Set up a method request in API Gateway

PDF | RSS

Setting up a method request involves performing the following tasks, after creating a RestApi resource:

1. Creating a new API or choosing an existing API Resource entity.
2. Creating an API Method resource that is a specific HTTP verb on the new or chosen API `Resource`. This task can be further divided into the following sub tasks:
   - Adding an HTTP method to the method request
   - Configuring request parameters
   - Defining a model for the request body
   - Enacting an authorization scheme
   - Enabling request validation

You can perform these tasks using the following methods:

- API Gateway console
- AWS CLI commands (create-resource and put-method)
- AWS SDK functions (for example, in Node.js, createResource and putMethod)
- API Gateway REST API (resource:create and method:put).

Topics

- Set up method request validation

Set up API resources

In an API Gateway API, you expose addressable resources as a tree of API Resources entities, with the root resource (`/`) at the top of the hierarchy. The root resource is relative to the API's base URL, which consists of the API endpoint and a stage name. In the API Gateway console, this base URI is referred to as the **Invoke URI** and is displayed in the API's stage editor after the API is deployed.

The API endpoint can be a default host name or a custom domain name. The default host name is of the following format:

```
{api-id}.execute-api.{region}.amazonaws.com
```

In this format, the `{api-id}` represents the API identifier that is generated by API Gateway. The `{region}` variable represents the AWS Region (for example, `us-east-1`) that you chose when creating the API. A custom domain name is any user-friendly name under a valid internet domain. For example, if you have registered an internet domain of `example.com`, any of `*.example.com` is a valid custom domain name. For more information, see create a custom domain name.

For the PetStore sample API, the root resource (`/`) exposes the pet store. The `/pets` resource represents the collection of pets available in the pet store. The `/pets/{petId}` exposes an individual pet of a given identifier (`petId`). The path parameter of `{petId}` is part of the request parameters.

To set up an API resource, you choose an existing resource as its parent and then create the child resource under this parent resource. You start with the root resource as a parent, add a resource to this parent, add another resource to this child resource as the new parent, and so on, to its parent identifier. Then you add the named resource to the parent.

With AWS CLI, you can call the `get-resources` command to

find out which resources of an API are available:

```
aws apigateway get-resources --rest-api-id <api
                             --region <region>
```

The result is a list of the currently available resources of the API. For our PetStore sample API, this list looks like the following:

```
{
    "items": [
        {
            "path": "/pets",
            "resourceMethods": {
                "GET": {}
            },
            "id": "6sxz2j",
            "pathPart": "pets",
            "parentId": "svzr2028x8"
        },
        {
            "path": "/pets/{petId}",
            "resourceMethods": {
                "GET": {}
            },
            "id": "rjkmth",
            "pathPart": "{petId}",
            "parentId": "6sxz2j"
        },
        {
            "path": "/",
            "id": "svzr2028x8"
        }
    ]
}
```

Each item lists the identifiers of the resource (`id`) and, except for the root resource, its immediate parent (`parentId`), as well as the resource name (`pathPart`). The root resource is special in that it does not have any parent. After choosing a resource

as the parent, call the following command to add a child resource.

```
aws apigateway create-resource --rest-api-id <
                               --region <region> \
                               --parent-id <parentId
                               --path-part <resource
```

For example, to add pet food for sale on the PetStore website, add a `food` resource to the root (/) by setting `path-part` to `food` and `parent-id` to `svzr2028x8`. The result looks like the following:

```
{
    "path": "/food",
    "pathPart": "food",
    "id": "xdsvhp",
    "parentId": "svzr2028x8"
}
```

Use a proxy resource to streamline API setup

As business grows, the PetStore owner may decide to add food, toys, and other pet-related items for sale. To support this, you can add `/food`, `/toys`, and other resources under the root resource. Under each sale category, you may also want to add more resources, such as `/food/{type}/{item}`, `/toys/{type}/{item}`, etc. This can get tedious. If you decide to add a middle layer `{subtype}` to the resource paths to change the path hierarchy into `/food/{type}/{subtype}/{item}`, `/toys/{type}/{subtype}/{item}`, etc., the changes will break the existing API set up. To avoid this, you can use an API Gateway proxy resource to expose a set of API resources all at once.

API Gateway defines a proxy resource as a placeholder for a resource to be specified when the request is submitted. A proxy resource is expressed by a special path parameter of

`{proxy+}`, often referred to as a greedy path parameter. The `+` sign indicates whichever child resources are appended to it. The `/parent/{proxy+}` placeholder stands for any resource matching the path pattern of `/parent/*`. The greedy path parameter name, `proxy`, can be replaced by another string in the same way you treat a regular path parameter name.

Using the AWS CLI, you call the following command to set up a proxy resource under the root (`/{proxy+}`):

```
aws apigateway create-resource --rest-api-id <
                               --region <region> \
                               --parent-id <rootReso
                               --path-part {proxy+}
```

The result is similar to the following:

```
{
    "path": "/{proxy+}",
    "pathPart": "{proxy+}",
    "id": "234jdr",
    "parentId": "svzr2028x8"
}
```

For the `PetStore` API example, you can use `/{proxy+}` to represent both the `/pets` and `/pets/{petId}`. This proxy resource can also reference any other (existing or to-be-added) resources, such as `/food/{type}/{item}`, `/toys/{type}/{item}`, etc., or `/food/{type}/{subtype}/{item}`, `/toys/{type}/{subtype}/{item}`, etc. The backend developer determines the resource hierarchy and the client developer is responsible for understanding it. API Gateway simply passes whatever the client submitted to the backend.

An API can have more than one proxy resource. For example, the following proxy resources are allowed within an API.

```
/{proxy+}
/parent/{proxy+}
```

```
/parent/{child}/{proxy+}
```

When a proxy resource has non-proxy siblings, the sibling resources are excluded from the representation of the proxy resource. For the preceding examples, `/{proxy+}` refers to any resources under the root resource except for the `/parent[/*]` resources. In other words, a method request against a specific resource takes precedence over a method request against a generic resource at the same level of the resource hierarchy.

A proxy resource cannot have any child resource. Any API resource after `{proxy+}` is redundant and ambiguous. The following proxy resources are not allowed within an API.

```
/{proxy+}/child
/parent/{proxy+}/{child}
/parent/{child}/{proxy+}/{grandchild+}
```

Set up an HTTP method

An API method request is encapsulated by the API Gateway Method resource. To set up the method request, you must first instantiate the `Method` resource, setting at least an HTTP method and an authorization type on the method.

Closely associated with the proxy resource, API Gateway supports an HTTP method of `ANY`. This `ANY` method represents any HTTP method that is to be supplied at run time. It allows you to use a single API method setup for all of the supported HTTP methods of `DELETE`, `GET`, `HEAD`, `OPTIONS`, `PATCH`, `POST`, and `PUT`.

You can set up the `ANY` method on a non-proxy resource as well. Combining the `ANY` method with a proxy resource, you get a single API method setup for all of the supported HTTP methods against any resources of an API. Furthermore, the backend can evolve without breaking the existing API setup.

Before setting up an API method, consider who can call the method. Set the authorization type according to your plan. For

open access, set it to `NONE`. To use IAM permissions, set the authorization type to `AWS_IAM`. To use a Lambda authorizer function, set this property to `CUSTOM`. To use an Amazon Cognito user pool, set the authorization type to `COGNITO_USER_POOLS`.

The following AWS CLI command shows how to create a method request of the `ANY` verb against a specified resource (`6sxz2j`), using the IAM permissions to control its access.

```
aws apigateway put-method --rest-api-id vaz7da
       --resource-id 6sxz2j \
       --http-method ANY \
       --authorization-type AWS_IAM \
       --region us-west-2
```

To create an API method request with a different authorization type, see Set up method request authorization.

Set up method request parameters

Method request parameters are a way for a client to provide input data or execution context necessary to complete the method request. A method parameter can be a path parameter, a header, or a query string parameter. As part of method request setup, you must declare required request parameters to make them available for the client. For non-proxy integration, you can translate these request parameters to a form that is compatible with the backend requirement.

For example, for the `GET /pets/{petId}` method request, the `{petId}` path variable is a required request parameter. You can declare this path parameter when calling the `put-method` command of the AWS CLI. This is illustrated as follows:

```
aws apigateway put-method --rest-api-id vaz7da
       --resource-id rjkmth \
       --http-method GET \
       --authorization-type "NONE" \
```

```
        --region us-west-2 \
        --request-parameters method.request.path.pe
```

If a parameter is not required, you can set it to `false` in `request-parameters`. For example, if the `GET /pets` method uses an optional query string parameter of `type`, and an optional header parameter of `breed`, you can declare them using the following CLI command, assuming that the `/pets` resource `id` is `6sxz2j`:

```
aws apigateway put-method --rest-api-id vaz7da
        --resource-id 6sxz2j \
        --http-method GET \
        --authorization-type "NONE" \
        --region us-west-2 \
        --request-parameters method.request.querystr
```

Instead of this abbreviated form, you can use a JSON string to set the `request-parameters` value:

```
'{"method.request.querystring.type":false,"meth
```

With this setup, the client can query pets by type:

```
GET /pets?type=dog
```

And the client can query dogs of the poodle breed as follows:

```
GET /pets?type=dog
breed:poodle
```

For information on how to map method request parameters to integration request parameters, see Setting up REST API integrations.

Set up method request model

For an API method that can take input data in a payload, you can use a model. A model is expressed in a JSON schema draft 4 and describes the data structure of the request body. With a model, a client can determine how to construct a method request payload as input. More importantly, API Gateway uses the model to validate a request, generate an SDK, and initialize a mapping template for setting up the integration in the API Gateway console. For information about how to create a model, see Understanding data models.

Depending on the content types, a method payload can have different formats. A model is indexed against the media type of the applied payload. API Gateway uses the `Content-Type` request header to determine the content type. To set up method request models, add key-value pairs of the `"<media-type>":"<model-name>"` format to the `requestModels` map when calling the AWS CLI `put-method` command.

To use the same model regardless of the content type, specify `$default` as the key.

For example, to set a model on the JSON payload of the `POST /pets` method request of the PetStore example API, you can call the following AWS CLI command:

```
aws apigateway put-method \
       --rest-api-id vaz7da96z6 \
       --resource-id 6sxz2j \
       --http-method POST \
       --authorization-type "NONE" \
       --region us-west-2 \
       --request-models '{"application/json":"petM
```

Here, `petModel` is the `name` property value of a `Model` resource describing a pet. The actual schema definition is expressed as a JSON string value of the `schema` property of the `Model` resource.

In a Java, or other strongly typed SDK, of the API, the input

data is cast as the `petModel` class derived from the schema definition. With the request model, the input data in the generated SDK is cast into the `Empty` class, which is derived from the default `Empty` model. In this case, the client cannot instantiate the correct data class to provide the required input.

Set up method request authorization

To control who can call the API method, you can configure the authorization type on the method. You can use this type to enact one of the supported authorizers, including IAM roles and policies (`AWS_IAM`), an Amazon Cognito user pool (`COGNITO_USER_POOLS`), or a Lambda authorizer (`CUSTOM`).

To use IAM permissions to authorize access to the API method, set the `authorization-type` input property to **`AWS_IAM`**. When you set this option, API Gateway verifies the caller's signature on the request based on the caller's credentials. If the verified user has permission to call the method, it accepts the request. Otherwise, it rejects the request and the caller receives an unauthorized error response. The call to the method doesn't succeed unless the caller has permission to invoke the API method. The following IAM policy grants permission to the caller to call any API methods created within the same AWS account:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "execute-api:Invoke"
            ],
            "Resource": "arn:aws:execute-api:*:*:*"
        }
    ]
}
```

For more information, see Control access to an API with IAM permissions.

Currently, you can only grant this policy to the users, groups, and roles within the API owner's AWS account. Users from a different AWS account can call the API methods only if allowed to assume a role within the API owner's AWS account with the necessary permissions to call the `execute-api:Invoke` action. For information on cross-account permissions, see Using IAM Roles.

You can use AWS CLI, an AWS SDK, or a REST API client, such as Postman, which implements Signature Version 4 (SigV4) signing.

To use a Lambda authorizer to authorize access to the API method, set the `authorization-type` input property to `CUSTOM` and set the `authorizer-id` input property to the `id` property value of a Lambda authorizer that already exists. The referenced Lambda authorizer can be of the `TOKEN` or `REQUEST` type. For information about creating a Lambda authorizer, see Use API Gateway Lambda authorizers.

To use an Amazon Cognito user pool to authorize access to the API method, set the `authorization-type` input property to `COGNITO_USER_POOLS` and set the `authorizer-id` input property to the `id` property value of the `COGNITO_USER_POOLS` authorizer that was already created. For information about creating an Amazon Cognito user pool authorizer, see Control access to a REST API using Amazon Cognito user pools as authorizer.

Set up method request validation

You can enable request validation when setting up an API method request. You need to first create a request validator:

```
aws apigateway create-request-validator \
    --rest-api-id 7zw9uyk9kl \
    --name bodyOnlyValidator \
    --validate-request-body  \
```

```
    --no-validate-request-parameters
```

This CLI command creates a body-only request validator. Example output is as follows:

```
{
    "validateRequestParameters": false,
    "validateRequestBody": true,
    "id": "jgpyy6",
    "name": "bodyOnlyValidator"
}
```

With this request validator, you can enable request validation as part of the method request setup:

```
aws apigateway put-method \
    --rest-api-id 7zw9uyk9kl
    --region us-west-2
    --resource-id xdsvhp
    --http-method PUT
    --authorization-type "NONE"
    --request-parameters '{"method.request.querystr
    --request-models '{"application/json":"petMode
    --request-validator-id jgpyy6
```

To be included in request validation, a request parameter must be declared as required. If the query string parameter for the page is used in request validation, the `request-parameters` map of the preceding example must be specified as `'{"method.request.querystring.type": false, "method.request.querystring.page":true}'`.

Next topic: Set up method response

Previous topic: Methods

Need help?

- Try AWS re:Post
- Connect with an AWS IQ expert