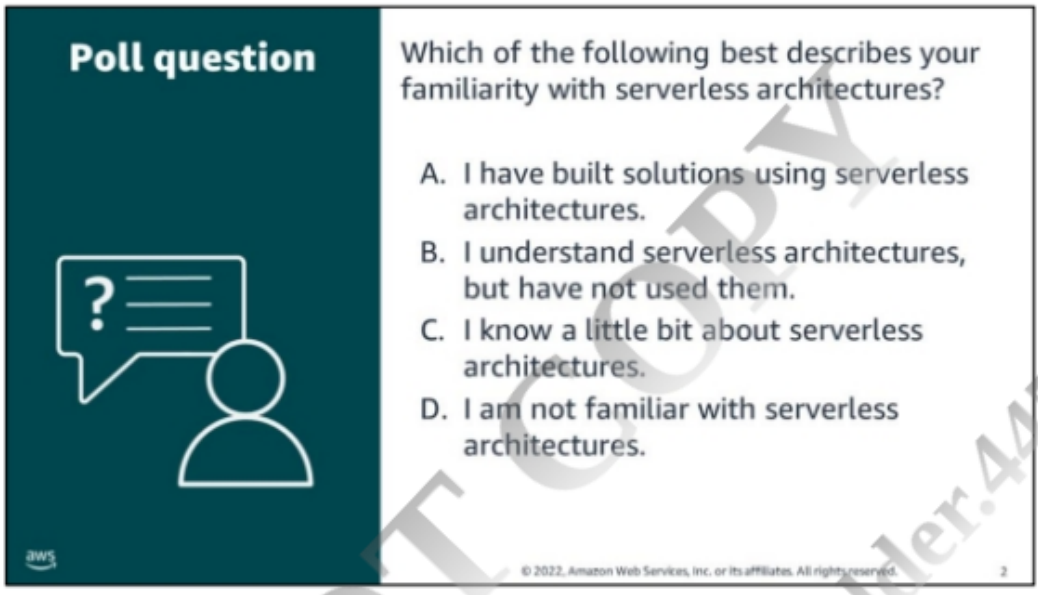
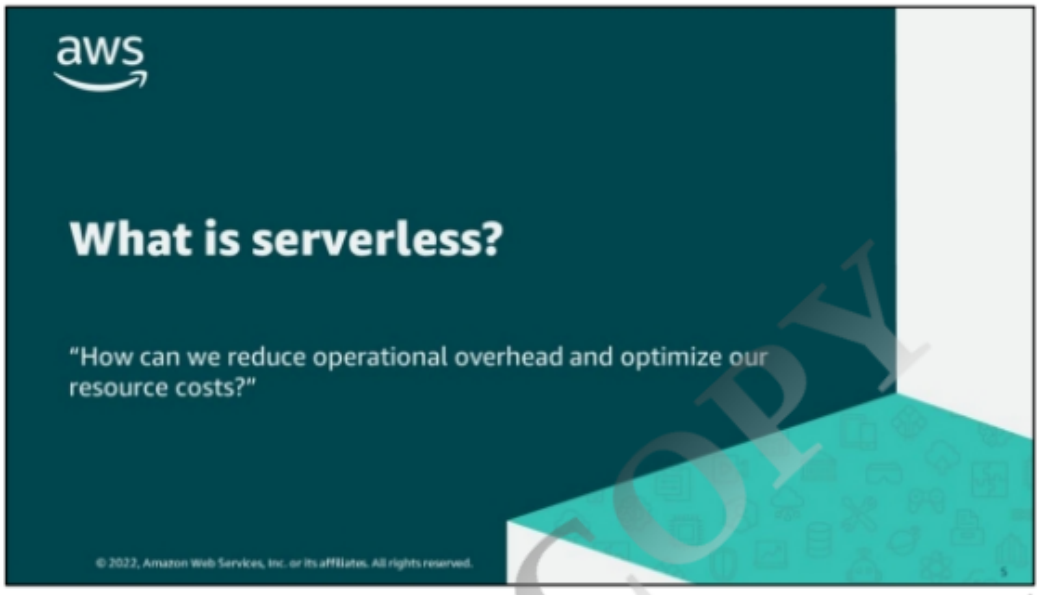
Printed by: 2d35e8483186bd2@placeholder.44518.edu. Printing is for personal, private use only. No part of this book may be reproduced or transmitted without publisher's prior permission. Violators will be prosecuted.



Module overview

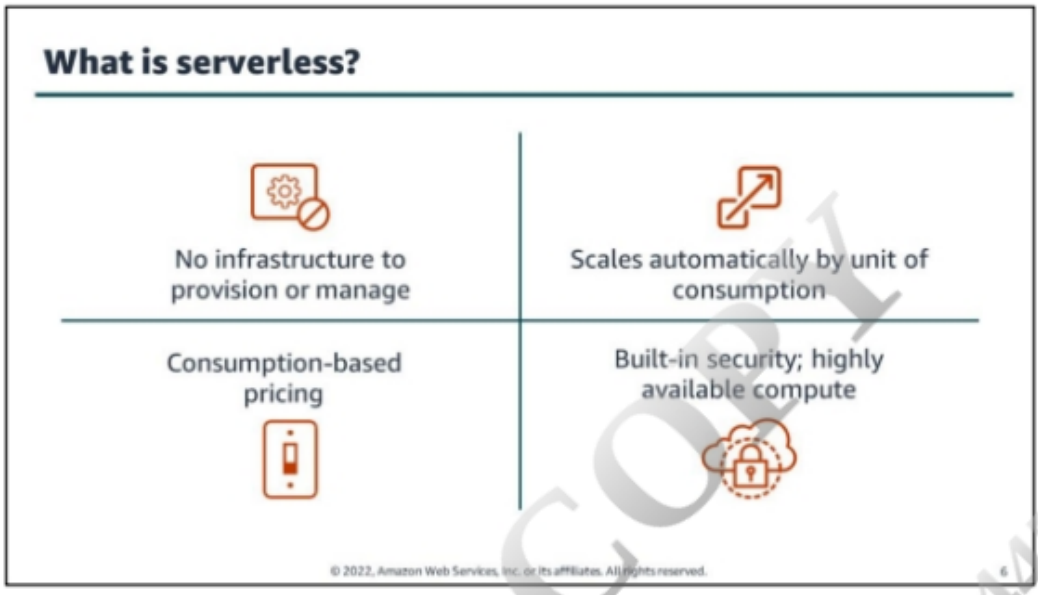
- Business request
- What is serverless?
- Amazon API Gateway
- Amazon Simple Queue Service (Amazon SQS)
- Amazon Simple Notification Service (Amazon SNS)
- Amazon Kinesis
- AWS Step Functions
- Present solutions
- Knowledge check
- Lab 5: Build a serverless architecture

Imagine you are meeting with an application development manager who is preparing to build on AWS. The manager is interested in the benefits of serverless architectures. During this module, you learn about topics that answer these questions.



The application development manager asks, "How can we reduce operational overhead and optimize our resource costs?"

The company is looking for ways to shift resources towards developing new features and services. They are concerned about the additional infrastructure and cost that this could create. The company is asking you to research different ways to build in the cloud.

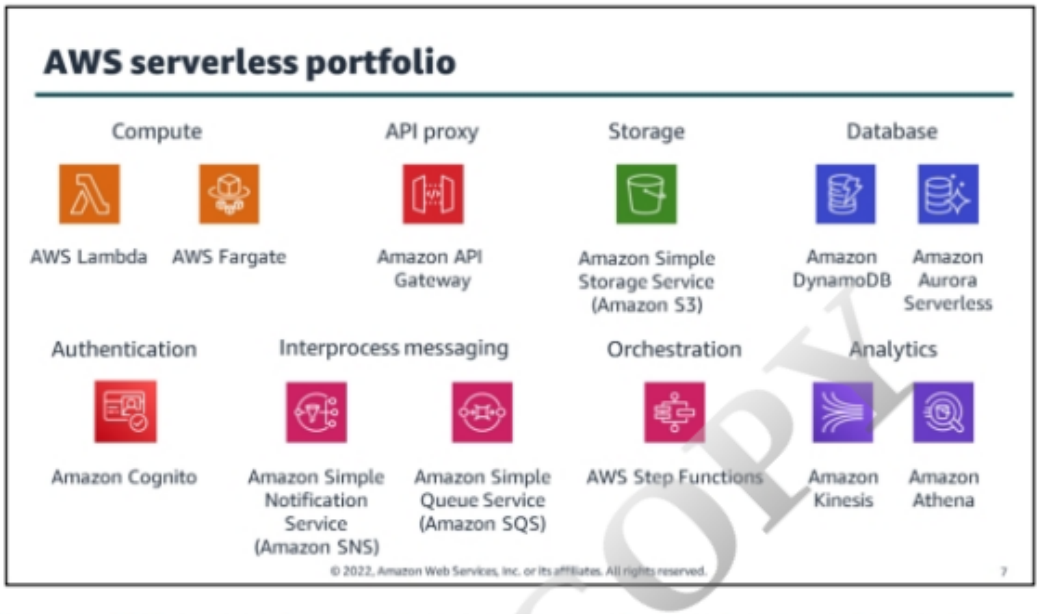


Serverless is a way to describe the services, practices, and strategies that you can use to build more agile applications. With serverless computing, you can innovate and respond to change faster. AWS handles infrastructure management tasks (such as capacity provisioning and patching), so you can focus on writing code that serves your customers.

Advantages of using serverless:

- No infrastructure to provision or manage
- No servers to provision, operate, or patch
- Scales automatically by unit of consumption, rather than by server unit
- Pay-for-value billing model (pay for the unit, rather than by server unit)
- Built-in availability and fault tolerance
- No need to architect for availability because it is built into the service

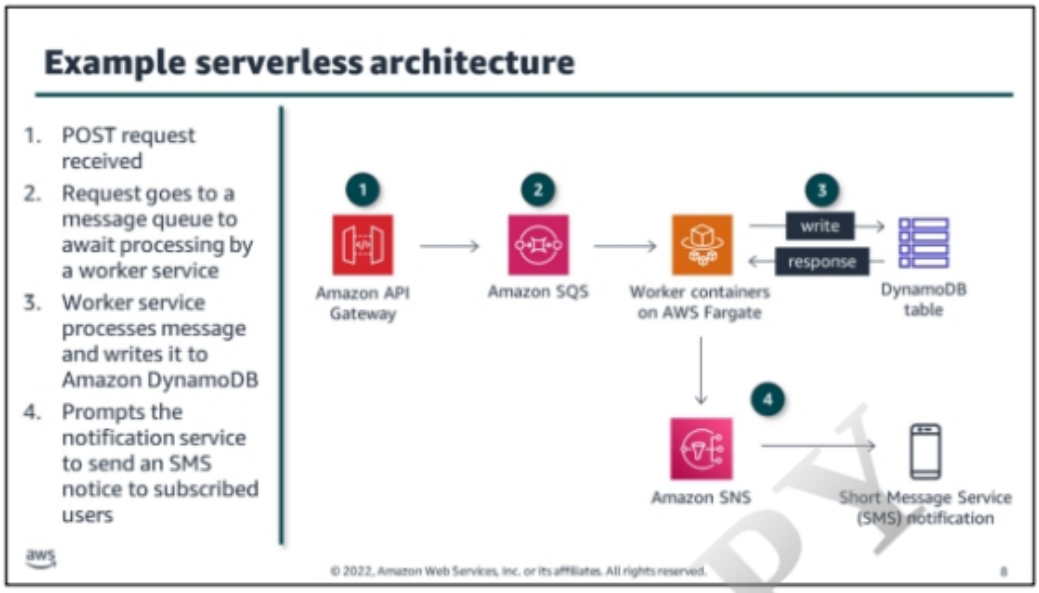
For more information, see "Serverless on AWS" (https://aws.amazon.com/serverless/).



AWS provides a set of fully managed services that you can use to build and run serverless applications. Serverless applications don't require you to provision, maintain, and administer servers for backend components, such as compute, databases, storage, stream processing, messaging, and queueing. You also no longer need to worry about ensuring application fault tolerance and availability. AWS provides all these capabilities for you so you can focus on product innovation while enjoying faster time to market.

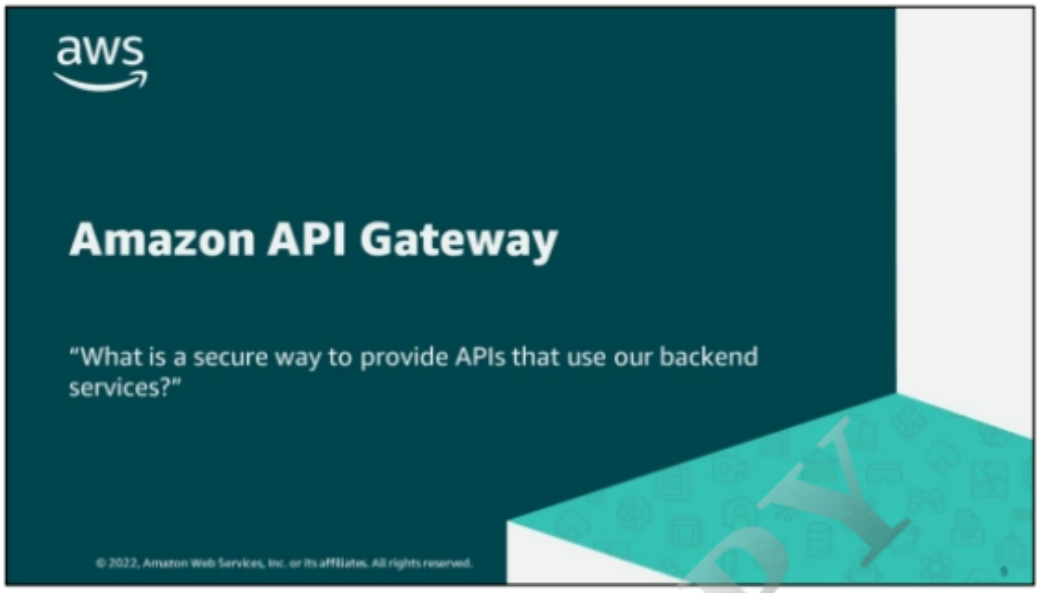
You learned about compute, storage, and database services in previous modules. In this module, you learn about the following:

- API Gateway
- Amazon SQS
- Amazon SNS
- Kinesis
- Step Functions



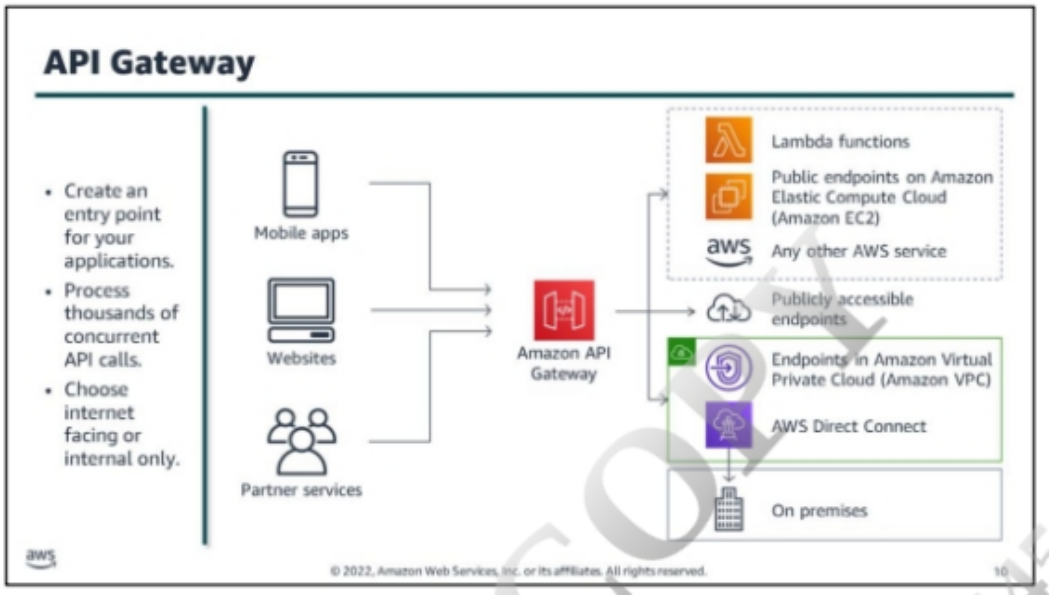
This example of a serverless architecture demonstrates a messaging application.

It uses Amazon API Gateway to handle all of the tasks involved in accepting and processing API calls. It feeds messages to an application messaging queue built in Amazon SQS, where the messages are processed by a worker service. This queue provides a buffer for the worker service in the event of traffic spikes. The containerized worker service uses serverless hosting in AWS Fargate, which automatically scales hosting capacity with each newly launched container. The worker service writes the processed message to an Amazon DynamoDB table. Once the worker receives a success response, the worker notifies Amazon SNS, which pushes an SMS notification to all subscribed users.



The application development manager asks, "What is a secure way to provide APIs that use our backend services?"

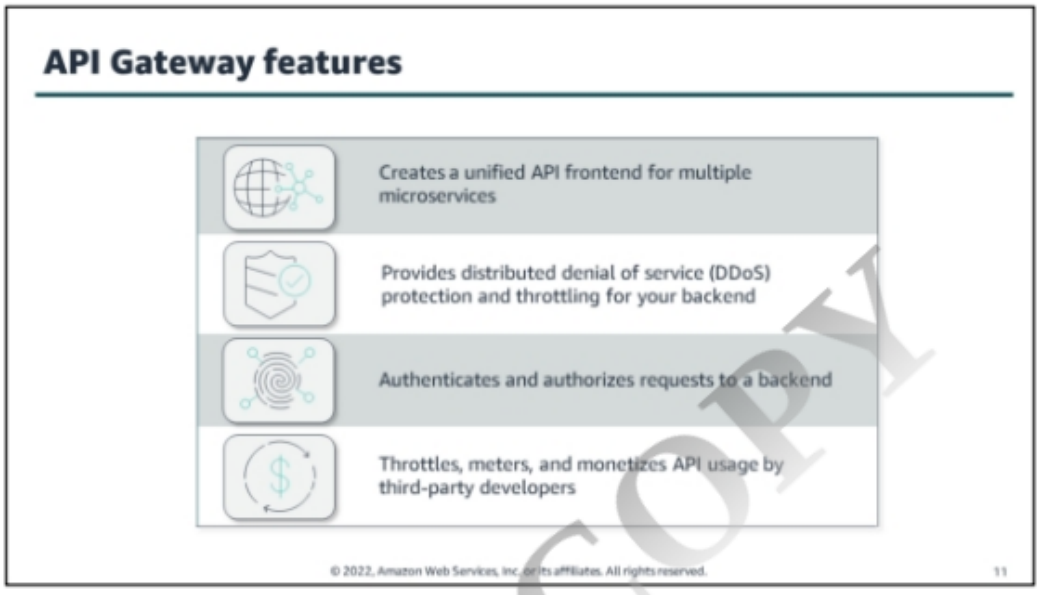
The company has some services that they would like to expose for third-party developers using a REST API. They have concerns about how to do this in a secure way while also managing scale, throttling, and monitoring. They are asking you to recommend a service that minimizes operational overhead and can protect against security threats.



With Amazon API Gateway, you can create, publish, maintain, monitor, and secure APIs.

With API Gateway, you can connect your applications to AWS services and other public or private websites. It provides consistent RESTful and HTTP APIs for mobile and web applications to access AWS services and other resources hosted outside of AWS.

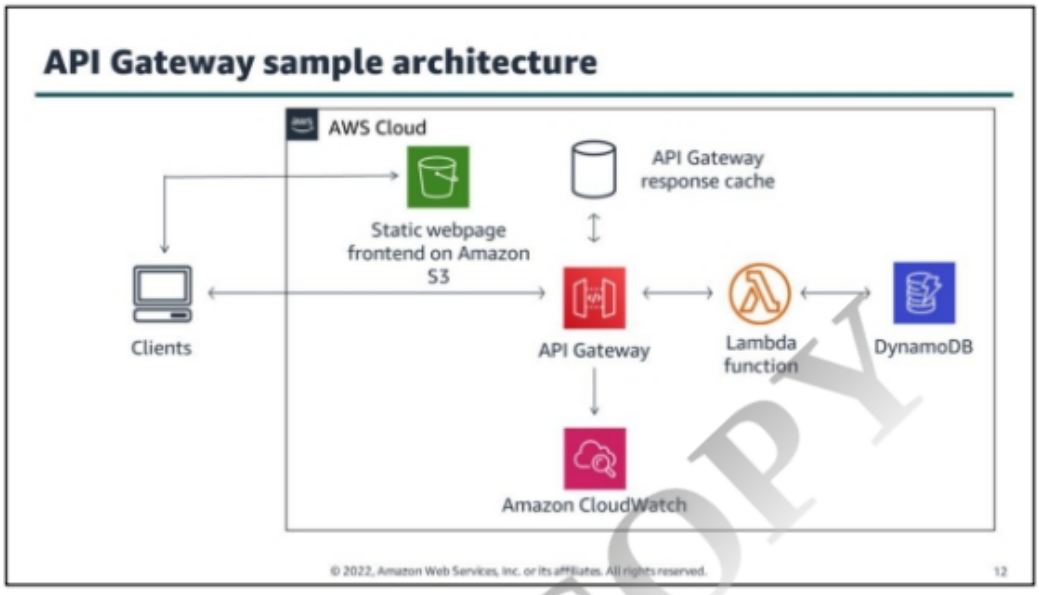
As a gateway, it handles all of the tasks involved in accepting and processing up to hundreds of thousands of concurrent API calls. These include traffic management, authorization and access control, monitoring, and API version management.



With API Gateway, you can do the following:

- Host and use multiple versions and stages of your APIs.
- Create and distribute API keys to developers.
- Use Signature Version 4 (SigV4) to authorize access to APIs.
- Use RESTful or WebSocket APIs.

For more information, see "Signature Version 4 signing process" in the *AWS General Reference* (https://docs.aws.amazon.com/general/latest/gr/signature-version-4.html).



In this example, the client browser requests a static webpage hosted in Amazon S3. Using this webpage, the client browser communicates with API Gateway using a REST API. API Gateway authenticates and authorizes the request, and invokes a Lambda function that communicates with DynamoDB. This example includes the optional API Gateway cache to reduce backend load and reduce latency when serving recurring requests. API Gateway also sends logs to Amazon CloudWatch.

API Gateway can send logs to CloudWatch for each stage in your API or for each method. You can set the verbosity of the logging (Error or Info), and if full request and response data should be logged.

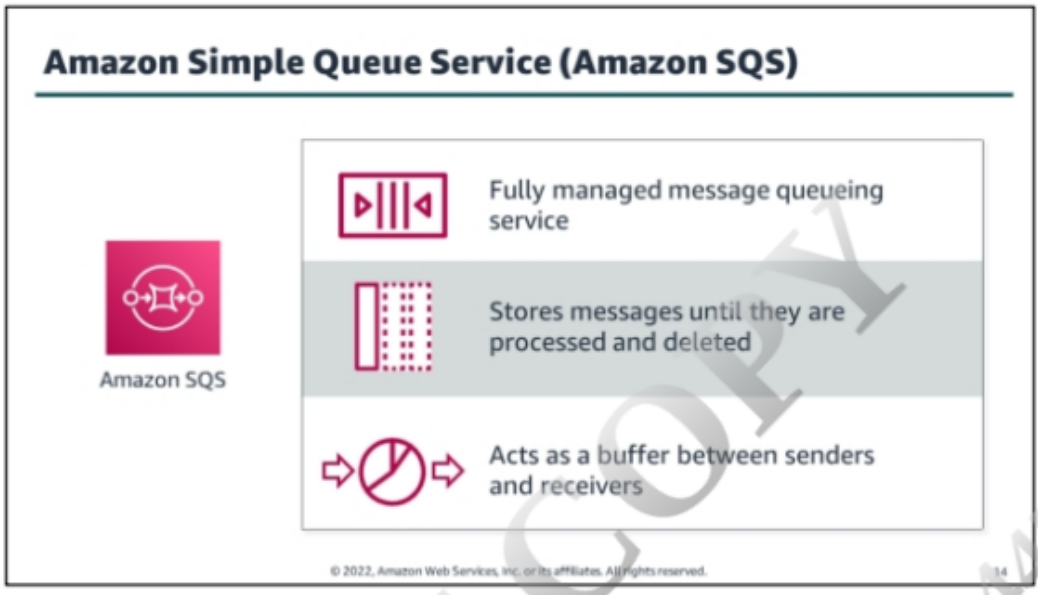
The detailed metrics that API Gateway can send to CloudWatch are the following:

- Number of API calls
- Latency
- Integration latency
- HTTP 400 and 500 errors

You can also activate access logging to log who has accessed your API and how they accessed it.

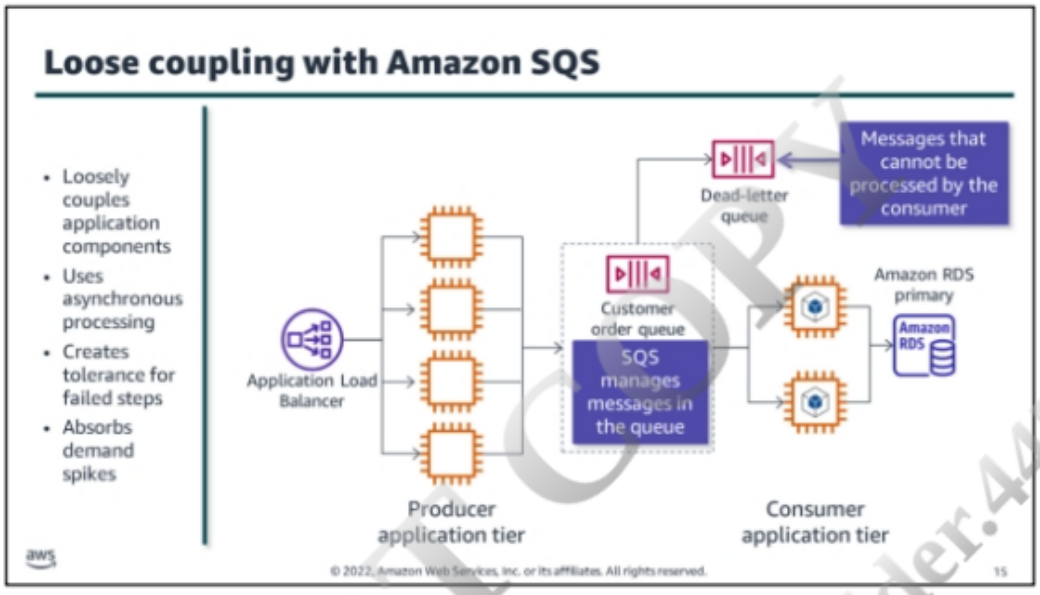
The application development manager asks, "How do we create a message queue for reliable service-to-service communication?"

The company is converting some of their monolithic applications into microservices. The development team has decided that these services can use asynchronous communication to communicate with one another. The company is asking you to recommend a solution for building a message queue.



Amazon Simple Queue Service (Amazon SQS) is a fully managed service that requires no administrative overhead and little configuration. The service works on a massive scale, processing billions of messages per day. It stores all message queues and messages within a single, highly available AWS Region with multiple redundant Availability Zones. This prevents a single computer, network, or Availability Zone failure from making messages inaccessible. You can send and read messages simultaneously.

Developers can securely share SQS queues anonymously or with specific AWS accounts. Queue sharing can also be restricted by IP address and time of day. Streaming Single Instruction Multiple Data (SIMD) Extensions (SSE) protects the contents of messages in SQS queues using keys managed in AWS KMS. SIMD Extensions encrypts messages as soon as Amazon SQS receives them. The messages are stored in encrypted form, and Amazon SQS decrypts messages only when they are sent to an authorized consumer.



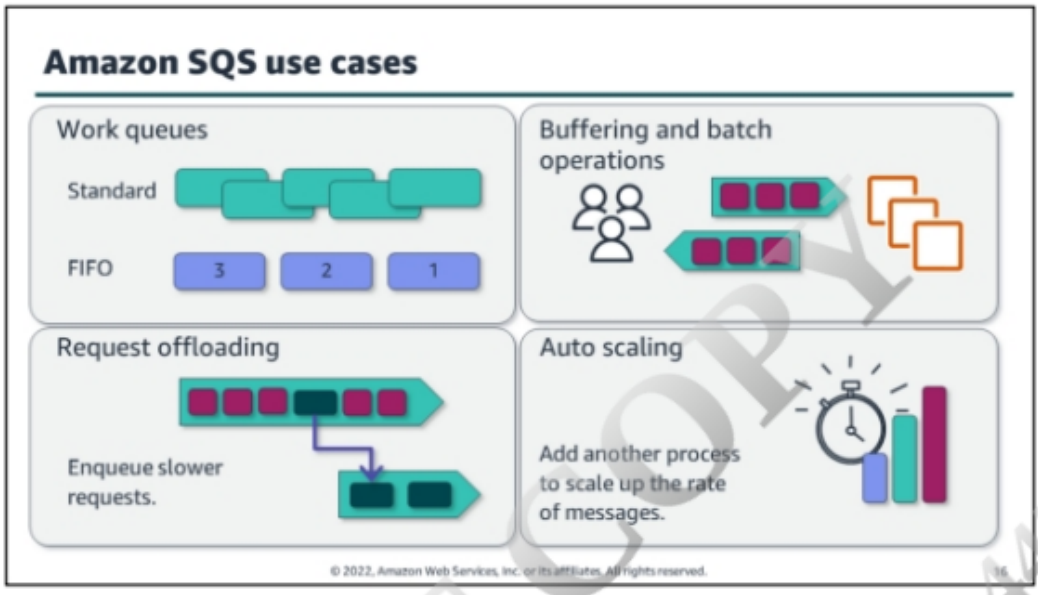
In this example, a producer application creates customer orders and sends them to an Amazon SQS queue. A consumer application processes orders from the producer application tier. The consumer application polls the queue and receives messages. It then records the messages in an Amazon Relational Database Service (Amazon RDS) database and deletes the processed messages from the SQS queue. Amazon SQS sends messages that cannot be processed to a dead-letter queue where they can be processed later.

Using an SQS queue provides the following benefits:

- **Loose coupling** – With Amazon SQS, you can decouple preprocessing steps from compute steps and postprocessing steps. Using asynchronous processing isolates the producer logic into its own component separate from the consumer logic.
- **Absorbs spikes** – An Amazon SQS queue makes the system more resilient. The queue acts as a buffer to absorb spikes in traffic. This gives your application additional time to complete scale-out actions. It is also cost effective because you don't need to provision as much idle compute to absorb spikes.
- **Failure tolerance** – In the event of an application exception or transaction failure, the order processing can be retried. Once the maximum number of retries is reached, SQS can redirect the message to an Amazon SQS dead-letter queue where you can reprocess or debug it later. The loss of one node or job in a loosely coupled workload usually doesn't delay the entire calculation.

For more information about this Amazon SQS use case, see "Building Loosely Coupled, Scalable, C# Applications with Amazon SQS and Amazon SNS" in the *AWS Compute Blog* (https://aws.amazon.com/blogs/compute/building-loosely-coupled-scalable-c-applications-with-amazon-sqs-and-amazon-sns/).

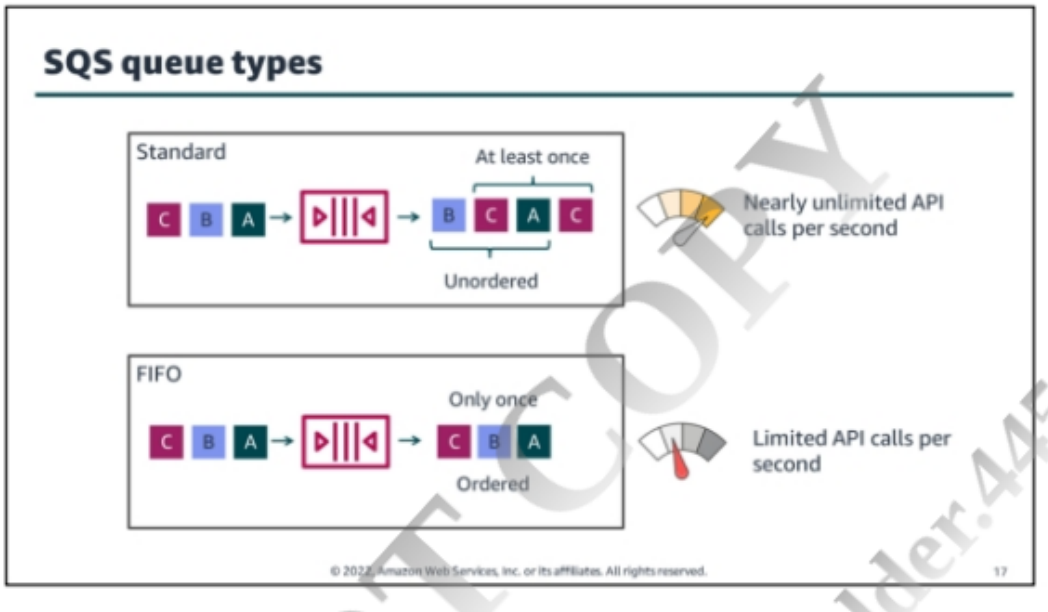
For more information about loosely coupled architectures, see "Loosely Coupled Scenarios" in the *High Performance Computing Lens: AWS Well-Architected Framework* guide (https://docs.aws.amazon.com/wellarchitected/latest/high-performance-computing-lens/loosely-coupled-scenarios.html).



There are many different ways to use SQS queues.

Review the following use cases:

- **Work queues** – Decouple components of a distributed application that might not all process the same amount of work simultaneously. You can choose a standard queue or a First-In-First-Out (FIFO) queue depending on the requirements of your application.
- **Buffering and batch operations** – Add scalability and reliability to your architecture and smooth out temporary volume spikes without losing messages or increasing latency.
- **Request offloading** – Move slow operations off of interactive request paths by enqueueing the request.
- **Auto scaling instances** – Use SQS queues to help determine the load on an application. When combined with auto scaling, you can scale the number of Amazon EC2 instances in or out, depending on the volume of traffic.



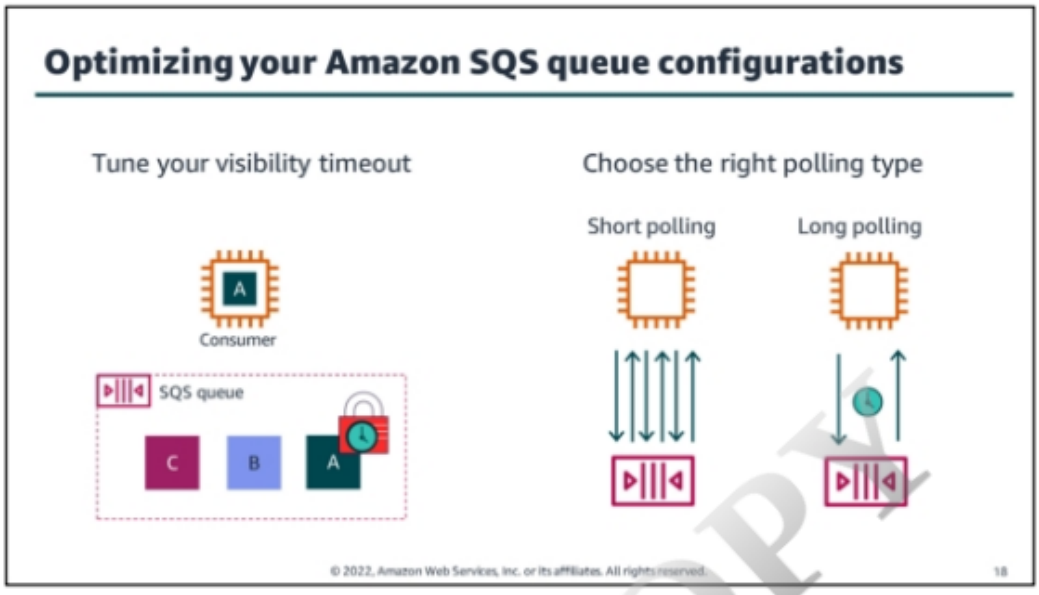
Amazon SQS offers two types of message queues.

Standard queues support at-least-once message delivery and provide best-effort ordering. Messages are generally delivered in the same order in which they are sent. However, because of the highly distributed architecture, more than one copy of a message might be delivered out of order. Standard queues can handle a nearly unlimited number of API calls per second. You can use standard message queues if your application can process messages that arrive more than once and out of order.

FIFO queues are designed to enhance messaging between applications when the order of operations and events is critical or where duplicates can't be tolerated. FIFO queues also provide exactly-once processing, but have a limited number of API calls per second. FIFO queues are designed to enhance messaging between applications when the order of operations and events is critical.

For more information about Amazon SQS standard queues, see "Amazon SQS Standard queues" in the *Amazon Simple Queue Service Developer Guide* (http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/standard-queues.html).

For more information about Amazon SQS FIFO, see "Amazon SQS FIFO (First-In-First-Out) queues" in the *Amazon Simple Queue Service Developer Guide* (http://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/FIFO-queues.html).



When creating an Amazon SQS queue, you need to consider how your application interacts with the queue. This information will help you optimize the configuration of your queue to control costs and increase performance.

Visibility timeout

When a consumer receives an SQS message, that message remains in the queue until the consumer deletes it. You can configure the SQS queue's *visibility timeout* setting to make that message invisible to other consumers for a period of time. This helps to prevent another consumer from processing the same message. The default visibility timeout is 30 seconds. The consumer deletes the message once it completes processing the message. If the consumer fails to delete the message before the visibility timeout expires, it becomes visible to other consumers and can be processed again.

Typically, you should set the visibility timeout to the maximum time that it takes your application to process and delete a message from the queue. Setting too short of a timeout increases the possibility of your application processing a message twice. Too long of a visibility timeout delays subsequent attempts at processing a message.

Polling type

You can configure an Amazon SQS queue to use either short polling or long polling.

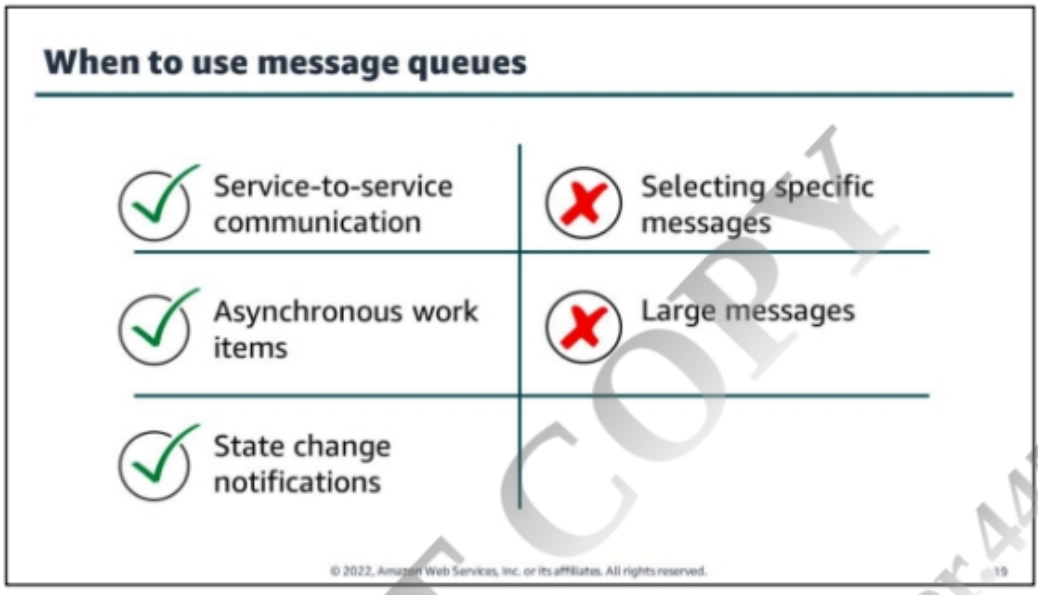
SQS queues with short polling:

- Sends a response to the consumer immediately after receiving a request providing a faster response
- Increases the number of responses and therefore costs

SQS queues with long polling:

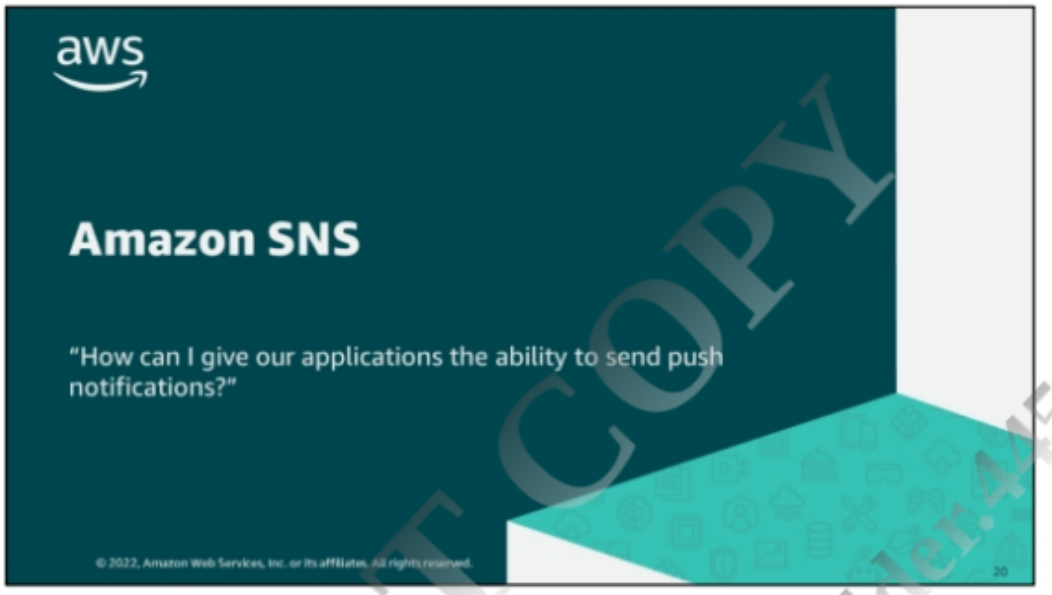
- Does not return a response until at least one message arrives or the poll times out
- Provides less frequent responses but decreases costs

Depending on the frequency of messages arriving in your queue, many of the responses from a queue using short polling could just be reporting an empty queue. Unless your application requires an immediate response to its poll requests, long polling is the preferable option.



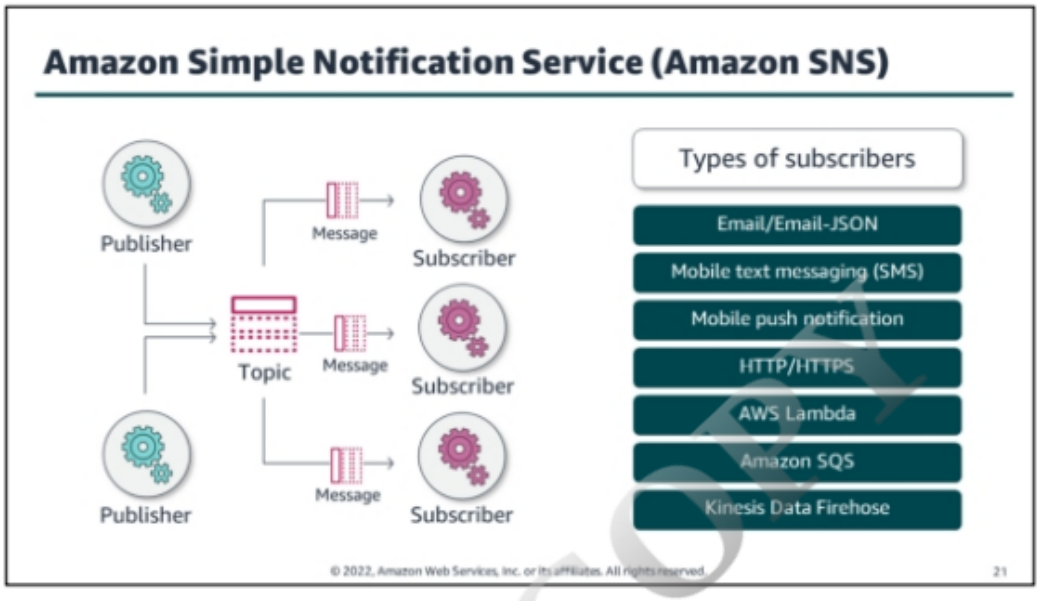
It's important to know when a particular technology won't fit well with your use case. Messaging has its own set of commonly encountered anti-patterns. It's tempting to retrieve messages selectively that match a particular set of attributes, or even match a one-time logical query. For example, a service requests a message with a particular attribute because it contains a response to another message that the service sent out. This can lead to a scenario where there are messages in the queue that no one is polling for and are never consumed.

Most messaging protocols and implementations work best with reasonably sized messages (in the tens or hundreds of kilobytes). As message sizes grow, it's best to use a dedicated storage system, such as Amazon S3, and pass a reference to an object in that store in the message itself.



The application development manager asks, "How can we give our applications the ability to send push notifications?"

The company would like to notify users when certain events occur, such as application updates or promotional messages. They would like to deliver these messages using email and text messaging. The company is asking you to research how this can most easily be done in the AWS Cloud.



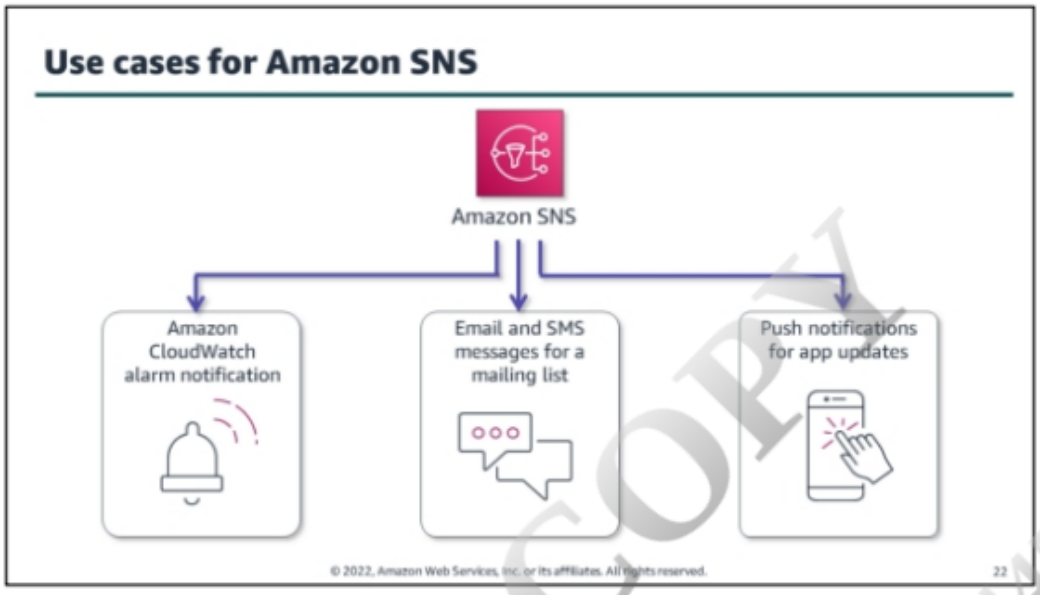
Amazon Simple Notification Service (Amazon SNS) is a web service that makes it easy to set up, operate, and send notifications from the cloud. The service follows the *publish-subscribe* (pub-sub) messaging paradigm, with notifications being delivered to clients using a *push* mechanism.

You create a topic and control access to it by defining policies that determine which publishers and subscribers can communicate with the topic. A publisher sends messages to topics they have created, or to topics to which they have permission to publish.

Instead of including a specific destination address in each message, a publisher sends a message to the topic. Amazon SNS matches the topic to a list of subscribers for that topic, and delivers the message to each subscriber.

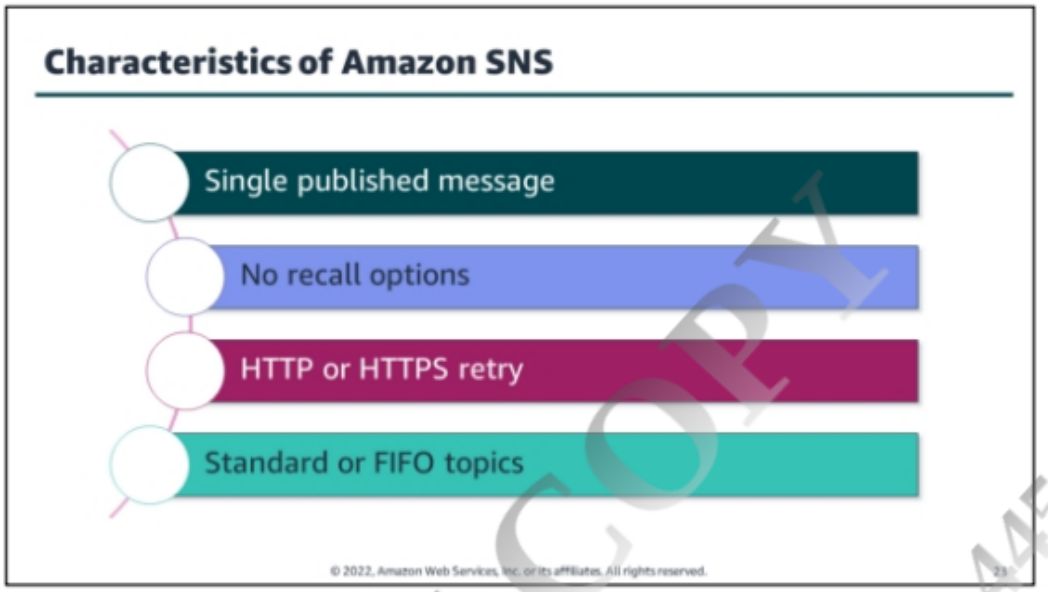
Each topic has a unique name that identifies the Amazon SNS endpoint where publishers post messages, and where subscribers register for notifications. Subscribers receive all messages published to their subscribed topics. All subscribers to a topic receive the same messages.

Amazon SNS supports encrypted topics. When you publish messages to encrypted topics, Amazon SNS uses AWS Key Management Service (AWS KMS) keys to encrypt your messages.



You can use Amazon SNS notifications in many ways, for example, the following:

- You can receive immediate notification when an event occurs, such as a specific change to your Auto Scaling group.
- You can push targeted news headlines to subscribers by email or SMS. Upon receiving the email or SMS text, interested readers can choose to learn more by visiting a website or launching an application.
- You can send notifications to an app, indicating that an update is available. The notification message can include a link to download and install the update.

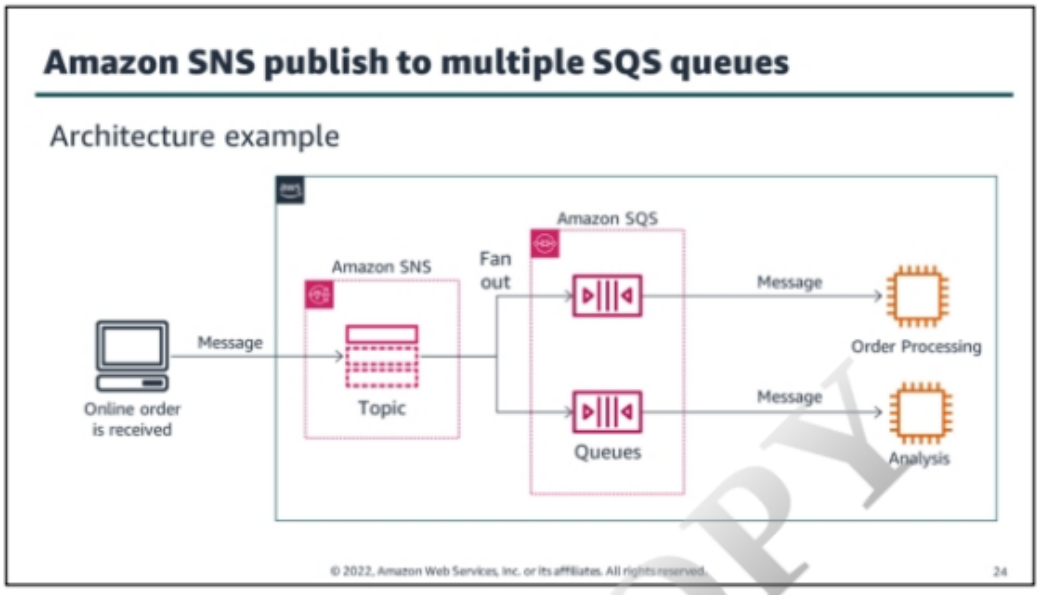


All notification messages contain a single published message.

Amazon SNS will attempt to deliver messages from the publisher in the order they were published into the topic. However, network issues could potentially result in out-of-order messages at the subscriber end. Use Amazon SNS FIFO topics if you require strict message ordering and deduplicated message delivery to one or more subscribers.

Characteristics of Amazon SNS:

- When a message is delivered successfully, there is no way to recall it.
- You can use an Amazon SNS Delivery Policy to control the retry pattern: linear, geometric, exponential backoff, maximum and minimum retry delays, and other patterns.
- To prevent messages from being lost, all messages are stored redundantly across multiple servers and data centers.
- Amazon SNS is designed to meet the needs of the largest and most demanding applications, allowing applications to publish a large number of messages at any time.
- Amazon SNS allows applications and end users on different devices to receive notifications by Mobile Push notification. This includes Apple, Google, and Kindle Fire Devices; HTTP or HTTPS; email or email-JSON; SMS or Amazon SQS queues; or Lambda functions.
- Amazon SNS provides access control mechanisms to protect topics and messages from unauthorized access. Topic owners can set policies for a topic that restrict who can publish or subscribe to a topic. Topic owners can encrypt notifications by specifying that the delivery mechanism must be HTTPS.



Using highly available services such as Amazon SNS to perform basic message routing is an effective way of distributing messages to microservices. The two main forms of communications between microservices are *request-response* and *observer*.

This example uses the observer type and illustrates a *fan-out* scenario using Amazon SNS and Amazon SQS. In a *fan-out* scenario, a message is sent to an SNS topic and then replicated and pushed to multiple SQS queues, HTTP endpoints, or email addresses. This allows for parallel asynchronous processing.

In this example, Amazon SNS fans out orders to two different SQS queues. Two Amazon EC2 instances each observe a queue. One of the instances handles the processing or fulfillment of the order, while the other is attached to a data warehouse for analysis of all orders received.

To deliver Amazon SNS notifications to an SQS queue, you subscribe to a topic and specify Amazon SQS as the transport and a valid SQS queue as the endpoint. To permit the SQS queue to receive notifications from Amazon SNS, the SQS queue owner must subscribe the SQS queue to the topic for Amazon SNS. If the user owns the Amazon SNS topic being subscribed to and the SQS queue receiving the notifications, nothing else is required. Any message published to the topic will automatically be delivered to the specified SQS queue. If the owner of the SQS queue is not the owner of the topic, Amazon SNS requires an explicit confirmation to the subscription request.

Amazon SNS and Amazon SQS

Features	Amazon SNS	Amazon SQS
Message persistence	No	Yes
Delivery mechanism	Push (passive)	Poll (active)
Producer and consumer	Publisher and subscriber	Send or receive
Distribution model	One to many	One to one

 25

Amazon SNS messages are not persistent. Amazon SNS defines a delivery policy for each delivery protocol. The delivery policy defines how Amazon SNS retries the delivery of messages when server-side errors occur. When the delivery policy is exhausted, Amazon SNS stops retrying the delivery and discards the message unless a dead-letter queue is attached to the subscription.

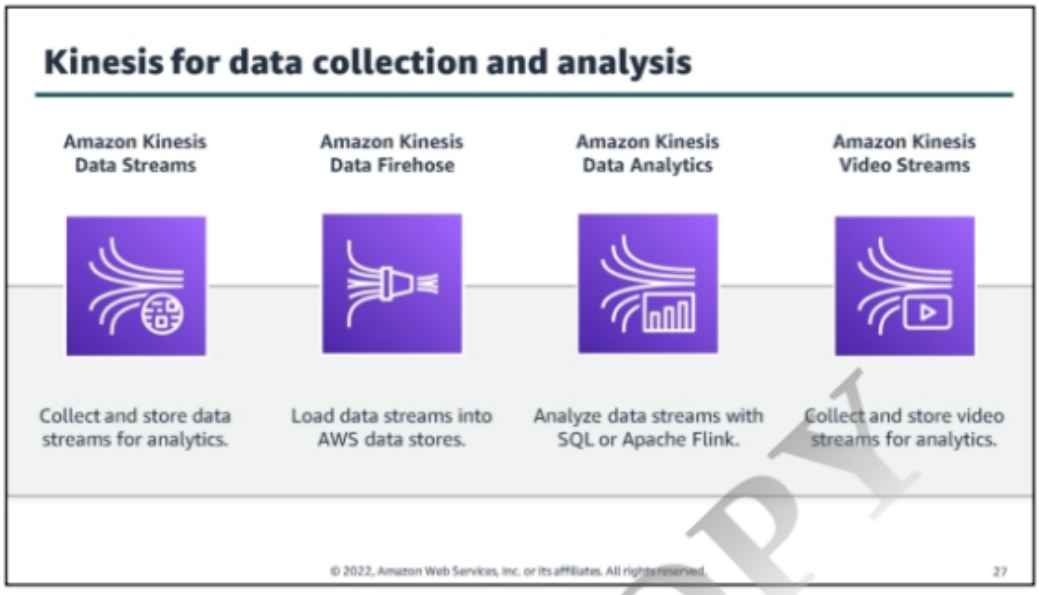
Amazon SNS permits applications to send time-critical messages to multiple subscribers through a push mechanism.

Amazon SQS exchanges messages through a polling model: sending and receiving components are decoupled.

Amazon SQS provides flexibility for distributed components of applications to send and receive messages without requiring each component to be concurrently available.

The application development manager asks, "How do we ingest streaming data to power our real-time applications?"

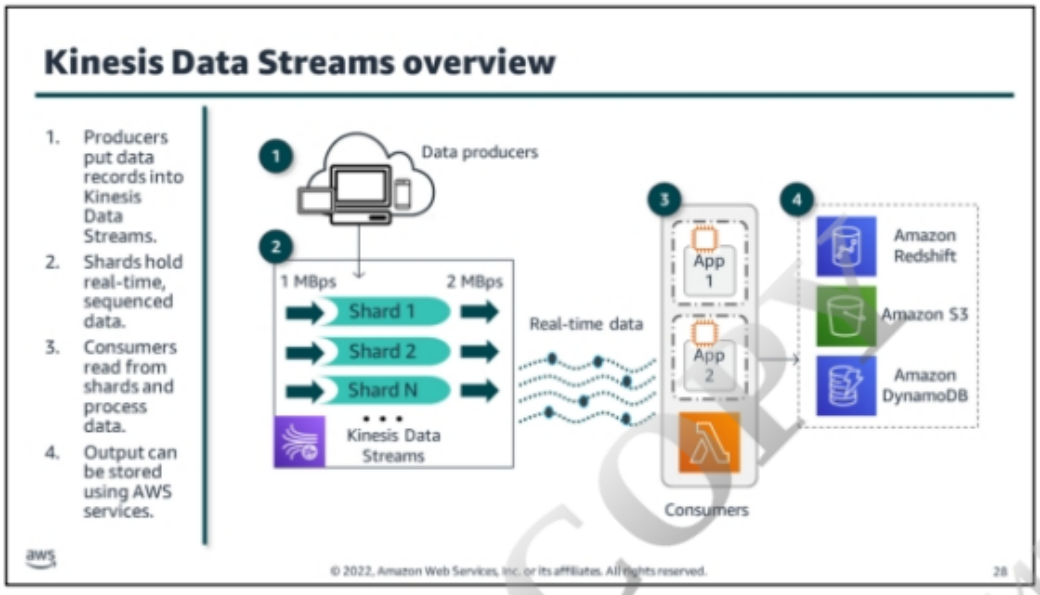
The company would like to capture clickstream data from their web applications and perform real-time analytics. The company is asking you identify a solution for ingesting streaming data.



With Amazon Kinesis, you can do the following:

- Collect, process, and analyze data streams in real time. Kinesis has the capacity to process streaming data at any scale. It provides you the flexibility to choose the tools that best suit the requirements of your application in a cost-effective way.
- Ingest real-time data such as video, audio, application logs, website clickstreams, and Internet of Things (IoT) telemetry data. The ingested data can be used for machine learning, analytics, and other applications.
- Process and analyze data as it arrives and respond instantly. You don't have to wait until all data is collected before the processing begins.

This module describes Amazon Kinesis Data Streams and Amazon Kinesis Data Firehose. For more information about Amazon Kinesis Data Analytics and Amazon Kinesis Video Streams, explore "Amazon Kinesis" (https://aws.amazon.com/kinesis/).



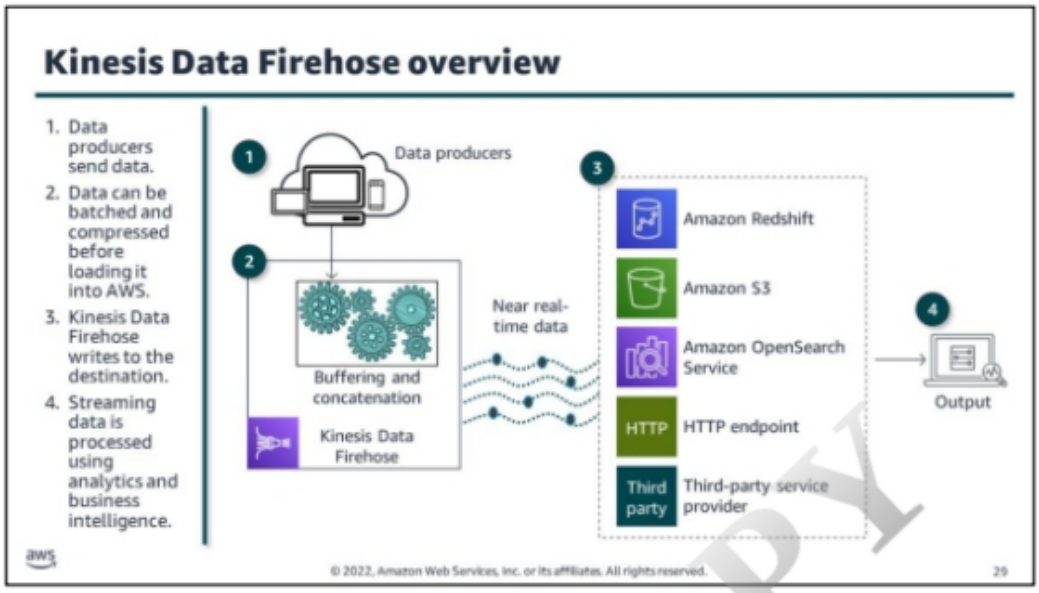
To get started using Kinesis Data Streams, create a stream and specify the number of shards. Each shard is a uniquely identified sequence of data records in a stream. Your stream can receive 1 MB per second per shard. Each shard has a read limit of 2 MB per second for your applications. The total capacity of a stream is the sum of the capacities of its shards. Use resharding to increase or decrease the number of shards in your stream as needed.

Producers write data into the stream. A producer might be an EC2 instance, a mobile client, an on-premises server, or an IoT device. You can send data such as infrastructure logs, application logs, market data feeds, and web clickstream data.

Consumers read the streaming data that the producers generate. A consumer might be an application running on an EC2 instance or AWS Lambda. An application on an EC2 instance will need to scale as the amount of streaming data increases. If this is the case, run it in an Auto Scaling group. Another way to write a consumer application is to use a Lambda function, which you can use to run code without having to provision or manage servers. There can be more than one application processing the same data in the stream.

The results of the consumer applications can be stored using AWS services such as Amazon S3, Amazon DynamoDB, and Amazon Redshift.

For more information about what you can do with Kinesis Data Streams and the benefits, see "What is Amazon Kinesis Data Streams?" in the *Amazon Kinesis Data Streams Developer Guide* (https://docs.aws.amazon.com/streams/latest/dev/introduction.html).



Kinesis Data Firehose starts to process data in near real time. Kinesis Data Firehose can send records to Amazon S3, Amazon Redshift, Amazon OpenSearch Service, and any HTTP endpoint owned by you. It can also send records to any of your third-party service providers, including Datadog, New Relic, and Splunk.

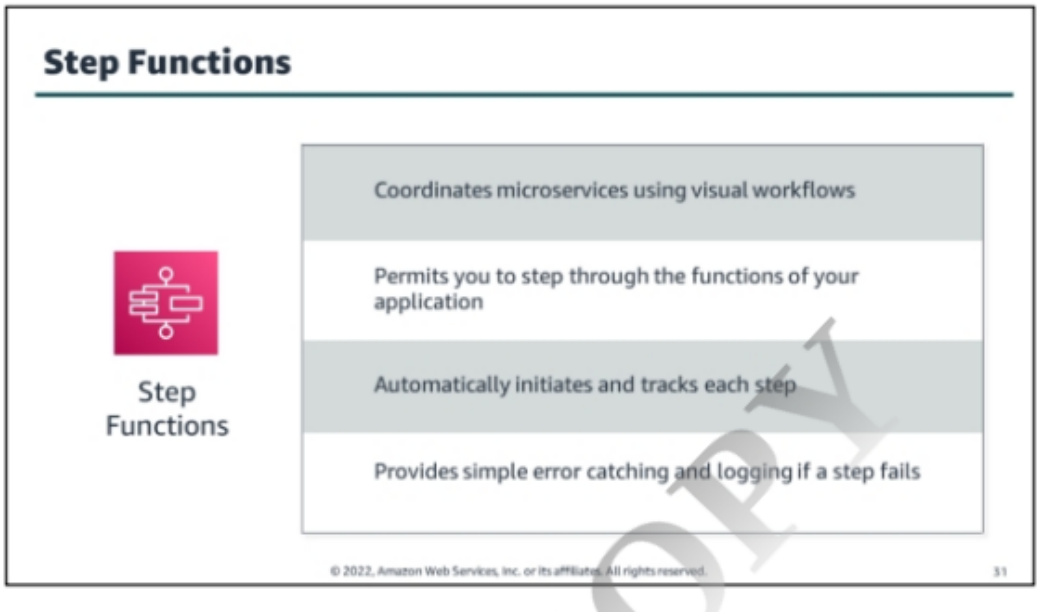
For data delivery to Amazon S3, Kinesis Data Firehose concatenates multiple incoming records based on the buffering configuration of your delivery stream. It then delivers the concatenated record to Amazon S3 as an S3 object.

To deliver data to Amazon Redshift, Kinesis Data Firehose does the following:
1. Delivers incoming data to your S3 bucket in the format described earlier
2. Issues an Amazon Redshift COPY command to load the data from your S3 bucket to your Amazon Redshift cluster

Amazon OpenSearch Service is a fully managed service that delivers OpenSearch APIs and real-time analytics capabilities. With the processed data, you can produce near real-time analytics using your existing business intelligence tools and dashboards. With Amazon OpenSearch Service, Kinesis Data Firehose buffers incoming records based on the buffering configuration of your delivery stream. Then Kinesis Data Firehose generates a bulk request to index multiple records to your Amazon OpenSearch Service cluster.

The application development manager asks, "What is an easy way to orchestrate multi-step workflows?"

The company Is rearchitecting some of its monolithic applications and decoupling their logic into individual functions and services. They are looking for ways to coordinate workflows through these newly independent components. The company is asking you to identify a tool that facilitates workflows between functions and services.

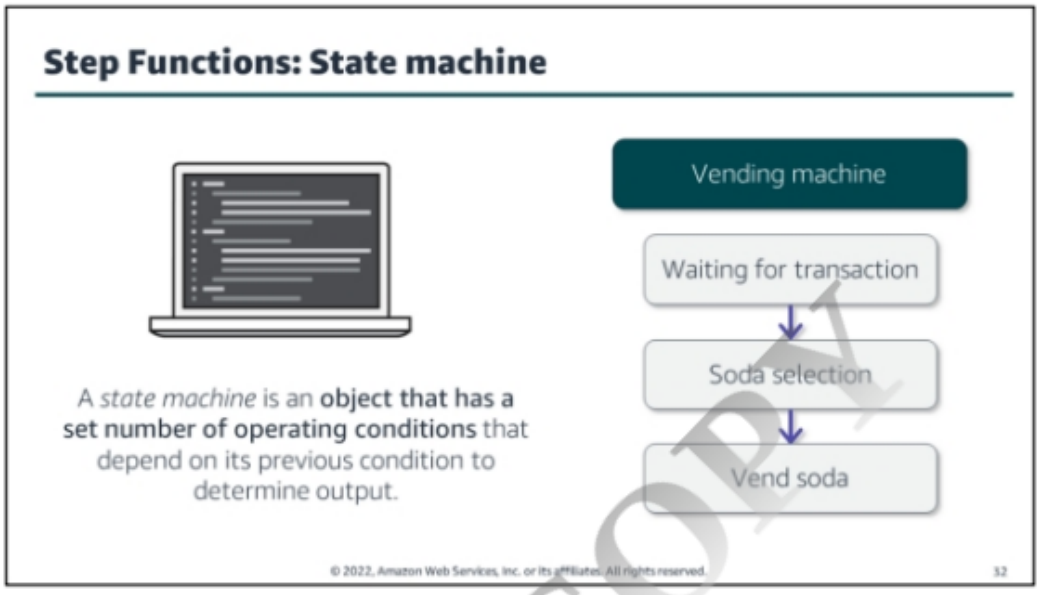


It's common for modern cloud applications to be composed of many services and components. As applications grow, an increasing amount of code must be written to coordinate the interaction of all components. With Step Functions, you can focus on defining the component interactions rather than writing all of the software to make the interactions work.

Step Functions integrates with the following AWS services. You can directly call API actions in Step Functions and pass parameters to the APIs of these services.

- Compute services: AWS Lambda, Amazon Elastic Container Service (Amazon ECS), Amazon Elastic Kubernetes Service (Amazon EKS), and AWS Fargate
- Database services: Amazon DynamoDB
- Messaging services: Amazon SNS and Amazon SQS
- Data processing and analytics services: Amazon Athena, AWS Batch, AWS Glue, Amazon EMR, and AWS Glue DataBrew
- Machine learning services: SageMaker
- APIs created by API Gateway

You can configure your Step Functions workflow to call other AWS services using Step Functions service tasks. For more information on AWS service integrations, explore "Call other AWS services" in the *AWS Step Functions Developer Guide* (https://docs.aws.amazon.com/step-functions/latest/dg/connect-to-services.html).



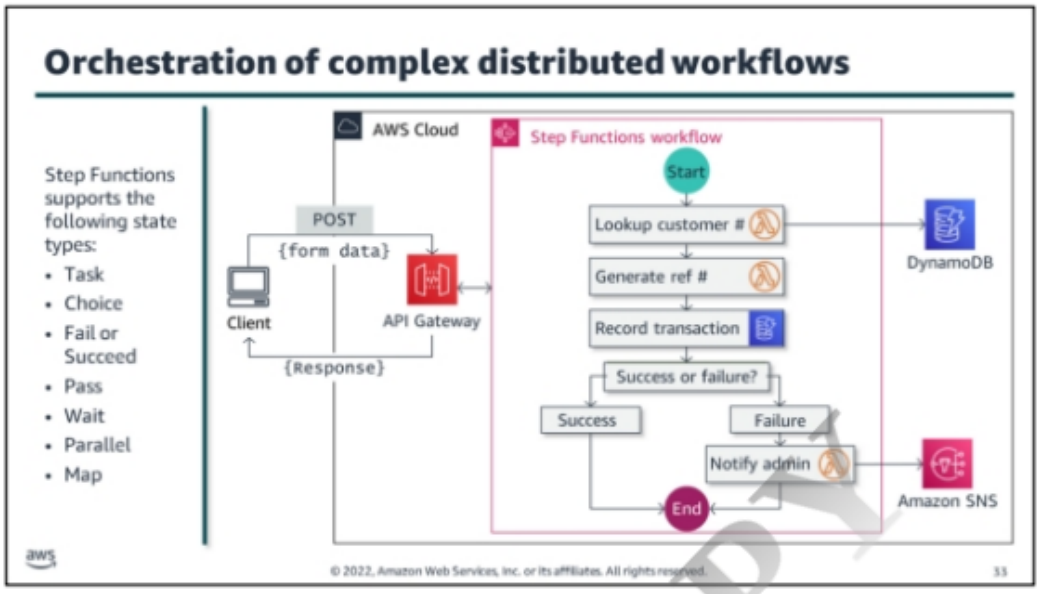
A *state machine* is an object that has a set number of operating conditions that depend on its previous condition to determine output.

A common example of a state machine is the soda vending machine. The machine starts in the operating state (waiting for a transaction), and then moves to soda selection when money is added. After that, it enters a vending state, where the soda is deployed to the customer. After completion, the state returns to operating.

With Step Functions, you can create and automate your own state machines within the AWS environment. It does this with the use of a JSON-based Amazon States Language, which contains a structure made of various states, tasks, choices, error handling, and more.

For more information about states, see "States" in the *AWS Step Functions Developer Guide* (https://docs.aws.amazon.com/step-functions/latest/dg/concepts-states.html).

For more information about Amazon States Language, see "Amazon States Language" (https://states-language.net/spec.html).



States are elements in your state machine. States can perform a variety of functions in your state machine. A state can do the following:

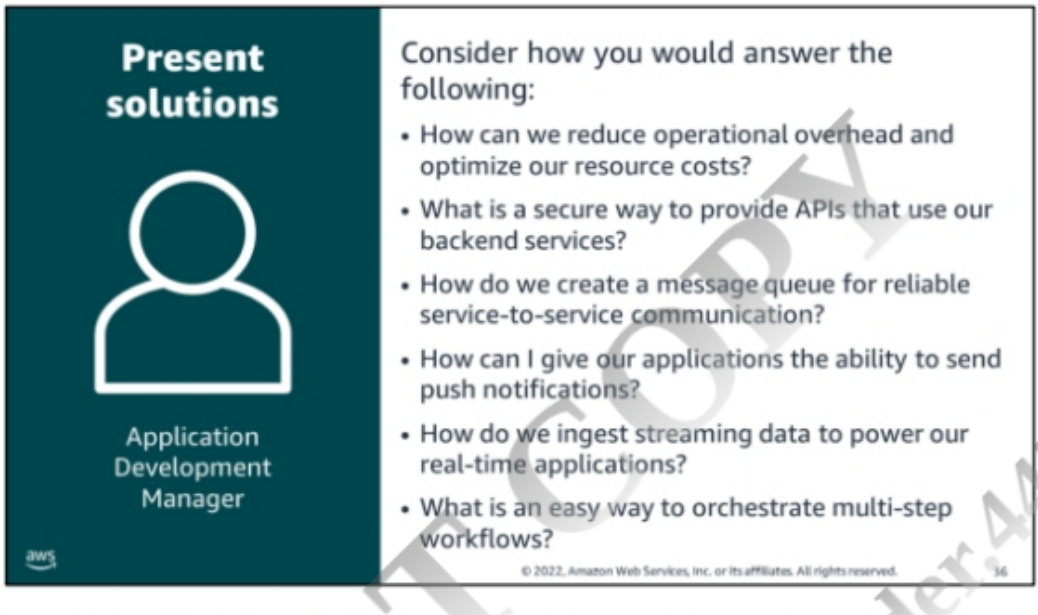
- Do some work in your state machine (a Task state).
- Make a choice between different branches to run (a Choice state).
- Stop with a failure or success (a Fail or Succeed state).
- Pass its input to its output or inject some fixed data (a Pass state).
- Provide a delay for a certain amount of time or until a specified time or date (a Wait state).
- Begin parallel branches (a Parallel state).
- Dynamically iterate steps (a Map state).

With Step Functions you can create two types of workflows, Standard and Express.

- You use the *Standard Workflows* type for long-running, durable, and auditable workflows. These workflows can run for up to a year and you can access the full history of workflow activity for up to 90 days after a workflow completes. Standard Workflows use an exactly-once model, where your tasks and states are never run more than once unless you specify a Retry behavior.
- You use the *Express Workflows* type for high-volume, event-processing workloads such as IoT data ingestion, streaming data processing and transformation, and mobile application backends. They can run for up to five minutes. Express Workflows use an at-least-once model, where there is a possibility that an execution might be run more than once.

In this example, a client completes a transaction, which sends a POST request to API Gateway. This event in API Gateway initiates a Step Functions workflow to record the transaction in DynamoDB. The workflow moves through each state until it completes the workflow and ends.

For more information about Express Workflows, see "Standard vs. Express Workflows" in the *AWS Step Functions Developer Guide* (https://docs.aws.amazon.com/step-functions/latest/dg/concepts-standard-vs-express.html).

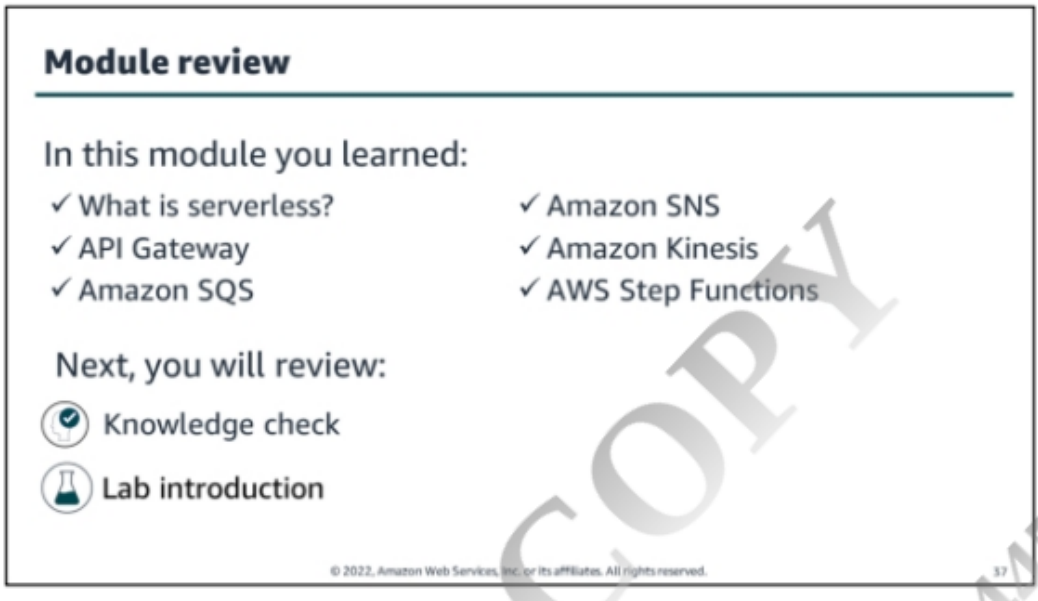


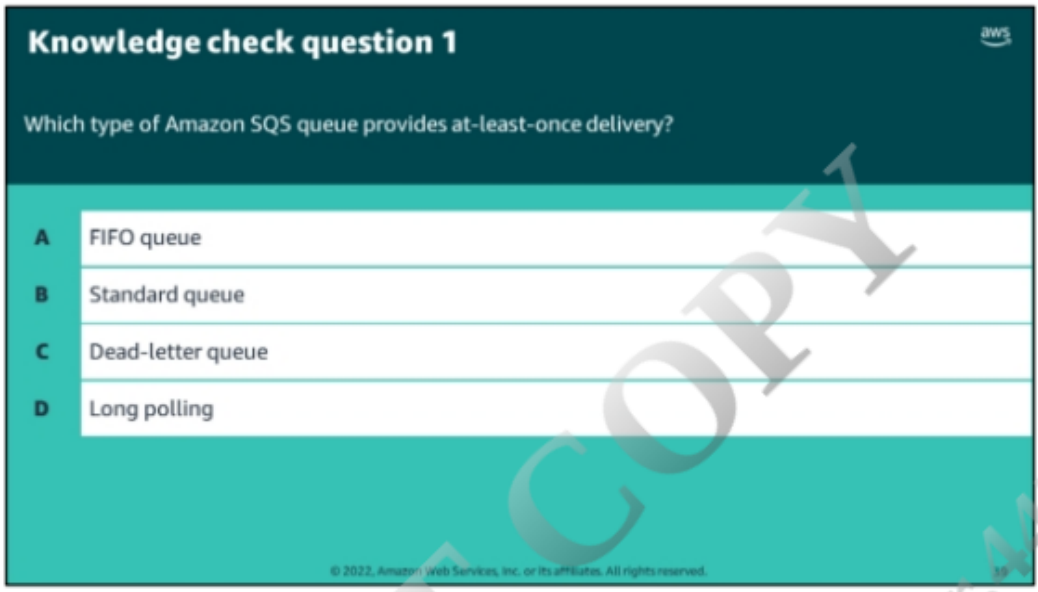
Imagine you are now ready to talk to the application development manager and present solutions that meet their architectural needs.

Think about how you would answer the questions from the beginning of the lesson.

Your answers should include the following solutions:

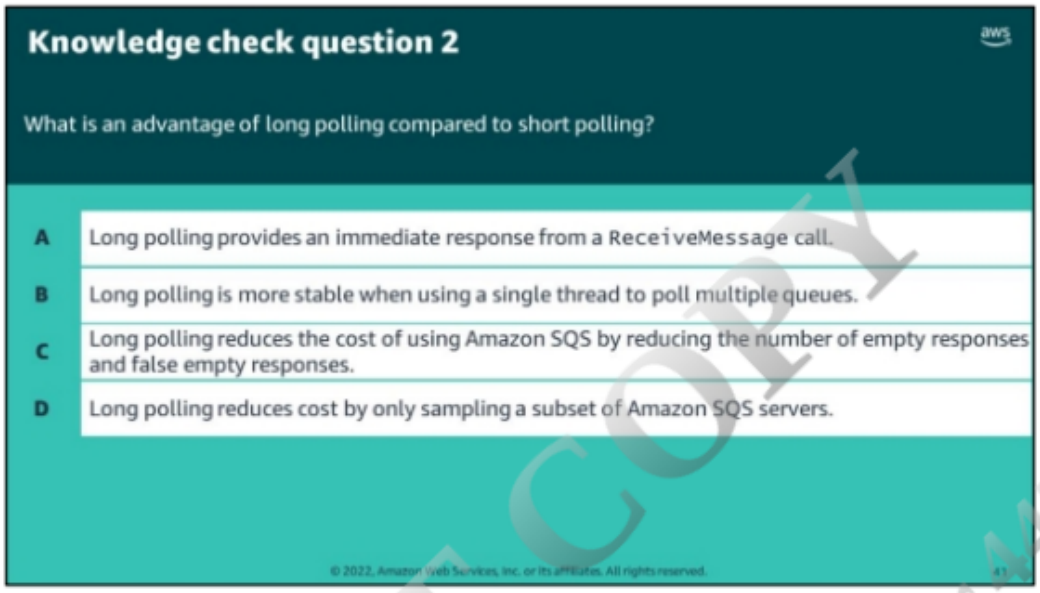
- One way to reduce operational overhead and costs is to adopt serverless architectures. With serverless you do not have to manage or provision infrastructure. It can scale automatically and you can control costs by paying only for the resources you use.
- You can use API Gateway to provide both REST APIs and WebSocket APIs that interact with your backend services. API Gateway can accept and process up to hundreds of thousands of concurrent API calls. It also handles traffic management, cross-origin resource sharing (CORS) support, authorization and access control, throttling, monitoring, and API version management.
- With Amazon SQS you can send, store, and receive messages between software components. Because Amazon SQS is a managed service, you do not have to manage and operate message-oriented middleware.
- You can use Amazon SNS for application-to-application and application-to-person communication. Amazon SNS supports messaging between distributed systems and event-driven architectures. It can also send messages to users at scale over SMS, mobile push, and email.
- You can use Amazon Kinesis to collect, process, and analyze real-time, streaming data.
- With Step Functions, you can build stateful workflows that connect services and systems. Step Functions integrates easily with many AWS services.

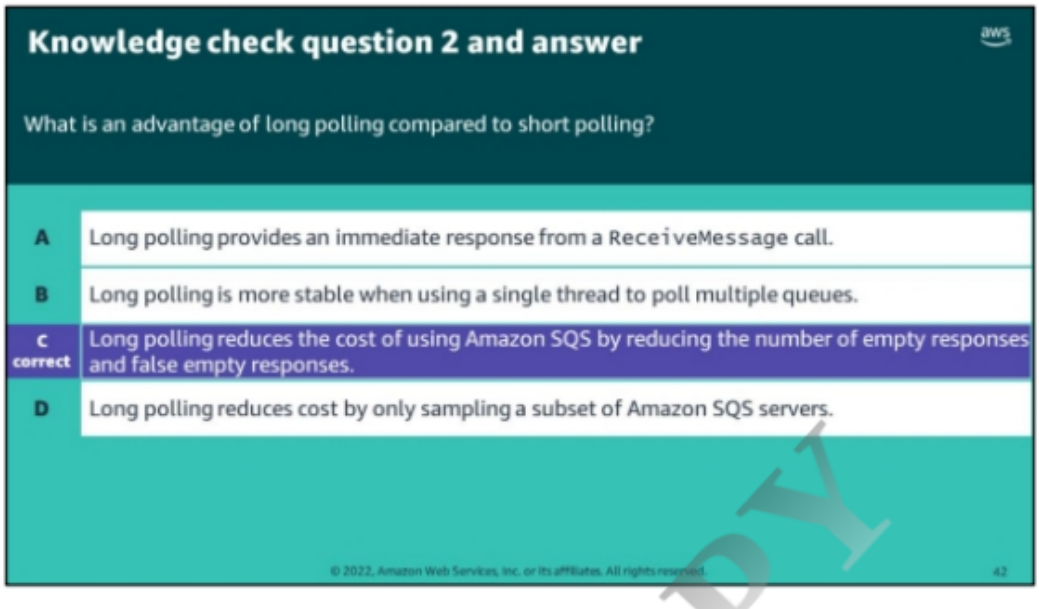




The correct answer is B, standard queue.

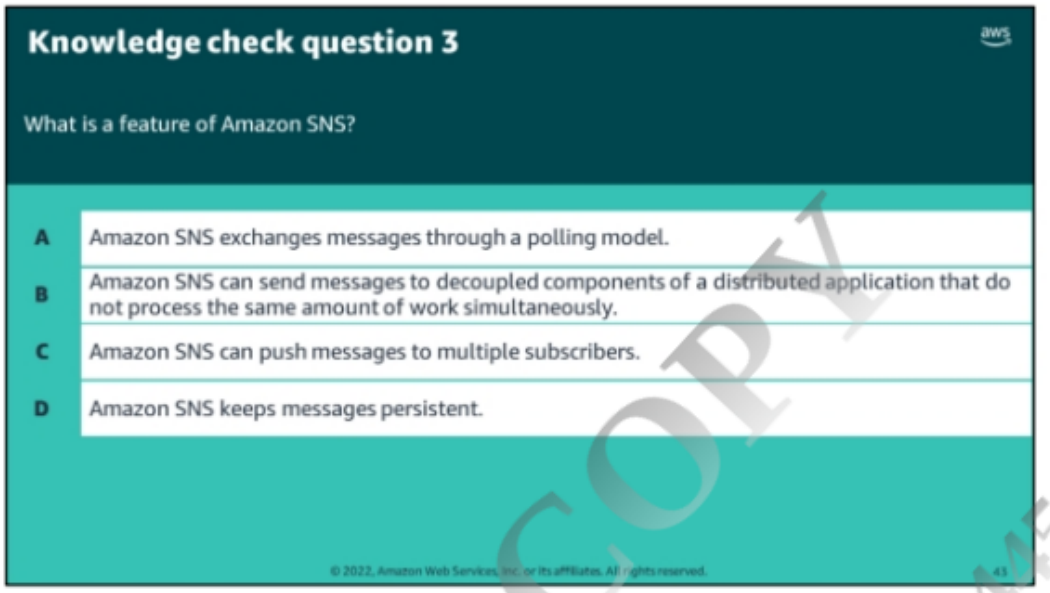
Standard queues support at-least-once message delivery and provide best-effort ordering. Messages are generally delivered in the same order in which they are sent. However, because of the highly distributed architecture, more than one copy of a message might be delivered out of order.

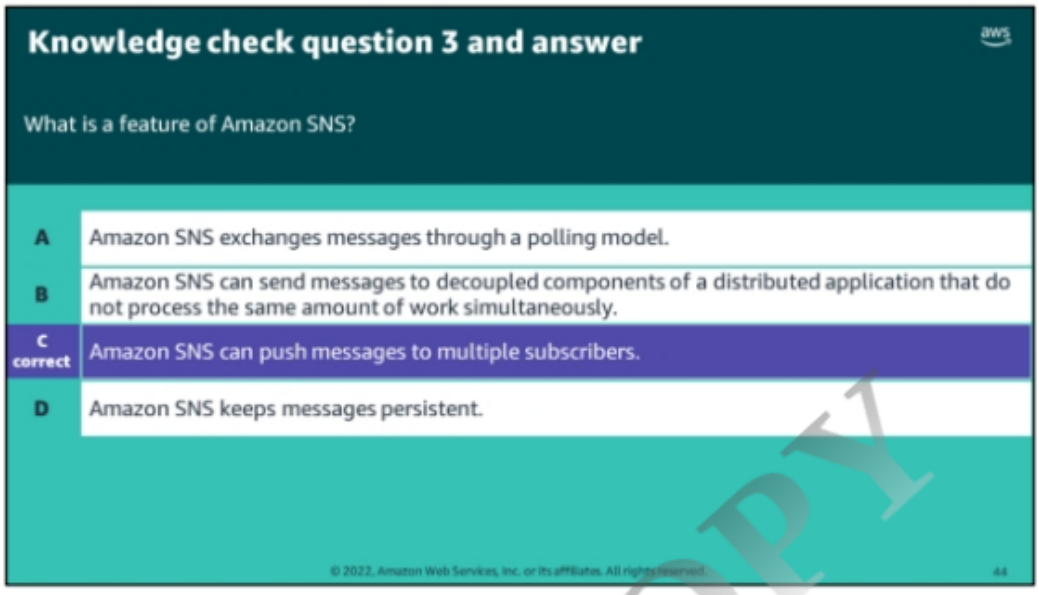




The correct answer is C, long polling reduces the cost of using Amazon SQS by reducing the number of empty responses and false empty responses.

Long polling doesn't return a response until a message arrives in the queue or the long poll times out. Long polling reduces the cost of using Amazon SQS by reducing the number of empty responses and false empty responses.

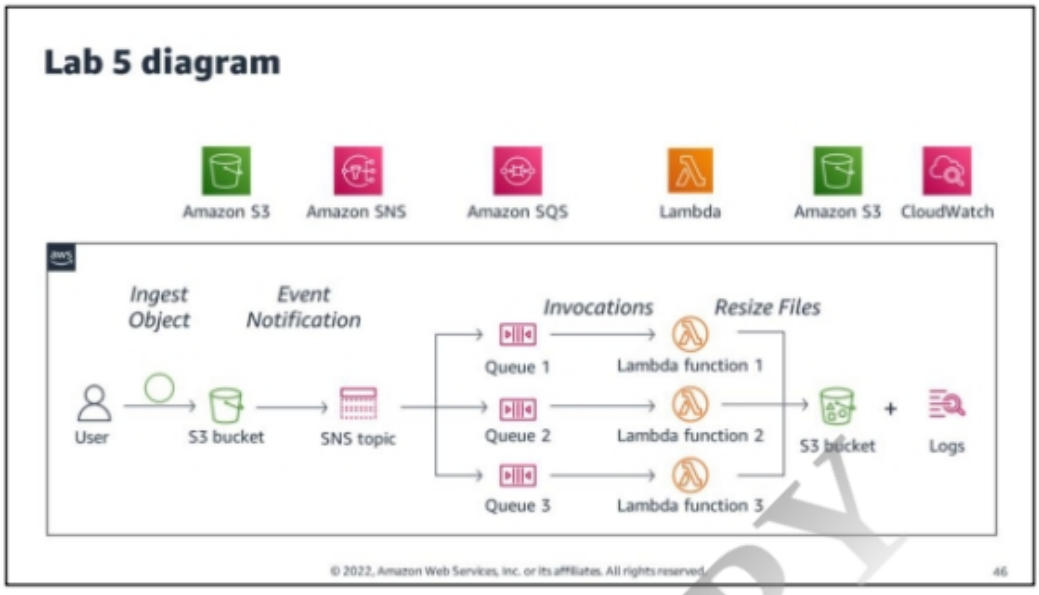




The correct answer is C, Amazon SNS can push messages to multiple subscribers.

Amazon SNS can push messages to multiple subscribers. Each account can support 1,000 FIFO topics, and each topic supports up to 100 subscriptions.

For more information, explore "Amazon SNS features" (https://aws.amazon.com/sns/features/).



In this lab, you create a serverless architecture. When a user adds an image to an Amazon S3 bucket, the event is published to an SNS topic. Three SQS queues are subscribed to the topic. Each queue has a Lambda function that is invoked when its SQS queue sends a message. Each Lambda function resizes the image and adds it to another S3 bucket. These events are logged in Amazon CloudWatch.