## **What is Azure API Management?**

Azure API Management is a service that allows us to manage all your APIs from one place.

**Component of Azure API** -Developer Portal, API Gateway, Azure Portal

Group

**Create an Instance of API**: Azure Portal, Azure CLI, ARM, Azure PowerShell, VSCode

**Group**: Administrator, Developer, Guest

Azure API Management helps customers meet these challenges:

* Abstract backend architecture diversity and complexity from API consumers
* Securely expose services hosted on and outside of Azure as APIs
* Protect, accelerate, and see APIs
* Enable API discovery and consumption by internal and external users

## **Products, API’s and Operations**

**Operation**

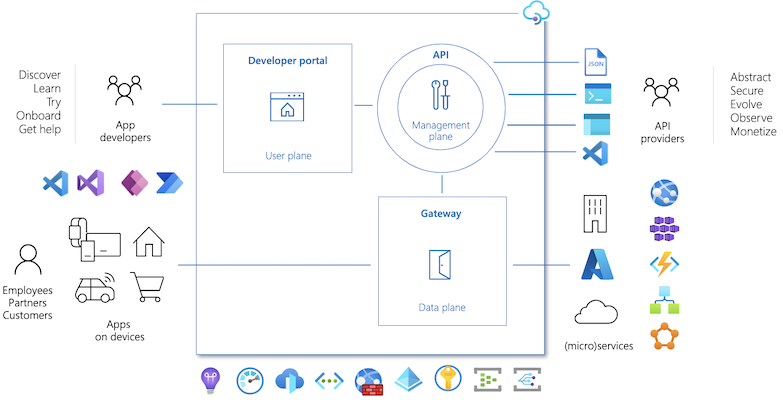
An operation is an endpoint that is consumed by the client for HTTP requests, it could be Post, Get, Delete, Head, etc.

**Apis**

When you group together the operations, then they became an API like Coffee API has multiple operations to Add, Delete Get Coffee.

**Product**

The collection of APIs is called a Product.



*You can create only****20 Consumption Plan****API Management services in an Azure subscription. Each Consumption tier service can manage up to 50 APIs.*

**Azure API Management is a fully managed service that helps developers to securely expose their APIs to external and internal customers**. It provides a set of tools and services for creating, publishing, and managing APIs, as well as for enforcing security, scaling, and monitoring API usage.

## **DEVELOPER PORTAL(reading API docs)**

The developer portal provides a portal with an auto-generated API catalog, documentation, and code samples. A developer portal is a place where API consumers come to find and learn about API details. The developer can find the API key to subscribe to APIs and provide a console for testing API endpoints.

## **API Gateway (accept API call and routes)**

API Gateway resides in a top layer of our backend services, it is a frontend that works like a proxy. All requests come to the gateway, route them to appropriate services, and return the results. Authentication, authorization, and restrictions are done in this layer. When a request is received that proper with the validations and limitations, it is forwarded to the backend. It enables us to easily access needs such as Cache, Logging, Request and response transformation, and analytical data.

## **Azure Portal (Import API Schema)**

It is an interface that allows developers to configure and manage their APIs. It allows us to separate the management of users, analytical data, policy definitions, APIs in different projects as products.

<https://medium.com/devopsturkiye/what-is-azure-api-management-why-should-we-use-it-what-are-the-benefits-part-1-11233376ac52>

# 2. Integrate API Management in an internal virtual network

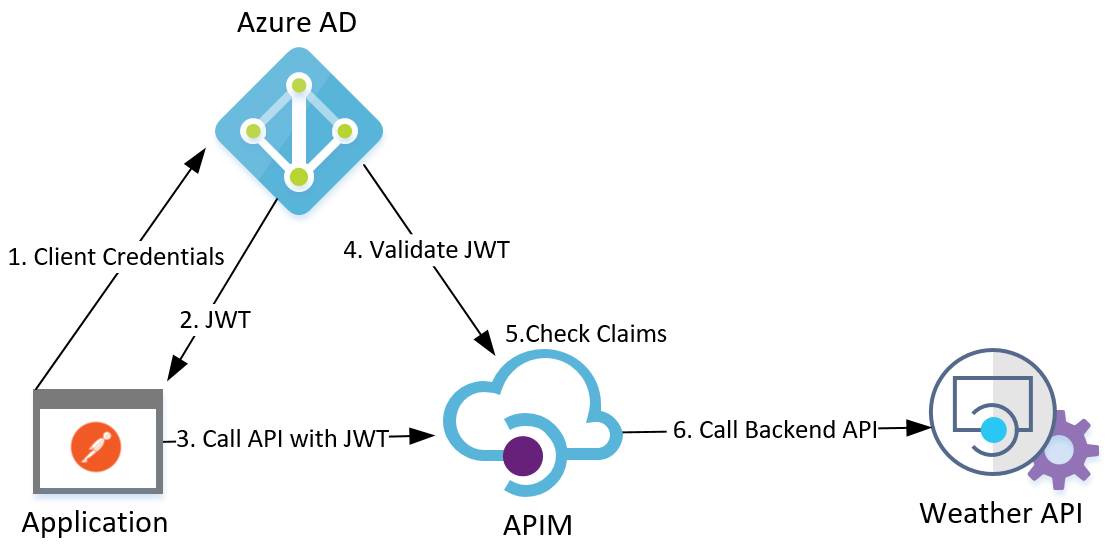
# with Application Gateway

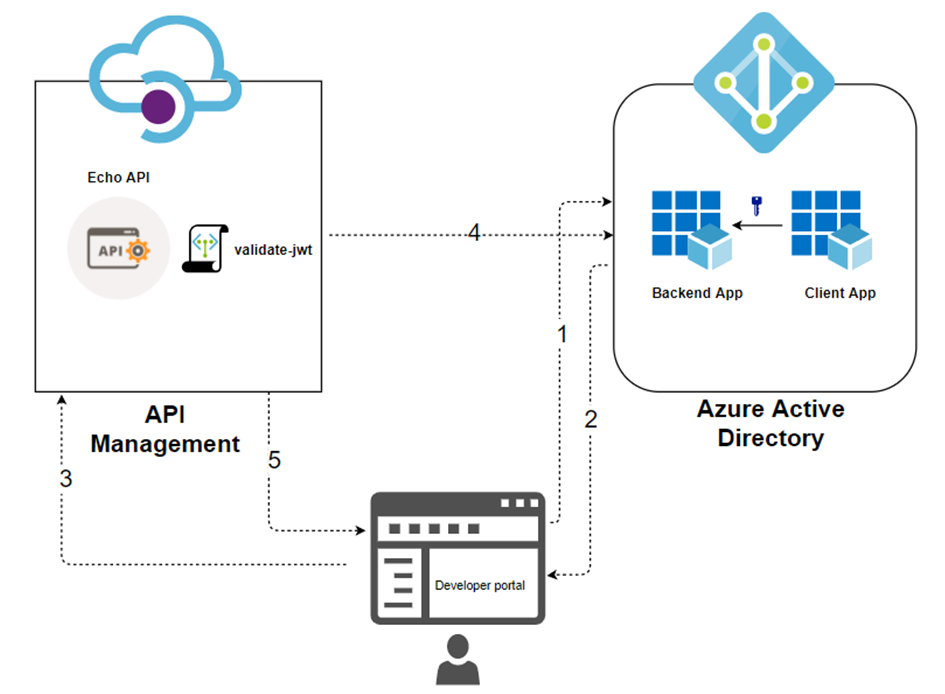
<https://learn.microsoft.com/en-us/azure/api-management/api-management-howto-integrate-internal-vnet-appgateway>

* API Management deployed in “internal” VNET mode
* Application Gateway (WAF) for exposing a subset of API’s externally

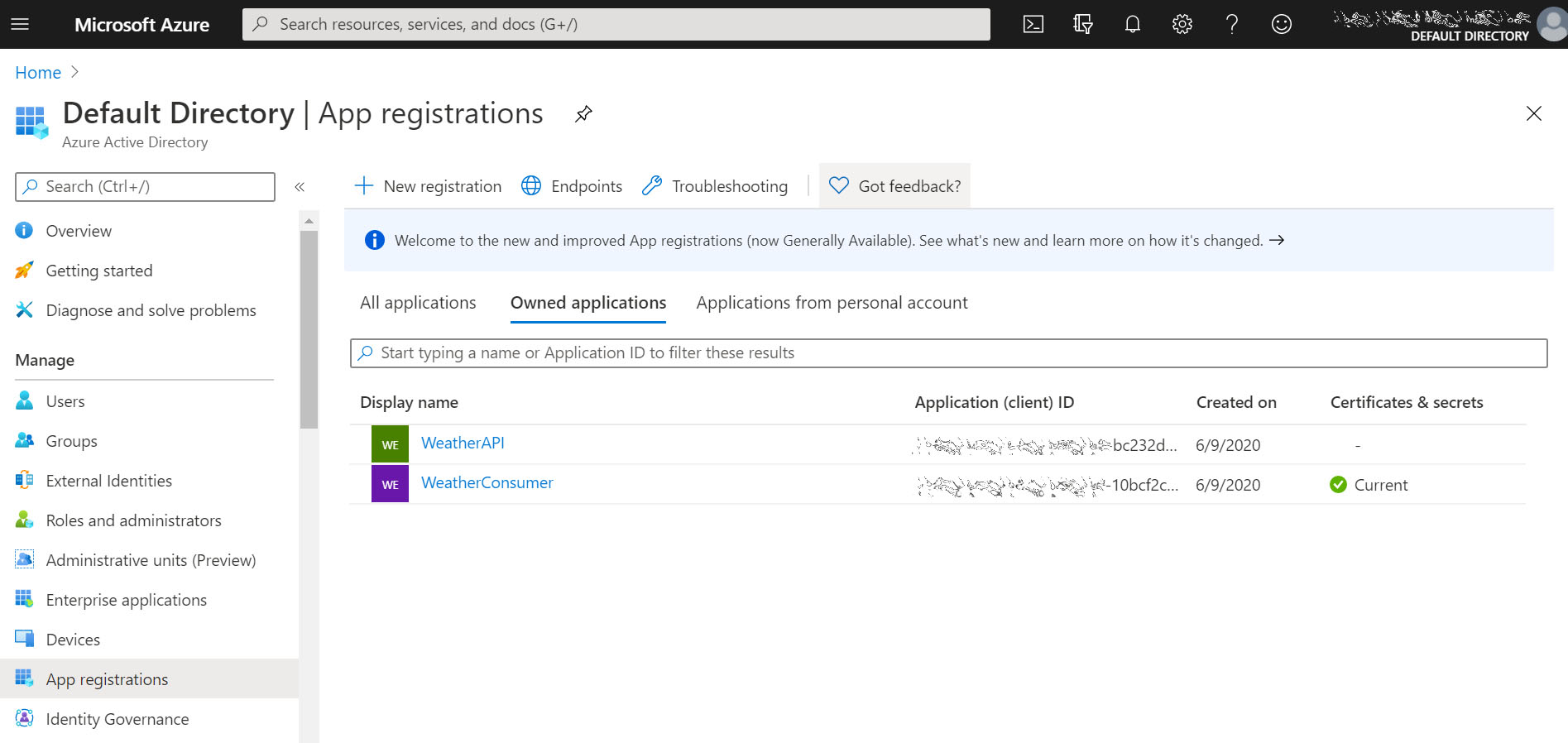
# 3. Authorization with Azure API Management

<https://tointegrationandbeyond.com/blogs/index.php/2020/06/13/authorization-with-azure-api-management/>

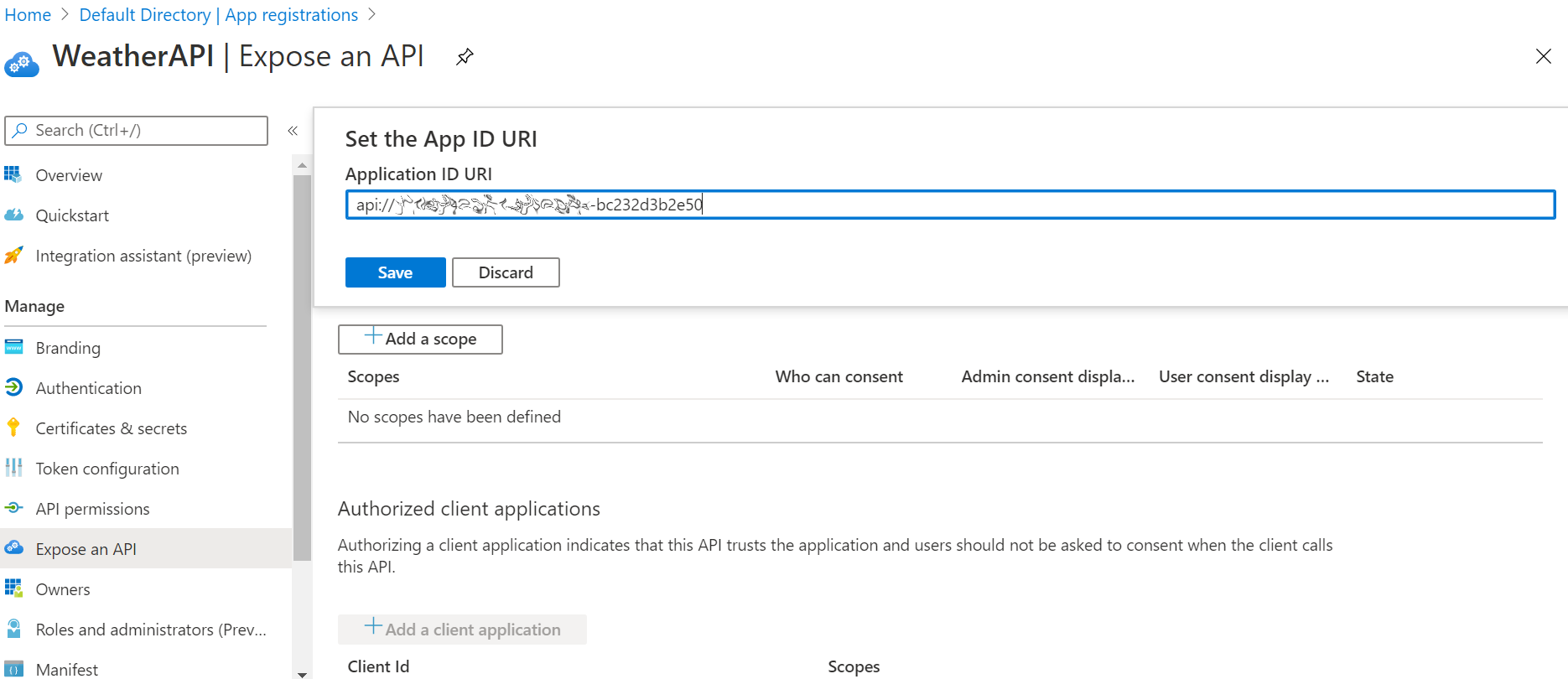




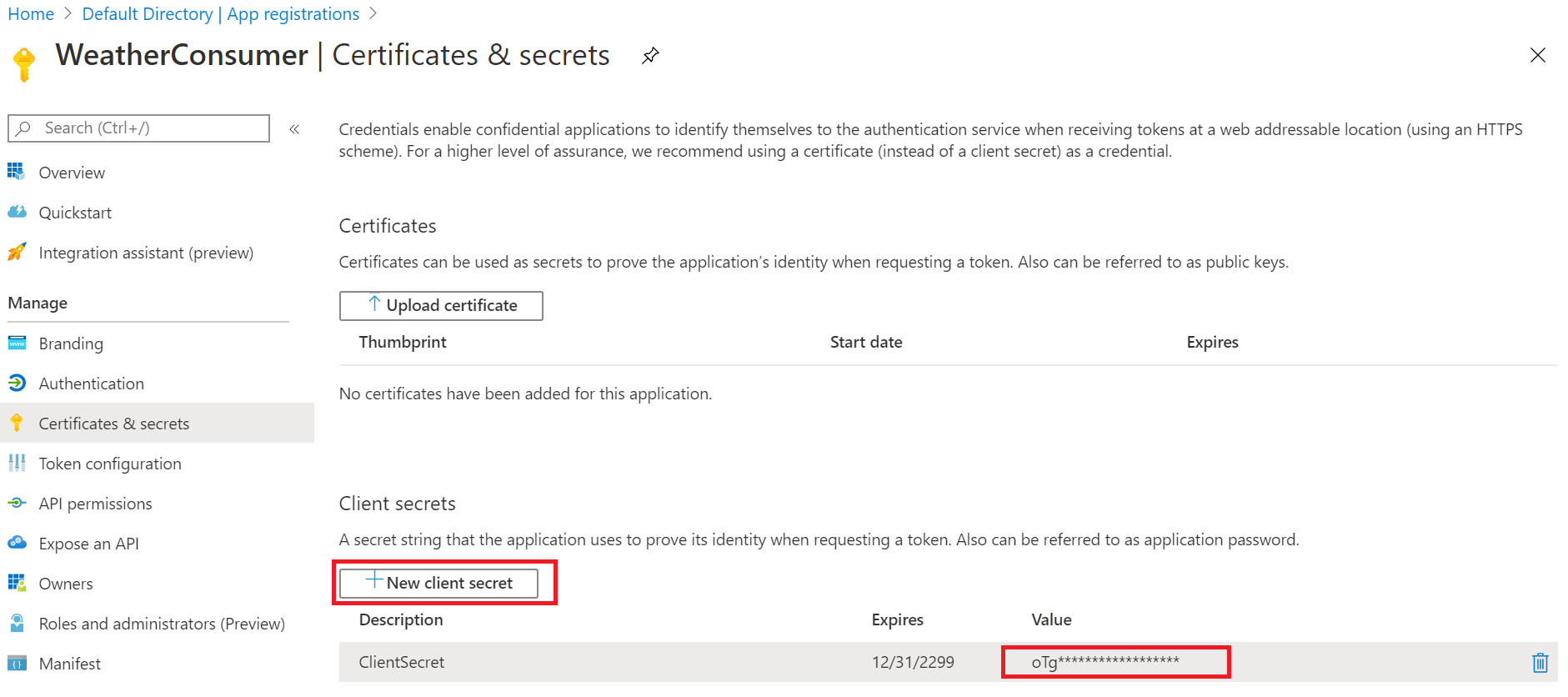
### Step 1: Register the Client and the API Resource in AAD



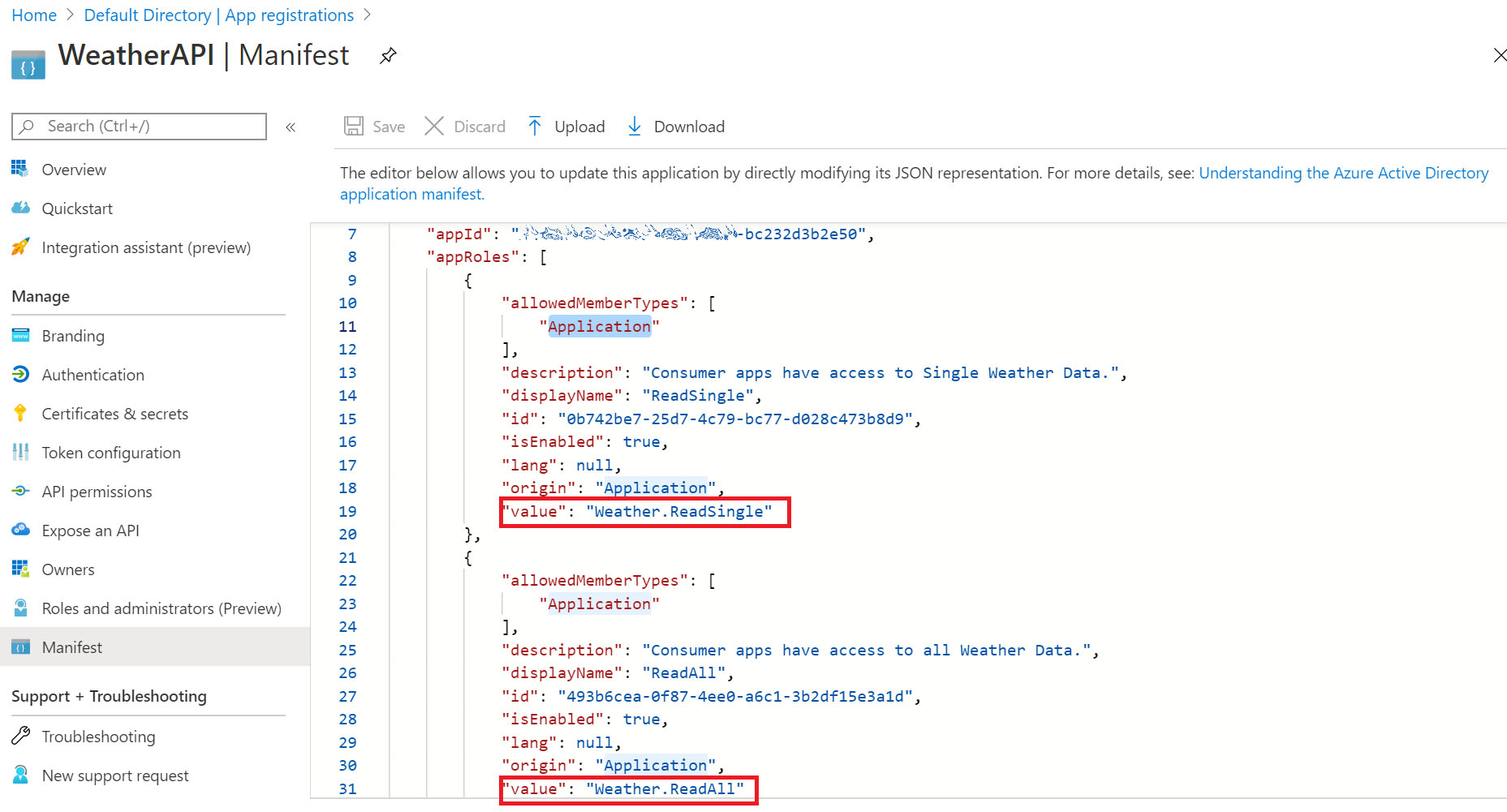
##### 2.2 Expose the API Application

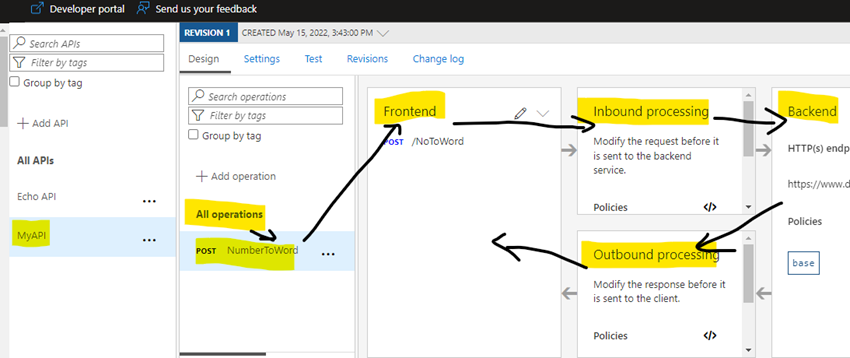


To create the client secret, in the Client AAD application > [Certificates & secrets] > [New client secret], copy the secret once it is generated as you won’t be able to view it again after you leave this page.

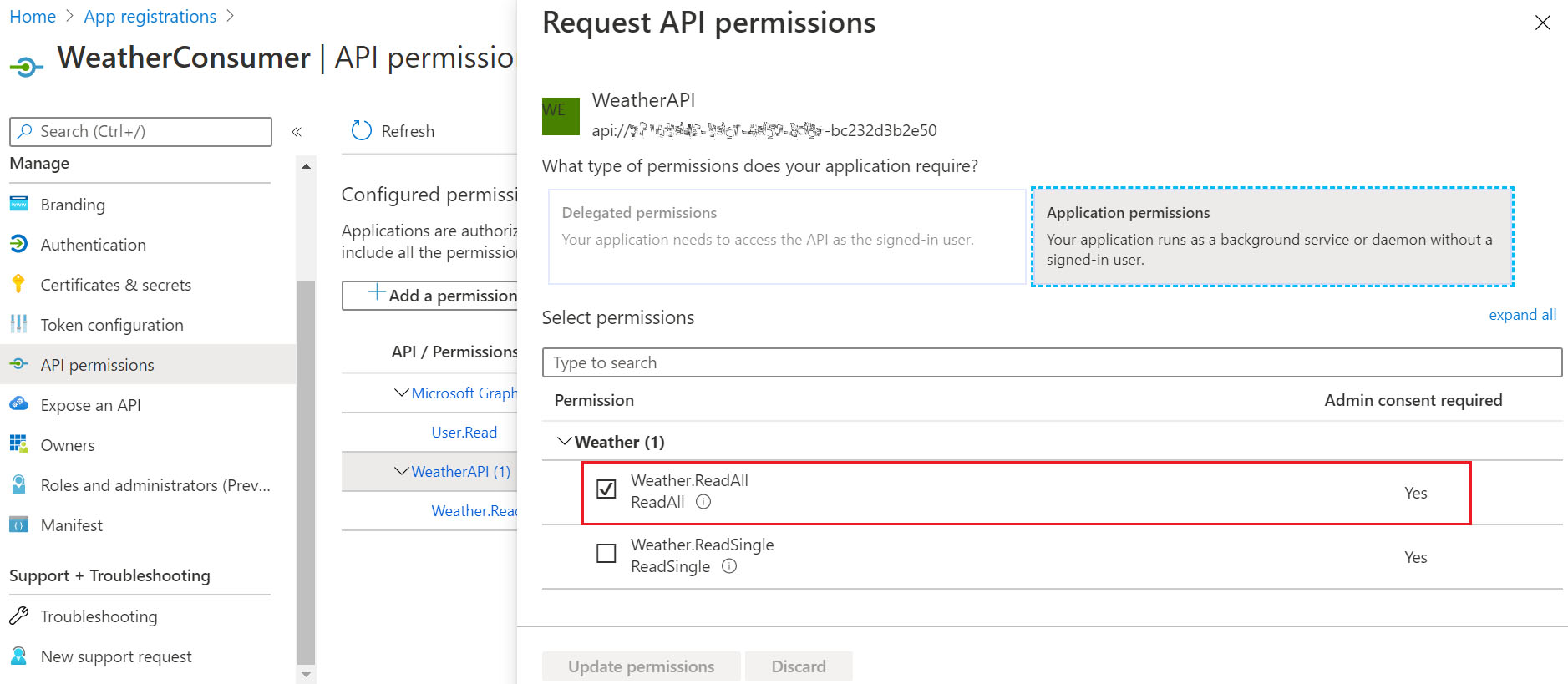


##### 2.4 Define Application Roles for the API Application

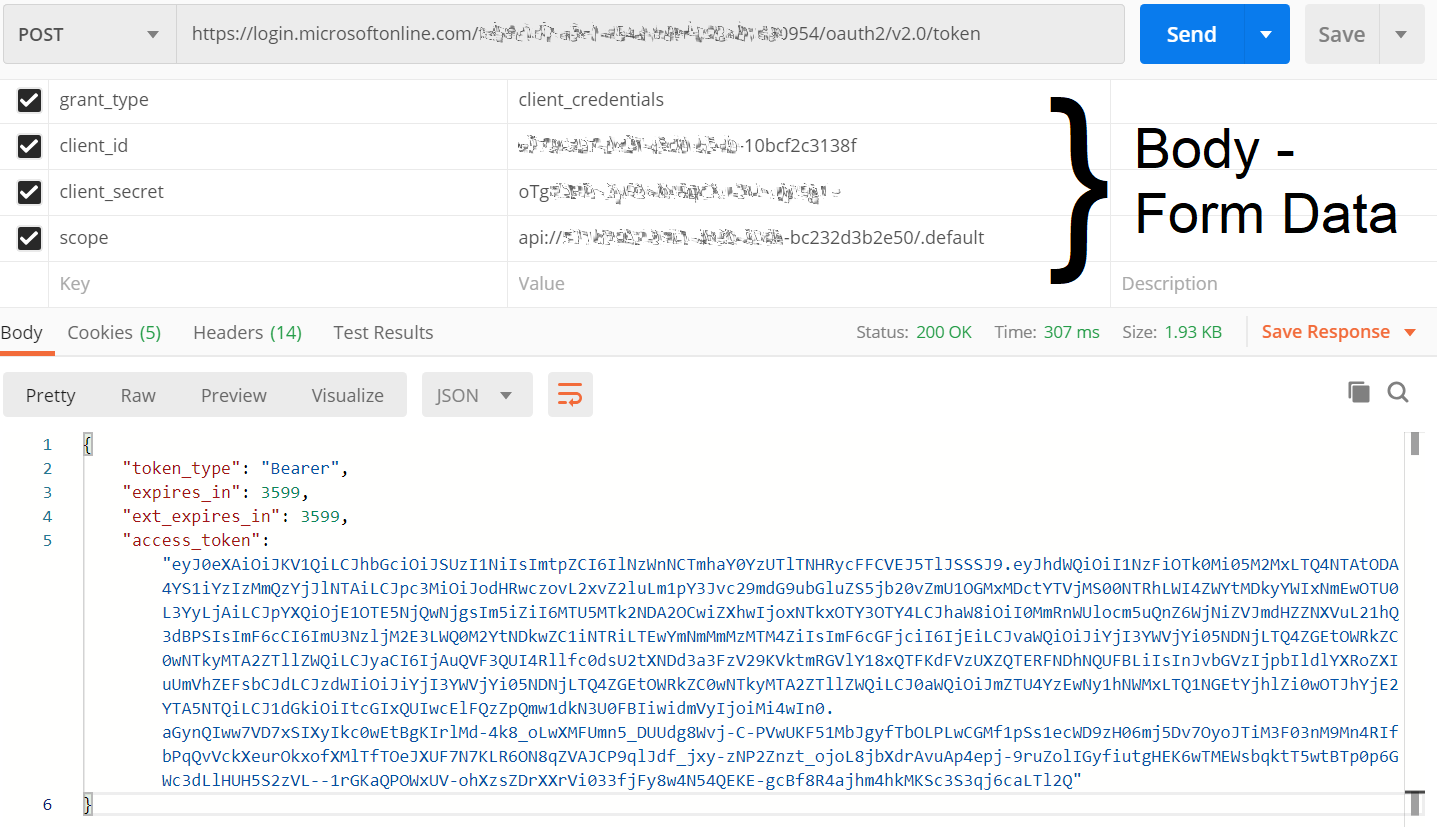


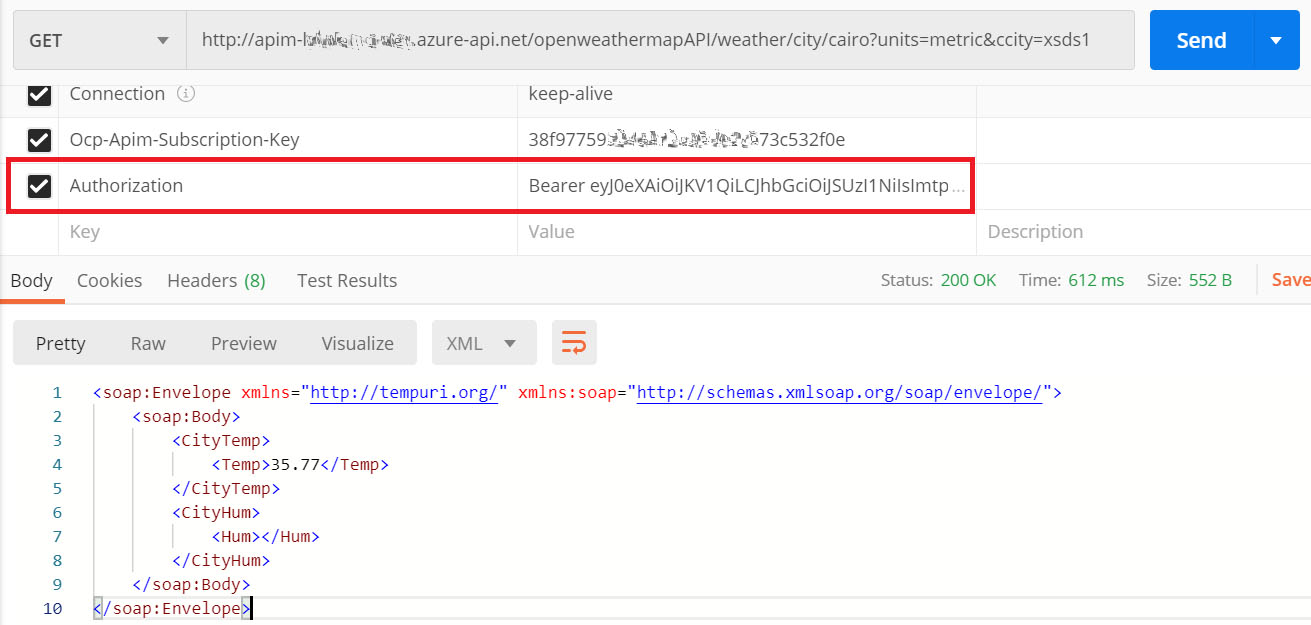


##### 2.5 Grant the Client Application the needed permissions



##### 2.6 AAD issued JWT Quick Test

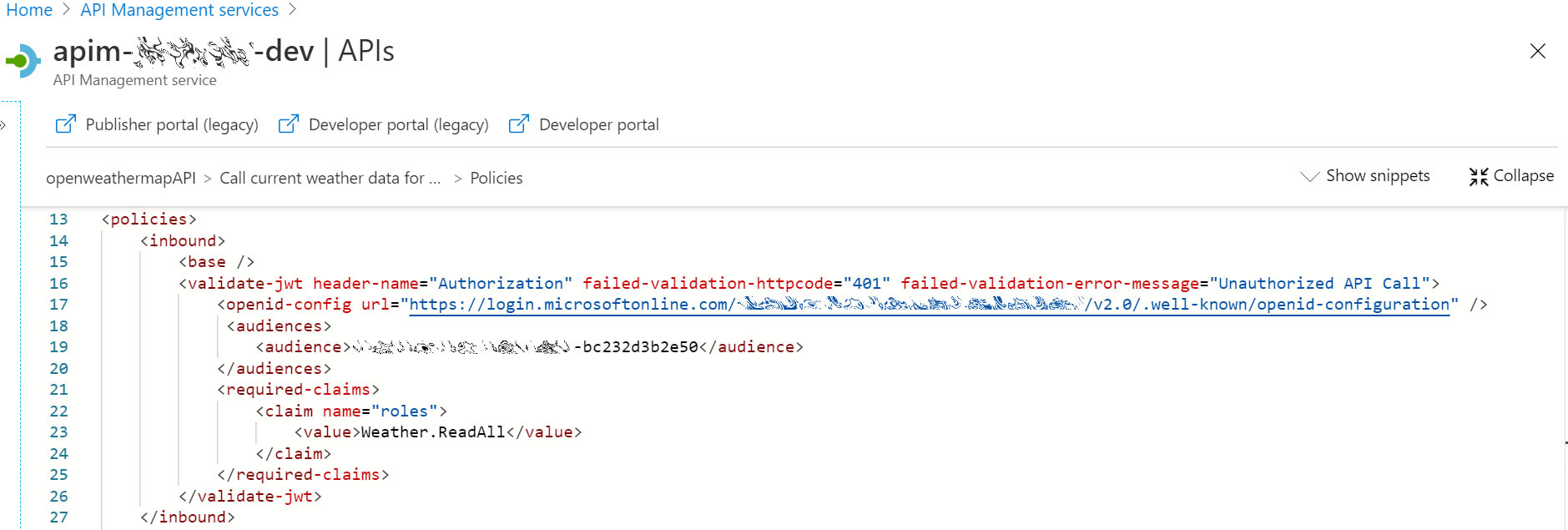




##### Apply the Operation-Level Access Restriction Policy

We start by applying the [JWT validation policy](https://docs.microsoft.com/en-us/azure/api-management/api-management-access-restriction-policies#ValidateJWT) on the operation-level, which will validate the JWT and will also inspect the role claims in the incoming request against the required role claims defined in the policy, before authorizing the API call.

Select the target API operation in APIM and apply the JWT validation policy in the inbound policy section, as shown below.



The URL attribute in the [openid-config] element sets the full URL for your AAD metadata endpoint, this endpoint provides a JSON document containing metadata information like AAD endpoint URLs, supported features, signing keys and issuer information, among other information.

This endpoint URL is available in Azure Portal in AAD > [App registrations] > Endpoints, and it will be in the following format: https://login.microsoftonline.com /{tenantID}/v2.0/.well-known/openid-configuration

In the [required-claims] element, I set the exact role claim to be checked against the incoming JWT payload [roles] section, and this is what will actually enforce the role-based access control over the selected API operation. It is worth mentioning that [claim] element has an optional attribute [match] where it can validate (all) claims or (any) matched claim.

<**validate-jwt** header-name="Authorization" failed-validation-httpcode="401" failed-validation-error-message="Unauthorized API Call">

<**openid-config** url="https://login.microsoftonline.com/{TenantID}/v2.0/.well-known/openid-configuration" />

<**audiences**>

<**audience**>{WeatherAPIApplicationID}</**audience**>

</**audiences**>

<**required-claims**>

<**claim** name="roles">

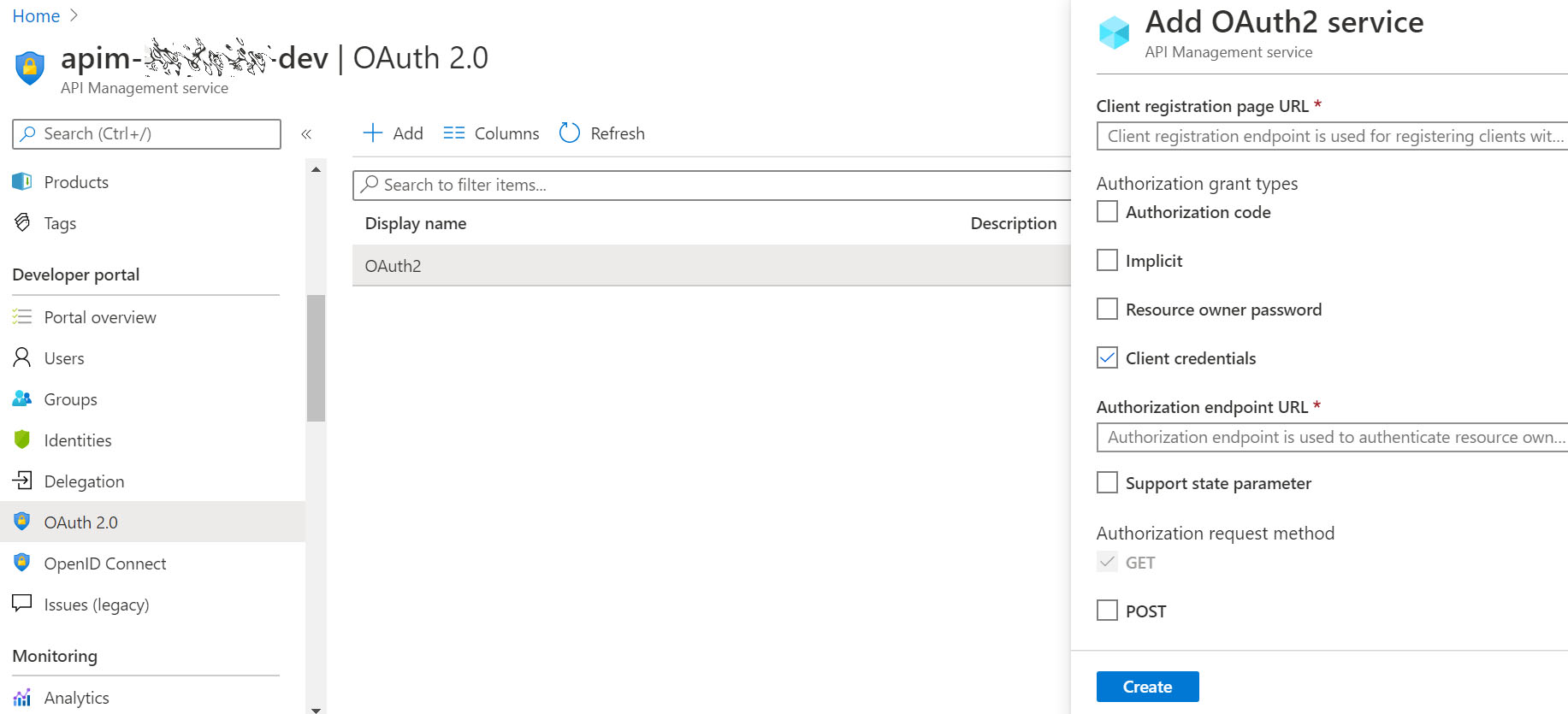
<**value**>Weather.ReadAll</**value**>

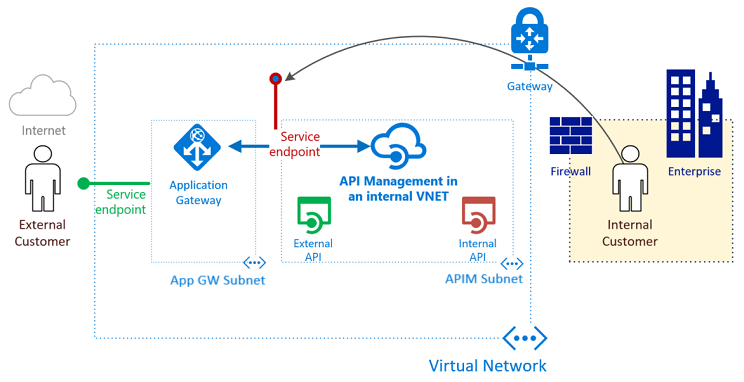
</**claim**>

</**required-claims**>

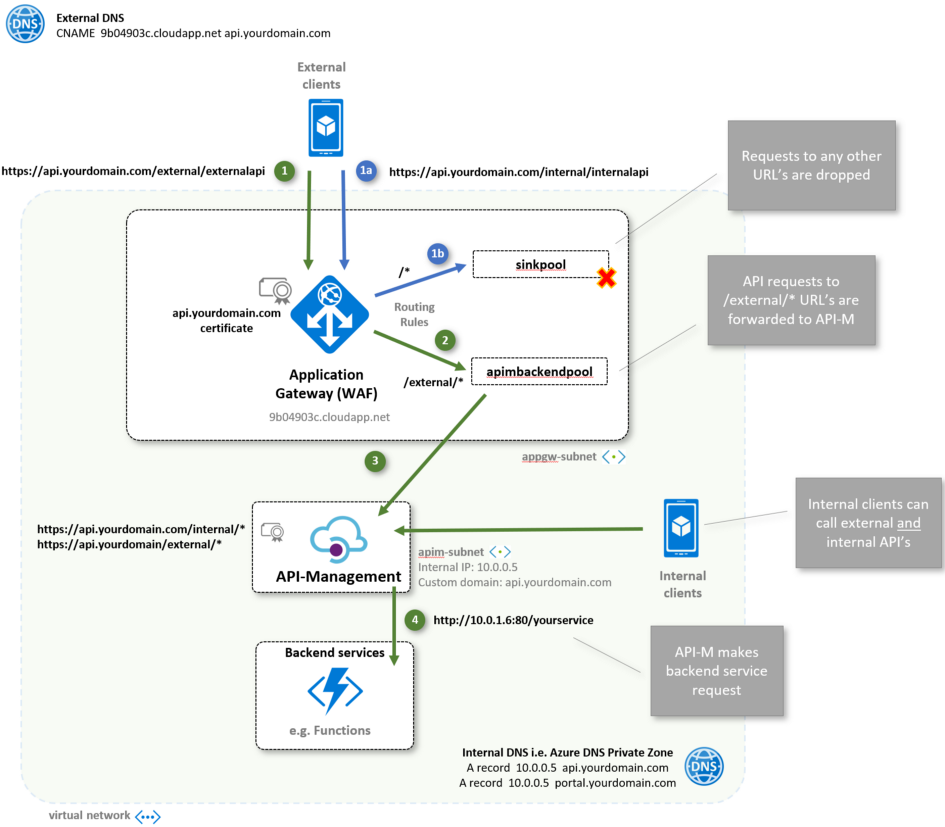
</**validate-jwt**>

Next, in APIM instance > [Developer Portal > [OAuth 2.0], add a new service, hopefully with a more relevant name than the one I chose here.





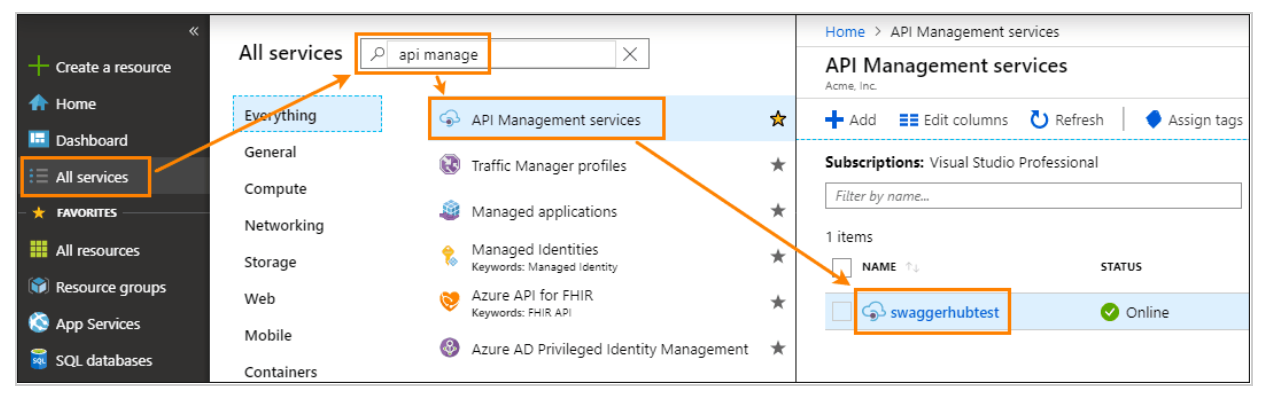
* Within API-M, APIs are created with separate base URL’s, i.e. **/external** and **/internal**
* Within Application Gateway, a **path-based routing rule** is created that redirects any API requests that contain **/external** to the API-M back-end
* The same routing rule drops requests to any other API requests, including **/internal**



Azure API Management Integration

<https://learn.microsoft.com/en-us/azure/api-management/import-and-publish>

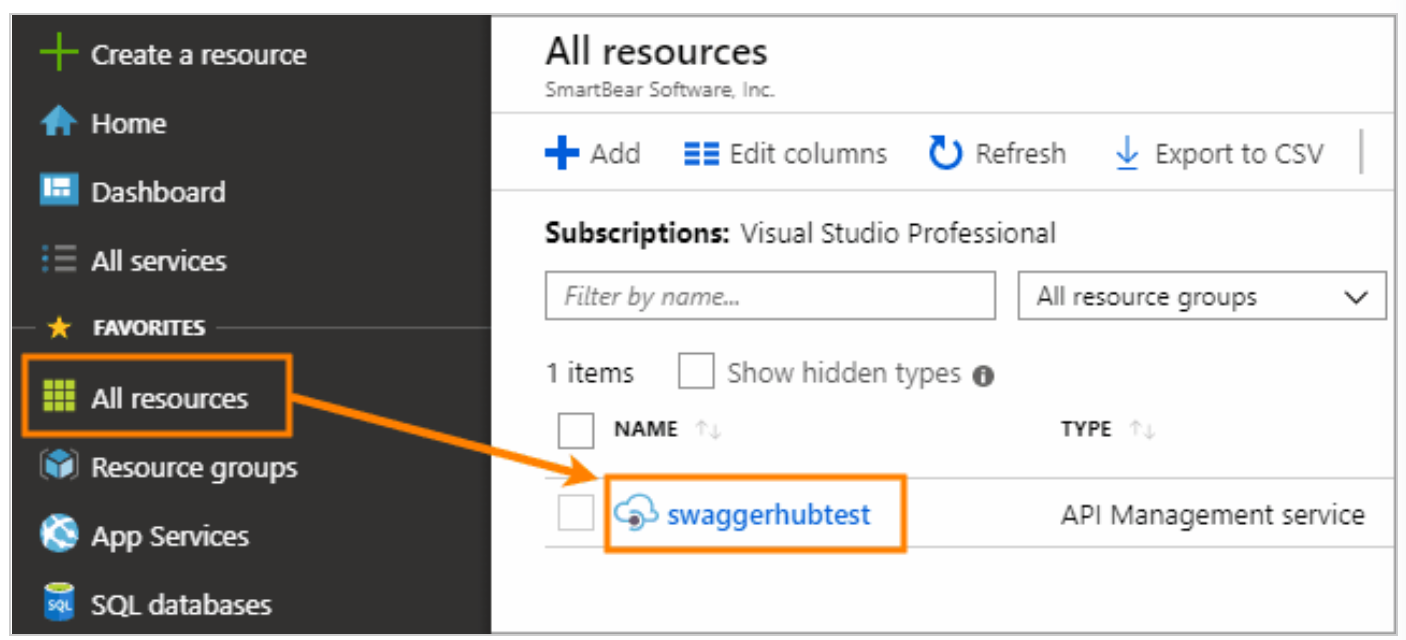
<https://support.smartbear.com/swaggerhub/docs/integrations/azure-api-management.html>



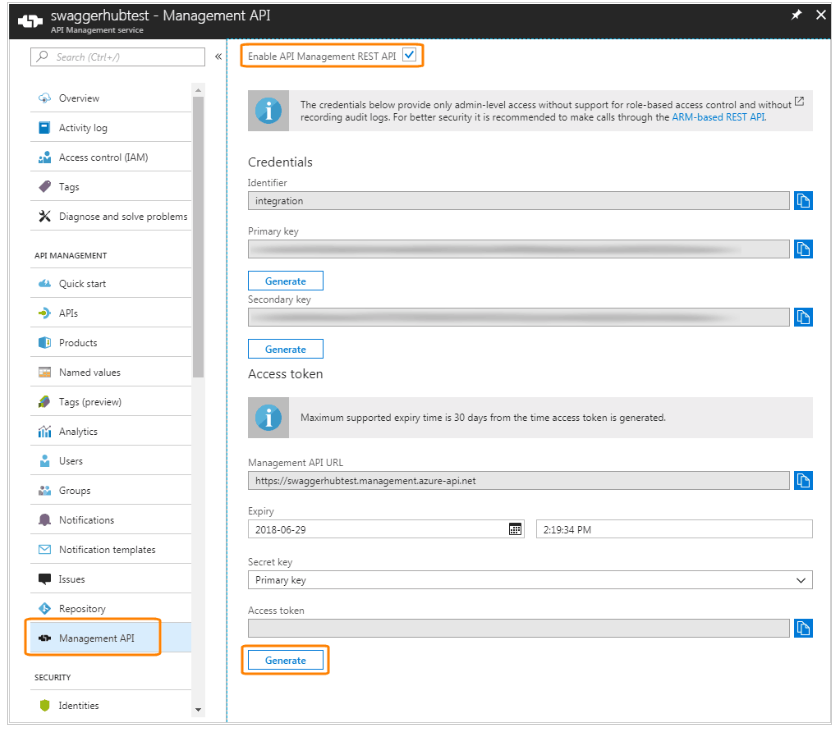
#### **Generate an Azure access token**

To generate an access token:

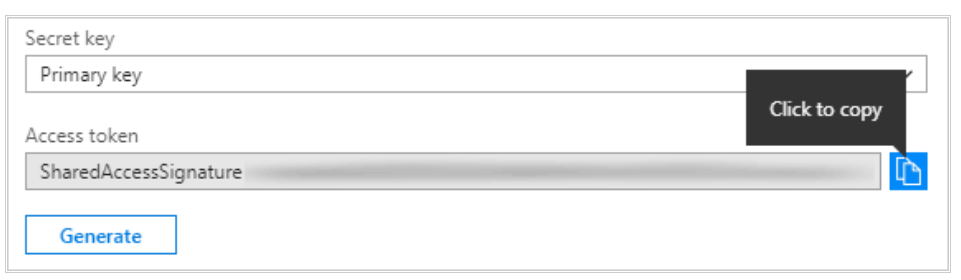
1. Open the Azure portal, [https://portal.azure.com](https://portal.azure.com/).
2. On the left, select **All resources**.



1. On the left, select **Management API**.
2. Select the **Enable API Management REST API** check box.
3. At the bottom, under **Access token**, click **Generat**



Copy the generated token (the entire string), because you will not be able to see it later.

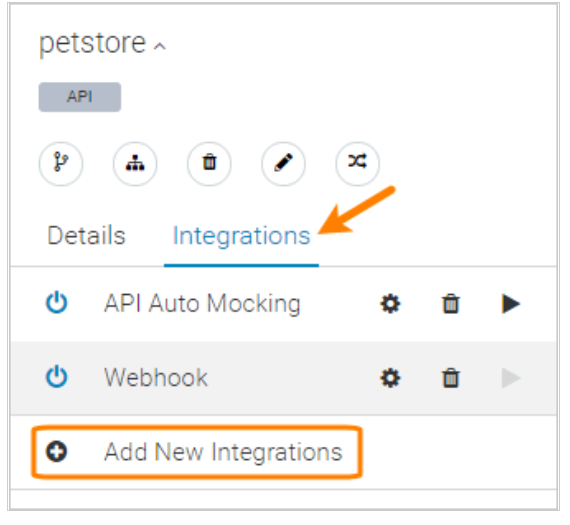


### **Configure the integration**

1. Open your API in the [SwaggerHub editor](https://support.smartbear.com/swaggerhub/docs/ui/editor.html).
2. If the API has several versions, select the version you want to push to Azure API Management.

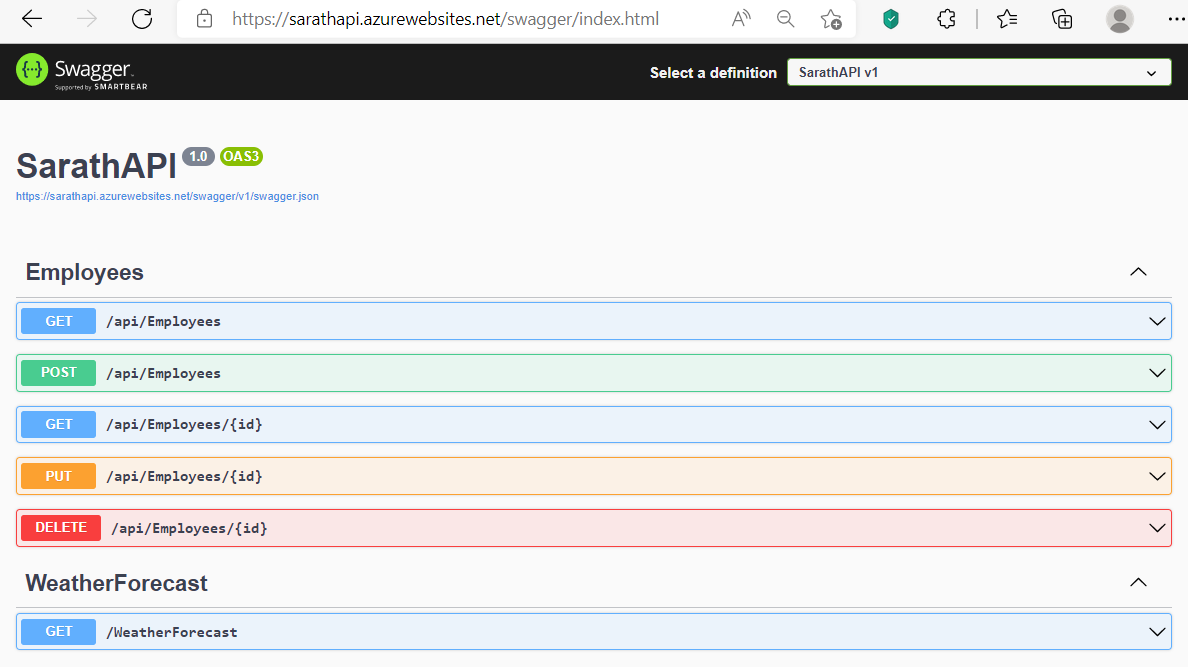


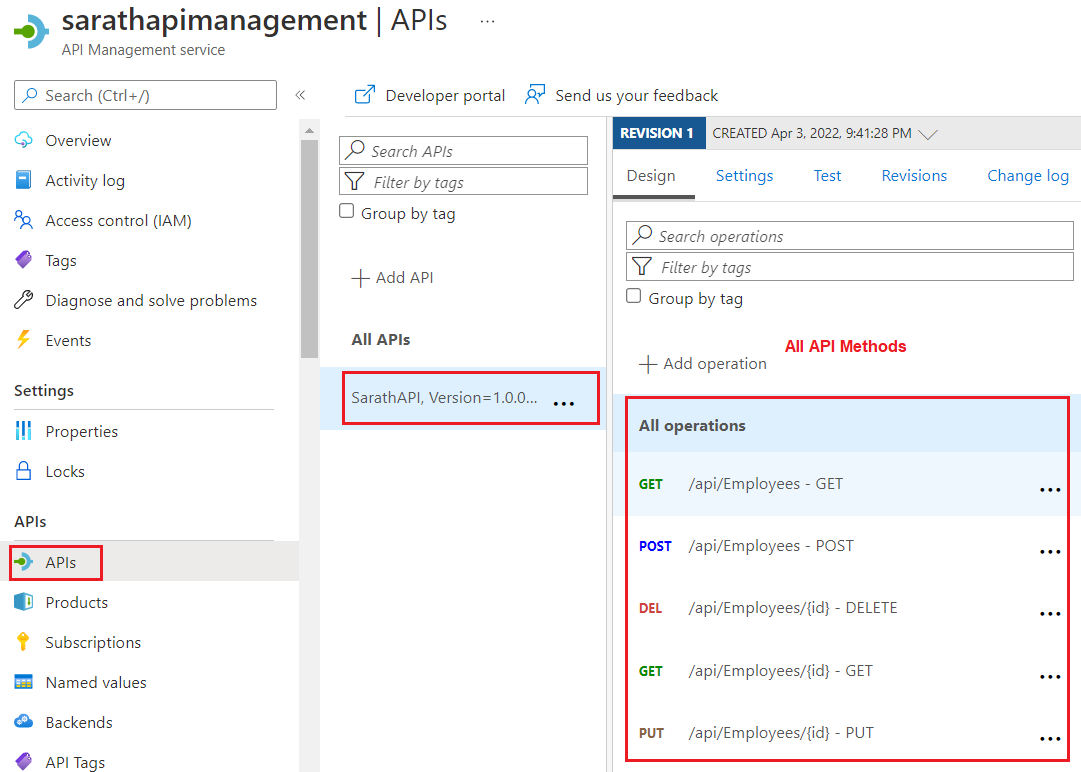
Click the Click API name, switch to the **Integrations** tab, and click **Add New Integrations**:



**Management API Access Token**

Click **Sign in with Microsoft**. You will be prompted to log into your Azure account and authorize the connection with SwaggerHub.



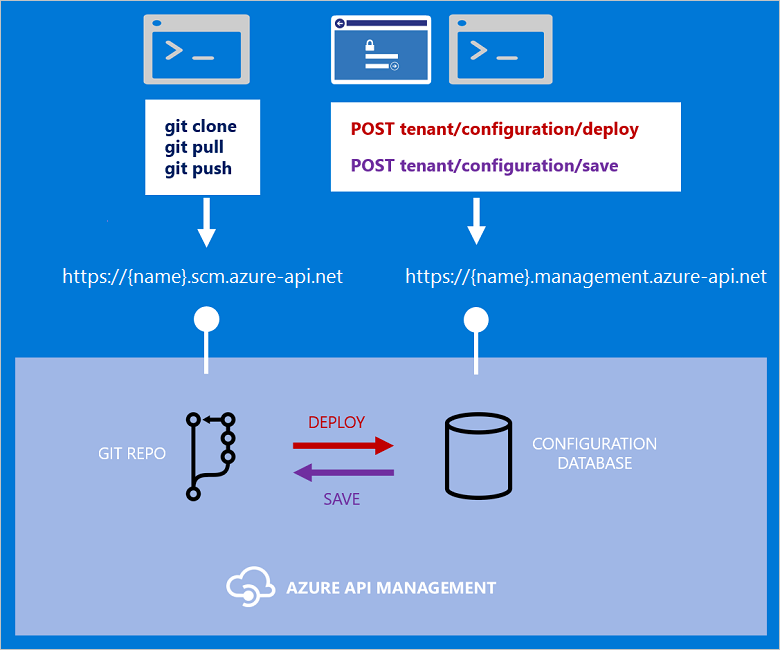


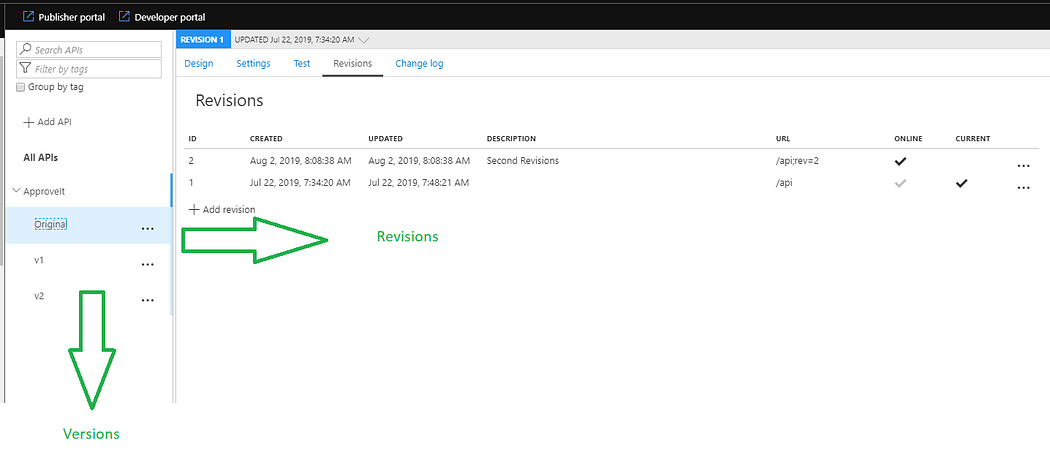
Example:

<https://www.c-sharpcorner.com/article/easily-understand-azure-api-management/>

Azure API Management, versioning

<https://thegreenerman.medium.com/azure-api-management-versioning-b062a5f5bb07>





**Add revision**

<https://learn.microsoft.com/en-us/azure/api-management/api-management-get-started-revise-api?tabs=azure-portal>

