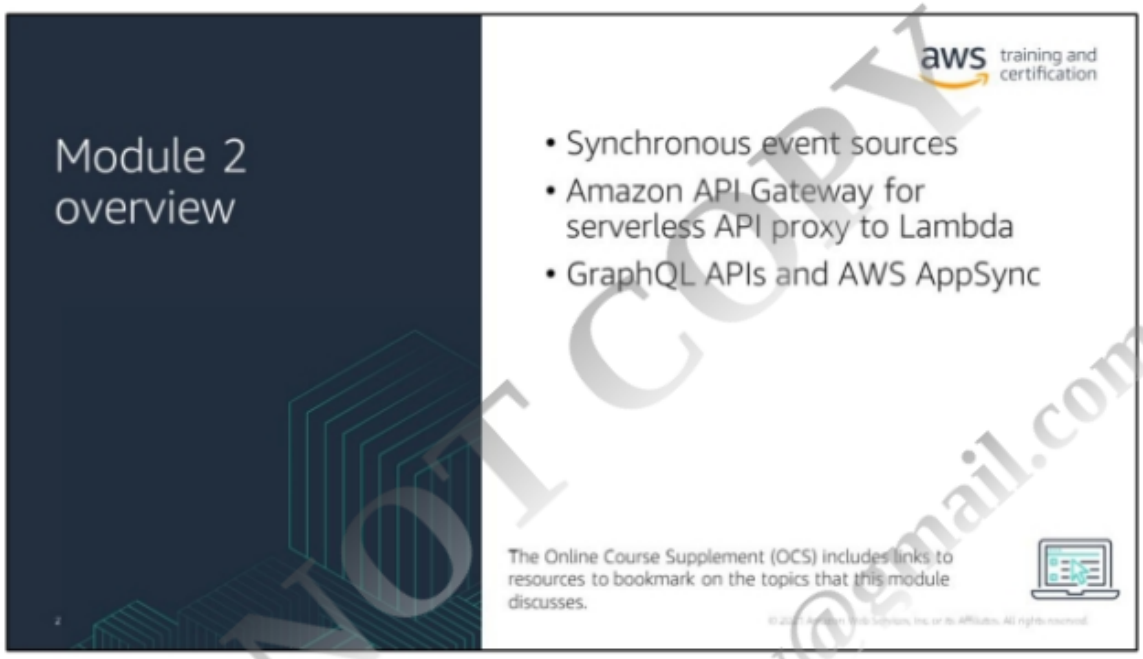
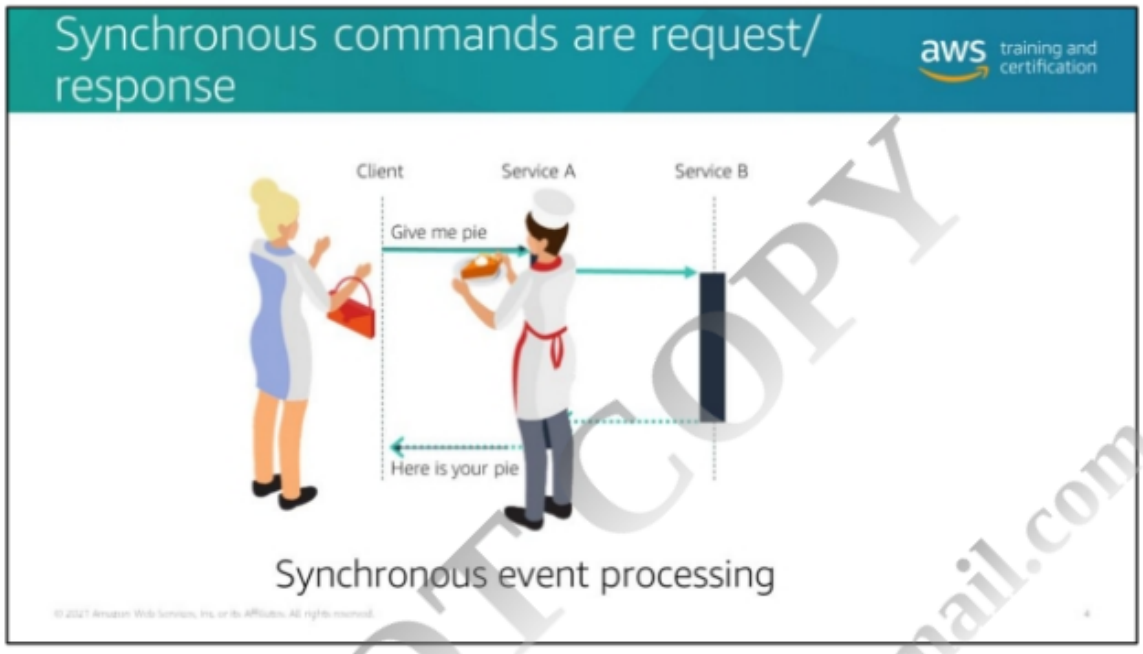


As noted earlier, an event source initiates a Lambda invocation.

You can trigger Lambda functions synchronously or asynchronously or for polling events through an event mapping.

When you choose an Amazon Web Services (AWS) service as an event source, the service determines how Lambda is invoked.

Let's look at synchronous events first.



In a synchronous model, the client makes a request and waits for a response. This is a relatively simple interaction common to create, read, update, and delete (CRUD) API actions.

If the client makes a request to service A, service A turns around and calls service B. Service A then waits for service B to finish doing whatever it has to do before it continues and eventually responds to service A. That response then goes back to the client.

The advantage of this approach is the immediate response to your API call. You can also use the API to control aspects of the request and response. Examples are validating characteristics of the request before the consumer processes it or reformatting the response before passing it off to the client.

The disadvantage is that an error or delay downstream holds up the whole process. Problems that happen in service B result in an error or a long wait time for the client.

In the bakery example, a synchronous event might be the customer buying a pie from within a display case. They make the request and receive the pie immediately in response before the next customer is served. The client makes a request and receives results immediately.

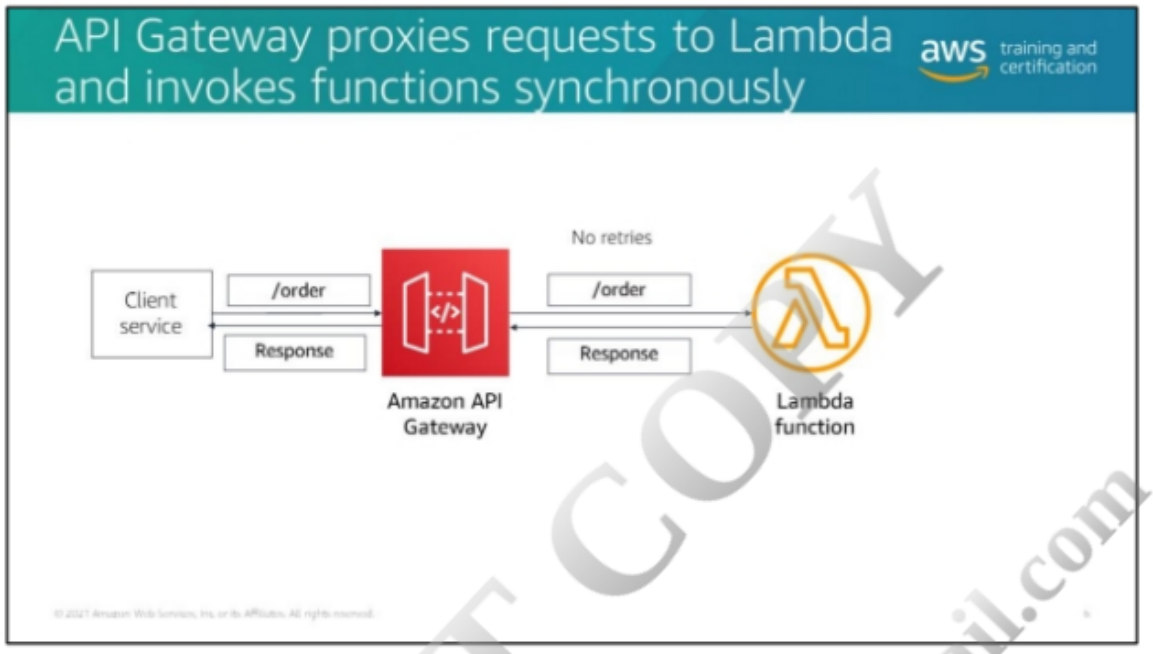
In this course you, use Amazon API Gateway as a synchronous event source.

API Gateway is a fully managed service that helps you manage your APIs.

API Gateway is often the front door to a serverless application. It can proxy API requests to Lambda and some other AWS services.

It can also be a good way to route some traffic to your new serverless component while routing other traffic to a legacy system.

The OCS has links to digital training courses and webinars that go deeper in to all of the features of API Gateway. You'll cover a few key points here before setting up an API using the AWS Management Console.



In a synchronous invocation, the Lambda service sends the event directly to the function and then sends the response back to the invoker.

There are no retries built in, and the requesting event must wait for the response. This means it's the developer's job to write any error handling into the producer service or client that's making the request. To invoke a function synchronously with the AWS CLI, use the invoke command with the RequestResponse invoke type.

When you use the AWS command line interface (CLI) and AWS software development kit (SDK) to make your requests, use the built-in options for retrying on client timeouts, throttling, and service errors.

API Gateway has rich integration with Lambda.

API Gateway invokes Lambda functions synchronously and has a built-in timeout of 30 seconds. If the Lambda service cannot successfully invoke the function in 30 seconds or if the function runs but generates an error, the error is returned to the API and the calling client.

For more information about API Gateway concepts, see https://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway-basic-concept.html.

For more information about using Lambda with API Gateway, see https://docs.aws.amazon.com/lambda/latest/dg/services-apigateway.html.

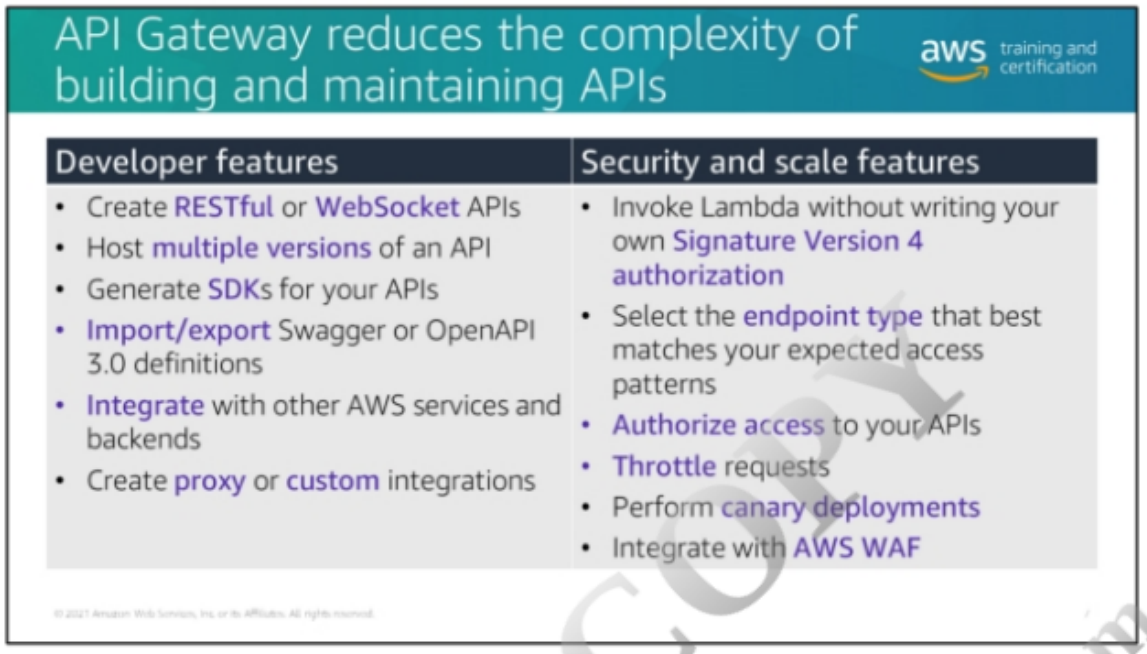
For more information about a list of AWS services that invoke Lambda synchronously, see https://docs.aws.amazon.com/lambda/latest/dg/lambda-services.html.

For more details about synchronous invocations, see the following information:

AWS Lambda Developer Guide
- Synchronous invocations: https://docs.aws.amazon.com/lambda/latest/dg/invocation-sync.html
- API reference: https://docs.aws.amazon.com/lambda/latest/dg/API_Invoke.html

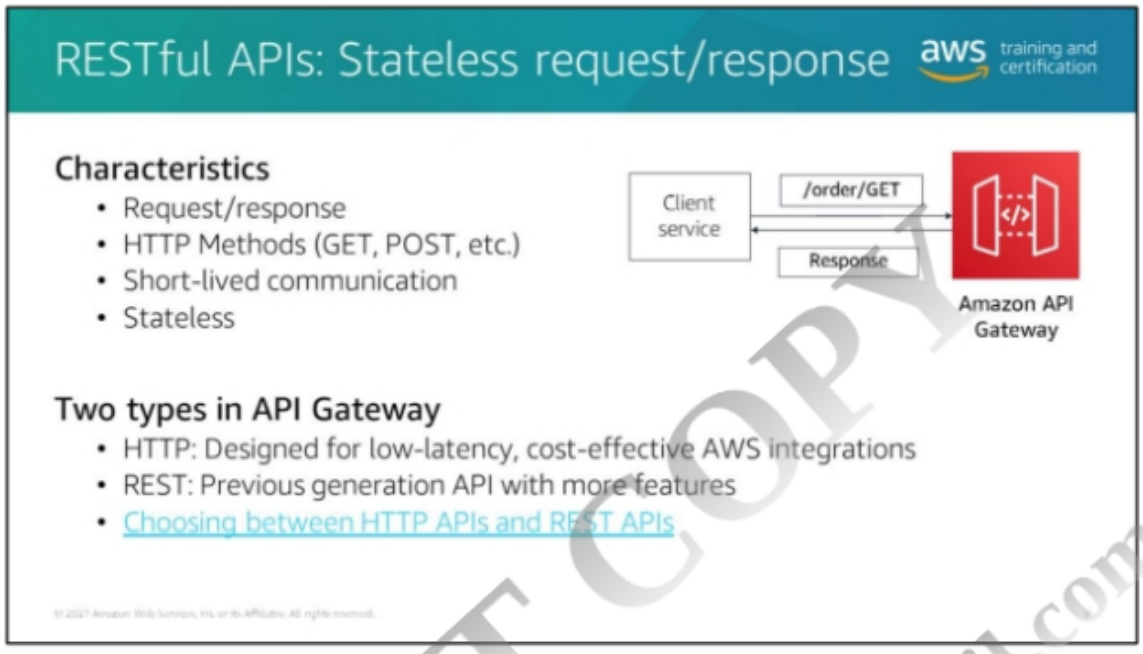
AWS Architecture Blog
- Understanding the Different Ways to Invoke Lambda Functions: https://aws.amazon.com/blogs/architecture/understanding-the-different-ways-to-invoke-lambda-functions/

These links are also available in the OCS.



API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. It frees you from the operational burden of implementation, offers reliable network protection, and centralizes authorization decisions within policies so bugs and code concerns are minimized.

The main security benefit that API Gateway provides is performing AWS Identity and Access Management (IAM) signature invokes on the client's behalf. Without this, direct calls to Lambda (or any AWS service) require you to write Signature Version 4 authorization.



As noted, API Gateway supports REST APIs and WebSocket APIs.

REST APIs provide stateless request/response APIs that use standard HTTP methods.

API Gateway lets you choose from two versions of REST APIs: REST or HTTP.

HTTP APIs are designed for low-latency, cost-effective integrations with AWS services, including AWS Lambda, and HTTP endpoints.

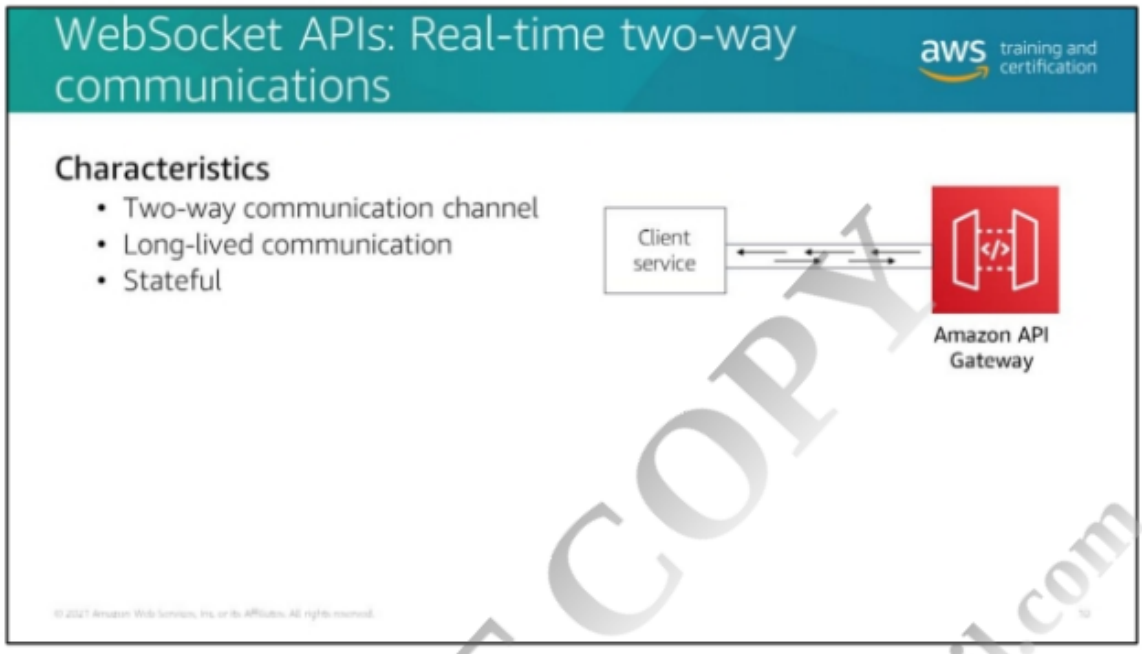
The REST API type gives you full control over API requests and responses and has some features that HTTP APIs do not.

For more information about choosing between HTTP APIs and REST APIs, see https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api-vs-rest.html.

For more information about working with HTTP APIs, see https://docs.aws.amazon.com/apigateway/latest/developerguide/http-api.html.

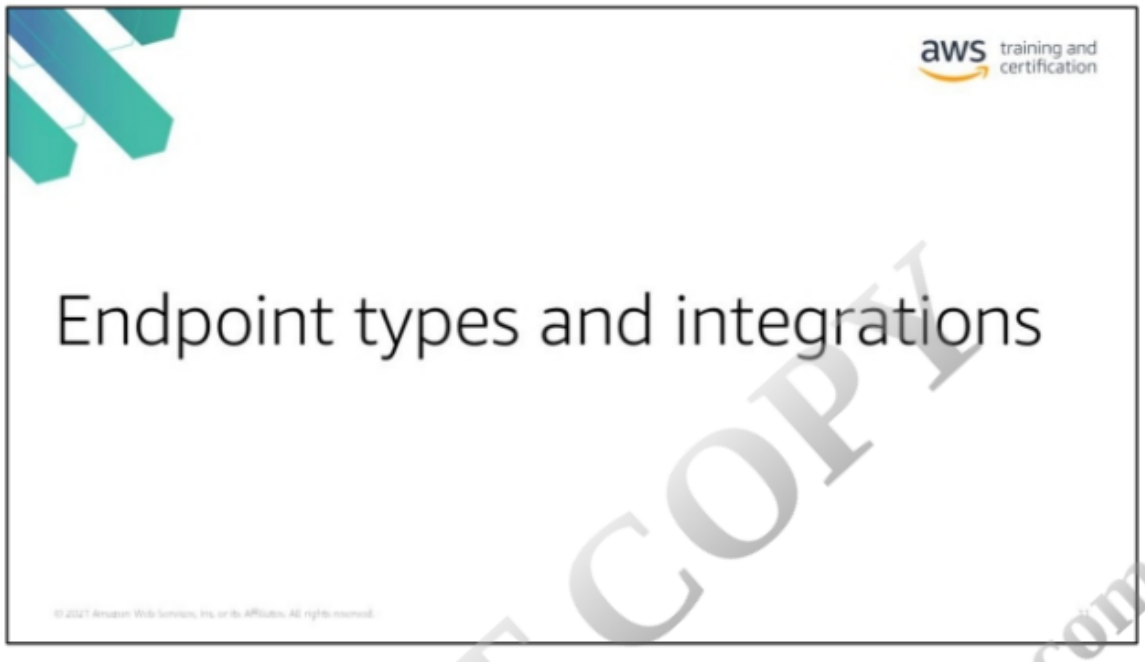
For more information about working with REST APIs, see https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-rest-api.html.

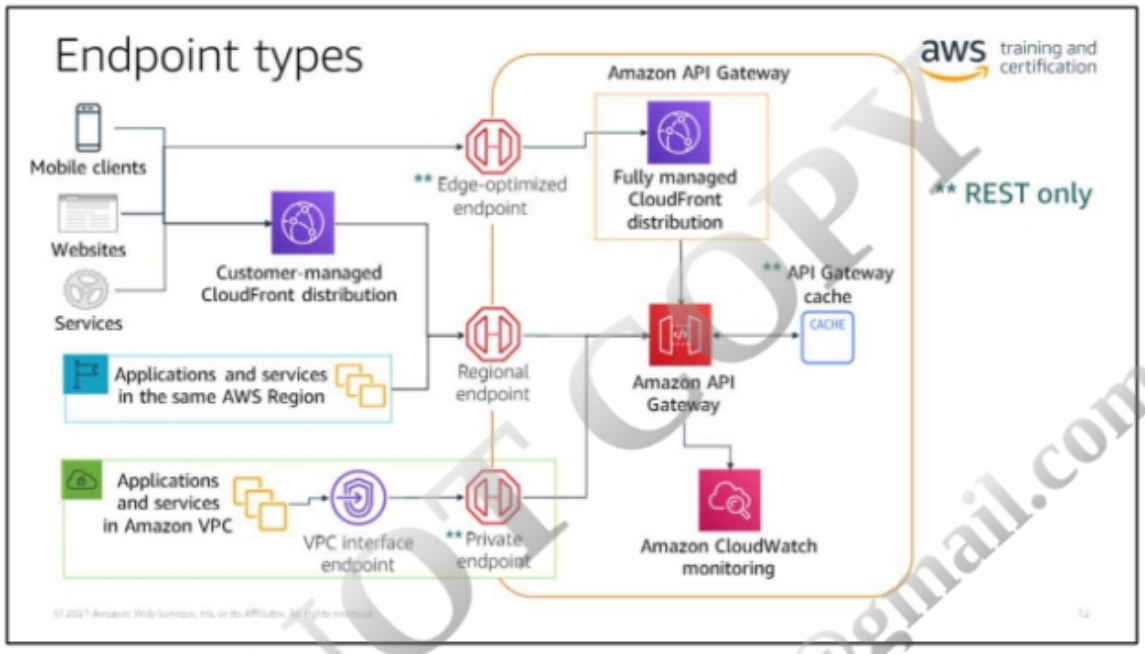
These links are also available in the OCS.



WebSocket APIs allow you to build real-time two-way communication applications, such as chat apps and streaming dashboards. API Gateway maintains a persistent connection to handle message

For more information about working with WebSocket APIs, see https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-websocket-api.html.





Let's go slightly deeper into the How API Gateway Works diagram presented on the https://aws.amazon.com/api-gateway/ page.

API Gateway handles all the tasks involved in accepting and processing API calls.

All of the API types (REST, HTTP, and WebSocket) support Regional endpoints. Regional endpoints are recommended for general use cases and are designed for building APIs for clients that are in the same AWS Region.

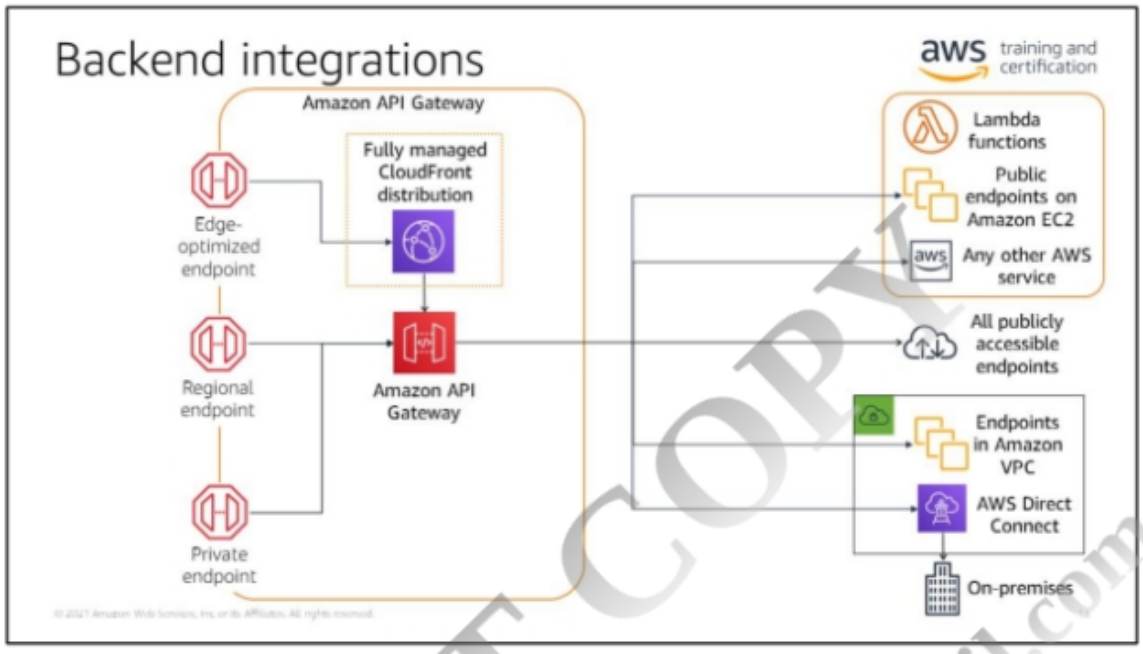
API Gateway also provides configurable logging and metrics via Amazon CloudWatch for all API types.

As the diagram illustrates, it is a best practice to set up an Amazon CloudFront distribution in front of Regional endpoints. These endpoints will be accessed by websites and other externally focused services to protect against distributed denial of service (DDoS) attacks and to reduce the roundtrip time for requests and responses.

REST APIs also have an optional cache to reduce backend load and reduce latency when serving recurring requests.

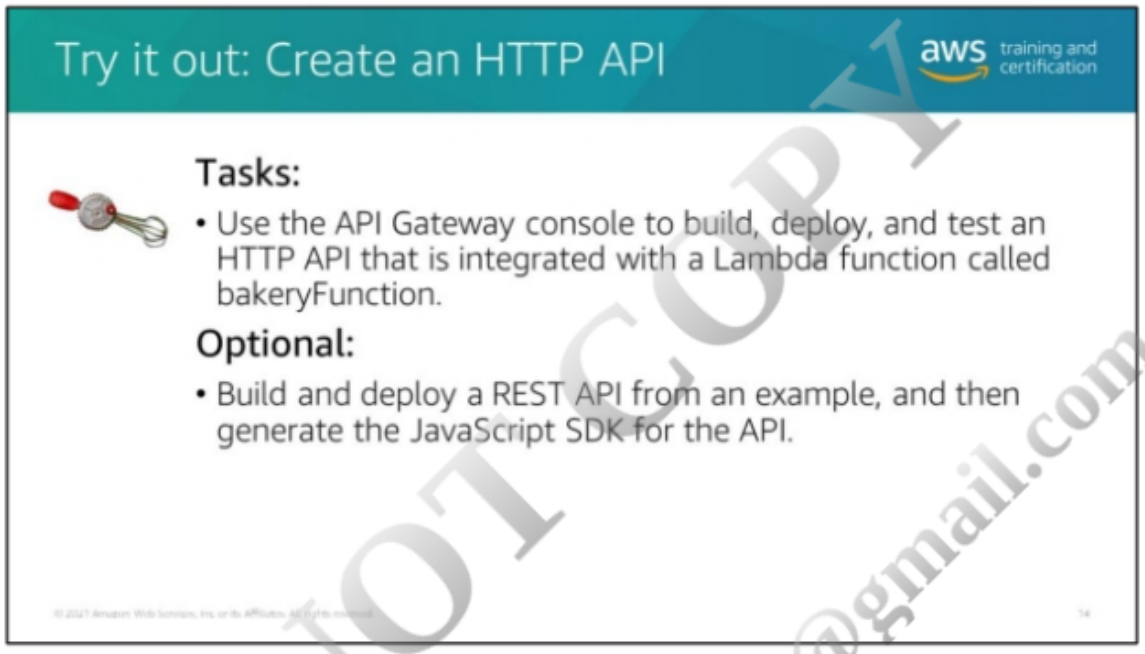
With REST APIs, you also have the option of an edge-optimized endpoint. With this endpoint type, API Gateway uses its own CloudFront distribution to reduce roundtrip time for your requests and responses. This endpoint type is designed for globally distributed clients and gives you built-in DDoS protection via its CloudFront distribution without your needing to set up a separate CloudFront distribution. This option is not available for HTTP or WebSocket APIs.

REST APIs also provide the option of private APIs that are accessible from only within your Amazon virtual private cloud (VPC). This endpoint type is designed for building APIs that are used internally or by private microservices. The Serverless Application Security module also discusses this information.



Regardless of the endpoint type you choose for your API, you can integrate the API directly with Lambda and some other AWS services using API Gateway as a proxy to those services.

With both REST and HTTP APIs, you also have the ability to reach endpoints within your Amazon VPC. You will review this information in a bit more detail in the Serverless Application Security module.



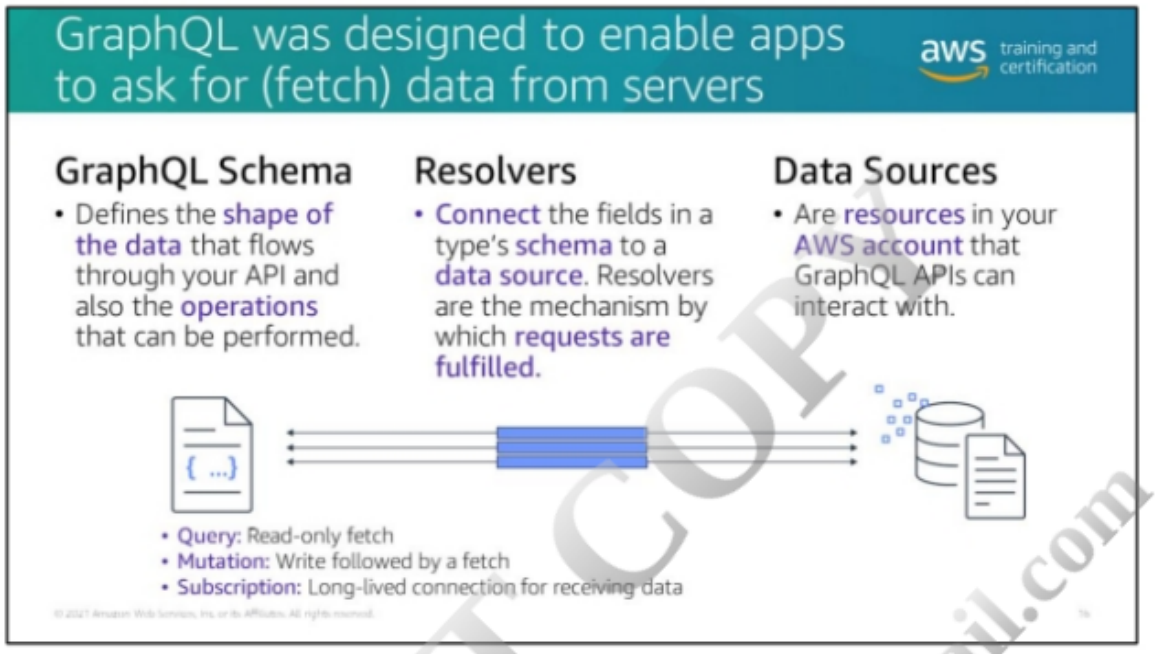
In this task, you'll build an HTTP API with a GET route that is integrated with a Lambda function called bakeryFunction. When you hit the endpoint, the function will randomly suggest a snack pick that you might buy from the diner.

The lab guide provides detailed steps.

You aren't going to cover GraphQL APIs in detail in this course, but they are a better option than REST or WebSockets APIs for some applications. You'll review the main characteristics of GraphQL APIs and how you might use AWS AppSync to implement them.

GraphQL is a query language for APIs and a runtime for fulfilling those queries.

AWS AppSync is a fully managed GraphQL service with real-time data synchronization and offline programming features.



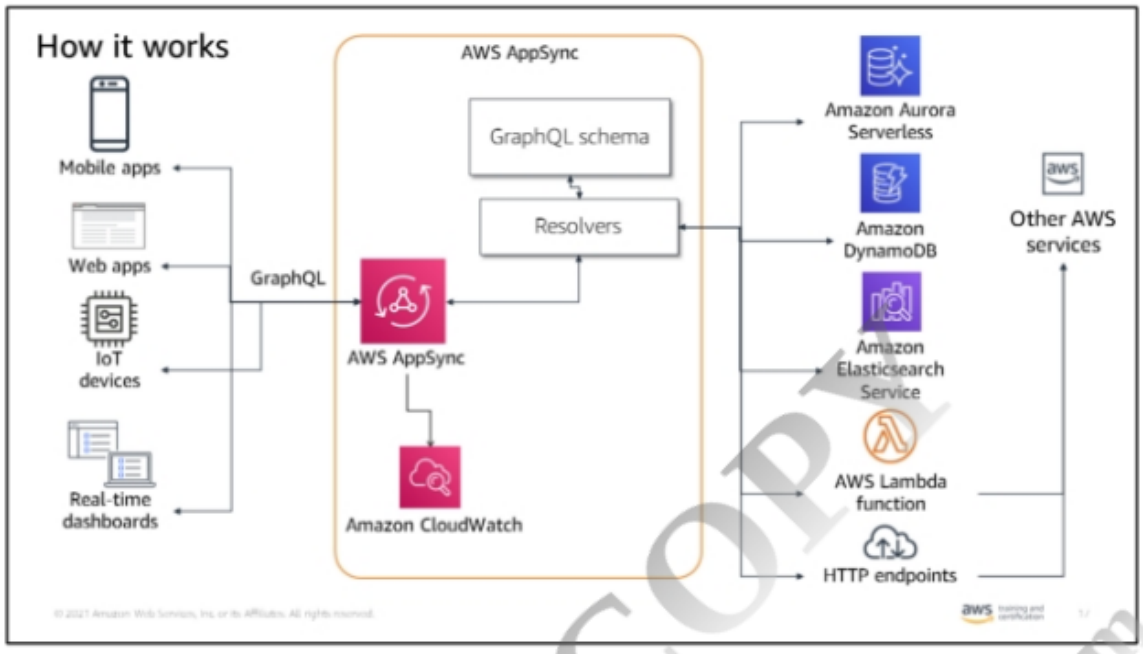
GraphQL was designed to enable apps to ask for (fetch) data from servers.

The GraphQL schema defines the data and the operations that can be performed. There are three top-level operations:

- Query: Read-only fetch
- Mutation: Write followed by a fetch
- Subscription: Long-lived connection for receiving data

With GraphQL, your API can interact with multiple data sources, and resolvers provide the connection between your schema fields and the data sources.

For more information about designing a GraphQL API, see https://docs.aws.amazon.com/appsync/latest/devguide/designing-a-graphql-api.html.



Let's go slightly deeper into the **How it works** diagram on the following page: https://aws.amazon.com/appsync/.

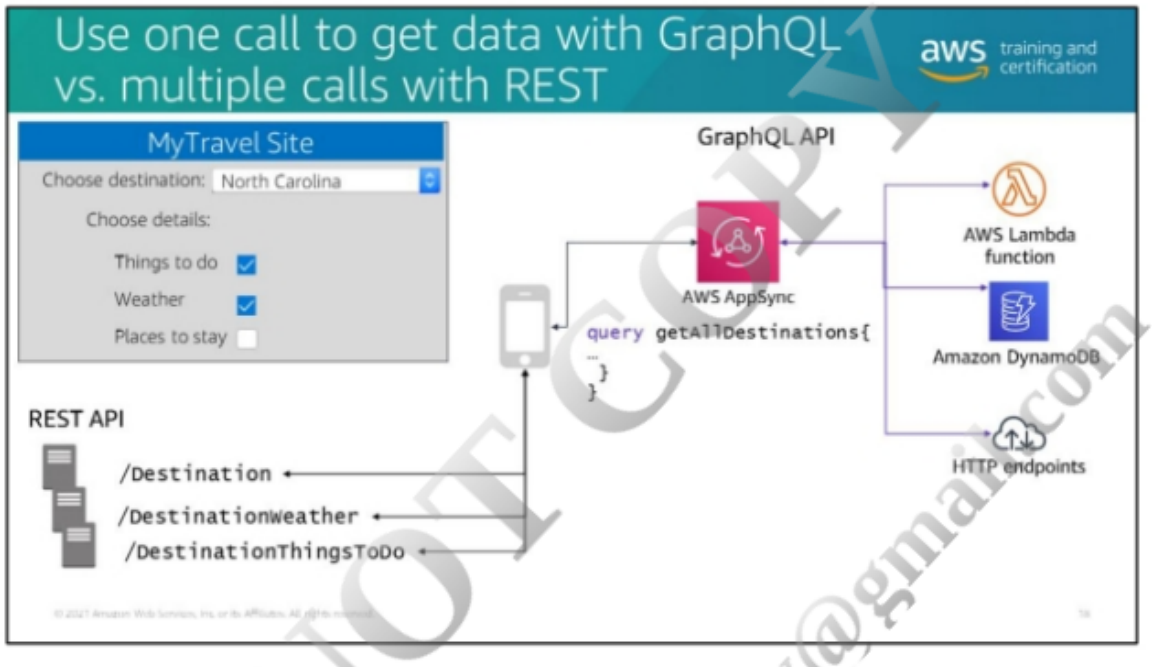
A GraphQL APIs is not structured around resources and HTTP verbs; instead, you define your data types and configure where the operations will retrieve data through a resolver.

An operation is either a query or a mutation. Queries retrieve data, and mutations are used when you want to modify data.

AWS AppSync can automatically create and connect resolvers from a schema or create a schema and connect resolvers from an existing table.

Data sources are resources in your AWS account that GraphQL APIs can interact with.

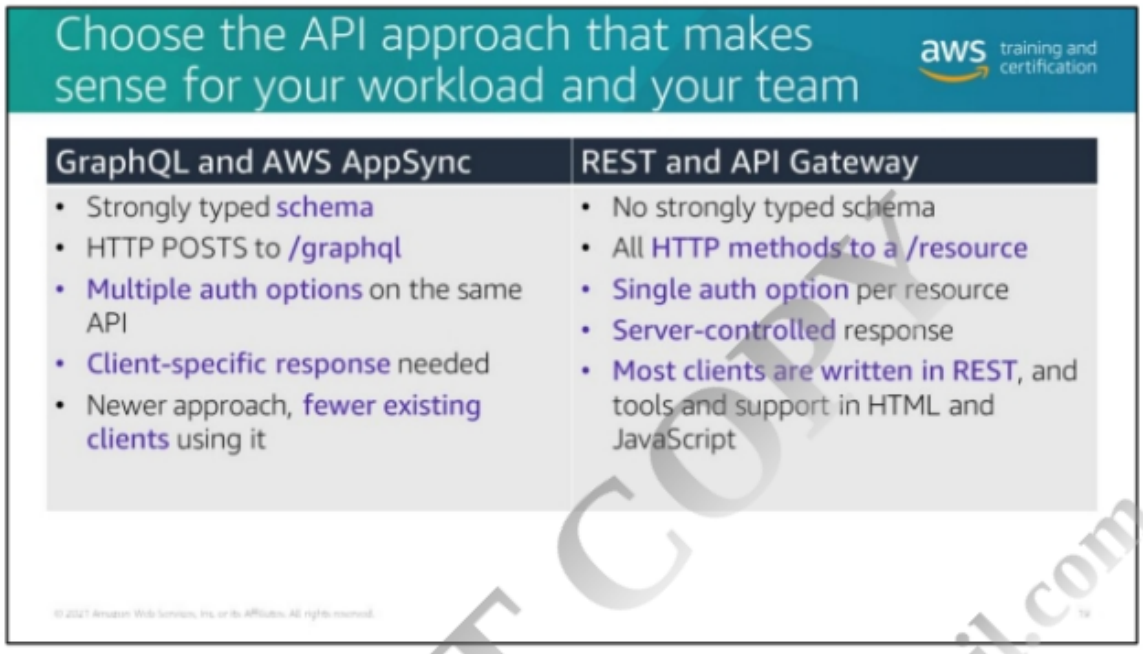
An AWS AppSync API can be configured to interact with multiple data sources, enabling you to aggregate data in a single location. AWS AppSync can use AWS resources from your account that already exist or can provision Amazon DynamoDB tables on your behalf from a schema definition.



With a REST API, you may need to make multiple sequential calls to get the data that you need. In this example, you have a travel site where users can select destinations and get details about the destination to help them make decisions about where to go.

Typically with REST, each distinct type of information to be returned is handled by its own resource that you need to call. To fulfill this request, you would need three different calls: the first to retrieve details about the selected destination, the second to get the weather for the destination, and the third to get the list of things to do associated with that destination.

With GraphQL, the schema for getAllDestinations includes fields for destinations, activities, weather, and hotels. You can make a single call and request only the fields you want (in this example, activities and weather for the specific destination, North Carolina). Resolvers connect to the different data sources that provide that information, and AWS AppSync provides the combined results to the client.



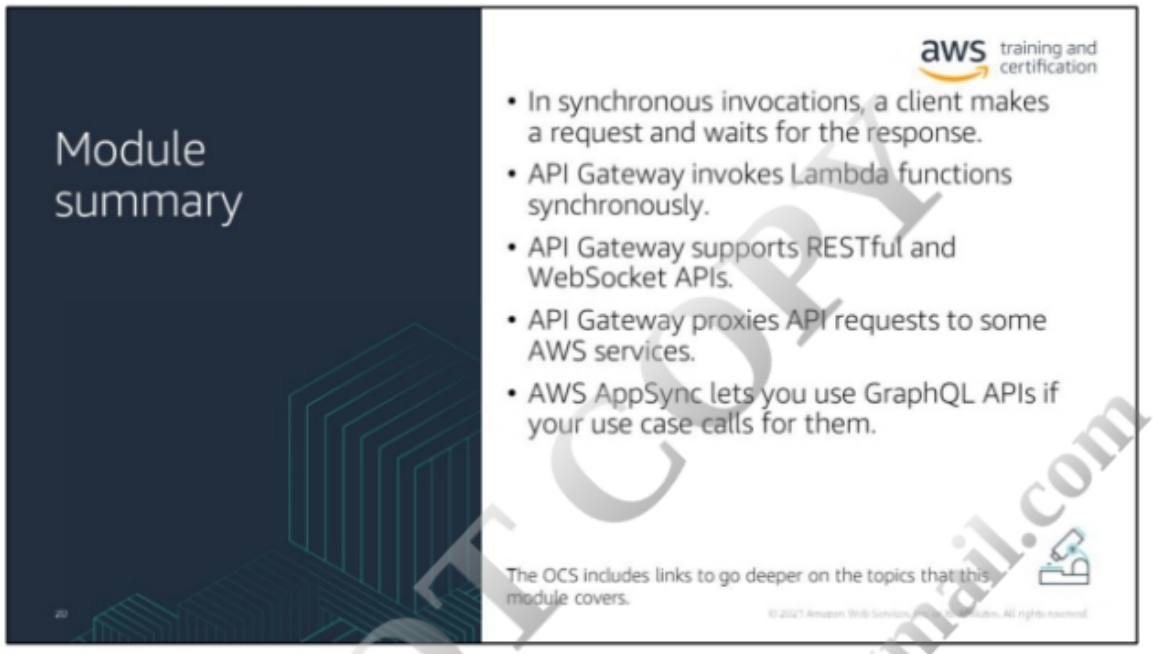
With GraphQL, you have to define the schema and data types in advance. If it's not in the schema, you can't query for it. Developers can download the schema and generate source code off the schema to work with it. When making calls to the APIs, developers can use HTTP POST to /graphql for all requests and modify the query details for only what they want.

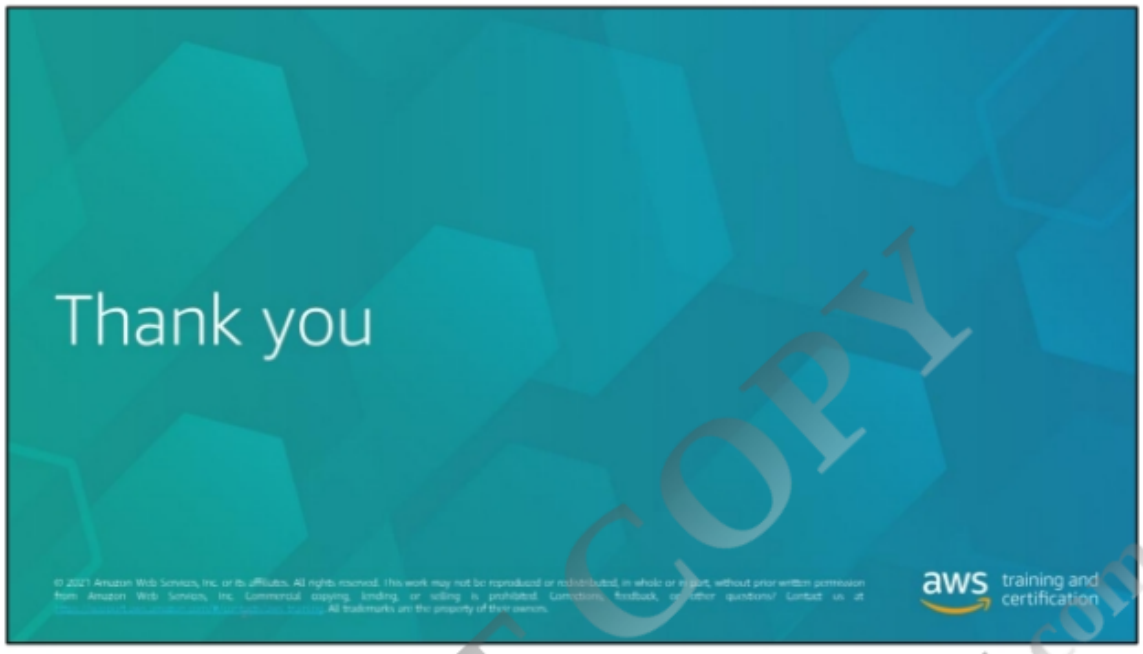
With REST, you can create data structures and validate what's being requested, but to write your code, all you have is the API documentation. You need to understand the model and write your APIs based on the documentation, and then you need to make calls to each method or resource.

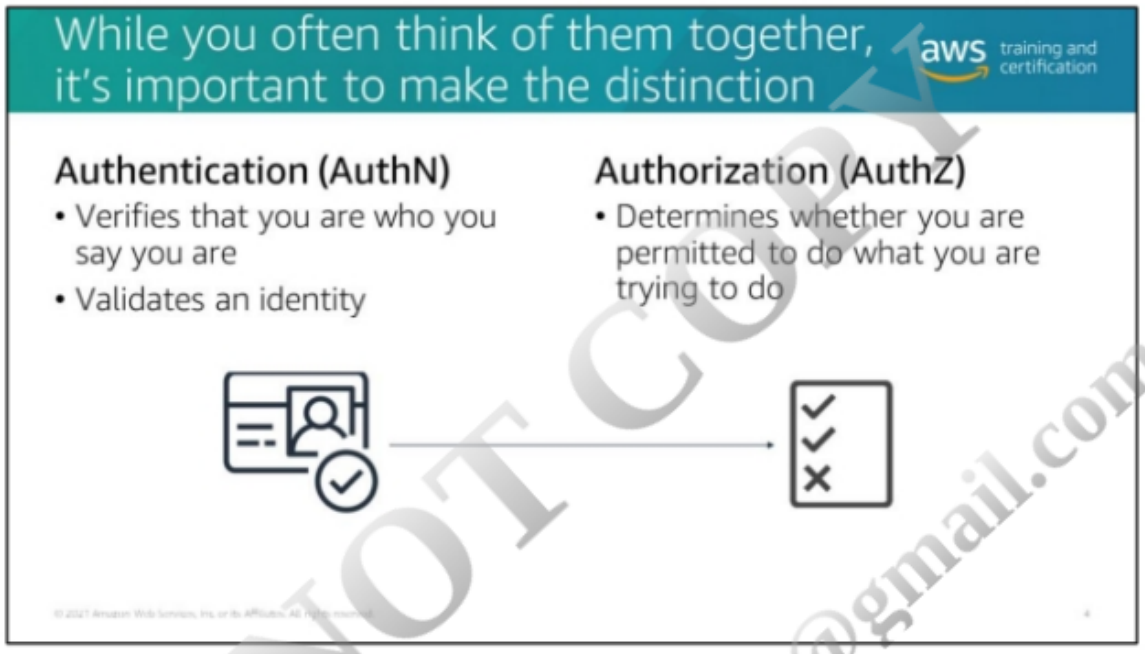
AWS AppSync allows you to use multiple authentication options on the same API, but API Gateway allows you to associate only one authentication option per resource.

Consider GraphQL for applications where you need a client-specific response that needs data from lots of backend sources.

If you have to integrate with existing clients, REST is much more mature, and there are more tools in which to use it. Most clients are written for REST.

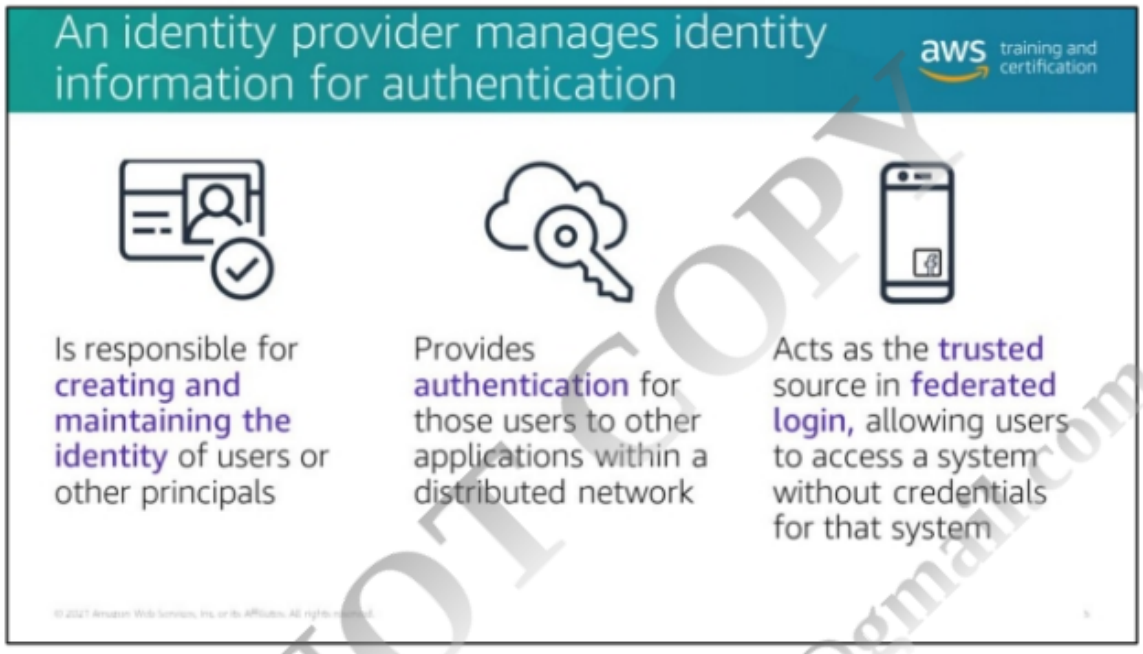




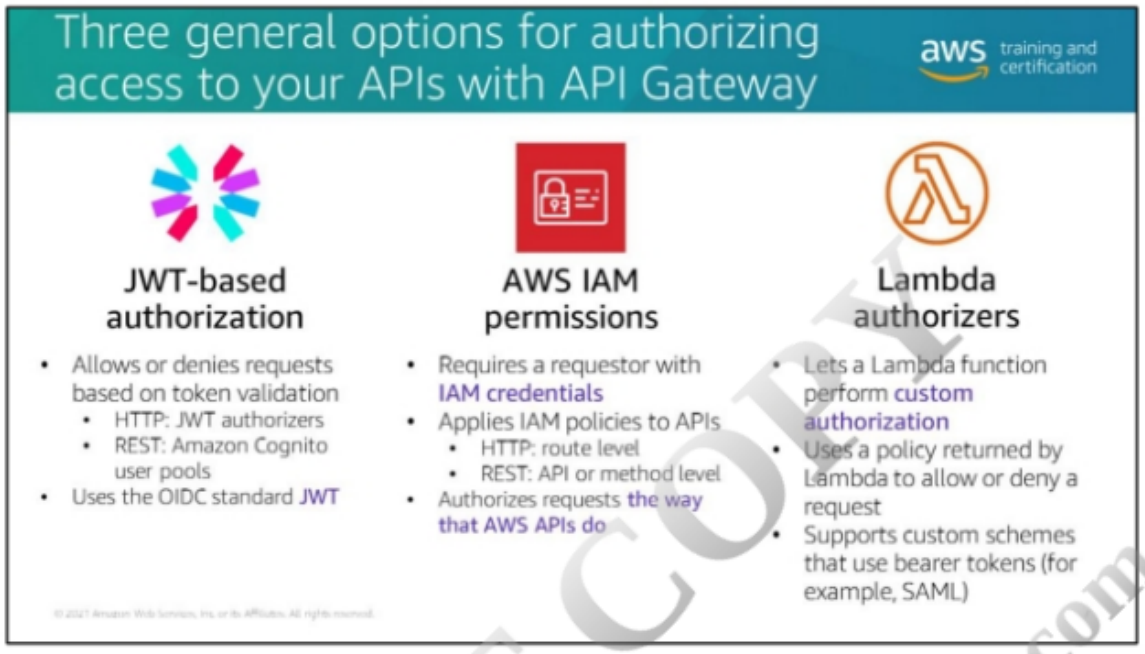


Authentication (AuthN) is about verifying who you are.

Authorization (AuthZ) is about what you're permitted to do.



The identity provider (IdP) is the source of truth for the identity of users in a system. It also facilitates single sign-on across applications within your enterprise and federated login (for example, using your Amazon, Facebook, or Google ID to sign in to another application).



Open ID Connect (OIDC, https://openid.net/connect/) is a simple identity layer on top of the OAuth 2.0 protocol. Developers often use it for web and mobile apps where they don't want the responsibility of managing identities and want to support federated identities. It enables clients to verify the identity of the user based on the authentication that an authorization server performs and to obtain basic profile information about the user in an interoperable and REST-like manner. It was released in 2014.

OIDC uses the open standard JSON Web Token (JWT) to securely transmit information between two parties. Amazon API Gateway provides JWT-based authorization for both HTTP and REST APIs. HTTP APIs can use JWT authorizers, and REST APIs can use Amazon Cognito user pools (which use JWT authorizers).

With JWT authorizers, API Gateway validates the JWTs that clients submit with API requests.

With Amazon Cognito user pools, the user is authenticated against the user pool and gets an OIDC token. The API call is successful only if the required JWT token is supplied and is valid.

API Gateway allows or denies requests based on token validation and optionally scopes in the token. You can configure distinct authorizers for each route of an API or use the same authorizer for multiple routes.

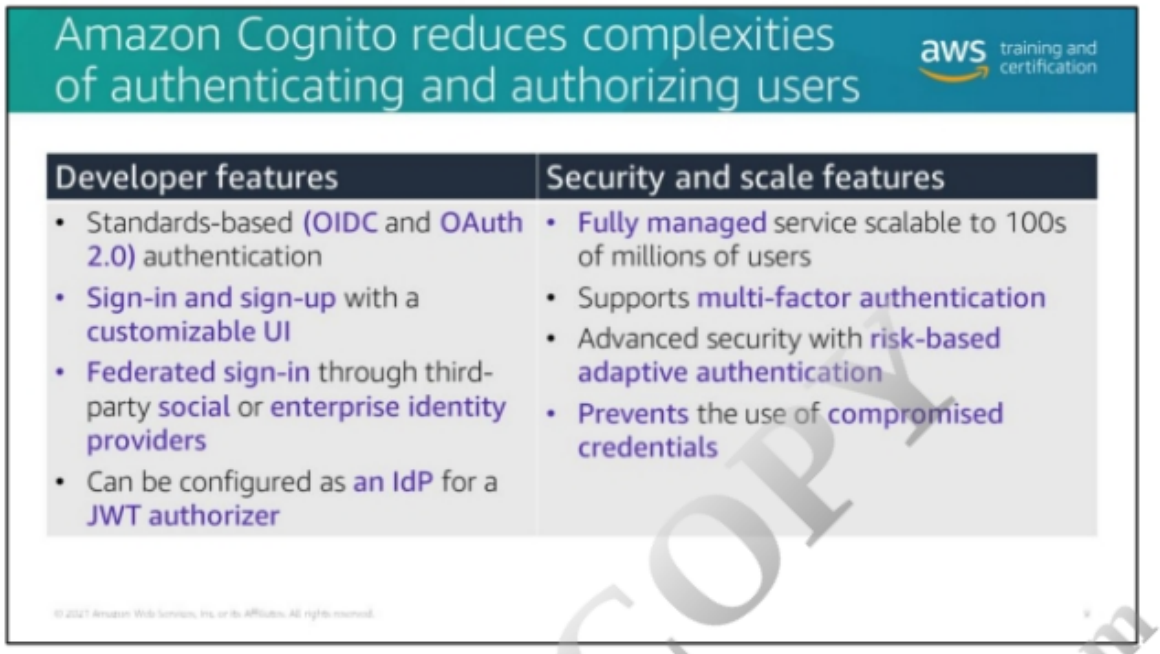
With permissions based on AWS Identity and Access Management (IAM), requestors must have IAM credentials.
Key information is added to the authorization header, and API Gateway determines whether or not the IAM user or role can invoke the API.

With Lambda authorizers, a Lambda function performs custom authorization. If a Lambda authorizer is configured, API Gateway uses it to call the API. API Gateway supplies the authorization information, and the Lambda function returns a policy to allow or deny the request. 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 Security Assertion Markup Language (SAML), or that uses request parameters to determine the caller's identity.

This course does not use SAML (http://docs.oasis-open.org/security/saml/Post2.0/sstc-saml-tech-overview-2.0-cd-02.html), but organizations frequently use it for enterprise single sign-on and federation across systems where the enterprise wants or needs to control identities. It defines an XML-based framework for describing and exchanging security information between online business partners and expresses security information in the form of portable SAML assertions that applications working across security domain boundaries can trust. It was released in 2002.

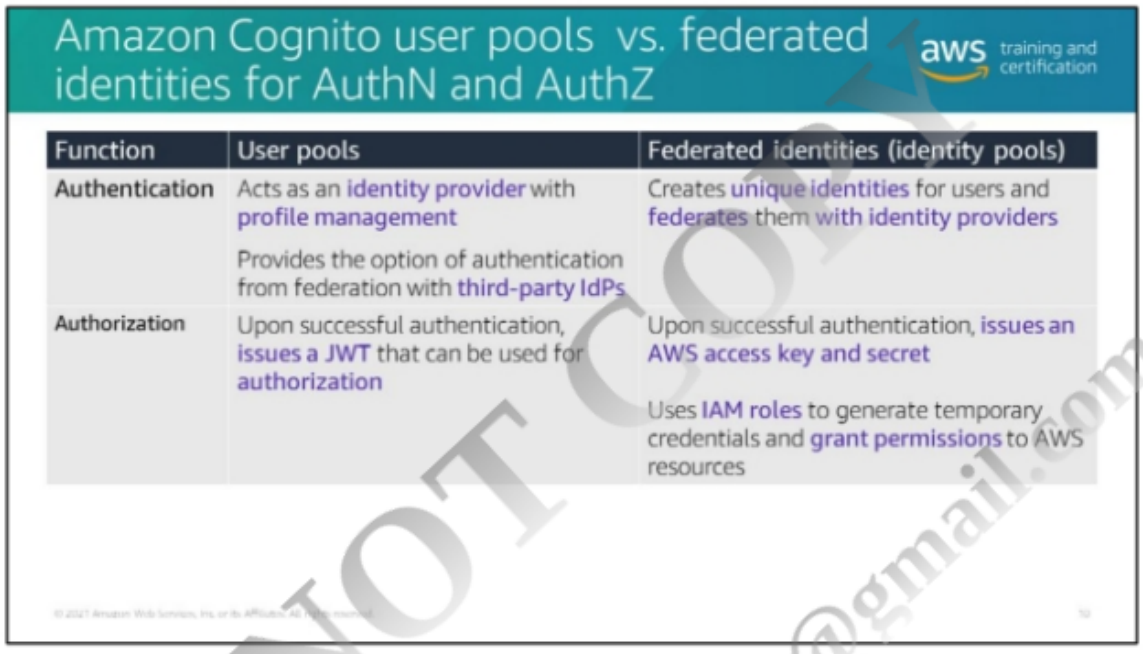
The Serverless Application Security module discusses API Gateway access options in more detail.

Amazon Cognito supports authentication and authorization in your serverless applications.



For more information about Amazon Cognito features, see https://aws.amazon.com/cognito/details/.

For code snippets for various mobile platforms, see the section titled "It really is this easy" at https://aws.amazon.com/cognito/.



There are two features in Amazon Cognito that provide different aspects of authentication and authorization. It's important to distinguish between their roles.

Amazon Cognito user pools can act as your IdP. User pools allow users to sign in to web or mobile applications and provide the option of authentication from federation with third-party IdPs.

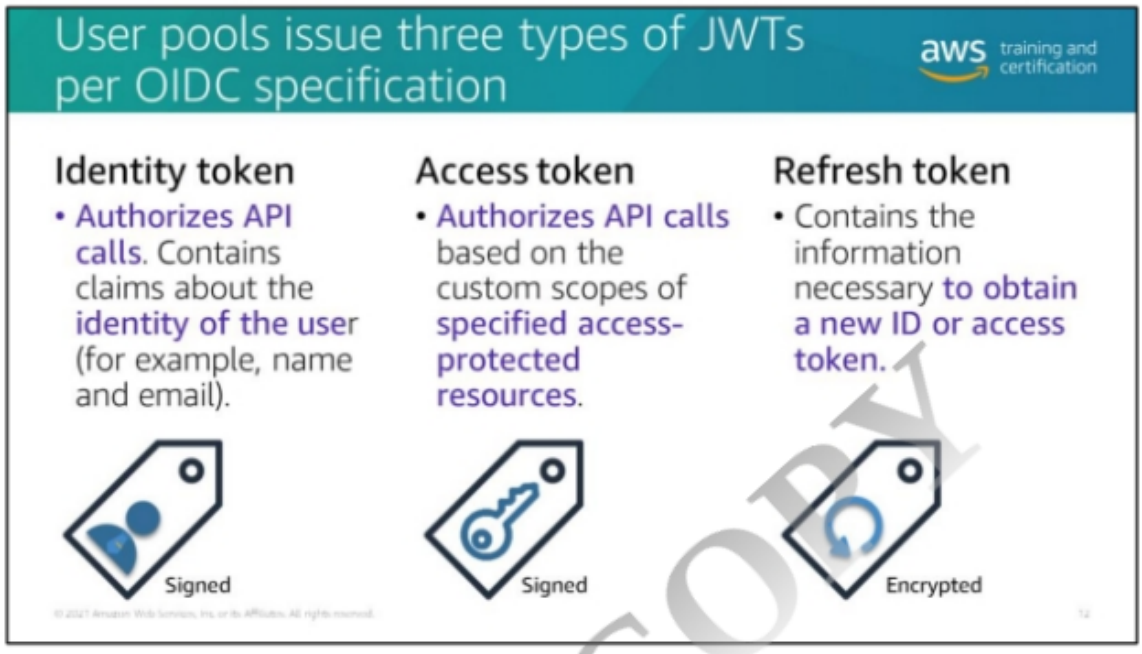
From an authorization perspective, user pools issue JWTs that can be used for authorization. The following can validate these tokens:

- An API Gateway
- An Application Load Balancer (ALB)
- Your own application or code

Federated identities (also called identity pools) create unique identities for users and federate them with identity providers but do not act as an IdP. Identity providers might be:

- An Amazon Cognito user pool or other OIDC provider
- A public provider (such as Facebook or Google)
- A SAML identity provider
- Developer-authenticated identities

Identity pools are useful when you want to directly integrate with AWS services like Amazon Simple Storage Service (Amazon S3) or Amazon DynamoDB rather than tunnel requests through an API Gateway or Application Load Balancer.



When you sign in using a user pool, regardless of where the identity lives (in the pool or coming from a federated third-party IdP), Amazon Cognito gives you three JWTs:

- ID
- Access
- Refresh

ID and access tokens are signed, not encrypted. The refresh token is encrypted.

The developer's job is to take the ID and access tokens and pass them along to authorize access to application resources.

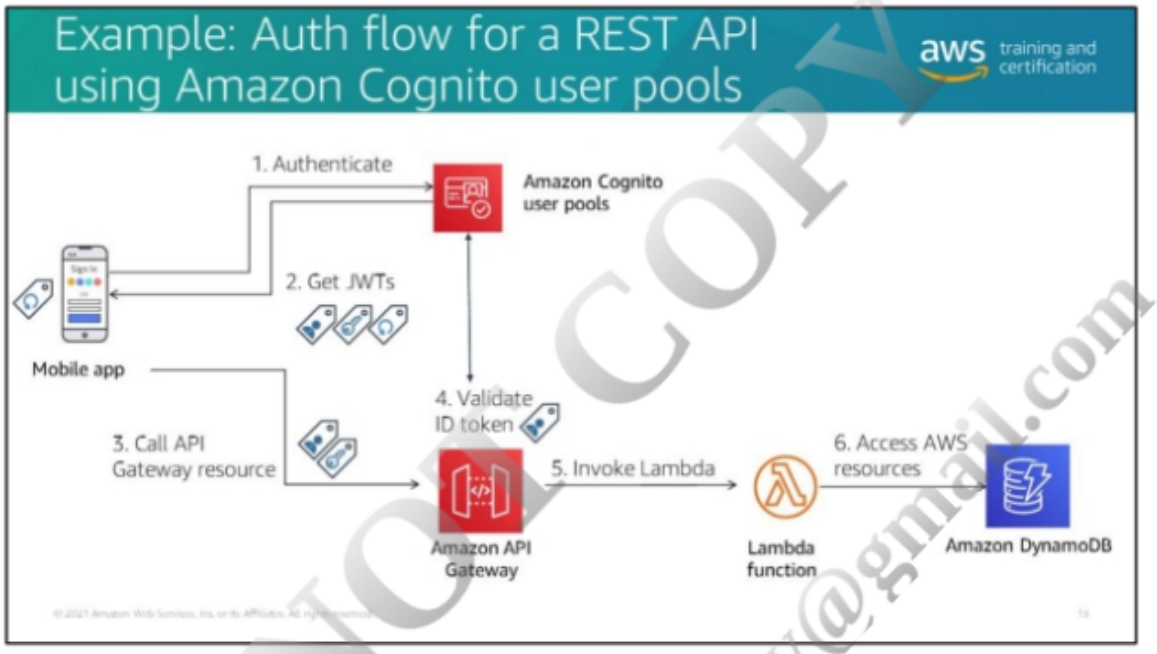
The client must save the refresh token to silently refresh the ID and access tokens behind the scenes.

By design, tokens live for a relatively short period of time. You can choose how long your access and refresh tokens remain valid.

- You can configure access tokens to expire in as little as 5 minutes or as long as 24 hours.
- You can configure refresh tokens to expire in as little as 1 hour or as long as 10 years.

These tokens simplify identity for developers and allow them, in a lightweight manner, to pass user context to downstream services so that developers can easily get this user context.

When you use the user pool directory as your IdP, AWS hosts the entire experience. If you set up user pools to federate with a third-party IdP, Amazon Cognito user pools is trusting those identities and redirecting them as if the users were in its directory.



This diagram depicts an authorization flow that uses Amazon Cognito user pools.

The client first authenticates with the user pool and gets the three JWTs in return.

The client then passes the ID and access token in the header as part of the call to API Gateway.

API Gateway validates the token before invoking the resource that it integrates with on the backend (in this example, a Lambda function).

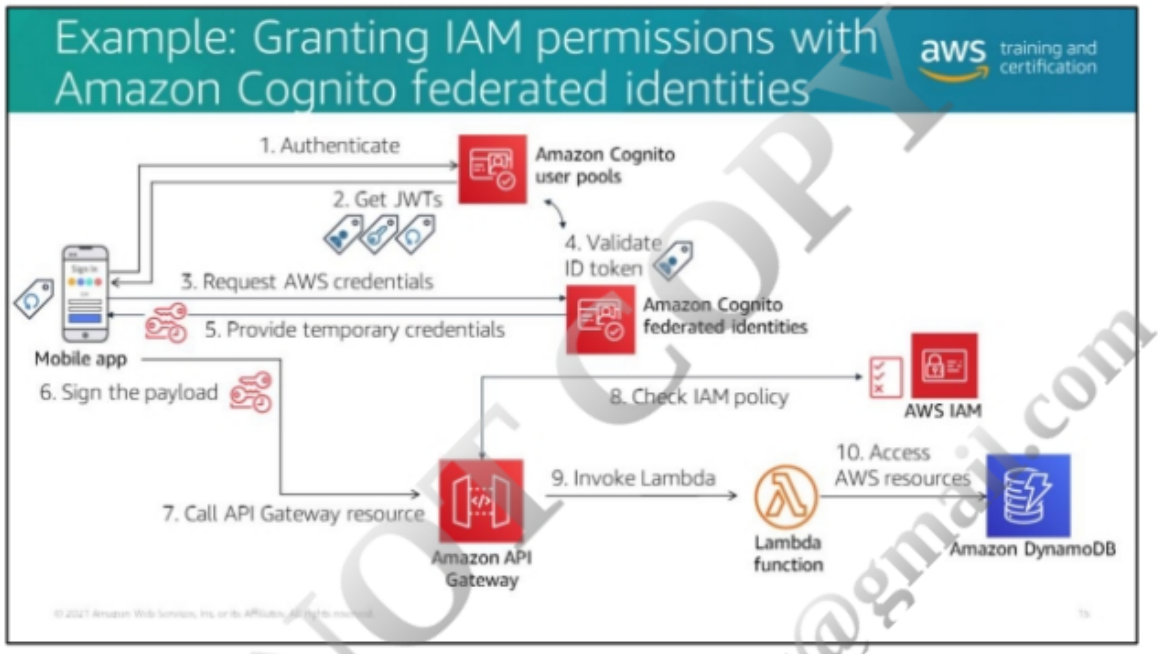
Depending on how you've written your API and application, API Gateway may pass on the ID token or the access token.

If you don't need to further scope the access allowed, pass the ID token.

If you need to scope access further, you can use the access token and configure predefined attributes within the API method request using OAuth scopes.

For details about configuring these options on this page, see https://docs.aws.amazon.com/apigateway/latest/developerguide/apigateway-enable-cognito-user-pool.html.

The flow for an HTTP API is similar except that a third-party JWT authorizer rather than Amazon Cognito user pools provides the JWT tokens.



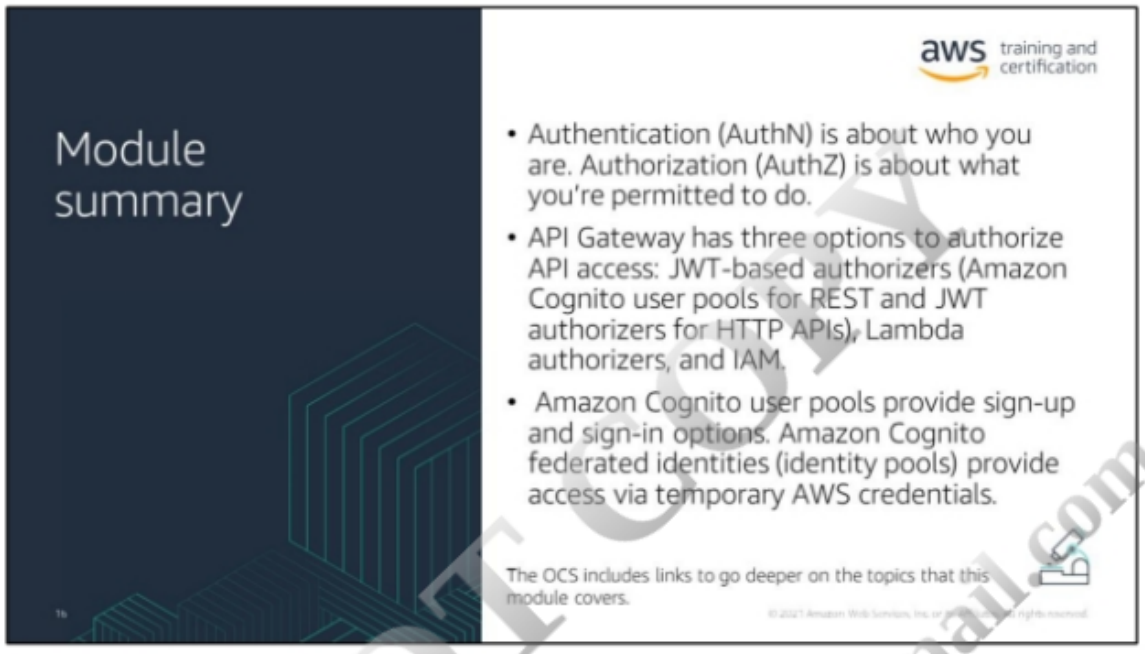
In this example, you use federated identities with IAM-based authorization to provide temporary IAM credentials.

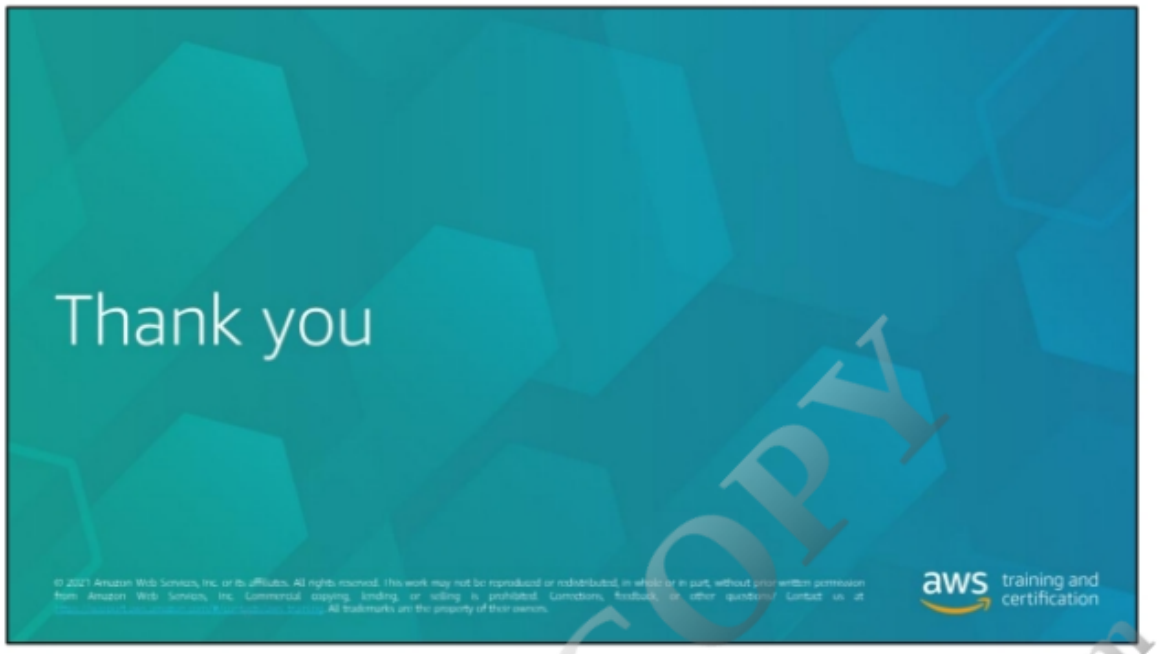
The client gets JWTs from the Amazon Cognito user pool and uses those to request temporary AWS credentials. Amazon Cognito federated identities validate the ID token and then return temporary credentials.

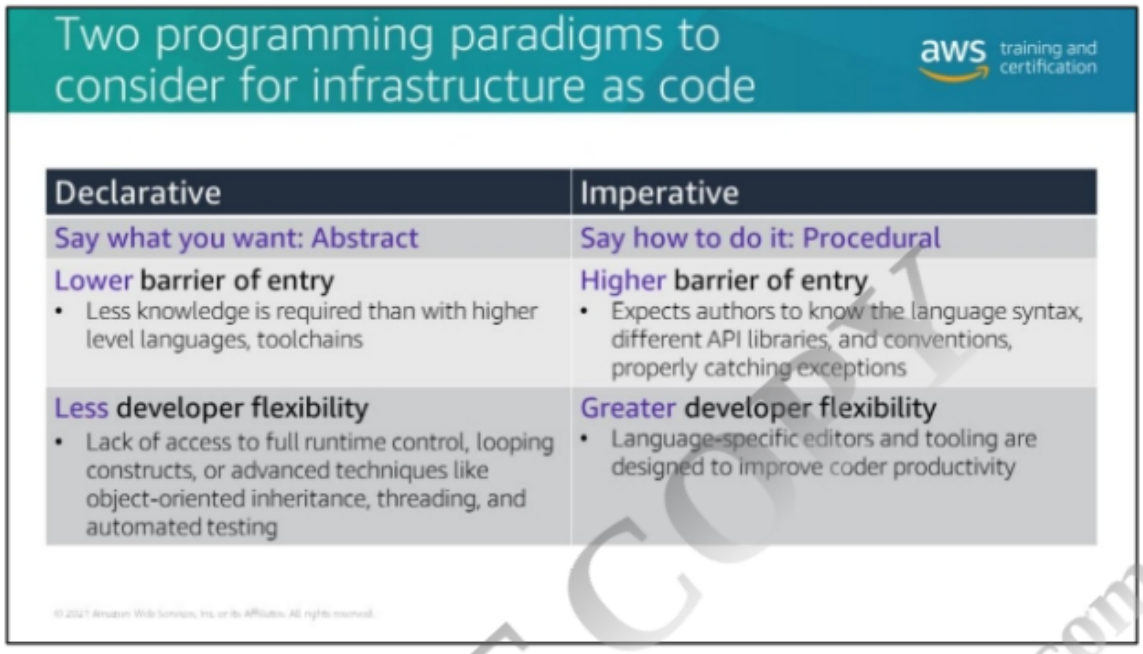
The client uses these credentials to sign the payload and call the API.

API Gateway checks the IAM policy associated with the temporary credentials and allows the actions that the policy provides.

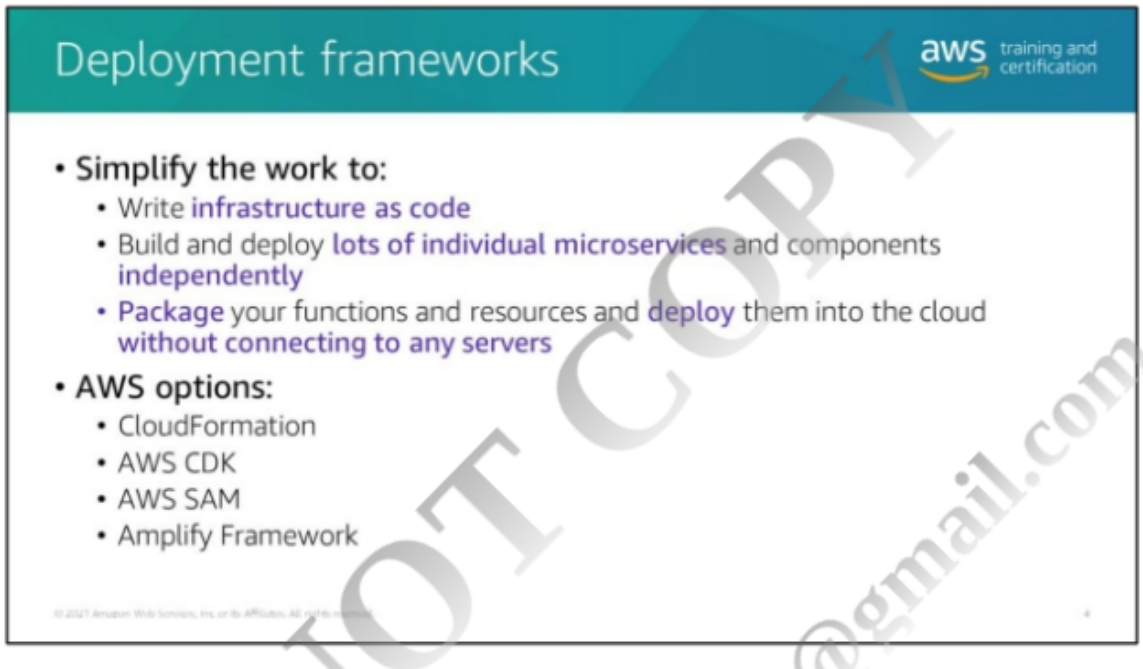
For more information about identity pools (federated identities) authentication flow, see https://docs.aws.amazon.com/cognito/latest/developerguide/authentication-flow.html.







Before you look at specific services, let's clarify two approaches to writing code that come into play: declarative and imperative.



One of the new practices that comes with working with lots of individual microservices and components that you want to be able to build independently is using a framework to simplify the deployment of your resources and code.

You no longer hand off a compiled binary for deployment to multiple servers. Instead, you need to tell AWS Lambda where to get your function code, you need to set up the other resources required by your application (APIs, tables, etc.), and you need to ensure that the correct permissions are set between the services that make up your application. You need to package your functions and resources and then deploy them into the cloud without connecting to any servers.

In this module, you look at how serverless frameworks and an infrastructure as code approach simplify the work to build and deploy your serverless stacks.

The course discusses four AWS options, but there are other frameworks in the market, so you should choose an approach that integrates well with your existing tool set and that suits the type of development that you need to do.