SKILLS | DevOps and Cloud Computing

AWS Messaging Services and Serverless Application Development

Objective

- Understand the use cases and configurations of Amazon SQS queues (Standard and FIFO).
- Learn how Amazon SNS facilitates publish-subscribe messaging and fan-out patterns.
- Gain knowledge of using Amazon SES for sending and receiving emails in cloud applications.
- Explore the basics of AWS SAM, including how to build
- and deploy serverless applications using infrastructure as code.

PW SKILLS

Explaining how SQS helps decouple microservices through message queues.

Let’s see

Amazon SQS (Simple Queue Service) helps decouple microservices by acting as a message queue between them.

- Instead of one service directly calling another (which creates tight coupling), it sends messages to a queue.
- The receiving service processes messages from the queue independently and at its own pace.
- This improves system reliability, scalability, and fault tolerance by allowing services to operate asynchronously and recover from failures without blocking others.

Pop Quiz

Q. Why is SQS useful in building fault-tolerant microservices?

A

It stores and retries messages if consumers fail

B

It prevents microservices from scaling

Pop Quiz

Q. Why is SQS useful in building fault-tolerant microservices?

A

It stores and retries messages if consumers fail

B

It prevents microservices from scaling

Differentiating between Standard (at-least-once delivery) and FIFO (exactly-once order) queues.

Let’s discuss

- **Standard Queue** offers at-least-once delivery and best-effort ordering, making it suitable for high-throughput, non-sequential tasks.

- **FIFO Queue** ensures exactly-once processing and preserves message order, ideal for use cases where order and duplication control are critical.

PW SKILLS

Introducing Visibility Timeout and its importance in preventing duplicate processing.

Let’s see

Visibility Timeout in Amazon SQS is the period during which a message is hidden from other consumers after being retrieved by one consumer.

Importance:

- It prevents duplicate processing by ensuring that once a message is picked up for processing, it won't be visible to other consumers until the timeout expires.

- If the message is not deleted (due to a failure), it becomes visible again for reprocessing. This helps ensure reliable, exactly-once processing logic when combined with proper handling.

PW SKILLS

Explaining Dead-letter Queues for failed messages and retries.

Let’s see

Dead-letter Queues (DLQs) in Amazon SQS are used to store messages that fail to be processed successfully after a specified number of retry attempts.

Purpose:
They help isolate and analyze problematic messages without blocking the main queue. DLQs improve reliability by preventing endless retry loops and allowing developers to debug and fix issues separately.

PW SKILLS

Showing how SQS can buffer requests between services in a distributed system.

Let’s see

Amazon SQS buffers requests by acting as a message queue between producer and consumer services in a distributed system.

How it works:

- The producer sends messages to the SQS queue, and consumers pull messages at their own pace.

- This decouples services, allowing producers to continue sending requests even if consumers are temporarily slow or unavailable, ensuring smooth and scalable communication.

Illustrating real-world examples like order processing or logging services.

Let’s see

1. Order Processing System (E-commerce)
Scenario: Customers place orders on a website.

How SQS helps:

- Orders are sent to an SQS queue.
- Backend workers (e.g., payment, inventory, shipping) consume messages from the queue asynchronously.
- This ensures scalability and fault tolerance, even during traffic spikes.

Let’s see

2. Centralized Logging Service
Scenario: Multiple microservices generate logs.

How SQS helps:

- Each service pushes log messages to a central SQS queue.
- A log processor reads from the queue and writes logs to a storage or analysis tool (e.g., S3, Elasticsearch).
- This prevents log loss and avoids overloading the logging system.

Introducing SNS as a pub/sub (publish-subscribe) service.

Let’s see

Amazon SNS (Simple Notification Service) is a publish-subscribe (pub/sub) messaging service.

How it works:
A publisher sends a message to an SNS topic, and multiple subscribers (like SQS queues, Lambda functions, email, or SMS) receive the message simultaneously.
It enables real-time, one-to-many communication across distributed systems.

Pop Quiz

Q. What is the primary communication model used by Amazon SNS?

A

Publish-subscribe

B

Request-response

Pop Quiz

Q. What is the primary communication model used by Amazon SNS?

A

Publish-subscribe

B

Request-response

PW SKILLS

Explaining topics and subscriptions, and contrast it with direct messaging like SQS.

Let’s see

In SNS, a topic is a communication channel that publishers send messages to. Subscriptions define the endpoints (e.g., SQS, Lambda, email) that receive those messages.

SNS vs. SQS:

- SNS (Pub/Sub): One message is broadcast to multiple subscribers in real time.
- SQS (Point-to-Point): One message is consumed by a single receiver, ideal for decoupling tasks.

Showing fan-out patterns: SNS topic publishing to multiple SQS queues or Lambda functions.

Let’s do it

In a fan-out pattern, an SNS topic publishes a message to multiple subscribers at once.

Example:
An event (e.g., new user signup) is published to an SNS topic.

The topic delivers this message to:

- An SQS queue for logging,
- A Lambda function for sending a welcome email,
- Another SQS queue for analytics.

PW SKILLS

Take A 5-Minute Break!

- Stretch and relax
- Hydrate
- Clear your mind
- Be back in 5 minutes

PW SKILLS

Discussing how SNS integrates with Lambda, SQS, HTTP endpoints, email, and other services.

Let's discuss

Amazon SNS (Simple Notification Service) integrates with various services using its publish-subscribe model:

- **Lambda:** SNS can trigger AWS Lambda functions to run serverless code in response to published messages.
- **SQS:** SNS can fan out messages to multiple SQS queues for decoupled processing.
- **HTTP/S Endpoints:** SNS delivers POST requests to subscribed web servers or applications via HTTP/S.
- **Email:** SNS can send notification emails to subscribed email addresses.
- **Other Services:** SNS can integrate with SMS, mobile push (e.g., Firebase, APNs),

PW SKILLS

Walk through real-world use cases like real-time alerts, Lambda-based processing, or failover notifications.

Let’s see

Here are real-world use cases of Amazon SNS:

- **Real-time alerts:** CloudWatch alarms publish to SNS, which sends SMS/email alerts to admins on high CPU or billing thresholds.
- **Lambda-based processing:** An app uploads images to S3 → triggers SNS → invokes a Lambda function to process or resize the images.
- **Failover notifications:** Health checks fail on a service → SNS notifies support teams via email/SMS and triggers Lambda to launch a backup server.
- **Fan-out pattern:** An order event in e-commerce → SNS publishes to multiple subscribers like inventory (SQS), analytics (Lambda), and customer notification (email).

PW SKILLS

Explaining the purpose of Simple Email Service (SES) for applications needing email functionality.

Let’s see

Amazon SES (Simple Email Service) enables applications to send, receive, and forward emails at scale.

It’s used for:

- Transactional emails (e.g., password resets, order confirmations)
- Marketing campaigns (e.g., newsletters, promotions)
- Notification systems (e.g., alerts, updates)

Pop Quiz

Q. How can applications interact with Amazon SES?

A

By manually uploading emails via S3

B

Using SMTP or SES APIs

Pop Quiz

Q. How can applications interact with Amazon SES?

A

By manually uploading emails via S3

B

Using SMTP or SES APIs

PW SKILLS

Demonstrating email sending features with templates and customization.

Let’s do it

Amazon SES supports sending customized emails using:

- **Templates:** Predefined email content with placeholders (e.g., {{name}}, {{order id}}).
- **Personalization:** Replace placeholders with user-specific data via the Send Templated
  Email API.
- **Bulk sending:** Send personalized messages to multiple recipients in a single API call using SendBulkTemplatedEmail.

PW SKILLS

Discussing receiving workflows, email rules, configuration sets, and delivery optimization

Let’s discuss

Amazon SES receiving workflows enable apps to process incoming emails:

- **Email receiving:** SES can accept mail for verified domains.
- **Rules:** Set up rule sets to filter and route emails (e.g., save to S3, trigger Lambda, publish to SNS).
- **Configuration sets:** Track and manage sending events (opens, clicks, bounces) using event destinations (CloudWatch, Kinesis, SNS).
- **Delivery optimization:** Use DKIM/SPF, dedicated IPs, reputation metrics, and feedback loops to improve inbox placement and reduce bounces.

PW SKILLS

Introducing AWS SAM as an abstraction over CloudFormation for simplified serverless deployment.

Let's introduce

AWS SAM (Serverless Application Model) is an abstraction over CloudFormation that simplifies defining and deploying serverless applications.

- Uses concise YAML syntax to define Lambda functions, APIs, DynamoDB tables, etc.
- Supports sam build, sam deploy, and sam local for easy packaging, deployment, and local testing.
- Converts SAM templates into standard CloudFormation stacks.

PW SKILLS

Explaining key concepts: SAM templates, resource definitions, and Lambda packaging.

Let’s see

Key concepts in AWS SAM:

- **SAM Templates:** YAML files that define serverless resources using simplified syntax (e.g., AWS::Serverless::Function).
- **Resource Definitions:** Declare functions, APIs, tables, etc., with minimal config—SAM expands them into full CloudFormation.
- **Lambda Packaging:** sam build compiles and packages code; sam deploy uploads it to S3 and deploys the stack.

Describing the build → package → deploy flow using the SAM CLI.

Let’s describe

Let’s discuss

The SAM CLI follows this flow:

- sam build – Compiles source code and prepares artifacts (e.g., dependencies) for deployment.
- sam package (optional) – Packages the app and uploads artifacts to an S3 bucket. (Handled internally by sam deploy)
- sam deploy – Deploys the app as a CloudFormation stack using the built artifacts.

Time for case study!

Important

- Complete the post-class assessment
- Complete assignments (if any)
- Practice the concepts and techniques taught in this session
- Review your lecture notes
- Note down questions and queries regarding this session and consult the teaching assistants

Thanks!