AWS notes

A. AWS STORAGE

KMS is for encryption at rest only (not in transit, use SSL).
Exam tip: Encryption keys are regional.

EXAM TIP: Know which services use interface endpoints and gateway endpoints. The easiest way to remember this is that Gateway Endpoints are for Amazon S3 and DynamoDB only.

Exam tip: AWS Lambda and Amazon ECS deployments cannot use an in-place deployment type.

Note: CodeDeploy does not provision the resources – it deploys applications not EC2 instances.

Secrets Manager is used for rotating credentials, not encryption keys.

Exam tip: If an exam scenario requires a unified development toolchain, and mentions collaboration between team members, synchronization, and centralized management of the CI/CD pipeline this will be CodeStar rather than CodePipeline or CodeCommit.

Exam tip: To make it easier to remember the different between User Pools and Identity Pools, think of Users Pools as being similar to IAM Users or Active Directory and an Identity Pools as being similar to an IAM Role.

Exam tip: AWS AppSync is a similar service that has additional capabilities. With AppSync you can synchronize mobile app data

across devices and users (Cognito Sync cannot synchronize across users, only devices), it has support for additional devices and data types, and is based on GraphQL.

Exam tip: Remember that annotations can be used for adding system or user-defined data to segments and subsegments that you want to index for search. Metadata is not indexed and cannot be used for searching.

1. Amazon S3 default encryption provides a way to set the default encryption behavior for an S3 bucket. You can set default encryption on a bucket so that all new objects are encrypted when they are stored in the bucket. The objects are encrypted using server-side encryption with either Amazon S3-managed keys (SSE-S3) or customer master keys (CMKs) stored in AWS Key Management

   Service (AWS KMS).
2. Server-Side Encryption with Customer Master Keys (CMKs) Stored in AWS Key Management Service (SSE-KMS) is similar to SSE-S3, but with some additional benefits and charges for using this service. There are separate permissions for the use of a CMK that provides added protection against unauthorized access of

your objects in Amazon S3. SSE-KMS also provides you with an audit trail that shows when your CMK was used and by whom.Therefore, the key to answering this question correctly is understanding that you do not get an audit trail of key usage when using S3-managed keys. You can still track API usage if you're using S3-managed keys, but not key usage. For this solution we need to use server-side encryption and AWS KMS-managed keys.

3. Amazon S3 Transfer Acceleration enables fast, easy, and secure transfers of files over long distances between your client and an S3 bucket. Transfer Acceleration takes advantage of Amazon CloudFront's globally distributed edge locations. As the data arrives at an edge location, data is routed to Amazon S3 over an optimized network path.You might want to use Transfer Acceleration on a bucket for various reasons, including the following:
   You have customers that upload to a centralized bucket from all over the world.

   You transfer gigabytes to terabytes of data on a regular basis across continents.

   You are unable to utilize all of your available bandwidth over the Internet when uploading to Amazon S3.

   Therefore, Amazon S3 Transfer Acceleration is an ideal solution for this use case and will result in improved throughput and upload times.

4. When you create an object, you specify the key name, which uniquely identifies the object in the bucket. For example, in the Amazon S3 console, when you highlight a bucket, a list of objects in your bucket appears. These names are the object keys. The name for a key is a sequence of Unicode characters whose UTF-8 encoding is at most 1024 bytes long.
   The Amazon S3 data model is a flat structure: you create a bucket, and the bucket stores objects. There is no hierarchy of subbuckets or subfolders. However, you can infer logical hierarchy using key name prefixes and delimiters as the Amazon S3 console does. The Amazon S3 console supports a concept of folders. Suppose that your bucket (admin-created) has four objects with the following object keys:

   · Development/Projects.xls

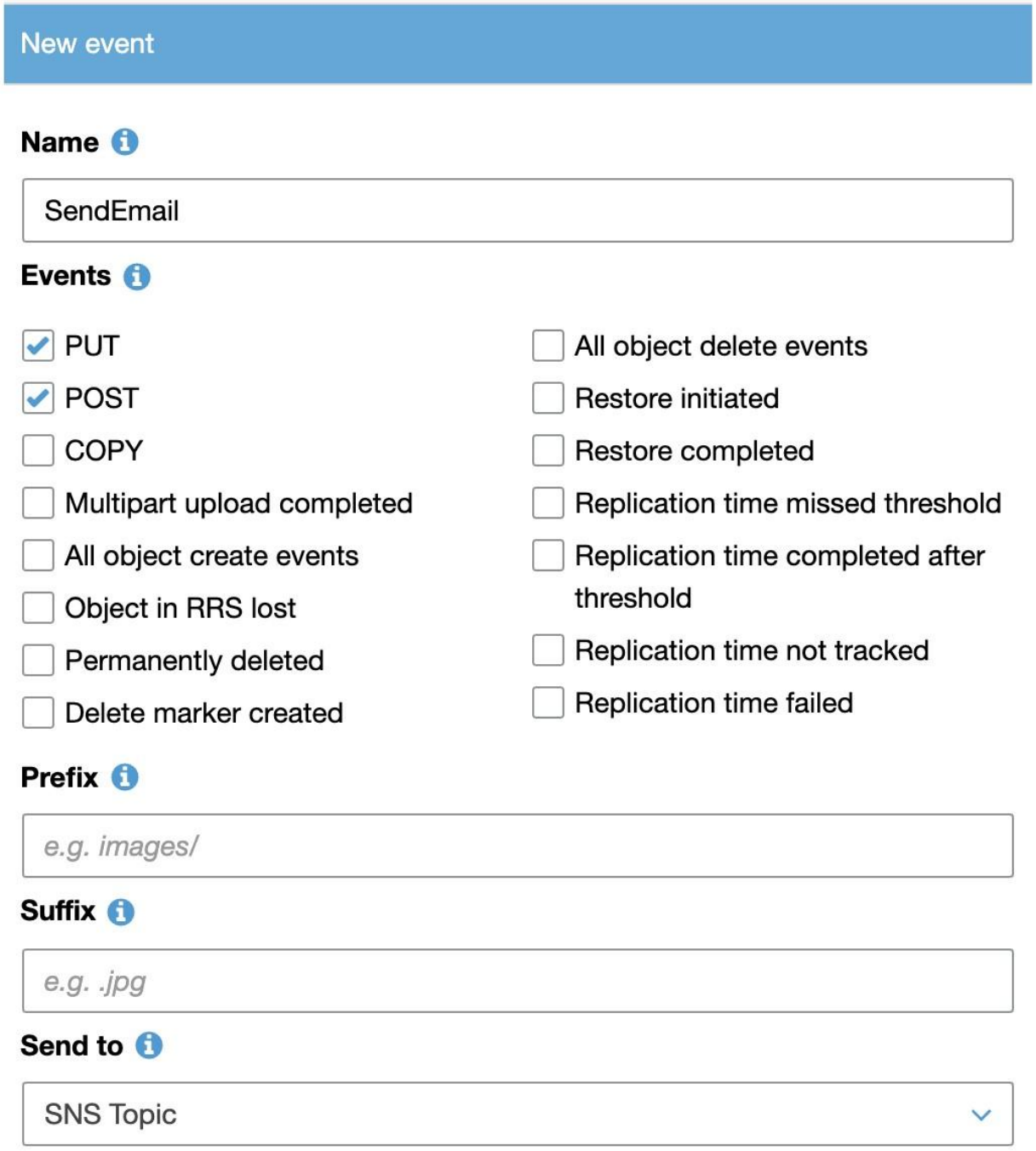
   · Finance/statement1.pdf

   · Private/taxdocument.pdf

· s3-dg.pdf

The console uses the key name prefixes (Development/, Finance/, and Private/) and delimiter ('/') to present a folder structure as shown.

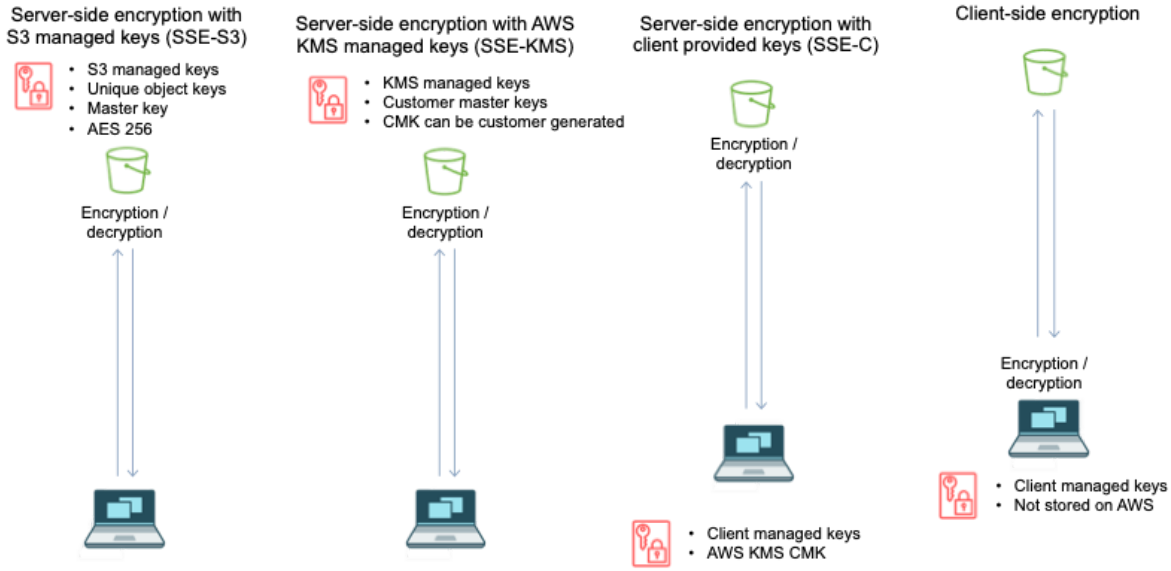
5. Amazon Elastic Block Store (Amazon EBS) provides block level storage volumes for use with EC2 instances. EBS volumes behave like raw, unformatted block devices.
   You can mount these volumes as devices on your instances. You can mount multiple volumes on the same instance, and you can mount a volume to multiple instances at a time.
   You can create a file system on top of these volumes or use them in any way you would use a block device (like a hard drive). You can dynamically change the configuration of a volume attached to an instance.
6. The Amazon S3 notification feature enables you to receive notifications when certain events happen in your bucket. To enable notifications, you must first add a notification configuration that identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications. You store this configuration in the *notification* subresource that is associated with a bucket.



Amazon Simple Notification Service (SNS) is a service that allows you to send notifications from the cloud in a publisher / subscriber model. It supports multiple transport protocols including email.

7. Server-side encryption is the encryption of data at its destination by the application or service that receives it. Amazon S3 encrypts your data at the object level as it writes it to disks in its data centers and decrypts it for you when

you access it. As long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects.



As you can see in the image above, there are three options for server-side encryption:

· **Server-Side Encryption with Amazon S3-Managed Keys (SSE-S3)** – the data is encrypted by Amazon S3 using keys that are managed through S3

· **Server-Side Encryption with Customer Master Keys (CMKs) Stored in AWS Key Management Service (SSE-KMS)** – this option uses CMKs managed in AWS KMS. There are additional benefits such as auditing and permissions associated with the CMKs but also additional charges

· **Server-Side Encryption with Customer-Provided Keys (SSE-C)** – you manage the encryption keys and Amazon S3 manages the encryption, as it writes to disks, and decryption, when you access your objects.

8. DynamoDB only supports a maximum item size of 400 KB.
9. S3 Standard offers high durability, availability, and performance object storage for frequently accessed data.

   Key Features:

   • Low latency and high throughput performance

• Designed for durability of 99.999999999% of objects across multiple Availability Zones

• Resilient against events that impact an entire Availability Zone

• Designed for 99.99% availability over a given year

• Backed with the Amazon S3 Service Level Agreement for availability

• Supports SSL for data in transit and encryption of data at rest

• S3 Lifecycle management for automatic migration of objects to other S3 Storage Classes

Amazon S3 Standard is the best solution for this scenario as the data is accessed directly by applications with varying frequency. Therefore, the S3 Glacier storage class would not be suitable as it would archive the data and data must be restored before being accessible.

10. You can use Amazon S3 to host a static website. On a *static* website, individual web pages include static content. They might also contain client-side scripts.

    To host a static website on Amazon S3, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. When you configure a bucket as a static website, you enable static website hosting, set permissions, and add an index document. Depending on your website requirements, you can also configure other options, including redirects, web traffic logging, and custom error documents.

11. The Amazon S3 notification feature enables you to receive notifications when certain events happen in your bucket. To enable notifications, you must first add a notification configuration that identifies the events you want Amazon S3 to publish and the destinations where you want Amazon S3 to send the notifications. You store this configuration in the *notification* subresource that is associated with a bucket.

    Currently, Amazon S3 can publish notifications for the following events:

    New object created events — Amazon S3 supports multiple APIs to create objects. You can request notification when only a specific API is used (for example, s3:ObjectCreated:Put), or you can use a wildcard (for example,

s3:ObjectCreated:*) to request notification when an object is created regardless of the API used.

Object removal events — Amazon S3 supports deletes of versioned and unversioned objects. For information about object versioning, see Object Versioning and Using Versioning.

Restore object events — Amazon S3 supports the restoration of objects archived to the S3 Glacier storage class. You request to be notified of object restoration completion by using s3:ObjectRestore:Completed. You use s3:ObjectRestore:Post to request notification of the initiation of a restore.

Reduced Redundancy Storage (RRS) object lost events — Amazon S3 sends a notification message when it detects that an object of the RRS storage class has been lost.

Replication events — Amazon S3 sends event notifications for replication configurations that have S3 Replication Time Control (S3 RTC) enabled. It sends these notifications when an object fails replication, when an object exceeds the 15-minute threshold, when an object is replicated after the 15-minute threshold, and when an object is no longer tracked by replication metrics. It publishes a second event when that object replicates to the destination Region.

12. Cross-origin resource sharing (CORS) defines a way for client web applications that are loaded in one domain to interact with resources in a different domain. In this scenario the S3 bucket is the requestor and is requesting access to resources served by Amazon API Gateway and AWS Lambda. Therefore, the CORS configuration must be enabled on the requested endpoint which is the method in API Gateway.

13. You can use Amazon S3 to host a static website. On a *static* website, individual web pages include static content. They might also contain client-side scripts.

    To host a static website on Amazon S3, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. When you configure a bucket as a static website, you enable static website hosting, set permissions, and add an index document.

    When you enable static website hosting for your bucket, you enter the name of the index document (for example, index.html). After you enable static website

hosting for your bucket, you upload an HTML file with the index document name to your bucket. Note that an error document is optional.

To provide permissions, it is necessary to disable “block public access” settings and then create a bucket policy that grants everyone the `s3:GetObject` permission. For example:

```
{
"Version": "2012-10-17",
"Statement": [
{
"Sid": "PublicReadGetObject",
"Effect": "Allow",
"Principal": "*",
"Action": [
"s3:GetObject"
],
"Resource": [
"arn:aws:s3:::example.com/*"
]
}
]
}
```

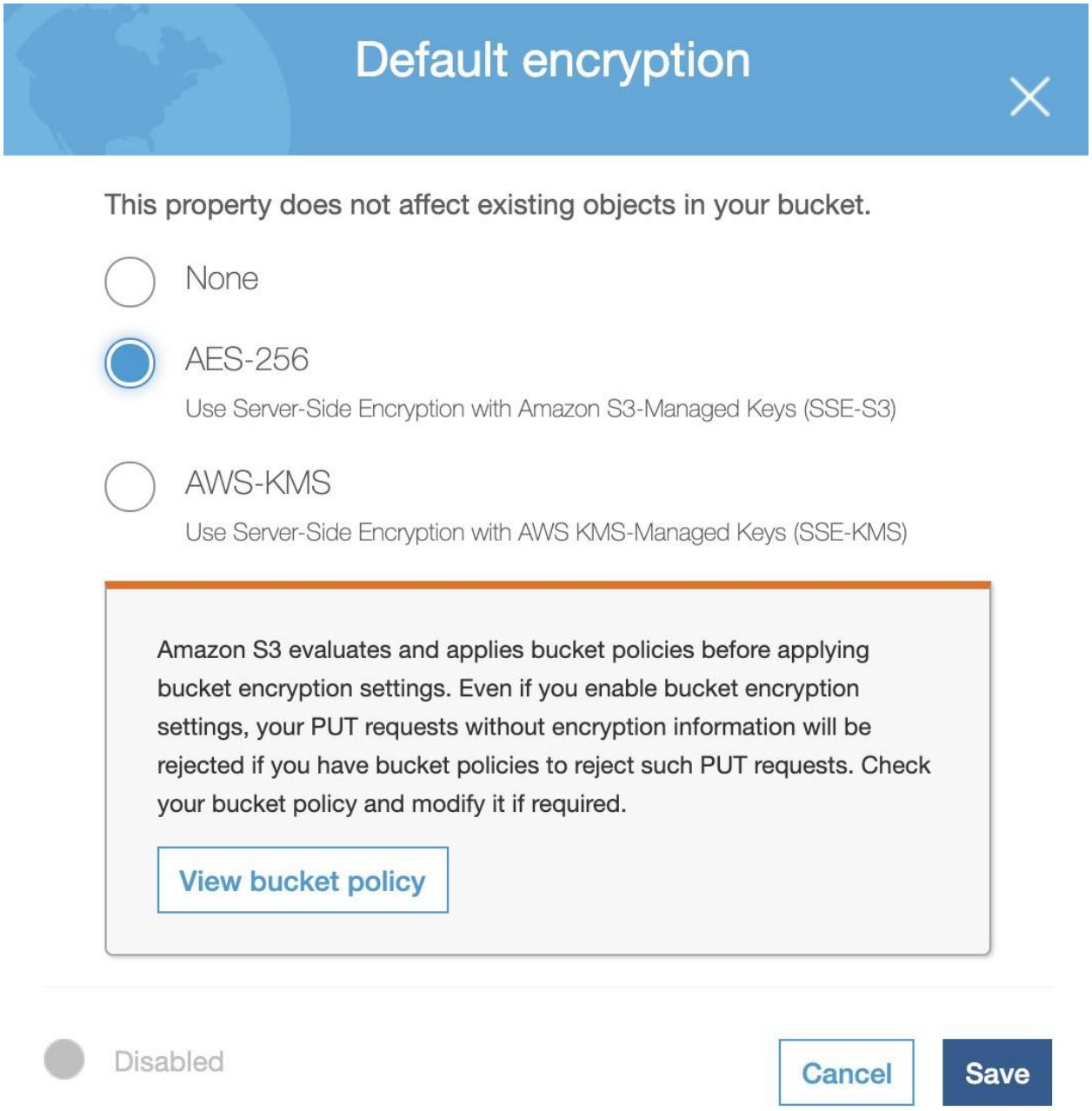
14. Amazon S3 provides read-after-write consistency for PUTS of new objects in your S3 bucket in all Regions with one caveat. The caveat is that if you make a HEAD or GET request to a key name before the object is created, then create the object shortly after that, a subsequent GET might not return the object due to eventual consistency.

    Amazon S3 offers eventual consistency for overwrite PUTS and DELETES in all Regions. Therefore, the most likely explanation for this issue is that the old object version was retrieved due to Amazon’s eventual consistency model for overwrite PUTS.

15. To encrypt an object at the time of upload, you need to add a header called x-amz-server-side-encryption to the request to tell S3 to encrypt the object using SSE-C, SSE-S3, or SSE-KMS. The following code example shows a Put request using SSE-S3.

```
PUT /example-object HTTP/1.1
Host: myBucket.s3.amazonaws.com
Date: Wed, 8 Jun 2016 17:50:00 GMT
Authorization: authorization string
Content-Type: text/plain
Content-Length: 11434
x-amz-meta-author: Janet
Expect: 100-continue
x-amz-server-side-encryption: AES256
[11434 bytes of object data]
```

Enabling encryption on an S3 bucket does not enforce encryption however, so it is still necessary to take extra steps to force compliance with the policy. As the message in the image below states, bucket policies are applied before encryption settings so PUT requests without encryption information can be rejected by a bucket policy:



Therefore, we need to create an S3 bucket policy that denies any S3 Put request that does not include the x-amz-server-side-encryption header. There are two possible values for the x-amz-server-side-encryption header: AES256, which tells S3 to use S3-managed keys, and aws:kms, which tells S3 to use AWS KMS–managed keys.

16. Replication enables automatic, asynchronous copying of objects across Amazon S3 buckets. Buckets that are configured for object replication can be owned by

the same AWS account or by different accounts. You can copy objects between different AWS Regions or within the same Region.

To enable object replication, you add a replication configuration to your source bucket. The minimum configuration must provide the following:

The destination bucket where you want Amazon S3 to replicate objects

An AWS Identity and Access Management (IAM) role that Amazon S3 can assume to replicate objects on your behalf

You can replicate objects between different AWS Regions or within the same AWS Region.

17. When Amazon S3 objects are private, only the object owner has permission to access these objects. However, the object owner can optionally share objects with others by creating a presigned URL, using their own security credentials, to grant time-limited permission to download the objects.

    When you create a presigned URL for your object, you must provide your security credentials, specify a bucket name, an object key, specify the HTTP method (GET to download the object) and expiration date and time. The presigned URLs are valid only for the specified duration.

    Anyone who receives the presigned URL can then access the object. In this scenario, the photos can be shared with the owner's website but not with any other 3rd parties. This will stop other sites from linking to the photos as they will not display anywhere else.

18. The first time you invoke your function, AWS Lambda creates an instance of the function and runs its handler method to process the event. When the function returns a response, it stays active and waits to process additional events. If you invoke the function again while the first event is being processed, Lambda initializes another instance, and the function processes the two events concurrently.

    Your functions' concurrency is the number of instances that serve requests at a given time. For an initial burst of traffic, your functions' cumulative concurrency in a Region can reach an initial level of between 500 and 3000, which varies per Region.

Burst Concurrency Limits:

• 3000 – US West (Oregon), US East (N. Virginia), Europe (Ireland).

• 1000 – Asia Pacific (Tokyo), Europe (Frankfurt).

• 500 – Other Regions.

After the initial burst, your functions' concurrency can scale by an additional 500 instances each minute. This continues until there are enough instances to serve all requests, or until a concurrency limit is reached.

The default account limit is up to 1000 executions per second, per region (can be increased).

19. A presigned URL gives you access to the object identified in the URL, provided that the creator of the presigned URL has permissions to access that object. That is, if you receive a presigned URL to upload an object, you can upload the object only if the creator of the presigned URL has the necessary permissions to upload that object.

    You can use the AWS SDK for Java to generate a presigned URL that you, or anyone you give the URL, can use to upload an object to Amazon S3. When you use the URL to upload an object, Amazon S3 creates the object in the specified bucket.

    If an object with the same key that is specified in the presigned URL already exists in the bucket, Amazon S3 replaces the existing object with the uploaded object. To successfully complete an upload, you must do the following:

    • Specify the HTTP PUT verb when creating the GeneratePresignedUrlRequest and HttpURLConnection objects.

    • Interact with the HttpURLConnection object in some way after finishing the upload. The following example accomplishes this by using the HttpURLConnection object to check the HTTP response code.

20. Amazon S3 is object storage built to store and retrieve any amount of data from anywhere on the Internet. Amazon S3 uses standards-based REST and SOAP interfaces designed to work with any internet-development toolkit.

Amazon S3 is a simple key-based object store. The key is the name of the object and the value is the actual data itself. Keys can be any string, and they can be constructed to mimic hierarchical attributes.

B. AWS SECURITY

1. When you access AWS programmatically, you use an access key to verify your identity and the identity of your applications. An access key consists of an access key ID (something like AKIAIOSFODNN7EXAMPLE) and a secret access key (something like wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY).

   Anyone who has your access key has the same level of access to your AWS resources that you do. Steps to protect access keys include the following:

   • Remove (or Don't Generate) Account Access Key – this is especially important for the root account.

   • Use Temporary Security Credentials (IAM Roles) Instead of Long-Term Access Keys.

   • Don't embed access keys directly into code.

   • Use different access keys for different applications.

   • Rotate access keys periodically.

   • Remove unused access keys.

   • Configure multi-factor authentication for your most sensitive operations.

2. According to the principal of least privilege the Developer needs to provide the minimum permissions that application requires. The application needs read-only access and therefore an IAM role with an AmazonDynamoDBReadOnlyAccess policy applied that only provides read-only access to DynamoDB is secure.

   This role can be applied to the EC2 instance through the management console or programmatically by creating an instance profile and attaching the role to the instance profile. The EC2 instance can then assume the role and get read-only access to DynamoDB.

3. Amazon Cognito lets you add user sign-up, sign-in, and access control to your web and mobile apps quickly and easily. Amazon Cognito provides authentication, authorization, and user management for your web and mobile apps.

   Amazon Cognito identity pools enable you to create unique identities for your users and authenticate them with identity providers. With an identity, you can obtain temporary, limited-privilege AWS credentials to access other AWS services.

   Amazon Cognito supports developer authenticated identities, in addition to web identity federation through Facebook (Identity Pools), Google (Identity Pools), and Login with Amazon (Identity Pools).

   With developer authenticated identities, you can register and authenticate users via your own existing authentication process, while still using Amazon Cognito to synchronize user data and access AWS resources. Using developer authenticated identities involves interaction between the end user device, your backend for authentication, and Amazon Cognito.

4. Access keys consist of two parts:

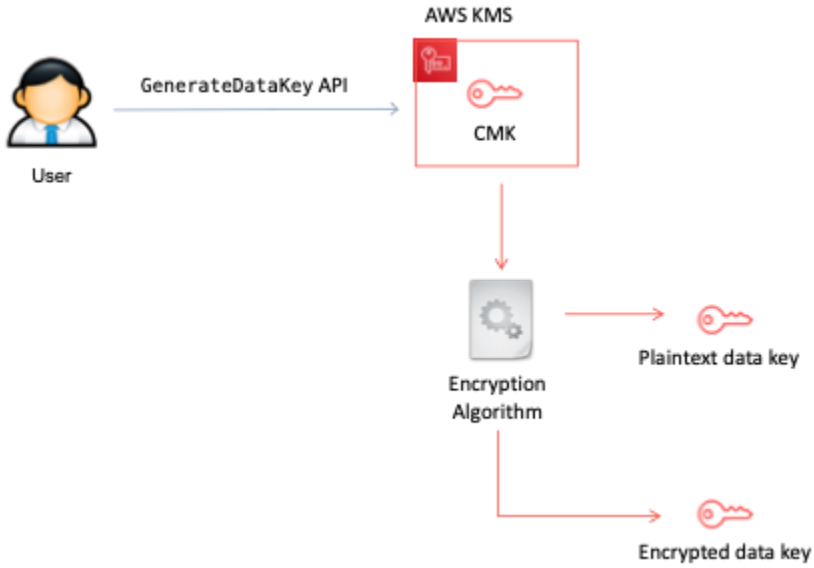
   The access key identifier. This is not a secret, and can be seen in the IAM console wherever access keys are listed, such as on the user summary page.

   The secret access key. This is provided when you initially create the access key pair. Just like a password, it ***cannot be retrieved later***. If you lost your secret access key, then you must create a new access key pair. If you already have the maximum number of access keys, you must delete an existing pair before you can create another.

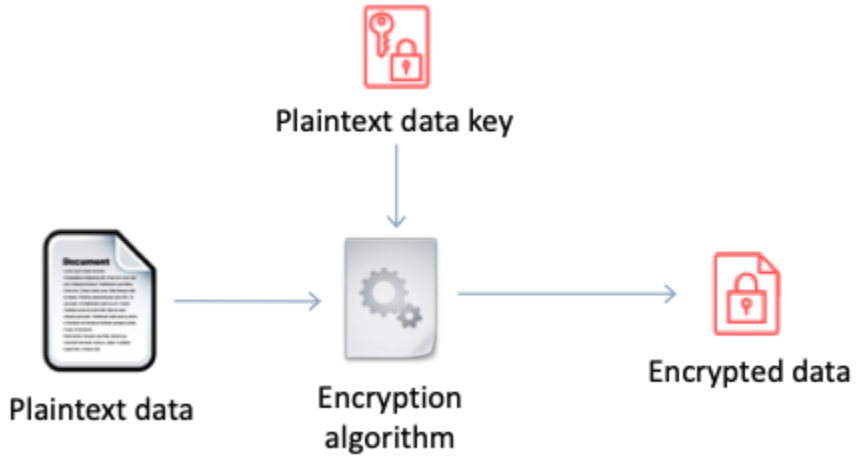
   Therefore, the Developer should disable and delete their access keys and generate a new set.

5. To encrypt large quantities of data with the AWS Key Management Service (KMS), you must use a data encryption key rather than a customer master keys (CMK). This is because a CMK can only encrypt up to 4KB in a single operation and in this scenario the objects are 1 GB in size.

To create a data key, call the GenerateDataKey operation. AWS KMS uses the CMK that you specify to generate a data key. The operation returns a plaintext copy of the data key and a copy of the data key encrypted under the CMK. The following image shows this operation.



AWS KMS cannot use a data key to encrypt data. But you can use the data key outside of KMS, such as by using OpenSSL or a cryptographic library like the AWS Encryption SDK. Data can then be encrypted using the plaintext data key as depicted below:



Therefore, the Developer should make a GenerateDataKey API call that returns a plaintext key and an encrypted copy of a data key, and then use the plaintext key to encrypt the data.

6. In some cases, you might not know the exact name of the resource when you write the policy. You might want to generalize the policy so it works for many

users without having to make a unique copy of the policy for each user. For example, consider writing a policy to allow each user to have access to his or her own objects in an Amazon S3 bucket.

Instead of that explicitly specifies the user's name as part of the resource, create a single group policy that works for any user in that group. You can do this by using *policy variables*, a feature that lets you specify placeholders in a policy. When the policy is evaluated, the policy variables are replaced with values that come from the context of the request itself.

The following example shows a policy for an Amazon S3 bucket that uses a policy variable.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${aws:username}/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws:s3:::mybucket/${aws:username}/*"]
    }
  ]
}
```

When this policy is evaluated, IAM replaces the variable ${aws:username}with the friendly name of the actual current user. This means that a single policy applied to a group of users can control access to a bucket by using the username as part of the resource's name.

7. When you encrypt your data, your data is protected, but you have to protect your encryption key. One strategy is to encrypt it. *Envelope encryption* is the practice of encrypting plaintext data with a data key, and then encrypting the data key under another key.

You can even encrypt the data encryption key under another encryption key and encrypt that encryption key under another encryption key. But, eventually, one key must remain in plaintext so you can decrypt the keys and your data. This top-level plaintext key encryption key is known as the *master key*.

Envelope encryption offers several benefits:

• **Protecting data keys**

When you encrypt a data key, you don't have to worry about storing the encrypted data key, because the data key is inherently protected by encryption. You can safely store the encrypted data key alongside the encrypted data.

• **Encrypting the same data under multiple master keys**

Encryption operations can be time consuming, particularly when the data being encrypted are large objects. Instead of re-encrypting raw data multiple times with different keys, you can re-encrypt only the data keys that protect the raw data.

• **Combining the strengths of multiple algorithms**

In general, symmetric key algorithms are faster and produce smaller ciphertexts than public key algorithms. But public key algorithms provide inherent separation of roles and easier key management. Envelope encryption lets you combine the strengths of each strategy.

As described above, the process that should be implemented is to encrypt plaintext data with a data key and then encrypt the data key with a top-level plaintext master key.

8. The Lambda function needs the privileges to use the SendMessage API action on the Amazon SQS queue. The permissions should be assigned to the function's

execution role. The AWSLambdaSQSQueueExecutionRole AWS managed policy cannot be used as this policy does not provide the SendMessage action.

The Developer should therefore create a customer managed policy with read/write permissions to SQS and attach the policy to the function's execution role.

9. For this scenario, the Developers will be connecting programmatically to AWS resources and will therefore be required to use an access key ID and secret access key.

10. A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito. Your users can also sign in through social identity providers like Google, Facebook, Amazon, or Apple, and through SAML identity providers.

    Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through a Software Development Kit (SDK).

    User pools provide:

    • Sign-up and sign-in services.

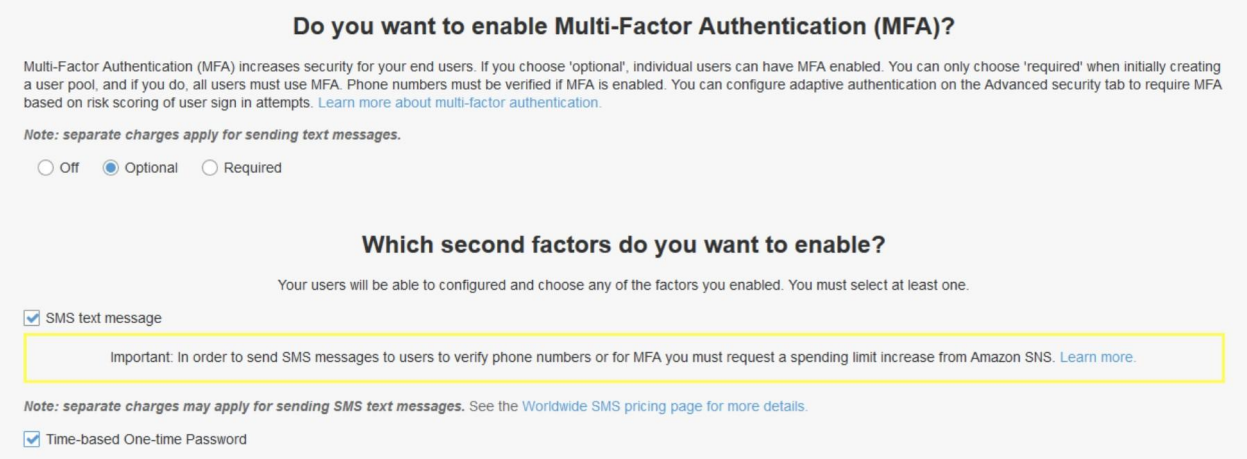
    • A built-in, customizable web UI to sign in users.

    • Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple, as well as sign-in with SAML identity providers from your user pool.

    • User directory management and user profiles.

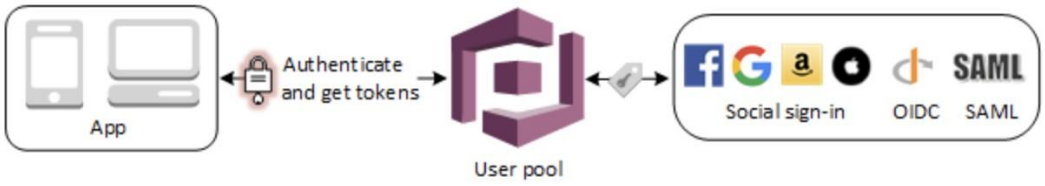
    • Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification.

Multi-factor authentication (MFA) increases security for your app by adding another authentication method, and not relying solely on username and password. You can choose to use SMS text messages, or time-based one-time (TOTP) passwords as second factors in signing in your users.



• Customized workflows and user migration through AWS Lambda triggers.

After successfully authenticating a user, Amazon Cognito issues JSON web tokens (JWT) that you can use to secure and authorize access to your own APIs, or exchange for AWS credentials.

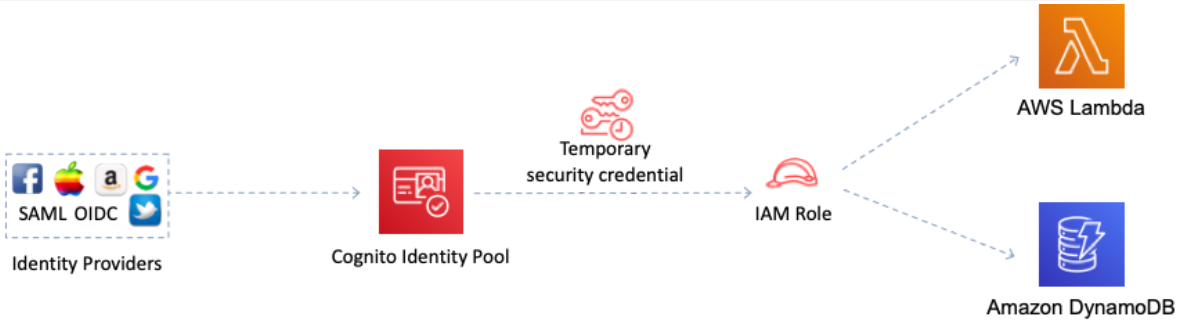


Therefore, an Amazon Cognito user pool is the best solution for enabling sign-up to the new web application.

11. Amazon Inspector is an automated security assessment service that helps improve the security and compliance of applications deployed on AWS. Amazon Inspector automatically assesses applications for exposure, vulnerabilities, and deviations from best practices.

    After performing an assessment, Amazon Inspector produces a detailed list of security findings prioritized by level of severity. These findings can be reviewed directly or as part of detailed assessment reports which are available via the Amazon Inspector console or API.

12. Amazon Cognito identity pools provide temporary AWS credentials for users who are guests (unauthenticated) and for users who have been authenticated and received a token. An identity pool is a store of user identity data specific to your account.



Amazon Cognito identity pools support both authenticated and unauthenticated identities. Authenticated identities belong to users who are authenticated by any supported identity provider. Unauthenticated identities typically belong to guest users.

• To configure authenticated identities with a public login provider, see Identity Pools (Federated Identities) External Identity Providers.

• To configure your own backend authentication process, see Developer Authenticated Identities (Identity Pools).

Therefore, the Developer should create a new identity pool, enable access to unauthenticated identities, and grant access to AWS resources.

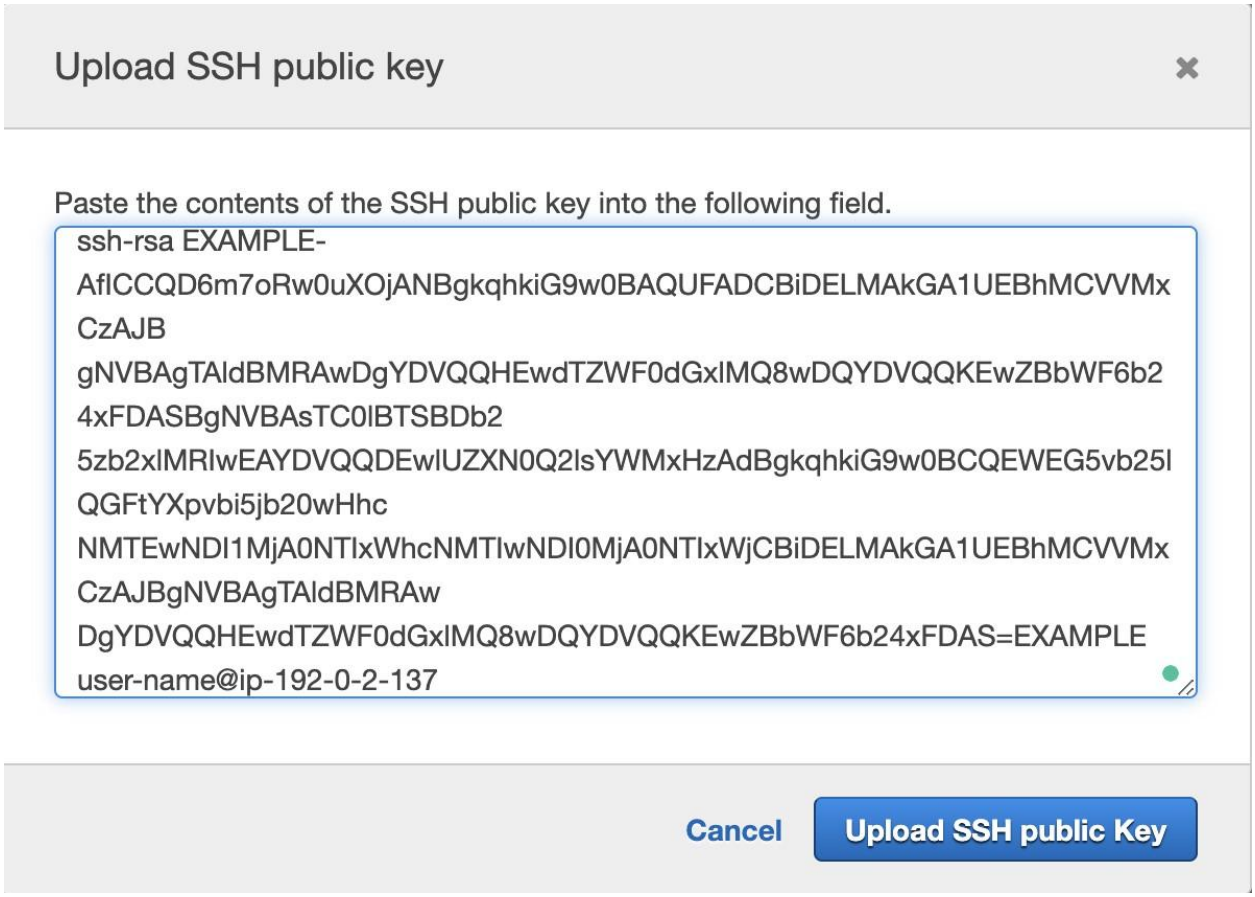
13. You need to configure your Git client to communicate with CodeCommit repositories. As part of this configuration, you provide IAM credentials that CodeCommit can use to authenticate you. IAM supports CodeCommit with three types of credentials:

• Git credentials, an IAM -generated user name and password pair you can use to communicate with CodeCommit repositories over HTTPS.

• SSH keys, a locally generated public-private key pair that you can associate with your IAM user to communicate with CodeCommit repositories over SSH.

• AWS access keys, which you can use with the credential helper included with the AWS CLI to communicate with CodeCommit repositories over HTTPS.

As the Developer is going to use SSH (in the given question), he first needs to generate an SSH private and public key. These can then be used for authentication. The method of creating these depends on the operating system the Developer is using. Then, the Developer can upload the public key (by copying the contents of the file) into his IAM account under security credentials.



14. You use AWS WAF to control how an Amazon CloudFront distribution, an Amazon API Gateway API, or an Application Load Balancer responds to web requests.

• **Web ACLs** – You use a web access control list (ACL) to protect a set of AWS resources. You create a web ACL and define its protection strategy by adding rules. Rules define criteria for inspecting web requests and specify how to handle requests that match the criteria. You set a default action for the web ACL that indicates whether to block or allow through those requests that pass the rules inspections.

• **Rules** – Each rule contains a statement that defines the inspection criteria, and an action to take if a web request meets the criteria. When a web request meets the criteria, that's a match. You can use rules to block matching requests or to allow matching requests through. You can also use rules just to count matching requests.

• **Rules groups** – You can use rules individually or in reusable rule groups. AWS Managed Rules and AWS Marketplace sellers provide managed rule groups for your use. You can also define your own rule groups.

After you create your web ACL, you can associate it with one or more AWS resources. The resource types that you can protect using AWS WAF web ACLs are Amazon CloudFront distributions, Amazon API Gateway APIs, and Application Load Balancers.

15. Service control policies (SCPs) are one type of policy that you can use to manage your organization. SCPs offer central control over the maximum available permissions for all accounts in your organization, allowing you to ensure your accounts stay within your organization's access control guidelines.

    You can configure the SCPs in your organization to work as either of the following:

    • A deny list – actions are allowed by default, and you specify what services and actions are prohibited

    • An allow list – actions are prohibited by default, and you specify what services and actions are allowed

    As there are only a few API actions to restrict the most efficient strategy for this scenario is to create a deny list and specify the specific actions that are prohibited.

16. In this case (the question) the Developer's access keys may have been compromised so the first step would be to invalidate the access keys by deleting them.

    The next step would then be to determine if any temporary security credentials have been issued and invalidating those too to prevent any further misuse.

The user account and user account password have not been compromised so they do not need to be deleted / changed as a first step. However, changing the account password would typically be recommended as a best practice in this situation.

17. At the Amazon S3 bucket level, you can configure permissions through a bucket policy. For example, you can limit access to the objects in a bucket by IP address range or specific IP addresses. Alternatively, you can make the objects accessible only through HTTPS.

    The following bucket policy allows access to Amazon S3 objects only through HTTPS (the policy was generated with the AWS Policy Generator).

```
{
    "Version": "2012-10-17",
    "Id": "Policy1504640911349",
    "Statement": [
        {
            "Sid": "Stmt1504640908907",
            "Effect": "Deny",
            "Principal": "*",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::/*",
            "Condition": {
                "Bool": {
                    "aws:SecureTransport": "false"
                }
            }
        }
    ]
}
```

    Here the bucket policy explicitly denies ("Effect": "Deny") all read access ("Action": "s3:GetObject") from anybody who browses ("Principal": "*") to Amazon S3 objects within an Amazon S3 bucket if they are not accessed through HTTPS ("aws:SecureTransport": "false").

18. AWS KMS establishes quotas for the number of API operations requested in each second. When you exceed an API request quota, AWS KMS *throttles* the request,

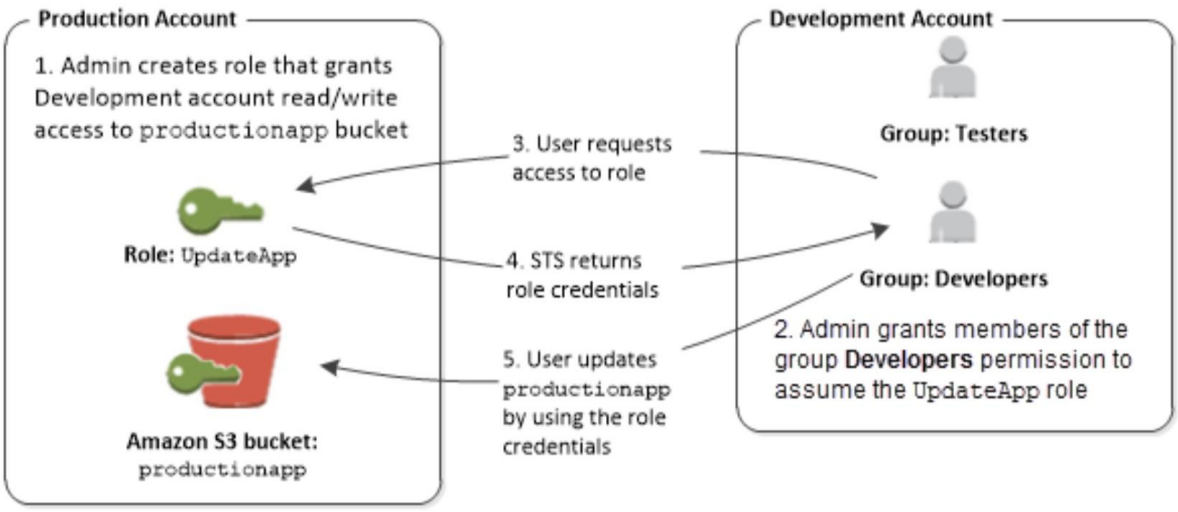
that is, it rejects an otherwise valid request and returns a ThrottlingException error like the following one.

```
You have exceeded the rate at which you may call KMS. Reduce the frequency of your calls.
(Service: AWSKMS; Status Code: 400; Error Code: ThrottlingException; Request ID: <ID>
```

As the error indicates, one of the recommendations is to reduce the frequency of calls which can be implemented by using exponential backoff logic in the application code. It is also possible to contact AWS and request an increase in the quota.

19. This should be implemented using a role in the Production account and a group in the Development account. The developer in the Development account would then be added to the group. The role in the Production account would provide the necessary access and would allow the group in the Development account to assume the role.

    The following image depicts this setup:



    Therefore, the most secure way to achieve the required access is to use a role in the Production account that the user is able to assume and then the user can request short-lived credentials from the Security Token Service (STS).

20. The AWS CloudHSM service helps you meet corporate, contractual, and regulatory compliance requirements for data security by using dedicated Hardware Security Module (HSM) instances within the AWS cloud.

    A Hardware Security Module (HSM) provides secure key storage and cryptographic operations within a tamper-resistant hardware device. CloudHSM

allows you to securely generate, store, and manage cryptographic keys used for data encryption in a way that keys are accessible only by you.

A large quantity of sensitive data must be encrypted. A Developer will use a custom CMK to generate the encryption key. The key policy currently looks like this:

```
{
"Sid": "Allow Key Usage",
"Effect": "Allow",
"Principal": {"AWS": [
"arn:aws:iam::111122223333:user/CMKUser"
]},
"Action": [
"kms:Encrypt",
"kms:Decrypt",
"kms:ReEncrypt*",
"kms:DescribeKey"
],
"Resource": "*"
}
```

What API action must be added to the key policy?

- `kms:GetKeyPolicy`
-
- `kms:EnableKey`
-
- `kms:GenerateDataKey`
  - **(Correct)**
-
- `kms:CreateKey`
  - **(Incorrect)**

Explanation

A key policy is a document that uses JSON (JavaScript Object Notation) to specify permissions. You can work with these JSON documents directly, or you can use the AWS Management Console to work with them using a graphical interface called the *default view*.

The key policy supplied with this question is missing the GenerateDataKey API action which is a permission that is required to generate a data encryption key. A data encryption key is required to encrypt large amounts of data as a CMK can only encrypt up to 4 KB.

The GenerateDataKey API Generates a unique symmetric data key. This operation returns a plaintext copy of the data key and a copy that is encrypted under a customer master key (CMK) that you specify. You can use the plaintext key to encrypt your data outside of AWS KMS and store the encrypted data key with the encrypted data.

21. Use the Principal element in a policy to specify the principal that is allowed or denied access to a resource. You cannot use the Principal element in an IAM identity-based policy. You can use it in the trust policies for IAM roles and in resource-based policies. Resource-based policies are policies that you embed directly in an IAM resource.
22. Amazon Cognito identity pools (federated identities) enable you to create unique identities for your users and federate them with identity providers. With an identity pool, you can obtain temporary, limited-privilege AWS credentials to access other AWS services. Amazon Cognito identity pools support the following identity providers:

    • Public providers: Login with Amazon (Identity Pools), Facebook (Identity Pools), Google (Identity Pools) Sign in with Apple (Identity Pools).

    • Amazon Cognito User Pools

    • Open ID Connect Providers (Identity Pools)

    • SAML Identity Providers (Identity Pools)

    • Developer Authenticated Identities (Identity Pools)

    With the temporary, limited-privilege AWS credentials users will be able to access the images in the S3 bucket. Therefore, the Developer should use Amazon Cognito with web identity federation

23. AWS WAF is a web application firewall service that helps protect your web apps from common exploits that could affect app availability, compromise security, or consume excessive resources.

    AWS WAF helps protect web applications from attacks by allowing you to configure rules that allow, block, or monitor (count) web requests based on conditions that you define. These conditions include IP addresses, HTTP headers, HTTP body, URI strings, SQL injection and cross-site scripting.

    AWS WAF gives you control over which traffic to allow or block to your web applications by defining customizable web security rules.

24. AWS provide a number of best practices for AWS IAM that help you to secure your resources. The key best practices referenced in this scenario are as follows:

    • Use groups to assign permissions to users – this is correct as you should create permissions policies and assign them to groups. Users can be added to the groups to get the permissions they need to perform their jobs.

    • Create standalone policies instead of using inline policies (Use Customer Managed Policies Instead of Inline Policies in the AWS best practices) – this refers to creating your own policies that are standalone policies which can be reused multiple times (assigned to multiple entities such as groups, and users). This is better than using inline policies which are directly attached to a single entity.

25. Amazon Cognito lets you save end user data in datasets containing key-value pairs. This data is associated with an Amazon Cognito identity, so that it can be accessed across logins and devices. To sync this data between the Amazon Cognito service and an end user's devices, invoke the synchronize method. Each dataset can have a maximum size of 1 MB. You can associate up to 20 datasets with an identity.

    The Amazon Cognito Sync client creates a local cache for the identity data. Your app talks to this local cache when it reads and writes keys. This guarantees that all of your changes made on the device are immediately available on the device, even when you are offline. When the synchronize method is called, changes from the service are pulled to the device, and any local changes are pushed to the service. At this point the changes are available to other devices to synchronize.

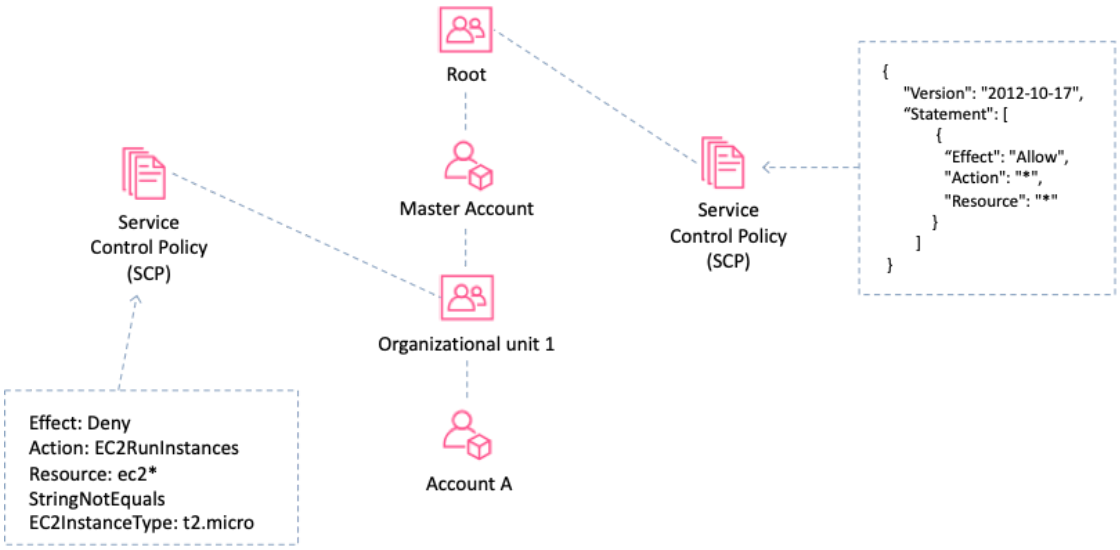
26. The GetSessionToken API call returns a set of temporary credentials for an AWS account or IAM user. The credentials consist of an access key ID, a secret access key, and a security token. Typically, you use GetSessionToken if you want to use MFA to protect programmatic calls to specific AWS API operations

    Therefore, the Developer can use GetSessionToken with an MFA device to make secure API calls using the AWS SDK.

27. As an administrator of the master account of an organization, you can use service control policies (SCPs) to specify the maximum permissions for member accounts in the organization.

    In SCPs, you can restrict which AWS services, resources, and individual API actions the users and roles in each member account can access. You can also define conditions for when to restrict access to AWS services, resources, and API actions.

    The following example shows how an SCP can be created to restrict the EC2 instance types that any user can run in the account:



    These restrictions even override the administrators of member accounts in the organization. When AWS Organizations blocks access to a service, resource, or API action for a member account, a user or role in that account can't access it. This block remains in effect even if an administrator of a member account explicitly grants such permissions in an IAM policy.

28. The AWS STS decode-authorization-message API decodes additional information about the authorization status of a request from an encoded message returned in response to an AWS request. The output is then decoded into a more human-readable output that can be viewed in a JSON editor.

    The following example is the decoded output from the error shown in the question:


```
{
"DecodedMessage":
"{\"allowed\":false,\"explicitDeny\":false,\"matchedStatements\":{\"
items\":[]},\"failures\":{\"items\":[]},\"context\":{\"principal\":{
\"id\":\"AIDAXP4J2EKU7YXXG3EJ4\",\"name\":\"Paul\",\"arn\":\"arn:aws
:iam::515148227241:user/Paul\"},\"action\":\"ec2:RunInstances\",\"re
source\":\"arn:aws:ec2:ap-southeast-2:515148227241:instance/*\",\"co
nditions\":{\"items\":[{\"key\":\"ec2:InstanceMarketType\",\"values\
":{\"items\":[{\"value\":\"on-demand\"}]}},{\"key\":\"aws:Resource\"
,\"values\":{\"items\":[{\"value\":\"instance/*\"}]}},{\"key\":\"aws
:Account\",\"values\":{\"items\":[{\"value\":\"515148227241\"}]}},{\
"key\":\"ec2:AvailabilityZone\",\"values\":{\"items\":[{\"value\":\"
ap-southeast-2a\"}]}},{\"key\":\"ec2:ebsOptimized\",\"values\":{\"it
ems\":[{\"value\":\"false\"}]}},{\"key\":\"ec2:IsLaunchTemplateResou
rce\",\"values\":{\"items\":[{\"value\":\"false\"}]}},{\"key\":\"ec2
:InstanceType\",\"values\":{\"items\":[{\"value\":\"t2.micro\"}]}},{
\"key\":\"ec2:RootDeviceType\",\"values\":{\"items\":[{\"value\":\"e
bs\"}]}},{\"key\":\"aws:Region\",\"values\":{\"items\":[{\"value\":\
"ap-southeast-2\"}]}},{\"key\":\"aws:Service\",\"values\":{\"items\"
:[{\"value\":\"ec2\"}]}},{\"key\":\"ec2:InstanceID\",\"values\":{\"i
tems\":[{\"value\":\"*\"}]}},{\"key\":\"aws:Type\",\"values\":{\"ite
ms\":[{\"value\":\"instance\"}]}},{\"key\":\"ec2:Tenancy\",\"values\
":{\"items\":[{\"value\":\"default\"}]}},{\"key\":\"ec2:Region\",\"v
alues\":{\"items\":[{\"value\":\"ap-southeast-2\"}]}},{\"key\":\"aws
:ARN\",\"values\":{\"items\":[{\"value\":\"arn:aws:ec2:ap-southeast-
2:515148227241:instance/*\"}]}}]}}}"
}
```


    Therefore, the best answer is to use the AWS STS decode-authorization-message API to decode the message.

29. A user pool is a user directory in Amazon Cognito. With a user pool, your users can sign into your web or mobile app through Amazon Cognito. Your users can also sign in through social identity providers like Google, Facebook, Amazon, or Apple, and through SAML identity providers. Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through a Software Development Kit (SDK).

As an alternative to using IAM roles and policies or Lambda authorizers (formerly known as custom authorizers), you can use an Amazon Cognito user pool to control who can access your API in Amazon API Gateway.

To use an Amazon Cognito user pool with your API, you must first create an authorizer of the COGNITO_USER_POOLS type and then configure an API method to use that authorizer. After the API is deployed, the client must first sign the user in to the user pool, obtain an identity or access token for the user, and then call the API method with one of the tokens, which are typically set to the request's Authorization header. The API call succeeds only if the required token is supplied and the supplied token is valid, otherwise, the client isn't authorized to make the call because the client did not have credentials that could be authorized.

30. Amazon Cognito Identity Pools can support unauthenticated identities by providing a unique identifier and AWS credentials for users who do not authenticate with an identity provider. If your application allows users who do not log in, you can enable access for unauthenticated identities.

    This is the most efficient and secure way to allow unauthenticated access as the process to set it up is simple and the IAM role can be configured with permissions allowing only the access permitted for unauthenticated users.

31. Amazon Cognito supports developer authenticated identities, in addition to web identity federation through Facebook (Identity Pools), Google (Identity Pools), Login with Amazon (Identity Pools), and Sign in with Apple (identity Pools).

    With developer authenticated identities, you can register and authenticate users via your own existing authentication process, while still using Amazon Cognito to synchronize user data and access AWS resources.

    Using developer authenticated identities involves interaction between the end user device, your backend for authentication, and Amazon Cognito.

    Therefore, the Developer can implement developer-authenticated identities by using Amazon Cognito, and get credentials for these identities.

32. The `GenerateDataKey` API is used with the AWS KMS services and generates a unique symmetric data key. This operation returns a plaintext copy of the data key and a copy that is encrypted under a customer master key (CMK) that you

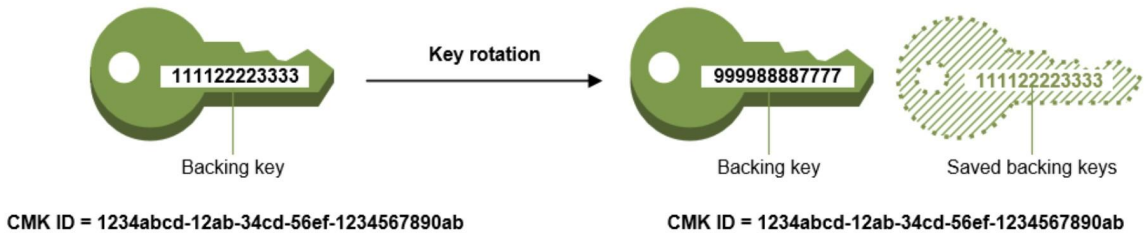
specify. You can use the plaintext key to encrypt your data outside of AWS KMS and store the encrypted data key with the encrypted data.

For this scenario we can use `GenerateDataKey` to obtain an encryption key from KMS that we can then use within the function code to encrypt the file. This ensures that the file is encrypted BEFORE it is uploaded to Amazon S3.

33. Cryptographic best practices discourage extensive reuse of encryption keys. To create new cryptographic material for your AWS Key Management Service (AWS KMS) customer master keys (CMKs), you can create new CMKs, and then change your applications or aliases to use the new CMKs. Or, you can enable automatic key rotation for an existing customer managed CMK.

    When you enable *automatic key rotation* for a customer managed CMK, AWS KMS generates new cryptographic material for the CMK every year. AWS KMS also saves the CMK's older cryptographic material in perpetuity so it can be used to decrypt data that it encrypted. AWS KMS does not delete any rotated key material until you delete the CMK.

    Key rotation changes only the CMK's *backing key*, which is the cryptographic material that is used in encryption operations. The CMK is the same logical resource, regardless of whether or how many times its backing key changes. The properties of the CMK do not change, as shown in the following image.

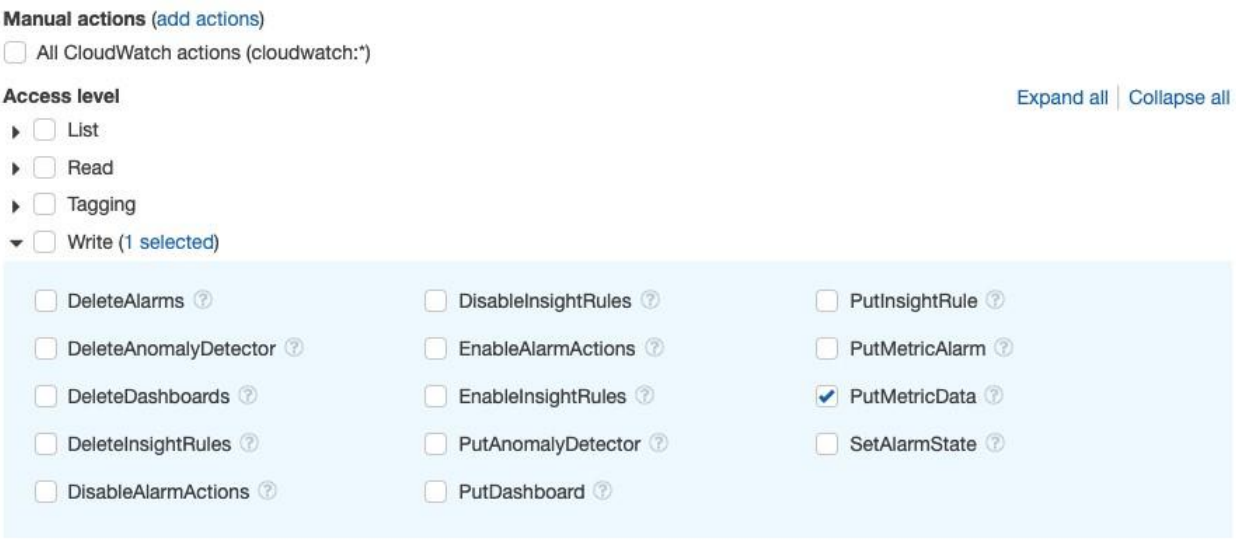


    Therefore, the easiest way to meet this requirement is to use AWS KMS with automatic key rotation.

34. The most secure configuration to authenticate the request is to create an IAM role with a permissions policy that only provides the minimum permissions requires (least privilege). This IAM role should have a customer-managed permissions policy applied with the `PutMetricData` allowed.

The `PutMetricData` API publishes metric data points to Amazon CloudWatch. CloudWatch associates the data points with the specified metric. If the specified metric does not exist, CloudWatch creates the metric. When CloudWatch creates a metric, it can take up to fifteen minutes for the metric to appear in calls to `ListMetrics`.

The following images shows a permissions policy being created with the permission:

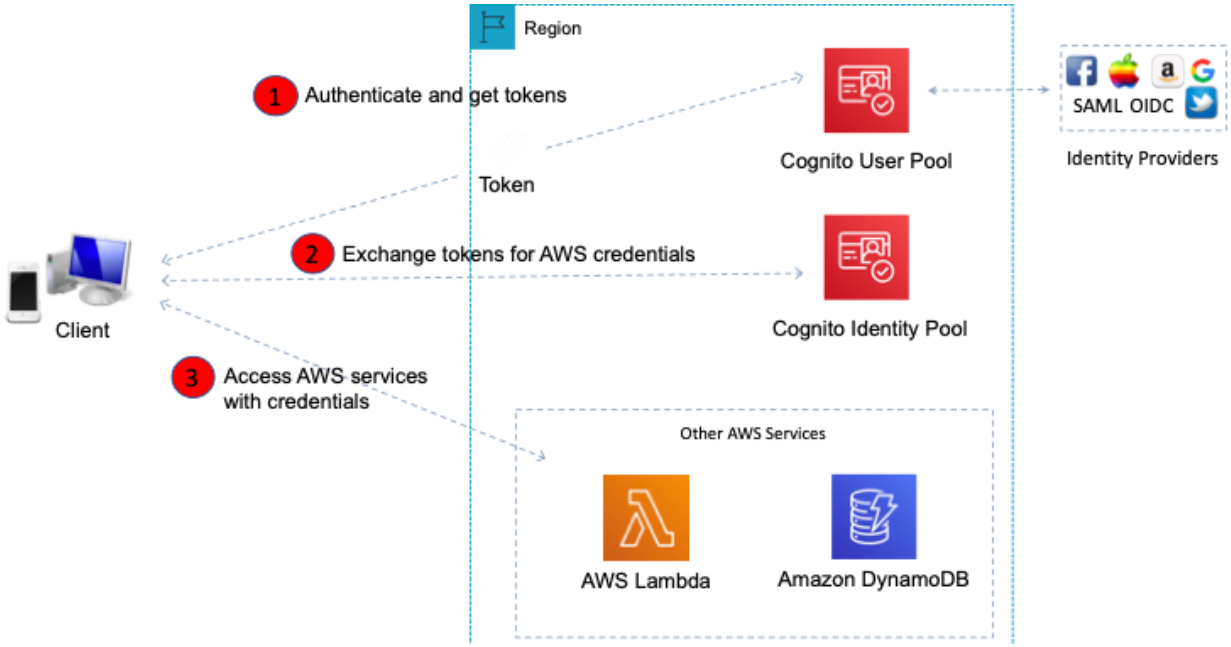


35.

36. There are two key requirements in this scenario. Firstly the company wants to manage user accounts using a system that allows users to reset their own passwords. The company also wants to authorize users to access AWS services.

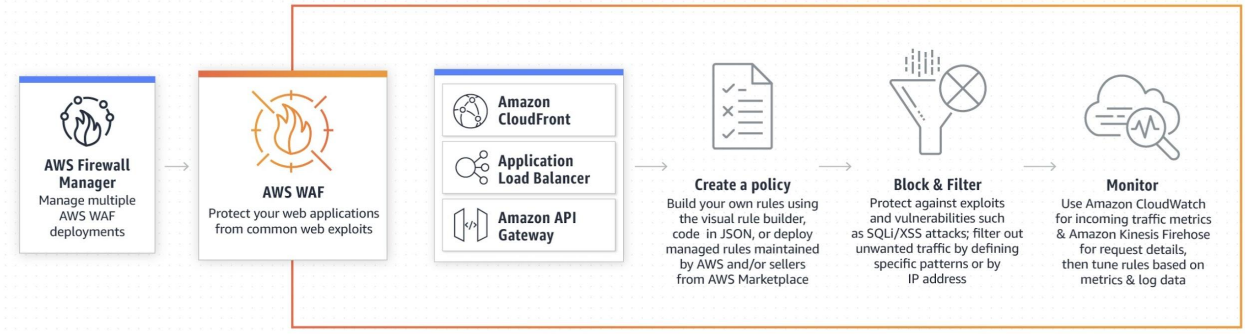
    The first requirement is provided by an Amazon Cognito User Pool. With a Cognito user pool you can add sign-up and sign-in to mobile and web apps and it also offers a user directory so user accounts can be created directly within the user pool. Users also have the ability to reset their passwords.

    To access AWS services you need a Cognito Identity Pool. An identity pool can be used with a user pool and enables a user to obtain temporary limited-privilege credentials to access AWS services.



Therefore, the best answer is to use Amazon Cognito user pools and identity pools.

37. AWS WAF is a web application firewall that helps protect your web applications or APIs against common web exploits that may affect availability, compromise security, or consume excessive resources.



AWS WAF gives you control over how traffic reaches your applications by enabling you to create security rules that block common attack patterns, such as SQL injection or cross-site scripting, and rules that filter out specific traffic patterns you define.

38. **An instance profile is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts. Using an instance profile you can attach an IAM Role to an EC2 instance that the instance can then assume in order to gain access to AWS services.**

39. The `Action` element describes the specific action or actions that will be allowed or denied. Statements must include either an `Action` or `NotAction` element. Each AWS service has its own set of actions that describe tasks that you can perform with that service.

    For this scenario(this question), the `Action` element might include the following JSON"

    ```
    "Action": [ "ec2:StartInstances", "ec2:StopInstances" ]
    ```

40. Applications that run on an EC2 instance must include AWS credentials in their AWS API requests. You could have your developers store AWS credentials directly within the EC2 instance and allow applications in that instance to use those credentials. But developers would then have to manage the credentials and ensure that they securely pass the credentials to each instance and update each EC2 instance when it's time to rotate the credentials. That's a lot of additional work.

    Instead, you can and should use an IAM role to manage *temporary* credentials for applications that run on an EC2 instance. When you use a role, you don't have to distribute long-term credentials (such as a user name and password or access keys) to an EC2 instance. Instead, the role supplies temporary permissions that applications can use when they make calls to other AWS resources. When you launch an EC2 instance, you specify an IAM role to associate with the instance. Applications that run on the instance can then use the role-supplied temporary credentials to sign API requests.

    There are two answers that would work in this scenario. In one a customer-managed policy is used and in the other an AWS managed policy is used. The customer-managed policy is more secure in this situation as it can be locked down with more granularity to ensure the EC2 instances can only read and write to the specific bucket.

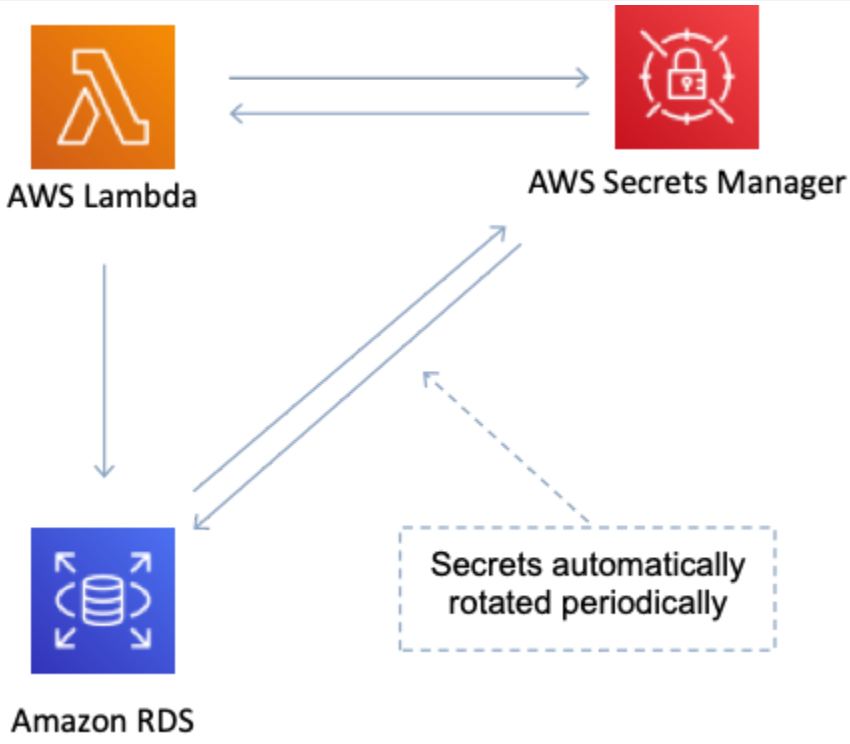
    With an AWS managed policy, you must choose from read only or full access and full access would provide more access than is required:

41. 

42. AWS Secrets Manager encrypts secrets at rest using encryption keys that you own and store in AWS Key Management Service (KMS). When you retrieve a

secret, Secrets Manager decrypts the secret and transmits it securely over TLS to your local environment.

With AWS Secrets Manager, you can rotate secrets on a schedule or on demand by using the Secrets Manager console, AWS SDK, or AWS CLI.



For example, to rotate a database password, you provide the database type, rotation frequency, and master database credentials when storing the password in Secrets Manager. Secrets Manager natively supports rotating credentials for databases hosted on Amazon RDS and Amazon DocumentDB and clusters hosted on Amazon Redshift.

43. Access keys are long-term credentials for an IAM user or the AWS account root user. You can use access keys to sign programmatic requests to the AWS CLI or AWS API (directly or using the AWS SDK).

    Access keys are stored in one of the locations on a client that needs to make authenticated API calls to AWS services:

    · Linux: ~/.aws/credentials

    · Windows: %UserProfle%\.aws\credentials

In this scenario the application server is running on-premises. Therefore, you cannot assign an IAM role (which would be the preferable solution for Amazon EC2 instances). In this case it is therefore better to use access keys.

44. The user needs to verify that the IAM policy grants permission to call sts:AssumeRole for the role that they want to assume. The Action element of an IAM policy must allow you to call the AssumeRole action. In addition, the Resource element of your IAM policy must specify the role that you want to assume.

For this situation the IAM policy would need to include a statement such as this:

```
"Effect": "Allow",
"Action": "sts:AssumeRole",
"Resource": "arn:aws:iam::5123399432:role/MyCARole"
```

45. Applications must sign their API requests with AWS credentials. Therefore, if you are an application developer, you need a strategy for managing credentials for your applications that run on EC2 instances. For example, you can securely distribute your AWS credentials to the instances, enabling the applications on those instances to use your credentials to sign requests, while protecting your credentials from other users.

However, it's challenging to securely distribute credentials to each instance, especially those that AWS creates on your behalf, such as Spot Instances or instances in Auto Scaling groups. You must also be able to update the credentials on each instance when you rotate your AWS credentials.

IAM roles are designed so that your applications can securely make API requests from your instances, without requiring you to manage the security credentials that the applications use.

Instead of creating and distributing your AWS credentials, you can delegate permission to make API requests using IAM roles as follows:

1. Create an IAM role.

2. Define which accounts or AWS services can assume the role.

3. Define which API actions and resources the application can use after assuming the role.

4. Specify the role when you launch your instance or attach the role to an existing instance.

5. Have the application retrieve a set of temporary credentials and use them.

For example, you can use IAM roles to grant permissions to applications running on your instances that need to use a bucket in Amazon S3. You can specify permissions for IAM roles by creating a policy in JSON format. These are similar to the policies that you create for IAM users. If you change a role, the change is propagated to all instances.

Therefore, the best solution is to create an AWS IAM Role with the necessary privileges (through an IAM policy) and attach the role to the instance's instance profile.

46. **With Amazon Cognito User Pools your app users can sign in either directly through a user pool or federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs.**

    After successful authentication, Amazon Cognito returns user pool tokens to your app. You can use the tokens to grant your users access to your own server-side resources, or to the Amazon API Gateway. Or, you can exchange them for AWS credentials to access other AWS services.

    The ID token is a JSON Web Token (JWT) that contains claims about the identity of the authenticated user such as name, email, and phone_number. You can use this identity information inside your application. The ID token can also be used to authenticate users against your resource servers or server applications.

47. **With AWS Systems Manager Parameter Store, you can create secure string parameters, which are parameters that have a plaintext parameter name and an encrypted parameter value. Parameter Store uses AWS KMS to encrypt and decrypt the parameter values of secure string parameters.**

    With Parameter Store you can create, store, and manage data as parameters with values. You can create a parameter in Parameter Store and use it in multiple applications and services subject to policies and permissions that you design. When you need to change a parameter value, you change one instance, rather than managing error-prone changes to numerous sources. Parameter

Store supports a hierarchical structure for parameter names, so you can qualify a parameter for specific uses.

To manage sensitive data, you can create secure string parameters. Parameter Store uses AWS KMS customer master keys (CMKs) to encrypt the parameter values of secure string parameters when you create or change them. It also uses CMKs to decrypt the parameter values when you access them. You can use the AWS managed CMK that Parameter Store creates for your account or specify your own customer managed CMK.

Therefore, you can use a combination of AWS Systems Manager Parameter Store and AWS Key Management Store to store the credentials securely. These keys can be then be referenced in the Lambda function code or through environment variables.

NOTE: Systems Manager Parameter Store does not natively perform rotation of credentials so this must be done in the application. AWS Secrets Manager does perform credential rotation however it is not an answer option for this question.

48. the minimum permissions required for Lambda to create an Amazon CloudWatch Logs group and stream and write logs into the stream. The minimum permissions required are as follows:

· "`logs:CreateLogGroup`" - Creates a log group with the specified name.

· "`logs:CreateLogStream`" - Creates a log stream for the specified log group.

· "`logs:PutLogEvents`" - Uploads a batch of log events to the specified log stream.

49. Client-side encryption is the act of encrypting data before sending it to Amazon S3. You have the following options:

Use a customer master key (CMK) stored in AWS Key Management Service (AWS KMS).

Use a master key you store within your application.

Additionally, using HTTPS/SSL to encrypt the data as it is transmitted over the Internet adds an additional layer of protection.

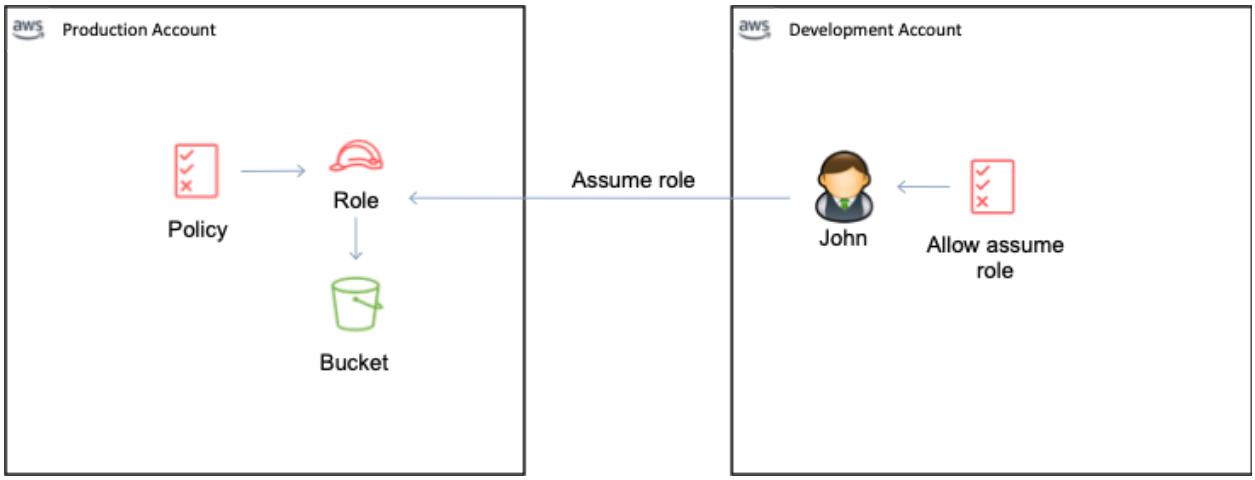
50. The `GetSessionToken` API operation returns a set of temporary security credentials to an existing IAM user. This is useful for providing enhanced security, such as allowing AWS requests only when MFA is enabled for the IAM

user. However, this is used within an account, not across accounts so it not useful here.

In this case the `AssumeRole` API operation should be used to assume the role in the other account. The developer can use the `DurationSeconds` parameter to specify the duration of the role session from 900 seconds (15 minutes) up to the maximum session duration setting for the role.

51. You can grant your IAM users' permission to switch to roles within your AWS account or to roles defined in other AWS accounts that you own. This is known as cross-account access.

    In the image below a user in the Development account needs to access an S3 bucket in the Production account:



    The user is able to assume the role in the Production account and access the S3 bucket. This is more efficient than providing the user with multiple accounts. In this scenario the user requests to switch to the role through either the console or the API/CLI.

52. With AWS KMS you can encrypt files directly with a customer master key (CMK). A CMK can encrypt up to 4KB (4096 bytes) of data in a single encrypt, decrypt, or reencrypt operation. As CMKs cannot be exported from KMS this is a very safe way to encrypt small amounts of data.

    Customer managed CMKs are CMKs in your AWS account that you create, own, and manage. You have full control over these CMKs, including establishing and maintaining their key policies, IAM policies, and grants, enabling and disabling them, rotating their cryptographic material, adding tags, creating aliases that refer to the CMK, and scheduling the CMKs for deletion.

AWS managed CMKs **are CMKs in your account that are created, managed, and used on your behalf by an AWS service that is integrated with AWS KMS. Some AWS services support only an AWS managed CMK. In this example the Amazon EC2 instance is saving files on a proprietary network-attached file system and this will not have support for AWS managed CMKs.**

Data keys **are encryption keys that you can use to encrypt data, including large amounts of data and other data encryption keys. You can use AWS KMS CMKs to generate, encrypt, and decrypt data keys. However, AWS KMS does not store, manage, or track your data keys, or perform cryptographic operations with data keys. You must use and manage data keys outside of AWS KMS – this is potentially less secure as you need to manage the security of these keys.**

53. **With AWS KMS you can choose to have AWS KMS automatically rotate CMKs every year, provided that those keys were generated within AWS KMS HSMs. Automatic key rotation is not supported for imported keys, asymmetric keys, or keys generated in an AWS CloudHSM cluster using the AWS KMS custom key store feature.**

    If you choose to import keys to AWS KMS or asymmetric keys or use a custom key store, you can manually rotate them by creating a new CMK and mapping an existing key alias from the old CMK to the new CMK.

    If you choose to have AWS KMS automatically rotate keys, you don't have to re-encrypt your data. AWS KMS automatically keeps previous versions of keys to use for decryption of data encrypted under an old version of a key. All new encryption requests against a key in AWS KMS are encrypted under the newest version of the key.
54. **AWS Secrets Manager is used for rotating credentials, not encryption keys.**
55. **User pools are for authentication (identify verification). With a user pool, your app users can sign in through the user pool or federate through a third-party identity provider (IdP).**

    Identity pools are for authorization (access control). You can use identity pools to create unique identities for users and give them access to other AWS services.

    User pool use cases:

    Use a user pool when you need to:

    Design sign-up and sign-in webpages for your app.

Access and manage user data.

Track user device, location, and IP address, and adapt to sign-in requests of different risk levels.

Use a custom authentication flow for your app.

Identity pool use cases:

Use an identity pool when you need to:

Give your users access to AWS resources, such as an Amazon Simple Storage Service (Amazon S3) bucket or an Amazon DynamoDB table.

Generate temporary AWS credentials for unauthenticated users.

Therefore, a user pool is the correct service to use as in this case we are not granting access to AWS services, just providing sign-up and sign-in capabilities for a mobile app.

56. **The simplest way to set up connections to AWS CodeCommit repositories is to configure Git credentials for CodeCommit in the IAM console, and then use those credentials for HTTPS connections.**

    You can also use these same credentials with any third-party tool or individual development environment (IDE) that supports HTTPS authentication using a static user name and password. For examples, see For Connections from Development Tools.

57. **You can enable device remembering for Amazon Cognito user pools. A remembered device can serve in place of the security code delivered via SMS as a second factor of authentication. This suppresses the second authentication challenge from remembered devices and thus reduces the friction users experience with multi-factor authentication (MFA).**

Therefore, Amazon Cognito is the best answer and will support all of the requirements in the scenario.

58. The Condition element (or Condition *block*) lets you specify conditions for when a policy is in effect. The Condition element is optional. In the Condition element, you build expressions in which you use condition operators (equal, less than, etc.) to match the condition keys and values in the policy against keys and values in the request context.

    For example, in this scenario the following condition statement could be used:

```
"Condition": {
"DateLessThanEquals" : {
"aws:CurrentTime" : "2020-04-18T12:00:00Z"
}
}
```

59. For general use, the aws configure command is the fastest way to set up your AWS CLI installation. The following example shows sample values:

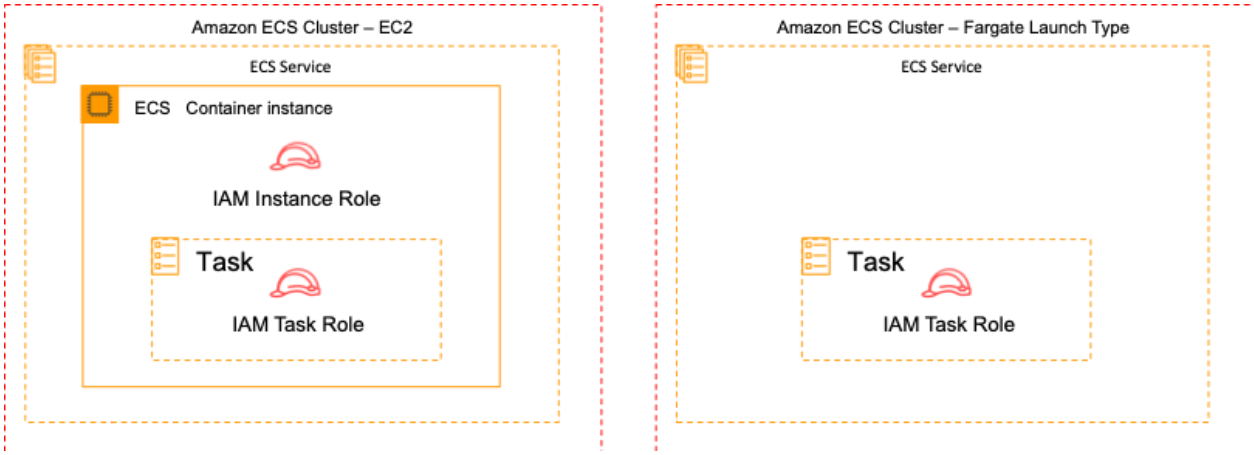
```
$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
```

    You can configure the AWS CLI on Linux, MacOS, and Windows. Computers can be located anywhere as long as they can connect to the AWS API.

    For this scenario, the best solution is to run aws configure and use the IAM user's access key ID and secret access key. This will mean that commands run using the AWS CLI will use the user's IAM permissions as required.

60. With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task. Applications must sign their AWS API requests

with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances.

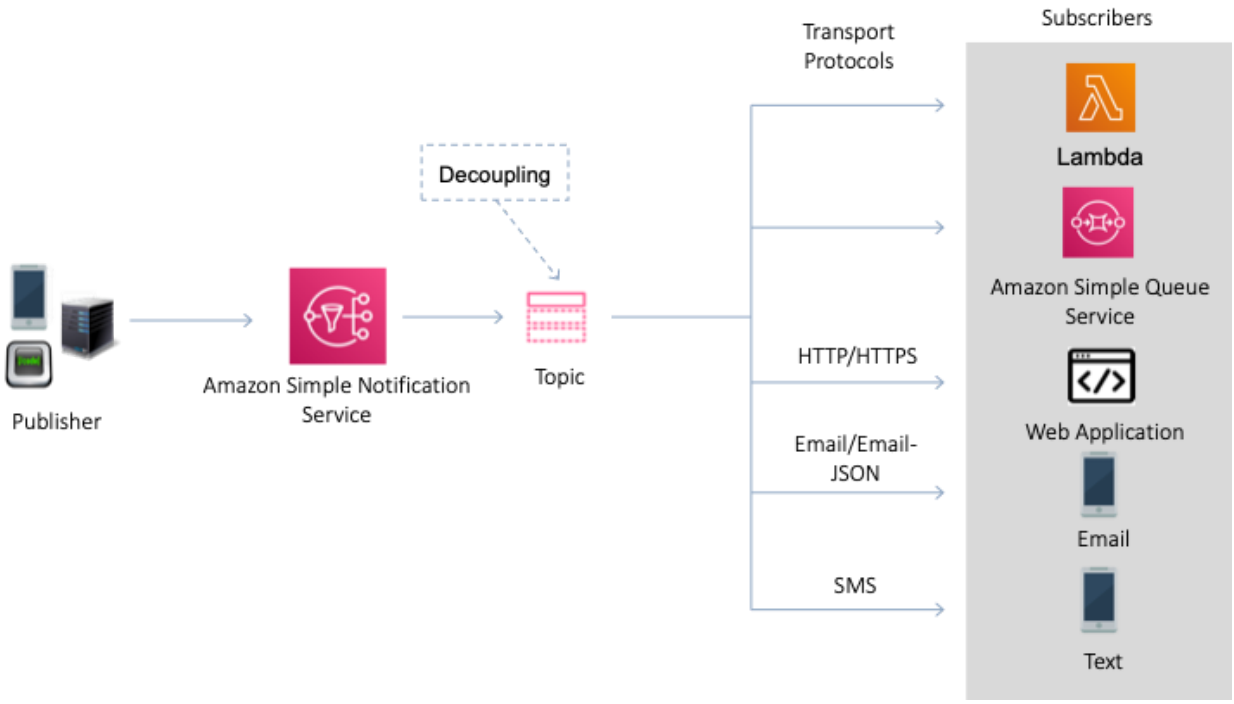


Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance's role, you can associate an IAM role with an ECS task definition or RunTask API operation. The applications in the task's containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

In this case each service requires access to different AWS services so following the principal of least privilege it is best to assign as a separate role to each task definition.

AWS APPLICATION INTEGRATION

1. **Amazon Simple Notification Service (Amazon SNS) is a web service that coordinates and manages the delivery or sending of messages to subscribing endpoints or clients. In Amazon SNS, there are two types of clients—publishers and subscribers—also referred to as producers and consumers.**



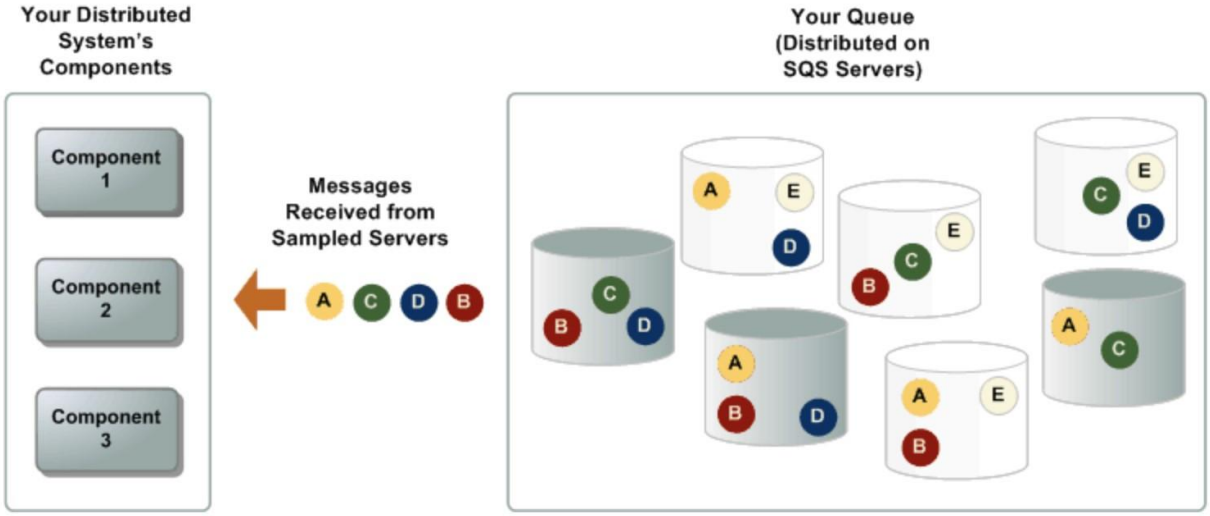
Publishers communicate asynchronously with subscribers by producing and sending a message to a topic, which is a logical access point and communication channel.

Subscribers (that is, web servers, email addresses, Amazon SQS queues, AWS Lambda functions) consume or receive the message or notification over one of the supported protocols (that is, Amazon SQS, HTTP/S, email, SMS, Lambda) when they are subscribed to the topic.

2. **The process of consuming messages from a queue depends on whether you use short or long polling. By default, Amazon SQS uses *short polling*, querying only a subset of its servers (based on a weighted random distribution) to determine whether any messages are available for a response. You can use *long polling* to reduce your costs while allowing your consumers to receive messages as soon as they arrive in the queue.**

   When you consume messages from a queue using short polling, Amazon SQS samples a subset of its servers (based on a weighted random distribution) and returns messages from only those servers. Thus, a particular ReceiveMessage request might not return all of your messages. However, if you have fewer than 1,000 messages in your queue, a subsequent request will return your messages. If you keep consuming from your queues, Amazon SQS samples all of its servers, and you receive all of your messages.

The following diagram shows the short-polling behavior of messages returned from a standard queue after one of your system components makes a receive request. Amazon SQS samples several of its servers (in gray) and returns messages A, C, D, and B from these servers. Message E isn't returned for this request but is returned for a subsequent request.



When the wait time for the `ReceiveMessage` API action is greater than 0, *long polling* is in effect. The maximum long polling wait time is 20 seconds. Long polling helps reduce the cost of using Amazon SQS by eliminating the number of empty responses (when there are no messages available for a `ReceiveMessage` request) and false empty responses (when messages are available but aren't included in a response).

Long polling occurs when the `WaitTimeSeconds` parameter of a `ReceiveMessage` request is set to a value greater than 0 in one of two ways:

The `ReceiveMessage` call sets `WaitTimeSeconds` to a value greater than 0.

The `ReceiveMessage` call doesn't set `WaitTimeSeconds`, but the queue attribute `ReceiveMessageWaitTimeSeconds` is set to a value greater than 0.

Therefore, the Developer should set the `ReceiveMessage` API with a `WaitTimeSeconds` of 20.

3. You can configure a dead letter queue (DLQ) on AWS Lambda to give you more control over message handling for all asynchronous invocations, including those delivered via AWS events (S3, SNS, IoT, etc.).

A dead-letter queue saves discarded events for further processing. A dead-letter queue acts the same as an on-failure destination in that it is used when an event fails all processing attempts or expires without being processed.

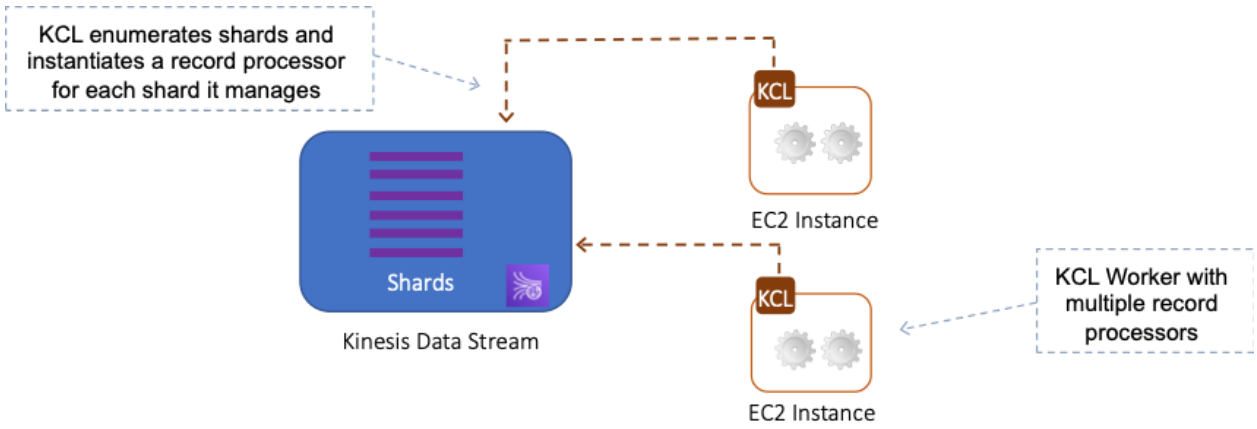
However, a dead-letter queue is part of a function's version-specific configuration, so it is locked in when you publish a version. On-failure destinations also support additional targets and include details about the function's response in the invocation record.

You can setup a DLQ by configuring the 'DeadLetterConfig' property when creating or updating your Lambda function. You can provide an SQS queue or an SNS topic as the 'TargetArn' for your DLQ, and AWS Lambda will write the event object invoking the Lambda function to this endpoint after the standard retry policy (2 additional retries on failure) is exhausted.

4. The `ReceiveMessage` API call retrieves one or more messages (up to 10), from the specified queue. This should be the first step to resolve the issue. With more messages received with each call the application should be able to process messages faster.

   If the application still fails to keep up with the messages, and speed is important (remember this is one of the reasons for using an SQS queue, to shield your processing layer for the front-end), then additional queues can be added to scale horizontally.

5. The Kinesis Client Library (KCL) helps you consume and process data from a Kinesis data stream. This type of application is also referred to as a *consumer*. The KCL takes care of many of the complex tasks associated with distributed computing, such as load balancing across multiple instances, responding to instance failures, checkpointing processed records, and reacting to resharding. The KCL enables you to focus on writing record-processing logic.



The KCL is different from the Kinesis Data Streams API that is available in the AWS SDKs. The Kinesis Data Streams API helps you manage many aspects of Kinesis Data Streams (including creating streams, resharding, and putting and getting records). The KCL provides a layer of abstraction specifically for processing data in a consumer role.

Therefore, the correct answer is to use the Kinesis Client Library.

6. The `DeleteMessageBatch` API deletes the specified message from the specified queue. To select the message to delete, use the `ReceiptHandle` of the message (*not* the `MessageId` which you receive when you send the message). Amazon SQS can delete a message from a queue even if a visibility timeout setting causes the message to be locked by another consumer. Amazon SQS automatically deletes messages left in a queue longer than the retention period configured for the queue.

   The `DeleteMessageBatch` API deletes up to ten messages from the specified queue. This is a batch version of `DeleteMessage`. The result of the action on each message is reported individually in the response.

   In this scenario the Developer has some specific messages that must be deleted. Therefore, we can assume the Developer is able to identify which messages need to be deleted by receipt handle. Using this information, the Developer can issue an API call with `DeleteMessageBatch` and specify up to 10 messages to delete per API call. This is the most efficient option available.
7. The message deduplication ID is the token used for deduplication of sent messages. If a message with a particular message deduplication ID is sent successfully, any messages sent with the same message deduplication ID are accepted successfully but aren't delivered during the 5-minute deduplication interval.

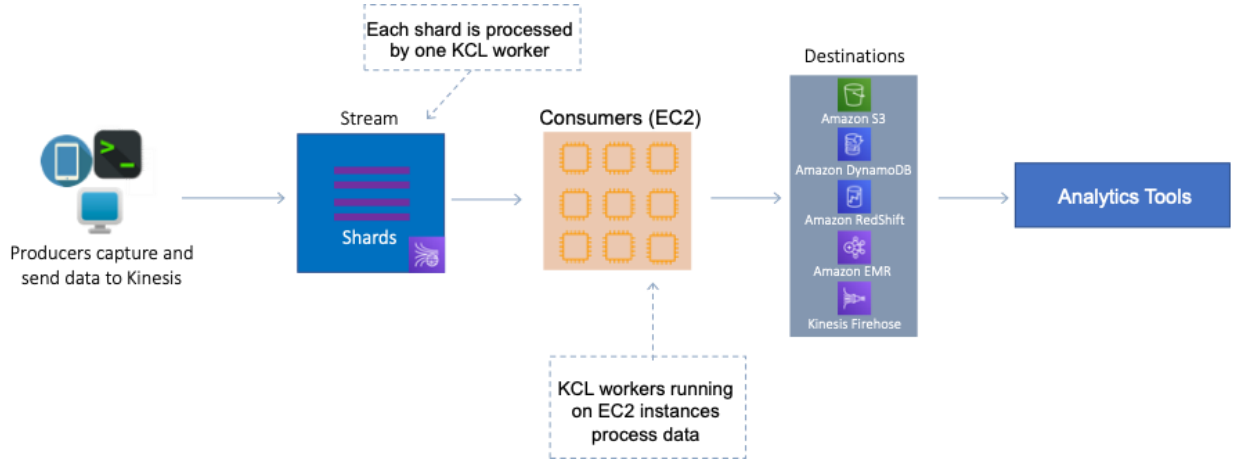
   The message deduplication ID is used with FIFO queues which also provide exactly-once processing (unlike standard queues which only provide “at least once delivery”). The producer should provide message deduplication ID values for each message.

   Therefore, the best answer is to use a FIFO queue and configure the producer to provide a message deduplication ID.

8. **AWS Step Functions makes it easy to coordinate the components of distributed applications as a series of steps in a visual workflow. You can quickly build and run state machines to execute the steps of your application in a reliable and scalable fashion.**

   AWS Step Functions is built on Lambda and is and orchestration service that lets you easily coordinate multiple Lambda functions into flexible workflows that are easy to debug and easy to change.

9. ***Resharding*** **enables you to increase or decrease the number of shards in a stream in order to adapt to changes in the rate of data flowing through the stream. Resharding is typically performed by an administrative application that monitors shard data-handling metrics. Although the KCL itself doesn't initiate resharding operations, it is designed to adapt to changes in the number of shards that result from resharding.**



   Typically, when you use the KCL, you should ensure that the number of instances does not exceed the number of shards (except for failure standby purposes). Each shard is processed by exactly one KCL worker and has exactly one corresponding record processor, so you never need multiple instances to process one shard. However, one worker can process any number of shards, so it's fine if the number of shards exceeds the number of instances.

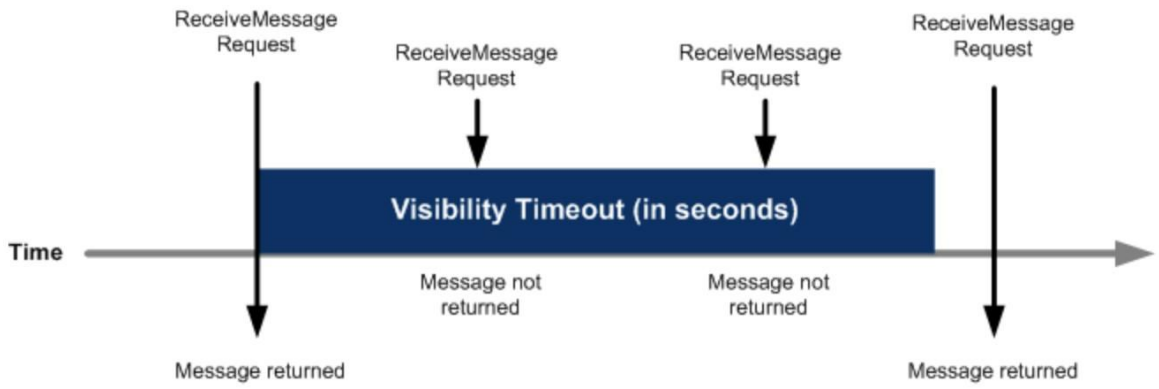
   To scale up processing in your application, you should test a combination of these approaches:

   Increasing the instance size (because all record processors run in parallel within a process)

Increasing the number of instances up to the maximum number of open shards (because shards can be processed independently)

Increasing the number of shards (which increases the level of parallelism)

10. **When a consumer receives and processes a message from a queue, the message remains in the queue. Amazon SQS doesn't automatically delete the message. Because Amazon SQS is a distributed system, there's no guarantee that the consumer actually receives the message (for example, due to a connectivity issue, or due to an issue in the consumer application). Thus, the consumer must delete the message from the queue after receiving and processing it.**

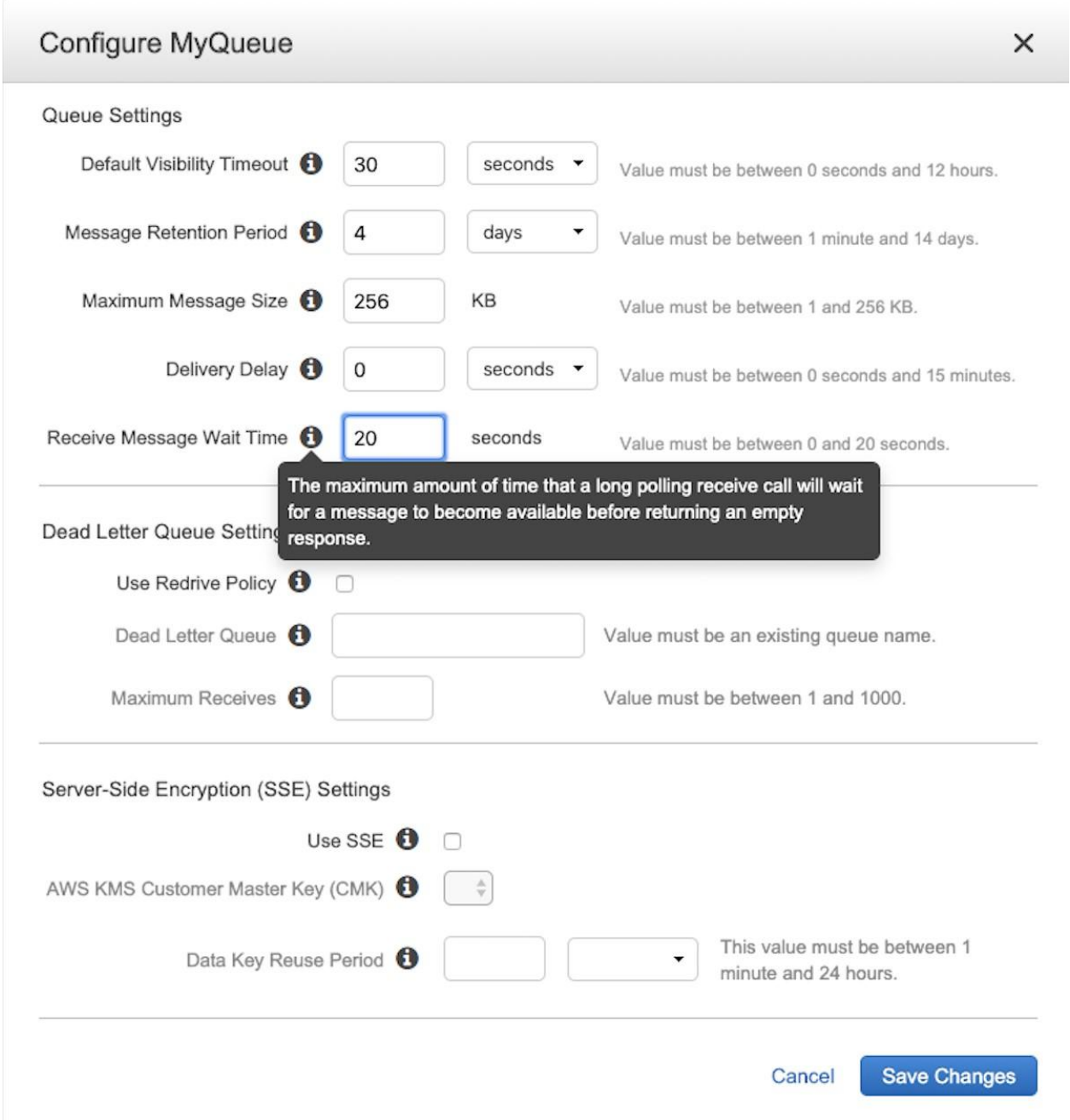


Immediately after a message is received, it remains in the queue. To prevent other consumers from processing the message again, Amazon SQS sets a *visibility timeout*, a period of time during which Amazon SQS prevents other consumers from receiving and processing the message. The default visibility timeout for a message is 30 seconds. The minimum is 0 seconds. The maximum is 12 hours.

For this scenario, the best way to minimize the chances of duplicate processing, the Developer should configure the visibility timeout to be at least 5 minutes to allow time for processing of the message. This will prevent the message from becoming visible while it is still being processed (which could result in another consumer processing it).

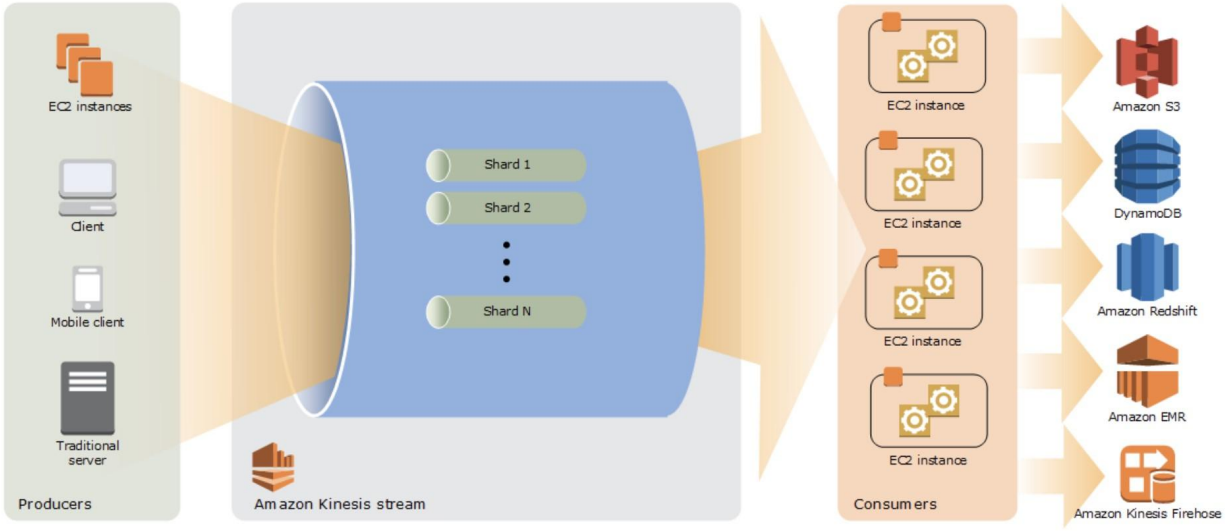
11. **The process of consuming messages from a queue depends on whether you use short or long polling. By default, Amazon SQS uses *short polling*, querying only a subset of its servers (based on a weighted random distribution) to determine whether any messages are available for a response. You can use *long polling* to reduce your costs while allowing your consumers to receive messages as soon as they arrive in the queue.**

When the wait time for the ReceiveMessage API action is greater than 0, *long polling* is in effect. The maximum long polling wait time is 20 seconds. Long polling helps reduce the cost of using Amazon SQS by eliminating the number of empty responses (when there are no messages available for a ReceiveMessage request) and false empty responses (when messages are available but aren't included in a response).



Therefore, setting the queue attribute ReceiveMessageWaitTimeSeconds to 20 will result in fewer attempts to poll the queue which is more efficient and will reduce cost.

12. The purpose of resharding in Amazon Kinesis Data Streams is to enable your stream to adapt to changes in the rate of data flow. You split shards to increase the capacity (and cost) of your stream. You merge shards to reduce the cost (and capacity) of your stream.



One approach to resharding could be to split every shard in the stream—which would double the stream's capacity. However, this might provide more additional capacity than you actually need and therefore create unnecessary cost.

You can also use metrics to determine which are your "hot" or "cold" shards, that is, shards that are receiving much more data, or much less data, than expected. You could then selectively split the hot shards to increase capacity for the hash keys that target those shards. Similarly, you could merge cold shards to make better use of their unused capacity.

You can obtain some performance data for your stream from the Amazon CloudWatch metrics that Kinesis Data Streams publishes. However, you can also collect some of your own metrics for your streams. One approach would be to log the hash key values generated by the partition keys for your data records.

For this scenario, the company has identified that there are shards which are underutilized, therefore the best approach to minimizing the cost of the Kinesis Data Stream is to merge the “cold” shards. This will reduce cost as you pay for each shard.

13. The `MessageDeduplicationId` is available in an Amazon SQS FIFO queue. With a FIFO queue, the order in which messages are sent and received is strictly

preserved and a message is delivered once and remains available until a consumer processes and deletes it.

Duplicates are not introduced into the queue. FIFO queues also support message groups that allow multiple ordered message groups within a single queue.

Deduplication with FIFO queues:

- Provide a `MessageDeduplicationId` with the message.

- The de-duplication interval is 5 minutes.

- Content based duplication – the `MessageDeduplicationId` is generated as the SHA-256 with the message body.

Sequencing with FIFO queues:

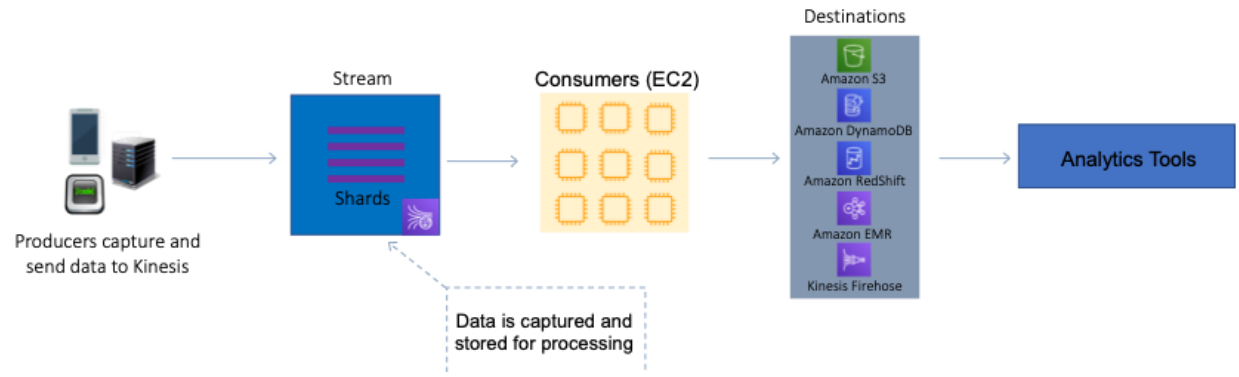
- To ensure strict ordering between messages, specify a `MessageGroupId`.

- Messages with a different Group ID may be received out of order.

- Messages with the same Group ID are delivered to one consumer at a time.

In this case, we need to ensure the messages from a specific producer are processed in order. Therefore, we need to configure each producer to send messages with a unique `MessageGroupId`.

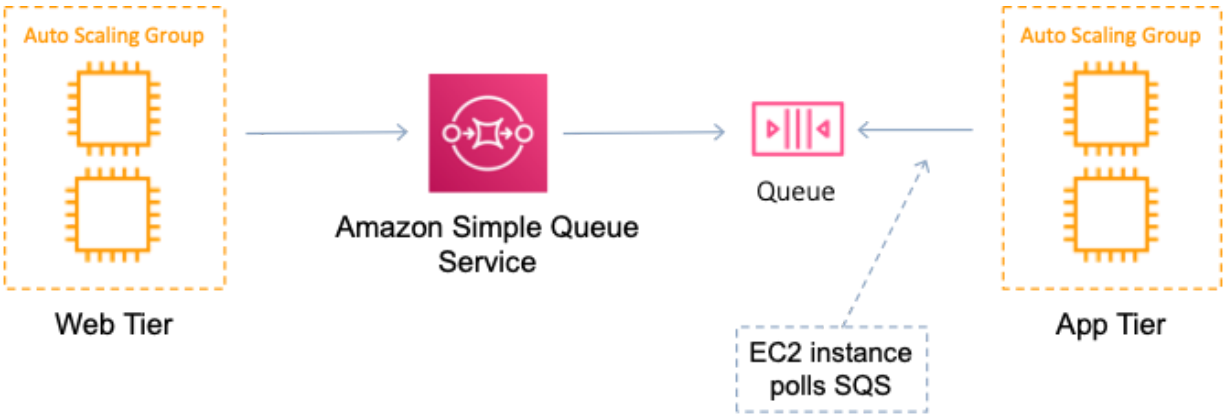
14. **The policy allows the user to use all Amazon SQS actions, but only with queues whose names are prefixed with the literal string "staging-queue". This policy is useful to provide a queue creator the ability to use Amazon SQS actions. Any user who has permissions to create a queue must also have permissions to use other Amazon SQS actions in order to do anything with the created queues.**
15. **You can use Amazon Kinesis Data Streams to collect and process large streams of data records in real time. You can create data-processing applications, known as *Kinesis Data Streams applications*. A typical Kinesis Data Streams application reads data from a *data stream* as data records.**



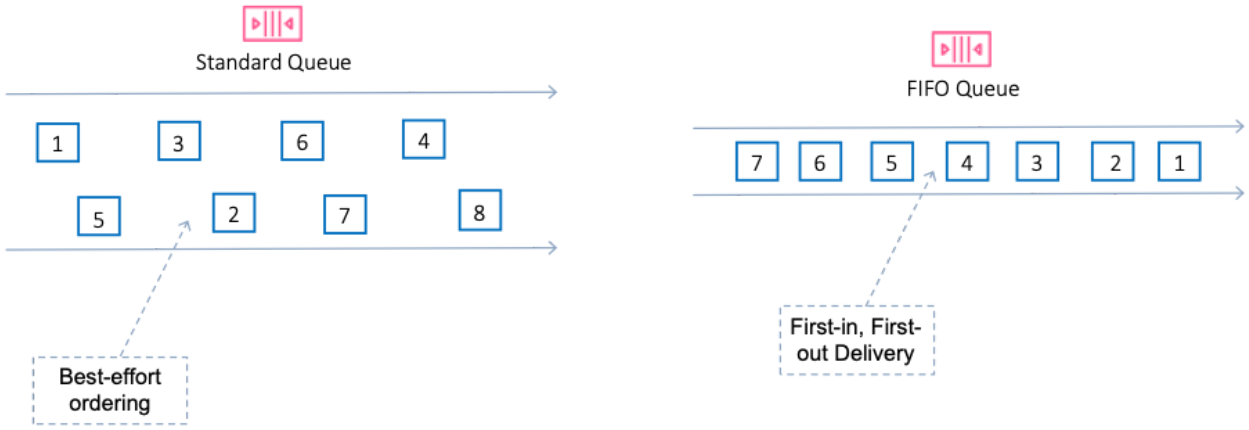
These applications can use the Kinesis Client Library, and they can run on Amazon EC2 instances. You can send the processed records to dashboards, use them to generate alerts, dynamically change pricing and advertising strategies, or send data to a variety of other AWS services.

This scenario is an ideal use case for Kinesis Data Streams as large volumes of real time streaming data are being ingested. Therefore, the best approach is to use Amazon Kinesis Data Streams with Kinesis Client Library to ingest and deliver messages

16. Amazon Simple Queue Service (Amazon SQS) offers a secure, durable, and available hosted queue that lets you integrate and decouple distributed software systems and components. Amazon SQS can process each buffered request independently, scaling transparently to handle any load increases or spikes without any provisioning instructions.



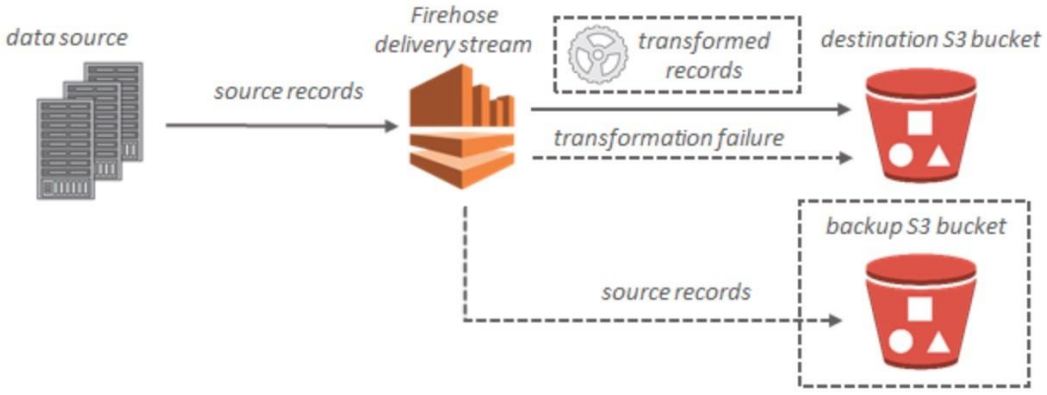
There are two different types of queue: standard queues and FIFO queues. Standard queues offer best-effort ordering whereas first-in-first-out (FIFO) queues offer strict preservation of the message order.



Therefore, Amazon SQS with a FIFO queue is a perfect solution for this scenario as it will provide a decoupling service that scales dynamically and provides strict preservation of the message order.

17. **Amazon Kinesis Data Firehose is the easiest way to reliably load streaming data into data lakes, data stores and analytics tools. It can capture, transform, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk, enabling near real-time analytics with existing business intelligence tools and dashboards you're already using today.**

    You can configure Amazon Kinesis Data Firehose to prepare your streaming data before it is loaded to data stores. Simply select an AWS Lambda function from the Amazon Kinesis Data Firehose delivery stream configuration tab in the AWS Management console. Amazon Kinesis Data Firehose will automatically apply that function to every input data record and load the transformed data to destinations.



Therefore, the best solution for these requirements is to use Amazon Kinesis Firehose with AWS Lambda for the transformation/processing of data. Kinesis Firehose can then load the data to Amazon S3.

18. **Amazon Simple Queue Service (SQS) is a fully managed message queuing service that enables you to decouple and scale microservices, distributed systems, and serverless applications.**

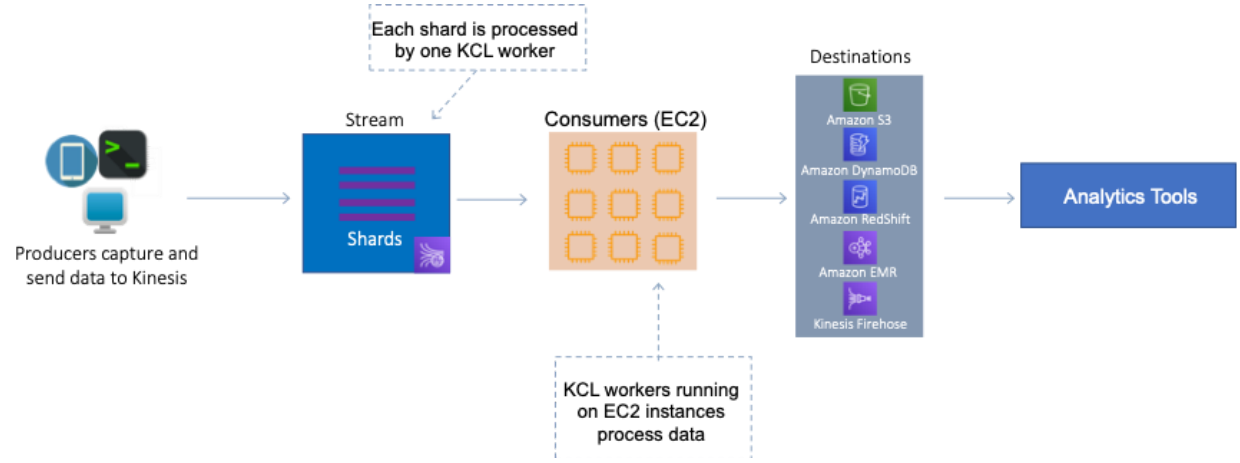
    Amazon Simple Notification Service (SNS) is a highly available, durable, secure, fully managed pub/sub messaging service that enables you to decouple microservices, distributed systems, and serverless applications.

    These services both enable asynchronous message passing in the form of a message bus (SQS) and notifications (SNS)
19. **Amazon Kinesis Data Streams (KDS) is a massively scalable and durable real-time data streaming service. KDS can continuously capture gigabytes of data per second from hundreds of thousands of sources such as website clickstreams, database event streams, financial transactions, social media feeds, IT logs, and location-tracking events.**

    Server-side encryption is a feature in Amazon Kinesis Data Streams that automatically encrypts data before it's at rest by using an AWS KMS customer master key (CMK) you specify. Data is encrypted before it's written to the Kinesis stream storage layer and decrypted after it's retrieved from storage. As a result, your data is encrypted at rest within the Kinesis Data Streams service. This allows you to meet strict regulatory requirements and enhance the security of your data.

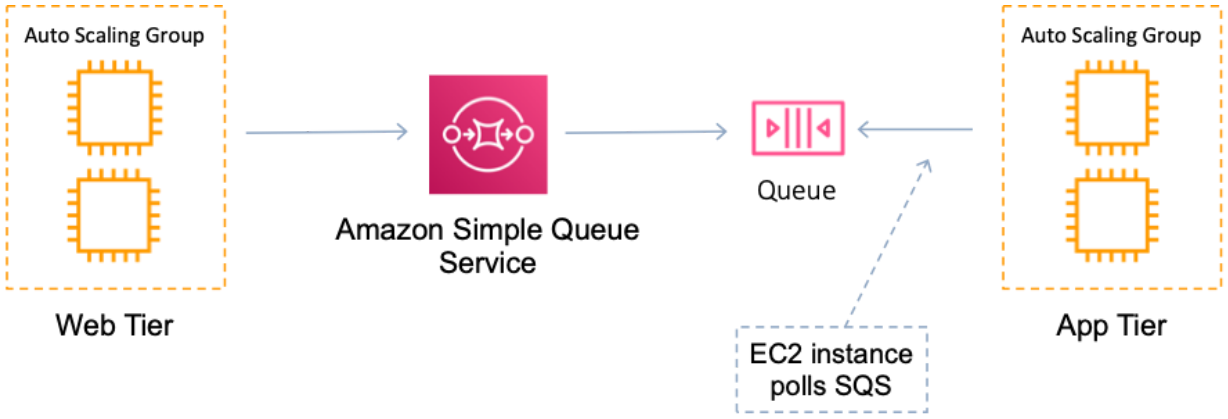
    With server-side encryption, your Kinesis stream producers and consumers don't need to manage master keys or cryptographic operations. Your data is automatically encrypted as it enters and leaves the Kinesis Data Streams service, so your data at rest is encrypted. AWS KMS provides all the master keys that are used by the server-side encryption feature. AWS KMS makes it easy to use a CMK for Kinesis that is managed by AWS, a user-specified AWS KMS CMK, or a master key imported into the AWS KMS service.

    Therefore, in this scenario the Developer can enable server-side encryption on Kinesis Data Streams with an AWS KMS CMK
20. **By increasing the instance size and number of shards in the Kinesis stream, the developer can allow the instances to handle more record processors, which are running in parallel within the instance. It also allows the stream to properly accommodate the rate of data being sent in. The data capacity of your stream is a function of the number of shards that you specify for the stream. The total capacity of the stream is the sum of the capacities of its shards.**



Therefore, the best answer is to increase both the EC2 instance size and add shards to the stream.

21. **Amazon SQS queues messages received from one application component ready for consumption by another component. A queue is a temporary repository for messages that are awaiting processing. The queue acts as a buffer between the component producing and saving data, and the component receiving the data for processing.**



With this scenario the best choice for the Developer is to implement an Amazon SQS queue between the web tier and the application tier. This will mean when the web tier receives bursts of traffic the messages will not overburden the application tier. Instead, they will be placed in the queue and can be processed by the app tier.

22. **The UpdateShardCount API action updates the shard count of the specified stream to the specified number of shards.**

    Updating the shard count is an asynchronous operation. Upon receiving the request, Kinesis Data Streams returns immediately and sets the status of the

stream to UPDATING. After the update is complete, Kinesis Data Streams sets the status of the stream back to ACTIVE.

Depending on the size of the stream, the scaling action could take a few minutes to complete. You can continue to read and write data to your stream while its status is UPDATING.

To update the shard count, Kinesis Data Streams performs splits or merges on individual shards. This can cause short-lived shards to be created, in addition to the final shards. These short-lived shards count towards your total shard limit for your account in the Region.

When using this operation, we recommend that you specify a target shard count that is a multiple of 25% (25%, 50%, 75%, 100%). You can specify any target value within your shard limit. However, if you specify a target that isn't a multiple of 25%, the scaling action might take longer to complete.

This operation has the following default limits. By default, you cannot do the following:

• Scale more than ten times per rolling 24-hour period per stream

• Scale up to more than double your current shard count for a stream

• Scale down below half your current shard count for a stream

• Scale up to more than 500 shards in a stream

• Scale a stream with more than 500 shards down unless the result is less than 500 shards

Scale up to more than the shard limit for your account

Note that the question specifically states that the Kinesis data stream cannot keep up with incoming data. This indicates that the producers are attempting to add records to the stream but there are not enough shards to keep up with demand. Therefore, we need to add additional shards and can do this using the UpdateShardCount API action.

23. **AWS Step Functions lets you coordinate multiple AWS services into serverless workflows so you can build and update apps quickly. Using Step Functions, you can design and run workflows that stitch together services, such as AWS Lambda, AWS Fargate, and Amazon SageMaker, into feature-rich applications.**

Workflows are made up of a series of steps, with the output of one step acting as input into the next. Application development is simpler and more intuitive using Step Functions, because it translates your workflow into a state machine diagram that is easy to understand, easy to explain to others, and easy to change.

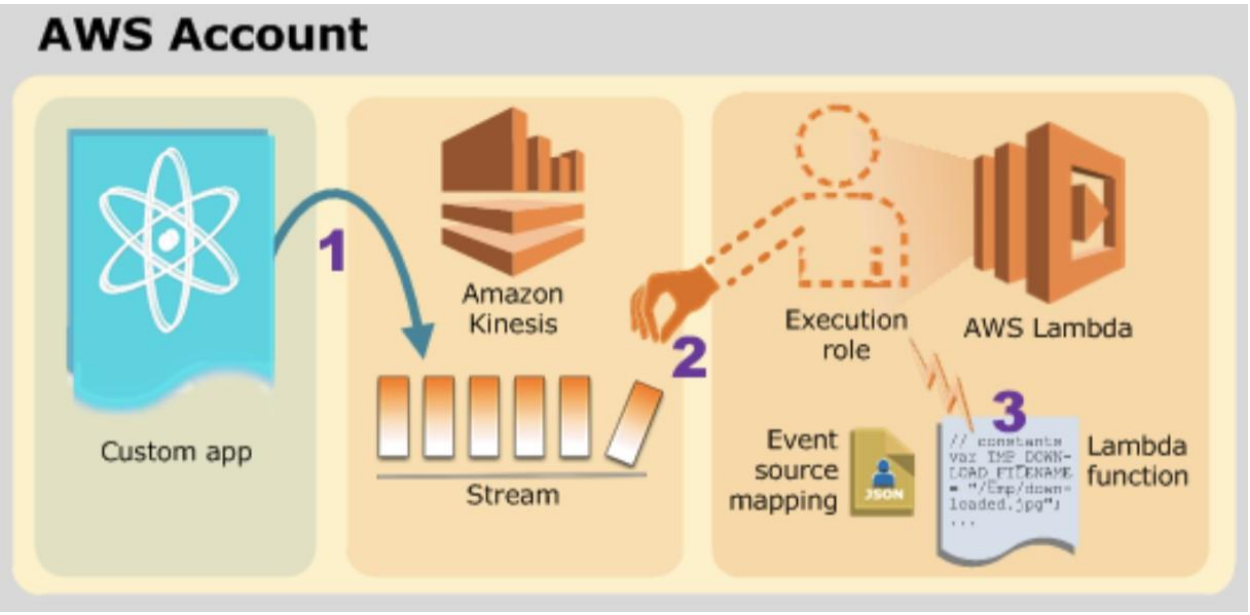
Step Functions automatically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected. With Step Functions, you can craft long-running workflows such as machine learning model training, report generation, and IT automation.

Therefore, AWS Step Functions is the best AWS service to use when refactoring the application away from the legacy code.

24. **Amazon Kinesis Data Streams (KDS) is a massively scalable and durable real-time data streaming service. KDS can continuously capture gigabytes of data per second from hundreds of thousands of sources such as website clickstreams, database event streams, financial transactions, social media feeds, IT logs, and location-tracking events.**

KDS receives data from producers, and the data is stored in shards. Consumers then take the data and process it. In this case the AWS Lambda function is consuming the records from the shards.

In this scenario an application will be producing records and placing them in the stream as in step 1 of the image below. The AWS Lambda function will then consume the records (step 2) and will then execute the function by assuming the execution role specified (step 3).



A shard is an append-only log and a unit of streaming capability. A shard contains an ordered sequence of records ordered by arrival time. The order is guaranteed within a shard but not across shards.

Therefore, the best answer to this question is that AWS Lambda will receive each record in the exact order it was placed into the shard but there is no guarantee of order across shards

25. Amazon Kinesis Data Streams supports changes to the data record retention period of your stream. A Kinesis data stream is an ordered sequence of data records meant to be written to and read from in real time. Data records are therefore stored in shards in your stream temporarily. The time period from when a record is added to when it is no longer accessible is called the *retention period*. A Kinesis data stream stores records from 24 hours by default, up to 168 hours.

    You can increase the retention period up to 168 hours using the IncreaseStreamRetentionPeriod operation. You can decrease the retention period down to a minimum of 24 hours using the DecreaseStreamRetentionPeriod operation. The request syntax for both operations includes the stream name and the retention period in hours. Finally, you can check the current retention period of a stream by calling the DescribeStream operation.

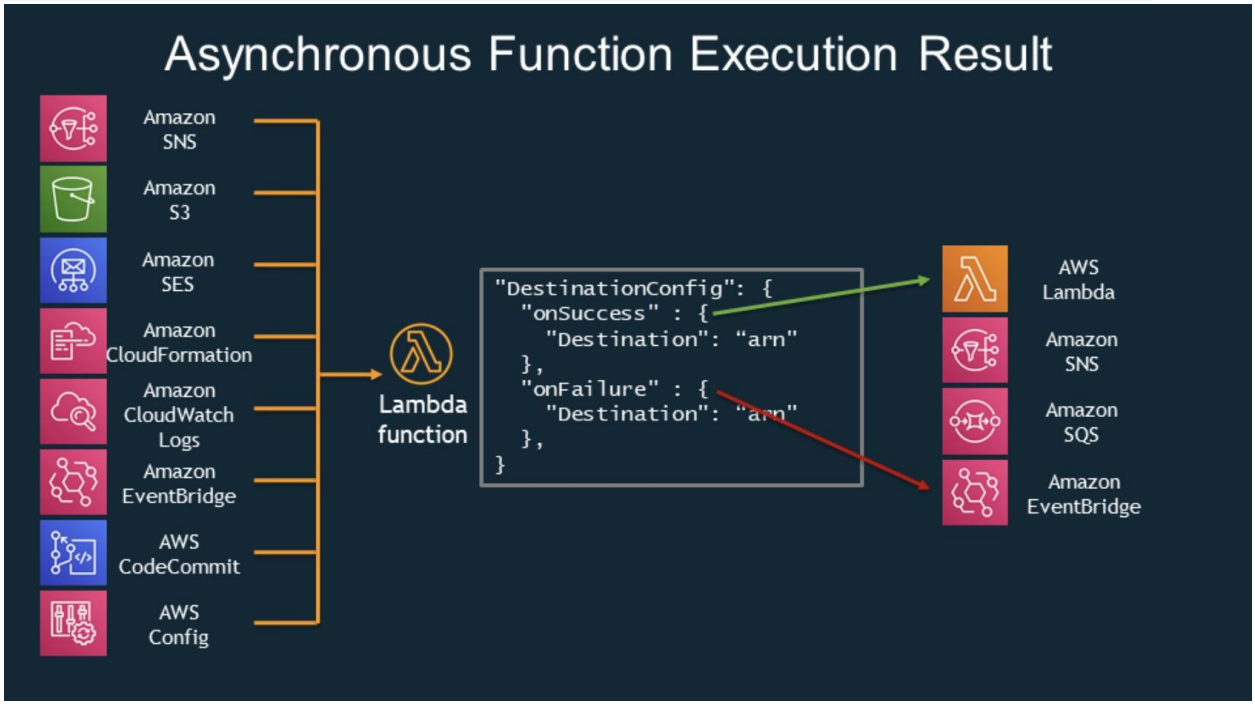
Both operations are easy to use. The following is an example of changing the retention period using the AWS CLI:

```
aws kinesis increase-stream-retention-period --stream-name retentionPeriodDemo --retention-period-hours 72
```

Therefore, the most likely explanation is that the message retention period is set at the 24-hour default.

26. With Destinations, you can route asynchronous function results as an execution record to a destination resource without writing additional code. An execution record contains details about the request and response in JSON format including version, timestamp, request context, request payload, response context, and response payload.

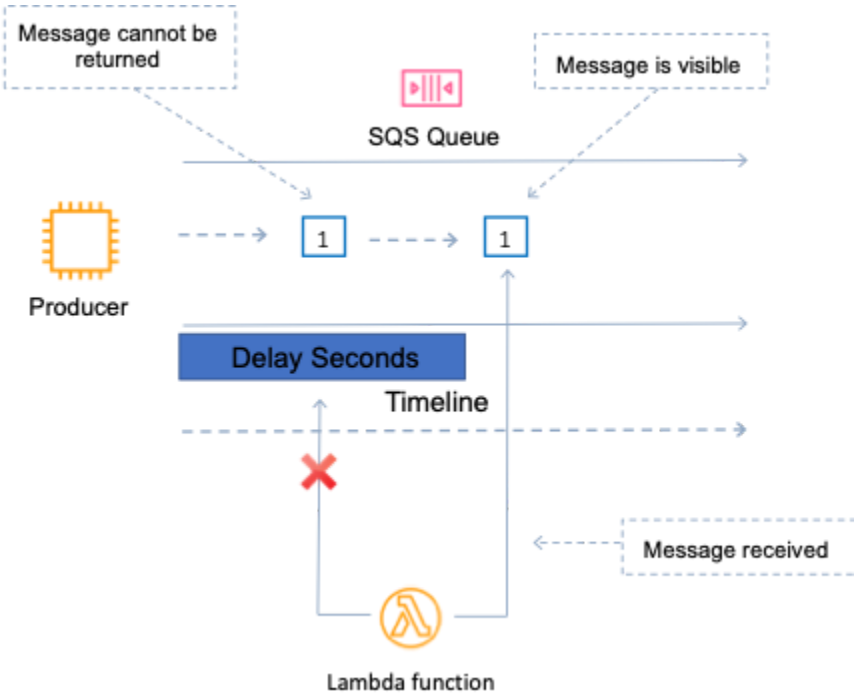
For each execution status such as *Success* or *Failure* you can choose one of four destinations: another Lambda function, SNS, SQS, or EventBridge. Lambda can also be configured to route different execution results to different destinations.



In this scenario the Developer can publish the processed data to an Amazon SNS topic by using an Amazon SNS destination.

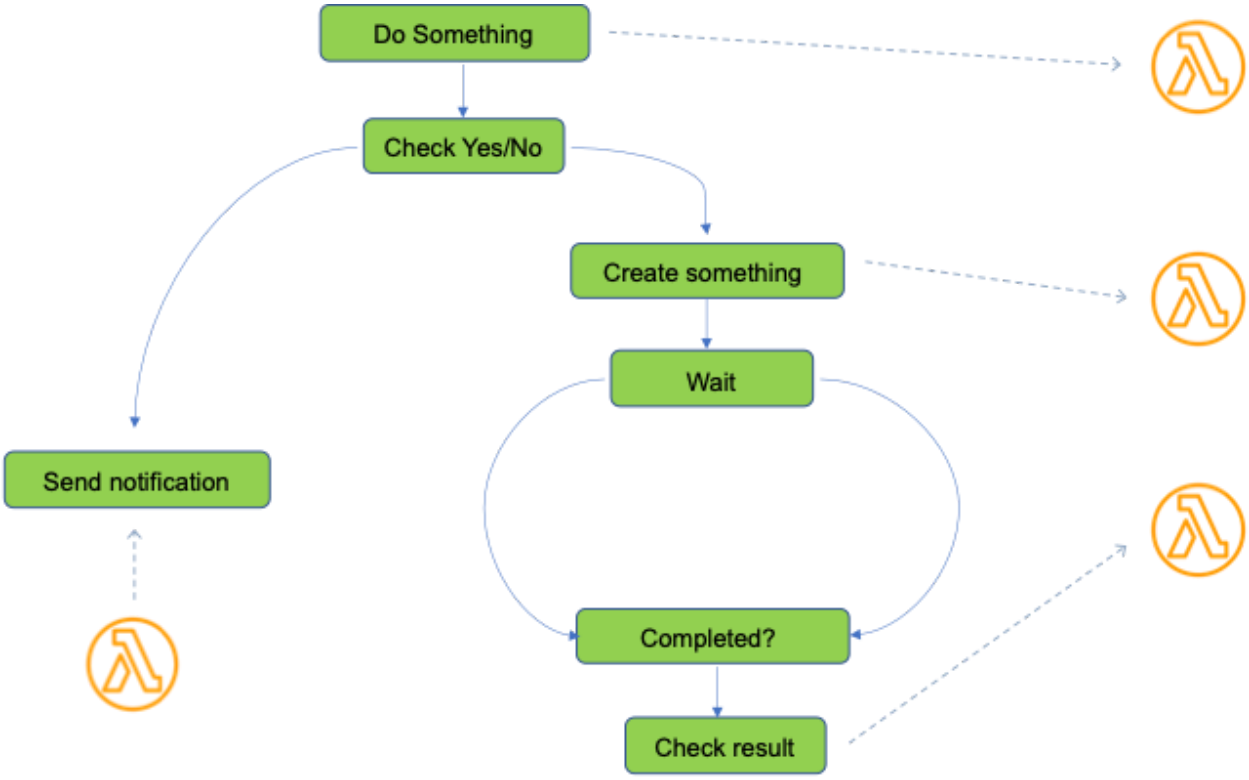
27. Delay queues let you postpone the delivery of new messages to a queue for a number of seconds, for example, when your consumer application needs additional time to process messages.

If you create a delay queue, any messages that you send to the queue remain invisible to consumers for the duration of the delay period. The default (minimum) delay for a queue is 0 seconds. The maximum is 15 minutes.



Therefore, the correct explanation is that with an Amazon SQS Delay Queue messages are hidden for a configurable amount of time when they are first added to the queue

28. **AWS Step Functions is a web service that enables you to coordinate the components of distributed applications and microservices using visual workflows. You build applications from individual components that each perform a discrete function, or *task*, allowing you to scale and change applications quickly.**



Step Functions provides a reliable way to coordinate components and step through the functions of your application. Step Functions offers a graphical console to visualize the components of your application as a series of steps. It auto matically triggers and tracks each step, and retries when there are errors, so your application executes in order and as expected, every time. Step Functions logs the state of each step, so when things go wrong, you can diagnose and debug problems quickly.

29. An Amazon Kinesis Data Streams producer is an application that puts user data records into a Kinesis data stream (also called *data ingestion*). The Kinesis Producer Library (KPL) simplifies producer application development, allowing developers to achieve high write throughput to a Kinesis data stream.

    The KPL is an easy-to-use, highly configurable library that helps you write to a Kinesis data stream. It acts as an intermediary between your producer application code and the Kinesis Data Streams API actions. The KPL performs the following primary tasks:

    • Writes to one or more Kinesis data streams with an automatic and configurable retry mechanism

    • Collects records and uses `PutRecords` to write multiple records to multiple shards per request

• Aggregates user records to increase payload size and improve throughput

• Integrates seamlessly with the Kinesis Client Library (KCL) to de-aggregate batched records on the consumer

• Submits Amazon CloudWatch metrics on your behalf to provide visibility into producer performance

The question states that the producers are not making best use of the available shards. Therefore, we understand that there are adequate shards available but the producers are either not discovering them or are not writing records at sufficient speed to best utilize the shards.

We therefore need to install the Kinesis Producer Library (KPL) for ingesting data into the stream.

30. **Amazon Kinesis Data Firehose is the easiest way to reliably load streaming data into data lakes, data stores and analytics tools. It can capture, transform, and load streaming data into Amazon S3, Amazon Redshift, Amazon Elasticsearch Service, and Splunk, enabling near real-time analytics with existing business intelligence tools and dashboards.**

    A destination is the data store where your data will be delivered. Firehose Destinations include:

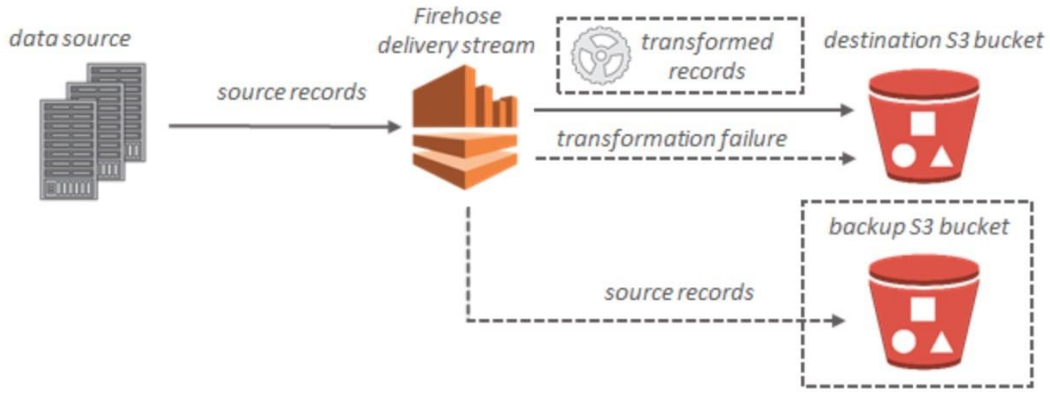
    • Amazon S3.

    • Amazon Redshift.

    • Amazon Elasticsearch Service.

    Splunk.

    For Amazon S3 destinations, streaming data is delivered to your S3 bucket. If data transformation is enabled, you can optionally back up source data to another Amazon S3 bucket:



The best choice of AWS service for this scenario is to use Amazon Kinesis Data Firehose as it can ingest large amounts of data at extremely high throughput and load that data into an Amazon S3 bucket

31. **Firehose is a fully managed service that automatically scales to match the throughput of your data and requires no ongoing administration. It can also batch, compress, and encrypt the data before loading it, minimizing the amount of storage used at the destination and increasing security.**
32. **With Amazon SNS, you have the ability to send push notification messages directly to apps on mobile devices. Push notification messages sent to a mobile endpoint can appear in the mobile app as message alerts, badge updates, or even sound alerts.**

    You send push notification messages to both mobile devices and desktops using one of the following supported push notification services:

    • Amazon Device Messaging (ADM)

    • Apple Push Notification Service (APNs) for both iOS and Mac OS X

    • Baidu Cloud Push (Baidu)

    • Firebase Cloud Messaging (FCM)

    • Microsoft Push Notification Service for Windows Phone (MPNS)

    • Windows Push Notification Services (WNS)

    To send a notification to an Amazon SNS subscriber, the application needs to send the notification to an Amazon SNS Topic. Amazon SNS will then send the notification to the relevant subscribers.

33. ***FIFO (First-In-First-Out)* queues are designed to enhance messaging between applications when the order of operations and events is critical, or where duplicates can't be tolerated.**

In FIFO queues, messages are ordered based on message group ID. If multiple hosts (or different threads on the same host) send messages with the same message group ID to a FIFO queue, Amazon SQS stores the messages in the order in which they arrive for processing. To ensure that Amazon SQS preserves the order in which messages are sent and received, ensure that each producer uses a unique message group ID to send all its messages.

FIFO queue logic applies only per message group ID. Each message group ID represents a distinct ordered message group within an Amazon SQS queue. For each message group ID, all messages are sent and received in strict order. However, messages with different message group ID values might be sent and received out of order. You must associate a message group ID with a message. If you don't provide a message group ID, the action fails. If you require a single group of ordered messages, provide the same message group ID for messages sent to the FIFO queue.

Therefore, the Developer can use a FIFO queue and configure the Lambda function to add a message deduplication token to the message body. This will ensure that the messages are deduplicated before being picked up for processing by the Amazon EC2 instance.

AWS NETWORKING AND CONTENT DELIVERY

1. If you need to remove a file from CloudFront edge caches before it expires, you can do one of the following:

   Invalidate the file from edge caches. The next time a viewer requests the file, CloudFront returns to the origin to fetch the latest version of the file.

   Use file versioning to serve a different version of the file that has a different name. For more information, see Updating Existing Files Using Versioned File Names.

In this case, the best option available is to invalidate all the application objects from the edge caches. This will result in the new objects being cached next time a request is made for them.

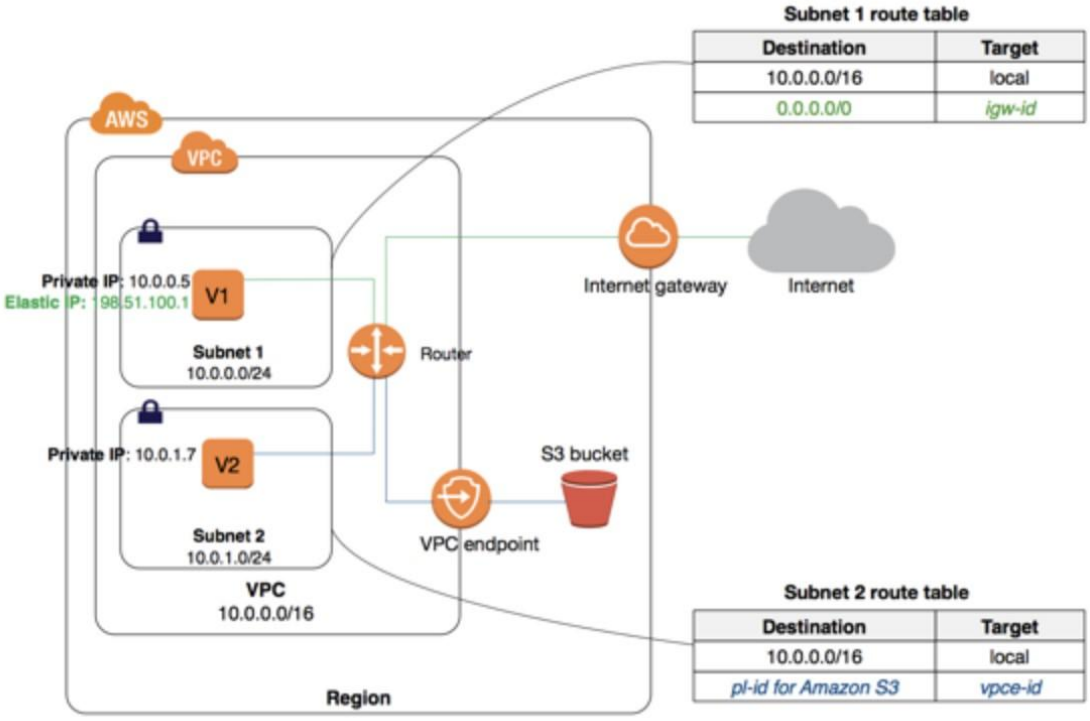
2. If your application is hosted in multiple AWS Regions, you can improve performance for your users by serving their requests from the AWS Region that provides the lowest latency.

   To use latency-based routing, you create latency records for your resources in multiple AWS Regions. When Route 53 receives a DNS query for your domain or subdomain (example.com or acme.example.com), it determines which AWS Regions you've created latency records for, determines which region gives the user the lowest latency, and then selects a latency record for that region. Route 53 responds with the value from the selected record, such as the IP address for a web server.
3. To determine the number of client-side errors captured in a given period the Developer should look at the 4XXError metric. To determine the number of server-side errors captured in a given period the Developer should look at the 5XXError.
4. Please note that the question specifically asks how to enable connectivity so this is not about permissions. When using a private subnet with no Internet connectivity there are only two options available for connecting to Amazon S3 (which remember, is a service with a public endpoint, it's not in your VPC).

   The first option is to enable Internet connectivity through either a NAT Gateway or a NAT Instance. However, there is no answer offering either of these as a solution. The other option is to enable a VPC endpoint for S3.

   The specific type of VPC endpoint to S3 is a Gateway Endpoint. EC2 instances running in private subnets of a VPC can use the endpoint to enable controlled access to S3 buckets, objects, and API functions that are in the same region as the VPC. You can then use an S3 bucket policy to indicate which VPCs and which VPC Endpoints have access to your S3 buckets.

   In the following diagram, instances in subnet 2 can access Amazon S3 through the gateway endpoint.



Therefore, the only answer that presents a solution to this challenge is to provision an VPC endpoint for S3.

5. ou can enable API caching in Amazon API Gateway to cache your endpoint's responses. With caching, you can reduce the number of calls made to your endpoint and also improve the latency of requests to your API.

   When you enable caching for a stage, API Gateway caches responses from your endpoint for a specified time-to-live (TTL) period, in seconds. API Gateway then responds to the request by looking up the endpoint response from the cache instead of making a request to your endpoint. The default TTL value for API caching is 300 seconds. The maximum TTL value is 3600 seconds. TTL=0 means caching is disabled.

   A client of your API can invalidate an existing cache entry and reload it from the integration endpoint for individual requests. The client must send a request that contains the `Cache-Control: max-age=0` header.

   The client receives the response directly from the integration endpoint instead of the cache, provided that the client is authorized to do so. This replaces the existing cache entry with the new response, which is fetched from the integration endpoint.

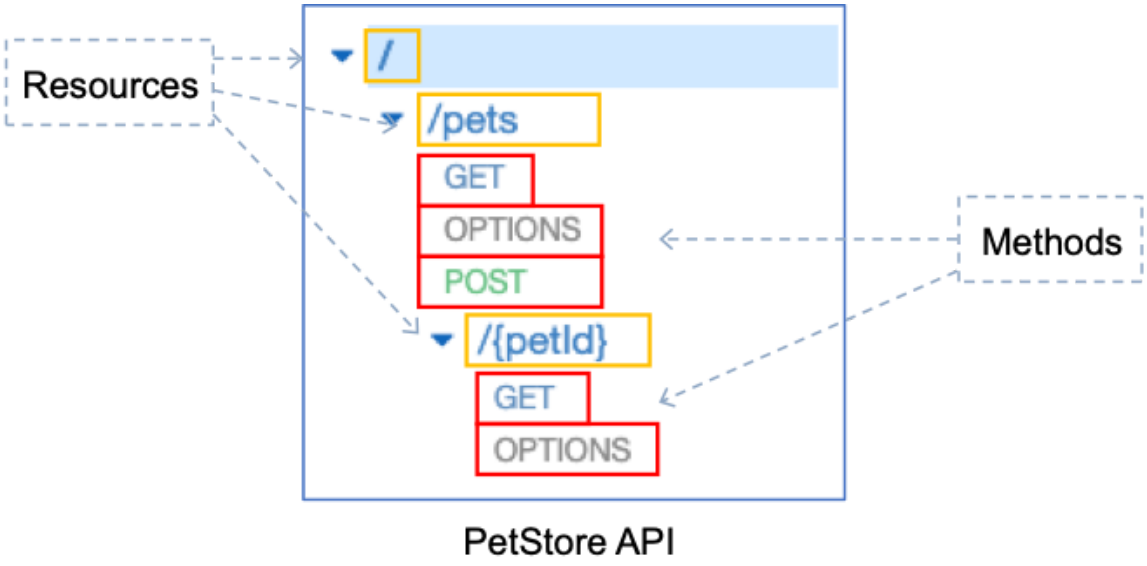
To grant permission for a client, attach a policy of the following format to an IAM execution role for the user.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "execute-api:InvalidateCache"
      ],
      "Resource": [
        "arn:aws:execute-api:region:account-id:api-id/stage-name/GET/resource-path-specifier"
      ]
    }
  ]
}
```

This policy allows the API Gateway execution service to invalidate the cache for requests on the specified resource (or resources).

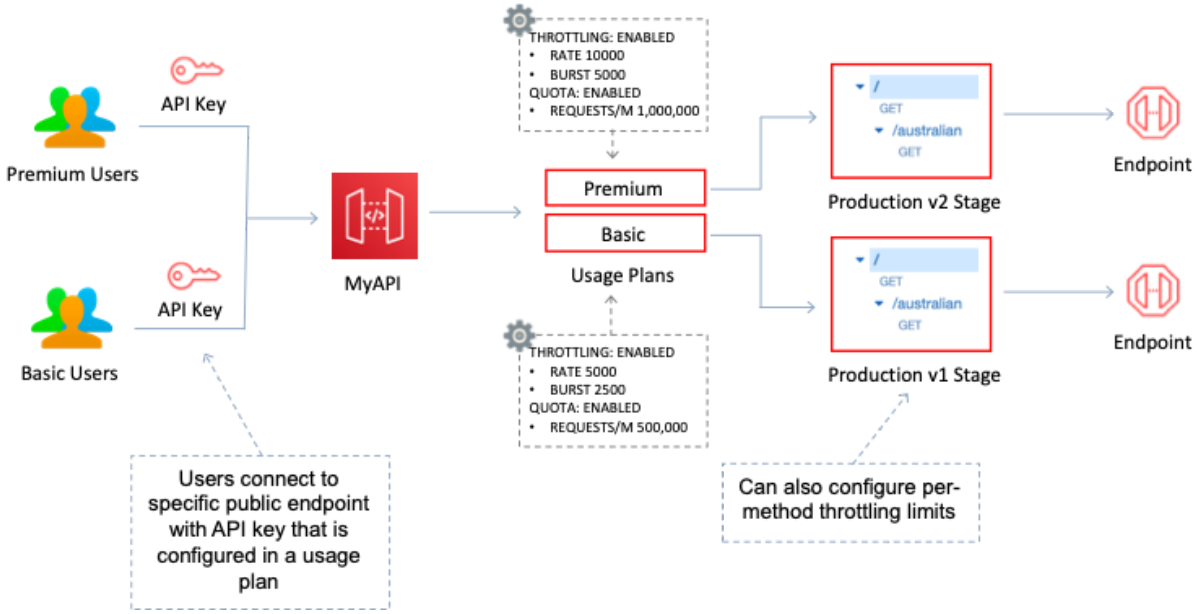
Therefore, as described above the solution is to get the partners to pass the HTTP header `Cache-Control: max-age=0`.

6. An API Gateway REST API is a collection of HTTP resources and methods that are integrated with backend HTTP endpoints, Lambda functions, or other AWS services. You can deploy this collection in one or more stages. Typically, API resources are organized in a resource tree according to the application logic. Each API resource can expose one or more API methods that have unique HTTP verbs supported by API Gateway.



As you can see from the image above, the Developer would need to create a resource which in this case would be /products. The Developer would then create a GET method within the resource.

7. A *usage plan* specifies who can access one or more deployed API stages and methods—and also how much and how fast they can access them. The plan uses API keys to identify API clients and meters access to the associated API stages for each key. It also lets you configure throttling limits and quota limits that are enforced on individual client API keys.



API keys are alphanumeric string values that you distribute to application developer customers to grant access to your API. You can use API keys together with usage plans or Lambda authorizers to control access to your APIs. API Gateway can generate API keys on your behalf, or you can import them from a CSV file. You can generate an API key in API Gateway, or import it into API Gateway from an external source.

To associate the newly created key with a usage plan the CreatUsagePlanKey API can be called. This creates a usage plan key for adding an existing API key to a usage plan.

8. Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services as well as data stored in the AWS Cloud.

   A stage is a named reference to a deployment, which is a snapshot of the API. You use a Stage to manage and optimize a particular deployment. For example, you can set up stage settings to enable caching, customize request throttling, configure logging, define stage variables or attach a canary release for testing.

   You deploy your API to a stage and it is given a unique URL that contains the stage name. This URL can be used to direct customers to your URL based on the stage (or version) you'd like them to use.

   The following invocation URLs can be used to direct customers to version 1 or version 2 of an API:

   Therefore, the best approach is to use API Gateway to deploy a new stage named v2 to the API and provide users with its URL. (for the given question)

9. The Latency metric measures the time between when API Gateway receives a request from a client and when it returns a response to the client. The latency includes the integration latency and other API Gateway overhead.

10. Standard API Gateway parameter and response code mapping templates allow you to map parameters one-to-one and map a family of integration response status codes (matched by a regular expression) to a single response status code.

    Mapping template overrides provides you with the flexibility to perform many-to-one parameter mappings; override parameters after standard API Gateway mappings have been applied; conditionally map parameters based on body content or other parameter values; programmatically create new parameters on the fly; and override status codes returned by your integration endpoint.

    Any type of request parameter, response header, or response status code may be overridden.

    Following are example uses for a mapping template override:

    To create a new header (or overwrite an existing header) as a concatenation of two parameters

    To override the response code to a success or failure code based on the contents of the body

    To conditionally remap a parameter based on its contents or the contents of some other parameter

    To iterate over the contents of a json body and remap key value pairs to headers or query strings

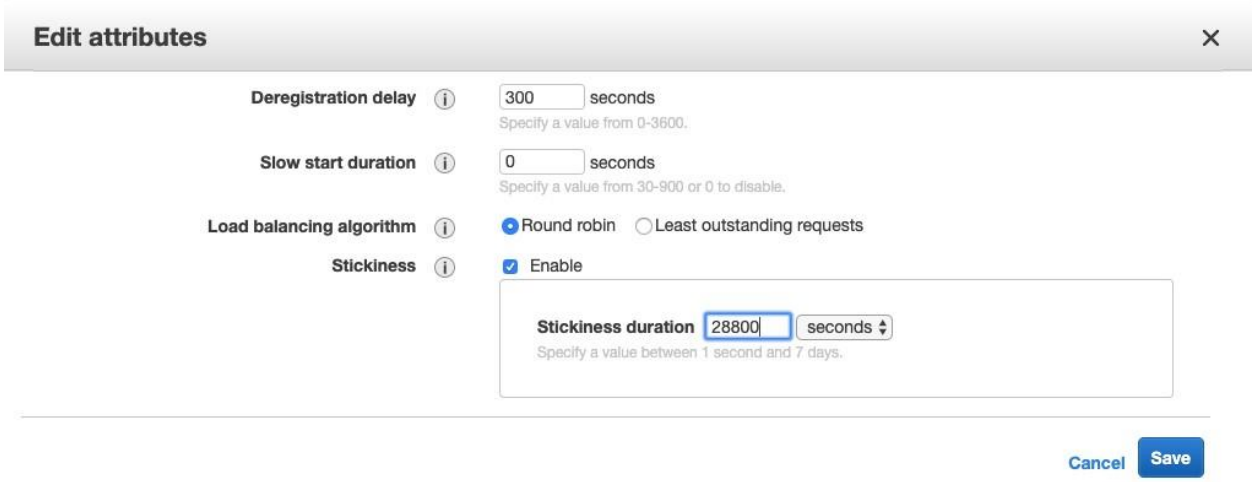
    Therefore, the Developer can convert the query string parameters by creating a mapping template.

11. CloudFront is a web service that gives businesses and web application developers an easy and cost-effective way to distribute content with low latency and high data transfer speeds. CloudFront is a good choice for distribution of frequently accessed static content that benefits from edge delivery—like popular website images, videos, media files or software downloads.

12. Sticky sessions are a mechanism to route requests to the same target in a target group. This is useful for servers that maintain state information in order to provide a continuous experience to clients. To use sticky sessions, the clients must support cookies.

In this case, it is likely that the clients authenticate to the back-end instance and when they are reconnecting without sticky sessions enabled they may be load balanced to a different instance and need to authenticate again.

The most obvious first step in troubleshooting this issue is to enable sticky sessions on the target group. The following image shows this setting:



13.
14. To enable connectivity to an application in a private subnet and the Internet you must first allow the function to connect to the private subnet (which has already been done).

    Lambda needs the following VPC configuration information so that it can connect to the VPC:

    · Private subnet ID.

    · Security Group ID (with required access).

    Lambda uses this information to setup an Elastic Network Interface (ENI) using an available IP address from your private subnet. Next you need to add a NAT Gateway for Internet access (no public IP).

    The NAT Gateway should be connected to a public subnet and a route needs to be added to the private subnet.

15. Amazon API Gateway is an AWS service for creating, publishing, maintaining, monitoring, and securing REST and WebSocket APIs at any scale. API developers can create APIs that access AWS or other web services as well as data stored in the AWS Cloud.

In API Gateway, an API's method request can take a payload in a different format from the corresponding integration request payload, as required in the backend. Similarly, the backend may return an integration response payload different from the method response payload, as expected by the frontend.

API Gateway lets you use mapping templates to map the payload from a method request to the corresponding integration request and from an integration response to the corresponding method response.

If an existing legacy service returns XML-style data, you can use the API Gateway to transform the output to JSON as part of your modernization effort. The API Gateway can be configured to transform the output of legacy services from XML to JSON, allowing them to make a move that is seamless and non-disruptive. The transformation is specified using JSON-Schema.

Therefore, the technique the Developer should use is to create a RESTful API with the API Gateway and transform the incoming JSON into a valid XML message for the SOAP interface using mapping templates.

16. VPC Flow Logs is a feature that enables you to capture information about the IP traffic going to and from network interfaces in your VPC. Flow log data can be published to Amazon CloudWatch Logs or Amazon S3. After you've created a flow log, you can retrieve and view its data in the chosen destination.

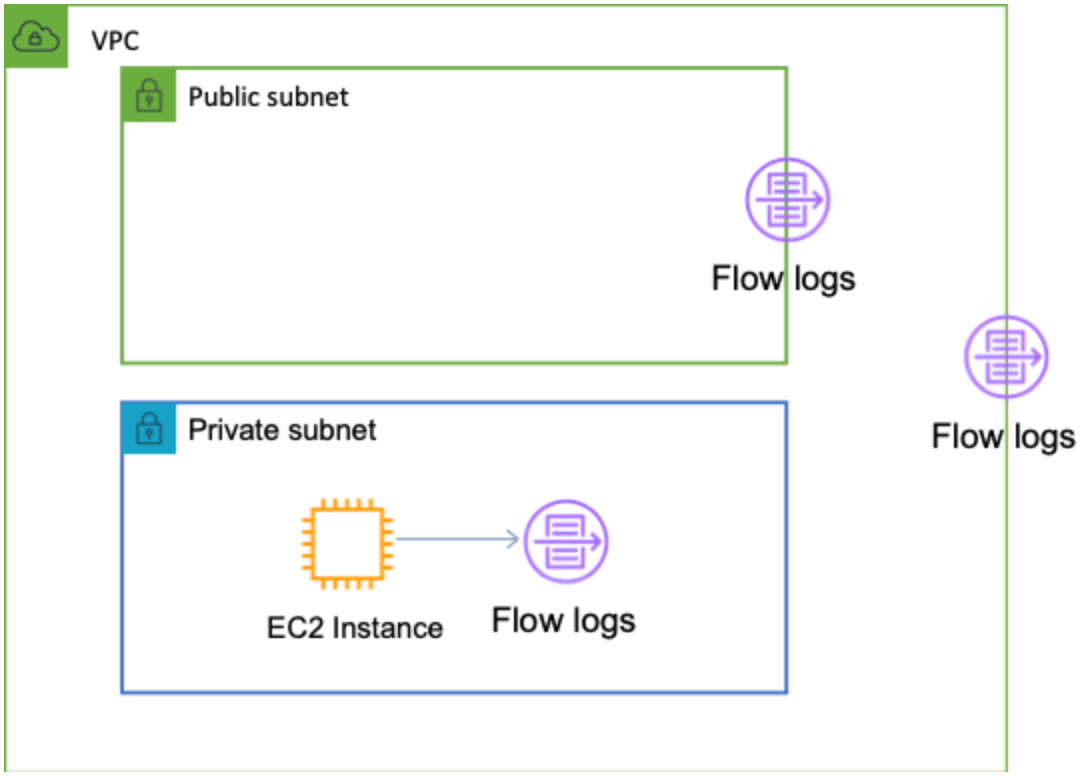
    Flow logs can help you with a number of tasks, such as:

    • Diagnosing overly restrictive security group rules

    • Monitoring the traffic that is reaching your instance

    • Determining the direction of the traffic to and from the network interfaces

    As you can see in the image below, you can create a flow log for a VPC, a subnet, or a network interface. If you create a flow log for a subnet or VPC, each network interface in that subnet or VPC is monitored.



Therefore, the Developer should create a flow log in the VPC and publish data to Amazon S3. The Developer could also choose CloudWatch Logs as a destination for publishing the data, but this is not presented as an option.

17. In Amazon Route 53 when you create an A record you must supply an IP address for the resource to connect to. For a public hosted zone this must be a public IP address.

There are three types of IP address that can be assigned to an Amazon EC2 instance:

• Public – public address that is assigned automatically to instances in public subnets and reassigned if instance is stopped/started.

• Private – private address assigned automatically to all instances.

• Elastic IP – public address that is static.

To ensure ongoing connectivity the Developer needs to use an Elastic IP address for the EC2 instance and DNS A record as this is the only type of static, public IP address you can assign to an Amazon EC2 instance.

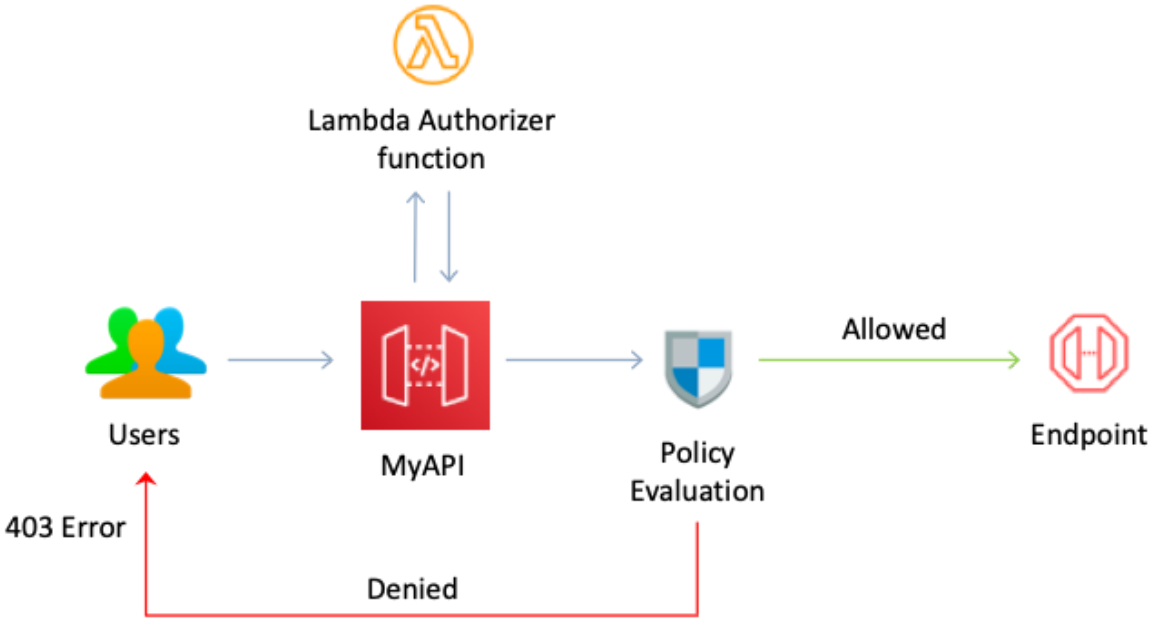
18. Some applications might need to perform many kinds of queries, using a variety of different attributes as query criteria. To support these requirements, you can create one or more global secondary indexes and issue Query requests against these indexes in Amazon DynamoDB.

    When items from a primary table are written to the GSI they consume write capacity units. It is essential to ensure the GSI has sufficient WCUs (typically, at least as many as the primary table). If writes are throttled on the GSI, the main table will be throttled (even if there's enough WCUs on the main table). LSIs do not cause any special throttling considerations.

    In this scenario, it is likely that the Developer assumed that the GSI would need fewer WCUs as it is more read-intensive and neglected to factor in the WCUs required for writing data into the GSI. Therefore, the most likely explanation is that the write capacity units on the GSI are under provisioned

19. A *Lambda authorizer* (formerly known as a *custom authorizer*) is an API Gateway feature that uses a Lambda function to control access to your API.

    A Lambda authorizer is useful if you want to implement a custom authorization scheme that uses a bearer token authentication strategy such as OAuth or SAML, or that uses request parameters to determine the caller's identity.



    When a client makes a request to one of your API's methods, API Gateway calls your Lambda authorizer, which takes the caller's identity as input and returns an IAM policy as output.

There are two types of Lambda authorizers:

• A *token-based* Lambda authorizer (also called a TOKEN authorizer) receives the caller's identity in a bearer token, such as a JSON Web Token (JWT) or an OAuth token.

• A *request parameter-based* Lambda authorizer (also called a REQUEST authorizer) receives the caller's identity in a combination of headers, query string parameters, stageVariables, and $context variables.

• For WebSocket APIs, only request parameter-based authorizers are supported.

In this scenario, the authentication is using headers in the request and therefore the request parameter-based Lambda authorizer should be used.

20. A *Lambda authorizer* (formerly known as a *custom authorizer*) is an API Gateway feature that uses a Lambda function to control access to your API.

    A Lambda authorizer is useful if you want to implement a custom authorization scheme that uses a bearer token authentication strategy such as OAuth or SAML, or that uses request parameters to determine the caller's identity.

    When a client makes a request to one of your API's methods, API Gateway calls your Lambda authorizer, which takes the caller's identity as input and returns an IAM policy as output.

    There are two types of Lambda authorizers:

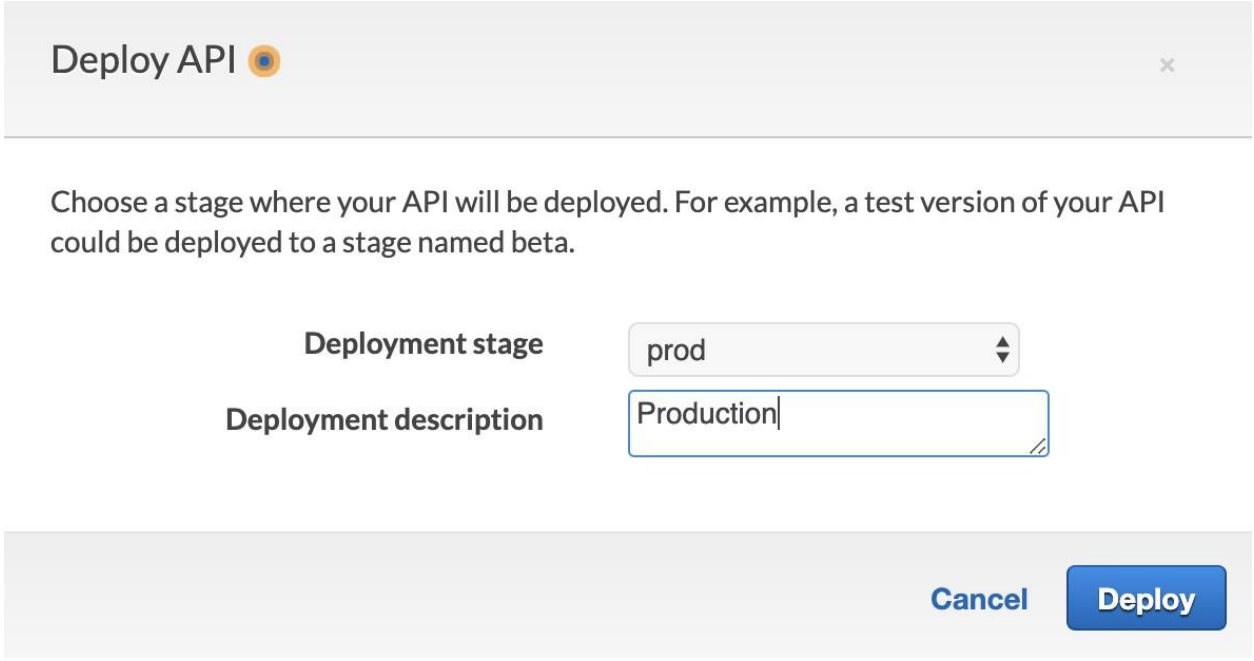
    A *token-based* Lambda authorizer (also called a TOKEN authorizer) receives the caller's identity in a bearer token, such as a JSON Web Token (JWT) or an OAuth token.

    A *request parameter-based* Lambda authorizer (also called a REQUEST authorizer) receives the caller's identity in a combination of headers, query string parameters, stageVariables, and $context variables.

    For this scenario, a Lambda authorizer is the most secure method available. It can also be used with usage plans and AWS recommends that you don't rely only on API keys, so a Lambda authorizer is a better solution.

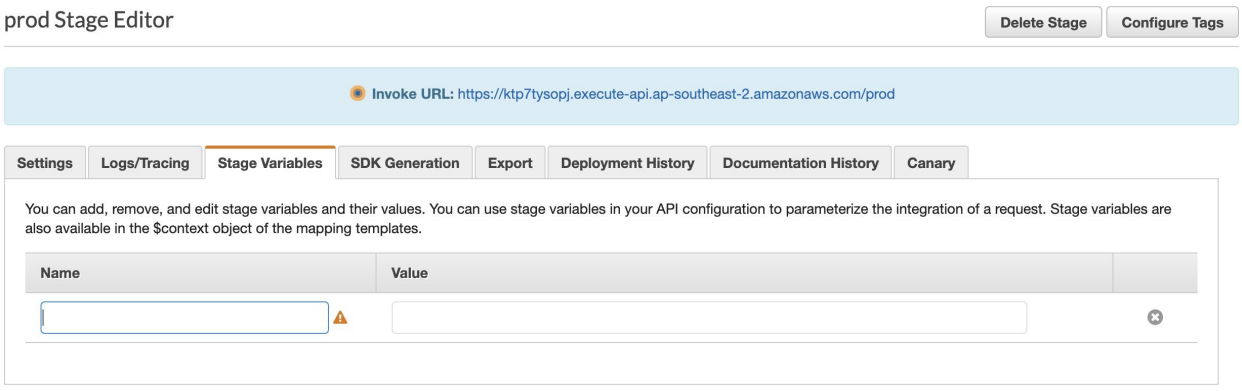
21. A stage is a named reference to a deployment, which is a snapshot of the API. You use a Stage to manage and optimize a particular deployment. For example, you can set up stage settings to enable caching, customize request throttling,

configure logging, define stage variables or attach a canary release for testing. APIs are deployed to stages:



Stage variables are name-value pairs that you can define as configuration attributes associated with a deployment stage of a REST API. They act like environment variables and can be used in your API setup and mapping templates.

With deployment stages in API Gateway, you can manage multiple release stages for each API, such as alpha, beta, and production. Using stage variables you can configure an API deployment stage to interact with different backend endpoints. For example, your API can pass a GET request as an HTTP proxy to the backend web host (for example, http://example.com).



In this case, the backend web host is configured in a stage variable so that when developers call your production endpoint, API Gateway calls example.com. When

you call your beta endpoint, API Gateway uses the value configured in the stage variable for the beta stage, and calls a different web host (for example, beta.example.com). Similarly, stage variables can be used to specify a different AWS Lambda function name for each stage in your API.

Therefore, for this scenario the Developers can deploy the API versions as unique stages with unique endpoints and use stage variables to provide further context such as connections to different backend services.

22. You can enable API caching in Amazon API Gateway to cache your endpoint's responses. With caching, you can reduce the number of calls made to your endpoint and also improve the latency of requests to your API.

    Caching is enabled for a stage. When you enable caching for a stage, API Gateway caches responses from your endpoint for a specified time-to-live (TTL) period, in seconds. API Gateway then responds to the request by looking up the endpoint response from the cache instead of making a request to your endpoint.

    The default TTL value for API caching is 300 seconds. The maximum TTL value is 3600 seconds. TTL=0 means caching is disabled.

    In this scenario we are asked to choose the most cost-efficient solution. Therefore, the best answer is to use a single API Gateway with three stages and, as caching is enabled per stage, we can choose to save cost by only enabling the cache on DEV and TEST when we need to perform tests relating to that functionality.

23. For web distributions, you can configure CloudFront to require that viewers use HTTPS to request your objects, so that connections are encrypted when CloudFront communicates with viewers. You also can configure CloudFront to use HTTPS to get objects from your origin, so that connections are encrypted when CloudFront communicates with your origin.

    If you configure CloudFront to require HTTPS both to communicate with viewers and to communicate with your origin, here's what happens when CloudFront receives a request for an object:

    1. A viewer submits an HTTPS request to CloudFront. There's some SSL/TLS negotiation here between the viewer and CloudFront. In the end, the viewer submits the request in an encrypted format.

2. If the object is in the CloudFront edge cache, CloudFront encrypts the response and returns it to the viewer, and the viewer decrypts it.

3. If the object is not in the CloudFront cache, CloudFront performs SSL/TLS negotiation with your origin and, when the negotiation is complete, forwards the request to your origin in an encrypted format.

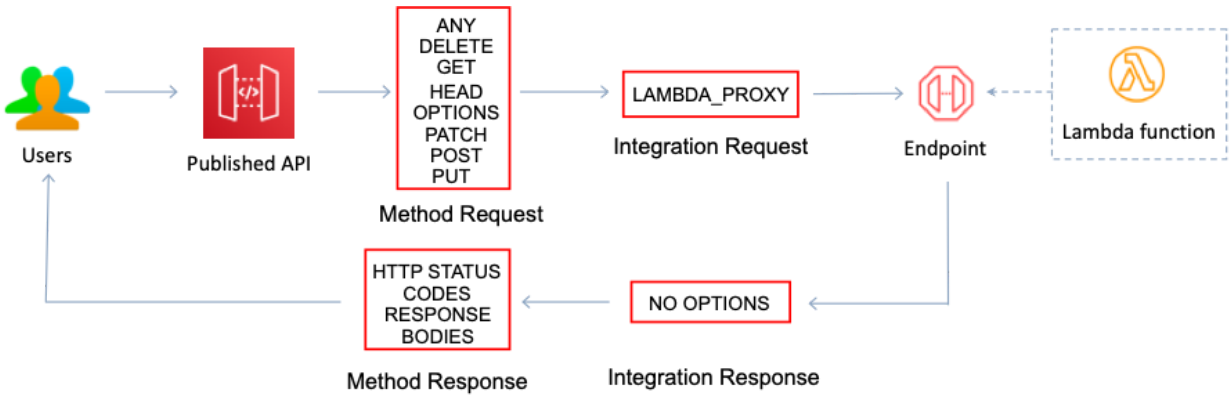
4. Your origin decrypts the request, encrypts the requested object, and returns the object to CloudFront.

5. CloudFront decrypts the response, re-encrypts it, and forwards the object to the viewer. CloudFront also saves the object in the edge cache so that the object is available the next time it's requested.

6. The viewer decrypts the response.

To enable SSL between the origin and the distribution the Developer can configure the Origin Protocol Policy. Depending on the domain name used (CloudFront default or custom), the steps are different. To enable SSL between the end-user and CloudFront the Viewer Protocol Policy should be configured.

24. Amazon API Gateway Lambda proxy integration is a simple, powerful, and nimble mechanism to build an API with a setup of a single API method. The Lambda proxy integration allows the client to call a single Lambda function in the backend. The function accesses many resources or features of other AWS services, including calling other Lambda functions.



In Lambda proxy integration, when a client submits an API request, API Gateway passes to the integrated Lambda function the raw request as-is, except that the order of the request parameters is not preserved. This request data includes the request headers, query string parameters, URL path variables, payload, and API configuration data.

This solution provides a front end that can listen for HTTP GET requests and then proxy them to the Lambda function and is the simplest option to implement and also the most cost-effective.

25. You choose an API integration type according to the types of integration endpoint you work with and how you want data to pass to and from the integration endpoint. For a Lambda function, you can have the Lambda proxy integration, or the Lambda custom integration.

    For an HTTP endpoint, you can have the HTTP proxy integration or the HTTP custom integration. For an AWS service action, you have the AWS integration of the non-proxy type only. API Gateway also supports the mock integration, where API Gateway serves as an integration endpoint to respond to a method request.

    As this is a Docker deployment running on Elastic Beanstalk the HTTP integration types are applicable. There are two options:

    HTTP: This type of integration lets an API expose HTTP endpoints in the backend. With the HTTP integration, also known as the HTTP custom integration, you must configure both the integration request and integration response. You must set up necessary data mappings from the method request to the integration request, and from the integration response to the method response.

    HTTP_PROXY: The HTTP proxy integration allows a client to access the backend HTTP endpoints with a streamlined integration setup on single API method. You do not set the integration request or the integration response. API Gateway passes the incoming request from the client to the HTTP endpoint and passes the outgoing response from the HTTP endpoint to the client.

    As we can see from the above explanation, the most suitable integration type for this deployment is going to be the HTTP_PROXY.

26. If you need to remove a file from CloudFront edge caches before it expires, you can do one of the following:

    • Invalidate the file from edge caches. The next time a viewer requests the file, CloudFront returns to the origin to fetch the latest version of the file.

• Use file versioning to serve a different version of the file that has a different name. For more information, see Updating Existing Files Using Versioned File Names.

To invalidate files, you can specify either the path for individual files or a path that ends with the * wildcard, which might apply to one file or to many, as shown in the following examples:

• /images/image1.jpg

• /images/image*

• /images/*

Therefore, the Developer should invalidate the old versions of the images on the edge cache as this will remove the cached images and the new versions of the images will then be cached when the next request is received.

27. This scenario requires encryption of in-flight data which can be done by implementing HTTPS. To do this the organization must configure the Origin Protocol Policy and the Viewer Protocol Policy on the CloudFront Distribution.

The Origin Protocol Policy can be used to select whether you want CloudFront to connect to your origin using only HTTP, only HTTPS, or to connect by matching the protocol used by the viewer. For example, if you select Match Viewer for the Origin Protocol Policy, and if the viewer connects to CloudFront using HTTPS, CloudFront will connect to your origin using HTTPS.

If you want CloudFront to allow viewers to access your web content using either HTTP or HTTPS, specify HTTP and HTTPS. If you want CloudFront to redirect all HTTP requests to HTTPS, specify Redirect HTTP to HTTPS. If you want CloudFront to require HTTPS, specify HTTPS Only.

28. Weighted routing lets you associate multiple resources with a single domain name (example.com) or subdomain name (acme.example.com) and choose how much traffic is routed to each resource. This can be useful for a variety of purposes, including load balancing and testing new versions of software.

Create Record Set

Name: dctlabs.com.

Type: A – IPv4 address

Alias: Yes No

Alias Target: dualstack.MyALB-1458270708.ap-sou

Alias Hosted Zone ID: Z1GM3OXH4ZPM65

You can also type the domain name for the resource. Examples:
- CloudFront distribution domain name: d111111abcdef8.cloudfront.net
- Elastic Beanstalk environment CNAME: example.elasticbeanstalk.com
- ELB load balancer DNS name: example-1.us-east-2.elb.amazonaws.com
- S3 website endpoint: s3-website.us-east-2.amazonaws.com
- Resource record set in this hosted zone: www.example.com
- VPC endpoint: example.us-east-2.vpce.amazonaws.com
- API Gateway custom regional API: d-abcde12345.execute-api.us-west-2.amazonaws.com
- Global Accelerator DNS name: a012345abc.awsglobalaccelerator.com

Learn More

Routing Policy: Weighted

Route 53 responds to queries based on weighting that you specify in this and other record sets that have the same name and type. Learn More

Weight: 204

Set ID: 1

Description of this record set that is unique within the group of weighted sets.

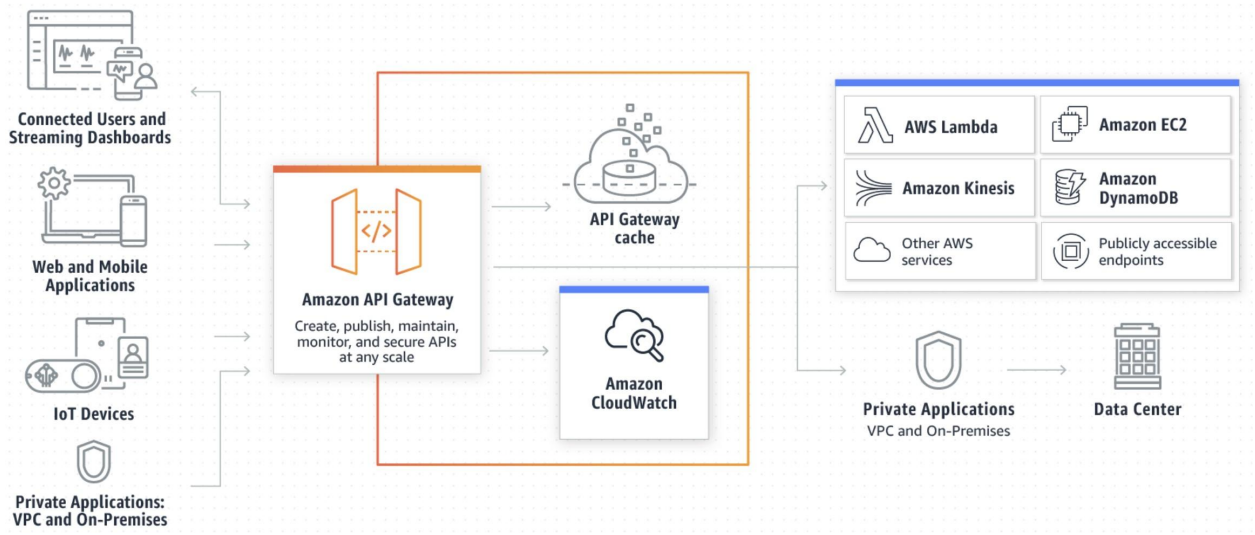
Example:

My Seattle Data Center

In this case the Developer can use a weighted routing policy to direct 20% of the incoming traffic to the new site as required.

29. Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services.

    Using API Gateway, you can create RESTful APIs and WebSocket APIs that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

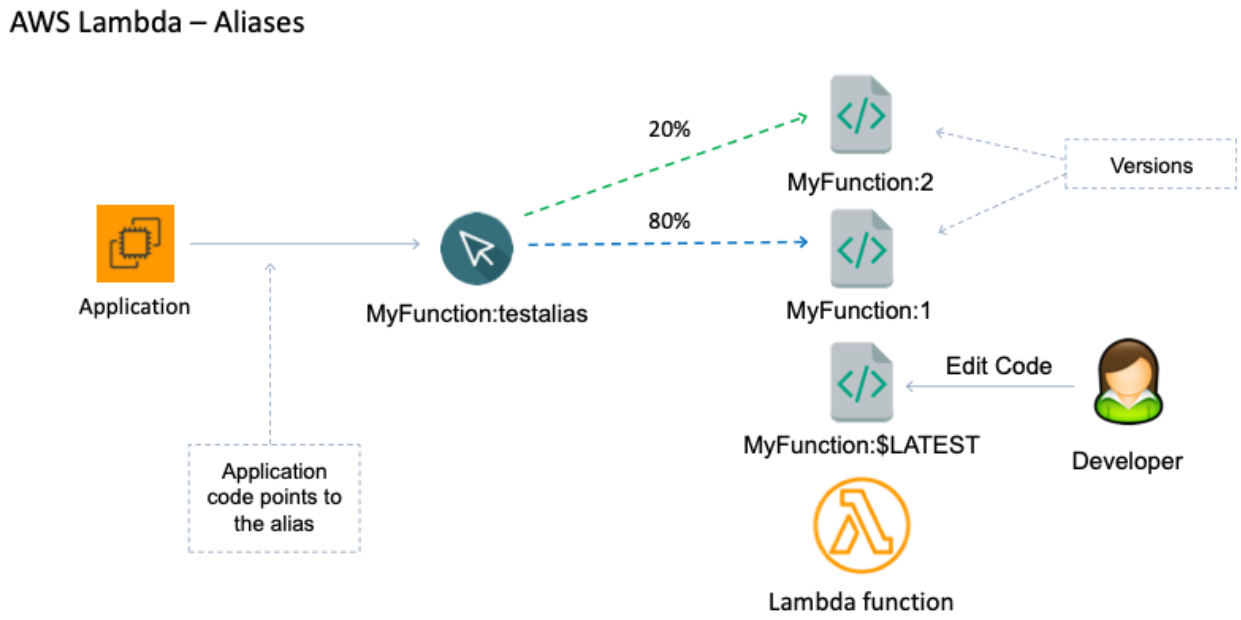


    API Gateway can be used as the single interface for consumers of the services provided by the organization in this scenario. This solution will simplify the architecture.

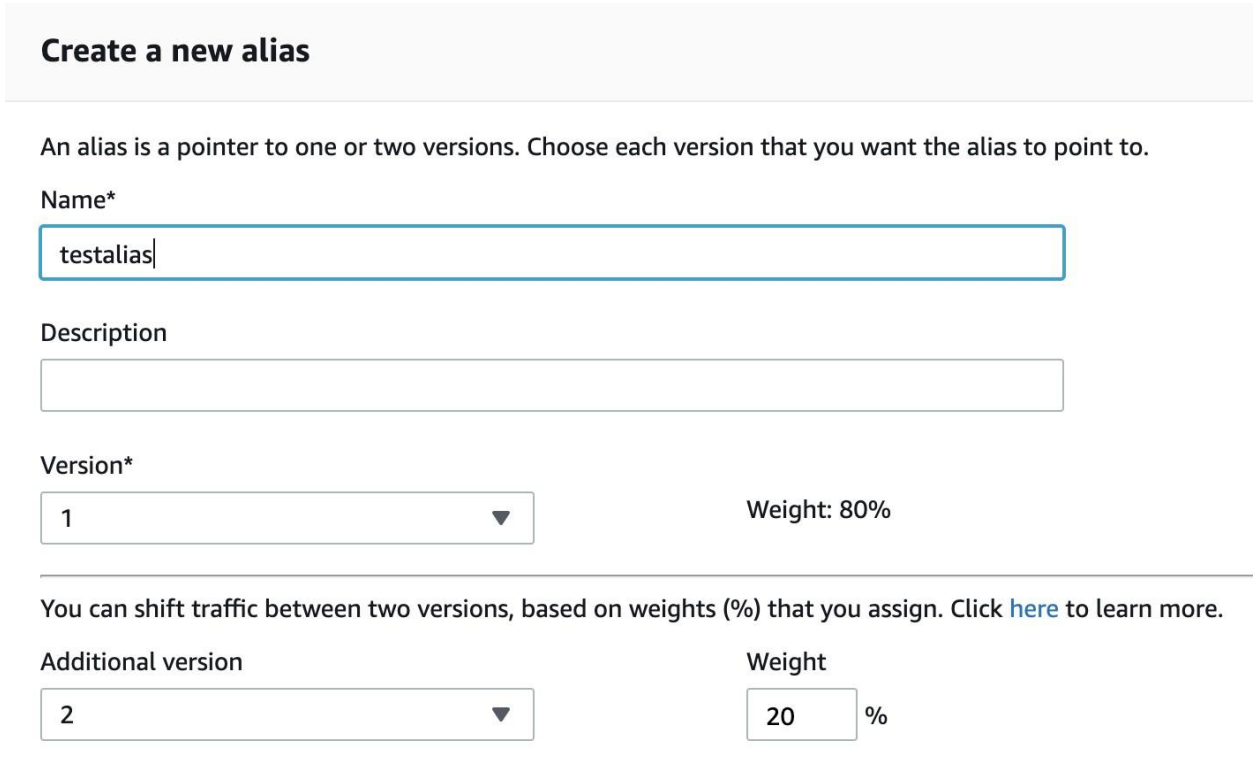
30. To connect to AWS services from a private subnet with no internet access, use VPC endpoints. A *VPC endpoint* for DynamoDB enables resources in a VPC to use their private IP addresses to access DynamoDB with no exposure to the public internet.

    When you create a VPC endpoint for DynamoDB, any requests to a DynamoDB endpoint within the Region (for example, *dynamodb.us-west-2.amazonaws.com*) are routed to a private DynamoDB endpoint within the Amazon network.

31. You can create one or more aliases for your AWS Lambda function. A Lambda alias is like a pointer to a specific Lambda function version. Users can access the function version using the alias ARN.



This is the best way to setup the Lambda function so you don't need to modify the application code when a new version is published. Instead, the developer will simply need to update the Alias to point to the new version:



As you can see above you can also point to multiple versions and send a percentage of traffic to each. This is great for testing new code.

32. To serve a static website hosted on Amazon S3, you can deploy a CloudFront distribution using one of these configurations:

    • Using a REST API endpoint as the origin with access restricted by an origin access identity (OAI)

    • Using a website endpoint as the origin with anonymous (public) access allowed

    • Using a website endpoint as the origin with access restricted by a Referer header

    All assets of this website are static (HTML, images, client-side JavaScript), therefore this website is compatible with both S3 static websites and Amazon CloudFront. The simplest way to minimize latency is to create a CloudFront distribution and configure the static website as an origin.

33. (in this question)The company needs to add security to their website by encrypting traffic in-transit using HTTPS. This requires adding SSL/TLS certificates to enable the encryption. The process of encrypting and decrypting data is CPU intensive and therefore the company needs to avoid adding certificates to the EC2 instances as that will place further load on their CPUs.

    Therefore, the solution is to configure SSL certificates on the Elastic Load Balancer and then configure SSL termination. This can be done by adding a certificate to a HTTPS listener on the load balancer.

34. This solution can be achieved by adding the AWS Lambda function to a VPC through the function configuration and by creating a VPC endpoint for Amazon SQS. This will result in the services using purely private IP addresses to communicate without traversing the public Internet.
35. To restrict access to content that you serve from Amazon S3 buckets, you create CloudFront signed URLs or signed cookies to limit access to files in your Amazon S3 bucket, and then you create a special CloudFront user called an origin access identity (OAI) and associate it with your distribution. Then you configure permissions so that CloudFront can use the OAI to access and serve files to your users, but users can't use a direct URL to the S3 bucket to access a file there.

Taking these steps help you maintain secure access to the files that you serve through CloudFront.

36. To ensure high availability and fault tolerance the Developer should create a subnet within each availability zone. The EC2 instances should then be distributed between these subnets.

    The Developer would likely use Amazon EC2 Auto Scaling which will automatically launch instances in each subnet and then Elastic Load Balancing to distributed incoming traffic.

37. You can use Amazon S3 to host a static website. On a *static* website, individual webpages include static content. They might also contain client-side scripts.

    To host a static website on Amazon S3, you configure an Amazon S3 bucket for website hosting and then upload your website content to the bucket. When you configure a bucket as a static website, you enable static website hosting, set permissions, and add an index document.

    To get content closer to users for better performance you can also use Amazon CloudFront in front of the S3 static website. To serve a static website hosted on Amazon S3, you can deploy a CloudFront distribution using one of these configurations:

    • Using a REST API endpoint as the origin with access restricted by an origin access identity (OAI)

    • Using a website endpoint as the origin with anonymous (public) access allowed

    • Using a website endpoint as the origin with access restricted by a Referer header

    Therefore, the combination of services should be Amazon S3 and Amazon CloudFront

38. This scenario includes a web application that will use RESTful API calls to determine the status of orders and dynamically return the results back to the company's customers. Therefore, the two best options are as per below:

    • Amazon API Gateway; AWS Lambda – this choice includes API Gateway which is provides managed REST APIs and Lambda which can run the backend code for the application. This is a good solution for this scenario.

• Elastic Load Balancing; Amazon EC2 – with this choice the ELB can load balance to one or more EC2 instances which can run the RESTful APIs and compute functions. This is also a good choice but could be more costly (operationally and financially).

None of the other options provide a workable solution for this scenario.

AWS DATABASE

1. When you request a strongly consistent read, DynamoDB returns a response with the most up-to-date data, reflecting the updates from all prior write operations that were successful.

   The GetItem operation returns a set of attributes for the item with the given primary key. If there is no matching item, GetItem does not return any data and there will be no Item element in the response.

   GetItem provides an eventually consistent read by default. If your application requires a strongly consistent read, set ConsistentRead to true. Although a strongly consistent read might take more time than an eventually consistent read, it always returns the last updated value.

2. In general, Scan operations are less efficient than other operations in DynamoDB. A Scan operation always scans the entire table or secondary index. It then filters out values to provide the result you want, essentially adding the extra step of removing data from the result set.

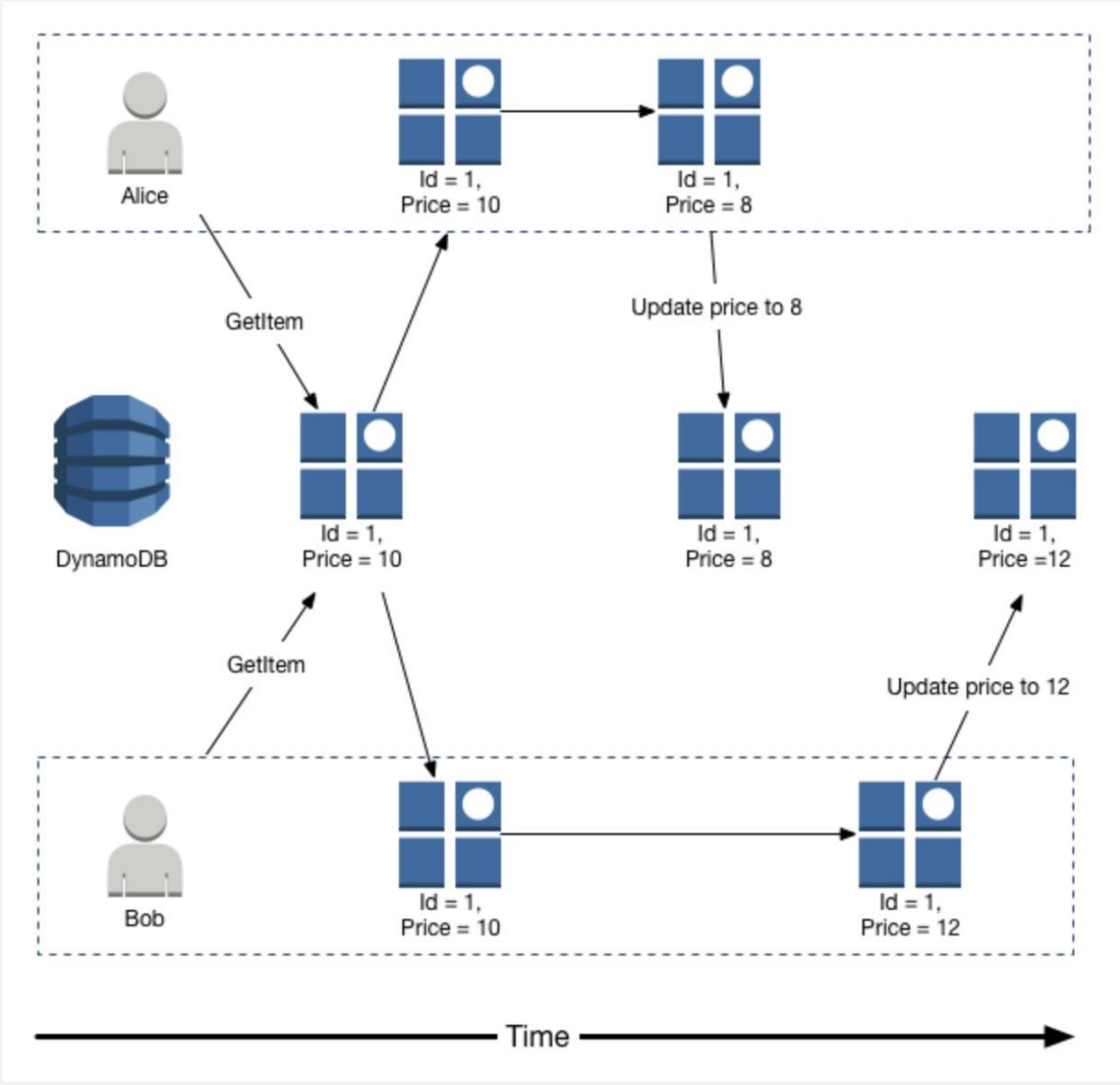
   If possible, you should avoid using a Scan operation on a large table or index with a filter that removes many results. Also, as a table or index grows, the Scan operation slows. The Scan operation examines every item for the requested values and can use up the provisioned throughput for a large table or index in a single operation. For faster response times, design your tables and indexes so that your applications can use Query instead of Scan. (For tables, you can also consider using the GetItem and BatchGetItem APIs.)

Additionally, eventual consistency consumes fewer RCUs than strong consistency. Therefore, the application should be refactored to use query APIs with eventual consistency.

3. By default, the DynamoDB write operations (PutItem, UpdateItem, DeleteItem) are *unconditional*: Each operation overwrites an existing item that has the specified primary key.

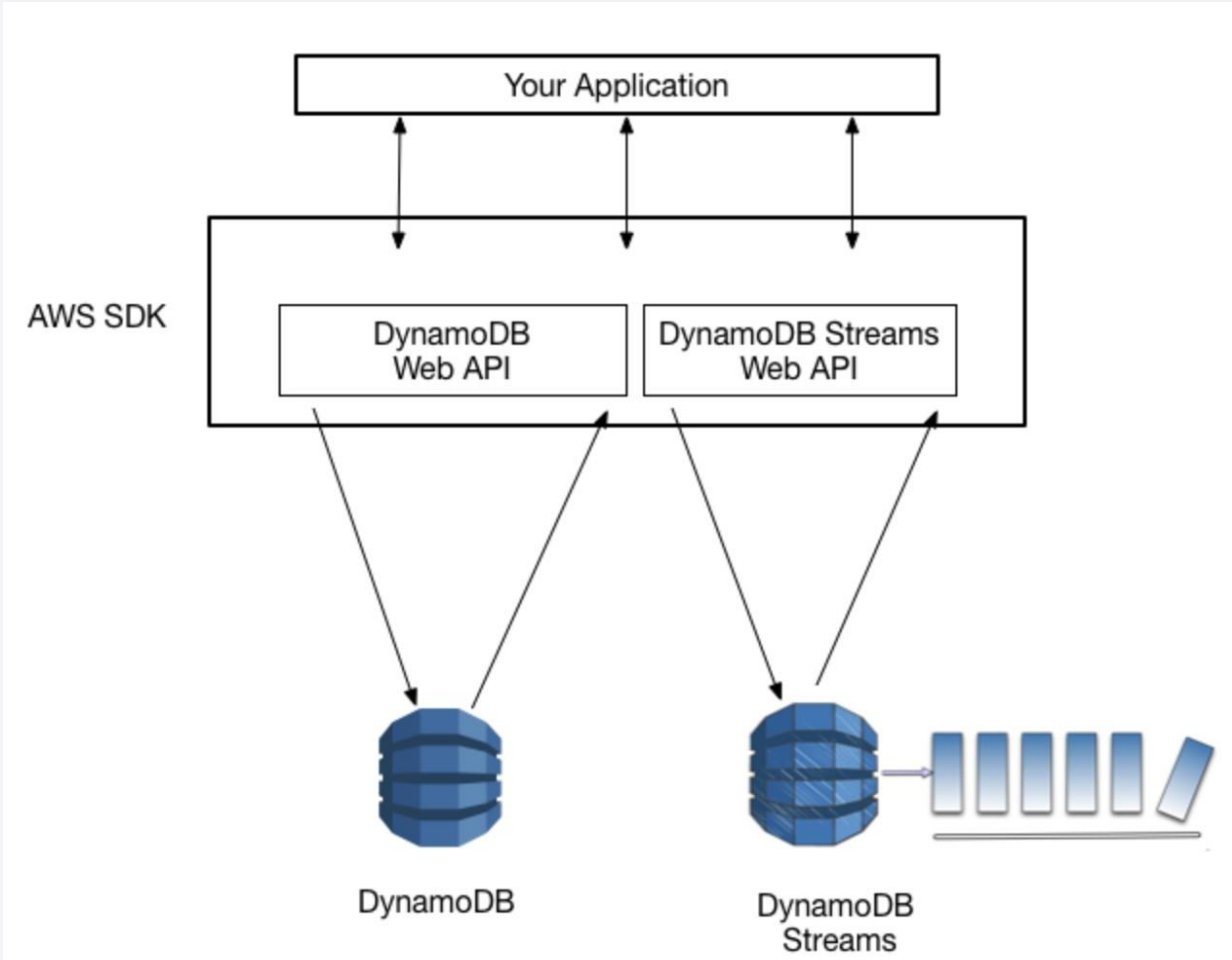
   DynamoDB optionally supports conditional writes for these operations. A conditional write succeeds only if the item attributes meet one or more expected conditions. Otherwise, it returns an error. Conditional writes are helpful in many situations. For example, you might want a PutItem operation to succeed only if there is not already an item with the same primary key. Or you could prevent an UpdateItem operation from modifying an item if one of its attributes has a certain value.

   Conditional writes are helpful in cases where multiple users attempt to modify the same item. Consider the following diagram, in which two users (Alice and Bob) are working with the same item from a DynamoDB table.



Therefore, conditional writes should be used to prevent the overwriting that has been occurring.

4. DynamoDB Streams captures a time-ordered sequence of item-level modifications in any DynamoDB table and stores this information in a log for up to 24 hours. Applications can access this log and view the data items as they appeared before and after they were modified, in near-real time.



Whenever an application creates, updates, or deletes items in the table, DynamoDB Streams writes a stream record with the primary key attributes of the items that were modified. A stream record contains information about a data modification to a single item in a DynamoDB table. You can configure the stream so that the stream records capture additional information, such as the "before" and "after" images of modified items.

For this scenario, we can enable a DynamoDB stream on the “orders” table and then configure the “customers” microservice to read records from the stream and then write those records, or relevant attributes of those records, to the “customers’ table.

5. Amazon DynamoDB Accelerator (DAX) is a fully managed, highly available, in-memory cache for DynamoDB that delivers up to a 10x performance improvement – from milliseconds to microseconds – even at millions of requests per second. DAX does all the heavy lifting required to add

in-memory acceleration to your DynamoDB tables, without requiring developers to manage cache invalidation, data population, or cluster management.

How it works:

DAX is a write-through caching service – this means the data is written to the cache as well as the back end store at the same time.

Allows you to point your DynamoDB API calls at the DAX cluster and if the item is in the cache (cache hit), DAX returns the result to the application.

If the item requested is not in the cache (cache miss) then DAX performs an Eventually Consistent GetItem operations against DynamoDB

Retrieval of data from DAX reduces the read load on DynamoDB tables.

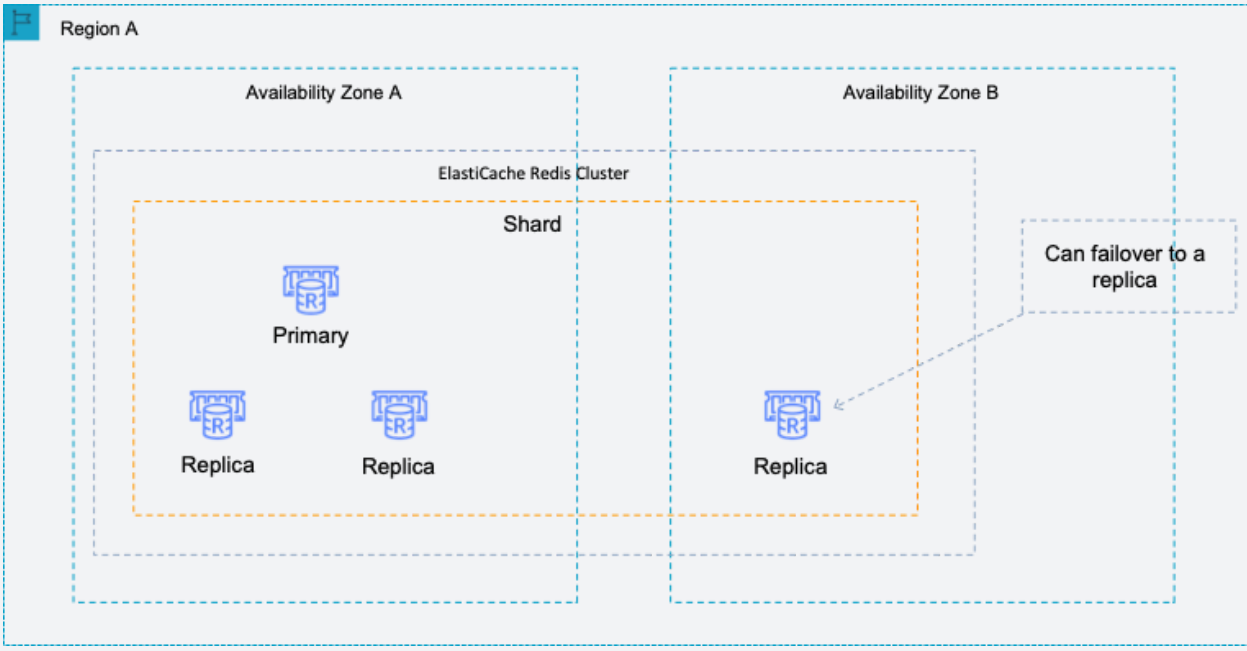
This may result in being able to reduce the provisioned read capacity on the table.

DynamoDB DAX is the correct solution for best performance for a read-heavy workload.

6. Single-node Amazon ElastiCache Redis clusters are in-memory entities with limited data protection services (AOF). If your cluster fails for any reason, you lose all the cluster's data.

   However, if you're running the Redis engine, you can group 2 to 6 nodes into a cluster with replicas where 1 to 5 read-only nodes contain replicate data of the group's single read/write primary node.

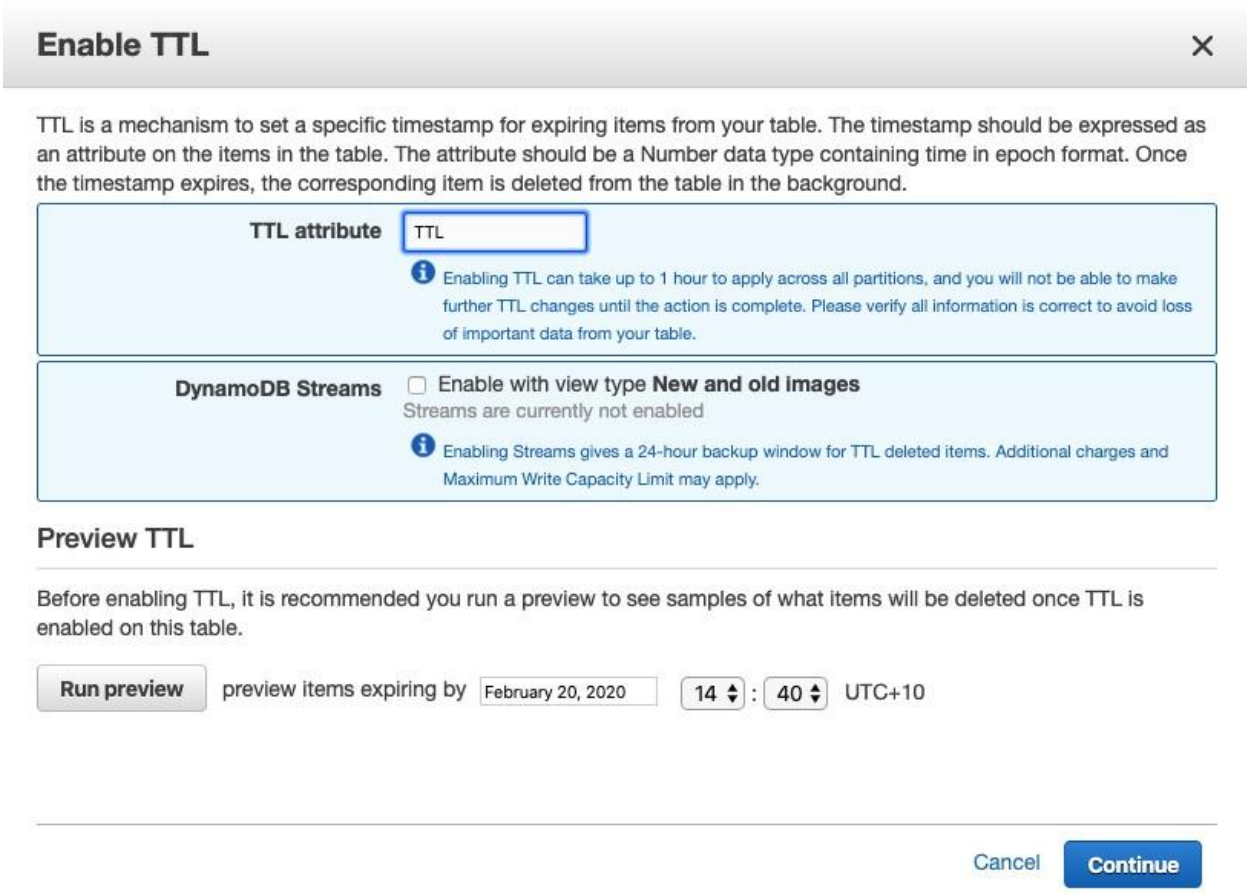
   In this scenario, if one node fails for any reason, you do not lose all your data since it is replicated in one or more other nodes. Due to replication latency, some data may be lost if it is the primary read/write node that fails.



Therefore, the best solution is to use ElastiCache Redis with replicas.

7. Amazon Relational Database Service (Amazon RDS) is a managed service that makes it easy to set up, operate, and scale a relational database in the cloud. RDS is an Online Transaction Processing (OLTP) type of database. The primary use case is a transactional database (rather than analytical) and it is best for structured, relational data store requirements.
8. Time to Live (TTL) for Amazon DynamoDB lets you define when items in a table expire so that they can be automatically deleted from the database. With TTL enabled on a table, you can set a timestamp for deletion on a per-item basis, allowing you to limit storage usage to only those records that are relevant.

   TTL is useful if you have continuously accumulating data that loses relevance after a specific time period (for example, session data, event logs, usage patterns, and other temporary data). If you have sensitive data that must be retained only for a certain amount of time according to contractual or regulatory obligations, TTL helps you ensure that it is removed promptly and as scheduled.



Therefore, using a TTL is the best solution as it will automatically purge items after their useful lifetime.

9. You can monitor Amazon DynamoDB using CloudWatch, which collects and processes raw data from DynamoDB into readable, near real-time metrics. These statistics are retained for a period of time, so that you can access historical information for a better perspective on how your web application or service is performing. By default, DynamoDB metric data is sent to CloudWatch automatically.

   To determine how much of the provisioned capacity is being used you can monitor ConsumedReadCapacityUnits or ConsumedWriteCapacityUnits over the specified time period.

10. A *read capacity unit* represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size. For example, suppose that you create a table with 10 provisioned read

capacity units. This allows you to perform 10 strongly consistent reads per second, or 20 eventually consistent reads per second, for items up to 4 KB.

Reading an item larger than 4 KB consumes more read capacity units. For example, a strongly consistent read of an item that is 8 KB (4 KB × 2) consumes 2 read capacity units. An eventually consistent read on that same item consumes only 1 read capacity unit.

Item sizes for reads are rounded up to the next 4 KB multiple. For example, reading a 3,500-byte item consumes the same throughput as reading a 4 KB item. Therefore, the smaller (1 KB) items in this scenario would consume the same number of RCUs as the 4 KB items. Also, we know that eventually consistent reads consume half the RCUs of strongly consistent reads.

The following bullets provide the read throughput for each configuration:

· Eventually consistent, 15 RCUs, 1 KB item = 30 items read per second.

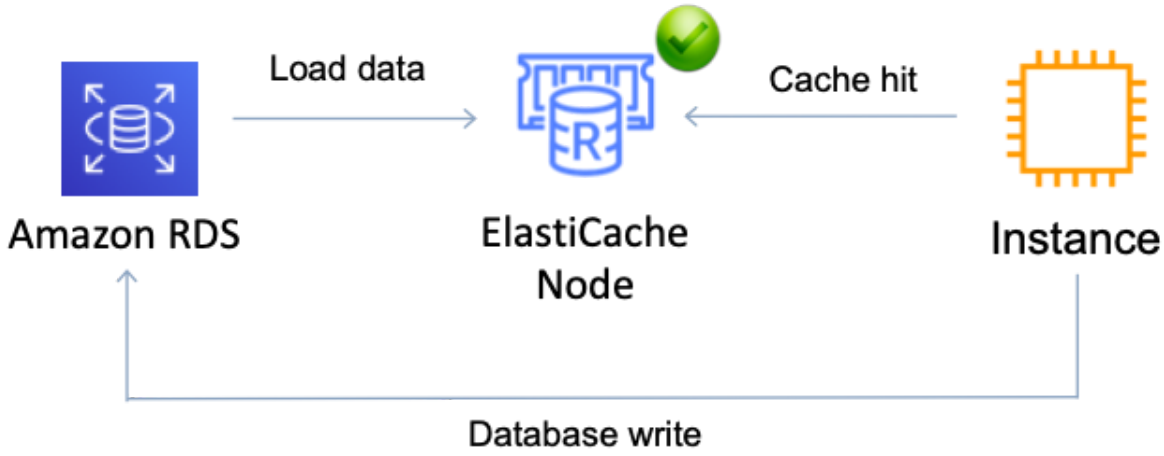
· Strongly consistent, 15 RCUs, 1 KB item = 15 items read per second.

· Eventually consistent, 5 RCUs, 4 KB item = 10 items read per second.

· Strongly consistent, 5 RCUs, 4 KB item = 5 items read per second.

Therefore, the Developer should choose the option to enable eventually consistent reads of 15 RCUs reading items that are 1 KB in size as this will result in the highest number of items read per second.

11. There are two options that can assist with improving performance for read requests: Amazon RDS read replicas, and Amazon ElastiCache. Both of these solutions will provide horizontal scaling for read requests to reduce the impact on the main database.



However, only Amazon ElastiCache is an in-memory database so the best solution is for the Developer to use Amazon ElastiCache to improve performance for repeated read requests.

12. With this scenario we have a table that has a large number of items quickly written to it on a recurring schedule. These items are no longer of use after they have been processed (within 2 hours) so from that point on until the next job the table is not being used. The items need to be deleted and we need to choose the most efficient (think cost as well as operations) way of doing this.

    Any delete operation will consume RCUs to scan/query the table and WCUs to delete the items. It will be much cheaper and simpler to just delete the table and recreate it again ahead of the next batch job. This can easily be automated through the API.

13. In an Amazon DynamoDB table, the primary key that uniquely identifies each item in the table can be composed not only of a partition key, but also of a sort key.

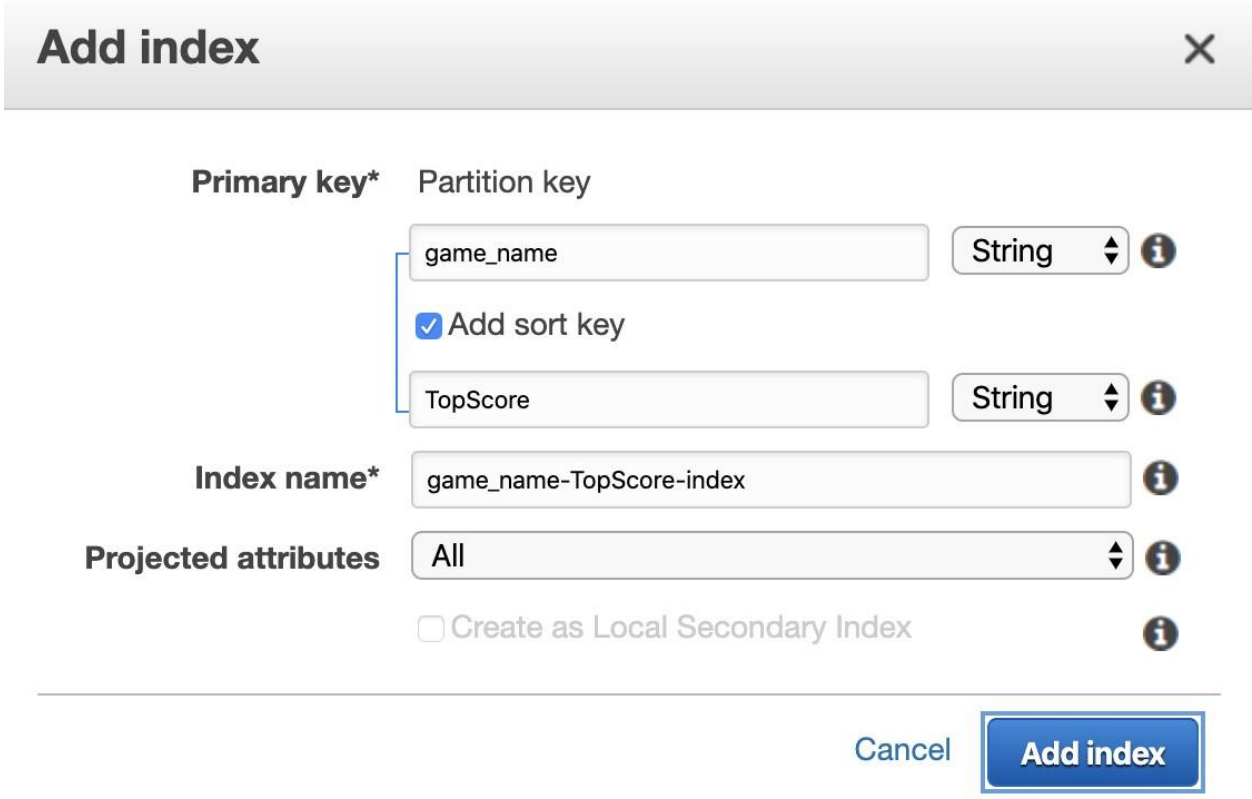
    Well-designed sort keys have two key benefits:

    - They gather related information together in one place where it can be queried efficiently. Careful design of the sort key lets you retrieve commonly needed groups of related items using range queries with operators such as begins_with, between, >, <, and so on.

    - Composite sort keys let you define hierarchical (one-to-many) relationships in your data that you can query at any level of the hierarchy.

To speed up queries on non-key attributes, you can create a global secondary index. A global secondary index contains a selection of attributes from the base table, but they are organized by a primary key that is different from that of the table. The index key does not need to have any of the key attributes from the table. It doesn't even need to have the same key schema as a table.

For this scenario we need to identify the top achieved score for each game. The most efficient way to do this is to create a global secondary index using “game_name” as the partition key and “TopScore” as the sort key. We can then efficiently query the global secondary index to find the top achieved score for each game.



14. With provisioned capacity mode, you specify the number of data reads and writes per second that you require for your application.

Read capacity unit (RCU):

• Each API call to read data from your table is a read request.

• Read requests can be strongly consistent, eventually consistent, or transactional.

• For items up to 4 KB in size, one RCU can perform one *strongly consistent* read request per second.

• Items larger than 4 KB require additional RCUs.

• For items up to 4 KB in size, one RCU can perform two *eventually consistent* read requests per second.

• *Transactional* read requests require two RCUs to perform one read per second for items up to 4 KB.

• For example, a strongly consistent read of an 8 KB item would require two RCUs, an eventually consistent read of an 8 KB item would require one RCU, and a transactional read of an 8 KB item would require four RCUs.

Write capacity unit (WCU):

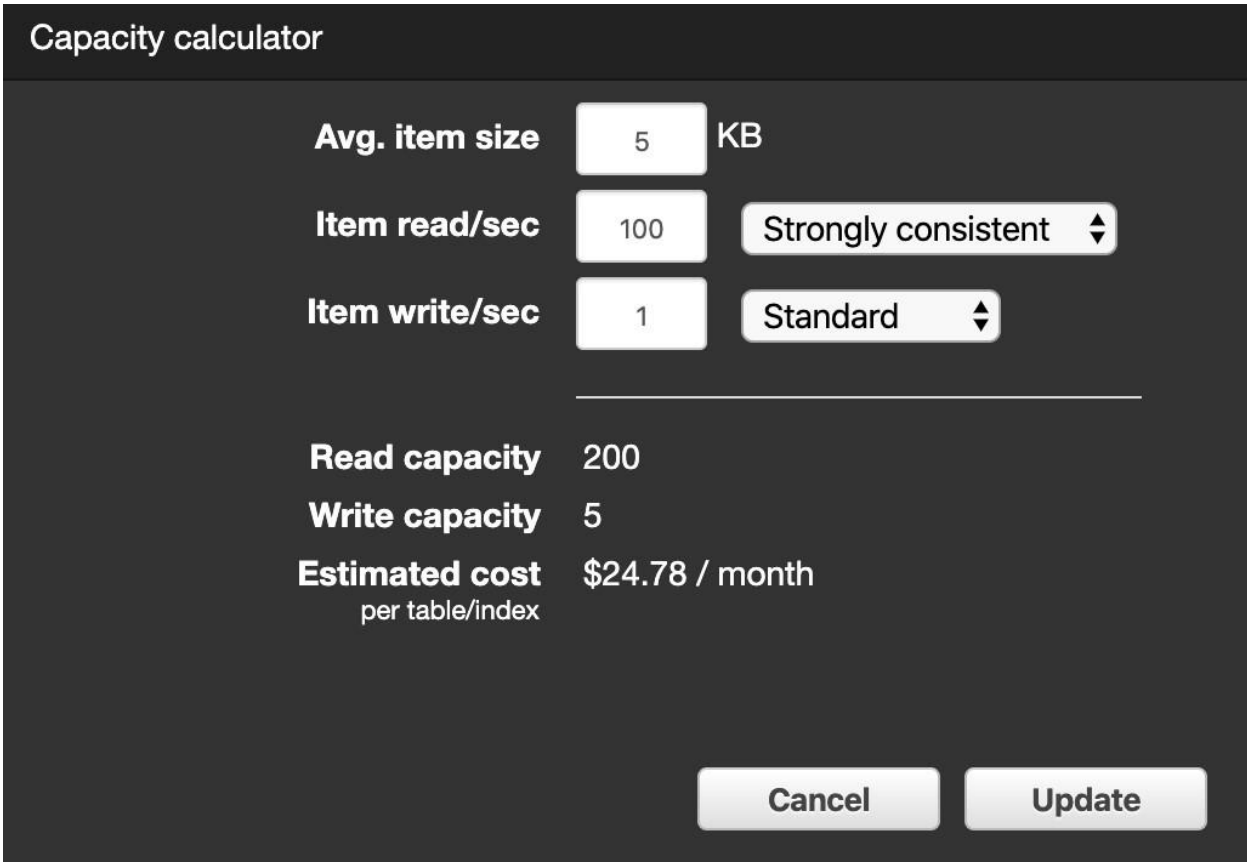
• Each API call to write data to your table is a write request.

• For items up to 1 KB in size, one WCU can perform one *standard* write request per second.

• Items larger than 1 KB require additional WCUs.

• *Transactional* write requests require two WCUs to perform one write per second for items up to 1 KB.

• For example, a standard write request of a 1 KB item would require one WCU, a standard write request of a 3 KB item would require three WCUs, and a transactional write request of a 3 KB item would require six WCUs.



To determine the number of RCUs required to handle 100 strongly consistent reads per/second with an average item size of 5KB, perform the following steps:

1. Determine the average item size by rounding up the next multiple of 4KB (5KB rounds up to 8KB).

2. Determine the RCU per item by dividing the item size by 4KB (8KB/4KB = 2).

3. **Multiply the value from step 2 with the number of reads required per second (2x100 = 200).**

15. In this scenario the Developer needs to maximize efficiency of RCUs. Therefore, the Developer will need to consider the item size and consistency model to determine the most efficient usage of RCUs.

Item size/consistency model: we know that both 1 KB items and 4 KB items consume the same number of RCUs as a *read capacity unit*

represents one strongly consistent read per second, or two eventually consistent reads per second, for an item up to 4 KB in size.

The following bullets provide the read throughput for each configuration:

• Eventually consistent, 15 RCUs, 1 KB item = 30 items/s = 2 items per RCU

• Strongly consistent, 15 RCUs, 1 KB item = 15 items/s = 1 item per RCU

• Eventually consistent, 5 RCUs, 4 KB item = 10 items/s = 2 items per RCU

• Strongly consistent, 5 RCUs, 4 KB item = 5 items/s = 1 item per RCU

From the above we can see that 4 KB items with eventually consistent reads is the most efficient option. Therefore, the Developer should choose the option “Eventually consistent reads of 5 RCUs reading items that are 4 KB in size”. This will achieve 2x 4 KB items per RCU.

16. The **DynamoDB Session Handler** is a custom session handler for PHP that allows developers to use Amazon DynamoDB as a session store. Using DynamoDB for session storage alleviates issues that occur with session handling in a distributed web application by moving sessions off of the local file system and into a shared location. DynamoDB is fast, scalable, easy to setup, and handles replication of your data automatically.
17. Amazon RDS read replicas are used for offloading reads from the primary database instance. Read replicas provide a read-only copy of the database. In this scenario this represents the simplest way of achieving the required outcome. The application will need to be modified to point to the read replica for all read requests. This requires some code changes, but they are minimal.
18. ElastiCache is a fully managed, low latency, in-memory data store that supports either Memcached or Redis. With ElastiCache, management tasks such as provisioning, setup, patching, configuration, monitoring, backup, and failure recovery are taken care of, so you can focus on application development.

    Amazon ElastiCache is a popular choice for real-time use cases like Caching, Session Stores, Gaming, Geospatial Services, Real-Time Analytics, and Queuing. For this scenario, the company is currently running an in-memory database and needs to ensure similar performance, so this is an ideal use case for ElastiCache.

19. Note:- Amazon RDS is not an in-memory database
20. Amazon DynamoDB" is not an in-memory database.
21. Exponential backoff can improve an application's reliability by using progressively longer waits between retries. When using the AWS SDK, this logic is built-in. However, in this case the application is incompatible with the AWS SDK so it is necessary to manually implement exponential backoff.
22. Amazon DynamoDB is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability. It is a non-relational (schema-less), key-value type of database. This is the most suitable solution for this requirement.
23. Time to Live (TTL) for Amazon DynamoDB lets you define when items in a table expire so that they can be automatically deleted from the database. With TTL enabled on a table, you can set a timestamp for deletion on a per-item basis, allowing you to limit storage usage to only those records that are relevant.

    TTL is useful if you have continuously accumulating data that loses relevance after a specific time period (for example, session data, event logs, usage patterns, and other temporary data). If you have sensitive data that must be retained only for a certain amount of time according to contractual or regulatory obligations, TTL helps you ensure that it is removed promptly and as scheduled.

    SessionData

UserName	SessionId	CreationTime	ExpirationTime (TTL)	SessionInfo	...
user1	74686572652773	1461931200	1461938400	{JSON Document}	...
user2	6e6f7468696e67	1461920400	1461927600	{JSON Document}	...
user3	746f2073656520	1461922200	1461929400	{JSON Document}	...
user4	68657265212121	1461925380	1461932580	{JSON Document}	...
user5	6e6572642e2e2e	1461920400	1461927600	{JSON Document}	...

    When Time to Live (TTL) is enabled on a table in Amazon DynamoDB, a background job checks the TTL attribute of items to determine whether they are expired.

    DynamoDB compares the current time, in epoch time format, to the value stored in the user-defined Number attribute of an item. If the attribute's

value is in the epoch time format, is less than the current time, and is not older than 5 years, the item is deleted.

Processing takes place automatically, in the background, and doesn't affect read or write traffic to the table. In addition, deletes performed via TTL are not counted towards capacity units or request units. TTL deletes are available at no additional cost.

For this requirement, the Developer must add an attribute to each item with the expiration time in epoch format and then enable the Time To Live (TTL) feature based on that attribute.

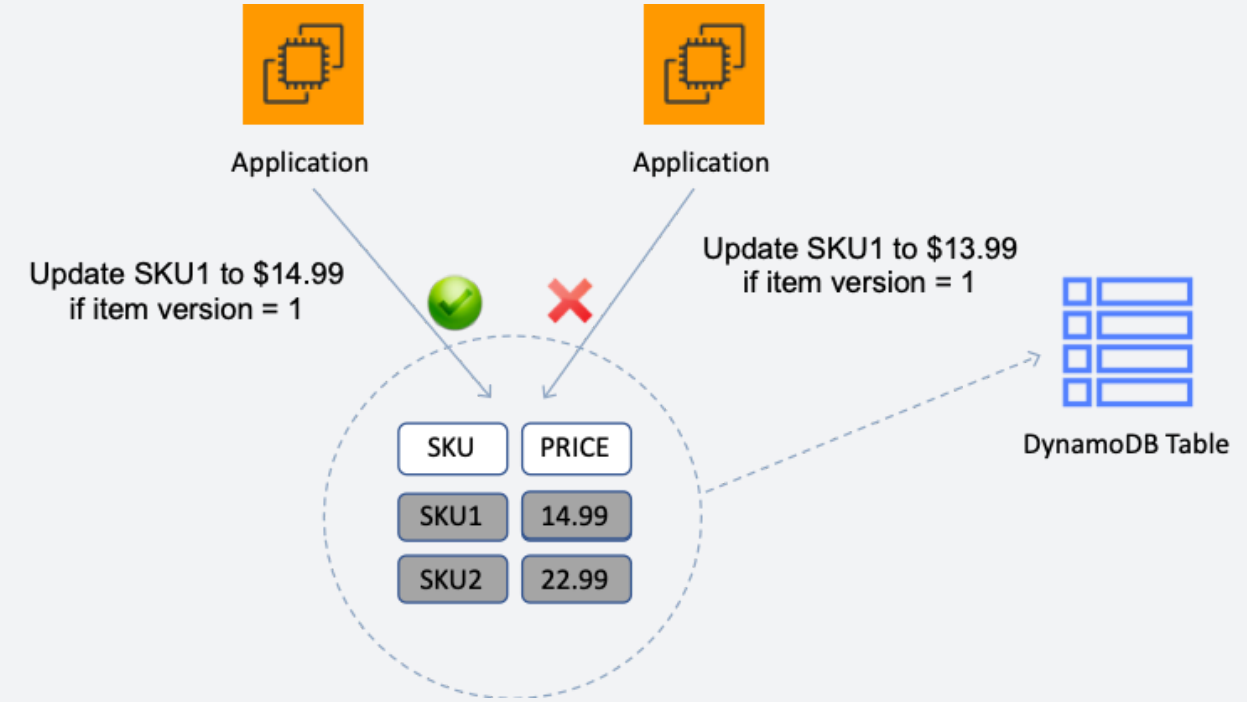
24. **Amazon DynamoDB is integrated with AWS Lambda so that you can create *triggers*—pieces of code that automatically respond to events in DynamoDB Streams. With triggers, you can build applications that react to data modifications in DynamoDB tables.**

    If you enable DynamoDB Streams on a table, you can associate the stream Amazon Resource Name (ARN) with an AWS Lambda function that you write. Immediately after an item in the table is modified, a new record appears in the table's stream. AWS Lambda polls the stream and invokes your Lambda function synchronously when it detects new stream records.

25. **Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. *Optimistic locking* is a strategy to ensure that the client-side item that you are updating (or deleting) is the same as the item in Amazon DynamoDB. If you use this strategy, your database writes are protected from being overwritten by the writes of others, and vice versa.**

    With optimistic locking, each item has an attribute that acts as a version number. If you retrieve an item from a table, the application records the version number of that item. You can update the item, but only if the version number on the server side has not changed. If there is a version mismatch, it means that someone else has modified the item before you did.

    In the diagram below, the application on the left updates an item and increments the version number. Then, the application on the right attempts to update the item but only if the version number is 1.



The update attempt fails, because the application has a stale version of the item. Optimistic locking prevents you from accidentally overwriting changes that were made by others. It also prevents others from accidentally overwriting your changes.

26. **The Developer needs the permissions to retrieve items, update/modify items, and create items. Therefore permissions for the following API actions are required:**

• GetItem - The GetItem operation returns a set of attributes for the item with the given primary key.

• UpdateItem - Edits an existing item's attributes, or adds a new item to the table if it does not already exist. You can put, delete, or add attribute values.

• PutItem - Creates a new item, or replaces an old item with a new item. If an item that has the same primary key as the new item already exists in the specified table, the new item completely replaces the existing item.

27. **Amazon ElastiCache provides fully managed implementations of two popular in-memory data stores – Redis and Memcached. ElastiCache is a**

web service that makes it easy to deploy and run Memcached or Redis protocol-compliant server nodes in the cloud.

The in-memory caching provided by ElastiCache can be used to significantly improve latency and throughput for many read-heavy application workloads or compute-intensive workloads. It is common to use ElastiCache as a cache in front of databases such as Amazon RDS.

The two implementations, Memcached, and Redis, each offer different capabilities and limitations. As you can see from the table below, only Redis supports read replicas and auto-failover:

Feature	Memcached	Redis (cluster mode disabled)	Redis (cluster mode enabled)
Data persistence	No	Yes	Yes
Data types	Simple	Complex	Complex
Data partitioning	Yes	No	Yes
Encryption	No	Yes	Yes
High availability (replication)	No	Yes	Yes
Multi-AZ	Yes, place nodes in multiple AZs. No failover or replication	Yes, with auto-failover. Uses read replicas (0-5 per shard)	Yes, with auto-failover. Uses read replicas (0-5 per shard)
Scaling	Up (node type); out (add nodes)	Single shard (can add replicas)	Add shards
Multithreaded	Yes	No	No
Backup and restore	No (and no snapshots)	Yes, automatic and manual snapshots	Yes, automatic and manual snapshots

28. The Redis implementation must be used if high availability is required, as is necessary for this scenario. Therefore the correct answer is to use Amazon ElastiCache Redis.
29. DynamoDB stores and retrieves data based on a Primary key. There are two types of Primary key:

•Partition key – unique attribute (e.g. user ID).

• Value of the Partition key is input to an internal hash function which determines the partition or physical location on which the data is stored.

• If you are using the Partition key as your Primary key, then no two items can have the same partition key.

Composite key – Partition key + Sort key in combination.

• Example is user posting to a forum. Partition key would be the user ID, Sort key would be the timestamp of the post.

• 2 items may have the same Partition key, but they must have a different Sort key.

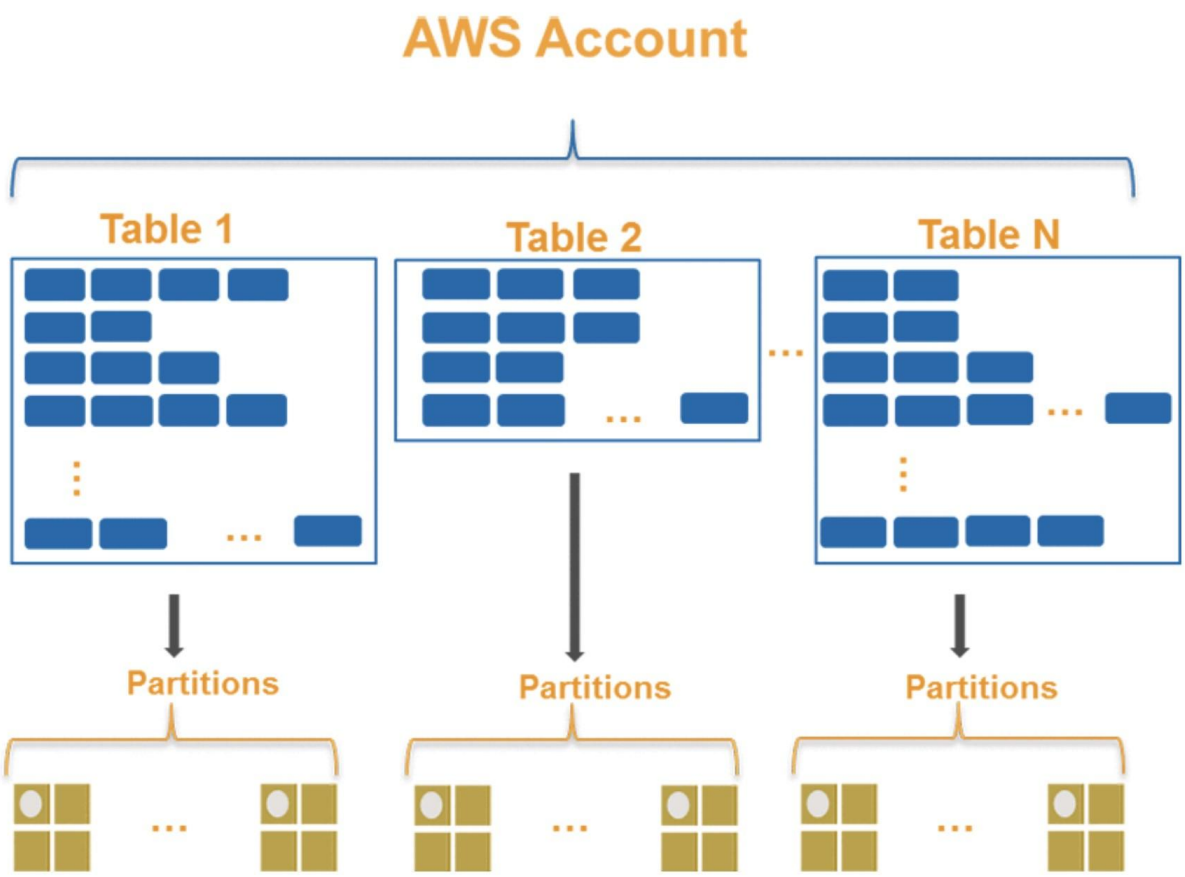
• All items with the same Partition key are stored together, then sorted according to the Sort key value.

• Allows you to store multiple items with the same partition key.

As stated above, if using a partition key alone as per the configuration provided with the question, then you cannot have two items with the same partition key. The only resolution is to recreate the table with a composite key consisting of the userid and timestamp attributes. In that case the Developer will be able to add multiple items with the same userid as long as the timestamp is unique.

30. **DynamoDB stores data as groups of attributes, known as *items*. Items are similar to rows or records in other database systems. DynamoDB stores and retrieves each item based on the primary key value, which must be unique.**

    Items are distributed across 10-GB storage units, called partitions (physical storage internal to DynamoDB). Each table has one or more partitions, as shown in the following illustration.



DynamoDB uses the partition key's value as an input to an internal hash function. The output from the hash function determines the partition in which the item is stored. Each item's location is determined by the hash value of its partition key.

All items with the same partition key are stored together, and for composite partition keys, are ordered by the sort key value. DynamoDB splits partitions by sort key if the collection size grows bigger than 10 GB.

DynamoDB evenly distributes provisioned throughput—read capacity units (RCUs) and write capacity units (WCUs)—among partitions and automatically supports your access patterns using the throughput you have provisioned. However, if your access pattern exceeds 3000 RCU or 1000 WCU for a single partition key value, your requests might be throttled with a ProvisionedThroughputExceededException error.

To avoid request throttling, design your DynamoDB table with the right partition key to meet your access requirements and provide even

distribution of data. Recommendations for doing this include the following:

• Use high cardinality attributes (e.g. email_id, employee_no, customer_id etc.)

• Use composite attributes

• Cache popular items

• Add random numbers or digits from a pre-determined range for write-heavy use cases

In this case there is a hot partition due to the order date being used as the partition key and this is causing writes to be throttled. Therefore, the best solution to ensure the writes are more evenly distributed in this scenario is to add a random number suffix to the partition key values.

31. **Amazon DynamoDB supports identity-based policies only. The best practice method to assign permissions to the table is to create a permissions policy that grants access to the table and assigning that policy to an IAM group that contains the Developer's user accounts.**

    This will provide all users with accounts in the IAM group with the access required to access the DynamoDB table.

32. **The Query operation finds items based on primary key values. You can query any table or secondary index that has a composite primary key (a partition key and a sort key).**

    For items up to 4 KB in size, one RCU equals one strongly consistent read request per second or two eventually consistent read requests per second. Therefore, eventually consistent reads uses fewer RCUs.
    By default, the Scan operation processes data sequentially. Amazon DynamoDB returns data to the application in 1 MB increments, and an application performs additional Scan operations to retrieve the next 1 MB of data.

    The larger the table or index being scanned, the more time the Scan takes to complete. In addition, a sequential Scan might not always be able to fully use the provisioned read throughput capacity: Even though

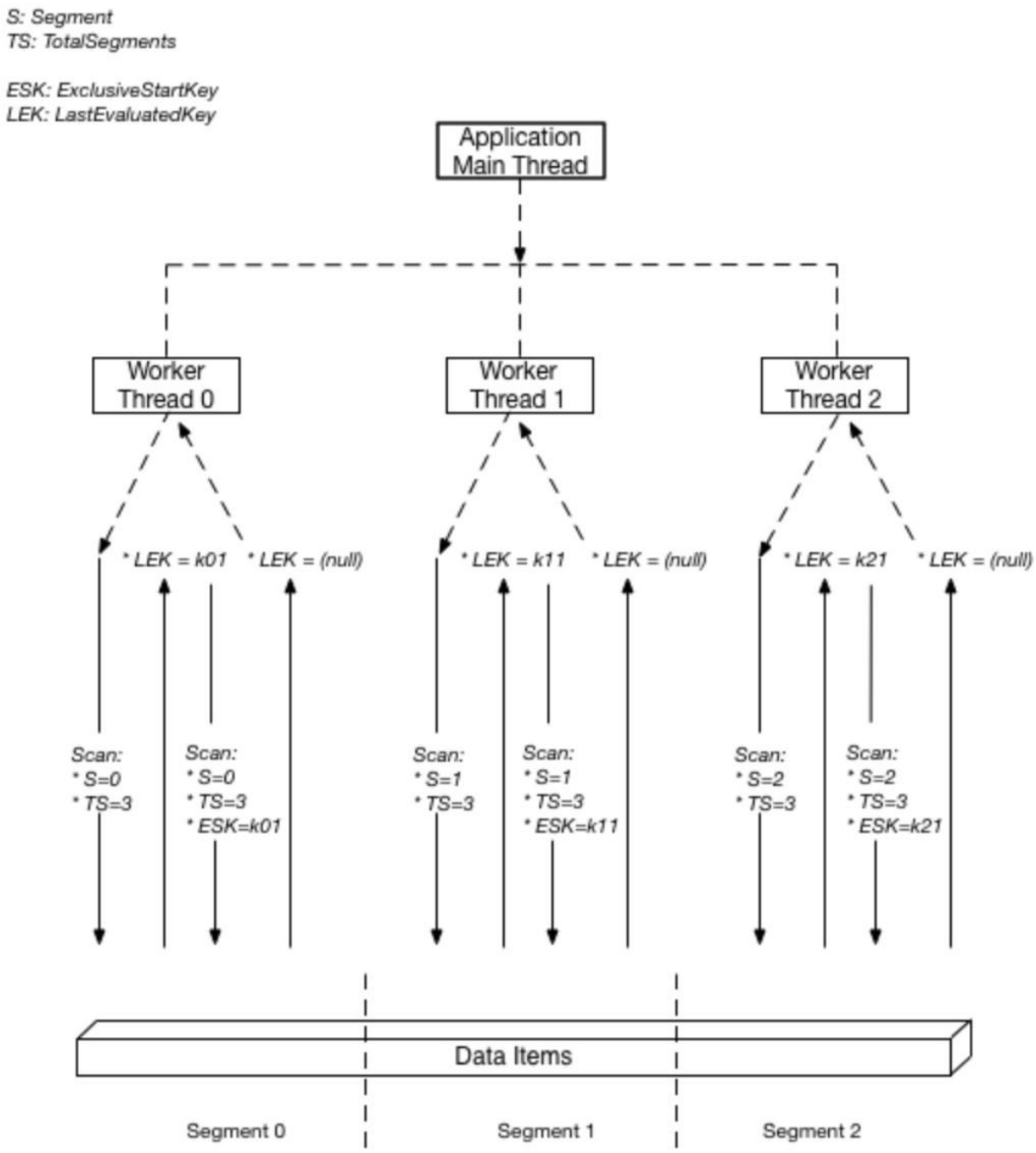
DynamoDB distributes a large table's data across multiple physical partitions, a Scan operation can only read one partition at a time. For this reason, the throughput of a Scan is constrained by the maximum throughput of a single partition.

To address these issues, the Scan operation can logically divide a table or secondary index into multiple *segments*, with multiple application workers scanning the segments in parallel. Each worker can be a thread (in programming languages that support multithreading) or an operating system process. To perform a parallel scan, each worker issues its own Scan request with the following parameters:

• Segment — A segment to be scanned by a particular worker. Each worker should use a different value for Segment.

• TotalSegments — The total number of segments for the parallel scan. This value must be the same as the number of workers that your application will use.

The following diagram shows how a multithreaded application performs a parallel Scan with three degrees of parallelism.



To make the most of your table's provisioned throughput, you'll want to use the Parallel Scan API operation so that your scan is distributed across your table's partitions. However, you also need to ensure the scan doesn't consume your table's provisioned throughput and cause the critical parts of your application to be throttled.

To control the amount of data returned per request, use the Limit parameter. This can help prevent situations where one worker consumes all of the provisioned throughput, at the expense of all other workers.

Therefore, the best solution to this problem is to use a parallel scan API operation with the Limit parameter.

33. The `GetItem` operation returns a set of attributes for the item with the given primary key. If there is no matching item, `GetItem` does not return any data and there will be no Item element in the response.

    `PutItem` creates a new item or replaces an old item with a new item. If an item that has the same primary key as the new item already exists in the specified table, the new item completely replaces the existing item.

    `UpdateItem` edits an existing item's attributes or adds a new item to the table if it does not already exist. You can put, delete, or add attribute values. You can also perform a conditional update on an existing item (insert a new attribute name-value pair if it doesn't exist or replace an existing name-value pair if it has certain expected attribute values).

34. The BatchGetItem operation returns the attributes of one or more items from one or more tables. You identify requested items by primary key.

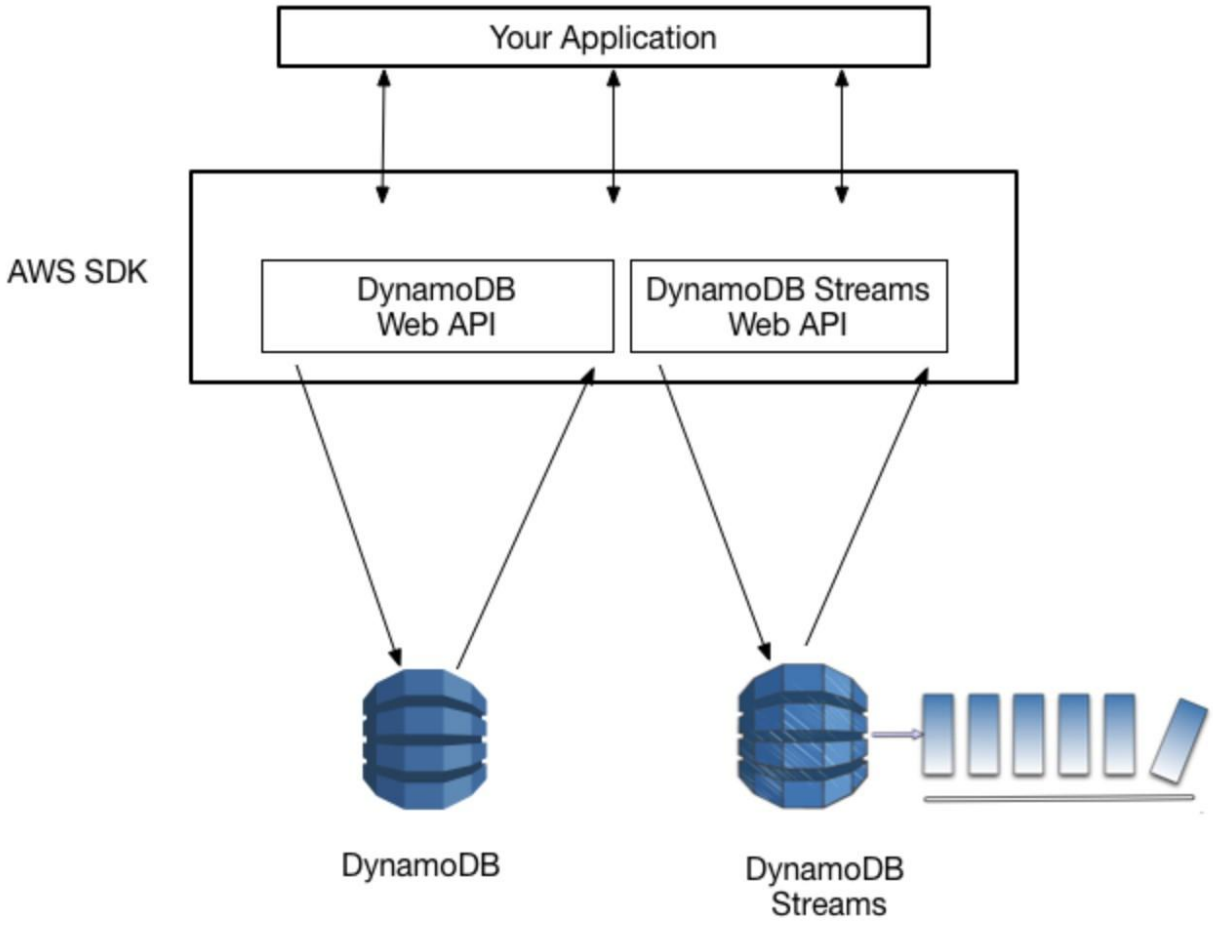
    A single operation can retrieve up to 16 MB of data, which can contain as many as 100 items. In order to minimize response latency, BatchGetItem retrieves items in parallel.

    By default, BatchGetItem performs eventually consistent reads on every table in the request. If you want strongly consistent reads instead, you can set ConsistentRead to true for any or all tables.

35. A *DynamoDB stream* is an ordered flow of information about changes to items in a DynamoDB table. When you enable a stream on a table, DynamoDB captures information about every modification to data items in the table.

    Whenever an application creates, updates, or deletes items in the table, DynamoDB Streams writes a stream record with the primary key attributes of the items that were modified. A *stream record* contains information about a data modification to a single item in a DynamoDB

table. You can configure the stream so that the stream records capture additional information, such as the "before" and "after" images of modified items.



This is the best way to capture a record of new changes made to the DynamoDB table. Another table can then be populated with this data so the data is stored persistently.

36. With Amazon RDS, you can create a MariaDB, MySQL, Oracle, or PostgreSQL read replica in a different AWS Region than the source DB instance. Creating a cross-Region read replica isn't supported for SQL Server on Amazon RDS.

    You create a read replica in a different AWS Region to do the following:

    • Improve your disaster recovery capabilities.

    • Scale read operations into an AWS Region closer to your users.

• Make it easier to migrate from a data center in one AWS Region to a data center in another AWS Region.

Creating a read replica in a different AWS Region from the source instance is similar to creating a replica in the same AWS Region. You can use the AWS Management Console, run the create-db-instance-read-replica command, or call the CreateDBInstanceReadReplica API operation.

Creating read replica in the region where the reporting application is going to run will provide the best performance as latency will be much lower than connecting across regions. As the database is a replica it will also be continuously updated using asynchronous replication so the reporting application will have the latest data available.

37. With ElastiCache the company can move the session data to a high-performance, in-memory data store that is well suited to this use case. This will provide high availability for the session data in the case of EC2 instance failure and will reduce downtime for users.
38. The key requirements are to add high availability and fault tolerance to the application. To do this the Developer should put the website into an Auto Scaling group of EC2 instances across multiple AZs. An Elastic Load Balancer can be deployed in front of the EC2 instances to distribute incoming connections. This solution is highly available and fault tolerant.

    For the MySQL database the Developer should use Amazon RDS with the MySQL engine. To provide fault tolerance the Developer should configure Amazon RDS as a Multi-AZ deployment which will create a standby instance in another AZ that can be failed over to.
39. The Developer needs to be able to sort the scores for each sport and then extract the highest performing athletes. In this case BOTH the partition key and sort key must be different which means a Global Secondary index is required (as a Local Secondary index only has a different sort key). The GSI would be configured as follows:

    · Partition key: sport_name

    · Sort Key: score

    The results will then be listed in order of the highest score for each sport which is exactly what is required.

40. **In DynamoDB, you have the option to specify conditions when granting permissions using an IAM policy. For example, you can:**

    • Grant permissions to allow users read-only access to certain items and attributes in a table or a secondary index.

    • Grant permissions to allow users write-only access to certain attributes in a table, based upon the identity of that user.

    To implement this kind of fine-grained access control, you write an IAM permissions policy that specifies conditions for accessing security credentials and the associated permissions. You then apply the policy to IAM users, groups, or roles that you create using the IAM console. Your IAM policy can restrict access to individual items in a table, access to the attributes in those items, or both at the same time.

    You use the IAM Condition element to implement a fine-grained access control policy. By adding a Condition element to a permissions policy, you can allow or deny access to items and attributes in DynamoDB tables and indexes, based upon your particular business requirements. You can also grant permissions on a table, but restrict access to specific items in that table based on certain primary key values.
41. **There are two caching strategies available: Lazy Loading and Write-Through:**

    Lazy Loading

    Loads the data into the cache only when necessary (if a cache miss occurs).

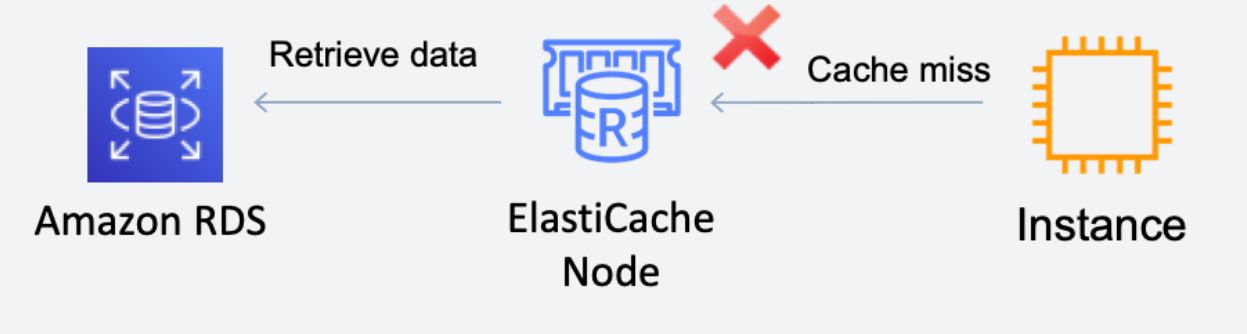
    Lazy loading avoids filling up the cache with data that won't be requested.

    If requested data is in the cache, ElastiCache returns the data to the application.

    If the data is not in the cache or has expired, ElastiCache returns a null.

    The application then fetches the data from the database and writes the data received into the cache so that it is available for next time.

    Data in the cache can become stale if Lazy Loading is implemented without other strategies (such as TTL).



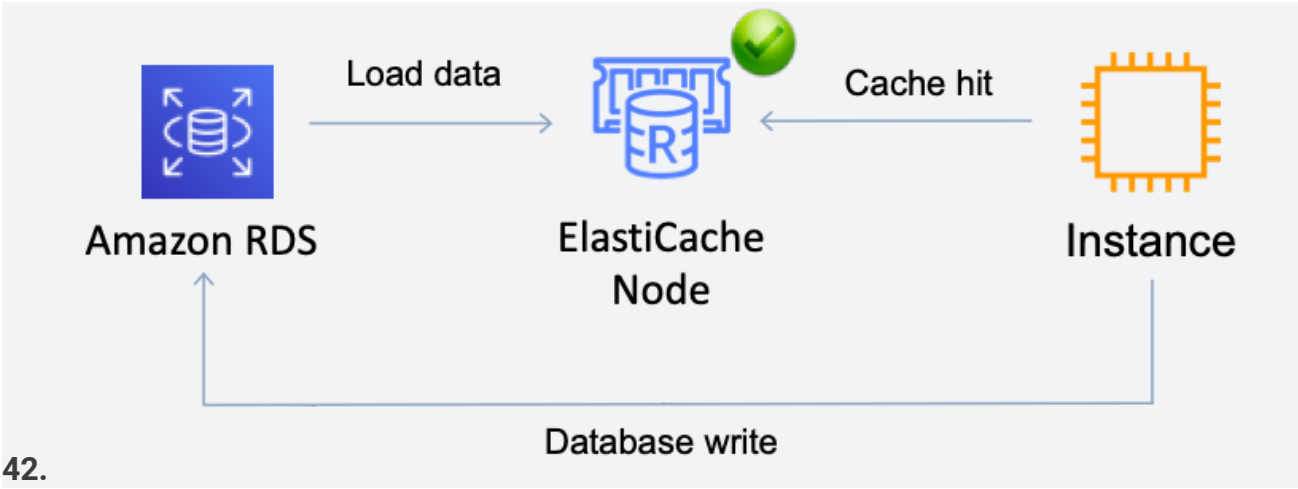
Write Through

When using a write through strategy, the cache is updated whenever a new write or update is made to the underlying database.

Allows cache data to remain up-to-date.

Can add wait time to write operations in your application.

Without a TTL you can end up with a lot of cached data that is never read.



42.

AWS DEVELOPER TOOLS

1. The application specification file (AppSpec file) is a YAML-formatted or JSON-formatted file used by CodeDeploy to manage a deployment.

   If your application uses the Amazon ECS compute platform, the AppSpec file is named appspec.yaml. It is used by CodeDeploy to determine:

Your Amazon ECS task definition file. This is specified with its ARN in the TaskDefinition instruction in the AppSpec file.

The container and port in your replacement task set where your Application Load Balancer or Network Load Balancer reroutes traffic during a deployment. This is specified with the LoadBalancerInfo instruction in the AppSpec file.

Optional information about your Amazon ECS service, such the platform version on which it runs, its subnets, and its security groups.

Optional Lambda functions to run during hooks that correspond with lifecycle events during an Amazon ECS deployment. For more information, see AppSpec 'hooks' Section for an Amazon ECS Deployment.

2. AWS X-Ray is a service that collects data about requests that your application serves, and provides tools you can use to view, filter, and gain insights into that data to identify issues and opportunities for optimization. For any traced request to your application, you can see detailed information not only about the request and response, but also about calls that your application makes to downstream AWS resources, microservices, databases and HTTP web APIs.

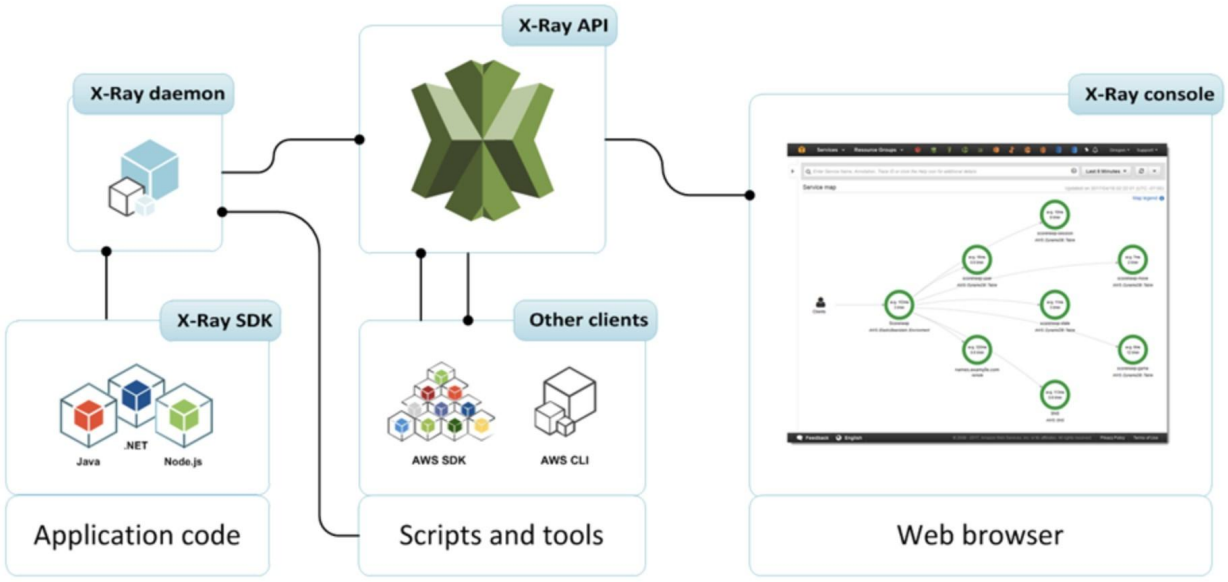
   - You can run the X-Ray daemon on the following operating systems on Amazon EC2:

   - Amazon Linux

   - Ubuntu

   Windows Server (2012 R2 and newer)

   The X-Ray daemon must be running on the EC2 instance in order to collect data. You can use a user data script to run the daemon automatically when you launch the instance. The X-Ray daemon uses the AWS SDK to upload trace data to X-Ray, and it needs AWS credentials with permission to do that.



On Amazon EC2, the daemon uses the instance's instance profile role automatically. The IAM role or user that the daemon's credentials belong to must have permission to write data to the service on your behalf.

- To use the daemon on Amazon EC2, create a new instance profile role or add the managed policy to an existing one.

- To use the daemon on Elastic Beanstalk, add the managed policy to the Elastic Beanstalk default instance profile role.

- To run the daemon locally, create an IAM user and save its access keys on your computer.

Therefore, the most likely cause of the issues being experienced in this scenario is that the instance's instance profile role does not have permission to upload trace data to X-Ray or the X-Ray daemon is not running on the EC2 instance.

3. AWS CodePipeline is a continuous delivery service you can use to model, visualize, and automate the steps required to release your software. You can quickly model and configure the different stages of a software release process. CodePipeline automates the steps required to release your software changes continuously.

   Specifically, you can:

   Automate your release processes: CodePipeline fully automates your release process from end to end, starting from your source repository through build, test,

and deployment. You can prevent changes from moving through a pipeline by including a manual approval action in any stage except a Source stage. You can release when you want, in the way you want, on the systems of your choice, across one instance or multiple instances.

Establish a consistent release process: Define a consistent set of steps for every code change. CodePipeline runs each stage of your release according to your criteria.

Speed up delivery while improving quality: You can automate your release process to allow your developers to test and release code incrementally and speed up the release of new features to your customers.

Use your favorite tools: You can incorporate your existing source, build, and deployment tools into your pipeline.

View progress at a glance: You can review real-time status of your pipelines, check the details of any alerts, retry failed actions, view details about the source revisions used in the latest pipeline execution in each stage, and manually rerun any pipeline.

View pipeline history details: You can view details about executions of a pipeline, including start and end times, run duration, and execution IDs.

Therefore, AWS CodePipeline is the perfect tool for the Developer's requirements.

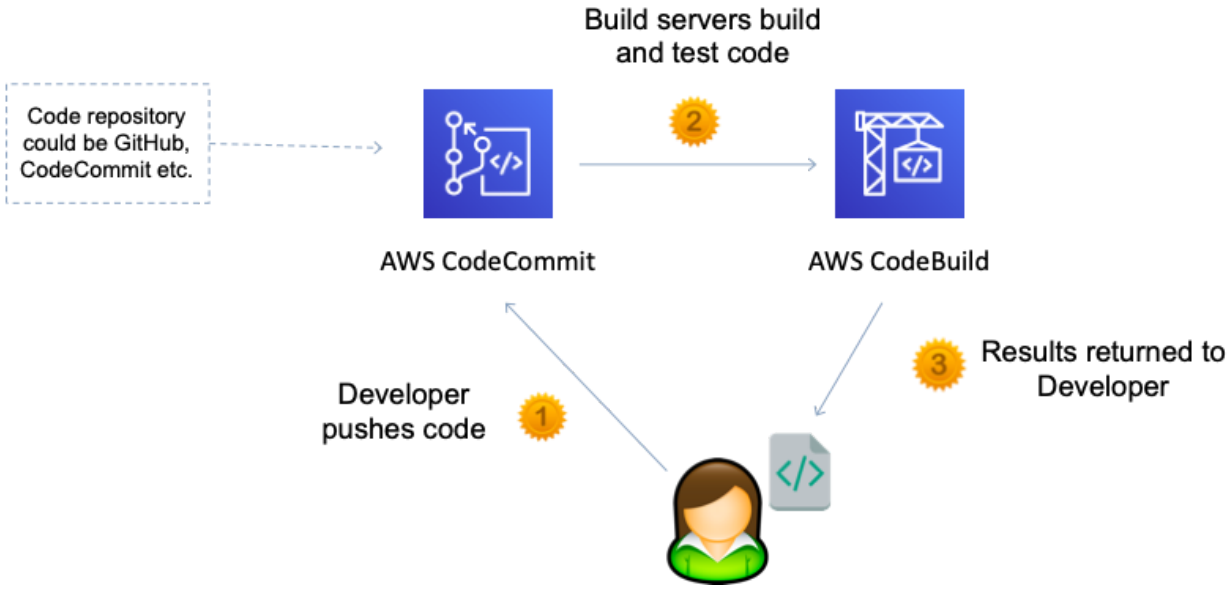
4. A **segment document** conveys information about a segment to X-Ray. A segment document can be up to 64 kB and contain a whole segment with subsegments, a fragment of a segment that indicates that a request is in progress, or a single subsegment that is sent separately. You can send segment documents directly to X-Ray by using the PutTraceSegments API.

   Example minimally complete segment:

```
{
"name" : "example.com",
"id" : "70de5b6f19ff9a0a",
"start_time" : 1.478293361271E9,
"trace_id" : "1-581cf771-a006649127e371903a2de979",
"end_time" : 1.478293361449E9
}
```

A subset of segment fields are indexed by X-Ray for use with filter expressions. For example, if you set the user field on a segment to a unique identifier, you can search for segments associated with specific users in the X-Ray console or by using the `GetTraceSummaries` API.

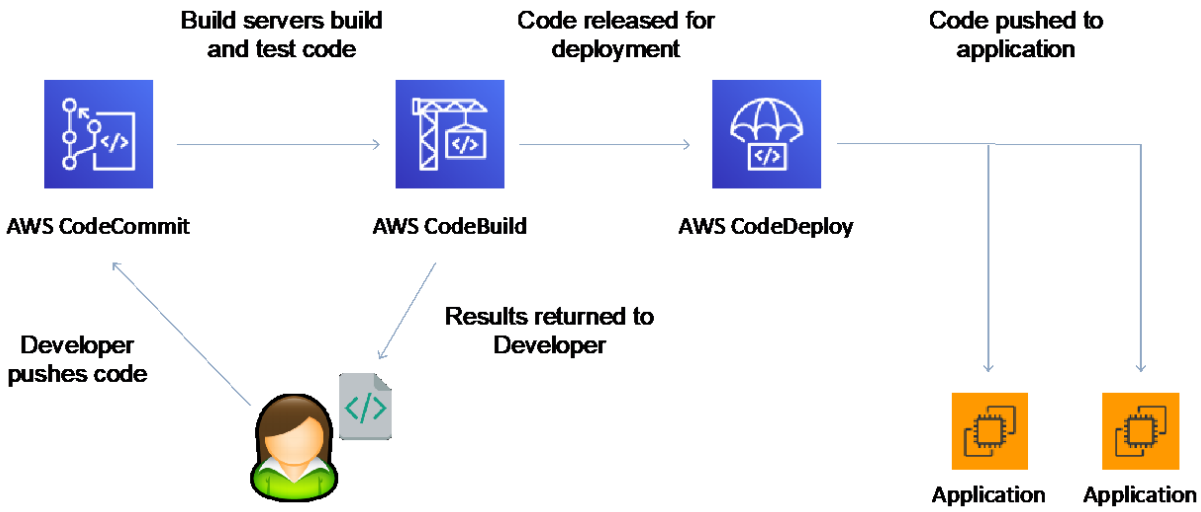
5. AWS CodeCommit is a version control service hosted by Amazon Web Services that you can use to privately store and manage assets (such as documents, source code, and binary files) in the cloud.
6. =



CodeCommit is optimized for team software development. It manages batches of changes across multiple files, which can occur in parallel with changes made by other developers.

7. AWS CodeBuild is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild eliminates the need to provision, manage, and scale your own build servers. It provides pre-packaged build environments for popular programming languages and build tools such as Apache Maven, Gradle, and more.

   CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.



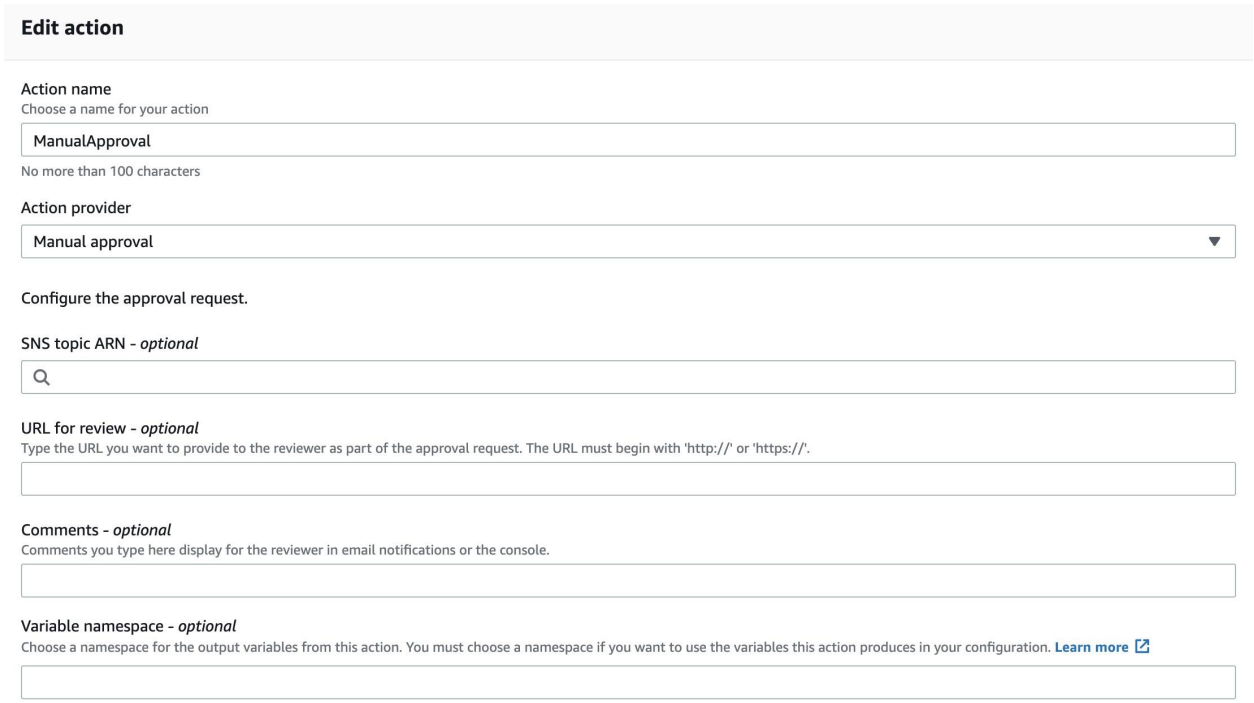
8. You can record additional information about requests, the environment, or your application with annotations and metadata. You can add annotations and metadata to the segments that the X-Ray SDK creates, or to custom subsegments that you create.

   Annotations are key-value pairs with string, number, or Boolean values. Annotations are indexed for use with filter expressions. Use annotations to record data that you want to use to group traces in the console, or when calling the GetTraceSummaries API.

   Metadata are key-value pairs that can have values of any type, including objects and lists, but are not indexed for use with filter expressions. Use metadata to record additional data that you want stored in the trace but don't need to use with search.

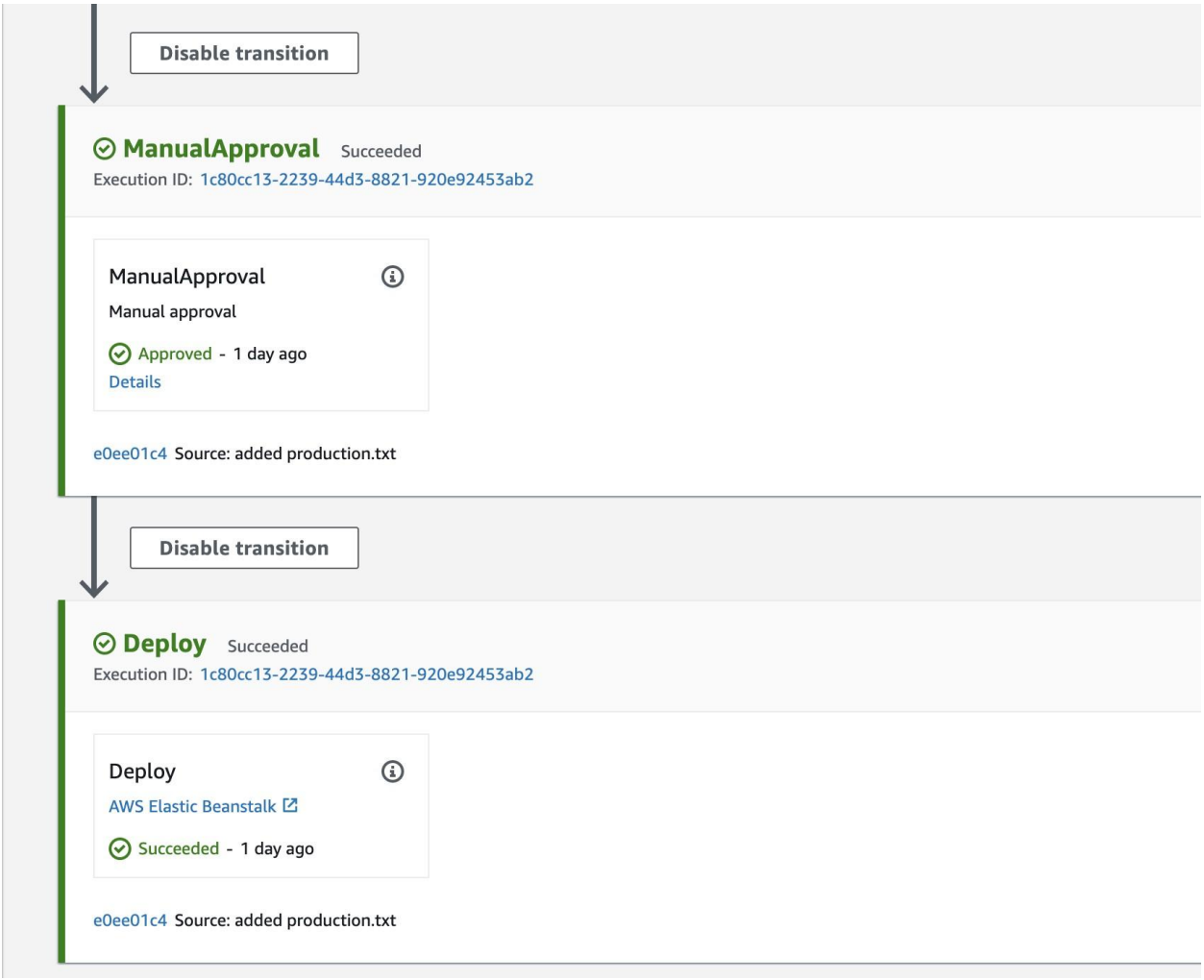
   Annotations can be used with filter expressions, so this is the best solution for this requirement. The Developer can add annotations to the custom subsegments and will then be able to use filter expressions to filter the results in AWS X-Ray.

9. In AWS CodePipeline, you can add an approval action to a stage in a pipeline at the point where you want the pipeline execution to stop so that someone with the required AWS Identity and Access Management permissions can approve or reject the action.



If the action is approved, the pipeline execution resumes. If the action is rejected—or if no one approves or rejects the action within seven days of the pipeline reaching the action and stopping—the result is the same as an action failing, and the pipeline execution does not continue.

10. In this scenario, the manual approval stage would be placed in the pipeline before the deployment stage that deploys the application update into production:



Therefore, the best answer is to use an approval action in a stage before deployment to production

11. If you use AWS SAM to create your serverless application, it comes built-in with CodeDeploy to provide gradual Lambda deployments. With just a few lines of configuration, AWS SAM does the following for you:

• Deploys new versions of your Lambda function, and automatically creates aliases that point to the new version.

• Gradually shifts customer traffic to the new version until you're satisfied that it's working as expected, or you roll back the update.

• Defines pre-traffic and post-traffic test functions to verify that the newly deployed code is configured correctly and your application operates as expected.

• Rolls back the deployment if CloudWatch alarms are triggered.

There are several options for how CodeDeploy shifts traffic to the new Lambda version. You can choose from the following:

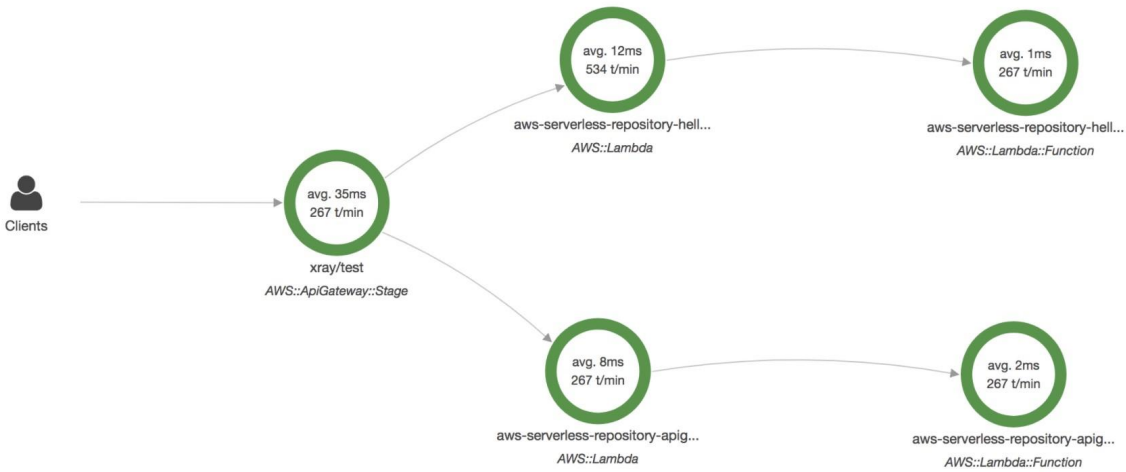
• **Canary**: Traffic is shifted in two increments. You can choose from predefined canary options. The options specify the percentage of traffic that's shifted to your updated Lambda function version in the first increment, and the interval, in minutes, before the remaining traffic is shifted in the second increment.

• **Linear**: Traffic is shifted in equal increments with an equal number of minutes between each increment. You can choose from predefined linear options that specify the percentage of traffic that's shifted in each increment and the number of minutes between each increment.

All-at-once: All traffic is shifted from the original Lambda function to the updated Lambda function version at once.

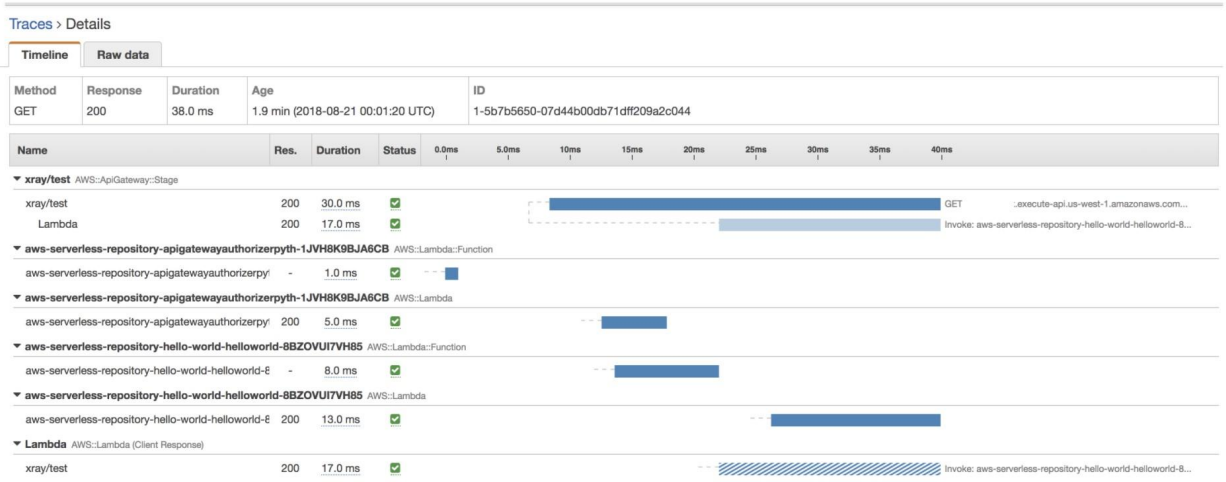
Therefore `CodeDeployDefault.LambdaCanary10Percent5Minutes` is the best answer as this will shift 10 percent of the traffic and then after 5 minutes shift the remainder of the traffic. The entire deployment will take 5 minutes to cut over.

12. You can use AWS X-Ray to trace and analyze user requests as they travel through your Amazon API Gateway APIs to the underlying services. API Gateway supports X-Ray tracing for all API Gateway endpoint types: Regional, edge-optimized, and private. You can use X-Ray with Amazon API Gateway in all AWS Regions where X-Ray is available.



Because X-Ray gives you an end-to-end view of an entire request, you can analyze latencies in your APIs and their backend services. You can use an X-Ray service map to view the latency of an entire request and that of the downstream services that are integrated with X-Ray. You can also configure sampling rules to tell X-Ray which requests to record and at what sampling rates, according to criteria that you specify.

The following diagram shows a trace view generated for the example API described above, with a Lambda backend function and a Lambda authorizer function. A successful API method request is shown with a response code of 200.



13.
14. CodeCommit is a secure, highly scalable, managed source control service that hosts private Git repositories. CodeCommit eliminates the need for you to manage your own source control system or worry about scaling its infrastructure.

    You can use CodeCommit to store anything from code to binaries. It supports the standard functionality of Git, so it works seamlessly with your existing Git-based tools.

    With CodeCommit, you can:

    • **Benefit from a fully managed service hosted by AWS**. CodeCommit provides high service availability and durability and eliminates the administrative overhead of managing your own hardware and software. There is no hardware to provision and scale and no server software to install, configure, and update.

    • **Store your code securely**. CodeCommit repositories are encrypted at rest as well as in transit.

• **Work collaboratively on code.** CodeCommit repositories support pull requests, where users can review and comment on each other's code changes before merging them to branches; notifications that automatically send emails to users about pull requests and comments; and more.

• **Easily scale your version control projects**. CodeCommit repositories can scale up to meet your development needs. The service can handle repositories with large numbers of files or branches, large file sizes, and lengthy revision histories.

• **Store anything, anytime**. CodeCommit has no limit on the size of your repositories or on the file types you can store.

• **Integrate with other AWS and third-party services**. CodeCommit keeps your repositories close to your other production resources in the AWS Cloud, which helps increase the speed and frequency of your development lifecycle. It is integrated with IAM and can be used with other AWS services and in parallel with other repositories. **Easily migrate files from other remote repositories**. You can migrate to CodeCommit from any Git-based repository.

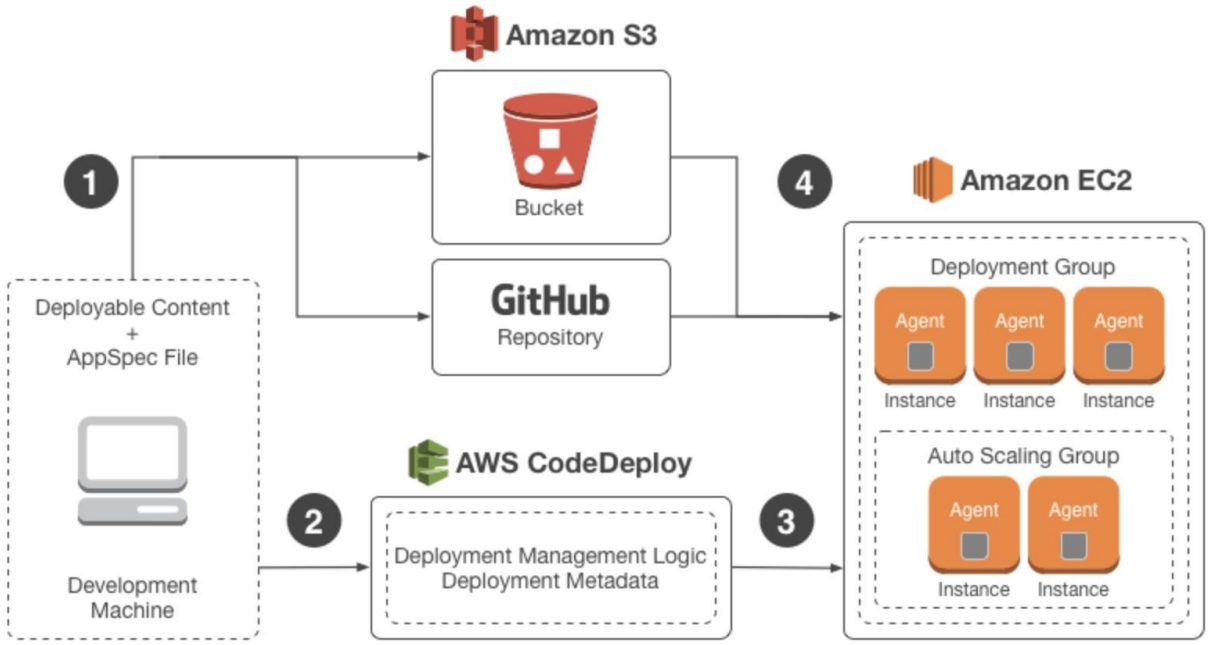
• **Use the Git tools you already know**. CodeCommit supports Git commands as well as its own AWS CLI commands and APIs.

Therefore, the development team should select AWS CodeCommit as the repository they use for storing code related to the new project.

15. CodeDeploy is a deployment service that automates application deployments to Amazon EC2 instances, on-premises instances, serverless Lambda functions, or Amazon ECS services.

    CodeDeploy can deploy application content that runs on a server and is stored in Amazon S3 buckets, GitHub repositories, or Bitbucket repositories. CodeDeploy can also deploy a serverless Lambda function. You do not need to make changes to your existing code before you can use CodeDeploy.

    The image below shows the flow of a typical CodeDeploy in-place deployment.



The above deployment could also be directed at on-premises servers. Therefore, the best answer is to use AWS CodeDeploy to deploy the software package to both EC2 instances and on-premises virtual servers.

16. The permissions assigned to the user account are missing the privileges to create and delete branches in AWS CodeCommit. The Developer needs to be assigned these permissions but according to the principal of least privilege it's important to ensure no additional permissions are assigned.

The following API actions can be used to work with branches:

• CreateBranch , which creates a branch in a specified repository.

• DeleteBranch , which deletes the specified branch in a repository unless it is the default branch.

• GetBranch , which returns information about a specified branch.

• ListBranches , which lists all branches for a specified repository.

• UpdateDefaultBranch , which changes the default branch for a repository.

Therefore, the best answer is to add the "`codecommit:CreateBranch`" and "`codecommit:DeleteBranch`" permissions to the permissions policy.

17. AWS CodeBuild is a fully managed build service in the cloud. CodeBuild compiles your source code, runs unit tests, and produces artifacts that are ready to deploy. CodeBuild eliminates the need to provision, manage, and scale your own build servers. It provides prepackaged build environments for popular programming languages and build tools such as Apache Maven, Gradle, and more.

    You can also customize build environments in CodeBuild to use your own build tools. CodeBuild scales automatically to meet peak build requests.

    CodeBuild provides these benefits:

    • **Fully managed** – CodeBuild eliminates the need to set up, patch, update, and manage your own build servers.

    • **On demand** – CodeBuild scales on demand to meet your build needs. You pay only for the number of build minutes you consume.

    • **Out of the box** – CodeBuild provides preconfigured build environments for the most popular programming languages. All you need to do is point to your build script to start your first build.

    Therefore, AWS CodeBuild is the best service to use to compile the Development team's source code, run tests, and produce software packages that are ready for deployment.

18. You can send trace data to X-Ray in the form of segment documents. A segment document is a JSON formatted string that contains information about the work that your application does in service of a request. Your application can record data about the work that it does itself in segments, or work that uses downstream services and resources in subsegments.

    Segments record information about the work that your application does. A segment, at a minimum, records the time spent on a task, a name, and two IDs. The trace ID tracks the request as it travels between services. The segment ID tracks the work done for the request by a single service.

    Example Minimal complete segment:

```
{
  "name" : "Scorekeep",
  "id" : "70de5b6f19ff9a0a",
  "start_time" : 1.478293361271E9,
  "trace_id" : "1-581cf771-a006649127e371903a2de979",
  "end_time" : 1.478293361449E9
}
```

You can upload segment documents with the `PutTraceSegments` API. The API has a single parameter, `TraceSegmentDocuments`, that takes a list of JSON segment documents.

Therefore, the Developer should use the `PutTraceSegments` API action.

19. AWS CodeStar enables you to quickly develop, build, and deploy applications on AWS. AWS CodeStar provides a unified user interface, enabling you to easily manage your software development activities in one place. With AWS CodeStar, you can set up your entire continuous delivery toolchain in minutes, allowing you to start releasing code faster. AWS CodeStar makes it easy for your whole team to work together securely, allowing you to easily manage access and add owners, contributors, and viewers to your projects.

    Each AWS CodeStar project comes with a project management dashboard, including an integrated issue tracking capability powered by Atlassian JIRA Software. With the AWS CodeStar project dashboard, you can easily track progress across your entire software development process, from your backlog of work items to teams' recent code deployments.

20. The Developer can use the AWS SDK to instantiate a CodeCommit client. For instance, the code might include the following:

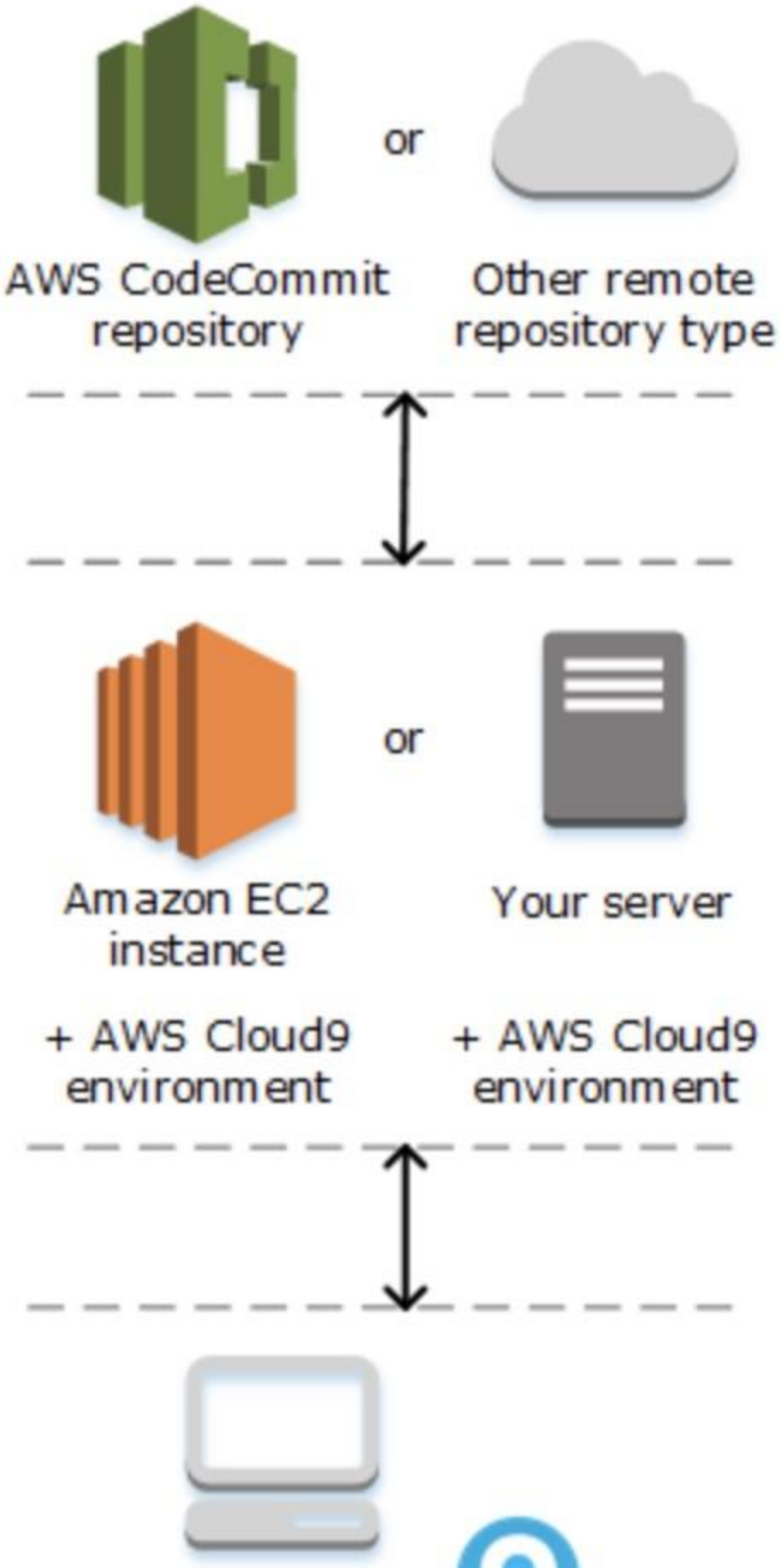
```
import boto3

client = boto3.client('codecommit')
```

The client can then be used with put_file which adds or updates a file in a branch in an AWS CodeCommit repository, and generates a commit for the addition in the specified branch.

The request syntax is as follows:

```
response = client.put_file(
    repositoryName='string',
    branchName='string',
    fileContent=b'bytes',
    filePath='string',
    fileMode='EXECUTABLE'|'NORMAL'|'SYMLINK',
    parentCommitId='string',
    commitMessage='string',
    name='string',
    email='string'
)
```

21. AWS Cloud9 is an integrated development environment, or *IDE*. The AWS Cloud9 IDE offers a rich code-editing experience with support for several programming languages and runtime debuggers, and a built-in terminal. It contains a collection of tools that you use to code, build, run, test, and debug software, and helps you release software to the cloud.


or
AWS CodeCommit repository
Other remote repository type
or
Amazon EC2 instance
+ AWS Cloud9 environment
Your server
+ AWS Cloud9 environment

You access the AWS Cloud9 IDE through a web browser. You can configure the IDE to your preferences. You can switch color themes, bind shortcut keys, enable programming language-specific syntax coloring and code formatting, and more.

AWS COMPUTE

1. AWS Elastic Beanstalk provides several options for how deployments are processed, including deployment policies (**All at once**, **Rolling**, **Rolling with additional batch**, and **Immutable**) and options that let you configure batch size and health check behavior during deployments.

   For this scenario we need to ensure that no downtime occurs if the update fails and there is a quick way to rollback. In the table below you can see the different deployment policies available and how they impact downtime and rollback:

Deployment Policy	Deploy Time	Zero Downtime	Rollback	Extra Cost	Reduction in capacity
All at once	🕓	NO	Manual redeploy	NONE	YES (total)
Rolling	🕓🕓	YES	Manual redeploy	NONE	YES (batch size)
Rolling with additional batch	🕓🕓🕓	YES	Manual redeploy	YES (batch size)	NO
Immutable	🕓🕓🕓🕓	YES	Terminate new instances	YES (total)	NO
Blue/green	🕓🕓🕓🕓	YES	Swap URL	YES (varies)	NO

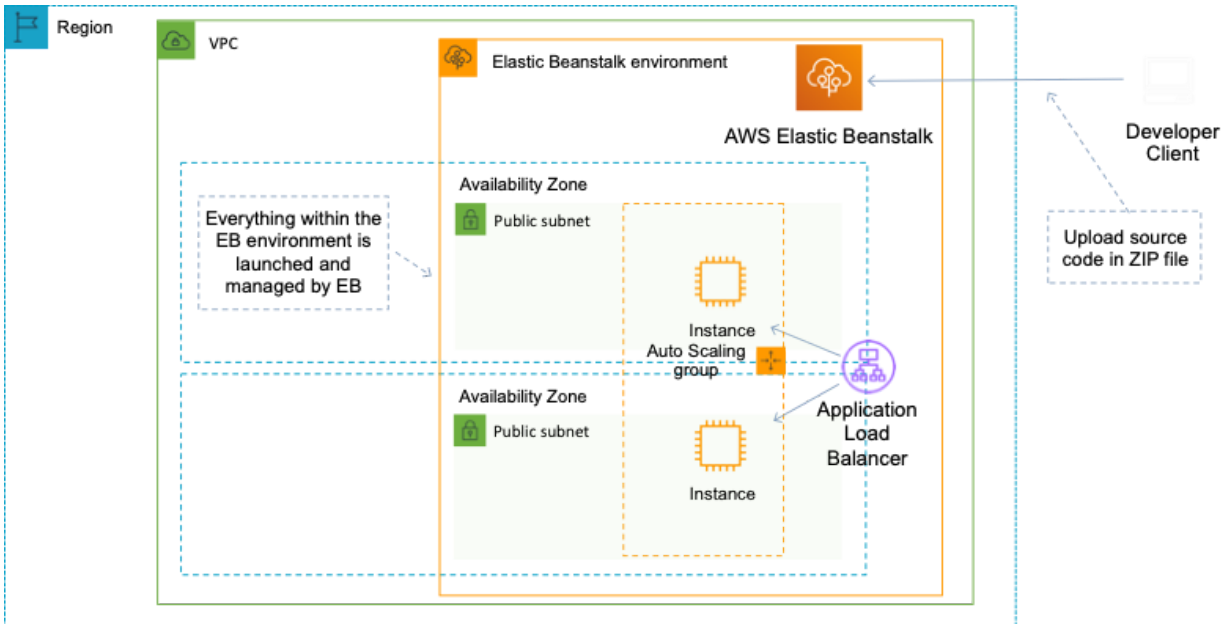
   All policies except for Immutable and Blue/Green require manual redeployment of the previous version of the code which will take time and result in downtime. The blue/green option is not actually an Elastic Beanstalk policy but it is a method you can use, however it is not offered as an answer choice

   Therefore, the best deployment policy to use for this scenario is the **Immutable** deployment policy.
2. If you would like to run a Docker image that is available in Amazon ECR, you can pull it to your local environment with the docker pull command. You can do this

from either your default registry or from a registry associated with another AWS account.

Docker CLI does not support standard AWS authentication methods, so client authentication must be handled so that ECR knows who is requesting to push or pull an image. To do this you can issue the aws `ecr get-login` or aws `ecr get-login-password` (AWS CLI v2) and then use the output to login using `docker login` and then issue a `docker pull` command specifying the repository and image with the REPOSITORY URI : TAG format.

3. The Developers do not want to manage the infrastructure so the best AWS service for them to use to create a website for a development environment is AWS Elastic Beanstalk. This will allow the Developers to simply upload their Node.js code to Elastic Beanstalk and it will handle the provisioning and management of the underlying infrastructure.



AWS Elastic Beanstalk can be used to quickly deploy and manage applications in the AWS Cloud. Developers upload applications and Elastic Beanstalk handles the deployment details of capacity provisioning, load balancing, auto-scaling, and application health monitoring. AWS Elastic Beanstalk leverages Elastic Load Balancing and Auto Scaling to automatically scale your application in and out based on your application's specific needs.

4. Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high

resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

User activity is not a standard CloudWatch metric and as stated above for the resolution we need in this scenario a custom CloudWatch metric is required anyway. Therefore, for this scenario the Developer should create a high-resolution custom Amazon CloudWatch metric for user activity data and publish the data every 10 seconds.

5. You can use the Amazon EC2 console to determine the private IPv4 addresses, public IPv4 addresses, and Elastic IP addresses of your instances. You can also determine the public IPv4 and private IPv4 addresses of your instance from within your instance by using instance metadata.

   You can access the instance metadata by using the following URL: http://169.254.169.254/latest/meta-data/

6. Cluster queries are expressions that enable you to group objects. For example, you can group container instances by attributes such as Availability Zone, instance type, or custom metadata.

   After you have defined a group of container instances, you can customize Amazon ECS to place tasks on container instances based on group. You can also apply a group filter when listing container instances.

   As an example, the following cluster query expressions selects instances with the specified instance type:

   attribute:ecs.instance-type == t2.small

   The following cluster query expressions selects instances in the us-east-1a or us-east-1b availability zones:

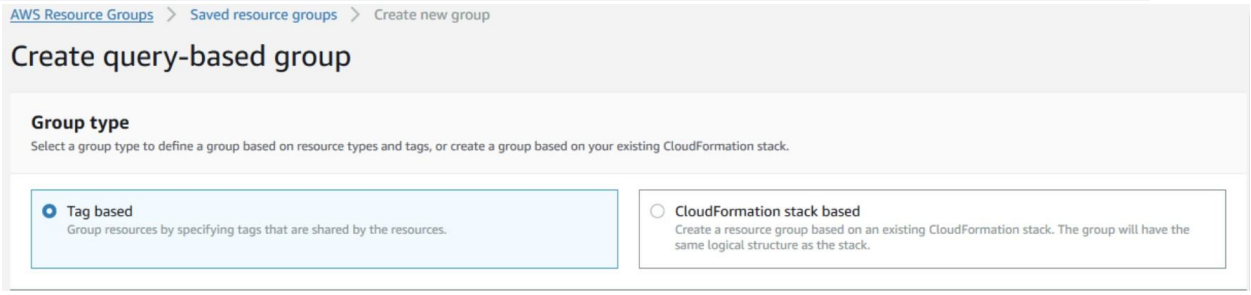
   attribute:ecs.availability-zone in [us-east-1a, us-east-1b]

   The Developer should therefore use the cluster query language to generate expressions that can group the container instances by instance type and availability zone.

7. A serverless application can include one or more **nested applications**. You can deploy a nested application as a stand-alone artifact or as a component of a larger application.

   As serverless architectures grow, common patterns emerge in which the same components are defined in multiple application templates. You can now separate out common patterns as dedicated applications, and then nest them as part of new or existing application templates. With nested applications, you can stay more focused on the business logic that's unique to your application.

   To define a nested application in your serverless application, use the AWS::Serverless::Application resource type.
8. In AWS, a *resource* is an entity that you can work with. Examples include an Amazon EC2 instance, an AWS CloudFormation stack, or an Amazon S3 bucket. If you work with multiple resources, you might find it useful to manage them as a group rather than move from one AWS service to another for each task.



   By default, the AWS Management Console is organized by AWS service. But with Resource Groups, you can create a custom console that organizes and consolidates information based on criteria specified in tags, or the resources in an AWS CloudFormation stack. The following list describes some of the cases in which resource grouping can help organize your resources.

   An application that has different phases, such as development, staging, and production.

   Projects managed by multiple departments or individuals.

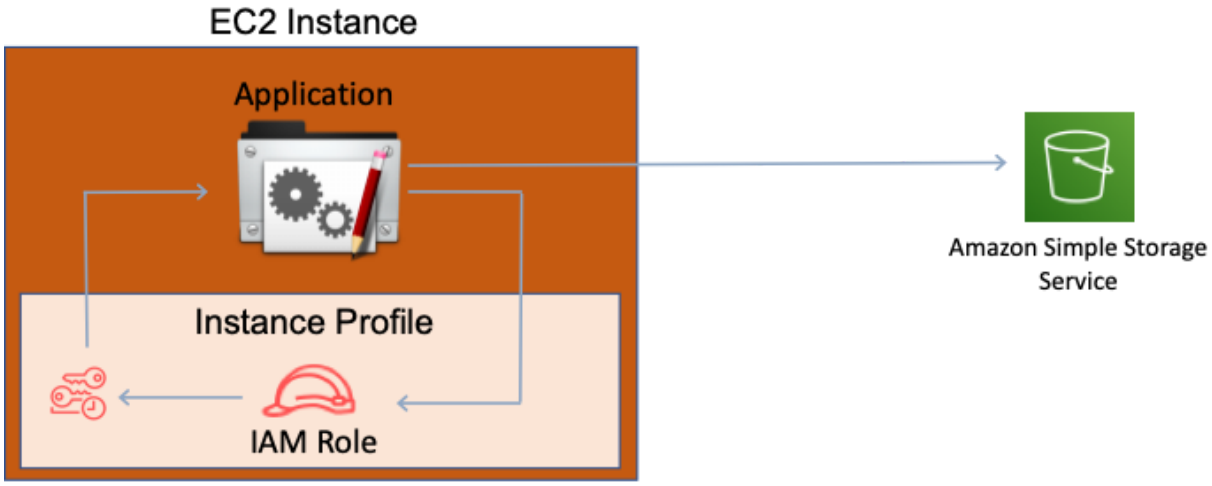
   A set of AWS resources that you use together for a common project or that you want to manage or monitor as a group.

   A set of resources related to applications that run on a specific platform, such as Android or iOS.
9. When you launch an instance in Amazon EC2, you have the option of passing user data to the instance that can be used to perform common automated

configuration tasks and even run scripts after the instance starts. You can pass two types of user data to Amazon EC2: shell scripts and cloud-init directives. You can also pass this data into the launch wizard as plain text, as a file (this is useful for launching instances using the command line tools), or as base64-encoded text (for API calls).

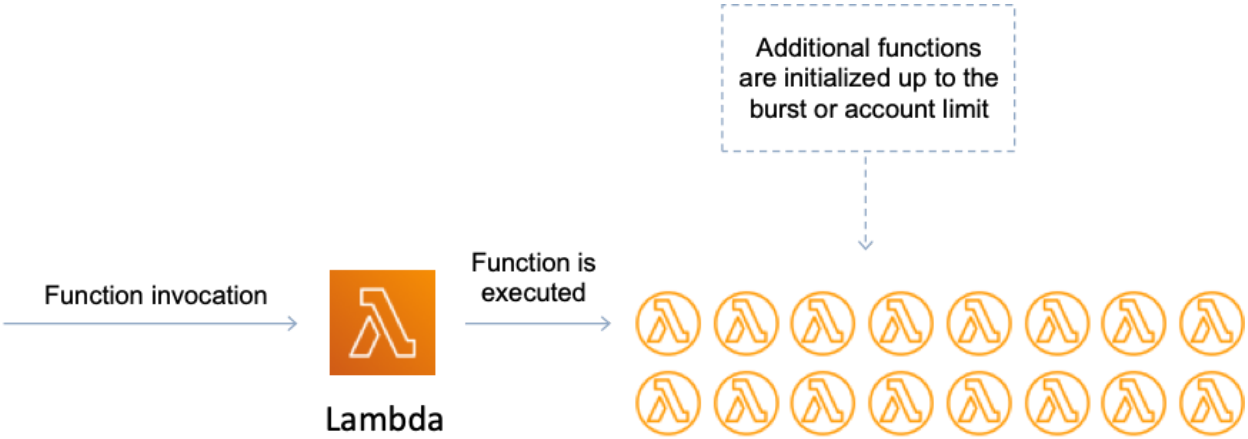
Using user data is definitely the simplest method of achieving this requirement. If using an Auto Scaling Group, you would also only have to place the data in the user data field once for the launch configuration.

10. An instance profile is a container for an IAM role that you can use to pass role information to an EC2 instance when the instance starts. This is a secure way to authorize and EC2 instance to access AWS services.



Instance profiles are created automatically if you use the console to add a role to an instance. You can also create instance profiles using the AWS CLI or API and assign roles to them.

11. Concurrency is the number of requests that your function is serving at any given time. When your function is invoked, Lambda allocates an instance of it to process the event. When the function code finishes running, it can handle another request. If the function is invoked again while a request is still being processed, another instance is allocated, which increases the function's concurrency.



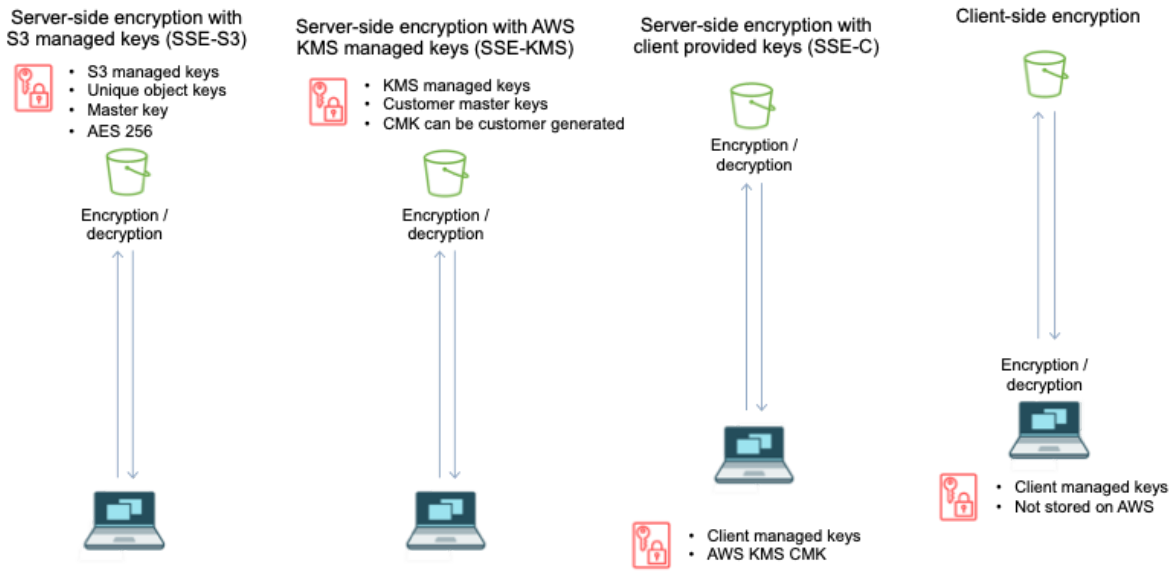
In this scenario, the average execution time is 100 seconds and 50 requests are received per second. This means the concurrency requirement is 100 x 50 = 5,000. As 5,000 is well above the default allowed concurrency of 1,000 executions a second. Therefore, the Developer will need to contact AWS Support to increase the concurrent execution limits.

12. The /tmp directory can be used for storing temporary files within the execution context. This can be used for storing static assets that can be used by subsequent invocations of the function. If the assets must be deleted before the function is invoked again the function code should take care of deleting them.

    There is a limit of 512 MB storage space in the /tmp directory, but this is more than adequate for this scenario.

13. Server-side encryption is about protecting data at rest. Server-side encryption encrypts only the object data, not object metadata. Using server-side encryption with customer-provided encryption keys (SSE-C) allows you to set your own encryption keys.

With the encryption key you provide as part of your request, Amazon S3 manages the encryption as it writes to disks and decryption when you access your objects. Therefore, you don't need to maintain any code to perform data encryption and decryption. The only thing you do is manage the encryption keys you provide.



Therefore, SSE-C is the best choice as AWS will manage all encryption and decryption operations whilst the company get to supply keys that they can manage.

14. AWS Fargate is a serverless compute engine for containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS). Fargate makes it easy for you to focus on building your applications. Fargate removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design.

    There are two launch types with Amazon ECS and the following table describes the differences:

Amazon EC2	Amazon Fargate
You explicitly provision EC2 instances	The control plane asks for resources and Fargate automatically provisions
You're responsible for upgrading, patching, care of EC2 pool	Fargate provisions compute as needed
You must handle cluster optimization	Fargate handles cluster optimization
More granular control over infrastructure	Limited control, as infrastructure is automated

As you can see with the EC2 launch type you must manage the infrastructure layer (Amazon EC2 instances), whereas with Amazon Fargate you do not. Therefore, for this scenario the Fargate launch type should be used.

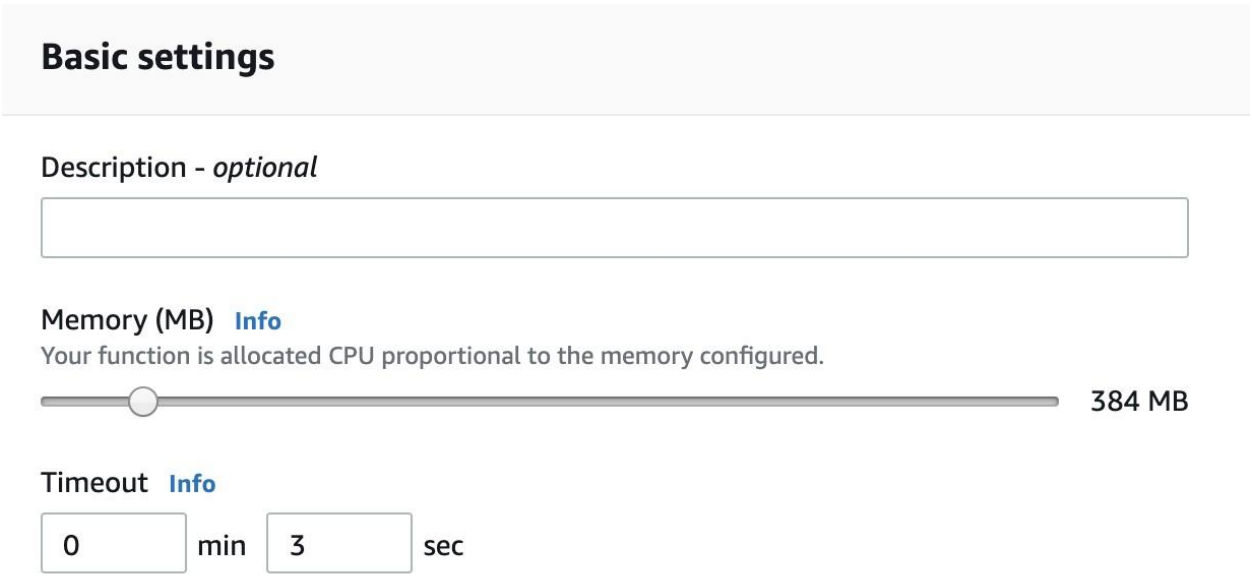
15. AWS Fargate is a serverless compute engine for containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS). Fargate makes it easy for you to focus on building your applications. Fargate removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design.

    There are two launch types with Amazon ECS and the following table describes the differences:

Amazon EC2	Amazon Fargate
You explicitly provision EC2 instances	The control plane asks for resources and Fargate automatically provisions
You're responsible for upgrading, patching, care of EC2 pool	Fargate provisions compute as needed
You must handle cluster optimization	Fargate handles cluster optimization
More granular control over infrastructure	Limited control, as infrastructure is automated

    As you can see with the EC2 launch type you must manage the infrastructure layer (Amazon EC2 instances), whereas with Amazon Fargate you do not. Therefore, for this scenario the Fargate launch type should be used.

16. You can allocate memory between 128 MB and 3,008 MB in 64-MB increments. AWS Lambda allocates CPU power linearly in proportion to the amount of memory configured. At 1,792 MB, a function has the equivalent of one full vCPU (one vCPU-second of credits per second).



Therefore, the way provide more compute capacity to this function is to allocate more memory.

17. The “all at once” policy will deploy the update in the fastest time but will incur downtime.

    All at once:

    Deploys the new version to all instances simultaneously.

    All of your instances are out of service while the deployment takes place.

    Fastest deployment.

    Good for quick iterations in development environment.

    You will experience an outage while the deployment is taking place – not ideal for mission-critical systems.

    If the update fails, you need to roll back the changes by re-deploying the original version to all of your instances.

    No additional cost.

    For this scenario downtime is acceptable and deploying in the fastest possible time is required so the “all at once” policy is the best choice.

18. **The primary differences between AWS SAM templates and AWS CloudFormation templates are the following:**

• Transform declaration. The declaration Transform: AWS::Serverless-2016-10-31 is required for AWS SAM templates. This declaration identifies an AWS CloudFormation template as an AWS SAM template.

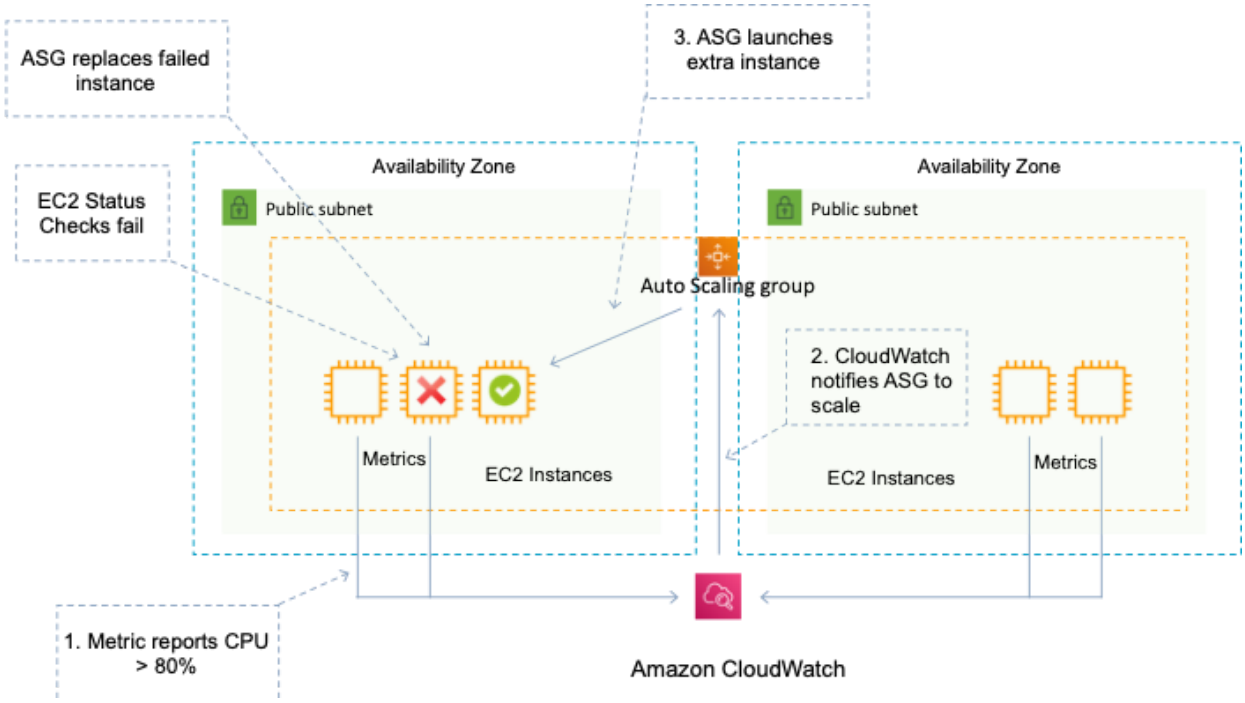
• Globals section. The Globals section is unique to AWS SAM. It defines properties that are common to all your serverless functions and APIs. All the AWS::Serverless::Function, AWS::Serverless::Api, and AWS::Serverless::SimpleTable resources inherit the properties that are defined in the Globals section.

• Resources section. In AWS SAM templates the Resources section can contain a combination of AWS CloudFormation resources and AWS SAM resources.

Of these three sections, only the Transform section and Resources sections are required; the Globals section is optional.

19. **Amazon EC2 Auto Scaling helps you maintain application availability and allows you to automatically add or remove EC2 instances according to conditions you define. You can use the fleet management features of EC2 Auto Scaling to maintain the health and availability of your fleet.**

    You can also use the dynamic and predictive scaling features of EC2 Auto Scaling to add or remove EC2 instances. Dynamic scaling responds to changing demand and predictive scaling automatically schedules the right number of EC2 instances based on predicted demand. Dynamic scaling and predictive scaling can be used together to scale faster.



A *launch configuration* is an instance configuration template that an Auto Scaling group uses to launch EC2 instances. When you create a launch configuration, you specify information for the instances. Include the ID of the Amazon Machine Image (AMI), the instance type, a key pair, one or more security groups, and a block device mapping. If you've launched an EC2 instance before, you specified the same information in order to launch the instance.

You can specify your launch configuration with multiple Auto Scaling groups. However, you can only specify one launch configuration for an Auto Scaling group at a time, and you can't modify a launch configuration after you've created it. To change the launch configuration for an Auto Scaling group, you must create a launch configuration and then update your Auto Scaling group with it.

Therefore, the Developer should create a launch configuration and use Amazon EC2 Auto Scaling.

20. **To add a role to an Amazon EC2 instance using the AWS CLI you must first create an instance profile. Then you need to add the role to the instance profile and finally assign the instance profile to the Amazon EC2 instance.**

The following example commands would achieve this outcome:

```
aws iam create-instance-profile --instance-profile-name
EXAMPLEPROFILENAME
```

```
aws iam add-role-to-instance-profile --instance-profile-name
EXAMPLEPROFILENAME --role-name EXAMPLEROLENAME
aws ec2 associate-iam-instance-profile --iam-instance-profile
Name=EXAMPLEPROFI
```

21. **Rolling with additional batch:**

    Like Rolling but launches new instances in a batch ensuring that there is full availability.

    Application is running at capacity.

    Can set the bucket size.

    Application is running both versions simultaneously.

    Small additional cost.

    Additional batch is removed at the end of the deployment.

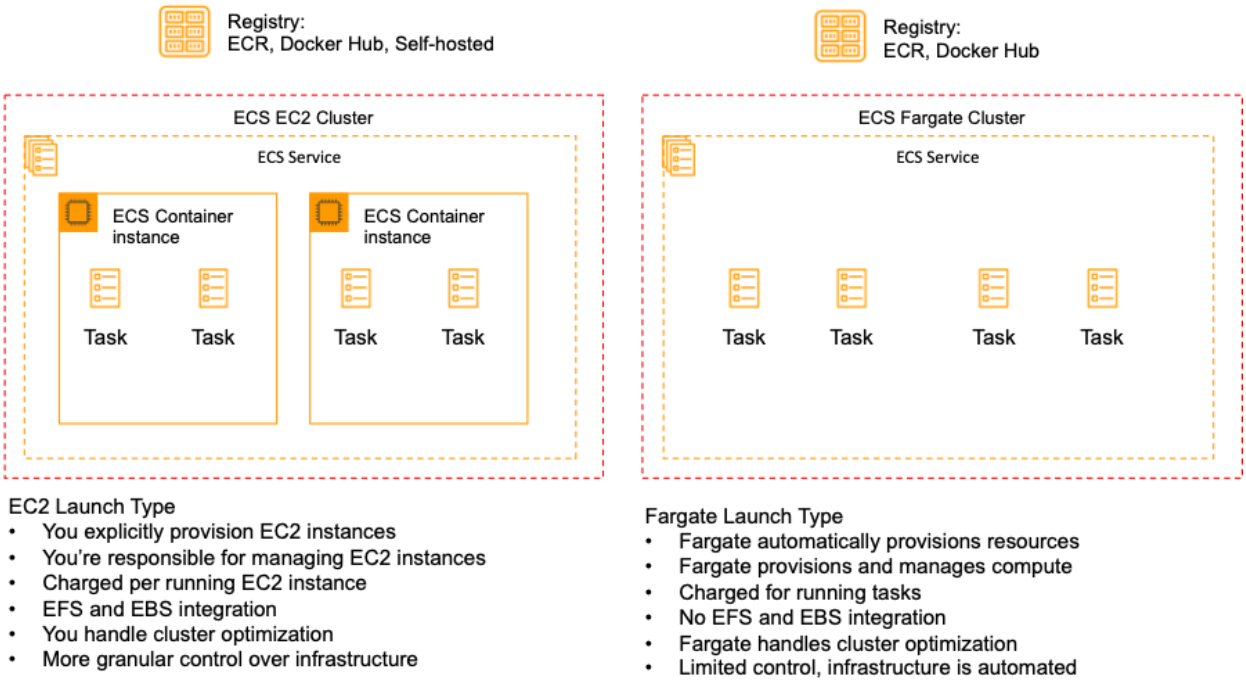
    Longer deployment.

    Good for production environments.

    For this scenario there can be no reduction in application performance and availability during the update. The question also asks for the most cost-effective choice.

    Therefore, the "rolling with additional batch" is the best choice as it will ensure fully availability of the application but minimize cost as the additional batch size can be kept small.

22. **AWS Fargate is a serverless compute engine for Docker containers that works with both Amazon Elastic Container Service (ECS) and Amazon Elastic Kubernetes Service (EKS). Fargate makes it easy for you to focus on building your applications. Fargate removes the need to provision and manage servers, lets you specify and pay for resources per application, and improves security through application isolation by design.**



EC2 Launch Type

- You explicitly provision EC2 instances
- You're responsible for managing EC2 instances
- Charged per running EC2 instance
- EFS and EBS integration
- You handle cluster optimization
- More granular control over infrastructure

Fargate Launch Type

- Fargate automatically provisions resources
- Fargate provisions and manages compute
- Charged for running tasks
- No EFS and EBS integration
- Fargate handles cluster optimization
- Limited control, infrastructure is automated

As you can see in the image above, AWS Fargate is serverless, however, the EC2 launch type requires the management of EC2 instances. Therefore, the most suitable service for the Developer's requirements is AWS Fargate.

23. **The requirements clearly state that we cannot impact the performance of the EC2 instances at all. Therefore, we will not be able to add certificates to the EC2 instances as that would place a burden on the CPU when encrypting and decrypting data.**

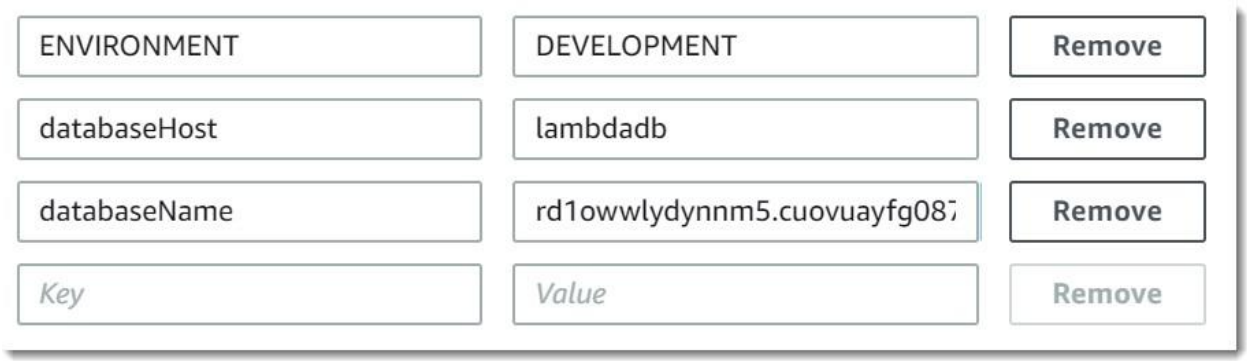
    We are therefore left with configuring SSL on the Elastic Load Balancer itself. For this we need to add an SSL certificate to the ELB and then configure the ELB for SSL termination.

    You can create an HTTPS listener, which uses encrypted connections (also known as *SSL offload*). This feature enables traffic encryption between your load balancer and the clients that initiate SSL or TLS sessions.

    To use an HTTPS listener, you must deploy at least one SSL/TLS server certificate on your load balancer. The load balancer uses a server certificate to terminate the front-end connection and then decrypt requests from clients before sending them to the targets.

    This is the most secure solution we can created without adding any performance impact to the EC2 instances.

24. **Use environment variables to pass environment-specific settings to your code. For example, you can have two functions with the same code but different configuration. One function connects to a test database, and the other connects to a production database. In this situation, you use environment variables to tell the function the hostname and other connection details for the database. You might also set an environment variable to configure your test environment to use more verbose logging or more detailed tracing.**



Therefore, using environment variables is the correct place to store the connection strings associated with the external endpoints.

25. **An event source mapping is an AWS Lambda resource that reads from an event source and invokes a Lambda function. You can use event source mappings to process items from a stream or queue in services that don't invoke Lambda functions directly. Lambda provides event source mappings for the following services.**

    Services That Lambda Reads Events From

    • Amazon Kinesis

    • Amazon DynamoDB

    • Amazon Simple Queue Service

    An event source mapping uses permissions in the function's execution role to read and manage items in the event source. Permissions, event structure, settings, and polling behavior vary by event source.

26. **Immutable:**

    • Launches new instances in a new ASG and deploys the version update to these instances before swapping traffic to these instances once healthy.

    • Zero downtime.

Blue / Green deployment:

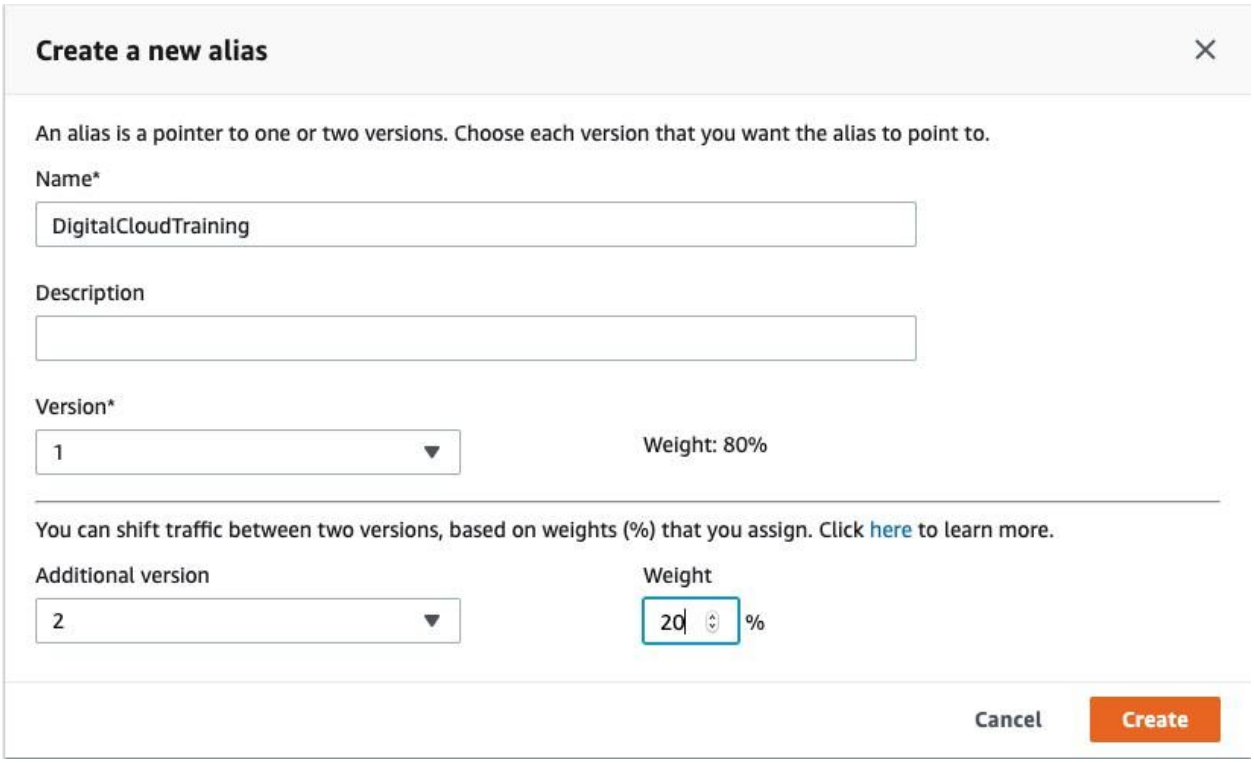
• Zero downtime and release facility.

• Create a new "stage" environment and deploy updates there.

The immutable and blue/green options both provide zero downtime as they will deploy the new version to a new version of the application. These are also the only two options that will ONLY deploy the updates to new EC2 instances.

27. **You can create one or more aliases for your AWS Lambda function. A Lambda alias is like a pointer to a specific Lambda function version. Users can access the function version using the alias ARN.**

You can point an alias a multiple versions of your function code and then assign a weighting to direct certain amounts of traffic to each version. This enables a blue/green style of deployment and means it's easy to roll back to the older version by simply updating the weighting if issues occur.



28. A **serverless application** is a combination of Lambda functions, event sources, and other resources that work together to perform tasks. Note that a serverless application is more than just a Lambda function—it can include additional resources such as APIs, databases, and event source mappings.

AWS SAM templates are an extension of AWS CloudFormation templates, with some additional components that make them easier to work with.

To create a Lambda function using an AWS SAM template the Developer can use the AWS::Serverless::Function resource type. The AWS::Serverless::Function resource type can be used to Create a Lambda function, IAM execution role, and event source mappings that trigger the function.

To create a DynamoDB table using an AWS SAM template the Developer can use the AWS::Serverless::SimpleTable resource type which creates a DynamoDB table with a single attribute primary key. It is useful when data only needs to be accessed via a primary key.

29. The AWS Serverless Application Model (AWS SAM) is an open-source framework that you can use to build serverless applications on AWS. A serverless application is a combination of Lambda functions, event sources, and other resources that work together to perform tasks.

    Note: A serverless application is more than just a Lambda function—it can include additional resources such as APIs, databases, and event source mappings.

AWS SAM provides you with a simple and clean syntax to describe the functions, APIs, permissions, configurations, and events that make up a serverless application.

The example AWS SAM template file below creates an AWS Lambda function and a simple Amazon API Gateway API with a Get method and a /greeting resource:

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Transform: 'AWS::Serverless-2016-10-31'
Description: 'Creates simple function and API resource'
Resources:
    LambdaFunctionWithAPI:
      Type: AWS::Serverless::Function
      Properties:
        Handler: index.handler
        Runtime: nodejs12.x
        Policies:
        Events:
          HttpPost:
            Type: Api
            Properties:
              Path: '/greeting'
              Method: get
```

You can use AWS SAM to define your serverless applications. AWS SAM consists of the following components:

• **AWS SAM template specification**. You use this specification to define your serverless application. It provides you with a simple and clean syntax to describe the functions, APIs, permissions, configurations, and events that make up a serverless application.

• **AWS SAM command line interface (AWS SAM CLI)**. You use this tool to build serverless applications that are defined by AWS SAM templates.

30. The application currently resides in a single subnet and that is within a single availability zone. To increase fault tolerance the application instances should be split across subnets that are in different availability zones. This will protect against any faults that occur within a single AZ.

    To do this, a subnet in another AZ can be added to both the ASG and the ALB. The ASG will automatically launch instances in the new subnet and try and balance the number of instances between these subnets. The ALB will distribute connections across both subnets/AZs.

31. The “immutable” policy will create a new ASG with a whole new set of instances and deploy the updates there.

    Immutable:

    • Launches new instances in a new ASG and deploys the version update to these instances before swapping traffic to these instances once healthy.

    • Zero downtime.

    • New code is deployed to new instances using an ASG.

    • High cost as double the number of instances running during updates.

    • Longest deployment.

    • Quick rollback in case of failures.

    • Great for production environments.

    For this scenario a quick rollback must be prioritized over all other considerations. Therefore, the best choice is “immutable”. This deployment policy is the most expensive and longest (duration) option. However, you can roll back quickly and safely as the original instances are all available and unmodified.

32. As this is a stateful application the session data needs to be stored somewhere. Amazon DynamoDB is designed to be used for storing session data and it highly scalable. To add elasticity to the architecture an Amazon Elastic Load Balancer (ELB) and Amazon EC2 Auto Scaling group (ASG) can be used.

    With this architecture the web service can scale elastically using the ASG and the ELB will distribute traffic to all new instances that the ASG launches. This is a good example of utilizing some of the key benefits of refactoring applications into the AWS cloud.

33. A *task placement constraint* is a rule that is considered during task placement. Task placement constraints can be specified when either running a task or creating a new service.

    Amazon ECS supports the following types of task placement constraints:

    `distinctInstance`

Place each task on a different container instance. This task placement constraint can be specified when either running a task or creating a new service.

`memberOf`

Place tasks on container instances that satisfy an expression. For more information about the expression syntax for constraints, see Cluster Query Language.

The `memberOf` task placement constraint can be specified with the following actions:

Running a task

Creating a new service

Creating a new task definition

Creating a new revision of an existing task definition

The following code can be used in a task definition to specify a task placement constraint that ensures that each task will run on a distinct instance:

```
"placementConstraints": [
{
"type": "distinctInstance"
}
]
```

34. The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings. With just a few lines per resource, you can define the application you want and model it using YAML. During deployment, SAM transforms and expands the SAM syntax into AWS CloudFormation syntax, enabling you to build serverless applications faster.

    To get started with building SAM-based applications, use the AWS SAM CLI. SAM CLI provides a Lambda-like execution environment that lets you locally build, test, and debug applications defined by SAM templates. You can also use the SAM CLI to deploy your applications to AWS.

    With the SAM CLI you can package and deploy your source code using two simple commands:

    • sam package

• sam deploy

Alternatively, you can use:

• aws cloudformation package

• aws cloudformation deploy

The SAM CLI is therefore the easiest way to deploy serverless applications on AWS.

35. You can invoke Lambda functions directly with the Lambda console, the Lambda API, the AWS SDK, the AWS CLI, and AWS toolkits.

You can also configure other AWS services to invoke your function, or you can configure Lambda to read from a stream or queue and invoke your function.

When you invoke a function, you can choose to invoke it synchronously or asynchronously.

• Synchronous invocation:

o You wait for the function to process the event and return a response.

o To invoke a function synchronously with the AWS CLI, use the invoke command.

o The Invocation-type can be used to specify a value of “RequestResponse”. This instructs AWS to execute your Lambda function and wait for the function to complete.

• Asynchronous invocation:

o When you invoke a function asynchronously, you don’t wait for a response from the function code.

o For asynchronous invocation, Lambda handles retries and can send invocation records to a destination.

o To invoke a function asynchronously, set the invocation type parameter to Event.

The fastest way to process all the files is to use asynchronous invocation and process the files in parallel. To do this you should specify the invocation type of Event

36. The /tmp directory provides 512 MB of storage space that can be used by a function. When a file is cached by a function in the /tmp directory it is available to be used by subsequent executions of the function which will reduce latency.
37. You can add AWS Elastic Beanstalk configuration files (.ebextensions) to your web application's source code to configure your environment and customize the AWS resources that it contains.

Configuration files are YAML- or JSON-formatted documents with a .config file extension that you place in a folder named .ebextensions and deploy in your application source bundle.

For example, you could include a configuration file for setting the load balancer type into:

.ebextensions/load-balancer.config

This example makes a simple configuration change. It modifies a configuration option to set the type of your environment's load balancer to Network Load Balancer:

Requirements

• **Location** – Place all of your configuration files in a single folder, named .ebextensions, in the root of your source bundle. Folders starting with a dot can be hidden by file browsers, so make sure that the folder is added when you create your source bundle.

• **Naming** – Configuration files must have the .config file extension.

• **Formatting** – Configuration files must conform to YAML or JSON specifications.

• **Uniqueness** – Use each key only once in each configuration file.

Therefore, the Developer should place the file in the .ebextensions folder in the application source bundle.

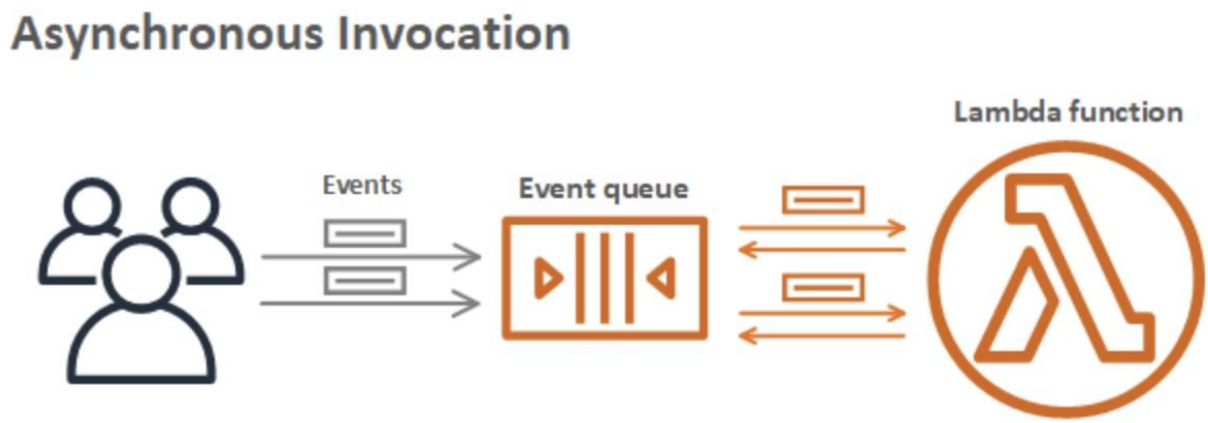
38. You can launch a cluster of multicontainer instances in a single-instance or autoscaling Elastic Beanstalk environment using the Elastic Beanstalk console. The single container and multicontainer Docker platforms for Elastic Beanstalk support the use of Docker images stored in a public or private online image repository.

    You specify images by name in the Dockerrun.aws.json file and save it in the root of your source directory.

39. Several AWS services, such as Amazon Simple Storage Service (Amazon S3) and Amazon Simple Notification Service (Amazon SNS), invoke functions asynchronously to process events.

    When you invoke a function asynchronously, you don't wait for a response from the function code. You hand off the event to Lambda and Lambda handles the rest. You can configure how Lambda handles errors, and can send invocation records to a downstream resource to chain together components of your application.

    The following diagram shows clients invoking a Lambda function asynchronously. Lambda queues the events before sending them to the function.



    For asynchronous invocation, Lambda places the event in a queue and returns a success response without additional information. A separate process reads events from the queue and sends them to your function. To invoke a function asynchronously, set the invocation type parameter to Event.

In this scenario the Event parameter has been used so we know the function has been invoked asynchronously. For asynchronous invocation the status code 202 indicates a successful execution.

40. When you initialize a new service client without supplying any arguments, the AWS SDK for Java attempts to find AWS credentials by using the *default credential provider chain* implemented by the DefaultAWSCredentialsProviderChain class. The default credential provider chain looks for credentials in this order:

1. Environment variables–AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY. The AWS SDK for Java uses the EnvironmentVariableCredentialsProvider class to load these credentials.

2. Java system properties–aws.accessKeyId and aws.secretKey. The AWS SDK for Java uses the SystemPropertiesCredentialsProvider to load these credentials.

3. The default credential profiles file– typically located at ~/.aws/credentials (location can vary per platform) and shared by many of the AWS SDKs and by the AWS CLI. The AWS SDK for Java uses the ProfileCredentialsProvider to load these credentials.

4. Amazon ECS container credentials– loaded from the Amazon ECS if the environment variable AWS_CONTAINER_CREDENTIALS_RELATIVE_URI is set. The AWS SDK for Java uses the ContainerCredentialsProvider to load these credentials. You can specify the IP address for this value.

5. Instance profile credentials– used on EC2 instances and delivered through the Amazon EC2 metadata service. The AWS SDK for Java uses the InstanceProfileCredentialsProvider to load these credentials. You can specify the IP address for this value.

Therefore, the AWS SDK for Java will find the credentials stored in environment variables before it checks for instance provide credentials and will allow access to the extra S3 buckets.

NOTE: The Default Credential Provider Chain is very similar for other SDKs and the CLI as well. Check the references below for an article showing the steps for the AWS CLI.

41. A **serverless application** is a combination of Lambda functions, event sources, and other resources that work together to perform tasks. Note that a serverless application is more than just a Lambda function—it can include additional resources such as APIs, databases, and event source mappings.

    AWS SAM templates are an extension of AWS CloudFormation templates, with some additional components that make them easier to work with. To create a Lambda function using an AWS SAM template the Developer can use the AWS::Serverless::Function resource type.

    The AWS::Serverless::Function resource type can be used to Create a Lambda function, IAM execution role, and event source mappings that trigger the function.

42. Several AWS services, such as Amazon Simple Storage Service (Amazon S3) and Amazon Simple Notification Service (Amazon SNS), invoke functions asynchronously to process events.

    When you invoke a function asynchronously, you don't wait for a response from the function code. You hand off the event to Lambda and Lambda handles the rest. You can configure how Lambda handles errors and can send invocation records to a downstream resource to chain together components of your application.

    The following code snippet is an example of invoking the “my-function” function asynchronously:

    ```
    $ aws lambda invoke --function-name my-function --invocation-type Event --payload '{ "key": "value" }' response.json
    {
    "StatusCode": 202
    }
    ```

    The Developer will therefore need to set the `-invocation-type` option to `Event`.

43. A *task placement strategy* is an algorithm for selecting instances for task placement or tasks for termination. Task placement strategies can be specified when either running a task or creating a new service.

    Amazon ECS supports the following task placement strategies:

    binpack - place tasks based on the least available amount of CPU or memory. This minimizes the number of instances in use.

random - place tasks randomly.

spread - place tasks evenly based on the specified value. Accepted values are instanceId (or host, which has the same effect), or any platform or custom attribute that is applied to a container instance, such as attribute:ecs.availability-zone. Service tasks are spread based on the tasks from that service. Standalone tasks are spread based on the tasks from the same task group.

To minimize the number of instances in use, the binpack placement strategy is the best choice for this scenario.

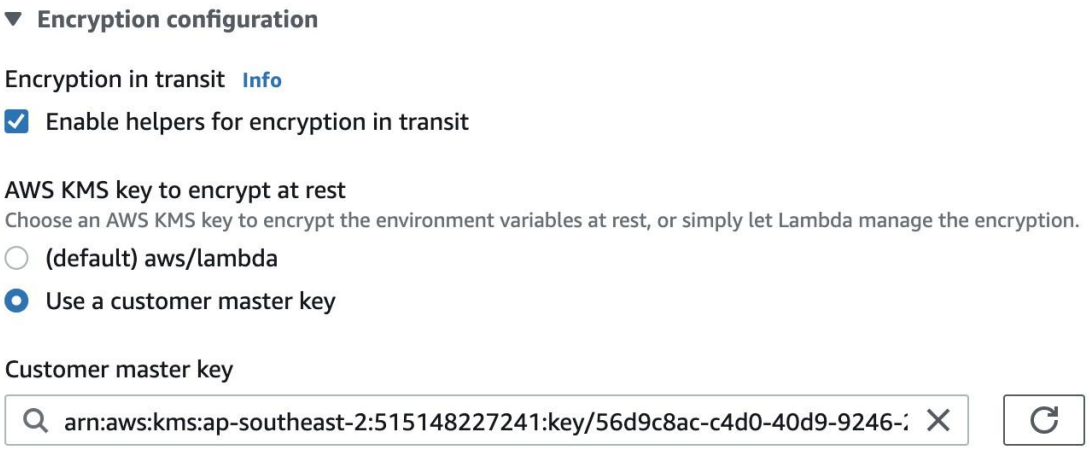
44. You can use environment variables to store secrets securely for use with Lambda functions. Lambda always encrypts environment variables at rest.

Additionally, you can use the following features to customize how environment variables are encrypted.

• **Key configuration** – On a per-function basis, you can configure Lambda to use an encryption key that you create and manage in AWS Key Management Service. These are referred to as *customer managed* customer master keys (CMKs) or customer managed keys. If you don't configure a customer managed key, Lambda uses an AWS managed CMK named aws/lambda, which Lambda creates in your account.

• **Encryption helpers** – The Lambda console lets you encrypt environment variable values client side, before sending them to Lambda. This enhances security further by preventing secrets from being displayed unencrypted in the Lambda console, or in function configuration that's returned by the Lambda API. The console also provides sample code that you can adapt to decrypt the values in your function handler.

The configuration for using encryption helps to encrypt data client-side looks like this:



This is the best way to achieve this outcome and minimizes complexity as the encryption infrastructure will still use AWS KMS and be able to decrypt the values during function execution.

45. AWS Lambda automatically monitors Lambda functions on your behalf, reporting metrics through Amazon CloudWatch. To help you troubleshoot failures in a function, Lambda logs all requests handled by your function and also automatically stores logs generated by your code through Amazon CloudWatch Logs.

    Lambda automatically integrates with CloudWatch Logs and pushes all logs from your code to a CloudWatch Logs group associated with a Lambda function, which is named /aws/lambda/*<function name>*.

    An AWS Lambda function's execution role grants it permission to access AWS services and resources. You provide this role when you create a function, and Lambda assumes the role when your function is invoked. You can create an execution role for development that has permission to send logs to Amazon CloudWatch and upload trace data to AWS X-Ray.

    For the lambda function to create log stream and publish logs to cloudwatch, the lambda execution role needs to have the following permissions:

```
{
    "Statement": [
        {
            "Action": [
                "logs:CreateLogGroup",
                 "logs:CreateLogStream",
                 "logs:PutLogEvents"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:logs:*:*:*"
        }
    ]
}
```

The most likely cause of this issue is that the execution role assigned to the Lambda function does not have the permissions (shown above) to write to CloudWatch Logs.

46. The content in the 'hooks' section of the AppSpec file varies, depending on the compute platform for your deployment. The 'hooks' section for an EC2/On-Premises deployment contains mappings that link deployment lifecycle event hooks to one or more scripts.

    The 'hooks' section for a Lambda or an Amazon ECS deployment specifies Lambda validation functions to run during a deployment lifecycle event. If an event hook is not present, no operation is executed for that event. This section is required only if you are running scripts or Lambda validation functions as part of the deployment.

47. The following code snippet shows a valid example of the structure of hooks for an Amazon ECS deployment:

```
Hooks:
  - BeforeInstall: "LambdaFunctionToValidateBeforeInstall"
  - AfterInstall: "LambdaFunctionToValidateAfterTraffic"
  - AfterAllowTestTraffic: "LambdaFunctionToValidateAfterTestTrafficStarts"
  - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeAllowingProductionTraffic"
  - AfterAllowTraffic: "LambdaFunctionToValidateAfterAllowingProductionTraffic"
```

    Therefore, in this scenario a valid structure for the order of hooks that should be specified in the appspec.yml file is: BeforeInstall > AfterInstall > AfterAllowTestTraffic > BeforeAllowTraffic > AfterAllowTraffic

48. The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings. With just a few lines per resource, you can define the application you want and model it using YAML.

    The “Transform” header indicates that the developer is creating a SAM template as it has the value: Transform: '`AWS::Serverless-2016-10-31`'

    Therefore there are two sets of commands that can be used to package and deploy using SAM:

    Use either:

    • sam package

    • sam deploy

    Or use:

    • aws cloudformation package

    • aws cloudformation deploy

49. When you invoke a function synchronously, Lambda runs the function and waits for a response. When the function execution ends, Lambda returns the response from the function's code with additional data, such as the version of the function that was executed. To invoke a function synchronously with the AWS CLI, use the invoke command.

    The following diagram shows clients invoking a Lambda function synchronously. Lambda sends the events directly to the function and sends the function's response back to the invoker.

We know the function has been run synchronously as the `--invocation-type Event` parameter has not been included. Also, the status code 200 indicates a successful execution of a synchronous execution.

50. With Destinations, you can send asynchronous function execution results to a destination resource without writing code. A function execution result includes version, timestamp, request context, request payload, response context, and response payload. For each execution status (i.e. Success and Failure), you can choose one destination from four options: another Lambda function, an SNS topic, an SQS standard queue, or EventBridge.

    For this scenario, the code will be run by Lambda and the execution result will then be sent to the SNS topic. The application that is subscribed to the SNS topics will then receive the notification.

51. A *task placement constraint* is a rule that is considered during task placement. Task placement constraints can be specified when either running a task or creating a new service.

    The memberOf task placement constraint places tasks on container instances that satisfy an expression.

    The memberOf task placement constraint can be specified with the following actions:

• Running a task

• Creating a new service

• Creating a new task definition

• Creating a new revision of an existing task definition

The example JSON code uses the memberOf constraint to place tasks on instances in the databases task group. It can be specified with the following actions: CreateService, UpdateService, RegisterTaskDefinition, and RunTask.

52. The `sam init` command initializes a serverless application with an AWS SAM template. The template provides a folder structure for your Lambda functions and is connected to an event source such as APIs, S3 buckets, or DynamoDB tables. This application includes everything you need to get started and to eventually extend it into a production-scale application.

    This is the simplest way for the Developer to quickly get started with testing AWS SAM. Before the Developer can use the "sam" commands it is necessary to install the AWS SAM CLI. This is separate to the AWS CLI.

53. The key requirements in this scenario are to add fault tolerances and elasticity to the web tier and application tier. Note that no specific requirements for the back end have been included.

    To add elasticity to the web and application tiers the Developer should create Auto Scaling groups of EC2 instances. We know that the application tier runs on Linux and the web tier runs on Apache Tomcat (which could be on Linux or Windows). Therefore, these workloads are suitable for an ASG and this will ensure the number of instances dynamically scales out and in based on actual usage.

    To add fault tolerance to the web and application tiers the Developer should add an Elastic Load Balancer. This will ensure that if the number of EC2 instances are changed by the ASG, the load balancer is able to distribute traffic to them. This also assists with elasticity

54. When a task that uses the EC2 launch type is launched, Amazon ECS must determine where to place the task based on the requirements specified in the

task definition, such as CPU and memory. Similarly, when you scale down the task count, Amazon ECS must determine which tasks to terminate. You can apply task placement strategies and constraints to customize how Amazon ECS places and terminates tasks. Task placement strategies and constraints are not supported for tasks using the Fargate launch type. By default, Fargate tasks are spread across Availability Zones.

A *task placement strategy* is an algorithm for selecting instances for task placement or tasks for termination. For example, Amazon ECS can select instances at random, or it can select instances such that tasks are distributed evenly across a group of instances.

Amazon ECS supports the following task placement strategies:

• binpack

Place tasks based on the least available amount of CPU or memory. This minimizes the number of instances in use.

• random

Place tasks randomly.

• spread

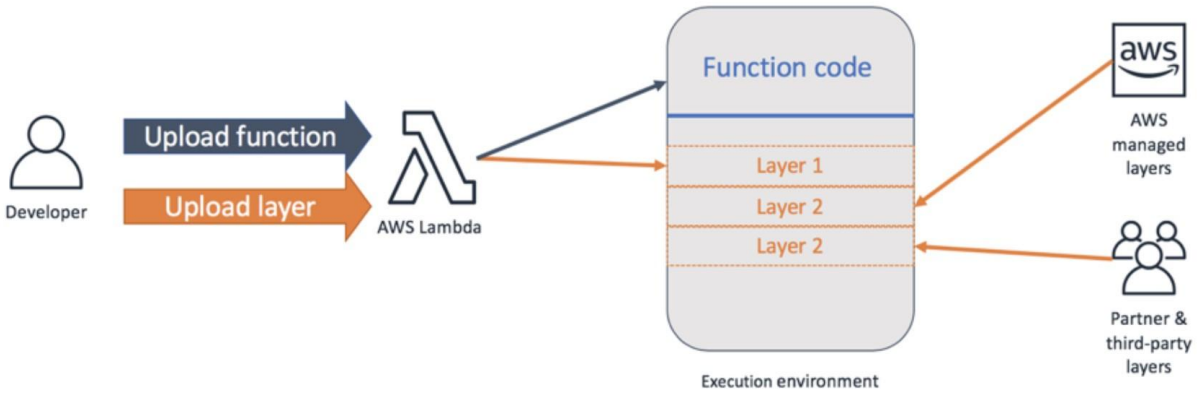
Place tasks evenly based on the specified value. Accepted values are instanceId (or host, which has the same effect), or any platform or custom attribute that is applied to a container instance, such as attribute:ecs.availability-zone. Service tasks are spread based on the tasks from that service. Standalone tasks are spread based on the tasks from the same task group.

Therefore, for this scenario the random task placement strategy is most suitable as it requires the least configuration.

55. You can configure your Lambda function to pull in additional code and content in the form of layers. A layer is a ZIP archive that contains libraries, a custom runtime, or other dependencies. With layers, you can use libraries in your function without needing to include them in your deployment package.

    Layers let you keep your deployment package small, which makes development easier. You can avoid errors that can occur when you install and package dependencies with your function code.

When a Lambda function configured with a Lambda layer is executed, AWS downloads any specified layers and extracts them to the /opt directory on the function execution environment. Each runtime then looks for a language-specific folder under the /opt directory.



One of the best practices for AWS Lambda functions is to minimize your deployment package size to its runtime necessities in order to reduce the amount of time that it takes for your deployment package to be downloaded and unpacked ahead of invocation.

Therefore, it is preferable to use layers to store the external libraries to optimize performance of the function. Using layers means that the external library will also be available to all of the Lambda functions that the Developer is creating.

56. With IAM roles for Amazon ECS tasks, you can specify an IAM role that can be used by the containers in a task. Applications must sign their AWS API requests with AWS credentials, and this feature provides a strategy for managing credentials for your applications to use, similar to the way that Amazon EC2 instance profiles provide credentials to EC2 instances.

Instead of creating and distributing your AWS credentials to the containers or using the EC2 instance's role, you can associate an IAM role with an ECS task definition or RunTask API operation. The applications in the task's containers can then use the AWS SDK or CLI to make API requests to authorized AWS services.

Therefore, the most secure solution is to use a separate IAM role with the specific permissions required for an individual service and associate that role to the relevant ECS task definition. This should then be repeated for the remaining 5 services.

57. With an Application Load Balancer it is possible to route requests based on the domain name specified in the Host header. This means you can route traffic coming in to forum.example.com and myaccount.example.com to different target groups.

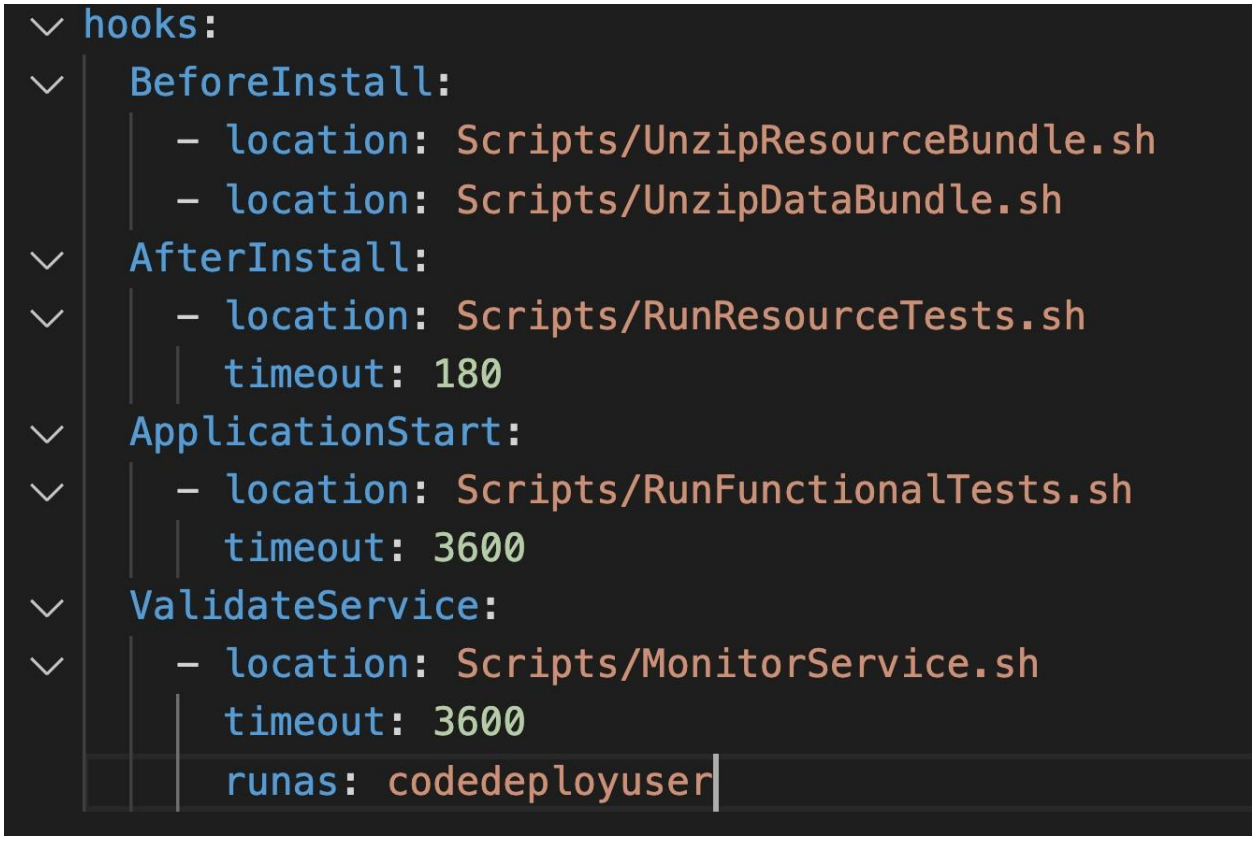
    You can see an example of a couple of similar rules depicted below:

    The Application Load Balancer is the only Elastic Load Balancer provided by AWS that can perform host-based routing.

58. The content in the 'hooks' section of the AppSpec file varies, depending on the compute platform for your deployment. The 'hooks' section for an EC2/On-Premises deployment contains mappings that link deployment lifecycle event hooks to one or more scripts.

    The 'hooks' section for a Lambda or an Amazon ECS deployment specifies Lambda validation functions to run during a deployment lifecycle event. If an event hook is not present, no operation is executed for that event. This section is required only if you are running scripts or Lambda validation functions as part of the deployment.

    The following code snippet shows a valid example of the structure of hooks for an Amazon EC2 deployment:



59. Therefore, in this scenario a valid structure for the order of hooks that should be specified in the appspec.yml file is: **BeforeInstall > AfterInstall > ApplicationStart > ValidateService ------> For amazon EC2**
BeforeAllowTraffic > AfterAllowTraffic" --> this would be valid for AWS Lambda.
BeforeInstall > AfterInstall > AfterAllowTestTraffic > BeforeAllowTraffic > AfterAllowTraffic" → this would be valid for Amazon ECS.

60. **In this scenario the Developer is using an AWS Lambda function to process images that are uploaded to an Amazon S3 bucket. The AWS Lambda function has been created and the notification settings on the bucket have been configured. The last thing to do is to grant permissions for the Amazon S3 service principal to invoke the function.**

    The Lambda CLI add-permission command grants the Amazon S3 service principal (s3.amazonaws.com) permissions to perform the lambda:InvokeFunction action.
61. **When you invoke a function, two types of error can occur. Invocation errors occur when the invocation request is rejected before your function receives it. Function errors occur when your function's code or runtime returns an error.**

Depending on the type of error, the type of invocation, and the client or service that invokes the function, the retry behavior and the strategy for managing errors varies. Function errors occur when your function code or the runtime that it uses return an error.

In this case, with an event source mapping from a stream (Kinesis Data Stream), Lambda retries the entire batch of items. Therefore, the best explanation is that the Lambda function did not handle the error, and the Lambda service attempted to reprocess the data.

62. **A deployment package is a ZIP archive that contains your function code and dependencies. You need to create a deployment package if you use the Lambda API to manage functions, or if you need to include libraries and dependencies other than the AWS SDK.**

    You can upload the package directly to Lambda, or you can use an Amazon S3 bucket, and then upload it to Lambda. If the deployment package is larger than 50 MB, you must use Amazon S3.

    If your function depends on libraries not included in the Lambda runtime, you can install them to a local directory and include them in your deployment package.

63. **Port mappings allow containers to access ports on the host container instance to send or receive traffic. Port mappings are specified as part of the container definition.**

    The container definition settings are specified within the task definition. The relevant settings are:

    containerPort - the port number on the container that is bound to the user-specified or automatically assigned host port.

    hostPort - the port number on the container instance to reserve for your container.

    With an ALB you can use Dynamic port mapping which makes it easier to run multiple tasks on the same Amazon ECS service on an Amazon ECS cluster. This is configured by setting the host port to 0, as in the image below:

64. In CodeDeploy, a revision contains a version of the source files CodeDeploy will deploy to your instances or scripts CodeDeploy will run on your instances. You plan the revision, add an AppSpec file to the revision, and then push the revision to Amazon S3 or GitHub. After you push the revision, you can deploy it.

    For a deployment to an Amazon ECS compute platform:

    • The AppSpec file specifies the Amazon ECS task definition used for the deployment, a container name and port mapping used to route traffic, and optional Lambda functions run after deployment lifecycle events.

    • A revision is the same as an AppSpec file.

    • An AppSpec file can be written using JSON or YAML.

    • An AppSpec file can be saved as a text file or entered directly into a console when you create a deployment.

    Therefore, the appspec.yaml file needs to be updated by the Developer.

65. With target tracking scaling policies, you select a scaling metric and set a target value. Amazon EC2 Auto Scaling creates and manages the CloudWatch alarms that trigger the scaling policy and calculates the scaling adjustment based on the metric and the target value. The scaling policy adds or removes capacity as required to keep the metric at, or close to, the specified target value. In addition to keeping the metric close to the target value, a target tracking scaling policy also adjusts to the changes in the metric due to a changing load pattern.

    For example, you can use target tracking scaling to:

    • Configure a target tracking scaling policy to keep the average aggregate CPU utilization of your Auto Scaling group at 40 percent.

    • Configure a target tracking scaling policy to keep the request count per target of your Elastic Load Balancing target group at 1000 for your Auto Scaling group.

The target tracking scaling policy is therefore the best choice for this scenario.

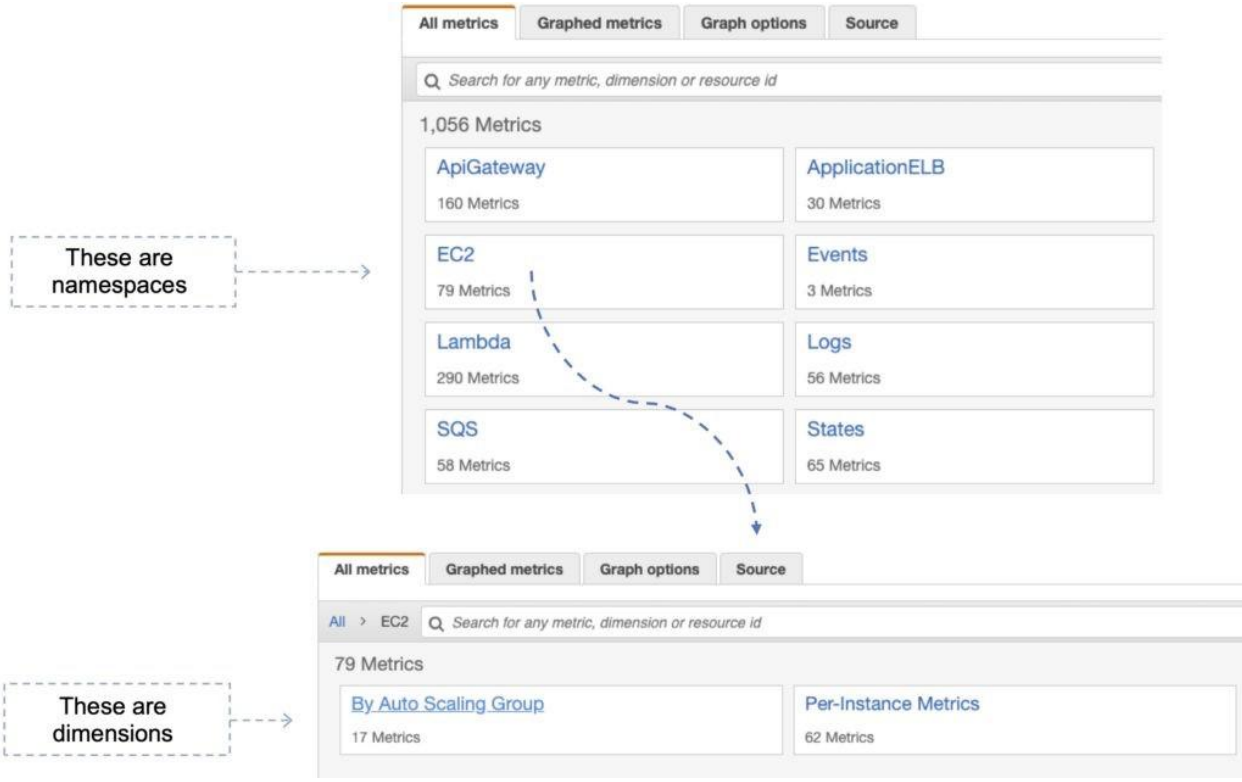
With step scaling and simple scaling, you choose scaling metrics and threshold values for the CloudWatch alarms that trigger the scaling process. You also define how your Auto Scaling group should be scaled when a threshold is in breach for a specified number of evaluation periods.

AWS MANAGEMENT AND GOVERNANCE

1. You can publish your own metrics to CloudWatch using the AWS CLI or an API. You can view statistical graphs of your published metrics with the AWS Management Console.

   CloudWatch stores data about a metric as a series of data points. Each data point has an associated time stamp. You can even publish an aggregated set of data points called a *statistic set*.

   In custom metrics, the --dimensions parameter is common. A dimension further clarifies what the metric is and what data it stores. You can have up to 10 dimensions in one metric, and each dimension is defined by a name and value pair.



As you can see in the above example there are two dimensions associated with the EC2 namespace. These organize the metrics by Auto Scaling Group and Per-Instance metrics. Therefore the Developer should use the --dimensions parameter.

2. The AWS Serverless Application Model (SAM) is an open-source framework for building serverless applications. It provides shorthand syntax to express functions, APIs, databases, and event source mappings. With just a few lines per resource, you can define the application you want and model it using YAML. During deployment, SAM transforms and expands the SAM syntax into AWS CloudFormation syntax, enabling you to build serverless applications faster.

   To get started with building SAM-based applications, use the AWS SAM CLI. SAM CLI provides a Lambda-like execution environment that lets you locally build, test, and debug applications defined by SAM templates. You can also use the SAM CLI to deploy your applications to AWS.

   With the SAM CLI you can package and deploy your source code using two simple commands:

   • sam package

• sam deploy

Alternatively, you can use:

• aws cloudformation package

• aws cloudformation deploy

Therefore, the Developer can use either the sam or aws cloudformation CLI commands to package and deploy the serverless application.

3. The unified CloudWatch agent can be installed on both on-premises servers and Amazon EC2 instances using multiple operating system versions. It enables you to do the following:

   • Collect more system-level metrics from Amazon EC2 instances across operating systems. The metrics can include in-guest metrics, in addition to the metrics for EC2 instances.

   • Collect system-level metrics from on-premises servers. These can include servers in a hybrid environment as well as servers not managed by AWS.

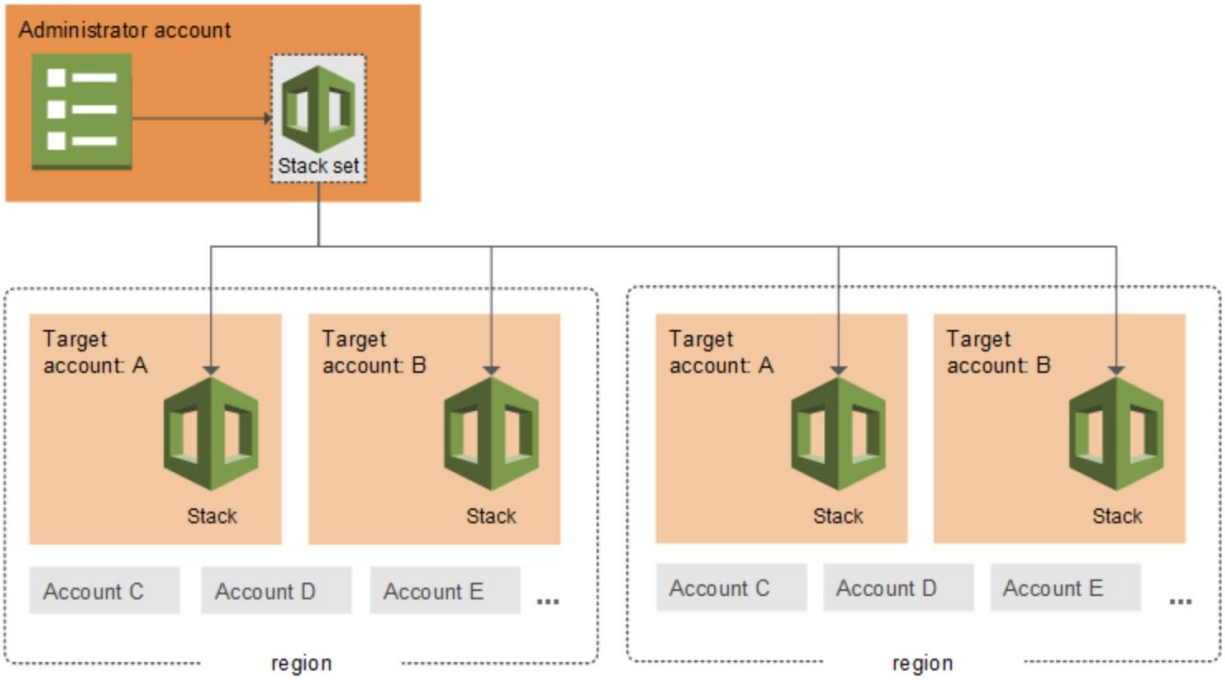
   • Retrieve custom metrics from your applications or services using the StatsD and collectd protocols.

   • Collect logs from Amazon EC2 instances and on-premises servers, running either Linux or Windows Server.

   Therefore, the Development team should install the CloudWatch agent on the on-premises servers and EC2 instances. This will allow them to collect system-level metrics from servers and instances across the hybrid cloud environment.

4. AWS CloudFormation StackSets extends the functionality of stacks by enabling you to create, update, or delete stacks across multiple accounts and regions with a single operation.

   Using an administrator account, you define and manage an AWS CloudFormation template, and use the template as the basis for provisioning stacks into selected target accounts across specified regions.



Using StackSets for this scenario will work well and result in the least operational overhead in creating, updating and deleting CloudFormation stacks across multiple accounts.

5. You can use CloudWatch Logs to monitor applications and systems using log data. For example, CloudWatch Logs can track the number of errors that occur in your application logs and send you a notification whenever the rate of errors exceeds a threshold you specify.

   CloudWatch Logs uses your log data for monitoring; so, no code changes are required. For example, you can monitor application logs for specific literal terms (such as "NullReferenceException") or count the number of occurrences of a literal term at a particular position in log data (such as "404" status codes in an Apache access log).

   When the term you are searching for is found, CloudWatch Logs reports the data to a CloudWatch metric that you specify. Log data is encrypted while in transit and while it is at rest.

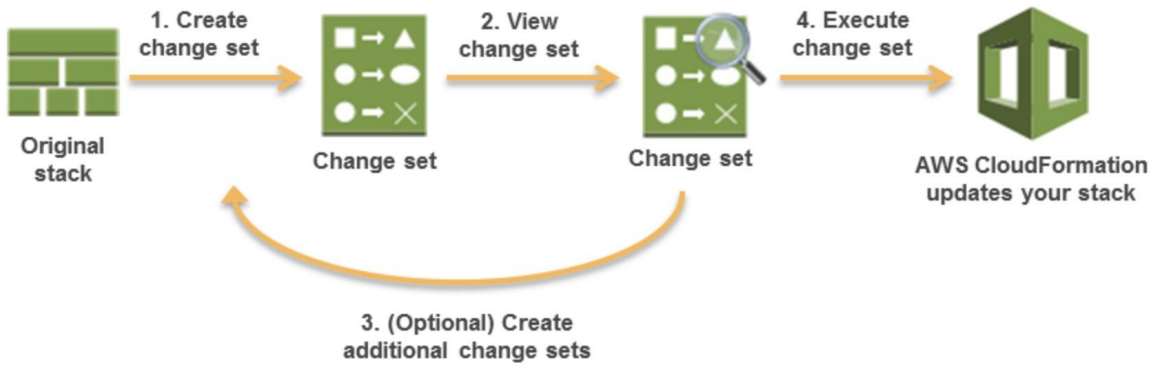
6. You can use CloudWatch Logs to monitor applications and systems using log data. For example, CloudWatch Logs can track the number of errors

that occur in your application logs and send you a notification whenever the rate of errors exceeds a threshold you specify.

CloudWatch Logs uses your log data for monitoring; so, no code changes are required. For example, you can monitor application logs for specific literal terms (such as "NullReferenceException") or count the number of occurrences of a literal term at a particular position in log data (such as "404" status codes in an Apache access log).

When the term you are searching for is found, CloudWatch Logs reports the data to a CloudWatch metric that you specify. Log data is encrypted while in transit and while it is at rest.

7. When you need to update a stack, understanding how your changes will affect running resources before you implement them can help you update stacks with confidence. Change sets allow you to preview how proposed changes to a stack might impact your running resources.

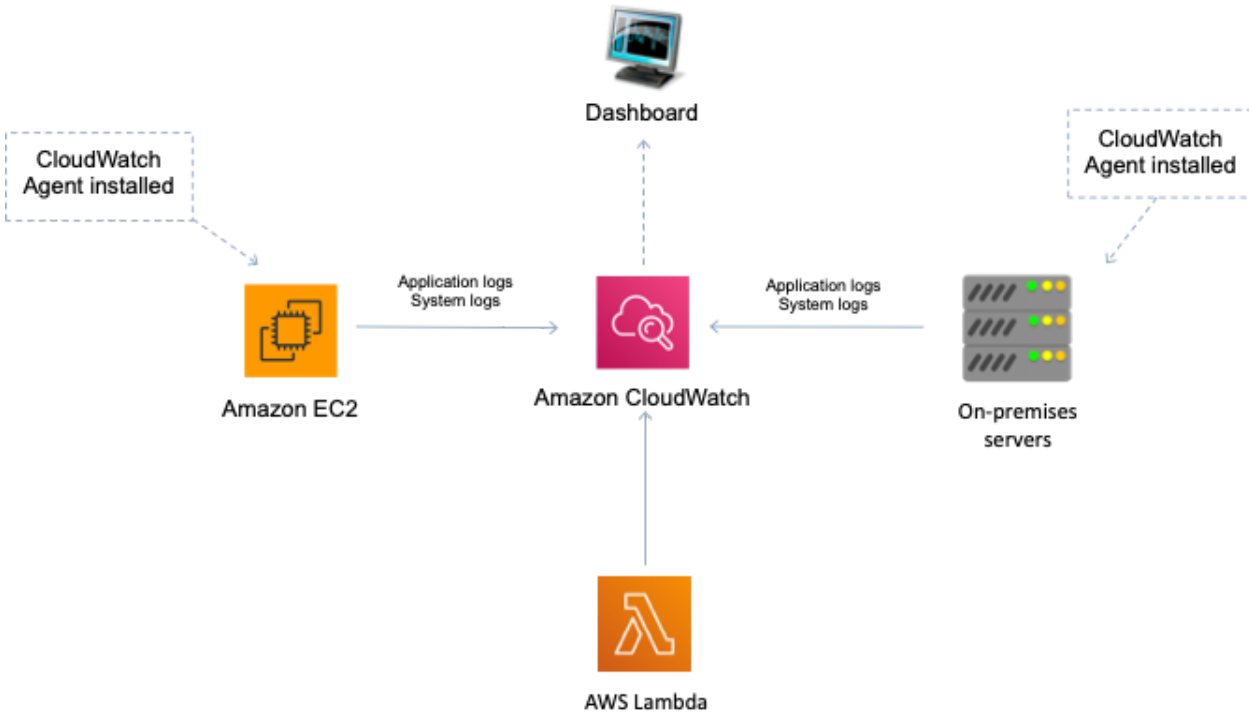


AWS CloudFormation makes the changes to your stack only when you decide to execute the change set, allowing you to decide whether to proceed with your proposed changes or explore other changes by creating another change set. You can create and manage change sets using the AWS CloudFormation console, AWS CLI, or AWS CloudFormation API.

8. Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. Using simple rules that you can quickly set up, you can match events and route them to one or more target functions or streams.

You can use Amazon CloudWatch Events to invoke the Lambda function on a recurring schedule of 15 minutes. This solution is entirely automated and serverless.

9. You can use Amazon CloudWatch Logs to monitor, store, and access your log files from Amazon Elastic Compute Cloud (Amazon EC2) instances, AWS CloudTrail, Route 53, and other sources.



CloudWatch Logs enables you to centralize the logs from all of your systems, applications, and AWS services that you use, in a single, highly scalable service. You can then easily view them, search them for specific error codes or patterns, filter them based on specific fields, or archive them securely for future analysis.

To collect logs from Amazon EC2 and on-premises instances it is necessary to install an agent. There are two options: the unified CloudWatch Agent which collects logs and advanced metrics (such as memory usage), or the older CloudWatch Logs agent which only collects logs from Linux servers.

10. AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management and secrets management.

You can store data such as passwords, database strings, and license codes as parameter values.

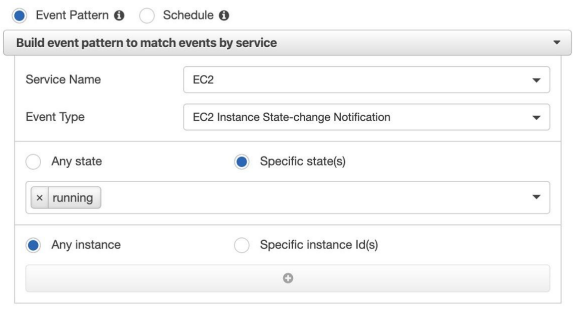
You can store values as plaintext (unencrypted data) or ciphertext (encrypted data). You can then reference values by using the unique name that you specified when you created the parameter.

A secure string parameter is any sensitive data that needs to be stored and referenced in a secure manner. If you have data that you don't want users to alter or reference in plaintext, such as passwords or license keys, create those parameters using the SecureString datatype.

If you choose the SecureString datatype when you create a parameter, then Parameter Store uses an AWS Key Management Service (KMS) customer master key (CMK) to encrypt the parameter value.

This is the most secure and operationally efficient way to meet this requirement. The connection string will be encrypted and only needs to be managed in one place where it can be shared by the multiple Lambda functions.

11. Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources. Using simple rules that you can quickly set up, you can match events and route them to one or more target functions or streams. CloudWatch Events becomes aware of operational changes as they occur. CloudWatch Events responds to these operational changes and takes corrective action as necessary, by sending messages to respond to the environment, activating functions, making changes, and capturing state information.



In this scenario the only workable solution is to create a CloudWatch Event with an event pattern looking for EC2 state changes and a target set to use the Lambda function.

12. AWS CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of your AWS account. With CloudTrail, you can log, continuously monitor, and retain account activity related to actions across your AWS infrastructure.

    CloudTrail provides event history of your AWS account activity, including actions taken through the AWS Management Console, AWS SDKs, command line tools, and other AWS services.

    This event history simplifies security analysis, resource change tracking, and troubleshooting. In addition, you can use CloudTrail to detect unusual activity in your AWS accounts. These capabilities help simplify operational analysis and troubleshooting.

    CloudWatch vs CloudTrail:

CloudWatch	CloudTrail
Performance monitoring	Auditing
Log events across AWS services – think operations	Log API activity across AWS services – think activities
Higher-level comprehensive monitoring and eventing	More low-level granular
Log from multiple accounts	Log from multiple accounts
Logs stored indefinitely	Logs stored to S3 or CloudWatch indefinitely
Alarms history for 14 days	No native alarming; can use CloudWatch alarms

    As this scenario requests that a history of API calls are retained (auditing), AWS CloudTrail is the correct solution to use.

13. You can publish your own metrics to CloudWatch using the AWS CLI or an API. You can view statistical graphs of your published metrics with the AWS Management Console.

    CloudWatch stores data about a metric as a series of data points. Each data point has an associated time stamp. You can even publish an aggregated set of data points called a *statistic set*.

    Each metric is one of the following:

• Standard resolution, with data having a one-minute granularity

• High resolution, with data at a granularity of one second

Metrics produced by AWS services are standard resolution by default. When you publish a custom metric, you can define it as either standard resolution or high resolution. When you publish a high-resolution metric, CloudWatch stores it with a resolution of 1 second, and you can read and retrieve it with a period of 1 second, 5 seconds, 10 seconds, 30 seconds, or any multiple of 60 seconds.

High-resolution metrics can give you more immediate insight into your application's sub-minute activity. Keep in mind that every PutMetricData call for a custom metric is charged, so calling PutMetricData more often on a high-resolution metric can lead to higher charges.

Therefore, the best action to take is to Create custom metrics and configure them as high resolution. This will ensure that granularity can be down to 1 second.

14. Of the options presented there are two workable procedures for deploying the Lambda function.

    Firstly, you can create an `AWS::Lambda::Function` resource in the template, then write the code directly inside the CloudFormation template. This is possible for simple functions using Node.js or Python which allow you to declare the code inline in the CloudFormation template. For example:

```
Resources:
  MyFunction:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        ZipFile: |
          import json
          def handler(event, context):
              print("Event: %s" % json.dumps(event))
```

The other option is to upload a ZIP file containing the function code to Amazon S3, then add a reference to it in an AWS::Lambda::Function resource in the template. To declare this in your AWS CloudFormation template, you can use the following syntax (within AWS::Lambda::Function Code):

```
{
  "S3Bucket" : String,
  "S3Key" : String,
  "S3ObjectVersion" : String,
  "ZipFile" : String
}
```

15. AWS Systems Manager Parameter Store provides secure, hierarchical storage for configuration data management and secrets management. You can store data such as passwords, database strings, and license codes as parameter values. It is highly scalable, available, and durable.

    You can store values as plaintext (unencrypted data) or ciphertext (encrypted data). You can then reference values by using the unique name that you specified when you created the parameter.

    There are no additional charges for using SSM Parameter Store. However, there are limit of 10,000 parameters per account

16. The template shown is an AWS SAM template for deploying a serverless application. This can be identified by the template header: *Transform: 'AWS::Serverless-2016-10-31'*

    The Developer will need to package and then deploy the template. To do this the source code must be available in the same directory or referenced using the “codeuri” parameter. Then, the Developer can use the “aws cloudformation package” or “sam package” commands to prepare the local artifacts (local paths) that your AWS CloudFormation template references.

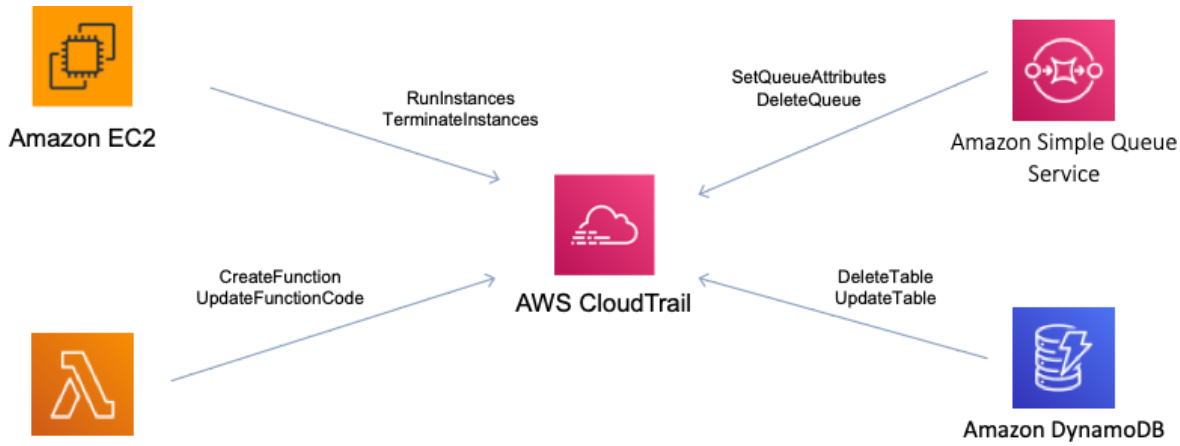
    The command uploads local artifacts, such as source code for an AWS Lambda function or a Swagger file for an AWS API Gateway REST API, to an S3 bucket. The command returns a copy of your template, replacing references to local artifacts with the S3 location where the command uploaded the artifacts.

    Once that is complete the template can be deployed using the “aws cloudformation deploy” or “sam deploy” commands. Therefore, the developer has two options to prepare and then deploy this package:

    1. Run `aws cloudformation package` and then aws cloudformation deploy

    2. Run `sam package` and then `sam deploy`

17. AWS CloudTrail is a web service that records activity made on your account. A CloudTrail trail can be created which delivers log files to an Amazon S3 bucket. CloudTrail is about logging and saves a history of API calls for your AWS account. It enables governance, compliance, and operational and risk auditing of your AWS account.



Therefore, AWS CloudTrail is the best solution for maintaining a log of API calls for the security team.

18. After the CloudWatch Logs agent begins publishing log data to Amazon CloudWatch, you can begin searching and filtering the log data by creating one or more metric filters. Metric filters define the terms and patterns to look for in log data as it is sent to CloudWatch Logs.

    CloudWatch Logs uses these metric filters to turn log data into numerical CloudWatch metrics that you can graph or set an alarm on. You can use any type of CloudWatch statistic, including percentile statistics, when viewing these metrics or setting alarms.

    Filters do not retroactively filter data. Filters only publish the metric data points for events that happen after the filter was created. Filtered results return the first 50 lines, which will not be displayed if the timestamp on the filtered results is earlier than the metric creation time.

    Therefore, the filtered results are not being returned as CloudWatch Logs only publishes metric data for events that happen after the filter is created.

19. A *namespace* is a container for CloudWatch metrics. Metrics in different namespaces are isolated from each other, so that metrics from different applications are not mistakenly aggregated into the same statistics.

    Therefore, the Developer should create a custom namespace with a unique metric name for each application. This namespace will then allow the metrics for each individual application to be shown in a single view through CloudWatch.

20. The template shown is an AWS SAM template for deploying a serverless application. This can be identified by the template header: *Transform: 'AWS::Serverless-2016-10-31'*

    The Developer will need to package and then deploy the template. To do this the source code must be available in the same directory or referenced using the “codeuri” parameter. Then, the Developer can use the “aws cloudformation package” or “sam package” commands to prepare the local artifacts (local paths) that your AWS CloudFormation template references.

    The command uploads local artifacts, such as source code for an AWS Lambda function or a Swagger file for an AWS API Gateway REST API, to an S3 bucket. The command returns a copy of your template, replacing references to local artifacts with the S3 location where the command uploaded the artifacts.

    Once that is complete the template can be deployed using the “aws cloudformation deploy” or “sam deploy” commands. Therefore, the next step in this scenario is for the Developer to run the “aws cloudformation” package command to upload the source code to an Amazon S3 bucket and produce a modified CloudFormation template. An example of this command is provided below:

    aws cloudformation package --template-file `/path_to_template/template.json` *--s3-bucket* `bucket-name` *--output-template-file* `packaged-template.json`

21. You can create *cross-account cross-Region dashboards*, which summarize your CloudWatch data from multiple AWS accounts and multiple Regions into one dashboard. From this high-level dashboard you can get a view of your entire application, and also drill down into more specific dashboards without having to log in and out of accounts or switch Regions.

    You can create cross-account cross-Region dashboards in the AWS Management Console and programmatically.

22. Occasionally ,you may receive the 400 ThrottlingException error for PutMetricData API calls in Amazon CloudWatch with a detailed response similar the following:

```
<ErrorResponse xmlns="http://monitoring.amazonaws.com/doc/2010-08-01/">
  <Error>
    <Type>Sender</Type>
    <Code>Throttling</Code>
    <Message>Rate exceeded</Message>
  </Error>
  <RequestId>2f85f68d-980b-11e7-a296-21716fd2d2e3</RequestId>
</ErrorResponse>
```

CloudWatch requests are throttled for each Amazon Web Services (AWS) account on a per-Region basis to help service performance. For current PutMetricData API request limits, see CloudWatch Limits.

All calls to the PutMetricData API in an AWS Region count towards the maximum allowed request rate. This number includes calls from any custom or third-party application, such as calls from the CloudWatch Agent, the AWS Command Line Interface (AWS CLI), or the AWS Management Console.

Resolutions: It's a best practice to use the following methods to reduce your call rate and avoid API throttling:

- Distribute your API calls evenly over time rather than making several API calls in a short time span. If you require data to be available with a one-minute resolution, you have an entire minute to emit that metric. Use jitter (randomized delay) to send data points at various times.

- Combine as many metrics as possible into a single API call. For example, a single PutMetricData call can include 20 metrics and 150 data points. You can also use pre-aggregated data sets, such as StatisticSet, to publish aggregated data points, thus reducing the number of PutMetricData calls per second.

- Retry your call with exponential backoff and jitter.

Following attempting the above resolutions AWS suggest the following: “If you still require a higher limit, you can request a limit increase. Increasing the rate limit can have a high financial impact on your AWS bill.”

Therefore, the first thing the Developer should do, from the list of options presented, is to retry the call with exponential backoff.

23. You can create a custom CloudWatch metric for your EC2 Linux instance statistics by creating a script through the AWS Command Line Interface (AWS CLI). Then, you can monitor that metric by pushing it to CloudWatch. In this scenario you could then monitor the number of users currently logged in.
24. The optional Mappings section matches a key to a corresponding set of named values. For example, if you want to set values based on a region, you can create a mapping that uses the region name as a key and contains the values you want to specify for each specific region. You use the Fn::FindInMap intrinsic function to retrieve values in a map.

    The following example shows a Mappings section with a map RegionMap, which contains five keys that map to name-value pairs containing single string values. The keys are region names. Each name-value pair is the AMI ID for the HVM64 AMI in the region represented by the key.

```
"Mappings" : {
  "RegionMap" : {
    "us-east-1"      : { "HVM64" : "ami-0ff8a91507f77f867"},
    "us-west-1"      : { "HVM64" : "ami-0bdb828fd58c52235"},
    "eu-west-1"      : { "HVM64" : "ami-047bb4163c506cd98"},
    "ap-southeast-1" : { "HVM64" : "ami-08569b978cc4dfa10"},
    "ap-northeast-1" : { "HVM64" : "ami-06cd52961ce9f0d85"}
  }
}
```