AWS Compute Blog

Building well-architected serverless applications: Controlling serverless API access – part 1

by Julian Wood | on 29 JUL 2020 | in Amazon API Gateway, Amazon CloudFront, Amazon Cognito, Amazon VPC, AWS Amplify, AWS AppSync, AWS CloudFormation, AWS Identity and Access Management (IAM), AWS Lambda, AWS Serverless Application Model, AWS Well-Architected Tool, Best Practices, Serverless | Permalink | Share

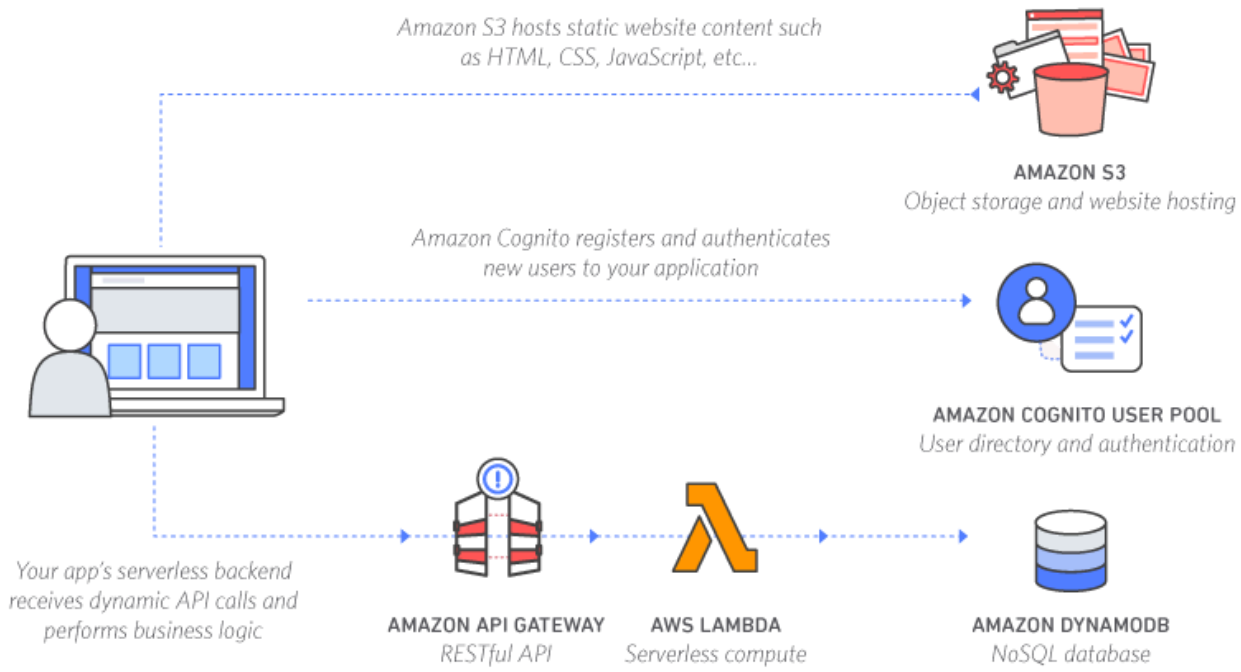
This series of blog posts uses the AWS Well-Architected Tool with the Serverless Lens to help customers build and operate applications using best practices. In each post, I address the nine serverless-specific questions identified by the Serverless Lens along with the recommended best practices. See the Introduction post for a table of contents and explanation of the example application.

Security question SEC1: How do you control access to your serverless API?

> Use authentication and authorization mechanisms to prevent unauthorized access, and enforce quota for public resources. By controlling access to your API, you can help protect against unauthorized access and prevent unnecessary use of resources.

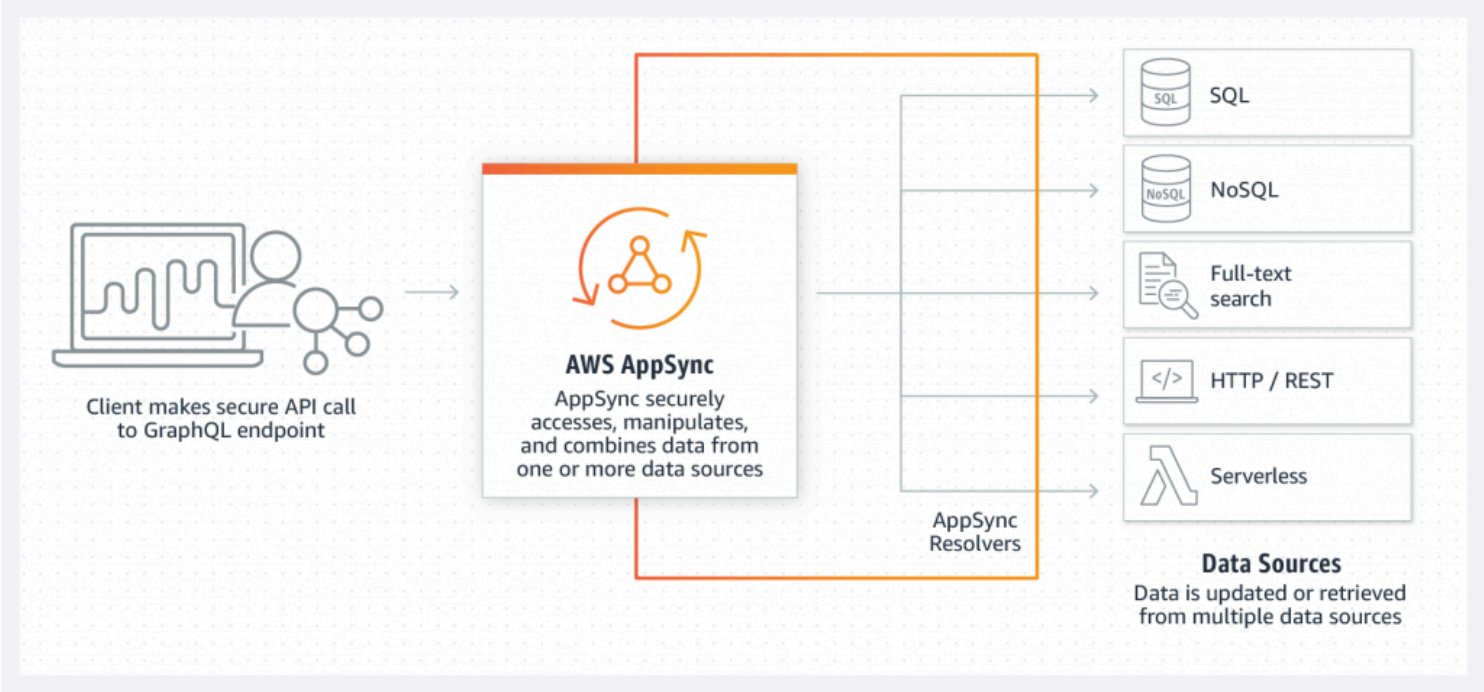
AWS has a number of services to provide API endpoints including Amazon API Gateway and AWS AppSync.

Use Amazon API Gateway for RESTful and WebSocket APIs. Here is an example serverless web application architecture using API Gateway.



Example serverless application architecture using API Gateway

Use AWS AppSync for managed GraphQL APIs.



AWS AppSync overview diagram

The serverless airline example in this series uses AWS AppSync to provide the frontend, user-facing public API. The application also uses API Gateway to provide backend, internal, private REST APIs for the loyalty and payment services.

Good practice: Use an authentication and an authorization mechanism

Authentication and authorization are mechanisms for controlling and managing access to a resource. In this well-architected question, that is a serverless API. Authentication is verifying who a client or user is. Authorization is deciding whether they have the permission to access a resource. By enforcing authorization, you can prevent unauthorized access to your workload from non-authenticated users.

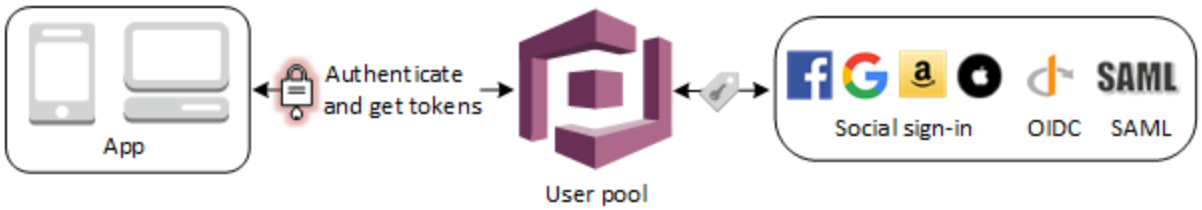
Integrate with an identity provider that can validate your API consumer's identity. An identity provider is a system that provides user authentication as a service. The identity provider may use the XML-based Security Assertion Markup Language (SAML), or JSON Web Tokens (JWT) for authentication. It may also federate with other identity management systems. JWT is an open standard that defines a way for securely transmitting information between parties as a JSON object. JWT uses frameworks such as OAuth 2.0 for authorization and OpenID Connect (OIDC), which builds on OAuth2, and adds authentication.

Only authorize access to consumers that have successfully authenticated. Use an identity provider rather than API keys as a primary authorization method. API keys are more suited to rate limiting and throttling.

Evaluate authorization mechanisms

Use AWS Identity and Access Management (IAM) for authorizing access to internal or private API consumers, or other AWS Managed Services like AWS Lambda.

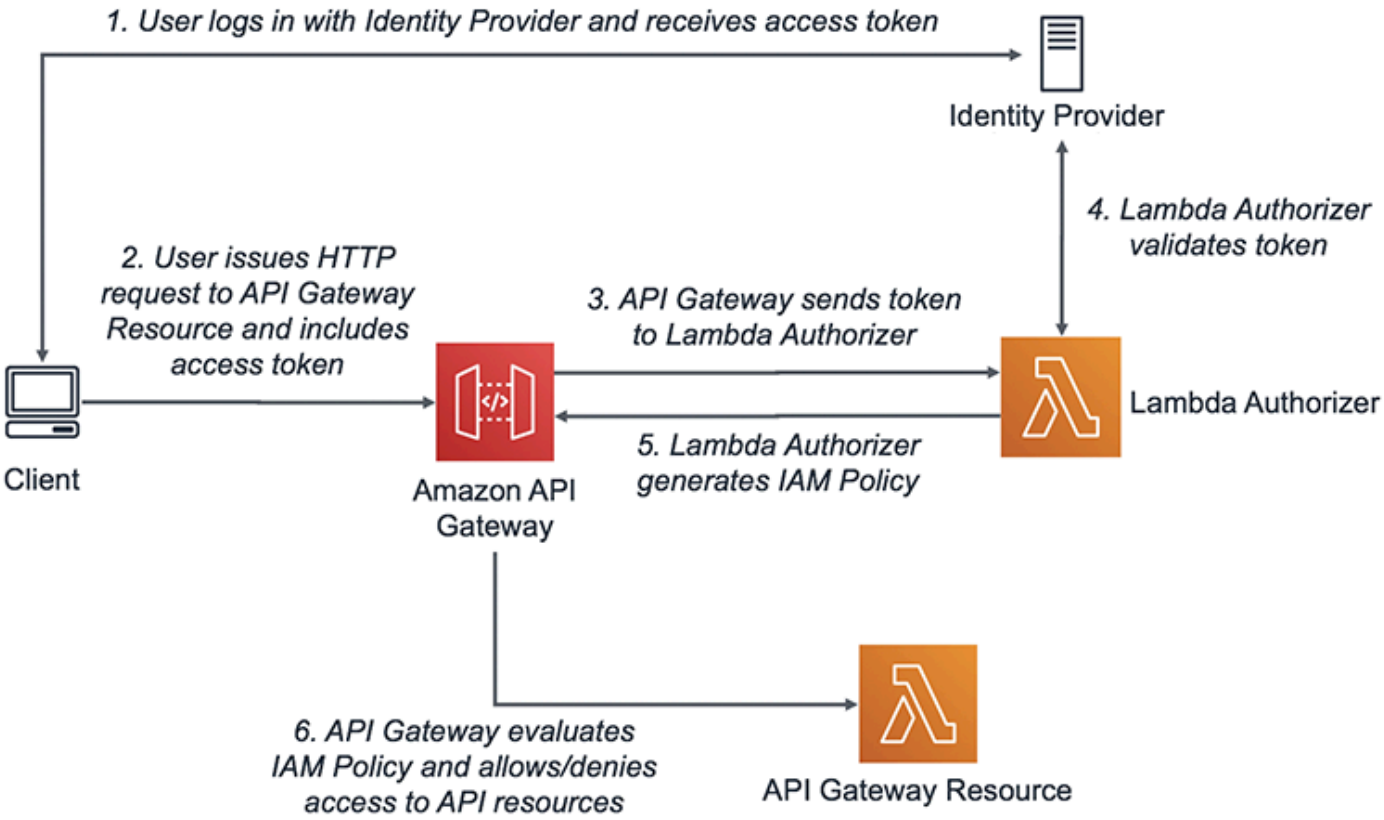
For public, user facing web applications, API Gateway accepts JWT authorizers for authenticating consumers. You can use either Amazon Cognito or OpenID Connect (OIDC).



App client authenticates and gets tokens

For custom authorization needs, you can use Lambda authorizers.

A Lambda authorizer (previously called a custom authorizer) is an AWS Lambda function which API Gateway calls for an authorization check when a client makes a request to an API method. This means you do not have to write custom authorization logic in a function behind an API. The Lambda authorizer function can validate a bearer token such as JWT, OAuth, or SAML, or request parameters and grant access. Lambda authorizers can be used when using an identity provider other than Amazon Cognito or AWS IAM, or when you require additional authorization customization.



Lambda authorizers

For more information, see the AWS Hero blog post, "The Complete Guide to Custom Authorizers with AWS Lambda and API Gateway".

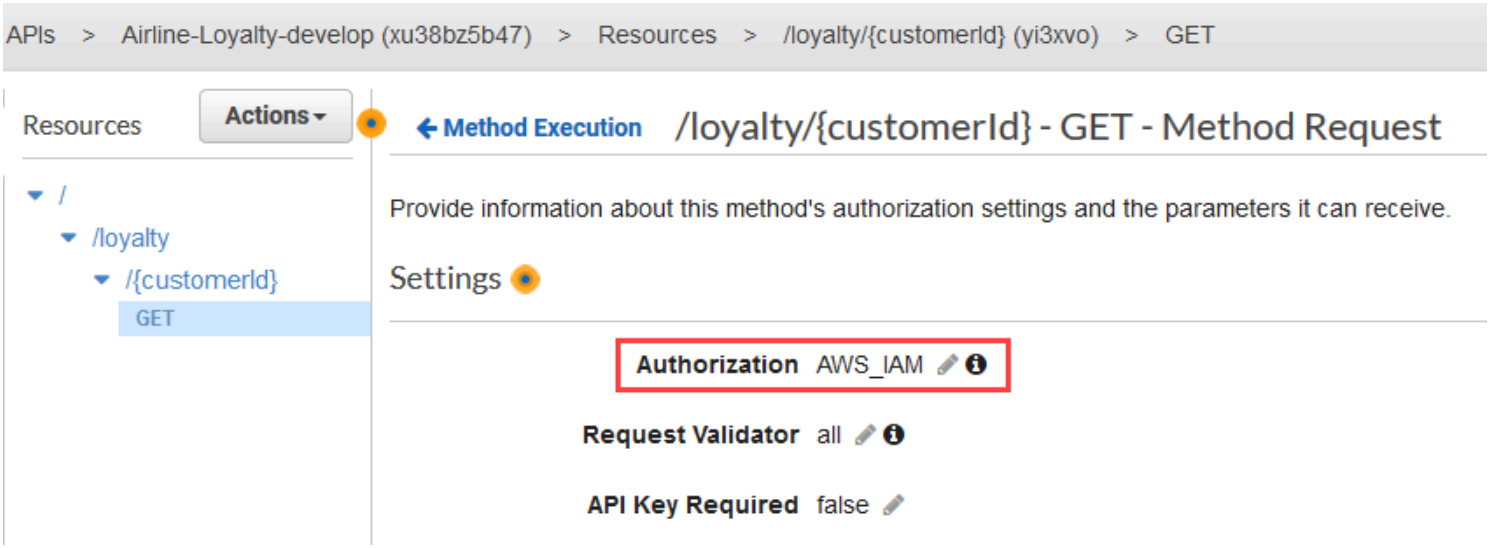
The AWS documentation also has a useful section on "Understanding Lambda Authorizers Auth Workflow with Amazon API Gateway".

Enforce authorization for non-public resources within your API

Within API Gateway, you can enable native authorization for users authenticated using Amazon Cognito or AWS IAM. For authorizing users authenticated by other identity providers, use Lambda authorizers.

For example, within the serverless airline, the loyalty service uses a Lambda function to fetch loyalty points and next tier progress. AWS AppSync acts as the client using an HTTP resolver, via an API Gateway REST API `/loyalty/{customerId}/get` resource, to invoke the function.

To ensure only AWS AppSync is authorized to invoke the API, IAM authorization is set within the API Gateway method request.



Viewing API Gateway IAM authorization

The serverless airline uses the AWS Serverless Application Model (AWS SAM) to deploy the backend infrastructure as code. This makes it easier to know which IAM role has access to the API. One of the benefits of using infrastructure as code is visibility into all deployed application resources, including IAM roles.

The loyalty service AWS SAM template contains the `AppsyncLoyaltyRestApiIamRole`.

YAML

```
AppsyncLoyaltyRestApiIamRole:
Type: AWS::IAM::Role
Properties:
AssumeRolePolicyDocument:
Version: 2012-10-17
Statement:
- Effect: Allow
  AppsyncLoyaltyRestApiIamRole:
    Type: AWS::IAM::Role
```

```
    Properties:
      AssumeRolePolicyDocument:
        Version: 2012-10-17
        Statement:
          - Effect: Allow
            Principal:
              Service: appsync.amazonaws.com
            Action: sts:AssumeRole
      Path: /
      Policies:
        - PolicyName: LoyaltyApiInvoke
          PolicyDocument:
            Version: 2012-10-17
            Statement:
              - Effect: Allow
                Action:
                  - execute-api:Invoke
                # arn:aws:execute-api:region:account-id:api-id/stage/METHOD_HTTP_VERB/I
                Resource: !Sub arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${I
```

The IAM role specifies that `appsync.amazonaws.com` can perform an `execute-api:Invoke` on the specific API Gateway resource `arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${LoyaltyApi}/*/*/*`

Within AWS AppSync, you can enable native authorization for users authenticating using Amazon Cognito or AWS IAM. You can also use any external identity provider compliant with OpenID Connect (OIDC).

Improvement plan summary:

1. Evaluate authorization mechanisms.
2. Enforce authorization for non-public resources within your API

Required practice: Use appropriate endpoint type and mechanisms to secure access to your API

APIs may have public or private endpoints. Consider public endpoints to serve consumers where they may not be part of your network perimeter. Consider private endpoints to serve consumers within your network perimeter where you may not want to expose the API publicly. Public and private endpoints may have different levels of security.

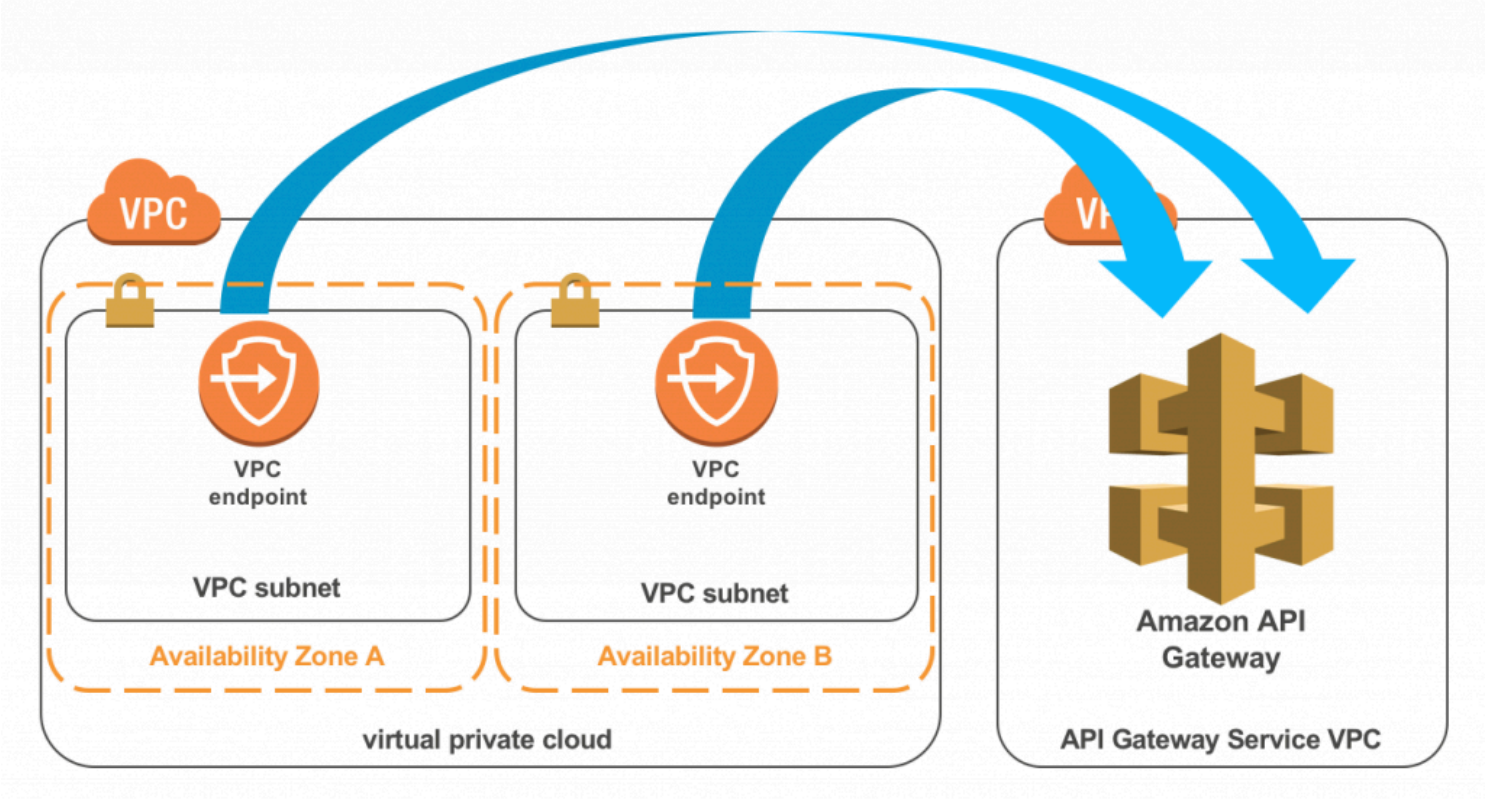
Determine your API consumer and choose an API endpoint type

For providing public content, use Amazon API Gateway or AWS AppSync public endpoints.

For providing content with restricted access, use Amazon API Gateway with authorization to specific resources, methods, and actions you want to restrict. For example, the serverless airline application uses AWS IAM to restrict access to the private *loyalty* API so only AWS AppSync can call it.

With AWS AppSync providing a GraphQL API, restrict access to specific data types, data fields, queries, mutations, or subscriptions.

You can create API Gateway private REST APIs that you can only access from your AWS Virtual Private Cloud(VPC) by using an interface VPC endpoint.



API Gateway private endpoints

For more information, see "Choose an endpoint type to set up for an API Gateway API".

Implement security mechanisms appropriate to your API endpoint

With Amazon API Gateway and AWS AppSync, for both public and private endpoints, there are a number of mechanisms for access control.

For providing content with restricted access, API Gateway REST APIs support native authorization using AWS IAM, Amazon Cognito user pools, and Lambda authorizers. Amazon Cognito user pools is a feature that provides a managed user directory for authentication. For more detailed information, see the AWS Hero blog post, "Picking the correct authorization mechanism in Amazon API Gateway".

You can also use resource policies to restrict content to a specific VPC, VPC endpoint, a data center, or a specific AWS Account.

API Gateway resource policies are different from IAM identity policies. IAM identity policies are attached to IAM users, groups, or roles. These policies define what that identity can do on which resources. For example, in the serverless airline, the IAM role `AppsyncLoyaltyRestApiIamRole` specifies that `appsync.amazonaws.com` can perform an `execute-api:Invoke` on the specific API Gateway resource `arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${LoyaltyApi}/*/*/*`

Resource policies are attached to resources such as an Amazon S3 bucket, or an API Gateway resource or method. The policies define what identities can access the resource.

IAM access is determined by a combination of identity policies and resource policies.

For more information on the differences, see "Identity-Based Policies and Resource-Based Policies". To see which services support resource-based policies, see "AWS Services That Work with IAM".

API Gateway HTTP APIs support JWT authorizers as a part of OpenID Connect (OIDC) and OAuth 2.0 frameworks.

API Gateway WebSocket APIs support AWS IAM and Lambda authorizers.

With AWS AppSync public endpoints, you can enable authorization with the following:

- AWS IAM
- Amazon Cognito User pools for email and password functionality
- Social providers (Facebook, Google+, and Login with Amazon)
- Enterprise federation with SAML

Within the serverless airline, AWS Amplify Console hosts the public user facing site. Amplify Console provides a git-based workflow for building, deploying, and hosting serverless web applications. Amplify Console manages the hosting of the frontend assets for single page app (SPA) frameworks in addition to static websites, along with an optional serverless backend. Frontend assets are stored in S3 and the Amazon CloudFront global edge network distributes the web app globally.

The AWS Amplify CLI toolchain allows you to add backend resources using AWS CloudFormation.

Using Amplify CLI to add authentication

For the serverless airline, I use the Amplify CLI to add authentication using Amazon Cognito with the following command:

```
Bash
amplify add auth
```

When prompted, I specify the authentication parameters I require.

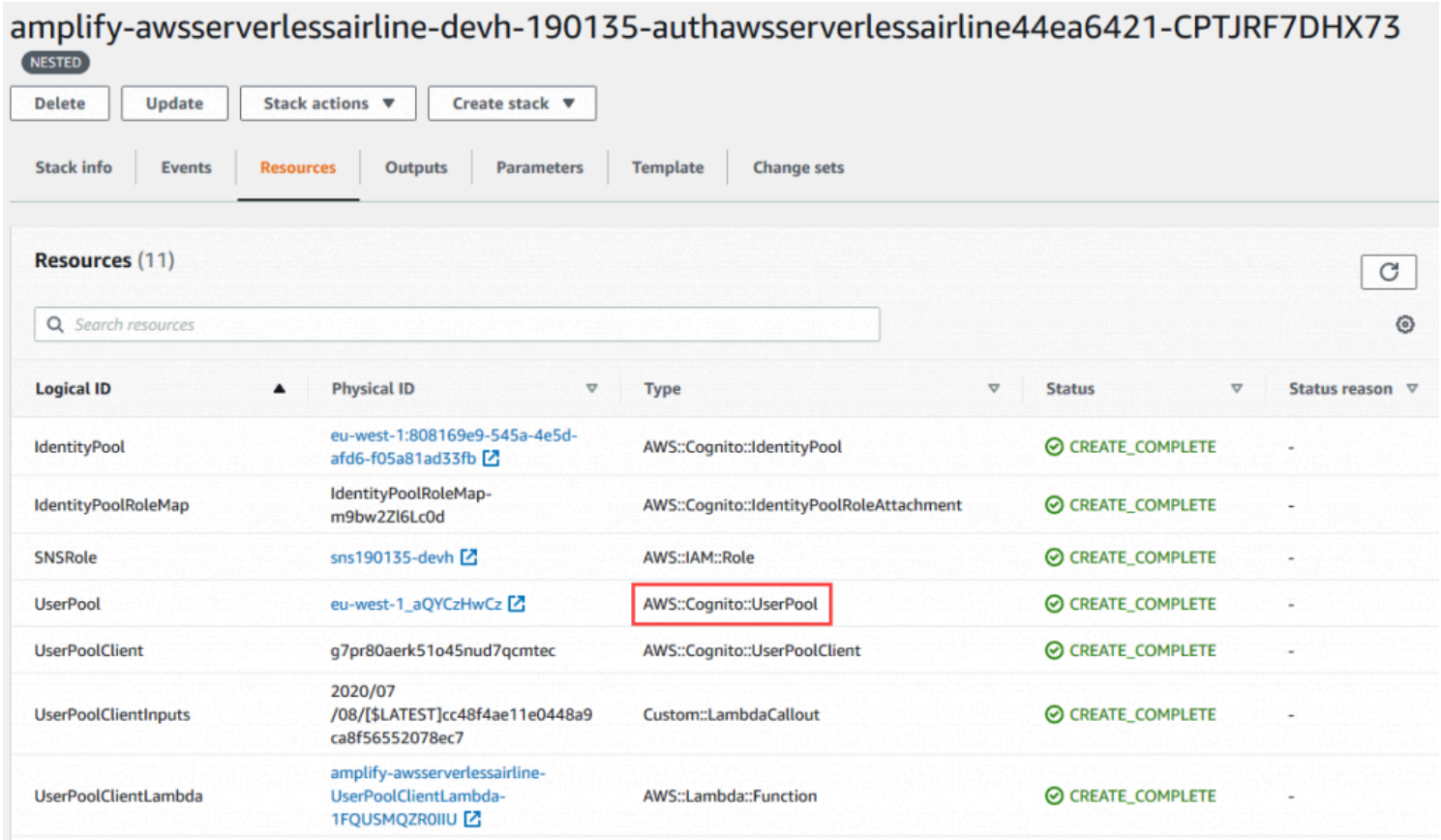
```
> amplify add auth
Using service: Cognito, provided by: awscloudformation

 The current configured provider is Amazon Cognito.

 Do you want to use the default authentication and security configuration? Default configuration
 Warning: you will not be able to edit these selections.
 How do you want users to be able to sign in? Email
 Do you want to configure advanced settings? Yes, I want to make some additional changes.
 Warning: you will not be able to edit these selections.
 What attributes are required for signing up? Email, Family Name (This attribute is not supported by Login With
 Amazon.), Given Name (This attribute is not supported by Login With Amazon.), Phone Number (This attribute is
not supported by Facebook, Login With Amazon.)
 Do you want to enable any of the following capabilities?
Successfully added resource awsserverlessairlinebooking74b61c6e locally

Some next steps:
"amplify push" will build all your local backend resources and provision it in the cloud
"amplify publish" will build all your local backend and frontend resources (if you have hosting category added)
 and provision it in the cloud
```

Amplify add auth

Amplify CLI creates a local CloudFormation template. Use the following command to deploy the updated authentication configuration to the cloud:

```
Bash
amplify push
```

Once the deployment is complete, I view the deployed authentication nested stack resources from within the CloudFormation Console. I see the Amazon Cognito user pool.



View Amplify authentication CloudFormation nested stack resources

For a more detailed walkthrough using Amplify CLI to add authentication for the serverless airline, see the build video.

For more information on Amplify CLI and authentication, see "Authentication with Amplify".

Conclusion

To help protect against unauthorized access and prevent unnecessary use of serverless API resources, control access using authentication and authorization mechanisms.

In this post, I cover the different mechanisms for authorization available for API Gateway and AWS AppSync. I explain the different approaches for public or private endpoints and show how to use IAM to control access to internal or private API consumers. I walk through how to use the Amplify CLI to create an Amazon Cognito user pool.

This well-architected question continues in part 2 where I continue using the Amplify CLI to add a GraphQL API. I explain how to view JSON Web Tokens (JWT) claims, and how to use Cognito identity pools to grant temporary access to AWS services. I will also show how to use API keys and API Gateway usage plans for rate limiting and throttling requests.

TAGS: Amazon API Gateway, Amazon CloudFront, Amazon Cognito, Amazon VPC, AppSync, AWS Amplify, AWS CloudFormation, AWS Lambda, AWS Serverless Application Model, IAM, serverless, well-architected