Secure MQTT broker communication using BrokerListener

Article • 11/19/2024

Important

This page includes instructions for managing Azure IoT Operations components using Kubernetes deployment manifests, which is in **preview**. This feature is provided with several limitations, and shouldn't be used for production workloads.

See the Supplemental Terms of Use for Microsoft Azure Previews for legal terms that apply to Azure features that are in beta, preview, or otherwise not yet released into general availability.

A BrokerListener corresponds to a network endpoint that exposes the broker to clients over the network. You can have one or more BrokerListener resources for each Broker, with multiple ports and different access control on each.

Each BrokerListener port can have its own authentication and authorization rules that define who can connect on that listener port and what actions they can perform on the broker. You can use BrokerAuthentication and BrokerAuthorization resources to specify the access control policies for each listener port. This flexibility allows you to fine-tune the permissions and roles for your MQTT clients based on their needs and use cases.

Tip

The default MQTT broker deployment is a cluster IP service that requires clients to connect with TLS and authenticate with service account tokens. Clients connecting from outside the cluster need extra configuration before they can connect.

Broker listeners have the following characteristics:

- **Name**: Name of the listener. This name is also the Kubernetes service name unless overridden.
- **Service type**: You can have up to three listeners, one per service type. The default listener is service type `ClusterIp`.

- **Ports**: Each listener supports multiple ports. Ports can't conflict over different listeners.
- **Authentication and Authorization references**: BrokerAuthentication and BrokerAuthorization are configured per port.
- **TLS**: Manual or automatic TLS configuration is applied per port.
- **Protocol**: MQTT over WebSockets can be enabled per port.

For a list of all available settings, see the Broker Listener API reference.

Default BrokerListener

When you deploy Azure IoT Operations, the deployment creates a BrokerListener resource named `default`. This listener is used for encrypted internal communication between Azure IoT Operations components. The default listener is part of the default Broker.

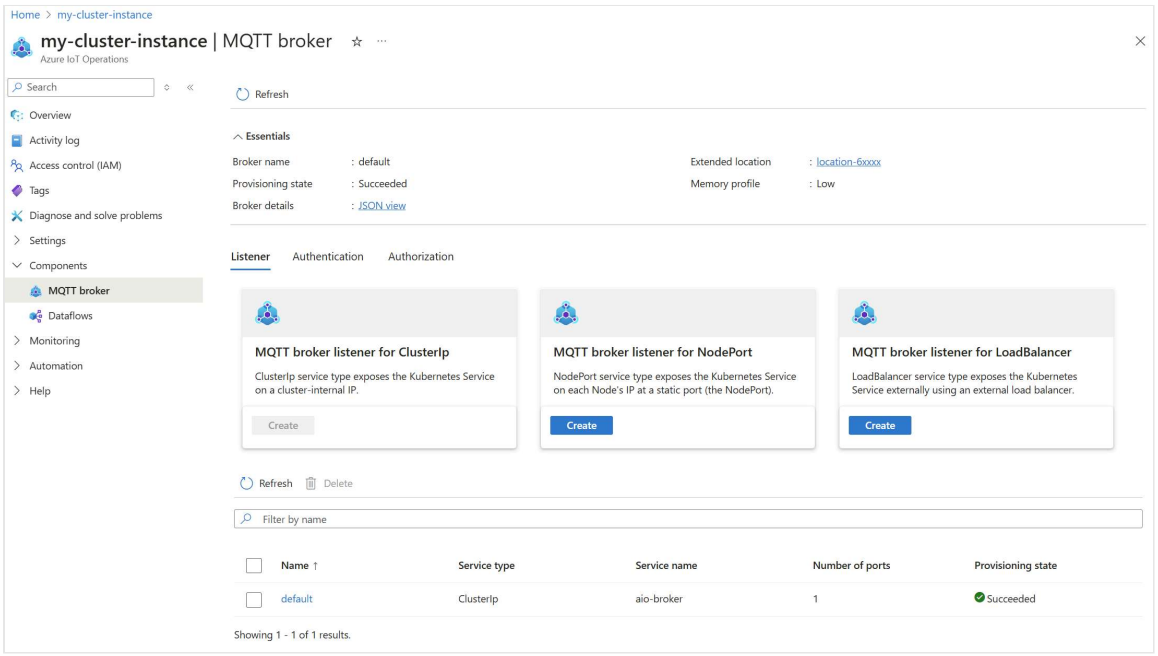
- It exposes a ClusterIp service on port 18883
- It requires clients to use Kubernetes service account authentication
- It has an automatically managed TLS certificate
- It doesn't configure any client authorization policies

> **⊗ Caution**
>
> To avoid disrupting internal Azure IoT Operations communication, keep the default listener unchanged and dedicated for internal use. For external communication, **create a new listener**. If you need to use the ClusterIp service, add more ports to the default listener without changing any of the existing settings.

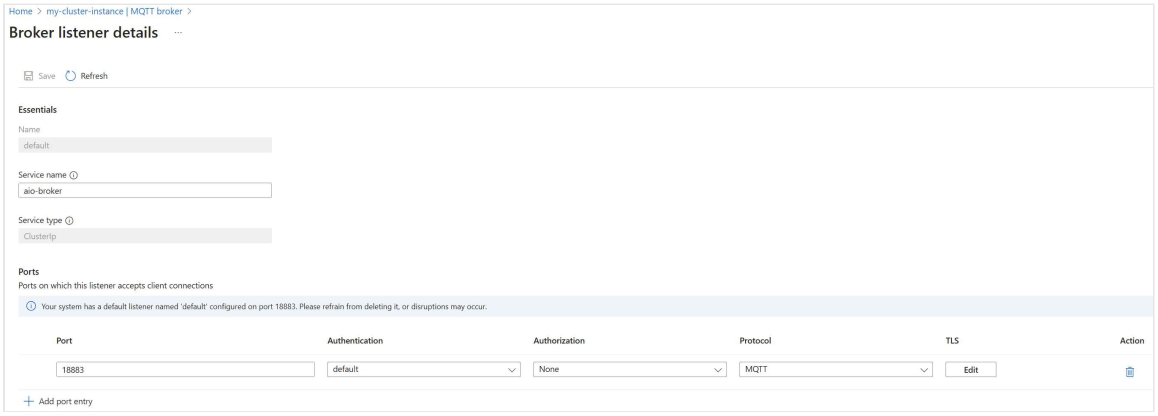
To view or edit the default listener:

Portal

1. In the Azure portal, navigate to your IoT Operations instance.
2. Under **Components**, select **MQTT Broker**.



3. From the broker listener list, select the **default** listener.



4. Review the listener settings, but avoid modifying any of the existing settings. Instead, create a new port and configure it as needed.

Avoid modifying default broker listener

To prevent disrupting internal Azure IoT Operations communication, keep the default listener unchanged and dedicated for internal use. For external communication, create a new listener.

Since the default broker listener uses the service type ClusterIp, and you can have only one listener per service type, add more ports to the default listener without changing any of the existing settings if you need to use the ClusterIp service.

Create new broker listeners

To create a new listener, specify the following settings:

- **Name**: Name of the listener. This name is also the Kubernetes service name unless overridden.
- **Service type**: Type of the Kubernetes service. See Service type.
- **Ports**: List of ports that to listen on. See Ports.
- (Optional) **Service name**: override the Kubernetes service name. See Service name.

Example: Create a new listener with two ports

This example shows how to create a new listener with load balancer service type. The BrokerListener resource defines two ports that accept MQTT connections from clients.

- The first port listens on port 1883 without TLS and authentication. This setup is suitable for testing only. Don't use this configuration in production.
- The second port listens on port 8883 with TLS and authentication enabled. Only authenticated clients with a Kubernetes service account token can connect. TLS is set to automatic mode, using cert-manager to manage the server certificate from the default issuer. This setup is closer to a production configuration.

Portal

1. In the Azure portal, navigate to your IoT Operations instance.
2. Under **Components**, select **MQTT Broker**.
3. Select **MQTT broker listener for LoadBalancer** > **Create**.

   Enter the following settings:

Expand table

Setting	Description
Name	Name of the BrokerListener resource.
Service name	Name of the Kubernetes service. Leave empty to use the listener name as service name.

Setting	Description
Service type	**LoadBalancer** already selected.

4. Under **Ports**, enter the following settings for the first port:

Expand table

Setting	Description
Port	Enter 1883
Authentication	Choose **None**
Authorization	Choose **None**
Protocol	Choose **MQTT**
TLS	Don't add

5. Select **Add port entry** to add a second port and enter the following settings:

Expand table

Setting	Description
Port	Enter 8883
Authentication	Choose **default**
Authorization	Choose **None**
Protocol	Choose **MQTT**
TLS	Select **Add**

6. In the **TLS configuration** pane, enter the following settings:

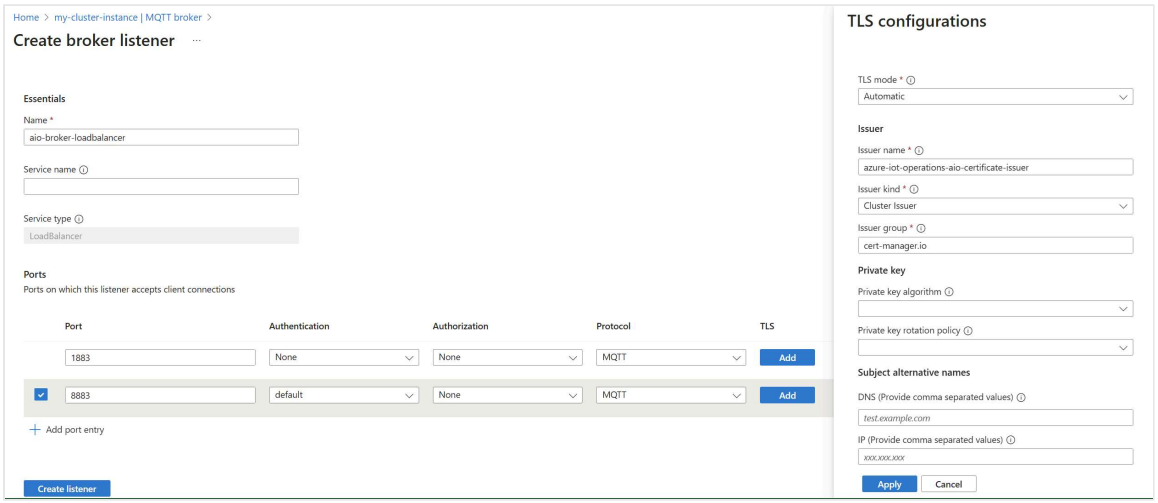
Expand table

Setting	Description
TLS Mode	Choose **Automatic**
Issuer name	Enter `azure-iot-operations-aio-certificate-issuer`

Setting	Description
Issuer kind	Choose **ClusterIssuer**

Leave other settings as default, and select **Apply**.

7. Select **Create listener**.



Service type

Each BrokerListener maps to a Kubernetes service . The service type determines how the broker is exposed to the network. The supported service types are:

- **ClusterIp**: Exposes the broker on a cluster-internal IP. Clients can connect to the broker from within the cluster. This is the default service type for the default listener.
- **NodePort**: Exposes the broker on each node's IP at a static port. Clients can connect to the broker from outside the cluster. This service type is useful for development and testing.
- **LoadBalancer**: Exposes the broker externally. The service is assigned an external IP address that clients can use to connect to the broker. This is the most common service type for production deployments.

Only one listener per service type

Only one listener per service type is allowed. If you need more connectivity of the same service type, add more ports to the existing listener of that service type.

Service name

The service name is the name of the Kubernetes service associated with the broker. If not specified, the broker listener name is used as the service name. The service name must be unique within the namespace.

> **Tip**
>
> To prevent management overhead, we recommend leaving the service name empty. The listener name is unique and can be used to identify the service. Use the service name as an override only when you can't name the service after the listener.

Ports

Each listener can have multiple ports, and each port can have its own settings for authentication, authorization, protocol, and TLS.

To use your own authentication or authorization setting for a port, you must create the corresponding resources before using it with a listener. For more information, see Configure MQTT broker authentication and Configure MQTT broker authorization.

To use TLS, see Configure TLS with automatic certificate management or Configure TLS with manual certificate management sections.

Using same port across listeners

Using the same port number across different listeners isn't supported. Each port number must be unique within the Azure IoT Operations MQTT broker.

For example, if you have a listener using port 1883, you can't create another listener with port 1883. Similarly, the default listener uses port 18883, so you can't create another listener with port 18883.

WebSockets support

Azure IoT Operations MQTT broker supports MQTT over WebSockets. To enable WebSockets, set the protocol to `WebSockets` for the port.

Configure TLS with automatic certificate management

To enable TLS with automatic certificate management, specify the TLS settings on a listener port.

Verify cert-manager installation

With automatic certificate management, you use cert-manager to manage the TLS server certificate. By default, cert-manager is installed alongside Azure IoT Operations in the `cert-manager` namespace already. Verify the installation before proceeding.

1. Use `kubectl` to check for the pods matching the cert-manager app labels.

   Bash

   ```
   kubectl get pods --namespace cert-manager -l 'app in (cert-manager,cainjector,webhook)'
   ```

   Output

   ```
   NAME                                       READY   STATUS    RESTARTS
   AGE
   aio-cert-manager-64f9548744-5fwdd          1/1     Running   4 (145m
   ago)   4d20h
   aio-cert-manager-cainjector-6c7c546578-p6vgv   1/1     Running   4 (145m
   ago)   4d20h
   aio-cert-manager-webhook-7f676965dd-8xs28      1/1     Running   4 (145m
   ago)   4d20h
   ```

2. If you see the pods shown as ready and running, cert-manager is installed and ready to use.

> **Tip**
>
> To further verify the installation, check the cert-manager documentation verify installation . Remember to use the `cert-manager` namespace.

Create an Issuer for the TLS server certificate

The cert-manager *Issuer* resource defines how certificates are automatically issued. Cert-manager supports several Issuers types natively. It also supports an external issuer type for extending functionality beyond the natively supported issuers. MQTT broker can be used with any type of cert-manager issuer.

> **Important**
>
> During initial deployment, Azure IoT Operations is installed with a default Issuer for TLS server certificates. You can use this issuer for development and testing. For more information, see **Default root CA and issuer with Azure IoT Operations**. The steps below are only required if you want to use a different issuer.

The approach to create the issuer is different depending on your scenario. The following sections list examples to help you get started.

Development or test

The CA issuer is useful for development and testing. It must be configured with a certificate and private key stored in a Kubernetes secret.

Set up the root certificate as a Kubernetes secret

If you have an existing CA certificate, create a Kubernetes secret with the CA certificate and CA private key PEM files. Run the following command to import the CA certificate as a Kubernetes secret and skip the next section.

Bash

```
kubectl create secret tls test-ca --cert tls.crt --key tls.key -n azure-iot-operations
```

If you don't have a CA certificate, cert-manager can generate a CA certificate for you. Using cert-manager to generate a CA certificate is known as bootstrapping a CA issuer with a self-signed certificate.

1. Start by creating `ca.yaml`:

   YAML

```yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: selfsigned-ca-issuer
  namespace: azure-iot-operations
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: selfsigned-ca-cert
  namespace: azure-iot-operations
spec:
  isCA: true
  commonName: test-ca
  secretName: test-ca
  issuerRef:
    # Must match Issuer name above
    name: selfsigned-ca-issuer
    # Must match Issuer kind above
    kind: Issuer
    group: cert-manager.io
  # Override default private key config to use an EC key
  privateKey:
    rotationPolicy: Always
    algorithm: ECDSA
    size: 256
```

2. Create the self-signed CA certificate with the following command:

Bash

```
kubectl apply -f ca.yaml
```

Cert-manager creates a CA certificate using its defaults. The properties of this certificate can be changed by modifying the Certificate specification. See cert-manager documentation for a list of valid options.

Distribute the root certificate

The prior example stores the CA certificate in a Kubernetes secret called `test-ca`. The certificate in PEM format can be retrieved from the secret and stored in a file `ca.crt` with the following command:

Bash

```
kubectl get secret test-ca -n azure-iot-operations -o json | jq -r
'.data["tls.crt"]' | base64 -d > ca.crt
```

This certificate must be distributed and trusted by all clients. For example, use the `--cafile` flag for a mosquitto client.

Create issuer based on CA certificate

Cert-manager needs an issuer based on the CA certificate generated or imported in the earlier step. Create the following file as `issuer-ca.yaml`:

YAML

```yaml
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: my-issuer
  namespace: azure-iot-operations
spec:
  ca:
    # Must match secretName of generated or imported CA cert
    secretName: test-ca
```

Create the issuer with the following command:

Bash

```
kubectl apply -f issuer-ca.yaml
```

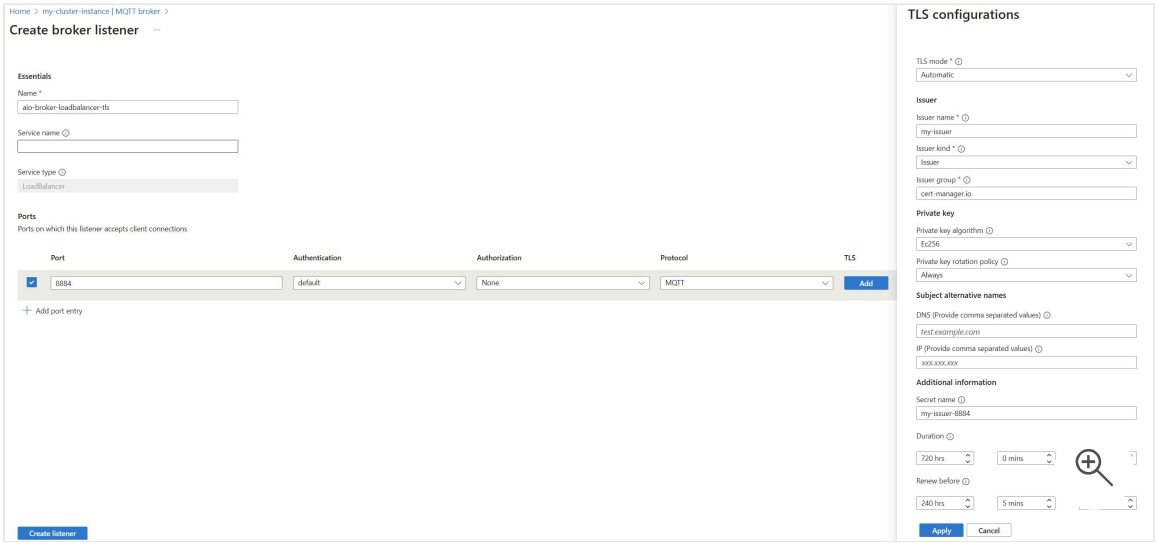
The prior command creates an issuer for issuing TLS server certificates. Note the name and kind of the issuer. In the example, the name is `my-issuer` and the kind is `Issuer`. These values are set in the BrokerListener resource later.

Enable TLS automatic certificate management for a port

The following is an example of a BrokerListener resource that enables TLS on port 8884 with automatic certificate management.

Portal

1. In the Azure portal, go to your IoT Operations instance.
2. Under **Components**, select **MQTT Broker**.
3. Select or create a listener. You can only create one listener per service type. If you already have a listener of the same service type, you can add more ports to the existing listener.
4. You can add TLS settings to the listener by selecting the **TLS** on an existing port or by adding a new port.



Enter the following settings:

Expand table

Setting	Description
Name	Name of the BrokerListener resource. Enter `aio-broker-loadbalancer-tls`.
Port	Port number on which the BrokerListener listens for MQTT connections. Enter 8884.
Authentication	The authentication resource reference.
Authorization	The authorization resource reference.
TLS	Select the *Add* button.

Setting	Description
Issuer name	Name of the cert-manager issuer. Required.
Issuer kind	Kind of the cert-manager issuer. Required.
Issuer group	Group of the cert-manager issuer. Required.
Private key algorithm	Algorithm for the private key.
Private key rotation policy	Policy for rotating the private key.
DNS names	DNS subject alternate names for the certificate.
IP addresses	IP addresses of the subject alternate names for the certificate.
Secret name	Kubernetes secret containing an X.509 client certificate.
Duration	Total lifetime of the TLS server certificate Defaults to 90 days.
Renew before	When to begin renewing the certificate.

5. Select **Apply** to save the TLS settings.

Once the BrokerListener resource is configured, MQTT broker automatically creates a new service with the specified port and TLS enabled.

Optional: Configure server certificate parameters

The only required parameters are Issuer name and Issuer kind. All other properties of the generated TLS server certificates are automatically chosen. However, MQTT broker allows certain properties to be customized following the same syntax as cert-manager Certificates. For example, you can specify the private key algorithm and rotation policy. These settings are under `tls.certManagerCertificateSpec` or the **TLS configuration** pane in the Azure portal.

For a full list of these settings, see Broker Listener CertManagerCertificateSpec API reference.

Verify deployment

Use kubectl to check that the service associated with the BrokerListener resource is running. From the example above, the service name is `aio-broker-loadbalancer-tls` and the namespace is `azure-iot-operations`. The following command checks the service status:

Console

```
$ kubectl get service my-new-tls-listener -n azure-iot-operations
NAME                            TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)          AGE
aio-broker-loadbalancer-tls     LoadBalancer   10.X.X.X        172.X.X.X
8884:32457/TCP   33s
```

Connect to the broker with TLS

Once the server certificate is configured, TLS is enabled. To test with mosquitto:

Bash

```
mosquitto_pub -h $HOST -p 8884 -V mqttv5 -i "test" -t "test" -m "test" --cafile
ca.crt
```

The `--cafile` argument enables TLS on the mosquitto client and specifies that the client should trust all server certificates issued by the given file. You must specify a file that contains the issuer of the configured TLS server certificate.

Replace `$HOST` with the appropriate host:

- If connecting from within the same cluster, replace with the service name given (`my-new-tls-listener` in example) or the service `CLUSTER-IP`.
- If connecting from outside the cluster, the service `EXTERNAL-IP`.

Remember to specify authentication methods if needed.

Default root CA and issuer

To help you get started, Azure IoT Operations is deployed with a default "quickstart" CA certificate and issuer for TLS server certificates. You can use this issuer for development and testing. For more information, see Default root CA and issuer for TLS server certificates.

For production, you must configure a CA issuer with a certificate from a trusted CA, as described in the previous sections.

Configure TLS with manual certificate management

To manually configure MQTT broker to use a specific TLS certificate, specify it in a BrokerListener resource with a reference to a Kubernetes secret and deploy it using kubectl. This article shows an example that configures TLS with self-signed certificates for testing.

Create certificate authority with Step CLI

Step is a certificate manager that can quickly get you up and running when creating and managing your own private CA.

1. Install Step CLI and create a root certificate authority (CA) certificate and key.

   Bash

   ```
   step certificate create --profile root-ca "Example Root CA" root_ca.crt
   root_ca.key
   ```

2. Create an intermediate CA certificate and key signed by the root CA.

   Bash

   ```
   step certificate create --profile intermediate-ca "Example Intermediate
   CA" intermediate_ca.crt intermediate_ca.key \
   --ca root_ca.crt --ca-key root_ca.key
   ```

Create server certificate

Use Step CLI to create a server certificate from the signed by the intermediate CA.

Bash

```
step certificate create mqtts-endpoint mqtts-endpoint.crt mqtts-endpoint.key \
--profile leaf \
--not-after 8760h \
--san mqtts-endpoint \
```

```
--san localhost \
--ca intermediate_ca.crt --ca-key intermediate_ca.key \
--no-password --insecure
```

Here, `mqtts-endpoint` and `localhost` are the Subject Alternative Names (SANs) for MQTT broker's frontend in Kubernetes and local clients, respectively. To connect over the Internet, add a `--san` with an external IP. The `--no-password --insecure` flags are used for testing to skip password prompts and disable password protection for the private key because it's stored in a Kubernetes secret. For production, use a password and store the private key in a secure location like Azure Key Vault.

Certificate key algorithm requirements

Both EC and RSA keys are supported, but all certificates in the chain must use the same key algorithm. If you import your own CA certificates, ensure that the server certificate uses the same key algorithm as the CAs.

Import server certificate chain as a Kubernetes secret

1. Create a full server certificate chain, where the order of the certificates matters: the server certificate is the first one in the file, the intermediate is the second.

   Bash

   ```
   cat  mqtts-endpoint.crt intermediate_ca.crt  > server_chain.crt
   ```

2. Create a Kubernetes secret with the server certificate chain and server key using kubectl.

   Bash

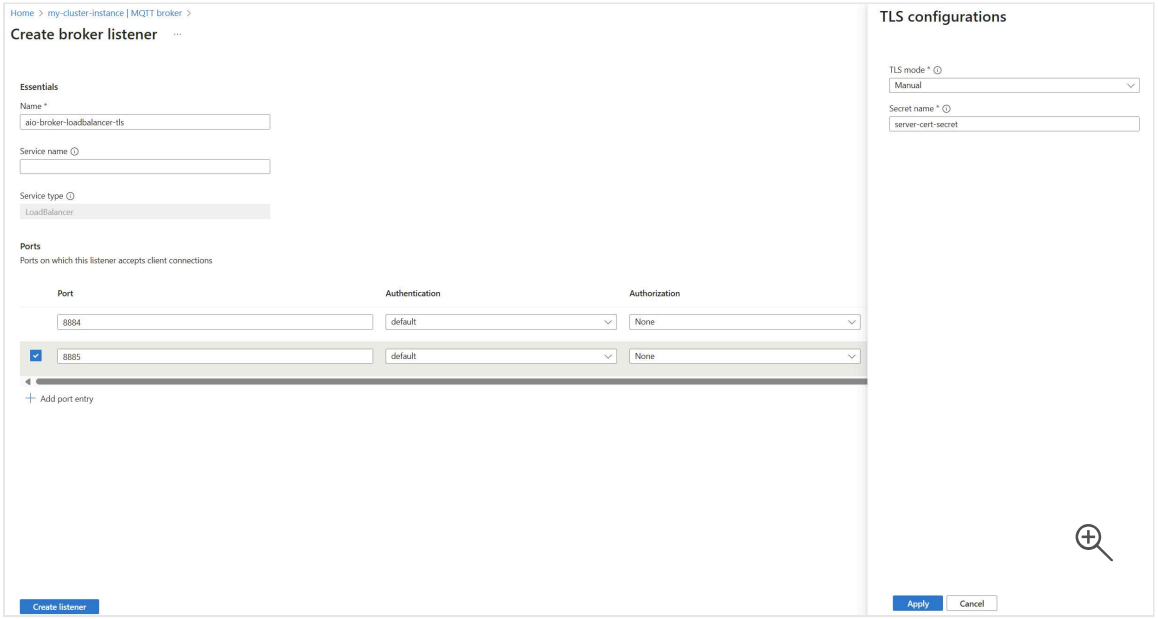
   ```
   kubectl create secret tls server-cert-secret -n azure-iot-operations \
   --cert server_chain.crt \
   --key mqtts-endpoint.key
   ```

Enable TLS manual certificate management for a port

The following is an example of a BrokerListener resource that enables TLS on port 8884 with manual certificate management.

Portal

1. In the Azure portal, navigate to your IoT Operations instance.
2. Under **Components**, select **MQTT Broker**.
3. Select or create a listener. You can only create one listener per service type. If you already have a listener of the same service type, you can add more ports to the existing listener. To follow the example, specify the listener service name as `mqtts-endpoint`.
4. You can add TLS settings to the listener by selecting the **TLS** on an existing port or by adding a new port.



Enter the following settings:

Expand table

Setting	Description
Port	Port number on which the BrokerListener listens for MQTT connections. Required.
Authentication	The authentication resource reference.
Authorization	The authorization resource reference.
TLS	Select the *Add* button.

Setting	Description
Secret name	Kubernetes secret containing an X.509 client certificate.

5. Select **Apply** to save the TLS settings.

Connect to the broker with TLS

To test the TLS connection with mosquitto client, publish a message and pass the root CA certificate in the parameter `--cafile`.

Bash

```
mosquitto_pub -d -h localhost -p 8885 -i "my-client" -t "test-topic" -m
"Hello" --cafile root_ca.crt
```

Remember to specify username, password, etc. if MQTT broker authentication is enabled.

Output

```
Client my-client sending CONNECT
Client my-client received CONNACK (0)
Client my-client sending PUBLISH (d0, q0, r0, m1, 'test-topic', ... (5 bytes))
Client my-client sending DISCONNECT
```

Tip

To use localhost, the port must be available on the host machine. For example, `kubectl port-forward svc/mqtts-endpoint 8885:8885 -n azure-iot-operations`. With some Kubernetes distributions like K3d, you can add a forwarded port with `k3d cluster edit $CLUSTER_NAME --port-add 8885:8885@loadbalancer`.

Note

To connect to the broker, you need to distribute root of trust, also known as trust bundle, to all clients. In this case, the root of trust is the self-signed root CA created by Step CLI. Distribution of root of trust is required for the client to verify the server

certificate chain. If your MQTT clients are workloads on the Kubernetes cluster you also need to create a ConfigMap with the root CA and mount it in your Pod.

Use external IP for the server certificate

To connect with TLS over the Internet, MQTT broker's server certificate must have its external hostname or IP address as a SAN. In production, this is usually a DNS name or a well-known IP address. However, during dev/test, you might not know what hostname or external IP is assigned before deployment. To solve this, deploy the listener without the server certificate first, create the server certificate and secret with the external IP, and import the secret to the listener.

If you try to deploy the example TLS listener `manual-tls-listener` but the referenced Kubernetes secret `server-cert-secret` doesn't exist, the associated service gets created, but the pods don't start. The service is created because the operator needs to reserve the external IP for the listener.

Bash

```
kubectl get svc mqtts-endpoint -n azure-iot-operations
```

Output

```
NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
mqtts-endpoint        LoadBalancer   10.X.X.X        172.X.X.X
8885:30674/TCP      1m15s
```

However, this behavior is expected while we import the server certificate. The health manager logs mention MQTT broker is waiting for the server certificate.

Bash

```
kubectl logs -l app=health-manager -n azure-iot-operations
```

Output

```
...
<6>2023-11-06T21:36:13.634Z [INFO] [1] - Server certificate server-cert-secret
```

```
not found. Awaiting creation of secret.
```

> **Note**
>
> Generally, in a distributed system, pods logs aren't deterministic and should be used with caution. The right way for information like this to surface is through Kubernetes events and custom resource status, which is in the backlog. Consider the previous step as a temporary workaround.

Even though the frontend pods aren't up, the external IP is already available.

Bash

```
kubectl get svc mqtts-endpoint -n azure-iot-operations
```

Output

```
NAME                  TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
AGE
mqtts-endpoint        LoadBalancer   10.X.X.X        172.X.X.X
8885:30674/TCP      1m15s
```

From here, follow the same steps as previously to create a server certificate with this external IP in `--san` and create the Kubernetes secret in the same way. Once the secret is created, it's automatically imported to the listener.

Related content

- Configure MQTT broker authorization
- Configure MQTT broker authentication
- Tutorial: TLS, X.509 client authentication, and attribute-based access control (ABAC) authorization

---

Feedback

Was this page helpful? Yes No

Provide product feedback | Get help at Microsoft Q&A