1. OBJETIVO

Proporcionar los requerimientos necesarios para la publicación de APIs basadas en REST y/o en protocolo SOAP y estándar WSDL, en la plataforma de Azure “API Management” del ICBF, con el fin de realizar intercambio de datos entre sistemas de información.

2. ALCANCE

Este documento proporciona un acercamiento mínimo necesario para que desarrolladores, arquitectos y otros actores técnicos, que requieran intercambiar información mediante la publicación de funcionalidades especificas por medio de APIs, a través de la plataforma API Management de Azure, puedan interoperar sus sistemas mediante una serie de políticas de uso, políticas de seguridad, monitoreo y gestión del ciclo de vida.

A continuación, se definen los escenarios, en los cuales se debe evaluar el uso de API Management:

- Entre aplicaciones internas del ICBF. Ejemplo: Integración SIM - Contratos.
- Para funcionalidades que requieran transformar el protocolo y/o estilo origen (REST a SOAP y SOAP a REST), con el fin que pueda ser interpretado por un cliente en específico.
- Para aplicaciones nativas en la nube.
- Para la creación de flujos de trabajo que requieren la intervención de varias aplicaciones y por ello una orquestación de servicios.

3. DEFINICIONES

API: Siglas de Application Programming Interface, se usa para la publicación y exposición de servicios a determinado usuario o sistema.

API Gateway: Es la puerta de entrada de un API, generalmente se define como un componente donde se encapsula un conjunto de APIs

API Management: Componente de Azure que enmascara y brinda una amplia gama de funcionalidades a APIs publicadas.

APIM: Sigla de API Management.

GET: Método HTTP usado generalmente en la publicación de servicios por REST. Su uso se da para obtener información.

CLASIFICADA

Policy: Política definida como normas o funcionalidades específicas que siempre se cumplen en el API Management.

POST: Método HTTP usado para publicación de servicios REST. Su uso se da para envío de información.

POSTMAN: Software usado generalmente para la ejecución de métodos REST de HTTP, es de gran utilidad debido a que permite manipular la información enviada (header, body, tipos de envío del mensaje), y almacena toda la información.

REST: Es un estándar de creación de servicios sobre el protocolo HTTP, generalmente su respuesta se da en formato XML y JSON.

Serialization/Deserialization: Es el proceso de convertir el estado de un objeto en bytes para su transporte, la deserialización se refiere a lo contrario, convertir bytes en objetos para su comprensión o entendimiento.

SOAP (Simple Object Access Protocol): Es un protocolo de comunicación de mensajes usado comúnmente y con estándar soportado por la W3C, este protocolo agrega capas estándar de seguridad.

SWAGGER: Framework que contiene herramientas de soporte a desarrolladores para consumir, diseñar, construir y documentar servicios REST.

4. DESARROLLO

A continuación, se realiza una definición sobre lo que es un api management system, así como sus componentes básicos, las diferentes políticas de uso y de seguridad, características de publicación, monitoreo, ciclo de vida de las apis así como una serie de ejemplos de su uso.

4.1. INTRODUCCIÓN

El APIM es el servicio de “Api Management System” proporcionado por la plataforma de nube publica Azure en la modalidad de PaaS. Este componente está conformado en tres subcomponentes principales, como son:

- **Intercambiador de APIs:** Componente cuya principal función es la de habilitar la interconexión entre los servicios y los consumidores, a través de las APIs publicadas en él.
- **Gestor de APIs:** Componente cuya principal responsabilidad es la de ofrecer a los proveedores capacidades de alta configuración y publicación de sus APIs en el componente de intercambiador de apis.

CLASIFICADA

- **Portal de APIs:** Componente dedicado a recopilar toda la información necesaria para los consumidores sobre las APIs publicadas en el intercambiador de APIs.

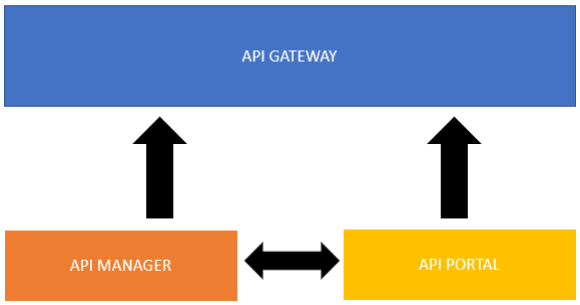


Figura 1. Arquitectura Alto Nivel Api Management System.

El APIM, mediante la unión de estos tres componentes macro proporciona un gran apoyo en la publicación de APIs, brindando políticas de publicación, control de acceso, recopilando y analizando estadísticas de uso, e informando el rendimiento de estas APIs. Proporciona además mayor seguridad ya que enmascara en una sola dirección IP o dominio todas las APIs.

4.2. PARTICIPANTES Y RESPONSABILIDADES

A continuación, se describen los roles básicos que intervienen, en el proceso de creación y publicación de los componentes técnicos para implementación de apis a través de la plataforma APIM:

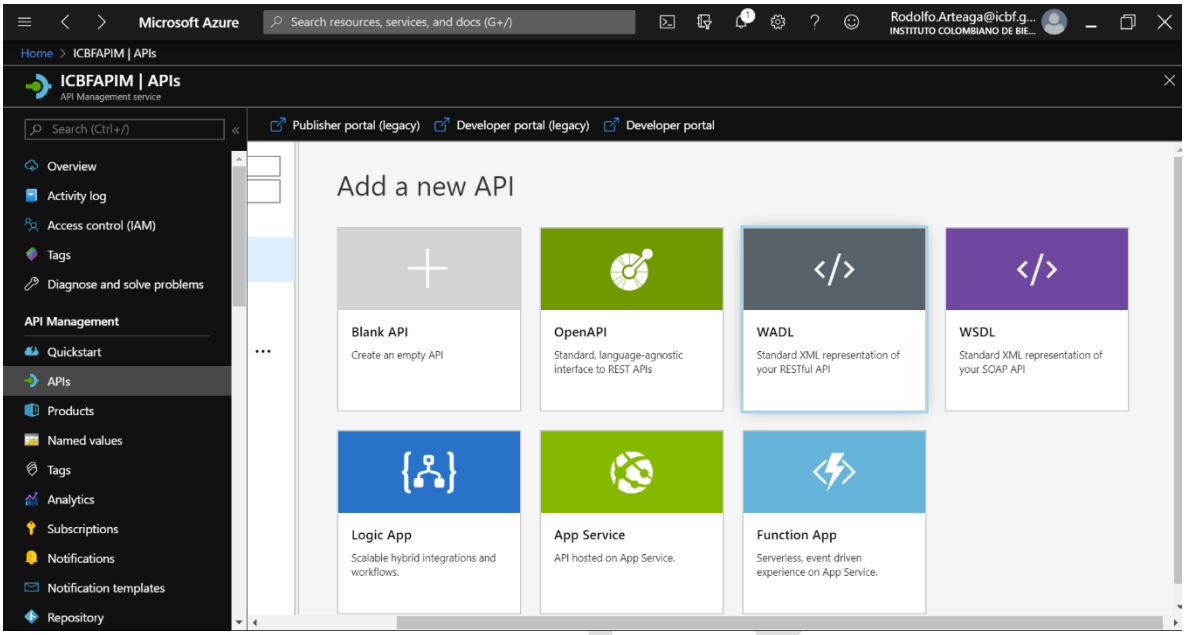
- **Administrador de la plataforma.** Persona encargada del aprovisionamiento de los recursos necesarios para implementar el APIM, así como el encargado de la gestión de permisos y el monitoreo de la plataforma.
- **Arquitecto.** Persona encargada de la definición de las políticas de uso y seguridad, así como de proponer y evaluar las estrategias de publicación de apis sobre la plataforma.
- **Desarrollador.** Persona encargada de la creación de las apis y su correspondiente publicación en la plataforma.

4.3. ESQUEMA DE PUBLICACIÓN DE API

Para publicar una API a través del API Management pueden seguirse los siguientes pasos.

4.3.1. Crear y probar el desarrollo de APIs. Explorar desde el portal de Azure las diferentes opciones de publicación.

CLASIFICADA



4.3.2. Test. Es recomendable realizar pruebas sobre la API publicada, ya sea por POSTMAN o directamente en el API Management.

1. Establecer políticas sobre las APIs publicadas
   a. Directivas básicas
      Políticas sobre el header de la solicitud.
      Restricción de acceso por IP.
      Limitar frecuencia de llamadas.
      Validación por JWT.
   b. Directivas avanzadas.
      Registro en log.
      Respuesta Mock.
      Política de reintentos.
   c. Directivas de autenticación.
      Autenticación básica.
      Autenticación con certificado del cliente.
      Autenticación basada en identidad.
   d. Políticas de caché.
      Obtener (valor) de caché.
      Almacenar (valor) en caché.
      Quitar (valor) de caché.
   e. Directivas entre dominios.
      CORS.

CLASIFICADA

JSONP.

f. Directivas de transformación.
   Conversión de JSON a XML o viceversa.
   Reemplazo de cadenas en el cuerpo.
   Enmascarar URL en el contenido.
   Enrutamiento de servicio.
   Agrega información al cuerpo del mensaje entrante o saliente.
   Transformación de XML mediante XLST.

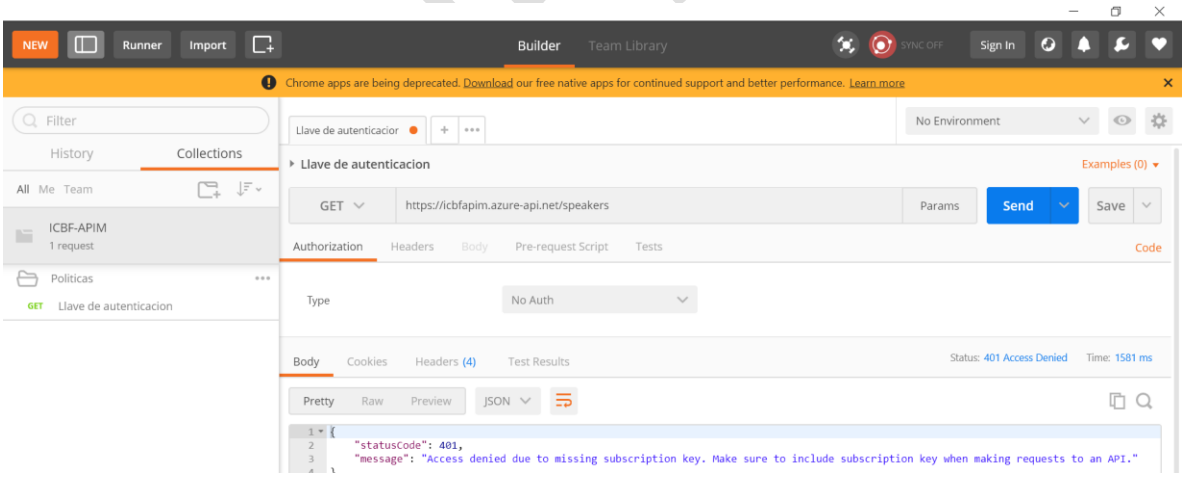
2. Versionamiento de APIs publicadas.
3. Monitoreo de APIs publicadas.

Estas políticas fueron tomadas de: https://docs.microsoft.com/es-es/azure/api-management/api-management-policies

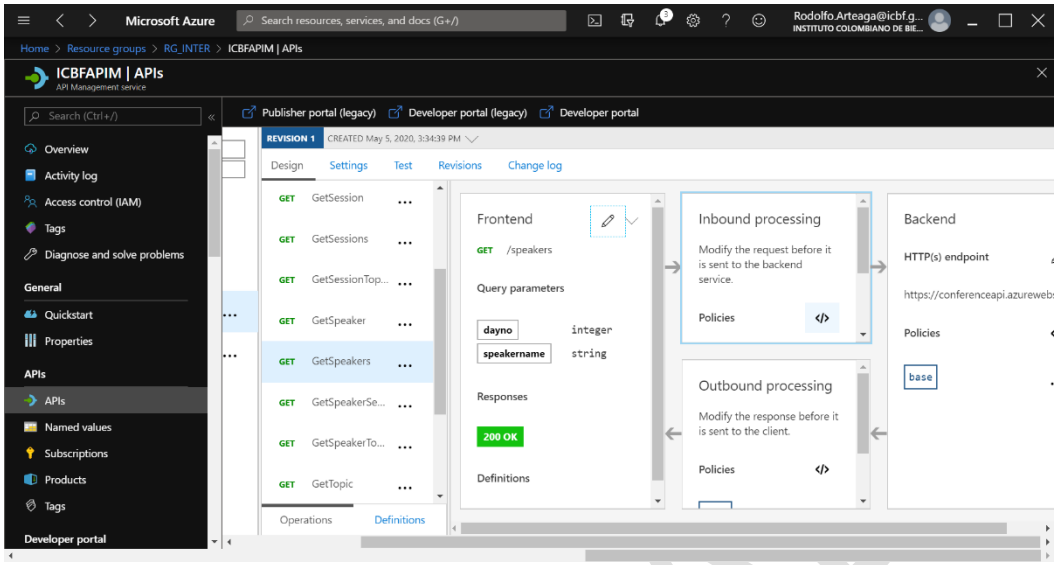
4.4. EJEMPLO DE IMPLEMENTACIÓN DE API EN API MANAGEMENT

Para cada una de las políticas establecidas en la parte superior, se decide realizar una prueba de su uso e implementación, se basa en un ejemplo que se tomó de Microsoft, siguiendo el enlace siguiente: https://docs.microsoft.com/es-es/azure/api-management/import-and-publish

Acceso a método sin llave de autenticación.



*Para acceder a las políticas de entrada del servicio o del método necesario, puede accederse a través de Azure, en la descripción del método, y haciendo clic en el botón “</>” para editar las políticas de procesamiento de entrada, como se muestra a continuación:



Directivas básicas.

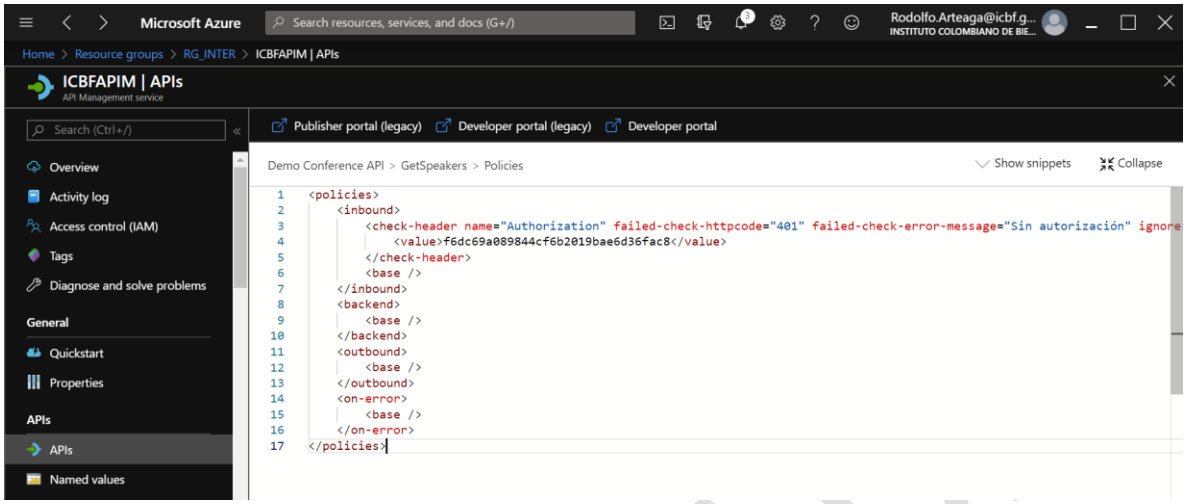
Política sobre el header de la solicitud.

Además de la llave de autenticación para la suscripción al APIM, es posible indicar todas las llaves adicionales necesarias para el uso de los servicios publicados, y de ellos devolver los mensajes y códigos de error que se requieran. Esta política agrega mayor seguridad a la ejecución de servicios.

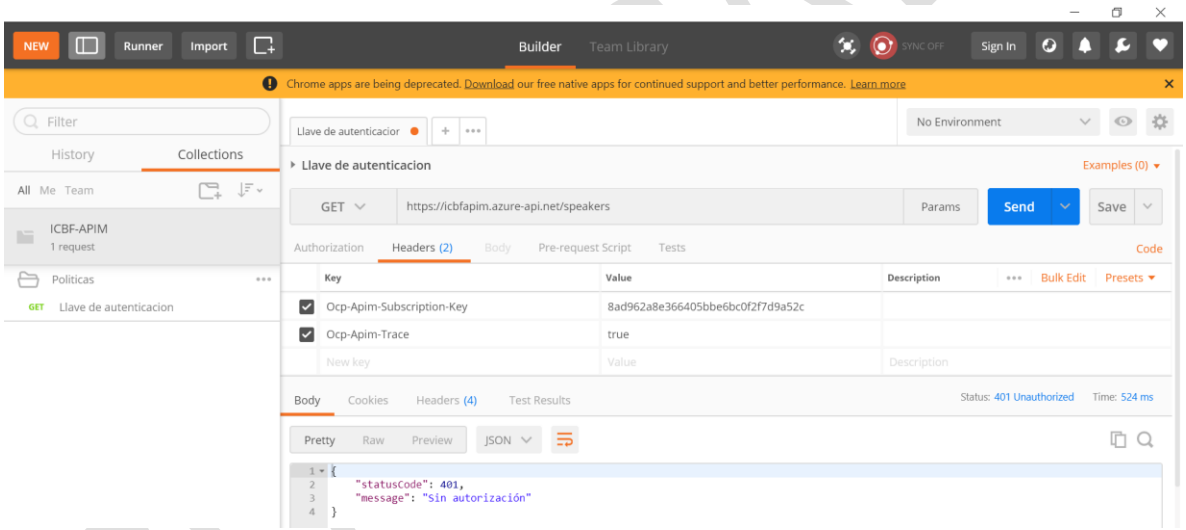
Para incluir esta política es necesario agregar el siguiente código dentro de la política de entrada del APIM.

```
<check-header name="Authorization" failed-check-httpcode="401" failed-check-error-message="Sin autorización" ignore-case="false"><value>f6dc69a089844cf6b2019bae6d36fac8</value></check-header>
```

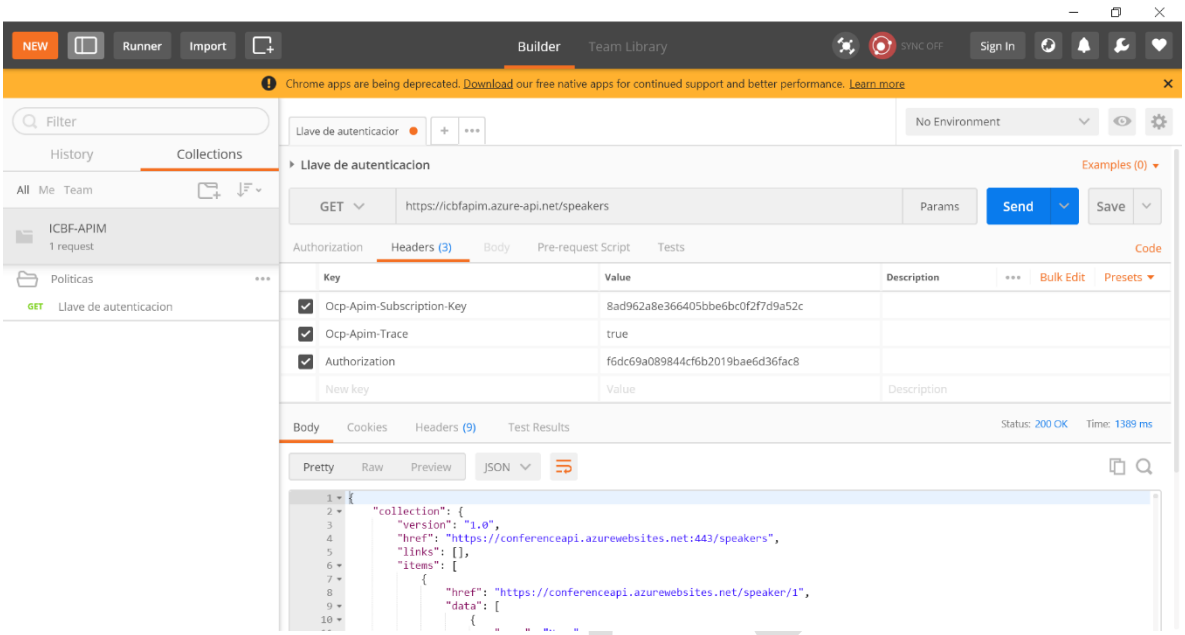
CLASIFICADA



Sin incluir esta llave la respuesta de la invocación del servicio es la siguiente.



Incluyendo la llave agregada a la política, la respuesta a la invocación al servicio es correcta.

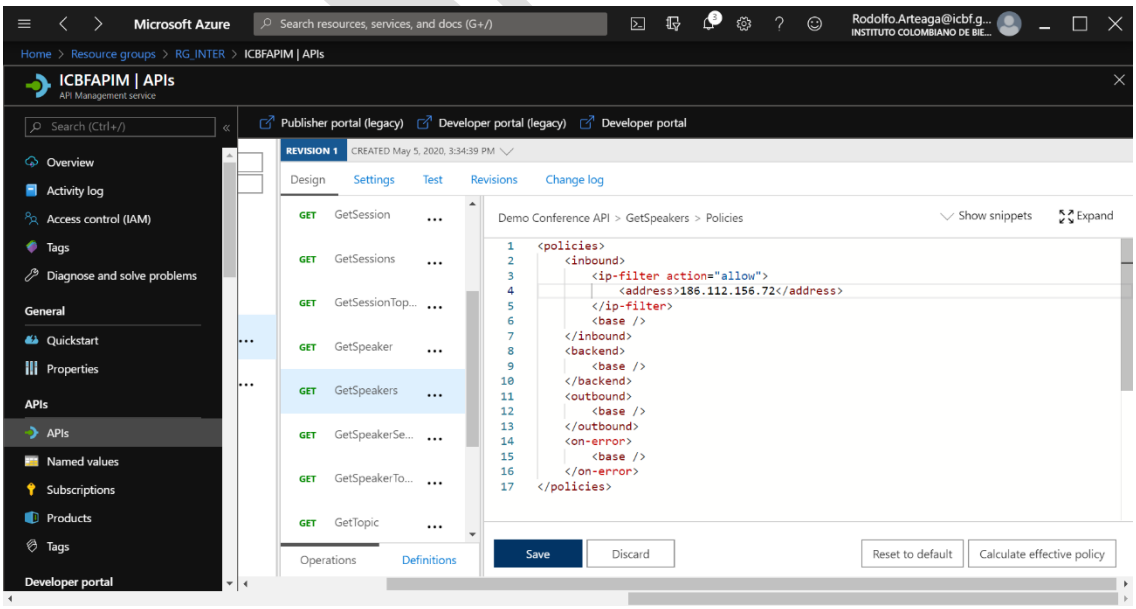


Política de restricción de acceso por IP.

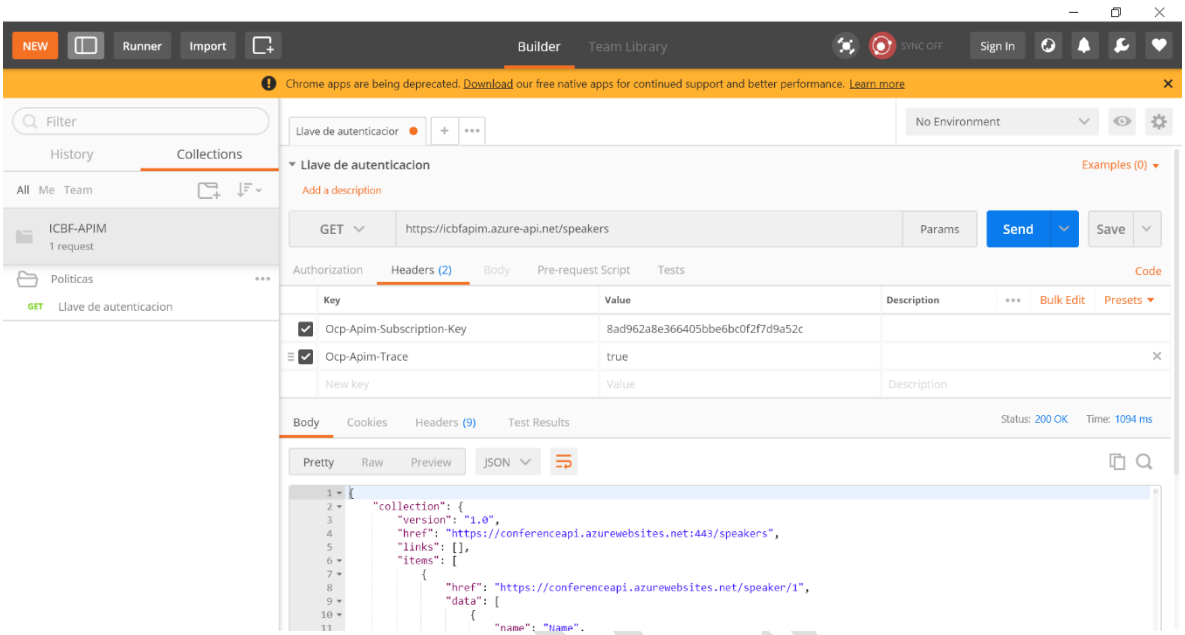
Es posible establecer a quien se le permite o niega el acceso a un servicio o método determinado mediante su dirección IP o un rango de IP al que pertenezca.

Para agregar esta política es necesario agregar el siguiente fragmento XML:

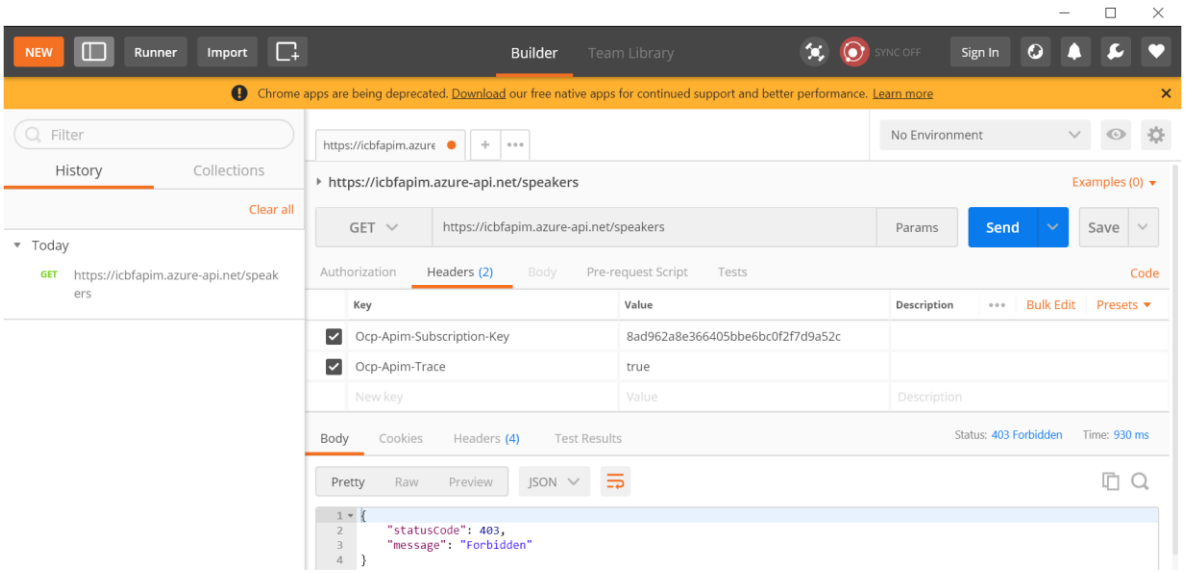
<ip-filter action="allow"><address>186.112.156.72</address></ip-filter>



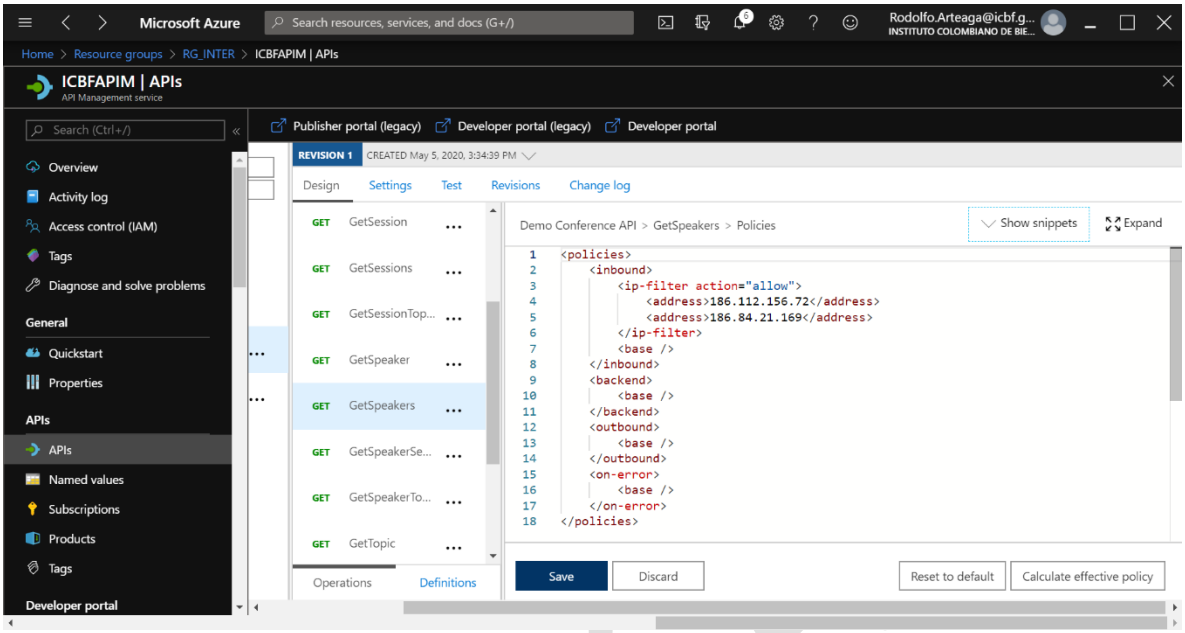
Y al ejecutar el servicio desde la IP proporcionada, su ejecución es correcta.



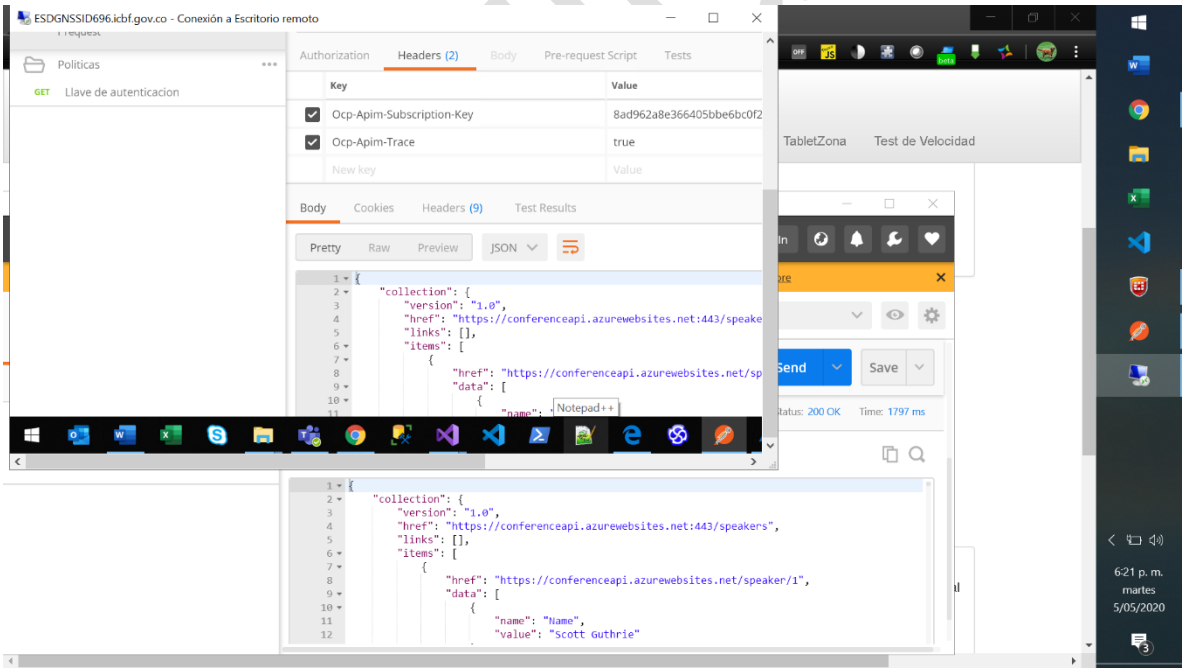
De ser ejecutado el servicio desde otra dirección ocurre lo siguiente:



También es posible agregar varias direcciones IP a la política.



Lo cual permite la ejecución del servicio desde las dos Ips indicadas.



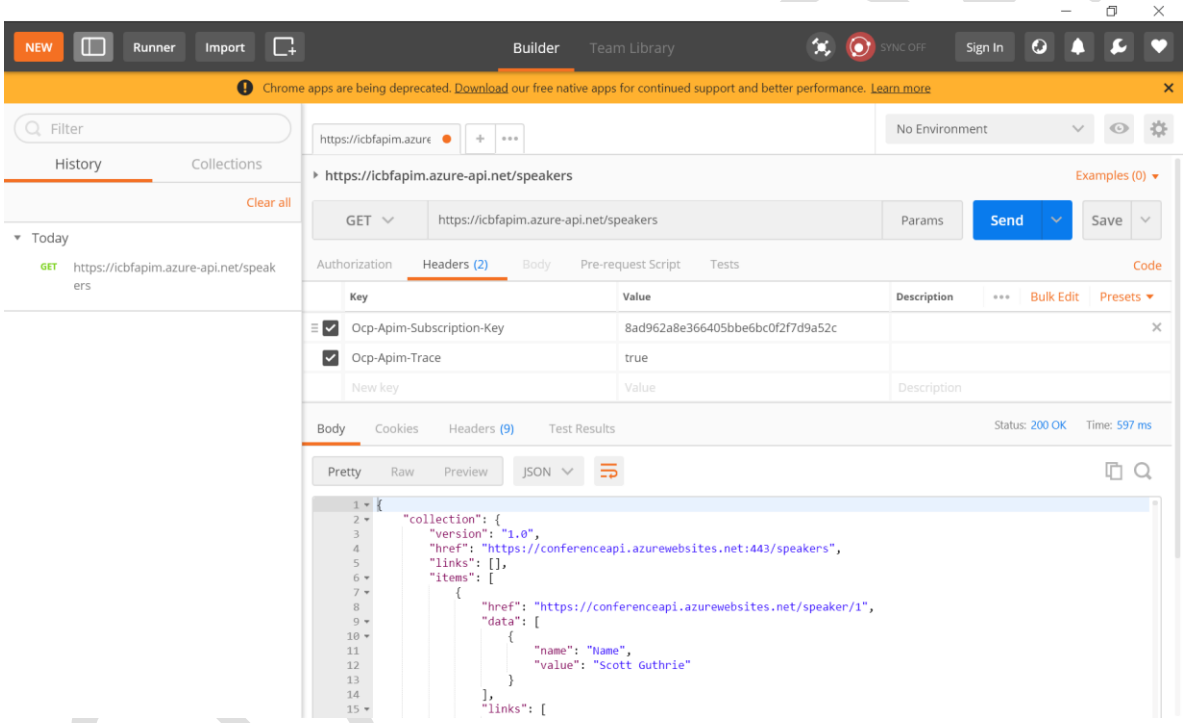
Para establecer un rango de Ips desde el cual puede ejecutarse el servicio debe agregarse por ejemplo el siguiente segmento XML dentro del nodo ip-filter.

<address-range from="13.66.140.128" to="13.66.140.143" />**Limitar por frecuencia de llamadas.**

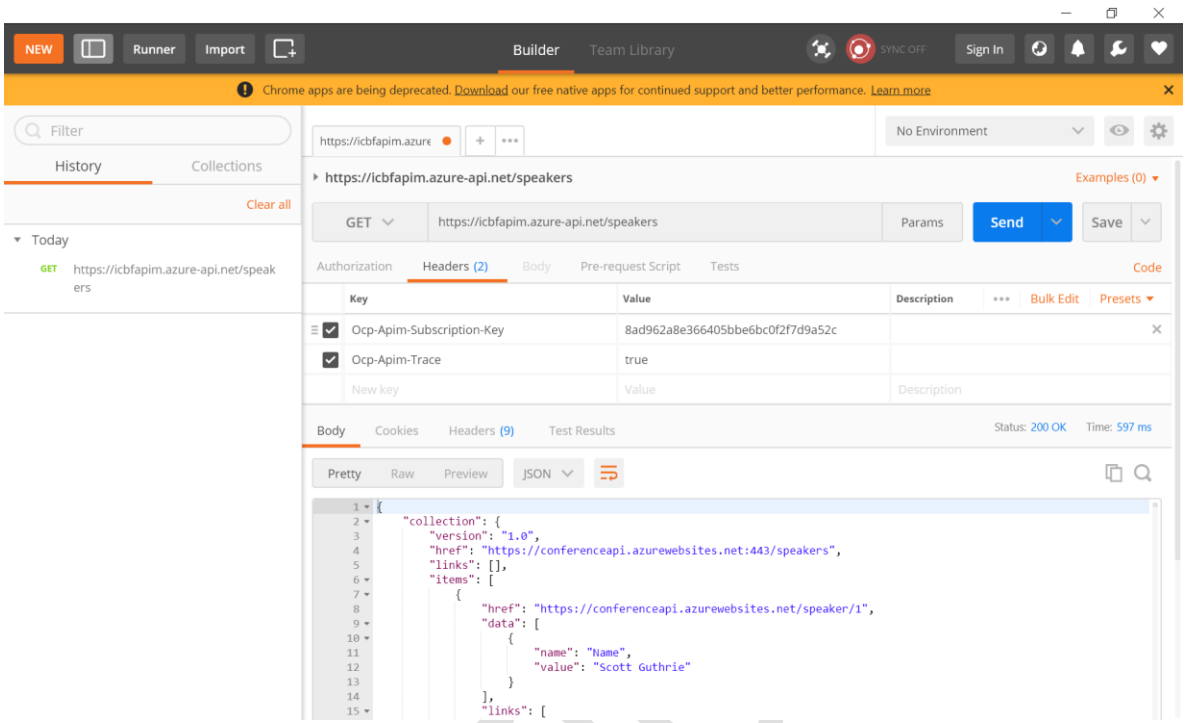
Es posible limitar el uso del API por llamadas y periodo de tiempo, es útil esta política para restringir la carga sobre las APIs desarrolladas. Para implementar esta API, puede hacerse como se muestra en el ejemplo siguiente:

```
<rate-limit calls="5" renewal-period="60" />
```

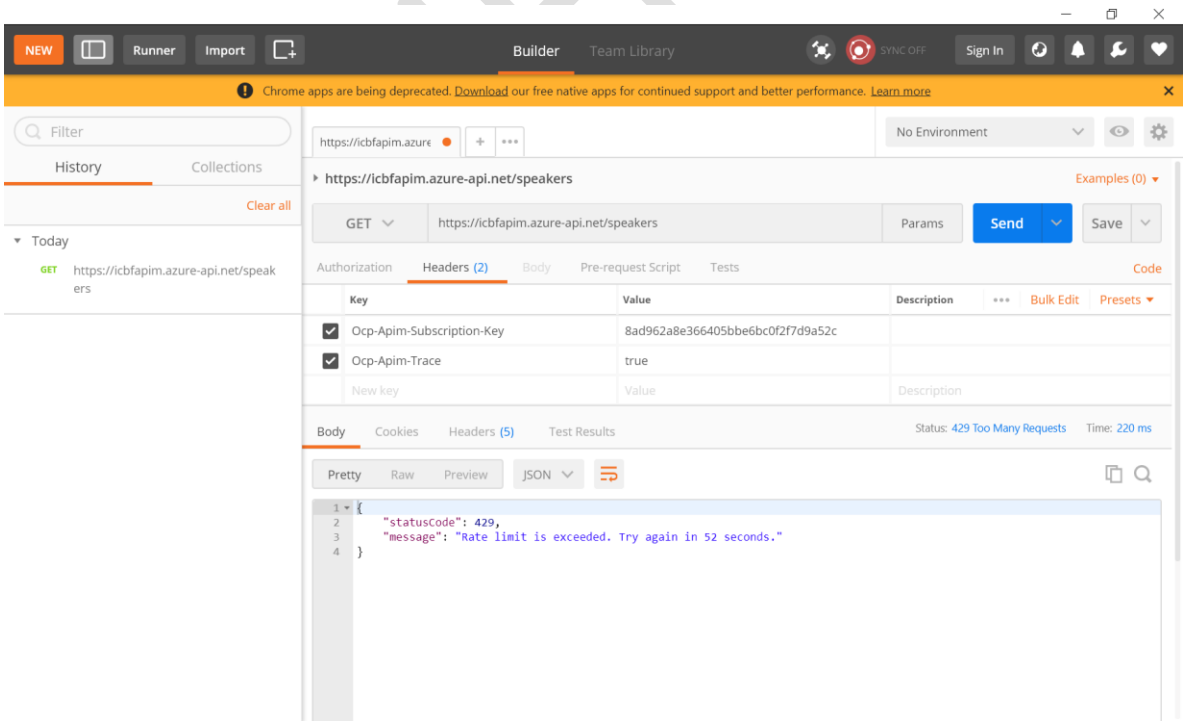
Donde el rate-limit calls indica el número de peticiones, y el renewal-period el tiempo en segundos.



Que al ejecutar más de 5 veces sin ningún problema el servicio responde.



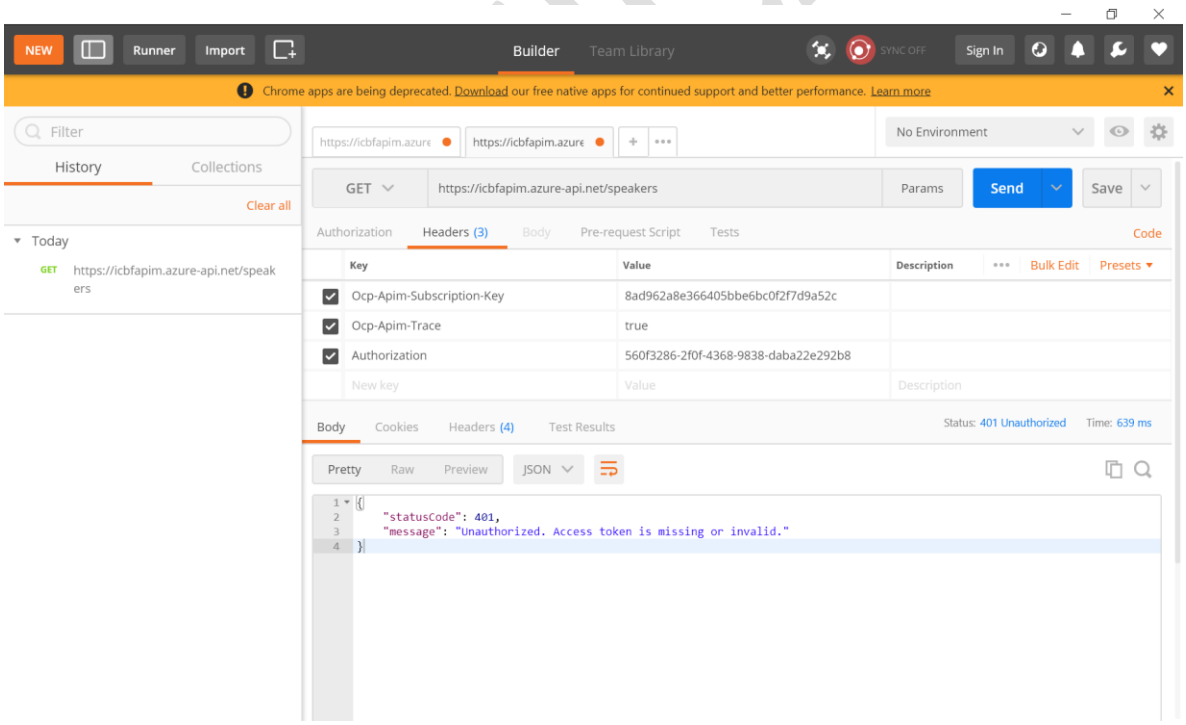
Al aplicar la política se presenta el siguiente mensaje al ejecutar el API más de 5 veces:



Validación de tokens JWT.

Es posible validar un token en específico contra una autoridad determinada. Además de ello, es posible validar la expiración de estos. Para usar un token en el servicio se debe usar por ejemplo de la siguiente forma.

```
<validate-jwt header-name="Authorization" failed-validation-httpcode="401" failed-validation-error-message="Unauthorized. Access token is missing or invalid.">
    <openid-config url="https://login.microsoftonline.com/contoso.onmicrosoft.com/.well-known/openid-configuration" />
    <audiences>
        <audience>25eef6e4-c905-4ª07-8eb4-0d08d5df8b3f</audience>
    </audiences>
    <required-claims>
        <claim name="id" match="all">
            <value>insert claim here</value>
        </claim>
    </required-claims>
</validate-jwt>
```



Registro de log a EventHub.

Además del registro que hace el monitoreo, es posible registrar manualmente los eventos de ejecución del servicio a un EventHub, para ello se debe incluir, por ejemplo, la siguiente estructura:

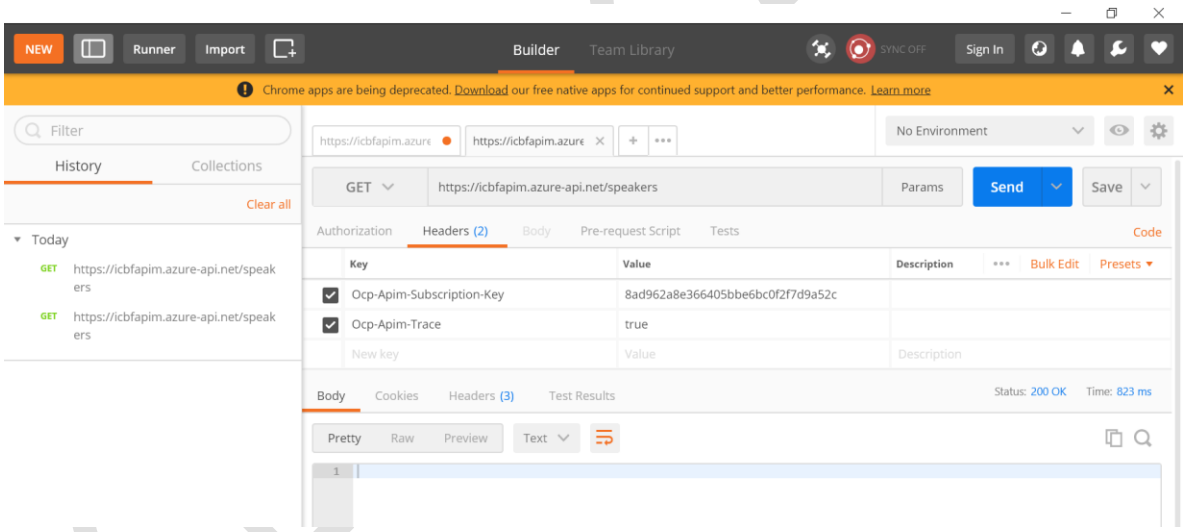
```
<log-to-eventhub logger-id ='contoso-logger'>
    @( string.Join(",", DateTime.UtcNow, context.Deployment.ServiceName,
context.RequestId, context.Request.IpAddress, context.Operation.Name) )
  </log-to-eventhub>
```

Respuesta Mock

Se usa comúnmente en el desarrollo de un componente que requiere una respuesta simulada. Su estructura es muy sencilla, ya que devuelve únicamente el código HTTP y el esquema indicado.

```
<mock-response status-code='200' content-type='application/json'/>
```

Devuelve la siguiente respuesta:

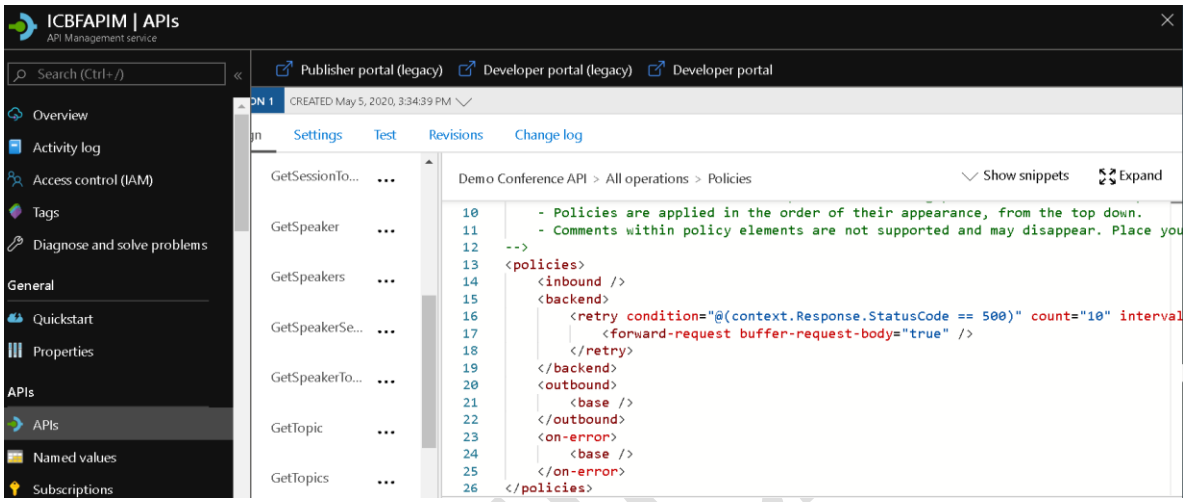


Política de reintentos.

El API Management realiza un reintento de devolución mientras la respuesta sea dada una condición en específico. Para el siguiente ejemplo se muestra una política de reintentos de hasta 10 veces mientras la respuesta sea un código HTTP 500.

```
<retry
  condition="@(context.Response.StatusCode == 500)"
  count="10"
  interval="10"
```

```
  max-interval="100"
  delta="10"
  first-fast-retry="false">
    <forward-request buffer-request-body="true" />
</retry>
```



Autenticación básica

Establece un usuario y contraseña predeterminado para usar un API de backend conectado al API Management. Por ende, la invocación del servicio no requiere dicho usuario y contraseña, sin embargo el API Management agrega al header de la solicitud del backend username y password. Para su uso debe incluirse el siguiente segmento XML:

```
<authentication-basic username="testuser" password="testpassword" />
```

Autenticación con certificado.

Realiza la autenticación contra un servicio del backend agregando además un certificado digital, el certificado digital debe ser instalado anteriormente en el API Management siguiendo la siguiente guía:

https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-mutual-certificates

Debe ser incluido el siguiente fragmento de código XML, para poder realizar la autenticación con el uso de un certificado digital.

```
<authentication-certificate
thumbprint="CA06F56B258B7A0D4F2B05470939478651151984" />
```

Con id del certificado.

<authentication-certificate certificate-id=”544fe9ddf3b8f30fb490d90f” />

Autenticación basada en identidad.

Es posible usar esta política para delegar la autenticación, recibiendo un token de acceso enviado por Azure Active Directory. Un ejemplo para el manejo de identidad es el siguiente:

<authentication-managed-identity resource=” http://icbfgob.onmicrosoft.com “/>

Para las siguientes políticas de cache, se realizarán pruebas inicialmente en almacenamiento y posteriormente en obtención de información.

Obtener (valor) de caché.

Es posible obtener un valor previamente almacenado en caché, usando la siguiente directiva.

```
<cache-lookup-value
   key=”@(“userprofile-“ + context.Variables[“enduserid”])”
   variable-name=”userprofile” />
```

Almacenar (valor) en caché.

Esta política se revisó en base al siguiente artículo: https://docs.microsoft.com/es-mx/azure/api-management/api-management-sample-cache-by-key

Indica que es posible el almacenamiento en caché de respuestas que permanecen estáticas durante un periodo de tiempo, esto reduce el consumo de ancho de banda y el procesamiento en backend, además de latencia de respuesta.

Este servicio proporciona varias opciones, como:

a. Almacenar en caché dependiendo quién la esté invocando.
b. Almacenar en caché dependiendo de un grupo de quienes invocan el servicio.
c. Almacenar en caché dependiendo de los parámetros.
d. Duración de tiempo en segundos que será almacenada una petición en caché.

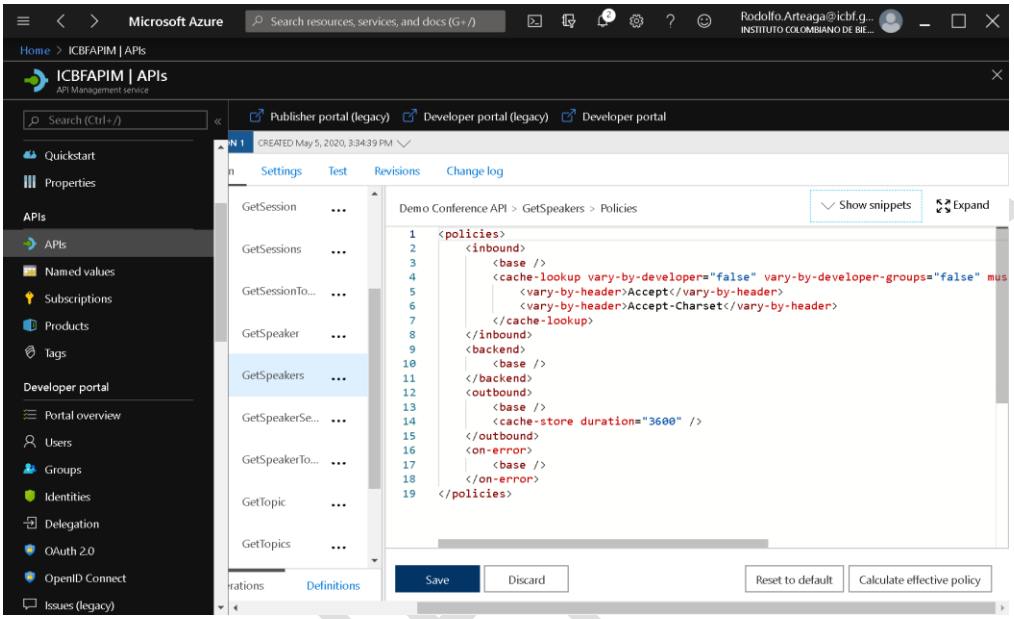
También es posible definir una expresión que indique el tiempo que va a durar una petición en caché, dependiendo en este caso de la respuesta “cache-control” del backend, como se muestra a continuación:

```
<cache-store duration=”@{
   var header = context.Response.Headers.GetValueOrDefault(“Cache-Control”,””);
   var maxAge = Regex.Match(header, @”max-
age=(¿<maxAge>\d+)”).Groups[“maxAge”]?.Value;
   return (¡string.IsNullOrEmpty(maxAge))?int.Parse(maxAge):300;
```


CLASIFICADA

```
}"
/>
```

En el siguiente ejemplo de implementación se muestra el almacenamiento en caché por 1 minuto:



Quitar (valor) de caché.

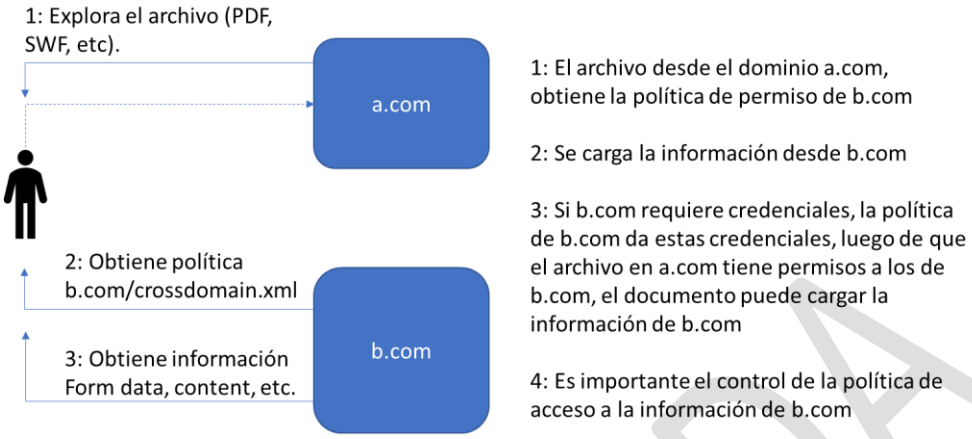
Para eliminar un valor del caché del APIM, se debe usar la siguiente instrucción:

```
<cache-remove-value key="@("userprofile-" + context.Variables["enduserid"])"/>
```

Directivas entre dominios.

CORS.

La directiva CORS o "Cross Domain", permite el acceso al API desde diferentes clientes o dominios, un flujo típico de CORS se refiere a un redireccionamiento, por ejemplo, un usuario trata de cargar un archivo desde un dominio "a.com", y este trata de cargar información desde otro dominio "b.com", esta política de permiso de un dominio a otro se hace a través de un archivo "crossdomain.xml". Para aclarar un poco más este flujo del proceso se puede ver en el siguiente diagrama:



Para el API Management, es importante establecer que políticas, y desde que orígenes es posible cargar la información, para ello en el siguiente ejemplo, se muestra la manera en cómo puede realizarse:

```
<cors allow-credentials="true">
  <allowed-origins>
    <!—Localhost useful for development →
    <origin>http://localhost:8080/</origin>
    <origin>http://example.com/</origin>
  </allowed-origins>
  <allowed-methods preflight-result-max-age="300">
    <method>GET</method>
    <method>POST</method>
    <method>PATCH</method>
    <method>DELETE</method>
  </allowed-methods>
  <allowed-headers>
    <!—Examples below show Azure Mobile Services headers →
    <header>x-zumo-installation-id</header>
    <header>x-zumo-application</header>
    <header>x-zumo-version</header>
    <header>x-zumo-auth</header>
    <header>content-type</header>
    <header>accept</header>
  </allowed-headers>
  <expose-headers>
    <!—Examples below show Azure Mobile Services headers →
    <header>x-zumo-installation-id</header>
    <header>x-zumo-application</header>
  </expose-headers>
```

ASFICADA

</cors>

Esto brinda el control desde otros dominios de acceso al API.

JSONP.

Es un método utilizado en Javascript para permitir llamadas entre dominios desde otro servidor. Esta instrucción agrega compatibilidad con JSONP que agrega relleno a la instrucción de Script entre varios servidores, por ejemplo:

```
<script type="application/javascript"
    src="http://server.example.com/Users/1234">
</script>
```

Directivas de transformación.

Conversión de JSON a XML y viceversa.

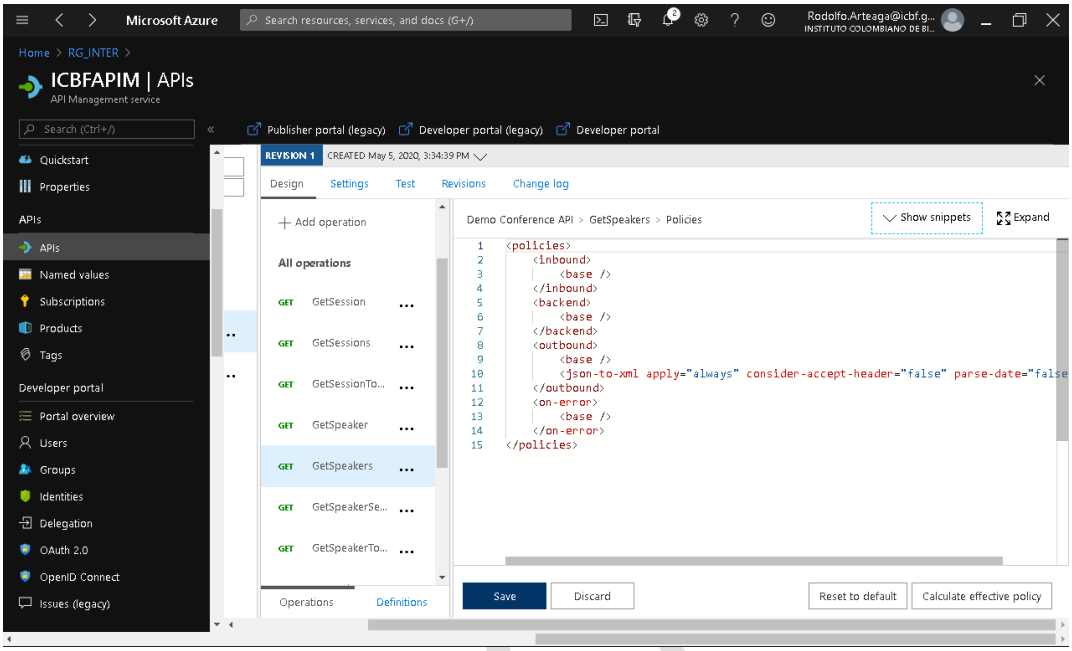
Tal como se indica realiza la conversión de JSON a XML utilizando la siguiente instrucción:

<json-to-xml apply="always | content-type-json" consider-accept-header="true | false" parse-date="true | false"/>

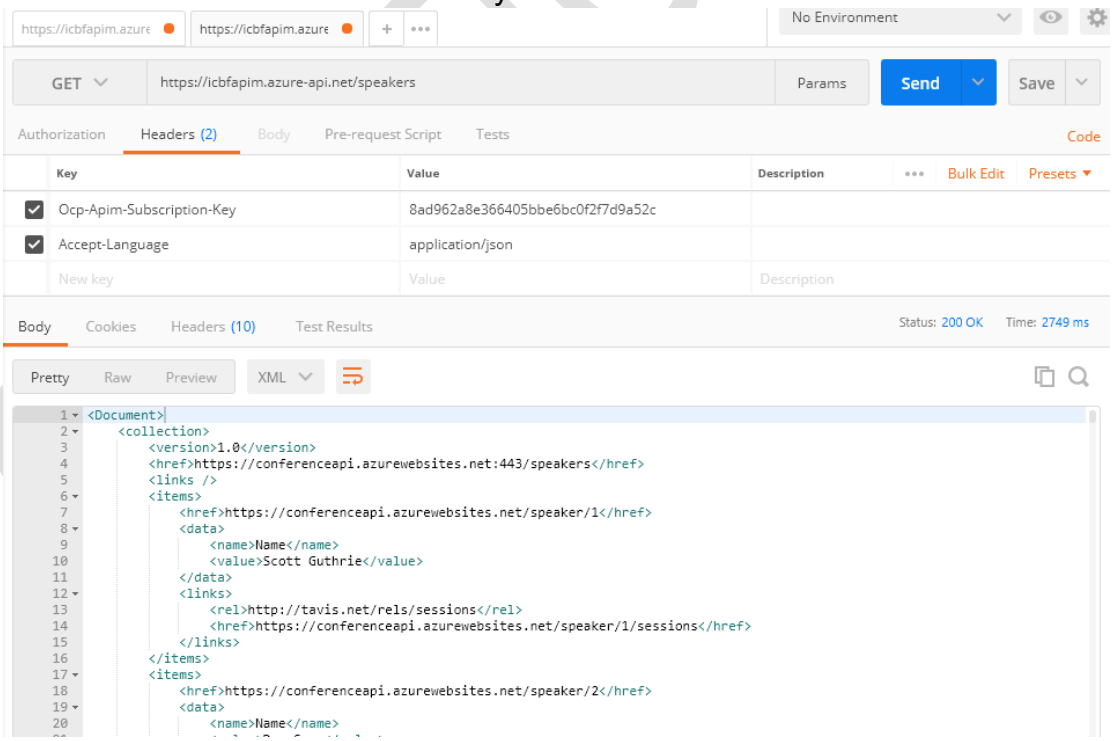
Se puede realizar la prueba de conversión utilizando el ejemplo anterior:

a. Agregar la política al APIM.

CLASIFICADA



b. Ejecutar desde POSTMAN el servicio y se evidencia la devolución en formato XML:



Exponer la respuesta con conversión de XML a JSON, funciona de manera similar usando la siguiente instrucción.

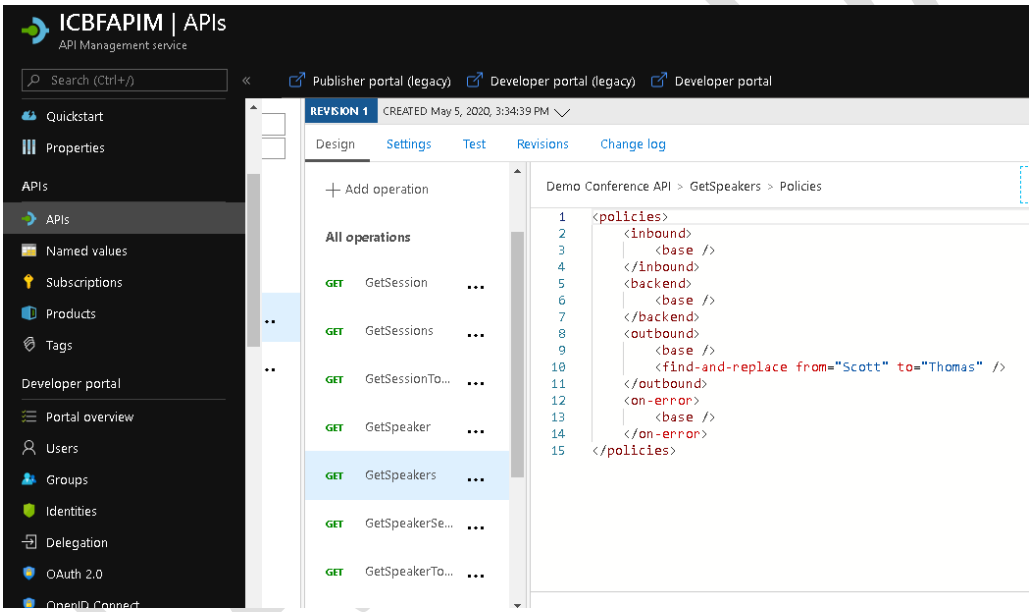
<xml-to-json kind=”javascript-friendly | direct” apply=”always | content-type-xml” consider-accept-header=”true | false”/>

Reemplazo de cadenas en el cuerpo.

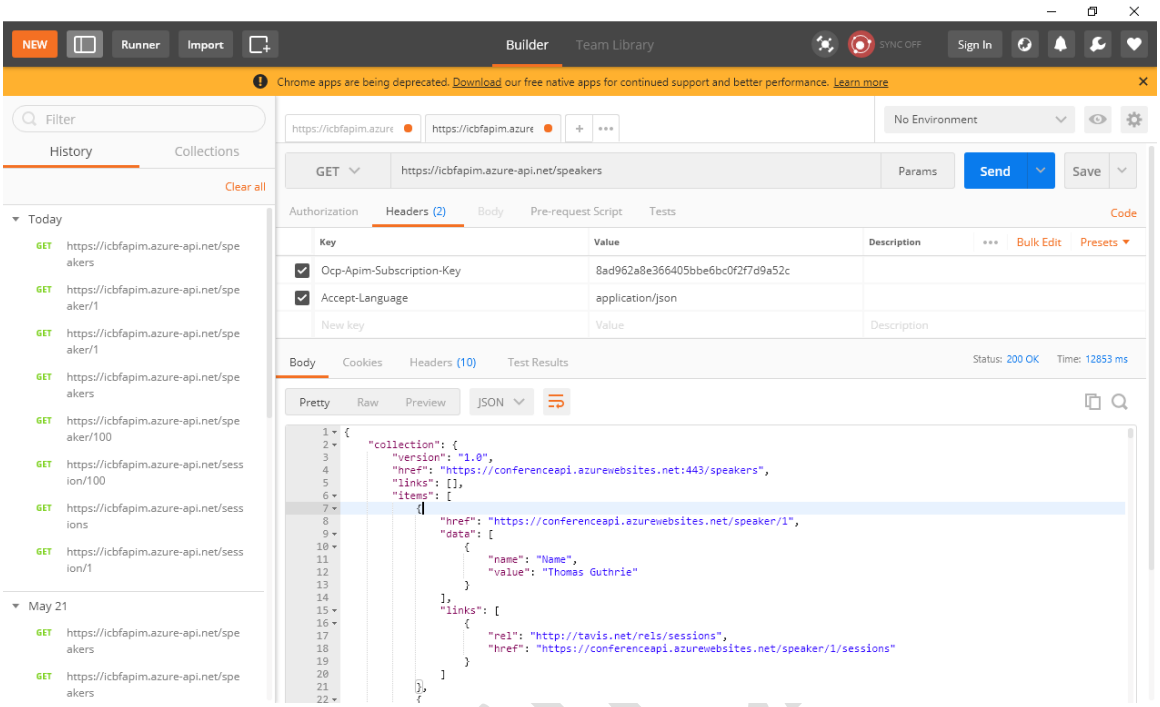
De una forma sencilla, es posible reemplazar cadenas de texto en la respuesta (XML o JSON) usando por ejemplo la siguiente instrucción:

<find-and-replace from=”what to replace” to=”replacement” />

Tomamos el ejemplo anterior y reemplazamos la cadena “Scott” por “Thomas”.



Luego se ejecuta por POSTMAN el servicio anterior. Se visualiza el reemplazo en la respuesta del primer participante.



Enmascarar URL en el contenido.

Esta directiva sirve como Gateway indicando el enmascaramiento de la dirección del contenido, para usarse simplemente se debe agregar la siguiente instrucción a las políticas del APIM:

```
<redirect-content-urls />
```

Enrutamiento de servicio.

Esta directiva indica quien da respuesta a las solicitudes entrantes, y además de ello puede establecer enrutamiento a diferentes servicios de Backend dependiendo de los parámetros entrantes, a continuación, se muestra un ejemplo de su uso:

```
<policies>
  <inbound>
    <choose>
      <when condition="@(context.Request.Url.Query.GetValueOrDefault("22ersión") == "2013-05")">
        <set-backend-service base-url="http://contoso.com/api/8.2/" />
      </when>
      <when condition="@(context.Request.Url.Query.GetValueOrDefault("22ersión") == "2014-03")">
        <set-backend-service base-url="http://contoso.com/api/9.1/" />
```

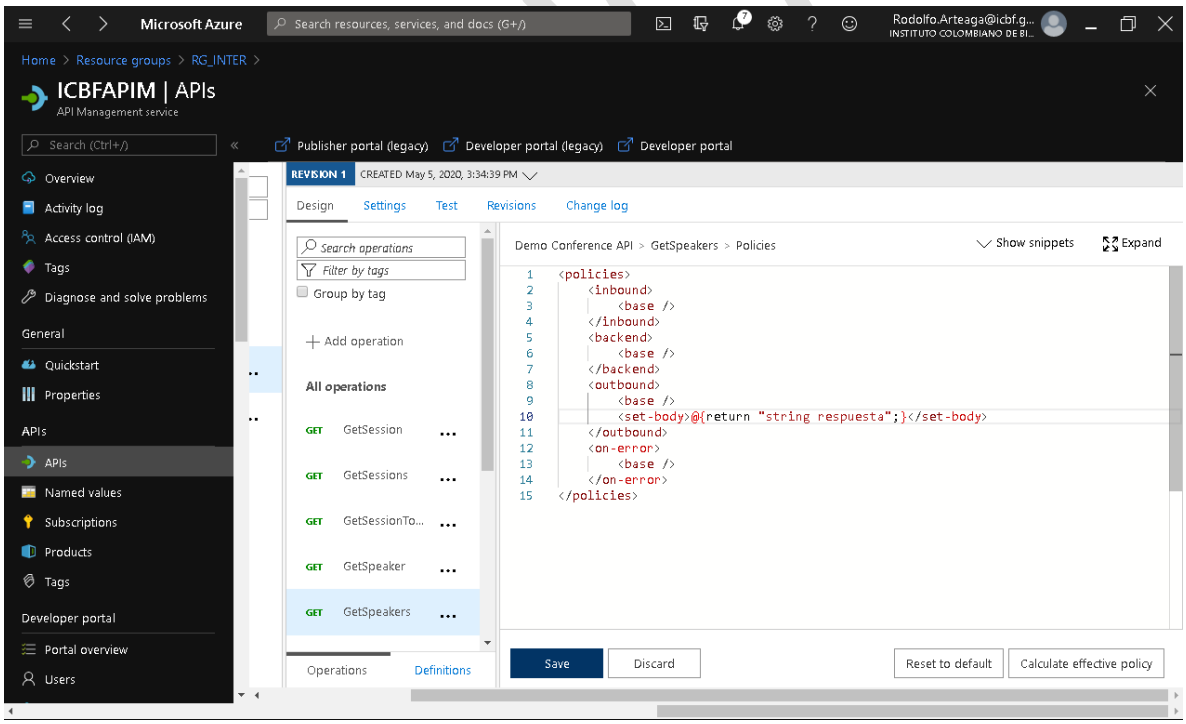
```
        </when>
      </choose>
      <base />
    </inbound>
    <outbound>
      <base />
    </outbound>
</policies>
```

Agrega información al cuerpo del mensaje entrante o saliente.

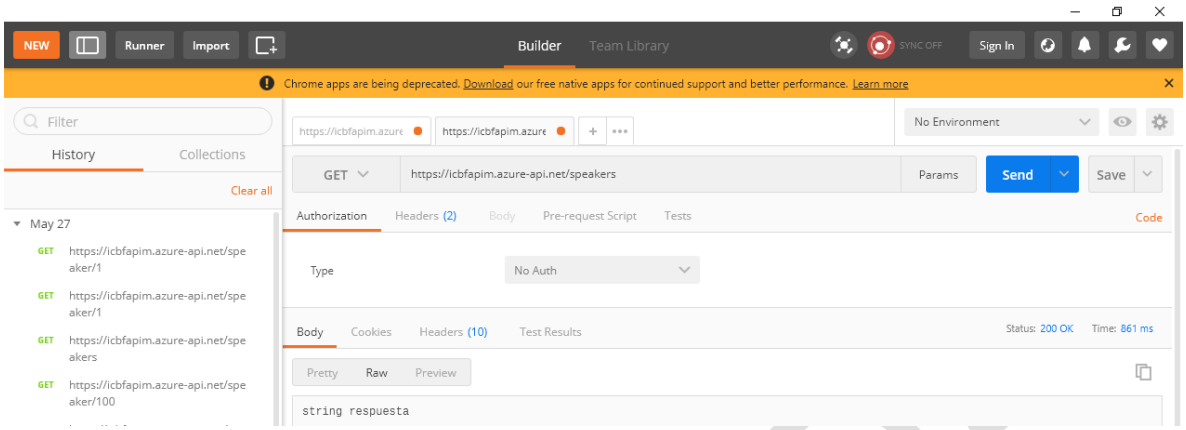
Establece el cuerpo de respuesta, puede ser usado para respuestas Mock o para dar alguna respuesta que un servicio de Backend no devuelve. Para usar esta instrucción se debe usar el siguiente script:

```
<set-body>new body value as text</set-body>
```

El siguiente ejemplo da una respuesta a un código en el API Management que inclusive puede agregársele código en C#:



Respuesta a invocación.



Transformación de XML mediante XLST

A los servicios que devuelven XML es posible aplicarles una transformación mediante una plantilla XLST, al igual que a las respuestas JSON es posible aplicarles una transformación usando Liquid.

Un ejemplo de aplicación de una transformación de un XML puede ser el siguiente:

```
<xsl-transform>
      <xsl:stylesheet 24ersión=”1.0” xmlns:xsl=”http://www.w3.org/1999/XSL/Transform”>
         <xsl:output omit-xml-declaration=”yes” method=”xml” indent=”yes” />
         <!—Copy all nodes directly→
         <xsl:template match=”node()| @*|*”>
            <xsl:copy>
               <xsl:apply-templates select=”@* | node()|*” />
            </xsl:copy>
         </xsl:template>
      </xsl:stylesheet>
   </xsl-transform>
```

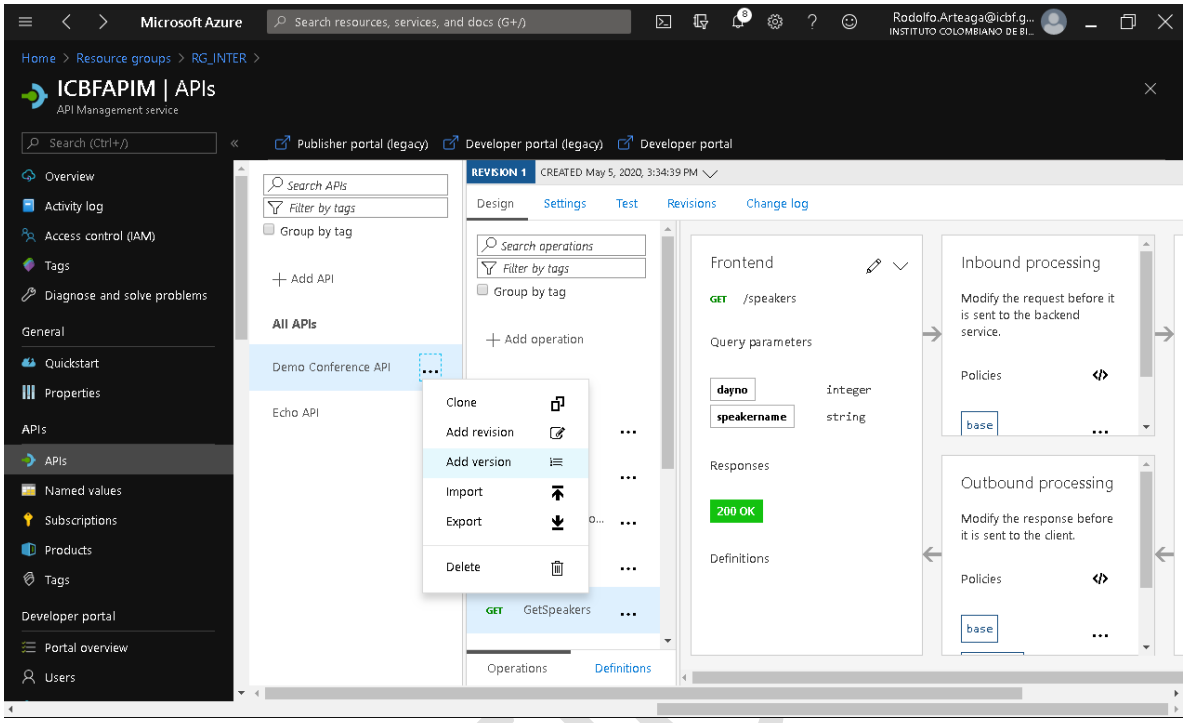
Como se puede evidenciar, todas estas políticas ayudan al API Management al comportamiento similar a un bus de datos, ayudando a enriquecer los mensajes, transformarlos e incluso enrutarlos dependiendo de los parámetros de un servicio.

4.5. VERSIONAMIENTO DE APIS PUBLICADAS

El API Management puede apoyar en el versionamiento de APIs publicadas, de manera sencilla, es posible realizar el versionamiento de la siguiente forma:

En el siguiente ejemplo se visualiza la manera de cómo se hace:

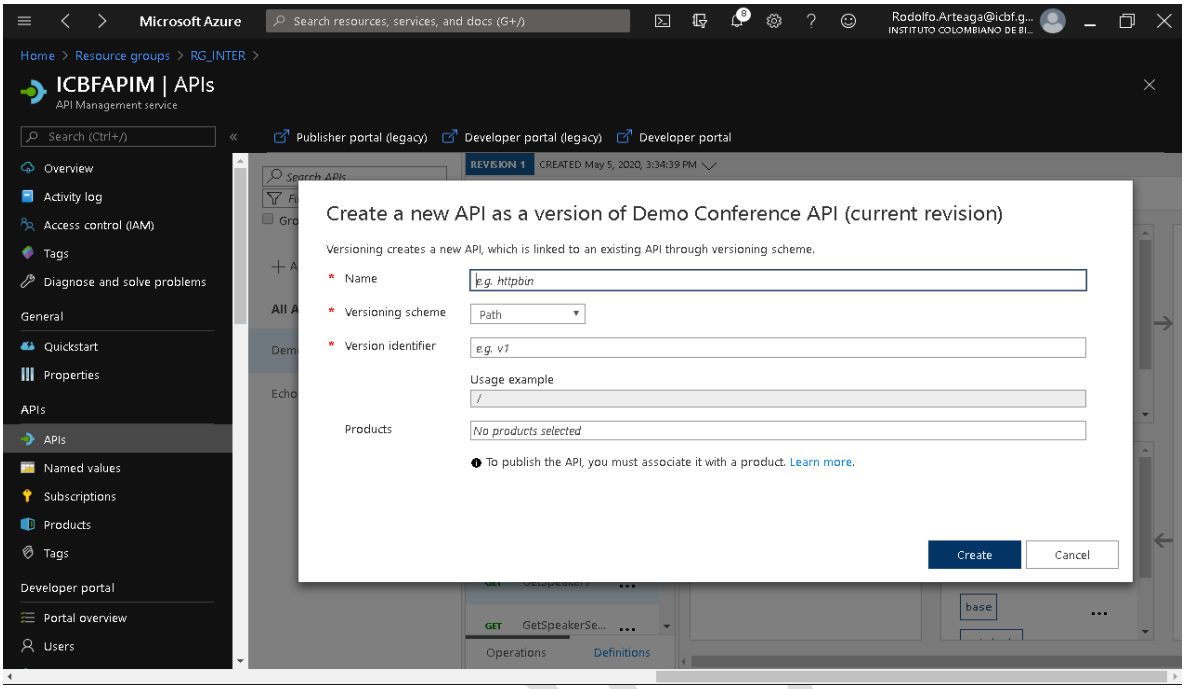
1. Seleccionar la API de prueba que se ha trabajado.



Seleccionar la opción ”Add 25ersión”.

2. Luego de ello se despliega una pantalla donde se puede registrar la siguiente información:

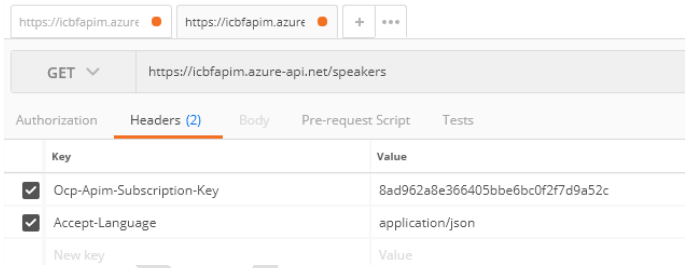
CLASIFICACIÓN



Registrar en el campo name (nombre): demo-conference-api-v1

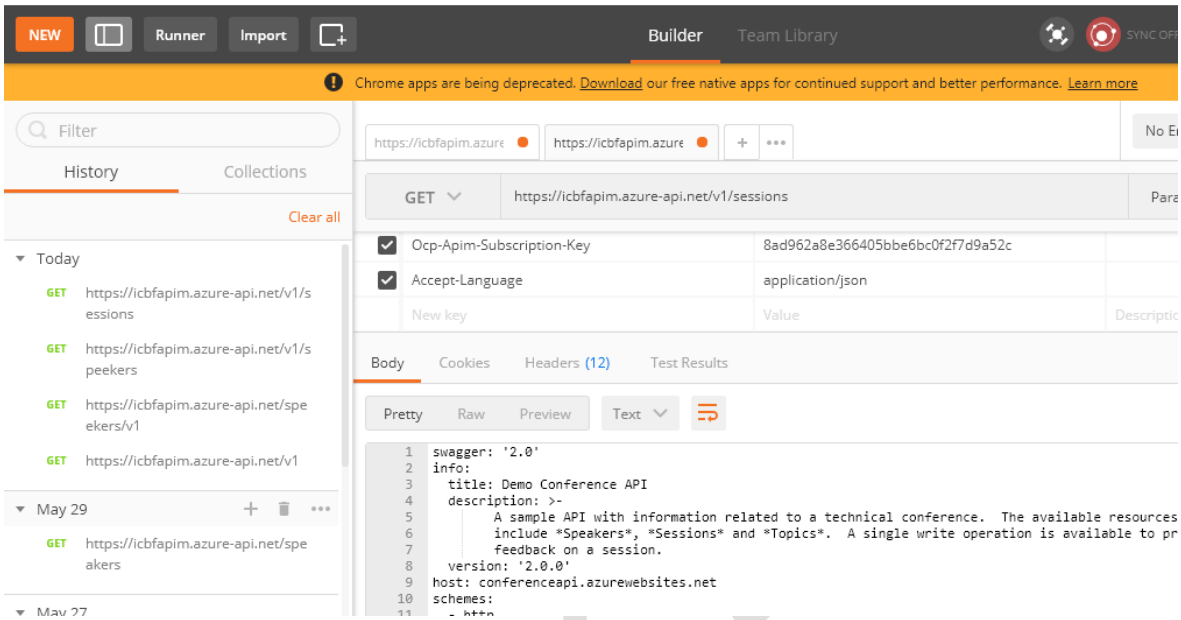
En el campo Versioning scheme (Esquema de versionamiento): Path

*Se puede elegir entre Path, header, o Query String, esto define cómo se versionará el API, por Path severa en la URL del API (algo así: miapi/v1/…), por header, en la invocación será enviada la versión.



Por último, si es por “Query String”, se evidencia en la URL similar a este ejemplo: “miapi/26ersión=v1”.

Finalmente, al crear la API se puede ver publicada con el nombre de versión que se haya indicado.

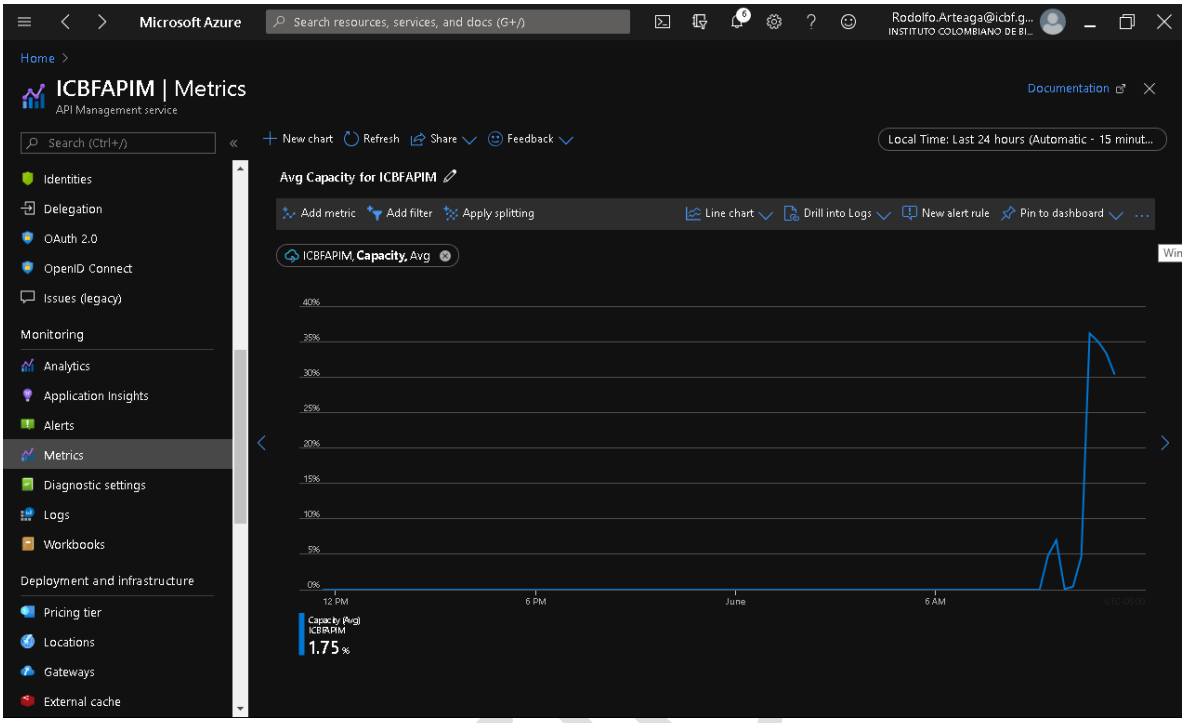


4.6. MONITOREO DE APIS PUBLICADAS

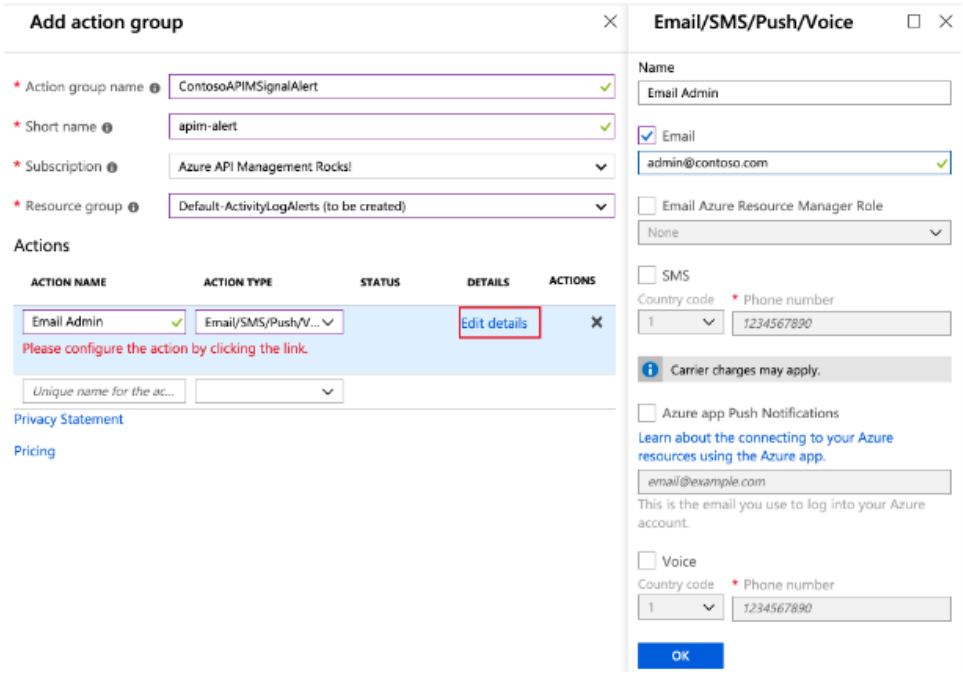
Para monitorizar APIs, visualizando, consultando, ver la ruta, tomar acciones o revisar logs, se pueden tomar acciones revisando qué pasa por el API Management, ya que este guarda información cada minuto. Para ello se puede medir lo siguiente.

- Capacidad: Ayuda a tomar decisiones de cómo hacer upgrade/downgrade de los servicios del API Management, refleja la capacidad del Gateway. Se mide de 0 a 100 en porcentaje de capacidad.
- Solicitudes totales al Gateway: Número de llamados al API en un periodo de tiempo determinado.
- Solicitudes satisfactorias al Gateway: Número de llamados correctos al API en un periodo de tiempo determinado (respuesta HTTP con código 304, y menores a 301, por ejemplo 200).
- Solicitudes fallidas al Gateway: Número de llamados fallidos al API en un periodo de tiempo determinado (Con código 400 o superiores a código 500).
- Solicitudes no autorizadas al Gateway: Mide el número de llamados no autorizados, estos devuelven un código de error 401, 403, y 439.

Podemos medir todas estas características en el menú **Metrics** del API Management. Por ejemplo, la Capacidad del Gateway es mostrada a continuación:

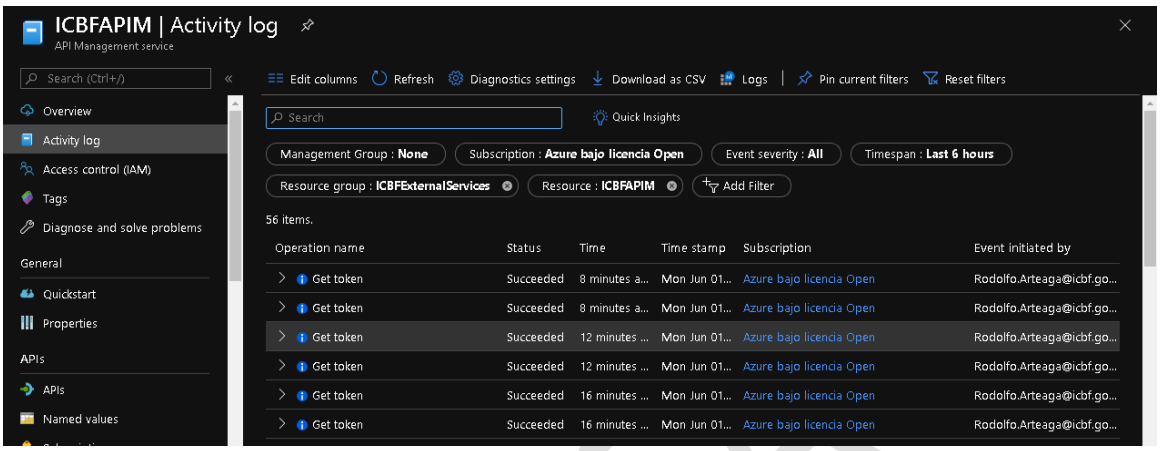


A este monitoreo también pueden añadírseles alarmas para que a los administradores de las APIs puedan ser notificados por Email o SMS.



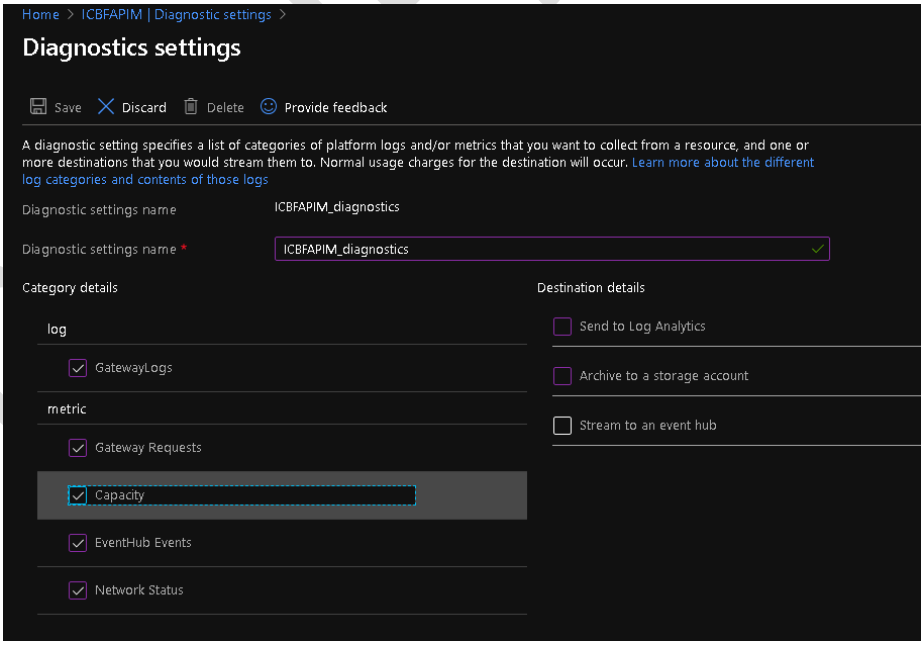
Logs de eventos.

Es posible revisar el log de eventos de la API haciendo clic en el menú “Activity Log”, este muestra toda la información de acceso al API Management.



Log de recursos.

En este log es posible revisar toda la información relacionada a la ejecución y a errores en las operaciones invocadas, para acceder es posible hacerse a través del menú “diagnostic settings”, y posterior a ello haciendo clic en “add diagnostic”.



Como se puede evidenciar, es posible enviar información de la capacidad del Gateway, logs, solicitudes, eventos a eventhub y estado de la red, a un Log Analytics, a una storage account o a un evento hub.

5. DOCUMENTOS DE REFERENCIA

https://azure.microsoft.com/es-es/services/api-management/
https://docs.microsoft.com/es-es/azure/api-management/api-management-get-started-publish-versions
https://docs.microsoft.com/es-es/azure/api-management/api-management-policies
https://docs.microsoft.com/es-mx/azure/api-management/api-management-sample-cache-by-key
https://docs.microsoft.com/en-us/azure/api-management/api-management-get-started-publish-versions

6. CONTROL DE CAMBIOS

Fecha	Versión	Descripción del Cambio
N/A	N/A	N/A

CLASIFICADA