**Azure APIM**

**Azure API Management** offers a scalable, multi-cloud API management platform for securing, publishing, and analyzing APIs.

Azure API Management (APIM) is a robust, cloud-based service provided by Microsoft that helps developers create, secure, monitor, and distribute APIs (Application Programming Interfaces) in a scalable and reliable manner.

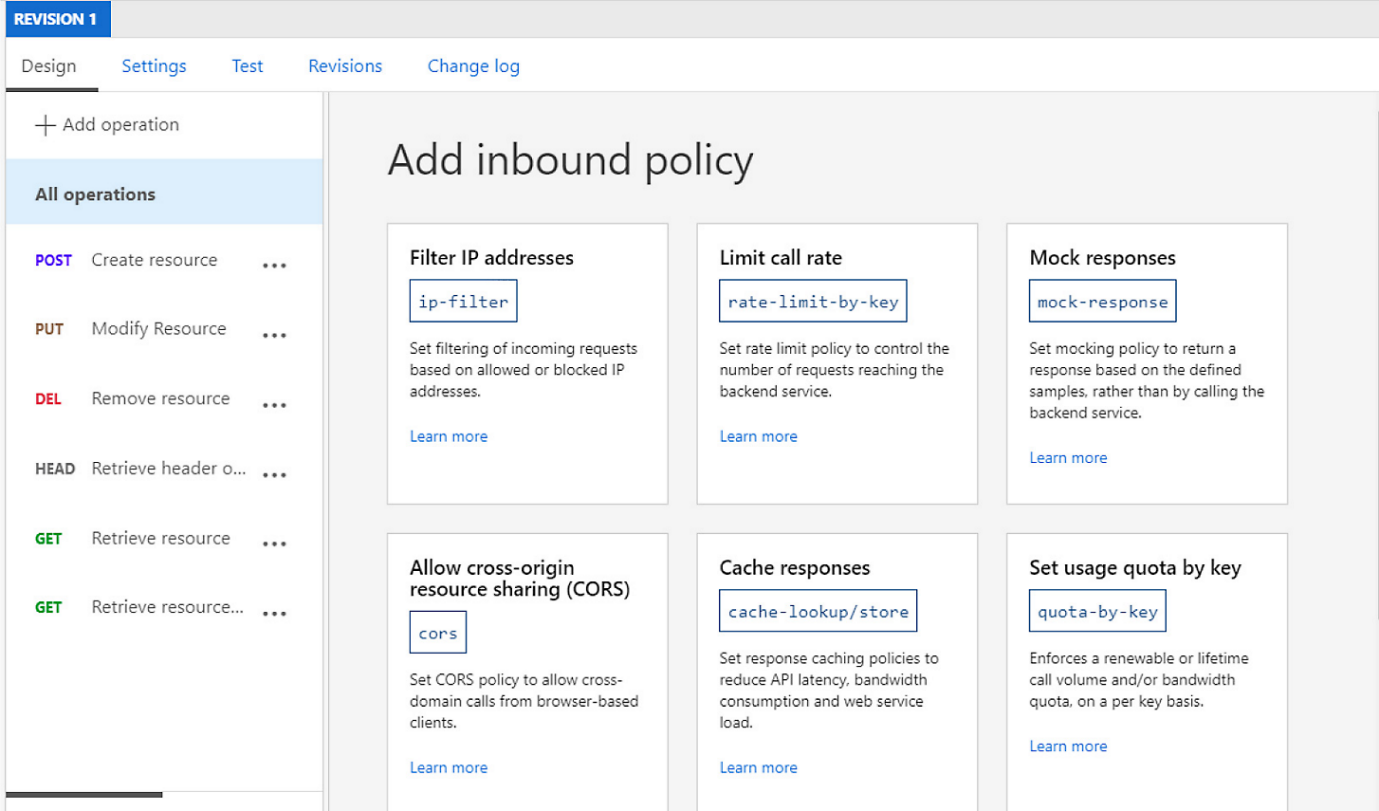
Microservices are perfect for building APIs. With Azure Kubernetes Service (AKS), you can quickly deploy and operate a microservices-based architecture in the cloud. You can then leverage Azure API Management (API Management) to publish your microservices as APIs for internal and external consumption.

Manage APIs across clouds and on-premises

Deploy API gateways side-by-side with the APIs hosted in Azure, other clouds, and on-premises, optimizing API traffic flow. Meet security and compliance requirements while enjoying a unified management experience and full observability across all internal and external APIs.

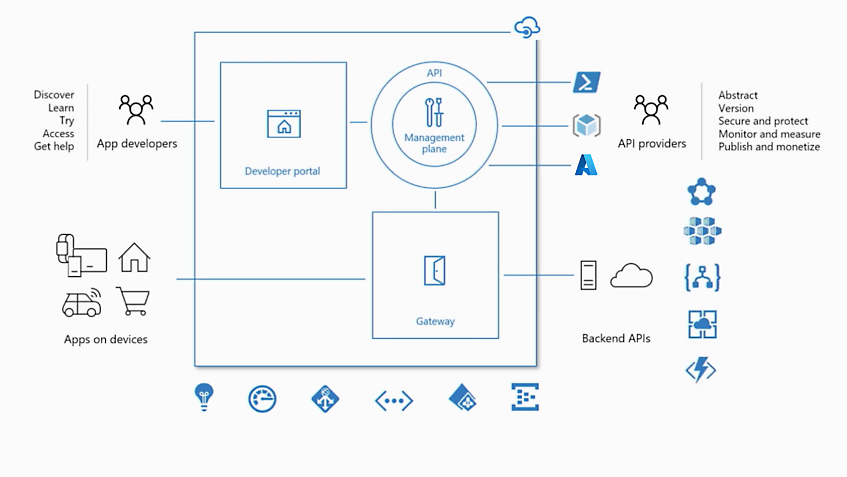
2. Help protect resources

Control how data and services are exposed to employees, partners, and customers by applying authentication, authorization, and usage limits.



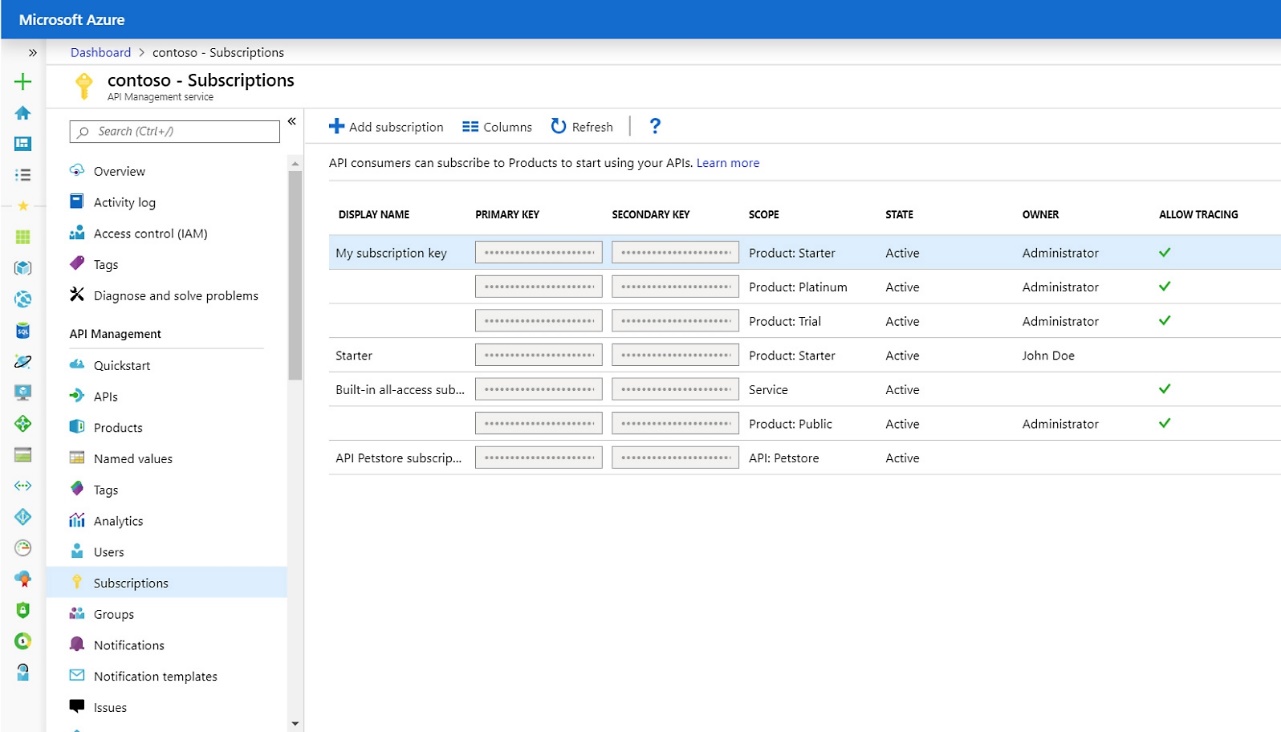
Accelerate your business

Build apps faster and deliver immediate value to your customers through API-first approaches. Decouple front- and back-end teams through API mocking, API revisions and versioning, and automated API documentation



Help secure your APIs

Keep all your APIs behind a single static IP or domain, and help protect them with keys, tokens, and IP filtering.



Cost:



|  |  |
| --- | --- |
| Azure APIM | create, manage, secure, and analyze APIs at scale.  completely managed solution called Azure API Management gives users the ability to publish, secure, adapt, maintain, and track APIs. It enables businesses to use their current investments, safeguard their assets, and securely expose their backend services and data to internal and external customers.  What is the benefit of Ocelot API gateway?  Ocelot API Gateway With .NET. Firstly we should know what is ...  These are some benefits of using API Gateway in project :  Routing and Aggregation.  Protocol Translation.  Authentication and Authorization.  Traffic Management and Load Balancing.  Caching and Performance Optimization.  Monitoring and Analytics.  Design APIs for Scalability and Performance. ...  Use Versioning and Revisions. ...  Leverage Policies. ...  Monitor and Diagnose. ...  Secure Your APIs. ...  Use Developer Portal. ...  Optimize for Cost. ...  Plan for High Availability and Disaster Recovery.  API Management components  API gateway,  management plane,  developer portal. |
|  |  |
| Best Practices for Azure API Management | 1) Versioning and revision  2) Caching  3) Policy Define  4) Traffic management and Load balancer  5) error handling,Monitor and analytics (Azure Monitor tool, Azure application insight)  6) Routing & Aggregation  7) Authentication & Authorization  8) Scalability & Performance  9) Secure APIs  10) High Available  11) Disaster recovery  12) Publish, Export and Monitor API  13) Throtlling Ratelimit, Quata  14) Pricing tier( developer, Consumption, premium)  15) CORS  Azure API Management is a powerful tool that, when used correctly, can significantly enhance the management, distribution, and security of your APIs. By following these best practices, you can ensure that your operations are as efficient, effective, and secure as possible.  <https://topuzas.medium.com/best-practices-for-azure-api-management-1ae3968722db>  1. Design APIs for Scalability and Performance  2. Use Versioning and Revisions  3. Monitor and Diagnose - zure Monitor, API Management analytics, and Application Insights to track your API’s performance, identify issues, and debug them.  4. Secure Your APIs - OAuth 2.0, OpenID Connect, and Azure Active Directory for user authentication and authorization  Use Developer Portal: Developer portal is a self-service portal for API documentation and guides, where developers can learn about your APIs, obtain keys, and track their usage. It’s a good practice to provide comprehensive API documentation and keep it up to date.  7. Optimize for Cost  Azure API Management comes with different pricing tiers — Consumption, Basic, Standard, and Premium. Evaluate your requirements carefully to select the most cost-effective tier that meets your needs. Remember, features like Virtual Network (VNet) are only available on higher tiers, so choose wisely.  8. Plan for High Availability and Disaster Recovery  Ensure that your APIs are available when they are needed by leveraging Azure’s capabilities for high availability and disaster recovery. This includes deploying your APIs across multiple regions and using Azure Traffic Manager to distribute traffic.  9. Leverage API Management CI/CD  Azure API Management supports integration with common CI/CD (Continuous Integration/Continuous Deployment) tools. Use this feature to automate your deployment processes and ensure consistent and reliable API updates.  10. Use Named Values for Secrets  Never hard-code secrets like connection strings or passwords in your policy expressions. Instead, use Named Values, which are securely stored and can be used in policies. Policy scopes |
|  |  |

In Azure API Management, **rate limiting and throttling** are used to control the amount of incoming traffic to your APIs.

You can **set quotas, rate limits, and burst limits to manage how many requests can be made to your APIs within a specific time frame**.

These limits can be applied at various levels, including API level, operation level, or even per subscription.

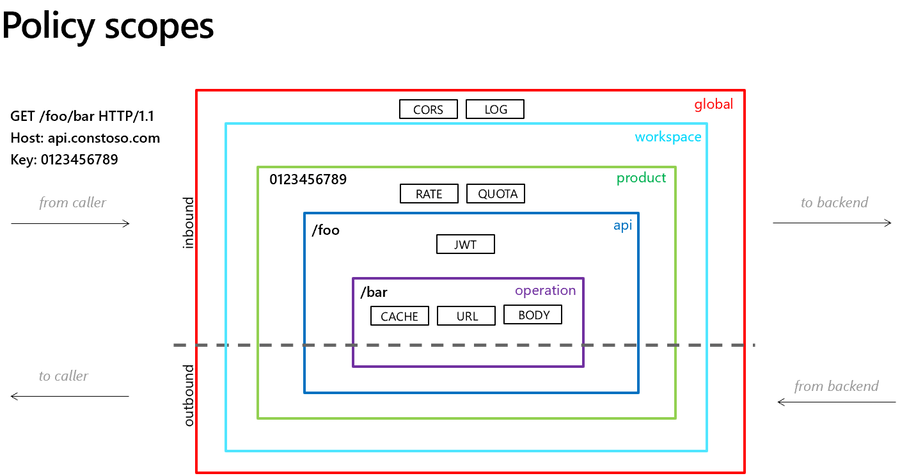
**Definition**:

**Policies**: You can apply policies at the product level, API level, or operation level. This flexibility allows you to enforce different policies, including authentication, rate limiting, and transformation, for different APIs or operations within a single subscription.

**Products**: APIM allows you to create products, which are logical groupings of APIs and their associated policies. You can include one or more APIs in a product. This allows you to control access to multiple APIs with a single subscription key.

**RATE** (to allow a maximum of 100 requests per minute (RPM) for each client),

**Quata** (want to limit a client to 10,000 requests per day)



Operation: Cache, URL, Body

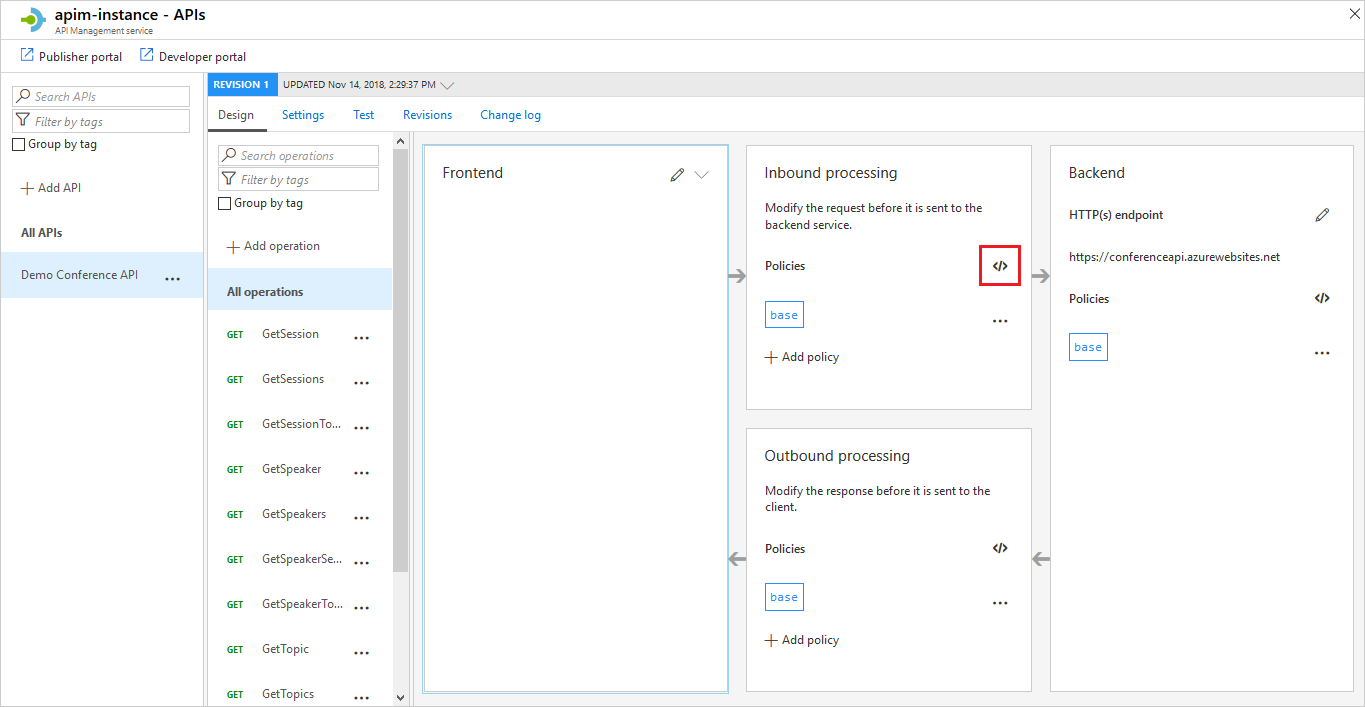
API : JWT

Product: RATE, Quata

Workspace:

Global: CORS, LOG





In the **Inbound processing** section, click the **</>** icon.

**<cache-lookup vary-by-developer="false" vary-by-developer-groups="false">**

<vary-by-header>Accept</vary-by-header>

<vary-by-header>Accept-Charset</vary-by-header>

<vary-by-header>Authorization</vary-by-header>

</cache-lookup>

In the **outbound** element, add the following policy:

<cache-store duration="20" />

In this example, we set the **Cache-Control** header to enable caching for one hour (**max-age=3600**) for responses from the backend. You can adjust the cache duration according to your requirements.

<policies>

<inbound>

<base />

</inbound>

<backend>

<!-- This sets the cache control header to enable caching for 1 hour -->

**<set-header name="Cache-Control" exists-action="override">**

<value>public, max-age=3600</value>

</set-header>

</backend>

<outbound>

<!-- Add other policies here -->

</outbound>

</policies>

<policies>

<inbound>

<base />

<!-- Check if the URL path starts with /api/{version}/ -->

<choose>

<when condition="@(context.Request.Url.Path.StartsWith("/api/"))">

<choose>

<!-- Check if the version query parameter is present -->

<when condition="@(context.Request.Url.Query.GetValueOrDefault("version", null) != null)">

<!-- Continue processing if validation passes -->

<return-response />

</when>

<!-- Reject request if version parameter is missing -->

<otherwise>

<return-response>

<set-status code="400" reason="Bad Request" />

<set-body>

{

"error": "Missing version parameter in the URL."

}

</set-body>

</return-response>

</otherwise>

</choose>

</when>

<!-- Reject request if URL path doesn't match the expected structure -->

<otherwise>

<return-response>

<set-status code="400" reason="Bad Request" />

<set-body>

{

"error": "Invalid URL path. It should start with /api/{version}/."

}

</set-body>

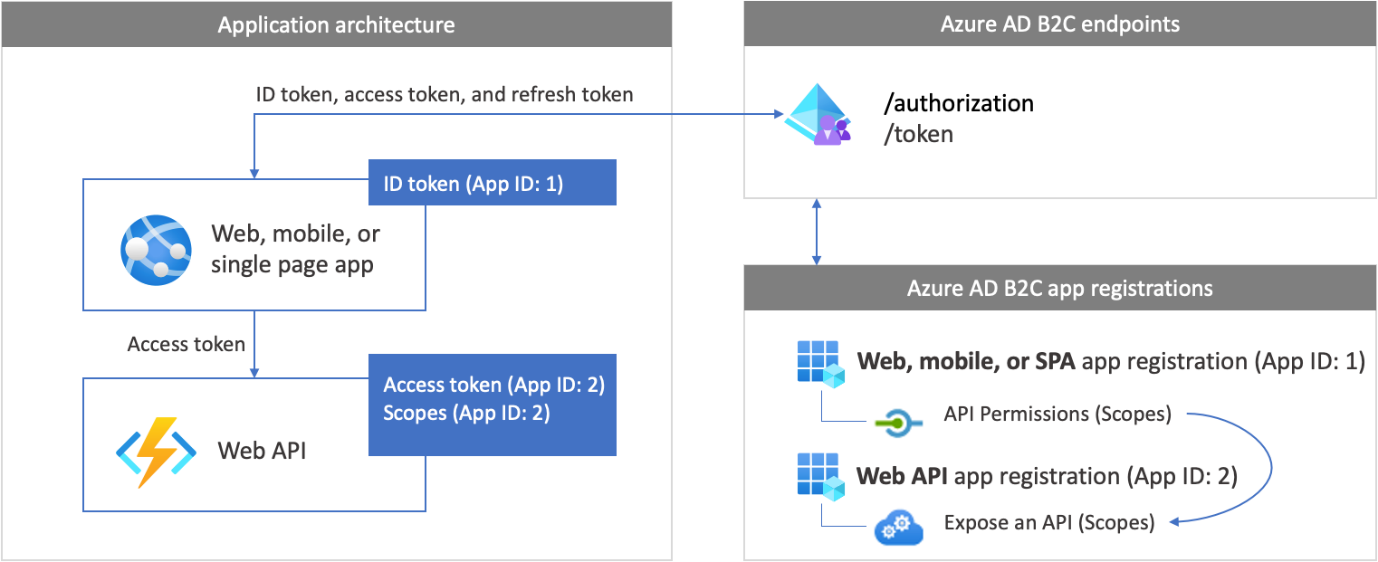
</return-response>

</otherwise>

</choose>

</inbound>

</policies>



1. **Long-Running Workflow**:

Use Case: You need to implement a workflow that involves multiple steps, such as data validation, approval, and notification, which can take a long time to complete.

Solution: You can use Durable Functions to define an orchestrated function that manages the workflow. Each step of the workflow can be implemented as a separate activity function. Durable Functions provides checkpoints and state management to ensure that the workflow progresses correctly, even if it spans multiple days or weeks.

1. **E-commerce Order Processing**:

Use Case: In an e-commerce application, you receive orders that require processing, including payment validation, inventory check, order fulfillment, and shipping confirmation.

Solution: Implement a Durable Function orchestration that manages the order processing workflow. Each step, such as payment validation or order fulfillment, can be implemented as an activity function. The orchestration ensures that these steps are executed in the correct sequence, and it can handle retries and compensation logic if errors occur.

1. **Fan-Out/Fan-In Processing**:

Use Case: You have a batch processing task that involves processing a large number of items in parallel and then aggregating the results.

Solution: Use a Durable Function pattern called Fan-Out/Fan-In. The orchestrator function can fan out by starting multiple activity functions to process items concurrently. Each activity function processes an individual item and returns a result. The orchestrator then collects and aggregates the results from all activity functions using a Fan-In pattern.

1. **Human Interaction in Workflow**:

Use Case: You need to incorporate human interaction, such as approvals or manual data entry, into an automated workflow.

Solution: Durable Functions allows you to implement human interaction by using external events. For example, an orchestrator can start an approval process and wait for an external event (e.g., an email reply or an approval from a web form). Once the external event occurs, the orchestrator can resume and continue the workflow accordingly.

1. **IoT Device Telemetry Processing**:

Use Case: You have a fleet of IoT devices sending telemetry data, and you want to process and analyze this data in real-time.

Solution: Implement a Durable Function that listens for incoming telemetry data and processes it as it arrives. The orchestrator can use a timer trigger to aggregate and analyze the data at predefined intervals. This allows you to perform real-time analytics on IoT device data without maintaining complex state management.

1. **Chained Function Execution**:

Use Case: You need to execute a series of functions in a specific order, where the output of one function serves as input to the next.

Solution: Create an orchestrator function that chains together multiple activity functions. Each activity function performs a specific task and returns data to be passed to the next function in the chain. This pattern is useful for implementing complex data transformations or data processing pipelines.

1. **Escalation Workflow**:

Use Case: Implement an escalation workflow where a support ticket starts with basic troubleshooting and escalates to higher-level support if the issue is not resolved.

Solution: Use an orchestrator to manage the ticket resolution process. Start with basic troubleshooting as an activity function, and if the issue persists, escalate to a higher-level support activity function. The orchestrator can handle the decision-making process and ensure that the correct steps are taken.

These examples illustrate how Azure Durable Functions can be applied to solve various real-world problems involving complex workflows, long-running processes, and human interaction within distributed systems. Durable Functions provide the necessary state management and coordination capabilities to make these solutions reliable and scalable.

**1. Rate Limit Example:**

Let's say you want to limit a specific API to allow a maximum of 100 requests per minute (RPM) for each client:

<inbound>

<base />

**<rate-limit-by-key calls="100" renewal-period="60" counter-key="@(context.Request.IpAddress)" />**

</inbound>

In this policy:

* **rate-limit-by-key** is used to limit the rate.
* **calls** specifies the maximum number of requests allowed per **renewal-period**.
* **renewal-period** is the time window in seconds.
* **counter-key** uses the client's IP address to track requests.

**2. Quota Example:**

For quotas, let's say you **want to limit a client to 10,000 requests per** day:

<inbound>

<base />

**<quota-by-key calls="10000" renewal-period="86400" counter**-key="@(context.Request.Headers.GetValueOrDefault("Ocp-Apim-Subscription-Key"))" />

</inbound>

Configuring Cross-Origin Resource Sharing (**CORS**) in Azure API Management (APIM) allows you to control which domains are permitted to access your APIs. CORS is essential when you want to enable web applications hosted on different domains to make requests to your APIs. Here's an example of how to set up CORS in Azure APIM using policies:

<inbound>

<!-- Base policy for all incoming requests -->

<base />

<!-- Enable CORS for a specific origin (e.g., https://example.com) -->

<cors>

<allowed-origins>

<origin>https://example.com</origin>

</allowed-origins>

<allowed-methods>

<method>GET</method>

<method>POST</method>

</allowed-methods>

<allowed-headers>

<header>Content-Type</header>

<header>Authorization</header>

</allowed-headers>

<expose-headers>

<header>Location</header>

</expose-headers>

</cors>

</inbound>

* **<allowed-origins>** specifies the domains that are allowed to access your API. You can list multiple origins if needed.
* **<allowed-methods>** defines the HTTP methods that are allowed for cross-origin requests (e.g., GET and POST).
* **<allowed-headers>** lists the HTTP headers that can be included in the request. You should include headers such as "Content-Type" and "Authorization" if your API expects them.
* **<expose-headers>** specifies which response headers are exposed to the client. These are headers that the client can access in the response.

However, using **\*** is generally discouraged because it allows any website to access your API, potentially posing security risks.

<origin>\*</origin>

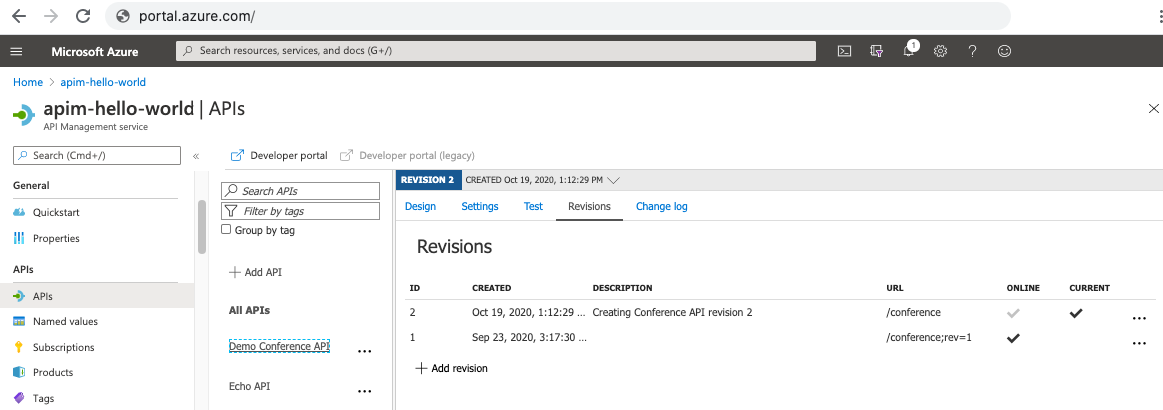
**Secure azure APIM via postman token**

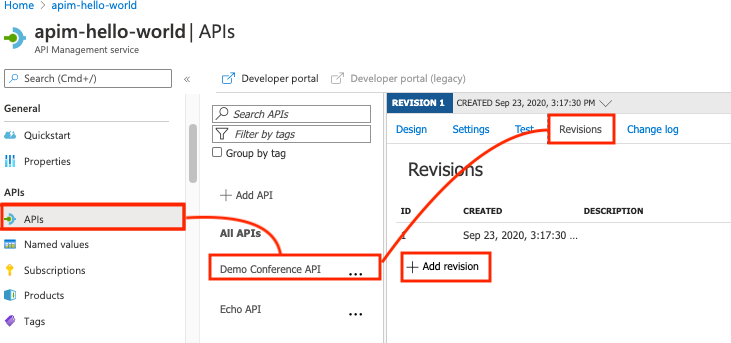
To secure Azure API Management (APIM) and make authenticated requests to your APIs using Postman, there are several authentication methods and tokens you can use. Here are some common ways to secure APIM via Postman:

1. **API Key (Subscription Key)**:
   * APIM allows you to secure your APIs using API keys (also known as subscription keys). You can add the API key as a query parameter or in the request headers.
   * In Postman, you can include the API key as a query parameter or in the request headers, such as "Ocp-Apim-Subscription-Key."
2. **OAuth 2.0**:
   * If your APIs use OAuth 2.0 for authentication, you can obtain an OAuth 2.0 access token from your identity provider (e.g., Azure Active Directory, Auth0) and use it in Postman.
   * Configure Postman to request an access token using OAuth 2.0 authorization flows like Authorization Code, Implicit, or Client Credentials. Then, include the obtained token in the request headers.
3. **JWT Token**:
   * If your APIs use JSON Web Tokens (JWT) for authentication, you can obtain a JWT token from your authentication server or identity provider.
   * In Postman, you can include the JWT token in the request headers, typically in the "Authorization" header as "Bearer {your\_token}".
4. **Client Certificates**:
   * APIM can be configured to require client certificates for authentication. You can generate a client certificate and configure Postman to use it when making requests.
5. **API Management Policies**:
   * APIM policies allow you to add custom authentication and authorization logic to your APIs. You can use policies to validate tokens, check API keys, or perform other authentication checks.
   * In Postman, you may need to include additional headers, parameters, or tokens required by the policies in your requests.
6. **Username/Password**:
   * If your APIs support basic authentication with a username and password, you can include these credentials in the request headers using Postman.
7. **Custom Tokens**:
   * If your APIs use custom tokens or a non-standard authentication mechanism, you'll need to follow the specific authentication process and include the token or credentials in Postman accordingly.

**Version & Revision**

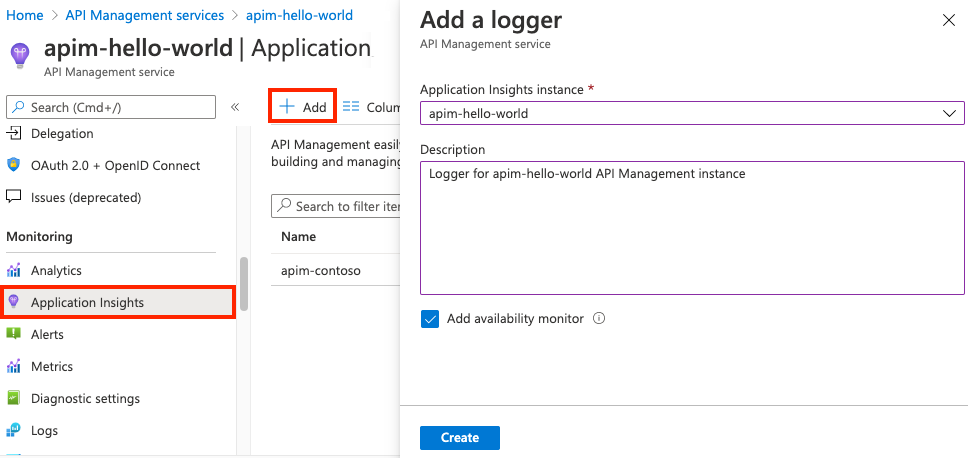
**Revisions allow you to make changes to your APIs in a controlled and safe way. When you want to make changes, create a new revision. You can then edit and test API without disturbing your API consumers. When you're ready, you** then make your revision current. At the same time, you can optionally post an entry to the change log, to keep your API consumers up to date with what has changed. The change log is published to your developer portal.

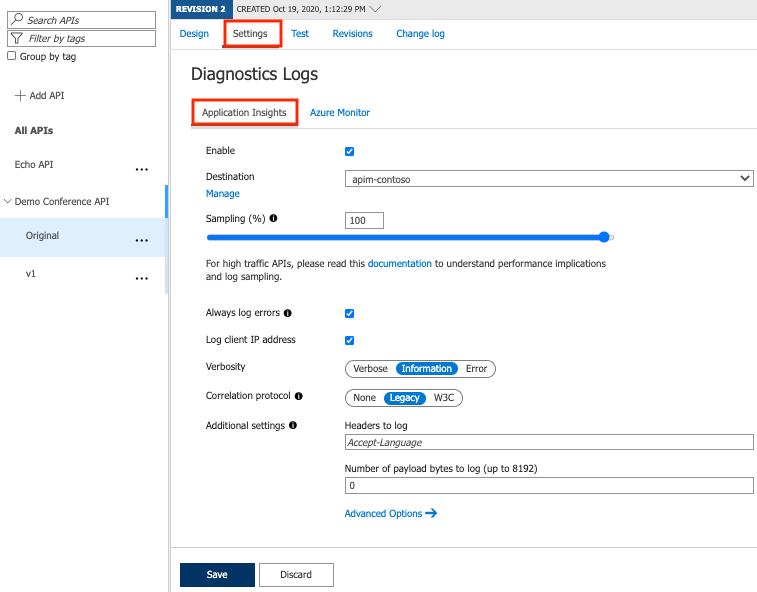


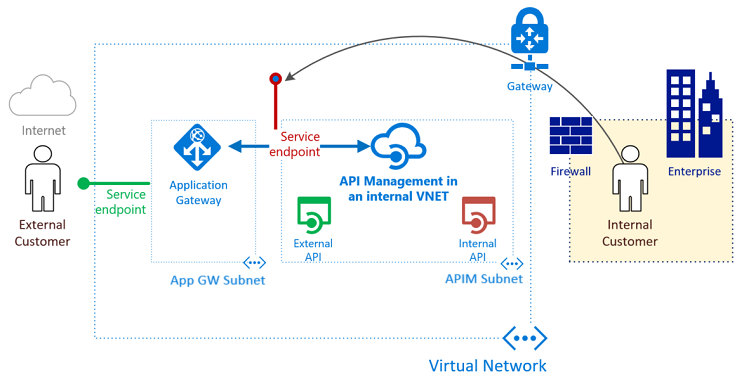


Within "v1" of your API, you identify a small issue that needs to be fixed or a minor improvement to be made. Instead of creating a new version, you create a new revision. Clients accessing "v1" will automatically receive the revision with the fix or improvement.

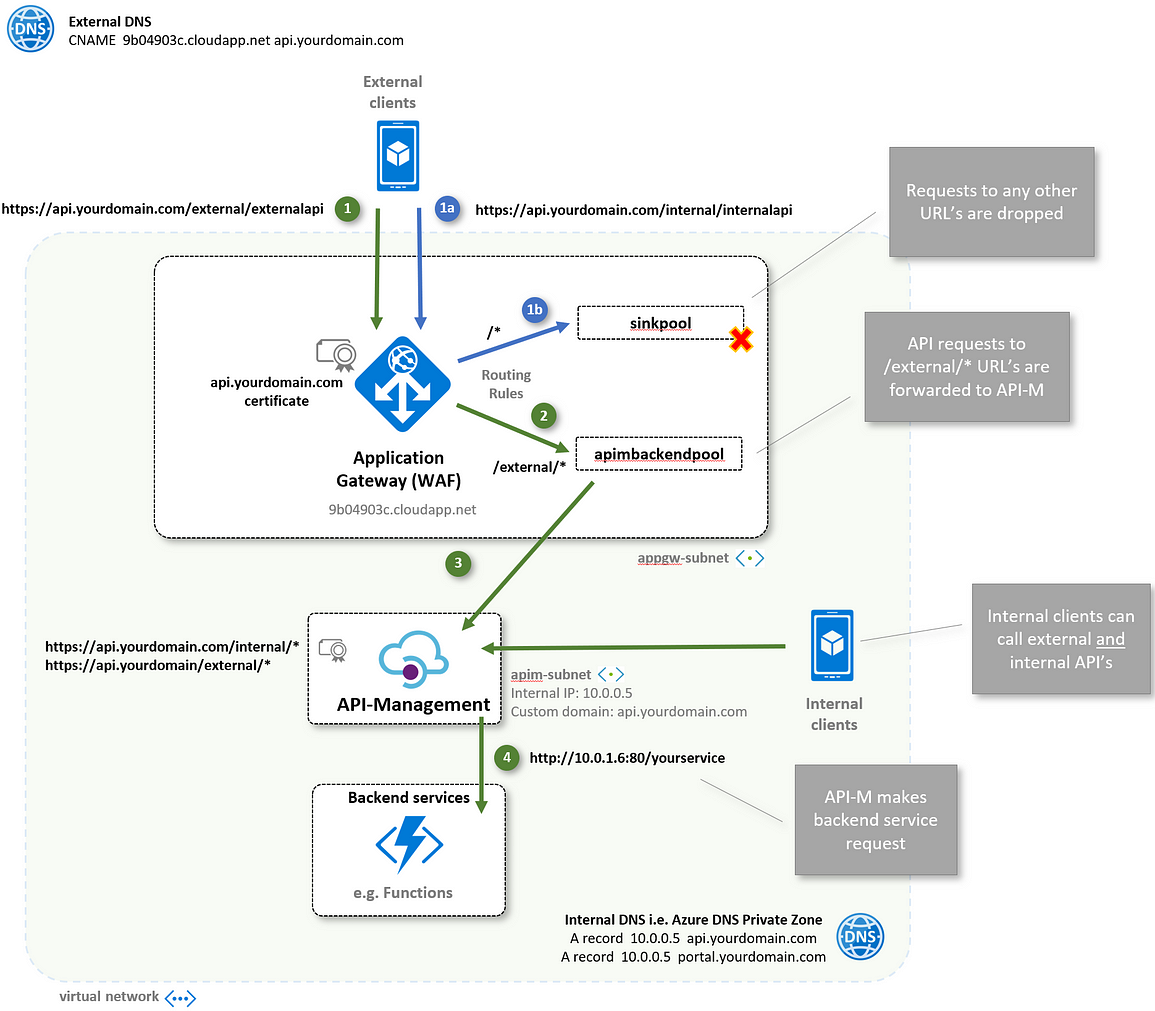
# integrate Azure API Management with Azure Application Insights







<https://medium.com/azure-architects/azure-api-management-and-application-gateway-integration-a31fde80f3db>



**Question and Answer:**

1. **What is Azure API Management, and why is it used in modern application development?**

Azure API Management (APIM) is a cloud-based service provided by Microsoft Azure that enables organizations to publish, secure, analyze, and manage APIs (Application Programming Interfaces). It acts as a gateway and a set of tools for managing APIs, making them accessible to external and internal developers, partners, and customers.

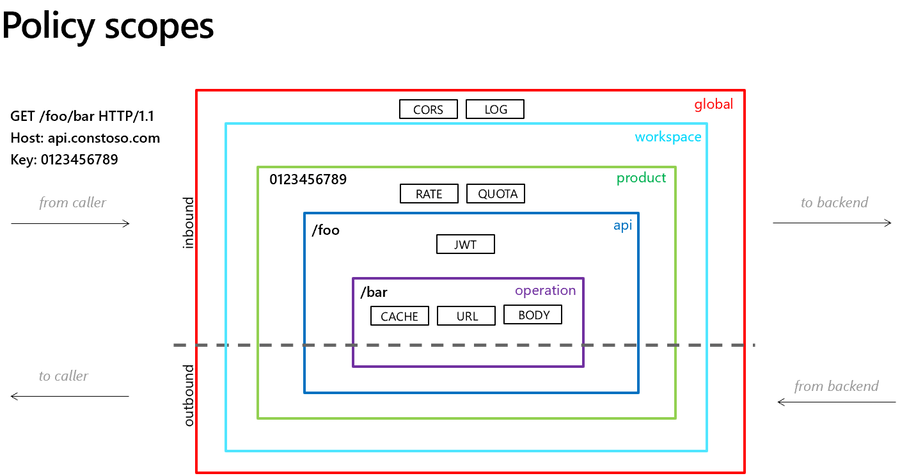
1. **API Gateway**: APIM serves as an API gateway, allowing you to expose your APIs to external consumers over the internet while managing the communication, security, and traffic routing. It acts as a bridge between your backend services and the clients consuming those services.
2. **Security and Authentication**: APIM provides robust security features, including authentication and authorization, to protect your APIs. You can implement policies such as OAuth, JWT validation, API keys, and client certificates to ensure that only authorized clients can access your APIs.
3. **Rate Limiting and Throttling**: You can set rate limits and throttling policies to control the rate at which clients can make requests to your APIs. This helps prevent abuse, ensures fair usage, and protects your backend services from overloading.
4. **Analytics and Monitoring**: APIM offers powerful analytics and monitoring capabilities. You can track API usage, performance, error rates, and other metrics to gain insights into how your APIs are being used and identify areas for improvement.
5. **Developer Portal**: APIM provides a developer portal where you can publish API documentation, interactive API consoles, and code samples. This makes it easier for developers to discover, understand, and test your APIs.
6. **Versioning and Lifecycle Management**: APIM supports API versioning, allowing you to introduce changes to your APIs while maintaining backward compatibility. You can also create multiple versions and revisions of your APIs, making it easier to manage the evolution of your APIs.
7. **Transformation and Routing**: APIM allows you to modify request and response data, route requests to different backend services, and perform content transformation, enabling you to adapt your APIs to different client needs and scenarios.
8. **Developer and Partner Ecosystem**: APIM enables you to expose your APIs to a broader audience, including internal developers, partners, and third-party developers. This can foster innovation, collaboration, and revenue generation through API monetization.
9. **Scalability and Availability**: APIM is a fully managed service in Azure, which means it benefits from Azure's scalability and high availability. It can automatically scale to handle increased traffic and provides redundancy for reliability.
10. **Cost Optimization**: By offloading tasks such as rate limiting, caching, and security to APIM, you can optimize the usage of your backend resources and reduce operational costs.
11. **Explain the key components of Azure API Management, including APIs, Products, and Developers.**

. It consists of several key components, including APIs, Products, and Developers, each playing a crucial role in the API management lifecycle:

1. **APIs**:
   * **APIs**: An API in APIM represents a single RESTful web service or an API endpoint. It is the core component that you want to expose, manage, and secure. Each API definition includes information such as the API's backend service URL, the available operations (GET, POST, PUT, DELETE, etc.), and other metadata.
   * **Operations**: Within an API, operations define the individual actions that can be performed, such as retrieving data, creating records, or updating resources. Operations are associated with specific HTTP methods (e.g., GET, POST) and endpoint URLs.
   * **Schemas**: APIs often have request and response data structures. Schemas define the data models for input and output, making it easier to generate documentation and validate payloads.
   * **Policies**: Policies are configurable rules and transformations applied to API requests and responses. They allow you to enforce security, rate limiting, caching, request/response transformation, and other behaviors at the API level.
2. **Products**:
   * **Products**: Products are logical containers that group one or more APIs together. They provide a way to package APIs with specific configurations and access controls for different audiences. For example, you can create products for free trial users, premium customers, or partner developers.
   * **Policies**: Like APIs, products can have policies applied at their level. These policies can include rate limiting, subscription requirements, and custom transformations.
   * **Access Control**: Products allow you to control who can access the associated APIs. You can require developers to subscribe to a product and obtain an API key for access.
   * **Pricing**: Products can be associated with pricing tiers, allowing you to define different usage plans and monetize your APIs if needed.
3. **Developers**:
   * **Developers**: Developers represent individuals or organizations that consume your APIs. APIM provides tools to manage developer identities, including registration, authentication, and subscription management.
   * **Subscriptions**: Developers obtain subscriptions to specific products, which grant them access to the associated APIs. Subscriptions are typically associated with API keys or OAuth tokens.
   * **Access Control**: Developers can be assigned varying levels of access, such as administrator, contributor, or reader roles, which determine their ability to manage APIs and products within APIM.
   * **Developer Portal**: APIM offers a developer portal where developers can discover APIs, read documentation, obtain API keys, and access interactive API consoles for testing and development.
4. **How do you onboard an API into Azure API Management?**

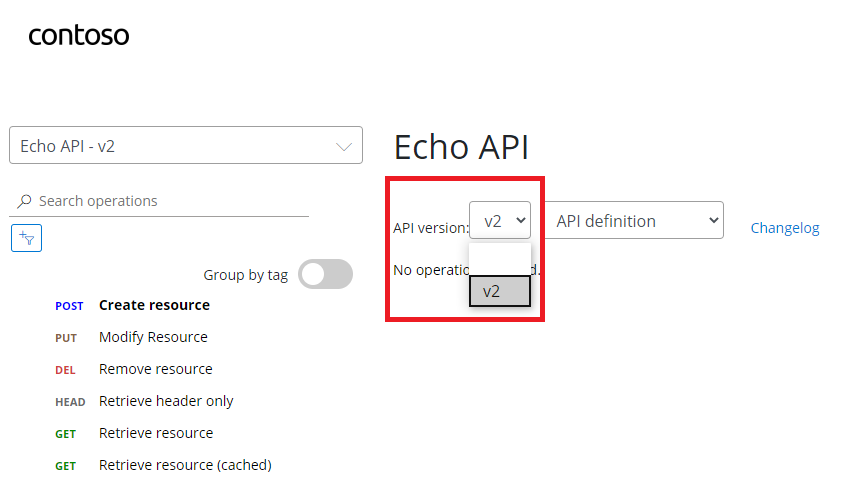
Navigate to your API Management service in the Azure portal and select APIs from the menu. From the left menu, select + Add API. Select HTTP from the list. Enter the backend Web service URL (for example, https://httpbin.org ) and other settings for the API.

1. **What is the purpose of API policies in Azure APIM, and how can they be applied to APIs?**



1. **Discuss the importance of versioning APIs in Azure APIM. How can you manage API versioning effectively?**

**Inside developer portal**



1. **Explain the concept of API throttling and rate limiting in Azure API Management. How can you configure these policies?**

**throttling** is the process of limiting the number of API requests a user can make in a certain period.

rate limiting and throttling are used to control the amount of incoming traffic to your APIs.

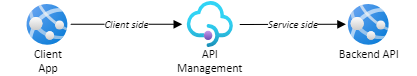
You can set quotas, rate limits, and burst limits to manage how many requests can be made to your APIs within a specific time frame.

These limits can be applied at various levels, including API level, operation level, or even per subscription.

RATE (to allow a maximum of 100 requests per minute (RPM) for each client),

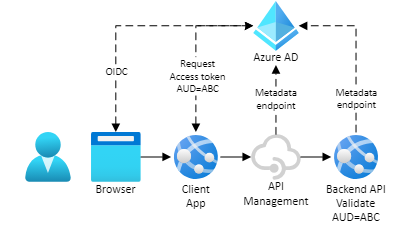
Quata (want to limit a client to 10,000 requests per day)

1. **What authentication and authorization options are available in Azure APIM to secure APIs?**

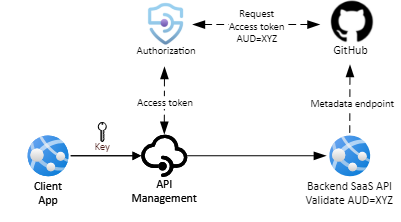
****

**Scenario 1 - Client app authorizes directly to backend**

common authorization scenario is when the calling application requests access to the backend API directly and presents an OAuth 2.0 token in an authorization header to the gateway.

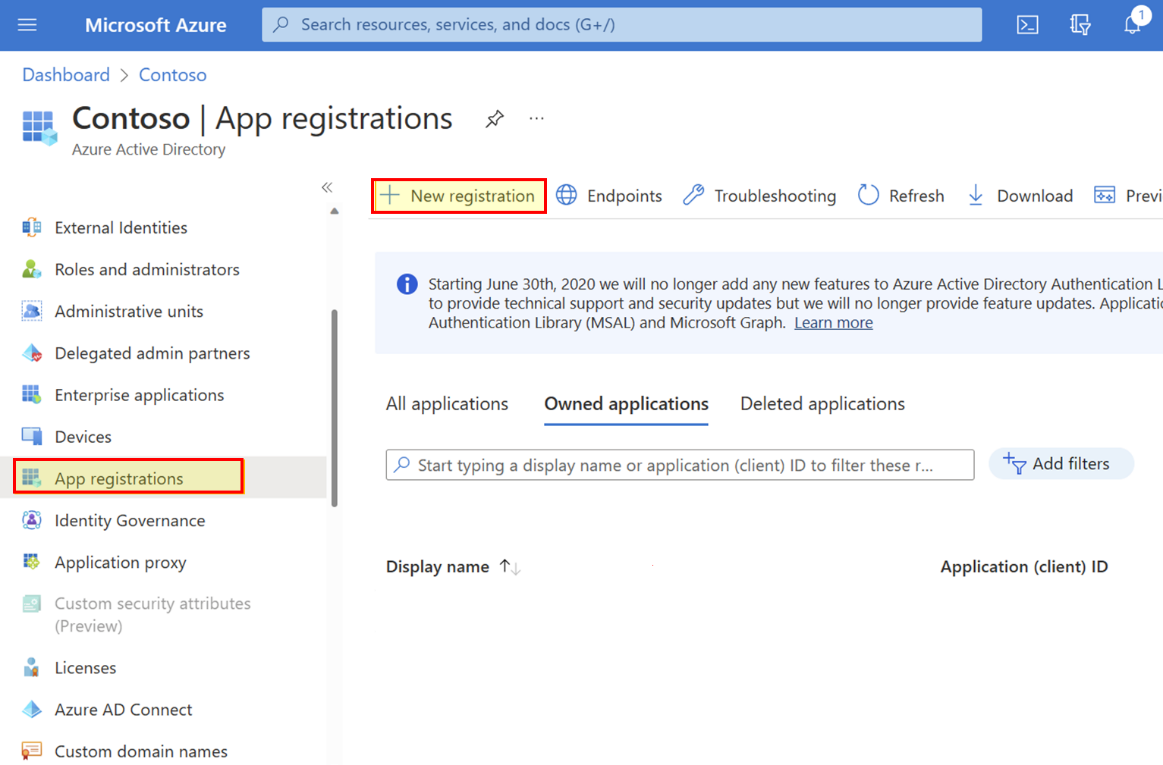
****

### Scenario 3: API management authorizes to backend

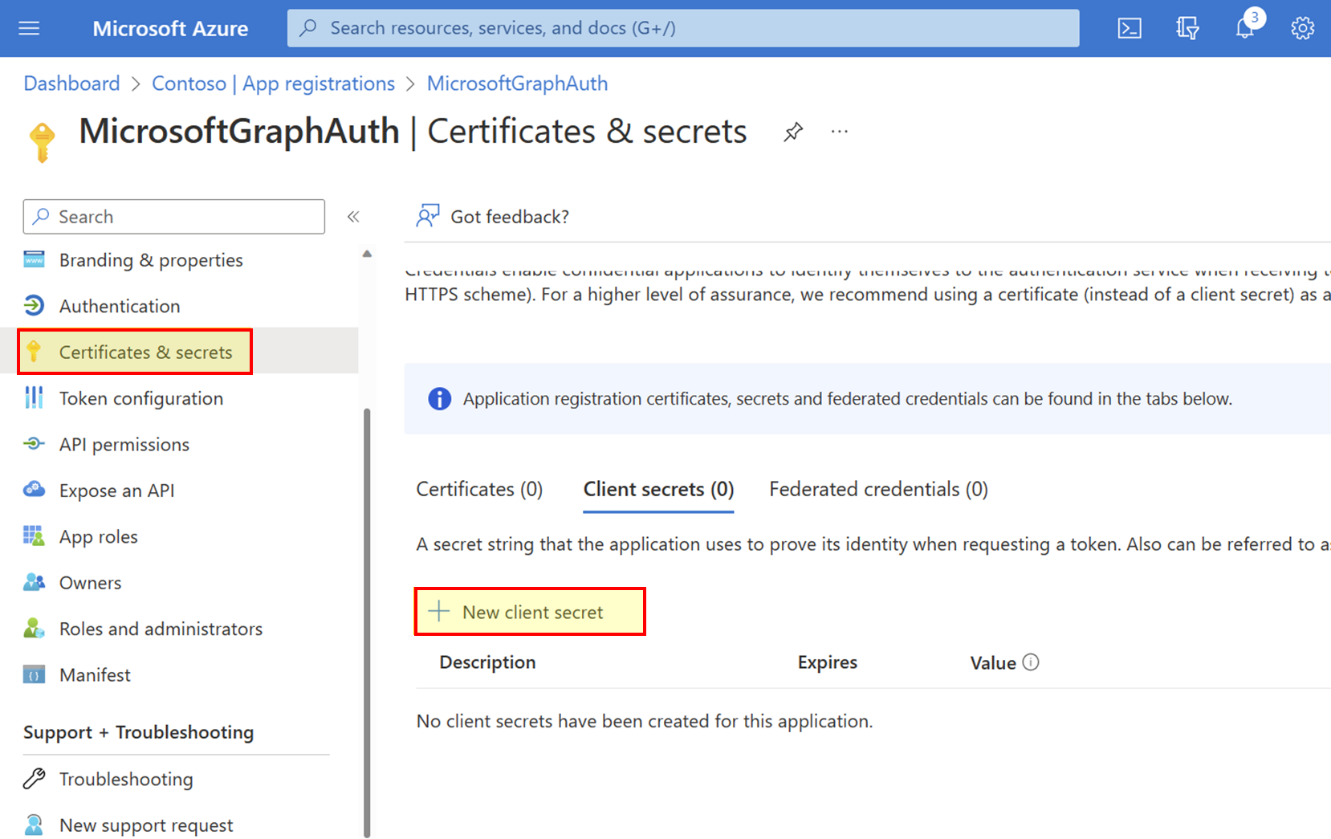
****

Steps:

* Create an Azure AD application
* Create and configure an authorization in API Management
* Configure an access policy
* Create a Microsoft Graph API in API Management and configure a policy
* Test your Microsoft Graph API in API Management

****

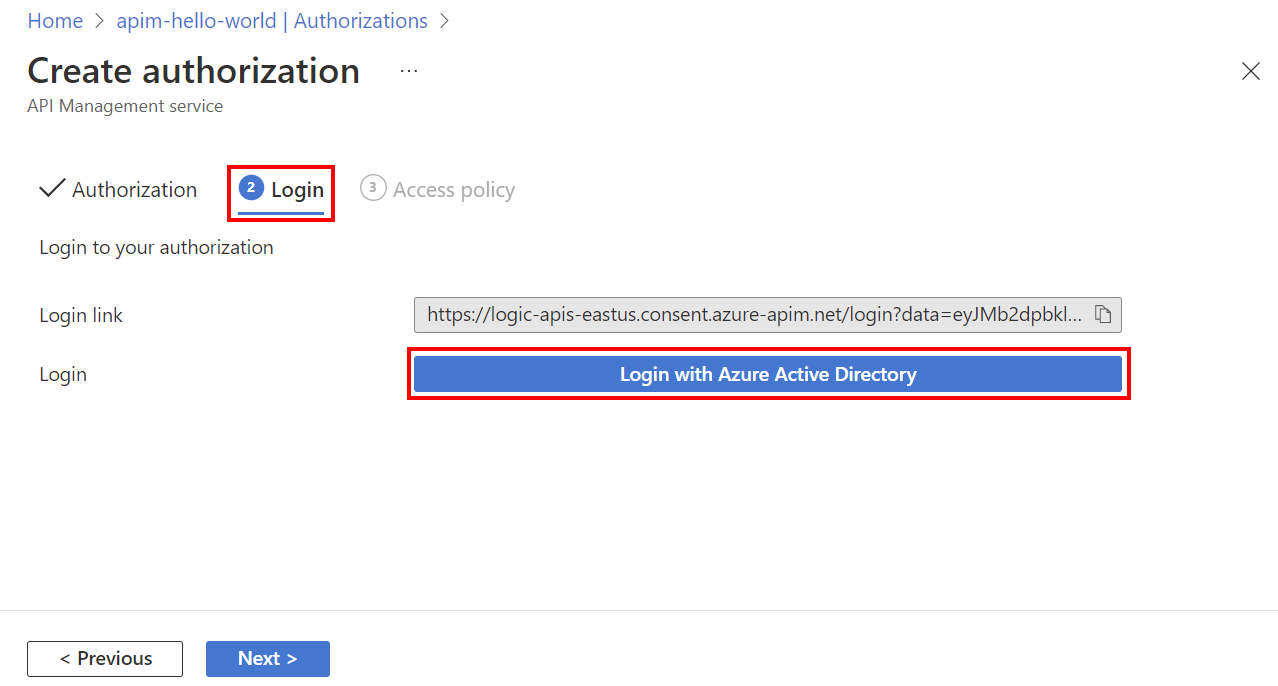
****

****

| **Settings** | **Value** |
| --- | --- |
| **Provider name** | A name of your choice, such as *aad-01* |
| **Identity provider** | Select **Azure Active Directory v1** |
| **Grant type** | Select **Authorization code** |
| **Client id** | Paste the value you copied earlier from the app registration |
| **Client secret** | Paste the value you copied earlier from the app registration |
| **Resource URL** | https://graph.microsoft.com |
| **Tenant ID** | Optional for Azure AD identity provider. Default is *Common* |
| **Scopes** | Optional for Azure AD identity provider. Automatically configured from AD app's API permissions. |
| **Authorization name** | A name of your choice, such as *aad-auth-01* |

**Step 2: Configure an authorization in API Management**

## Step 3: Authorize with Azure AD and configure an access policy

****

## Step 4: Create a Microsoft Graph API in API Management and configure a policy

**<policies>**

**<inbound>**

**<base />**

**<get-authorization-context provider-id="aad-01" authorization-id="aad-auth-01" context-variable-name="auth-context" identity-type="managed" ignore-error="false" />**

**<set-header name="authorization" exists-action="override">**

**<value>@("Bearer " + ((Authorization)context.Variables.GetValueOrDefault("auth-context"))?.AccessToken)</value>**

**</set-header>**

**</inbound>**

**<backend>**

**<base />**

**</backend>**

**<outbound>**

**<base />**

**</outbound>**

**<on-error>**

**<base />**

**</on-error>**

**</policies>**

[**https://learn.microsoft.com/en-us/azure/api-management/authorizations-how-to-azure-ad**](https://learn.microsoft.com/en-us/azure/api-management/authorizations-how-to-azure-ad)

**Other Way:**

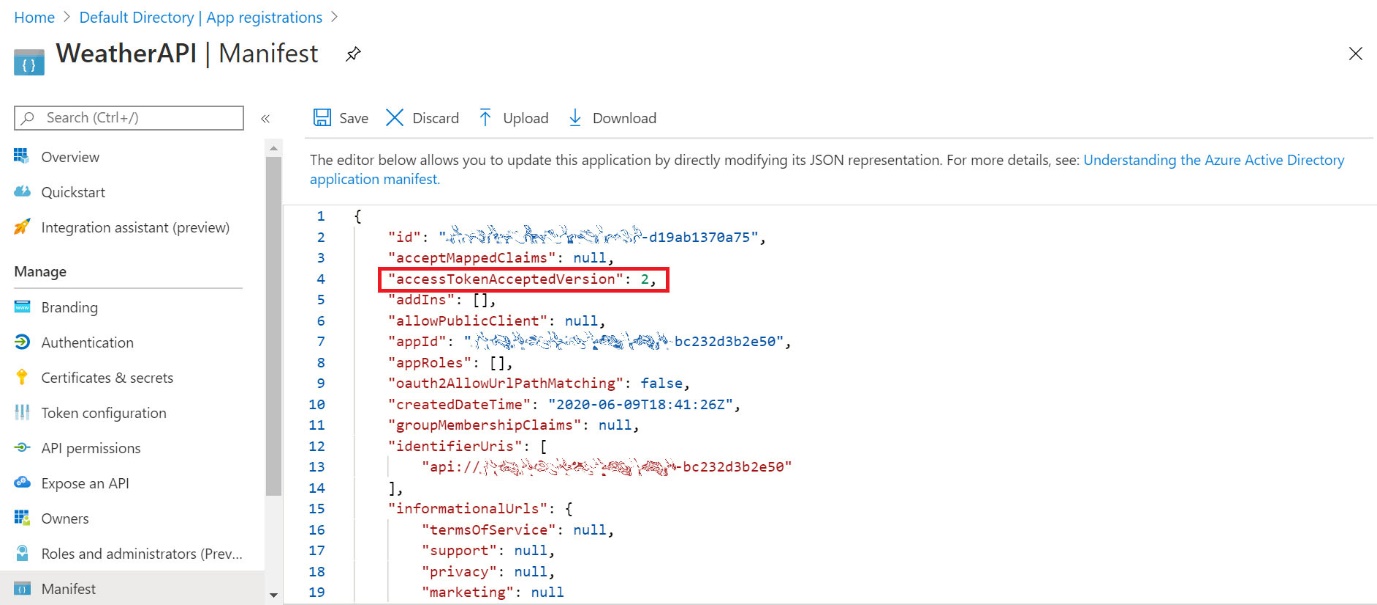
# Authorization with Azure API Management



### 1. Register the Client and the API Resource in AAD

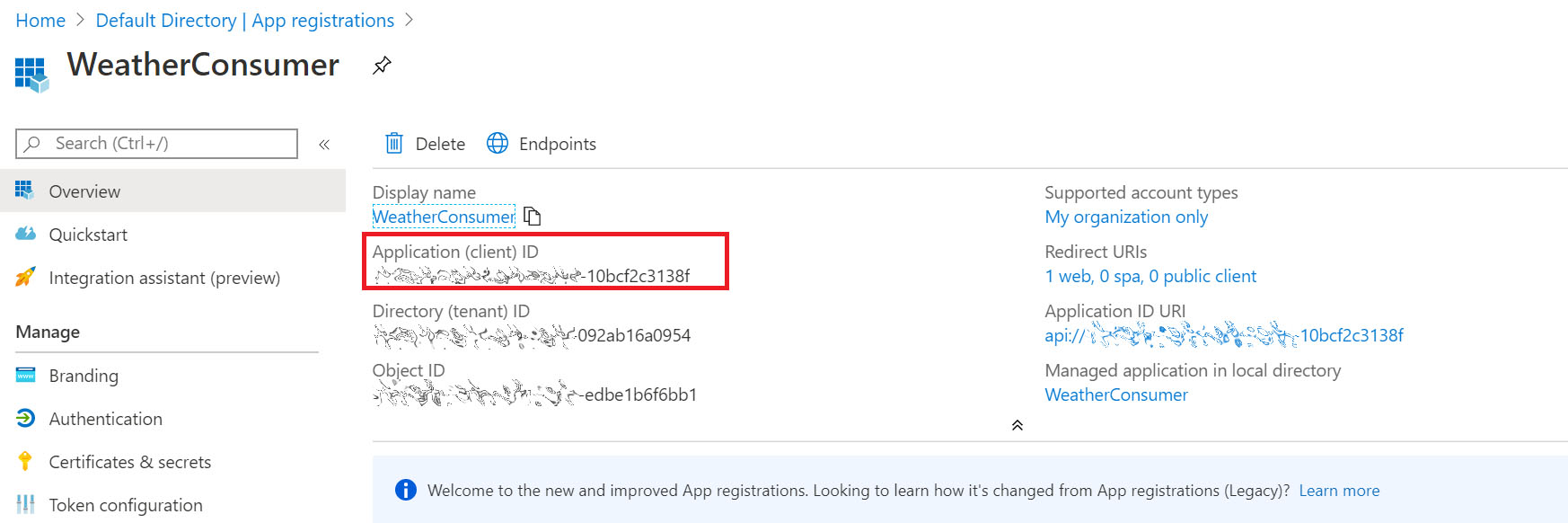
### 2. Configure the AAD Applications

##### 2.1 Set Access Token Version in API Resource AAD Application

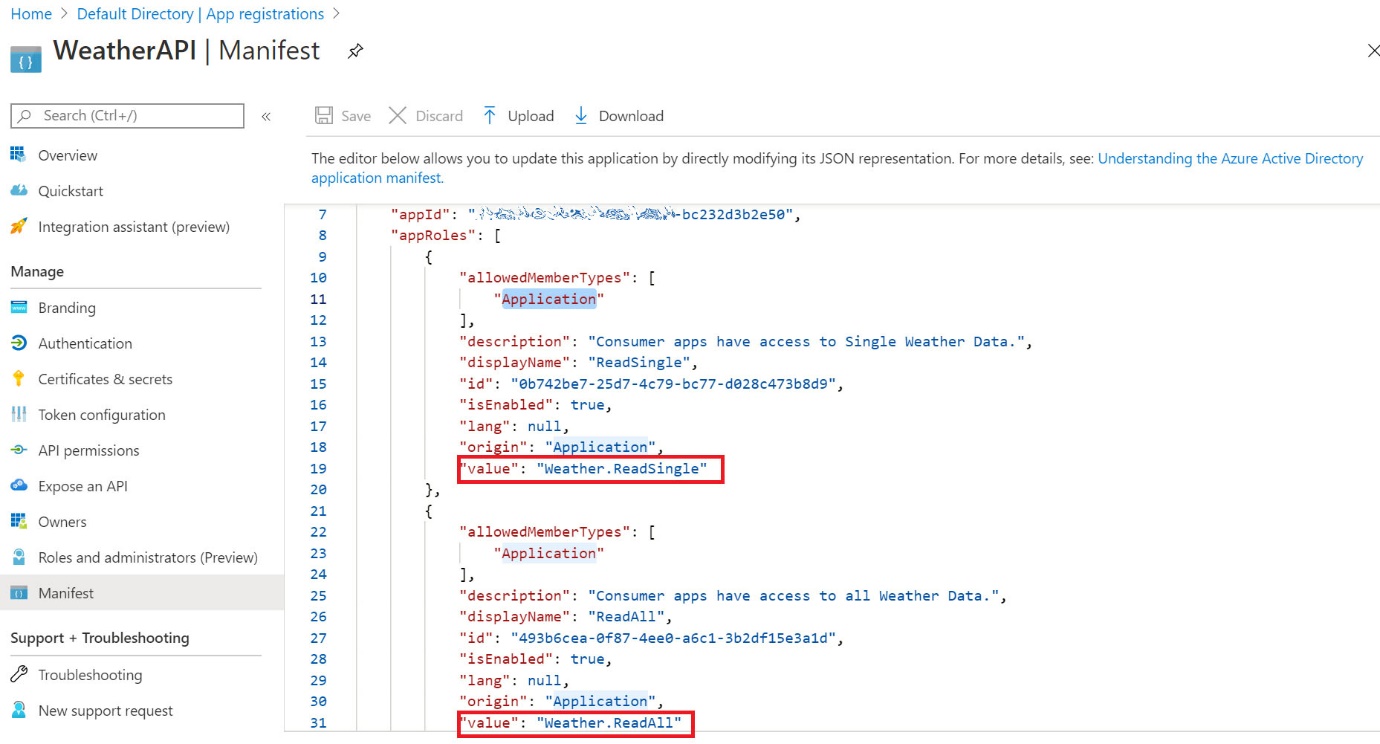
 

##### 2.2 Expose the API Application

##### 2.3 Set the Client Secret in Client AAD Application



**2.4 Define Application Roles for the API Application**

I created two roles in the [[appRoles](https://docs.microsoft.com/en-us/azure/active-directory/develop/howto-add-app-roles-in-azure-ad-apps" \t "_blank)] field: **Weather.ReadSingle**, and **Weather.ReadAll** in the WeatherAPI [Application Manifest](https://docs.microsoft.com/en-us/azure/active-directory/develop/reference-app-manifest) editor, one role for each APIM operation I have. 

**"appRoles":** [

{

**"allowedMemberTypes":** [

"Application"

],

**"description":** "Consumer apps have access to all Weather Data.",

**"displayName":** "ReadAll",

**"id":** "493b6cea-0f87-4ee0-a6c1-3b2df15e3a1d",

**"isEnabled":** **true**,

**"lang":** **null**,

**"origin":** "Application",

**"value":** "Weather.ReadAll"

},

{

**"allowedMemberTypes":** [

"Application"

],

**"description":** "Consumer apps have access to Single Weather Data.",

**"displayName":** "ReadSingle",

**"id":** "0b742be7-25d7-4c79-bc77-d028c473b8d9",

**"isEnabled":** **true**,

**"lang":** **null**,

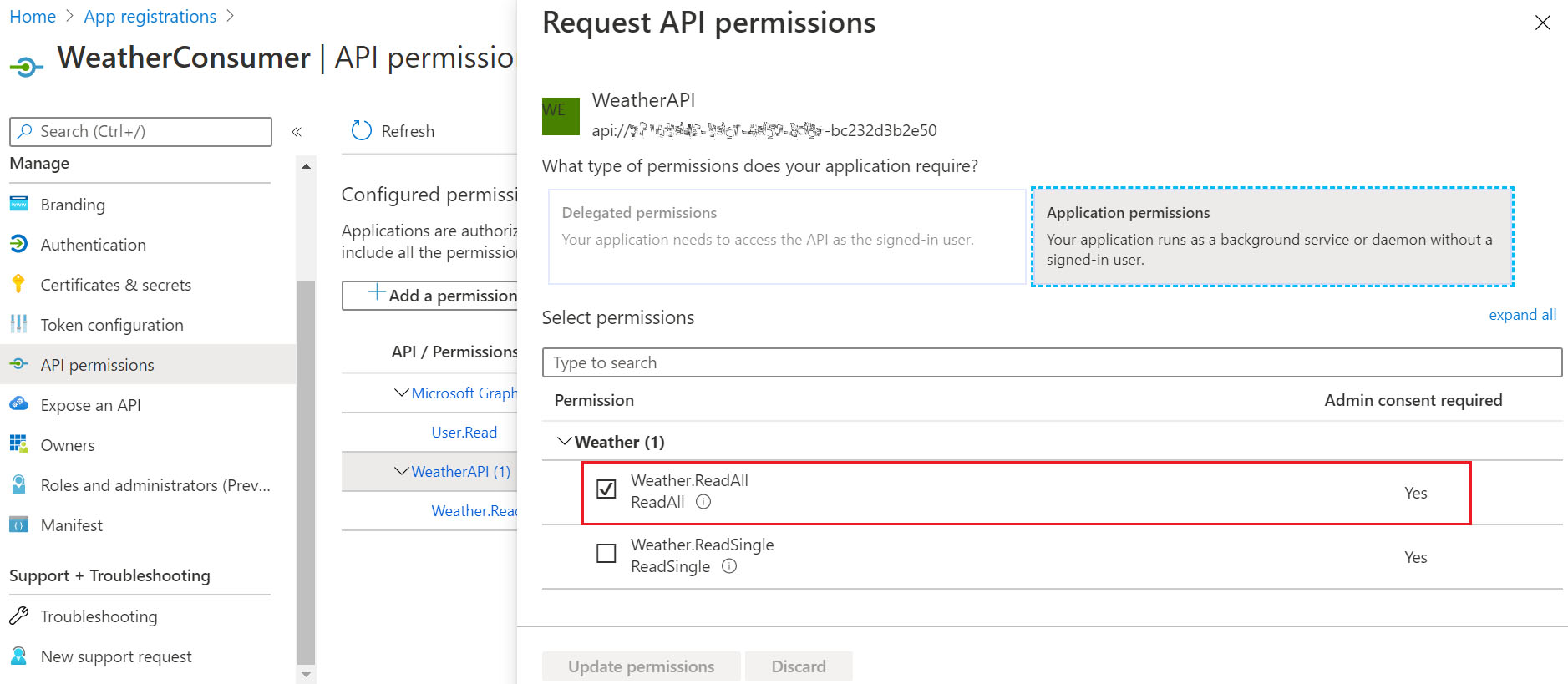
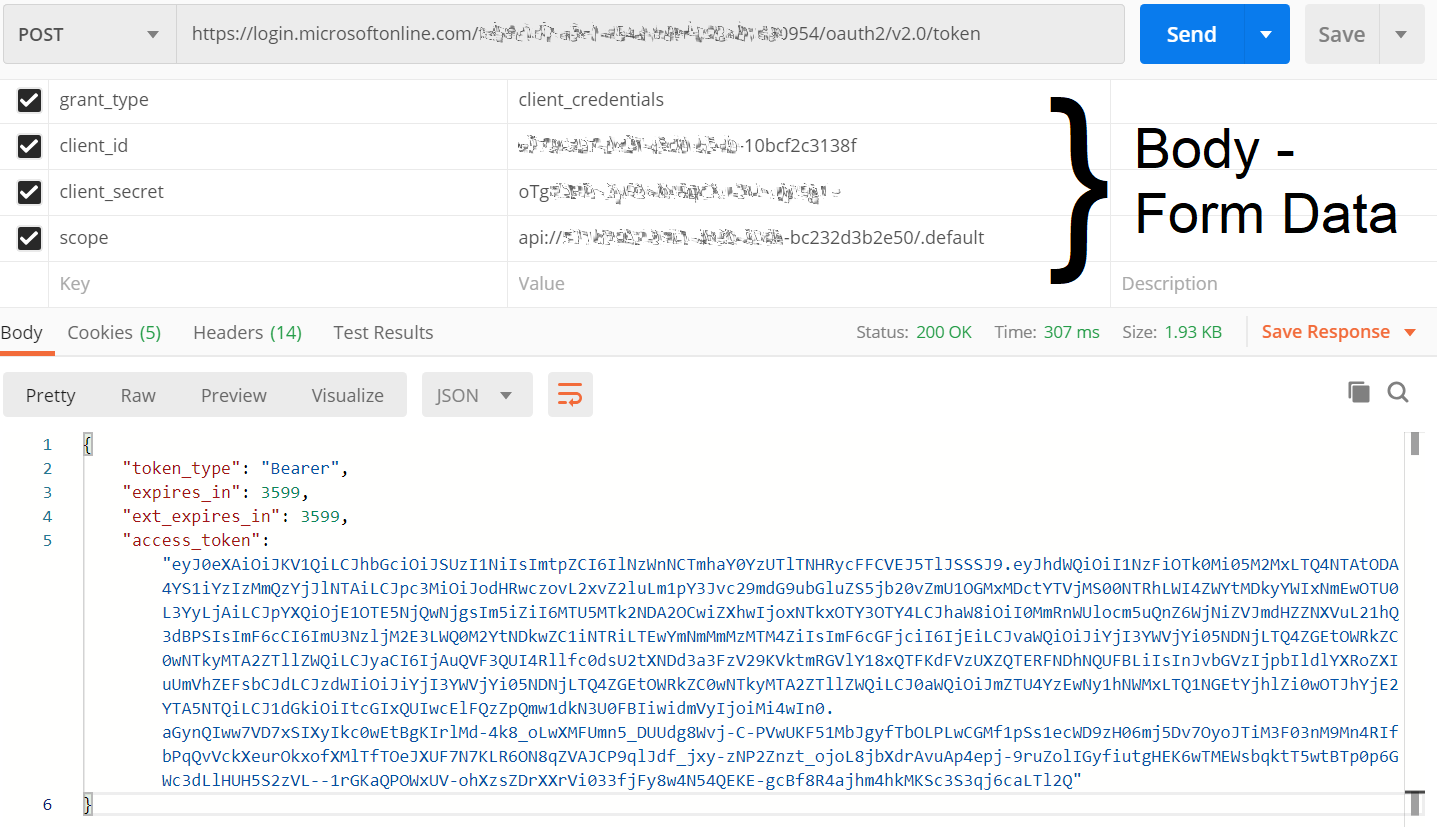
**"origin":** "Application",

**"value":** "Weather.ReadSingle"

}

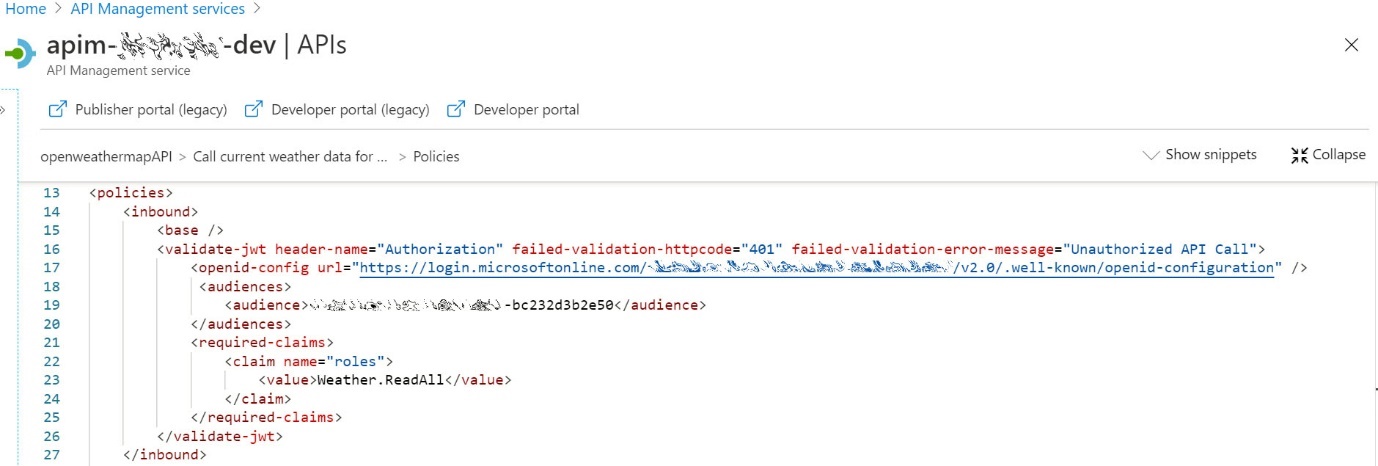
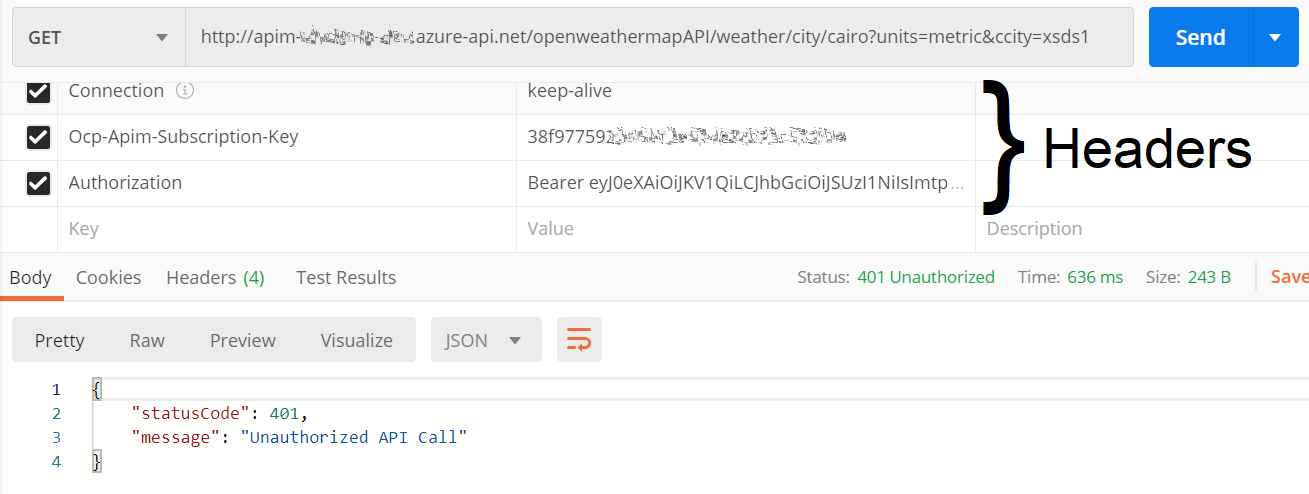
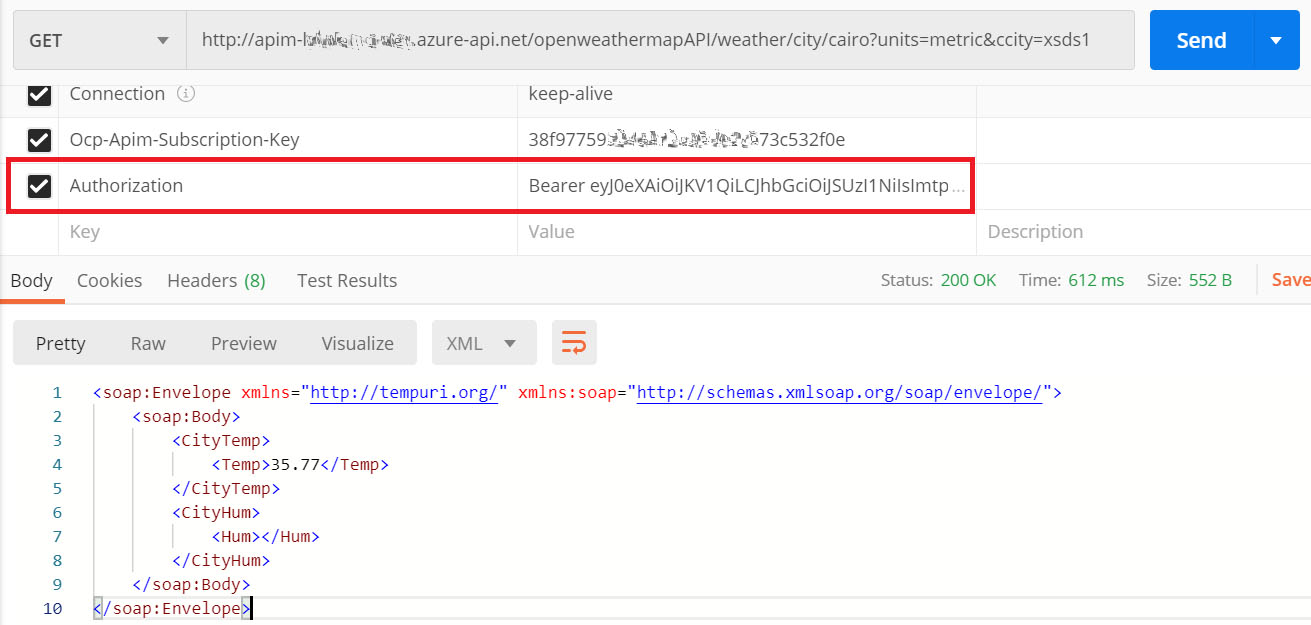
]

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

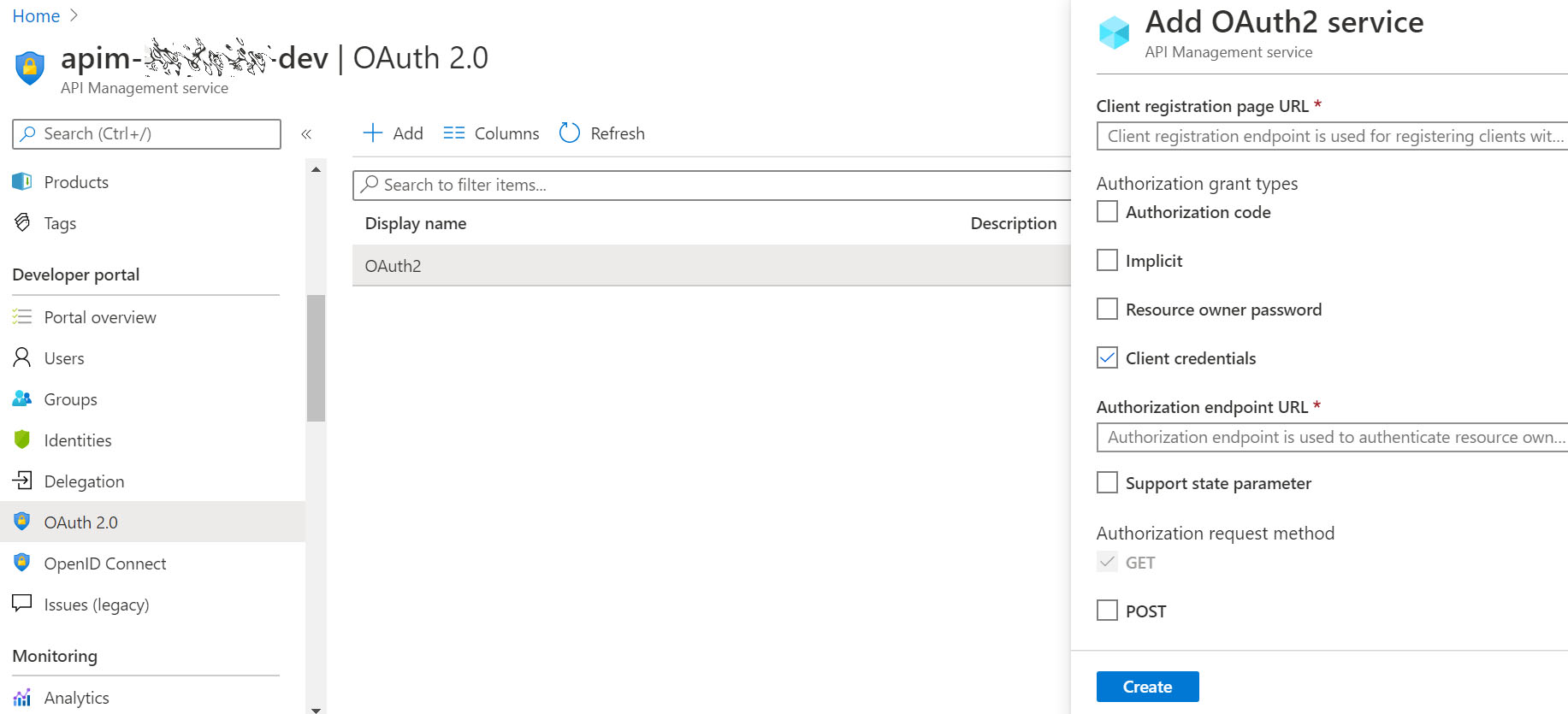
### 3. Configuring APIM Instance

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

##### 3.3 APIM Developer Portal OAuth 2.0 settings

it will allow developers to test OAuth 2.0 APIs in the APIM developer portal interactive console by retrieving the access token on their behalf.

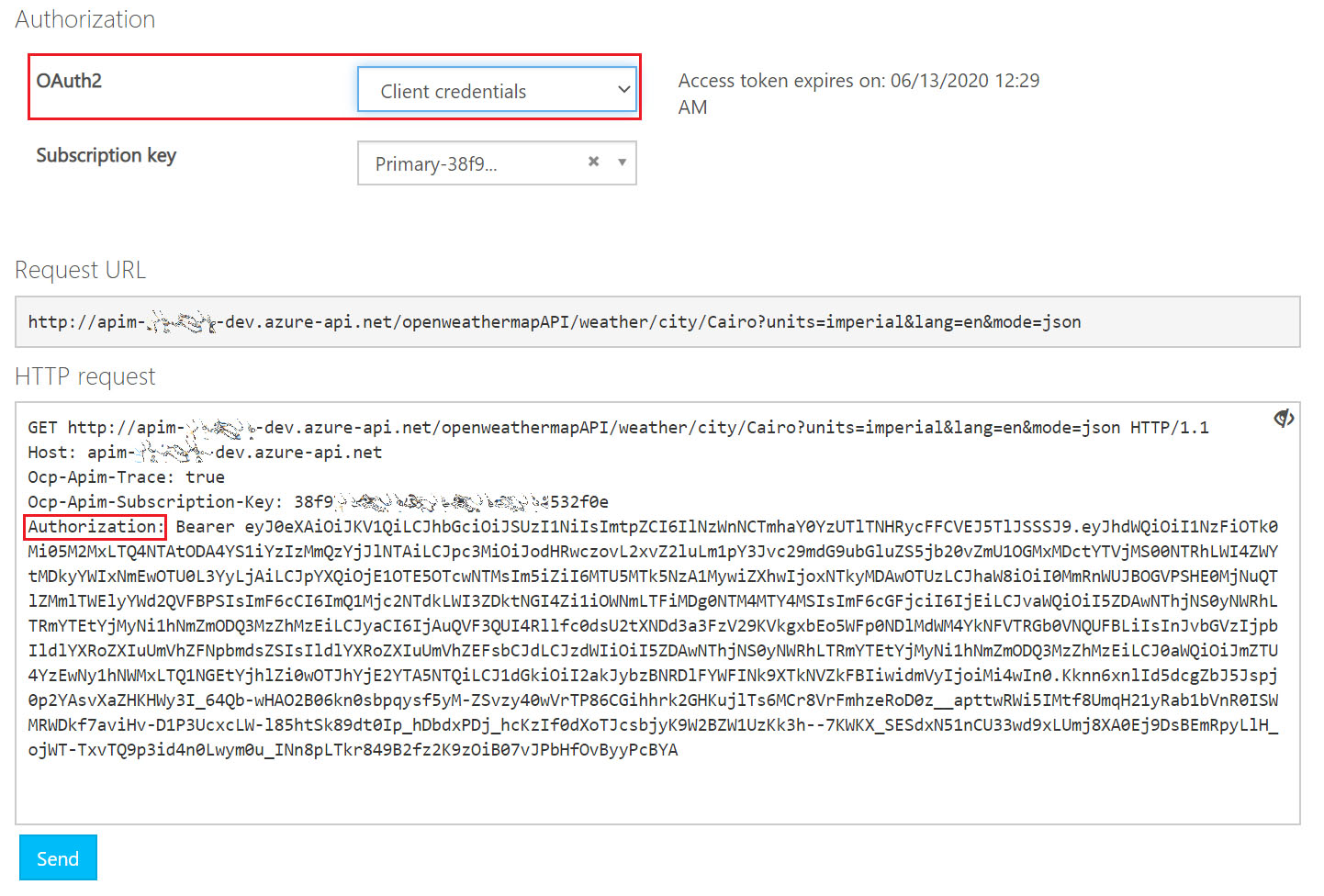
 We need to set the following fields, and leave the default ones for the rest. Note that some fields are irrelevant for our grant type, however these are mandatory in this blade and need to be filled anyway.

* Client registration page URL: http://localhost (just a placeholder)
* Authorization grant types: **Client credentials**
* Authorization endpoint URL: https://login.microsoftonline.com /{tenantID}/oauth2/v2.0/authorize
* Token endpoint URL: https://login.microsoftonline.com /{tenantID}/oauth2/v2.0/token
* Default scope: api://{APIResourceApplicationID}/.default
* Client ID: {DevConsoleApplicationID}
* Client secret: {DevConsoleApplicationSecret}

Finally, we will need to set the Security settings on the API-level and set the [User authorization]: **OAuth 2.0**and set the [OAuth 2.0 server] to the authorization service we just created, as shown below.



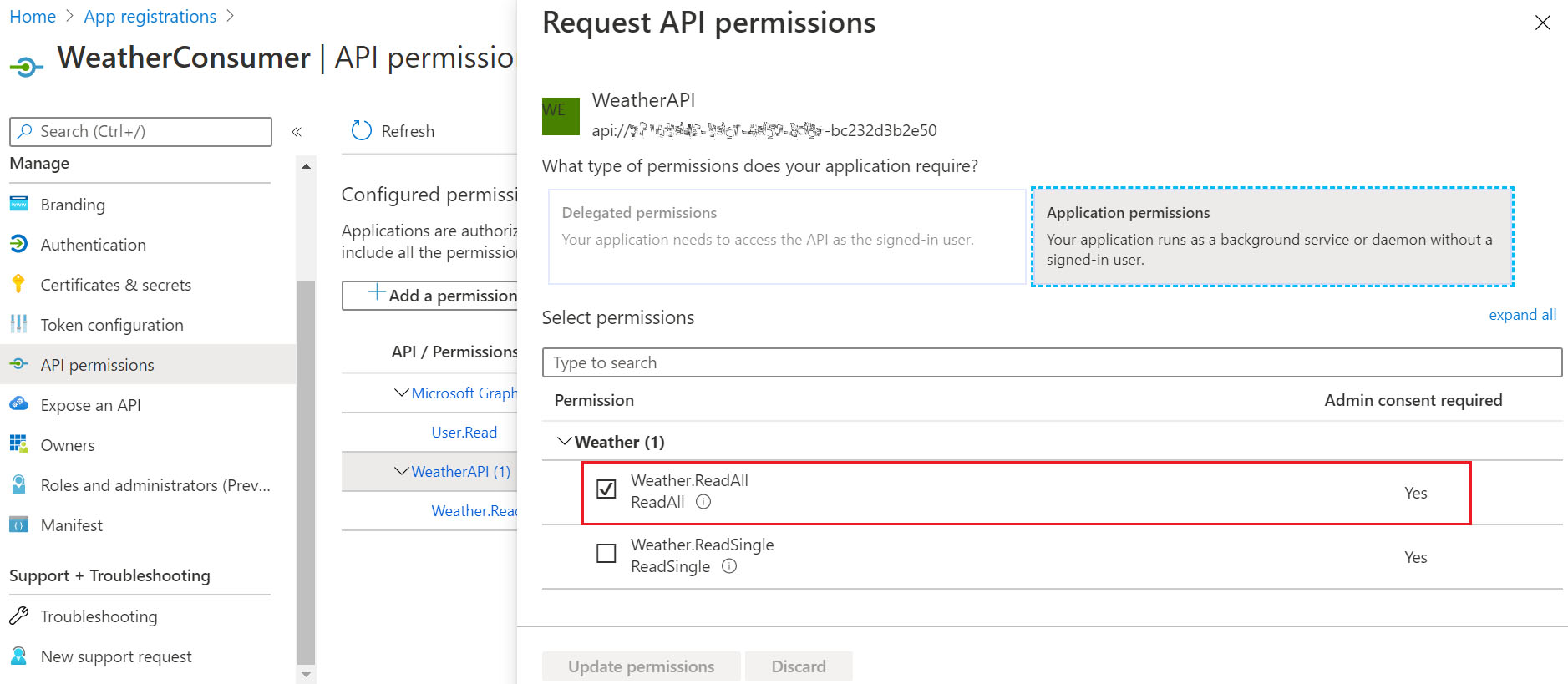
Now, API developers can test the protected APIs by simply selecting the pre-configured authorization service in the developer portal and it will automatically retrieve the JWT and include it in the API request to be tested, as shown below.



[**https://tointegrationandbeyond.com/blogs/index.php/2020/06/13/authorization-with-azure-api-management/**](https://tointegrationandbeyond.com/blogs/index.php/2020/06/13/authorization-with-azure-api-management/)

1. **How can you integrate Azure Active Directory (Azure AD) with Azure API Management for authentication and authorization?**

**Azure app registration 🡪(frontend and backend api) 🡪Expose API (URL)**

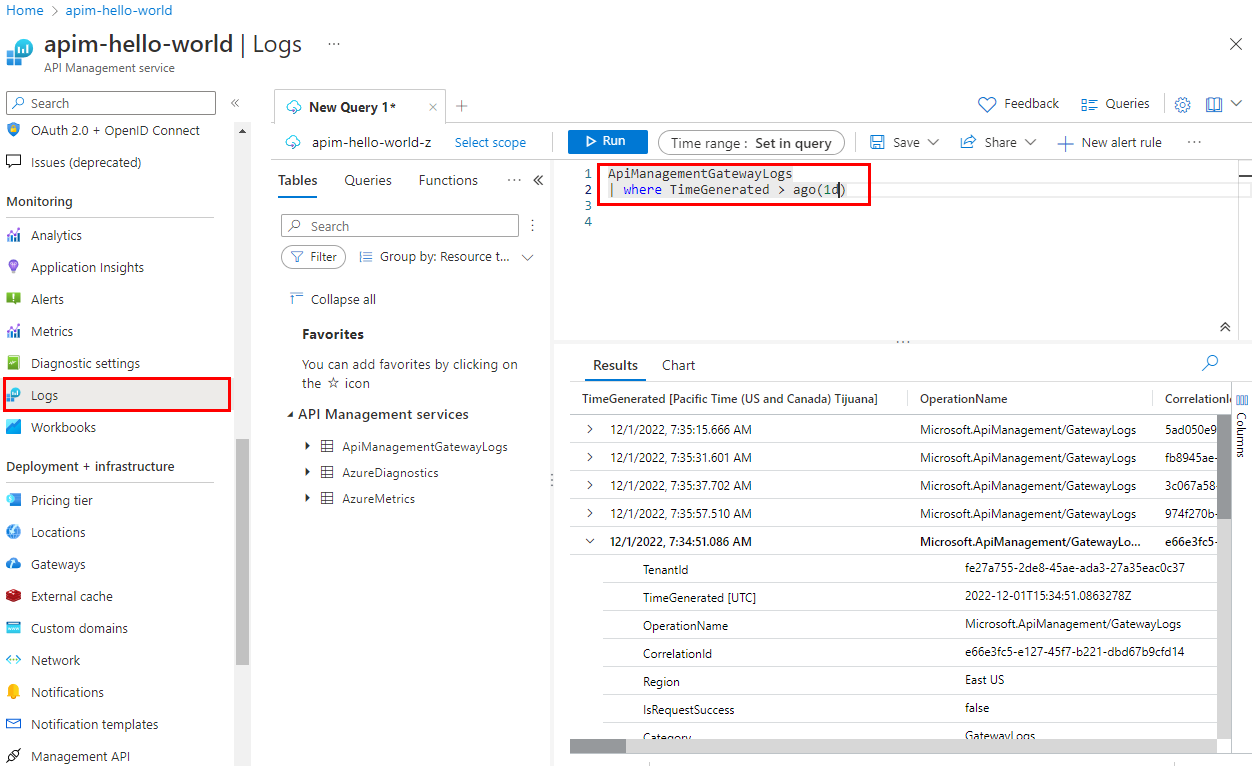


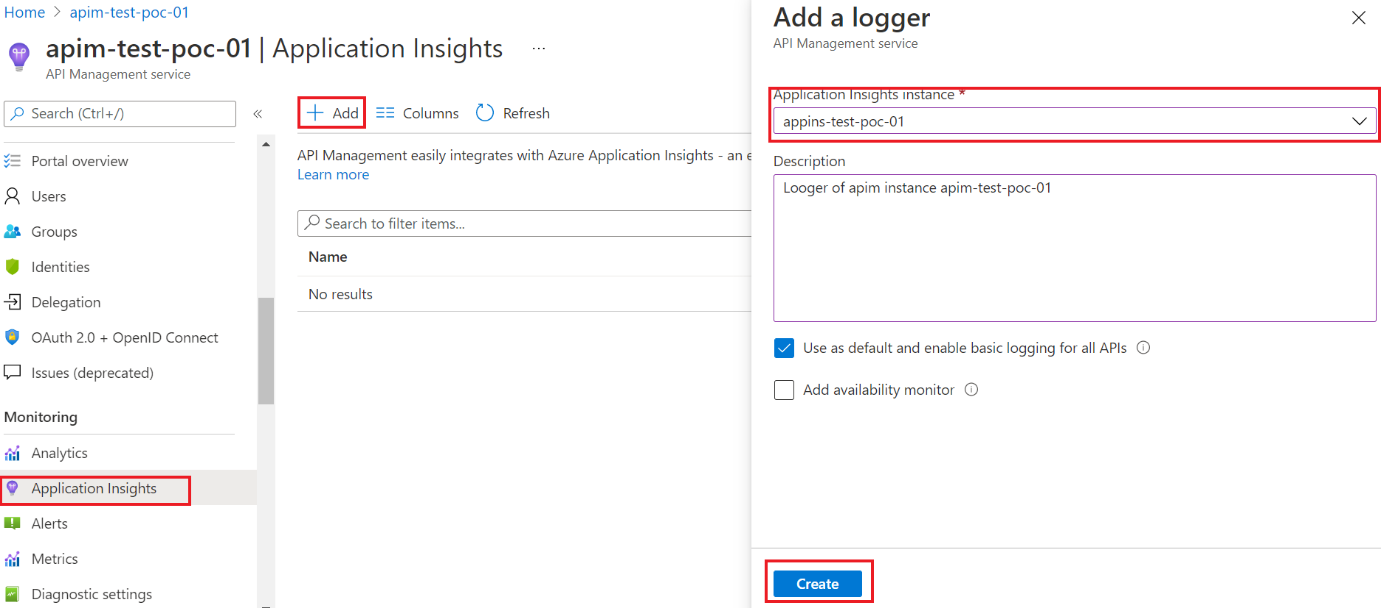
1. **What is the Developer Portal in Azure API Management, and why is it valuable for API consumers?**

**Developer Portal:** Azure API Management offers a developer portal for API documentation, testing, and onboarding.

1. **What is caching in Azure APIM, and when should you use it to improve API performance?**
2. **Discuss the role of transformation policies in Azure APIM and how they can be used to modify API requests and responses.**
3. **How can you monitor and analyze API usage and performance in Azure API Management?**

**Azure Monitor, Azure application Insight, API Management Analytics**





1. **Explain the concept of API mocking in Azure APIM and when it might be useful during API development.**
2. **What are API revisions in Azure API Management, and how do they differ from versions?**
3. **Describe the process of importing and exporting API configurations in Azure APIM.**
4. **How do you scale Azure API Management to handle increased traffic and ensure high availability?**

**Scale Up (Vertical Scaling):**

* **Performance Tiers**: Azure API Management offers different performance tiers (e.g., Developer, Basic, Standard, Premium) with varying levels of resources. Start by selecting an appropriate tier that meets your initial needs.
* **SKU Size**: Within a tier, you can choose the SKU size that determines the number of virtual CPUs and memory allocated to your API Management instance. You can vertically scale by upgrading to a larger SKU size when needed.
* **Auto-scaling**: Configure auto-scaling policies to automatically adjust the number of instances or the SKU size based on predefined conditions, such as CPU usage or request rate.

**2. Scale Out (Horizontal Scaling):**

* **Increase the Number of Instances**: You can add more API Management instances to distribute the traffic load across multiple instances. Use Azure Traffic Manager or Azure Application Gateway for load balancing between instances.
* **Regional Deployment**: Deploy API Management instances in different Azure regions to provide geographic redundancy and reduce latency for users in different locations.

**3. Caching and Content Delivery:**

* **Caching**: Implement caching for frequently accessed data to reduce the load on your backend services. Azure API Management supports caching policies that can be configured at the API level.
* **Content Delivery**: Utilize Azure Content Delivery Network (CDN) to cache and deliver static content, reducing the load on your API Management instance.

**4. Rate Limiting and Quotas:**

* Implement rate limiting and quotas to control the number of requests from each client or subscription, preventing overuse of your API resources.

**5. Content Compression:**

* Enable content compression to reduce the amount of data transmitted over the network, improving response times and reducing bandwidth consumption.

**6. Health Monitoring and Alerts:**

* Set up health probes to monitor the availability of your API Management instances and backend services.
* Configure alerts to notify you of any issues or performance anomalies. Azure Monitor and Azure Application Insights are valuable tools for monitoring and alerting.

**7. Redundancy and Failover:**

* Implement redundancy by configuring multiple API Management instances in different Azure regions to ensure high availability. Use Azure Traffic Manager for automatic failover.
* Enable the active-standby configuration to have a standby instance ready to take over in case the primary instance becomes unavailable.

**8. Geo-replication:**

* Implement geo-replication to have a hot standby instance in another Azure region. This approach provides disaster recovery capabilities in addition to high availability.

**9. Backups:**

* Regularly back up your API Management configuration and data to ensure that you can quickly recover in case of data loss or corruption.

**10. Application Gateway Integration:**

* Consider integrating Azure API Management with Azure Application Gateway for advanced traffic management, SSL offloading, and Web Application Firewall (WAF) capabilities.

**11. Azure Front Door Service:**

* Utilize Azure Front Door Service for global load balancing, security, and fast failover capabilities to ensure high availability.

**12. Disaster Recovery Plan:**

* Develop and test a disaster recovery plan to ensure business continuity in the event of an outage.

By implementing these scaling and high availability strategies, you can ensure that your Azure API Management instance can handle increased traffic and provide a highly available and reliable API service for your users.

1. **Discuss the use of custom domains and SSL certificates in Azure APIM for branding and security purposes.**
2. **Explain the integration options for connecting Azure APIM to backend services and how you handle API routing.**
3. **What is the developer experience like when working with Azure APIM, and how can it be customized for different users?**
4. **Can you provide an example of a real-world scenario where Azure API Management solved a specific problem or improved an organization's API strategy?**

**Problem:** Many organizations have legacy systems or services that were developed using older technologies and protocols. These legacy systems may not be easily accessible or usable by modern applications, especially those built using RESTful APIs or other modern standards. The organization needs to find a way to expose these legacy services to support digital transformation initiatives, mobile applications, and partner integrations.

1. **Legacy System Integration:** **Azure API Management allows organizations to create a bridge between legacy systems and modern applications**. The organization connects Azure API Management to the legacy service using various connectors, such as SOAP, OData, or custom connectors, depending on the legacy system's technology.
2. **Transformation and Aggregation:** Azure API Management can transform the data and protocols used by the legacy service into more modern formats like JSON and REST. It can also aggregate data from multiple legacy services into a single API endpoint, simplifying the integration for developers.
3. **Security and Authentication:** Azure API Management provides robust security features, including authentication, authorization, and rate limiting. It ensures that only authorized users and applications can access the exposed APIs, maintaining security for legacy systems.
4. **Developer Portal:** The developer portal in Azure API Management allows developers to discover and access the newly exposed APIs. Detailed documentation, interactive testing, and code samples help developers understand how to use the APIs effectively.
5. **Traffic Management:** Azure API Management helps manage the traffic to the legacy systems, preventing overloading and ensuring high availability. Rate limiting, caching, and request/response policies can be configured to optimize performance.

**Benefits:**

* **Modernization**: The organization can modernize its existing IT infrastructure and make legacy systems accessible to modern applications and clients without the need for extensive redevelopment.
* **Time-to-Market**: By using Azure API Management, the organization accelerates the delivery of new services and integrations, reducing development time and costs.
* **Security and Control**: Azure API Management offers advanced security features, allowing the organization to maintain control over who accesses the legacy systems and how they do so.
* **Developer Experience**: Developers have a user-friendly and consistent experience when working with the APIs, which can lead to increased developer productivity.
* **Scalability**: Azure API Management can scale to handle increased traffic and ensure high availability, critical for organizations with growing API demands.

This scenario highlights how Azure API Management can be a crucial tool in solving the challenge of exposing legacy services in a secure and modern way, supporting an organization's digital transformation efforts and enhancing its API strategy.

Top of Form

1. **What best practices would you recommend for designing APIs and managing them effectively in Azure API Management?**
2. **How do you handle versioning and deprecation of APIs in Azure APIM without disrupting existing consumers?**
3. **What are the cost considerations when using Azure API Management, and how can you optimize costs while delivering value?**

# Publish and protect APIs running on Azure VMware Solution VMs

# Diagram showing an external API Management deployment for Azure VMware Solution

