1. You can integrate an API method in your API Gateway with a custom HTTP endpoint of your application in two ways:- HTTP proxy integration - HTTP custom integration
2. With proxy integration, the setup is simple. You only need to set the HTTP method and the HTTP endpoint URI, according to the backend requirements, if you are not concerned with content encoding or caching.
3. With custom integration, setup is more involved. In addition to the proxy integration setup steps, you need to specify how the incoming request data is mapped to the integration request and how the resulting integration response data is mapped to the method response. API Gateway supports the following endpoint ports: 80, 443 and 1024-65535.
4. Programmatically, you choose an integration type by setting the [type](https://docs.aws.amazon.com/apigateway/api-reference/resource/integration/#type) property on the [Integration](https://docs.aws.amazon.com/apigateway/api-reference/resource/integration/) resource. For the Lambda proxy integration, the value is **AWS\_PROXY**. For the Lambda custom integration and all other AWS integrations, it is **AWS**. For the HTTP proxy integration and HTTP integration, the value is **HTTP\_PROXY** and **HTTP**, respectively. For the mock integration, the type value is **MOCK**.
5. To avoid potential throttling, the provisioned write capacity for a global secondary index should be equal or greater than the write capacity of the base table since new updates will write to both the base table and global secondary index.
6. With an atomic counter, the numeric value will increment each time you call UpdateItem.
7. To assume a role, an application calls the AWS STS AssumeRole API operation and passes the ARN of the role to use. The operation creates a new session with temporary credentials. This session has the same permissions as the identity-based policies for that role.
8. **AssumeRoleWithWebIdentity**  returns a set of temporary security credentials for federated users who are authenticated through public identity providers
9. To create a Lambda function, you first create a Lambda function deployment package, a .zip or .jar file consisting of your code and any dependencies.
10. If you are using a CloudFormation template, you can configure the AWS::Lambda::Function resource which creates a Lambda function. To create a function, you need a deployment package and an execution role. The deployment package contains your function code. The execution role grants the function permission to use AWS services, such as Amazon CloudWatch Logs for log streaming and AWS X-Ray for request tracing.
11. Hence, ***including your function source inline in the ZipFile parameter of the AWS::Lambda::Function resource in the CloudFormation template*** is the easiest way to deploy the Lambda function to AWS.
12. *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 DynamoDB. If you use this strategy, then your database writes are protected from being overwritten by the writes of others — and vice-versa.
13. ***Implementing pessimistic locking with read locking***is incorrect because this type of locking can interrupt user operations. This is an approach where an entity is locked in the database for the entire time that it is in application memory (often in the form of an object). This can prevent certain use
14. You can use the --page-size option to specify that the AWS CLI request a smaller number of items from each call to the AWS service. The CLI still retrieves the full list, but performs a larger number of service API calls in the background and retrieves a smaller number of items with each call. This gives the individual calls a better chance of succeeding without a timeout.
15. To include fewer items at a time in the AWS CLI output, use the --max-items option. The AWS CLI still handles pagination with the service as described above, but prints out only the number of items at a time that you specify. If the number of items output is fewer than the total number of items returned by the underlying API calls, the output includes a *NextToken* that you can pass to a subsequent command to retrieve the next set of items.
16. X-Ray compiles and processes segment documents to generate queryable **trace summaries** and **full traces** that you can access by using the [GetTraceSummaries](https://docs.aws.amazon.com/xray/latest/api/API_GetTraceSummaries.html) and [BatchGetTraces](https://docs.aws.amazon.com/xray/latest/api/API_BatchGetTraces.html) APIs, respectively. In addition to the segments and subsegments that you send to X-Ray, the service uses information in subsegments to generate **inferred segments** and adds them to the full trace. Inferred segments represent downstream services and resources in the service map.
17. inline policies are useful if you want to maintain a strict one-to-one relationship between a policy and the principal entity that it's applied to.
18. By default, a Scan operation returns all of the data attributes for every item in the table or index. You can use the ProjectionExpression parameter so that Scan only returns some of the attributes, rather than all of them. On the other hand, 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).
19. DynamoDB calculates the number of read capacity units consumed based on item size, not on the amount of data that is returned to an application.
20. In the AWS Lambda resource model, you choose the amount of memory you want for your function and are allocated proportional CPU power and other resources. An increase in memory size triggers an equivalent increase in CPU available to your function.
21. The IAM policy simulator evaluates the policies that you choose and determines the effective permissions for each of the actions that you specify.
22. **Time To Live (TTL)** for DynamoDB allows you to define when items in a table expire so that they can be automatically deleted from the database.
23. AWS Lambda will keep the unreserved concurrency pool at a minimum of 100 concurrent executions, so that functions that do not have specific limits set can still process requests. So, in practice, if your total account limit is 1000, you are limited to allocating 900 to individual functions.
24. 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](https://docs.aws.amazon.com/lambda/latest/dg/runtimes-custom.html), or other dependencies. With layers, you can use libraries in your function without needing to include them in your deployment package.
25. A function can use up to 5 layers at a time. The total unzipped size of the function and all layers can't exceed the unzipped deployment package size limit of 250 MB.
26. CodeDeploy agent is required only if you deploy to an EC2/On-Premises compute platform. The agent is not required for deployments that use the Amazon ECS or AWS Lambda compute platform.
27. A Lambda authorizer is an API Gateway feature that uses a Lambda function to control access to your API. 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.
28. 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.
29. 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.
30. The **Kinesis Adapter** is the recommended way to consume streams from DynamoDB for real-time processing.
31. When an item in the table is modified, StreamViewType determines what information is written to the stream for this table. Valid values for StreamViewType are:
32. **KEYS\_ONLY** - Only the key attributes of the modified item are written to the stream.
33. **NEW\_IMAGE** - The entire item, as it appears after it was modified, is written to the stream.
34. **OLD\_IMAGE** - The entire item, as it appeared before it was modified, is written to the stream.
35. **NEW\_AND\_OLD\_IMAGES** - Both the new and the old item images of the item are written to the stream.
36. When you query a local secondary index, you can choose either eventual consistency or strong consistency. GSI only supports eventual consistency and not strong consistency.
37. ***Using a Lazy Loading caching strategy***is a strategy to load data into the cache only when necessary.
38. The number of concurrent executions for poll-based event sources is different from push-based event sources. Remember that the Kinesis and Lambda integration is using a poll-based event source, which means that the number of shards is the unit of concurrency for the function.
39. **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.
40. Optimistic locking strategy simply ensures that the client-side item that you are updating (or deleting) is the same as the item in DynamoDB.
41. 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.
42. To **decrypt** data locally:**1. Use the**[**Decrypt**](https://docs.aws.amazon.com/kms/latest/APIReference/API_Decrypt.html)**operation to decrypt the encrypted data key. The operation returns a plaintext copy of the data key. 2. Use the plaintext data key to decrypt data locally, then erase the plaintext data key from memory.**
43. CodeDeploy provides two deployment type options:

**In-place deployment**: The application on each instance in the deployment group is stopped, the latest application revision is installed, and the new version of the application is started and validated. You can use a load balancer so that each instance is deregistered during its deployment and then restored to service after the deployment is complete. Only deployments that use the EC2/On-Premises compute platform can use in-place deployments. AWS Lambda compute platform deployments cannot use an in-place deployment type.

**Blue/green deployment**: The behavior of your deployment depends on which compute platform you use:

- Blue/green on an EC2/On-Premises compute platform: The instances in a deployment group (the original environment) are replaced by a different set of instances (the replacement environment). If you use an EC2/On-Premises compute platform, be aware that blue/green deployments work with Amazon EC2 instances only.

- Blue/green on an AWS Lambda compute platform: Traffic is shifted from your current serverless environment to one with your updated Lambda function versions. You can specify Lambda functions that perform validation tests and choose the way in which the traffic shift occurs. All AWS Lambda compute platform deployments are blue/green deployments. For this reason, you do not need to specify a deployment type.

- Blue/green on an Amazon ECS compute platform: Traffic is shifted from the task set with the original version of a containerized application in an Amazon ECS service to a replacement task set in the same service. The protocol and port of a specified load balancer listener are used to reroute production traffic. During deployment, a test listener can be used to serve traffic to the replacement task set while validation tests are run.

The CodeDeploy agent is a software package that, when installed and configured on an instance, makes it possible for that instance to be used in CodeDeploy deployments. The CodeDeploy agent communicates outbound using HTTPS over port 443.

It is also important to note that the CodeDeploy agent is required only if you deploy to an EC2/On-Premises compute platform. The agent is not required for deployments that use the Amazon ECS or AWS Lambda compute platform.

1. Monitor the IntegrationLatency metrics to measure the responsiveness of the backend.

- Monitor the Latency metrics to measure the overall responsiveness of your API calls.

- Monitor the CacheHitCount and CacheMissCount metrics to optimize cache capacities to achieve a desired performance. CacheMissCount tracks the number of requests served from the backend in a given period, when API caching is enabled. On the other hand, CacheHitCount track the number of requests served from the API cache in a given period.

1. 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
2. Change Sets only allow you to preview how proposed changes to a stack might impact your running resources.
3. CloudWatch does not monitor the memory, swap, and disk space utilization of your instances. If you need to track these metrics, you can install a CloudWatch agent in your EC2 instances.
4. Your sign-in page URL has the following format, by defaull [https://Your\_AWS\_Account\_ID.signin.aws.amazon.com/console/](https://your_aws_account_id.signin.aws.amazon.com/console/)
5. If you create an AWS account alias for your AWS account ID, your sign-in page URL looks like the following example. [https://Your\_Alias.signin.aws.amazon.com/console/](https://your_alias.signin.aws.amazon.com/console/)
6. 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.
7. Parameters stored in Systems Manager are mutable. Any time you use a template containing Systems Manager parameters to create/update your stacks, CloudFormation uses the values for these Systems Manager parameters at the time of the create/update operation. So, as parameters are updated in Systems Manager, you can have the new value of the parameter take effect by just executing a stack update operation.
8. Amazon CloudWatch Events delivers a near real-time stream of system events that describe changes in Amazon Web Services (AWS) resources, it cannot provide tracking of the DynamoDB's table activities.
9. **INTEGRATION\_FAILURE** - The gateway response for an integration failed error. If the response type is unspecified, this response defaults to the DEFAULT\_5XX type.
10. **INTEGRATION\_TIMEOUT** - The gateway response for an integration timed out error. If the response type is unspecified, this response defaults to the DEFAULT\_5XX type
11. For the integration timeout, the range is from 50 milliseconds to 29 seconds for all integration types, including Lambda, Lambda proxy, HTTP, HTTP proxy, and AWS integrations.
12. "There is an authorization failure occurring between API Gateway and the Lambda function." is incorrect because an authentication issue usually produces HTTP 403 errors and not 504s. The gateway response for authorization failures for missing authentication token error, invalid AWS signature error, or Amazon Cognito authentication problems is HTTP 403.
13. Amazon OpenSearch Service can be used for full-text search, structured search, analytics, and all three in combination.
14. Cloud trail data are easy to send to CloudWatch logs for real time purpose or can use s3 bucket which will take several minutes.
15. put-bucket-policy command can only be used to apply policy at the bucket level, not on objects.
16. CloudFormation can certainly be used to deploy Lambda, API Gateway, DynamoDB, and other AWS resources of your serverless application, it doesn't have the capability to locally build, test, and debug your application like what AWS SAM has.
17. Configure an SQS Delay Queue with a value of 40 seconds" is primarily configured if you want to postpone the delivery of new messages to the SQS queue for a number of seconds.
18. AWS X-Ray is an AWS service that allows you to detect, analyze, and optimize performance issues with your AWS Lambda applications.
19. Global secondary index — an index with a partition key and a sort key that can be different from those on the base table. A global secondary index is considered "global" because queries on the index can span all of the data in the base table, across all partitions.
20. Local secondary index — an index that has the same partition key as the base table, but a different sort key. A local secondary index is "local" in the sense that every partition of a local secondary index is scoped to a base table partition that has the same partition key value.
21. **RequestResponse** (default) - Invoke the function synchronously. Keep the connection open until the function returns a response or times out. The API response includes the function response and additional data.
22. Event - Invoke the function asynchronously. Send events that fail multiple times to the function's dead-letter queue (if it's configured). The API response only includes a status code.
23. **DryRun** - Validate parameter values and verify that the user or role has permission to invoke the function.
24. using Query operations instead of Scan and reducing the page size re the more cost-effective solutions to improve the performance of the application using DynamoDB
25. All at once – Deploy the new version to all instances simultaneously. All instances in your environment are out of service for a short time while the deployment occurs.
26. Rolling – Deploy the new version in batches. Each batch is taken out of service during the deployment phase, reducing your environment's capacity by the number of instances in a batch.
27. Rolling with additional batch – Deploy the new version in batches, but first launch a new batch of instances to ensure full capacity during the deployment process.
28. Immutable – Deploy the new version to a fresh group of instances by performing an immutable update.
29. Blue/Green - Deploy the new version to a separate environment, and then swap CNAMEs of the two environments to redirect traffic to the new version instantly.
30. 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.
31. 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.
32. To define a nested application in your serverless application, use the AWS::Serverless::Application resource type.
33. Monitor the **IntegrationLatency** metrics to measure the responsiveness of the backend. Monitor the **Latency** metrics to measure the overall responsiveness of your API calls. Monitor the **CacheHitCount** and **CacheMissCount** metrics to optimize cache capacities to achieve a desired performance.
34. To read data from a table, you use operations such as GetItem, Query, or Scan. DynamoDB returns all of the item attributes by default. To get just some, rather than all of the attributes, use a projection expression.
35. You can use the **--page-size** option to specify that the AWS CLI request a smaller number of items from each call to the AWS service.
36. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools enable you to grant your users access to other AWS services. You can use identity pools and user pools separately or together.
37. Amazon Cognito identity pools (federated identities) support user authentication through Amazon Cognito user pools, federated identity providers—including Amazon, Facebook, Google, and SAML identity providers—as well as unauthenticated identities. This feature also supports Developer Authenticated Identities (Identity Pools), which lets you register and authenticate users via your own back-end authentication process.
38. Programmatically, you choose an integration type by setting the type property on the Integration resource. For the Lambda proxy integration, the value is AWS\_PROXY. For the Lambda custom integration and all other AWS integrations, it is AWS. For the HTTP proxy integration and HTTP integration, the value is HTTP\_PROXY and HTTP, respectively. For the mock integration, the type value is MOCK.
39. Implement the **routing-config** parameter of the Lambda alias that allows you to point to two different versions of the Lambda function and dictate what percentage of incoming traffic is sent to each version.
40. For serverless applications (also referred to as Lambda-based applications), the optional Transform section specifies the version of the AWS Serverless Application Model (AWS SAM) to use.
41. The **InvalidParameterValueException** will be returned if one of the parameters in the request is invalid.
42. Annotations are simple key-value pairs that 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. X-Ray indexes up to 50 annotations per trace.
43. INTEGRATION\_FAILURE error if your AWS Lambda integration does not work at all in the first place.
44. INTEGRATION\_TIMEOUT - The gateway response for an integration timed out error. If the response type is unspecified, this response defaults to the DEFAULT\_5XX type.
45. For the integration timeout, the range is from 50 milliseconds to 29 seconds for all integration types, including Lambda, Lambda proxy, HTTP, HTTP proxy, and AWS integrations.
46. You can monitor the MySQL error log, slow query log, and the general log. The MySQL error log is generated by default; you can generate the slow query and general logs by setting parameters in your DB parameter group. Amazon RDS rotates all of the MySQL log files; the intervals for each type are given following.
47. Setting up elastic network interfaces (ENIs) to enable your Lambda function to connect securely to other resources within your private VPC is incorrect because this is already done automatically by AWS Lambda.
48. 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.
49. With synchronous invocation, you wait for the function to process the event and return a response. With asynchronous invocation, Lambda queues the event for processing and returns a response immediately. For asynchronous invocation, Lambda handles retries and can send invocation records to a destination.
50. When you invoke a function synchronously, Lambda runs the function and waits for a response. When the function completes, Lambda returns the response from the function's code with additional data, such as the version of the function that was invoked.
51. 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.
52. Synchronous invocation -> we want to know the result (response)
53. Asynchronous invocation -> we don’t want to know the result (response)
54. If lambda function is polling for sqs, lambda needs to have execution IAM role to do that.
55. If s3 invokes lambda, lambda needs to have RBP with principal -> s3 and source being s3 bucket.
56. API integrating with Lambda proxy to get the full event back from the API gateway.
57. Authentication verifies the identity of a user or service, and authorization determines their access rights. Authentication is done before the authorization process.
58. A segment can break down the data about the work done into **subsegments.** Subsegments provide more granular timing information and details about downstream calls that your application made to fulfill the original request. A subsegment can contain additional details about a call to an AWS service, an external HTTP API, or an SQL database. You can even define arbitrary subsegments to instrument specific functions or lines of code in your application.
59. When using server-side encryption with customer-provided encryption keys (SSE-C), you must provide encryption key information using the following request headers:
60. **x-amz-server-side-encryption-customer-algorithm** - This header specifies the encryption algorithm. The header value must be "AES256".
61. **x-amz-server-side-encryption-customer-key** - This header provides the 256-bit, base64-encoded encryption key for Amazon S3 to use to encrypt or decrypt your data.
62. x**-amz-server-side-encryption-customer-key-MD5** - This header provides the base64-encoded 128-bit MD5 digest of the encryption key according to RFC 1321. Amazon S3 uses this header for a message integrity check to ensure the encryption key was transmitted without error.
63. **x-amz-server-side-encryption** and **x-amz-server-side-encryption-aws-kms-key-id** headers in the upload request headers are primarily used in Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS) and not for Server-Side Encryption with Customer-Provided Keys (SSE-C).
64. The **GetSessionToken** API 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 like Amazon EC2 StopInstances. MFA-enabled IAM users would need to call GetSessionToken and submit an MFA code that is associated with their MFA device.
65. Immutable is incorrect because this is only applicable in Elastic Beanstalk and not for Lambda.
66. AWS X-Ray receives data from services as segments. X-Ray then groups segments that have a common request into traces. X-Ray processes the traces to generate a service graph that provides a visual representation of your application.
67. AWS X-Ray will fetch the client IP address from the **X-Forwarded-For** header of the request.
68. For applications that need to read or write multiple items, DynamoDB provides the B**atchGetItem** and **BatchWriteItem** operations.
69. You can use the **UpdateItem** operation to implement an atomic counter — a numeric attribute that is incremented, unconditionally, without interfering with other write requests.
70. An atomic counter would not be appropriate where overcounting or undercounting cannot be tolerated (For example, in a banking application). In this case, it is safer to use a conditional update instead of an atomic counter.
71. Envelope encryption is the practice of encrypting plaintext data with a data key and then encrypting the data key under another key.
72. 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.
73. In the Invoke API, you have 3 options to choose from for the InvocationType:
74. RequestResponse (default) - Invoke the function synchronously. Keep the connection open until the function returns a response or times out. The API response includes the function response and additional data.
75. Event - Invoke the function asynchronously. Send events that fail multiple times to the function's dead-letter queue (if it's configured). The API response only includes a status code.
76. DryRun - Validate parameter values and verify that the user or role has permission to invoke the function.
77. AWS Lambda will keep the unreserved concurrency pool at a minimum of 100 concurrent executions, so that functions that do not have specific limits set can still process requests. So, in practice, if your total account limit is 1000, you are limited to allocating 900 to individual functions
78. create a Docker image that runs the X-Ray daemon, upload it to a Docker image repository, and then deploy it to your Amazon ECS cluster. In addition, you also have to configure the port mappings and network mode settings in your task definition file to allow traffic on UDP port 2000.
79. The objective is to publish two live endpoints (one for version 1 and another for version 2) that the users can access. Therefore, we need to create two stages. Deploying the new version to the same stage would overwrite the old version.
80. 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.
81. The GenerateDataKey generates a unique symmetric data key for client-side encryption. 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.
82. When a state reports an error and either there is no Retry field, or if retries fail to resolve the error, Step Functions scans through the catchers in the order listed in the array. When the error name appears in the value of a catcher's ErrorEquals field, the state machine transitions to the state named in the Next field.

Task and Parallel states can have a field named **Retry**, whose value must be an array of objects known as *retriers*. An individual retrier represents a certain number of retries, usually at increasing time intervals.

A retrier contains the following fields:

**ErrorEquals**

A non-empty array of strings that match error names. When a state reports an error, Step Functions scans through the retriers. When the error name appears in this array, it implements the retry policy described in this retrier.

**IntervalSeconds**

An integer that represents the number of seconds before the first retry attempt (1 by default).

**MaxAttempts**

A positive integer that represents the maximum number of retry attempts (3 by default). If the error recurs more times than specified, retries cease and normal error handling resumes. A value of 0 specifies that the error or errors are never retried.

**BackoffRate**

The multiplier by which the retry interval increases during each attempt (2.0 by default)

Task and Parallel states can have a field named **Catch**. This field's value must be an array of objects, known as *catchers*.

A catcher contains the following fields.

**ErrorEquals**

A non-empty array of strings that match error names, specified exactly as they are with the retrier field of the same name.

**Next**

A string that must exactly match one of the state machine's state names.

**ResultPath**

A path that determines what input is sent to the state specified in the Next field.

1. When Lambda invokes your function handler, the Lambda runtime passes two arguments to the function handler. The first argument is the event object. An event is a JSON-formatted document that contains data for a Lambda function to process. The Lambda runtime converts the event to an object and passes it to your function code. The event object contains information from the invoking service. When you invoke a function, you determine the structure and contents of the event. When an AWS service invokes your function, the service defines the event structure.
2. The second argument is the **context object**. A context object is passed to your function by Lambda at runtime. This object provides methods and properties that provide information about the invocation, function, and runtime environment.
3. 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.
4. Standard generic and preconfigured Docker platforms on Elastic Beanstalk support only a single Docker container per Elastic Beanstalk environment. In order to get the most out of Docker, Elastic Beanstalk lets you create an environment where your Amazon EC2 instances run multiple Docker containers side by side.
5. Container instances—Amazon EC2 instances running Multicontainer Docker in an Elastic Beanstalk environment—require a configuration file named **Dockerrun.aws.json**. This file is specific to Elastic Beanstalk and can be used alone or combined with source code and content in a source bundle to create an environment on a Docker platform.
6. AWS Lambda natively supports Java, Go, PowerShell, Node.js, C#, Python, and Ruby code, and provides a Runtime API, which allows you to use any additional programming languages to author your functions. You can use the custom runtime to create a Lambda function if your preferred language is not available. You can include a runtime in your function's deployment package in the form of an executable file named bootstrap.
7. You can pass two types of user data to Amazon EC2: shell scripts and cloud-init directives.
8. 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.
9. 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.
10. Trust policies define which principal entities (accounts, users, roles, and federated users) can assume the role.
11. The --dry-run parameter checks whether you have the required permissions for the action, without actually making the request, and provides an error response. If you have the required permissions, the error response is DryRun-Operation. Otherwise, it is UnauthorizedOperation.
12. To run CodeBuild with a proxy server, you install a proxy server in a public subnet and CodeBuild in a private subnet in a VPC.