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.