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.