

API Management Custom Policies - Exposing Azure Service Bus

Objective

In this lab, we will be exposing Azure Service Bus as a REST endpoint through API Management. To do this, we will create a [custom policy](#) for API Management. Azure Service Bus exposes several [REST operations](#), which can be used to manage the namespace, as well as do message operations. In this lab we will use an API call to send a message to a queue, while abstracting the Service Bus' specific [authentication](#) method from the API consumer. This means that our consumer does not need to know how to authenticate and communicate with the Service Bus REST API, but instead only needs to know how to do this with API Management, just like any other API which would be published through here. For more information around this and similar scenario's, check this [blog post](#) by [Eldert Grootenboer](#).



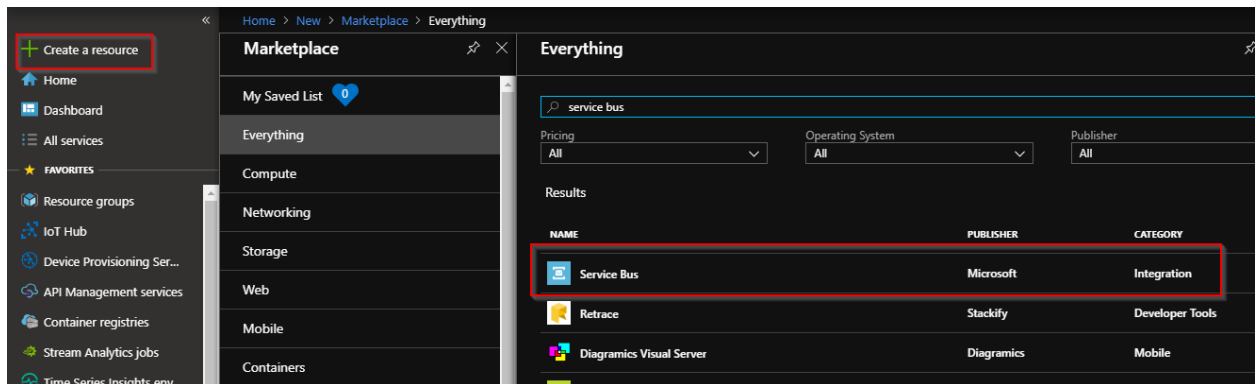
Prerequisites

- Azure account - You can [Open an Azure account for free](#) or [Activate Visual Studio subscriber benefits](#).

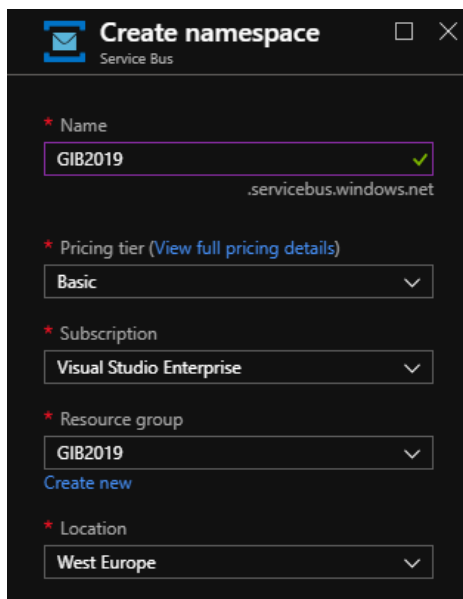
Create Service Bus namespace

The first step in building the solution in this lab is create a Service Bus namespace in Azure.

1. Go to the Azure Portal: <https://portal.azure.com/>
2. Login into the Azure portal with your account.
3. In the Market Place enter Service Bus and select it from the list as shown below.



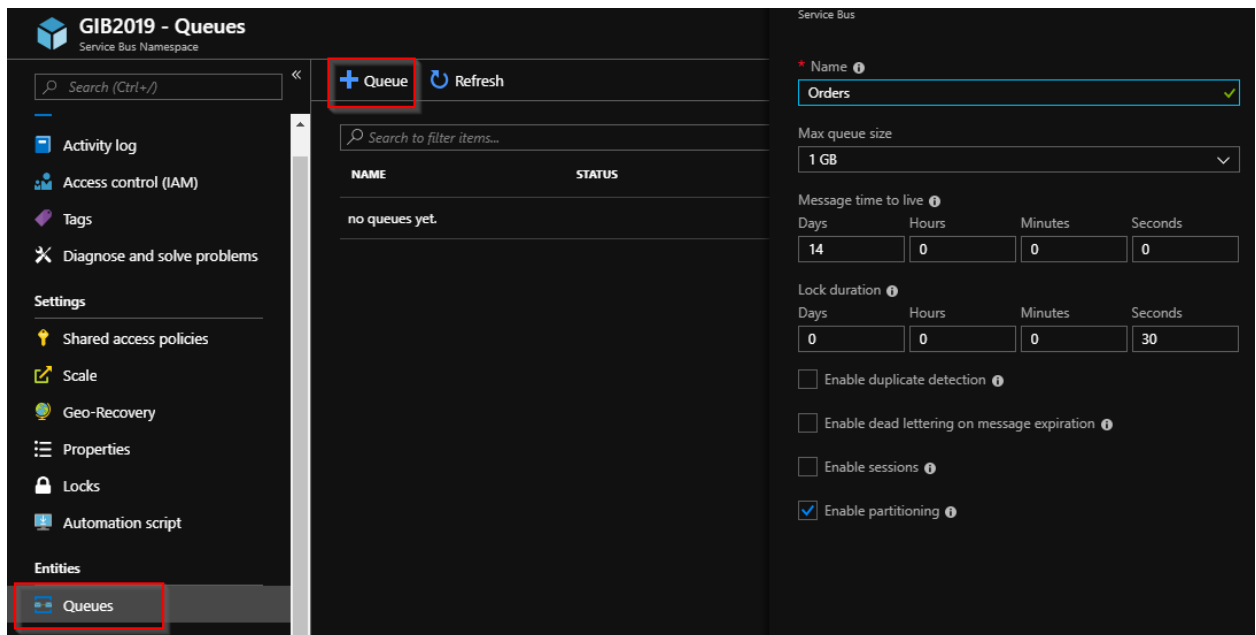
4. Click Create.
5. Specify a name, a Resource Group (you can create a new one here if you haven't created a resource group yet) and a location. You can use the basic tier, as we will only be using queues. Subsequently, click on Review + create.



Create queue

Once the Service Bus namespace has been provisioned you can navigate to it.

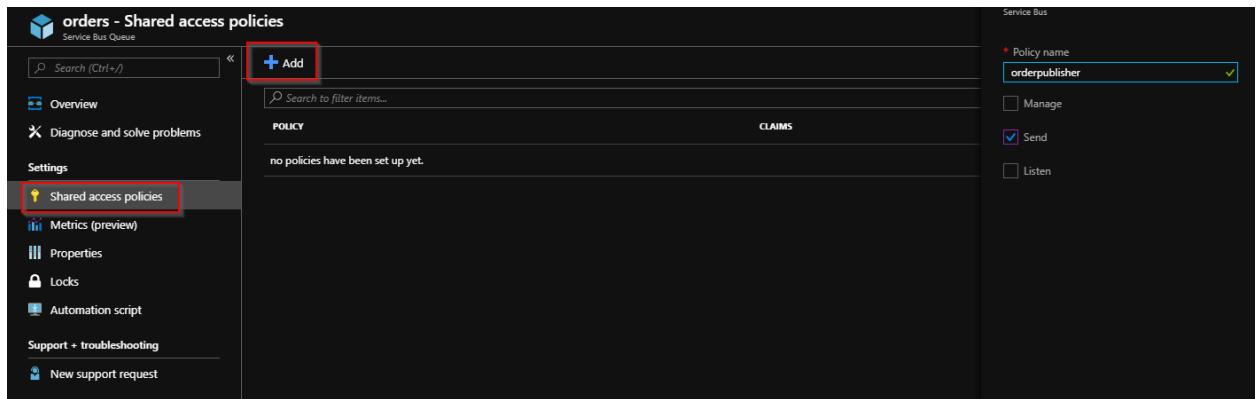
1. In the namespace click on Queues.
2. Click on + Container and specify the name.



Create SAS key

Once the queue has been created, you can click on it to navigate to it.

1. Open Shared access policies.
2. Add a new shared access policy.
3. Specify a policy name, enable the Send permissions and click Create.

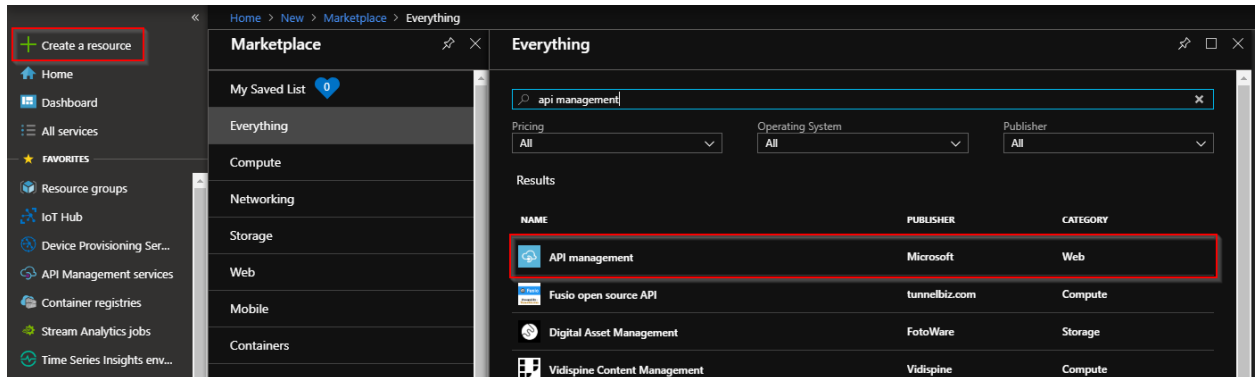


4. Once created, grab the Primary Key value for the key.

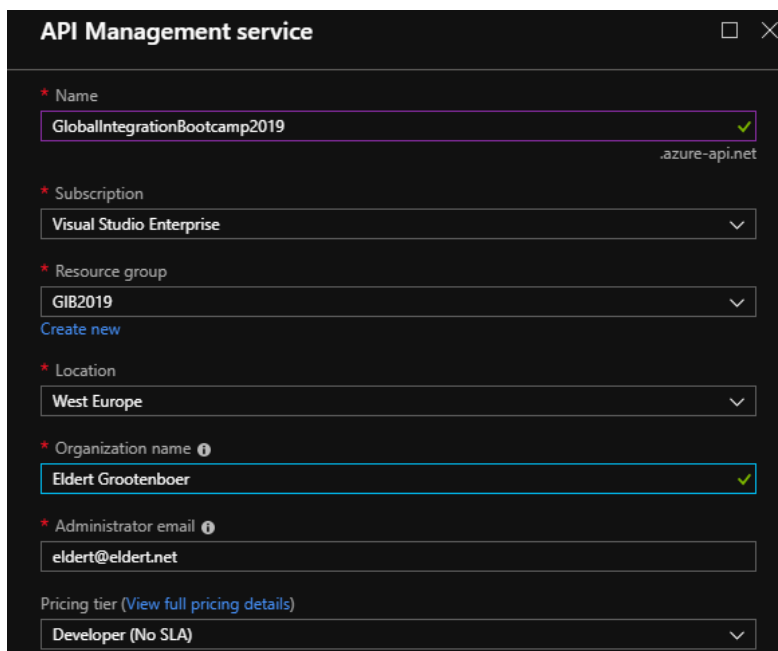
Create API Management instance

Next, we will create an API management instance which will be used to expose our Service Bus queue to our consumer.

1. Go to the Azure Portal: <https://portal.azure.com/>
2. Login into the Azure portal with your account.
3. In the Market Place enter API Management and select it from the list as shown below.



6. Click Create.
7. Specify a name, a Resource Group (you can create a new one here or use the resource group we created in the previous steps) and a location. Subsequently, click on Create.



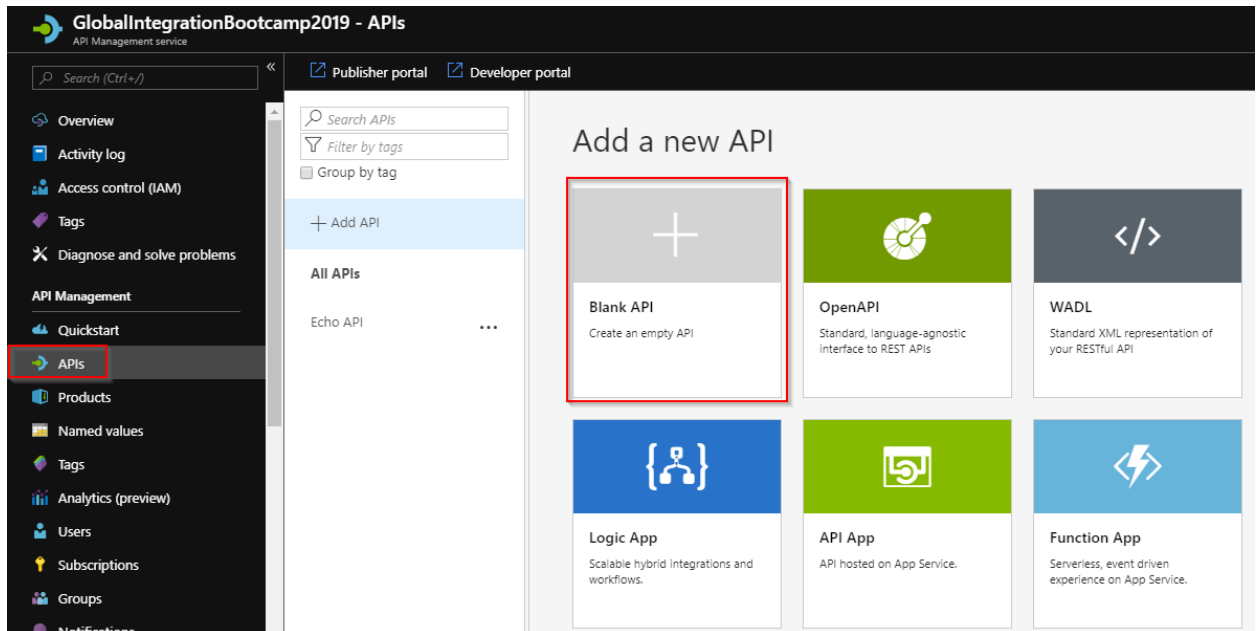
The screenshot shows the 'API Management service' creation form. The form fields are as follows:

- Name:** GlobalIntegrationBootcamp2019 (with a green checkmark icon)
- Subscription:** Visual Studio Enterprise
- Resource group:** GIB2019 (with a 'Create new' link below it)
- Location:** West Europe
- Organization name:** Eldert Grootenboer (with a green checkmark icon)
- Administrator email:** eldert@eldert.net
- Pricing tier:** Developer (No SLA) (with a link to 'View full pricing details')

Create Order API

Once the instance has been created, you can click on it to navigate to your API Management.

1. Open APIs tab.
2. Click on Add API – Blank API



3. Give the API a name, suffix and add it to the Unlimited product, and click Create. We won't need to set the web service URL, as we will be defining this from our custom policy.

Create a blank API

Basic | **Full**

*

Display name

*

Name

Web service URL

API URL suffix

Products

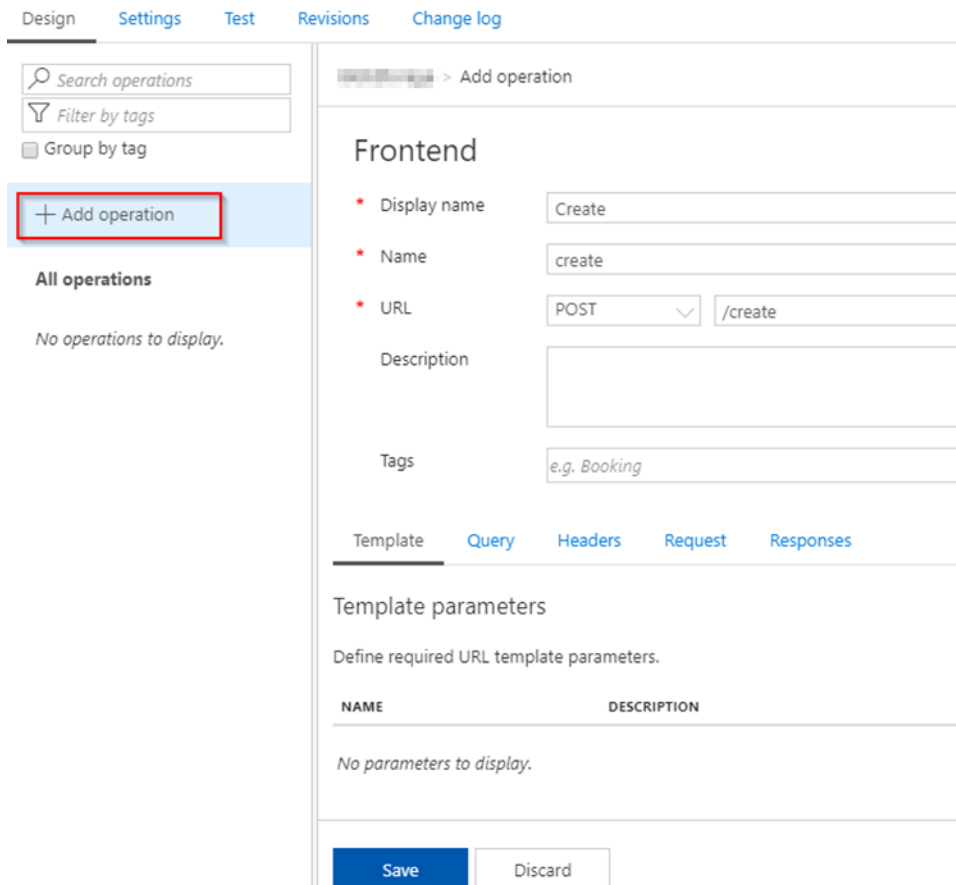
Create

Cancel

Add operation

Navigate to the API we just created.

1. Click on Add operation.
2. Provide a display name and name for the operation.
3. Set the method to Post and provide the suffix.
4. Click on Save.



Design Settings Test Revisions Change log

Search operations

Filter by tags

Group by tag

+ Add operation

All operations

No operations to display.

> Add operation

Frontend

- * Display name: Create
- * Name: create
- * URL: POST /create
- Description:
- Tags: e.g. Booking

Template Query Headers Request Responses

Template parameters

Define required URL template parameters.

NAME	DESCRIPTION
No parameters to display.	

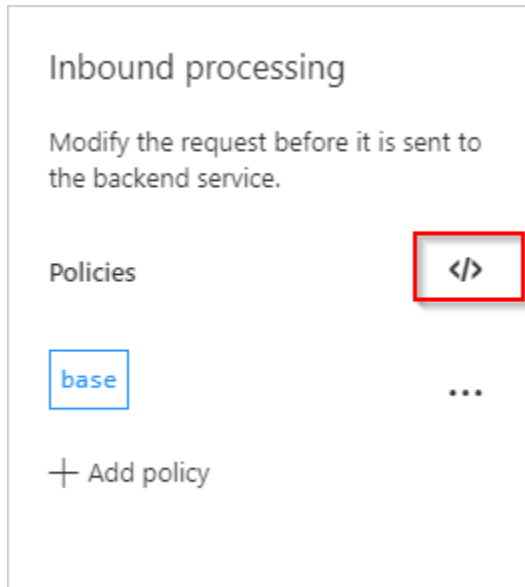
Save Discard



Set policy

Navigate to the operation we just added.

1. Open the Inbound processing policy for the operation which was just created.



2. We will start by creating some variables in the inbound policy section. These variables will be used to calculate the authorization header, which is needed when communicating with Azure Service Bus through the REST API. Everything we do in the following steps need to be inside the `<inbound>` section, right after the `<base />` statement.
 - a. Add a variable for the name of your Service Bus namespace.
`<set-variable name="namespace" value="yourNamespaceName" />`
 - b. Add a variable for the name of your Service Bus queue.
`<set-variable name="queue" value="yourQueueName" />`
 - c. Add a variable for the name of your Service Bus SAS key.
`<set-variable name="sasKeyName" value="yourSASKeyName" />`
 - d. Add a variable for the value of your Service Bus SAS key.
`<set-variable name="sasKeyValue" value="yourSASKeyValue" />`

- Now we will add the policy which calculates the WRAP token for the authentication and adds it to the header keys.

```

set-header name="Authorization" exists-action="override">
    <value>@{
        // Load variables
        string resourceUri = String.Format("https://{0}.servicebus.windows.net/{1}",
            (string)context.Variables.GetValueOrDefault("namespace"),
            (string)context.Variables.GetValueOrDefault("queue"));
        string sasKeyName = (string)context.Variables.GetValueOrDefault("sasKeyName");
        string sasKeyValue = (string)context.Variables.GetValueOrDefault("sasKeyValue");
        // Set the token lifespan
        System.TimeSpan sinceEpoch = System.DateTime.UtcNow.Subtract(new System.DateTime(1970,
            1, 1));
        var expiry = System.Convert.ToString((int)sinceEpoch.TotalSeconds + 60); //1 minute
        string stringToSign = System.Uri.EscapeDataString(resourceUri) + "\n" + expiry;
        System.Security.Cryptography.HMACSHA256 hmac = new
            System.Security.Cryptography.HMACSHA256(System.Text.Encoding.UTF8.GetBytes(sasKey));
        var signature =
            System.Convert.ToBase64String(hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(stringToSign)));
        // Format the sas token
        var sasToken = String.Format("SharedAccessSignature sr={0}&sig={1}&se={2}&skn={3}",
            System.Uri.EscapeDataString(resourceUri), System.Uri.EscapeDataString(signature), expiry,
            sasKeyName);
        return sasToken;
    }</value>

```

- And finally, we will set the backend service to the endpoint of our Service Bus namespace, and append the correct suffix.

```

<set-backend-service base-url="https://yourNamespaceName.servicebus.windows.net/" />
<rewrite-uri template="/yourQueueName/messages" />

```

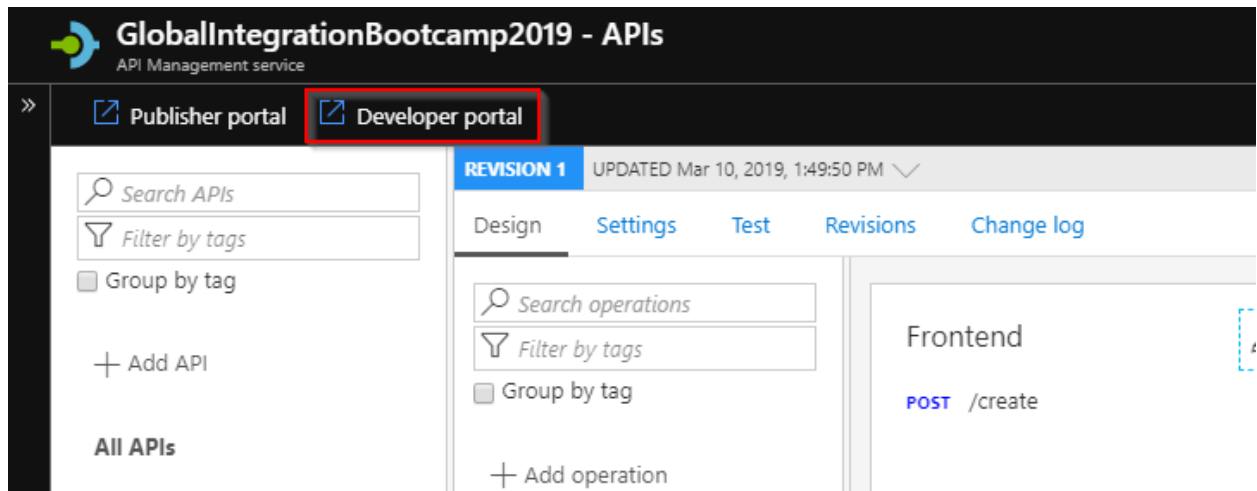
5. The complete policy should now look like the following.

```
<policies>
  <inbound>
    <base />
    <set-variable name="namespace" value="yourNamespaceName" />
    <set-variable name="queue" value="yourQueueName" />
    <set-variable name="sasKeyName" value="yourSASKeyName" />
    <set-variable name="sasKeyValue" value="yourSASKeyValue" />
    <set-header name="Authorization" exists-action="override">
      <value>@{
// Load variables
string resourceUri = String.Format("https://{0}.servicebus.windows.net/{1}",
(string)context.Variables.GetValueOrDefault("namespace"),
(string)context.Variables.GetValueOrDefault("queue"));
string sasKeyName = (string)context.Variables.GetValueOrDefault("sasKeyName");
string sasKeyValue = (string)context.Variables.GetValueOrDefault("sasKeyValue");
// Set the token lifespan
System.TimeSpan sinceEpoch = System.DateTime.UtcNow.Subtract(new System.DateTime(1970, 1, 1));
var expiry = System.Convert.ToString((int)sinceEpoch.TotalSeconds + 60); //1 minute
string stringToSign = System.Uri.EscapeDataString(resourceUri) + "\n" + expiry;
System.Security.Cryptography.HMACSHA256 hmac = new
System.Security.Cryptography.HMACSHA256(System.Text.Encoding.UTF8.GetBytes(sasKeyValue));
var signature =
System.Convert.ToBase64String(hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(stringToSign)))
;
// Format the sas token
var sasToken = String.Format("SharedAccessSignature sr={0}&sig={1}&se={2}&skn={3}",
System.Uri.EscapeDataString(resourceUri), System.Uri.EscapeDataString(signature), expiry, sasKeyName);
return sasToken;
}</value>
    </set-header>
    <set-backend-service base-url="https://yourNamespaceName.servicebus.windows.net/" />
    <rewrite-uri template="/yourQueueName/messages" />
  </inbound>
</backend>
  <base />
</backend>
<outbound>
  <base />
</outbound>
<on-error>
  <base />
</on-error>
</policies>
```

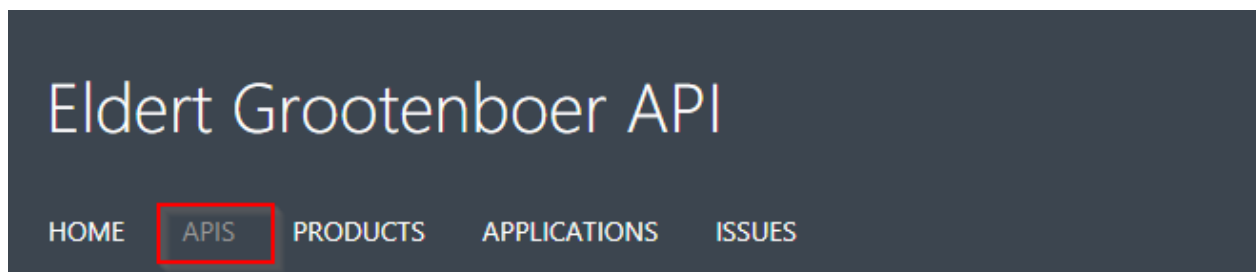
Test

To test the solution, you can use the developer portal of API Management.

1. Open the developer portal.



2. Navigate to the API we just created.



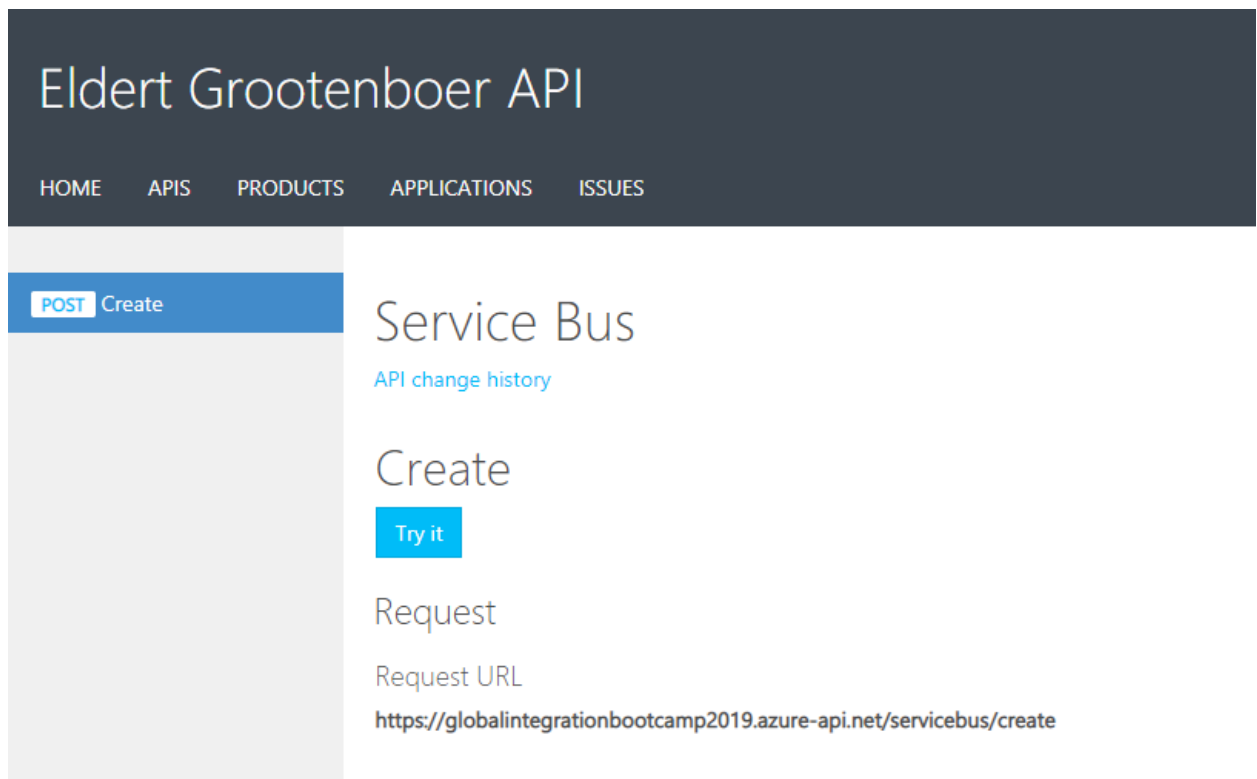
APIs

WebStorage

Echo API

Service Bus

3. Click on Try it.



4. Use the following message body and click on Send.

```
{
  "name" : "John Smith",
  "sku" : "20223",
  "price" : 23.95,
  "shipTo" : {
    "name" : "Jane Smith",
    "address" : "123 Maple Street",
    "city" : "Pretendville",
    "state" : "NY",
    "zip" : "12345"
  },
  "billTo" : {
    "name" : "John Smith",
    "address" : "123 Maple Street",
    "city" : "Pretendville",
    "state" : "NY",
    "zip" : "12345"
  }
}
```

API Management Custom Policies - Exposing Azure Service Bus

5. You should now get 201 Created response, indicating the message has been delivered to the queue. You can use tooling like [Service Bus Explorer](#) or [Serverless360](#) to check if the message was indeed delivered in the queue.

Response

Trace

Response status

201 Created

Response latency

195 ms

Response content

```
Transfer-Encoding: chunked
Strict-Transport-Security: max-age=31536000
Ocp-Apim-Trace-Location: https://apimgmtsttgry0y7kqh4trh.blob.core.windows.net/apiinspectorcontainer/CeBZmDLtT5JYGta-Xb1bSA2-38?sv=2017-04-17&sr=b&sig=8jiXGjFXr8dSntfgkg%2BiuMegD%2BRHcZod9K6FvbHU054%3D&se=2019-03-11T13%3A38%3A32Z&sp=r&traceId=c32f435372de48b2bf28ae7b972b1378
Date: Sun, 10 Mar 2019 13:38:32 GMT
Content-Type: application/xml; charset=utf-8
```