

Building well-architected serverless applications: Controlling serverless API access – part 2

by Julian Wood | on 30 JUL 2020 | in [Amazon API Gateway](#), [Amazon Cognito](#), [AWS Amplify](#), [AWS AppSync](#), [AWS CloudFormation](#), [AWS Identity and Access Management \(IAM\)](#), [AWS Lambda](#), [AWS WAF](#), [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?

This post continues [part 1](#) of this security question. Previously, I cover the different mechanisms for authentication and authorization available for [Amazon API Gateway](#) and [AWS AppSync](#). I explain the different approaches for public or private endpoints and show how to use [AWS Identity and Access Management \(IAM\)](#) to control access to internal or private API consumers.

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

I continue to show how to implement security mechanisms appropriate for your API endpoint.

Using AWS Amplify CLI to add a GraphQL API

After adding authentication in [part 1](#), I use the AWS Amplify CLI to add a GraphQL AWS AppSync API with the following command:

```
Bash
amplify add api
```

When prompted, I specify an [Amazon Cognito](#) user pool for authorization.

```
> amplify add api
? Please select from one of the below mentioned services: GraphQL
? Provide API name: awsserverlessairline
? Choose the default authorization type for the API Amazon Cognito User Pool
Use a Cognito user pool configured as a part of this project.
? Do you want to configure advanced settings for the GraphQL API Yes, I want to make some additional changes.
? Configure additional auth types? No
? Configure conflict detection? No
? Do you have an annotated GraphQL schema? Yes
? Provide your schema file path: C:\GitProjects\aws-serverless-airline-booking\amplify\backend\api\awsserverlessairline\schema.graphql

The following types do not have '@auth' enabled. Consider using @auth with @model
- Flight
Learn more about @auth here: https://docs.amplify.aws/cli/graphql-transformer/directives#auth

GraphQL schema compiled successfully.

Edit your schema at C:\GitProjects\aws-serverless-airline-booking\amplify\backend\api\awsserverlessairline\schema.graphql or place .graphql files in a directory at C:\GitProjects\aws-serverless-airline-booking\amplify\backend\api\awsserverlessairline\schema
Successfully added resource awsserverlessairline locally
```

Amplify add Amazon Cognito user pool for authorization

To deploy the AWS AppSync API configuration to the AWS Cloud, I enter:

```
Bash
amplify push
```

Once the deployment is complete, I view the GraphQL API from within the [AWS AppSync console](#) and navigate to **Settings**. I see the AWS AppSync API uses the authorization configuration added during the [part 1](#)

`amplify add auth`. This uses the Amazon Cognito user pool to store the user sign-up information.

Default authorization mode

API-level

The primary authorization mode for your schema. This applies to all fields and types by default.

Amazon Cognito User Pool

Configuration

AWS Region

Select the region your user pool is located in.

EU-WEST-1

Select a user pool

eu-west-1_aQYCzHwCz

Default action

Select the default permission for requests to your API.

ALLOW

AppId client regex

(Optional) Type a regular expression to allow or block requests to this API.

View AWS AppSync authorization settings with Amazon Cognito

For a more detailed walkthrough using Amplify CLI to add an AWS AppSync API for the serverless airline, see the [build video](#).

Viewing JWT tokens

When I create a new account from the serverless airline web frontend, Amazon Cognito creates a user within the user pool. It handles the 3-stage sign-up process for new users. This includes account creation, confirmation, and user sign-in.

The diagram illustrates the Amazon Cognito sign-in process, divided into three main steps:

- Create a new account (Step 1):** This section includes fields for First name (Julian), Family name (Wood), Username (julianwood), Password (masked), Email (jrwood@amazon.com), and Phone number (UK (+44) 12345678). A red **CREATE ACCOUNT** button is at the bottom right, and a **Sign in** link is at the bottom left.
- Confirm Sign Up (Step 2):** This section includes fields for Username (julianwood) and Confirmation Code (338711). A red **CONFIRM** button is at the bottom right. Links for **Resend Code** and **Back to Sign In** are also present.
- Sign in to your account (Step 3):** This section includes fields for Username (julianwood) and Password (masked). A red **SIGN IN** button is at the bottom right. Links for **Reset password** and **Create account** are also present.

Serverless airline Amazon Cognito based sign-in process

Once the account is created, I browse to the [Amazon Cognito console](#) and choose **Manage User Pools**. I navigate to *Users and groups* under *General settings* and view my user account.

Users > **julianwood**

[Add to group](#) [Reset password](#) [Enable SMS MFA](#) [Disable user](#)

Groups -

Account Status Enabled / CONFIRMED

SMS MFA Status Disabled

Last Modified Jul 15, 2020 3:19:42 PM

Created Jul 15, 2020 3:17:40 PM

sub a900ae6c-99b9-4112-9677-b66fe79d792c

email_verified true

phone_number_verified false

phone_number +4412345678

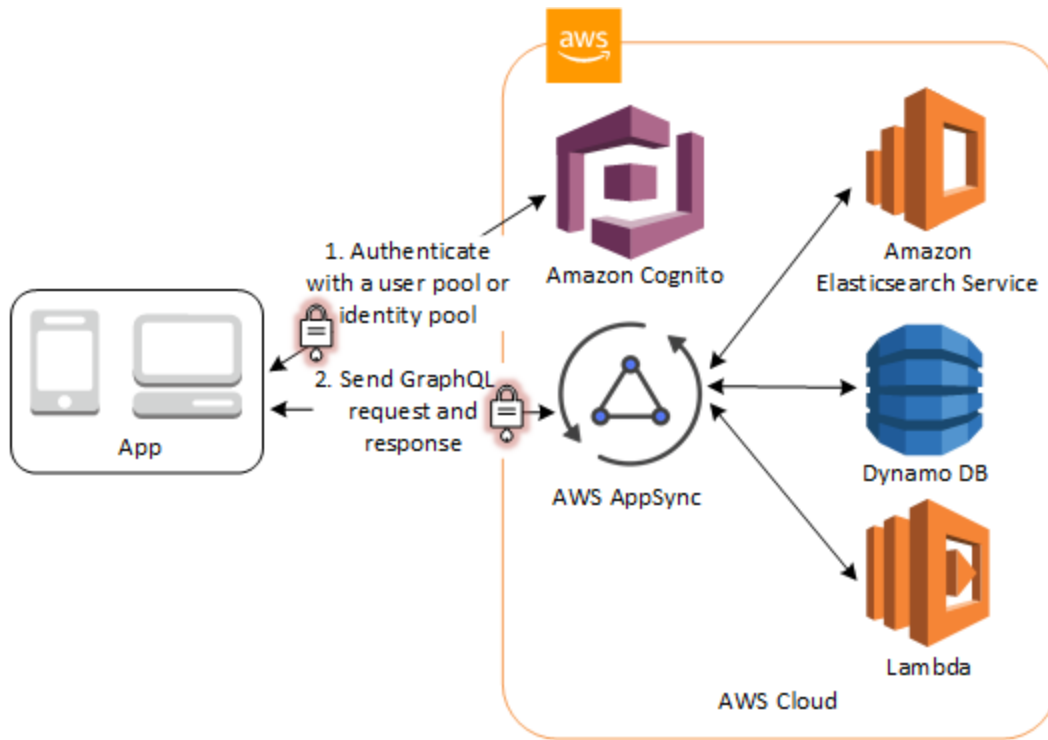
given_name Julian

family_name Wood

email jrwood@amazon.com

[View User Account](#)

When I sign in to the serverless airline web app, I authenticate with Amazon Cognito, and the client receives user pool tokens. The client then calls the AWS AppSync API, which authorizes access using the tokens, connects to data sources, and resolves the queries.



Amazon Cognito tokens used by AWS AppSync

During the sign-in process, I can use the browser developer tools to view the three JWT tokens Amazon Cognito generates and returns to the client. These are the `accessToken`, `idToken`, and `refreshToken`.

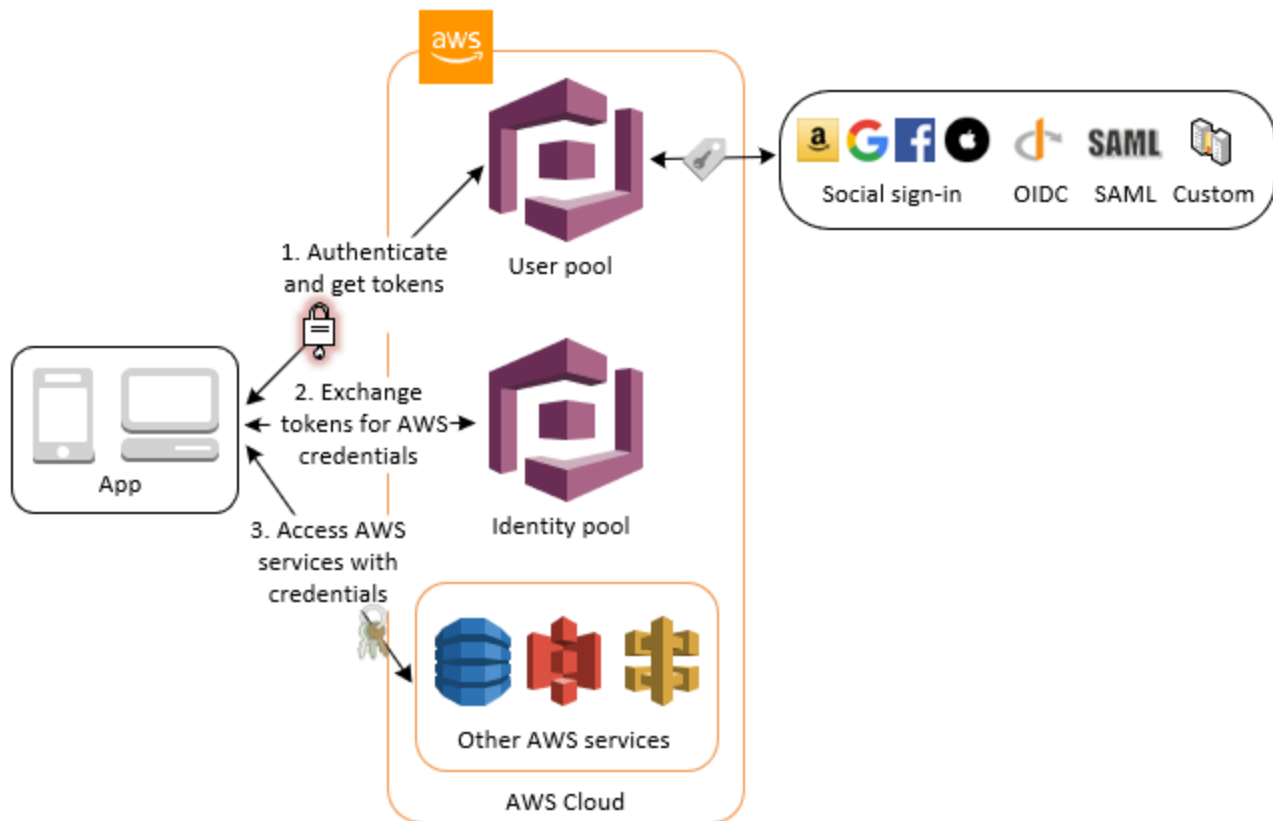
Key	Value
amplify-signin-with-hostedUI	false
aws.cognito.identity-id.eu-west-1:808169e9-545a-4e5d-afd6-f05a81ad33fb	eu-west-1:a9b285d4-bf65-4b52-b5bd-2cc3df235ea3
aws.cognito.identity-providers.eu-west-1:808169e9-545a-4e5d-afd6-f05a81ad33fb	cognito-idp.eu-west-1.amazonaws.com/eu-west-1_aQfCzHwCz
CognitoIdentityServiceProvider.3fe6quse1kki13s6nebi0ncnin.julianwood.accessToken	eyJraWQlOUIJSld3RHZPdWJDZ24zN0w1R0xMRnRnQjNHK0gJMU5qb1I3SWWhUSIBcL2Fqbz0iLCJhb...
CognitoIdentityServiceProvider.3fe6quse1kki13s6nebi0ncnin.julianwood.clockDrift	1
CognitoIdentityServiceProvider.3fe6quse1kki13s6nebi0ncnin.julianwood.idToken	eyJraWQlOUIxcGpXa1VEbW5MN2IsMGpDZ2ZaV2d5TzVBTK1nWFd6Tk91N1FnK0I1aXk4PSlmlFszY...
CognitoIdentityServiceProvider.3fe6quse1kki13s6nebi0ncnin.julianwood.refreshToken	eyJjdHkiOUKV1QlLCJlbmMiOUIBMjU2R0Nniw1VxnljoiUINBLU9BRVAIfQ.gfLEfwOvzsTONhTaQID...
CognitoIdentityServiceProvider.3fe6quse1kki13s6nebi0ncnin.julianwood.userData	[{"UserAttributes":{"Name":"sub","Value":"a900ae6c-99b9-4112-9677-b66fe79d792c"},"Name":"e...
CognitoIdentityServiceProvider.3fe6quse1kki13s6nebi0ncnin.LastAuthUser	julianwood

View tokens with browser developer tools

I copy the `.idToken` value and use the decoder at <https://jwt.io/> to view the contents.

Identity pools have identities that are either authenticated or unauthenticated. Unauthenticated identities typically belong to guest users. Authenticated identities belong to authenticated users who have received a token by a login provider, such as a user pool. The Amazon Cognito issued user pool tokens are exchanged for AWS access credentials from an identity pool.

Identity pools have identities that are either authenticated or unauthenticated. Every user who uses the client is given a unique identity. If they have not authenticated then that identity is an “unauthenticated identity”. Once they authenticate, the identity becomes an authenticated identity. The Amazon Cognito issued user pool tokens are exchanged for AWS access credentials from an identity pool.



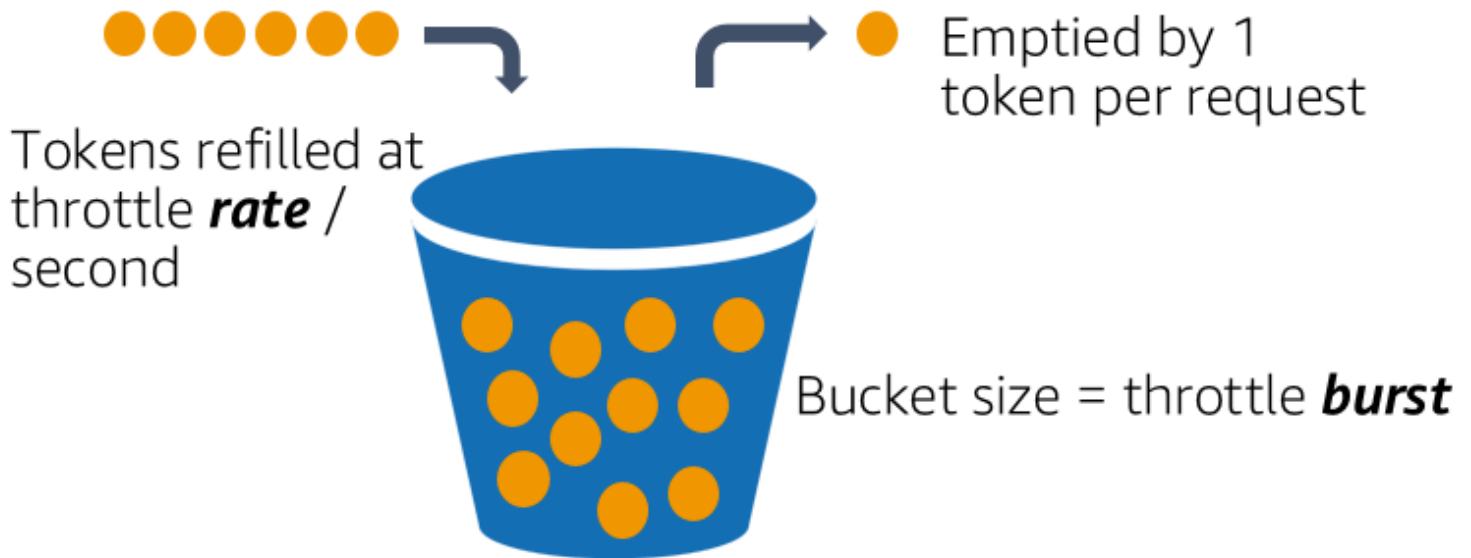
JWT-tokens-from-Amazon-Cognito-user-pool-exchanged-for-AWS-credentials-from-Amazon-Cognito-identity-pool

API keys

For public content and unauthenticated access, both Amazon API Gateway and AWS AppSync provide API keys that can be used to track usage. API keys should not be used as a primary authorization method for production applications, however they should still be treated as secrets. Instead, use these for rate limiting and throttling. Unauthenticated APIs require stricter throttling than authenticated APIs.

API Gateway usage plans specify who can access API stages and methods, and also how much and how fast they can access them. API keys are then associated with the usage plans to identify API clients and meter access for each key. Throttling and quota limits are enforced on individual keys.

Throttling limits determine how many requests per second are allowed for a usage plan. This is useful to prevent a client from overwhelming a downstream resource. There are two API Gateway values to control this, the throttle *rate* and throttle *burst*, which use the [token bucket algorithm](#). The algorithm is based on an analogy of filling and emptying a bucket of tokens representing the number of available requests that can be processed. The bucket in the algorithm has a fixed size based on the throttle *burst* and is filled at the token *rate*. Each API request removes a token from the bucket. The throttle *rate* then determines how many requests are allowed per second. The throttle *burst* determines how many concurrent requests are allowed and is shared across all APIs per Region in an account.



Token bucket algorithm

Quota limits allow you to set a maximum number of requests for an API key within a fixed time period. When billing for usage, this also allows you to enforce a limit when a client pays by monthly volume.

API keys are passed using the `x-api-key` header. API Gateway rejects requests without them.

For example, within the [serverless airline](#), the [loyalty](#) service uses an [AWS Lambda function](#) to fetch loyalty points and next tier progress via an API Gateway REST API `/loyalty/{customerId}/get` resource.

I can use this API to simulate the effect of usage plans with API keys.

1. I navigate to the *airline-loyalty* API `/loyalty/{customerId}/get` resource in [API Gateway console](#).
2. I change the **API Key Required** value to be `true`.

The screenshot shows the AWS API Gateway console. The breadcrumb trail at the top is: APIs > Airline-Loyalty-develop (xu38bz5b47) > Resources > /loyalty/{customerId} (yi3xvo) > GET. On the left, the 'Resources' tree shows the path: / > /loyalty > /{customerId} > GET. The 'Actions' menu is open, and 'Method Execution' is selected. The main panel is titled '/loyalty/{customerId} - GET - Method Request'. It contains a description: 'Provide information about this method's authorization settings and the parameters it can receive.' Below this is the 'Settings' section. It includes 'Authorization' set to 'AWS_IAM', 'Request Validator' set to 'all', and 'API Key Required' set to 'true'. The 'API Key Required' setting is highlighted with a red rectangular box.

Setting API Key Required on API Gateway method

1. I choose **Deploy API** from the *Actions* menu.
2. I create a usage plan in the *Usage Plans* section of the API Gateway Console.
3. I choose **Create** and enter a name for the usage plan.
4. I select **Enable throttling** and set the rate to one request per second and the burst to two requests. These are artificially low numbers to simulate the effect.
5. I select **Enable quota** and set the limit to 10 requests per day.

Create Usage Plan

Usage Plans help you meter API usage. With Usage Plans, you can enforce a throttling and quota limit on each API key. Throttling limits define the maximum number of requests per second available to each key. Quota limits define the number of requests each API key is allowed to make over a period.

Name* RestrictUsage

Description

Throttling

Enable throttling ☒ ⓘ

Rate* 1 requests per second ⓘ

Burst* 2 requests ⓘ

Quota

Enable quota ☒ ⓘ

10 requests per Day ⓘ

Create API Gateway usage plan

1. I click **Next**.
2. I associate an API Stage by choosing **Add API Stage**, and selecting the airline *Loyalty API* and *Prod* Stage.

Associated API Stages

Associate API stages to this usage plan. Subscribers will only be allowed to access the API stages that are associated with the plan. Choose "Add API Stage" below, then use the dropdown to select an API and stage to enable for this usage plan.

Add API Stage

API	Stage	Method Throttling	
Airline-Loyalty-develop	Prod	No Methods Configured	Configure Method Throttling 

Associate usage plan to API Gateway stage


1. I click **Next**, and choose **Create API Key and add to Usage Plan**

Usage Plan API Keys

Subscribe an API key to this usage plan. Choose "Add API Key" below to search through your existing API keys. Once a key is associated with a plan, API Gateway will meter all requests from the key and apply the plan's throttling and quota limits.

Add API Key to Usage Plan

Create API Key and add to Usage Plan

Results per page 100 

Name

No associated API keys

Create API key and add to usage plan.

1. I name the API Key and ensure it is set to *Auto Generate*.

Create an API Key and add it to the Usage Plan

Name*

API key* ☒ Auto Generate ☐ Custom

1. I choose **Save** then **Done** to associate the API key with the usage plan.

The screenshot displays the AWS API Gateway console for a usage plan named 'RestrictUsage'. The 'Details' tab shows the plan's ID (jcfq33), name, description, and throttling settings: Rate (1 request per second), Burst (2 requests), and Quota (10 requests per day). The 'Associated API Stages' section shows a table with one stage, 'Prod', associated with the 'Airline-Loyalty-develop' API. The 'API Keys' tab is also visible, showing a search bar and a list of API keys, with 'julianwood' being the only key listed.

API key associated with usage plan

I test the API authentication, in addition to the throttles and limits using [Postman](#).

I issue a **GET** request against the API Gateway URL using a **customerId** from the airline *Airline-LoyaltyData* [Amazon DynamoDB](#) table. I don't specify any authorization or API key.

The screenshot shows the Postman interface for an unauthenticated GET request. The URL is `https://[redacted].execute-api.eu-west-1.amazonaws.com/Prod/loyalty/a900ae6c-99b9-4112-9677-b66fe79d792c`. The request is sent without any authorization. The response status is **403 Forbidden** with a message: `"message": "Missing Authentication Token"`.

Postman unauthenticated GET request

I receive a **Missing Authentication Token** reply, which I expect as the API uses IAM authentication and I haven't authenticated.

I then configure authentication details within the *Authorization* tab, using an *AWS Signature*. I enter my AWS user account's **AccessKey** and **SecretKey**, which has an associated IAM identity policy to access the API.

The screenshot shows the Postman interface for a GET request to `https://[redacted].execute-api.eu-west-1.amazonaws.com/Prod/loyalty/a900ae6c-99b9-4112-9677-b66fe79d792c`. The **Authorization** tab is selected, showing the **AWS Signature** type. The **AccessKey** and **SecretKey** fields are highlighted with a red box. The **ADVANCED** section shows the **AWS Region** set to `eu-west-1`, **Service Name** set to `e.g. s3`, and **Session Token** set to `Session Token`. The **Body** tab shows the response: `{ "message": "Forbidden" }`, which is also highlighted with a red box. The status is **403 Forbidden**, time is **27 ms**, and size is **349 B**.

Postman authenticated GET request without access key

I receive a **Forbidden** reply. I have successfully authenticated, but the API Gateway method rejects the request as it requires an API key, which I have not provided.

I retrieve and copy my previously created API key from the API Gateway console *API Keys* section, and display it by choosing **Show**.

The screenshot shows the AWS API Gateway console for an API key named **julianwood**. The **Delete API Key** and **Configure Tags** buttons are visible. The **ID** is `fr92a5qpbj`, the **Name** is `julianwood`, and the **API key** is `67sfLSq19U1GuGk1RB5DwaS7EIYsWvBU3fZbtvQT`, which is highlighted with a red box. The **Description** is `No description.` and the **Enabled** status is `Enabled` with an information icon.

Retrieve API key.

I then configure an `x-api-key` header in the Postman *Headers* section and paste the API key value.

Having authenticated and specifying the required API key, I receive a response from the API with the loyalty points value.

The screenshot shows the Postman interface for a GET request to the URL `https://[redacted].execute-api.eu-west-1.amazonaws.com/Prod/loyalty/a900ae6c-99b9-4112-9677-b66fe79d792c`. The **Headers** tab is selected, showing a table with one header: `x-api-key` with the value `67sfLSq19U1GuGk1RB5DwaS7EIYsWvBU3fZbtvQT`. The **Body** tab is also visible, showing a JSON response: `{ "points": 900, "level": "bronze", "remainingPoints": 49100 }`. The status bar at the bottom indicates a **200 OK** response with a time of 86 ms and a size of 336 B.

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> x-api-key	67sfLSq19U1GuGk1RB5DwaS7EIYsWvBU3fZbtvQT	

Key	Value	Description


```
{
  "points": 900,
  "level": "bronze",
  "remainingPoints": 49100
}
```

Postman successful authenticated GET request with access key

I then call the API with a number of quick successive requests.

When I exceed the throttle rate limit of one request per second, and the throttle burst limit of two requests, I receive:

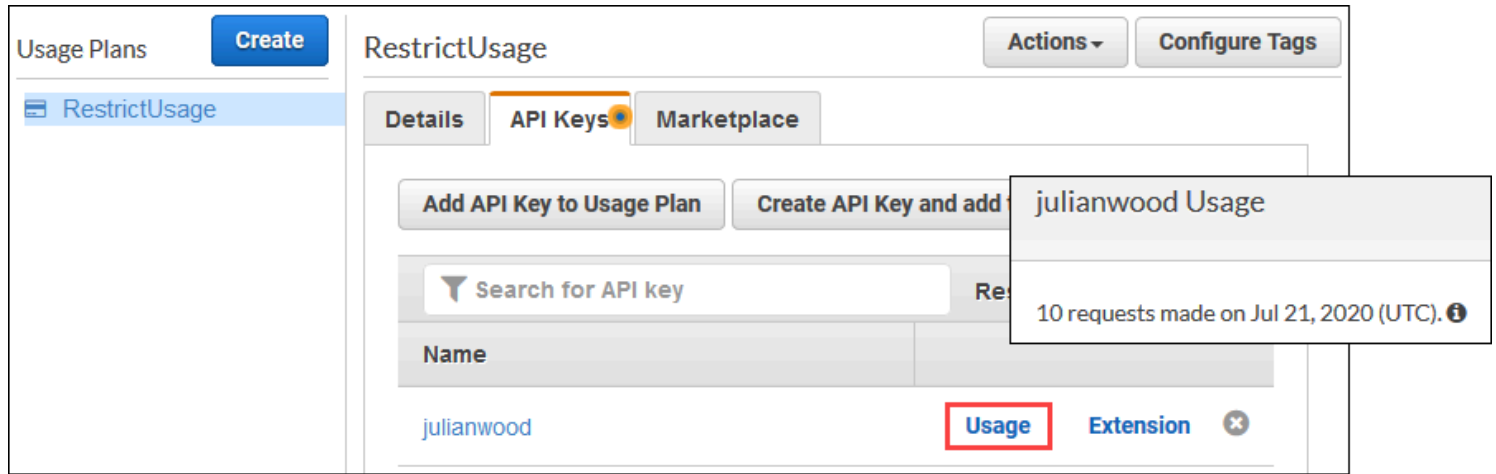
```
{"message": "Too Many Requests"}
```

When I then exceed the quota of 10 requests per day, I receive:

```
{"message": "Limit Exceeded"}
```

I view the API key usage within the API Gateway console *Usage Plan* section.

I select the usage plan, choose the *API Keys* section, then choose **Usage**. I see how many requests I have made.



View API key usage

If necessary, I can also grant a temporary rate extension for this key.

For more information on using API Keys for unauthenticated access for AWS AppSync, see the [documentation](#).

API Gateway also has support for [AWS Web Application Firewall](#) (AWS WAF) which helps protect web applications and APIs from attacks. It is another mechanism to apply rate-based rules to prevent public API consumers exceeding a configurable request threshold. AWS WAF also protects your website from common attack techniques like SQL injection and Cross-Site Scripting (XSS). You can also create rules that can block attacks from specific user-agents, bad bots, content scrapers, or geographies. AWS WAF rules are evaluated before other access control features, such as resource policies, IAM policies, Lambda authorizers, and Amazon Cognito authorizers. For more information, see [“Using AWS WAF with Amazon API Gateway”](#).

AWS AppSync APIs have [built-in DDoS protection](#) to protect all GraphQL API endpoints from attacks.

Improvement plan summary:

1. Determine your API consumer and choose an API endpoint type.
2. Implement security mechanisms appropriate to your API endpoint

Conclusion

Controlling serverless application API access using authentication and authorization mechanisms can help protect against unauthorized access and prevent unnecessary use of resources.

In this post, I cover using Amplify CLI to add a GraphQL API with an Amazon Cognito user pool handling authentication. I explain how to view JSON Web Token (JWT) claims, and how to use identity pools to grant temporary access to AWS services. I also show how to use API keys and API Gateway usage plans for rate limiting and throttling requests.

This well-architected question continues in [part 3](#) where I look at separating authenticated users into logical groups. I show how to use Amazon Cognito user pool groups to separate users with an Amazon Cognito authorizer to control

access to an API Gateway method. I then show how to pass JWTs to a Lambda function to perform authorization within a function. I then explain how to also segregate users using custom scopes by defining an Amazon Cognito resource server.

TAGS: [Amazon API Gateway](#), [Amazon Cognito](#), [AppSync](#), [AWS Amplify](#), [AWS CloudFormation](#), [AWS Lambda](#), [AWS WAF](#), [IAM](#), [serverless](#), [well-architected](#)