poll its respective queue.
B. Use Amazon Kinesis Data Firehose with Lambda consumers for each service.
C. Use Amazon SNS to fan out the order message to multiple SQS queues, each subscribed by a different service.
D. Use Amazon MQ to broadcast messages to multiple endpoints using the MQTT protocol.

A healthcare company needs to ingest and process large volumes of patient monitoring data. The system must support encryption, low-latency data processing, and the ability to run custom transformations before storing the data in Amazon S3. Which solution best fits this requirement?

A. Amazon SQS with Lambda triggers writing data to S3

B. Amazon Kinesis Data Firehose with Lambda transformation and delivery to S3

C. Amazon SNS with multiple Lambda subscriptions writing to S3

D. Amazon MQ with a consumer application writing to S3

ECS Overview

Amazon ECS (Elastic Container Service) is a fully managed container orchestration service that lets you run and manage Docker containers on a cluster of EC2 instances or using Fargate.

Key Concepts:

- Cluster: A logical group of compute resources (EC2 or Fargate).
- Task Definition: A blueprint that defines your application (Docker image, CPU, memory, networking, etc.).
- Task: An instantiation of the task definition (i.e., a running container).
- Service: Ensures a specified number of tasks run and restarts failed tasks automatically.

Two Launch Types:

1. EC2 Launch – You manage the EC2 instances.
2. Fargate Launch – AWS manages the compute for you (serverless).

ECS is integrated with IAM, CloudWatch, Load Balancers, and can run behind a VPC with high security.

Use Case:

- ECS is ideal for teams using Docker containers that want to manage container workloads with tight AWS service integration, including load balancing, IAM, and auto-scaling.

IAM Role for ECS

In ECS, IAM roles are used to grant permissions to tasks and services, allowing them to interact securely with other AWS services.

There are two main types of IAM roles for ECS:

- Task Execution Role
    - Used by ECS to pull container images from Amazon ECR, and to log to CloudWatch.
    - Attached in the task definition.
    - Example permissions: ecr:GetAuthorizationToken, logs:CreateLogStream.

- Task Role
    - Used by the running application inside the container to call AWS services like S3, DynamoDB, or Secrets Manager.
    - Also defined in the task definition and assumed by the task at runtime.
    - This separation follows the principle of least privilege and gives granular control.

Use Case:

- A task that processes images from S3 and stores results in DynamoDB needs a task role with permissions to s3:GetObject and dynamodb:PutItem, while the execution role handles pulling the image and logging.

ECS Auto Scaling

Amazon ECS supports two types of auto scaling:

Service Auto Scaling:

- Scales the number of running tasks in a service based on metrics like CPU usage, memory usage, or custom CloudWatch alarms.
- Ensures that your service runs the desired number of healthy tasks depending on load.

Cluster Auto Scaling (for EC2 launch type):

- Scales the EC2 instances in your ECS cluster automatically using the Capacity Provider and an Auto Scaling group.
- ECS can launch or terminate EC2 instances based on task demand.

For Fargate, you only need Service Auto Scaling, since the infrastructure scales automatically under the hood.

Use Case:
A web app running on ECS EC2 tasks gets traffic spikes in the evening. ECS service auto scaling increases the number of tasks, and ECS cluster auto scaling provisions more EC2 instances to support them.

Amazon ECR

Amazon ECR is a fully managed Docker container registry that allows you to store, manage, and deploy container images securely at scale.

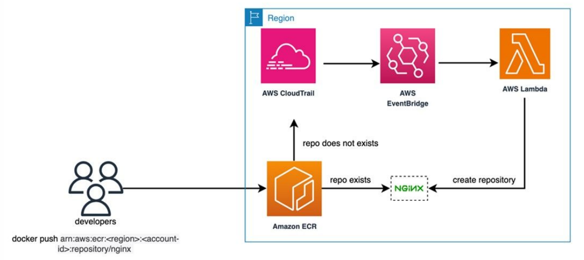
Key Features:

- Integrated with ECS, EKS, Fargate, and CodePipeline for seamless DevOps workflows.
- Supports image versioning, scanning for vulnerabilities, and lifecycle policies for cleanup.
- Authentication is done via IAM roles or token-based login (aws ecr get-login-password).

Push/Pull Workflow:

1. Developers push Docker images to ECR from their local machine or CI/CD pipelines.
2. ECS or EKS pulls the image during task or pod execution.

Use Case:
A CI/CD pipeline builds a Docker image on every code commit and pushes it to ECR. ECS automatically pulls the updated image during deployment.



FlipTheScript

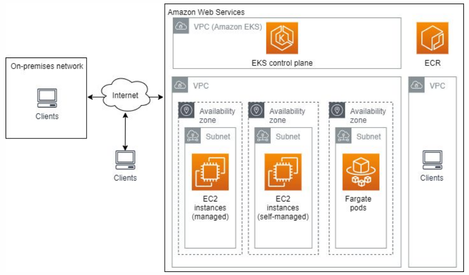
Amazon EKS

Amazon EKS is a fully managed Kubernetes service that makes it easy to run Kubernetes clusters on AWS without needing to install or operate your own control plane.

Key Features:

- AWS manages the Kubernetes control plane, including upgrades, patches, and scalability.
- You can run worker nodes on:
    - EC2 instances
    - Fargate (serverless pods)
- Integrated with IAM, VPC, CloudWatch, and other AWS services.
- Supports standard Kubernetes tooling (kubectl, Helm).

With EKS, you get the flexibility of Kubernetes and the security, scalability, and reliability of AWS.



Use Case:
A SaaS company wants to migrate its containerized microservices from an on-premises Kubernetes cluster to AWS. They use EKS to simplify management and leverage native AWS integrations.

EKS Node Types

In Amazon EKS, you can choose different types of worker nodes to run your Kubernetes workloads:

- Self-Managed EC2 Nodes
  - You provision and manage EC2 instances yourself.
  - Full control over instance type, AMI, patching, and lifecycle.
  - Best for custom configurations or specific compliance needs.

- Managed Node Groups
  - AWS provisions and manages EC2 instances in an Auto Scaling group.
  - Simplifies updates and lifecycle management.
  - Best for general-purpose workloads that need easier administration.

- Fargate (Serverless)
  - No need to manage servers.
  - AWS launches pods on demand.
  - Ideal for bursty or unpredictable workloads where you pay per pod usage.

Use Case:
A startup runs predictable workloads on managed node groups for simplicity and cost efficiency but uses Fargate for scheduled jobs that only run once a day.

AWS App Runner

AWS App Runner is a fully managed service that makes it easy to deploy web applications and APIs directly from source code or a container image — without needing to manage servers, load balancers, or scaling.

Key Features:

- Automatically builds and deploys from GitHub or Amazon ECR.
- Handles load balancing, auto scaling, and TLS out of the box.
- You only pay for what you use (active request time & provisioned capacity).
- Supports custom domains and HTTPS.

Use Case:
A developer wants to deploy a Python Flask web app with no infrastructure setup. They connect their GitHub repo to App Runner, which builds and deploys it automatically, handling scaling and TLS without manual configuration.

ECS + EKS Exam Questions

A company is running a microservices application using Amazon ECS. They want full control over the underlying infrastructure, including the AMI, security group settings, and patch management. They also want to save costs by using Reserved Instances.
Which ECS launch type should they choose?

A. Fargate

B. Fargate Spot

C. EC2

D. App Runner

Your team is deploying a Kubernetes-based application on Amazon EKS. You want AWS to manage the lifecycle of worker nodes, including automated updates and integration with Auto Scaling Groups.
Which EKS compute option should you choose?

A. Fargate

B. Self-managed EC2 nodes

C. AWS Lambda

D. Managed Node Groups

Lambda

AWS Lambda is a serverless compute service that runs code in response to events without provisioning or managing servers.

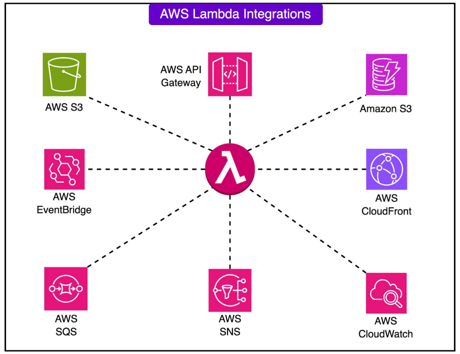
Language Support:
Supports multiple runtimes including Python, Node.js, Java, Go, .NET, Ruby, and custom runtimes via Lambda Runtime API.

Key Integrations:

- Event sources: API Gateway, S3, DynamoDB, EventBridge, CloudWatch, Kinesis, Cognito, SNS, SQS.
- Destinations (after execution): SNS, SQS, EventBridge, another Lambda.

Important Limits (default, can be increased):

- Execution timeout: up to 15 minutes
- Memory: 128 MB to 10 GB
- Ephemeral disk (/tmp): 512 MB by default, up to 10 GB
- Concurrency: 1,000 (soft limit per region)



Package size:

- Direct upload: 50 MB
- With layers or S3: up to 250 MB unzipped

Use Case:
Trigger a Lambda function using S3 event notifications to resize images uploaded by users.

Lambda SnapStart

Lambda SnapStart is a feature that improves cold start performance for Java-based Lambda functions.

How it works:

- During function publish, AWS initializes the function, captures a snapshot of the memory and execution state after the Init phase, and stores it.
- When invoked, Lambda restores the snapshot rather than reinitializing, reducing cold starts significantly (up to 10× faster).

Important Notes:

- Only supports Java 11 at the moment.
- Not supported with features like ephemeral /tmp storage, extensions, or custom runtimes.
- Designed for high-performance, low-latency Java apps like financial APIs or e-commerce search.

Use Case:
A Java-based Lambda function used for processing payments needs millisecond response time. SnapStart helps minimize latency during first-time or infrequent invocations.

Lambda@Edge with CloudFront

Lambda@Edge is an extension of AWS Lambda that allows you to run functions closer to your users by executing them at AWS edge locations — integrated with Amazon CloudFront.

Key Capabilities:

- Modify or inspect HTTP requests/responses before or after CloudFront forwards them to origin.
- Supports dynamic web personalization, A/B testing, authentication, custom headers, redirects, and geo-targeting.

- Executes in response to four trigger points:
  - Viewer request
  - Viewer response
  - Origin request
  - Origin response

Use Case:
Personalize web content based on user location by modifying requests with Lambda@Edge before reaching the origin.

CloudFront Functions vs Lambda@Edge

Both are used to run code at AWS edge locations with Amazon CloudFront, but they serve different purposes and have different capabilities.

CloudFront Functions:

- Ultra-lightweight JavaScript functions
- Designed for high performance, low-latency tasks
- Executes only at Viewer Request stage
- Common use: URL rewrites, redirects, header manipulation

Lambda@Edge:

- More powerful, supports multiple runtimes (Python, Node.js, etc.)
- Executes at all four stages (viewer/origin request/response)
- Use for complex logic like authentication, dynamic content, geo personalization

Use Case:
Use CloudFront Functions for quick redirects, and Lambda@Edge for inspecting and modifying request headers based on user device type or authentication.

Lambda in a VPC

By default, Lambda functions run in an isolated AWS-managed environment outside of your VPC. If you want your Lambda to access resources inside a private VPC (e.g., RDS, EC2, Redis), you must explicitly configure VPC access.

When you attach a Lambda function to a VPC:

- You must specify subnet(s) and security group(s).
- Lambda creates Elastic Network Interfaces (ENIs) in your subnets.
- The function no longer has internet access by default unless routed via a NAT Gateway or NAT instance.

Key Considerations:

- Slightly increased cold start time due to ENI attachment.
- Best practice: Use private subnets + NAT Gateway for secure internet access.

Use Case:
A Lambda function that queries a MySQL RDS database inside a private subnet must be attached to the VPC to connect securely.

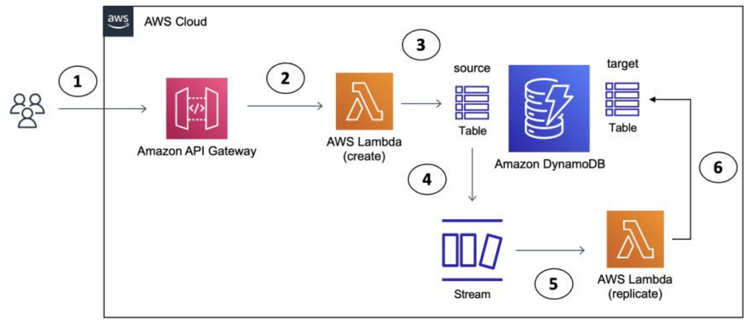
DynamoDB

Amazon DynamoDB is a fully managed NoSQL database service offering single-digit millisecond latency at any scale.

Key features:

- Key-value and document-based storage
- Serverless – no need to provision infrastructure
- Highly available & scalable by default
- Built-in encryption at rest, backup and restore, and multi-Region replication
- Integrated with other AWS services like Lambda, API Gateway, and Cognito

Use Case:
Used in high-throughput applications such as shopping carts, gaming leaderboards, IoT data ingestion, or user session storage, where performance and scale are critical.



DynamoDB Capacity Modes

DynamoDB offers two capacity modes to control how read/write throughput is managed:

1. On-Demand Mode
    - No capacity planning required.
    - Automatically scales with usage.
    - You pay per request (read/write units).
    - Ideal for unpredictable or bursty workloads.

2. Provisioned Mode
    - You specify read and write capacity (RCU/WCU).
    - Can enable Auto Scaling.
    - Cheaper for predictable workloads.
    - Throttling can occur if limits are exceeded.

Use Case:
Use On-Demand for new apps with unknown traffic patterns.
Use Provisioned for stable workloads like a product catalog API with consistent traffic.

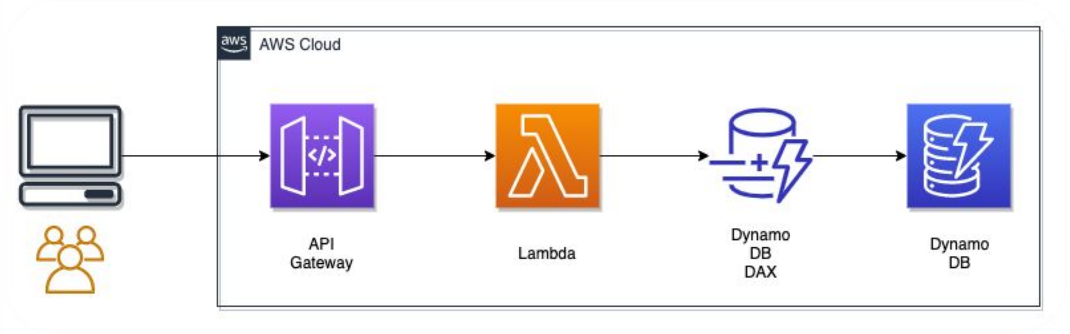
DAX – DynamoDB Accelerator

DynamoDB Accelerator (DAX) is a fully managed in-memory caching service that improves read performance for DynamoDB.

Key benefits:

- Reduces read latency from milliseconds to microseconds
- Fully compatible with existing DynamoDB API calls
- Ideal for read-heavy or latency-sensitive workloads
- Managed by AWS – no need to handle cache invalidation or cluster management
- DAX is not a write-through cache; it handles read operations and caches results.

Use Case:
An e-commerce application that displays product information rapidly to many users benefits from DAX, as it minimizes read latency and database load.



DynamoDB Streams

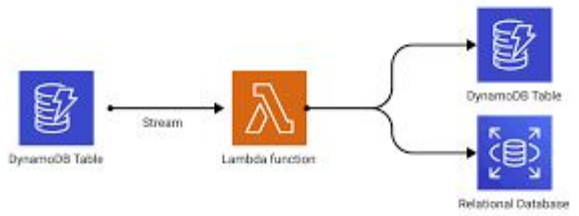
DynamoDB Streams capture a time-ordered sequence of item-level changes in a table and allow them to be consumed asynchronously.

Key features:

- Captures events for insert, update, delete operations
- Stores events for 24 hours
- Each stream record includes before and after images (configurable)
- Integrated with AWS Lambda for real-time processing

Common use: trigger downstream actions when changes happen in DynamoDB — for example, updating a search index or notifying another service.

Use Case:
When a new user is added to the Users table, a Lambda function triggered by the stream can send them a welcome email or log analytics data.



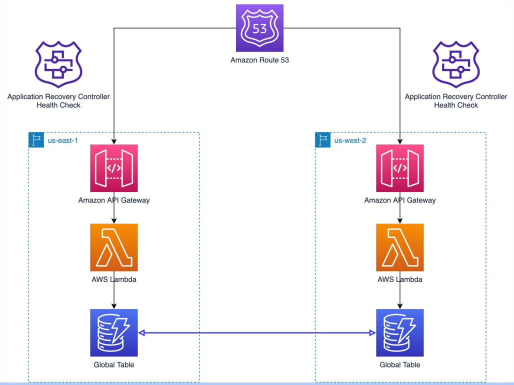
DynamoDB Global Tables

Global Tables in DynamoDB provide multi-Region, fully replicated tables to support global applications with low-latency data access and high availability.

Key features:

- Active-active replication: changes made in one Region are automatically replicated to all others.
- Conflict-free: uses last writer wins strategy for resolving concurrent writes.
- Great for disaster recovery and geo-distributed apps.
- Requires DynamoDB Streams to be enabled.

Use Case:
A mobile game used worldwide can store player data in a Global Table, ensuring fast access and real-time sync whether users are in Europe, the US, or Asia.



FlipTheScript

DynamoDB Disaster Recovery

DynamoDB provides several built-in features for disaster recovery (DR) to ensure durability, availability, and business continuity:

Point-in-Time Recovery (PITR):
Automatically backs up data continuously and lets you restore to any second in the last 35 days.

On-Demand Backups:
Manual full backups of the table at any time, retained until deleted.

Global Tables:
Offer multi-Region redundancy — if one Region fails, traffic can be routed to another.

Cross-Region Replication:
Achieved via Global Tables for high availability and DR.

Use Case:
A financial app can enable PITR to recover from accidental writes or deletions and use Global Tables to remain online even if a Region becomes unavailable.

API Gateway

Amazon API Gateway is a fully managed service for creating, publishing, maintaining, and securing APIs at any scale.

Key features:

- Supports REST, HTTP, and WebSocket APIs
- Handles throttling, authorization, monitoring, and API versioning
- Automatically scales to handle millions of requests
- Natively integrates with AWS services like Lambda, DynamoDB, and Step Functions

Use Case:
Use API Gateway as a frontend to expose your backend logic (e.g., Lambda functions) securely to clients such as mobile apps or web browsers.

API Gateway Security + Endpoint Types

Amazon API Gateway offers multiple security features and endpoint types to protect and manage your APIs:

Security Options:

- IAM Authentication – restricts access to AWS users/roles.
- Cognito User Pools – handles user sign-in/sign-up.
- Lambda Authorizers – run custom logic to allow/deny requests.
- API Keys – used for throttling and identifying usage.
- Resource Policies – control access from specific IPs or VPCs.

Endpoint Types:

- Edge-Optimized – best for public clients (uses CloudFront).
- Regional – best for clients within the same Region.
- Private – accessible only from within your VPC (via VPC Endpoint).

Use Case:
Expose your Lambda functions to the internet securely using an Edge-Optimized endpoint with Cognito for user auth, and use API Keys to throttle free-tier users.

AWS Step Functions

AWS Step Functions is a serverless orchestration service that lets you coordinate multiple AWS services into workflows using state machines.

Key features:

- Define workflows using Amazon States Language (ASL) (JSON-based)
- Built-in support for error handling, timeouts, retries, and parallel execution
- Integrates with services like Lambda, ECS, SageMaker, DynamoDB, SNS, SQS, Glue, etc.

Two workflow types:

1. Standard Workflows – long-running (up to 1 year), detailed execution history
2. Express Workflows – short duration, high-throughput (ideal for event-driven apps)

Use Case:
Automate an ETL pipeline: use Step Functions to orchestrate Lambda for data extraction, Glue for transformation, and S3 for storage.

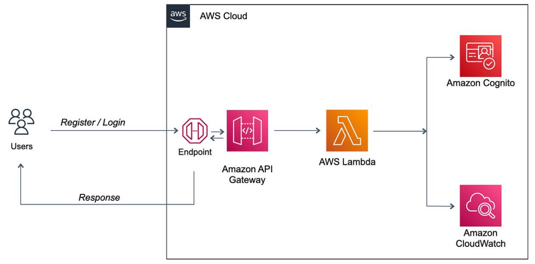
Amazon Cognito

Amazon Cognito is a serverless identity service that provides user authentication, authorization, and user management for your web and mobile apps.

Key features:

- Handles sign-up, sign-in, password recovery, and MFA.
- Scales to millions of users.
- Supports federated identities – users can log in using Facebook, Google, SAML, etc.
- Fully integrates with API Gateway, Lambda, and IAM for access control.

Use Case:
A web app uses Cognito to allow users to register and sign in via their Google accounts, then receive tokens to securely access a protected API.



Cognito User Pools vs Identity Pools – Comparison

User Pools:

- User Pools are for authentication – they manage user sign-up, sign-in, and user profiles.
- Issue JWT tokens (ID, Access)
- Integrate directly with API Gateway, Lambda, and App clients
- Support MFA, password policies, and OAuth providers

Identity Pools:

- Identity Pools are for authorization – they provide temporary AWS credentials to authenticated or unauthenticated users.
- Use IAM roles to access AWS services (like S3, DynamoDB)
- Can federate identities from User Pools, SAML, Google, Facebook, etc.

Main Difference:
User Pools = Who you are
Identity Pools = What you can access

Use Case:
Use User Pool to handle login and get tokens, then exchange them in an Identity Pool to grant access to upload a file to an S3 bucket.

Serverless Exam Questions

An application uses Amazon Cognito for authentication. After users log in, they need temporary access to upload documents to an S3 bucket. The solution should be fully managed and secure.

Which Cognito component provides the necessary credentials?

A. Cognito User Pools

B. Cognito Identity Pools

C. IAM User

D. Lambda Authorizer

A company is building a RESTful web service to handle customer support tickets. The service must scale automatically, be cost-effective, and require minimal infrastructure management. It should expose HTTP endpoints to clients, store data in a NoSQL database, and allow business logic execution during requests.

Which AWS services should be used to build this serverless architecture?

A. Amazon EC2 + Amazon RDS + Elastic Load Balancer
B. AWS Lambda + Amazon API Gateway + Amazon DynamoDB
C. Amazon ECS Fargate + Application Load Balancer + Amazon Aurora Serverless
D. AWS Lambda + Amazon S3 + Amazon Redshift