TLS Proxy: Custom domain support for Event Hubs using Application Gateway

Contents

Overview

EventHubs support access via standard eventhub FQDN (ex. sb://eventhubbackend.servicebus.windows.net/). It does not support custom domain URLs like sb://eventhubbackend.contoso.com and certificates for them. This feature can be supported by adding Application Gateway as frontend for EventHub and map the desired custom domain URL to EventHub URL in the backend.

Supported scenarios when going through Application Gateway:

1. Access via AMQP or AMQPOverWebSocket protocols
2. Authentication via SAS, AAD Apps or Managed identities

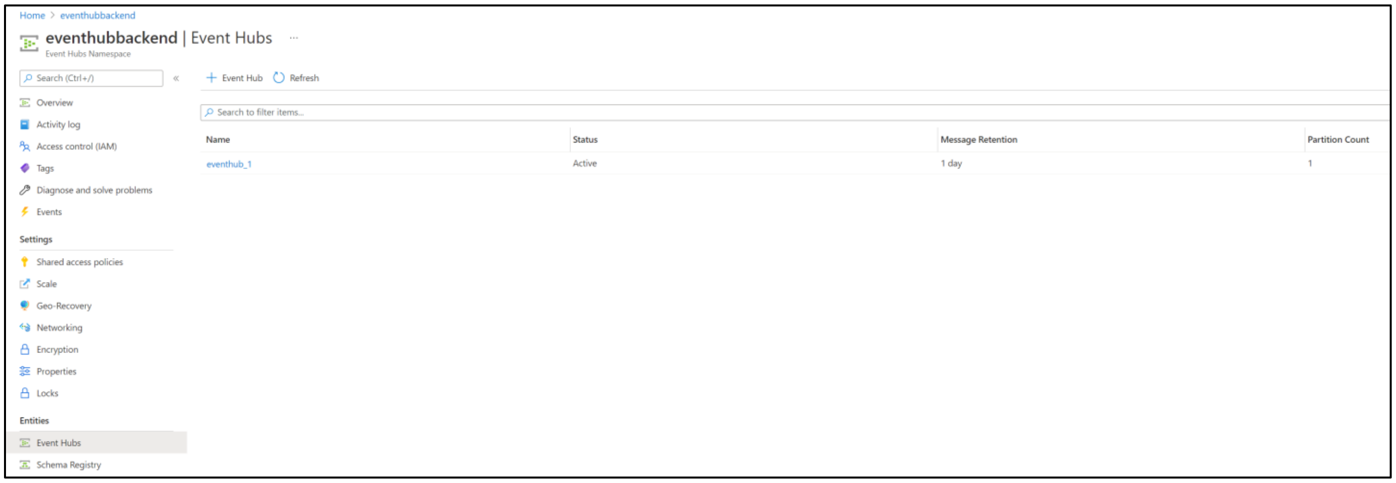
Pre-requisites

1. EventHub Namespace with an Event Hub instance
2. Application Gateway using L7 (AMQP over WebSockets) and L4 capabilities (AMQP)
3. Client SDK (Documentation)

Solution – Detailed Process

Create Event Hubs

1. Create an Event Hub Namespace and an Event Hub instance in it. In this example, we have **eventhubbackend** namespace with **eventhub_1** as an instance.



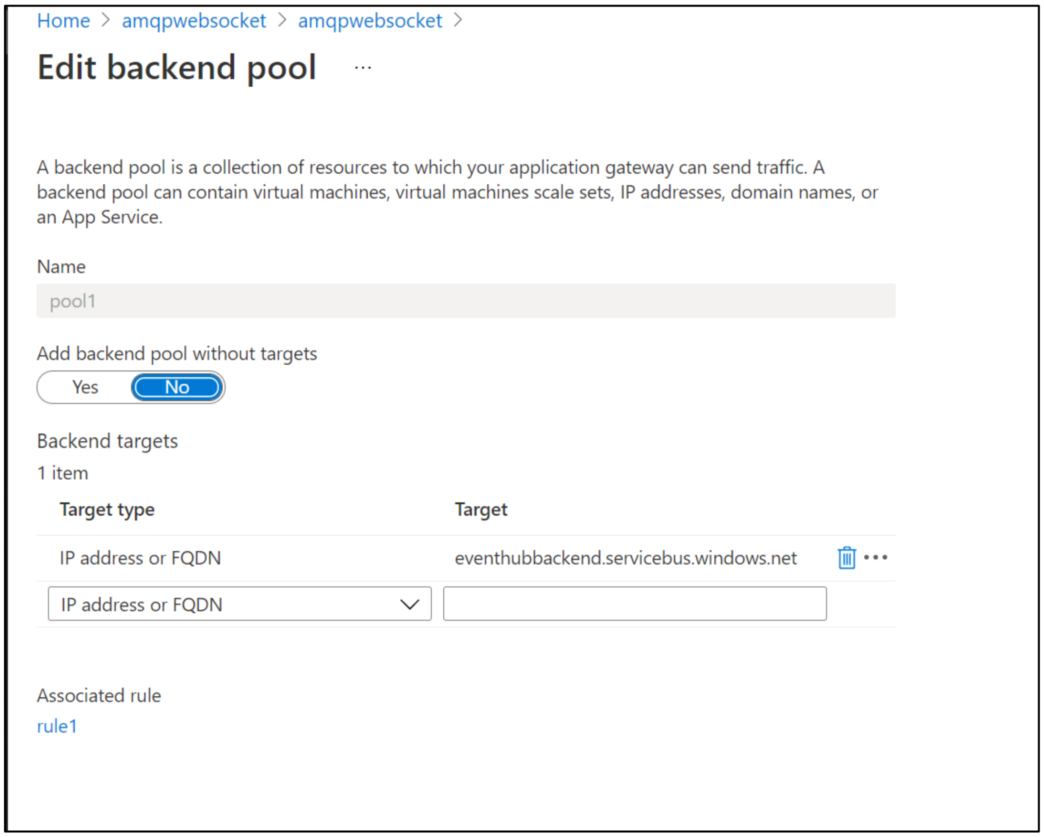
Name	Status	Message Retention	Partition Count
eventhub_1	Active	1 day	1

Setting up Application Gateway with AMQP Protocol

Note: For this to work with Application Gateway, TLS/TCP proxying support is required. Since the feature is still under development, this section is descriptive without any actual examples.

1. Setup a TLS listener (by selecting TLS protocol for listener) with custom certificate on port 5671 (it can be a different port)

2. Add a well-known Event Hub URI for the backend pool (the backend service is private link-enabled in this case)


Home > amqpwebsocket > amqpwebsocket >

Edit backend pool ...

A backend pool is a collection of resources to which your application gateway can send traffic. A backend pool can contain virtual machines, virtual machines scale sets, IP addresses, domain names, or an App Service.

Name
pool1

Add backend pool without targets
Yes [✓ No]

Backend targets
1 item

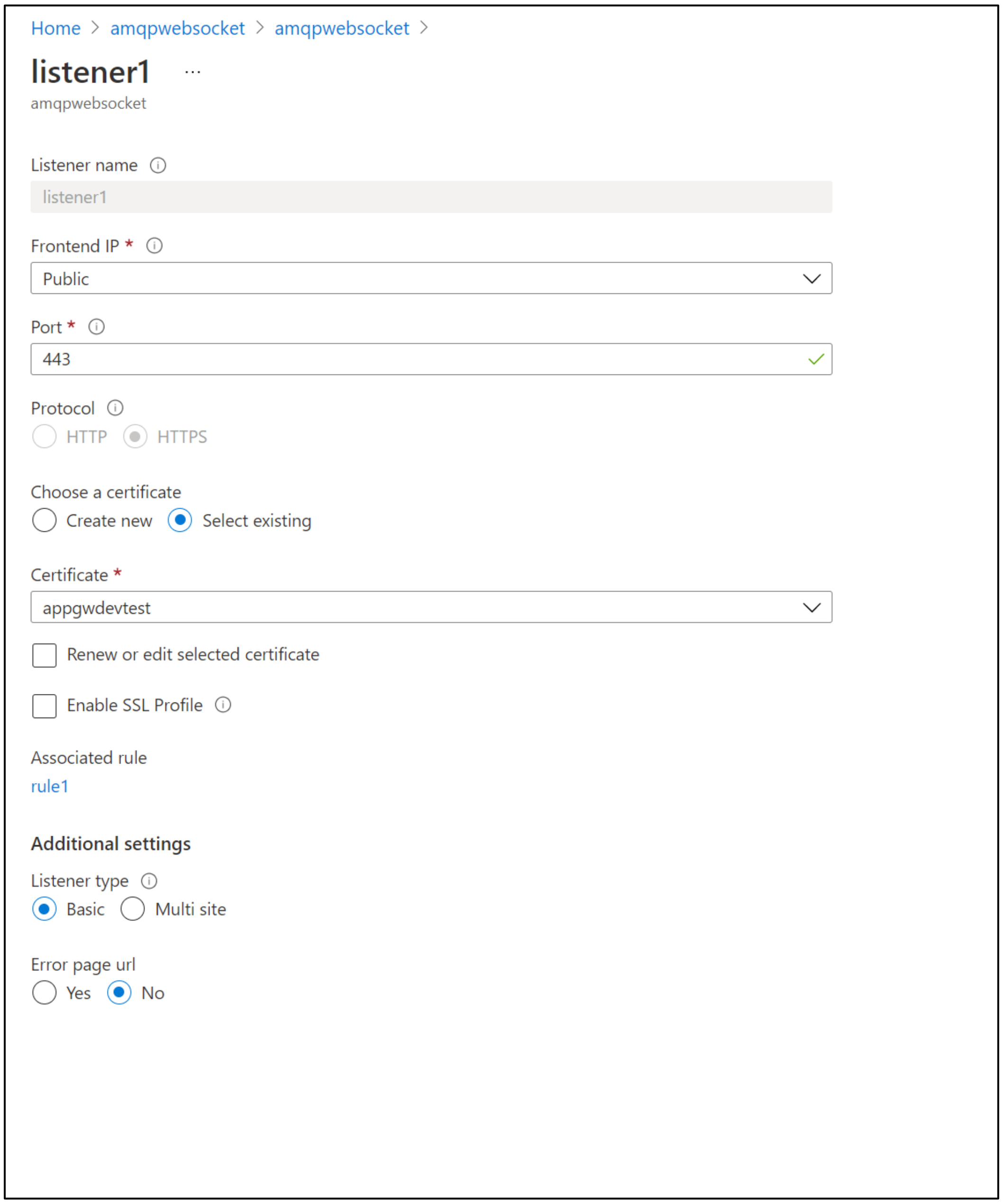
Target type	Target
IP address or FQDN	eventhubbackend.servicebus.windows.net
IP address or FQDN	

Associated rule
rule1

3. Setup Backend Settings with Protocol as TLS and Port as 5671. Select “yes” for well-known trusted root certificate.
4. Setup Custom Probe and connect it to the configured Backend Settings.
5. Set a Rule connecting Backend Settings with a Pool and Listener created in the previous steps.

Setting up Application Gateway with AMQP over WebSockets protocol

1. Setup an HTTPS listener (Layer7) with custom certificate on port 443



Home > amqpwebsocket > amqpwebsocket >

listener1 ...

amqpwebsocket

Listener name: listener1

Frontend IP *: Public

Port *: 443

Protocol: HTTP [✓ HTTPS]

Choose a certificate: Create new [✓ Select existing]

Certificate *: appgwdevtest

☐ Renew or edit selected certificate

☐ Enable SSL Profile

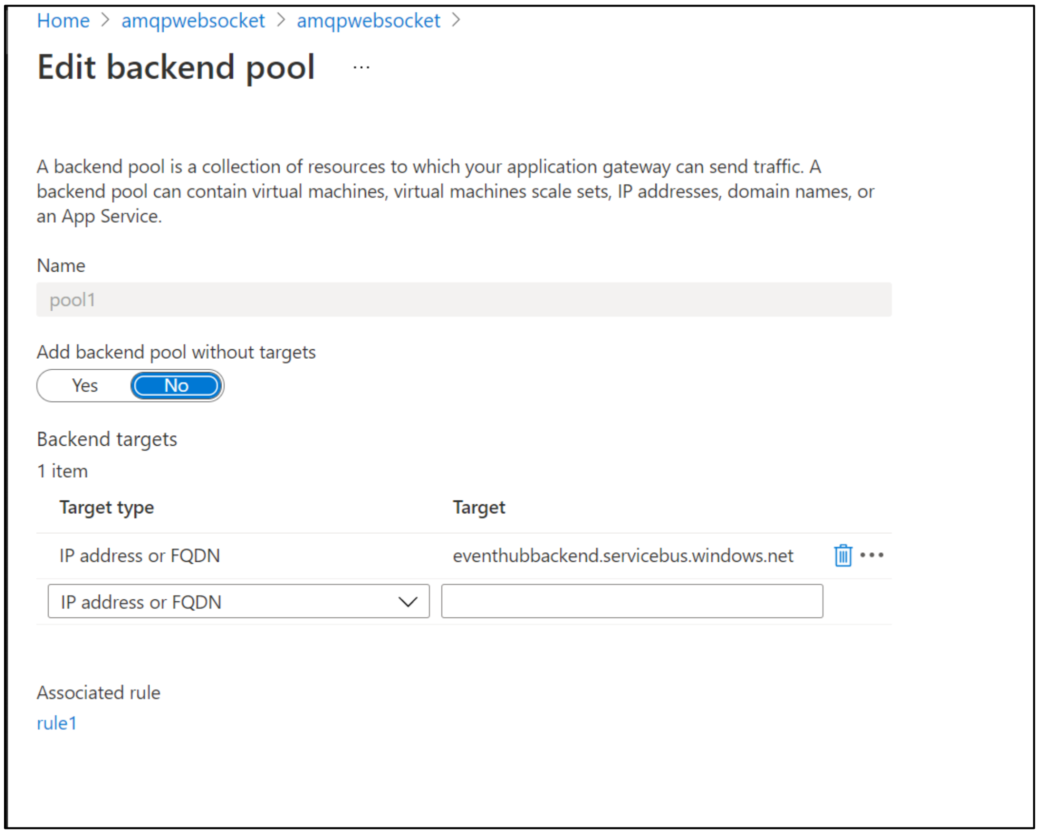
Associated rule: rule1

Additional settings

Listener type: [✓ Basic] Multi site

Error page url: Yes [✓ No]

2. Add a well-known Event Hub URI for the backend pool (the backend service is private link-enabled in this case)


Home > amqpwebsocket > amqpwebsocket >

Edit backend pool ...

A backend pool is a collection of resources to which your application gateway can send traffic. A backend pool can contain virtual machines, virtual machines scale sets, IP addresses, domain names, or an App Service.

Name

pool1

Add backend pool without targets

Yes [✓ No]

Backend targets

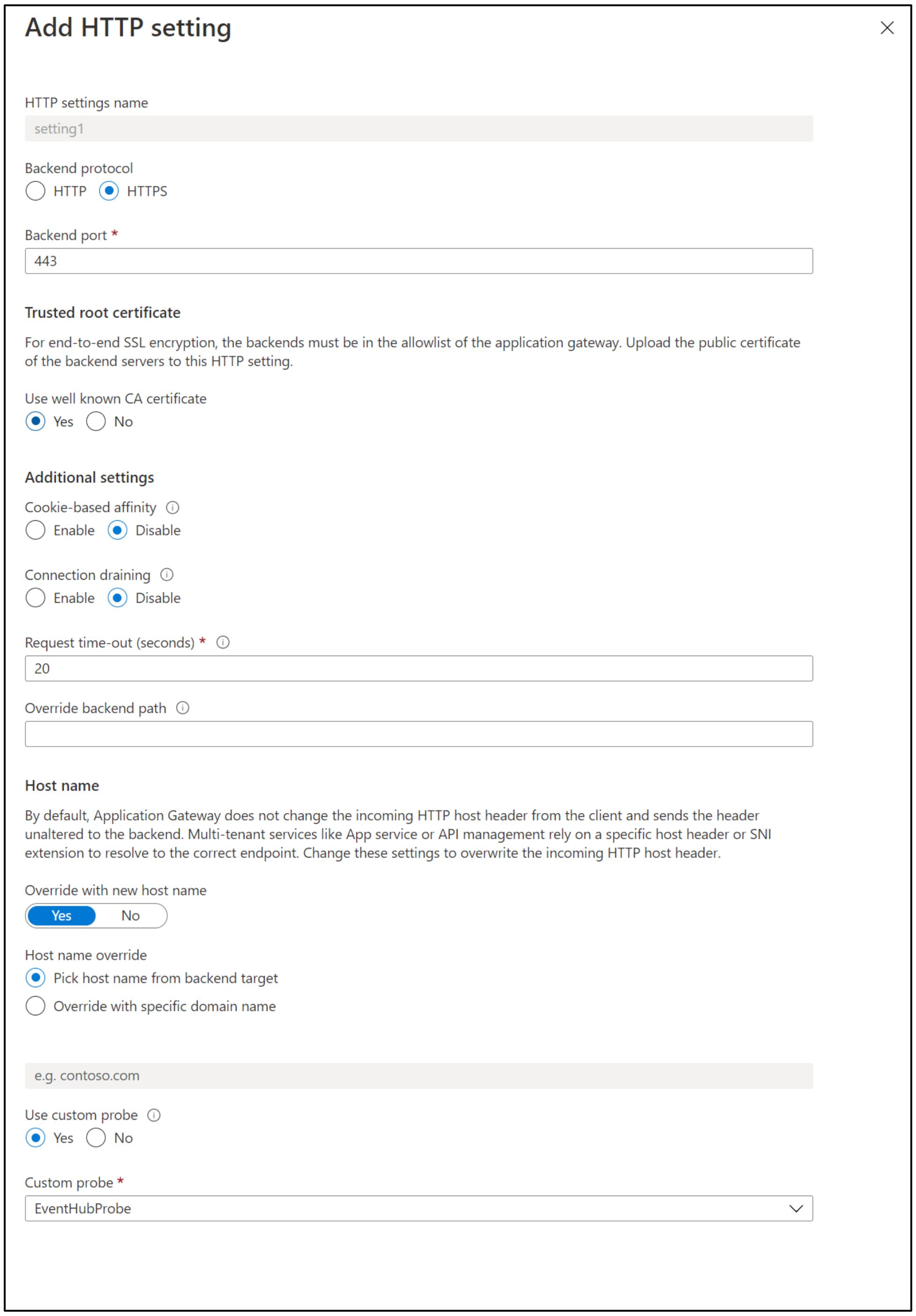
1 item

Target type	Target
IP address or FQDN	eventhubbackend.servicebus.windows.net
IP address or FQDN	

Associated rule

rule1

3. Setup backend HTTP settings for port 443 as show below.



Add HTTP setting

HTTP settings name
setting1

Backend protocol
HTTP [✓ HTTPS]

Backend port *
443

Trusted root certificate

For end-to-end SSL encryption, the backends must be in the allowlist of the application gateway. Upload the public certificate of the backend servers to this HTTP setting.

Use well known CA certificate
[✓ Yes] No

Additional settings

Cookie-based affinity
Enable [✓ Disable]

Connection draining
Enable [✓ Disable]

Request time-out (seconds) *
20

Override backend path

Host name

By default, Application Gateway does not change the incoming HTTP host header from the client and sends the header unaltered to the backend. Multi-tenant services like App service or API management rely on a specific host header or SNI extension to resolve to the correct endpoint. Change these settings to overwrite the incoming HTTP host header.

Override with new host name
[✓ Yes] No

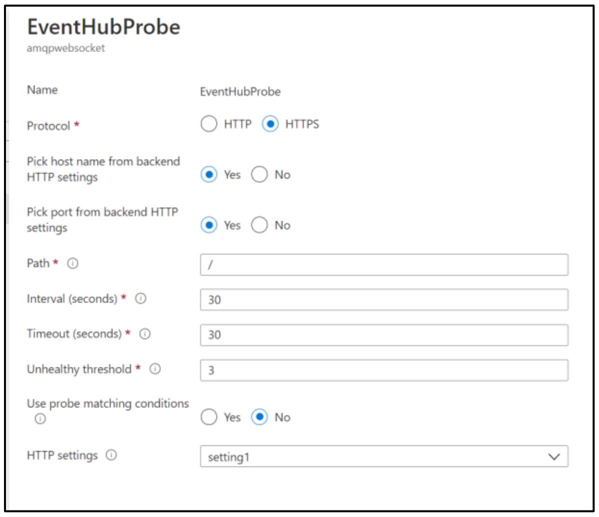
Host name override
[✓ Pick host name from backend target]
Override with specific domain name

e.g. contoso.com

Use custom probe
[✓ Yes] No

Custom probe *
EventHubProbe

4. Setup a Custom Probe and connect it with the above HTTP setting.



EventHubProbe
amqpwebsocket

Name	EventHubProbe
Protocol *	HTTP [✓ HTTPS]
Pick host name from backend HTTP settings	[✓ Yes] No
Pick port from backend HTTP settings	[✓ Yes] No
Path *	/
Interval (seconds) *	30
Timeout (seconds) *	30
Unhealthy threshold *	3
Use probe matching conditions	Yes [✓ No]
HTTP settings	setting1

5. Create a Rule connecting HTTP setting with the Backend pool and the Listener created in previous steps.

Before we proceed, the custom domain's DNS should be setup to resolve to Application Gateway IP.

Connecting using an SDK Client

When using the SDK for connection establishment via Application Gateway, following are the requirements.

1. On the connections object set CustomEndpointAddress to point to Application Gateway.

```
  // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
  // The eh:// can be any string. SDK requires something to be presented
  options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudapp.net:5671");
```

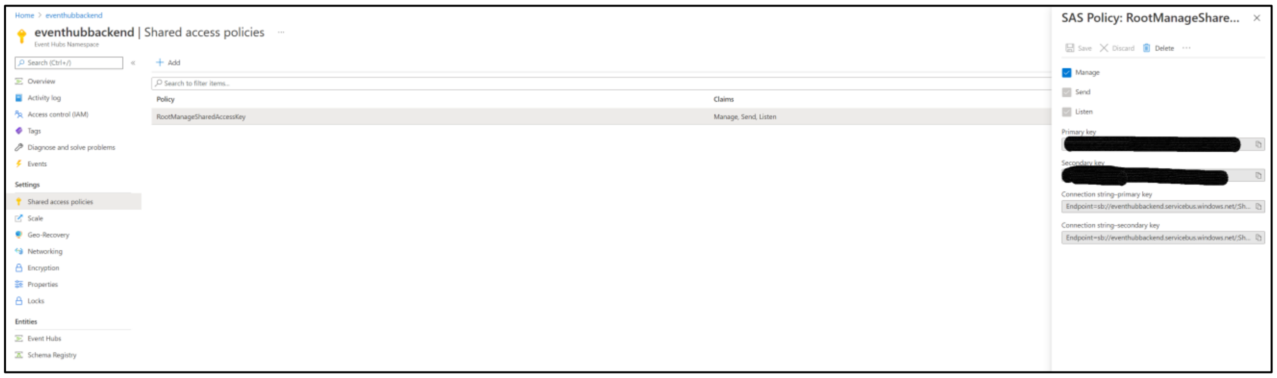
2. When creating Producer/Consumer clients use actual EventHub name instead of Application Gateway custom domain. As CustomEndpointAddress is set on the connection option traffic will be routed via Application Gateway but underneath it will establish the connection to actual EventHub

```
 var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", eventHubName, new DefaultAzureCredential(), options);
```

Authentication

SAS Keys

1. Get the SAS key from Azure Portal on the Event Hub >> Shared access policies blade.



Home > eventhubbackend

eventhubbackend | Shared access policies

Event Hubs Namespace

Policy	Claims
RootManageSharedAccessKey	Manage, Send, Listen

SAS Policy: RootManageShare...

[✓ Manage] Send Listen

Primary key: [redacted]

Secondary key: [redacted]

Connection string–primary key: Endpoint=sb://eventhubbackend.servicebus.windows.net/;Sh...

Connection string–secondary key: Endpoint=sb://eventhubbackend.servicebus.windows.net/;Sh...

2. Create a connection in the SDK and send events:

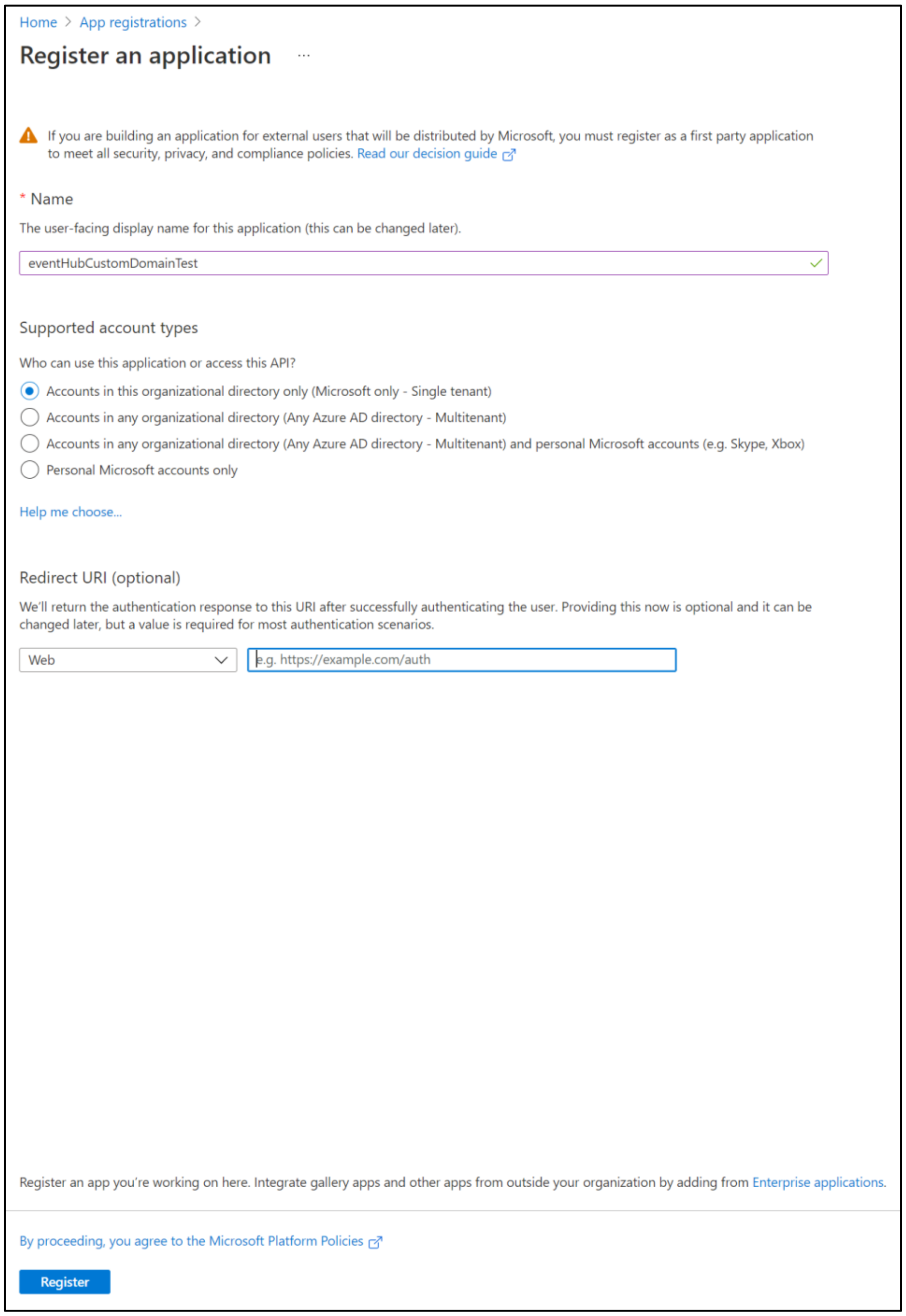
```
  private static async Task SendUsingSASKey()
  {
      // Create a producer client that you can use to send events to an event hub
      var credentials = new AzureNamedKeyCredential("RootManageSharedAccessKey", "<SAS KEY FROM Portal>");
      var options = new EventHubProducerClientOptions();

      // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
      // The eh:// can be any string. SDK requires something to be presented
      options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudapp.net:5671");
      var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", eventHubName, credentials, options);

      try
      {
          await SendEvent(producerClient, "Send using SAS auth");
      }
      finally
      {
          await producerClient.DisposeAsync();
      }
  }
```

AAD Application

1. Create AAD Application via App registrations:



Home > App registrations >

Register an application ...

⚠ If you are building an application for external users that will be distributed by Microsoft, you must register as a first party application to meet all security, privacy, and compliance policies. Read our decision guide

* Name

The user-facing display name for this application (this can be changed later).

eventHubCustomDomainTest ✓

Supported account types

Who can use this application or access this API?

- ● Accounts in this organizational directory only (Microsoft only - Single tenant)
- ○ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ○ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ○ Personal Microsoft accounts only

Help me choose...

Redirect URI (optional)

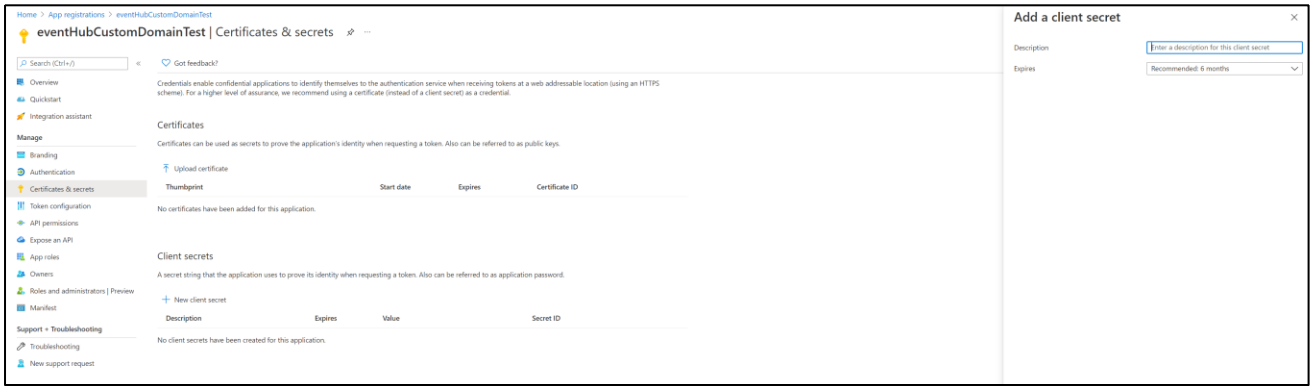
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web | e.g. https://example.com/auth

Register an app you're working on here. Integrate gallery apps and other apps from outside your organization by adding from Enterprise applications.

By proceeding, you agree to the Microsoft Platform Policies

2. Add a client secret for authentication. (Auth can also be done with a client certificate)



Home > App registrations > eventHubCustomDomainTest

eventHubCustomDomainTest | Certificates & secrets

Search (Ctrl+/)

- Overview
- Quickstart
- Integration assistant

Manage

- Branding
- Authentication
- Certificates & secrets
- Token configuration
- API permissions
- Expose an API
- App roles
- Owners
- Roles and administrators | Preview
- Manifest

Support + Troubleshooting

- Troubleshooting
- New support request

Got feedback?

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Upload certificate

Thumbprint	Start date	Expires	Certificate ID

No certificates have been added for this application.

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

New client secret

Description	Expires	Value	Secret ID

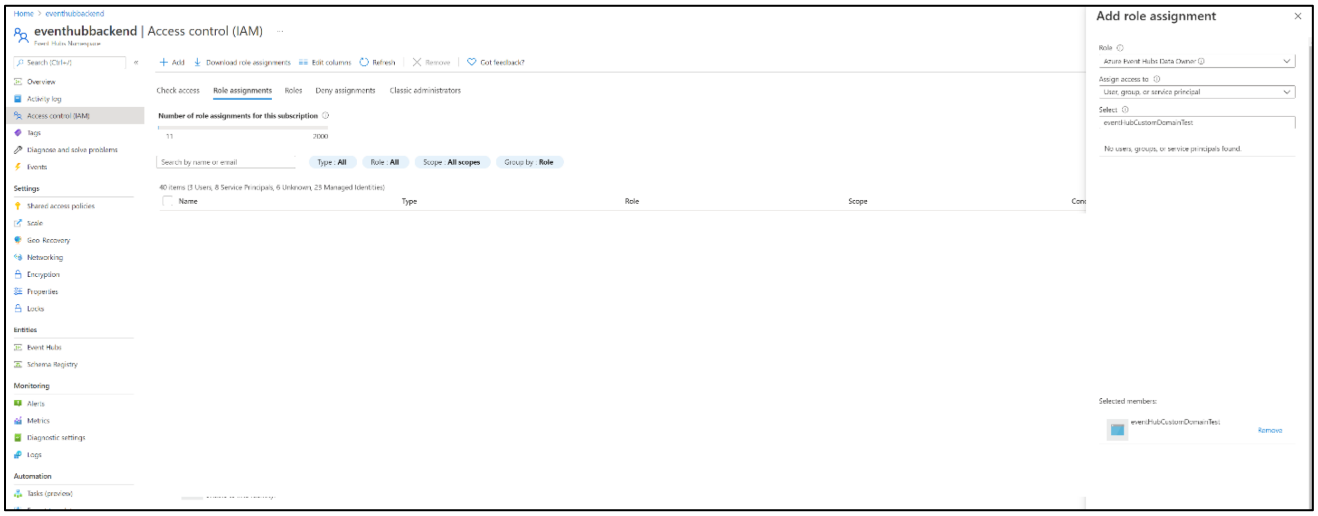
No client secrets have been created for this application.

Add a client secret

Description: Enter a description for this client secret

Expires: Recommended: 6 months

3. Allow AAD Application access to Event Hub:



Home > eventhubbackend

eventhubbackend | Access control (IAM)

Event Hubs Namespace

Search (Ctrl+/)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Events

Settings

- Shared access policies
- Scale
- Geo Recovery
- Networking
- Encryption
- Properties
- Locks

Entities

- Event Hubs
- Schema Registry

Monitoring

- Alerts
- Metrics
- Diagnostic settings
- Logs

Automation

- Tasks (preview)

Add | Download role assignments | Edit columns | Refresh | Remove | Got feedback?

Check access | Role assignments | Roles | Deny assignments | Classic administrators

Number of role assignments for this subscription

11 2000

Search by name or email

Type : All | Role : All | Scope : All scopes | Group by : Role

40 items (3 Users, 8 Service Principals, 6 Unknown, 23 Managed Identities)

Name	Type	Role	Scope	Con

Add role assignment

Role: Azure Event Hubs Data Owner

Assign access to: User, group, or service principal

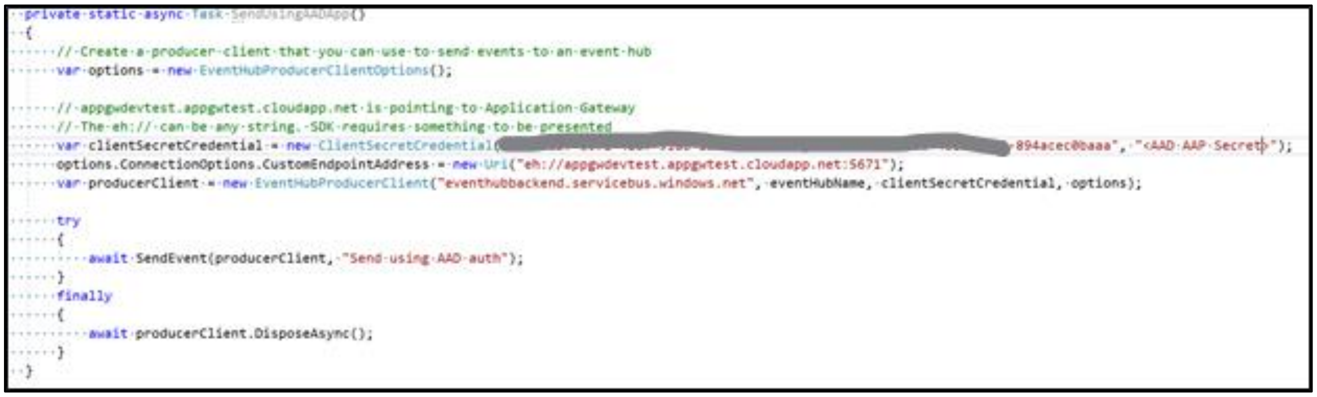
Select: eventHubCustomDomainTest

No users, groups, or service principals found.

Selected members:

eventHubCustomDomainTest Remove

4. Create a connection in the SDK and send/receive events:



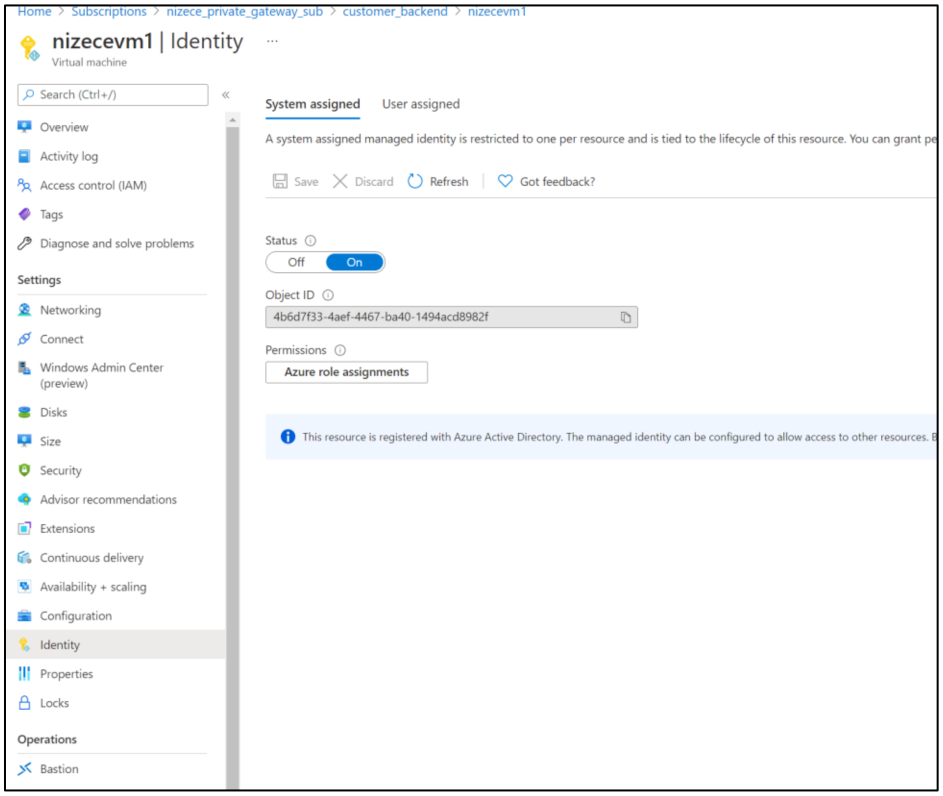
```
  private static async Task SendUsingAADApp()
  {
      // Create a producer client that you can use to send events to an event hub
      var options = new EventHubProducerClientOptions();

      // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
      // The eh:// can be any string. SDK requires something to be presented
      var clientSecretCredential = new ClientSecretCredential([redacted]894acec0baaa", "<AAD AAP Secret>");
      options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudapp.net:5671");
      var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", eventHubName, clientSecretCredential, options);

      try
      {
          await SendEvent(producerClient, "Send using AAD auth");
      }
      finally
      {
          await producerClient.DisposeAsync();
      }
  }
```

Managed Identity

1. Create your client side with Managed Identity. In this example I've created a VM with system assigned managed identity:



A system assigned managed identity is restricted to one per resource and is tied to the lifecycle of this resource. You can grant pe

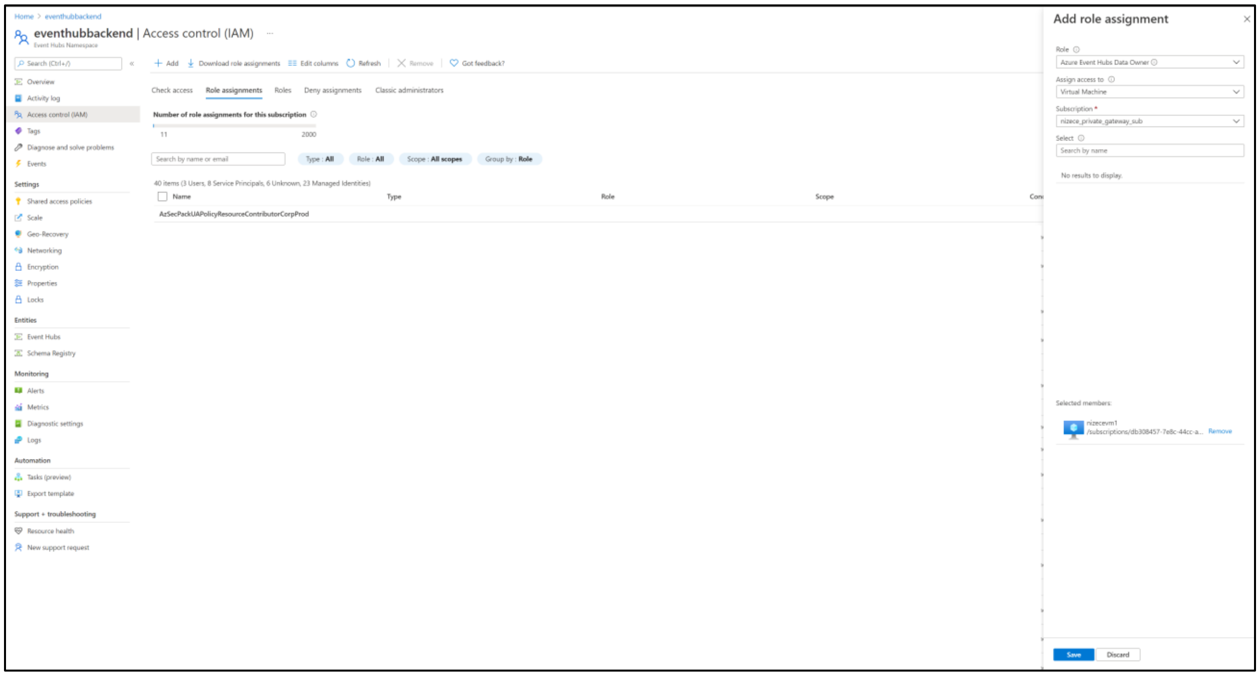
Status: Off [✓ On]

Object ID: 4b6d7f33-4aef-4467-ba40-1494acd8982f

Permissions

This resource is registered with Azure Active Directory. The managed identity can be configured to allow access to other resources. B

2. Allow that Managed Identity access to Event Hub:



Number of role assignments for this subscription: 11 / 2000

Search by name or email · Type : All · Role : All · Scope : All scopes · Group by : Role

40 items (3 Users, 8 Service Principals, 6 Unknown, 23 Managed Identities)

Name	Type	Role	Scope	Con
AzSecPackUAPolicyResourceContributorCorpProd				

Role: Azure Event Hubs Data Owner

Assign access to: Virtual Machine

Subscription *: nizece_private_gateway_sub

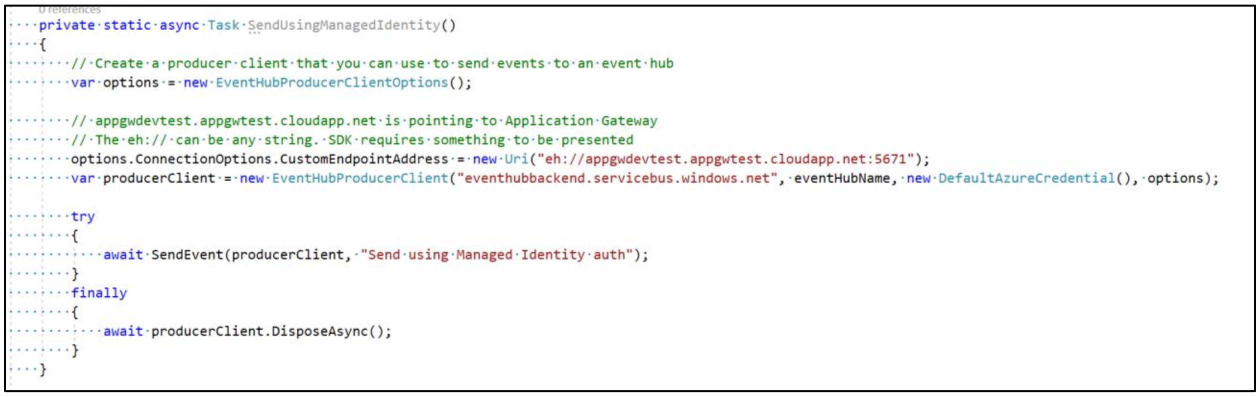
Select: Search by name

No results to display.

Selected members:

nizecevm1
/subscriptions/db308457-7e8c-44cc-a...

3. Create a connection in the SDK and send events:



```
0 references
    private static async Task SendUsingManagedIdentity()
    {
        // Create a producer client that you can use to send events to an event hub
        var options = new EventHubProducerClientOptions();

        // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
        // The eh:// can be any string. SDK requires something to be presented
        options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudapp.net:5671");
        var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", eventHubName, new DefaultAzureCredential(), options);

        try
        {
            await SendEvent(producerClient, "Send using Managed Identity auth");
        }
        finally
        {
            await producerClient.DisposeAsync();
        }
    }
```

Full SDK Code Example

```
namespace EventHubSender
{
    using System;
    using System.Text;
    using System.Threading.Tasks;
    using Azure;
    using Azure.Identity;
    using Azure.Messaging.EventHubs;
    using Azure.Messaging.EventHubs.Producer;
    using Azure.Messaging.EventHubs.Consumer;

    class Program
    {
        // name of the event hub
        private const string eventHubName = "eventhub_1";

        static async Task Main()
        {
            await SendUsingSASKey();
            Console.WriteLine("=================================================");
            await ReceiveUsingSASKey();
            Console.WriteLine("=================================================");
            await SendUsingAADApp();
            Console.WriteLine("=================================================");
            await ReceiveUsingAADApp();
            Console.WriteLine("=================================================");
            await SendUsingAADAppWithWebSockets();
            Console.WriteLine("=================================================");
            await ReceiveUsingAADAppWithWebSockets();
            Console.WriteLine("=================================================");
            await SendUsingManagedIdentity();
            Console.WriteLine("=================================================");
            await ReceiveUsingManagedIdentity();
            Console.WriteLine("=================================================");
        }

        private static async Task SendUsingSASKey()
        {
            // Create a producer client that you can use to send events to an event hub
            var credentials = new AzureNamedKeyCredential("RootManageSharedAccessKey", "<SAS KEY>");
            var options = new EventHubProducerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", even
tHubName, credentials, options);

            try
            {
                await SendEvent(producerClient, "Send using SAS auth");
            }
            finally
            {
                await producerClient.DisposeAsync();
            }
        }

        // TODO: Lets also test with AmqpOverWebSockets
        private static async Task SendUsingAADApp()
        {
            // Create a producer client that you can use to send events to an event hub
            var options = new EventHubProducerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            var clientSecretCredential = new ClientSecretCredential("<TenantID>",
"<AAD APP ID>", "<AAD SECRET>");
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", even
tHubName, clientSecretCredential, options);

            try
            {
                await SendEvent(producerClient, "Send using AAD auth");
            }
            finally
            {
                await producerClient.DisposeAsync();
            }
        }

        private static async Task SendUsingAADAppWithWebSockets()
        {
            // Create a producer client that you can use to send events to an event hub
            var options = new EventHubProducerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            var clientSecretCredential = new ClientSecretCredential(“<TenantID>”, "<AAD APP ID>", "<AAD SE
CRET>");
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            options.ConnectionOptions.TransportType = EventHubsTransportType.AmqpWebSockets;
            var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", even
tHubName, clientSecretCredential, options);

            try
            {
                await SendEvent(producerClient, "Send using Web Socket AAD auth");
            }
            finally
            {
                await producerClient.DisposeAsync();
            }
        }

        private static async Task SendUsingManagedIdentity()
        {
            // Create a producer client that you can use to send events to an event hub
            var options = new EventHubProducerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            var producerClient = new EventHubProducerClient("eventhubbackend.servicebus.windows.net", even
tHubName, new DefaultAzureCredential(), options);
```

```
            try
            {
                await SendEvent(producerClient, "Send using Managed Identity auth");
            }
            finally
            {
                await producerClient.DisposeAsync();
            }
        }


        private static async Task ReceiveUsingSASKey()
        {
            // Create a producer client that you can use to send events to an event hub
            var credentials = new AzureNamedKeyCredential("RootManageSharedAccessKey", "<SAS KEY>");
            var options = new EventHubConsumerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            var consumerClient = new EventHubConsumerClient(EventHubConsumerClient.DefaultConsumerGroupNam
e, "eventhubbackend.servicebus.windows.net", eventHubName, credentials, options);

            try
            {
                await ReceiveEvent(consumerClient, "SAS authentication");
            }
            finally
            {
                await consumerClient.CloseAsync();
            }
        }

        private static async Task ReceiveUsingAADApp()
        {
            // Create a producer client that you can use to send events to an event hub
            var options = new EventHubConsumerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            var clientSecretCredential = new ClientSecretCredential("<TenantID>",
"<AAD APP ID>", "<AAD SECRET>");
```

```
            var consumerClient = new EventHubConsumerClient(EventHubConsumerClient.DefaultConsumerGroupNam
e, "eventhubbackend.servicebus.windows.net", eventHubName, clientSecretCredential, options);

            try
            {
                await ReceiveEvent(consumerClient, "AAD authentication");
            }
            finally
            {
                await consumerClient.CloseAsync();
            }
        }

        private static async Task ReceiveUsingAADAppWithWebSockets()
        {
            // Create a producer client that you can use to send events to an event hub
            var options = new EventHubConsumerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            options.ConnectionOptions.TransportType = EventHubsTransportType.AmqpWebSockets;
            var clientSecretCredential = new ClientSecretCredential("<TenantID>", "<AAD APP ID>", "<AAD SE
CRET>");
            var consumerClient = new EventHubConsumerClient(EventHubConsumerClient.DefaultConsumerGroupNam
e, "eventhubbackend.servicebus.windows.net", eventHubName, clientSecretCredential, options);

            try
            {
                await ReceiveEvent(consumerClient, "WebSocket AAD authentication");
            }
            finally
            {
                await consumerClient.CloseAsync();
            }
        }

        private static async Task ReceiveUsingManagedIdentity()
        {
            // Create a producer client that you can use to send events to an event hub
            var options = new EventHubConsumerClientOptions();

            // appgwdevtest.appgwtest.cloudapp.net is pointing to Application Gateway
            // The eh:// can be any string. SDK requires something to be presented
            options.ConnectionOptions.CustomEndpointAddress = new Uri("eh://appgwdevtest.appgwtest.cloudap
p.net:5671");
            var consumerClient = new EventHubConsumerClient(EventHubConsumerClient.DefaultConsumerGroupNam
e, "eventhubbackend.servicebus.windows.net", eventHubName, new DefaultAzureCredential(), options);

            try
            {
                await ReceiveEvent(consumerClient, "Managed Identity authentication");
            }
            finally
            {
                await consumerClient.CloseAsync();
            }
        }

        private static async Task ReceiveEvent(EventHubConsumerClient consumerClient, string authOption)
        {
            var eventsRead = 0;
            // Use the producer client to send the batch of events to the event hub
            await foreach (PartitionEvent partitionEvent in consumerClient.ReadEventsAsync())
            {
                string readFromPartition = partitionEvent.Partition.PartitionId;
                byte[] eventBodyBytes = partitionEvent.Data.EventBody.ToArray();

                var readEvent = string.Empty;
                foreach (var utf8Byte in eventBodyBytes)
                {
                    readEvent += (char)utf8Byte;
                }

                Console.WriteLine($"Read event: {readEvent}");
                eventsRead++;
                if (eventsRead >= 3)
                {
                    Console.WriteLine($"A batch of 3 events has been read using {authOption}");
                    break;
                }
            }
        }

        private static async Task SendEvent(EventHubProducerClient producerClient, string eventName)
        {
            // Create a batch of events
            using EventDataBatch eventBatch = await producerClient.CreateBatchAsync();
            for (int i = 1; i <= 3; i++)
            {
                if (!eventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes($"{eventName}{i}"))))
                {
                    // if it is too large for the batch
                    throw new Exception($"Event {i} is too large for the batch and cannot be sent.");
                }
            }

            // Use the producer client to send the batch of events to the event hub
            await producerClient.SendAsync(eventBatch);
            Console.WriteLine($"A batch of 3 events has been published using {eventName} and custom endpoi
nt address.");
        }
    }
}
```

Callouts/Caveats

1. Although the Private Endpoint configuration is not covered in this doc, the configuration test includes the Private Link for Event Hubs.

2. This document assumes that the Private DNS is configured in the Application Gateway virtual network. This Private DNS contains the required A record for <accountname>.privatelink.servicebus.windows.net

3. While configuring Private Link from clients other than portal, the subnet in which the endpoint would reside must have "privateLinkServiceNetworkPolicies" property disabled. More on this here.

4. The DNS of the custom domain should point to the Application Gateway public IP.

5. Due to current limitation with Application Gateway, any PaaS service's default FQDN should be added to the backend pool **after** configuring the private link, otherwise the default FQDN continues to point to its public IP. [ETA for this fix: Early 2022. As a workaround, you can even perform any PUT operation on gateway which will refresh its DNS.]

ETA

Private Preview: 15 Feb, 2022

---------------------------------------End of File----------------------------------