Chapter 8
SQS, SWF, and SNS

THE AWS CERTIFIED SOLUTIONS ARCHITECT ASSOCIATE EXAM OBJECTIVES COVERED IN THIS CHAPTER MAY INCLUDE, BUT ARE NOT LIMITED TO, THE FOLLOWING:

1 Domain 1.0: Designing highly available, cost-efficient, fault-tolerant, scalable systems

✓ **1.1 Identify and recognize cloud architecture considerations, such as fundamental components and effective designs.**

Content may include the following:

- How to design cloud services
- Planning and design
- Monitoring and logging
- Familiarity with:
  - Best practices for AWS architecture
  - Architectural trade-off decisions (e.g., high availability vs. cost, Amazon Relational Database Service [Amazon RDS] vs. installing your own database on Amazon Elastic Compute Cloud [Amazon EC2])
  - Elasticity and scalability (e.g., Auto Scaling, Amazon Simple Queue Service [Amazon SQS], Elastic Load Balancing, Amazon CloudFront)

Domain 2.0: Implementation/Deployment

✓ **2.1 Identify the appropriate techniques and methods using Amazon EC2, Amazon Simple Storage Service (Amazon S3), AWS Elastic Beanstalk, AWS CloudFormation, AWS OpsWorks, Amazon VPC, and AWS Identity and Access Management (IAM) to code and implement a cloud solution.**

Domain 4.0: Troubleshooting

Content may include the following:

- General troubleshooting information and questions

There are a number of services under the Application and Mobile Services section of the AWS Management Console. At the time of writing this chapter, application

services include Amazon Simple Queue Service (Amazon SQS), Amazon Simple Workflow Service (Amazon SWF), Amazon AppStream, Amazon Elastic Transcoder, Amazon Simple Email Service (Amazon SES), Amazon CloudSearch, and Amazon API Gateway. Mobile services include Amazon Cognito, Amazon Simple Notification Service (Amazon SNS), AWS Device Farm, and Amazon Mobile Analytics. This chapter focuses on the core services you are required to be familiar with to pass the exam: Amazon SQS, Amazon SWF, and Amazon SNS.

Amazon Simple Queue Service (Amazon SQS)

Amazon SQS is a fast, reliable, scalable, and fully managed message queuing service. Amazon SQS makes it simple and cost effective to decouple the components of a cloud application. You can use Amazon SQS to transmit any volume of data, at any level of throughput, without losing messages or requiring other services to be continuously available.

With Amazon SQS, you can offload the administrative burden of operating and scaling a highly available messaging cluster while paying a low price for only what you use. Using Amazon SQS, you can store application messages on reliable and scalable infrastructure, enabling you to move data between distributed components to perform different tasks as needed.

An *Amazon SQS queue* is basically a buffer between the application components that receive data and those components that process the data in your system. If your processing servers cannot process the work fast enough (perhaps due to a spike in traffic), the work is queued so that the processing servers can get to it when they are ready. This means that work is not lost due to insufficient resources.

Amazon SQS ensures delivery of each message at least once and supports multiple readers and writers interacting with the same queue. A single queue can be used simultaneously by many distributed application components, with no need for those components to coordinate with one another to share the queue. Although most of the time each message will be delivered to your application exactly once, you should design your system to be *idempotent* (that is, it must not be adversely affected if it processes the same message more than once).

Amazon SQS is engineered to be highly available and to deliver messages reliably and efficiently; however, the service does not guarantee First In, First Out (FIFO) delivery of messages. For many distributed applications, each message can stand on its own and, if all messages are delivered, the order is not important. If your system requires that order be preserved, you can place sequencing information in each message so that you can reorder the messages when they are retrieved from the queue.

Message Lifecycle

The diagram and process shown in Figure 8.1 describes the lifecycle of an Amazon SQS message, called Message A, from creation to deletion. Assume that a queue already exists.

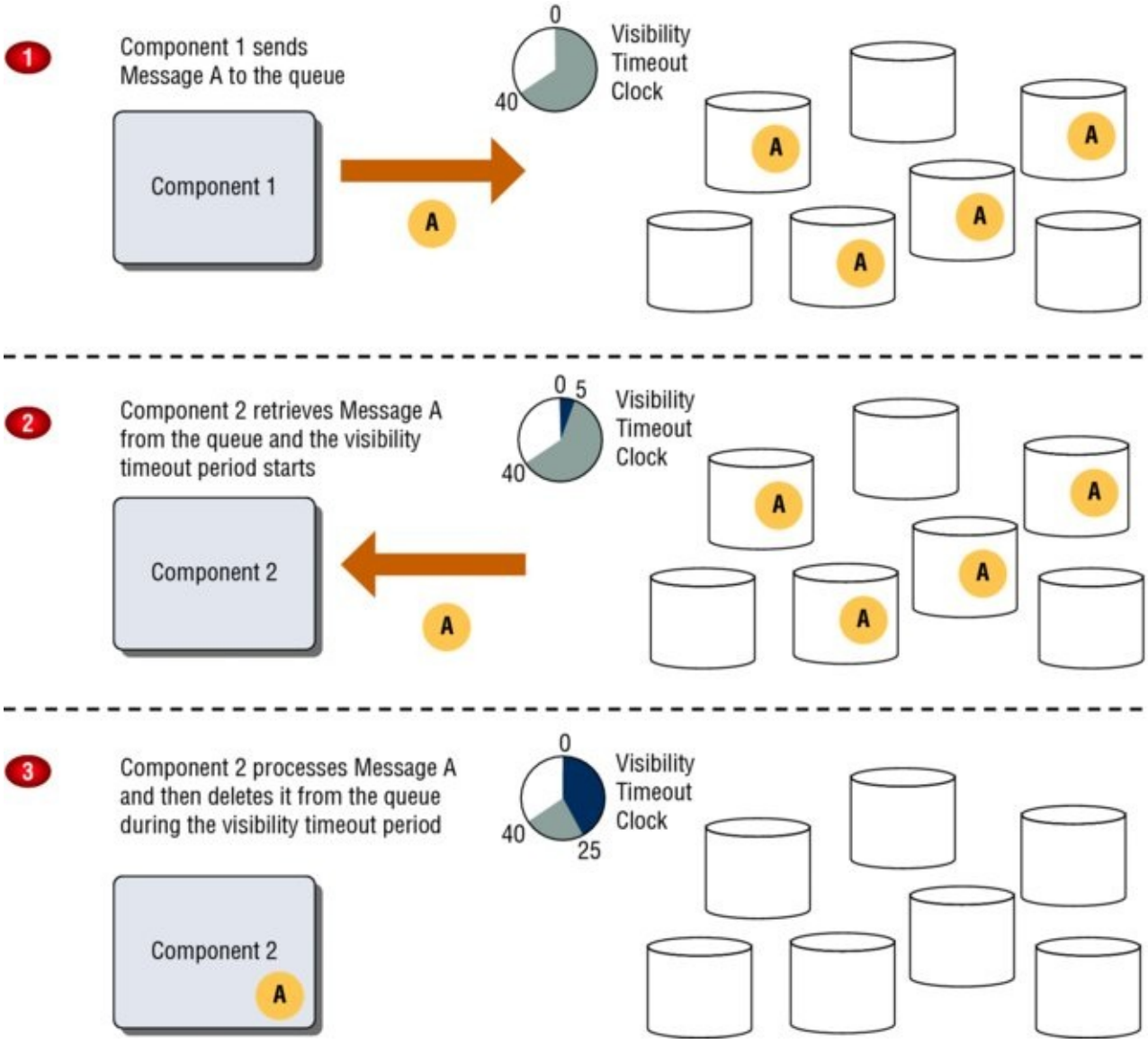

FIGURE 8.1 Message lifecycle

1. Component 1 sends Message A to a queue, and the message is redundantly distributed across the Amazon SQS servers.
2. When Component 2 is ready to process a message, it retrieves messages from the queue, and Message A is returned. While Message A is being processed, it remains in the queue and is not returned to subsequently receive requests for the duration of the visibility timeout.
3. Component 2 deletes Message A from the queue to prevent the message from being received and processed again after the visibility timeout expires.

Delay Queues and Visibility Timeouts

Delay queues allow you to postpone the delivery of new messages in a queue for a specific number of seconds. If you create a delay queue, any message that you send to that queue will be invisible to consumers for the duration of the delay period. To create a delay queue, use `CreateQueue` and set the `DelaySeconds` attribute to any value between 0 and 900 (15 minutes). You can also turn an existing queue into a delay queue by using `SetQueueAttributes` to set the queue's `DelaySeconds` attribute. The default value for `DelaySeconds` is 0.

Delay queues are similar to visibility timeouts in that both features make messages

unavailable to consumers for a specific period of time. The difference is that a delay queue hides a message when it is first added to the queue, whereas a visibility timeout hides a message only after that message is retrieved from the queue. Figure 8.2 illustrates the functioning of a visibility timeout.

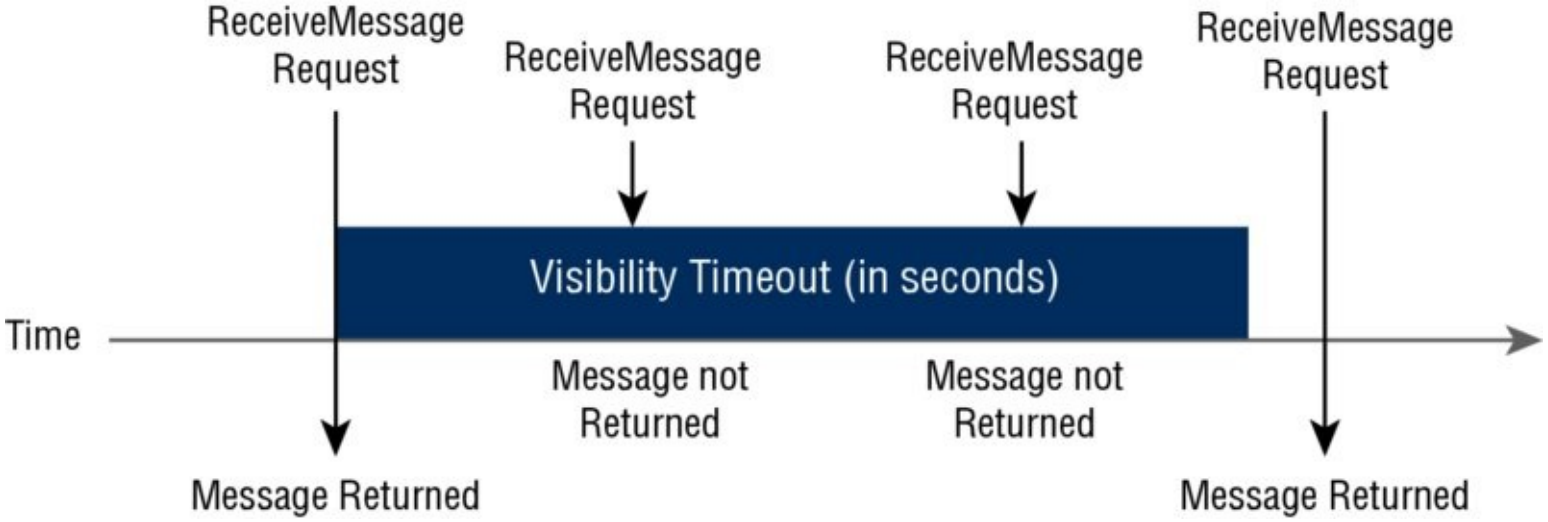

FIGURE 8.2 Diagram of visibility timeout

When a message is in the queue but is neither delayed nor in a visibility timeout, it is considered to be "in flight." You can have up to 120,000 messages in flight at any given time. Amazon SQS supports up to 12 hours' maximum visibility timeout.

Separate Throughput from Latency

Like many other AWS Cloud services, Amazon SQS is accessed through HTTP request-response, and a typical Amazon SQS request-response takes a bit less than 20ms from Amazon Elastic Compute Cloud (Amazon EC2). This means that from a single thread, you can, on average, issue 50+ Application Programming Interface (API) requests per second (a bit fewer for batch API requests, but those do more work). The throughput scales horizontally, so the more threads and hosts you add, the higher the throughput. Using this scaling model, some AWS customers have queues that process thousands of messages every second.

Queue Operations, Unique IDs, and Metadata

The defined operations for Amazon SQS queues are `CreateQueue`, `ListQueues`, `DeleteQueue`, `SendMessage`, `SendMessageBatch`, `ReceiveMessage`, `DeleteMessage`, `DeleteMessageBatch`, `PurgeQueue`, `ChangeMessageVisibility`, `ChangeMessageVisibilityBatch`, `SetQueueAttributes`, `GetQueueAttributes`, `GetQueueUrl`, `ListDeadLetterSourceQueues`, `AddPermission`, and `RemovePermission`. Only the AWS account owner or an AWS identity that has been granted the proper permissions can perform operations.

Your messages are identified via a globally unique ID that Amazon SQS returns when the message is delivered to the queue. The ID isn't required in order to perform any further actions on the message, but it's useful for tracking whether a particular message in the queue has been received. When you receive a message from the queue, the response includes a

receipt handle, which you must provide when deleting the message.

Queue and Message Identifiers

Amazon SQS uses three identifiers that you need to be familiar with: queue URLs, message IDs, and receipt handles.

When creating a new queue, you must provide a queue name that is unique within the scope of all of your queues. Amazon SQS assigns each queue an identifier called a *queue URL*, which includes the queue name and other components that Amazon SQS determines. Whenever you want to perform an action on a queue, you must provide its queue URL.

Amazon SQS assigns each message a unique ID that it returns to you in the `SendMessage` response. This identifier is useful for identifying messages, but note that to delete a message, you need the message's receipt handle instead of the message ID. The maximum length of a message ID is 100 characters.

Each time you receive a message from a queue, you receive a receipt handle for that message. The handle is associated with the act of receiving the message, not with the message itself. As stated previously, to delete the message or to change the message visibility, you must provide the receipt handle and not the message ID. This means you must always receive a message before you can delete it (that is, you can't put a message into the queue and then recall it). The maximum length of a receipt handle is 1,024 characters.

Message Attributes

Amazon SQS provides support for *message attributes*. Message attributes allow you to provide structured *metadata* items (such as timestamps, geospatial data, signatures, and identifiers) about the message. Message attributes are optional and separate from, but sent along with, the message body. The receiver of the message can use this information to help decide how to handle the message without having to process the message body first. Each message can have up to 10 attributes. To specify message attributes, you can use the AWS Management Console, AWS Software Development Kits (SDKs), or a query API.

Long Polling

When your application queries the Amazon SQS queue for messages, it calls the function `ReceiveMessage`. `ReceiveMessage` will check for the existence of a message in the queue and return immediately, either with or without a message. If your code makes periodic calls to the queue, this pattern is sufficient. If your SQS client is just a loop that repeatedly checks for new messages, however, then this pattern becomes problematic, as the constant calls to `ReceiveMessage` burn CPU cycles and tie up a thread.

In this situation, you will want to use *long polling*. With long polling, you send a `WaitTimeSeconds` argument to `ReceiveMessage` of up to 20 seconds. If there is no message in the queue, then the call will wait up to `WaitTimeSeconds` for a message to appear before returning. If a message appears before the time expires, the call will return the message right away. Long polling drastically reduces the amount of load on your client.

Dead Letter Queues

Amazon SQS provides support for *dead letter queues*. A dead letter queue is a queue that other (source) queues can target to send messages that for some reason could not be successfully processed. A primary benefit of using a dead letter queue is the ability to sideline and isolate the unsuccessfully processed messages. You can then analyze any messages sent to the dead letter queue to try to determine the cause of failure.

Messages can be sent to and received from a dead letter queue, just like any other Amazon SQS queue. You can create a dead letter queue from the Amazon SQS API and the Amazon SQS console.

Access Control

While IAM can be used to control the interactions of different AWS identities with queues, there are often times when you will want to expose queues to other accounts. These situations may include:

- You want to grant another AWS account a particular type of access to your queue (for example, `SendMessage`).
- You want to grant another AWS account access to your queue for a specific period of time.
- You want to grant another AWS account access to your queue only if the requests come from your Amazon EC2 instances.
- You want to deny another AWS account access to your queue.

While close coordination between accounts may allow these types of actions through the use of IAM roles, that level of coordination is frequently unfeasible.

Amazon SQS Access Control allows you to assign policies to queues that grant specific interactions to other accounts without that account having to assume IAM roles from your account. These policies are written in the same JSON language as IAM. For example, the following sample policy gives the developer with AWS account number 111122223333 the `SendMessage` permission for the queue named `444455556666/queue1` in the US East (N. Virginia) region.

```
{
"Version": "2012&#x02013;10-17",
"Id": "Queue1_Policy_UUID",
"Statement": [
{
"Sid":"Queue1_SendMessage",
"Effect": "Allow",
"Principal": {
"AWS": "111122223333"
},
"Action": "sqs:SendMessage",
"Resource": "arn:aws:sqs:us-east-1:444455556666:queue1"
}
]
}
```

Tradeoff Message Durability and Latency

Amazon SQS does not return success to a `SendMessage` API call until the message is durably stored in Amazon SQS. This makes the programming model very simple with no doubt about the safety of messages, unlike the situation with an asynchronous messaging model. If you don't need a durable messaging system, however, you can build an asynchronous, client-side batching on top of Amazon SQS libraries that delays enqueue of messages to Amazon SQS and transmits a set of messages in a batch. Please be aware that with a client-side batching approach, you could potentially lose messages when your client process or client host dies for any reason.

Amazon Simple Workflow Service (Amazon SWF)

Amazon SWF makes it easy to build applications that coordinate work across distributed components. In Amazon SWF, a task represents a logical unit of work that is performed by a component of your application. Coordinating tasks across the application involves managing inter-task dependencies, scheduling, and concurrency in accordance with the logical flow of the application. Amazon SWF gives you full control over implementing and coordinating tasks without worrying about underlying complexities such as tracking their progress and maintaining their state.

When using Amazon SWF, you implement workers to perform *tasks*. These workers can run either on cloud infrastructure, such as Amazon EC2, or on your own premises. You can create long-running tasks that might fail, time out, or require restarts, or tasks that can complete with varying throughput and latency. Amazon SWF stores tasks, assigns them to workers when they are ready, monitors their progress, and maintains their state, including details on their completion. To coordinate tasks, you write a program that gets the latest state of each task from Amazon SWF and uses it to initiate subsequent tasks. Amazon SWF maintains an application's execution state durably so that the application is resilient to failures in individual components. With Amazon SWF, you can implement, deploy, scale, and modify these application components independently.

Workflows

Using Amazon SWF, you can implement distributed, asynchronous applications as *workflows*. Workflows coordinate and manage the execution of activities that can be run asynchronously across multiple computing devices and that can feature both sequential and parallel processing.

When designing a workflow, analyze your application to identify its component tasks, which are represented in Amazon SWF as activities. The workflow's coordination logic determines the order in which activities are executed.

Workflow Domains

Domains provide a way of scoping Amazon SWF resources within your AWS account. You must specify a domain for all the components of a workflow, such as the workflow type and activity types. It is possible to have more than one workflow in a domain; however, workflows in different domains cannot interact with one another.

Workflow History

The workflow history is a detailed, complete, and consistent record of every event that occurred since the workflow execution started. An event represents a discrete change in your workflow execution's state, such as scheduled and completed activities, task timeouts, and signals.

Actors

Amazon SWF consists of a number of different types of programmatic features known as

actors. Actors can be workflow starters, deciders, or activity workers. These actors communicate with Amazon SWF through its API. You can develop actors in any programming language.

A workflow starter is any application that can initiate workflow executions. For example, one workflow starter could be an e-commerce website where a customer places an order. Another workflow starter could be a mobile application where a customer orders takeout food or requests a taxi.

Activities within a workflow can run sequentially, in parallel, synchronously, or asynchronously. The logic that coordinates the tasks in a workflow is called the *decider*. The decider schedules the activity tasks and provides input data to the activity workers. The decider also processes events that arrive while the workflow is in progress and closes the workflow when the objective has been completed.

An *activity worker* is a single computer process (or thread) that performs the activity tasks in your workflow. Different types of activity workers process tasks of different activity types, and multiple activity workers can process the same type of task. When an activity worker is ready to process a new activity task, it polls Amazon SWF for tasks that are appropriate for that activity worker. After receiving a task, the activity worker processes the task to completion and then returns the status and result to Amazon SWF. The activity worker then polls for a new task.

Tasks

Amazon SWF provides activity workers and deciders with work assignments, given as one of three types of tasks: activity tasks, AWS Lambda tasks, and decision tasks.

An activity task tells an activity worker to perform its function, such as to check inventory or charge a credit card. The activity task contains all the information that the activity worker needs to perform its function.

An AWS Lambda task is similar to an activity task, but executes an AWS Lambda function instead of a traditional Amazon SWF activity. For more information about how to define an AWS Lambda task, see the AWS documentation on AWS Lambda tasks.

A decision task tells a decider that the state of the workflow execution has changed so that the decider can determine the next activity that needs to be performed. The decision task contains the current workflow history.

Amazon SWF schedules a decision task when the workflow starts and whenever the state of the workflow changes, such as when an activity task completes. Each decision task contains a paginated view of the entire workflow execution history. The decider analyzes the workflow execution history and responds back to Amazon SWF with a set of decisions that specify what should occur next in the workflow execution. Essentially, every decision task gives the decider an opportunity to assess the workflow and provide direction back to Amazon SWF.

Task Lists

Task lists provide a way of organizing the various tasks associated with a workflow. You could think of task lists as similar to dynamic queues. When a task is scheduled in Amazon SWF, you can specify a queue (task list) to put it in. Similarly, when you poll Amazon SWF for a

task, you determine which queue (task list) to get the task from.

Task lists provide a flexible mechanism to route tasks to workers as your use case necessitates. Task lists are dynamic in that you don't need to register a task list or explicitly create it through an action—simply scheduling a task creates the task list if it doesn't already exist.

Long Polling

Deciders and activity workers communicate with Amazon SWF using long polling. The decider or activity worker periodically initiates communication with Amazon SWF, notifying Amazon SWF of its availability to accept a task, and then specifies a task list to get tasks from. Long polling works well for high-volume task processing. Deciders and activity workers can manage their own capacity.

Object Identifiers

Amazon SWF objects are uniquely identified by workflow type, activity type, decision and activity tasks, and workflow execution:

- A registered workflow type is identified by its domain, name, and version. Workflow types are specified in the call to `RegisterWorkflowType`.
- A registered activity type is identified by its domain, name, and version. Activity types are specified in the call to `RegisterActivityType`.
- Each decision task and activity task is identified by a unique task token. The task token is generated by Amazon SWF and is returned with other information about the task in the response from `PollForDecisionTask` or `PollForActivityTask`. Although the token is most commonly used by the process that received the task, that process could pass the token to another process, which could then report the completion or failure of the task.
- A single execution of a workflow is identified by the domain, workflow ID, and run ID. The first two are parameters that are passed to `StartWorkflowExecution`. The run ID is returned by `StartWorkflowExecution`.

Workflow Execution Closure

After you start a workflow execution, it is open. An open workflow execution can be closed as completed, canceled, failed, or timed out. It can also be continued as a new execution, or it can be terminated. The decider, the person administering the workflow, or Amazon SWF can close a workflow execution.

Lifecycle of a Workflow Execution

From the start of a workflow execution to its completion, Amazon SWF interacts with actors by assigning them appropriate tasks: either activity tasks or decision tasks.

Figure 8.3 shows the lifecycle of an order-processing workflow execution from the perspective of components that act on it.

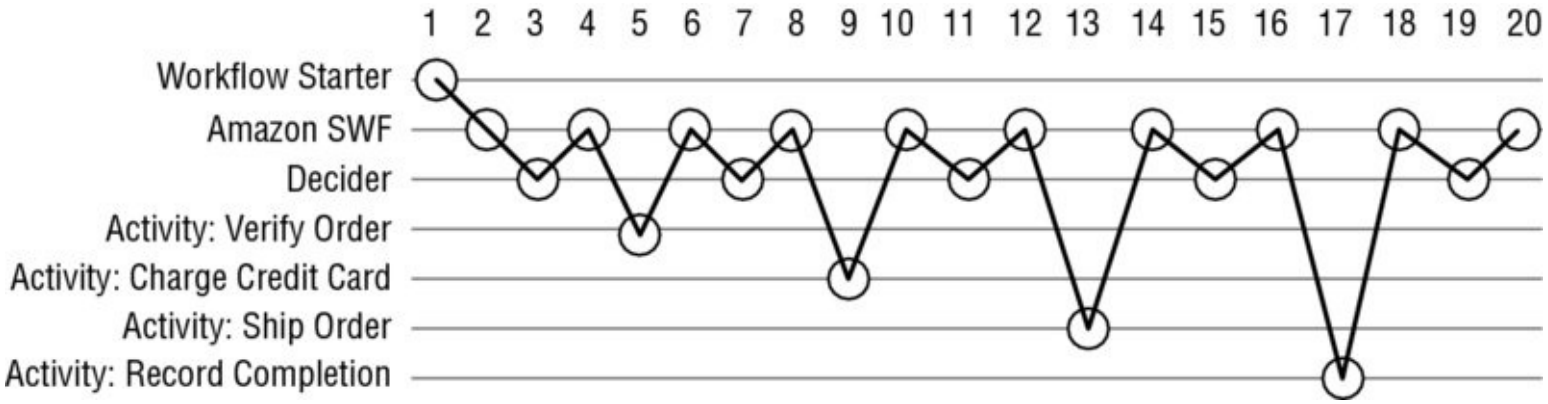

FIGURE 8.3 Amazon SWF workflow illustration

The following 20 steps describe the workflow detailed in Figure 8.3:

1. A workflow starter calls an Amazon SWF action to start the workflow execution for an order, providing order information.
2. Amazon SWF receives the start workflow execution request and then schedules the first decision task.
3. The decider receives the task from Amazon SWF, reviews the history, and applies the coordination logic to determine that no previous activities occurred. It then makes a decision to schedule the Verify Order activity with the information the activity worker needs to process the task and returns the decision to Amazon SWF.
4. Amazon SWF receives the decision, schedules the Verify Order activity task, and waits for the activity task to complete or time out.
5. An activity worker that can perform the Verify Order activity receives the task, performs it, and returns the results to Amazon SWF.
6. Amazon SWF receives the results of the Verify Order activity, adds them to the workflow history, and schedules a decision task.
7. The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to schedule a Charge Credit Card activity task with information the activity worker needs to process the task, and returns the decision to Amazon SWF.
8. Amazon SWF receives the decision, schedules the Charge Credit Card activity task, and waits for it to complete or time out.
9. An activity worker activity receives the Charge Credit Card task, performs it, and returns the results to Amazon SWF.
10. Amazon SWF receives the results of the Charge Credit Card activity task, adds them to the workflow history, and schedules a decision task.
11. The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to schedule a Ship Order activity task with the information the activity worker needs to perform the task, and returns the decision to Amazon SWF.
12. Amazon SWF receives the decision, schedules a Ship Order activity task, and waits for it

to complete or time out.

13. An activity worker that can perform the Ship Order activity receives the task, performs it, and returns the results to Amazon SWF.
14. Amazon SWF receives the results of the Ship Order activity task, adds them to the workflow history, and schedules a decision task.
15. The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to schedule a Record Completion activity task with the information the activity worker needs, performs the task, and returns the decision to Amazon SWF.
16. Amazon SWF receives the decision, schedules a Record Completion activity task, and waits for it to complete or time out.
17. An activity worker Record Completion receives the task, performs it, and returns the results to Amazon SWF.
18. Amazon SWF receives the results of the Record Completion activity task, adds them to the workflow history, and schedules a decision task.
19. The decider receives the task from Amazon SWF, reviews the history, applies the coordination logic, makes a decision to close the workflow execution, and returns the decision along with any results to Amazon SWF.
20. Amazon SWF closes the workflow execution and archives the history for future reference.

Amazon Simple Notification Service (Amazon SNS)

Amazon SNS is a web service for mobile and enterprise messaging that enables you to set up, operate, and send notifications. It is designed to make web-scale computing easier for developers. Amazon SNS follows the publish-subscribe (pub-sub) messaging paradigm, with notifications being delivered to clients using a push mechanism that eliminates the need to check periodically (or poll) for new information and updates. For example, you can send notifications to Apple, Android, Fire OS, and Windows devices. In China, you can send messages to Android devices with Baidu Cloud Push. You can use Amazon SNS to send Short Message Service (SMS) messages to mobile device users in the United States or to email recipients worldwide.

Amazon SNS consists of two types of clients: publishers and subscribers (sometimes known as producers and consumers). Publishers communicate to subscribers asynchronously by sending a message to a topic. A topic is simply a logical access point/communication channel that contains a list of subscribers and the methods used to communicate to them. When you send a message to a topic, it is automatically forwarded to each subscriber of that topic using the communication method configured for that subscriber.

Figure 8.4 shows this process at a high level. A publisher issues a message on a topic. The message is then delivered to the subscribers of that topic using different methods, such as Amazon SQS, HTTP, HTTPS, email, SMS, and AWS Lambda.

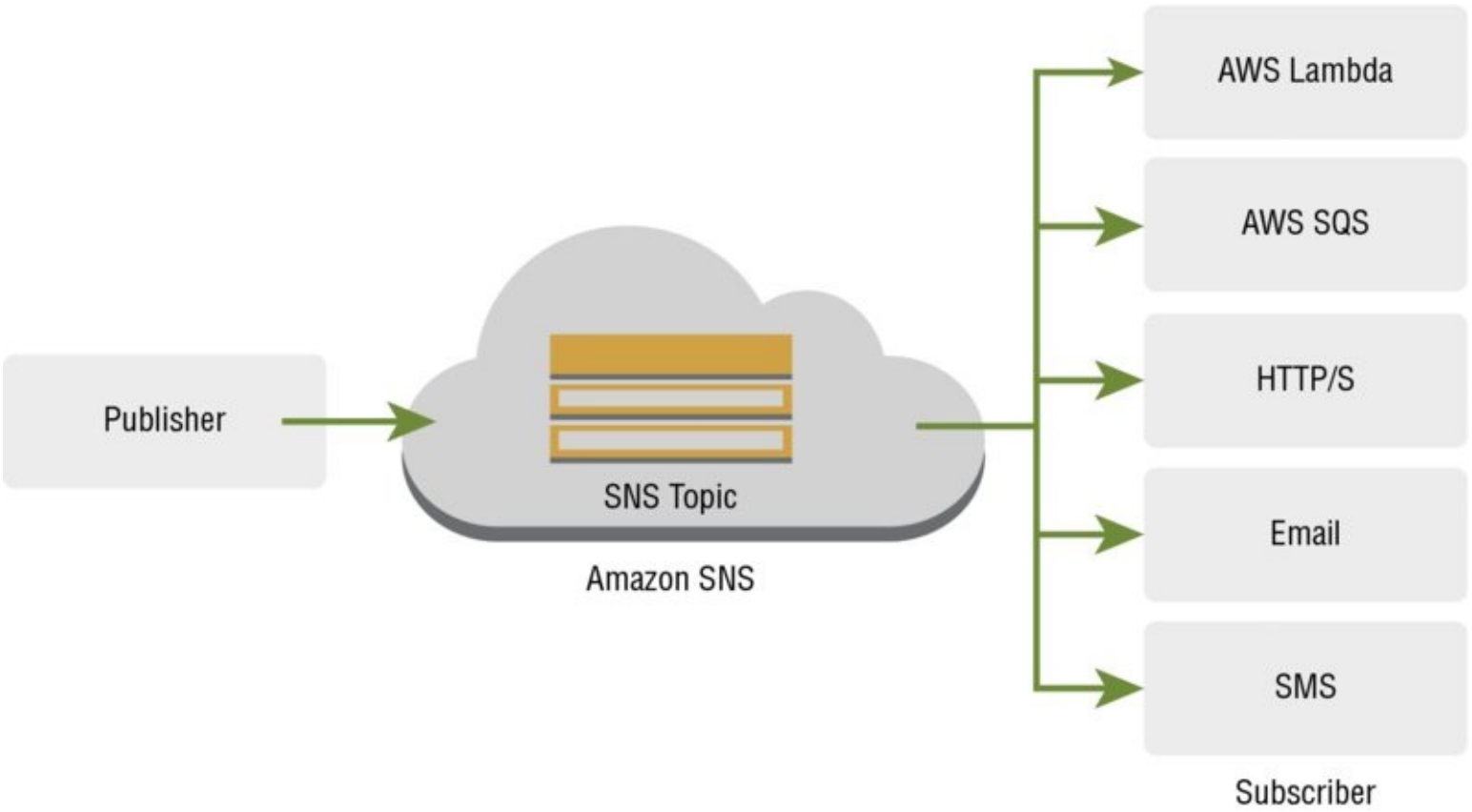

FIGURE 8.4 Diagram of topic delivery

When using Amazon SNS, you (as the owner) create a topic and control access to it by defining policies that determine which publishers and subscribers can communicate with the topic and via which technologies. Publishers send messages to topics that they created or that

they have permission to publish to. Instead of including a specific destination address in each message, a publisher sends a message to the topic, and Amazon SNS delivers the message to each subscriber for that topic. Each topic has a unique name that identifies the Amazon SNS endpoint where publishers post messages and subscribers register for notifications. Subscribers receive all messages published to the topics to which they subscribe, and all subscribers to a topic receive the same messages.

Common Amazon SNS Scenarios

Amazon SNS can support a wide variety of needs, including monitoring applications, workflow systems, time-sensitive information updates, mobile applications, and any other application that generates or consumes notifications. For example, you can use Amazon SNS to relay events in workflow systems among distributed computer applications, move data between data stores, or update records in business systems. Event updates and notifications concerning validation, approval, inventory changes, and shipment status are immediately delivered to relevant system components and end users. Another example use for Amazon SNS is to relay time-critical events to mobile applications and devices. Because Amazon SNS is both highly reliable and scalable, it provides significant advantages to developers who build applications that rely on real-time events.

To help illustrate, the following sections describe some common Amazon SNS scenarios, including fanout scenarios, application and system alerts, push email and text messaging, and mobile push notifications.

Fanout

A fanout scenario is when an Amazon SNS message is sent to a topic and then replicated and pushed to multiple Amazon SQS queues, HTTP endpoints, or email addresses (see Figure 8.5). This allows for parallel asynchronous processing. For example, you can develop an application that sends an Amazon SNS message to a topic whenever an order is placed for a product. Then the Amazon SQS queues that are subscribed to that topic will receive identical notifications for the new order. An Amazon EC2 instance attached to one of the queues handles the processing or fulfillment of the order, while an Amazon EC2 instance attached to a parallel queue sends order data to a data warehouse application/service for analysis.

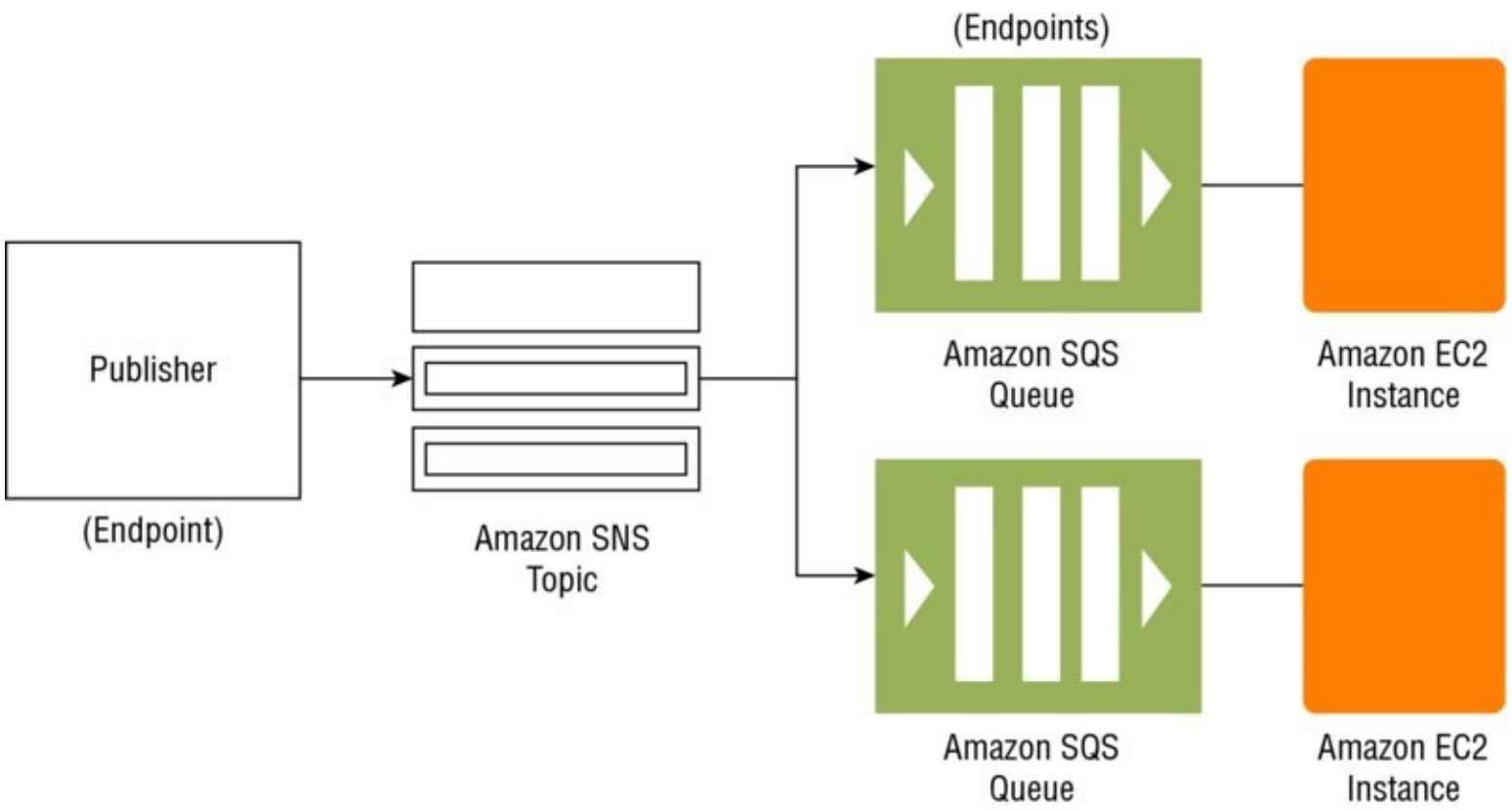

FIGURE 8.5 Diagram of fanout scenario

Another way to use fanout is to replicate data sent to your production environment and integrate it with your development environment. Expanding upon the previous example, you can subscribe yet another queue to the same topic for new incoming orders. Then, by attaching this new queue to your development environment, you can continue to improve and test your application using data received from your production environment.

Application and System Alerts

Application and system alerts are SMS and/or email notifications that are triggered by predefined thresholds. For example, because many AWS Cloud services use Amazon SNS, you can receive immediate notification when an event occurs, such as a specific change to your Auto Scaling group in AWS.

Push Email and Text Messaging

Push email and text messaging are two ways to transmit messages to individuals or groups via email and/or SMS. For example, you can use Amazon SNS to push targeted news headlines to subscribers by email or SMS. Upon receiving the email or SMS text, interested readers can then choose to learn more by visiting a website or launching an application.

Mobile Push Notifications

Mobile push notifications enable you to send messages directly to mobile applications. For example, you can use Amazon SNS for sending notifications to an application, indicating that an update is available. The notification message can include a link to download and install the update.

Summary

In this chapter, you learned about the core application and mobile services that you will be tested on in your AWS Certified Solutions Architect – Associate exam.

Amazon SQS is a unique service designed by Amazon to help you decouple your infrastructure. Using Amazon SQS, you can store messages on reliable and scalable infrastructure as they travel between distributed components of your applications that perform different tasks, without losing messages or requiring each component to be continuously available.

Understand Amazon SQS queue operations, unique IDs, and metadata. Be familiar with queue and message identifiers such as queue URLs, message IDs, and receipt handles. Understand related concepts such as delay queues, message attributes, long polling, message timers, dead letter queues, access control, and the overall message lifecycle.

Amazon SWF allows you to create applications that coordinate work across distributed components. Amazon SWF is driven by tasks, which are logical units of work that different components of your application perform. To manage tasks across your application, you need to be aware of inter-task dependencies, scheduling of tasks, and using tasks concurrently. Amazon SWF simplifies the coordination of workflow tasks, giving you full control over their implementation without worrying about underlying complexities such as tracking their progress and maintaining their state.

You must be familiar with the following Amazon SWF components and the lifecycle of a workflow execution:

- Workers, starters, and deciders
- Workflows
- Workflow history
- Actors
- Tasks
- Domains
- Object identifiers
- Task lists
- Workflow execution closure
- Long polling

Amazon SNS is a push notification service that lets you send individual or multiple messages to large numbers of recipients. Amazon SNS consists of two types of clients: publishers and subscribers (sometimes known as producers and consumers). Publishers communicate to subscribers asynchronously by sending a message to a topic. A topic is simply a logical access point/communication channel that contains a list of subscribers and the methods used to communicate to them. When you send a message to a topic, it is automatically forwarded to each subscriber of that topic using the communication method configured for that subscriber.

Amazon SNS can support a wide variety of needs, including monitoring applications, workflow systems, time-sensitive information updates, mobile applications, and any other application that generates or consumes notifications. Understand some common Amazon SNS scenarios, including:

- Fanout
- Application and system alerts
- Push email and text messaging
- Mobile push notifications

Exam Essentials

Know how to use Amazon SQS. Amazon SQS is a unique service designed by Amazon to help you to decouple your infrastructure. Using Amazon SQS, you can store messages on reliable and scalable infrastructure as they travel between your servers. This allows you to move data between distributed components of your applications that perform different tasks without losing messages or requiring each component always to be available.

Understand Amazon SQS visibility timeouts. Visibility timeout is a period of time during which Amazon SQS prevents other components from receiving and processing a message because another component is already processing it. By default, the message visibility timeout is set to 30 seconds, and the maximum that it can be is 12 hours.

Know how to use Amazon SQS long polling. Long polling allows your Amazon SQS client to poll an Amazon SQS queue. If nothing is there, `ReceiveMessage` waits between 1 and 20 seconds. If a message arrives in that time, it is returned to the caller as soon as possible. If a message does not arrive in that time, you need to execute the `ReceiveMessage` function again. This helps you avoid polling in tight loops and prevents you from burning through CPU cycles, keeping costs low.

Know how to use Amazon SWF. Amazon SWF allows you to make applications that coordinate work across distributed components. Amazon SWF is driven by tasks, which are logical units of work that part of your application performs. To manage tasks across your application, you need to be aware of inter-task dependencies, scheduling of tasks, and using tasks concurrently. This is where Amazon SWF can help you. It gives you full control over implementing tasks and coordinating them without worrying about underlying complexities such as tracking their progress and maintaining their state.

Know the basics of an Amazon SWF workflow. A workflow is a collection of activities (coordinated by logic) that carry out a specific goal. For example, a workflow receives a customer order and takes whatever actions are necessary to fulfill it. Each workflow runs in an AWS resource called a domain, which controls the scope of the workflow. An AWS account can have multiple domains, each of which can contain multiple workflows, but workflows in different domains cannot interact.

Understand the different Amazon SWF actors. Amazon SWF interacts with a number of different types of programmatic actors. Actors can be activity workers, workflow starters, or deciders.

Understand Amazon SNS basics. Amazon SNS is a push notification service that lets you send individual or multiple messages to large numbers of recipients. Amazon SNS consists of two types of clients: publishers and subscribers (sometimes known as producers and consumers). Publishers communicate to subscribers asynchronously by sending a message to a topic.

Know the different protocols used with Amazon SNS. You can use the following protocols with Amazon SNS: HTTP, HTTPS, SMS, email, email-JSON, Amazon SQS, and AWS Lambda.

Exercises

In this section, you create a topic and subscription in Amazon SNS and then publish a message to your topic.

EXERCISE 8.1

Create an Amazon SNS Topic

In this exercise, you will create an Amazon SNS message.

1. Open a browser, and navigate to the AWS Management Console. Sign in to your AWS account.
2. Navigate to Mobile Services and then Amazon SNS to load the Amazon SNS dashboard.
3. Create a new topic, and use `MyTopic` for both the topic name and the display name.
4. Note that an Amazon Resource Name (ARN) is specified immediately.

Congratulations! You have created your first topic.

EXERCISE 8.2

Create a Subscription to Your Topic

In this exercise, you will create a subscription to the newly created topic using your email address. Then you confirm your email address.

1. In the Amazon SNS dashboard of the AWS Management Console, navigate to Topics.
2. Select the ARN that you just created. Create a Subscription with the protocol of Email, and enter your email address.
3. Create the Subscription.
4. The service sends a confirmation email to your email address. Before this subscription can go live, you need to click on the link in the email that AWS sent you to confirm your email address. Check your email, and confirm your address.

Congratulations! You have now confirmed your email address and created a subscription to a topic.

EXERCISE 8.3

Publish to a Topic

In this exercise, you will publish a message to your newly created topic.

1. In the Amazon SNS dashboard of the AWS Management Console, navigate to Topics.
2. Navigate to the ARN link for your newly created topic.
3. Update the subject with My Test Message, leave the message format to set to Raw, and use a Time to Live (TTL) field to 300.
4. Publish the message.
5. You should receive an email from your topic name with the subject that you specified. If you do not receive this email, check your junk folder.

Congratulations! In this exercise, you created a new topic, added a new subscription, and then published a message to your new topic. Note the different formats in which you can publish messages, including HTTP and AWS Lambda. Delete your newly created topic and subscriptions after you are finished.

EXERCISE 8.4

Create Queue

1. In the AWS Management Console, navigate to Application Services and then to Amazon SQS to load the Amazon SQS dashboard.
2. Create a new queue with `input` as the queue name, 60 seconds for the default visibility, and 5 minutes for the message retention period. Leave the remaining default values for this exercise.
3. Create the queue.

Congratulations! In this exercise, you created a new queue. You will publish to this queue in the following exercise.

EXERCISE 8.5

Subscribe Queue to SNS Topic

1. In the AWS Management Console, navigate to Application Services and then to Amazon SQS to load the Amazon SQS dashboard.
2. Subscribe your queue to your Amazon SNS topic.
3. Now return to the Amazon SNS dashboard (in the AWS Management Console under Mobile Services).
4. Publish to your new topic, and use the defaults.
5. Return to the Amazon SQS dashboard (in the AWS Management Console under Application Services).
6. You will notice there is “1 Message Available” in the input queue. Check the input box to the left of the input queue name.
7. Start polling for messages. You should see the Amazon SNS message in your queue.
8. Click the More Details link to see the details of the message.
9. Review your message, and click Close.
10. Delete your message.

Congratulations! In this exercise, you subscribed your input queue to an Amazon SNS topic and viewed your message in your Amazon SQS queue in addition to receiving the message in subscribed email.

Review Questions

1. Which of the following is not a supported Amazon Simple Notification Service (Amazon SNS) protocol?
   A. HTTPS
   B. AWS Lambda
   C. Email-JSON
   D. Amazon DynamoDB
2. When you create a new Amazon Simple Notification Service (Amazon SNS) topic, which of the following is created automatically?
   A. An Amazon Resource Name (ARN)
   B. A subscriber
   C. An Amazon Simple Queue Service (Amazon SQS) queue to deliver your Amazon SNS topic
   D. A message
3. Which of the following are features of Amazon Simple Notification Service (Amazon SNS)? (Choose 3 answers)
   A. Publishers
   B. Readers
   C. Subscribers
   D. Topic
4. What is the default time for an Amazon Simple Queue Service (Amazon SQS) visibility timeout?
   A. 30 seconds
   B. 60 seconds
   C. 1 hour
   D. 12 hours
5. What is the longest time available for an Amazon Simple Queue Service (Amazon SQS) visibility timeout?
   A. 30 seconds
   B. 60 seconds
   C. 1 hour
   D. 12 hours
6. Which of the following options are valid properties of an Amazon Simple Queue Service

(Amazon SQS) message? (Choose 2 answers)

A. Destination

B. Message ID

C. Type

D. Body

7. You are a solutions architect who is working for a mobile application company that wants to use Amazon Simple Workflow Service (Amazon SWF) for their new takeout ordering application. They will have multiple workflows that will need to interact. What should you advise them to do in structuring the design of their Amazon SWF environment?

   A. Use multiple domains, each containing a single workflow, and design the workflows to interact across the different domains.

   B. Use a single domain containing multiple workflows. In this manner, the workflows will be able to interact.

   C. Use a single domain with a single workflow and collapse all activities to within this single workflow.

   D. Workflows cannot interact with each other; they would be better off using Amazon Simple Queue Service (Amazon SQS) and Amazon Simple Notification Service (Amazon SNS) for their application.

8. In Amazon Simple Workflow Service (Amazon SWF), which of the following are actors? (Choose 3 answers)

   A. Activity workers

   B. Workflow starters

   C. Deciders

   D. Activity tasks

9. You are designing a new application, and you need to ensure that the components of your application are not tightly coupled. You are trying to decide between the different AWS Cloud services to use to achieve this goal. Your requirements are that messages between your application components may not be delivered more than once, tasks must be completed in either a synchronous or asynchronous fashion, and there must be some form of application logic that decides what do when tasks have been completed. What application service should you use?

   A. Amazon Simple Queue Service (Amazon SQS)

   B. Amazon Simple Workflow Service (Amazon SWF)

   C. Amazon Simple Storage Service (Amazon S3)

   D. Amazon Simple Email Service (Amazon SES)

10. How does Amazon Simple Queue Service (Amazon SQS) deliver messages?

    A. Last In, First Out (LIFO)

B. First In, First Out (FIFO)

C. Sequentially

D. Amazon SQS doesn't guarantee delivery of your messages in any particular order.

11. Of the following options, what is an efficient way to fanout a single Amazon Simple Notification Service (Amazon SNS) message to multiple Amazon Simple Queue Service (Amazon SQS) queues?

A. Create an Amazon SNS topic using Amazon SNS. Then create and subscribe multiple Amazon SQS queues sent to the Amazon SNS topic.

B. Create one Amazon SQS queue that subscribes to multiple Amazon SNS topics.

C. Amazon SNS allows exactly one subscriber to each topic, so fanout is not possible.

D. Create an Amazon SNS topic using Amazon SNS. Create an application that subscribes to that topic and duplicates the message. Send copies to multiple Amazon SQS queues.

12. Your application polls an Amazon Simple Queue Service (Amazon SQS) queue frequently and returns immediately, often with empty ReceiveMessageResponses. What is one thing that can be done to reduce Amazon SQS costs?

A. Pricing on Amazon SQS does not include a cost for service requests; therefore, there is no concern.

B. Increase the timeout value for short polling to wait for messages longer before returning a response.

C. Change the message visibility value to a higher number.

D. Use long polling by supplying a WaitTimeSeconds of greater than 0 seconds when calling ReceiveMessage.

13. What is the longest time available for an Amazon Simple Queue Service (Amazon SQS) long polling timeout?

A. 10 seconds

B. 20 seconds

C. 30 seconds

D. 1 hour

14. What is the longest configurable message retention period for Amazon Simple Queue Service (Amazon SQS)?

A. 30 minutes

B. 4 days

C. 30 seconds

D. 14 days

15. What is the default message retention period for Amazon Simple Queue Service (Amazon SQS)?

A. 30 minutes

B. 4 days

C. 30 seconds

D. 14 days

16. Amazon Simple Notification Service (Amazon SNS) is a push notification service that lets you send individual or multiple messages to large numbers of recipients. What types of clients are supported?

A. Java and JavaScript clients that support publisher and subscriber types

B. Producers and consumers supported by C and C++ clients

C. Mobile and AMQP support for publisher and subscriber client types

D. Publisher and subscriber client types

17. In Amazon Simple Workflow Service (Amazon SWF), a decider is responsible for what?

A. Executing each step of the work

B. Defining work coordination logic by specifying work sequencing, timing, and failure conditions

C. Executing your workflow

D. Registering activities and workflow with Amazon SWF

18. Can an Amazon Simple Notification Service (Amazon SNS) topic be recreated with a previously used topic name?

A. Yes. The topic name should typically be available after 24 hours after the previous topic with the same name has been deleted.

B. Yes. The topic name should typically be available after 1–3 hours after the previous topic with the same name has been deleted.

C. Yes. The topic name should typically be available after 30–60 seconds after the previous topic with the same name has been deleted.

D. At this time, this feature is not supported.

19. What should you do in order to grant a different AWS account permission to your Amazon Simple Queue Service (Amazon SQS) queue?

A. Share credentials to your AWS account and have the other account's applications use your account's credentials to access the Amazon SQS queue.

B. Create a user for that account in AWS Identity and Access Management (IAM) and establish an IAM policy that grants access to the queue.

C. Create an Amazon SQS policy that grants the other account access.

D. Amazon Virtual Private Cloud (Amazon VPC) peering must be used to achieve this.

20. Can an Amazon Simple Notification Service (Amazon SNS) message be deleted after being published to a topic?

A. Only if a subscriber(s) has/have not read the message yet
B. Only if the Amazon SNS recall message parameter has been set
C. No. After a message has been successfully published to a topic, it cannot be recalled.
D. Yes. However it can be deleted only if the subscribers are Amazon SQS queues.