Azure REST API reference

Article • 04/04/2023

Welcome to the Azure REST API reference documentation.

Representational State Transfer (REST) APIs are service endpoints that support sets of HTTP operations (methods), which provide create, retrieve, update, or delete access to the service's resources. This article walks you through:

- How to call Azure REST APIs with Postman
- The basic components of a REST API request/response pair.
- How to register your client application with Azure Active Directory (Azure AD) to secure your REST requests.
- Overviews of creating and sending a REST request, and handling the response.

> **Tip**
>
> Most Azure service REST APIs have client libraries that provide a native interface for using Azure services:
>
> .NET | Java | Node.js | Python

How to call Azure REST APIs with Postman

The following video will show you how to quickly authenticate with the Azure REST APIs via the client id/secret method. We encourage you continue reading below to learn about what constitutes a REST operation, but if you need to quickly call the APIs, this video is for you.
https://www.youtube-nocookie.com/embed/6b1J03fDnOg

You can read the full walk-through on Jon Gallant's blog here: Azure REST APIs with Postman

How to call Azure REST APIs with curl

The process described in the following blog entry is similar to the one used for Postman, but shows how to call an Azure REST API using curl .You might consider using curl in unattended scripts, for example in DevOps automation scenarios.

Calling Azure REST API via curl

Components of a REST API request/response

A REST API request/response pair can be separated into five components:

1. The **request URI**, which consists of: `{URI-scheme} :// {URI-host} / {resource-path} ? {query-string}`. Although the request URI is included in the request message header, we call it out separately here because most languages or frameworks require you to pass it separately from the request message.

    - URI scheme: Indicates the protocol used to transmit the request. For example, `http` or `https`.
    - URI host: Specifies the domain name or IP address of the server where the REST service endpoint is hosted, such as `graph.microsoft.com`.
    - Resource path: Specifies the resource or resource collection, which may include multiple segments used by the service in determining the selection of those resources. For example: `beta/applications/00003f25-7e1f-4278-9488-efc7bac53c4a/owners` can be used to query the list a specific application's owners within the applications collection.
    - Query string (optional): Provides additional simple parameters, such as the API version or resource selection criteria.

2. HTTP **request message header** fields:

    - A required HTTP method (also known as an operation or verb), which tells the service what type of operation you are requesting. Azure REST APIs support GET, HEAD, PUT, POST, and PATCH methods.
    - Optional additional header fields, as required by the specified URI and HTTP method. For example, an Authorization header that provides a bearer token containing client authorization information for the request.

3. Optional HTTP **request message body** fields, to support the URI and HTTP operation. For example, POST operations contain MIME-encoded objects that are passed as complex parameters. For POST or PUT operations, the MIME-encoding

type for the body should be specified in the `Content-type` request header as well. Some services require you to use a specific MIME type, such as `application/json`.

4. HTTP **response message header** fields:

    - An HTTP status code , ranging from 2xx success codes to 4xx or 5xx error codes. Alternatively, a service-defined status code may be returned, as indicated in the API documentation.
    - Optional additional header fields, as required to support the request's response, such as a `Content-type` response header.

5. Optional HTTP **response message body** fields:

    - MIME-encoded response objects are returned in the HTTP response body, such as a response from a GET method that is returning data. Typically, these objects are returned in a structured format such as JSON or XML, as indicated by the `Content-type` response header. For example, when you request an access token from Azure AD, it is returned in the response body as the `access_token` element, one of several name/value paired objects in a data collection. In this example, a response header of `Content-Type: application/json` is also included.

Register your client application with Azure AD

Most Azure services (such as Azure Resource Manager providers and the classic deployment model) require your client code to authenticate with valid credentials before you can call the service's API. Authentication is coordinated between the various actors by Azure AD, and provides your client with an access token as proof of the authentication. The token is then sent to the Azure service in the HTTP Authorization header of subsequent REST API requests. The token's claims also provide information to the service, allowing it to validate the client and perform any required authorization.

If you are using a REST API that does not use integrated Azure AD authentication, or you've already registered your client, skip to the Create the request section.

Prerequisites

Your client application must make its identity configuration known to Azure AD before run-time by registering it in an Azure AD tenant. Before you register your client with Azure AD, consider the following prerequisites:

- If you do not have an Azure AD tenant yet, see Set up an Azure Active Directory tenant.
- In accordance with the OAuth2 Authorization Framework, Azure AD supports two types of clients. Understanding each helps you decide which is most appropriate for your scenario:
  - web/confidential clients run on a web server and can access resources under their own identity (for example, a service or daemon), or obtain delegated authorization to access resources under the identity of a signed-in user (for example, a web app). Only a web client can securely maintain and present its own credentials during Azure AD authentication to acquire an access token.
  - native/public clients are installed and run on a device. They can access resources only under delegated authorization, using the identity of the signed-in user to acquire an access token on behalf of the user.
- The registration process creates two related objects in the Azure AD tenant where the application is registered: an application object and a service principal object. For more background on these components and how they are used at run-time, see Application and service principal objects in Azure Active Directory.

You are now ready to register your client application with Azure AD.

Client registration

To register a client that accesses an Azure Resource Manager REST API, see Use portal to create Active Directory application and service principal that can access resources. The article (also available in PowerShell and CLI versions for automating registration) shows you how to:

- Register the client application with Azure AD.
- Set permission requests to allow the client to access the Azure Resource Manager API.
- Configure Azure Resource Manager Role-Based Access Control (RBAC) settings for authorizing the client.

If your client accesses an API other than an Azure Resource Manager API, refer to:

- Register an application with the Microsoft identity platform
  - Register the client application with Azure AD, in the "Register an application" section.
  - Create a secret key (if you are registering a web client), in the "Add credentials" section.
- Configure an application to expose a web API
  - Add permissions to your web API, exposing them as scopes
- Configure a client application to access a web API
  - Add permission requests as required by the scopes defined for the API, in the "Add permissions to access your web API" section.

Now that you've completed registration of your client application, move on to your client code where you create the REST request and handle the response.

Create the request

This section covers the first three of the five components that we discussed earlier. You first need to acquire the access token from Azure AD, which you use to assemble your request message header.

Acquire an access token

After you have a valid client registration, you have two ways to integrate with Azure AD to acquire an access token:

- Platform- and language-neutral OAuth2 service endpoints, which we use in this article. The instructions provided in this section assume nothing about your client's platform or language/script when you use the Azure AD OAuth endpoints. The only requirement is that you can send/receive HTTPS requests to/from Azure AD, and parse the response message.
- The platform- and language-specific Microsoft Authentication Libraries (MSAL), which is beyond the scope of this article. The libraries provide asynchronous wrappers for the OAuth2 endpoint requests, and robust token-handling features such as caching and refresh token management. For more information, see the Overview of Microsoft Authentication Library (MSAL).

The two Azure AD endpoints that you use to authenticate your client and acquire an access token are referred to as the OAuth2 `/authorize` and `/token` endpoints. How you use them depends on your application's registration and the type of OAuth2 authorization grant flow you need to support your application at run-time. For the purposes of this article, we assume that your client uses one of the following authorization grant flows: authorization code or client credentials. To acquire an access token used in the remaining sections, follow the instructions for the flow that best matches your scenario.

Authorization code grant (interactive clients)

This grant is used by both web and native clients, requiring credentials from a signed-in user in order to delegate resource access to the client application. It uses the `/authorize` endpoint to obtain an authorization code (in response to user sign-in/consent), followed by the `/token` endpoint to exchange the authorization code for an access token.

1. First, your client needs to request an authorization code from Azure AD. For details on the format of the HTTPS GET request to the `/authorize` endpoint, and example request/response messages, see Request an authorization code. The URI contains the following query-string parameters, which are specific to your client application:

    - `client_id`: A GUID that was assigned to your client application during registration, also known as an application ID.

    - `redirect_uri`: A URL-encoded version of one of the reply/redirect URIs, specified during registration of your client application. The value you pass must match your registration value exactly.

    - `resource`: A URL-encoded identifier URI that's specified by the REST API you are calling. Web/REST APIs (also known as resource applications) can expose one or more application ID URIs in their configuration. For example:
        - Azure Resource Manager provider (and classic deployment model) APIs use `https://management.core.windows.net/`.
        - For any other resources, see the API documentation or the resource application's configuration in the Azure portal. For more information, see the `identifierUris` property of the Microsoft Graph API application object.

The request to the `/authorize` endpoint first triggers a sign-in prompt to authenticate the user. The response you get back is delivered as a redirect (302) to the URI that you specified in `redirect_uri`. The response header message contains a `location` field, containing the redirect URI followed by a `code` query parameter. The `code` parameter contains the authorization code that you need for step 2.

2. Next, your client needs to redeem the authorization code for an access token. For details on the format of the HTTPS POST request to the `/token` endpoint and request/response examples, see Request an access token. Because this is a POST request, you package your application-specific parameters in the request body. In addition to some of the previously mentioned parameters (along with other new ones), you will pass:

   - `code`: This query parameter contains the authorization code that you obtained in step 1.
   - `client_secret`: You need this parameter only if your client is configured as a web application. This is the same secret/key value that you generated earlier, in client registration.

Client credentials grant (non-interactive clients)

This grant is used only by web clients, allowing the application to access resources directly (no user delegation) using the client's credentials, which are provided at registration time. The grant is typically used by non-interactive clients (no UI) that run as a service or daemon. It requires only the `/token` endpoint to acquire an access token.

The client/resource interactions for this grant are similar to step 2 of the authorization code grant. For details on the format of the HTTPS POST request to the `/token` endpoint and request/response examples, see the "Get a token" section in Microsoft identity platform and the OAuth 2.0 client credentials flow.

Assemble the request message

Most programming languages or frameworks and scripting environments make it easy to assemble and send the request message. They typically provide a web/HTTP class or API that abstracts the creation or formatting of the request, making it easier to write the client code (the HttpWebRequest class in the .NET Framework, for example). For brevity, and because most of the task is handled for you, this section covers only the important elements of the request.

Request URI

Because sensitive information is being transmitted and received, all REST requests require the HTTPS protocol for the URI scheme, giving the request and response a secure channel. The information (that is, the Azure AD authorization code, access/bearer token, and sensitive request/response data) is encrypted by a lower transport layer, ensuring the privacy of the messages.

The remainder of your service's request URI (the host, resource path, and any required query-string parameters) are determined by its related REST API specification. For example, Azure Resource Manager provider APIs use `https://management.azure.com/`, and Azure classic deployment model uses `https://management.core.windows.net/`. Both require an `api-version` query-string parameter.

Request header

The request URI is bundled in the request message header, along with any additional fields required by your service's REST API specification and the HTTP specification .
Your request might require the following common header fields:

- `Authorization`: Contains the OAuth2 bearer token to secure the request, as acquired earlier from Azure AD.
- `Content-Type`: Typically set to "application/json" (name/value pairs in JSON format), and specifies the MIME type of the request body.
- `Host`: The domain name or IP address of the server where the REST service endpoint is hosted.

Request body

As mentioned earlier, the request message body is optional, depending on the specific operation you're requesting and its parameter requirements. If it's required, the API specification for the service you are requesting also specifies the encoding and format.

The request body is separated from the header by an empty line, formatted in accordance with the `Content-Type` header field. An example of an "application/json" formatted body would appear as follows:

JSON

```json
{
  "<name>": "<value>"
}
```

Send the request

Now that you have the service's request URI and have created the related request message header and body, you are ready to send the request to the REST service endpoint.

For example, you might send an HTTPS GET request method for an Azure Resource Manager provider by using request header fields that are similar to the following (note that the request body is empty):

HTTP

```http
GET /subscriptions?api-version=2014-04-01-preview HTTP/1.1
Authorization: Bearer <bearer-token>
Host: management.azure.com

<no body>
```

And you might send an HTTPS PUT request method for an Azure Resource Manager provider, by using request header *and* body fields similar to the following example:

HTTP

```
PUT /subscriptions/.../resourcegroups/ExampleResourceGroup?api-
version=2016-02-01  HTTP/1.1
Authorization: Bearer <bearer-token>
Content-Length: 29
Content-Type: application/json
Host: management.azure.com

{
  "location": "West US"
}
```

After you make the request, the response message header and optional body are returned.

Process the response message

The process concludes with the final two of the five components.

To process the response, parse the response header and, optionally, the response body (depending on the request). In the HTTPS GET example provided in the preceding section, you used the /subscriptions endpoint to retrieve the list of subscriptions for a user. Assuming that the response was successful, you should receive response header fields that are similar to the following example:

```
HTTP/1.1 200 OK
Content-Length: 303
Content-Type: application/json;
```

And you should receive a response body that contains a list of Azure subscriptions and their individual properties encoded in JSON format, similar to:

JSON

```
{
    "value":[
        {
        "id":"/subscriptions/...",
        "subscriptionId":"...",
        "displayName":"My Azure Subscription",
        "state":"Enabled",
"subscriptionPolicies":{
            "locationPlacementId":"Public_2015-09-01",
            "quotaId":"MSDN_2014-05-01",
            "spendingLimit":"On"}
        }
    ]
}
```

Similarly, for the HTTPS PUT example, you should receive a response header similar to the following, confirming that your PUT operation to add the "ExampleResourceGroup" was successful:

```
HTTP/1.1 200 OK
Content-Length: 193
Content-Type: application/json;
```

And you should receive a response body that confirms the content of your newly added resource group encoded in JSON format, similar to:

JSON

```json
{
    "id":"/subscriptions/.../resourceGroups/ExampleResourceGroup",
    "name":"ExampleResourceGroup",
    "location":"westus",
    "properties":
        {
        "provisioningState":"Succeeded"
        }
}
```

As with the request, most programming languages and frameworks make it easy to process the response message. They typically return this information to your application following the request, allowing you to process it in a typed/structured format. Mainly, you are interested in confirming the HTTP status code in the response header, and parsing the response body according to the API specification (or the `Content-Type` and `Content-Length` response header fields).

Async operations, throttling, and paging

The Create/Send/Process-Response pattern that's discussed in this article is synchronous and applies to all REST messages. However, some services also support an asynchronous pattern, which requires additional processing of response headers to monitor or complete the asynchronous request. For more information, see Track asynchronous Azure operations.

Resource Manager applies a limit on the number of read and write requests per hour to prevent an application from sending too many requests. If your application exceeds those limits, requests are throttled. The response header includes the number of remaining requests for your scope. For more information, see Throttling Resource Manager requests.

Some list operations return a property called `nextLink` in the response body. You see this property when the results are too large to return in one response. Typically, the response includes the nextLink property when the list operation returns more than 1,000 items. When nextLink isn't present in the results, the returned results are complete. When nextLink contains a URL, the returned results are just part of the total result set.

The response is in the format:

JSON

```json
{
  "value": [
    <returned-items>
  ],
  "nextLink": "https://management.azure.com/{operation}?api-version=
{version}&%24skiptoken={token}"
}
```

To get the next page of the results, send a GET request to the URL in the nextLink property. The URL includes a continuation token to indicate where you are in the results. Continue sending requests to the nextLink URL until it no longer contains a URL in the returned results.

Resiliency of Azure APIs

The Azure REST APIs are designed for resiliency and continuous availability. Control plane operations (requests sent to management.azure.com) in the REST API are:

- Distributed across regions. Some services are regional.
- Distributed across Availability Zones (as well regions) in locations that have multiple Availability Zones.
- Not dependent on a single logical data center.
- Never taken down for maintenance activities.

Related content

That's it. After you register your Azure AD application and have a modular technique for acquiring an access token and handling HTTP requests, it's fairly easy to replicate your code to take advantage of new REST APIs. For more information about application registration and the Azure AD programming model, see the Microsoft identity platform documentation.

For information about testing HTTP requests/responses, see:

- Fiddler . Fiddler is a free web debugging proxy that can intercept your REST requests, making it easy to diagnose the HTTP request/ response messages.
- JWT.ms , which make it quick and easy to dump the claims in your bearer token so you can validate their contents.