

# Contents

## Azure IoT Hub Device Provisioning Service Documentation

### Overview

[What is IoT Hub Device Provisioning Service?](#)

### Quickstarts

[Set up a DPS](#)

[Azure portal](#)

[Azure CLI](#)

[Azure RM template](#)

[Provision a single device](#)

[Symmetric Key attestation](#)

[C](#)

[Java](#)

[Python](#)

[X.509 Certificate attestation](#)

[C](#)

[Java](#)

[C#](#)

[Node.js](#)

[Python](#)

[Simulated TPM device attestation](#)

[C](#)

[Java](#)

[C#](#)

[Node.js](#)

[Python](#)

[Create device enrollments using DPS APIs](#)

[Group enrollment using X.509 Certificate Attestation](#)

[Java](#)

[C#](#)

[Node.js](#)

[Python](#)

[Individual device enrollment using TPM attestation](#)

[Java](#)

[C#](#)

[Node.js](#)

[Python](#)

## Tutorials

[1 - Set up cloud resources](#)

[2 - Set up a device](#)

[3 - Provision a device to an IoT hub](#)

[C](#)

[C#](#)

[4 - Provision devices to multiple hubs](#)

[5 - Set up group enrollments](#)

## Concepts

[Auto-provisioning](#)

[Reprovisioning](#)

[Devices](#)

[Security](#)

[TLS support](#)

[Virtual networks support](#)

[Service](#)

[Understanding DPS IP addresses](#)

[TPM attestation](#)

[Symmetric key attestation](#)

[Security practices for device manufacturers](#)

## How-to guides

[Connect MXChip IoT DevKit to IoT Hub](#)

[Manage Device Provisioning Service configuration](#)

[Manage enrollments - Portal](#)

[Manage enrollments - Service SDKs](#)

- [Configure verified CA certificates](#)
- [Roll device certificates](#)
- [Reprovision devices](#)
- [Manage disenrollment](#)
- [Manage deprovisioning](#)
- [Manage Device Provisioning Service](#)
  - [Configure Device Provisioning Service using Azure CLI](#)
  - [Control access to Provisioning Service APIs](#)
  - [Configure IP filtering](#)
- [Provision IoT Edge devices](#)
  - [Linux](#)
  - [Windows](#)
- [Use attestation mechanisms with SDK](#)
- [Provision for multitenancy](#)
- [Use custom allocation policies](#)
- [Use symmetric keys with legacy devices](#)
- [Use SDK tools for development](#)
- [How to send additional data from devices](#)
- [Troubleshooting](#)
- [Communicate with your DPS using MQTT protocol](#)

## Reference

- [REST API](#)
- [Resource Manager template](#)
- [Azure IoT SDK for C](#)
- [Azure IoT SDK for Python](#)
- [Azure IoT SDK for Node.js](#)
- [Azure IoT SDK for Java](#)
- [Azure IoT SDK for .NET](#)
- [Azure CLI](#)

## Resources

- [Support and help options](#)
- [Azure IoT services](#)

- [IoT Hub](#)
- [IoT Hub Device Provisioning Service](#)
- [IoT Central](#)
- [IoT Edge](#)
- [IoT solution accelerators](#)
- [IoT Plug and Play](#)
- [Azure Maps](#)
- [Time Series Insights](#)
- [Azure IoT SDKs](#)
  - [IoT Service SDKs](#)
  - [IoT Device SDKs](#)
- [Azure IoT samples](#)
  - [C# \(.NET\)](#)
  - [Node.js](#)
  - [Java](#)
  - [Python](#)
  - [iOS Platform](#)
- [Azure Certified for IoT device catalog](#)
- [Azure IoT Developer Center](#)
- [Customer data requests](#)
- [Azure Roadmap](#)
- [Azure IoT Explorer tool](#)
- [iothub-diagnostics tool](#)
- [Pricing](#)
- [Pricing calculator](#)
- [Service updates](#)
- [Technical case studies](#)
- [Videos](#)

# Provisioning devices with Azure IoT Hub Device Provisioning Service

4/20/2020 • 7 minutes to read • [Edit Online](#)

Microsoft Azure provides a rich set of integrated public cloud services for all your IoT solution needs. The IoT Hub Device Provisioning Service (DPS) is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention. DPS enables the provisioning of millions of devices in a secure and scalable manner.

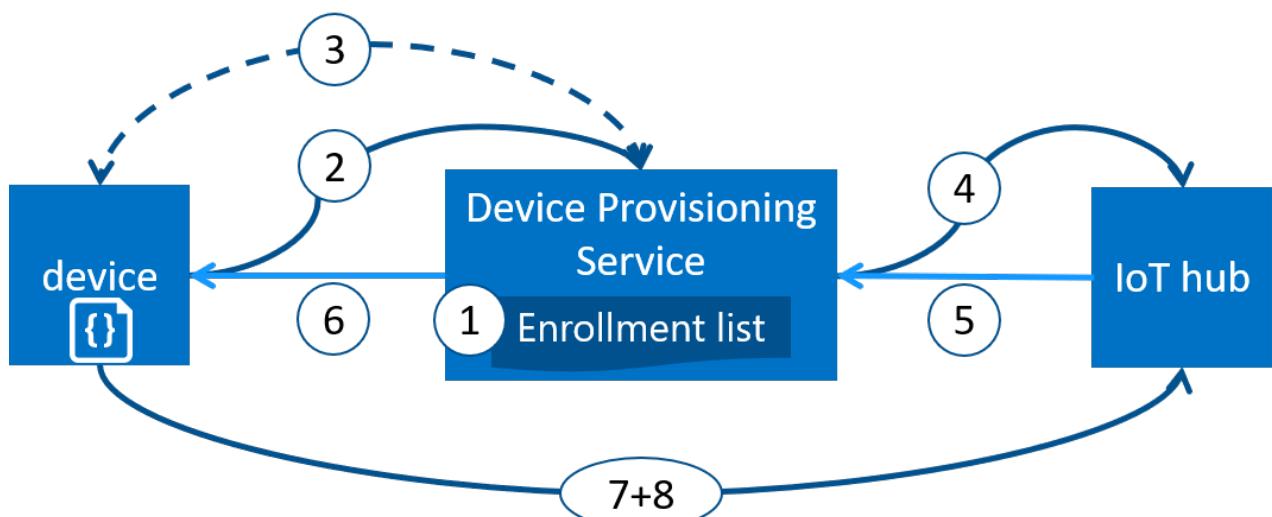
## When to use Device Provisioning Service

There are many provisioning scenarios in which DPS is an excellent choice for getting devices connected and configured to IoT Hub, such as:

- Zero-touch provisioning to a single IoT solution without hardcoding IoT Hub connection information at the factory (initial setup)
- Load-balancing devices across multiple hubs
- Connecting devices to their owner's IoT solution based on sales transaction data (multitenancy)
- Connecting devices to a particular IoT solution depending on use-case (solution isolation)
- Connecting a device to the IoT hub with the lowest latency (geo-sharding)
- Reprovisioning based on a change in the device
- Rolling the keys used by the device to connect to IoT Hub (when not using X.509 certificates to connect)

## Behind the scenes

All the scenarios listed in the previous section can be done using DPS for zero-touch provisioning with the same flow. Many of the manual steps traditionally involved in provisioning are automated with DPS to reduce the time to deploy IoT devices and lower the risk of manual error. The following section describes what goes on behind the scenes to get a device provisioned. The first step is manual, all of the following steps are automated.



1. Device manufacturer adds the device registration information to the enrollment list in the Azure portal.
2. Device contacts the DPS endpoint set at the factory. The device passes the identifying information to DPS to prove its identity.
3. DPS validates the identity of the device by validating the registration ID and key against the enrollment list

- entry using either a nonce challenge ([Trusted Platform Module](#)) or standard X.509 verification (X.509).
4. DPS registers the device with an IoT hub and populates the device's [desired twin state](#).
  5. The IoT hub returns device ID information to DPS.
  6. DPS returns the IoT hub connection information to the device. The device can now start sending data directly to the IoT hub.
  7. The device connects to IoT hub.
  8. The device gets the desired state from its device twin in IoT hub.

## Provisioning process

There are two distinct steps in the deployment process of a device in which DPS takes a part that can be done independently:

- The **manufacturing step** in which the device is created and prepared at the factory, and
- The **cloud setup step** in which the Device Provisioning Service is configured for automated provisioning.

Both these steps fit in seamlessly with existing manufacturing and deployment processes. DPS even simplifies some deployment processes that involve manual work to get connection information onto the device.

### Manufacturing step

This step is all about what happens on the manufacturing line. The roles involved in this step include silicon designer, silicon manufacturer, integrator and/or the end manufacturer of the device. This step is concerned with creating the hardware itself.

DPS does not introduce a new step in the manufacturing process; rather, it ties into the existing step that installs the initial software and (ideally) the HSM on the device. Instead of creating a device ID in this step, the device is programmed with the provisioning service information, enabling it to call the provisioning service to get its connection info/IoT solution assignment when it is switched on.

Also in this step, the manufacturer supplies the device deployer/operator with identifying key information. Supplying that information could be as simple as confirming that all devices have an X.509 certificate generated from a signing certificate provided by the device deployer/operator, or as complicated as extracting the public portion of a TPM endorsement key from each TPM device. These services are offered by many silicon manufacturers today.

### Cloud setup step

This step is about configuring the cloud for proper automatic provisioning. Generally there are two types of users involved in the cloud setup step: someone who knows how devices need to be initially set up (a device operator), and someone else who knows how devices are to be split among the IoT hubs (a solution operator).

There is a one-time initial setup of the provisioning that must occur, which is usually handled by the solution operator. Once the provisioning service is configured, it does not have to be modified unless the use case changes.

After the service has been configured for automatic provisioning, it must be prepared to enroll devices. This step is done by the device operator, who knows the desired configuration of the device(s) and is in charge of making sure the provisioning service can properly attest to the device's identity when it comes looking for its IoT hub. The device operator takes the identifying key information from the manufacturer and adds it to the enrollment list. There can be subsequent updates to the enrollment list as new entries are added or existing entries are updated with the latest information about the devices.

## Registration and provisioning

*Provisioning* means various things depending on the industry in which the term is used. In the context of provisioning IoT devices to their cloud solution, provisioning is a two part process:

1. The first part is establishing the initial connection between the device and the IoT solution by registering the device.
2. The second part is applying the proper configuration to the device based on the specific requirements of the solution it was registered to.

Once both of those two steps have been completed, we can say that the device has been fully provisioned. Some cloud services only provide the first step of the provisioning process, registering devices to the IoT solution endpoint, but do not provide the initial configuration. DPS automates both steps to provide a seamless provisioning experience for the device.

## Features of the Device Provisioning Service

DPS has many features, making it ideal for provisioning devices.

- **Secure attestation** support for both X.509 and TPM-based identities.
- **Enrollment list** containing the complete record of devices/groups of devices that may at some point register. The enrollment list contains information about the desired configuration of the device once it registers, and it can be updated at any time.
- **Multiple allocation policies** to control how DPS assigns devices to IoT hubs in support of your scenarios: Lowest latency, evenly weighted distribution (default), and static configuration via the enrollment list. Latency is determined using the same method as [Traffic Manager](#).
- **Monitoring and diagnostics logging** to make sure everything is working properly.
- **Multi-hub support** allows DPS to assign devices to more than one IoT hub. DPS can talk to hubs across multiple Azure subscriptions.
- **Cross-region support** allows DPS to assign devices to IoT hubs in other regions.
- **Encryption for data at rest** allows data in DPS to be encrypted and decrypted transparently using 256-bit AES encryption, one of the strongest block ciphers available, and is FIPS 140-2 compliant.

You can learn more about the concepts and features involved in device provisioning in [device concepts](#), [service concepts](#), and [security concepts](#).

## Cross-platform support

Just like all Azure IoT services, DPS works cross-platform with a variety of operating systems. Azure offers open-source SDKs in a variety of [languages](#) to facilitate connecting devices and managing the service. DPS supports the following protocols for connecting devices:

- HTTPS
- AMQP
- AMQP over web sockets
- MQTT
- MQTT over web sockets

DPS only supports HTTPS connections for service operations.

## Regions

DPS is available in many regions. The updated list of existing and newly announced regions for all services is at [Azure Regions](#). You can check availability of the Device Provisioning Service on the [Azure Status](#) page.

**NOTE**

DPS is global and not bound to a location. However, you must specify a region in which the metadata associated with your DPS profile will reside.

## Availability

There is a 99.9% Service Level Agreement for DPS, and you can [read the SLA](#). The full [Azure SLA](#) explains the guaranteed availability of Azure as a whole.

## Quotas

Each Azure subscription has default quota limits in place that could impact the scope of your IoT solution. The current limit on a per-subscription basis is 10 Device Provisioning Services per subscription.

The following table lists the limits that apply to Azure IoT Hub Device Provisioning Service resources.

RESOURCE	LIMIT
Maximum device provisioning services per Azure subscription	10
Maximum number of enrollments	1,000,000
Maximum number of registrations	1,000,000
Maximum number of enrollment groups	100
Maximum number of CAs	25
Maximum number of linked IoT hubs	50
Maximum size of message	96 KB

**NOTE**

To increase the number of enrollments and registrations on your provisioning service, contact [Microsoft Support](#).

**NOTE**

Increasing the maximum number of CAs is not supported.

The Device Provisioning Service throttles requests when the following quotas are exceeded.

THROTTLE	PER-UNIT VALUE
Operations	200/min/service
Device registrations	200/min/service
Device polling operation	5/10 sec/device

For more details on quota limits:

- [Azure Subscription Service Limits](#)

## Related Azure components

DPS automates device provisioning with Azure IoT Hub. Learn more about [IoT Hub](#).

## Next steps

You now have an overview of provisioning IoT devices in Azure. The next step is to try out an end-to-end IoT scenario.

[Set up IoT Hub Device Provisioning Service with the Azure portal](#) [Create and provision a simulated device](#) [Set up device for provisioning](#)

# Quickstart: Set up the IoT Hub Device Provisioning Service with the Azure portal

1/15/2020 • 6 minutes to read • [Edit Online](#)

The IoT Hub Device Provisioning Service can be used with IoT Hub to enable zero-touch, just-in-time provisioning to the desired IoT hub without requiring human intervention, enabling customers to provision millions of IoT devices in a secure and scalable manner. Azure IoT Hub Device Provisioning Service supports IoT devices with TPM, symmetric key and X.509 certificate authentications. For more information, please refer to [IoT Hub Device Provisioning Service overview](#)

In this quickstart, you will learn how to set up the IoT Hub Device Provisioning Service in the Azure Portal for provisioning your devices with the following steps:

- Use the Azure portal to create an IoT Hub
- Use the Azure portal to create an IoT Hub Device Provisioning Service and get the ID scope
- Link the IoT hub to the Device Provisioning Service

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Create an IoT hub

This section describes how to create an IoT hub using the [Azure portal](#).

1. Sign in to the [Azure portal](#).
2. From the Azure homepage, select the **+ Create a resource** button, and then enter *IoT Hub* in the **Search the Marketplace** field.
3. Select **IoT Hub** from the search results, and then select **Create**.
4. On the **Basics** tab, complete the fields as follows:
  - **Subscription:** Select the subscription to use for your hub.
  - **Resource Group:** Select a resource group or create a new one. To create a new one, select **Create new** and fill in the name you want to use. To use an existing resource group, select that resource group. For more information, see [Manage Azure Resource Manager resource groups](#).
  - **Region:** Select the region in which you want your hub to be located. Select the location closest to you. Some features, such as [IoT Hub device streams](#), are only available in specific regions. For these limited features, you must select one of the supported regions.
  - **IoT Hub Name:** Enter a name for your hub. This name must be globally unique. If the name you enter is available, a green check mark appears.

### IMPORTANT

Because the IoT hub will be publicly discoverable as a DNS endpoint, be sure to avoid entering any sensitive or personally identifiable information when you name it.

Home > New > IoT hub

## IoT hub

Microsoft

[Basics](#) [Size and scale](#) [Tags](#) [Review + create](#)

Create an IoT Hub to help you connect, monitor, and manage billions of your IoT assets. [Learn more](#)

**Project details**

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

Subscription \* ⓘ Personal IoT items

Resource group \* ⓘ  [Create new](#)

Region \* ⓘ East Asia

IoT hub name \* ⓘ Once your hub is created, this name can't be changed

[Review + create](#) [< Previous](#) [Next: Size and scale >](#) [Automation options](#)

5. Select **Next: Size and scale** to continue creating your hub.

Home > New > IoT hub

## IoT hub

Microsoft

[Basics](#) [Size and scale](#) [Tags](#) [Review + create](#)

Each IoT hub is provisioned with a certain number of units in a specific tier. The tier and number of units determine the maximum daily quota of messages that you can send. [Learn more](#)

**Scale tier and units**

Pricing and scale tier \* ⓘ S1: Standard tier [Learn how to choose the right IoT hub tier for your solution](#)

Number of S1 IoT hub units ⓘ  1  
Determines how your IoT hub can scale. You can change this later if your needs increase.

Azure Security Center  On  
Turn on Azure Security Center for IoT and add an extra layer of threat protection to IoT Hub, IoT Edge, and your devices. [Learn more](#)

Pricing and scale tier ⓘ	S1	Device-to-cloud-messages ⓘ	Enabled
Messages per day ⓘ	400,000	Message routing ⓘ	Enabled
Cost per month	25.00 USD	Cloud-to-device commands ⓘ	Enabled
Azure Security Center ⓘ	0.001 USD per device per month	IoT Edge ⓘ	Enabled
		Device management ⓘ	Enabled

[Advanced settings](#)

[Review + create](#) [< Previous: Basics](#) [Next: Tags >](#) [Automation options](#)

You can accept the default settings here. If desired, you can modify any of the following fields:

- **Pricing and scale tier:** Your selected tier. You can choose from several tiers, depending on how many features you want and how many messages you send through your solution per day. The free tier is intended for testing and evaluation. It allows 500 devices to be connected to the hub and up to 8,000 messages per day. Each Azure subscription can create one IoT hub in the free tier.  
If you are working through a Quickstart for IoT Hub device streams, select the free tier.
- **IoT Hub units:** The number of messages allowed per unit per day depends on your hub's pricing tier. For example, if you want the hub to support ingress of 700,000 messages, you choose two S1 tier units. For details about the other tier options, see [Choosing the right IoT Hub tier](#).
- **Azure Security Center:** Turn this on to add an extra layer of threat protection to IoT and your devices. This option is not available for hubs in the free tier. For more information about this feature, see [Azure Security Center for IoT](#).
- **Advanced Settings > Device-to-cloud partitions:** This property relates the device-to-cloud messages to the number of simultaneous readers of the messages. Most hubs need only four partitions.

6. Select **Next: Tags** to continue to the next screen.

Tags are name/value pairs. You can assign the same tag to multiple resources and resource groups to categorize resources and consolidate billing. For more information, see [Use tags to organize your Azure resources](#).

Name ⓘ	Value ⓘ	Resource	⋮
department	: accounting	IoT Hub	
	:	IoT Hub	

**Review + create**    < Previous: Size and scale    Next: Review + create >    Automation options

7. Select **Next: Review + create** to review your choices. You see something similar to this screen, but with the values you selected when creating the hub.

The screenshot shows the 'Review + create' step of the Azure IoT hub creation wizard. At the top, there are tabs for 'Basics', 'Size and scale', 'Tags', and 'Review + create'. The 'Review + create' tab is highlighted with a red dashed border. Below the tabs, there are two sections: 'Basics' and 'Size and scale'. Under 'Basics', the following details are listed:

Subscription	Personal testing
Resource group	iot-hubs
Region	West US 2
IoT hub name	you-hub-name

Under 'Size and scale', the following details are listed:

Pricing and scale tier	S1
Number of S1 IoT hub units	1
Messages per day	400,000
Cost per month	25.00 USD
Azure Security Center	0.001 USD per device per month

Under 'Tags', the following detail is listed:

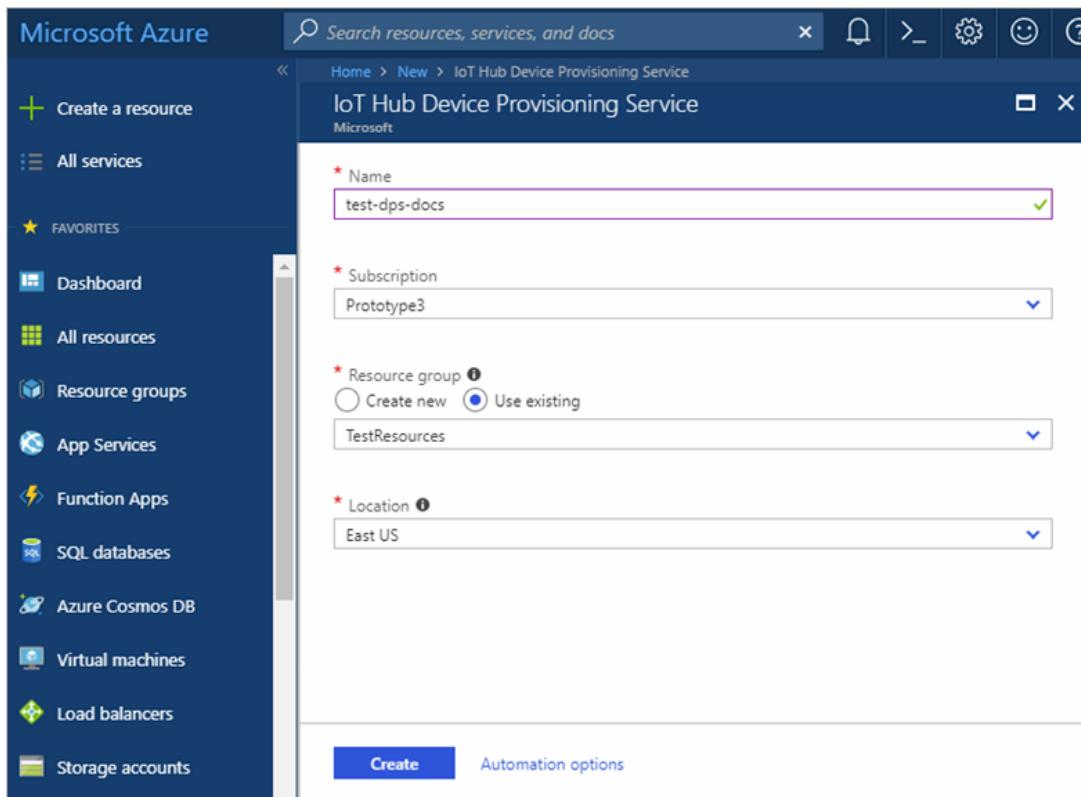
department	accounting
------------	------------

At the bottom of the screen, there are several buttons: a large blue 'Create' button with a red border, a 'Previous: Tags' button, a 'Next >' button, and an 'Automation options' link.

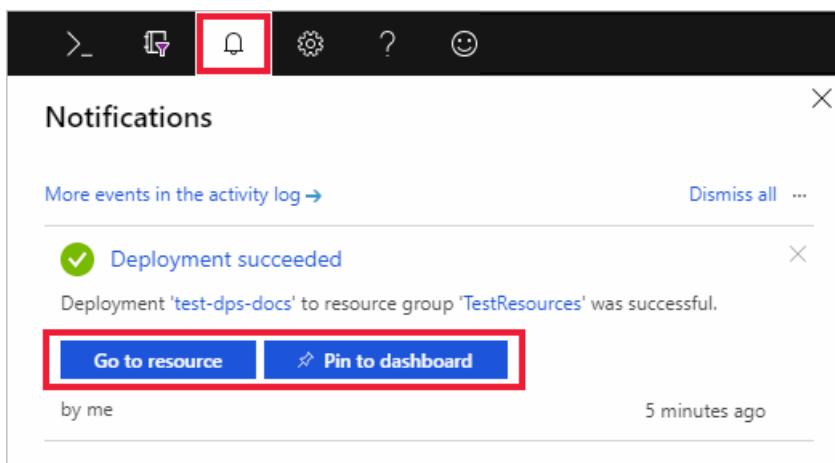
8. Select **Create** to create your new hub. Creating the hub takes a few minutes.

## Create a new IoT Hub Device Provisioning Service

1. Select the **+ Create a resource** button again.
2. *Search the Marketplace* for the **Device Provisioning Service**. Select **IoT Hub Device Provisioning Service** and hit the **Create** button.
3. Provide the following information for your new Device Provisioning Service instance and hit **Create**.
  - **Name:** Provide a unique name for your new Device Provisioning Service instance. If the name you enter is available, a green check mark appears.
  - **Subscription:** Choose the subscription that you want to use to create this Device Provisioning Service instance.
  - **Resource group:** This field allows you to create a new resource group, or choose an existing one to contain the new instance. Choose the same resource group that contains the IoT hub you created above, for example, **TestResources**. By putting all related resources in a group together, you can manage them together. For example, deleting the resource group deletes all resources contained in that group. For more information, see [Manage Azure Resource Manager resource groups](#).
  - **Location:** Select the closest location to your devices.



- Select the notification button to monitor the creation of the resource instance. Once the service is successfully deployed, select **Pin to dashboard**, and then **Go to resource**.



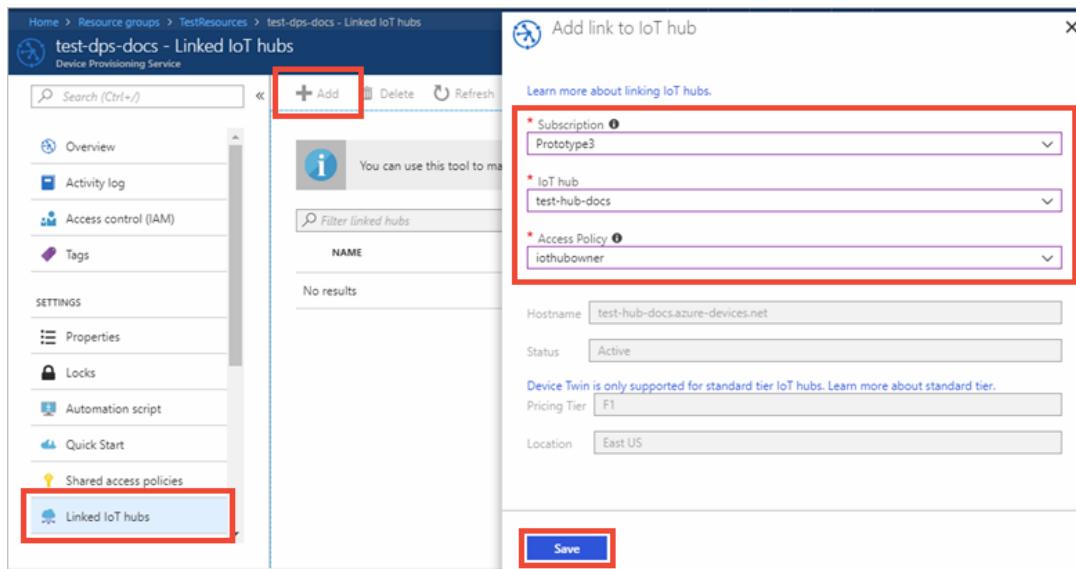
## Link the IoT hub and your Device Provisioning Service

In this section, you will add a configuration to the Device Provisioning Service instance. This configuration sets the IoT hub for which devices will be provisioned.

- Select the **All resources** button from the left-hand menu of the Azure portal. Select the Device Provisioning Service instance that you created in the preceding section.  
If your menu is configured using **Flyout** instead of the **Docked** mode in the portal settings, you will need to click the 3 lines at the top left to open the portal menu on the left.
- From the Device Provisioning Service's menu, select **Linked IoT hubs**. Hit the **+ Add** button seen at the top.
- On the **Add link to IoT hub** page, provide the following information to link your new Device Provisioning Service instance to an IoT hub. Then hit **Save**.
  - Subscription:** Select the subscription containing the IoT hub that you want to link with your new Device Provisioning Service instance.

Device Provisioning Service instance.

- **IoT hub:** Select the IoT hub to link with your new Device Provisioning Service instance.
- **Access Policy:** Select **iothubowner** as the credentials for establishing the link with the IoT hub.



4. Now you should see the selected hub under the **Linked IoT hubs** blade. You might need to hit **Refresh** for it to show up.

## Clean up resources

Other quickstarts in this collection build upon this quickstart. If you plan to continue on to work with subsequent quickstarts or with the tutorials, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart in the Azure portal.

1. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning Service. At the top of the device detail pane, select **Delete**.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. At the top of the hub detail pane, select **Delete**.

## Next steps

In this quickstart, you've deployed an IoT hub and a Device Provisioning Service instance, and linked the two resources. To learn how to use this setup to provision a simulated device, continue to the quickstart for creating a simulated device.

[Quickstart to create a simulated device](#)

# Quickstart: Set up the IoT Hub Device Provisioning Service with Azure CLI

2/20/2020 • 5 minutes to read • [Edit Online](#)

The Azure CLI is used to create and manage Azure resources from the command line or in scripts. This quickstart details using the Azure CLI to create an IoT hub and an IoT Hub Device Provisioning Service, and to link the two services together.

If you don't have an Azure subscription, create a [free account](#) before you begin.

## IMPORTANT

Both the IoT hub and the provisioning service you create in this quickstart will be publicly discoverable as DNS endpoints. Make sure to avoid any sensitive information if you decide to change the names used for these resources.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

## Create a resource group

Create a resource group with the [az group create](#) command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named *my-sample-resource-group* in the *westus* location.

```
az group create --name my-sample-resource-group --location westus
```

**TIP**

The example creates the resource group in the West US location. You can view a list of available locations by running the command `az account list-locations -o table`.

## Create an IoT hub

Create an IoT hub with the [az iot hub create](#) command.

The following example creates an IoT hub named *my-sample-hub* in the *westus* location. An IoT hub name must be globally unique in Azure, so you may want to add a unique prefix or suffix to the example name, or choose a new name altogether. Make sure your name follows proper naming conventions for an IoT hub: it should be 3-50 characters in length, and can contain only upper or lower case alphanumeric characters or hyphens ('-').

```
az iot hub create --name my-sample-hub --resource-group my-sample-resource-group --location westus
```

## Create a Device Provisioning Service

Create a Device Provisioning Service with the [az iot dps create](#) command.

The following example creates a provisioning service named *my-sample-dps* in the *westus* location. You will also need to choose a globally unique name for your own provisioning service. Make sure it follows proper naming conventions for an IoT Hub Device Provisioning Service: it should be 3-64 characters in length and can contain only upper or lower case alphanumeric characters or hyphens ('-').

```
az iot dps create --name my-sample-dps --resource-group my-sample-resource-group --location westus
```

**TIP**

The example creates the provisioning service in the West US location. You can view a list of available locations by running the command

```
az provider show --namespace Microsoft.Devices --query "resourceTypes[?resourceType=='ProvisioningServices'].locations | [0]" --out table
```

or by going to the [Azure Status](#) page and searching for "Device Provisioning Service". In commands, locations can be specified either in one word or multi-word format; for example: *westus*, *West US*, *WEST US*, etc. The value is not case sensitive. If you use multi-word format to specify location, enclose the value in quotes; for example, `--location "West US"`.

## Get the connection string for the IoT hub

You need your IoT hub's connection string to link it with the Device Provisioning Service. Use the [az iot hub show-connection-string](#) command to get the connection string and use its output to set a variable that you will use when you link the two resources.

The following example sets the *hubConnectionString* variable to the value of the connection string for the primary key of the hub's *iothubowner* policy (the `--policy-name` parameter can be used to specify a different policy). Trade out *my-sample-hub* for the unique IoT hub name you chose earlier. The command uses the Azure CLI [query](#) and [output](#) options to extract the connection string from the command output.

```
hubConnectionString=$(az iot hub show-connection-string --name my-sample-hub --key primary --query connectionstring -o tsv)
```

You can use the `echo` command to see the connection string.

```
echo $hubConnectionString
```

#### NOTE

These two commands are valid for a host running under Bash. If you are using a local Windows/CMD shell or a PowerShell host, you need to modify the commands to use the correct syntax for that environment.

## Link the IoT hub and the provisioning service

Link the IoT hub and your provisioning service with the `az iot dps linked-hub create` command.

The following example links an IoT hub named *my-sample-hub* in the *westus* location and a Device Provisioning Service named *my-sample-dps*. Trade out these names for the unique IoT hub and Device Provisioning Service names you chose earlier. The command uses the connection string for your IoT hub that was stored in the *hubConnectionString* variable in the previous step.

```
az iot dps linked-hub create --dps-name my-sample-dps --resource-group my-sample-resource-group --connection-string $hubConnectionString --location westus
```

The command may take a few minutes to complete.

## Verify the provisioning service

Get the details of your provisioning service with the `az iot dps show` command.

The following example gets the details of a provisioning service named *my-sample-dps*. Trade out this name for your own Device Provisioning Service name.

```
az iot dps show --name my-sample-dps
```

The linked IoT hub is shown in the *properties.iotHubs* collection.

```
user@Azure:~$ az iot dps show --name my-sample-dps
{
  "etag": "AAAAAAAGPDJY=",
  "id": "/subscriptions/<subscription ID>/resourceGroups/my-sample-resource-group/providers/Microsoft.Devices/provisioningServices/my-sample-dps",
  "location": "westus",
  "name": "my-sample-dps",
  "properties": {
    "allocationPolicy": "Hashed",
    "authorizationPolicies": null,
    "deviceProvisioningHostName": "global.azure-devices-provisioning.net",
    "idScope": "0ne000956C5",
    "iotHubs": [
      {
        "allocationWeight": null,
        "applyAllocationPolicy": null,
        "connectionString": "HostName=my-sample-hub.azure-devices.net;SharedAccessKeyName=iothubowner;SharedAccessKey=*****",
        "location": "westus",
        "name": "my-sample-hub.azure-devices.net"
      }
    ],
    "provisioningState": null,
    "serviceOperationsHostName": "my-sample-dps.azure-devices-provisioning.net",
    "state": "Active"
  },
  "resourcegroup": "my-sample-resource-group",
  "sku": {
    "capacity": 1,
    "name": "S1",
    "tier": "Standard"
  },
  "subscriptionid": "<subscription ID>",
  "tags": {},
  "type": "Microsoft.Devices/provisioningServices"
}
```

## Clean up resources

Other quickstarts in this collection build upon this quickstart. If you plan to continue on to work with subsequent quickstarts or with the tutorials, do not clean up the resources created in this quickstart. If you do not plan to continue, you can use the following commands to delete the provisioning service, the IoT hub or the resource group and all of its resources. Replace the names of the resources written below with the names of your own resources.

To delete the provisioning service, run the [az iot dps delete](#) command:

```
az iot dps delete --name my-sample-dps --resource-group my-sample-resource-group
```

To delete the IoT hub, run the [az iot hub delete](#) command:

```
az iot hub delete --name my-sample-hub --resource-group my-sample-resource-group
```

To delete a resource group and all its resources, run the [az group delete](#) command:

```
az group delete --name my-sample-resource-group
```

## Next steps

In this quickstart, you've deployed an IoT hub and a Device Provisioning Service instance, and linked the two resources. To learn how to use this setup to provision a simulated device, continue to the quickstart for creating a simulated device.

[Quickstart to create a simulated device](#)

# Quickstart: Set up the IoT Hub Device Provisioning Service with an Azure Resource Manager template

12/10/2019 • 8 minutes to read • [Edit Online](#)

You can use [Azure Resource Manager](#) to programmatically set up the Azure cloud resources necessary for provisioning your devices. These steps show how to create an IoT hub and a new IoT Hub Device Provisioning Service, and link the two services together using an Azure Resource Manager template. This quickstart uses [Azure CLI](#) to perform the programmatic steps necessary to create a resource group and deploy the template, but you can easily use the [Azure portal](#), [PowerShell](#), .NET, Ruby, or other programming languages to perform these steps and deploy your template.

## Prerequisites

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- This quickstart requires that you run the Azure CLI locally. You must have the Azure CLI version 2.0 or later installed. Run `az --version` to find the version. If you need to install or upgrade the CLI, see [Install the Azure CLI](#).

## Sign in to Azure and create a resource group

Sign in to your Azure account and select your subscription.

1. At the command prompt, run the [login command](#):

```
az login
```

Follow the instructions to authenticate using the code and sign in to your Azure account through a web browser.

2. If you have multiple Azure subscriptions, signing in to Azure grants you access to all the Azure accounts associated with your credentials. Use the following [command to list the Azure accounts](#) available for you to use:

```
az account list
```

Use the following command to select subscription that you want to use to run the commands to create your IoT hub. You can use either the subscription name or ID from the output of the previous command:

```
az account set --subscription {your subscription name or id}
```

3. When you create Azure cloud resources like IoT hubs and provisioning services, you create them in a resource group. Either use an existing resource group, or run the following [command to create a resource group](#):

```
az group create --name {your resource group name} --location westus
```

**TIP**

The previous example creates the resource group in the West US location. You can view a list of available locations by running the command `az account list-locations -o table`.

## Create a Resource Manager template

Use a JSON template to create a provisioning service and a linked IoT hub in your resource group. You can also use an Azure Resource Manager template to make changes to an existing provisioning service or IoT hub.

1. Use a text editor to create an Azure Resource Manager template called **template.json** with the following skeleton content.

```
{  
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
  "contentVersion": "1.0.0.0",  
  "parameters": {},  
  "variables": {},  
  "resources": []  
}
```

2. Replace the **parameters** section with the following content. The parameters section defines parameters whose values can be passed in from another file. This section defines the name of the IoT hub and provisioning service to create. It also defines the location for both the IoT hub and provisioning service. The values will be restricted to Azure regions that support IoT hubs and provisioning services. For a list of supported locations for Device Provisioning Service, you can run the following command

```
az provider show --namespace Microsoft.Devices --query "resourceTypes[?  
resourceType=='ProvisioningServices'].locations | [0]" --out table
```

or go to the [Azure Status](#) page and search on "Device Provisioning Service".

```
"parameters": {  
  "iotHubName": {  
    "type": "string"  
  },  
  "provisioningServiceName": {  
    "type": "string"  
  },  
  "hubLocation": {  
    "type": "string",  
    "allowedValues": [  
      "eastus",  
      "westus",  
      "westeurope",  
      "northeurope",  
      "southeastasia",  
      "eastasia"  
    ]  
  }  
},
```

3. Replace the **variables** section with the following content. This section defines values that are used later to construct the IoT hub connection string, which is needed to link the provisioning service and the IoT hub.

```

"variables": {
    "iotHubResourceId": "[resourceId('Microsoft.Devices/Iothubs', parameters('iotHubName'))]",
    "iotHubKeyName": "iothubowner",
    "iotHubKeyResource": "[resourceId('Microsoft.Devices/Iothubs/Iothubkeys', parameters('iotHubName'), variables('iotHubKeyResource'))]"
},

```

- To create an IoT hub, add the following lines to the **resources** collection. The JSON specifies the minimum properties required to create an IoT hub. The **name** and **location** values will be passed as parameters from another file. To learn more about the properties you can specify for an IoT hub in a template, see [Microsoft.Devices/IotHubs template reference](#).

```

{
    "apiVersion": "2017-07-01",
    "type": "Microsoft.Devices/IotHubs",
    "name": "[parameters('iotHubName')]",
    "location": "[parameters('hubLocation')]",
    "sku": {
        "name": "S1",
        "capacity": 1
    },
    "tags": {},
    "properties": {}
},

```

- To create the provisioning service, add the following lines after the IoT hub specification in the **resources** collection. The **name** and **location** of the provisioning service will be passed in as parameters. The **iotHubs** collection specifies the IoT hubs to link to the provisioning service. At a minimum, you must specify the **connectionString** and **location** properties for each linked IoT hub. You can also set properties like **allocationWeight** and **applyAllocationPolicy** on each IoT hub, as well as properties like **allocationPolicy** and **authorizationPolicies** on the provisioning service itself. To learn more, see [Microsoft.Devices/provisioningServices template reference](#).

The **dependsOn** property is used to ensure that Resource Manager creates the IoT hub before it creates the provisioning service. The template requires the connection string of the IoT hub to specify its linkage to the provisioning service, so the hub and its keys must be created first. The template uses functions like **concat** and **listKeys** to create the connection string from parameterized variables. To learn more, see [Azure Resource Manager template functions](#).

```
{
  "type": "Microsoft.Devices/provisioningServices",
  "sku": {
    "name": "S1",
    "capacity": 1
  },
  "name": "[parameters('provisioningServiceName')]",
  "apiVersion": "2017-11-15",
  "location": "[parameters('hubLocation')]",
  "tags": {},
  "properties": {
    "iotHubs": [
      {
        "connectionString": "[concat('HostName=',
reference(variables('iotHubResourceId')).hostName, ';SharedAccessKeyName=' , variables('iotHubKeyName'),
';SharedAccessKey=' , listkeys(variables('iotHubKeyResource'), '2017-07-01').primaryKey)]",
        "location": "[parameters('hubLocation')]",
        "name": "[concat(parameters('iotHubName'), '.azure-devices.net')]"
      }
    ]
  },
  "dependsOn": "[["parameters('iotHubName')]]"
}
}
```

- Save the template file. The finished template should look like the following:

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "iotHubName": {
      "type": "string"
    },
    "provisioningServiceName": {
      "type": "string"
    },
    "hubLocation": {
      "type": "string",
      "allowedValues": [
        "eastus",
        "westus",
        "westeurope",
        "northeurope",
        "southeastasia",
        "eastasia"
      ]
    }
  },
  "variables": {
    "iotHubResourceId": "[resourceId('Microsoft.Devices/Iothubs', parameters('iotHubName'))]",
    "iotHubKeyOwner": "iothubowner",
    "iotHubKeyResource": "[resourceId('Microsoft.Devices/Iothubs/Iothubkeys',
parameters('iotHubName'), variables('iotHubKeyName'))]"
  },
  "resources": [
    {
      "apiVersion": "2017-07-01",
      "type": "Microsoft.Devices/IotHubs",
      "name": "[parameters('iotHubName')]",
      "location": "[parameters('hubLocation')]",
      "sku": {
        "name": "S1",
        "capacity": 1
      },
      "tags": {
        "name": "iotHub"
      }
    }
  ]
}
```

```

        },
        "properties": {
        }
    },
    {
        "type": "Microsoft.Devices/provisioningServices",
        "sku": {
            "name": "S1",
            "capacity": 1
        },
        "name": "[parameters('provisioningServiceName')]",
        "apiVersion": "2017-11-15",
        "location": "[parameters('hubLocation')]",
        "tags": {},
        "properties": {
            "iotHubs": [
                {
                    "connectionString": "[concat('HostName=', reference(variables('iotHubResourceId')).hostName, ';SharedAccessKeyName=', variables('iotHubKeyName'), ';SharedAccessKey=', listkeys(variables('iotHubKeyResource'), '2017-07-01').primaryKey)]",
                    "location": "[parameters('hubLocation')]",
                    "name": "[concat(parameters('iotHubName'), '.azure-devices.net')]"
                }
            ]
        },
        "dependsOn": "[parameters('iotHubName')]"
    }
]
}

```

## Create a Resource Manager parameter file

The template that you defined in the last step uses parameters to specify the name of the IoT hub, the name of the provisioning service, and the location (Azure region) to create them. You pass these parameters into the template from a separate file. Doing so enables you to reuse the same template for multiple deployments. To create the parameter file, follow these steps:

1. Use a text editor to create an Azure Resource Manager parameter file called **parameters.json** with the following skeleton content:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {}
}
```

2. Add the **iotHubName** value to the parameter section. An IoT hub name must be globally unique in Azure, so you may want to add a unique prefix or suffix to the example name, or choose a new name altogether. Make sure your name follows proper naming conventions for an IoT hub: it should be 3-50 characters in length, and can contain only upper or lower case alphanumeric characters or hyphens ('-').

```
"parameters": {
    "iotHubName": {
        "value": "my-sample-iot-hub"
    }
}
```

3. Add the **provisioningServiceName** value to the parameter section. You will also need to choose a globally

unique name for your provisioning service. Make sure it follows proper naming conventions for an IoT Hub Device Provisioning Service: it should be 3-64 characters in length and can contain only upper or lower case alphanumeric characters or hyphens ('-').

```
"parameters": {  
    "iotHubName": {  
        "value": "my-sample-iot-hub"  
    },  
    "provisioningServiceName": {  
        "value": "my-sample-provisioning-service"  
    },  
}
```

4. Add the **hubLocation** value to the parameter section. This value specifies the location for both the IoT hub and provisioning service. The value must correspond to one of the locations specified in the **allowedValues** collection in the parameter definition in the template file. This collection restricts the values to Azure locations that support both IoT hubs and provisioning services. For a list of supported locations for Device Provisioning Service, you can run the command

```
az provider show --namespace Microsoft.Devices --query "resourceTypes[?  
resourceType=='ProvisioningServices'].locations | [0]" --out table
```

, or go to the [Azure Status](#) page and search on "Device Provisioning Service".

```
"parameters": {  
    "iotHubName": {  
        "value": "my-sample-iot-hub"  
    },  
    "provisioningServiceName": {  
        "value": "my-sample-provisioning-service"  
    },  
    "hubLocation": {  
        "value": "westus"  
    }  
}
```

5. Save the file.

#### IMPORTANT

Both the IoT hub and the provisioning service will be publicly discoverable as DNS endpoints, so make sure to avoid any sensitive information when naming them.

## Deploy the template

Use the following Azure CLI commands to deploy your templates and verify the deployment.

1. To deploy your template, navigate to the folder containing the template and parameter files, and run the following [command to start a deployment](#):

```
az group deployment create -g {your resource group name} --template-file template.json --parameters  
@parameters.json
```

This operation may take a few minutes to complete. Once it's done, look for the **provisioningState** property showing "Succeeded" in the output.

```
],
  "mode": "Incremental",
  "outputs": null,
  "parameters": {
    "hubLocation": {
      "type": "String",
      "value": "westus"
    },
    "iotHubName": {
      "type": "String",
      "value": "my-sample-iot-hub"
    },
    "provisioningServiceName": {
      "type": "String",
      "value": "my-sample-provisioning-service"
    }
  },
  "parametersLink": null,
  "providers": [
    {
      "id": null,
      "namespace": "Microsoft.Devices",
      "registrationState": null,
      "resourceTypes": [
        {
          "aliases": null,
          "apiVersions": null,
          "locations": [
            "westus"
          ],
          "properties": null,
          "resourceType": "IoTHubs"
        },
        {
          "aliases": null,
          "apiVersions": null,
          "locations": [
            "westus"
          ],
          "properties": null,
          "resourceType": "provisioningServices"
        }
      ]
    }
  ],
  "provisioningState": "Succeeded",
  "template": null,
  "templateLink": null,
  "timestamp": "2018-02-26T22:27:31.318400+00:00"
},
"resourceGroup": "my-sample-resource-group"
}
```

- To verify your deployment, run the following [command to list resources](#) and look for the new provisioning service and IoT hub in the output:

```
az resource list -g {your resource group name}
```

## Clean up resources

Other quickstarts in this collection build upon this quickstart. If you plan to continue on to work with subsequent quickstarts or with the tutorials, do not clean up the resources created in this quickstart. If you do not plan to continue, you can use the Azure CLI to [delete an individual resource](#), such as an IoT hub or a provisioning service, or to delete a resource group and all of its resources.

To delete the provisioning service, run the following command:

```
az iot hub delete --name {your provisioning service name} --resource-group {your resource group name}
```

To delete an IoT hub, run the following command:

```
az iot hub delete --name {your iot hub name} --resource-group {your resource group name}
```

To delete a resource group and all its resources, run the following command:

```
az group delete --name {your resource group name}
```

You can also delete resource groups and individual resources using the Azure portal, PowerShell, or REST APIs, as well as with supported platform SDKs published for Azure Resource Manager or IoT Hub Device Provisioning Service.

## Next steps

In this quickstart, you've deployed an IoT hub and a Device Provisioning Service instance, and linked the two resources. To learn how to use this setup to provision a simulated device, continue to the quickstart for creating a simulated device.

[Quickstart to create a simulated device](#)

# Quickstart: Provision a simulated device with symmetric keys

1/14/2020 • 7 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create and run a device simulator on a Windows development machine. You will configure this simulated device to use a symmetric key to authenticate with a Device Provisioning Service instance and be assigned to an IoT hub. Sample code from the [Azure IoT C SDK](#) will be used to simulate a boot sequence for the device that initiates provisioning. The device will be recognized based on an individual enrollment with a provisioning service instance and assigned to an IoT hub.

Although this article demonstrates provisioning with an individual enrollment, you can use enrollment groups. There are some differences when using enrollment groups. For example, you must use a derived device key with a unique registration ID for the device. Although symmetric key enrollment groups are not limited to legacy devices, [How to provision legacy devices using Symmetric key attestation](#) provides an enrollment group example. For more information, see [Group Enrollments for Symmetric Key Attestation](#).

If you're unfamiliar with the process of auto-provisioning, review [Auto-provisioning concepts](#).

Also, make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing with this quickstart. This quickstart requires you to have already created your Device Provisioning Service instance.

This article is oriented toward a Windows-based workstation. However, you can perform the procedures on Linux. For a Linux example, see [How to provision for multitenancy](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

The following prerequisites are for a Windows development environment. For Linux or macOS, see the appropriate section in [Prepare your development environment](#) in the SDK documentation.

- [Visual Studio 2019](#) with the 'Desktop development with C++' workload enabled. Visual Studio 2015 and Visual Studio 2017 are also supported.
- Latest version of [Git](#) installed.

## Prepare an Azure IoT C SDK development environment

In this section, you will prepare a development environment used to build the [Azure IoT C SDK](#).

The SDK includes the sample code for a simulated device. This simulated device will attempt provisioning during the device's boot sequence.

### 1. Download the CMake build system.

It is important that the Visual Studio prerequisites (Visual Studio and the 'Desktop development with C++' workload) are installed on your machine, **before** starting the `cmake` installation. Once the prerequisites are in place, and the download is verified, install the CMake build system.

Older versions of the CMake build system fail to generate the solution file used in this article. Make sure to use a newer version of CMake.

2. Click **Tags** and find the tag name for the latest release on the [Release page of the Azure IoT C SDK](#).
3. Open a command prompt or Git Bash shell. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. Create a `cmake` subdirectory in the root directory of the git repository, and navigate to that folder. Run the following commands from the `azure-iot-sdk-c` directory:

```
mkdir cmake
cd cmake
```

5. Run the following command, which builds a version of the SDK specific to your development client platform. A Visual Studio solution for the simulated device will be generated in the `cmake` directory.

```
cmake -Dhsm_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
```

If `cmake` does not find your C++ compiler, you might get build errors while running the above command. If that happens, try running this command in the [Visual Studio command prompt](#).

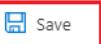
Once the build succeeds, the last few output lines will look similar to the following output:

```
$ cmake -Dhsm_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
-- Building for: Visual Studio 15 2017
-- Selecting Windows SDK version 10.0.16299.0 to target Windows 10.0.17134.
-- The C compiler identification is MSVC 19.12.25835.0
-- The CXX compiler identification is MSVC 19.12.25835.0

...
-- Configuring done
-- Generating done
-- Build files have been written to: E:/IoT Testing/azure-iot-sdk-c/cmake
```

## Create a device enrollment entry in the portal

1. Sign in to the [Azure portal](#), select the **All resources** button on the left-hand menu and open your Device Provisioning service.
2. Select the **Manage enrollments** tab, and then select the **Add individual enrollment** button at the top.
3. In the **Add Enrollment** panel, enter the following information, and press the **Save** button.
  - **Mechanism:** Select **Symmetric Key** as the identity attestation *Mechanism*.
  - **Auto-generate keys:** Check this box.
  - **Registration ID:** Enter a registration ID to identify the enrollment. Use only lowercase alphanumeric and dash ('-') characters. For example, **symm-key-device-007**.
  - **IoT Hub Device ID:** Enter a device identifier. For example, **device-007**.

 Add Enrollment

Mechanism \* ⓘ  
Symmetric Key

Auto-generate keys ⓘ

Primary Key ⓘ  
Enter your primary key

Secondary Key ⓘ  
Enter your secondary key

Registration ID \*  
symm-key-device-007

IoT Hub Device ID ⓘ  
device-007

IoT Edge device ⓘ  
 True  False

Select how you want to assign devices to hubs ⓘ  
Evenly weighted distribution

Select the IoT hubs this device can be assigned to: ⓘ  
test-docs-dps.azure-devices.net

[Link a new IoT hub](#)

Select how you want device data to be handled on re-provisioning \* ⓘ  
Re-provision and migrate data

Device Twin is only supported for standard tier IoT hubs. [Learn more about standard tier.](#)

- Once you have saved your enrollment, the **Primary Key** and **Secondary Key** will be generated and added to the enrollment entry. Your symmetric key device enrollment appears as **symm-key-device-007** under the *Registration ID* column in the *Individual Enrollments* tab.

Open the enrollment and copy the value of your generated **Primary Key**.

## Simulate first boot sequence for the device

In this section, update the sample code to send the device's boot sequence to your Device Provisioning Service instance. This boot sequence will cause the device to be recognized and assigned to an IoT hub linked to the Device Provisioning Service instance.

- In the Azure portal, select the **Overview** tab for your Device Provisioning service and note the *ID Scope* value.

The screenshot shows the Azure IoT Hub Device Provisioning Service Overview page. The left sidebar lists various settings like Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, Manage enrollments, and Manage allocation policy. The main pane displays resource details: Resource group (test-rg-dps), Service endpoint (test-docs-dps.azure-devices-provisioning.net), Status (Active), Location (East US), Global device endpoint (global.azure-devices-provisioning.net), ID Scope (0ne0000A0A), Subscription (Microsoft Azure Internal Consumption), Subscription ID (\*\*\*\*), Pricing and scale tier (S1). A red box highlights the 'ID Scope' field. Below the main pane, there's a 'Quick Links' section with links to documentation and service concepts.

2. In Visual Studio, open the `azure_iot_sdks.sln` solution file that was generated by running CMake. The solution file should be in the following location:

```
\azure-iot-sdk-c\cmake\azure_iot_sdks.sln
```

If the file was not generated in your `cmake` directory, make sure you used a recent version of the CMake build system.

3. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Samples** folder. Expand the sample project named **prov\_dev\_client\_sample**. Expand **Source Files**, and open **prov\_dev\_client\_sample.c**.
4. Find the `id_scope` constant, and replace the value with your **ID Scope** value that you copied earlier.

```
static const char* id_scope = "0ne00002193";
```

5. Find the definition for the `main()` function in the same file. Make sure the `hsm_type` variable is set to `SECURE_DEVICE_TYPE_SYMMETRIC_KEY` as shown below:

```
SECURE_DEVICE_TYPE hsm_type;
//hsm_type = SECURE_DEVICE_TYPE_TPM;
//hsm_type = SECURE_DEVICE_TYPE_X509;
hsm_type = SECURE_DEVICE_TYPE_SYMMETRIC_KEY;
```

6. Find the call to `prov_dev_set_symmetric_key_info()` in `prov_dev_client_sample.c` which is commented out.

```
// Set the symmetric key if using they auth type
//prov_dev_set_symmetric_key_info("<symm_registration_id>", "<symmetric_Key>");
```

Uncomment the function call, and replace the placeholder values (including the angle brackets) with your registration ID and primary key values.

```
// Set the symmetric key if using they auth type  
prov_dev_set_symmetric_key_info("symm-key-device-007", "your primary key here");
```

Save the file.

7. Right-click the `prov_dev_client_sample` project and select **Set as Startup Project**.
8. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the prompt to rebuild the project, select **Yes**, to rebuild the project before running.

The following output is an example of the simulated device successfully booting up, and connecting to the provisioning Service instance to be assigned to an IoT hub:

```
Provisioning API Version: 1.2.8  
  
Registering Device  
  
Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED  
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING  
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING  
  
Registration Information received from service:  
test-docs-hub.azure-devices.net, deviceId: device-007  
Press enter key to exit:
```

9. In the portal, navigate to the IoT hub your simulated device was assigned to and select the **IoT devices** tab. On successful provisioning of the simulated to the hub, its device ID appears on the **IoT Devices** blade, with *STATUS* as **enabled**. You might need to press the **Refresh** button at the top.

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION...	CLOUD TO DEVICE ...
device-007	Enabled	Mon Sep 17 201...	Sas	0	0

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this

quickstart.

1. Close the device client sample output window on your machine.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open **Manage Enrollments** for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open **IoT devices** for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a simulated device on your Windows machine and provisioned it to your IoT hub using Symmetric key with the Azure IoT Hub Device Provisioning Service on the portal. To learn how to enroll your device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.

[Azure quickstart - Enroll X.509 devices to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Provision a simulated device to IoT Hub with symmetric keys

7/30/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create and run a device simulator on a Windows development machine. You will configure this simulated device to use a symmetric key to authenticate with a Device Provisioning Service (DPS) instance and be assigned to an IoT hub. Sample code from the [Microsoft Azure IoT SDKs for Java](#) will be used to simulate a boot sequence for the device that initiates provisioning. The device will be recognized based on an individual enrollment with a DPS service instance and assigned to an IoT hub.

Although this article demonstrates provisioning with an individual enrollment, you can use enrollment groups. There are some differences when using enrollment groups. For example, you must use a derived device key with a unique registration ID for the device. Although symmetric key enrollment groups are not limited to legacy devices, [How to provision legacy devices using Symmetric key attestation](#) provides an enrollment group example. For more information, see [Group Enrollments for Symmetric Key Attestation](#).

If you're unfamiliar with the process of auto-provisioning, review [Auto-provisioning concepts](#).

Also, make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing with this quickstart. This quickstart requires you to have already created your Device Provisioning Service instance.

This article is oriented toward a Windows-based workstation. However, you can perform the procedures on Linux. For a Linux example, see [How to provision for multitenancy](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Make sure you have [Java SE Development Kit 8](#) or later installed on your machine.
- Download and install [Maven](#).
- Latest version of [Git](#) installed.

## Prepare the Java SDK environment

1. Make sure Git is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the [Git Bash](#), the command-line app that you can use to interact with your local Git repository.
2. Open a command prompt. Clone the GitHub repo for device simulation code sample:

```
git clone https://github.com/Azure/azure-iot-sdk-java.git --recursive
```

3. Navigate to the root `azure-iot-sdk-java` directory and build the project to download all needed packages.

```
cd azure-iot-sdk-java  
mvn install -DskipTests=true
```

## Create a device enrollment

1. Sign in to the [Azure portal](#), select the **All resources** button on the left-hand menu and open your Device Provisioning service (DPS) instance.
2. Select the **Manage enrollments** tab, and then select the **Add individual enrollment** button at the top.
3. In the **Add Enrollment** panel, enter the following information, and press the **Save** button.
  - **Mechanism:** Select **Symmetric Key** as the identity attestation *Mechanism*.
  - **Auto-generate keys:** Check this box.
  - **Registration ID:** Enter a registration ID to identify the enrollment. Use only lowercase alphanumeric and dash ('-') characters. For example, **symm-key-java-device-007**.
  - **IoT Hub Device ID:** Enter a device identifier. For example, **java-device-007**.

The screenshot shows the 'Add Enrollment' page in the Microsoft Azure portal. The 'Save' button is highlighted with a red box. The 'Mechanism' dropdown is set to 'Symmetric Key' and is also highlighted with a red box. The 'Auto-generate keys' checkbox is checked and is highlighted with a red box. The 'Registration ID' field contains 'symm-key-java-device-007' and the 'IoT Hub Device ID' field contains 'java-device-007', both of which are highlighted with red boxes.

4. Once you have saved your enrollment, the **Primary Key** and **Secondary Key** will be generated and added to the enrollment entry. Your symmetric key device enrollment appears as **symm-key-java-device-007** under the *Registration ID* column in the *Individual Enrollments* tab.

Open the enrollment and copy the value of your generated **Primary Key**. You will use this key value and the **Registration ID** later when you update the Java code for the device.

## Simulate device boot sequence

In this section, you will update the device sample code to send the device's boot sequence to your DPS instance. This boot sequence will cause the device to be recognized, authenticated, and assigned to an IoT hub linked to the DPS instance.

1. From the Device Provisioning Service menu, select **Overview** and note your *ID Scope* and *Provisioning Service Global Endpoint*.

2. Open the Java device sample code for editing. The full path to the device sample code is:

```
azure-iot-sdk-java/provisioning/provisioning-samples/provisioning-symmetrickey-sample/src/main/java/samples/com/microsoft/azure/sdk/iot/ProvisioningSymmetricKeySample.java
```

- Add the *ID Scope* and *Provisioning Service Global Endpoint* of your DPS instance. Also include the primary symmetric key and the registration ID you chose for your individual enrollment. Save your changes.

```
private static final String SCOPE_ID = "[Your scope ID here]";
private static final String GLOBAL_ENDPOINT = "[Your Provisioning Service Global Endpoint here]";
private static final String SYMMETRIC_KEY = "[Enter your Symmetric Key here]";
private static final String REGISTRATION_ID = "[Enter your Registration ID here];
```

3. Open a command prompt for building. Navigate to the provisioning sample project folder of the Java SDK repository.

```
cd azure-iot-sdk-java/provisioning/provisioning-samples/provisioning-symmetrickey-sample
```

4. Build the sample then navigate to the `target` folder to execute the created jar file.

```
mvn clean install
cd target
java -jar ./provisioning-symmetrickey-sample-{version}-with-deps.jar
```

5. The expected output should look similar to the following:

```
Starting...
Beginning setup.
Waiting for Provisioning Service to register
IoTHub Uri : <Your DPS Service Name>.azure-devices.net
Device ID : java-device-007
Sending message from device to IoT Hub...
Press any key to exit...
Message received! Response status: OK_EMPTY
```

6. In the Azure portal, navigate to the IoT hub linked to your provisioning service and open the **Device Explorer** blade. After successful provisioning the simulated symmetric key device to the hub, its device ID

appears on the Device Explorer blade, with *STATUS* as enabled. You might need to press the Refresh button at the top if you already opened the blade prior to running the sample device application.

The screenshot shows the 'test-docs-dps - IoT devices' blade in the Azure portal. On the left, there's a navigation menu with sections like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings (Shared access policies, Pricing and scale, IP Filter, Certificates, Built-in endpoints, Failover, Properties, Locks, Export template), Explorers (Query explorer, IoT devices), and a search bar. The 'IoT devices' item in the Explorers section is highlighted with a red box. In the main pane, there's a query builder with fields for Field (select or enter a property name), Operator (=), and Value (specify constraint). Below it is a 'Query devices' button. A table lists a single device: java-device-007, which is Enabled, has a last status update of '--', and uses Sas authentication. The 'DEVICE ID' column is also highlighted with a red box.

#### NOTE

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open **Manage Enrollments** for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open **IoT devices** for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a simulated device on your Windows machine and provisioned it to your IoT hub using Symmetric key with the Azure IoT Hub Device Provisioning Service on the portal. To learn how to enroll your device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.



# Quickstart: Provision a Python device with symmetric keys

7/30/2020 • 6 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to provision a Windows development machine as a device to an IoT hub using Python. This device will use a symmetric key to authenticate with a Device Provisioning Service (DPS) instance in order to be assigned to an IoT hub. The authenticated device will be recognized by DPS based on an individual enrollment and assigned to an IoT hub. Sample code from the [Azure IoT Python SDK](#) will be used to provision the device.

Although this article demonstrates provisioning with an individual enrollment, you can also use enrollment groups. There are some differences when using enrollment groups. For example, you must use a derived device key with a unique registration ID for the device. Although symmetric key enrollment groups are not limited to legacy devices, [How to provision legacy devices using Symmetric key attestation](#) provides an enrollment group example. For more information, see [Group Enrollments for Symmetric Key Attestation](#).

If you're unfamiliar with the process of auto-provisioning, review [Auto-provisioning concepts](#).

Also, make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing with this quickstart. This quickstart requires you to have already created your Device Provisioning Service instance.

This article is oriented toward a Windows-based workstation. However, you can perform the procedures on Linux. For a Linux example, see [How to provision for multitenancy](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Make sure you have [Python 3.7](#) or later installed on your Windows-based machine. You can check your version of Python by running `python --version`.
- Latest version of [Git](#) installed.

## Prepare the Python SDK environment

1. Make sure Git is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the [Git Bash](#), the command-line app that you can use to interact with your local Git repository.
2. Open a command prompt. Clone the GitHub repo for the Azure IoT Python SDK:

```
git clone https://github.com/Azure/azure-iot-sdk-python.git --recursive
```

3. Navigate to the `azure-iot-sdk-python\azure-iot-device\samples\async-hub-scenarios` directory where the sample file, `provision_symmetric_key.py`, is located.

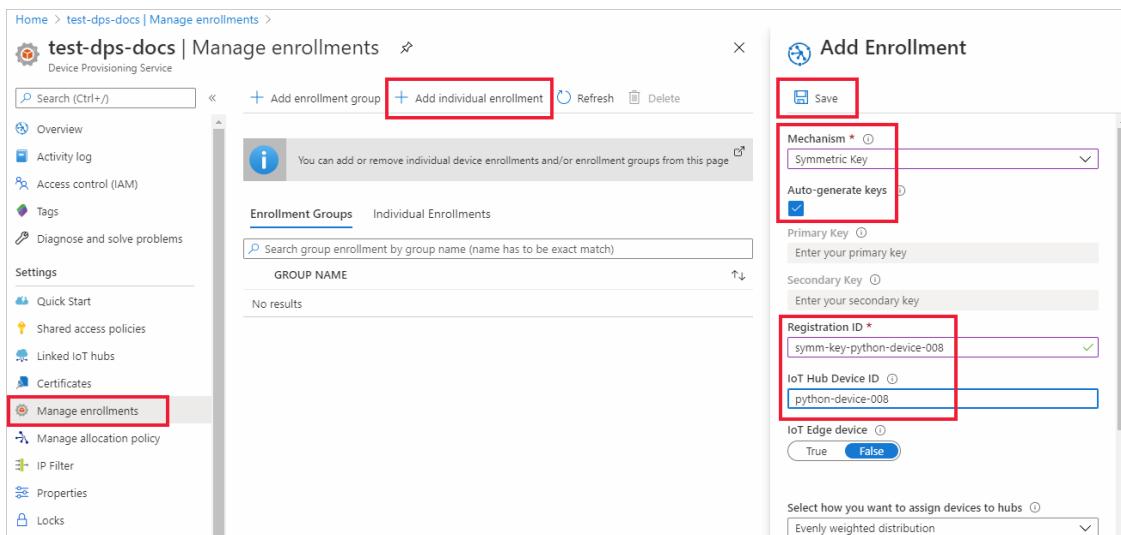
```
cd azure-iot-sdk-python\azure-iot-device\samples\async-hub-scenarios
```

4. Install the `azure-iot-device` library by running the following command.

```
pip install azure-iot-device
```

## Create a device enrollment

1. Sign in to the [Azure portal](#), select the **All resources** button on the left-hand menu and open your Device Provisioning service (DPS) instance.
2. Select the **Manage enrollments** tab, and then select the **Add individual enrollment** button at the top.
3. In the **Add Enrollment** panel, enter the following information, and press the **Save** button.
  - **Mechanism:** Select **Symmetric Key** as the identity attestation *Mechanism*.
  - **Auto-generate keys:** Check this box.
  - **Registration ID:** Enter a registration ID to identify the enrollment. Use only lowercase alphanumeric and dash ('-') characters. For example, **symm-key-python-device-008**.
  - **IoT Hub Device ID:** Enter a device identifier. For example, **python-device-008**.



4. Once you have saved your enrollment, the **Primary Key** and **Secondary Key** will be generated and added to the enrollment entry. Your symmetric key device enrollment appears as **symm-key-python-device-008** under the *Registration ID* column in the *Individual Enrollments* tab.
5. Open the enrollment and copy the value of your generated **Primary Key**. You will use this key value and the **Registration ID** later when you add environment variables for use with the device provisioning sample code.

## Prepare the device provisioning code

In this section, you will add the following four environment variables that will be used as parameters for the device provisioning sample code for your symmetric key device.

- `PROVISIONING_HOST`
- `PROVISIONING_IDSCOPE`
- `PROVISIONING_REGISTRATION_ID`
- `PROVISIONING_SYMMETRIC_KEY`

The provisioning code will contact the DPS instance based on these variables in order to authenticate your device.

The device will then be assigned to an IoT hub already linked to the DPS instance based on the individual enrollment configuration. Once provisioned, the sample code will send some test telemetry to the IoT hub.

1. In the [Azure portal](#), on your Device Provisioning Service menu, select **Overview** and copy your *Service Endpoint* and *ID Scope*. You will use these values for the `PROVISIONING_HOST` and `PROVISIONING_IDSCOPE` environment variables.

The screenshot shows the Azure portal interface for a 'Device Provisioning Service'. The left sidebar has a 'Overview' tab selected, indicated by a red box. The main content area displays various service details: Resource group (DocRelatedTestingResources), Status (Active), Location (East US), Subscription (Your Subscription Name), and Subscription ID (Your Subscription ID). On the right, specific settings are highlighted with red boxes: 'Service endpoint' (test-dps-docs.azure-devices-provisioning.net) and 'ID Scope' (One00000A0A).

2. In your Python command prompt, add the environment variables using the values you copied.

The following commands are examples to show command syntax. Make sure to use your correct values.

```
set PROVISIONING_HOST=test-dps-docs.azure-devices-provisioning.net
```

```
set PROVISIONING_IDSCOPE=One00000A0A
```

3. In your Python command prompt, add the environment variables for the registration ID and symmetric key you copied from the individual enrollment in the previous section.

The following commands are examples to show command syntax. Make sure to use your correct values.

```
set PROVISIONING_REGISTRATION_ID=symm-key-python-device-008
```

```
set  
PROVISIONING_SYMMETRIC_KEY=sbDDeEzRuEuGKag+kQKV+T1QGakRtHpsERLP0yPjwR93TrpEgEh/Y07CXstfha6dhIPWvdD1nRxK5  
T0KGKA+nQ==
```

4. Run the python sample code in *provision\_symmetric\_key.py*.

```
D:\azure-iot-sdk-python\azure-iot-device\samples\async-hub-scenarios>python provision_symmetric_key.py
```

5. The expected output should look similar to the following which shows the linked IoT hub that the device was assigned to based on the individual enrollment settings. Some example wind speed telemetry messages are also sent to the hub as a test:

```

D:\azure-iot-sdk-python\azure-iot-device\samples\async-hub-scenarios>python provision_symmetric_key.py
RegistrationStage(RequestAndResponseOperation): Op will transition into polling after interval 2.
Setting timer.
The complete registration result is
python-device-008
docs-test-iot-hub.azure-devices.net
initialAssignment
null
Will send telemetry from the provisioned device
sending message #8
sending message #9
sending message #3
sending message #10
sending message #4
sending message #2
sending message #6
sending message #7
sending message #1
sending message #5
done sending message #8
done sending message #9
done sending message #3
done sending message #10
done sending message #4
done sending message #2
done sending message #6
done sending message #7
done sending message #1
done sending message #5

```

6. In the Azure portal, navigate to the IoT hub linked to your provisioning service and open the **IoT devices** blade. After successfully provisioning the symmetric key device to the hub, the device ID is shown with **STATUS** as **enabled**. You might need to press the **Refresh** button at the top if you already opened the blade prior to running the device sample code.

The screenshot shows the 'IoT devices' blade in the Azure portal for the 'docs-test-iot-hub' IoT hub. The blade has a header with a search bar, a 'New' button, a 'Refresh' button (which is highlighted with a red box), and a 'Delete' button. Below the header is a section for querying devices. The main area displays a table of devices. One device, 'python-device-008', is highlighted with a red box. The table columns include DEVICE ID, STATUS, LAST ST..., AUTHE..., and CLOUD ... . The 'python-device-008' row shows 'Enabled' under STATUS, '--' under LAST ST..., 'Sas' under AUTHE..., and '0' under CLOUD ... . On the left side, there's a sidebar with sections like Pricing and scale, Networking, Certificates, Built-in endpoints, Failover, Properties, Locks, and Export template. The 'Explorers' section has 'Query explorer' and 'IoT devices' selected, with 'IoT devices' also highlighted with a red box. At the bottom, there's an 'Automatic Device Management' section and an 'IoT Edge' section.

Field	Operator	Value
select or enter a property name	=	specify constraint value

DEVICE ID	STATUS	LAST ST...	AUTHE...	CLOUD ...
python-device-008	Enabled	--	Sas	0

#### **NOTE**

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open **Manage Enrollments** for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open **IoT devices** for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you provisioned a Windows-based symmetric key device to your IoT hub using the IoT Hub Device Provisioning Service. To learn how to provision X.509 certificate devices using Python, continue with the quickstart below for X.509 devices.

[Azure quickstart - Provision X.509 devices using DPS and Python](#)

# Quickstart: Provision an X.509 simulated device using the Azure IoT C SDK

4/26/2020 • 7 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create and run an X.509 device simulator on a Windows development machine. You will configure this simulated device to be assigned to an IoT hub using an enrollment with a Device Provisioning Service instance. Sample code from the [Azure IoT C SDK](#) will be used to simulate a boot sequence for the device. The device will be recognized based on the enrollment with the provisioning service and assigned to the IoT hub.

If you're unfamiliar with the process of autopropvisioning, review [Auto-provisioning concepts](#). Also, make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing with this quickstart.

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual Enrollments](#): Used to enroll a single device.

This article will demonstrate individual enrollments.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

The following prerequisites are for a Windows development environment. For Linux or macOS, see the appropriate section in [Prepare your development environment](#) in the SDK documentation.

- [Visual Studio 2019](#) with the '[Desktop development with C++](#)' workload enabled. Visual Studio 2015 and Visual Studio 2017 are also supported.
- Latest version of [Git](#) installed.

## Prepare a development environment for the Azure IoT C SDK

In this section, you will prepare a development environment used to build the [Azure IoT C SDK](#), which includeS the sample code for the X.509 boot sequence.

1. Download the [CMake build system](#).

It is important that the Visual Studio prerequisites (Visual Studio and the 'Desktop development with C++' workload) are installed on your machine, **before** starting the `cmake` installation. Once the prerequisites are in place, and the download is verified, install the CMake build system.

2. Find the tag name for the [latest release](#) of the SDK.
3. Open a command prompt or Git Bash shell. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. Create a `cmake` subdirectory in the root directory of the git repository, and navigate to that folder. Run the following commands from the `azure-iot-sdk-c` directory:

```
mkdir cmake
cd cmake
```

5. The code sample uses an X.509 certificate to provide attestation via X.509 authentication. Run the following command to build a version of the SDK specific to your development platform that includes the device provisioning client. A Visual Studio solution for the simulated device is generated in the `cmake` directory.

```
cmake -Duse_prov_client:BOOL=ON ..
```

If `cmake` does not find your C++ compiler, you might get build errors while running the above command. If that happens, try running this command in the [Visual Studio command prompt](#).

Once the build succeeds, the last few output lines look similar to the following output:

```
$ cmake -Duse_prov_client:BOOL=ON ..
-- Building for: Visual Studio 16 2019
-- The C compiler identification is MSVC 19.23.28107.0
-- The CXX compiler identification is MSVC 19.23.28107.0

...
-- Configuring done
-- Generating done
-- Build files have been written to: C:/code/azure-iot-sdk-c/cmake
```

## Create a self-signed X.509 device certificate

In this section you will use a self-signed X.509 certificate, it is important to keep in mind the following points:

- Self-signed certificates are for testing only, and should not be used in production.
- The default expiration date for a self-signed certificate is one year.

You will use sample code from the Azure IoT C SDK to create the certificate to be used with the individual enrollment entry for the simulated device.

1. Launch Visual Studio and open the new solution file named `azure_iot_sdks.sln`. This solution file is located in the `cmake` folder you previously created in the root of the `azure-iot-sdk-c` git repository.
2. On the Visual Studio menu, select **Build > Build Solution** to build all projects in the solution.
3. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Tools** folder. Right-click the **device\_device\_enrollment** project and select **Set as Startup Project**.
4. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the output window, enter **i** for individual enrollment when prompted.

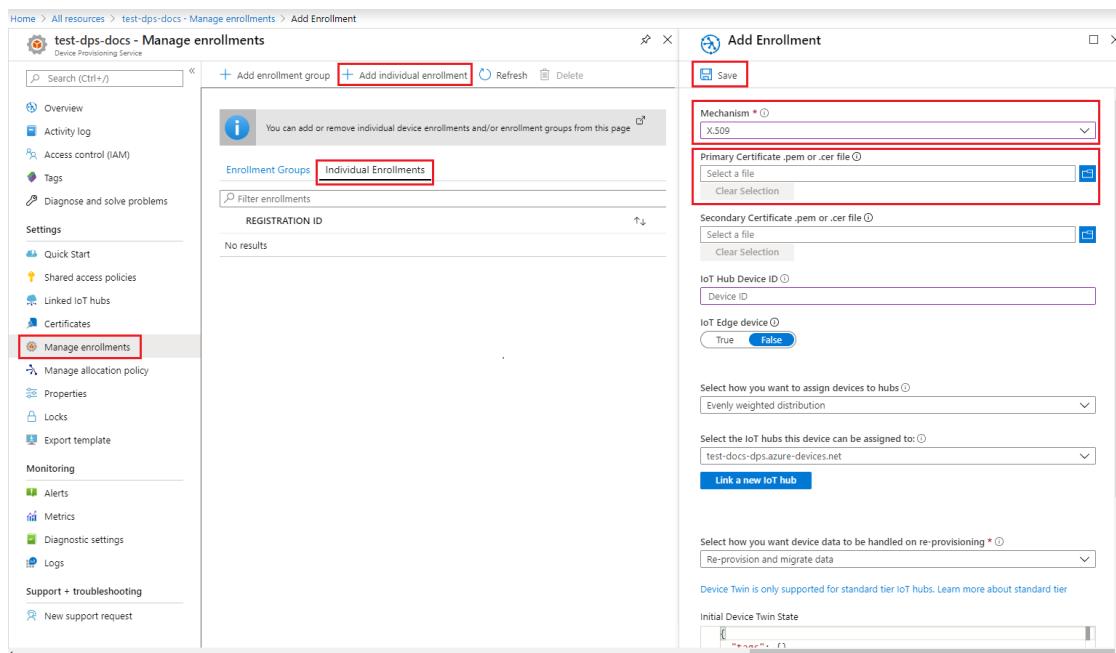
The output window displays a locally generated self-signed X.509 certificate for your simulated device.

Copy the output to clipboard, starting from -----BEGIN CERTIFICATE----- and ending with the first -----END CERTIFICATE-----, making sure to include both of these lines as well. You need only the first certificate from the output window.

5. Using a text editor, save the certificate to a new file named **X509testcert.pem**.

## Create a device enrollment entry in the portal

1. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service.
2. Select the **Manage enrollments** tab, then select the **Add individual enrollment** button at the top.
3. In the Add Enrollment panel, enter the following information, then press the **Save** button.
  - **Mechanism:** Select X.509 as the identity attestation *Mechanism*.
  - **Primary certificate .pem or .cer file:** Choose **Select a file** to select the certificate file, X509testcert.pem, you created earlier.
  - **IoT Hub Device ID:** Enter **test-docs-cert-device** to give the device an ID.



On successful enrollment, your X.509 device appears as **riot-device-cert** under the *Registration ID* column in the *Individual Enrollments* tab.

## Simulate first boot sequence for the device

In this section, update the sample code to send the device's boot sequence to your Device Provisioning Service instance. This boot sequence will cause the device to be recognized and assigned to an IoT hub linked to the Device Provisioning Service instance.

1. In the Azure portal, select the **Overview** tab for your Device Provisioning service and note the *ID Scope* value.

2. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Samples** folder. Expand the sample project named **prov\_dev\_client\_sample**. Expand **Source Files**, and open **prov\_dev\_client\_sample.c**.
3. Find the `id_scope` constant, and replace the value with your **ID Scope** value that you copied earlier.

```
static const char* id_scope = "0ne00002193";
```

4. Find the definition for the `main()` function in the same file. Make sure the `hsm_type` variable is set to `SECURE_DEVICE_TYPE_X509` instead of `SECURE_DEVICE_TYPE_TPM` as shown below.

```
SECURE_DEVICE_TYPE hsm_type;
//hsm_type = SECURE_DEVICE_TYPE_TPM;
hsm_type = SECURE_DEVICE_TYPE_X509;
```

5. Right-click the **prov\_dev\_client\_sample** project and select **Set as Startup Project**.
6. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the prompt to rebuild the project, select **Yes** to rebuild the project before running.

The following output is an example of the provisioning device client sample successfully booting up, and connecting to the provisioning Service instance to get IoT hub information and registering:

```
Provisioning API Version: 1.2.7

Registering... Press enter key to interrupt.

Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING

Registration Information received from service:
test-docs-hub.azure-devices.net, deviceId: test-docs-cert-device
```

7. In the portal, navigate to the IoT hub linked to your provisioning service and select the **IoT devices** tab. On

successful provisioning of the simulated X.509 device to the hub, its device ID appears on the IoT devices blade, with *STATUS* as **enabled**. You might need to press the **Refresh** button at the top.

DEVICE ID	STATUS	LAST STATUS UPDATE (UTC)	AUTHENTICATION TYPE	CLOUD TO DEVICE MESSAGE COUNT
test-docs-cert-device	Enabled	--	SelfSigned	0

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open **Manage Enrollments** for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open **IoT devices** for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a simulated X.509 device on your Windows machine and provisioned it to your IoT hub using the Azure IoT Hub Device Provisioning Service on the portal. To learn how to enroll your X.509 device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.

[Azure quickstart - Enroll X.509 devices to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision a simulated X.509 device using Java device SDK for IoT Hub Device Provisioning Service

7/30/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you create a simulated X.509 device on a Windows computer. You use device sample Java code to connect this simulated device with your IoT hub using an individual enrollment with the Device Provisioning Service (DPS).

## Prerequisites

- Review of [Auto-provisioning concepts](#).
- Completion of [Set up IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Java SE Development Kit 8](#).
- [Maven](#).
- [Git](#).

## Prepare the environment

1. Make sure you have [Java SE Development Kit 8](#) installed on your machine.
2. Download and install [Maven](#).
3. Make sure Git is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the [Git Bash](#), the command-line app that you can use to interact with your local Git repository.
4. Open a command prompt. Clone the GitHub repo for device simulation code sample:

```
git clone https://github.com/Azure/azure-iot-sdk-java.git --recursive
```

5. Navigate to the root `azure-iot-sdk-java` directory and build the project to download all needed packages.

```
cd azure-iot-sdk-java  
mvn install -DskipTests=true
```

6. Navigate to the certificate generator project and build the project.

```
cd azure-iot-sdk-java/provisioning/provisioning-tools/provisioning-x509-cert-generator  
mvn clean install
```

## Create a self-signed X.509 device certificate and individual enrollment entry

In this section you, will use a self-signed X.509 certificate, it is important to keep in mind the following:

- Self-signed certificates are for testing only, and should not be used in production.
- The default expiration date for a self-signed certificate is one year.

You will use sample code from the [Azure IoT SDK for Java](#) to create the certificate to be used with the individual enrollment entry for the simulated device.

The Azure IoT Device Provisioning Service supports two types of enrollments:

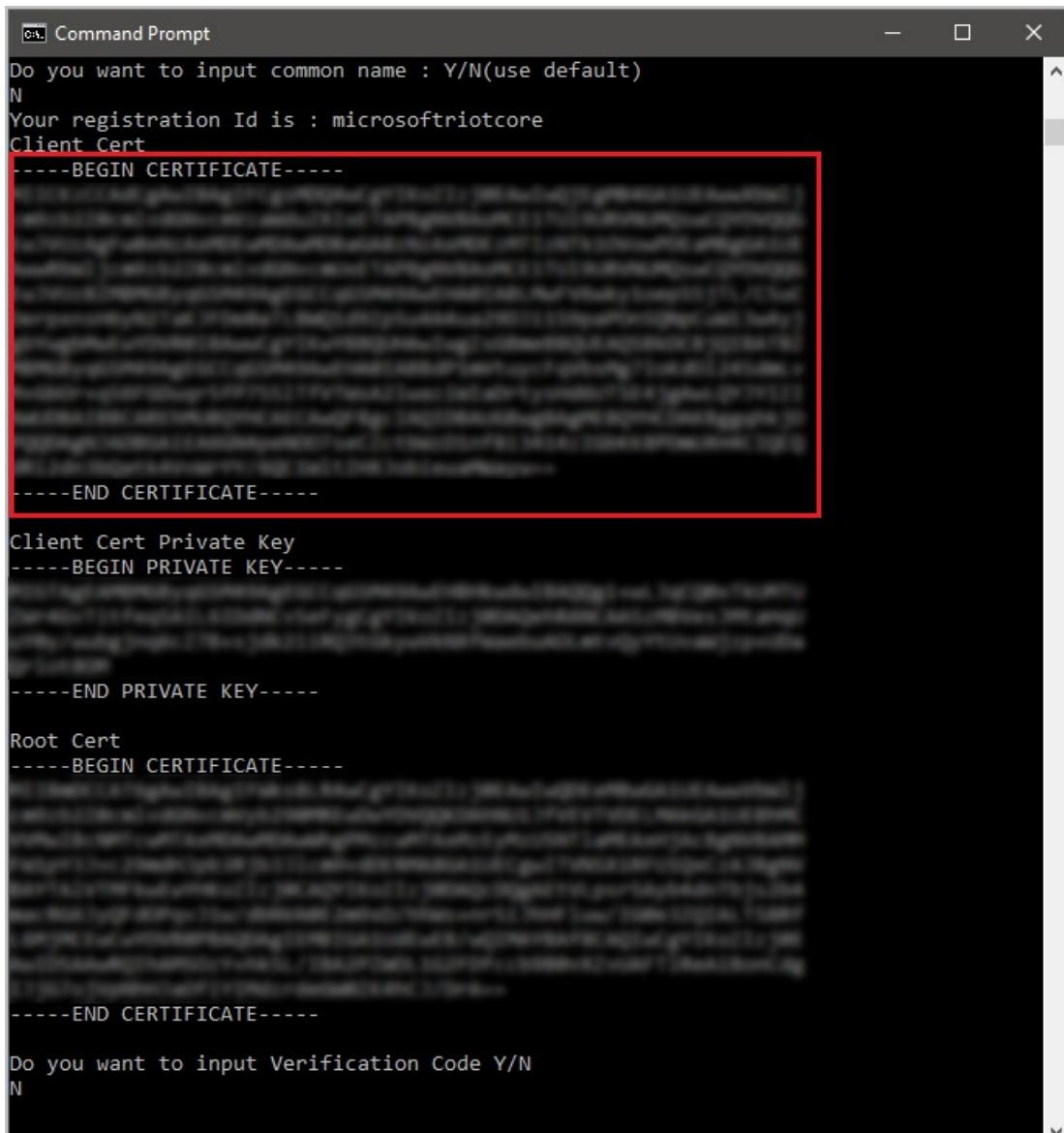
- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual enrollments](#): Used to enroll a single device.

This article demonstrates individual enrollments.

1. Using the command prompt from previous steps, navigate to the `target` folder, then execute the `.jar` file created in the previous step.

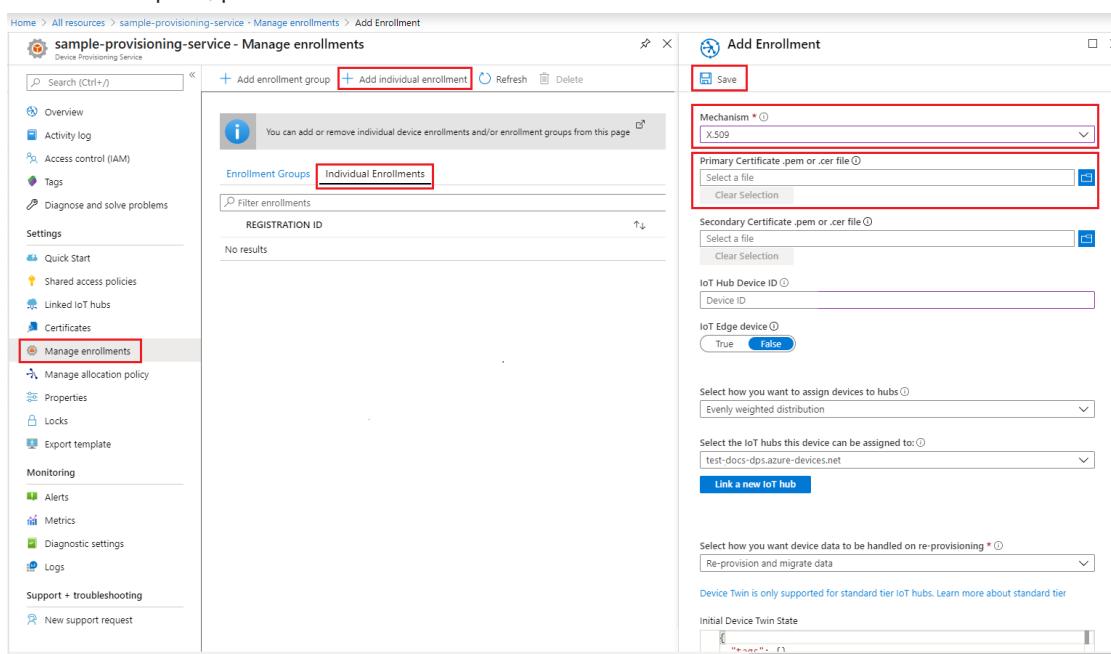
```
cd target  
java -jar ./provisioning-x509-cert-generator-{version}-with-deps.jar
```

2. Enter **N** for *Do you want to input common name*. Copy to the clipboard the output of `Client Cert`, starting from `-----BEGIN CERTIFICATE-----` through `-----END CERTIFICATE-----`.



```
OS Command Prompt  
Do you want to input common name : Y/N(use default)  
N  
Your registration Id is : microsoftiotcore  
Client Cert  
-----BEGIN CERTIFICATE-----  
[Redacted certificate content]  
-----END CERTIFICATE-----  
Client Cert Private Key  
-----BEGIN PRIVATE KEY-----  
[Redacted private key content]  
-----END PRIVATE KEY-----  
Root Cert  
-----BEGIN CERTIFICATE-----  
[Redacted root certificate content]  
-----END CERTIFICATE-----  
Do you want to input Verification Code Y/N  
N
```

3. Create a file named **X509individual.pem** on your Windows machine, open it in an editor of your choice, and copy the clipboard contents to this file. Save the file and close your editor.
4. In the command prompt, enter **N** for *Do you want to input Verification Code* and keep the program output open for reference later in the quickstart. Later you copy the **Client Cert** and **Client Cert Private Key** values, for use in the next section.
5. Sign in to the [Azure portal](#), select the **All resources** button on the left-hand menu and open your Device Provisioning Service instance.
6. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.
7. In the **Add Enrollment** panel, enter the following information:
  - Select **X.509** as the identity attestation *Mechanism*.
  - Under the *Primary certificate .pem or .cer file*, choose *Select a file* to select the certificate file **X509individual.pem** created in the previous steps.
  - Optionally, you may provide the following information:
    - Select an IoT hub linked with your provisioning service.
    - Enter a unique device ID. Make sure to avoid sensitive data while naming your device.
    - Update the **Initial device twin state** with the desired initial configuration for the device.
    - Once complete, press the **Save** button.



Upon successful enrollment, your X.509 device appears as **microsoftiotcore** under the *Registration ID* column in the *Individual Enrollments* tab.

## Simulate the device

1. From the Device Provisioning Service menu, select **Overview** and note your *ID Scope* and *Provisioning Service Global Endpoint*.

2. Open a command prompt. Navigate to the sample project folder of the Java SDK repository.

```
cd azure-iot-sdk-java/provisioning/provisioning-samples/provisioning-X509-sample
```

3. Enter the provisioning service and X.509 identity information in your code. This is used during autoprovisioning, for attestation of the simulated device, prior to device registration:

- Edit the file `/src/main/java/samples/com/microsoft/azure/sdk/iot/ProvisioningX509Sample.java`, to include your *ID Scope* and *Provisioning Service Global Endpoint* as noted previously. Also include *Client Cert* and *Client Cert Private Key* as noted in the previous section.

```
private static final String idScope = "[Your ID scope here]";
private static final String globalEndpoint = "[Your Provisioning Service Global Endpoint here]";
private static final ProvisioningDeviceClientTransportProtocol
PROVISIONING_DEVICE_CLIENT_TRANSPORT_PROTOCOL = ProvisioningDeviceClientTransportProtocol.HTTPS;
private static final String leafPublicPem = "<Your Public PEM Certificate here>";
private static final String leafPrivateKey = "<Your Private PEM Key here>";
```

- Use the following format when copying/pasting your certificate and private key:

```
private static final String leafPublicPem = "-----BEGIN CERTIFICATE-----\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"+XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"-----END CERTIFICATE-----\n";
private static final String leafPrivateKey = "-----BEGIN PRIVATE KEY-----\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXX\n" +
"-----END PRIVATE KEY-----\n";
```

4. Build the sample. Navigate to the `target` folder and execute the created .jar file.

```
mvn clean install
cd target
java -jar ./provisioning-x509-sample-{version}-with-deps.jar
```

5. In the Azure portal, navigate to the IoT hub linked to your provisioning service and open the Device

**Explorer** blade. Upon successful provisioning of the simulated X.509 device to the hub, its device ID appears on the **Device Explorer** blade, with *STATUS* as **enabled**. You might need to press the **Refresh** button at the top if you already opened the blade prior to running the sample device application.

The screenshot shows the 'test-docs-dps - IoT devices' blade in the Azure portal. On the left, a navigation menu includes 'SETTINGS' (Shared access policies, Pricing and scale, Operations monitoring, IP Filter, Certificates, Properties, Locks, Automation script), 'EXPLORERS' (Query explorer, IoT devices - highlighted with a red box), and 'AUTOMATIC DEVICE MANAGEMENT' (IoT Edge, IoT device configuration). The main area has a search bar and buttons for '+ Add', 'Refresh', and 'Delete'. A message box says: 'You can use this tool to view, create, update, and delete devices on your IoT Hub.' Below it is a query editor with a placeholder 'optional (e.g. tags.location='US')' and an 'Execute' button. A table lists devices with columns: DEVICE ID, STATUS, LAST ACTIVITY, LAST STATUS UPD..., AUTHENTICATION..., CLOUD TO DEVICE ... . A single row is shown for 'Java-device' with 'Enabled' status, 'Sas' authentication, and 0 cloud-to-device messages. The 'DEVICE ID' column for 'Java-device' is also highlighted with a red box.

#### NOTE

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open the **IoT devices** blade for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you created a simulated X.509 device on your Windows machine. You configured its enrollment in your Azure IoT Hub Device Provisioning Service, then autoprovisioned the device to your IoT hub. To learn how to enroll your X.509 device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.



# Quickstart: Create and provision a simulated X.509 device using C# device SDK for IoT Hub Device Provisioning Service

12/10/2019 • 4 minutes to read • [Edit Online](#)

These steps show you how to use the [Azure IoT Samples for C#](#) to simulate an X.509 device on a development machine running the Windows OS. The sample also connects the simulated device to an IoT Hub using the Device Provisioning Service.

If you're unfamiliar with the process of autopropvisioning, be sure to also review [Auto-provisioning concepts](#). Also make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing.

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual Enrollments](#): Used to enroll a single device.

This article will demonstrate individual enrollments.

## NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the development environment

1. Make sure you have the [.NET Core 2.1 SDK or later](#) installed on your machine.
2. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the **Git Bash**, the command-line app that you can use to interact with your local Git repository.
3. Open a command prompt or Git Bash. Clone the Azure IoT Samples for C# GitHub repo:

```
git clone https://github.com/Azure-Samples/azure-iot-samples-csharp.git
```

## Create a self-signed X.509 device certificate and individual enrollment entry

In this section you, will use a self-signed X.509 certificate, it is important to keep in mind the following:

- Self-signed certificates are for testing only, and should not be used in production.
- The default expiration date for a self-signed certificate is one year.

You will use sample code from the [Provisioning Device Client Sample - X.509 Attestation](#) to create the certificate to be used with the individual enrollment entry for the simulated device.

1. In a command prompt, change directories to the project directory for the X.509 device provisioning sample.

```
cd .\azure-iot-samples-csharp\provisioning\Samples\device\X509Sample
```

2. The sample code is set up to use X.509 certificates stored within a password-protected PKCS12 formatted file (certificate.pfx). Additionally, you need a public key certificate file (certificate.cer) to create an individual enrollment later in this quickstart. To generate a self-signed certificate and its associated .cer and .pfx files, run the following command:

```
powershell .\GenerateTestCertificate.ps1
```

3. The script prompts you for a PFX password. Remember this password, you must use it when you run the sample.

The screenshot shows a Windows Command Prompt window titled "Command Prompt". The command entered is "powershell .\GenerateTestCertificate.ps1". The output of the script is displayed, including certificate details like Subject (CN=iothubx509device1, O=TEST, C=US), Issuer (CN=iothubx509device1, O=TEST, C=US), Serial Number, Not Before (1/4/2018 4:38:50 PM), Not After (1/4/2019 4:58:50 PM), and Thumbprint. It also prompts for the PFX password with four asterisks. Finally, it lists the generated certificate file "certificate.pfx" in the current directory.

```
C:\Users\[REDACTED]\azure-iot-sdk-csharp\provisioning\device\samples\ProvisioningDeviceClientX509>powershell .\GenerateTestCertificate.ps1
Generated the certificate:
[Subject]
  CN=iothubx509device1, O=TEST, C=US

[Issuer]
  CN=iothubx509device1, O=TEST, C=US

[Serial Number]
[REDACTED]

[Not Before]
  1/4/2018 4:38:50 PM

[Not After]
  1/4/2019 4:58:50 PM

[Thumbprint]
[REDACTED]

Enter the PFX password:
****

      Directory: C:\Users\[REDACTED]\azure-iot-sdk-csharp\provisioning\device\samples\ProvisioningDeviceClientX509

Mode           LastWriteTime         Length Name
----           -----          ------
-a---  1/4/2018  4:48 PM        2726 certificate.pfx

C:\Users\[REDACTED]\azure-iot-sdk-csharp\provisioning\device\samples\ProvisioningDeviceClientX509>
```

4. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your provisioning service.
5. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.
6. In the **Add Enrollment** panel, enter the following information:
  - Select **X.509** as the identity attestation *Mechanism*.
  - Under the *Primary certificate .pem or .cer file*, choose *Select a file* to select the certificate file **certificate.cer** created in the previous steps.
  - Leave **Device ID** blank. Your device will be provisioned with its device ID set to the common name (CN) in the X.509 certificate, **iothubx509device1**. This will also be the name used for the registration ID for the individual enrollment entry.
  - Optionally, you may provide the following information:
    - Select an IoT hub linked with your provisioning service.
    - Update the **Initial device twin state** with the desired initial configuration for the device.
  - Once complete, press the **Save** button.

On successful enrollment, your X.509 enrollment entry appears as **iothubx509device1** under the *Registration ID* column in the *Individual Enrollments* tab.

## Provision the simulated device

- From the **Overview** blade for your provisioning service, note the *ID Scope* value.

- Type the following command to build and run the X.509 device provisioning sample. Replace the `<IDScope>` value with the ID Scope for your provisioning service.

```
dotnet run <IDScope>
```

- When prompted, enter the password for the PFX file that you created previously. Notice the messages that simulate the device booting and connecting to the Device Provisioning Service to get your IoT hub

information.

```
C:\Users\...\azure-iot-sdk-csharp\provisioning\device\samples\ProvisioningDeviceClientX509>dotnet run One00004D38
Enter the PFX password for certificate.pfx:
*****
RegistrationID = iothubx509device1
ProvisioningClient RegisterAsync . . . Assigned
ProvisioningClient AssignedHub: sample-iot-hub1.azure-devices.net; DeviceID: iothubx509device1
DeviceClient OpenAsync.
DeviceClient SendEventAsync.
DeviceClient CloseAsync.

C:\Users\...\azure-iot-sdk-csharp\provisioning\device\samples\ProvisioningDeviceClientX509>
```

4. Verify that the device has been provisioned. On successful provisioning of the simulated device to the IoT hub linked with your provisioning service, the device ID appears on the hub's **IoT devices** blade.

The screenshot shows the Azure portal interface for an IoT hub named "sample-iot-hub1". The left sidebar contains a navigation menu with various options such as Overview, Activity log, Access control (IAM), Tags, Events, Settings, Explorers (Query explorer, IoT devices), and Automatic Device Management. The "IoT devices" option under Explorers is highlighted with a red box. The main content area is titled "sample-iot-hub1 - IoT devices". It features a search bar, an "Add" button, a "Refresh" button, and a "Delete" button. A message box says, "You can use this tool to view, create, update, and delete devices on your IoT Hub." Below this is a "Query" section with a sample query: "SELECT \* FROM devices WHERE optional (e.g. tags.location='US')". An "Execute" button is present. A table below lists the device information:

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION ...	CLOUD TO DEVICE ...
iothubtomdevice1	Enabled		Sas	0	

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#)

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. Close the TPM simulator window on your machine.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. At the top of the **Overview** blade, press **Delete** at the top of the pane.
4. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. At the top of the **Overview** blade, press **Delete** at the top of the pane.

## Next steps

In this quickstart, you've created a simulated X.509 device on your Windows machine and provisioned it to your

IoT hub using the Azure IoT Hub Device Provisioning Service on the portal. To learn how to enroll your X.509 device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.

[Azure quickstart - Enroll X.509 devices to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision an X.509 simulated device using Node.js device SDK for IoT Hub Device Provisioning Service

7/30/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you create a simulated X.509 device on a Windows computer. You use device sample Node.js code to connect this simulated device with your IoT hub using an individual enrollment with the Device Provisioning Service (DPS).

## Prerequisites

- Review of [Auto-provisioning concepts](#).
- Completion of [Set up IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Node.js v4.0+](#).
- [Git](#).
- [OpenSSL](#).

### NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the environment

1. Complete the steps in the [Setup IoT Hub Device Provisioning Service with the Azure portal](#) before you proceed.
2. Make sure you have [Node.js v4.0 or above](#) installed on your machine.
3. Make sure [Git](#) is installed on your machine and is added to the environment variables accessible to the command window.
4. Make sure [OpenSSL](#) is installed on your machine and is added to the environment variables accessible to the command window. This library can either be built and installed from source or downloaded and installed from a [third party](#) such as [this](#).

### NOTE

If you have already created your *root*, *intermediate*, and/or *leaf* X.509 certificates, you may skip this step and all following steps regarding certificate generation.

## Create a self-signed X.509 device certificate and individual enrollment entry

In this section you, will use a self-signed X.509 certificate, it is important to keep in mind the following:

- Self-signed certificates are for testing only, and should not be used in production.
- The default expiration date for a self-signed certificate is one year.

You will use sample code from the [Azure IoT SDK for Node.js](#) to create the certificate to be used with the individual enrollment entry for the simulated device.

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual enrollments](#): Used to enroll a single device.

This article demonstrates individual enrollments.

1. Open a command prompt. Clone the GitHub repo for the code samples:

```
git clone https://github.com/Azure/azure-iot-sdk-node.git --recursive
```

2. Navigate to the certificate generator script and build the project.

```
cd azure-iot-sdk-node/provisioning/tools  
npm install
```

3. Create a *leafX.509* certificate by running the script using your own *certificate-name*. The leaf certificate's common name becomes the [Registration ID](#) so be sure to only use lower-case alphanumeric and hyphens.

```
node create_test_cert.js device {certificate-name}
```

4. Sign in to the [Azure portal](#), select the **All resources** button on the left-hand menu and open your Device Provisioning Service instance.

5. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.

6. In the **Add Enrollment** panel, enter the following information:

- Select **X.509** as the identity attestation *Mechanism*.
- Under the *Primary certificate .pem or .cer file*, choose *Select a file* to select the certificate file **{certificate-name}\_cert.pem** created in the previous steps.
- Optionally, you may provide the following information:
  - Select an IoT hub linked with your provisioning service.
  - Enter a unique device ID. Make sure to avoid sensitive data while naming your device.
  - Update the **Initial device twin state** with the desired initial configuration for the device.
  - Once complete, press the **Save** button.

On successful enrollment, your X.509 device appears as `{certificatename}` under the *Registration ID* column in the *Individual Enrollments* tab. Note this value for later.

## Simulate the device

The [Azure IoT Hub Node.js Device SDK](#) provides an easy way to simulate a device. For further reading, see [Device concepts](#).

1. In the Azure portal, select the **Overview** blade for your Device Provisioning service and note the *Global Device Endpoint* and *ID Scope* values.

2. Copy your *certificate* and *key* to the sample folder.

```
copy .\{certificate-name}_cert.pem ..\device\samples\{certificate-name}_cert.pem
copy .\{certificate-name}_key.pem ..\device\samples\{certificate-name}_key.pem
```

3. Navigate to the device test script and build the project.

```
cd ..\device\samples  
npm install
```

4. Edit the `register_x509.js` file. Save the file after making the following changes.

- Replace `provisioning host` with the *Global Device Endpoint* noted in Step 1 above.
- Replace `id scope` with the *ID Scope* noted in Step 1 above.
- Replace `registration id` with the *Registration ID* noted in the previous section.
- Replace `cert filename` and `key filename` with the files you copied in Step 2 above.

5. Execute the script and verify the device was provisioned successfully.

```
node register_x509.js
```

6. In the portal, navigate to the IoT hub linked to your provisioning service and open the **IoT devices** blade.

On successful provisioning of the simulated X.509 device to the hub, its device ID appears on the **IoT devices** blade, with *STATUS* as **enabled**. You might need to press the **Refresh** button at the top if you already opened the blade prior to running the sample device application.

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION...	CLOUD TO DEVICE ...
test-docs-device	Enabled	Mon Sep 17 201...	Sas	0	

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.

2. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open the **IoT devices** blade for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a simulated X.509 device and provisioned it to your IoT hub using the Azure IoT Hub Device Provisioning Service on the portal. To learn how to enroll your X.509 device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.

[Azure quickstart - Enroll X.509 devices to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision a simulated X.509 device using Python device SDK for IoT Hub Device Provisioning Service

7/30/2020 • 6 minutes to read • [Edit Online](#)

In this quickstart, you create a simulated X.509 device on a Windows computer. You use device sample Python code to connect this simulated device with your IoT hub using an individual enrollment with the Device Provisioning Service (DPS).

## Prerequisites

- Review of [Auto-provisioning concepts](#).
- Completion of [Set up IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Visual Studio 2015+](#) with Desktop development with C++.
- [CMake build system](#).
- [Git](#).

### IMPORTANT

This article only applies to the deprecated V1 Python SDK. Device and service clients for the IoT Hub Device Provisioning Service are not yet available in V2. The team is currently hard at work to bring V2 to feature parity.

### NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the environment

1. Make sure you have installed either [Visual Studio](#) 2015 or later, with the 'Desktop development with C++' workload enabled for your Visual Studio installation.
2. Download and install the [CMake build system](#).
3. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the **Git Bash**, the command-line app that you can use to interact with your local Git repository.
4. Open a command prompt or Git Bash. Clone the GitHub repo for device simulation code sample.

```
git clone https://github.com/Azure/azure-iot-sdk-python.git --recursive
```

5. Create a folder in your local copy of this GitHub repo for CMake build process.

```
cd azure-iot-sdk-python/c  
mkdir cmake  
cd cmake
```

6. Run the following command to create the Visual Studio solution for the provisioning client.

```
cmake -Duse_prov_client:BOOL=ON ..
```

## Create a self-signed X.509 device certificate and individual enrollment entry

In this section you will use a self-signed X.509 certificate. It is important to keep in mind the following points:

- Self-signed certificates are for testing only, and should not be used in production.
- The default expiration date for a self-signed certificate is one year.

You will use sample code from the Azure IoT C SDK to create the certificate to be used with the individual enrollment entry for the simulated device.

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual enrollments](#): Used to enroll a single device.

This article demonstrates individual enrollments.

1. Open the solution generated in the `cmake` folder named `azure_iot_sdks.sln`, and build it in Visual Studio.
2. Right-click the `dice_device_enrollment` project under the `Provision_Tools` folder, and select **Set as Startup Project**. Run the solution.
3. In the output window, enter `i` for individual enrollment when prompted. The output window displays a locally generated X.509 certificate for your simulated device.

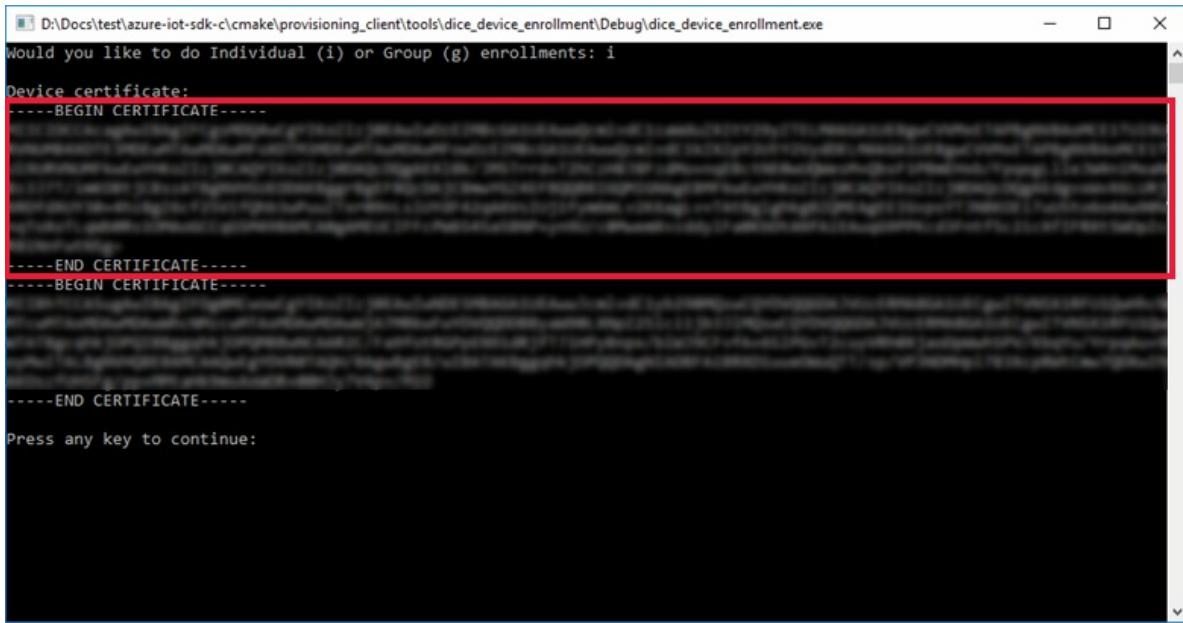
```
Copy the first certificate to clipboard. Begin with the first occurrence of:
```

```
-----BEGIN CERTIFICATE-----
```

```
End you copying after the first occurrence of:
```

```
-----END CERTIFICATE-----
```

```
Make sure to include both of those lines as well.
```



```
D:\Docs\test\azure-iot-sdk-c\cmake\provisioning_client\tools\dice_device_enrollment\Debug\dice_device_enrollment.exe
Would you like to do Individual (i) or Group (g) enrollments: i
Device certificate:
-----BEGIN CERTIFICATE-----
[REDACTED]
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
[REDACTED]
-----END CERTIFICATE-----
Press any key to continue:
```

4. Create a file named **X509testcertificate.pem** on your Windows machine, open it in an editor of your choice, and copy the clipboard contents to this file. Save the file.
5. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your provisioning service.
6. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.
7. In the **Add Enrollment** panel, enter the following information:
  - Select **X.509** as the identity attestation *Mechanism*.
  - Under the *Primary certificate .pem or .cer file*, choose *Select a file* to select the certificate file **X509testcertificate.pem** created in the previous steps.
  - Optionally, you may provide the following information:
    - Select an IoT hub linked with your provisioning service.
    - Enter a unique device ID. Make sure to avoid sensitive data while naming your device.
    - Update the **Initial device twin state** with the desired initial configuration for the device.
  - Once complete, press the **Save** button.

Upon successful enrollment, your X.509 device appears as **riot-device-cert** under the *Registration ID* column in the *Individual Enrollments* tab.

## Simulate the device

- From the Device Provisioning Service menu, select **Overview**. Note your *ID Scope* and *Global Service Endpoint*.

- Download and install [Python 2.x or 3.x](#). Make sure to use the 32-bit or 64-bit installation as required by your setup. When prompted during the installation, make sure to add Python to your platform-specific environment variables. If you are using Python 2.x, you may need to [install or upgrade pip](#), the Python package management system.

### NOTE

If you are using Windows, also install the [Visual C++ Redistributable for Visual Studio 2015](#). The pip packages require the redistributable in order to load/execute the C DLLs.

- Follow [these instructions](#) to build the Python packages.

**NOTE**

If using `pip` make sure to also install the `azure-iot-provisioning-device-client` package.

4. Navigate to the samples folder.

```
cd azure-iot-sdk-python/provisioning_device_client/samples
```

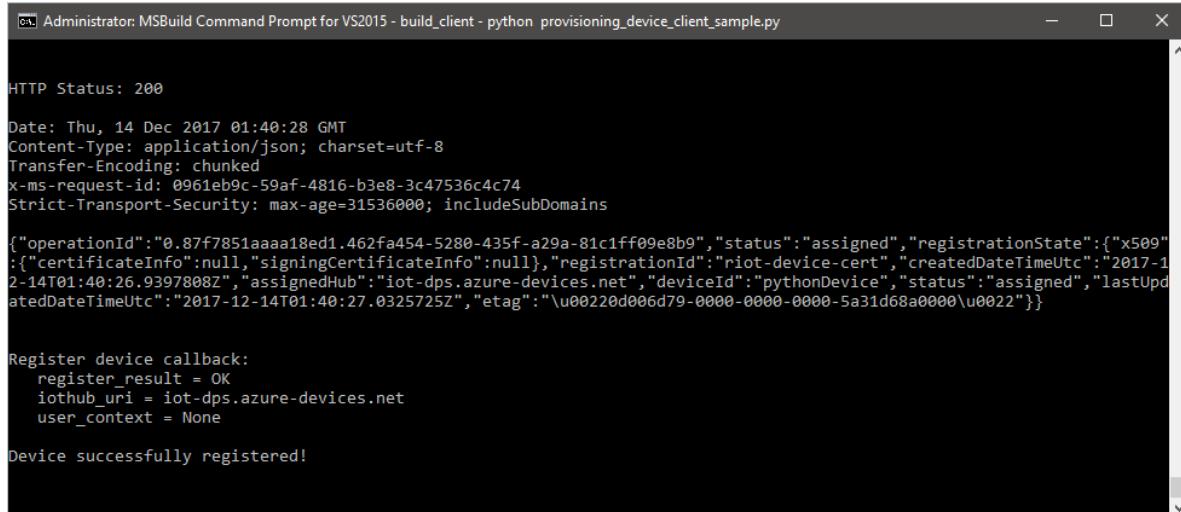
5. Using your Python IDE, edit the python script named `provisioning_device_client_sample.py`. Modify the `GLOBAL_PROV_URI` and `ID_SCOPE` variables to the values noted previously.

```
GLOBAL_PROV_URI = "{globalServiceEndpoint}"
ID_SCOPE = "{idScope}"
SECURITY_DEVICE_TYPE = ProvisioningSecurityDeviceType.X509
PROTOCOL = ProvisioningTransportProvider.HTTP
```

6. Run the sample.

```
python provisioning_device_client_sample.py
```

7. The application will connect, enroll the device, and display a successful enrollment message.



```
Administrator: MSBuild Command Prompt for VS2015 - build_client - python provisioning_device_client_sample.py

HTTP Status: 200
Date: Thu, 14 Dec 2017 01:40:28 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
x-ms-request-id: 0961eb9c-59af-4816-b3e8-3c47536c4c74
Strict-Transport-Security: max-age=31536000; includeSubDomains
{"operationId": "0.87f7851aaaa18ed1.462fa454-5280-435f-a29a-81c1ff09e8b9", "status": "assigned", "registrationState": {"x509": {"certificateInfo": null, "signingCertificateInfo": null}}, "registrationId": "riot-device-cert", "createdDateTimeUtc": "2017-12-14T01:40:26.9397808Z", "assignedHub": "iot-dps.azure-devices.net", "deviceId": "pythonDevice", "status": "assigned", "lastUpdatedDateTimeUtc": "2017-12-14T01:40:27.0325725Z", "etag": "\u0022d006d79-0000-0000-5a31d68a0000\u0022"}}

Register device callback:
register_result = OK
iothub_uri = iot-dps.azure-devices.net
user_context = None

Device successfully registered!
```

8. In the portal, navigate to the IoT hub linked to your provisioning service and open the **Device Explorer** blade. On successful provisioning of the simulated X.509 device to the hub, its device ID appears on the **Device Explorer** blade, with **STATUS** as **enabled**. You might need to press the **Refresh** button at the top if you already opened the blade prior to running the sample device application.

The screenshot shows the Azure IoT Hub Device Provisioning Service portal. On the left, there's a navigation menu with sections like SETTINGS, EXPLORERS, and AUTOMATIC DEVICE MANAGEMENT. Under EXPLORERS, 'IoT devices' is selected and highlighted with a red box. In the main pane, there's a search bar at the top with options to Add, Refresh, or Delete. Below that is a message: 'You can use this tool to view, create, update, and delete devices on your IoT Hub.' A query editor window is open, showing a sample SQL query: 'SELECT \* FROM devices WHERE optional (e.g. tags.location='US')'. There's a blue 'Execute' button below the query. At the bottom, a table lists a single device: 'python-device' (Device ID), 'Enabled' (Status), 'Mon Sep 17 201...' (Last Activity), 'Sas' (Authentication), and '0' (Cloud to Device). The 'python-device' row is also highlighted with a red box.

#### NOTE

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open the **IoT devices** blade for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a simulated X.509 device on your Windows machine and provisioned it to your IoT hub using the Azure IoT Hub Device Provisioning Service on the portal. To learn how to enroll your X.509 device programmatically, continue to the quickstart for programmatic enrollment of X.509 devices.

[Azure quickstart - Enroll X.509 devices to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Provision a simulated TPM device using the Azure IoT C SDK

4/20/2020 • 7 minutes to read • [Edit Online](#)

In this quickstart, you will learn how to create and run a Trusted Platform Module (TPM) device simulator on a Windows development machine. You will connect this simulated device to an IoT hub using a Device Provisioning Service instance. Sample code from the [Azure IoT C SDK](#) will be used to help enroll the device with a Device Provisioning Service instance and simulate a boot sequence for the device.

If you're unfamiliar with the process of autoprovisioning, review [Auto-provisioning concepts](#). Also, make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing with this quickstart.

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual Enrollments](#): Used to enroll a single device.

This article will demonstrate individual enrollments.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

The following prerequisites are for a Windows development environment. For Linux or macOS, see the appropriate section in [Prepare your development environment](#) in the SDK documentation.

- [Visual Studio](#) 2019 with the '[Desktop development with C++](#)' workload enabled. Visual Studio 2015 and Visual Studio 2017 are also supported.
- Latest version of [Git](#) installed.

## Prepare a development environment for the Azure IoT C SDK

In this section, you will prepare a development environment used to build the [Azure IoT C SDK](#) and the [TPM](#) device simulator sample.

### 1. Download the [CMake build system](#).

It is important that the Visual Studio prerequisites (Visual Studio and the '[Desktop development with C++](#)' workload) are installed on your machine, **before** starting the `CMake` installation. Once the prerequisites are in place, and the download is verified, install the CMake build system.

### 2. Find the tag name for the [latest release](#) of the SDK.

### 3. Open a command prompt or Git Bash shell. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. Create a `cmake` subdirectory in the root directory of the git repository, and navigate to that folder. Run the following commands from the `azure-iot-sdk-c` directory:

```
mkdir cmake  
cd cmake
```

## Build the SDK and run the TPM device simulator

In this section, you will build the Azure IoT C SDK, which includes the TPM device simulator sample code. This sample provides a TPM [attestation mechanism](#) via Shared Access Signature (SAS) Token authentication.

1. From the `cmake` subdirectory you created in the `azure-iot-sdk-c` git repository, run the following command to build the sample. A Visual Studio solution for the simulated device will also be generated by this build command.

```
cmake -Duse_prov_client:BOOL=ON -Duse_tpm_simulator:BOOL=ON ..
```

If `cmake` does not find your C++ compiler, you might get build errors while running the above command. If that happens, try running this command in the [Visual Studio command prompt](#).

Once the build succeeds, the last few output lines will look similar to the following output:

```
$ cmake -Duse_prov_client:BOOL=ON -Duse_tpm_simulator:BOOL=ON ..  
-- Building for: Visual Studio 15 2017  
-- Selecting Windows SDK version 10.0.16299.0 to target Windows 10.0.17134.  
-- The C compiler identification is MSVC 19.12.25835.0  
-- The CXX compiler identification is MSVC 19.12.25835.0  
  
...  
  
-- Configuring done  
-- Generating done  
-- Build files have been written to: E:/IoT Testing/azure-iot-sdk-c/cmake
```

2. Navigate to the root folder of the git repository you cloned, and run the [TPM](#) simulator using the path shown below. This simulator listens over a socket on ports 2321 and 2322. Do not close this command window; you will need to keep this simulator running until the end of this quickstart.

If you are in the `cmake` folder, then run the following commands:

```
cd ..  
.\\provisioning_client\\deps\\utpm\\tools\\tpm_simulator\\Simulator.exe
```

You will not see any output from the simulator. Let it continue to run simulating a TPM device.

## Read cryptographic keys from the TPM device

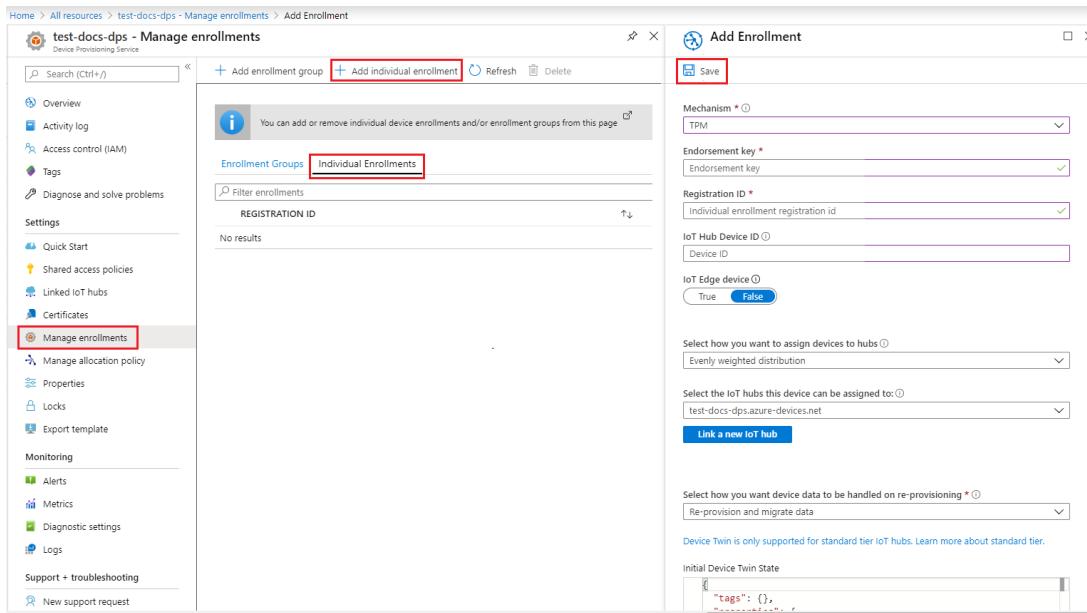
In this section, you will build and execute a sample that will read the endorsement key and registration ID from the TPM simulator you left running, and listening over ports 2321 and 2322. These values will be used for device enrollment with your Device Provisioning Service instance.

1. Launch Visual Studio and open the new solution file named `azure_iot_sdks.sln`. This solution file is located in the `cmake` folder you previously created in the root of the `azure-iot-sdk-c` git repository.

2. On the Visual Studio menu, select **Build > Build Solution** to build all projects in the solution.
3. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Tools** folder. Right-click the **tpm\_device\_provision** project and select **Set as Startup Project**.
4. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. The app reads and displays a *Registration ID* and an *Endorsement key*. Note or copy these values. They will be used in the next section for device enrollment.

## Create a device enrollment entry in the portal

1. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service.
2. Select the **Manage enrollments** tab, and then select the **Add individual enrollment** button at the top.
3. In the **Add Enrollment** panel, enter the following information:
  - Select **TPM** as the identity attestation *Mechanism*.
  - Enter the *Registration ID* and *Endorsement key* for your TPM device from the values you noted previously.
  - Select an IoT hub linked with your provisioning service.
  - Optionally, you may provide the following information:
    - Enter a unique *Device ID* (you can use the suggested **test-docs-device** or provide your own). Make sure to avoid sensitive data while naming your device. If you choose not to provide one, the registration ID will be used to identify the device instead.
    - Update the **Initial device twin state** with the desired initial configuration for the device.
  - Once complete, press the **Save** button.



On successful enrollment, the *Registration ID* of your device will appear in the list under the *Individual Enrollments* tab.

## Simulate first boot sequence for the device

In this section, you will configure sample code to use the [Advanced Message Queuing Protocol \(AMQP\)](#) to send the device's boot sequence to your Device Provisioning Service instance. This boot sequence will cause the

device to be recognized and assigned to an IoT hub linked to the Device Provisioning Service instance.

1. In the Azure portal, select the **Overview** tab for your Device Provisioning service and copy the **ID Scope** value.

The screenshot shows the Azure portal interface for a Device Provisioning Service named "test-docs-dps". The "Overview" tab is selected, indicated by a red box around it. In the main content area, the "ID Scope" field is also highlighted with a red box and contains the value "One0000A0A". Other visible information includes the resource group "test-rg-dps", status "Active", location "East US", subscription "Microsoft Azure Internal Consumption", and pricing tier "S1". On the left, there's a sidebar with links for Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, and Manage enrollments.

2. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Samples** folder. Expand the sample project named **prov\_dev\_client\_sample**. Expand **Source Files**, and open **prov\_dev\_client\_sample.c**.
3. Near the top of the file, find the `#define` statements for each device protocol as shown below. Make sure only `SAMPLE_AMQP` is uncommented.

Currently, the [MQTT protocol is not supported for TPM Individual Enrollment](#).

```
//  
// The protocol you wish to use should be uncommented  
//  
//#define SAMPLE_MQTT  
//#define SAMPLE_MQTT_OVER_WEBSOCKETS  
#define SAMPLE_AMQP  
//#define SAMPLE_AMQP_OVER_WEBSOCKETS  
//#define SAMPLE_HTTP
```

4. Find the `id_scope` constant, and replace the value with your ID Scope value that you copied earlier.

```
static const char* id_scope = "One00002193";
```

5. Find the definition for the `main()` function in the same file. Make sure the `hsm_type` variable is set to `SECURE_DEVICE_TYPE_TPM` instead of `SECURE_DEVICE_TYPE_X509` as shown below.

```
SECURE_DEVICE_TYPE hsm_type;  
hsm_type = SECURE_DEVICE_TYPE_TPM;  
//hsm_type = SECURE_DEVICE_TYPE_X509;
```

6. Right-click the **prov\_dev\_client\_sample** project and select **Set as Startup Project**.

7. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the

prompt to rebuild the project, select **Yes**, to rebuild the project before running.

The following output is an example of the provisioning device client sample successfully booting up, and connecting to a Device Provisioning Service instance to get IoT hub information and registering:

```
Provisioning API Version: 1.2.7
Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED

Registering... Press enter key to interrupt.

Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING

Registration Information received from service:
test-docs-hub.azure-devices.net, deviceId: test-docs-device
```

- Once the simulated device is provisioned to the IoT hub by your provisioning service, the device ID appears with the hub's **IoT devices**.

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION...	CLOUD TO DEVICE ...
test-docs-device	Enabled	Mon Sep 17 201...	Sas	0	

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

- Close the device client sample output window on your machine.
- Close the TPM simulator window on your machine.
- From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open **Manage Enrollments** for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
- From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open **IoT devices** for your hub, select the check box next to the *DEVICE ID* of the device you registered in this

quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a TPM simulated device on your machine and provisioned it to your IoT hub using the IoT Hub Device Provisioning Service. To learn how to enroll your TPM device programmatically, continue to the quickstart for programmatic enrollment of a TPM device.

[Azure quickstart - Enroll TPM device to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision a simulated TPM device using Java device SDK for Azure IoT Hub Device Provisioning Service

7/30/2020 • 4 minutes to read • [Edit Online](#)

In this quickstart, you create a simulated IoT device on a Windows computer. The simulated device includes a TPM simulator as a Hardware Security Module (HSM). You use device sample Java code to connect this simulated device with your IoT hub using an individual enrollment with the Device Provisioning Service (DPS).

## Prerequisites

- Review of [Auto-provisioning concepts](#).
- Completion of [Set up IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Java SE Development Kit 8](#).
- [Maven](#).
- [Git](#).

### NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the environment

1. Make sure you have [Java SE Development Kit 8](#) installed on your machine.
2. Download and install [Maven](#).
3. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the **Git Bash**, the command-line app that you can use to interact with your local Git repository.
4. Open a command prompt. Clone the GitHub repo for device simulation code sample.

```
git clone https://github.com/Azure/azure-iot-sdk-java.git --recursive
```

5. Run the **TPM** simulator to be the **HSM** for the simulated device. Click **Allow access** to allow changes to *Windows Firewall* settings. It listens over a socket on ports 2321 and 2322. Do not close this window; you need to keep this simulator running until the end of this quickstart guide.

```
.\azure-iot-sdk-java\provisioning\provisioning-tools\tpm-simulator\Simulator.exe
```

```
C:\Users\v-masebo\Desktop\azure-iot-sdk-java\provisioning\provisioning-tools\tpm-simulator\Simulator.exe
Platform server listening on port 2322
TPM command server listening on port 2321
```

6. In a separate command prompt, navigate to the root folder and build the sample dependencies.

```
cd azure-iot-sdk-java
mvn install -DskipTests=true
```

7. Navigate to the sample folder.

```
cd provisioning/provisioning-samples/provisioning-tpm-sample
```

8. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service. Note your *ID Scope* and *Provisioning Service Global Endpoint*.

The screenshot shows the Azure portal interface with the title 'Device Provisioning Service'. The URL in the address bar is <https://ms.portal.azure.com/#resource/subscriptions/.../resourceGroups/test-grp/providers/Microsoft.Devices/ProvisioningServices/dps-certificate>. The top navigation bar includes 'Report a bug', a search icon, a bell icon, a gear icon, a smiley face icon, a question mark icon, and a user profile for 'bailey@contoso.com'.

The main content area displays the 'dps-certificate' Device Provisioning Service resource. On the left, there is a sidebar with icons for Home, Overview, Activity log, Access control (IAM), Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, Manage enrollments, Manage allocation policy, and Monitoring. The 'Overview' tab is selected.

The main panel shows the following details:

- Resource group: test-grp
- Status: Active
- Location: East US
- Subscription: Microsoft Azure Internal Consumption
- Subscription ID: \*\*\*\*
- Service endpoint: dps-certificate.azure-devices-provisioning.net
- Global device endpoint: global.azure-devices-provisioning.net
- ID Scope: One0000A0A
- Pricing and scale tier: S1

A red box highlights the 'Global device endpoint', 'ID Scope', and 'Pricing and scale tier' fields.

Below the details, there is a 'Quick Links' section with four items:

- Azure IoT Hub Device Provisioning Service Documentation
- Learn more about IoT Hub Device Provisioning Service
- Device Provisioning concepts
- Pricing and scale details

9. Edit `src/main/java/samples/com/microsoft/azure/sdk/iot/ProvisioningTpmSample.java` to include your *ID Scope* and *Provisioning Service Global Endpoint* as noted before.

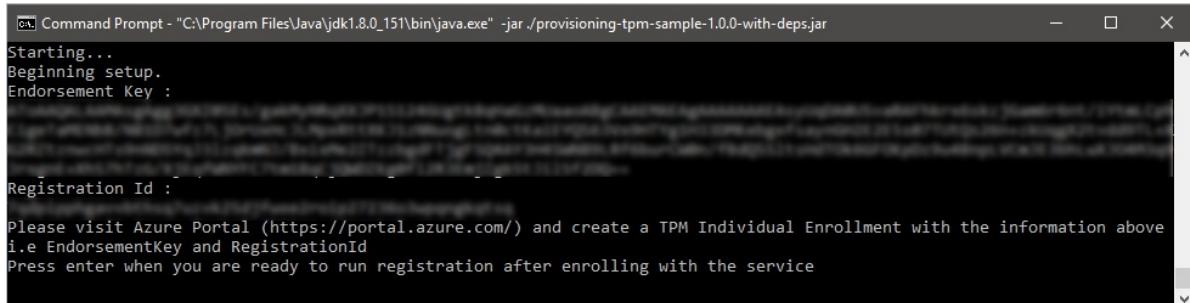
```
private static final String idScope = "[Your ID scope here]";
private static final String globalEndpoint = "[Your Provisioning Service Global Endpoint here]";
private static final ProvisioningDeviceClientTransportProtocol
PROVISIONING_DEVICE_CLIENT_TRANSPORT_PROTOCOL = ProvisioningDeviceClientTransportProtocol.HTTPS;
```

Save the file.

10. Use the following commands to build the project, navigate to the target folder, and execute the created .jar file. Replace the `version` placeholder with your version of Java.

```
mvn clean install  
cd target  
java -jar ./provisioning-tpm-sample-{version}-with-deps.jar
```

11. The program begins running. Note the *Endorsement key* and *Registration ID* for the next section and leave the program running.



The screenshot shows a Windows Command Prompt window titled "Command Prompt - 'C:\Program Files\Java\jdk1.8.0\_151\bin\java.exe' -jar ./provisioning-tpm-sample-1.0.0-with-deps.jar". The window displays the following text:

```
Starting...
Beginning setup.
Endorsement Key : [REDACTED]  
  
Registration Id : [REDACTED]  
  
Please visit Azure Portal (https://portal.azure.com/) and create a TPM Individual Enrollment with the information above  
i.e EndorsementKey and RegistrationId  
Press enter when you are ready to run registration after enrolling with the service
```

## Create a device enrollment entry

The Azure IoT Device Provisioning Service supports two types of enrollments:

- **Enrollment groups:** Used to enroll multiple related devices.
- **Individual enrollments:** Used to enroll a single device.

This article demonstrates individual enrollments.

1. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service.
2. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.
3. In the **Add Enrollment** panel, enter the following information:
  - Select **TPM** as the identity attestation *Mechanism*.
  - Enter the *Registration ID* and *Endorsement key* for your TPM device from the values you noted previously.
  - Select an IoT hub linked with your provisioning service.
  - Optionally, you may provide the following information:
    - Enter a unique *Device ID*. Make sure to avoid sensitive data while naming your device. If you choose not to provide one, the registration ID will be used to identify the device instead.
    - Update the **Initial device twin state** with the desired initial configuration for the device.
  - Once complete, press the **Save** button.

The screenshot shows two windows side-by-side. On the left is the 'dps-certificate - Manage enrollments' blade under 'Device Provisioning Service'. It has a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings, Quick Start, Shared access policies, Linked IoT hubs, and Certificates. The 'Manage enrollments' option is highlighted with a red box. The main area shows tabs for 'Enrollment Groups' and 'Individual Enrollments', with 'Individual Enrollments' selected. A red box highlights the '+ Add individual enrollment' button. On the right is the 'Add Enrollment' dialog. It includes fields for Mechanism (set to TPM), Endorsement key, Registration ID (set to 'Individual enrollment registration id'), IoT Hub Device ID (set to 'Device ID'), IoT Edge device (set to 'False'), and a dropdown for device assignment. Below these are sections for handling device data on re-provisioning (set to 'Re-provision and migrate data') and linking to an IoT hub (set to 'test-docs-dps.azure-devices.net'). A 'Save' button is at the top right of the dialog.

On successful enrollment, the *Registration ID* of your device appears in the list under the *Individual Enrollments* tab.

## Simulate the device

1. On the command window running the Java sample code on your machine, press *Enter* to continue running the application. Notice the messages that simulate the device booting and connecting to the Device Provisioning Service to get your IoT hub information.

```
C:\> Command Prompt - "C:\Program Files\Java\jdk1.8.0_151\bin\java.exe" -jar ./provisioning-tpm-sample-1.0.0-with-deps.jar
Press enter when you are ready to run registration after enrolling with the service
Waiting for Provisioning Service to register
Waiting for Provisioning Service to register
IotHub Uri :
Device ID :
log4j:WARN No appenders could be found for logger (com.microsoft.azure.sdk.iot.device.IotHubConnectionString).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Sending message from device to IoT Hub...
Press any key to exit...
Message received! Response status: OK_EMPTY
```

2. On successful provisioning of your simulated device to the IoT hub linked with your provisioning service, the device ID appears on the hub's **IoT devices** blade.

The screenshot shows the Azure IoT Hub Device Provisioning Service portal. On the left, there's a navigation menu with sections like SETTINGS, EXPLORERS, and AUTOMATIC DEVICE MANAGEMENT. Under EXPLORERS, the 'IoT devices' item is highlighted with a red box. The main pane displays a query editor with a sample SQL query and an 'Execute' button. Below it is a table showing device details:

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION...	CLOUD TO DEVICE ...
Java-device	Enabled	Sas	0		

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. Close the TPM simulator window on your machine.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
4. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open the **IoT devices** blade for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a TPM simulated device on your machine and provisioned it to your IoT hub using the IoT Hub Device Provisioning Service. To learn how to enroll your TPM device programmatically, continue to the quickstart for programmatic enrollment of a TPM device.

[Azure quickstart - Enroll TPM device to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision a simulated TPM device using C# device SDK for IoT Hub Device Provisioning Service

12/10/2019 • 4 minutes to read • [Edit Online](#)

These steps show you how to use the [Azure IoT Samples for C#](#) to simulate a TPM device on a development machine running the Windows OS. The sample also connects the simulated device to an IoT Hub using the Device Provisioning Service.

The sample code uses the Windows TPM simulator as the [Hardware Security Module \(HSM\)](#) of the device.

If you're unfamiliar with the process of autopropvisioning, be sure to also review [Auto-provisioning concepts](#). Also make sure you've completed the steps in [Set up IoT Hub Device Provisioning Service with the Azure portal](#) before continuing.

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual Enrollments](#): Used to enroll a single device.

This article will demonstrate individual enrollments.

## NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the development environment

1. Make sure you have the [.NET Core 2.1 SDK or later](#) installed on your machine.
2. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the **Git Bash**, the command-line app that you can use to interact with your local Git repository.
3. Open a command prompt or Git Bash. Clone the Azure IoT Samples for C# GitHub repo:

```
git clone https://github.com/Azure-Samples/azure-iot-samples-csharp.git
```

## Provision the simulated device

1. Sign in to the Azure portal. Select the **All resources** button on the left-hand menu and open your Device Provisioning service. From the **Overview** blade, note the **ID Scope** value.

2. In a command prompt, change directories to the project directory for the TPM device provisioning sample.

```
cd ..\azure-iot-samples-csharp\provisioning\Samples\device\TpmSample
```

3. Type the following command to build and run the TPM device provisioning sample. Replace the <IDScope> value with the ID Scope for your provisioning service.

```
dotnet run <IDScope>
```

This command will launch the TPM chip simulator in a separate command prompt. On Windows, you may encounter a Windows Security Alert that asks whether you want to allow Simulator.exe to communicate on public networks. For the purposes of this sample, you may cancel the request.

4. The original command window displays the *Endorsement key*, the *Registration ID*, and a suggested *Device ID* needed for device enrollment. Take note of these values. You will use these value to create an individual enrollment in your Device Provisioning Service instance.

#### NOTE

Do not confuse the window that contains command output with the window that contains output from the TPM simulator. You may have to select the original command window to bring it to the foreground.

```
Starting TPM simulator.
Extracting endorsement key.
In your Azure Device Provisioning Service please go to 'Manage enrollments' and select 'Individual Enrollments'. Select 'Add' then fill in the following:
  Mechanism: TPM
  Registration ID: testtpmregistration1
  Endorsement key:
Device ID: iothubtpmdevice1 (or any other valid DeviceID)
Press ENTER when ready.
```

5. In the Azure portal, from the Device Provisioning Service menu, select **Manage enrollments**. Select the

Individual Enrollments tab and select the Add individual enrollment button at the top.

6. In the Add Enrollment panel, enter the following information:

- Select TPM as the identity attestation *Mechanism*.
- Enter the *Registration ID* and *Endorsement key* for your TPM device from the values you noted previously.
- Select an IoT hub linked with your provisioning service.
- Optionally, you may provide the following information:
  - Enter a unique *Device ID* (you can use the suggested one or provide your own). Make sure to avoid sensitive data while naming your device. If you choose not to provide one, the registration ID will be used to identify the device instead.
  - Update the **Initial device twin state** with the desired initial configuration for the device.
- Once complete, press the **Save** button.

The screenshot shows two windows side-by-side. On the left is the 'sample-provisioning-service - Manage enrollments' blade, which has a sidebar with 'Manage enrollments' highlighted. The main area shows an 'Enrollment Groups' section with 'Individual Enrollments' selected. On the right is the 'Add Enrollment' dialog. It contains fields for 'Mechanism' (set to 'TPM'), 'Endorsement key' (set to 'Endorsement key'), 'Registration ID' (set to 'Individual enrollment registration id'), 'IoT Hub Device ID' (set to 'iothubtpmdevice1'), and 'IoT Edge device' (set to 'False'). Below these are dropdowns for 'Select how you want to assign devices to hubs' (set to 'Evenly weighted distribution') and 'Select the IoT hubs this device can be assigned to' (set to 'sample-iot-hub1.azure-devices.net'). A 'Link a new IoT hub' button is also present. At the bottom, there's a note about device data handling on re-provisioning ('Re-provision and migrate data') and a note about device twin support ('Device Twin is only supported for standard tier IoT hubs. Learn more about standard tier.'). The 'Initial Device Twin State' field is empty, showing a JSON object placeholder: { "tags": {} }.

On successful enrollment, the *Registration ID* of your device will appear in the list under the *Individual Enrollments* tab.

7. Press *Enter* in the command window (the one that displayed the *Endorsement key*, *Registration ID*, and suggested *Device ID*) to enroll the simulated device. Notice the messages that simulate the device booting and connecting to the Device Provisioning Service to get your IoT hub information.
8. Verify that the device has been provisioned. On successful provisioning of the simulated device to the IoT hub linked with your provisioning service, the device ID appears on the hub's **IoT devices** blade.

The screenshot shows the Azure IoT Hub Device Provisioning Service portal. The left sidebar has a tree view with nodes like Overview, Activity log, Access control (IAM), Tags, Events, Settings (Shared access policies, Pricing and scale, Operations monitoring, IP Filter, Certificates, Properties, Locks, Automation script), Explorers (Query explorer, IoT devices), and Automatic Device Management. The 'IoT devices' node is highlighted with a red box. The main content area has a search bar, an 'Add' button, a 'Refresh' button, and a 'Delete' button. A message says 'You can use this tool to view, create, update, and delete devices on your IoT Hub.' Below it is a 'Query' section with a sample query: 'SELECT \* FROM devices WHERE optional (e.g. tags.location='US')'. An 'Execute' button is below the query. A table lists devices with columns: DEVICE ID, STATUS, LAST ACTIVITY, LAST STATUS UPDATE, AUTHENTICATION TYPE, and CLOUD TO DEVICE MESS... The first row shows 'iothubtpmdevice1' with 'Enabled' status, 'Sas' authentication type, and 0 messages.

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPDATE	AUTHENTICATION TYPE	CLOUD TO DEVICE MESS...
iothubtpmdevice1	Enabled			Sas	0

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. Close the TPM simulator window on your machine.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. At the top of the **Overview** blade, press **Delete** at the top of the pane.
4. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. At the top of the **Overview** blade, press **Delete** at the top of the pane.

## Next steps

In this quickstart, you've created a TPM simulated device on your machine and provisioned it to your IoT hub using the IoT Hub Device Provisioning Service. To learn how to enroll your TPM device programmatically, continue to the quickstart for programmatic enrollment of a TPM device.

[Azure quickstart - Enroll TPM device to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision a simulated TPM device using Node.js device SDK for IoT Hub Device Provisioning Service

7/30/2020 • 7 minutes to read • [Edit Online](#)

In this quickstart, you create a simulated IoT device on a Windows computer. The simulated device includes a TPM simulator as a Hardware Security Module (HSM). You use device sample Node.js code to connect this simulated device with your IoT hub using an individual enrollment with the Device Provisioning Service (DPS).

## Prerequisites

- Review of [Auto-provisioning concepts](#).
- Completion of [Set up IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Node.js v4.0+](#).
- [Git](#).

### NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the environment

1. Make sure you have [Nodejs v4.0 or above](#) installed on your machine.
2. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the **Git Bash**, the command-line app that you can use to interact with your local Git repository.

## Simulate a TPM device

1. Open a command prompt or Git Bash. Clone the `azure-utpm-c` GitHub repo:

```
git clone https://github.com/Azure/azure-utpm-c.git --recursive
```

2. Navigate to the GitHub root folder and run the **TPM** simulator to be the **HSM** for the simulated device. It listens over a socket on ports 2321 and 2322. Do not close this command window; you need to keep this simulator running until the end of this quickstart guide:

```
.\azure-utpm-c\tools\tpm_simulator\Simulator.exe
```

3. Create a new empty folder called **registerdevice**. In the **registerdevice** folder, create a `package.json` file using the following command at your command prompt. Make sure to answer all questions asked by `npm` or accept the defaults if they suit you:

```
npm init
```

#### 4. Install the following precursor packages:

```
npm install node-gyp -g
npm install ffi -g
```

##### NOTE

There are some known issues to installing the above packages. To resolve these issues, run

```
npm install --global --production windows-build-tools
```

 using a command prompt in **Run as administrator** mode, run 

```
SET VCTargetsPath=C:\Program Files (x86)\MSBuild\Microsoft.Cpp\v4.0\V140
```

 after replacing the path with your installed version, and then rerun the above installation commands.

#### 5. Install the following packages containing the components used during registration:

- a security client that works with TPM: `azure-iot-security-tpm`
- a transport for the device to connect to the Device Provisioning Service: either `azure-iot-provisioning-device-http` or `azure-iot-provisioning-device-amqp`
- a client to use the transport and security client: `azure-iot-provisioning-device`

Once the device is registered, you can use the usual IoT Hub Device Client packages to connect the device using the credentials provided during registration. You will need:

- the device client: `azure-iot-device`
- a transport: any of `azure-iot-device-amqp`, `azure-iot-device-mqtt`, or `azure-iot-device-http`
- the security client that you already installed: `azure-iot-security-tpm`

##### NOTE

The samples below use the `azure-iot-provisioning-device-http` and `azure-iot-device-mqtt` transports.

You can install all of these packages at once by running the following command at your command prompt in the **registerdevice** folder:

```
npm install --save azure-iot-device azure-iot-device-mqtt azure-iot-security-tpm azure-iot-
provisioning-device-http azure-iot-provisioning-device
```

#### 6. Using a text editor, create a new **ExtractDevice.js** file in the **registerdevice** folder.

#### 7. Add the following `require` statements at the start of the **ExtractDevice.js** file:

```
'use strict';

var tpmSecurity = require('azure-iot-security-tpm');
var tssJs = require("tss.js");

var myTpm = new tpmSecurity.TpmSecurityClient(undefined, new tssJs.Tpm(true));
```

8. Add the following function to implement the method:

```
myTpm.getEndorsementKey(function(err, endorsementKey) {
  if (err) {
    console.log('The error returned from get key is: ' + err);
  } else {
    console.log('the endorsement key is: ' + endorsementKey.toString('base64'));
    myTpm.getRegistrationId((getRegistrationIdError, registrationId) => {
      if (getRegistrationIdError) {
        console.log('The error returned from get registration id is: ' + getRegistrationIdError);
      } else {
        console.log('The Registration Id is: ' + registrationId);
        process.exit();
      }
    });
  }
});
```

9. Save and close the `ExtractDevice.js` file. Run the sample:

```
node ExtractDevice.js
```

10. The output window displays the *Endorsement key* and the *Registration ID* needed for device enrollment. Note down these values.

## Create a device entry

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual enrollments](#): Used to enroll a single device.

This article demonstrates individual enrollments.

1. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service.
2. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.
3. In the **Add Enrollment** panel, enter the following information:
  - Select **TPM** as the identity attestation *Mechanism*.
  - Enter the *Registration ID* and *Endorsement key* for your TPM device from the values you noted previously.
  - Select an IoT hub linked with your provisioning service.
  - Optionally, you may provide the following information:
    - Enter a unique *Device ID*. Make sure to avoid sensitive data while naming your device. If you choose not to provide one, the registration ID will be used to identify the device instead.
    - Update the **Initial device twin state** with the desired initial configuration for the device.
  - Once complete, press the **Save** button.

On successful enrollment, the *Registration ID* of your device appears in the list under the *Individual Enrollments* tab.

## Register the device

1. In the Azure portal, select the **Overview** blade for your Device Provisioning service and note the *Global Device Endpoint* and *ID Scope* values.

2. Using a text editor, create a new **RegisterDevice.js** file in the **registerdevice** folder.
3. Add the following `require` statements at the start of the **RegisterDevice.js** file:

```
'use strict';

var ProvisioningTransport = require('azure-iot-provisioning-device-http').Http;
var iotHubTransport = require('azure-iot-device-mqtt').Mqtt;
var Client = require('azure-iot-device').Client;
var Message = require('azure-iot-device').Message;
var tpmSecurity = require('azure-iot-security-tpm');
var ProvisioningDeviceClient = require('azure-iot-provisioning-device').ProvisioningDeviceClient;
```

#### NOTE

The Azure IoT SDK for Node.js supports additional protocols like *AMQP*, *AMQP WS*, and *MQTT WS*. For more examples, see [Device Provisioning Service SDK for Node.js samples](#).

4. Add `globalDeviceEndpoint` and `idScope` variables and use them to create a `ProvisioningDeviceClient` instance. Replace `{globalDeviceEndpoint}` and `{idScope}` with the *Global Device Endpoint* and *ID Scope* values from Step 1:

```
var provisioningHost = '{globalDeviceEndpoint}';
var idScope = '{idScope}';

var tssJs = require("tss.js");
var securityClient = new tpmSecurity.TpmSecurityClient('', new tssJs.Tpm(true));
// if using non-simulated device, replace the above line with following:
//var securityClient = new tpmSecurity.TpmSecurityClient();

var provisioningClient = ProvisioningDeviceClient.create(provisioningHost, idScope, new
ProvisioningTransport(), securityClient);
```

5. Add the following function to implement the method on the device:

```

provisioningClient.register(function(err, result) {
  if (err) {
    console.log("error registering device: " + err);
  } else {
    console.log('registration succeeded');
    console.log('assigned hub=' + result.registrationState.assignedHub);
    console.log('deviceId=' + result.registrationState.deviceId);
    var tpmAuthenticationProvider =
      tpmSecurity.TpmAuthenticationProvider.fromTpmSecurityClient(result.registrationState.deviceId,
      result.registrationState.assignedHub, securityClient);
    var hubClient = Client.fromAuthenticationProvider(tpmAuthenticationProvider, iotHubTransport);

    var connectCallback = function (err) {
      if (err) {
        console.error('Could not connect: ' + err.message);
      } else {
        console.log('Client connected');
        var message = new Message('Hello world');
        hubClient.sendEvent(message, printResultFor('send'));
      }
    };

    hubClient.open(connectCallback);

    function printResultFor(op) {
      return function printResult(err, res) {
        if (err) console.log(op + ' error: ' + err.toString());
        if (res) console.log(op + ' status: ' + res.constructor.name);
        process.exit(1);
      };
    }
  });
});

```

6. Save and close the **RegisterDevice.js** file. Run the sample:

```
node RegisterDevice.js
```

7. Notice the messages that simulate the device booting and connecting to the Device Provisioning Service to get your IoT hub information. On successful provisioning of your simulated device to the IoT hub linked with your provisioning service, the device ID appears on the hub's **IoT devices** blade.

The screenshot shows the Azure IoT Hub Device Provisioning Service portal. On the left, there's a navigation menu with sections like SETTINGS, EXPLORERS, and AUTOMATIC DEVICE MANAGEMENT. Under EXPLORERS, the 'IoT devices' item is highlighted with a red box. The main pane displays a query editor with a sample SQL query and an 'Execute' button. Below that is a table showing device details:

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION...	CLOUD TO DEVICE ...
test-docs-device	Enabled	Mon Sep 17 201...	Sas	0	

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#)

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. Close the TPM simulator window on your machine.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
4. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open the **IoT devices** blade for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a TPM simulated device on your machine and provisioned it to your IoT hub using the IoT Hub Device Provisioning Service. To learn how to enroll your TPM device programmatically, continue to the quickstart for programmatic enrollment of a TPM device.

[Azure quickstart - Enroll TPM device to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Create and provision a simulated TPM device using Python device SDK for IoT Hub Device Provisioning Service

7/30/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you create a simulated IoT device on a Windows computer. The simulated device includes a TPM simulator as a Hardware Security Module (HSM). You use device sample Python code to connect this simulated device with your IoT hub using an individual enrollment with the Device Provisioning Service (DPS).

## Prerequisites

- Review of [Auto-provisioning concepts](#).
- Completion of [Set up IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Visual Studio 2015+](#) with Desktop development with C++.
- [CMake build system](#).
- [Git](#).

### IMPORTANT

This article only applies to the deprecated V1 Python SDK. Device and service clients for the IoT Hub Device Provisioning Service are not yet available in V2. The team is currently hard at work to bring V2 to feature parity.

### NOTE

The initial device twin state configuration is available only in the standard tier of IoT Hub. For more information about the basic and standard IoT Hub tiers, see [How to choose the right IoT Hub tier](#).

## Prepare the environment

1. Make sure you have installed either [Visual Studio](#) 2015 or later, with the 'Desktop development with C++' workload enabled for your Visual Studio installation.
2. Download and install the [CMake build system](#).
3. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the [Git Bash](#), the command-line app that you can use to interact with your local Git repository.
4. Open a command prompt or Git Bash. Clone the GitHub repo for device simulation code sample:

```
git clone --single-branch --branch v1-deprecated https://github.com/Azure/azure-iot-sdk-python.git --recursive
```

5. Create a folder in your local copy of this GitHub repo for CMake build process.

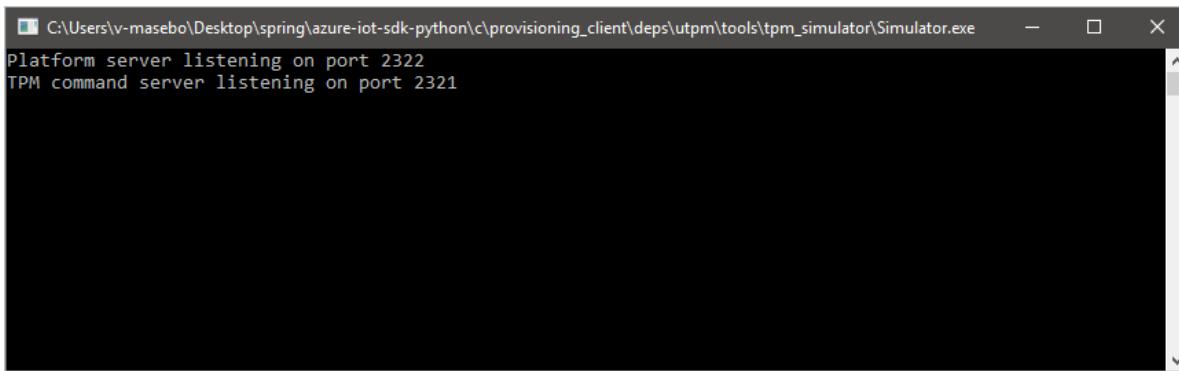
```
cd azure-iot-sdk-python/c  
mkdir cmake  
cd cmake
```

6. The code sample uses a Windows TPM simulator. Run the following command to enable the SAS token authentication. It also generates a Visual Studio solution for the simulated device.

```
cmake -Duse_prov_client:BOOL=ON -Duse_tpm_simulator:BOOL=ON ..
```

7. In a separate command prompt, navigate to the TPM simulator folder and run the **TPM** simulator to be the **HSM** for the simulated device. Click **Allow Access**. It listens over a socket on ports 2321 and 2322. Do not close this command window; you will need to keep this simulator running until the end of this quickstart guide.

```
.\azure-iot-sdk-python\c\provisioning_client\deps\utpm\tools\tpm_simulator\Simulator.exe
```



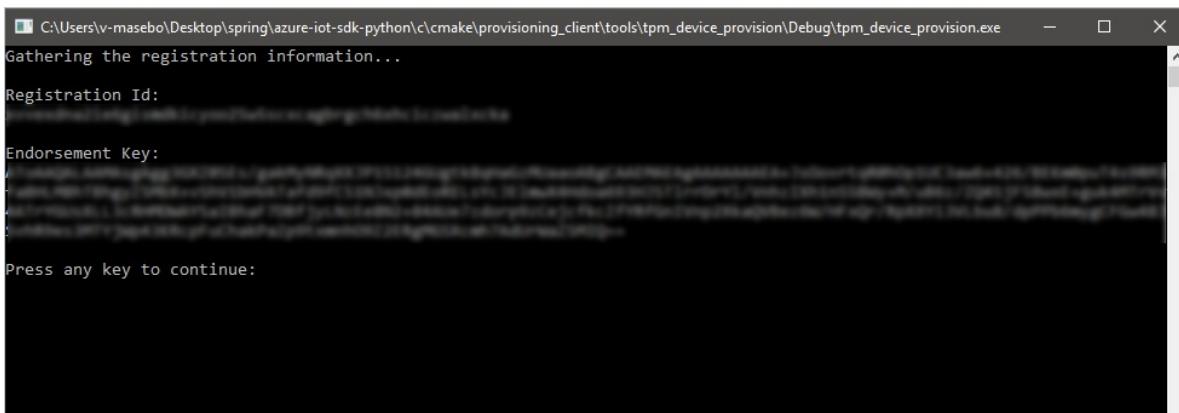
## Create a device enrollment entry

The Azure IoT Device Provisioning Service supports two types of enrollments:

- **Enrollment groups**: Used to enroll multiple related devices.
- **Individual enrollments**: Used to enroll a single device.

This article demonstrates individual enrollments.

1. Open the solution generated in the *cmake* folder named `azure_iot_sdks.sln`, and build it in Visual Studio.
2. Right-click the **tpm\_device\_provision** project and select **Set as Startup Project**. Run the solution. The output window displays the **Endorsement key** and the **Registration ID** needed for device enrollment. Note down these values.

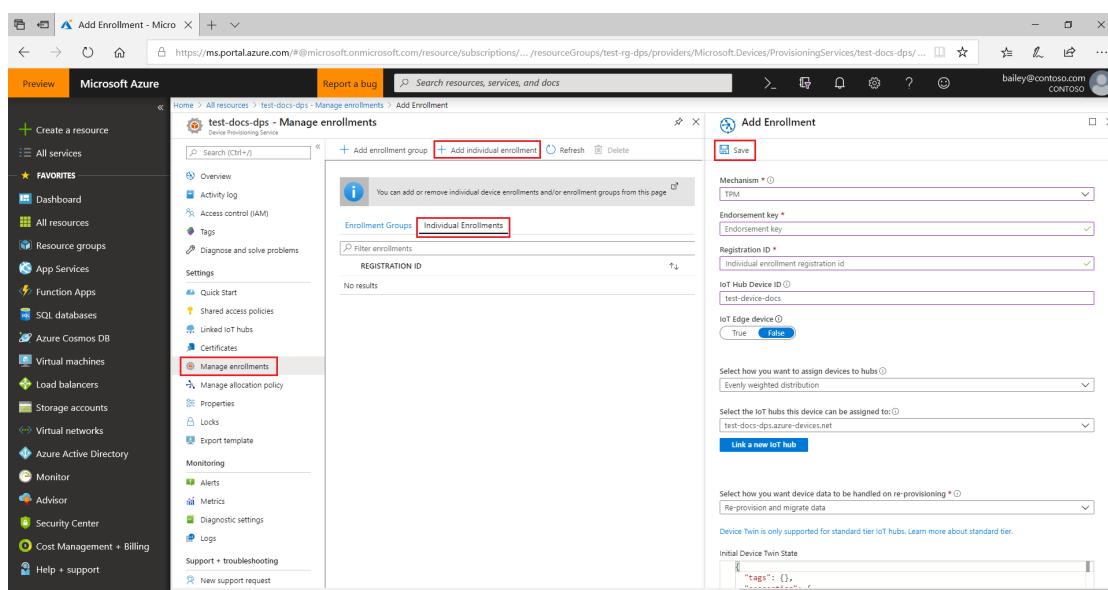


3. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service.

4. From the Device Provisioning Service menu, select **Manage enrollments**. Select **Individual Enrollments** tab and select the **Add individual enrollment** button at the top.

5. In the **Add Enrollment** panel, enter the following information:

- Select **TPM** as the identity attestation *Mechanism*.
- Enter the *Registration ID* and *Endorsement key* for your TPM device from the values you noted previously.
- Select an IoT hub linked with your provisioning service.
- Optionally, you may provide the following information:
  - Enter a unique *Device ID*. Make sure to avoid sensitive data while naming your device. If you choose not to provide one, the registration ID will be used to identify the device instead.
  - Update the **Initial device twin state** with the desired initial configuration for the device.
- Once complete, press the **Save** button.



On successful enrollment, the *Registration ID* of your device will appear in the list under the *Individual Enrollments* tab.

## Simulate the device

1. Download and install [Python 2.x or 3.x](#). Make sure to use the 32-bit or 64-bit installation as required by your setup. When prompted during the installation, make sure to add Python to your platform-specific environment variables.

- If you are using Windows OS, then [Visual C++ redistributable package](#) to allow the use of native DLLs from Python.

2. Follow [these instructions](#) to build the Python packages.

### NOTE

If running the `build_client.cmd` make sure to use the `--use-tpm-simulator` flag.

#### NOTE

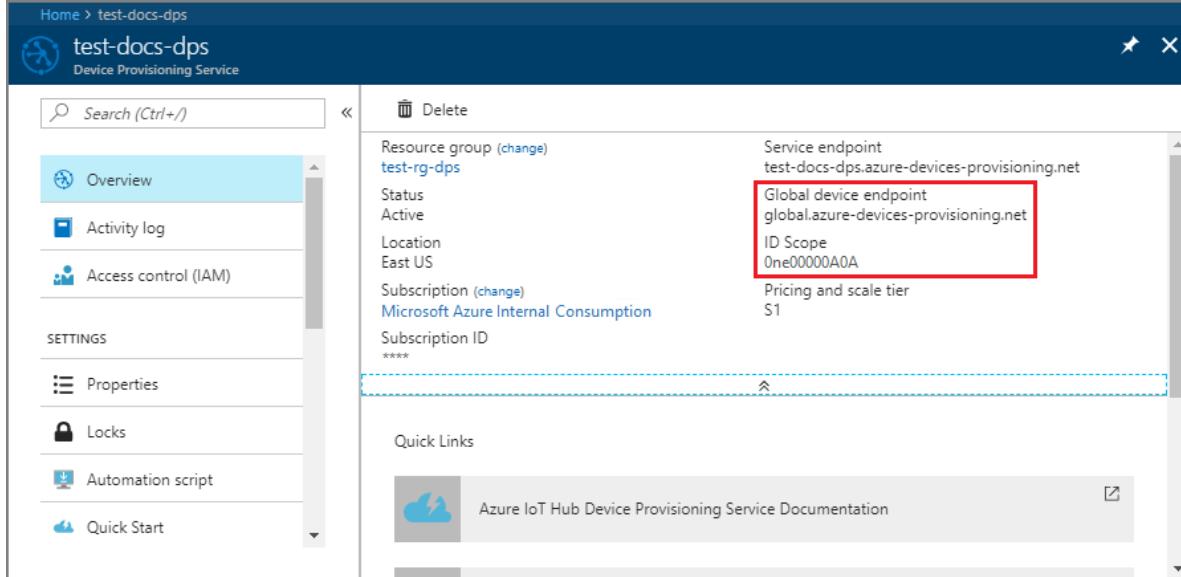
If using `pip` make sure to also install the `azure-iot-provisioning-device-client` package. Note that the released PIP packages are using the real TPM, not the simulator. To use the simulator you need to compile from the source using the `--use-tpm-simulator` flag.

3. Navigate to the samples folder.

```
cd azure-iot-sdk-python/provisioning_device_client/samples
```

4. Using your Python IDE, edit the python script named `provisioning_device_client_sample.py`. Modify the `GLOBAL_PROV_URI` and `ID_SCOPE` variables to the values noted previously. Also make sure `SECURITY_DEVICE_TYPE` is set to `ProvisioningSecurityDeviceType.TPM`

```
GLOBAL_PROV_URI = "{globalServiceEndpoint}"
ID_SCOPE = "{idScope}"
SECURITY_DEVICE_TYPE = ProvisioningSecurityDeviceType.TPM
PROTOCOL = ProvisioningTransportProvider.HTTP
```



5. Run the sample.

```
python provisioning_device_client_sample.py
```

6. Notice the messages that simulate the device booting and connecting to the Device Provisioning Service to get your IoT hub information.

```
Administrator: Developer Command Prompt for VS 2017 - python provisioning_device_client_sample.py
HTTP Status: 200

Date: Thu, 14 Dec 2017 18:35:23 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
x-ms-request-id: 3b882f2e-ac4e-429a-90ca-ba722686f51f
Strict-Transport-Security: max-age=31536000; includeSubDomains

{"operationId": "0.7020d3135a225f0e.45d32292-2b8c-4f60-b44e-23d4dfef5cd6", "status": "assigned", "registrationState": {"tpm": "authenticated"}, "authenticationKey": "ADQATIKWu5htwBudC5sanxtJG57wp4jH+rqvJQ2jwghLsLpdZFMjIHHKYEEfSYD0KzpE84MAQDM9LczAliunNzoxw9Vq4Qb8W2tf1B34xu165L8DXdydw+pebl78IPVgmxDGsrwaovcx2uwFpdodJBX3/eJgbUwJltXvh0ti0sy3Rc5yFS0Pw0zra9Ubiz6ztUmLT8NTLtnRV2bbI4zT6dIwqvSwjOas3z2gMs40dX91nCM690cm4osv75tlwQ4hoaCef45wAmXcNGo/27jmciIQIWamFpv75hQ2Wi6tCuARFzjJxgIcoqos9y0LhBw32w3L5R1AQgK0lwB5gNZC/oRw3XcJfYHgkCwgE0p0+epY77wBFdm+7k4033A40VT1f7fb8l1nTQzgjXNwKlhJAI4AIEgiJ20TwM3+XHH0+GVnt3+T+2uH/1fYw6+cxd2J/VP/H+70jzbS2mxHGRX41bi1fvH0xi17VrcW9pjqdK6esu/pIBUP4ZV1H/u5xGdFjt+Ca9vtRdczEphemD18V2DDLM3OCa0nvr/23zpsxSrwdPctInNj0c1nw4KMDtJjtTmmix8/rL4VK0DQl6a0CxCz8ZG2Sp5f0s564D+2niY4unVo4/r4ck8FU0i41fHFnZe4s8CUpmfS6/5NeRQwZrsV4Yc1Co5Rxos7j6/ijT2L3uxnSTD7zH7ZC181AmTxwU6t7D+FfVros77x", "iotHubConnectionString": "HostName=iot-dps.azure-devices.net;DeviceId=pythonDevice2;SharedAccessSignature=sv=2017-12-14&t=1513360000&r=https%3A%2F%2Fiot-dps.azure-devices.net%2Fdevices%2FpythonDevice2&sig=5a32c4690000\u0022"}}

Register device callback:
register_result = OK
iothub_uri = iot-dps.azure-devices.net
user_context = None

Device successfully registered!
```

- On successful provisioning of your simulated device to the IoT hub linked with your provisioning service, the device ID appears on the hub's **IoT devices** blade.

The screenshot shows the 'IoT devices' blade in the Azure IoT Hub. The left sidebar has a red box around the 'IoT devices' item under 'EXPLORERS'. The main area displays a summary message, a query editor, and a table of devices.

You can use this tool to view, create, update, and delete devices on your IoT Hub.

Query i

```
SELECT * FROM devices WHERE  
optional (e.g. tags.location='US')
```

Execute

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPD...	AUTHENTICATION...	CLOUD TO DEVICE ...
python-device	Enabled	Mon Sep 17 201...	Sas	0	

If you changed the *initial device twin state* from the default value in the enrollment entry for your device, it can pull the desired twin state from the hub and act accordingly. For more information, see [Understand and use device twins in IoT Hub](#).

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the device client sample output window on your machine.
2. Close the TPM simulator window on your machine.
3. From the left-hand menu in the Azure portal, select **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then select the **Individual Enrollments** tab. Select the check box next to the *REGISTRATION ID* of the device you enrolled in this quickstart, and press the **Delete** button at the top of the pane.
4. From the left-hand menu in the Azure portal, select **All resources** and then select your IoT hub. Open the **IoT devices** blade for your hub, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've created a TPM simulated device on your machine and provisioned it to your IoT hub using the IoT Hub Device Provisioning Service. To learn how to enroll your TPM device programmatically, continue to the quickstart for programmatic enrollment of a TPM device.

[Azure quickstart - Enroll TPM device to Azure IoT Hub Device Provisioning Service](#)

# Quickstart: Enroll X.509 devices to the Device Provisioning Service using Java

7/30/2020 • 5 minutes to read • [Edit Online](#)

In this quickstart, you use Java to programmatically enroll a group of X.509 simulated devices to the Azure IoT Hub Device Provisioning Service. Devices are enrolled to a provisioning service instance by creating an enrollment group, or an individual enrollment. This quickstart shows how to create both types of enrollments by using the Java Service SDK and a sample Java application.

## Prerequisites

- Completion of [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Java SE Development Kit 8](#). This quickstart installs the [Java Service SDK](#) below. It works on both Windows and Linux. This quickstart uses Windows.
- [Maven 3](#).
- [Git](#).

## Download and modify the Java sample code

This section uses a self-signed X.509 certificate, it is important to keep in mind the following points:

- Self-signed certificates are for testing only, and should not be used in production.
- The default expiration date for a self-signed certificate is one year.

The following steps show how to add the provisioning details of your X.509 device to the sample code.

1. Open a command prompt. Clone the GitHub repo for device enrollment code sample using the [Java Service SDK](#):

```
git clone https://github.com/Azure/azure-iot-sdk-java.git --recursive
```

2. In the downloaded source code, navigate to the sample folder *azure-iot-sdk-java/provisioning/provisioning-samples/service-enrollment-group-sample*. Open the file */src/main/java/samples/com/microsoft/azure/sdk/iot/ServiceEnrollmentGroupSample.java* in an editor of your choice, and add the following details:

- a. Add the `[Provisioning Connection String]` for your provisioning service, from the portal as following:
  - a. Navigate to your provisioning service in the [Azure portal](#).
  - b. Open the **Shared access policies**, and select a policy, which has the *EnrollmentWrite* permission.
  - c. Copy the **Primary key connection string**.

- d. In the sample code file *ServiceEnrollmentGroupSample.java*, replace the [Provisioning Connection String] with the Primary key connection string.

```
private static final String PROVISIONING_CONNECTION_STRING = "[Provisioning Connection String]";
```

- b. Add the root certificate for the group of devices. If you need a sample root certificate, use the *X.509 certificate generator* tool as follows:
- In a command window, navigate to the folder *azure-iot-sdk-java/provisioning/provisioning-tools/provisioning-x509-cert-generator*.
  - Build the tool by running the following command:

```
mvn clean install
```

- c. Run the tool using the following commands:

```
cd target
java -jar ./provisioning-x509-cert-generator-{version}-with-deps.jar
```

- d. When prompted, you may optionally enter a *Common Name* for your certificates.
- e. The tool locally generates a **Client Cert**, the **Client Cert Private Key**, and the **Root Cert**.
- f. Copy the **Root Cert**, including the lines **-----BEGIN CERTIFICATE-----** and **-----END CERTIFICATE-----**.
- g. Assign the value of the **Root Cert** to the parameter **PUBLIC\_KEY\_CERTIFICATE\_STRING** as shown below:

```

private static final String PUBLIC_KEY_CERTIFICATE_STRING =
"-----BEGIN CERTIFICATE-----\n" +
"xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx\n" +
"-----END CERTIFICATE-----\n";

```

h. Close the command window, or enter **n** when prompted for *Verification Code*.

c. Optionally, you may configure your provisioning service through the sample code:

- To add this configuration to the sample, follow these steps:

a. Navigate to the IoT hub linked to your provisioning service in the [Azure portal](#). Open the **Overview** tab for the hub, and copy the **Hostname**. Assign this **Hostname** to the **IOTHUB\_HOST\_NAME** parameter.

```
private static final String IOTHUB_HOST_NAME = "[Host name].azure-devices.net";
```

b. Assign a friendly name to the **DEVICE\_ID** parameter, and keep the **PROVISIONING\_STATUS** as the default **ENABLED** value.

- OR, if you choose not to configure your provisioning service, make sure to comment out or delete the following statements in the *ServiceEnrollmentGroupSample.java* file:

```

enrollmentGroup.setIoTHubHostName(IOTHUB_HOST_NAME);           // Optional
parameter.
enrollmentGroup.setProvisioningStatus(ProvisioningStatus.ENABLED); // Optional
parameter.

```

d. Study the sample code. It creates, updates, queries, and deletes a group enrollment for X.509 devices. To verify successful enrollment in portal, temporarily comment out the following lines of code at the end of the *ServiceEnrollmentGroupSample.java* file:

```

// **** Delete info of enrollmentGroup
*****
System.out.println("\nDelete the enrollmentGroup...");
provisioningServiceClient.deleteEnrollmentGroup(enrollmentGroupId);

```

e. Save the file *ServiceEnrollmentGroupSample.java*.

## Build and run sample group enrollment

The Azure IoT Device Provisioning Service supports two types of enrollments:

- **Enrollment groups**: Used to enroll multiple related devices.
- **Individual enrollments**: Used to enroll a single device.

This procedure uses an enrollment group. The next section uses an individual enrollment.

1. Open a command window, and navigate to the folder `azure-iot-sdk-java/provisioning/provisioning-samples/service-enrollment-group-sample`.

2. Build the sample code by using this command:

```
mvn install -DskipTests
```

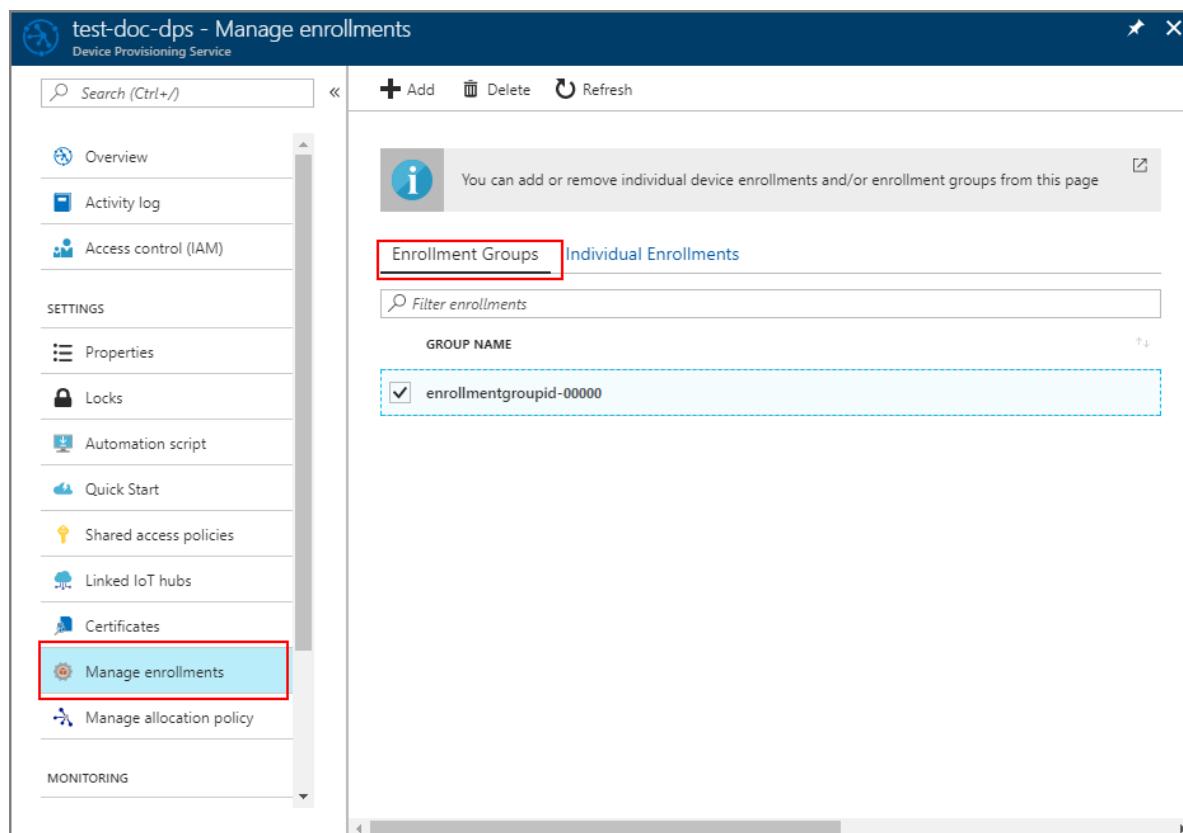
This command downloads the Maven package [com.microsoft.azure.sdk.iot.provisioning.service](#) to your machine. This package includes the binaries for the Java service SDK, that the sample code needs to build. If you ran the *X.509 certificate generator* tool in the preceding section, this package will be already downloaded on your machine.

3. Run the sample by using these commands at the command window:

```
cd target  
java -jar ./service-enrollment-group-sample-{version}-with-deps.jar
```

4. Observe the output window for successful enrollment.

5. Navigate to your provisioning service in the Azure portal. Click **Manage enrollments**. Notice that your group of X.509 devices appears under the **Enrollment Groups** tab, with an autogenerated *GROUP NAME*.



## Modifications to enroll a single X.509 device

To enroll a single X.509 device, modify the *individual enrollment* sample code used in [Enroll TPM device to IoT Hub Device Provisioning Service using Java service SDK](#) as follows:

1. Copy the *Common Name* of your X.509 client certificate to the clipboard. If you wish to use the *X.509 certificate generator* tool as shown in the [preceding sample code section](#), either enter a *Common Name* for your certificate, or use the default `microsoftiotcore`. Use this **Common Name** as the value for the `REGISTRATION_ID` variable.

```
// Use common name of your X.509 client certificate  
private static final String REGISTRATION_ID = "[RegistrationId]";
```

2. Rename the variable `TPM_ENDORSEMENT_KEY` as `PUBLIC_KEY_CERTIFICATE_STRING`. Copy your client certificate or the **Client Cert** from the output of the *X.509 certificate generator* tool, as the value for the `PUBLIC_KEY_CERTIFICATE_STRING` variable.

```
// Rename the variable *TPM_ENDORSEMENT_KEY* as *PUBLIC_KEY_CERTIFICATE_STRING*  
private static final String PUBLIC_KEY_CERTIFICATE_STRING =  
    "-----BEGIN CERTIFICATE-----\n" +  
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +  
    "-----END CERTIFICATE-----\n";
```

3. In the **main** function, replace the line `Attestation attestation = new TpmAttestation(TPM_ENDORSEMENT_KEY);` with the following to use the X.509 client certificate:

```
Attestation attestation = X509Attestation.createFromClientCertificates(PUBLIC_KEY_CERTIFICATE_STRING);
```

4. Save, build, and run the *individual enrollment* sample file, using the steps in the section [Build and run the sample code for individual enrollment](#).

## Clean up resources

If you plan to explore the Java service sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the Java sample output window on your machine.
2. Close the *X509 Cert Generator* window on your machine.
3. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select the **Enrollment Groups** tab. Select the check box next to the *GROUP NAME* for the X.509 devices you enrolled using this quickstart, and press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you enrolled a simulated group of X.509 devices to your Device Provisioning service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll X.509 devices to the Device Provisioning Service using C#

12/19/2019 • 6 minutes to read • [Edit Online](#)

This quickstart shows how to use C# to programmatically create an [Enrollment group](#) that uses intermediate or root CA X.509 certificates. The enrollment group is created by using the [Microsoft Azure IoT SDK for .NET](#) and a sample C# .NET Core application. An enrollment group controls access to the provisioning service for devices that share a common signing certificate in their certificate chain. To learn more, see [Controlling device access to the provisioning service with X.509 certificates](#). For more information about using X.509 certificate-based Public Key Infrastructure (PKI) with Azure IoT Hub and Device Provisioning Service, see [X.509 CA certificate security overview](#).

This quickstart expects you've already created an IoT hub and Device Provisioning Service instance. If you haven't already created these resources, complete the [Set up IoT Hub Device Provisioning Service with the Azure portal](#) quickstart before you continue with this article.

Although the steps in this article work on both Windows and Linux computers, this article uses a Windows development computer.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Install [Visual Studio 2019](#).
- Install [.NET Core SDK](#).
- Install [Git](#).

## Prepare test certificates

For this quickstart, you must have a .pem or a .cer file that contains the public portion of an intermediate or root CA X.509 certificate. This certificate must be uploaded to your provisioning service, and verified by the service.

The [Azure IoT C SDK](#) contains test tooling that can help you create an X.509 certificate chain, upload a root or intermediate certificate from that chain, and do proof-of-possession with the service to verify the certificate.

### Caution

Use certificates created with the SDK tooling for development testing only. Do not use these certificates in production. They contain hard-coded passwords, such as `1234`, that expire after 30 days. To learn about obtaining certificates suitable for production use, see [How to get an X.509 CA certificate](#) in the Azure IoT Hub documentation.

To use this test tooling to generate certificates, do the following steps:

1. Find the tag name for the [latest release](#) of the Azure IoT C SDK.
2. Open a command prompt or Git Bash shell, and change to a working folder on your machine. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

The test tooling is located in the `azure-iot-sdk-c/tools/CACertificates` of the repository you cloned.

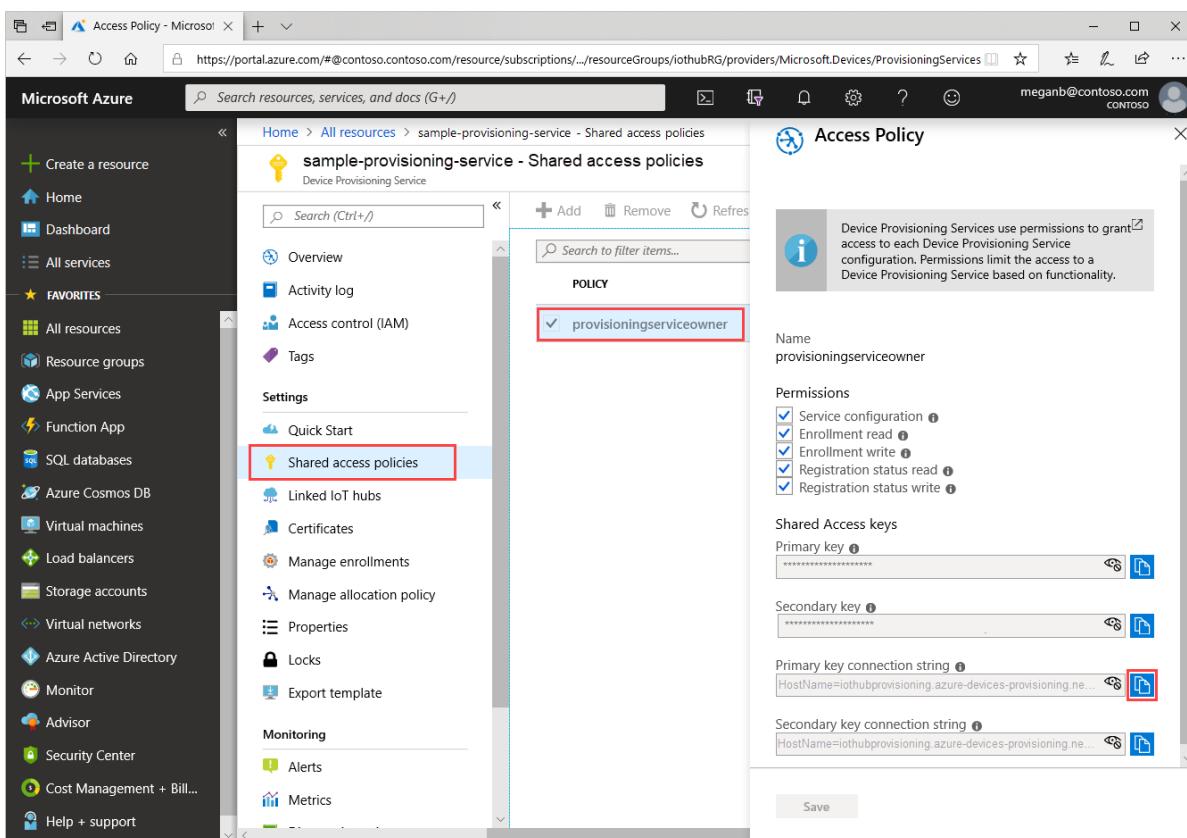
### 3. Follow the steps in [Managing test CA certificates for samples and tutorials](#).

In addition to the tooling in the C SDK, the [Group certificate verification sample](#) in the *Microsoft Azure IoT SDK for .NET* shows how to do proof-of-possession in C# with an existing X.509 intermediate or root CA certificate.

## Get the connection string for your provisioning service

For the sample in this quickstart, you need the connection string for your provisioning service.

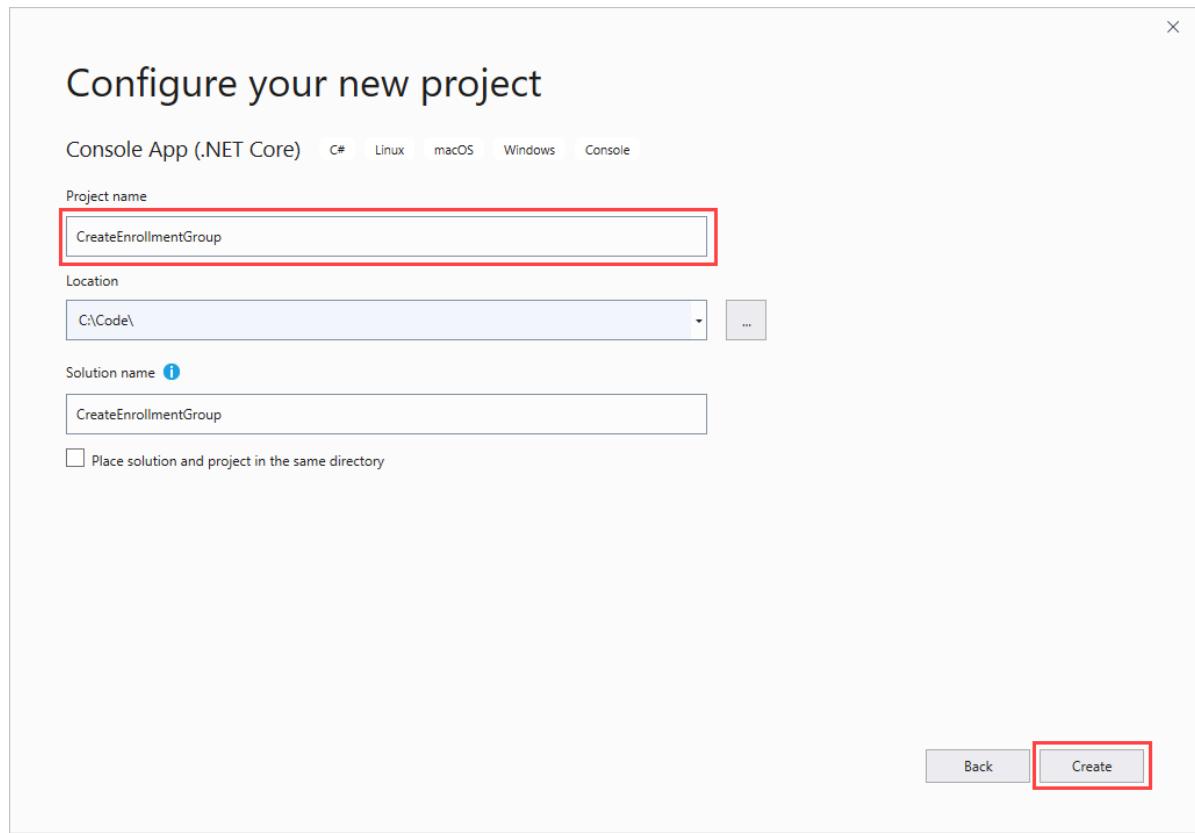
1. Sign in to the Azure portal, select **All resources**, and then your Device Provisioning Service.
2. Select **Shared access policies**, then choose the access policy you want to use to open its properties. In **Access Policy**, copy and save the primary key connection string.



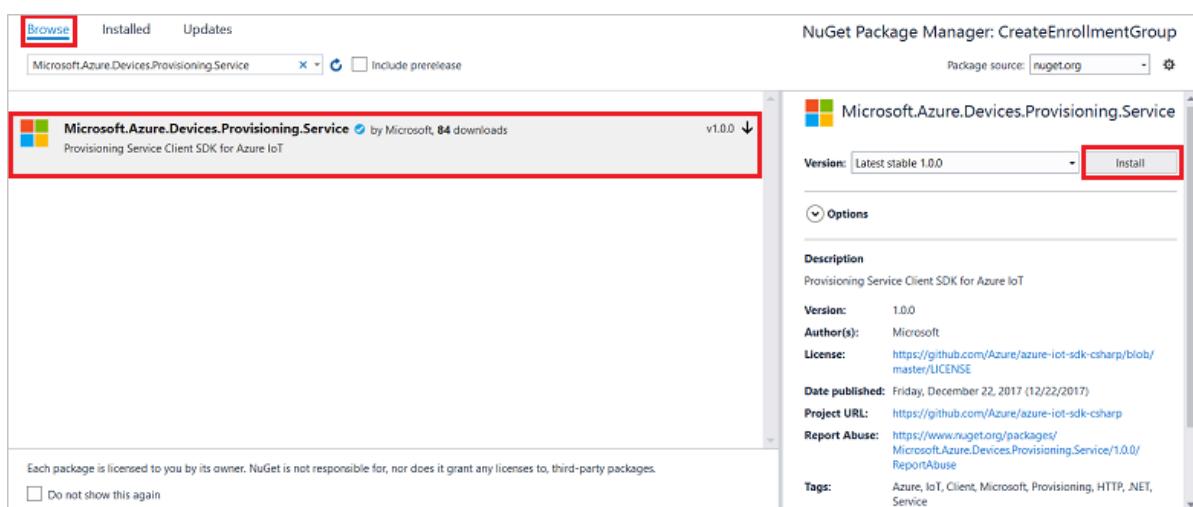
## Create the enrollment group sample

This section shows how to create a .NET Core console app that adds an enrollment group to your provisioning service. With some modification, you can also follow these steps to create a [Windows IoT Core](#) console app to add the enrollment group. To learn more about developing with IoT Core, see the [Windows IoT Core developer documentation](#).

1. Open Visual Studio and select **Create a new project**. In **Create a new project**, choose the **Console App (.NET Core)** for C# project template and select **Next**.
2. Name the project `CreateEnrollmentGroup`, and then press **Create**.



3. When the solution opens in Visual Studio, in the **Solution Explorer** pane, right-click the **CreateEnrollmentGroup** project, and then select **Manage NuGet Packages**.
4. In **NuGet Package Manager**, select **Browse**, search for and choose **Microsoft.Azure.Devices.Provisioning.Service**, and then press **Install**.



This step downloads, installs, and adds a reference to the [Azure IoT Provisioning Service Client SDK](#) NuGet package and its dependencies.

5. Add the following `using` statements after the other `using` statements at the top of `Program.cs`:

```
using System.Security.Cryptography.X509Certificates;
using System.Threading.Tasks;
using Microsoft.Azure.Devices.Provisioning.Service;
```

6. Add the following fields to the `Program` class, and make the listed changes.

```
private static string ProvisioningConnectionString = "{ProvisioningServiceConnectionString}";  
private static string EnrollmentGroupId = "enrollmentgroup1";  
private static string X509RootCertPath = @"{Path to a .cer or .pem file for a verified root CA or  
intermediate CA X.509 certificate}";
```

- Replace the `ProvisioningConnectionString` placeholder value with the connection string of the provisioning service that you want to create the enrollment for.
- Replace the `X509RootCertPath` placeholder value with the path to a .pem or .cer file. This file represents the public part of an intermediate or root CA X.509 certificate that has been previously uploaded and verified with your provisioning service.
- You may optionally change the `EnrollmentGroupId` value. The string can contain only lower case characters and hyphens.

#### **IMPORTANT**

In production code, be aware of the following security considerations:

- Hard-coding the connection string for the provisioning service administrator is against security best practices. Instead, the connection string should be held in a secure manner, such as in a secure configuration file or in the registry.
- Be sure to upload only the public part of the signing certificate. Never upload .pfx (PKCS12) or .pem files containing private keys to the provisioning service.

7. Add the following method to the `Program` class. This code creates an enrollment group entry and then calls the `CreateOrUpdateEnrollmentGroupAsync` method on `ProvisioningServiceClient` to add the enrollment group to the provisioning service.

```

public static async Task RunSample()
{
    Console.WriteLine("Starting sample...");

    using (ProvisioningServiceClient provisioningServiceClient =
        ProvisioningServiceClient.CreateFromConnectionString(ProvisioningConnectionString))
    {
        #region Create a new enrollmentGroup config
        Console.WriteLine("\nCreating a new enrollmentGroup...");
        var certificate = new X509Certificate2(X509RootCertPath);
        Attestation attestation = X509Attestation.CreateFromRootCertificates(certificate);
        EnrollmentGroup enrollmentGroup =
            new EnrollmentGroup(
                EnrollmentGroupId,
                attestation)
        {
            ProvisioningStatus = ProvisioningStatus.Enabled
        };
        Console.WriteLine(enrollmentGroup);
        #endregion

        #region Create the enrollmentGroup
        Console.WriteLine("\nAdding new enrollmentGroup...");
        EnrollmentGroup enrollmentGroupResult =
            await
provisioningServiceClient.CreateOrUpdateEnrollmentGroupAsync(enrollmentGroup).ConfigureAwait(false);
        Console.WriteLine("\nEnrollmentGroup created with success.");
        Console.WriteLine(enrollmentGroupResult);
        #endregion

    }
}

```

- Finally, replace the body of the `Main` method with the following lines:

```

RunSample().GetAwaiter().GetResult();
Console.WriteLine("\nHit <Enter> to exit ...");
Console.ReadLine();

```

- Build the solution.

## Run the enrollment group sample

Run the sample in Visual Studio to create the enrollment group. A Command Prompt window will appear and start showing confirmation messages. On successful creation, the Command Prompt window displays the properties of the new enrollment group.

You can verify that the enrollment group has been created. Go to the Device Provisioning Service summary, and select **Manage enrollments**, then select **Enrollment Groups**. You should see a new enrollment entry that corresponds to the registration ID you used in the sample.

The screenshot shows the Azure Device Provisioning Service portal. In the top left, there's a breadcrumb navigation: Home > All resources > sample-provisioning-service - Shared access policies. The main title is "sample-provisioning-service - Shared access policies" under "Device Provisioning Service". On the left sidebar, there are several navigation items: Overview, Activity log, Access control (IAM), Tags, Settings (with sub-options Quick Start, Shared access policies, Linked IoT hubs, Certificates), and Manage enrollments (which is highlighted with a red box). The main content area has a search bar and buttons for Add enrollment group, Add individual enrollment, Refresh, and Delete. A message box says, "You can add or remove individual device enrollments and/or enrollment groups from this page". Below this, there are tabs for "Enrollment Groups" (which is selected and highlighted with a red box) and "Individual Enrollments". There's also a "Filter enrollments" search bar. Under the "GROUP NAME" heading, there's a list with one item: "enrollmentgroupstest".

Select the entry to verify the certificate thumbprint and other properties for the entry.

## Clean up resources

If you plan to explore the C# service sample, don't clean up the resources created in this quickstart. Otherwise, use the following steps to delete all resources created by this quickstart.

1. Close the C# sample output window on your computer.
2. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select **Enrollment Groups**. Select the *Registration ID* for the enrollment entry you created using this quickstart and press **Delete**.
3. From your Device Provisioning service in the Azure portal, select **Certificates**, choose the certificate you uploaded for this quickstart, and press **Delete** at the top of **Certificate Details**.

## Next steps

In this quickstart, you created an enrollment group for an X.509 intermediate or root CA certificate using the Azure IoT Hub Device Provisioning Service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll X.509 devices to the Device Provisioning Service using Node.js

7/30/2020 • 4 minutes to read • [Edit Online](#)

In this quickstart, you use Node.js to programmatically create an enrollment group that uses intermediate or root CA X.509 certificates. The enrollment group is created using the IoT SDK for Node.js and a sample Node.js application.

## Prerequisites

- Completion of [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Node.js v4.0+](#). This quickstart installs the [IoT SDK for Node.js](#) below.
- [Git](#).
- [Azure IoT C SDK](#).

## Prepare test certificates

For this quickstart, you must have a .pem or a .cer file that contains the public portion of an intermediate or root CA X.509 certificate. This certificate must be uploaded to your provisioning service, and verified by the service.

For more information about using X.509 certificate-based Public Key Infrastructure (PKI) with Azure IoT Hub and Device Provisioning Service, see [X.509 CA certificate security overview](#).

The [Azure IoT C SDK](#) contains test tooling that can help you create an X.509 certificate chain, upload a root or intermediate certificate from that chain, and perform proof-of-possession with the service to verify the certificate. Certificates created with the SDK tooling are designed to use for **development testing only**. These certificates **must not be used in production**. They contain hard-coded passwords ("1234") that expire after 30 days. To learn about obtaining certificates suitable for production use, see [How to get an X.509 CA certificate](#) in the Azure IoT Hub documentation.

To use this test tooling to generate certificates, perform the following steps:

- Find the tag name for the [latest release](#) of the Azure IoT C SDK.
- Open a command prompt or Git Bash shell, and change to a working folder on your machine. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

The test tooling is located in the `azure-iot-sdk-c/tools/CACertificates` of the repository you cloned.

- Follow the steps in [Managing test CA certificates for samples and tutorials](#).

## Create the enrollment group sample

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual enrollments](#): Used to enroll a single device.

An enrollment group controls access to the provisioning service for devices that share a common signing certificate in their certificate chain. To learn more, see [Controlling device access to the provisioning service with X.509 certificates](#).

1. From a command window in your working folder, run:

```
npm install azure-iot-provisioning-service
```

2. Using a text editor, create a `create_enrollment_group.js` file in your working folder. Add the following code to the file and save:

```
'use strict';
var fs = require('fs');

var provisioningServiceClient = require('azure-iot-provisioning-service').ProvisioningServiceClient;

var serviceClient = provisioningServiceClient.fromConnectionString(process.argv[2]);

var enrollment = {
    enrollmentGroupId: 'first',
    attestation: {
        type: 'x509',
        x509: {
            signingCertificates: {
                primary: {
                    certificate: fs.readFileSync(process.argv[3], 'utf-8').toString()
                }
            }
        }
    },
    provisioningStatus: 'disabled'
};

serviceClient.createOrUpdateEnrollmentGroup(enrollment, function(err, enrollmentResponse) {
    if (err) {
        console.log('error creating the group enrollment: ' + err);
    } else {
        console.log("enrollment record returned: " + JSON.stringify(enrollmentResponse, null, 2));
        enrollmentResponse.provisioningStatus = 'enabled';
        serviceClient.createOrUpdateEnrollmentGroup(enrollmentResponse, function(err, enrollmentResponse) {
            if (err) {
                console.log('error updating the group enrollment: ' + err);
            } else {
                console.log("updated enrollment record returned: " + JSON.stringify(enrollmentResponse, null, 2));
            }
        });
    }
});
```

## Run the enrollment group sample

1. To run the sample, you need the connection string for your provisioning service.
  - a. Sign in to the Azure portal, select the **All resources** button on the left-hand menu and open your Device Provisioning service.

- b. Click **Shared access policies**, then select the access policy you want to use to open its properties. In the Access Policy window, copy and note down the primary key connection string.

**POLICY**

- provisioningserviceowner

Name  
provisioningserviceowner  
Permissions

- Service configuration
- Enrollment read
- Enrollment write
- Registration status read
- Registration status write

Shared Access keys

Primary key  
\*\*\*\*\*  
Secondary key  
\*\*\*\*\*

Primary key connection string  
HostName=sample-provisioning-service.azure-devices-provisioning.com  
Secondary key connection string  
HostName=sample-provisioning-service.azure-devices-provisioning.com

2. As stated in [Prepare test certificates](#), you also need a .pem file that contains an X.509 intermediate or root CA certificate that has been previously uploaded and verified with your provisioning service. To check that your certificate has been uploaded and verified, on the Device Provisioning Service summary page in the Azure portal, select **Certificates**. Find the certificate that you want to use for the group enrollment and ensure that its status value is *verified*.

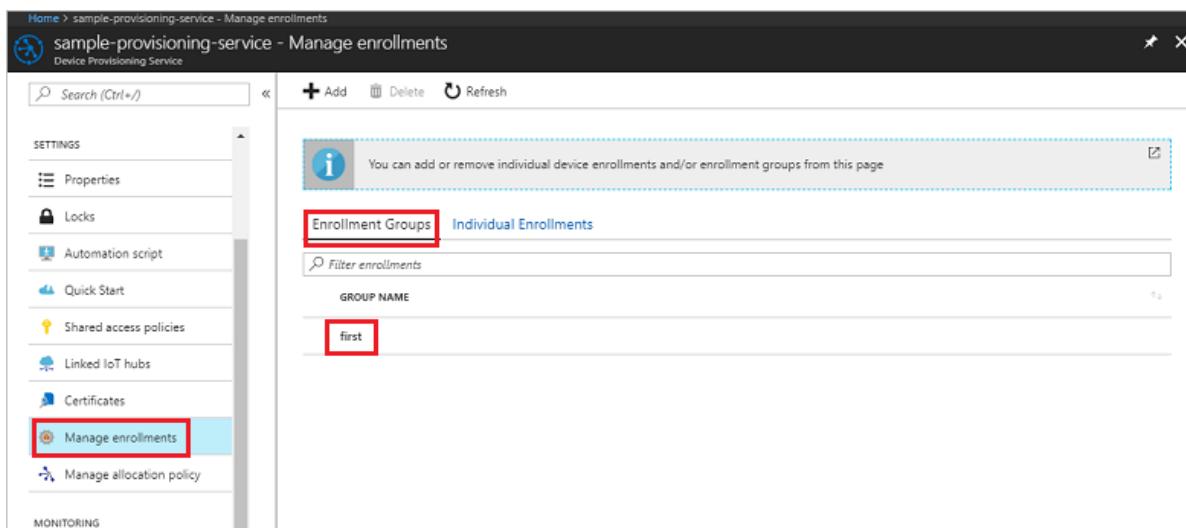
NAME	STATUS	EXPIRY	SUBJECT	THUMBPRINT	CREATED
TestRootCA	Verified	Fri Jan 12 2018 11:20:02 C	CN=Azure IoT CA TestO	*****	Wed Dec 13 2017 11:26:0

3. To create an [enrollment group](#) for your certificate, run the following command (include the quotes around the command arguments):

```
node create_enrollment_group.js "<the connection string for your provisioning service>" "<your certificate's .pem file>"
```

4. On successful creation, the command window displays the properties of the new enrollment group.

5. Verify that the enrollment group has been created. In the Azure portal, on the Device Provisioning Service summary blade, select **Manage enrollments**. Select the **Enrollment Groups** tab and verify that the new enrollment entry (*first*) is present.



# Clean up resources

If you plan to explore the Node.js service samples, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all Azure resources created by this quickstart.

1. Close the Node.js sample output window on your machine.
  2. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select the **Enrollment Groups** tab. Select the check box next to the *GROUP NAME* for the X.509 devices you enrolled using this quickstart, and press the **Delete** button at the top of the pane.
  3. From your Device Provisioning service in the Azure portal, select **Certificates**, select the certificate you

uploaded for this quickstart, and press the **Delete** button at the top of the **Certificate Details** window.

## Next steps

In this quickstart, you created a group enrollment for an X.509 intermediate or root CA certificate using the Azure IoT Hub Device Provisioning Service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

Also, see the [Node.js device provisioning sample](#).

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll X.509 devices to the Device Provisioning Service using Python

7/30/2020 • 4 minutes to read • [Edit Online](#)

In this quickstart, you use Python to programmatically create an enrollment group that uses intermediate or root CA X.509 certificates. An enrollment group controls access to the provisioning service for devices that share a common signing certificate in their certificate chain. The enrollment group is created using the Python Provisioning Service SDK and a sample Python application.

## Prerequisites

- Completion of [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Python 2.x or 3.x](#). Add Python to your platform-specific environment variables. This quickstart installs the [Python Provisioning Service SDK](#) below.
- [Pip](#), if not included with your distribution of Python.
- [Git](#).

### IMPORTANT

This article only applies to the deprecated V1 Python SDK. Device and service clients for the IoT Hub Device Provisioning Service are not yet available in V2. The team is currently hard at work to bring V2 to feature parity.

## Prepare test certificates

For this quickstart, you must have a .pem or a .cer file that contains the public portion of an intermediate or root CA X.509 certificate. This certificate must be uploaded to your provisioning service, and verified by the service.

For more information about using X.509 certificate-based Public Key Infrastructure (PKI) with Azure IoT Hub and Device Provisioning Service, see [X.509 CA certificate security overview](#).

The [Azure IoT C SDK](#) contains test tooling that can help you create an X.509 certificate chain, upload a root or intermediate certificate from that chain, and perform proof-of-possession with the service to verify the certificate. Certificates created with the SDK tooling are designed to use for **development testing only**. These certificates **must not be used in production**. They contain hard-coded passwords ("1234") that expire after 30 days. To learn about obtaining certificates suitable for production use, see [How to get an X.509 CA certificate](#) in the Azure IoT Hub documentation.

To use this test tooling to generate certificates, perform the following steps:

- Find the tag name for the [latest release](#) of the Azure IoT C SDK.
- Open a command prompt or Git Bash shell, and change to a working folder on your machine. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

The test tooling is located in the [azure-iot-sdk-c/tools/CACertificates](#) of the repository you cloned.

3. Follow the steps in [Managing test CA certificates for samples and tutorials](#).

## Modify the Python sample code

This section shows how to add the provisioning details of your X.509 device to the sample code.

1. Using a text editor, create a new `EnrollmentGroup.py` file.
2. Add the following `import` statements and variables at the start of the `EnrollmentGroup.py` file. Then replace `dpsConnectionString` with your connection string found under **Shared access policies** in your **Device Provisioning Service** on the [Azure portal](#). Replace the certificate placeholder with the certificate created previously in [Prepare test certificates](#). Finally, create a unique `registrationid` and be sure that it only consists of lower-case alphanumerics and hyphens.

```
from provisioningserviceclient import ProvisioningServiceClient
from provisioningserviceclient.models import EnrollmentGroup, AttestationMechanism

CONNECTION_STRING = "{dpsConnectionString}"

SIGNING_CERT = """----BEGIN CERTIFICATE----
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
GROUP_ID = "{registrationid}"
```

3. Add the following function and function call to implement the group enrollment creation:

```
def main():
    print ( "Initiating enrollment group creation..." )

    psc = ProvisioningServiceClient.create_from_connection_string(CONNECTION_STRING)
    att = AttestationMechanism.create_with_x509_signing_certs(SIGNING_CERT)
    eg = EnrollmentGroup.create(GROUP_ID, att)

    eg = psc.create_or_update(eg)

    print ( "Enrollment group created." )

if __name__ == '__main__':
    main()
```

4. Save and close the `EnrollmentGroup.py` file.

# Run the sample group enrollment

The Azure IoT Device Provisioning Service supports two types of enrollments:

- [Enrollment groups](#): Used to enroll multiple related devices.
- [Individual enrollments](#): Used to enroll a single device.

Creating Individual enrollments using the [Python Provisioning Service SDK](#) is still a work in progress. To learn more, see [Controlling device access to the provisioning service with X.509 certificates](#).

1. Open a command prompt, and run the following command to install the [azure-iot-provisioning-device-client](#).

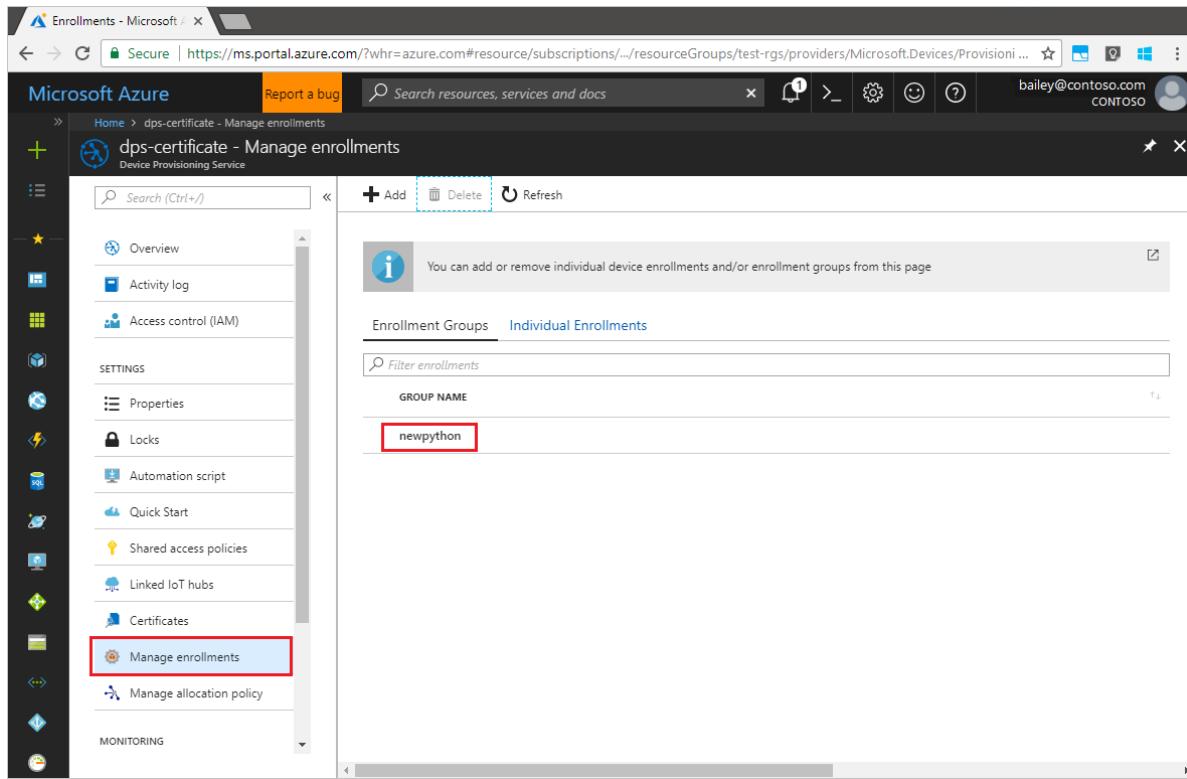
```
pip install azure-iothub-provisioningserviceclient
```

2. In the command prompt, and run the script.

```
python EnrollmentGroup.py
```

3. Observe the output for the successful enrollment.

4. Navigate to your provisioning service in the Azure portal. Click **Manage enrollments**. Notice that your group of X.509 devices appears under the **Enrollment Groups** tab, with the name `registrationid` created earlier.



## Clean up resources

If you plan to explore the Java service sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the Java sample output window on your machine.
2. Close the *X509 Cert Generator* window on your machine.
3. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select

the **Enrollment Groups** tab. Select the check box next to the *GROUP NAME* for the X.509 devices you enrolled using this quickstart, and press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you enrolled a simulated group of X.509 devices to your Device Provisioning service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll TPM device to IoT Hub Device Provisioning Service using Java Service SDK

7/30/2020 • 4 minutes to read • [Edit Online](#)

In this quickstart, you programmatically create an individual enrollment for a simulated TPM device in the Azure IoT Hub Device Provisioning Service using the Java Service SDK with the help of a sample Java application.

## Prerequisites

- Completion of [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).
- Completion of [Read cryptographic keys from the TPM device](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Java SE Development Kit 8](#). This quickstart installs the [Java Service SDK](#) below. It works on both Windows and Linux. This quickstart uses Windows.
- [Maven 3](#).
- [Git](#).

## Prepare the development environment

- Make sure you have [Java SE Development Kit 8](#) installed on your machine.
- Set up environment variables for your Java installation. The `PATH` variable should include the full path to `jdk1.8.x\bin` directory. If this is your machine's first Java installation, then create a new environment variable named `JAVA_HOME` and point it to the full path to the `jdk1.8.x` directory. On Windows machine, this directory is found in the `C:\Program Files\Java` folder, and you can create or edit environment variables by searching for [Edit the system environment variables](#) on the [Control panel](#) of your Windows machine.

You may check if Java is successfully set up on your machine by running the following command on your command window:

```
java -version
```

- Download and extract [Maven 3](#) on your machine.
- Edit environment variable `PATH` to point to the `apache-maven-3.x.x\bin` folder inside the folder where Maven is extracted. You may confirm that Maven is successfully installed by running this command on your command window:

```
mvn --version
```

- Make sure [git](#) is installed on your machine and is added to the environment variable `PATH`.

## Download and modify the Java sample code

This section shows how to add the provisioning details of your TPM device to the sample code.

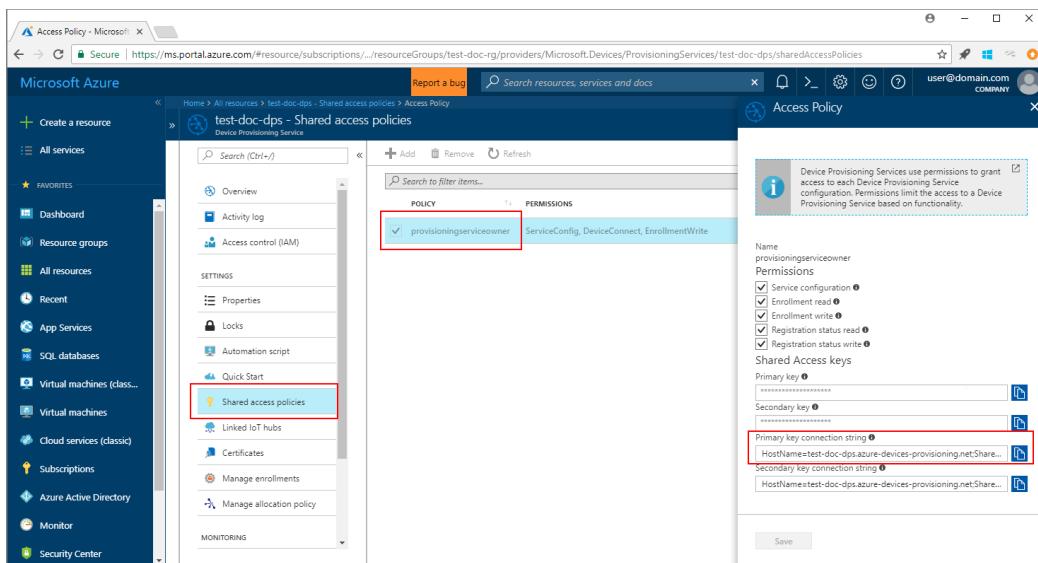
- Open a command prompt. Clone the GitHub repo for device enrollment code sample using the [Java Service SDK](#):

```
git clone https://github.com/Azure/azure-iot-sdk-java.git --recursive
```

2. In the downloaded source code, navigate to the sample folder **azure-iot-sdk-java/provisioning/provisioning-samples/service-enrollment-sample**. Open the file **/src/main/java/samples/com/microsoft/azure/sdk/iot/ServiceEnrollmentSample.java** in an editor of your choice, and add the following details:

- a. Add the [Provisioning Connection String] for your provisioning service, from the portal as following:

- Navigate to your provisioning service in the [Azure portal](#).
- Open the **Shared access policies**, and select a policy that has the *EnrollmentWrite* permission.
- Copy the **Primary key connection string**.



- d. In the sample code file **ServiceEnrollmentSample.java**, replace the [Provisioning Connection String] with the Primary key connection string.

```
private static final String PROVISIONING_CONNECTION_STRING = "[Provisioning Connection String]";
```

- b. Add the TPM device details:

- Get the *Registration ID* and the *TPM endorsement key* for a TPM device simulation, by following the steps leading to the section [Simulate TPM device](#).
- Use the *Registration ID* and the *Endorsement Key* from the output of the preceding step, to replace the [RegistrationId] and [TPM Endorsement Key] in the sample code file **ServiceEnrollmentSample.java**:

```
private static final String REGISTRATION_ID = "[RegistrationId]";
private static final String TPM_ENDORSEMENT_KEY = "[TPM Endorsement Key]";
```

- c. Optionally, you may configure your provisioning service through the sample code:

- To add this configuration to the sample, follow these steps:
  - Navigate to the IoT hub linked to your provisioning service in the [Azure portal](#). Open the

**Overview** tab for the hub, and copy the **Hostname**. Assign this **Hostname** to the **IOTHUB\_HOST\_NAME** parameter.

```
private static final String IOTHUB_HOST_NAME = "[Host name].azure-devices.net";
```

- b. Assign a friendly name to the **DEVICE\_ID** parameter, and keep the **PROVISIONING\_STATUS** as the default **ENABLED** value.
- OR, if you choose not to configure your provisioning service, make sure to comment out or delete the following statements in the *ServiceEnrollmentSample.java* file:

```
// The following parameters are optional. Remove it if you don't need.  
individualEnrollment.setDeviceId(DEVICE_ID);  
individualEnrollment.setIotHubHostName(IOTHUB_HOST_NAME);  
individualEnrollment.setProvisioningStatus(PROVISIONING_STATUS);
```

- d. Study the sample code. It creates, updates, queries, and deletes an individual TPM device enrollment. To verify successful enrollment in portal, temporarily comment out the following lines of code at the end of the *ServiceEnrollmentSample.java* file:

```
// **** Delete info of individualEnrollment  
*****  
System.out.println("\nDelete the individualEnrollment...");  
provisioningServiceClient.deleteIndividualEnrollment(REGISTRATION_ID);
```

- e. Save the file *ServiceEnrollmentSample.java*.

## Build and run the Java sample code

1. Open a command window, and navigate to the folder *azure-iot-sdk-java/provisioning/provisioning-samples/service-enrollment-sample*.
2. Build the sample code by using this command:

```
mvn install -DskipTests
```

This command downloads the Maven package [com.microsoft.azure.sdk.iot.provisioning.service](#) to your machine. This package includes the binaries for the **Java Service SDK**, that the sample code needs to build.

3. Run the sample by using these commands at the command window:

```
cd target  
java -jar ./service-enrollment-sample-{version}-with-deps.jar
```

4. Observe the output window for successful enrollment.
5. Navigate to your provisioning service in the Azure portal. Select **Manage enrollments**, and select the **Individual Enrollments** tab. Notice that the *Registration ID* of your simulated TPM device is now listed.

The screenshot shows the Azure Device Provisioning Service interface. On the left, there's a sidebar with various management options like Overview, Activity log, Access control (IAM), Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, and Manage enrollments. The 'Manage enrollments' option is highlighted with a red box. The main pane has tabs for 'Enrollment Groups' and 'Individual Enrollments', with 'Individual Enrollments' also highlighted with a red box. Below these tabs is a 'Filter enrollments' search bar. Underneath is a 'REGISTRATION ID' section containing a registration ID entry, which is also highlighted with a red box.

## Clean up resources

If you plan to explore the Java service sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the Java sample output window on your machine.
2. Close the TPM simulator window that you may have created to simulate your TPM device.
3. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select the **Individual Enrollments** tab. Select the check box next to the *Registration ID* for the enrollment entry you created using this quickstart, and press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you enrolled a simulated TPM device to your Device Provisioning service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll TPM device to IoT Hub Device Provisioning Service using C# service SDK

12/10/2019 • 5 minutes to read • [Edit Online](#)

This article shows how to programmatically create an individual enrollment for a TPM device in the Azure IoT Hub Device Provisioning Service by using the [C# Service SDK](#) and a sample C# .NET Core application. You can optionally enroll a simulated TPM device to the provisioning service by using this individual enrollment entry. Although these steps work on both Windows and Linux computers, this article uses a Windows development computer.

## Prepare the development environment

1. Verify you have [Visual Studio 2019](#) installed on your computer.
2. Verify you have the [.NET Core SDK](#) installed on your computer.
3. Complete the steps in [Set up the IoT Hub Device Provisioning Service with the Azure portal](#) before you continue.
4. (Optional) If you want to enroll a simulated device at the end of this quickstart, follow the procedure in [Create and provision a simulated TPM device using C# device SDK](#) up to the step where you get an endorsement key for the device. Save the endorsement key, registration ID, and, optionally, the device ID, because you need to use them later in this quickstart.

### NOTE

Don't follow the steps to create an individual enrollment by using the Azure portal.

## Get the connection string for your provisioning service

For the sample in this quickstart, you need the connection string for your provisioning service.

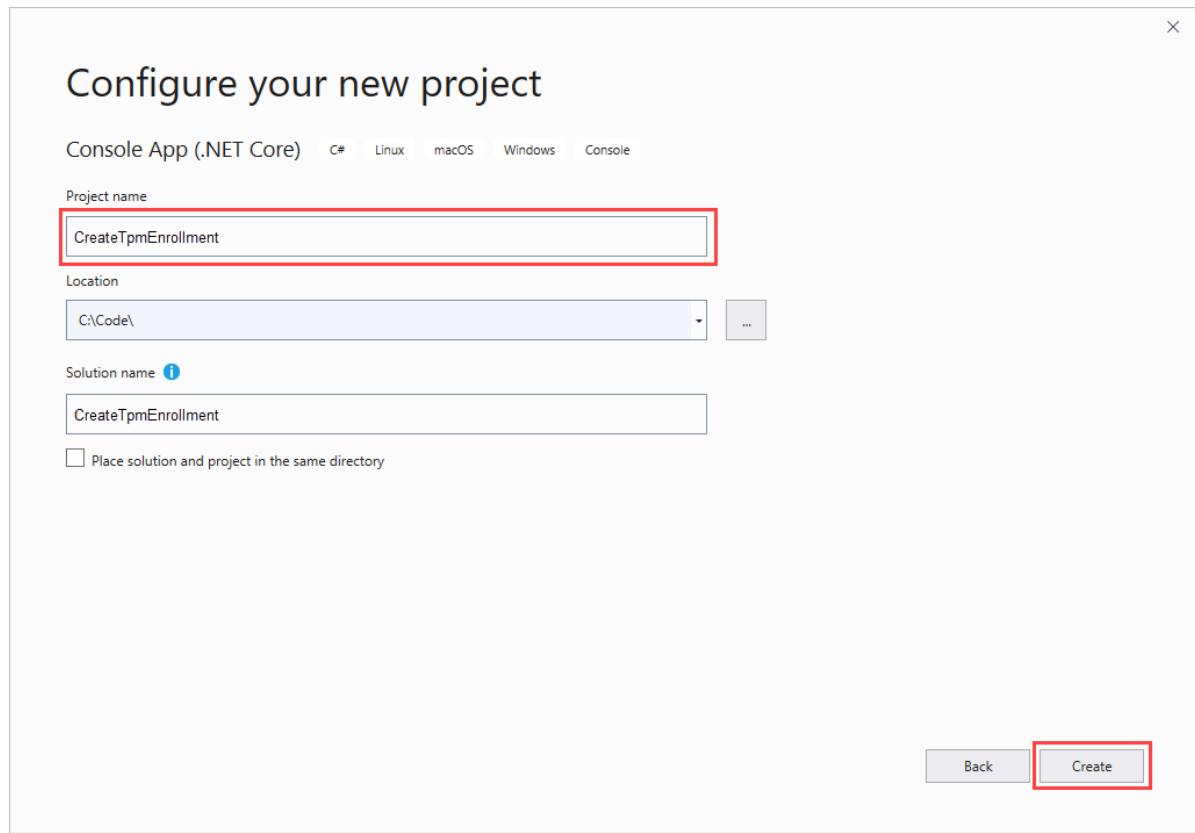
1. Sign in to the Azure portal, select **All resources**, and then your Device Provisioning Service.
2. Choose **Shared access policies**, then select the access policy you want to use to open its properties. In **Access Policy**, copy and save the primary key connection string.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu includes 'Create a resource', 'Home', 'Dashboard', 'All services', 'FAVORITES' (with 'All resources', 'Resource groups', 'App Services', 'Function App', 'SQL databases', 'Azure Cosmos DB', 'Virtual machines', 'Load balancers', 'Storage accounts', 'Virtual networks', 'Azure Active Directory', 'Monitor', 'Advisor', 'Security Center', 'Cost Management + Bill...', and 'Help + support'). The main content area shows 'sample-provisioning-service - Shared access policies' under 'Device Provisioning Service'. The 'Settings' section contains 'Quick Start', 'Shared access policies' (which is selected and highlighted with a red box), 'Linked IoT hubs', 'Certificates', 'Manage enrollments', 'Manage allocation policy', 'Properties', 'Locks', and 'Export template'. The 'Monitoring' section includes 'Alerts' and 'Metrics'. On the right, the 'Access Policy' blade is open, showing a list of policies with 'provisioningserviceowner' selected (highlighted with a red box). The blade also displays information about Device Provisioning Services using permissions to grant access, the policy name 'provisioningserviceowner', and a list of permissions (Service configuration, Enrollment read, Enrollment write, Registration status read, Registration status write) all checked. It also shows 'Shared Access keys' for Primary Key and Secondary key, and 'Key connection strings' for Primary key connection string and Secondary key connection string, both starting with 'HostName=iothubprovisioning.azure-devices-provisioning.ne...'. A 'Save' button is at the bottom.

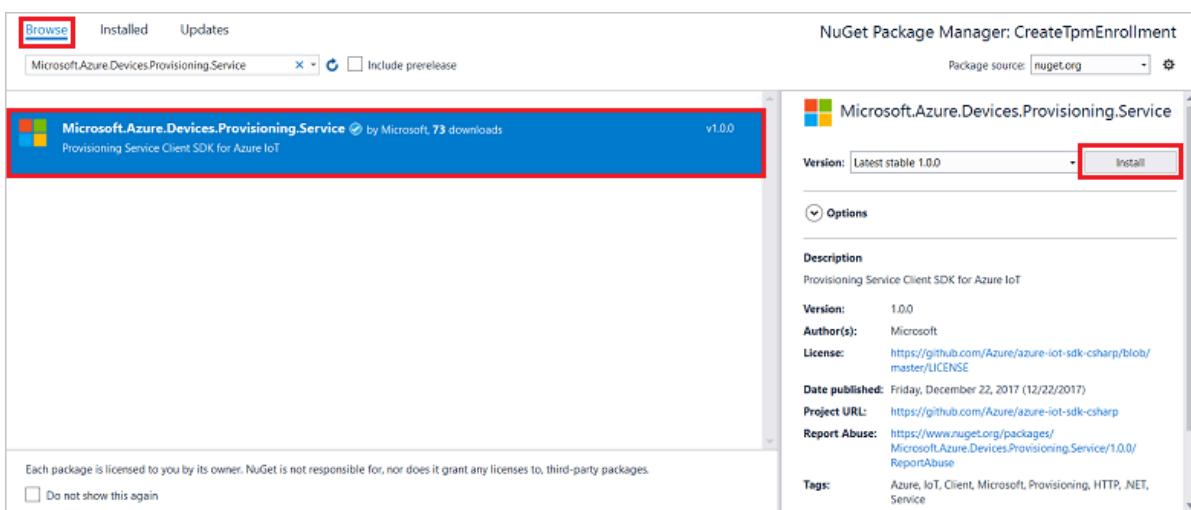
## Create the individual enrollment sample

This section shows how to create a .NET Core console app that adds an individual enrollment for a TPM device to your provisioning service. With some modification, you can also follow these steps to create a [Windows IoT Core](#) console app to add the individual enrollment. To learn more about developing with IoT Core, see [Windows IoT Core developer documentation](#).

1. Open Visual Studio and select **Create a new project**. In **Create a new project**, choose the **Console App (.NET Core)** project template for C# and select **Next**.
2. Name the project *CreateTpmEnrollment*, and press **Create**.



3. When the solution opens in Visual Studio, in the **Solution Explorer** pane, right-click the **CreateTpmEnrollment** project. Select **Manage NuGet Packages**.
4. In **NuGet Package Manager**, select **Browse**, search for and choose **Microsoft.Azure.Devices.Provisioning.Service**, and then press **Install**.



This step downloads, installs, and adds a reference to the [Azure IoT Provisioning Service Client SDK NuGet package](#) and its dependencies.

5. Add the following `using` statements after the other `using` statements at the top of `Program.cs`:

```
using System.Threading.Tasks;
using Microsoft.Azure.Devices.Provisioning.Service;
```

6. Add the following fields to the `Program` class, making the changes listed below.

```

private static string ProvisioningConnectionString = "{ProvisioningServiceConnectionString}";
private const string RegistrationId = "sample-registrationid-csharp";
private const string TpmEndorsementKey =
    "AToAAQALAAMASgg3GXZ0SEs/gakMyNRqXXJP1S124GUGtk8qHaGzMuaaoABgCAAEMAEAgAAAAAAEAsxj2gUS" +
    "cTk1UjuioeTlfGYZrrimExB+bSch75adUMRIi2UOMxG1kw4y+9RW/IVoMl4e620VxZad0ARX2gUqVjY07KPVt3d" +
    "yKhZS3dkcvfBisBhP1XH9B33VqHG9SHnbnQXdBuAxCgKAfxome8UmBKfe+naTsE5fkvb/do3/dD614sGBwFCnKR" +
    "dln4XpM03zLpoHFao8z0wt81/uP3qUIxmCYv9A7m69Ms+5/pCkTu/rK4mRDsfhZ0QLfbzVI6zQFOKF/rwsfBtFe" +
    "WlWtcuJMK1Xd8TXWE1Tzgh7JS4qhFzreLoc1mI0Gcj+Aws0usZh7dLIVPnlgZCBhgy1SSDQM==";

// Optional parameters
private const string OptionalDeviceId = "myCSharpDevice";
private const ProvisioningStatus OptionalProvisioningStatus = ProvisioningStatus.Enabled;

```

- Replace the `ProvisioningServiceConnectionString` placeholder value with the connection string of the provisioning service that you want to create the enrollment for.
- You may optionally change the registration ID, endorsement key, device ID, and provisioning status.
- If you're using this quickstart together with the [Create and provision a simulated TPM device using C# device SDK](#) quickstart to provision a simulated device, replace the endorsement key and registration ID with the values that you noted in that quickstart. You can replace the device ID with the value suggested in that quickstart, use your own value, or use the default value in this sample.

7. Add the following method to the `Program` class. This code creates individual enrollment entry and then calls the `CreateOrUpdateIndividualEnrollmentAsync` method on the `ProvisioningServiceClient` to add the individual enrollment to the provisioning service.

```

public static async Task RunSample()
{
    Console.WriteLine("Starting sample...");

    using (ProvisioningServiceClient provisioningServiceClient =
        ProvisioningServiceClient.CreateFromConnectionString(ProvisioningConnectionString))
    {
        #region Create a new individualEnrollment config
        Console.WriteLine("\nCreating a new individualEnrollment...");
        Attestation attestation = new TpmAttestation(TpmEndorsementKey);
        IndividualEnrollment individualEnrollment =
            new IndividualEnrollment(
                RegistrationId,
                attestation);

        // The following parameters are optional. Remove them if you don't need them.
        individualEnrollment.DeviceId = OptionalDeviceId;
        individualEnrollment.ProvisioningStatus = OptionalProvisioningStatus;
        #endregion

        #region Create the individualEnrollment
        Console.WriteLine("\nAdding new individualEnrollment...");
        IndividualEnrollment individualEnrollmentResult =
            await
provisioningServiceClient.CreateOrUpdateIndividualEnrollmentAsync(individualEnrollment).ConfigureAwait(
false);
        Console.WriteLine("\nIndividualEnrollment created with success.");
        Console.WriteLine(individualEnrollmentResult);
        #endregion

    }
}

```

8. Finally, replace the body of the `Main` method with the following lines:

```
RunSample().GetAwaiter().GetResult();
Console.WriteLine("\nHit <Enter> to exit ...");
Console.ReadLine();
```

9. Build the solution.

## Run the individual enrollment sample

Run the sample in Visual Studio to create the individual enrollment for your TPM device.

A Command Prompt window will appear and start showing confirmation messages. On successful creation, the Command Prompt window displays the properties of the new individual enrollment.

You can verify that the individual enrollment has been created. Go to the Device Provisioning Service summary, and select **Manage enrollments**, then select **Individual Enrollments**. You should see a new enrollment entry that corresponds to the registration ID you used in the sample.

The screenshot shows the Azure portal interface for the Device Provisioning Service. The left sidebar has a 'Manage enrollments' option highlighted with a red box. The main content area shows a list of individual enrollments. One entry, 'sample-registrationid-csharp', is selected and has a checked checkbox next to it. The 'Individual Enrollments' tab is also highlighted with a red box.

Select the entry to verify the endorsement key and other properties for the entry.

If you've been following the steps in the [Create and provision a simulated TPM device using C# device SDK](#) quickstart, you can continue with the remaining steps in that quickstart to enroll your simulated device. Be sure to skip the steps to create an individual enrollment using the Azure portal.

## Clean up resources

If you plan to explore the C# service sample, don't clean up the resources created in this quickstart. Otherwise, use the following steps to delete all resources created by this quickstart.

1. Close the C# sample output window on your computer.
2. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select the **Individual Enrollments** tab. Select the check box next to the *Registration ID* for the enrollment entry you created using this quickstart, and press the **Delete** button at the top of the pane.
3. If you followed the steps in [Create and provision a simulated TPM device using C# device SDK](#) to create a simulated TPM device, do the following steps:

- a. Close the TPM simulator window and the sample output window for the simulated device.
- b. In the Azure portal, navigate to the IoT Hub where your device was provisioned. In the menu under **Explorers**, select **IoT devices**, select the check box next to the *DEVICE ID* of the device you registered in this quickstart, and then press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've programmatically created an individual enrollment entry for a TPM device. Optionally, you created a TPM simulated device on your computer and provisioned it to your IoT hub using the Azure IoT Hub Device Provisioning Service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll TPM device to IoT Hub Device Provisioning Service using Node.js service SDK

7/30/2020 • 3 minutes to read • [Edit Online](#)

In this quickstart, you programmatically create an individual enrollment for a TPM device in the Azure IoT Hub Device Provisioning Service using the Node.js Service SDK and a sample Node.js application. You can optionally enroll a simulated TPM device to the provisioning service using this individual enrollment entry.

## Prerequisites

- Completion of [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Node.js v4.0+](#). This quickstart installs the [Node.js Service SDK](#) below.
- Endorsement key (optional). Follow the steps in [Create and provision a simulated device](#) until you get the key. Do not create an individual enrollment using the Azure portal.

## Create the individual enrollment sample

- From a command window in your working folder, run:

```
npm install azure-iot-provisioning-service
```

- Using a text editor, create a `create_individual_enrollment.js` file in your working folder. Add the following code to the file and save:

```
'use strict';

var provisioningServiceClient = require('azure-iot-provisioning-service').ProvisioningServiceClient;

var serviceClient = provisioningServiceClient.fromConnectionString(process.argv[2]);
var endorsementKey = process.argv[3];

var enrollment = {
    registrationId: 'first',
    attestation: {
        type: 'tpm',
        tpm: {
            endorsementKey: endorsementKey
        }
    }
};

serviceClient.createOrUpdateIndividualEnrollment(enrollment, function(err, enrollmentResponse) {
    if (err) {
        console.log('error creating the individual enrollment: ' + err);
    } else {
        console.log("enrollment record returned: " + JSON.stringify(enrollmentResponse, null, 2));
    }
});
```

## Run the individual enrollment sample

- To run the sample, you need the connection string for your provisioning service.
  - Sign in to the Azure portal, select the All resources button on the left-hand menu and open your Device Provisioning service.
  - Select Shared access policies, then select the access policy you want to use to open its properties. In the Access Policy window, copy and note down the primary key connection string.

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation pane is visible with various icons and sections like Overview, Activity log, and Shared access policies. The 'Shared access policies' section is highlighted with a red box. The main content area is titled 'sample-provisioning-service - Shared access policies' and shows a single policy named 'provisioningserviceowner'. This policy has several permissions checked under 'Permissions'. Below the policy details, there are sections for 'Shared Access keys' and 'Primary key connection string'. The 'Primary key connection string' field contains the value 'HostName=sample-provisioning-service.azure-devices-provisioning.com;'. A blue box highlights this connection string field.

- You also need the endorsement key for your device. If you have followed the [Create and provision a simulated device](#) quickstart to create a simulated TPM device, use the key created for that device. Otherwise, to create a sample individual enrollment, you can use the following endorsement key supplied with the [Node.js Service SDK](#):

```
AToAAQALAAASgAgg3GXZ0SEs/gakMyNRqXXJP1S124GUgtk8qHaGzMuaaoABgCAAEMAEAgAAAAAAEAsxj2gUScTk1UjuioeTlfGYZ
rrimExB+bScH75adUMRIi2UOMxG1kw4y+9RW/IVoM14e620VxZad0ARX2gUqvjY07KPVt3dyKhZS3dkcvfBisBhP1XH9B33VqHG9SHn
bnQXdBUaCgKAfxome8UmBKfe+naTsE5fkvjb/do3/dD614sGBwFCnKRd1n4XpM03zLpoHFao8z0wt81/uP3qUIxmCYv9A7m69Ms+5/p
CkTu/rK4mRDsfhZ0QLfbzVI6zQFOKF/rwsfBtFeWlWtcuJMK1XdD8TXWE1Tzgh7JS4qhFzreL0c1mI0GCj+Aws0usZh7dLIVPnlgZcB
hgy1SSDQM==
```

- To create an individual enrollment for your TPM device, run the following command (include the quotes around the command arguments):

```
node create_individual_enrollment.js "<the connection string for your provisioning service>" "<endorsement key>"
```

- On successful creation, the command window displays the properties of the new individual enrollment.

```

enrollment record returned: {
  "registrationId": "first",
  "attestation": {
    "type": "tpm",
    "tpm": {
      "endorsementKey": "-----"
    }
  },
  "etag": "\"96006773-0000-0000-0000-5a4fcbe70000\"",
  "createdDateTimeUtc": "2018-01-05T19:03:03.6347982Z",
  "lastUpdatedDateTimeUtc": "2018-01-05T19:03:03.6347982Z"
}

C:\code\NodeTpm3Enrollment>

```

- Verify that an individual enrollment has been created. In the Azure portal, on the Device Provisioning Service summary blade, select **Manage enrollments**. Select the **Individual Enrollments** tab and select the new enrollment entry (*first*) to verify the endorsement key and other properties for the entry.

The screenshot shows the Azure portal interface for managing device enrollments. On the left, there's a navigation sidebar with various service links. The main area is titled 'sample-provisioning-service - Manage enrollments'. Under the 'Individual Enrollments' tab, a list shows a single entry named 'first' with a checked checkbox next to it. The right side of the screen displays detailed configuration settings for this enrollment, including fields for IoT Hub assignment and initial device twin state.

Now that you've created an individual enrollment for a TPM device, if you want to enroll a simulated device, you can continue with the remaining steps in [Create and provision a simulated device](#). Be sure to skip the steps to create an individual enrollment using the Azure portal in that quickstart.

## Clean up resources

If you plan to explore the Node.js service samples, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

- Close the Node.js sample output window on your machine.
- If you created a simulated TPM device, close the TPM simulator window.
- Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select the **Individual Enrollments** tab. Select the check box next to the *Registration ID* for the enrollment entry you created using this quickstart, and press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've programmatically created an individual enrollment entry for a TPM device, and, optionally, created a TPM simulated device on your machine and provisioned it to your IoT hub using the Azure IoT

Hub Device Provisioning Service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Quickstart: Enroll TPM device to IoT Hub Device Provisioning Service using Python provisioning service SDK

7/30/2020 • 3 minutes to read • [Edit Online](#)

In this quickstart, you programmatically create an individual enrollment for a TPM device in the Azure IoT Hub Device Provisioning Service, using the Python Provisioning Service SDK with the help of a sample Python application.

## Prerequisites

- Completion of [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).
- An Azure account with an active subscription. [Create one for free](#).
- [Python 2.x or 3.x](#). This quickstart installs the [Python Provisioning Service SDK](#) below.
- [Pip](#), if not included with your distribution of Python.
- Endorsement key. Use the steps in [Create and provision a simulated device](#) or use the endorsement key supplied with the SDK, described below.

### IMPORTANT

This article only applies to the deprecated V1 Python SDK. Device and service clients for the IoT Hub Device Provisioning Service are not yet available in V2. The team is currently hard at work to bring V2 to feature parity.

## Prepare the environment

- Download and install [Python 2.x or 3.x](#). Make sure to use the 32-bit or 64-bit installation as required by your setup. When prompted during the installation, make sure to add Python to your platform-specific environment variables.
- For the [Python Provisioning Service SDK](#), choose one of the following options:
  - Build and compile the [Azure IoT Python SDK](#). Follow [these instructions](#) to build the Python packages. If you are using Windows OS, then also install [Visual C++ redistributable package](#) to allow the use of native DLLs from Python.
  - [Install or upgrade pip](#), the Python package management system and install the package via the following command:

```
pip install azure-iothub-provisioningserviceclient
```

- You need the endorsement key for your device. If you have followed the [Create and provision a simulated device](#) quickstart to create a simulated TPM device, use the key created for that device. Otherwise, you can use the following endorsement key supplied with the SDK:

```
AToAAQALAAAsgAgg3GXZ0SEs/gakMyNRqXXJP1S124GUgtk8qHaGzMuaaoAbgCAAEMAEAgAAAAAAEAtW6MOyCu/Nih47atIIoZt1Y  
khLeCTiSrtRN3q6hqg011A979No4B0cDWF900yzJvjQknMfxS/Dx/IJIBnOrgCg1YX/j4EEt07Ase29Xd63HjvG8M94+u2XINu79rkT  
xeueqW7gPerZQPn1xYmqawYcyzJS6GKWKdoIdS+UWu6bJr58V3xwv0QI4NiBXKD7htvz07jLItWTFhsWnTdzbJ7PnmfCa2vbRH/9pZ  
Iow+CcAL9mNTNNN4FdzYwapNVO+6SY/W4XU0Q+dLMCKYarqVNH5GzAWDFKT8nKzg69yQejJM8oeUWag/8odWOfbszA+iFjw3wVNrA5n  
8grUieRkPQ==
```

## Modify the Python sample code

This section shows how to add the provisioning details of your TPM device to the sample code.

1. Using a text editor, create a new **TpmEnrollment.py** file.
2. Add the following `import` statements and variables at the start of the **TpmEnrollment.py** file. Then replace `dpsConnectionString` with your connection string found under **Shared access policies** in your **Device Provisioning Service** on the [Azure portal](#). Replace `endorsementKey` with the value noted previously in [Prepare the environment](#). Finally, create a unique `registrationid` and be sure that it only consists of lower-case alphanumerics and hyphens.

```
from provisioningserviceclient import ProvisioningServiceClient  
from provisioningserviceclient.models import IndividualEnrollment, AttestationMechanism  
  
CONNECTION_STRING = "{dpsConnectionString}"  
  
ENDORSEMENT_KEY = "{endorsementKey}"  
  
REGISTRATION_ID = "{registrationid}"
```

3. Add the following function and function call to implement the group enrollment creation:

```
def main():  
    print ( "Starting individual enrollment..." )  
  
    psc = ProvisioningServiceClient.create_from_connection_string(CONNECTION_STRING)  
  
    att = AttestationMechanism.create_with_tpm(ENDORSEMENT_KEY)  
    ie = IndividualEnrollment.create(REGISTRATION_ID, att)  
  
    ie = psc.create_or_update(ie)  
  
    print ( "Individual enrollment successful." )  
  
if __name__ == '__main__':  
    main()
```

4. Save and close the **TpmEnrollment.py** file.

## Run the sample TPM enrollment

1. Open a command prompt, and run the script.

```
python TpmEnrollment.py
```

2. Observe the output for the successful enrollment.
3. Navigate to your provisioning service in the Azure portal. Select **Manage enrollments**. Notice that your TPM device appears under the **Individual Enrollments** tab, with the name `registrationid` created earlier.

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a tree view with 'dps-certificate' selected. Under 'dps-certificate', 'Manage enrollments' is highlighted with a red box. The main content area shows the 'dps-certificate - Manage enrollments' page for the 'Device Provisioning Service'. At the top, there are buttons for '+ Add', 'Delete', and 'Refresh'. Below this is a message: 'You can add or remove individual device enrollments and/or enrollment groups from this page'. There are two tabs: 'Enrollment Groups' and 'Individual Enrollments', with 'Individual Enrollments' highlighted with a red box. A search bar labeled 'Filter enrollments' is present. The 'REGISTRATION ID' section contains a table with one row, where the value 'pythontpm' is also highlighted with a red box.

## Clean up resources

If you plan to explore the Java service sample, do not clean up the resources created in this quickstart. If you do not plan to continue, use the following steps to delete all resources created by this quickstart.

1. Close the Python sample output window on your machine.
2. If you created a simulated TPM device, close the TPM simulator window.
3. Navigate to your Device Provisioning service in the Azure portal, select **Manage enrollments**, and then select the **Individual Enrollments** tab. Select the check box next to the *Registration ID* for the enrollment entry you created using this quickstart, and press the **Delete** button at the top of the pane.

## Next steps

In this quickstart, you've programmatically created an individual enrollment entry for a TPM device, and, optionally, created a TPM simulated device on your machine and provisioned it to your IoT hub using the Azure IoT Hub Device Provisioning Service. To learn about device provisioning in depth, continue to the tutorial for the Device Provisioning Service setup in the Azure portal.

[Azure IoT Hub Device Provisioning Service tutorials](#)

# Tutorial: Configure cloud resources for device provisioning with the IoT Hub Device Provisioning Service

12/10/2019 • 7 minutes to read • [Edit Online](#)

This tutorial shows how to set up the cloud for automatic device provisioning using the IoT Hub Device Provisioning Service. In this tutorial, you learn how to:

- Use the Azure portal to create an IoT Hub Device Provisioning Service and get the ID scope
- Create an IoT hub
- Link the IoT hub to the Device Provisioning Service
- Set the allocation policy on the Device Provisioning Service

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Sign in to the Azure portal

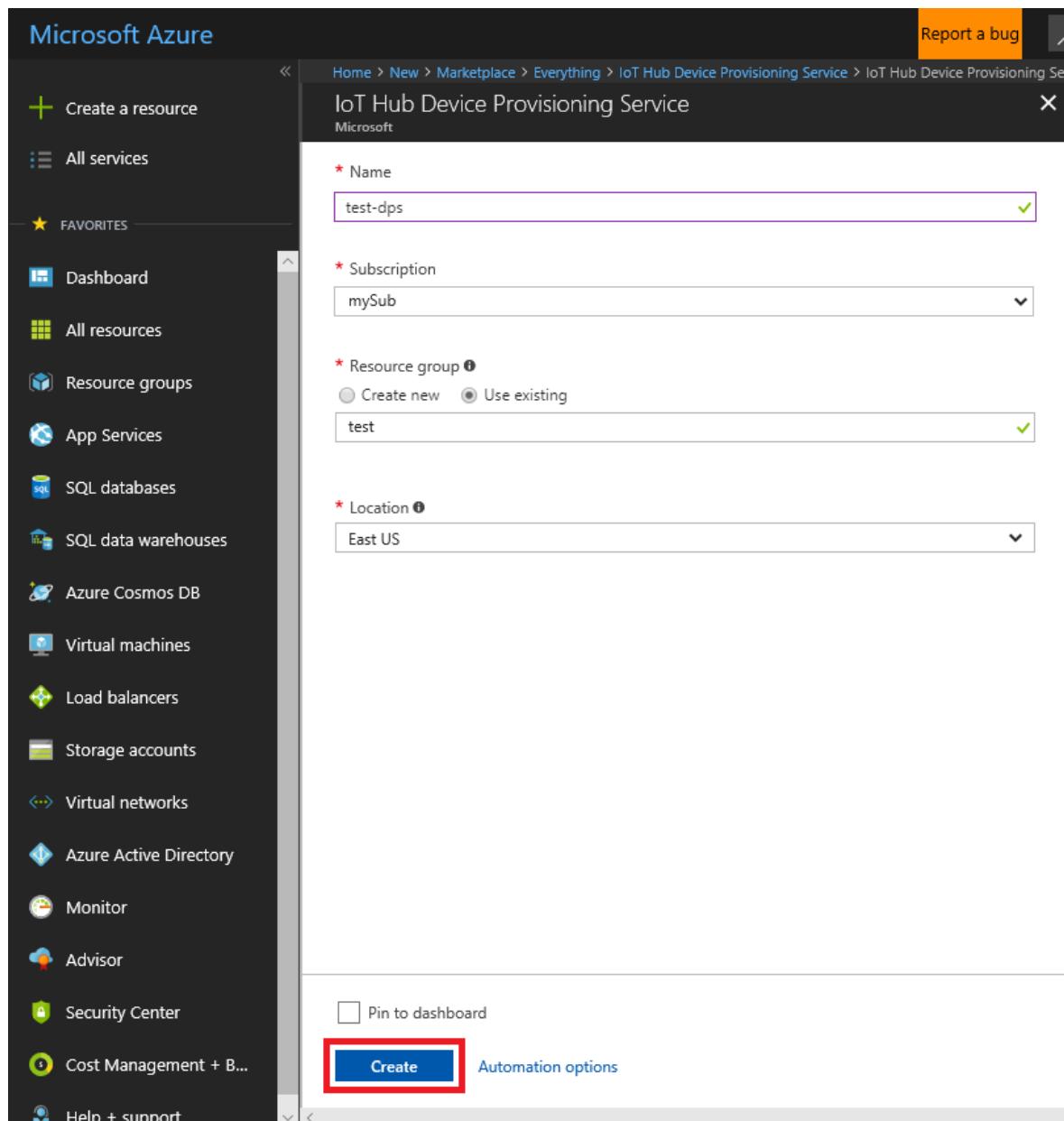
Sign in to the [Azure portal](#).

## Create a Device Provisioning Service instance and get the ID scope

Follow these steps to create a new Device Provisioning Service instance.

1. In the upper left-hand corner of the Azure portal, click **Create a resource**.
2. In the Search box, type **device provisioning**.
3. Click **IoT Hub Device Provisioning Service**.
4. Fill out the **IoT Hub Device Provisioning Service** form with the following information:

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	Any unique name	--
Subscription	Your subscription	For details about your subscriptions, see <a href="#">Subscriptions</a> .
Resource group	myResourceGroup	For valid resource group names, see <a href="#">Naming rules and restrictions</a> .
Location	Any valid location	For information about regions, see <a href="#">Azure Regions</a> .



5. Click **Create**. After a few moments, the Device Provisioning Service instance is created and the **Overview** page is displayed.
6. On the **Overview** page for the new service instance, copy the value for the **ID scope** for use later. That value is used to identify registration IDs, and provides a guarantee that the registration ID is unique.
7. Also, copy the **Service endpoint** value for later use.

## Create an IoT hub

This section describes how to create an IoT hub using the [Azure portal](#).

1. Sign in to the [Azure portal](#).
2. From the Azure homepage, select the **+ Create a resource** button, and then enter *IoTHub* in the **Search the Marketplace** field.
3. Select **IoT Hub** from the search results, and then select **Create**.
4. On the **Basics** tab, complete the fields as follows:
  - **Subscription:** Select the subscription to use for your hub.
  - **Resource Group:** Select a resource group or create a new one. To create a new one, select **Create**

**new** and fill in the name you want to use. To use an existing resource group, select that resource group. For more information, see [Manage Azure Resource Manager resource groups](#).

- **Region:** Select the region in which you want your hub to be located. Select the location closest to you. Some features, such as [IoT Hub device streams](#), are only available in specific regions. For these limited features, you must select one of the supported regions.
- **IoT Hub Name:** Enter a name for your hub. This name must be globally unique. If the name you enter is available, a green check mark appears.

#### IMPORTANT

Because the IoT hub will be publicly discoverable as a DNS endpoint, be sure to avoid entering any sensitive or personally identifiable information when you name it.

Home > New > IoT hub

## IoT hub

Microsoft

X

[Basics](#) [Size and scale](#) [Tags](#) [Review + create](#)

Create an IoT Hub to help you connect, monitor, and manage billions of your IoT assets. [Learn more](#)

**Project details**

Choose the subscription you'll use to manage deployments and costs. Use resource groups like folders to help you organize and manage resources.

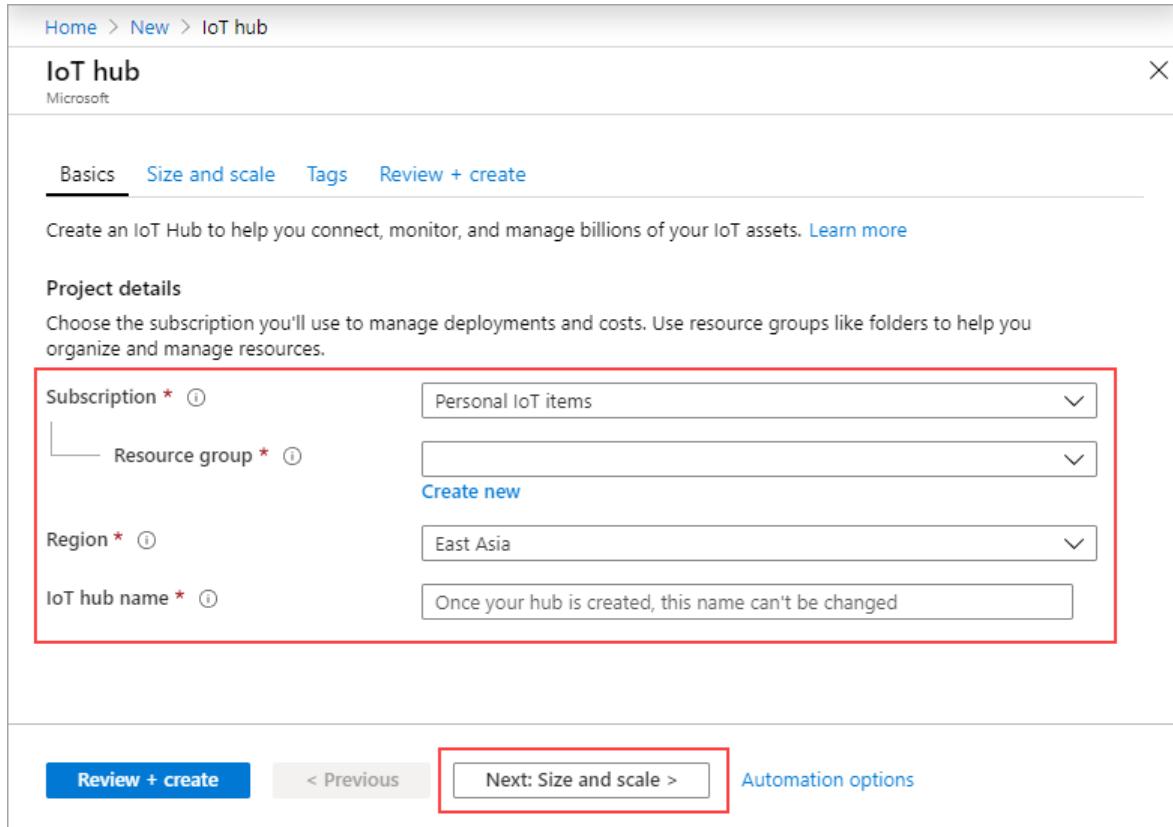
Subscription \* ⓘ Personal IoT items

Resource group \* ⓘ  Create new

Region \* ⓘ East Asia

IoT hub name \* ⓘ Once your hub is created, this name can't be changed

[Review + create](#) [Next: Size and scale >](#) Automation options



5. Select **Next: Size and scale** to continue creating your hub.

Home > New > IoT hub

## IoT hub

Microsoft

Basics Size and scale Tags Review + create

Each IoT hub is provisioned with a certain number of units in a specific tier. The tier and number of units determine the maximum daily quota of messages that you can send. [Learn more](#)

Scale tier and units

Pricing and scale tier \* ⓘ S1: Standard tier [Learn how to choose the right IoT hub tier for your solution](#)

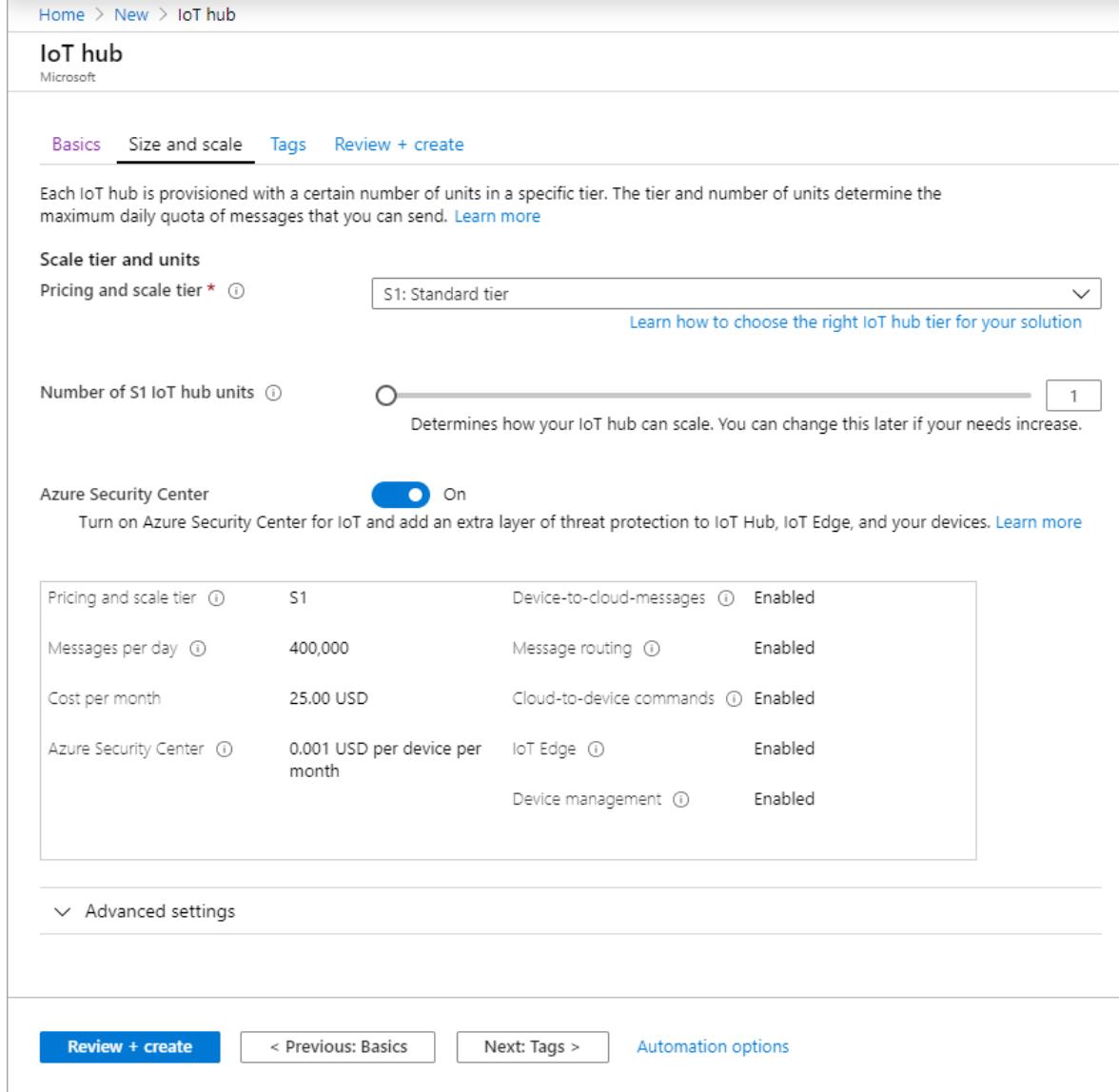
Number of S1 IoT hub units ⓘ 1  
Determines how your IoT hub can scale. You can change this later if your needs increase.

Azure Security Center  On  
Turn on Azure Security Center for IoT and add an extra layer of threat protection to IoT Hub, IoT Edge, and your devices. [Learn more](#)

Pricing and scale tier ⓘ	S1	Device-to-cloud-messages ⓘ	Enabled
Messages per day ⓘ	400,000	Message routing ⓘ	Enabled
Cost per month	25.00 USD	Cloud-to-device commands ⓘ	Enabled
Azure Security Center ⓘ	0.001 USD per device per month	IoT Edge ⓘ	Enabled
		Device management ⓘ	Enabled

Advanced settings

[Review + create](#) [< Previous: Basics](#) [Next: Tags >](#) [Automation options](#)



You can accept the default settings here. If desired, you can modify any of the following fields:

- **Pricing and scale tier:** Your selected tier. You can choose from several tiers, depending on how many features you want and how many messages you send through your solution per day. The free tier is intended for testing and evaluation. It allows 500 devices to be connected to the hub and up to 8,000 messages per day. Each Azure subscription can create one IoT hub in the free tier.  
If you are working through a Quickstart for IoT Hub device streams, select the free tier.
- **IoT Hub units:** The number of messages allowed per unit per day depends on your hub's pricing tier. For example, if you want the hub to support ingress of 700,000 messages, you choose two S1 tier units. For details about the other tier options, see [Choosing the right IoT Hub tier](#).
- **Azure Security Center:** Turn this on to add an extra layer of threat protection to IoT and your devices. This option is not available for hubs in the free tier. For more information about this feature, see [Azure Security Center for IoT](#).
- **Advanced Settings > Device-to-cloud partitions:** This property relates the device-to-cloud messages to the number of simultaneous readers of the messages. Most hubs need only four partitions.

6. Select **Next: Tags** to continue to the next screen.

Tags are name/value pairs. You can assign the same tag to multiple resources and resource groups to categorize resources and consolidate billing. For more information, see [Use tags to organize your Azure](#)

resources.

The screenshot shows the 'Tags' section of the IoT hub creation wizard. It displays a table with two rows of tags. The first row has 'Name' as 'department' and 'Value' as 'accounting'. The second row is partially visible. Below the table are navigation buttons: 'Review + create', '< Previous: Size and scale', 'Next: Review + create >', and 'Automation options'.

Name	Value	Resource
department	accounting	IoT Hub
		IoT Hub

**Review + create**    < Previous: Size and scale    Next: Review + create >    Automation options

7. Select **Next: Review + create** to review your choices. You see something similar to this screen, but with the values you selected when creating the hub.

The screenshot shows the 'Review + create' page for the IoT hub. It lists the chosen parameters under three sections: Basics, Size and scale, and Tags. The 'Review + create' button is highlighted with a red box. At the bottom, the 'Create' button is also highlighted with a red box.

**Basics**

Subscription	Personal testing
Resource group	iot-hubs
Region	West US 2
IoT hub name	you-hub-name

**Size and scale**

Pricing and scale tier	S1
Number of S1 IoT hub units	1
Messages per day	400,000
Cost per month	25.00 USD
Azure Security Center	0.001 USD per device per month

**Tags**

department	accounting
------------	------------

**Create**    < Previous: Tags    Next >    Automation options

8. Select **Create** to create your new hub. Creating the hub takes a few minutes.

### Retrieve connection string for IoT hub

After your hub has been created, retrieve the connection string for the hub. This is used to connect devices and applications to your hub.

1. Click on your hub to see the IoT Hub pane with Settings, and so on. Click Shared access policies.
2. In Shared access policies, select the **iothubowner** policy.
3. Under Shared access keys, copy the Connection string -- primary key to be used later.

The screenshot shows the Azure IoT Hub - Shared access policies blade for the ContosoHub IoT Hub. On the left, there's a sidebar with options like Overview, Activity log, Access control (IAM), Tags, Events, and Shared access policies (which is selected). The main area shows the 'iothubowner' policy details. It includes sections for 'Access policy name' (set to 'iothubowner'), 'Permissions' (with checkboxes for Registry read, Registry write, Service connect, and Device connect all checked), and 'Shared access keys'. The 'Primary key' field contains the connection string: 'HostName=ContosoHub.azure-devices.net;SharedAccessKey...'. This field is highlighted with a red box. Below it, the 'Secondary key' field also contains a connection string, and a link to 'Connection string—secondary key' is shown.

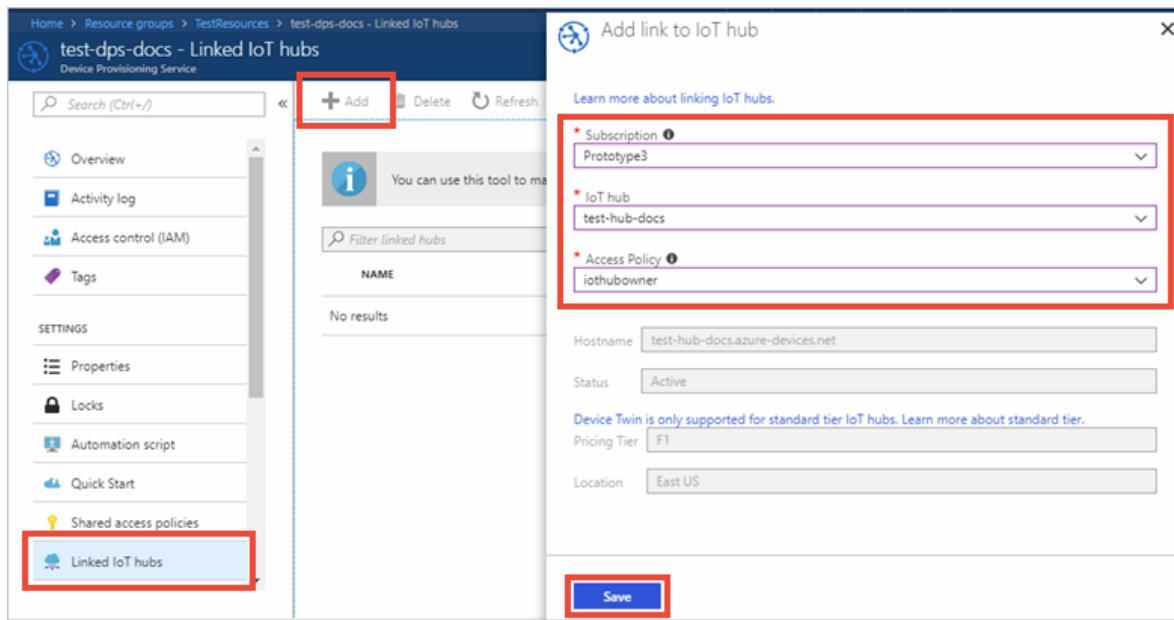
For more information, see [Access control](#) in the "IoT Hub developer guide."

You have now created your IoT hub, and you have the host name and IoT Hub connection string that you need to complete the rest of this tutorial.

## Link the Device Provisioning Service to an IoT hub

The next step is to link the Device Provisioning Service and IoT hub so that the IoT Hub Device Provisioning Service can register devices to that hub. The service can only provision devices to IoT hubs that have been linked to the Device Provisioning Service. Follow these steps.

1. In the **All resources** page, click the Device Provisioning Service instance you created previously.
2. In the Device Provisioning Service page, click **Linked IoT hubs**.
3. Click **Add**.
4. In the **Add link to IoT hub** page, provide the following information, and click **Save**:
  - Subscription:** Make sure the subscription that contains the IoT hub is selected. You can link to IoT hub that resides in a different subscription.
  - IoT hub:** Choose the name of the IoT hub that you want to link with this Device Provisioning Service instance.
  - Access Policy:** Select **iothubowner** as the credentials to use for establishing the link to the IoT hub.

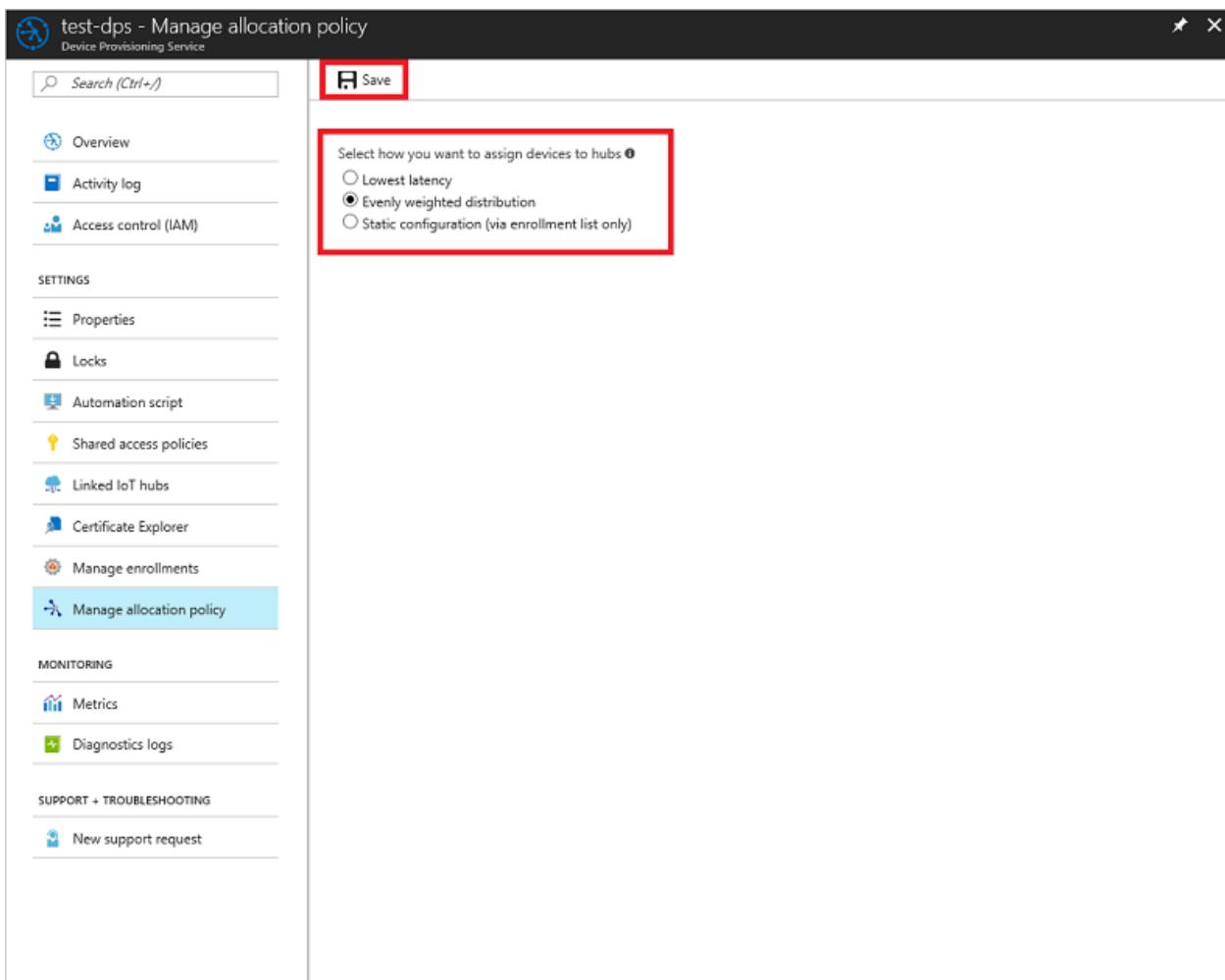


## Set the allocation policy on the Device Provisioning Service

The allocation policy is an IoT Hub Device Provisioning Service setting that determines how devices are assigned to an IoT hub. There are three supported allocation policies:

1. **Lowest latency**: Devices are provisioned to an IoT hub based on the hub with the lowest latency to the device.
2. **Evenly weighted distribution** (default): Linked IoT hubs are equally likely to have devices provisioned to them. This setting is the default. If you are provisioning devices to only one IoT hub, you can keep this setting.
3. **Static configuration via the enrollment list**: Specification of the desired IoT hub in the enrollment list takes priority over the Device Provisioning Service-level allocation policy.

To set the allocation policy, in the Device Provisioning Service page click **Manage allocation policy**. Make sure the allocation policy is set to **Evenly weighted distribution** (the default). If you make any changes, click **Save** when you are done.



## Clean up resources

Other tutorials in this collection build upon this tutorial. If you plan to continue on to work with subsequent quick starts or with the tutorials, do not clean up the resources created in this tutorial. If you do not plan to continue, use the following steps to delete all resources created by this tutorial in the Azure portal.

1. From the left-hand menu in the Azure portal, click **All resources** and then select your IoT Hub Device Provisioning Service instance. At the top of the **All resources** page, click **Delete**.
2. From the left-hand menu in the Azure portal, click **All resources** and then select your IoT hub. At the top of the **All resources** page, click **Delete**.

## Next steps

In this tutorial, you learned how to:

- Use the Azure portal to create an IoT Hub Device Provisioning Service and get the ID scope
- Create an IoT hub
- Link the IoT hub to the Device Provisioning Service
- Set the allocation policy on the Device Provisioning Service

Advance to the next tutorial to learn how to set up your device for provisioning.

[Set up device for provisioning](#)

# Tutorial: Set up a device to provision using the Azure IoT Hub Device Provisioning Service

12/19/2019 • 9 minutes to read • [Edit Online](#)

In the previous tutorial, you learned how to set up the Azure IoT Hub Device Provisioning Service to automatically provision your devices to your IoT hub. This tutorial shows you how to set up your device during the manufacturing process, enabling it to be auto-provisioned with IoT Hub. Your device is provisioned based on its [Attestation mechanism](#), upon first boot and connection to the provisioning service. This tutorial covers the following tasks:

- Build platform-specific Device Provisioning Services Client SDK
- Extract the security artifacts
- Create the device registration software

This tutorial expects that you have already created your Device Provisioning Service instance and an IoT hub, using the instructions in the previous [Set up cloud resources](#) tutorial.

This tutorial uses the [Azure IoT SDKs and libraries for C repository](#), which contains the Device Provisioning Service Client SDK for C. The SDK currently provides TPM and X.509 support for devices running on Windows or Ubuntu implementations. This tutorial is based on use of a Windows development client, which also assumes basic proficiency with Visual Studio.

If you're unfamiliar with the process of auto-provisioning, be sure to review [Auto-provisioning concepts](#) before continuing.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

The following prerequisites are for a Windows development environment. For Linux or macOS, see the appropriate section in [Prepare your development environment](#) in the SDK documentation.

- [Visual Studio](#) 2019 with the '[Desktop development with C++](#)' workload enabled. Visual Studio 2015 and Visual Studio 2017 are also supported.
- Latest version of [Git](#) installed.

## Build a platform-specific version of the SDK

The Device Provisioning Service Client SDK helps you implement your device registration software. But before you can use it, you need to build a version of the SDK specific to your development client platform and attestation mechanism. In this tutorial, you build an SDK that uses Visual Studio on a Windows development platform, for a supported type of attestation:

1. Download the [CMake build system](#).

It is important that the Visual Studio prerequisites (Visual Studio and the '[Desktop development with C++](#)' workload) are installed on your machine, **before** starting the [CMake](#) installation. Once the prerequisites are in place, and the download is verified, install the CMake build system.

2. Find the tag name for the [latest release](#) of the SDK.

3. Open a command prompt or Git Bash shell. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. Create a `cmake` subdirectory in the root directory of the git repository, and navigate to that folder. Run the following commands from the `azure-iot-sdk-c` directory:

```
mkdir cmake
cd cmake
```

5. Build the SDK for your development platform based on the attestation mechanisms you will be using. Use one of the following commands (also note the two trailing period characters for each command). Upon completion, CMake builds out the `/cmake` subdirectory with content specific to your device:

- For devices that use the TPM simulator for attestation:

```
cmake -Duse_prov_client:BOOL=ON -Duse_tpm_simulator:BOOL=ON ..
```

- For any other device (physical TPM/HSM/X.509, or a simulated X.509 certificate):

```
cmake -Duse_prov_client:BOOL=ON ..
```

Now you're ready to use the SDK to build your device registration code.

## Extract the security artifacts

The next step is to extract the security artifacts for the attestation mechanism used by your device.

### Physical devices

Depending on whether you built the SDK to use attestation for a physical TPM/HSM or using X.509 certificates, gathering the security artifacts is as follows:

- For a TPM device, you need to determine the **Endorsement Key** associated with it from the TPM chip manufacturer. You can derive a unique **Registration ID** for your TPM device by hashing the endorsement key.
- For an X.509 device, you need to obtain the certificates issued to your device(s). The provisioning service exposes two types of enrollment entries that control access for devices using the X.509 attestation mechanism. The certificates needed depend on the enrollment types you will be using.
  - Individual enrollments: Enrollment for a specific single device. This type of enrollment entry requires [end-entity, "leaf", certificates](#).
  - Enrollment groups: This type of enrollment entry requires intermediate or root certificates. For more information, see [Controlling device access to the provisioning service with X.509 certificates](#).

### Simulated devices

Depending on whether you built the SDK to use attestation for a simulated device using TPM or X.509 certificates,

gathering the security artifacts is as follows:

- For a simulated TPM device:

1. Open a Windows Command Prompt, navigate to the `azure-iot-sdk-c` subdirectory, and run the TPM simulator. It listens over a socket on ports 2321 and 2322. Do not close this command window; you will need to keep this simulator running until the end of the following Quickstart.

From the `azure-iot-sdk-c` subdirectory, run the following command to start the simulator:

```
.\provisioning_client\deps\utpm\tools\tpm_simulator\Simulator.exe
```

**NOTE**

If you use the Git Bash command prompt for this step, you'll need to change the backslashes to forward slashes, for example: `./provisioning_client/deps/utpm/tools/tpm_simulator/Simulator.exe`.

2. Using Visual Studio, open the solution generated in the `cmake` folder named `azure_iot_sdks.sln`, and build it using the "Build solution" command on the "Build" menu.
  3. In the *Solution Explorer* pane in Visual Studio, navigate to the folder **Provision\_Tools**. Right-click the `tpm_device_provision` project and select **Set as Startup Project**.
  4. Run the solution using either of the "Start" commands on the "Debug" menu. The output window displays the TPM simulator's **Registration ID** and the **Endorsement Key**, needed for device enrollment and registration. Copy these values for use later. You can close this window (with Registration ID and Endorsement Key), but leave the TPM simulator window running that you started in step #1.
- For a simulated X.509 device:
1. Using Visual Studio, open the solution generated in the `cmake` folder named `azure_iot_sdks.sln`, and build it using the "Build solution" command on the "Build" menu.
  2. In the *Solution Explorer* pane in Visual Studio, navigate to the folder **Provision\_Tools**. Right-click the `dice_device_enrollment` project and select **Set as Startup Project**.
  3. Run the solution using either of the "Start" commands on the "Debug" menu. In the output window, enter `i` for individual enrollment when prompted. The output window displays a locally generated X.509 certificate for your simulated device. Copy to clipboard the output starting from `-----BEGIN CERTIFICATE-----` and ending at the first `-----END CERTIFICATE-----`, making sure to include both of these lines as well. You only need the first certificate from the output window.
  4. Create a file named `X509testcert.pem`, open it in a text editor of your choice, and copy the clipboard contents to this file. Save the file as you will use it later for device enrollment. When your registration software runs, it uses the same certificate during auto-provisioning.

These security artifacts are required during enrollment your device to the Device Provisioning Service. The provisioning service waits for the device to boot and connect with it at any later point in time. When your device boots for the first time, the client SDK logic interacts with your chip (or simulator) to extract the security artifacts from the device, and verifies registration with your Device Provisioning service.

## Create the device registration software

The last step is to write a registration application that uses the Device Provisioning Service client SDK to register the device with the IoT Hub service.

## NOTE

For this step we will assume the use of a simulated device, accomplished by running an SDK sample registration application from your workstation. However, the same concepts apply if you are building a registration application for deployment to a physical device.

1. In the Azure portal, select the **Overview** blade for your Device Provisioning service and copy the *ID Scope* value. The *ID Scope* is generated by the service and guarantees uniqueness. It is immutable and used to uniquely identify the registration IDs.

The screenshot shows the Azure portal interface for a Device Provisioning Service resource group named "test-docs-dps". The left sidebar lists various management options: Overview (selected), Activity log, Access control (IAM), Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, and Manage enrollments. The main content area displays the "Overview" details for the resource group. Key information includes:

- Resource group: test-docs-dps
- Status: Active
- Location: East US
- Subscription: Microsoft Azure Internal Consumption
- Subscription ID: [REDACTED]
- Service endpoint: test-docs-dps.azure-devices-provisioning.net
- Global device endpoint: global.azure-devices-provisioning.net
- ID Scope: One00002193 (highlighted with a red box)
- Pricing and scale tier: S1

Below the main details, there is a "Quick Links" section with three items:

- Azure IoT Hub Device Provisioning Service Documentation
- Learn more about IoT Hub Device Provisioning Service
- Device Provisioning concepts

2. In the Visual Studio *Solution Explorer* on your machine, navigate to the folder **Provision\_Samples**. Select the sample project named **prov\_dev\_client\_sample** and open the source file **prov\_dev\_client\_sample.c**.
  3. Assign the *ID Scope* value obtained in step #1, to the `id_scope` variable (removing the left/ [ ] and right/ ] brackets):

```
static const char* global_prov_uri = "global.azure-devices-provisioning.net";
static const char* id_scope = "[ID Scope]";
```

For reference, the `global_prov_uri` variable, which allows the IoT Hub client registration API `IoTHubClient_LL_CreateFromDeviceAuth` to connect with the designated Device Provisioning Service instance.

4. In the `main()` function in the same file, comment/uncomment the `hsm_type` variable that matches the attestation mechanism being used by your device's registration software (TPM or X.509):

```
hsm_type = SECURE_DEVICE_TYPE_TPM;  
//hsm_type = SECURE_DEVICE_TYPE_X509;
```

5. Save your changes and rebuild the `prov_dev_client_sample` sample by selecting "Build solution" from the "Build" menu.
  6. Right-click the `prov_dev_client_sample` project under the `Provision_Samples` folder, and select `Set as Startup Project`. DO NOT run the sample application yet.

## IMPORTANT

Do not run/start the device yet! You need to finish the process by enrolling the device with the Device Provisioning Service first, before starting the device. The Next steps section below will guide you to the next article.

## SDK APIs used during registration (for reference only)

For reference, the SDK provides the following APIs for your application to use during registration. These APIs help your device connect and register with the Device Provisioning Service when it boots up. In return, your device receives the information required to establish a connection to your IoT Hub instance:

```
// Creates a Provisioning Client for communications with the Device Provisioning Client Service.  
PROV_DEVICE_LL_HANDLE Prov_Device_LL_Create(const char* uri, const char* scope_id,  
PROV_DEVICE_TRANSPORT_PROVIDER_FUNCTION protocol)  
  
// Disposes of resources allocated by the provisioning Client.  
void Prov_Device_LL_Destroy(PROV_DEVICE_LL_HANDLE handle)  
  
// Asynchronous call initiates the registration of a device.  
PROV_DEVICE_RESULT Prov_Device_LL_Register_Device(PROV_DEVICE_LL_HANDLE handle,  
PROV_DEVICE_CLIENT_REGISTER_DEVICE_CALLBACK register_callback, void* user_context,  
PROV_DEVICE_CLIENT_REGISTER_STATUS_CALLBACK reg_status_cb, void* status_user_ctxt)  
  
// Api to be called by user when work (registering device) can be done  
void Prov_Device_LL_DoWork(PROV_DEVICE_LL_HANDLE handle)  
  
// API sets a runtime option identified by parameter optionName to a value pointed to by value  
PROV_DEVICE_RESULT Prov_Device_LL_SetOption(PROV_DEVICE_LL_HANDLE handle, const char* optionName, const void*  
value)
```

You may also find that you need to refine your Device Provisioning Service client registration application, using a simulated device at first, and a test service setup. Once your application is working in the test environment, you can build it for your specific device and copy the executable to your device image.

## Clean up resources

At this point, you might have the Device Provisioning and IoT Hub services running in the portal. If you wish to abandon the device provisioning setup, and/or delay completion of this tutorial series, we recommend shutting them down to avoid incurring unnecessary costs.

1. From the left-hand menu in the Azure portal, click **All resources** and then select your Device Provisioning service. At the top of the **All resources** blade, click **Delete**.
2. From the left-hand menu in the Azure portal, click **All resources** and then select your IoT hub. At the top of the **All resources** blade, click **Delete**.

## Next steps

In this tutorial, you learned how to:

- Build platform-specific Device Provisioning Service Client SDK
- Extract the security artifacts
- Create the device registration software

Advance to the next tutorial to learn how to provision the device to your IoT hub by enrolling it to the Azure IoT Hub Device Provisioning Service for auto-provisioning.

[Provision the device to your IoT hub](#)

# Tutorial: Provision the device to an IoT hub using the Azure IoT Hub Device Provisioning Service

12/10/2019 • 4 minutes to read • [Edit Online](#)

In the previous tutorial, you learned how to set up a device to connect to your Device Provisioning service. In this tutorial, you learn how to use this service to provision your device to a single IoT hub, using auto-provisioning and **enrollment lists**. This tutorial shows you how to:

- Enroll the device
- Start the device
- Verify the device is registered

## Prerequisites

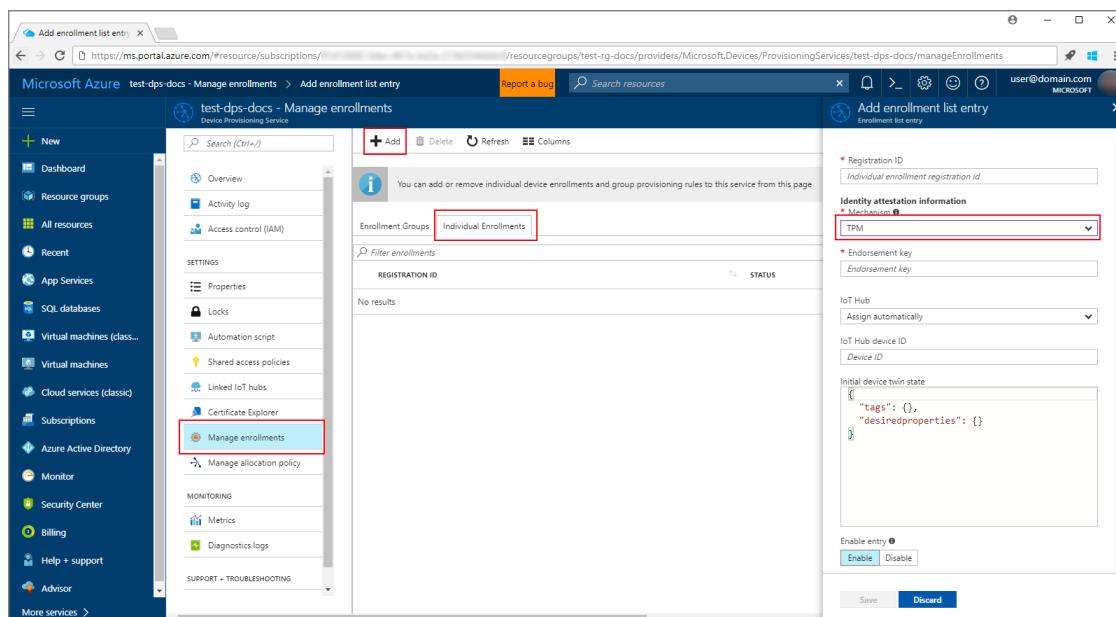
Before you proceed, make sure to configure your device as discussed in the tutorial [Setup a device to provision using Azure IoT Hub Device Provisioning Service](#).

If you're unfamiliar with the process of auto-provisioning, be sure to review [Auto-provisioning concepts](#) before continuing.

## Enroll the device

This step involves adding the device's unique security artifacts to the Device Provisioning Service. These security artifacts are based on the device's [Attestation mechanism](#) as follows:

- For TPM-based devices you need:
  - The *Endorsement Key* that is unique to each TPM chip or simulation, which is obtained from the TPM chip manufacturer. Read the [Understand TPM Endorsement Key](#) for more information.
  - The *Registration ID* that is used to uniquely identify a device in the namespace/scope. This ID may or may not be the same as the device ID. The ID is mandatory for every device. For TPM-based devices, the registration ID may be derived from the TPM itself, for example, an SHA-256 hash of the TPM Endorsement Key.



- For X.509 based devices you need:

- The [certificate issued to the X.509](#) chip or simulation, in the form of either a `.pem` or a `.cer` file. For individual enrollment, you need to use the per-device *signed certificate* for your X.509 system, while for enrollment groups, you need to use the *root certificate*.

The screenshot shows two windows side-by-side. On the left is the 'test-dps-docs - Manage enrollments' page, which has a sidebar with various settings like Overview, Activity log, Access control (IAM), Tags, Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, and Manage enrollments (which is highlighted with a red box). The main area shows tabs for 'Enrollment Groups' and 'Individual Enrollments' (also highlighted with a red box). On the right is the 'Add Enrollment' dialog, also with a sidebar. The 'Mechanism' dropdown is set to 'X.509'. Below it, the 'Primary Certificate .pem or .cer file' input field is highlighted with a red box. The 'Secondary Certificate .pem or .cer file' input field is below it. Under 'REGISTRATION ID', there's a dropdown for 'IoT Hub' set to 'Assign automatically' and an input field for 'IoT Hub Device ID'. A section for 'Initial Device Twin State' contains JSON code: 

```
{"tags": {}, "properties": { "desired": {} } }
```

. At the bottom of the dialog is a 'Save' button, which is also highlighted with a red box.

There are two ways to enroll the device to the Device Provisioning Service:

- Enrollment Groups** This represents a group of devices that share a specific attestation mechanism. We recommend using an enrollment group for a large number of devices, which share a desired initial configuration, or for devices all going to the same tenant. For more information on Identity attestation for enrollment groups, see [Security](#).

The screenshot shows the Azure Device Provisioning Service (DPS) portal. On the left, the navigation menu includes sections like Overview, Activity log, Access control (IAM), Tags, Settings (Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates), Monitoring (Metrics (preview), Diagnostics settings), and Support + Troubleshooting (New support request). The 'Certificates' section has 'Manage enrollments' highlighted with a red box.

The main area displays the 'Add Enrollment Group' dialog. It has tabs for 'Enrollment Groups' and 'Individual Enrollments'. A search bar says 'Filter enrollments'. Below it is a 'GROUP NAME' field containing 'test-enrollment-group', which is also highlighted with a red box. Under 'Identity Attestation Information', there are tabs for 'CA Certificate' (selected) and 'Intermediate Certificate'. The 'Primary Certificate' dropdown is set to 'No certificate selected' and is also highlighted with a red box. The 'Secondary Certificate' dropdown is also set to 'No certificate selected'. A note below states: 'Device Twin is only supported for standard tier IoT hubs. Learn more about standard tier.' Under 'Desired IoT Hub', a dropdown is set to 'Assign automatically'. The 'Initial Device Twin State' section contains a JSON object:

```
{
  "tags": {},
  "properties": {
    "desired": {}
  }
}
```

At the bottom of the dialog are 'Enable entry' buttons ('Enable' and 'Disable') and a large blue 'Save' button, which is also highlighted with a red box.

- **Individual Enrollments** This represents an entry for a single device that may register with the Device Provisioning Service. Individual enrollments may use either x509 certificates or SAS tokens (in a real or virtual TPM) as attestation mechanisms. We recommend using individual enrollments for devices that require unique initial configurations, and devices that can only use SAS tokens via TPM or virtual TPM as the attestation mechanism. Individual enrollments may have the desired IoT hub device ID specified.

Now you enroll the device with your Device Provisioning Service instance, using the required security artifacts based on the device's attestation mechanism:

1. Sign in to the Azure portal, click on the **All resources** button on the left-hand menu and open your Device Provisioning service.
2. On the Device Provisioning Service summary blade, select **Manage enrollments**. Select either **Individual Enrollments** tab or the **Enrollment Groups** tab as per your device setup. Click the **Add** button at the top. Select **TPM** or **X.509** as the identity attestation *Mechanism*, and enter the appropriate security artifacts as discussed previously. You may enter a new **IoT Hub device ID**. Once complete, click the **Save** button.
3. When the device is successfully enrolled, you should see it displayed in the portal as follows:

The screenshot shows the Azure Device Provisioning Service interface for managing device enrollments. The left sidebar contains various navigation links, with 'Manage enrollments' being the active link, indicated by a red box. The main content area displays a table for individual device enrollments, with the 'Individual Enrollments' tab selected, also highlighted by a red box. The 'Refresh' button at the top right of the main content area is also highlighted with a red box.

After enrollment, the provisioning service then waits for the device to boot and connect with it at any later point in time. When your device boots for the first time, the client SDK library interacts with your chip to extract the security artifacts from the device, and verifies registration with your Device Provisioning service.

## Start the IoT device

Your IoT device can be a real device, or a simulated device. Since the IoT device has now been enrolled with a Device Provisioning Service instance, the device can now boot up, and call the provisioning service to be recognized using the attestation mechanism. Once the provisioning service has recognized the device, it will be assigned to an IoT hub.

Simulated device examples, using both TPM and X.509 attestation, are included for C, Java, C#, Node.js, and Python. For example, a simulated device using TPM and the [Azure IoT C SDK](#) would follow the process covered in the [Simulate first boot sequence for a device](#) section. The same device using X.509 certificate attestation would refer to this [boot sequence](#) section.

Refer to the [How-to guide for the MXChip IoT DevKit](#) as an example for a real device.

Start the device to allow your device's client application to start the registration with your Device Provisioning service.

## Verify the device is registered

Once your device boots, the following actions should take place:

1. The device sends a registration request to your Device Provisioning service.
2. For TPM devices, the Device Provisioning Service sends back a registration challenge to which your device responds.
3. On successful registration, the Device Provisioning Service sends the IoT hub URI, device ID, and the encrypted key back to the device.
4. The IoT Hub client application on the device then connects to your hub.

5. On successful connection to the hub, you should see the device appear in the IoT hub's IoT Devices explorer.

The screenshot shows the 'test-hub-docs - IoT devices' blade in the Azure portal. The left sidebar contains navigation links for 'Shared access policies', 'Pricing and scale', 'Operations monitoring', 'IP Filter', 'Certificates', 'Properties', 'Locks', and 'Automation script'. Under 'EXPLORERS', 'Query explorer' is listed, and 'IoT devices' is highlighted with a red box. The main area features a 'Refresh' button with a red box around it, a query editor with a sample SQL query, and a table listing four devices: AZ3166, simDevice, test-docs-cert-device, and test-docs-device. Each device entry includes columns for DEVICE ID, STATUS, LAST ACTIVITY, LAST STATUS UPDATE, AUTHENTICATION TYPE, and C.

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS UPDATE	AUTHENTICATION TYPE	C
AZ3166	Enabled	Wed Jul 18 20...		SelfSigned	0
simDevice	Enabled	Fri Jun 22 201...		Sas	0
test-docs-cert-device	Enabled			SelfSigned	0
test-docs-device	Enabled			Sas	0

For more information, see the provisioning device client sample, [prov\\_dev\\_client\\_sample.c](#). The sample demonstrates provisioning a simulated device using TPM, X.509 certificates and symmetric keys. Refer back to the [TPM](#), [X.509](#), and [Symmetric key](#) attestation quickstarts for step-by-step instructions on using the sample.

## Next steps

In this tutorial, you learned how to:

- Enroll the device
- Start the device
- Verify the device is registered

Advance to the next tutorial to learn how to provision multiple devices across load-balanced hubs.

[Provision devices across load-balanced IoT hubs](#)

# Tutorial: Enroll the device to an IoT hub using the Azure IoT Hub Provisioning Service Client (.NET)

12/10/2019 • 6 minutes to read • [Edit Online](#)

In the previous tutorial, you learned how to set up a device to connect to your Device Provisioning service. In this tutorial, you learn how to use this service to provision your device to a single IoT hub, using both *Individual Enrollment* and *Enrollment Groups*. This tutorial shows you how to:

- Enroll the device
- Start the device
- Verify the device is registered

## Prerequisites

Before you proceed, make sure to configure your device and its *Hardware Security Module* as discussed in the tutorial [Set up a device to provision using Azure IoT Hub Device Provisioning Service](#).

- Visual Studio

### NOTE

Visual Studio is not required. The installation of .NET is sufficient and developers can use their preferred editor on Windows or Linux.

This tutorial simulates the period during or right after the hardware manufacturing process, when device information is added to the provisioning service. This code is usually run on a PC or a factory device that can run .NET code and should not be added to the devices themselves.

## Enroll the device

This step involves adding the device's unique security artifacts to the Device Provisioning Service. These security artifacts are as follows:

- For TPM-based devices:
  - The *Endorsement Key* that is unique to each TPM chip or simulation. Read the [Understand TPM Endorsement Key](#) for more information.
  - The *Registration ID* that is used to uniquely identify a device in the namespace/scope. This may or may not be the same as the device ID. The ID is mandatory for every device. For TPM-based devices, the registration ID may be derived from the TPM itself, for example, an SHA-256 hash of the TPM Endorsement Key.
- For X.509 based devices:
  - The [X.509 certificate issued to the device](#), in the form of either a *.pem* or a *.cer* file. For individual enrollment, you need to use the *leaf certificate* for your X.509 system, while for enrollment groups, you need to use the *root certificate* or an equivalent *signer certificate*.
  - The *Registration ID* that is used to uniquely identify a device in the namespace/scope. This may or may not be the same as the device ID. The ID is mandatory for every device. For X.509 based devices, the registration ID is derived from the certificate's common name (CN). For further information on these

requirements see [Device concepts](#).

There are two ways to enroll the device to the Device Provisioning Service:

- **Individual Enrollments** This represents an entry for a single device that may register with the Device Provisioning Service. Individual enrollments may use either X.509 certificates or SAS tokens (in a real or virtual TPM) as attestation mechanisms. We recommend using individual enrollments for devices, which require unique initial configurations, or for devices that can only use SAS tokens via TPM as the attestation mechanism. Individual enrollments may have the desired IoT hub device ID specified.
- **Enrollment Groups** This represents a group of devices that share a specific attestation mechanism. We recommend using an enrollment group for a large number of devices, which share a desired initial configuration, or for devices all going to the same tenant. Enrollment groups are X.509 only and all share a signing certificate in their X.509 certificate chain.

### Enroll the device using Individual Enrollments

1. In Visual Studio, create a Visual C# Console Application project by using the **Console App** project template. Name the project **DeviceProvisioning**.
2. In Solution Explorer, right-click the **DeviceProvisioning** project, and then click **Manage NuGet Packages....**
3. In the **NuGet Package Manager** window, select **Browse** and search for **microsoft.azure.devices.provisioning.service**. Select the entry and click **Install** to install the **Microsoft.Azure.Devices.Provisioning.Service** package, and accept the terms of use. This procedure downloads, installs, and adds a reference to the [Azure IoT Device Provisioning Service SDK](#) NuGet package and its dependencies.
4. Add the following `using` statements at the top of the **Program.cs** file:

```
using Microsoft.Azure.Devices.Provisioning.Service;
```

5. Add the following fields to the **Program** class. Replace the placeholder value with the Device Provisioning Service connection string noted in the previous section.

```
static readonly string ServiceConnectionString = "{Device Provisioning Service connection string}";

private const string SampleRegistrationId = "sample-individual-csharp";
private const string SampleTpmEndorsementKey =
    "AToAAQALAAMAsgAgg3GXZ0SEs/gakMyNrqXXJP1S124GUgtk8qHaGzMuaaoABgCAAEMAEAgAAAAAAEAsxsj2gUS" +
    "cTk1UjuioeTlfGYZrrimExB+bScH75adUMRIi2UOMxG1kw4y+9RW/IVoMl4e620VxZad0ARX2gUqVjY07KPvt3d" +
    "yKhZS3dkcvfBisBhP1XH9B33VqHG9SHnbnQXdBUaCgKAfxome8UmBKfe+naTsE5fkvjb/do3/db6l4sGBwFCnKR" +
    "dln4XpM03zLpoHfao8zOwt81/uP3qUIxmCYv9A7m69Ms+5/pCkTu/rK4mRDsfhZ0QLfbzVI6zQFOKF/rwsfBtFe" +
    "WlWtcuJMK1XdD8TXWE1Tzgh7JS4qhFzreL0c1mI0GCj+Aws0usZh7dLIVPnlgZcBhgy1SSDQM==";
private const string OptionalDeviceId = "myCSharpDevice";
private const ProvisioningStatus OptionalProvisioningStatus = ProvisioningStatus.Enabled;
```

6. Add the following to implement the enrollment for the device:

```

static async Task SetRegistrationDataAsync()
{
    Console.WriteLine("Starting SetRegistrationData");

    Attestation attestation = new TpmAttestation(SampleTpmEndorsementKey);

    IndividualEnrollment individualEnrollment = new IndividualEnrollment(SampleRegistrationId,
    attestation);

    individualEnrollment.DeviceId = OptionalDeviceId;
    individualEnrollment.ProvisioningStatus = OptionalProvisioningStatus;

    Console.WriteLine("\nAdding new individualEnrollment...");
    var serviceClient = ProvisioningServiceClient.CreateFromConnectionString(ServiceConnectionString);

    IndividualEnrollment individualEnrollmentResult =
        await
    serviceClient.CreateOrUpdateIndividualEnrollmentAsync(individualEnrollment).ConfigureAwait(false);

    Console.WriteLine("\nIndividualEnrollment created with success.");
    Console.WriteLine(individualEnrollmentResult);
}

```

- Finally, add the following code to the **Main** method to open the connection to your IoT hub and begin the enrollment:

```

try
{
    Console.WriteLine("IoT Device Provisioning example");

    SetRegistrationDataAsync().GetAwaiter().GetResult();

    Console.WriteLine("Done, hit enter to exit.");
}
catch (Exception ex)
{
    Console.WriteLine();
    Console.WriteLine("Error in sample: {0}", ex.Message);
}
Console.ReadLine();

```

- In the Visual Studio Solution Explorer, right-click your solution, and then click **Set StartUp Projects....** Select **Single startup project**, and then select the **DeviceProvisioning** project in the dropdown menu.
- Run the .NET device app **DeviceProvisioning**. It should set up provisioning for the device:

```

C:\Users\v-masebo\Desktop\DeviceProvisioning\DeviceProvisioning\bin\Debug\DeviceProvisioning.exe
IoT Device Provisioning example
Starting SetRegistrationData
Adding new individualEnrollment...
IndividualEnrollment created with success.
{
  "attestation": {
    "type": "tpm",
    "tpm": {
      "endorsementKey": "
    }
  },
  "registrationId": "sample-individual-csharp",
  "deviceId": "myCSharpDevice",
  "provisioningStatus": "enabled",
  "createdDateTimeUtc": "2018-01-09T00:28:26.1600452Z",
  "lastUpdatedDateTimeUtc": "2018-01-09T00:28:26.1600452Z",
  "etag": "\0a00cf32-0000-0000-0000-5a540caa0000\""
}
Done, hit enter to exit.

```

When the device is successfully enrolled, you should see it displayed in the portal as following:

## Enroll the device using Enrollment Groups

### NOTE

The enrollment group sample requires an X.509 certificate.

1. In the Visual Studio Solution Explorer, open the **DeviceProvisioning** project created above.
2. Add the following `using` statements at the top of the **Program.cs** file:

```
using System.Security.Cryptography.X509Certificates;
```

3. Add the following fields to the **Program** class. Replace the placeholder value with the X509 certificate location.

```
private const string X509RootCertPathVar = "{X509 Certificate Location}";
private const string SampleEnrollmentGroupId = "sample-group-csharp";
```

4. Add the following to **Program.cs** implement the enrollment for the group:

```
public static async Task SetGroupRegistrationDataAsync()
{
    Console.WriteLine("Starting SetGroupRegistrationData");

    using (ProvisioningServiceClient provisioningServiceClient =
        ProvisioningServiceClient.CreateFromConnectionString(ServiceConnectionString))
    {
        Console.WriteLine("\nCreating a new enrollmentGroup...");

        var certificate = new X509Certificate2(X509RootCertPathVar);

        Attestation attestation = X509Attestation.CreateFromRootCertificates(certificate);

        EnrollmentGroup enrollmentGroup = new EnrollmentGroup(SampleEnrollmentGroupId, attestation);

        Console.WriteLine(enrollmentGroup);
        Console.WriteLine("\nAdding new enrollmentGroup...");

        EnrollmentGroup enrollmentGroupResult =
            await
provisioningServiceClient.CreateOrUpdateEnrollmentGroupAsync(enrollmentGroup).ConfigureAwait(false);

        Console.WriteLine("\nEnrollmentGroup created with success.");
        Console.WriteLine(enrollmentGroupResult);
    }
}
```

5. Finally, replace the following code to the **Main** method to open the connection to your IoT hub and begin the group enrollment:

```
try
{
    Console.WriteLine("IoT Device Group Provisioning example");

    SetGroupRegistrationDataAsync().GetAwaiter().GetResult();

    Console.WriteLine("Done, hit enter to exit.");
    Console.ReadLine();
}
catch (Exception ex)
{
    Console.WriteLine();
    Console.WriteLine("Error in sample: {0}", ex.Message);
}
```

6. Run the .NET device app **DeviceProvisiong**. It should set up group provisioning for the device:

```

C:\Users\v-masebo\Desktop\DeviceProvisioning\DeviceProvisioning\bin\Debug\DeviceProvisioning.exe
IoT Device Group Provisioning example
Starting SetGroupRegistrationData

Creating a new enrollmentGroup...
{
  "attestation": {
    "type": "x509",
    "x509": {
      "signingCertificates": {
        "primary": {
          "certificate": "-----"
        }
      }
    }
  },
  "enrollmentGroupId": "sample-group-csharp"
}

Adding new enrollmentGroup...

EnrollmentGroup created with success.
{
  "attestation": {
    "type": "x509",
    "x509": {
      "signingCertificates": {
        "primary": {
          "certificate": null,
          "info": {
            "subjectName": "CN=microsoftiotcoreroot, O=MSR_TEST, C=US",
            "sha1Thumbprint": "-----",
            "sha256Thumbprint": "-----",
            "issuerName": "CN=microsoftiotcoreroot, O=MSR_TEST, C=US",
            "notBeforeUtc": "2017-01-01T00:00:00Z",
            "notAfterUtc": "3701-01-31T23:59:59Z",
            "serialNumber": "5A4B3C2D1E",
            "version": 3
          }
        }
      }
    }
  },
  "enrollmentGroupId": "sample-group-csharp",
  "provisioningStatus": 0,
  "createdDateTimeUtc": "2018-01-09T00:34:29.9919838Z",
  "lastUpdatedDateTimeUtc": "2018-01-09T00:34:29.9919838Z",
  "etag": "\"\\1400556d-0000-0000-0000-5a540e160000\""
}
Done, hit enter to exit.

```

When the device group is successfully enrolled, you should see it displayed in the portal as following:

The screenshot shows the Azure Device Provisioning Service interface. On the left, there's a navigation sidebar with several sections: Overview, Activity log, Access control (IAM), SETTINGS (Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates), and MONITORING (Metrics (preview), Diagnostics settings). The 'Manage enrollments' option under SETTINGS is highlighted with a red box. The main content area has a header 'dps-certificate - Manage enrollments' with a 'Refresh' button. Below the header, there's a message: 'You can add or remove individual device enrollments and/or enrollment groups from this page'. There are two tabs: 'Enrollment Groups' (which is selected and highlighted with a red box) and 'Individual Enrollments'. Under 'Enrollment Groups', there's a search bar labeled 'Filter enrollments' and a table with a single row containing the group name 'sample-group-csharp'. The entire screenshot is framed by a red border.

## Start the device

At this point, the following setup is ready for device registration:

1. Your device or group of devices are enrolled to your Device Provisioning service, and
2. Your device is ready with the security configured and accessible through the application using the Device Provisioning Service client SDK.

Start the device to allow your client application to start the registration with your Device Provisioning service.

## Verify the device is registered

Once your device boots, the following actions should take place. See the [Provisioning Device Client Sample](#) for more details.

1. The device sends a registration request to your Device Provisioning service.
2. For TPM devices, the Device Provisioning Service sends back a registration challenge to which your device responds.
3. On successful registration, the Device Provisioning Service sends the IoT hub URI, device ID, and the encrypted key back to the device.
4. The IoT Hub client application on the device then connects to your hub.
5. On successful connection to the hub, you should see the device appear in the IoT hub's **Device Explorer**.

The screenshot shows the 'test-hub-docs - Device Explorer' window in the Azure portal. The left sidebar contains navigation links: Activity log, Access control (IAM), SETTINGS (Shared access policies, Pricing and scale, Operations monitoring, IP Filter, Properties, Locks, Automation script), and EXPLORERS (Device Explorer, Query Explorer). The main area has a search bar and tool buttons (+ Add, Columns, Refresh, Delete). A message box says: 'You can use this tool to view, create, update, and delete devices on your IoT Hub.' Below it is a 'Query' section with a query editor containing: 'SELECT \* FROM devices WHERE optional (e.g. tags.location='US')' and an 'Execute' button. At the bottom, there is a table with columns 'DEVICE ID' and 'STATUS'. One row is shown: 'test-device-docs' and 'enabled'. A 'Filter by Device Id' input field is also present.

## Next steps

In this tutorial, you learned how to:

- Enroll the device
- Start the device
- Verify the device is registered

Advance to the next tutorial to learn how to provision multiple devices across load-balanced hubs.

[Provision devices across load-balanced IoT hubs](#)

# Tutorial: Provision devices across load-balanced IoT hubs

12/10/2019 • 2 minutes to read • [Edit Online](#)

This tutorial shows how to provision devices for multiple, load-balanced IoT hubs using the Device Provisioning Service. In this tutorial, you learn how to:

- Use the Azure portal to provision a second device to a second IoT hub
- Add an enrollment list entry to the second device
- Set the Device Provisioning Service allocation policy to **even distribution**
- Link the new IoT hub to the Device Provisioning Service

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Prerequisites

This tutorial builds on the previous [Provision device to a hub](#) tutorial.

## Use the Azure portal to provision a second device to a second IoT hub

Follow the steps in the [Provision device to a hub](#) tutorial to provision a second device to another IoT hub.

## Add an enrollment list entry to the second device

The enrollment list tells the Device Provisioning Service which method of attestation (the method for confirming a device identity) it is using with the device. The next step is to add an enrollment list entry for the second device.

1. In the page for your Device Provisioning Service, click **Manage enrollments**. The **Add enrollment list entry** page appears.
2. At the top of the page, click **Add**.
3. Complete the fields and then click **Save**.

## Set the Device Provisioning Service allocation policy

The allocation policy is a Device Provisioning Service setting that determines how devices are assigned to an IoT hub. There are three supported allocation policies:

1. **Lowest latency**: Devices are provisioned to an IoT hub based on the hub with the lowest latency to the device.
2. **Evenly weighted distribution** (default): Linked IoT hubs are equally likely to have devices provisioned to them. This is the default setting. If you are provisioning devices to only one IoT hub, you can keep this setting.
3. **Static configuration via the enrollment list**: Specification of the desired IoT hub in the enrollment list takes priority over the Device Provisioning Service-level allocation policy.

Follow these steps to set the allocation policy:

1. To set the allocation policy, in the Device Provisioning Service page click **Manage allocation policy**.
2. Set the allocation policy to **Evenly weighted distribution**.
3. Click **Save**.

## Link the new IoT hub to the Device Provisioning Service

Link the Device Provisioning Service and IoT hub so that the Device Provisioning Service can register devices to that hub.

1. In the **All resources** page, click the Device Provisioning service you created previously.
2. In the Device Provisioning Service page, click **Linked IoT hubs**.
3. Click **Add**.
4. In the **Add link to IoT hub** page, use the radio buttons to specify whether the linked IoT hub is located in the current subscription, or in a different subscription. Then, choose the name of the IoT hub from the **IoT hub** box.
5. Click **Save**.

## Next steps

In this tutorial, you learned how to:

- Use the Azure portal to provision a second device to a second IoT hub
- Add an enrollment list entry to the second device
- Set the Device Provisioning Service allocation policy to **even distribution**
- Link the new IoT hub to the Device Provisioning Service

# Tutorial: Create and provision a simulated X.509 device using Java device and service SDK and group enrollments for IoT Hub Device Provisioning Service

7/30/2020 • 6 minutes to read • [Edit Online](#)

These steps show how to simulate an X.509 device on your development machine running Windows OS, and use a code sample to connect this simulated device with the Device Provisioning Service and your IoT hub using enrollment groups.

Make sure to complete the steps in the [Setup IoT Hub Device Provisioning Service with the Azure portal](#) before you proceed.

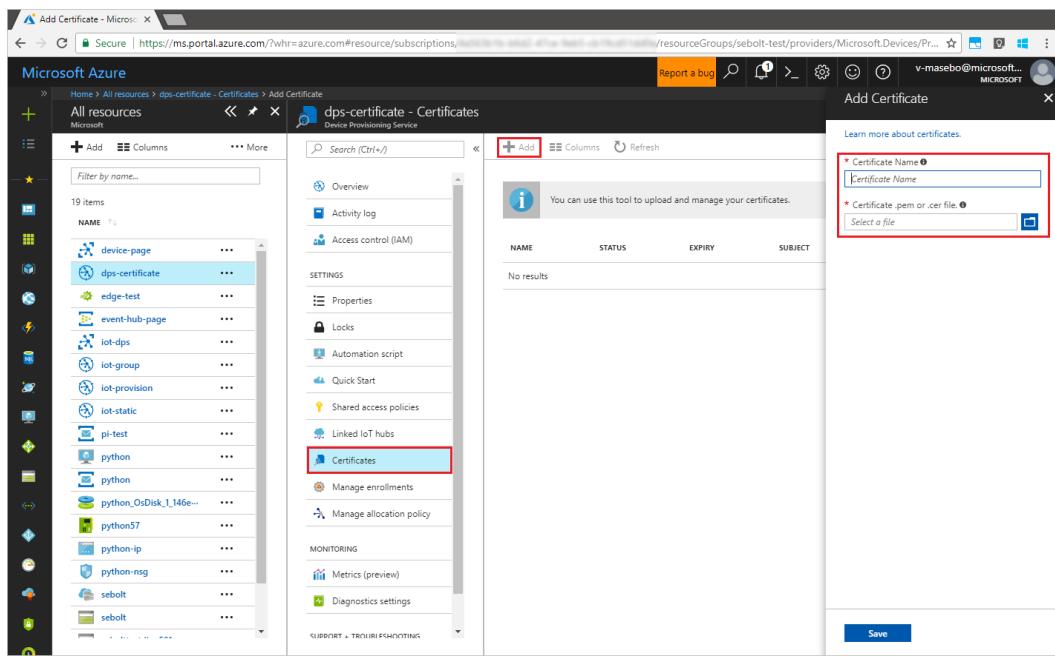
## Prepare the environment

1. Make sure you have [Java SE Development Kit 8](#) installed on your machine.
2. Download and install [Maven](#).
3. Make sure `git` is installed on your machine and is added to the environment variables accessible to the command window. See [Software Freedom Conservancy's Git client tools](#) for the latest version of `git` tools to install, which includes the **Git Bash**, the command-line app that you can use to interact with your local Git repository.
4. Use the following [Certificate Overview](#) to create your test certificates.

### NOTE

This step requires [OpenSSL](#), which can either be built and installed from source or downloaded and installed from a [3rd-party](#) such as [this](#). If you have already created your *root*, *intermediate* and *device* certificates you may skip this step.

- a. Run through the first two steps to create your *root* and *intermediate* certificates.
- b. Sign in to the Azure portal, click on the **All resources** button on the left-hand menu and open your provisioning service.
  - a. On the Device Provisioning Service summary blade, select **Certificates** and click the **Add** button at the top.
  - b. Under the **Add Certificate**, enter the following information:
    - Enter a unique certificate name.
    - Select the *RootCA.pem* file you created.
    - Once complete, click the **Save** button.



c. Select the newly created certificate:

- Click **Generate Verification Code**. Copy the code generated.
- Run the verification step. Enter the *verification code* or right-click to paste in your running PowerShell window. Press **Enter**.
- Select the newly created **verifyCert4.pem** file in the Azure portal. Click **Verify**.

**Certificate Details**

group-test

**Delete**

**Certificate Name** ?  
group-test

**ETag** ?  
AAAAAAAAG7WU=

**Subject** ?  
O=MSR\_TEST, C=US, CN=riot-root

**Expiry** ?  
Wed Dec 31 2036 16:00:00 GMT-0800 (Pacific Standard Time)

**Thumbprint** ?

**Created** ?  
Mon Dec 18 2017 12:26:32 GMT-0800 (Pacific Standard Time)

**Updated** ?  
Mon Dec 18 2017 13:19:51 GMT-0800 (Pacific Standard Time)

**Verification Code** ?

**Generate Verification Code**

**\* Verification Certificate .pem or .cer file.** ?

**Verify**

- c. Finish by running the steps to create your device certificates and clean-up resources.

**NOTE**

When creating device certificates be sure to use only lower-case alphanumerics and hyphens in your device name.

## Create a device enrollment entry

1. Open a command prompt. Clone the GitHub repo for Java SDK code samples:

```
git clone https://github.com/Azure/azure-iot-sdk-java.git --recursive
```

2. In the downloaded source code, navigate to the sample folder *azure-iot-sdk-java/provisioning/provisioning-samples/service-enrollment-group-sample*. Open the file */src/main/java/samples/com/microsoft/azure/sdk/iot/ServiceEnrollmentGroupSample.java* in an editor of your choice, and add the following details:

a. Add the [Provisioning Connection String] for your provisioning service, from the portal as following:

- a. Navigate to your provisioning service in the [Azure portal](#).
- b. Open the **Shared access policies**, and select a policy that has the *EnrollmentWrite* permission.
- c. Copy the **Primary key connection string**.

The screenshot shows the Azure portal interface for a provisioning service named 'test-doc-dps'. In the left sidebar, under 'Shared access policies', the 'provisioningserviceowner' policy is selected. The right pane displays the policy details, including its name ('provisioningserviceowner'), permissions (Service configuration, Enrollment read, Enrollment write, Registration status read, Registration status write), and shared access keys. The 'Primary key connection string' field is highlighted with a red box and contains the value 'HostName=test-doc-dps.azure-devices-provisioning.net;Share...'. The 'Secondary key connection string' field also contains a similar value.

d. In the sample code file *ServiceEnrollmentGroupSample.java*, replace the

[Provisioning Connection String] with the Primary key connection string.

```
private static final String PROVISIONING_CONNECTION_STRING = "[Provisioning Connection String]";
```

b. Open your intermediate signing certificate file in a text editor. Update the

PUBLIC\_KEY\_CERTIFICATE\_STRING value with the value of your intermediate signing certificate.

If you generated your device certificates with Bash shell, `./certs/azure-iot-test-only.intermediate.cert.pem` contains the intermediate certificate key. If your certs were generated with PowerShell, `./Intermediate1.pem` will be your intermediate certificate file.

```
private static final String PUBLIC_KEY_CERTIFICATE_STRING =
    "-----BEGIN CERTIFICATE-----\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "-----END CERTIFICATE-----\n";
```

c. Navigate to the IoT hub linked to your provisioning service in the [Azure portal](#). Open the **Overview** tab for the hub, and copy the **Hostname**. Assign this **Hostname** to the *IOTHUB\_HOST\_NAME* parameter.

```
private static final String IOTHUB_HOST_NAME = "[Host name].azure-devices.net";
```

- d. Study the sample code. It creates, updates, queries, and deletes a group enrollment for X.509 devices.

To verify successful enrollment in portal, temporarily comment out the following lines of code at the end of the *ServiceEnrollmentGroupSample.java* file:

```
// **** Delete info of enrollmentGroup
*****
System.out.println("\nDelete the enrollmentGroup...");
provisioningServiceClient.deleteEnrollmentGroup(enrollmentGroupId);
```

- e. Save the file *ServiceEnrollmentGroupSample.java*.

3. Open a command window, and navigate to the folder *azure-iot-sdk-java/provisioning/provisioning-samples/service-enrollment-group-sample*.
4. Build the sample code by using this command:

```
mvn install -DskipTests
```

5. Run the sample by using these commands at the command window:

```
cd target
java -jar ./service-enrollment-group-sample-{version}-with-deps.jar
```

6. Observe the output window for successful enrollment.

```
Command Prompt
Starting sample...
Create a new enrollmentGroup...
Add new enrollmentGroup...
EnrollmentGroup created with success...
{
  "enrollmentGroupId": "enrollmentgroupid-f1df3152-538d-4202-bc3f-955be263c4ba",
  "attestation": {
    "type": "x509",
    "x509": {
      "signingCertificates": {
        "primary": {
          "info": {
            "subjectName": "CN=microsoftiotcoreroot, O=MSR_TEST, C=US",
            "sha1Thumbprint": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
            "sha256Thumbprint": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
            "issuerName": "CN=microsoftiotcoreroot, O=MSR_TEST, C=US",
            "notBeforeUtc": "2017-01-01T00:00:00Z",
            "notAfterUtc": "3701-01-31T23:59:59Z",
            "serialNumber": "5A4B3C2D1E",
            "version": 3
          }
        }
      }
    }
  },
  "iotHubHostName": "iot-dps.azure-devices.net",
  "provisioningStatus": "enabled",
  "etag": "\"\\3e00bb24-0000-0000-0000-5a4fb7180000\""
}
Get the enrollmentGroup information...
```

7. Navigate to your provisioning service in the Azure portal. Click **Manage enrollments**. Notice that your group of X.509 devices appears under the **Enrollment Groups** tab, with an auto-generated *GROUP NAME*.

## Simulate the device

1. On the Device Provisioning Service summary blade, select **Overview** and note your *ID Scope* and *Provisioning Service Global Endpoint*.

The screenshot shows the Azure portal interface for a Device Provisioning Service named 'test-docs-dps'. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Properties, Locks, Automation script, and Quick Start. The main area displays details about the resource group 'test-rg-dps', which is active and located in East US. It shows the service endpoint 'test-docs-dps.azure-devices-provisioning.net' and the global device endpoint 'global.azure-devices-provisioning.net'. A red box highlights the 'Global device endpoint' field. Other visible details include the ID Scope (redacted), Pricing and scale tier S1, and a Subscription ID (redacted). At the bottom right, there's a link to 'Azure IoT Hub Device Provisioning Service Documentation'.

2. Open a command prompt. Navigate to the sample project folder.

```
cd azure-iot-sdk-java/provisioning/provisioning-samples/provisioning-X509-sample
```

3. Edit `/src/main/java/samples/com/microsoft/azure/sdk/iot/ProvisioningX509Sample.java` to include your *ID Scope* and *Provisioning Service Global Endpoint* that you noted previously.

```
private static final String idScope = "[Your ID scope here]";
private static final String globalEndpoint = "[Your Provisioning Service Global Endpoint here]";
private static final ProvisioningDeviceClientTransportProtocol
PROVISIONING_DEVICE_CLIENT_TRANSPORT_PROTOCOL = ProvisioningDeviceClientTransportProtocol.HTTPS;
private static final int MAX_TIME_TO_WAIT_FOR_REGISTRATION = 10000; // in milli seconds
private static final String leafPublicPem = "<Your Public PEM Certificate here>";
private static final String leafPrivateKey = "<Your Private PEM Key here>";
```

4. Update the `leafPublicPem` and `leafPrivateKey` variables with your public and private device certificates.

If you generated your device certificates with PowerShell, the files mydevice\* contain the public key, private key, and PFX for the device.

If you generated your device certificates with Bash shell, ./certs/new-device.cert.pem contains the public key. The device's private key will be in the ./private/new-device.key.pem file.

Open your public key file and update the `leafPublicPem` variable with that value. Copy the text from -----  
*BEGIN PRIVATE KEY*----- to -----*END PRIVATE KEY*-----.

```
private static final String leafPublicPem = "-----BEGIN CERTIFICATE-----\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
"-----END CERTIFICATE-----\n";
```

Open your private key file and update the `leafPrivateKey` variable with that value. Copy the text from -----  
*BEGIN RSA PRIVATE KEY*----- to -----*END RSA PRIVATE KEY*-----.

```

private static final String leafPrivateKey = "-----BEGIN RSA PRIVATE KEY-----\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "-----END RSA PRIVATE KEY-----\n";

```

- Add a new variable just below `leafPrivateKey` for your intermediate certificate. Name this new variable `intermediateKey`. Give it the value of your intermediate signing certificate.

If you generated your device certificates with Bash shell, `./certs/azure-iot-test-only.intermediate.cert.pem` contains the intermediate certificate key. If your certs were generated with PowerShell, `./Intermediate1.pem` will be your intermediate certificate file.

```

private static final String intermediateKey = "-----BEGIN CERTIFICATE-----\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX\n" +
    "-----END CERTIFICATE-----\n";

```

- In the `main` function, add the `intermediateKey` to the `signerCertificates` collection before the initialization of `securityProviderX509`.

```

public static void main(String[] args) throws Exception
{
    ...

    try
    {
        ProvisioningStatus provisioningStatus = new ProvisioningStatus();

        // Add intermediate certificate as part of the certificate key chain.
        signerCertificates.add(intermediateKey);

        SecurityProvider securityProviderX509 = new SecurityProviderX509Cert(leafPublicPem,
            leafPrivateKey, signerCertificates);
    }
}

```

- Save your changes and build the sample. Navigate to the target folder and execute the created jar file.

```

mvn clean install
cd target
java -jar ./provisioning-x509-sample-{version}-with-deps.jar

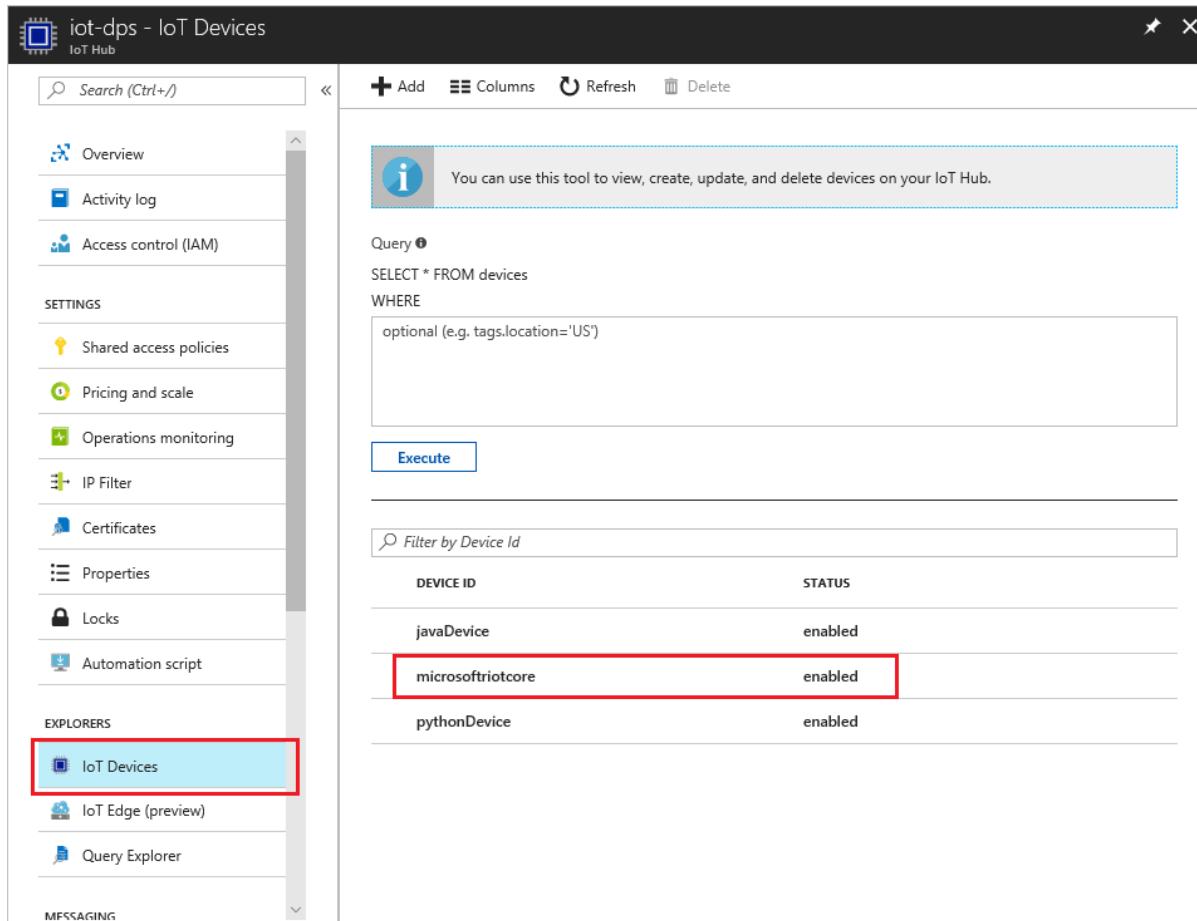
```

```

Starting...
Beginning setup.
Waiting for Provisioning Service to register
Waiting for Provisioning Service to register
IotHub Uri : iot-dps.azure-devices.net
Device ID : microsoftrioticore
log4j:WARN No appenders could be found for logger (com.microsoft.azure.sdk.iot.device.IotHubConnectionString).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
Sending message from device to IoT Hub...
Press any key to exit...
Message received!

```

8. In the portal, navigate to the IoT hub linked to your provisioning service and open the **Device Explorer** blade. On successful provisioning of the simulated X.509 device to the hub, its device ID appears on the **Device Explorer** blade, with *STATUS* as **enabled**. Note that you might need to click the **Refresh** button at the top if you already opened the blade prior to running the sample device application.



The screenshot shows the Azure IoT Devices blade for the hub 'iot-dps'. The left sidebar has a 'IoT Devices' section highlighted with a red box. The main area shows a query editor with a SELECT statement and a table of devices. One device, 'microsoftriotcore', is highlighted with a red box in the table.

DEVICE ID	STATUS
javaDevice	enabled
microsoftriotcore	enabled
pythonDevice	enabled

## Clean up resources

If you plan to continue working on and exploring the device client sample, do not clean up the resources created in this Quickstart. If you do not plan to continue, use the following steps to delete all resources created by this Quickstart.

1. Close the device client sample output window on your machine.
2. From the left-hand menu in the Azure portal, click **All resources** and then select your Device Provisioning service. Open the **Manage Enrollments** blade for your service, and then click the **Individual Enrollments** tab. Select the *REGISTRATION ID* of the device you enrolled in this Quickstart, and click the **Delete** button at the top.
3. From the left-hand menu in the Azure portal, click **All resources** and then select your IoT hub. Open the **IoT Devices** blade for your hub, select the *DEVICE ID* of the device you registered in this Quickstart, and then click **Delete** button at the top.

## Next steps

In this tutorial, you've created a simulated X.509 device on your Windows machine and provisioned it to your IoT hub using the Azure IoT Hub Device Provisioning Service and enrollment groups. To learn more about your X.509 device, continue to device concepts.

[IoT Hub Device Provisioning Service device concepts](#)

# Auto-provisioning concepts

7/30/2020 • 8 minutes to read • [Edit Online](#)

As described in the [Overview](#), the Device Provisioning Service is a helper service that enables just-in-time provisioning of devices to an IoT hub, without requiring human intervention. After successful provisioning, devices connect directly with their designated IoT Hub. This process is referred to as auto-provisioning, and provides an out-of-the-box registration and initial configuration experience for devices.

## Overview

Azure IoT auto-provisioning can be broken into three phases:

1. **Service configuration** - a one-time configuration of the Azure IoT Hub and IoT Hub Device Provisioning Service instances, establishing them and creating linkage between them.

### NOTE

Regardless of the size of your IoT solution, even if you plan to support millions of devices, this is a **one-time configuration**.

2. **Device enrollment** - the process of making the Device Provisioning Service instance aware of the devices that will attempt to register in the future. [Enrollment](#) is accomplished by configuring device identity information in the provisioning service, as either an "individual enrollment" for a single device, or a "group enrollment" for multiple devices. Identity is based on the [attestation mechanism](#) the device is designed to use, which allows the provisioning service to attest to the device's authenticity during registration:

- **TPM**: configured as an "individual enrollment", the device identity is based on the TPM registration ID and the public endorsement key. Given that TPM is a [specification](#), the service only expects to attest per the specification, regardless of TPM implementation (hardware or software). See [Device provisioning: Identity attestation with TPM](#) for details on TPM-based attestation.
- **X509**: configured as either an "individual enrollment" or "group enrollment", the device identity is based on an X.509 digital certificate, which is uploaded to the enrollment as a .pem or .cer file.

### IMPORTANT

Although not a prerequisite for using Device Provisioning Services, we strongly recommend that your device use a Hardware Security Module (HSM) to store sensitive device identity information, such as keys and X.509 certificates.

3. **Device registration and configuration** - initiated upon boot up by registration software, which is built using a Device Provisioning Service client SDK appropriate for the device and attestation mechanism. The software establishes a connection to the provisioning service for authentication of the device, and subsequent registration in the IoT Hub. Upon successful registration, the device is provided with its IoT Hub unique device ID and connection information, allowing it to pull its initial configuration and begin the telemetry process. In production environments, this phase can occur weeks or months after the previous two phases.

## Roles and operations

The phases discussed in the previous section can span weeks or months, due to production realities like manufacturing time, shipping, customs process, etc. In addition, they can span activities across multiple roles given the various entities involved. This section takes a deeper look at the various roles and operations related to each phase, then illustrates the flow in a sequence diagram.

Auto-provisioning also places requirements on the device manufacturer, specific to enabling the attestation mechanism. Manufacturing operations can also occur independent of the timing of auto-provisioning phases, especially in cases where new devices are procured after auto-provisioning has already been established.

A series of Quickstarts are provided in the table of contents to the left, to help explain auto-provisioning through hands-on experience. In order to facilitate/simplify the learning process, software is used to simulate a physical device for enrollment and registration. Some Quickstarts require you to fulfill operations for multiple roles, including operations for non-existent roles, due to the simulated nature of the Quickstarts.

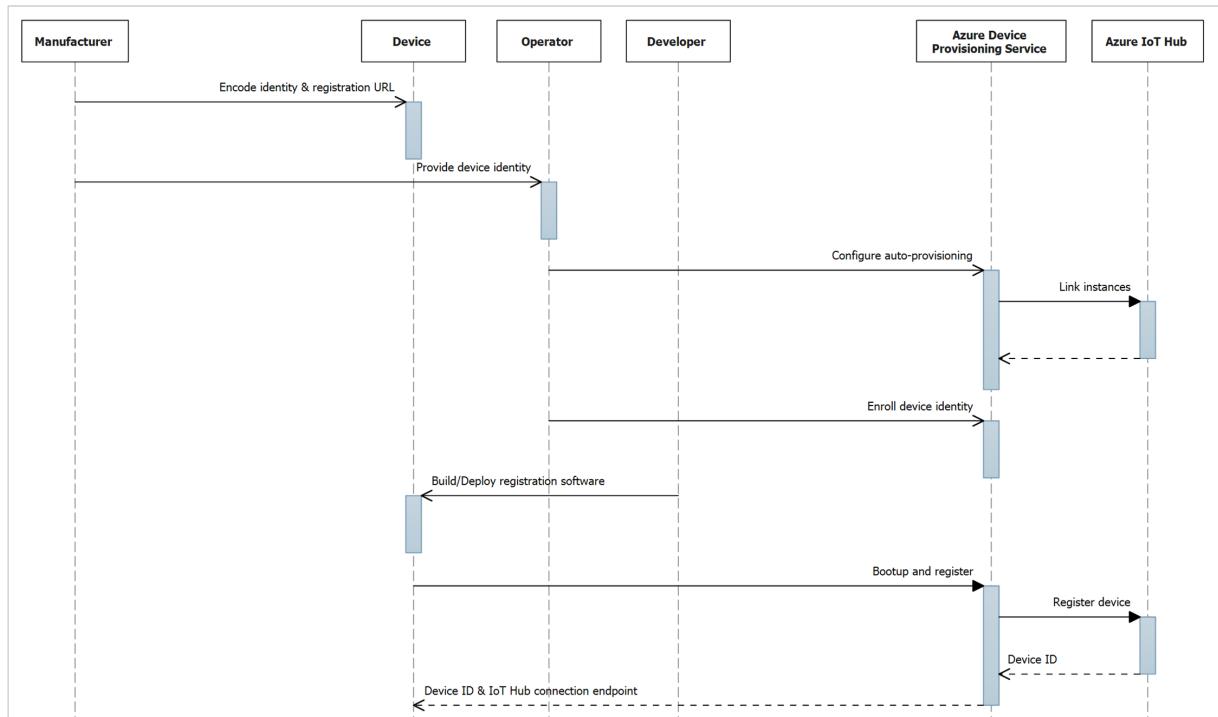
ROLE	OPERATION	DESCRIPTION
Manufacturer	Encode identity and registration URL	<p>Based on the attestation mechanism used, the manufacturer is responsible for encoding the device identity info, and the Device Provisioning Service registration URL.</p> <p><b>Quickstarts:</b> since the device is simulated, there is no Manufacturer role. See the Developer role for details on how you get this information, which is used in coding a sample registration application.</p>
	Provide device identity	<p>As the originator of the device identity info, the manufacturer is responsible for communicating it to the operator (or a designated agent), or directly enrolling it to the Device Provisioning Service via APIs.</p> <p><b>Quickstarts:</b> since the device is simulated, there is no Manufacturer role. See the Operator role for details on how you get the device identity, which is used to enroll a simulated device in your Device Provisioning Service instance.</p>
Operator	Configure auto-provisioning	<p>This operation corresponds with the first phase of auto-provisioning.</p> <p><b>Quickstarts:</b> You perform the Operator role, configuring the Device Provisioning Service and IoT Hub instances in your Azure subscription.</p>

ROLE	OPERATION	DESCRIPTION
	Enroll device identity	<p>This operation corresponds with the second phase of auto-provisioning.</p> <p><b>Quickstarts:</b> You perform the Operator role, enrolling your simulated device in your Device Provisioning Service instance. The device identity is determined by the attestation method being simulated in the Quickstart (TPM or X.509). See the Developer role for attestation details.</p>
Device Provisioning Service, IoT Hub	<all operations>	<p>For both a production implementation with physical devices, and Quickstarts with simulated devices, these roles are fulfilled via the IoT services you configure in your Azure subscription. The roles/operations function exactly the same, as the IoT services are indifferent to provisioning of physical vs. simulated devices.</p>

ROLE	OPERATION	DESCRIPTION
Developer	Build/Deploy registration software	<p>This operation corresponds with the third phase of auto-provisioning. The Developer is responsible for building and deploying the registration software to the device, using the appropriate SDK.</p> <p><b>Quickstarts:</b> The sample registration application you build simulates a real device, for your platform/language of choice, which runs on your workstation (instead of deploying it to a physical device). The registration application performs the same operations as one deployed to a physical device. You specify the attestation method (TPM or X.509 certificate), plus the registration URL and "ID Scope" of your Device Provisioning Service instance. The device identity is determined by the SDK attestation logic at runtime, based on the method you specify:</p> <ul style="list-style-type: none"> <li>• <b>TPM attestation</b> - your development workstation runs a <a href="#">TPM simulator application</a>. Once running, a separate application is used to extract the TPM's "Endorsement Key" and "Registration ID" for use in enrolling the device identity. The SDK attestation logic also uses the simulator during registration, to present a signed SAS token for authentication and enrollment verification.</li> <li>• <b>X509 attestation</b> - you use a tool to <a href="#">generate a certificate</a>. Once generated, you create the certificate file required for use in enrollment. The SDK attestation logic also uses the certificate during registration, to present for authentication and enrollment verification.</li> </ul>

ROLE	OPERATION	DESCRIPTION
Device	Bootup and register	<p>This operation corresponds with the third phase of auto-provisioning, fulfilled by the device registration software built by the Developer. See the Developer role for details. Upon first boot:</p> <ol style="list-style-type: none"> <li>1. The application connects with the Device Provisioning Service instance, per the global URL and service "ID Scope" specified during development.</li> <li>2. Once connected, the device is authenticated against the attestation method and identity specified during enrollment.</li> <li>3. Once authenticated, the device is registered with the IoT Hub instance specified by the provisioning service instance.</li> <li>4. Upon successful registration, a unique device ID and IoT Hub endpoint are returned to the registration application for communicating with IoT Hub.</li> <li>5. From there, the device can pull down its initial <a href="#">device twin</a> state for configuration, and begin the process of reporting telemetry data.</li> </ol> <p><b>Quickstarts:</b> since the device is simulated, the registration software runs on your development workstation.</p>

The following diagram summarizes the roles and sequencing of operations during device auto-provisioning:



#### **NOTE**

Optionally, the manufacturer can also perform the "Enroll device identity" operation using Device Provisioning Service APIs (instead of via the Operator). For a detailed discussion of this sequencing and more, see the [Zero touch device registration with Azure IoT video](#) (starting at marker 41:00)

## Roles and Azure accounts

How each role is mapped to an Azure account is scenario-dependent, and there are quite a few scenarios that can be involved. The common patterns below should help provide a general understanding regarding how roles are generally mapped to an Azure account.

### **Chip manufacturer provides security services**

In this scenario, the manufacturer manages security for level-one customers. This scenario may be preferred by these level-one customers as they don't have to manage detailed security.

The manufacturer introduces security into Hardware Security Modules (HSMs). This security can include the manufacturer obtaining keys, certificates, etc. from potential customers who already have DPS instances and enrollment groups setup. The manufacturer could also generate this security information for its customers.

In this scenario, there may be two Azure accounts involved:

- **Account #1:** Likely shared across the operator and developer roles to some degree. This party may purchase the HSM chips from the manufacturer. These chips are pointed to DPS instances associated with the Account #1. With DPS enrollments, this party can lease devices to multiple level-two customers by reconfiguring the device enrollment settings in DPS. This party may also have IoT hubs allocated for end-user backend systems to interface with in order to access device telemetry etc. In this latter case, a second account may not be needed.
- **Account #2:** End users, level-two customers may have their own IoT hubs. The party associated with Account #1 just points leased devices to the correct hub in this account. This configuration requires linking DPS and IoT hubs across Azure accounts, which can be done with Azure Resource Manager templates.

### **All-in-one OEM**

The manufacturer could be an "All-in-one OEM" where only a single manufacturer account would be needed. The manufacturer handles security and provisioning end to end.

The manufacturer may provide a cloud-based application to customers who purchase devices. This application would interface with the IoT Hub allocated by the manufacturer.

Vending machines or automated coffee machines represent examples for this scenario.

## Next steps

You may find it helpful to bookmark this article as a point of reference, as you work your way through the corresponding auto-provisioning Quickstarts.

Begin by completing a "Set up auto-provisioning" Quickstart that best suits your management tool preference, which walks you through the "Service configuration" phase:

- [Set up auto-provisioning using Azure CLI](#)
- [Set up auto-provisioning using the Azure portal](#)
- [Set up auto-provisioning using a Resource Manager template](#)

Then continue with an "Auto-provision a simulated device" Quickstart that suits your device attestation

mechanism and Device Provisioning Service SDK/language preference. In this Quickstart, you walk through the "Device enrollment" and "Device registration and configuration" phases:

SIMULATED DEVICE ATTESTATION MECHANISM	QUICKSTART SDK/LANGUAGE
Trusted Platform Module (TPM)	<a href="#">C</a> <a href="#">Java</a> <a href="#">C#</a> <a href="#">Python</a>
X.509 certificate	<a href="#">C</a> <a href="#">Java</a> <a href="#">C#</a> <a href="#">Nodejs</a> <a href="#">Python</a>

# IoT Hub Device reprovisioning concepts

12/10/2019 • 4 minutes to read • [Edit Online](#)

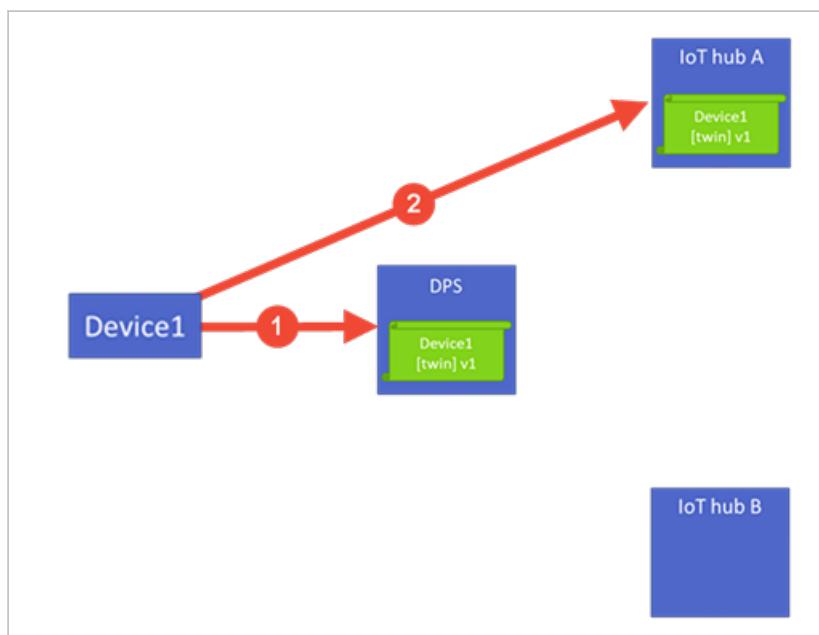
During the lifecycle of an IoT solution, it's common to move devices between IoT hubs. The reasons for this move may include the following scenarios:

- **Geolocation / GeoLatency:** As a device moves between locations, network latency is improved by having the device migrated to a closer IoT hub.
- **Multi-tenancy:** A device may be used within the same IoT solution and reassigned to a new customer, or customer site. This new customer may be serviced using a different IoT hub.
- **Solution change:** A device could be moved into a new or updated IoT solution. This reassignment may require the device to communicate with a new IoT hub that's connected to other back-end components.
- **Quarantine:** Similar to a solution change. A device that's malfunctioning, compromised, or out-of-date may be reassigned to an IoT hub that can only update and get back in compliance. Once the device is functioning properly, it's then migrated back to its main hub.

Reprovisioning support within the Device Provisioning Service addresses these needs. Devices can be automatically reassigned to new IoT hubs based on the reprovisioning policy that's configured on the device's enrollment entry.

## Device state data

Device state data is composed of the [device twin](#) and device capabilities. This data is stored in the Device Provisioning Service instance and the IoT hub that a device is assigned to.

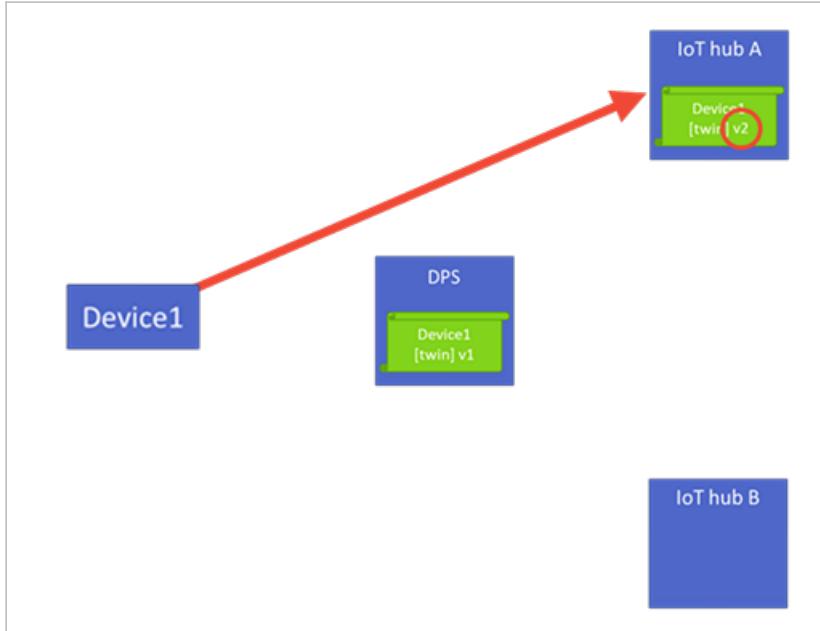


When a device is initially provisioned with a Device Provisioning Service instance, the following steps are done:

1. The device sends a provisioning request to a Device Provisioning Service instance. The service instance authenticates the device identity based on an enrollment entry, and creates the initial configuration of the device state data. The service instance assigns the device to an IoT hub based on the enrollment configuration and returns that IoT hub assignment to the device.

2. The provisioning service instance gives a copy of any initial device state data to the assigned IoT hub. The device connects to the assigned IoT hub and begins operations.

Over time, the device state data on the IoT hub may be updated by [device operations](#) and [back-end operations](#). The initial device state information stored in the Device Provisioning Service instance stays untouched. This untouched device state data is the initial configuration.

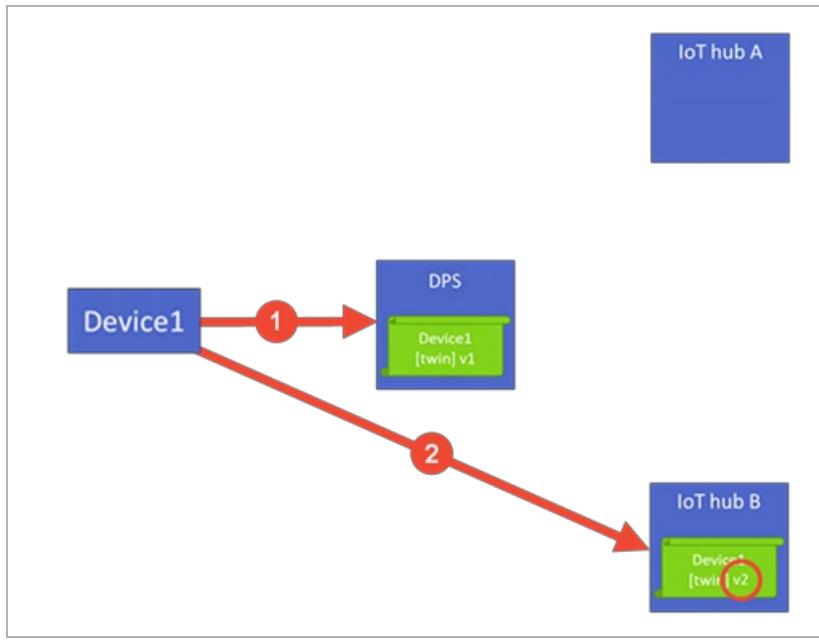


Depending on the scenario, as a device moves between IoT hubs, it may also be necessary to migrate device state updated on the previous IoT hub over to the new IoT hub. This migration is supported by reprovisioning policies in the Device Provisioning Service.

## Reprovisioning policies

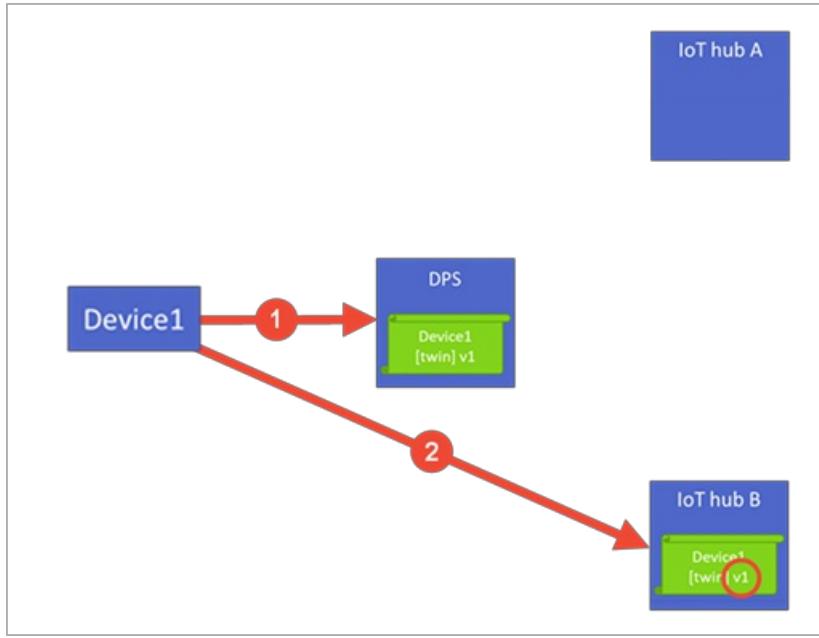
Depending on the scenario, a device usually sends a request to a provisioning service instance on reboot. It also supports a method to manually trigger provisioning on demand. The reprovisioning policy on an enrollment entry determines how the device provisioning service instance handles these provisioning requests. The policy also determines whether device state data should be migrated during reprovisioning. The same policies are available for individual enrollments and enrollment groups:

- **Re-provision and migrate data:** This policy is the default for new enrollment entries. This policy takes action when devices associated with the enrollment entry submit a new request (1). Depending on the enrollment entry configuration, the device may be reassigned to another IoT hub. If the device is changing IoT hubs, the device registration with the initial IoT hub will be removed. The updated device state information from that initial IoT hub will be migrated over to the new IoT hub (2). During migration, the device's status will be reported as **Assigning**.



- **Re-provision and reset to initial config:** This policy takes action when devices associated with the enrollment entry submit a new provisioning request (1). Depending on the enrollment entry configuration, the device may be reassigned to another IoT hub. If the device is changing IoT hubs, the device registration with the initial IoT hub will be removed. The initial configuration data that the provisioning service instance received when the device was provisioned is provided to the new IoT hub (2). During migration, the device's status will be reported as **Assigning**.

This policy is often used for a factory reset without changing IoT hubs.



- **Never re-provision:** The device is never reassigned to a different hub. This policy is provided for managing backwards compatibility.

### Managing backwards compatibility

Before September 2018, device assignments to IoT hubs had a sticky behavior. When a device went back through the provisioning process, it would only be assigned back to the same IoT hub.

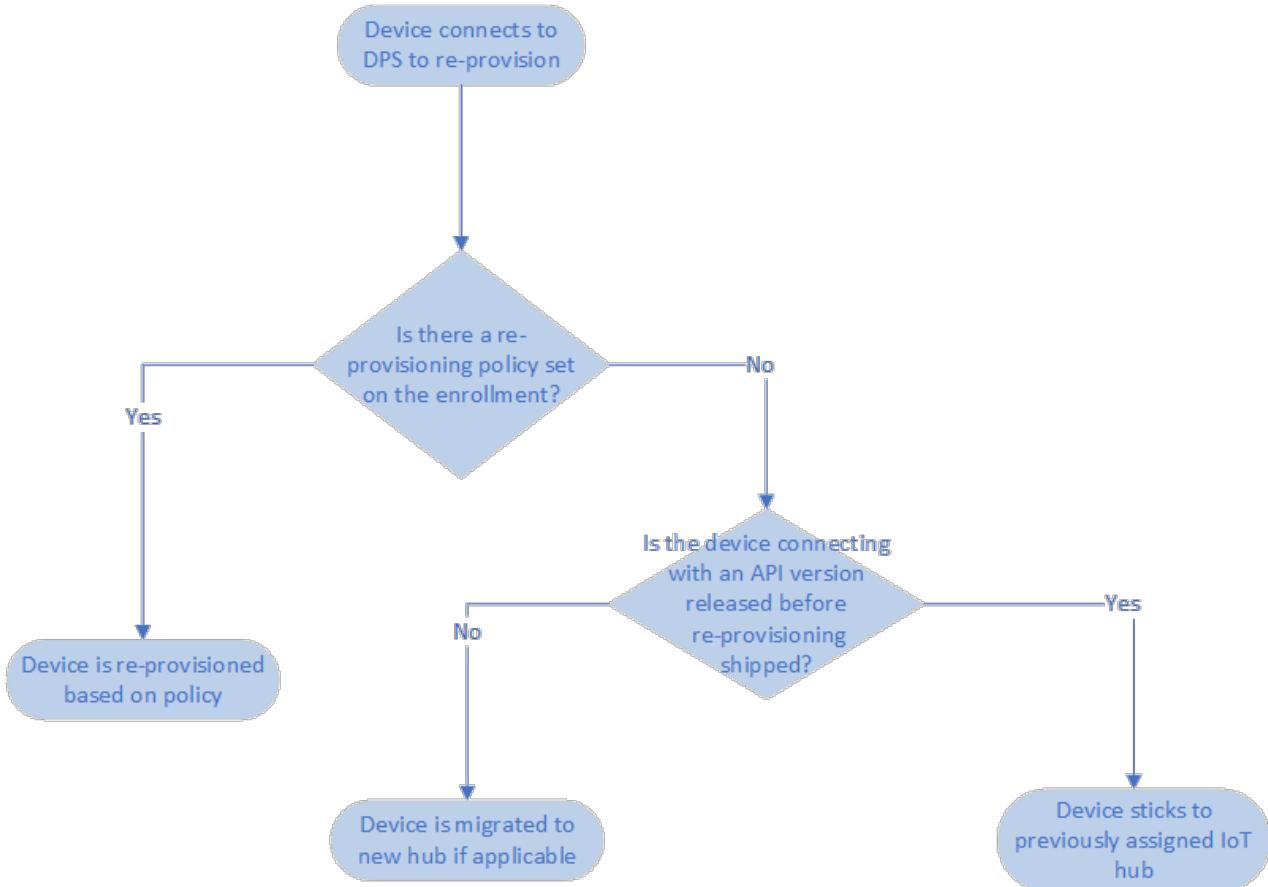
For solutions that have taken a dependency on this behavior, the provisioning service includes backwards compatibility. This behavior is presently maintained for devices according to the following criteria:

1. The devices connect with an API version before the availability of native reprovisioning support in the Device Provisioning Service. Refer to the API table below.

- The enrollment entry for the devices doesn't have a reprovisioning policy set on them.

This compatibility makes sure that previously deployed devices experience the same behavior that's present during initial testing. To preserve the previous behavior, don't save a reprovisioning policy to these enrollments. If a reprovisioning policy is set, the reprovisioning policy takes precedence over the behavior. By allowing the reprovisioning policy to take precedence, customers can update device behavior without having to reimagine the device.

The following flow chart helps to show when the behavior is present:



The following table shows the API versions before the availability of native reprovisioning support in the Device Provisioning Service:

REST API	C SDK	PYTHON SDK	NODE SDK	JAVA SDK	.NET SDK
2018-04-01 and earlier	1.2.8 and earlier	1.4.2 and earlier	1.7.3 or earlier	1.13.0 or earlier	1.1.0 or earlier

#### NOTE

These values and links are likely to change. This is only a placeholder attempt to determine where the versions can be determined by a customer and what the expected versions will be.

## Next steps

- How to reprovision devices

# IoT Hub Device Provisioning Service device concepts

12/10/2019 • 2 minutes to read • [Edit Online](#)

IoT Hub Device Provisioning Service is a helper service for IoT Hub that you use to configure zero-touch device provisioning to a specified IoT hub. With the Device Provisioning Service, you can provision millions of devices in a secure and scalable manner.

This article gives an overview of the *device* concepts involved in device provisioning. This article is most relevant to personas involved in the [manufacturing step](#) of getting a device ready for deployment.

## Attestation mechanism

The attestation mechanism is the method used for confirming a device's identity. The attestation mechanism is also relevant to the enrollment list, which tells the provisioning service which method of attestation to use with a given device.

### NOTE

IoT Hub uses "authentication scheme" for a similar concept in that service.

The Device Provisioning Service supports the following forms of attestation:

- **X.509 certificates** based on the standard X.509 certificate authentication flow.
- **Trusted Platform Module (TPM)** based on a nonce challenge, using the TPM standard for keys to present a signed Shared Access Signature (SAS) token. This does not require a physical TPM on the device, but the service expects to attest using the endorsement key per the [TPM spec](#).
- **Symmetric Key** based on shared access signature (SAS) [Security tokens](#), which include a hashed signature and an embedded expiration. For more information, see [Symmetric key attestation](#).

## Hardware security module

The hardware security module, or HSM, is used for secure, hardware-based storage of device secrets, and is the most secure form of secret storage. Both X.509 certificates and SAS tokens can be stored in the HSM. HSMs can be used with both attestation mechanisms the provisioning service supports.

### TIP

We strongly recommend using an HSM with devices to securely store secrets on your devices.

Device secrets may also be stored in software (memory), but it is a less secure form of storage than an HSM.

## Registration ID

The registration ID is used to uniquely identify a device in the Device Provisioning Service. The device ID must be unique in the provisioning service [ID scope](#). Each device must have a registration ID. The registration ID is alphanumeric, case insensitive, and may contain special characters including colon, period, underscore and hyphen.

- In the case of TPM, the registration ID is provided by the TPM itself.
- In the case of X.509-based attestation, the registration ID is provided as the subject name of the certificate.

## Device ID

The device ID is the ID as it appears in IoT Hub. The desired device ID may be set in the enrollment entry, but it is not required to be set. Setting the desired device ID is only supported in individual enrollments. If no desired device ID is specified in the enrollment list, the registration ID is used as the device ID when registering the device. Learn more about [device IDs in IoT Hub](#).

## ID scope

The ID scope is assigned to a Device Provisioning Service when it is created by the user and is used to uniquely identify the specific provisioning service the device will register through. The ID scope is generated by the service and is immutable, which guarantees uniqueness.

**NOTE**

Uniqueness is important for long-running deployment operations and merger and acquisition scenarios.

# IoT Hub Device Provisioning Service security concepts

12/10/2019 • 6 minutes to read • [Edit Online](#)

IoT Hub Device Provisioning Service is a helper service for IoT Hub that you use to configure zero-touch device provisioning to a specified IoT hub. With the Device Provisioning Service, you can [auto-provision](#) millions of devices in a secure and scalable manner. This article gives an overview of the *security* concepts involved in device provisioning. This article is relevant to all personas involved in getting a device ready for deployment.

## Attestation mechanism

The attestation mechanism is the method used for confirming a device's identity. The attestation mechanism is also relevant to the enrollment list, which tells the provisioning service which method of attestation to use with a given device.

### NOTE

IoT Hub uses "authentication scheme" for a similar concept in that service.

Device Provisioning Service supports the following forms of attestation:

- **X.509 certificates** based on the standard X.509 certificate authentication flow.
- **Trusted Platform Module (TPM)** based on a nonce challenge, using the TPM standard for keys to present a signed Shared Access Signature (SAS) token. This form of attestation does not require a physical TPM on the device, but the service expects to attest using the endorsement key per the [TPM spec](#).
- **Symmetric Key** based on shared access signature (SAS) [Security tokens](#), which include a hashed signature and an embedded expiration. For more information, see [Symmetric key attestation](#).

## Hardware security module

The hardware security module, or HSM, is used for secure, hardware-based storage of device secrets, and is the most secure form of secret storage. Both X.509 certificates and SAS tokens can be stored in the HSM. HSMs can be used with both attestation mechanisms the provisioning supports.

### TIP

We strongly recommend using an HSM with devices to securely store secrets on your devices.

Device secrets may also be stored in software (memory), but it is a less secure form of storage than an HSM.

## Trusted Platform Module

TPM can refer to a standard for securely storing keys used to authenticate the platform, or it can refer to the I/O interface used to interact with the modules implementing the standard. TPMs can exist as discrete hardware, integrated hardware, firmware-based, or software-based. Learn more about [TPMs and TPM attestation](#). Device Provisioning Service only supports TPM 2.0.

TPM attestation is based on a nonce challenge, which uses the endorsement and storage root keys to present a signed Shared Access Signature (SAS) token.

## **Endorsement key**

The endorsement key is an asymmetric key contained inside the TPM, which was internally generated or injected at manufacturing time and is unique for every TPM. The endorsement key cannot be changed or removed. The private portion of the endorsement key is never released outside of the TPM, while the public portion of the endorsement key is used to recognize a genuine TPM. Learn more about the [endorsement key](#).

## **Storage root key**

The storage root key is stored in the TPM and is used to protect TPM keys created by applications, so that these keys cannot be used without the TPM. The storage root key is generated when you take ownership of the TPM; when you clear the TPM so a new user can take ownership, a new storage root key is generated. Learn more about the [storage root key](#).

# X.509 certificates

Using X.509 certificates as an attestation mechanism is an excellent way to scale production and simplify device provisioning. X.509 certificates are typically arranged in a certificate chain of trust in which each certificate in the chain is signed by the private key of the next higher certificate, and so on, terminating in a self-signed root certificate. This arrangement establishes a delegated chain of trust from the root certificate generated by a trusted root certificate authority (CA) down through each intermediate CA to the end-entity "leaf" certificate installed on a device. To learn more, see [Device Authentication using X.509 CA Certificates](#).

Often the certificate chain represents some logical or physical hierarchy associated with devices. For example, a manufacturer may:

- issue a self-signed root CA certificate
- use the root certificate to generate a unique intermediate CA certificate for each factory
- use each factory's certificate to generate a unique intermediate CA certificate for each production line in the plant
- and finally use the production line certificate, to generate a unique device (end-entity) certificate for each device manufactured on the line.

To learn more, see [Conceptual understanding of X.509 CA certificates in the IoT industry](#).

## **Root certificate**

A root certificate is a self-signed X.509 certificate representing a certificate authority (CA). It is the terminus, or trust anchor, of the certificate chain. Root certificates can be self-issued by an organization or purchased from a root certificate authority. To learn more, see [Get X.509 CA certificates](#). The root certificate can also be referred to as a root CA certificate.

## **Intermediate certificate**

An intermediate certificate is an X.509 certificate, which has been signed by the root certificate (or by another intermediate certificate with the root certificate in its chain). The last intermediate certificate in a chain is used to sign the leaf certificate. An intermediate certificate can also be referred to as an intermediate CA certificate.

## **End-entity "leaf" certificate**

The leaf certificate, or end-entity certificate, identifies the certificate holder. It has the root certificate in its certificate chain as well as zero or more intermediate certificates. The leaf certificate is not used to sign any other certificates. It uniquely identifies the device to the provisioning service and is sometimes referred to as the device certificate. During authentication, the device uses the private key associated with this certificate to respond to a proof of possession challenge from the service.

Leaf certificates used with an [Individual enrollment](#) entry have a requirement that the **Subject Name** must be set to the registration ID of the Individual Enrollment entry. Leaf certificates used with an [Enrollment group](#) entry should have the **Subject Name** set to the desired device ID which will be shown in the [Registration Records](#) for

the authenticated device in the enrollment group.

To learn more, see [Authenticating devices signed with X.509 CA certificates](#).

## Controlling device access to the provisioning service with X.509 certificates

The provisioning service exposes two types of enrollment entry that you can use to control access for devices that use the X.509 attestation mechanism:

- **Individual enrollment** entries are configured with the device certificate associated with a specific device. These entries control enrollments for specific devices.
- **Enrollment group** entries are associated with a specific intermediate or root CA certificate. These entries control enrollments for all devices that have that intermediate or root certificate in their certificate chain.

When a device connects to the provisioning service, the service prioritizes more specific enrollment entries over less specific enrollment entries. That is, if an individual enrollment for the device exists, the provisioning service applies that entry. If there is no individual enrollment for the device and an enrollment group for the first intermediate certificate in the device's certificate chain exists, the service applies that entry, and so on, up the chain to the root. The service applies the first applicable entry that it finds, such that:

- If the first enrollment entry found is enabled, the service provisions the device.
- If the first enrollment entry found is disabled, the service does not provision the device.
- If no enrollment entry is found for any of the certificates in the device's certificate chain, the service does not provision the device.

This mechanism and the hierarchical structure of certificate chains provides powerful flexibility in how you can control access for individual devices as well as for groups of devices. For example, imagine five devices with the following certificate chains:

- *Device 1*: root certificate -> certificate A -> device 1 certificate
- *Device 2*: root certificate -> certificate A -> device 2 certificate
- *Device 3*: root certificate -> certificate A -> device 3 certificate
- *Device 4*: root certificate -> certificate B -> device 4 certificate
- *Device 5*: root certificate -> certificate B -> device 5 certificate

Initially, you can create a single enabled group enrollment entry for the root certificate to enable access for all five devices. If certificate B later becomes compromised, you can create a disabled enrollment group entry for certificate B to prevent *Device 4* and *Device 5* from enrolling. If still later *Device 3* becomes compromised, you can create a disabled individual enrollment entry for its certificate. This revokes access for *Device 3*, but still allows *Device 1* and *Device 2* to enroll.

# TLS support in Azure IoT Hub Device Provisioning Service (DPS)

7/30/2020 • 3 minutes to read • [Edit Online](#)

DPS uses [Transport Layer Security \(TLS\)](#) to secure connections from IoT devices.

Current TLS protocol versions supported by DPS are:

- TLS 1.2

TLS 1.0 and 1.1 are considered legacy and are planned for deprecation. For more information, see [Deprecating TLS 1.0 and 1.1 for IoT Hub](#).

## Restrict connections to TLS 1.2

For added security, it is advised to configure your DPS instances to *only* allow device client connections that use TLS version 1.2 and to enforce the use of [recommended ciphers](#).

To do this, provision a new DPS resource setting the `minTlsVersion` property to `1.2` in your Azure Resource Manager template's DPS resource specification. The following example template JSON specifies the `minTlsVersion` property for a new DPS instance.

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "resources": [  
        {  
            "type": "Microsoft.Devices/ProvisioningServices",  
            "apiVersion": "2020-01-01",  
            "name": "<provide-a-valid-DPS-resource-name>",  
            "location": "<any-region>",  
            "properties": {  
                "minTlsVersion": "1.2"  
            },  
            "sku": {  
                "name": "S1",  
                "capacity": 1  
            },  
        }  
    ]  
}
```

You can deploy the template with the following Azure CLI command.

```
az deployment group create -g <your resource group name> --template-file template.json
```

For more information on creating DPS resources with Resource Manager templates, see, [Set up DPS with an Azure Resource Manager template](#).

The DPS resource created using this configuration will refuse devices that attempt to connect using TLS versions 1.0 and 1.1. Similarly, the TLS handshake will be refused if the device client's HELLO message does not list any of the [recommended ciphers](#).

**NOTE**

The `minTlsVersion` property is read-only and cannot be changed once your DPS resource is created. It is therefore essential that you properly test and validate that *all* your IoT devices are compatible with TLS 1.2 and the [recommended ciphers](#) in advance.

**NOTE**

Upon failovers, the `minTlsVersion` property of your DPS will remain effective in the geo-paired region post-failover.

## Recommended ciphers

DPS instances that are configured to accept only TLS 1.2 will also enforce the use of the following cipher suites:

### TLS 1.2 cipher suites

**MINIMUM BAR**

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384

**OPPORTUNITY FOR EXCELLENCE**

TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_GCM\_SHA384  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256

### Cipher Suite Ordering Prior to Windows 10

**MINIMUM BAR**

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256\_P256  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384\_P384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256\_P256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384\_P384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA\_P256  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA\_P384  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA\_P256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA\_P384

## OPPORTUNITY FOR EXCELLENCE

TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA384\_P384  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256\_P256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA384\_P384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256\_P256  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA\_P384  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA\_P256  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA\_P384  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA\_P256

## Legacy Cipher Suites

### OPTION #1 (BETTER SECURITY)

TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA\_P384 (uses SHA-1)  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA\_P256 (uses SHA-1)  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA\_P384 (uses SHA-1)  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA\_P256 (uses SHA-1)  
TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (uses SHA-1, lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (uses SHA-1, lack of Perfect Forward Secrecy)

### OPTION #2 (BETTER PERFORMANCE)

TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA\_P256 (uses SHA-1)  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_256\_CBC\_SHA\_P384 (uses SHA-1)  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA\_P256 (uses SHA-1)  
TLS\_ECDHE\_RSA\_WITH\_AES\_256\_CBC\_SHA\_P384 (uses SHA-1)  
TLS\_RSA\_WITH\_AES\_128\_GCM\_SHA256 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_256\_GCM\_SHA384 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA256 (lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA (uses SHA-1, lack of Perfect Forward Secrecy)  
TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA (uses SHA-1, lack of Perfect Forward Secrecy)

## Use TLS 1.2 in the IoT SDKs

Use the links below to configure TLS 1.2 and allowed ciphers in the Azure IoT client SDKs.

LANGUAGE	VERSIONS SUPPORTING TLS 1.2	DOCUMENTATION
C	Tag 2019-12-11 or newer	<a href="#">Link</a>
Python	Version 2.0.0 or newer	<a href="#">Link</a>
C#	Version 1.21.4 or newer	<a href="#">Link</a>
Java	Version 1.19.0 or newer	<a href="#">Link</a>
NodeJS	Version 1.12.2 or newer	<a href="#">Link</a>

## Use TLS 1.2 with IoT Hub

IoT Hub can be configured to use TLS 1.2 when communicating with devices. For more information, see [Deprecating TLS 1.0 and 1.1 for IoT Hub](#).

## Use TLS 1.2 with IoT Edge

IoT Edge devices can be configured to use TLS 1.2 when communicating with IoT Hub and DPS. For more information, see the [IoT Edge documentation page](#).

# Azure IoT Hub Device Provisioning Service (DPS) support for virtual networks

7/30/2020 • 5 minutes to read • [Edit Online](#)

This article introduces the virtual network (VNET) connectivity pattern for IoT devices provisioning with IoT hubs using DPS. This pattern provides private connectivity between the devices, DPS, and the IoT hub inside a customer-owned Azure VNET.

In most scenarios where DPS is configured with a VNET, your IoT Hub will also be configured in the same VNET. For more specific information on VNET support and configuration for IoT Hubs, see, [IoT Hub virtual network support](#).

## Introduction

By default, DPS hostnames map to a public endpoint with a publicly routable IP address over the Internet. This public endpoint is visible to all customers. Access to the public endpoint can be attempted by IoT devices over wide-area networks as well as on-premises networks.

For several reasons, customers may wish to restrict connectivity to Azure resources, like DPS. These reasons include:

- Prevent connection exposure over the public Internet. Exposure can be reduced by introducing additional layers of security via network level isolation for your IoT hub and DPS resources
- Enabling a private connectivity experience from your on-premises network assets ensuring that your data and traffic is transmitted directly to Azure backbone network.
- Preventing exfiltration attacks from sensitive on-premises networks.
- Following established Azure-wide connectivity patterns using [private endpoints](#).

Common approaches to restricting connectivity include [DPS IP filter rules](#) and Virtual networking (VNET) with [private endpoints](#). This goal of this article is to describe the VNET approach for DPS using private endpoints.

Devices that operate in on-premises networks can use [Virtual Private Network \(VPN\)](#) or [ExpressRoute](#) private peering to connect to a VNET in Azure and access DPS resources through private endpoints.

A private endpoint is a private IP address allocated inside a customer-owned VNET by which an Azure resource is accessible. By having a private endpoint for your DPS resource, you will be able to allow devices operating inside your VNET to request provisioning by your DPS resource without allowing traffic to the public endpoint.

## Prerequisites

Before proceeding ensure that the following prerequisites are met:

- Your DPS resource is already created and linked to your IoT hubs. For guidance on setting up a new DPS resource, see, [Set up IoT Hub Device Provisioning Service with the Azure portal](#)
- You have provisioned an Azure VNET with a subnet in which the private endpoint will be created. For more information, see, [create a virtual network using Azure CLI](#).
- For devices that operate inside of on-premises networks, set up [Virtual Private Network \(VPN\)](#) or [ExpressRoute](#) private peering into your Azure VNET.

# Private endpoint limitations

Note the following current limitations for DPS when using private endpoints:

- Private endpoints will not work with DPS when the DPS resource and the linked Hub are in different clouds.  
For example, [Azure Government and global Azure](#).
- Currently, [custom allocation policies with Azure Functions](#) for DPS will not work a VNET and private endpoints.
- Current DPS VNET support is for data ingress into DPS only. Data egress, which is the traffic from DPS to IoT Hub, uses an internal service-to-service mechanism rather than a dedicated VNET. Support for full VNET-based egress lockdown between DPS and IoT Hub is not currently available.
- The lowest latency allocation policy is used to assign a device to the IoT hub with the lowest latency. This allocation policy is not reliable in a virtual network environment.

## Set up a private endpoint

To set up a private endpoint, follow these steps:

1. In the [Azure portal](#), open your DPS resource and click the **Networking** tab. Click **Private endpoint connections** and **+ Private endpoint**.

The screenshot shows the Azure portal interface for a DPS resource named "test-dps-docs". The left sidebar has a "Networking" section with several options: Quick Start, Shared access policies, Linked IoT hubs, Certificates, Manage enrollments, Manage allocation policy, Networking (which is highlighted with a red box), and Properties. The main content area has tabs for "Public access" and "Private endpoint connections", with "Private endpoint connections" being the active tab (also highlighted with a red box). Below the tabs is a toolbar with "Search (Ctrl+ /)", "+ Private endpoint" (highlighted with a red box), "Approve", "Reject", "Refresh", and "Remove". A dropdown menu shows "All connection states". A table below lists connection details: Connection Name, Connection State, Private Endpoint, and Description. The table header includes columns for CONNECTION NAME, CONNECTION STATE, PRIVATE ENDPOINT, and DESCRIPTION.

2. On the *Create a private endpoint* Basics page, enter the information mentioned in the table below.

## Create a private endpoint

**1 Basics**   **2 Resource**   **3 Configuration**   **4 Tags**   **5 Review + create**

Use private endpoints to privately connect to a service or resource. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to. [Learn more](#)

**Project details**

Subscription \* ⓘ

Resource group \* ⓘ  [Create new](#)

Instance details

Name \*

Region \*

< Previous   **Next : Resource >**

FIELD	VALUE
Subscription	Choose the desired Azure subscription to contain the private endpoint.
Resource group	Choose or create a resource group to contain the private endpoint
Name	Enter a name for your private endpoint
Region	The region chosen must be the same as the region that contains the VNET, but it does not have to be the same as the DPS resource.

Click **Next : Resource** to configure the resource that the private endpoint will point to.

3. On the *Create a private endpoint Resource* page, enter the information mentioned in the table below.

Home > test-dps-docs | Networking >

## Create a private endpoint

✓ Basics   **2 Resource**   **3 Configuration**   **4 Tags**   **5 Review + create**

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method ⓘ  Connect to an Azure resource in my directory.  Connect to an Azure resource by resource ID or alias.

Subscription \* ⓘ

Resource type \* ⓘ

Resource \* ⓘ

Target sub-resource \* ⓘ

< Previous   **Next : Configuration >**

FIELD	VALUE
Subscription	Choose the Azure subscription that contains the DPS resource that your private endpoint will point to.
Resource type	Choose <b>Microsoft.Devices/ProvisioningServices</b> .
Resource	Select the DPS resource that the private endpoint will map to.
Target sub-resource	Select <b>iotDps</b> .

#### TIP

Information on the **Connect to an Azure resource by resource ID or alias** setting is provided in the [Request a private endpoint](#) section in this article.

Click **Next : Configuration** to configure the VNET for the private endpoint.

4. On the *Create a private endpoint Configuration* page, choose your virtual network and subnet to create the private endpoint in.

Click **Next : Tags**, and optionally provide any tags for your resource.

Home > test-dps-docs | Networking >

## Create a private endpoint

Basics Resource Configuration Tags Review + create

**Networking**

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network \*

Subnet \*

If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

**Private DNS integration**

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone  Yes  No

Configuration name	Subscription	Private DNS zones
privatelink-azure-devi...	Your Azure Subscription	(New) privatelink.azure-devices-provisioning....

**Review + create** < Previous Next : Tags >

5. Click **Review + create** and then **Create** to create your private endpoint resource.

## Request a private endpoint

You can request a private endpoint to a DPS resource by resource ID. In order to make this request, you need the resource owner to supply you with the resource ID.

1. The resource ID is provided on to the properties tab for DPS resource as shown below.

DPS-test7 | Properties

Name: DPS-test7

Resource type: Microsoft.Devices/ProvisioningServices

Location ID: eastus

Resource ID: /subscriptions/Your Azure Subscription ID/resourceGroups/DocRelatedTestingResources/providers/Microsoft.Devices/ProvisioningServices/DPS-test7

Resource group: DocRelatedTestingResources

Resource group ID: /subscriptions/Your Azure Subscription ID/resourceGroups/DocRelatedTestingResources

Subscription: DocRelatedTesting

#### Caution

Be aware that the resource ID does contain the subscription ID.

- Once you have the resource ID, follow the steps above in [Set up a private endpoint](#) to step 3 on the *Create a private endpoint Resource* page. Click **Connect to an Azure resource by resource ID or alias** and enter the information in the following table.

FIELD	VALUE
Resource ID or alias	Enter the resource ID for the DPS resource.
Target sub-resource	Enter <code>iotDps</code>
Request message	<p>Enter a request message for the DPS resource owner. For example,</p> <pre>Please approve this new private endpoint for IoT devices in site 23 to access this DPS instance</pre>

Click **Next : Configuration** to configure the VNET for the private endpoint.

- On the *Create a private endpoint Configuration* page, choose the virtual network and subnet to create the private endpoint in.

Click **Next : Tags**, and optionally provide any tags for your resource.

- Click **Review + create** and then **Create** to create your private endpoint request.
- The DPS owner will see the private endpoint request in the **Private endpoint connections** list on DPS networking tab. On that page, the owner can **Approve** or **Reject** the private endpoint request as shown below.

The screenshot shows the DPS-test7 | Networking blade in the Azure portal. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Settings (Quick Start, Shared access policies, Linked IoT hubs, Certificates, Manage enrollments, Manage allocation policy), and Networking. The Networking option is highlighted with a red box. The main content area is titled 'Private endpoint connections'. It has a toolbar with 'Private endpoint', 'Approve' (which has a checkmark), 'Reject', 'Refresh', and 'Remove'. A dropdown menu shows 'All connection states'. A table lists two connections:

Connection Name	Connection State	Private Endpoint	Description
DPS-test7.0d757774-c071...	Approved	dps-test7-private-endpoint...	Auto-Approved
DPS-test7.8e461870-ad42...	Pending	dps-test7-private-endpoint...	Please approve this new private endpoint for IoT devices in site 23 to access this DPS instance

## Pricing private endpoints

For pricing details, see [Azure Private Link pricing](#).

## Next steps

Use the links below to learn more about DPS security features:

- [Security](#)
- [TLS 1.2 Support](#)

# IoT Hub Device Provisioning Service concepts

12/10/2019 • 3 minutes to read • [Edit Online](#)

IoT Hub Device Provisioning Service is a helper service for IoT Hub that you use to configure zero-touch device provisioning to a specified IoT hub. With the Device Provisioning Service, you can [auto-provision](#) millions of devices in a secure and scalable manner.

Device provisioning is a two part process. The first part is establishing the initial connection between the device and the IoT solution by *registering* the device. The second part is applying the proper *configuration* to the device based on the specific requirements of the solution. Once both steps have been completed, the device has been fully *provisioned*. Device Provisioning Service automates both steps to provide a seamless provisioning experience for the device.

This article gives an overview of the provisioning concepts most applicable to managing the *service*. This article is most relevant to personas involved in the [cloud setup step](#) of getting a device ready for deployment.

## Service operations endpoint

The service operations endpoint is the endpoint for managing the service settings and maintaining the enrollment list. This endpoint is only used by the service administrator; it is not used by devices.

## Device provisioning endpoint

The device provisioning endpoint is the single endpoint all devices use for auto-provisioning. The URL is the same for all provisioning service instances, to eliminate the need to reflash devices with new connection information in supply chain scenarios. The ID scope ensures tenant isolation.

## Linked IoT hubs

The Device Provisioning Service can only provision devices to IoT hubs that have been linked to it. Linking an IoT hub to an instance of the Device Provisioning service gives the service read/write permissions to the IoT hub's device registry; with the link, a Device Provisioning service can register a device ID and set the initial configuration in the device twin. Linked IoT hubs may be in any Azure region. You may link hubs in other subscriptions to your provisioning service.

## Allocation policy

The service-level setting that determines how Device Provisioning Service assigns devices to an IoT hub. There are three supported allocation policies:

- **Evenly weighted distribution:** linked IoT hubs are equally likely to have devices provisioned to them. The default setting. If you are provisioning devices to only one IoT hub, you can keep this setting.
- **Lowest latency:** devices are provisioned to an IoT hub with the lowest latency to the device. If multiple linked IoT hubs would provide the same lowest latency, the provisioning service hashes devices across those hubs
- **Static configuration via the enrollment list:** specification of the desired IoT hub in the enrollment list takes priority over the service-level allocation policy.

## Enrollment

An enrollment is the record of devices or groups of devices that may register through auto-provisioning. The enrollment record contains information about the device or group of devices, including:

- the [attestation mechanism](#) used by the device
- the optional initial desired configuration
- desired IoT hub
- the desired device ID

There are two types of enrollments supported by Device Provisioning Service:

### **Enrollment group**

An enrollment group is a group of devices that share a specific attestation mechanism. Enrollment groups support both X.509 as well as symmetric. All devices in the X.509 enrollment group present X.509 certificates that have been signed by the same root or intermediate Certificate Authority (CA). Each device in the symmetric key enrollment group present SAS tokens derived from the group symmetric key. The enrollment group name and certificate name must be alphanumeric, lowercase, and may contain hyphens.

#### **TIP**

We recommend using an enrollment group for a large number of devices that share a desired initial configuration, or for devices all going to the same tenant.

### **Individual enrollment**

An individual enrollment is an entry for a single device that may register. Individual enrollments may use either X.509 leaf certificates or SAS tokens (from a physical or virtual TPM) as attestation mechanisms. The registration ID in an individual enrollment is alphanumeric, lowercase, and may contain hyphens. Individual enrollments may have the desired IoT hub device ID specified.

#### **TIP**

We recommend using individual enrollments for devices that require unique initial configurations, or for devices that can only authenticate using SAS tokens via TPM attestation.

## **Registration**

A registration is the record of a device successfully registering/provisioning to an IoT Hub via the Device Provisioning Service. Registration records are created automatically; they can be deleted, but they cannot be updated.

## **Operations**

Operations are the billing unit of the Device Provisioning Service. One operation is the successful completion of one instruction to the service. Operations include device registrations and re-registrations; operations also include service-side changes such as adding enrollment list entries, and updating enrollment list entries.

# IoT Hub DPS IP addresses

3/12/2020 • 2 minutes to read • [Edit Online](#)

The IP address prefixes for the public endpoints of an IoT Hub Device Provisioning Service (DPS) are published periodically under the [AzureIoTHub service tag](#). You may use these IP address prefixes to control connectivity between an IoT DPS instance and devices or network assets in order to implement a variety of network isolation goals:

GOAL	APPROACH
Ensure your devices and services communicate with IoT Hub DPS endpoints only	Use the <a href="#">AzureIoTHub service tag</a> to discover IoT Hub DPS instances. Configure ALLOW rules on your devices' and services' firewall setting for those IP address prefixes accordingly. Configure rules to drop traffic to other destination IP addresses that you do not want devices or services to communicate with.
Ensure your IoT Hub DPS endpoint receives connections only from your devices and network assets	Use IoT DPS <a href="#">IP filter feature</a> to create filter rules for the device and DPS service APIs. These filter rules can be used to allow connections only from your devices and network asset IP addresses (see <a href="#">limitations</a> section).

## Best practices

- When adding ALLOW rules in your devices' firewall configuration, it is best to provide specific [ports used by applicable protocols](#).
- The IP address prefixes of IoT DPS instances are subject to change. These changes are published periodically via service tags before taking effect. It is therefore important that you develop processes to regularly retrieve and use the latest service tags. This process can be automated via the [service tags discovery API](#). The Service tags discovery API is still in preview and in some cases may not produce the full list of tags and IP addresses. Until discovery API is generally available, consider using the [service tags in downloadable JSON format](#).
- Use the [AzureIoTHub.\[region name\]](#) tag to identify IP prefixes used by DPS endpoints in a specific region. To account for datacenter disaster recovery, or [regional failover](#), ensure connectivity to IP prefixes of your DPS instance's geo-pair region is also enabled.
- Setting up firewall rules for a DPS instance may block off connectivity needed to run Azure CLI and PowerShell commands against it. To avoid these connectivity issues, you can add ALLOW rules for your clients' IP address prefixes to re-enable CLI or PowerShell clients to communicate with your DPS instance.

## Limitations and workarounds

- The DPS IP filter feature has a limit of 100 rules. This limit can be raised via requests through Azure Customer Support.
- Your configured [IP filtering rules](#) are only applied on your DPS endpoints and not on the linked IoT Hub endpoints. IP filtering for linked IoT Hubs must be configured separately. For more information, see, [IoT Hub IP filtering rules](#).

## Support for IPv6

IPv6 is currently not supported on IoT Hub or DPS.

## Next steps

To learn more about IP address configurations with DPS, see:

- [Configure IP filtering](#)

# TPM attestation

12/10/2019 • 4 minutes to read • [Edit Online](#)

IoT Hub Device Provisioning Service is a helper service for IoT Hub that you use to configure zero-touch device provisioning to a specified IoT hub. With the Device Provisioning Service, you can provision millions of devices in a secure manner.

This article describes the identity attestation process when using a [TPM](#). TPM stands for Trusted Platform Module and is a type of hardware security module (HSM). This article assumes you are using a discrete, firmware, or integrated TPM. Software emulated TPMs are well-suited for prototyping or testing, but they do not provide the same level of security as discrete, firmware, or integrated TPMs do. We do not recommend using software TPMs in production. For more information about types of TPMs, see [A Brief Introduction to TPM](#).

This article is only relevant for devices using TPM 2.0 with HMAC key support and their endorsement keys. It is not for devices using X.509 certificates for authentication. TPM is an industry-wide, ISO standard from the Trusted Computing Group, and you can read more about TPM at the [complete TPM 2.0 spec](#) or the [ISO/IEC 11889 spec](#). This article also assumes you are familiar with public and private key pairs, and how they are used for encryption.

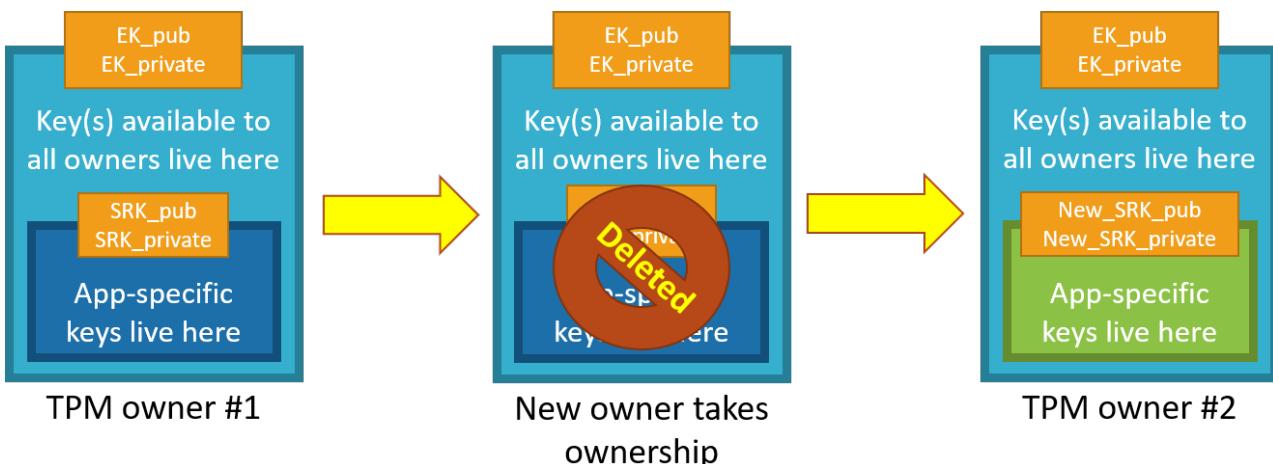
The Device Provisioning Service device SDKs handle everything that is described in this article for you. There is no need for you to implement anything additional if you are using the SDKs on your devices. This article helps you understand conceptually what's going on with your TPM security chip when your device provisions and why it's so secure.

## Overview

TPMs use something called the endorsement key (EK) as the secure root of trust. The EK is unique to the TPM and changing it essentially changes the device into a new one.

There's another type of key that TPMs have, called the storage root key (SRK). An SRK may be generated by the TPM's owner after it takes ownership of the TPM. Taking ownership of the TPM is the TPM-specific way of saying "someone sets a password on the HSM." If a TPM device is sold to a new owner, the new owner can take ownership of the TPM to generate a new SRK. The new SRK generation ensures the previous owner can't use the TPM. Because the SRK is unique to the owner of the TPM, the SRK can be used to seal data into the TPM itself for that owner. The SRK provides a sandbox for the owner to store their keys and provides access revocability if the device or TPM is sold. It's like moving into a new house: taking ownership is changing the locks on the doors and destroying all furniture left by the previous owners (SRK), but you can't change the address of the house (EK).

Once a device has been set up and ready to use, it will have both an EK and an SRK available for use.



One note on taking ownership of the TPM: Taking ownership of a TPM depends on many things, including TPM manufacturer, the set of TPM tools being used, and the device OS. Follow the instructions relevant to your system to take ownership.

The Device Provisioning Service uses the public part of the EK (EK\_pub) to identify and enroll devices. The device vendor can read the EK\_pub during manufacture or final testing and upload the EK\_pub to the provisioning service so that the device will be recognized when it connects to provision. The Device Provisioning Service does not check the SRK or owner, so "clearing" the TPM erases customer data, but the EK (and other vendor data) is preserved and the device will still be recognized by the Device Provisioning Service when it connects to provision.

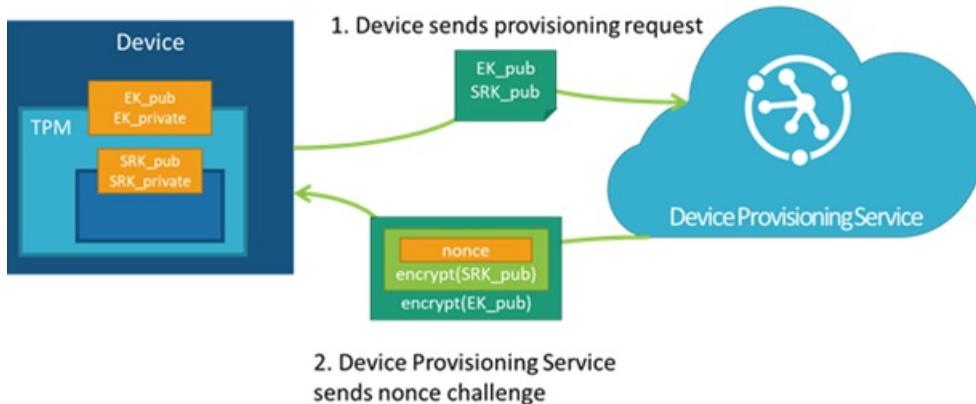
## Detailed attestation process

When a device with a TPM first connects to the Device Provisioning Service, the service first checks the provided EK\_pub against the EK\_pub stored in the enrollment list. If the EK\_pubs do not match, the device is not allowed to provision. If the EK\_pubs do match, the service then requires the device to prove ownership of the private portion of the EK via a nonce challenge, which is a secure challenge used to prove identity. The Device Provisioning Service generates a nonce and then encrypts it with the SRK and then the EK\_pub, both of which are provided by the device during the initial registration call. The TPM always keeps the private portion of the EK secure. This prevents counterfeiting and ensures SAS tokens are securely provisioned to authorized devices.

Let's walk through the attestation process in detail.

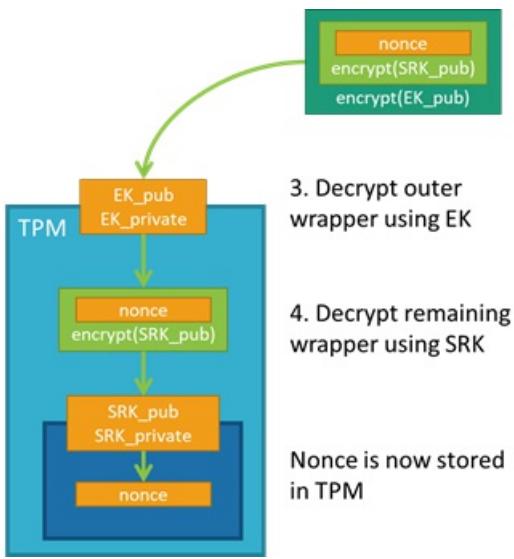
### Device requests an IoT Hub assignment

First the device connects to the Device Provisioning Service and requests to provision. In doing so, the device provides the service with its registration ID, an ID scope, and the EK\_pub and SRK\_pub from the TPM. The service passes the encrypted nonce back to the device and asks the device to decrypt the nonce and use that to sign a SAS token to connect again and finish provisioning.



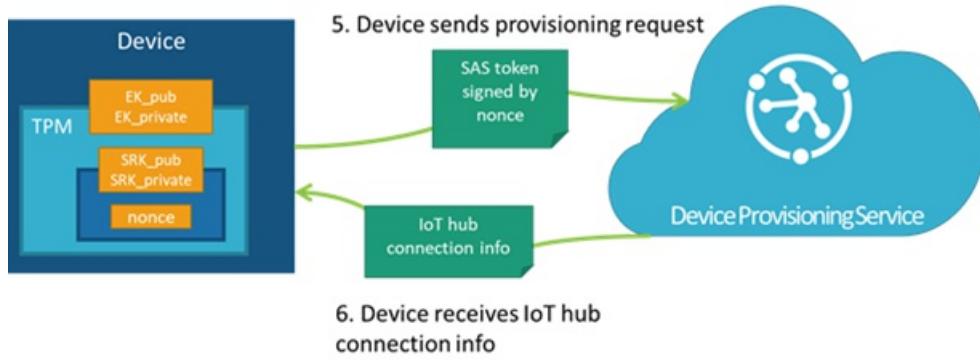
### Nonce challenge

The device takes the nonce and uses the private portions of the EK and SRK to decrypt the nonce into the TPM; the order of nonce encryption delegates trust from the EK, which is immutable, to the SRK, which can change if a new owner takes ownership of the TPM.



### Validate the nonce and receive credentials

The device can then sign a SAS token using the decrypted nonce and reestablish a connection to the Device Provisioning Service using the signed SAS token. With the Nonce challenge completed, the service allows the device to provision.



## Next steps

Now the device connects to IoT Hub, and you rest secure in the knowledge that your devices' keys are securely stored. Now that you know how the Device Provisioning Service securely verifies a device's identity using TPM, check out the following articles to learn more:

- [Learn about all the concepts in auto-provisioning](#)
- [Get started using auto-provisioning using the SDKs to take care of the flow.](#)

# Symmetric key attestation

7/30/2020 • 4 minutes to read • [Edit Online](#)

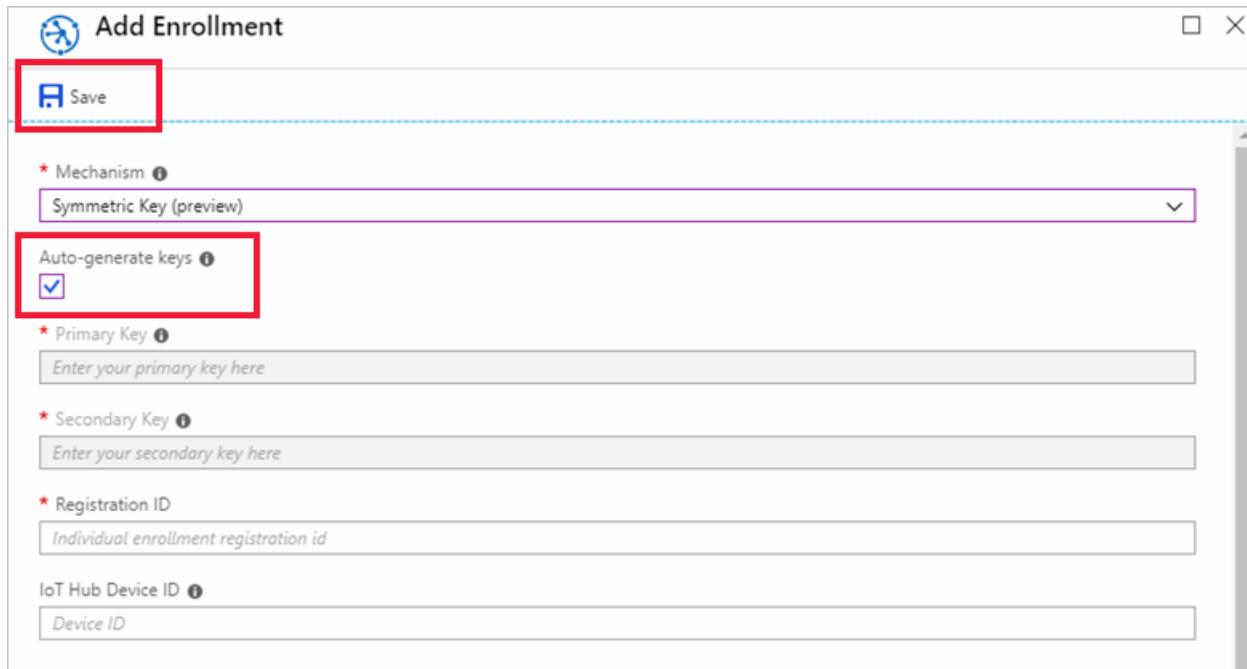
This article describes the identity attestation process when using symmetric keys with the Device Provisioning Service.

Symmetric key attestation is a simple approach to authenticating a device with a Device Provisioning Service instance. This attestation method represents a "Hello world" experience for developers who are new to device provisioning, or do not have strict security requirements. Device attestation using a [TPM](#) or an [X.509 certificate](#) is more secure, and should be used for more stringent security requirements.

Symmetric key enrollments also provide a great way for legacy devices, with limited security functionality, to bootstrap to the cloud via Azure IoT. For more information on symmetric key attestation with legacy devices, see [How to use symmetric keys with legacy devices](#).

## Symmetric key creation

By default, the Device Provisioning Service creates new symmetric keys with a default length of 64 bytes when new enrollments are saved with the **Auto-generate keys** option enabled.



You can also provide your own symmetric keys for enrollments by disabling this option. When specifying your own symmetric keys, your keys must have a key length between 16 bytes and 64 bytes. Also, symmetric keys must be provided in valid Base64 format.

## Detailed attestation process

Symmetric key attestation with the Device Provisioning Service is performed using the same [Security tokens](#) supported by IoT hubs to identify devices. These security tokens are [Shared Access Signature \(SAS\) tokens](#).

SAS tokens have a hashed *signature* that is created using the symmetric key. The signature is recreated by the Device Provisioning Service to verify whether a security token presented during attestation is authentic or not.

SAS tokens have the following form:

```
SharedAccessSignature sig={signature}&se={expiry}&skn={policyName}&sr={URL-encoded-resourceURI}
```

Here are the components of each token:

VALUE	DESCRIPTION
{signature}	An HMAC-SHA256 signature string. For individual enrollments, this signature is produced by using the symmetric key (primary or secondary) to perform the hash. For enrollment groups, a key derived from the enrollment group key is used to perform the hash. The hash is performed on a message of the form: <code>URL-encoded-resourceURI + "\n" + expiry</code> . <b>Important:</b> The key must be decoded from base64 before being used to perform the HMAC-SHA256 computation. Also, the signature result must be URL-encoded.
{resourceURI}	URI of the registration endpoint that can be accessed with this token, starting with scope ID for the Device Provisioning Service instance. For example, <code>{Scope ID}/registrations/{Registration ID}</code>
{expiry}	UTF8 strings for number of seconds since the epoch 00:00:00 UTC on 1 January 1970.
{URL-encoded-resourceURI}	Lower case URL-encoding of the lower case resource URI
{policyName}	The name of the shared access policy to which this token refers. The policy name used when provisioning with symmetric key attestation is <b>registration</b> .

When a device is attesting with an individual enrollment, the device uses the symmetric key defined in the individual enrollment entry to create the hashed signature for the SAS token.

For code examples that create a SAS token, see [Security Tokens](#).

Creating security tokens for symmetric key attestation is supported by the Azure IoT C SDK. For an example using the Azure IoT C SDK to attest with an individual enrollment, see [Provision a simulated device with symmetric keys](#).

## Group Enrollments

The symmetric keys for group enrollments are not used directly by devices when provisioning. Instead devices that belong to an enrollment group provision using a derived device key.

First, a unique registration ID is defined for each device attesting with an enrollment group. Valid characters for the registration ID are lowercase alphanumeric and dash ('-'). This registration ID should be something unique that identifies the device. For example, a legacy device may not support many security features. The legacy device may only have a MAC address or serial number available to uniquely identify that device. In that case, a registration ID can be composed of the MAC address and serial number similar to the following:

```
sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6
```

This exact example is used in the [How to provision legacy devices using symmetric keys](#) article.

Once a registration ID has been defined for the device, the symmetric key for the enrollment group is used to compute an **HMAC-SHA256** hash of the registration ID to produce a derived device key. The hashing of the registration ID can be performed with the following C# code:

```

using System;
using System.Security.Cryptography;
using System.Text;

public static class Utils
{
    public static string ComputeDerivedSymmetricKey(byte[] masterKey, string registrationId)
    {
        using (var hmac = new HMACSHA256(masterKey))
        {
            return Convert.ToBase64String(hmac.ComputeHash(Encoding.UTF8.GetBytes(registrationId)));
        }
    }
}

```

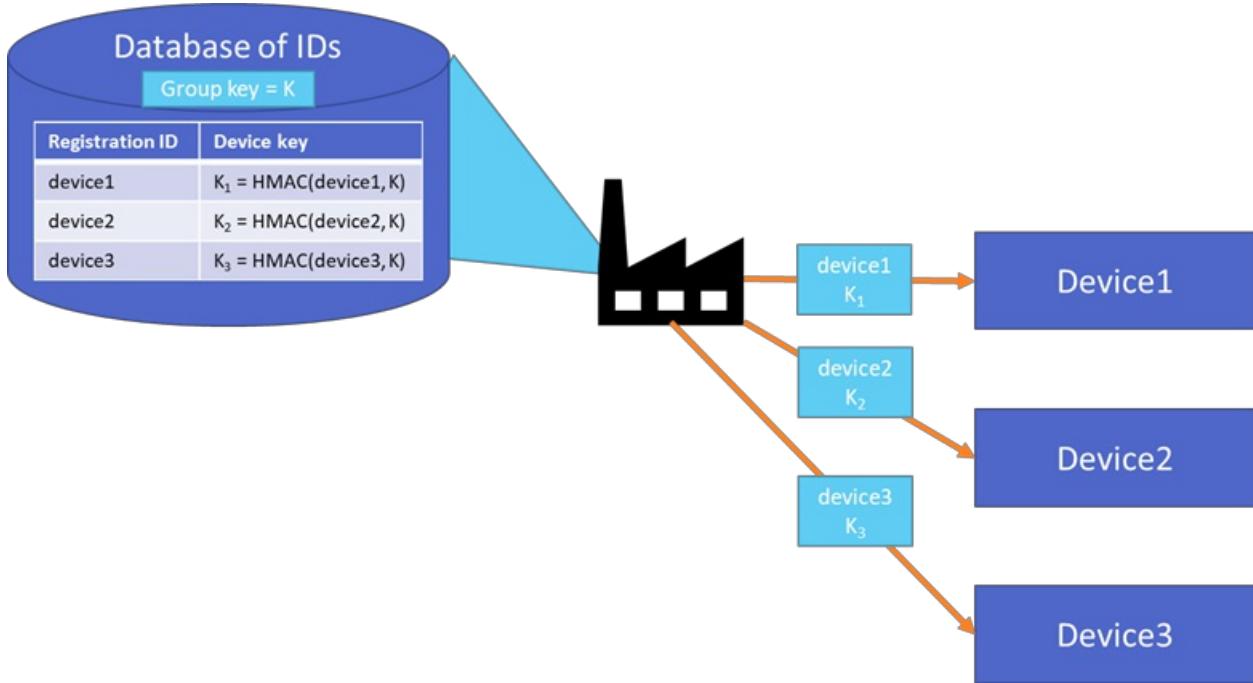
```
String deviceKey = Utils.ComputeDerivedSymmetricKey(Convert.FromBase64String(masterKey), registrationId);
```

The resulting device key is then used to generate a SAS token to be used for attestation. Each device in an enrollment group is required to attest using a security token generated from a unique derived key. The enrollment group symmetric key cannot be used directly for attestation.

#### **Installation of the derived device key**

Ideally the device keys are derived and installed in the factory. This method guarantees the group key is never included in any software deployed to the device. When the device is assigned a MAC address or serial number, the key can be derived and injected into the device however the manufacturer chooses to store it.

Consider the following diagram that shows a table of device keys generated in a factory by hashing each device registration ID with the group enrollment key (K).



The identity of each device is represented by the registration ID and derived device key that is installed at the factory. The device key is never copied to another location and the group key is never stored on a device.

If the device keys are not installed in the factory, a [hardware security module HSM](#) should be used to securely store the device identity.

## Next steps

Now that you have an understanding of Symmetric Key attestation, check out the following articles to learn more:

- Quickstart: Provision a simulated device with symmetric keys
- Learn about the concepts in auto-provisioning
- Get started using auto-provisioning

# Security practices for Azure IoT device manufacturers

4/1/2020 • 15 minutes to read • [Edit Online](#)

As more manufacturers release IoT devices, it's helpful to identify guidance around common practices. This article summarizes recommended security practices to consider when you manufacture devices for use with Azure IoT Device Provisioning Service (DPS).

- Selecting device authentication options
- Installing certificates on IoT devices
- Integrating a Trusted Platform Module (TPM) into the manufacturing process

## Selecting device authentication options

The ultimate aim of any IoT device security measure is to create a secure IoT solution. But issues such as hardware limitations, cost, and level of security expertise all impact which options you choose. Further, your approach to security impacts how your IoT devices connect to the cloud. While there are [several elements of IoT security](#) to consider, a key element that every customer encounters is what authentication type to use.

Three widely used authentication types are X.509 certificates, Trusted Platform Modules (TPM), and symmetric keys. While other authentication types exist, most customers who build solutions on Azure IoT use one of these three types. The rest of this article surveys pros and cons of using each authentication type.

### X.509 certificate

X.509 certificates are a type of digital identity you can use for authentication. The X.509 certificate standard is documented in [IETF RFC 5280](#). In Azure IoT, there are two ways to authenticate certificates:

- Thumbprint. A thumbprint algorithm is run on a certificate to generate a hexadecimal string. The generated string is a unique identifier for the certificate.
- CA authentication based on a full chain. A certificate chain is a hierarchical list of all certificates needed to authenticate an end-entity (EE) certificate. To authenticate an EE certificate, it's necessary to authenticate each certificate in the chain including a trusted root CA.

Pros for X.509:

- X.509 is the most secure authentication type supported in Azure IoT.
- X.509 allows a high level of control for purposes of certificate management.
- Many vendors are available to provide X.509 based authentication solutions.

Cons for X.509:

- Many customers may need to rely on external vendors for their certificates.
- Certificate management can be costly and adds to total solution cost.
- Certificate life-cycle management can be difficult if logistics are not well thought out.

### Trusted Platform Module (TPM)

TPM, also known as [ISO/IEC 11889](#), is a standard for securely generating and storing cryptographic keys. TPM also refers to a virtual or physical I/O device that interacts with modules that implement the standard. A TPM device can exist as discrete hardware, integrated hardware, a firmware-based module, or a software-based module.

There are two key differences between TPMs and symmetric keys:

- TPM chips can also store X.509 certificates.

- TPM attestation in DPS uses the TPM endorsement key (EK), a form of asymmetric authentication. With asymmetric authentication, a public key is used for encryption, and a separate private key is used for decryption. In contrast, symmetric keys use symmetric authentication, where the private key is used for both encryption and decryption.

Pros for TPM:

- TPMs are included as standard hardware on many Windows devices, with built-in support for the operating system.
- TPM attestation is easier to secure than shared access signature (SAS) token-based symmetric key attestation.
- You can easily expire and renew, or roll, device credentials. DPS automatically rolls the IoT Hub credentials whenever a TPM device is due for re-provisioning.

Cons for TPM:

- TPMs are complex and can be difficult to use.
- Application development with TPMs is difficult unless you have a physical TPM or a quality emulator.
- You may have to redesign the board of your device to include a TPM in the hardware.
- If you roll the EK on a TPM, it destroys the identity of the TPM and creates a new one. Although the physical chip stays the same, it has a new identity in your IoT solution.

### **Symmetric key**

With symmetric keys, the same key is used to encrypt and decrypt messages. As a result, the same key is known to both the device and the service that authenticates it. Azure IoT supports SAS token-based symmetric key connections. Symmetric key authentication requires significant owner responsibility to secure the keys and achieve an equal level of security with X.509 authentication. If you use symmetric keys, the recommended practice is to protect the keys by using a hardware security module (HSM).

Pros for symmetric key:

- Using symmetric keys is the simplest, lowest cost way to get started with authentication.
- Using symmetric keys streamlines your process because there's nothing extra to generate.

Cons for symmetric key:

- Symmetric keys take a significant degree of effort to secure the keys. The same key is shared between device and cloud, which means the key must be protected in two places. In contrast, the challenge with TPM and X.509 certificates is proving possession of the public key without revealing the private key.
- Symmetric keys make it easy to follow poor security practices. A common tendency with symmetric keys is to hard code the unencrypted keys on devices. While this practice is convenient, it leaves the keys vulnerable. You can mitigate some risk by securely storing the symmetric key on the device. However, if your priority is ultimately security rather than convenience, use X.509 certificates or TPM for authentication.

### **Shared symmetric key**

There's a variation of symmetric key authentication known as shared symmetric key. This approach involves using the same symmetric key in all devices. The recommendation is to avoid using shared symmetric keys on your devices.

Pro for shared symmetric key:

- Simple to implement and inexpensive to produce at scale.

Cons for shared symmetric key:

- Highly vulnerable to attack. The benefit of easy implementation is far outweighed by the risk.
- Anyone can impersonate your devices if they obtain the shared key.

- If you rely on a shared symmetric key that becomes compromised, you will likely lose control of the devices.

## Making the right choice for your devices

To choose an authentication method, make sure you consider the benefits and costs of each approach for your unique manufacturing process. For device authentication, usually there's an inverse relationship between how secure a given approach is, and how convenient it is.

# Installing certificates on IoT devices

If you use X.509 certificates to authenticate your IoT devices, this section offers guidance on how to integrate certificates into your manufacturing process. You'll need to make several decisions. These include decisions about common certificate variables, when to generate certificates, and when to install them.

If you're used to using passwords, you might ask why you can't use the same certificate in all your devices, in the same way that you'd be able to use the same password in all your devices. First, using the same password everywhere is dangerous. The practice has exposed companies to major DDoS attacks, including the one that took down DNS on the US East Coast several years ago. Never use the same password everywhere, even with personal accounts. Second, a certificate isn't a password, it's a unique identity. A password is like a secret code that anyone can use to open a door at a secured building. It's something you know, and you could give the password to anyone to gain entrance. A certificate is like a driver's license with your photo and other details, which you can show to a guard to get into a secured building. It's tied to who you are. Provided that the guard accurately matches people with driver's licenses, only you can use your license (identity) to gain entrance.

## Variables involved in certificate decisions

Consider the following variables, and how each one impacts the overall manufacturing process.

### Where the certificate root of trust comes from

It can be costly and complex to manage a public key infrastructure (PKI). Especially if your company doesn't have any experience managing a PKI. Your options are:

- Use a third-party PKI. You can buy intermediate signing certificates from a third-party certificate vendor. Or you can use a private Certificate Authority (CA).
- Use a self-managed PKI. You can maintain your own PKI system and generate your own certificates.
- Use the [Azure Sphere](#) security service. This option applies only to Azure Sphere devices.

### Where certificates are stored

There are a few factors that impact the decision on where certificates are stored. These factors include the type of device, expected profit margins (whether you can afford secure storage), device capabilities, and existing security technology on the device that you may be able to use. Consider the following options:

- In a hardware security module (HSM). Using an HSM is highly recommended. Check whether your device's control board already has an HSM installed. If you know you don't have an HSM, work with your hardware manufacturer to identify an HSM that meets your needs.
- In a secure place on disk such as a trusted execution environment (TEE).
- In the local file system or a certificate store. For example, the Windows certificate store.

### Connectivity at the factory

Connectivity at the factory determines how and when you'll get the certificates to install on the devices.

Connectivity options are as follows:

- Connectivity. Having connectivity is optimal, it streamlines the process of generating certificates locally.
- No connectivity. In this case, you use a signed certificate from a CA to generate device certificates locally and offline.
- No connectivity. In this case, you can obtain certificates that were generated ahead of time. Or you can use an offline PKI to generate certificates locally.

## Audit requirement

Depending on the type of devices you produce, you might have a regulatory requirement to create an audit trail of how device identities are installed on your devices. Auditing adds significant production cost. So in most cases, only do it if necessary. If you're unsure whether an audit is required, check with your company's legal department.

Auditing options are:

- Not a sensitive industry. No auditing is required.
- Sensitive industry. Certificates should be installed in a secure room according to compliance certification requirements. If you need a secure room to install certificates, you are likely already aware of how certificates get installed in your devices. And you probably already have an audit system in place.

## Length of certificate validity

Like a driver's license, certificates have an expiration date that is set when they are created. Here are the options for length of certificate validity:

- Renewal not required. This approach uses a long renewal period, so you'll never need to renew the certificate during the device's lifetime. While such an approach is convenient, it's also risky. You can reduce the risk by using secure storage like an HSM on your devices. However, the recommended practice is to avoid using long-lived certificates.
- Renewal required. You'll need to renew the certificate during the lifetime of the device. The length of the certificate validity depends on context, and you'll need a strategy for renewal. The strategy should include where you're getting certificates, and what type of over-the-air functionality your devices have to use in the renewal process.

## When to generate certificates

The internet connectivity capabilities at your factory will impact your process for generating certificates. You have several options for when to generate certificates:

- Pre-loaded certificates. Some HSM vendors offer a premium service in which the HSM vendor installs certificates for the customer. First, customers give the HSM vendor access to a signing certificate. Then the HSM vendor installs certificates signed by that signing certificate onto each HSM the customer buys. All the customer has to do is install the HSM on the device. While this service adds cost, it helps to streamline your manufacturing process. And it resolves the question of when to install certificates.
- Device-generated certificates. If your devices generate certificates internally, then you must extract the public X.509 certificate from the device to enroll it in DPS.
- Connected factory. If your factory has connectivity, you can generate device certificates whenever you need them.
- Offline factory with your own PKI. If your factory does not have connectivity, and you are using your own PKI with offline support, you can generate the certificates when you need them.
- Offline factory with third-party PKI. If your factory does not have connectivity, and you are using a third-party PKI, you must generate the certificates ahead of time. And it will be necessary to generate the certificates from a location that has connectivity.

## When to install certificates

After you generate certificates for your IoT devices, you can install them in the devices.

If you use pre-loaded certificates with an HSM, the process is simplified. After the HSM is installed in the device, the device code can access it. Then you'll call the HSM APIs to access the certificate that's stored in the HSM. This approach is the most convenient for your manufacturing process.

If you don't use a pre-loaded certificate, you must install the certificate as part of your production process. The simplest approach is to install the certificate in the HSM at the same time that you flash the initial firmware image. Your process must add a step to install the image on each device. After this step, you can run final quality checks and any other steps, before you package and ship the device.

There are software tools available that let you run the installation process and final quality check in a single step. You can modify these tools to generate a certificate, or to pull a certificate from a pre-generated certificate store. Then the software can install the certificate where you need to install it. Software tools of this type enable you to run production quality manufacturing at scale.

After you have certificates installed on your devices, the next step is to learn how to enroll the devices with [DPS](#).

## Integrating a TPM into the manufacturing process

If you use a TPM to authenticate your IoT devices, this section offers guidance. The guidance covers the widely used TPM 2.0 devices that have hash-based message authentication code (HMAC) key support. The TPM specification for TPM chips is an ISO standard that's maintained by the Trusted Computing Group. For more on TPM, see the specifications for [TPM 2.0](#) and [ISO/IEC 11889](#).

### Taking ownership of the TPM

A critical step in manufacturing a device with a TPM chip is to take ownership of the TPM. This step is required so that you can provide a key to the device owner. The first step is to extract the endorsement key (EK) from the device. The next step is to actually claim ownership. How you accomplish this depends on which TPM and operating system you use. If needed, contact the TPM manufacturer or the developer of the device operating system to determine how to claim ownership.

In your manufacturing process, you can extract the EK and claim ownership at different times, which adds flexibility. Many manufacturers take advantage of this flexibility by adding a hardware security module (HSM) to enhance the security of their devices. This section provides guidance on when to extract the EK, when to claim ownership of the TPM, and considerations for integrating these steps into a manufacturing timeline.

#### IMPORTANT

The following guidance assumes you use a discrete, firmware, or integrated TPM. In places where it's applicable, the guidance adds notes on using a non-discrete or software TPM. If you use a software TPM, there may be additional steps that this guidance doesn't include. Software TPMs have a variety of implementations that are beyond the scope of this article. In general, it's possible to integrate a software TPM into the following general manufacturing timeline. However, while a software emulated TPM is suitable for prototyping and testing, it can't provide the same level of security as a discrete, firmware, or integrated TPM. As a general practice, avoid using a software TPM in production.

### General manufacturing timeline

The following timeline shows how a TPM goes through a production process and ends up in a device. Each manufacturing process is unique, and this timeline shows the most common patterns. The timeline offers guidance on when to take certain actions with the keys.

#### Step 1: TPM is manufactured

- If you buy TPMs from a manufacturer for use in your devices, see if they'll extract public endorsement keys (EK\_pubs) for you. It's helpful if the manufacturer provides the list of EK\_pubs with the shipped devices.

#### NOTE

You could give the TPM manufacturer write access to your enrollment list by using shared access policies in your provisioning service. This approach lets them add the TPMs to your enrollment list for you. But that is early in the manufacturing process, and it requires trust in the TPM manufacturer. Do so at your own risk.

- If you manufacture TPMs to sell to device manufacturers, consider giving your customers a list of EK\_pubs along with their physical TPMs. Providing customers with EK\_pubs saves a step in their process.
- If you manufacture TPMs to use with your own devices, identify which point in your process is the most

convenient to extract the EK\_pub. You can extract the EK\_pub at any of the remaining points in the timeline.

#### **Step 2: TPM is installed into a device**

At this point in the production process, you should know which DPS instance the device will be used with. As a result, you can add devices to the enrollment list for automated provisioning. For more information about automatic device provisioning, see the [DPS documentation](#).

- If you haven't extracted the EK\_pub, now is a good time to do so.
- Depending on the installation process of the TPM, this step is also a good time to take ownership of the TPM.

#### **Step 3: Device has firmware and software installed**

At this point in the process, install the DPS client along with the ID scope and global URL for provisioning.

- Now is the last chance to extract the EK\_pub. If a third party will install the software on your device, it's a good idea to extract the EK\_pub first.
- This point in the manufacturing process is ideal to take ownership of the TPM.

##### **NOTE**

If you're using a software TPM, you can install it now. Extract the EK\_pub at the same time.

#### **Step 4: Device is packaged and sent to the warehouse**

A device can sit in a warehouse for 6-12 months before being deployed.

#### **Step 5: Device is installed into the location**

After the device arrives at its final location, it goes through automated provisioning with DPS.

For more information, see [Autoprovisioning concepts](#) and [TPM attestation](#).

## Resources

In addition to the recommended security practices in this article, Azure IoT provides resources to help with selecting secure hardware and creating secure IoT deployments:

- Azure IoT [security recommendations](#) to guide the deployment process.
- The [Azure Security Center](#) offers a service to help create secure IoT deployments.
- For help with evaluating your hardware environment, see the whitepaper [Evaluating your IoT Security](#).
- For help with selecting secure hardware, see [The Right Secure Hardware for your IoT Deployment](#).

# Use Azure IoT Hub Device Provisioning Service auto-provisioning to register the MXChip IoT DevKit with IoT Hub

12/10/2019 • 4 minutes to read • [Edit Online](#)

This article describes how to use Azure IoT Hub Device Provisioning Service [auto-provisioning](#), to register the MXChip IoT DevKit with Azure IoT Hub. In this tutorial, you learn how to:

- Configure the global endpoint of the Device Provisioning service on a device.
- Use a unique device secret (UDS) to generate an X.509 certificate.
- Enroll an individual device.
- Verify that the device is registered.

The [MXChip IoT DevKit](#) is an all-in-one Arduino-compatible board with rich peripherals and sensors. You can develop for it using [Azure IoT Device Workbench](#) or [Azure IoT Tools](#) extension pack in Visual Studio Code. The DevKit comes with a growing [projects catalog](#) to guide your prototype Internet of Things (IoT) solutions that take advantage of Azure services.

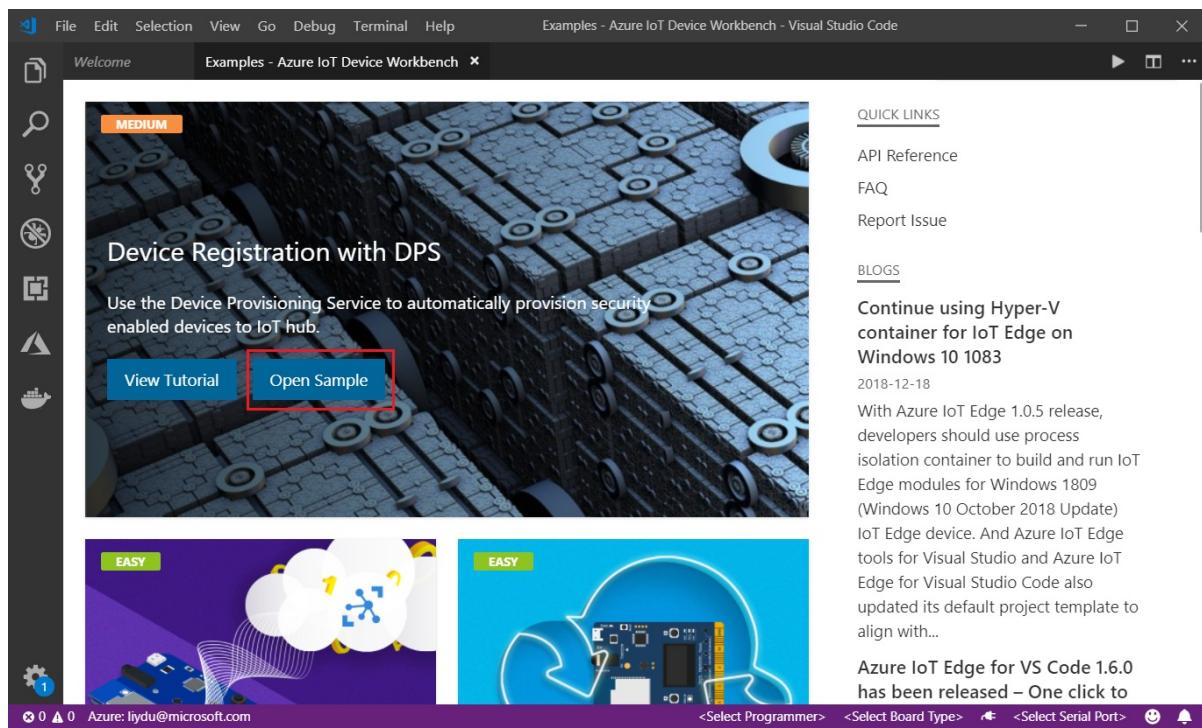
## Before you begin

To complete the steps in this tutorial, first do the following tasks:

- Configure your DevKit's Wi-Fi and prepare your development environment by following the "Prepare the development environment" section steps in [Connect IoT DevKit AZ3166 to Azure IoT Hub in the cloud](#).
- Upgrade to the latest firmware (1.3.0 or later) with the [Update DevKit firmware](#) tutorial.
- Create and link an IoT Hub with a Device Provisioning service instance by following the steps in [Set up the IoT Hub Device Provisioning Service with the Azure portal](#).

## Open sample project

1. Make sure your IoT DevKit is **not connected** to your computer. Start VS Code first, and then connect the DevKit to your computer.
2. Click **F1** to open the command palette, type and select **Azure IoT Device Workbench: Open Examples....** Then select **IoT DevKit** as board.
3. In the IoT Workbench Examples page, find **Device Registration with DPS** and click **Open Sample**. Then selects the default path to download the sample code.



## Save a Unique Device Secret on device security storage

Auto-provisioning can be configured on a device based on the device's [attestation mechanism](#). The MXChip IoT DevKit uses the [Device Identity Composition Engine](#) from the [Trusted Computing Group](#). A **Unique Device Secret** (UDS) saved in an STSAFE security chip ([STSAFE-A100](#)) on the DevKit is used to generate the device's unique [X.509 certificate](#). The certificate is used later for the enrollment process in the Device Provisioning service, and during registration at runtime.

A typical UDS is a 64-character string, as seen in the following sample:

```
19e25a259d0c2be03a02d416c05c48cc0cc7d1743458aae1cb488b074993eae
```

To save a UDS on the DevKit:

1. In VS Code, click on the status bar to select the COM port for the DevKit.

The screenshot shows the Visual Studio Code interface with the DevKitDPS.ino project open. In the top right, there is a dropdown menu labeled "Select a serial port". A red box highlights the option "COM3 STMMicroelectronics". The code editor shows the main file content, which includes #include statements for WiFi, Azure IoT Hub, MQTT Client, and DevkitDPSClient, along with configuration variables for Global\_Device\_Endpoint and ID\_Scope.

```
3 #include "AZ3166WiFi.h"
4 #include "AzureIoTHub.h"
5 #include "DevKitMQTTClient.h"
6 #include "DevkitDPSClient.h"
7
8 #include "config.h"
9 #include "utility.h"
10
11 // Input DPS instance info
12 char* Global_Device_Endpoint = "[Global_Device_Endpoint]";
13 char* ID_Scope = "[ID_Scope]";
14
15 // Input your preferred registrationId and only alphanumeric, lowercase
16 // If you leave it blank, one registrationId would be auto-generated by DPS
17 char* registrationId = "";
18
19 // Indicate whether WiFi is ready
20 static bool hasWifi = false;
21 int messageCount = 1;
22 static bool messageSending = true;
23 static uint64_t send_interval_ms;
24
25 static void InitWiFi()
26 {
27     Screen.print(2, "Connecting...");
28
29     if (WiFi.begin() == WL_CONNECTED)
```

2. On DevKit, hold down **button A**, push and release the **reset** button, and then release **button A**. Your DevKit enters configuration mode.
3. Click **F1** to open the command palette, type and select **Azure IoT Device Workbench: Configure Device Settings... > Config Unique Device String (UDS)**.

The screenshot shows the Visual Studio Code interface with the DevKitDPS.ino project open. The command palette is open, and the option "Config Unique Device String (UDS)" is highlighted with a red box. Other options like "Config Device Connection String" and "Config Connection String" are also visible.

4. Note down the generated UDS string. You will need it to generate the X.509 certificate. Then press **Enter**.

The screenshot shows the Visual Studio Code interface with the DevKitDPS.ino project open. The command palette is open, and a modal dialog box is prompting for a Unique Device String (UDS). The string "bd3a9a140f2f088c9e3ed75c120e575af4408bcae1568df70c93a03fd11300f9" is entered into the field. The modal says "Please input Unique Device String (UDS) here. (Press 'Enter' to confirm or 'Escape' to cancel)".

5. Confirm from the notification that UDS has been configured on the STSAFE successfully.

The screenshot shows the Visual Studio Code interface with the DevKitDPS.ino project open. A notification bar at the bottom displays the message "Configure Unique Device String (UDS) completely." with a red box highlighting it.

#### NOTE

Alternatively, you can configure UDS via serial port by using utilities such as Putty. Follow [Use configuration mode](#) to do so.

## Update the Global Device Endpoint and ID Scope

In device code, you need to specify the [Device provisioning endpoint](#) and ID scope to ensure the tenant isolation.

1. In the Azure portal, select the [Overview](#) pane of your Device Provisioning service and note down the **Global device endpoint** and **ID Scope** values.

The screenshot shows the Azure portal interface for a Device Provisioning service named "IoTDPSTest". The "Overview" tab is selected. Key details shown include:

- Resource group: IoT
- Status: Active
- Location: West US
- Subscription: Visual Studio Enterprise
- Service endpoint: azure-devices-provisioning.net
- Global device endpoint: global.azure-devices-provisioning.net (highlighted)
- ID Scope: One (highlighted)
- Pricing and scale tier: S1

2. Open `DevKitDPS.ino`. Find and replace `[Global_Device_Endpoint]` and `[ID_Scope]` with the values you just noted down.

The screenshot shows the Visual Studio Code editor with the file `DevKitDPS.ino` open. The code includes the following placeholder values:

```
// Copyright (c) Microsoft. All rights reserved.  
// Licensed under the MIT license.  
#include "AZ3166WiFi.h"  
#include "AzureIotHub.h"  
#include "DevKitMQTTClient.h"  
#include "DevkitDPSClient.h"  
  
#include "config.h"  
#include "utility.h"  
  
// Input DPS instance info  
char* Global_Device_Endpoint = "[Global_Device_Endpoint]";  
char* ID_Scope = "[ID_Scope]";  
  
// Input your preferred registrationId and only alphanumeric, lowercase,  
// If you leave it blank, one registrationId would be auto-generated by DPS  
char* registrationId = "";  
  
// Indicate whether WiFi is ready  
static bool hasWifi = false;  
int messageCount = 1;  
static bool messageSending = true;  
static uint64_t send_interval_ms;  
  
static void InitWiFi()  
{  
    Screen.print(2, "Connecting...");  
    if (WiFi.begin() == WL_CONNECTED)
```

3. Fill the `registrationId` variable in the code. Only alphanumeric, lowercase, and hyphen combination with a maximum of 128 characters is allowed. Also noted down the value.

The screenshot shows the Visual Studio Code editor with the file `DevKitDPS.ino` open. The `registrationId` variable has been filled with the value "mydevkit3166".

```
Global_Device_Endpoint = "IoTDPSTest.azure-devices-provisioning.net";  
ID_Scope = "One00027C0C";  
  
// Input your preferred registrationId and only alphanumeric, lowercase,  
// If you leave it blank, one registrationId would be auto-generated based on DPS  
registrationId = "mydevkit3166";  
  
// Indicate whether WiFi is ready
```

4. Click `F1`, type and select **Azure IoT Device Workbench: Upload Device Code**. It starts compiling and uploading the code to DevKit.

```

1 // Copyright (c) Microsoft. All rights reserved.
2 // Licensed under the MIT license.
3 #include "AZ3166WiFi.h"
4 #include "AzureIoTHub.h"
5 #include "DevKitMQTTClient.h"
6 #include "DevkitDPSCClient.h"
7
8 #include "config.h"
9 #include "utility.h"
10
11 #include <ut DPS instance info>
12 Global Device Endpoint = "IoTDPS-Test.azure-devices-provisioning.net";

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Arduino

Info : flash size = 1024kbytes  
wrote 737280 bytes from file  
c:\Users\liya\Documents\IoTWorkbenchProjects\examples\devkit\_dps\Device\.build\DevKitDPS.ino.bin in 17.524897s (41.084 KiB/s)  
\*\* Programming Finished \*\*  
\*\* Verify Started \*\*  
target halted due to breakpoint, current mode: Thread  
xPSR: 0x61000000 pc: 0x2000002e msp: 0x2000d650  
verified 644888 bytes in 1.194999s (527.008 KiB/s)  
\*\* Verified OK \*\*  
\*\* Resetting Target \*\*  
Info : Unable to match requested speed 2000 kHz, using 1800 kHz  
[Done] Uploaded the sketch: DevKitDPS.ino

0 A 0 Azure: liyu@microsoft.com Ln 13, Col 30 Spaces: 2 UTF-8 LF C++ <Select Programmer> DevKitDPS.ino MXCHIP AZ3166 COM3 1

## Generate X.509 certificate

The [attestation mechanism](#) used by this sample is X.509 certificate. You need to use a utility to generate it.

1. In VS Code, click **F1**, type and select **Open New Terminal** to open terminal window.
2. Run `dps_cert_gen.exe` in `tool` folder.
3. Specify the compiled binary file location as `..\build\DevKitDPS`. Then paste the `UDS` and `registrationId` you just noted down.

```
C:\Users\liya\Documents\IoTWorkbenchProjects\examples\devkit_dps\Device\tool>.\dps_cert_gen.exe
Input the name of your project, default name is "DevKitDPS" :..\build\DevKitDPS
Input the UDS you saved into security chip of your Devkit: :39f646d6abea19d74dc49+
Input your preferred registrationId as you input in DPS.ino(click Enter to skip if no registrationId provided in DPS.ino): |mydevkit3166
Writing to the Alias Key Certificate file mydevkit3166.pem is successful.
Press any key to continue:
```

4. A `.pem` X.509 certificate generates in the same folder.

```
C:\Users\liya\Documents\IoTWorkbenchProjects\examples\devkit_dps\Device\tool>dir
Volume in drive C has no label.
Volume Serial Number is 948B-6224

Directory of C:\Users\liya\Documents\IoTWorkbenchProjects\examples\devkit_dps\Device\tool

12/19/2018  05:43 PM    <DIR>      .
12/19/2018  05:43 PM    <DIR>      ..
10/18/2018  06:23 PM        53,760 dps_cert_gen.exe
10/18/2018  06:23 PM        96,404 dps_cert_gen_mac
12/19/2018  05:43 PM        814 mydevkit3166.pem
               3 File(s)     150,978 bytes
               2 Dir(s)   155,649,695,744 bytes free
```

## Create a device enrollment entry

1. In the Azure portal, open your Device Provision Service, navigate to Manage enrollments section, and click **Add individual enrollment**.

Home > IoT DPS-Test - Manage enrollments

**IoT DPS-Test - Manage enrollments**  
Device Provisioning Service

Add enrollment group + Add individual enrollment Refresh Delete

You can add or remove individual device enrollments and/or enrollment groups from this page.

Enrollment Groups Individual Enrollments

Filter enrollments

GROUP NAME

No results

- Click file icon next to Primary Certificate .pem or .cer file to upload the .pem file generated.

Home > IoT DPS-Test - Manage enrollments > Add Enrollment

**Add Enrollment**

Save

\* Mechanism X.509

Primary Certificate .pem or .cer file mydevkit3166.pem

Clear Selection

## Verify the DevKit is registered with Azure IoT Hub

Press the **Reset** button on your DevKit. You should see **DPS Connected!** on DevKit screen. After the device reboots, the following actions take place:

- The device sends a registration request to your Device Provisioning service.
- The Device Provisioning service sends back a registration challenge to which your device responds.
- On successful registration, the Device Provisioning service sends the IoT Hub URI, device ID, and the encrypted key back to the device.
- The IoT Hub client application on the device connects to your hub.
- On successful connection to the hub, you see the device appear in the IoT Hub Device Explorer.

d2s166-iotduo-c0c3 - IoT Devices

IoT Hub

Search (Ctrl+ /)

+ Add Columns Refresh Delete

You can use this tool to view, create, update, and delete devices on your IoT Hub.

Query SELECT \* FROM devices WHERE optional (e.g. tags.location='US')

Execute

Filter by Device Id

DEVICE ID	STATUS
AZ3166	enabled
Liya-DevKit	enabled
<Device ID>	enabled

## Problems and feedback

If you encounter problems, refer to the IoT DevKit [FAQs](#), or reach out to the following channels for support:

- [Gitter.im](#)
- [Stack Overflow](#)

## Next steps

In this tutorial, you learned to enroll a device securely to the Device Provisioning Service by using the Device Identity Composition Engine, so that the device can automatically register with Azure IoT Hub.

In summary, you learned how to:

- Configure the global endpoint of the Device Provisioning service on a device.
- Use a unique device secret to generate an X.509 certificate.
- Enroll an individual device.
- Verify that the device is registered.

Learn how to [Create and provision a simulated device](#).

# How to manage device enrollments with Azure Portal

12/10/2019 • 3 minutes to read • [Edit Online](#)

A *device enrollment* creates a record of a single device or a group of devices that may at some point register with the Azure IoT Hub Device Provisioning Service. The enrollment record contains the initial desired configuration for the device(s) as part of that enrollment, including the desired IoT hub. This article shows you how to manage device enrollments for your provisioning service.

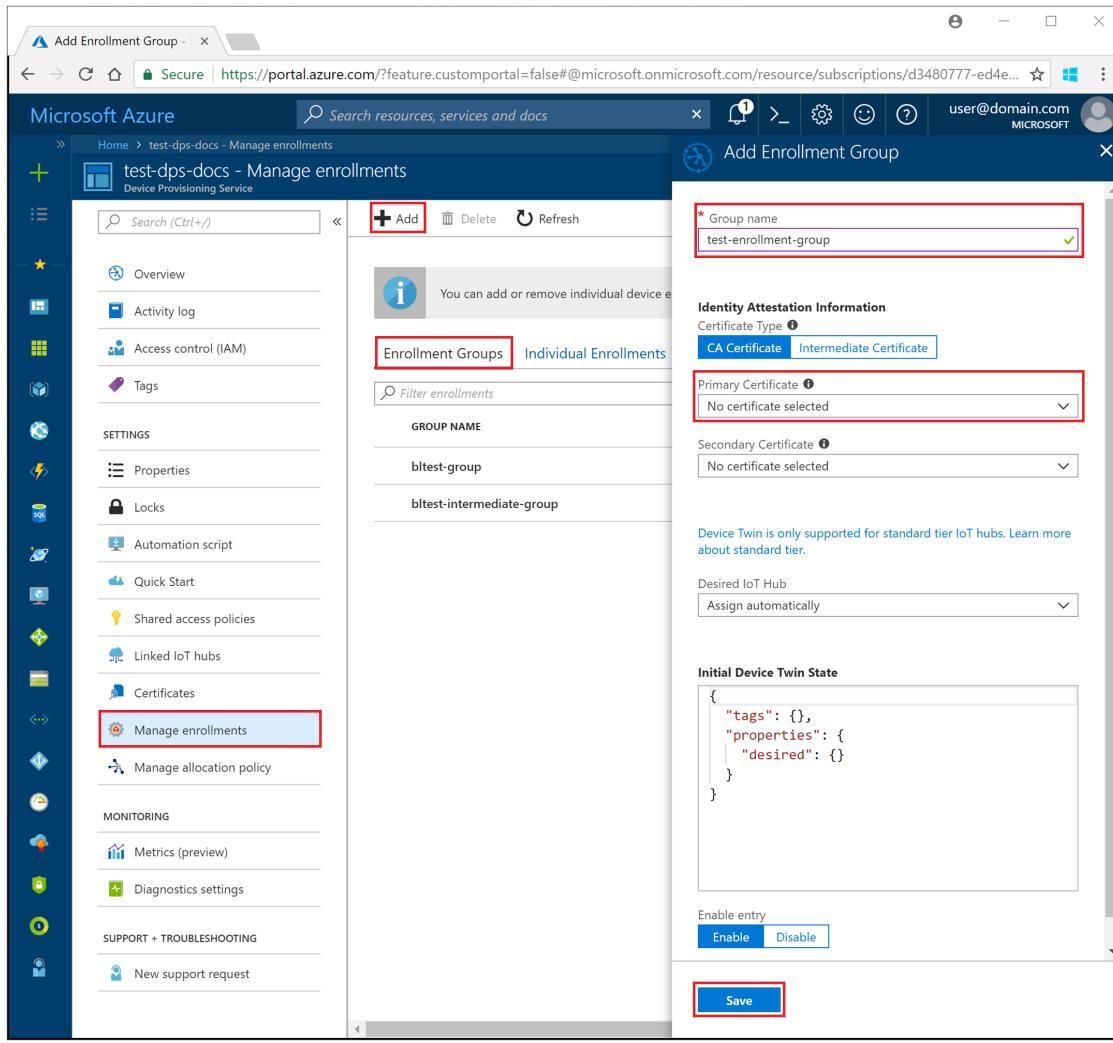
## Create a device enrollment

There are two ways you can enroll your devices with the provisioning service:

- An **Enrollment group** is an entry for a group of devices that share a common attestation mechanism of X.509 certificates, signed by the same signing certificate, which can be the [root certificate](#) or the [intermediate certificate](#), used to produce device certificate on physical device. We recommend using an enrollment group for a large number of devices which share a desired initial configuration, or for devices all going to the same tenant. Note that you can only enroll devices that use the X.509 attestation mechanism as *enrollment groups*.

You can create an enrollment group in the portal for a group of devices using the following steps:

1. Log in to the Azure portal and click **All resources** from the left-hand menu.
2. Click the Device Provisioning service you want to enroll your device to from the list of resources.
3. In your provisioning service:
  - a. Click **Manage enrollments**, then select the **Enrollment Groups** tab.
  - b. Click the **Add** button at the top.
  - c. When the "Add Enrollment Group" panel appears, enter the information for the enrollment list entry. **Group name** is required. Also select "CA or Intermediate" for **Certificate type**, and upload the root **Primary certificate** for the group of devices.
  - d. Click **Save**. On successful creation of your enrollment group, you should see the group name appear under the **Enrollment Groups** tab.



- An **Individual enrollment** is an entry for a single device that may register. Individual enrollments may use either x509 certificates or SAS tokens (from a physical or virtual TPM) as attestation mechanisms. We recommend using individual enrollments for devices which require unique initial configurations, or for devices which can only use SAS tokens via TPM or virtual TPM as the attestation mechanism. Individual enrollments may have the desired IoT hub device ID specified.

You can create an individual enrollment in the portal using the following steps:

- Log in to the Azure portal and click **All resources** from the left-hand menu.
- Click the Device Provisioning service you want to enroll your device to from the list of resources.
- In your provisioning service:
  - Click Manage enrollments**, then select the **Individual Enrollments** tab.
  - Click the Add button** at the top.
  - When the "Add Enrollment" panel appears, enter the information for the enrollment list entry. First select the attestation **Mechanism** for the device (X.509 or TPM). X.509 attestation requires you to upload the leaf **Primary certificate** for the device. TPM requires you to enter the **Attestation Key** and **Registration ID** for the device.
  - Click Save**. On successful creation of your enrollment group, you should see your device appear under the **Individual Enrollments** tab.

The screenshot shows the 'Add Enrollment' dialog in the Azure portal. The 'Mechanism' field is set to 'X.509'. The 'Primary Certificate (.pem or .cer file)' field has a 'Select a file' button. The 'Secondary Certificate (.pem or .cer file)' field also has a 'Select a file' button. The 'IoT Hub' dropdown is set to 'Assign automatically'. The 'IoT Hub Device ID' input field contains 'Device ID'. The 'Initial Device Twin State' section contains JSON code: 
 

```
{
        "tags": {},
        "properties": {
            "desired": {}
        }
    }
```

 There are 'Enable' and 'Disable' buttons for entry enablement. A red box highlights the 'Save' button at the bottom right.

## Update an enrollment entry

You can update an existing enrollment entry in the portal using the following steps:

1. Open your Device Provisioning service in the Azure portal and click **Manage Enrollments**.
2. Navigate to the enrollment entry you want to modify. Click the entry, which opens a summary information about your device enrollment.
3. On this page, you can modify items other than the security type and credentials, such as the IoT hub the device should be linked to, as well as the device ID. You may also modify the initial device twin state.
4. Once completed, click **Save** to update your device enrollment.

The screenshot shows the 'Enrollment list entry' dialog for an existing enrollment. The 'Registration status' section shows 'Status: Unassigned', 'Assigned hub: -', 'Device ID: -', and 'Last assigned: -'. The 'Identity attestation information' section shows 'Mechanism: TPM' and 'Endorsement key: [redacted]'. The 'IoT Hub' section shows 'test-hub-docs.azure-devices.net' selected in a dropdown. The 'IoT Hub device ID' input field contains 'test-device-docs'. The 'Initial device twin state' section contains JSON code: 
 

```
{
        "tags": {},
        "desiredproperties": {}
    }
```

 A red box highlights the 'Save' button at the bottom right.

## Remove a device enrollment

In cases where your device(s) do not need to be provisioned to any IoT hub, you can remove the related enrollment entry in the portal using the following steps:

1. Open your Device Provisioning service in the Azure portal and click **Manage Enrollments**.
2. Navigate to and select the enrollment entry you want to remove.
3. Click the **Delete** button at the top and then select **Yes** when prompted to confirm.
4. Once the action is completed, you will see your entry removed from the list of device enrollments.

The screenshot shows the 'Manage enrollments' page in the Azure Device Provisioning Service. The left sidebar has a 'SETTINGS' section with 'Manage enrollments' highlighted in blue. The main area has a search bar and a toolbar with 'Add', 'Delete' (which is highlighted with a red box), 'Refresh', and 'Columns'. A modal window titled 'Remove enrollment' contains 'Yes' and 'No' buttons, with 'Yes' highlighted with a red box. Below the modal is a table with columns 'REGISTRATION ID' and 'STATUS'. A row in the table is highlighted with a red box, indicating the target for deletion. The table includes a 'Load more' link at the bottom right.

# How to manage device enrollments with Azure Device Provisioning Service SDKs

12/10/2019 • 4 minutes to read • [Edit Online](#)

A *device enrollment* creates a record of a single device or a group of devices that may at some point register with the Device Provisioning Service. The enrollment record contains the initial desired configuration for the device(s) as part of that enrollment, including the desired IoT hub. This article shows you how to manage device enrollments for your provisioning service programmatically using the Azure IoT Provisioning Service SDKs. The SDKs are available on GitHub in the same repository as Azure IoT SDKs.

## Prerequisites

- Obtain the connection string from your Device Provisioning Service instance.
- Obtain the device security artifacts for the [attestation mechanism](#) used:
  - **Trusted Platform Module (TPM):**
    - Individual enrollment: Registration ID and TPM Endorsement Key from a physical device or from TPM Simulator.
    - Enrollment group does not apply to TPM attestation.
  - **X.509:**
    - Individual enrollment: The [Leaf certificate](#) from physical device or from the SDK [DICE](#) Emulator.
    - Enrollment group: The [CA/root certificate](#) or the [intermediate certificate](#), used to produce device certificate on a physical device. It can also be generated from the SDK DICE emulator.
- Exact API calls may be different due to language differences. Please review the samples provided on GitHub for details:
  - [Java Provisioning Service Client samples](#)
  - [Node.js Provisioning Service Client samples](#)
  - [.NET Provisioning Service Client samples](#)

## Create a device enrollment

There are two ways you can enroll your devices with the provisioning service:

- An **Enrollment group** is an entry for a group of devices that share a common attestation mechanism of X.509 certificates, signed by the [root certificate](#) or the [intermediate certificate](#). We recommend using an enrollment group for a large number of devices that share a desired initial configuration, or for devices all going to the same tenant. Note that you can only enroll devices that use the X.509 attestation mechanism as *enrollment groups*.

You can create an enrollment group with the SDKs following this workflow:

1. For enrollment group, the attestation mechanism uses X.509 root certificate. Call Service SDK API `X509Attestation.createFromRootCertificate` with root certificate to create attestation for enrollment. X.509 root certificate is provided in either a PEM file or as a string.
2. Create a new `EnrollmentGroup` variable using the `attestation` created and a unique `enrollmentGroupId`. Optionally, you can set parameters like `Device ID`, `IoTHubHostName`, `ProvisioningStatus`.
3. Call Service SDK API `createOrUpdateEnrollmentGroup` in your backend application with `EnrollmentGroup` to create an enrollment group.

- An **Individual enrollment** is an entry for a single device that may register. Individual enrollments may use either X.509 certificates or SAS tokens (from a physical or virtual TPM) as attestation mechanisms. We recommend using individual enrollments for devices that require unique initial configurations, or for devices which can only use SAS tokens via TPM or virtual TPM as the attestation mechanism. Individual enrollments may have the desired IoT hub device ID specified.

You can create an individual enrollment with the SDKs following this workflow:

- Choose your `attestation` mechanism, which can be TPM or X.509.
  - TPM:** Using the Endorsement Key from a physical device or from TPM Simulator as the input, you can call Service SDK API `TpmAttestation` to create attestation for enrollment.
  - X.509:** Using the client certificate as the input, you can call Service SDK API `X509Attestation.createFromClientCertificate` to create attestation for enrollment.
- Create a new `IndividualEnrollment` variable with using the `attestation` created and a unique `registrationId` as input, which is on your device or generated from the TPM Simulator. Optionally, you can set parameters like `Device ID`, `IoTHubHostName`, `ProvisioningStatus`.
- Call Service SDK API `createOrUpdateIndividualEnrollment` in your backend application with `IndividualEnrollment` to create an individual enrollment.

After you have successfully created an enrollment, the Device Provisioning Service returns an enrollment result. This workflow is demonstrated in the samples [mentioned previously](#).

## Update an enrollment entry

After you have created an enrollment entry, you may want to update the enrollment. Potential scenarios include updating the desired property, updating the attestation method, or revoking device access. There are different APIs for individual enrollment and group enrollment, but no distinction for attestation mechanism.

You can update an enrollment entry following this workflow:

- Individual enrollment:**

- Get the latest enrollment from the provisioning service first with Service SDK API `getIndividualEnrollment`.
- Modify the parameter of the latest enrollment as necessary.
- Using the latest enrollment, call Service SDK API `createOrUpdateIndividualEnrollment` to update your enrollment entry.

- Group enrollment:**

- Get the latest enrollment from the provisioning service first with Service SDK API `getEnrollmentGroup`.
- Modify the parameter of the latest enrollment as necessary.
- Using the latest enrollment, call Service SDK API `createOrUpdateEnrollmentGroup` to update your enrollment entry.

This workflow is demonstrated in the samples [mentioned previously](#).

## Remove an enrollment entry

- Individual enrollment** can be deleted by calling Service SDK API `deleteIndividualEnrollment` using `registrationId`.
- Group enrollment** can be deleted by calling Service SDK API `deleteEnrollmentGroup` using `enrollmentGroupId`.

This workflow is demonstrated in the samples [mentioned previously](#).

## Bulk operation on individual enrollments

You can perform bulk operation to create, update, or remove multiple individual enrollments following this workflow:

1. Create a variable that contains multiple `IndividualEnrollment`. Implementation of this variable is different for every language. Review the bulk operation sample on GitHub for details.
2. Call Service SDK API `runBulkOperation` with a `BulkOperationMode` for desired operation and your variable for individual enrollments. Four modes are supported: create, update, updateIfMatchEtag, and delete.

After you have successfully performed an operation, the Device Provisioning Service would return a bulk operation result.

This workflow is demonstrated in the samples [mentioned previously](#).

# How to do proof-of-possession for X.509 CA certificates with your Device Provisioning Service

12/10/2019 • 4 minutes to read • [Edit Online](#)

A verified X.509 Certificate Authority (CA) certificate is a CA certificate that has been uploaded and registered to your provisioning service and has gone through proof-of-possession with the service.

Proof-of-possession involves the following steps:

1. Get a unique verification code generated by the provisioning service for your X.509 CA certificate. You can do this from the Azure portal.
2. Create an X.509 verification certificate with the verification code as its subject and sign the certificate with the private key associated with your X.509 CA certificate.
3. Upload the signed verification certificate to the service. The service validates the verification certificate using the public portion of the CA certificate to be verified, thus proving that you are in possession of the CA certificate's private key.

Verified certificates play an important role when using enrollment groups. Verifying certificate ownership provides an additional security layer by ensuring that the uploader of the certificate is in possession of the certificate's private key. Verification prevents a malicious actor sniffing your traffic from extracting an intermediate certificate and using that certificate to create an enrollment group in their own provisioning service, effectively hijacking your devices. By proving ownership of the root or an intermediate certificate in a certificate chain, you're proving that you have permission to generate leaf certificates for the devices that will be registering as a part of that enrollment group. For this reason, the root or intermediate certificate configured in an enrollment group must either be a verified certificate or must roll up to a verified certificate in the certificate chain a device presents when it authenticates with the service. To learn more about enrollment groups, see [X.509 certificates](#) and [Controlling device access to the provisioning service with X.509 certificates](#).

## Register the public part of an X.509 certificate and get a verification code

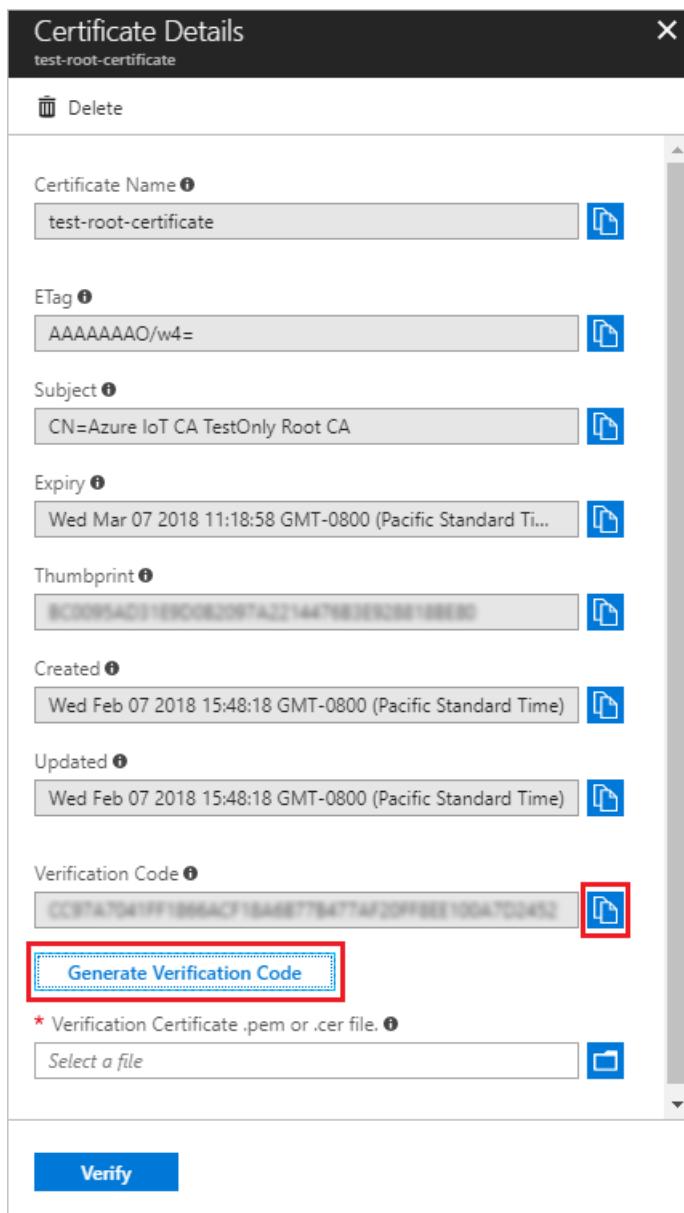
To register a CA certificate with your provisioning service and get a verification code that you can use during proof-of-possession, follow these steps.

1. In the Azure portal, navigate to your provisioning service and open **Certificates** from the left-hand menu.
2. Click **Add** to add a new certificate.
3. Enter a friendly display name for your certificate. Browse to the .cer or .pem file that represents the public part of your X.509 certificate. Click **Upload**.
4. Once you get a notification that your certificate is successfully uploaded, click **Save**.

The screenshot shows the Microsoft Azure portal interface. The left sidebar lists various service options, and the main area is titled "sample-provisioning-service - Certificates". A sub-menu on the left shows "Certificates" selected. The right pane is titled "Add Certificate" and contains fields for "Certificate Name" (set to "test-root-certificate") and "Certificate .pem or .cer file" (set to "RootCA.cer"). A "Save" button is at the bottom.

Your certificate will show in the **Certificate Explorer** list. Note that the **STATUS** of this certificate is *Unverified*.

5. Click on the certificate that you added in the previous step.
6. In **Certificate Details**, click **Generate Verification Code**.
7. The provisioning service creates a **Verification Code** that you can use to validate the certificate ownership.  
Copy the code to your clipboard.



## Digitally sign the verification code to create a verification certificate

Now, you need to sign the *Verification Code* with the private key associated with your X.509 CA certificate, which generates a signature. This is known as [Proof of possession](#) and results in a signed verification certificate.

Microsoft provides tools and samples that can help you create a signed verification certificate:

- The [Azure IoT Hub C SDK](#) provides PowerShell (Windows) and Bash (Linux) scripts to help you create CA and leaf certificates for development and to perform proof-of-possession using a verification code. You can download the [files](#) relevant to your system to a working folder and follow the instructions in the [Managing CA certificates readme](#) to perform proof-of-possession on a CA certificate.
- The [Azure IoT Hub C# SDK](#) contains the [Group Certificate Verification Sample](#), which you can use to do proof-of-possession.

### IMPORTANT

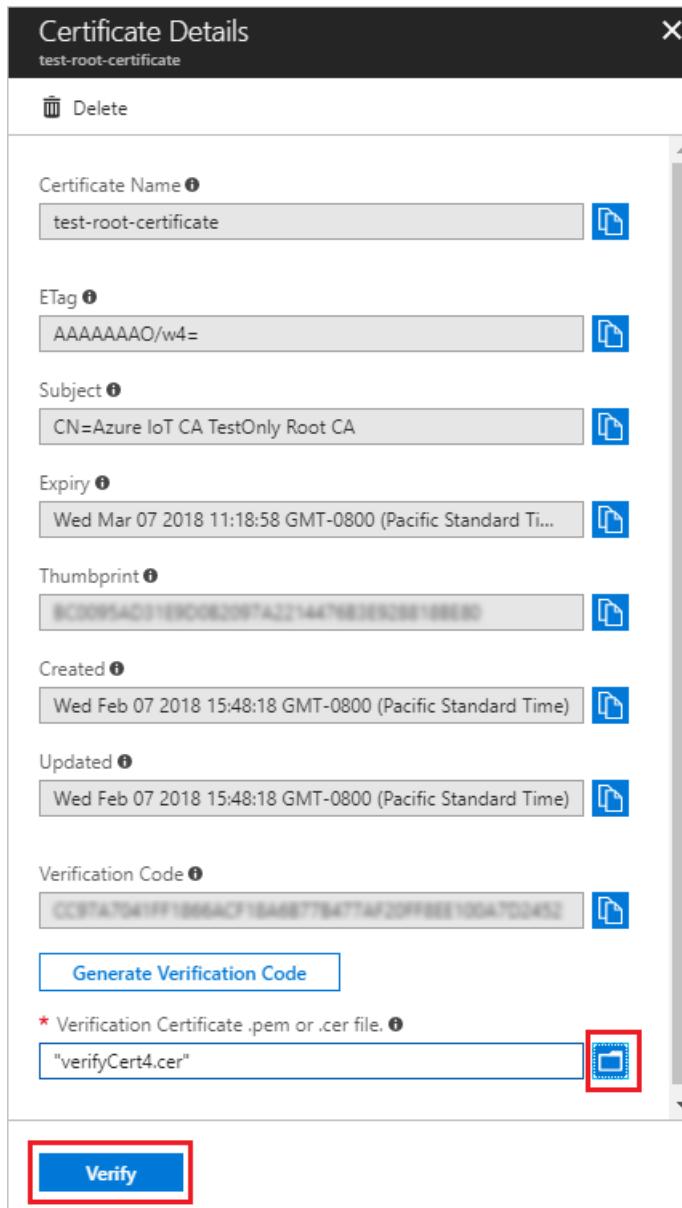
In addition to performing proof-of-possession, the PowerShell and Bash scripts cited previously also allow you to create root certificates, intermediate certificates, and leaf certificates that can be used to authenticate and provision devices. These certificates should be used for development only. They should never be used in a production environment.

The PowerShell and Bash scripts provided in the documentation and SDKs rely on [OpenSSL](#). You may also use

OpenSSL or other third-party tools to help you do proof-of-possession. For more information about tooling provided with the SDKs, see [How to use tools provided in the SDKs](#).

## Upload the signed verification certificate

1. Upload the resulting signature as a verification certificate to your provisioning service in the portal. In **Certificate Details** on the Azure portal, use the *File Explorer* icon next to the **Verification Certificate .pem or .cer file** field to upload the signed verification certificate from your system.
2. Once the certificate is successfully uploaded, click **Verify**. The **STATUS** of your certificate changes to **Verified** in the **Certificate Explorer** list. Click **Refresh** if it does not update automatically.



## Next steps

- To learn about how to use the portal to create an enrollment group, see [Managing device enrollments with Azure portal](#).
- To learn about how to use the service SDKs to create an enrollment group, see [Managing device enrollments with service SDKs](#).

# How to roll X.509 device certificates

12/10/2019 • 11 minutes to read • [Edit Online](#)

During the lifecycle of your IoT solution, you'll need to roll certificates. Two of the main reasons for rolling certificates would be a security breach, and certificate expirations.

Rolling certificates is a security best practice to help secure your system in the event of a breach. As part of [Assume Breach Methodology](#), Microsoft advocates the need for having reactive security processes in place along with preventative measures. Rolling your device certificates should be included as part of these security processes. The frequency in which you roll your certificates will depend on the security needs of your solution. Customers with solutions involving highly sensitive data may roll certificate daily, while others roll their certificates every couple years.

Rolling device certificates will involve updating the certificate stored on the device and the IoT hub. Afterwards, the device can reprovision itself with the IoT hub using normal [auto-provisioning](#) with the Device Provisioning Service.

## Obtain new certificates

There are many ways to obtain new certificates for your IoT devices. These include obtaining certificates from the device factory, generating your own certificates, and having a third party manage certificate creation for you.

Certificates are signed by each other to form a chain of trust from a root CA certificate to a [leaf certificate](#). A signing certificate is the certificate used to sign the leaf certificate at the end of the chain of trust. A signing certificate can be a root CA certificate, or an intermediate certificate in chain of trust. For more information, see [X.509 certificates](#).

There are two different ways to obtain a signing certificate. The first way, which is recommended for production systems, is to purchase a signing certificate from a root certificate authority (CA). This way chains security down to a trusted source.

The second way is to create your own X.509 certificates using a tool like OpenSSL. This approach is great for testing X.509 certificates but provides few guarantees around security. We recommend you only use this approach for testing unless you prepared to act as your own CA provider.

## Roll the certificate on the device

Certificates on a device should always be stored in a safe place like a [hardware security module \(HSM\)](#). The way you roll device certificates will depend on how they were created and installed in the devices in the first place.

If you got your certificates from a third party, you must look into how they roll their certificates. The process may be included in your arrangement with them, or it may be a separate service they offer.

If you're managing your own device certificates, you'll have to build your own pipeline for updating certificates. Make sure both old and new leaf certificates have the same common name (CN). By having the same CN, the device can reprovision itself without creating a duplicate registration record.

## Roll the certificate in the IoT hub

The device certificate can be manually added to an IoT hub. The certificate can also be automated using a Device Provisioning service instance. In this article, we'll assume a Device Provisioning service instance is being used to support auto-provisioning.

When a device is initially provisioned through auto-provisioning, it boots-up, and contacts the provisioning service. The provisioning service responds by performing an identity check before creating a device identity in an IoT hub

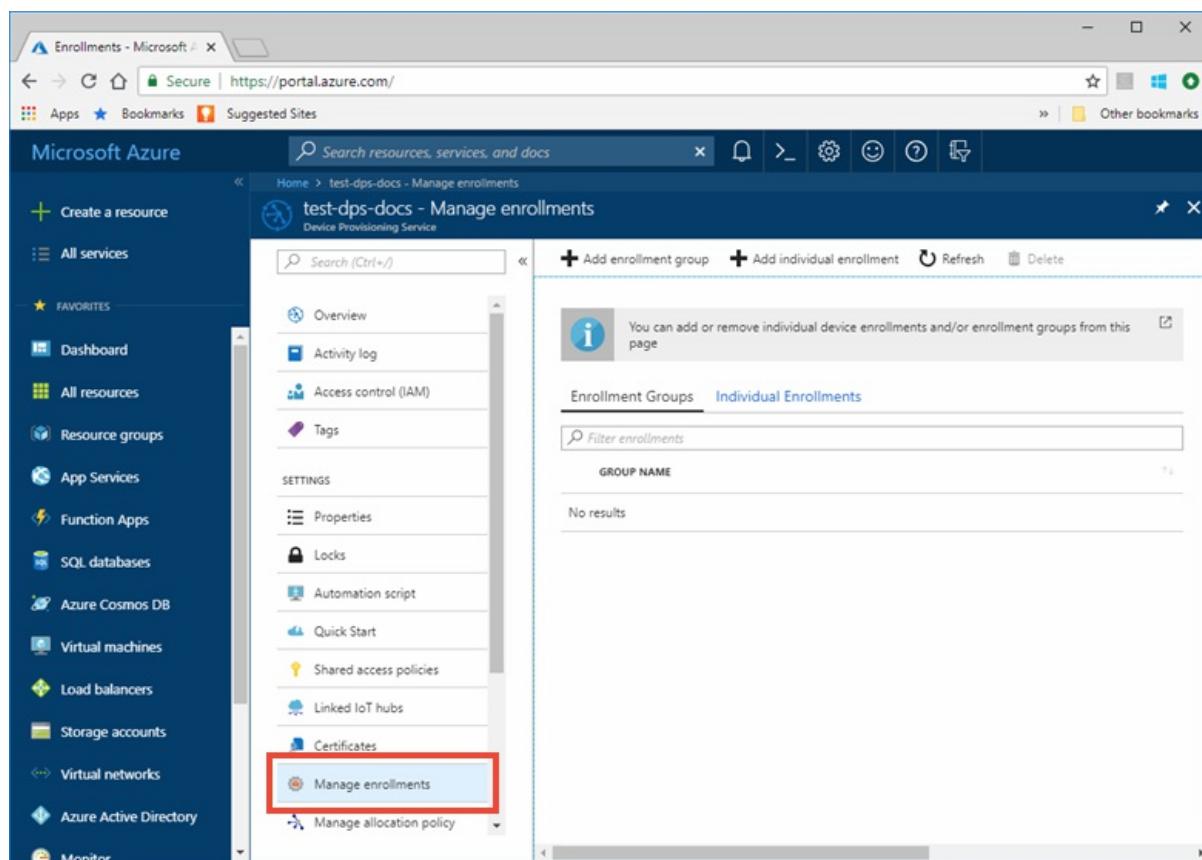
using the device's leaf certificate as the credential. The provisioning service then tells the device which IoT hub it's assigned to, and the device then uses its leaf certificate to authenticate and connect to the IoT hub.

Once a new leaf certificate has been rolled to the device, it can no longer connect to the IoT hub because it's using a new certificate to connect. The IoT hub only recognizes the device with the old certificate. The result of the device's connection attempt will be an "unauthorized" connection error. To resolve this error, you must update the enrollment entry for the device to account for the device's new leaf certificate. Then the provisioning service can update the IoT Hub device registry information as needed when the device is reprovisioned.

One possible exception to this connection failure would be a scenario where you've created an [Enrollment Group](#) for your device in the provisioning service. In this case, if you aren't rolling the root or intermediate certificates in the device's certificate chain of trust, then the device will be recognized if the new certificate is part of the chain of trust defined in the enrollment group. If this scenario arises as a reaction to a security breach, you should at least blacklist the specific device certificates in the group that are considered to be breached. For more information, see [Blacklist specific devices in an enrollment group](#).

Updating enrollment entries for rolled certificates is accomplished on the [Manage enrollments](#) page. To access that page, follow these steps:

1. Sign in to the [Azure portal](#) and navigate to the IoT Hub Device Provisioning Service instance that has the enrollment entry for your device.
2. Click **Manage enrollments**.



How you handle updating the enrollment entry will depend on whether you're using individual enrollments, or group enrollments. Also the recommended procedures differ depending on whether you're rolling certificates because of a security breach, or certificate expiration. The following sections describe how to handle these updates.

## Individual enrollments and security breaches

If you're rolling certificates in response to a security breach, you should use the following approach that deletes the current certificate immediately:

1. Click **Individual Enrollments**, and click the registration ID entry in the list.
2. Click the **Delete current certificate** button and then, click the folder icon to select the new certificate to be uploaded for the enrollment entry. Click **Save** when finished.

These steps should be completed for the primary and secondary certificate, if both are compromised.

The screenshot shows the 'test-dps-docs - Manage enrollments' blade. On the left, there's a navigation menu with items like Overview, Activity log, Access control (IAM), Tags, Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, Manage enrollments (which is selected and highlighted in blue), Manage allocation policy, Metrics (preview), Diagnostics settings, and New support request. The main area has tabs for 'Enrollment Groups' and 'Individual Enrollments'. Under 'Individual Enrollments', there's a 'REGISTRATION ID' section with a list containing 'abcde...g'. Below this is an 'Identity Attestation Information' section with fields for Primary Certificate (thumbprint and .pem/.cer file) and Secondary Certificate (IoT Hub assignment). There's also an 'Initial Device Twin State' JSON editor and 'Enable'/'Disable' buttons. The 'Save' button at the bottom is also highlighted with a red box.

3. Once the compromised certificate has been removed from the provisioning service, the certificate can still be used to make device connections to the IoT hub as long as a device registration for it exists there. You can address this two ways:

The first way would be to manually navigate to your IoT hub and immediately remove the device registration associated with the compromised certificate. Then when the device provisions again with an updated certificate, a new device registration will be created.

The screenshot shows the 'ExampleIoTHub - IoT devices' blade in the Azure portal. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Events, SETTINGS (Shared access policies, Pricing and scale, Operations monitoring, IP Filter, Certificates, Properties, Locks, Automation script), EXPLORERS (Query explorer, IoT devices). The main area has a toolbar with Add, Refresh, and Delete buttons (Delete is highlighted with a red box). A query editor window displays a sample SQL query: 'SELECT \* FROM devices WHERE optional (e.g. tags.location='US')'. Below it is a table listing devices:

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS ...	AUTHENTICA...	CLOUD TO DE...
AZ3166	Enabled	Wed Jul 18 ...		SelfSigned	0

The 'AZ3166' row is also highlighted with a red box. The entire 'IoT devices' section in the sidebar is also highlighted with a red box.

The second way would be to use reprovisioning support to reprovision the device to the same IoT hub. This approach can be used to replace the certificate for the device registration on the IoT hub. For more information, see [How to reprovision devices](#).

## Individual enrollments and certificate expiration

If you're rolling certificates to handle certificate expirations, you should use the secondary certificate configuration as follows to reduce downtime for devices attempting to provision.

Later when the secondary certificate also nears expiration, and needs to be rolled, you can rotate to using the primary configuration. Rotating between the primary and secondary certificates in this way reduces downtime for devices attempting to provision.

1. Click **Individual Enrollments**, and click the registration ID entry in the list.
2. Click **Secondary Certificate** and then, click the folder icon to select the new certificate to be uploaded for the enrollment entry. Click **Save**.

The screenshot shows the Azure Device Provisioning Service interface. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, Manage enrollments (which is selected), Manage allocation policy, Metrics (preview), Diagnostics settings, and New support request. The main content area has tabs for 'Enrollment Groups' and 'Individual Enrollments' (which is selected). Below these are sections for 'REGISTRATION ID' (with a checked checkbox for 'abcdefg') and 'Secondary Certificate' (with a red box around the 'Secondary Certificate Thumbprint' section and another red box around the 'Select a file' input field). At the bottom right, there's a 'Save' button.

- Later when the primary certificate has expired, come back and delete that primary certificate by clicking the **Delete current certificate** button.

## Enrollment groups and security breaches

To update a group enrollment in response to a security breach, you should use one of the following approaches that will delete the current root CA, or intermediate certificate immediately.

### Update compromised root CA certificates

- Click the **Certificates** tab for your Device Provisioning service instance.
- Click the compromised certificate in the list, and then click the **Delete** button. Confirm the delete by entering the certificate name and click **OK**. Repeat this process for all compromised certificates.

The screenshot shows the Azure Device Provisioning Service (DPS) Certificates blade. On the left, a navigation menu includes options like Overview, Activity log, Access control (IAM), Tags, Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates (which is selected and highlighted with a red box), and Manage enrollments. The main area displays a table titled 'Certificate Details' for a certificate named 'MyRootCert'. The table includes fields for Certificate Name (MyRootCert), ETag (ABCD...), Subject (O=MSR\_TEST, C=US, CN=riot-device...), Expiry (Wed Dec 31 2036 19:00:00 GMT-0500), Thumbprint (ABCDEF...), Created (Mon Aug 06 2018 04:48:06 GMT-0400), and Updated (Mon Aug 06 2018 04:48:06 GMT-0400). A large red box highlights the 'Delete' button at the top right of the details panel.

3. Follow steps outlined in [Configure verified CA certificates](#) to add and verify new root CA certificates.
4. Click the **Manage enrollments** tab for your Device Provisioning service instance, and click the **Enrollment Groups** list. Click your enrollment group name in the list.
5. Click **CA Certificate**, and select your new root CA certificate. Then click **Save**.

The screenshot shows the 'Enrollment Group Details' page for an enrollment group named 'example'. The 'Settings' tab is selected. A red box highlights the 'Save' button at the top left. Another red box highlights the 'CA Certificate' option under 'Certificate Type'. A third red box highlights the 'NewRootCA' certificate listed under 'Primary Certificate'.

example  
Enrollment Group Details

Save Refresh

Settings Registration Records

**i** You can view and update attestation information, set desired IoT Hub, and set the initial twin state of provisioning devices

Identity Attestation Information  
x509

Certificate Type ⓘ  
**CA Certificate** Intermediate Certificate

Primary Certificate ⓘ  
NewRootCA  
No certificate selected  
NewRootCA

Desired IoT Hub  
Assign automatically

Enable entry ⓘ  
**Enable** Disable

- Once the compromised certificate has been removed from the provisioning service, the certificate can still be used to make device connections to the IoT hub as long as device registrations for it exists there. You can address this two ways:

The first way would be to manually navigate to your IoT hub and immediately remove the device registration associated with the compromised certificate. Then when your devices provision again with updated certificates, a new device registration will be created for each one.

The screenshot shows the 'ExampleIoTHub - IoT devices' blade in the Azure portal. The left sidebar contains navigation links: Overview, Activity log, Access control (IAM), Tags, Events, SETTINGS (Shared access policies, Pricing and scale, Operations monitoring, IP Filter, Certificates, Properties, Locks, Automation script), EXPLORERS (Query explorer, IoT devices). The main area has a toolbar with Add, Refresh, and Delete buttons (Delete is highlighted with a red box). A query editor window is open with the text: 'Query: SELECT \* FROM devices WHERE optional (e.g. tags.location='US')' and an Execute button. Below is a table of device details:

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS ...	AUTHENTICA...	CLOUD TO DE...
AZ3166	Enabled	Wed Jul 18 ...		SelfSigned	0

The 'AZ3166' row is also highlighted with a red box.

The second way would be to use reprovisioning support to reprovision your devices to the same IoT hub. This approach can be used to replace certificates for device registrations on the IoT hub. For more information, see [How to reprovision devices](#).

#### Update compromised intermediate certificates

1. Click **Enrollment Groups**, and then click the group name in the list.
2. Click **Intermediate Certificate**, and **Delete current certificate**. Click the folder icon to navigate to the new intermediate certificate to be uploaded for the enrollment group. Click **Save** when you're finished. These steps should be completed for both the primary and secondary certificate, if both are compromised.

This new intermediate certificate should be signed by a verified root CA certificate that has already been added into provisioning service. For more information, see [X.509 certificates](#).

The screenshot shows the 'Enrollment Group Details' page with the following interface elements:

- Top Bar:** Includes a logo, the text 'example', and buttons for 'Save' (highlighted with a red box) and 'Refresh'.
- Navigation:** Tabs for 'Settings' (selected) and 'Registration Records'.
- Information Panel:** A grey box with an info icon containing text about viewing and updating attestation information, setting IoT Hub, and provisioning device initial twin state.
- Identity Attestation Information:** Shows 'x509'.
- Certificate Type:** A dropdown menu with 'CA Certificate' (highlighted with a red box) and 'Intermediate Certificate' selected.
- Primary Certificate Section:** Contains:
  - A button labeled 'Delete current certificate' (highlighted with a red box).
  - Text: 'Primary Certificate Common Name: CN=riot-device-cert, C=US, O=MSR\_TEST', 'Primary Certificate Thumbprint: ABCDEFABCDEFABCDEFABCDEFABCDEFABCD', and 'Primary Certificate Expiration: 2018-01-01T00:00:00Z'.
  - A file input field labeled 'Primary Certificate .pem or .cer file' with a 'Select a file' button (highlighted with a red box) and a 'Clear Selection' button.
- Secondary Certificate Section:** Contains a collapsed section labeled 'Secondary Certificate'.

- Once the compromised certificate has been removed from the provisioning service, the certificate can still be used to make device connections to the IoT hub as long as device registrations for it exists there. You can address this two ways:

The first way would be to manually navigate to your IoT hub and immediately remove the device registration associated with the compromised certificate. Then when your devices provision again with updated certificates, a new device registration will be created for each one.

The screenshot shows the 'ExampleIoTHub - IoT devices' blade in the Azure portal. On the left, a navigation menu includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Events', 'SETTINGS' (with options like 'Shared access policies', 'Pricing and scale', 'Operations monitoring', 'IP Filter', 'Certificates', 'Properties', 'Locks', and 'Automation script'), 'EXPLORERS' (with 'Query explorer' and 'IoT devices'), and a 'Help & feedback' link. The main area has a toolbar with 'Add', 'Refresh', and 'Delete' buttons, with 'Delete' highlighted by a red box. A query editor window displays a sample query: 'SELECT \* FROM devices WHERE optional (e.g. tags.location='US')'. Below it is a table listing devices:

DEVICE ID	STATUS	LAST ACTIVITY	LAST STATUS ...	AUTHENTICA...	CLOUD TO DE...
AZ3166	Enabled	Wed Jul 18 ...		SelfSigned	0

The 'AZ3166' row is also highlighted with a red box. The 'IoT devices' item in the 'EXPLORERS' section of the sidebar is also highlighted with a red box.

The second way would be to use reprovisioning support to reprovision your devices to the same IoT hub. This approach can be used to replace certificates for device registrations on the IoT hub. For more information, see [How to reprovision devices](#).

## Enrollment groups and certificate expiration

If you are rolling certificates to handle certificate expirations, you should use the secondary certificate configuration as follows to ensure no downtime for devices attempting to provision.

Later when the secondary certificate also nears expiration, and needs to be rolled, you can rotate to using the primary configuration. Rotating between the primary and secondary certificates in this way ensures no downtime for devices attempting to provision.

### Update expiring root CA certificates

1. Follow steps outlined in [Configure verified CA certificates](#) to add and verify new root CA certificates.
2. Click the **Manage enrollments** tab for your Device Provisioning service instance, and click the **Enrollment Groups** list. Click your enrollment group name in the list.
3. Click **CA Certificate**, and select your new root CA certificate under the **Secondary Certificate** configuration. Then click **Save**.

You can view and update attestation information, set desired IoT Hub, and set the initial twin state of provisioning devices

Identity Attestation Information  
x509

Certificate Type ⓘ  
CA Certificate **Intermediate Certificate**

Primary Certificate ⓘ  
ExpiringCert

Secondary Certificate ⓘ  
No certificate selected  
No certificate selected  
ExpiringCert  
NewRootCA

Assign automatically

Enable entry ⓘ  
Enable **Disable**

- Later when the primary certificate has expired, click the **Certificates** tab for your Device Provisioning service instance. Click the expired certificate in the list, and then click the **Delete** button. Confirm the delete by entering the certificate name, and click **OK**.

Home > test-dps-docs - Certificates

## test-dps-docs - Certificates

Device Provisioning Service

Search (Ctrl+ /)

Add Columns Refresh

Overview

Activity log

Access control (IAM)

Tags

SETTINGS

Properties

Locks

Automation script

Quick Start

Shared access policies

Linked IoT hubs

**Certificates**

Manage enrollments

**Certificate Details**

MyRootCert

**Delete**

Certificate Name ⓘ  
MyRootCert

ETag ⓘ  
ABCDEFGHJK=

Subject ⓘ  
O=MSR\_TEST, C=US, CN=riot-device-test

Expiry ⓘ  
Wed Dec 31 2036 19:00:00 GMT-0500 (Eastern Standard Time)

Thumbprint ⓘ  
ABCDEFABCDEFABCDEFABCDEFABC...

Created ⓘ  
Mon Aug 06 2018 04:48:06 GMT-0400 (Eastern Daylight Time)

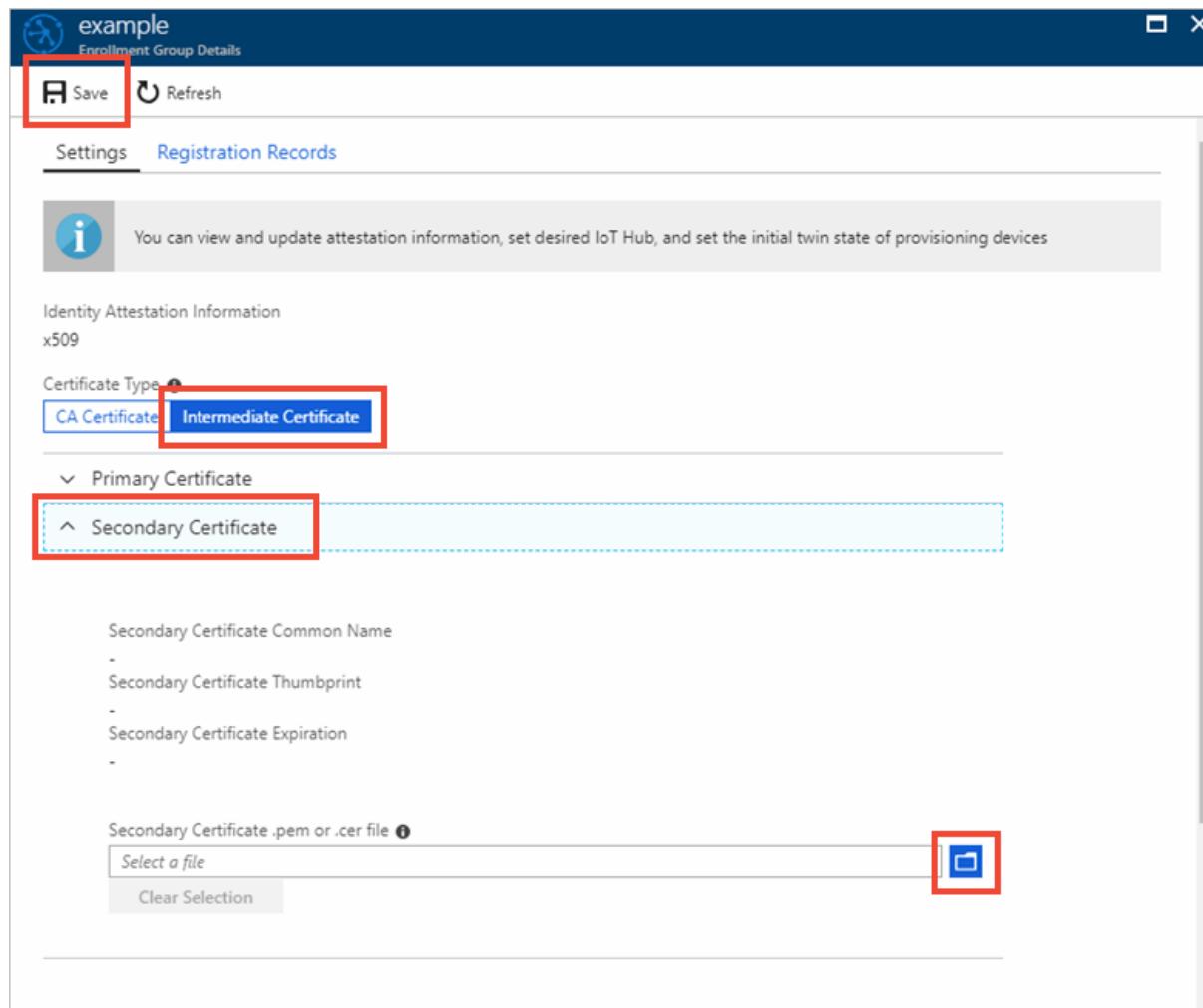
Updated ⓘ  
Mon Aug 06 2018 04:48:06 GMT-0400 (Eastern Daylight Time)

Verify

#### Update expiring intermediate certificates

1. Click **Enrollment Groups**, and click the group name in the list.
2. Click **Secondary Certificate** and then, click the folder icon to select the new certificate to be uploaded for the enrollment entry. Click **Save**.

This new intermediate certificate should be signed by a verified root CA certificate that has already been added into provisioning service. For more information, see [X.509 certificates](#).



3. Later when the primary certificate has expired, come back and delete that primary certificate by clicking the **Delete current certificate** button.

## Reprovision the device

Once the certificate is rolled on both the device and the Device Provisioning Service, the device can reprovision itself by contacting the Device Provisioning service.

One easy way of programming devices to reprovision is to program the device to contact the provisioning service to go through the provisioning flow if the device receives an "unauthorized" error from attempting to connect to the IoT hub.

Another way is for both the old and the new certificates to be valid for a short overlap, and use the IoT hub to send a command to devices to have them re-register via the provisioning service to update their IoT Hub connection information. Because each device can process commands differently, you will have to program your device to know what to do when the command is invoked. There are several ways you can command your device via IoT Hub, and we recommend using [direct methods or jobs](#) to initiate the process.

Once reprovisioning is complete, devices will be able to connect to IoT Hub using their new certificates.

## Blacklist certificates

In response to a security breach, you may need to blacklist a device certificate. To blacklist a device certificate, disable the enrollment entry for the target device/certificate. For more information, see blacklisting devices in the [Manage disenrollment](#) article.

Once a certificate is included as part of a disabled enrollment entry, any attempts to register with an IoT hub using that certificates will fail even if it is enabled as part of another enrollment entry.

## Next steps

- To learn more about X.509 certificates in the Device Provisioning Service, see [Security](#)
- To learn about how to do proof-of-possession for X.509 CA certificates with the Azure IoT Hub Device Provisioning Service, see [How to verify certificates](#)
- To learn about how to use the portal to create an enrollment group, see [Managing device enrollments with Azure portal](#).

# How to reprovision devices

12/10/2019 • 4 minutes to read • [Edit Online](#)

During the lifecycle of an IoT solution, it is common to move devices between IoT hubs. The reasons for this move may include the following scenarios:

- **Geolocation:** As a device moves between locations, network latency is improved by having the device migrated to an IoT hub closer to each location.
- **Multi-tenancy:** A device could be used within the same IoT solution but, reassigned or leased to a new customer, or customer site. This new customer may be serviced using a different IoT hub.
- **Solution change:** A device could be moved into a new or updated IoT solution. This reassignment may require that the device communicate with a new IoT hub that is connected to other backend components.
- **Quarantine:** Similar to a solution change. A device that is malfunctioning, compromised, or out-of-date may be reassigned to an IoT hub where all it can do is update and get back in compliance. Once the device is functioning properly, it is then migrated back to its main hub.

For more a more detailed overview of reprovisioning, see [IoT Hub Device reprovisioning concepts](#).

## Configure the enrollment allocation policy

The allocation policy determines how the devices associated with the enrollment will be allocated, or assigned, to an IoT hub once reprovisioned.

The following steps configure the allocation policy for a device's enrollment:

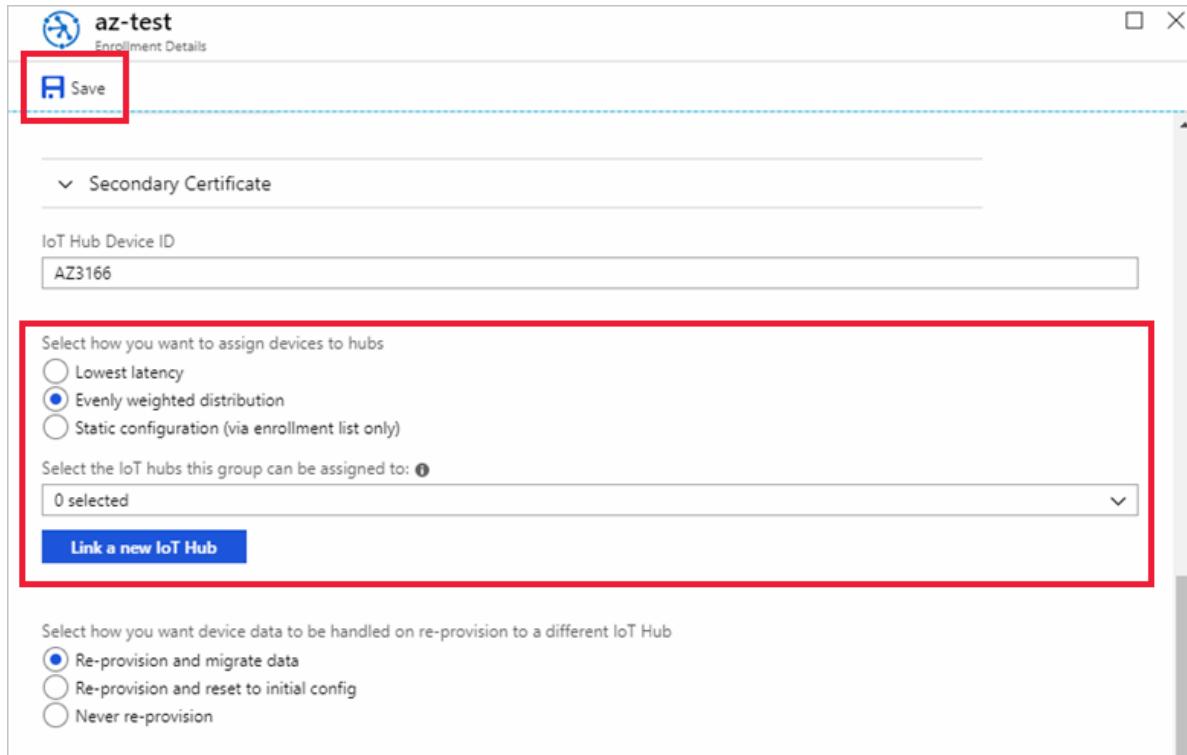
1. Sign in to the [Azure portal](#) and navigate to your Device Provisioning Service instance.
2. Click **Manage enrollments**, and click the enrollment group or individual enrollment that you want to configure for reprovisioning.
3. Under **Select how you want to assign devices to hubs**, select one of the following allocation policies:
  - **Lowest latency:** This policy assigns devices to the linked IoT Hub that will result in the lowest latency communications between device and IoT Hub. This option enables the device to communicate with the closest IoT hub based on location.
  - **Evenly weighted distribution:** This policy distributes devices across the linked IoT Hubs based on the allocation weight assigned to each linked IoT hub. This policy allows you to load balance devices across a group of linked hubs based on the allocation weights set on those hubs. If you are provisioning devices to only one IoT Hub, we recommend this setting. This setting is the default.
  - **Static configuration:** This policy requires a desired IoT Hub be listed in the enrollment entry for a device to be provisioned. This policy allows you to designate a single specific IoT hub that you want to assign devices to.
4. Under **Select the IoT hubs this group can be assigned to**, select the linked IoT hubs that you want included with your allocation policy. Optionally, add a new linked IoT hub using the **Link a new IoT Hub** button.

With the **Lowest latency** allocation policy, the hubs you select will be included in the latency evaluation to determine the closest hub for device assignment.

With the **Evenly weighted distribution** allocation policy, devices will be load balanced across the hubs you select based on their configured allocation weights and their current device load.

With the **Static configuration** allocation policy, select the IoT hub you want devices assigned to.

5. Click **Save**, or proceed to the next section to set the reprovisioning policy.



## Set the reprovisioning policy

1. Sign in to the [Azure portal](#) and navigate to your Device Provisioning Service instance.
2. Click **Manage enrollments**, and click the enrollment group or individual enrollment that you want to configure for reprovisioning.
3. Under **Select how you want device data to be handled on re-provision to a different IoT hub**, choose one of the following reprovisioning policies:
  - **Re-provision and migrate data:** This policy takes action when devices associated with the enrollment entry submit a new provisioning request. Depending on the enrollment entry configuration, the device may be reassigned to another IoT hub. If the device is changing IoT hubs, the device registration with the initial IoT hub will be removed. All device state information from that initial IoT hub will be migrated over to the new IoT hub. During migration, the device's status will be reported as **Assigning**.
  - **Re-provision and reset to initial config:** This policy takes action when devices associated with the enrollment entry submit a new provisioning request. Depending on the enrollment entry configuration, the device may be reassigned to another IoT hub. If the device is changing IoT hubs, the device registration with the initial IoT hub will be removed. The initial configuration data that the provisioning service instance received when the device was provisioned is provided to the new IoT hub. During migration, the device's status will be reported as **Assigning**.
4. Click **Save** to enable the reprovisioning of the device based on your changes.

The screenshot shows the 'Enrollment Details' page for an enrollment group named 'az-test'. At the top left is a blue circular icon with a white 'A' and a gear. To its right is the text 'az-test' and 'Enrollment Details'. In the top right corner are close and minimize buttons. A red box highlights the 'Save' button at the top left. Below it is a section titled 'Secondary Certificate' with a dropdown arrow. Underneath is a field for 'IoT Hub Device ID' containing 'AZ3166'. The next section, 'Select how you want to assign devices to hubs', contains three radio buttons: 'Lowest latency' (selected), 'Evenly weighted distribution', and 'Static configuration (via enrollment list only)'. Below this is a section titled 'Select the IoT hubs this group can be assigned to:' with a dropdown menu showing '2 selected'. A blue button labeled 'Link a new IoT Hub' is below this. The final section, highlighted by a red box, is 'Select how you want device data to be handled on re-provision to a different IoT Hub' with three radio buttons: 'Re-provision and migrate data' (selected), 'Re-provision and reset to initial config', and 'Never re-provision'.

## Send a provisioning request from the device

In order for devices to be reprovisioned based on the configuration changes made in the preceding sections, these devices must request reprovisioning.

How often a device submits a provisioning request depends on the scenario. However, it is advised to program your devices to send a provisioning request to a provisioning service instance on reboot, and support a [method](#) to manually trigger provisioning on demand. Provisioning could also be triggered by setting a [desired property](#).

The reprovisioning policy on an enrollment entry determines how the device provisioning service instance handles these provisioning requests, and if device state data should be migrated during reprovisioning. The same policies are available for individual enrollments and enrollment groups:

For example code of sending provisioning requests from a device during a boot sequence, see [Auto-provisioning a simulated device](#).

## Next steps

- To learn more Reprovisioning, see [IoT Hub Device reprovisioning concepts](#)
- To learn more Deprovisioning, see [How to deprovision devices that were previously auto-provisioned](#)

# How to disenroll a device from Azure IoT Hub Device Provisioning Service

12/10/2019 • 5 minutes to read • [Edit Online](#)

Proper management of device credentials is crucial for high-profile systems like IoT solutions. A best practice for such systems is to have a clear plan of how to revoke access for devices when their credentials, whether a shared access signatures (SAS) token or an X.509 certificate, might be compromised.

Enrollment in the Device Provisioning Service enables a device to be [auto-provisioned](#). A provisioned device is one that has been registered with IoT Hub, allowing it to receive its initial [device twin](#) state and begin reporting telemetry data. This article describes how to disenroll a device from your provisioning service instance, preventing it from being provisioned again in the future.

## NOTE

Be aware of the retry policy of devices that you revoke access for. For example, a device that has an infinite retry policy might continuously try to register with the provisioning service. That situation consumes service resources and possibly affects performance.

## Blacklist devices by using an individual enrollment entry

Individual enrollments apply to a single device and can use either X.509 certificates or SAS tokens (in a real or virtual TPM) as the attestation mechanism. (Devices that use SAS tokens as their attestation mechanism can be provisioned only through an individual enrollment.) To blacklist a device that has an individual enrollment, you can either disable or delete its enrollment entry.

To temporarily blacklist the device by disabling its enrollment entry:

1. Sign in to the Azure portal and select **All resources** from the left menu.
2. In the list of resources, select the provisioning service that you want to blacklist your device from.
3. In your provisioning service, select **Manage enrollments**, and then select the **Individual Enrollments** tab.
4. Select the enrollment entry for the device that you want to blacklist.

The screenshot shows the Microsoft Azure portal with the URL <https://portal.azure.com/>. The page title is "Enrollments - Microsoft". The left sidebar shows various service icons and the "Manage enrollments" option is highlighted with a red box. The main content area displays "test-dps-docs - Manage enrollments" with tabs for "Enrollment Groups" and "Individual Enrollments" (which is selected and highlighted with a red box). Below these tabs is a search bar labeled "Filter enrollments". The main table has columns for "REGISTRATION ID" and "Status". One row in the table contains the value "riot-device-cert" in the "REGISTRATION ID" column, which is also highlighted with a red box.

5. On your enrollment page, scroll to the bottom, and select **Disable** for the **Enable** entry switch, and then select **Save**.

The screenshot shows the "riot-device-cert" enrollment details page. The "Save" button is highlighted with a red box. Below it, there are sections for assigning devices to hubs and selecting IoT hubs. Further down, there is a section for handling device data on re-provisioning, which is currently empty. At the very bottom, there is an "Initial Device Twin State" JSON editor containing a basic template. The "Enable entry" switch is set to "Disable" and is also highlighted with a red box.

To permanently blacklist the device by deleting its enrollment entry:

1. Sign in to the Azure portal and select **All resources** from the left menu.
2. In the list of resources, select the provisioning service that you want to blacklist your device from.
3. In your provisioning service, select **Manage enrollments**, and then select the **Individual Enrollments** tab.
4. Select the check box next to the enrollment entry for the device that you want to blacklist.
5. Select **Delete** at the top of the window, and then select **Yes** to confirm that you want to remove the enrollment.

The screenshot shows the 'test-dps-docs - Manage enrollments' page. On the left, a sidebar lists various service management options. The 'Manage enrollments' option is highlighted with a red box. The main area displays a list of individual enrollments. At the top right, there are buttons for 'Add enrollment group', 'Add individual enrollment', 'Refresh', and 'Delete', with the 'Delete' button also highlighted by a red box. Below these buttons, a message states: 'You can add or remove individual device enrollments and/or enrollment groups'. Underneath, there are two tabs: 'Enrollment Groups' and 'Individual Enrollments', with 'Individual Enrollments' being the active tab and also highlighted by a red box. A search bar labeled 'Filter enrollments' is present. The 'REGISTRATION ID' section contains a list of entries, with one entry, 'riot-device-cert', having a checked checkbox, which is also highlighted by a red box.

After you finish the procedure, you should see your entry removed from the list of individual enrollments.

## Blacklist an X.509 intermediate or root CA certificate by using an enrollment group

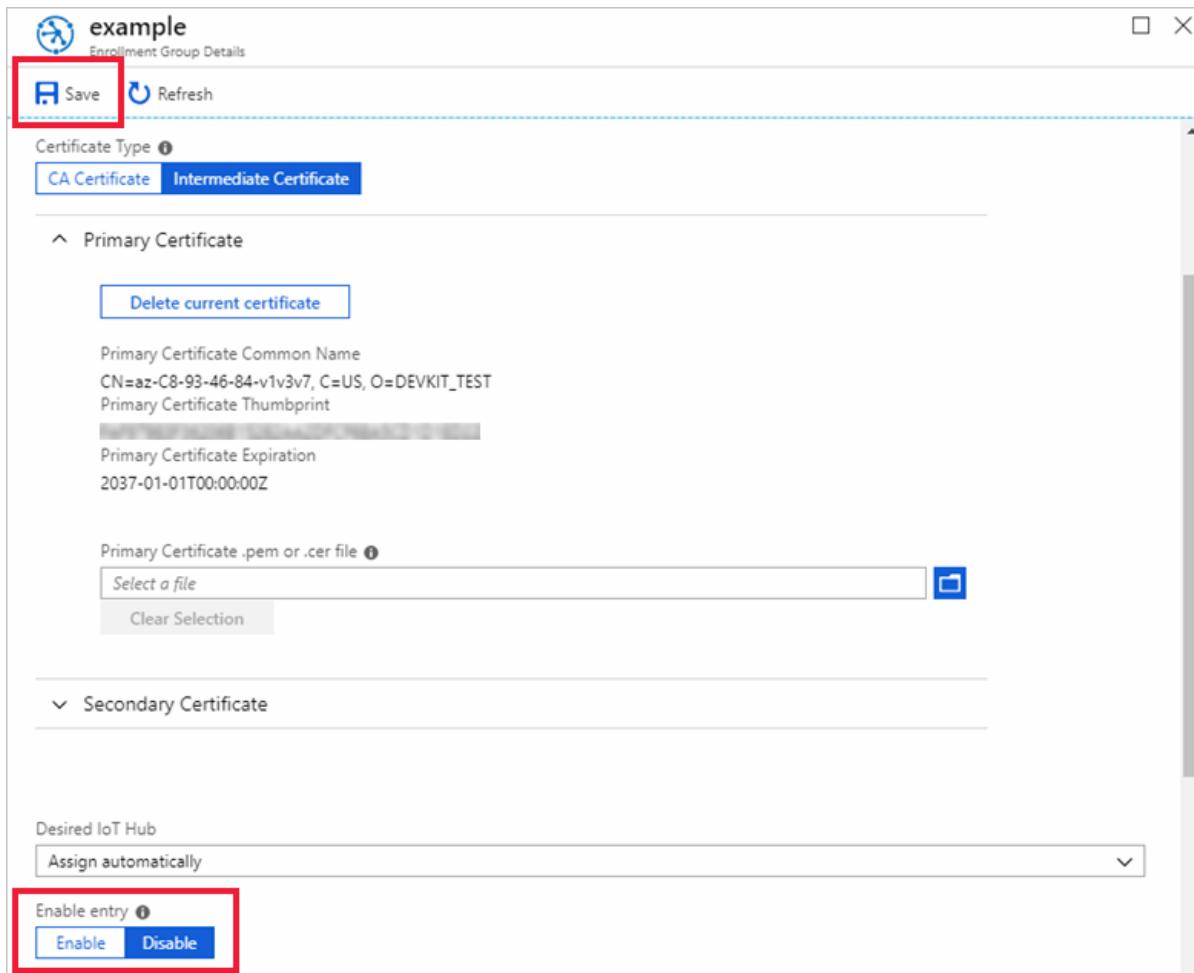
X.509 certificates are typically arranged in a certificate chain of trust. If a certificate at any stage in a chain becomes compromised, trust is broken. The certificate must be blacklisted to prevent Device Provisioning Service from provisioning devices downstream in any chain that contains that certificate. To learn more about X.509 certificates and how they are used with the provisioning service, see [X.509 certificates](#).

An enrollment group is an entry for devices that share a common attestation mechanism of X.509 certificates signed by the same intermediate or root CA. The enrollment group entry is configured with the X.509 certificate associated with the intermediate or root CA. The entry is also configured with any configuration values, such as twin state and IoT hub connection, that are shared by devices with that certificate in their certificate chain. To blacklist the certificate, you can either disable or delete its enrollment group.

To temporarily blacklist the certificate by disabling its enrollment group:

1. Sign in to the Azure portal and select **All resources** from the left menu.

2. In the list of resources, select the provisioning service that you want to blacklist the signing certificate from.
3. In your provisioning service, select **Manage enrollments**, and then select the **Enrollment Groups** tab.
4. Select the enrollment group using the certificate that you want to blacklist.
5. Select **Disable** on the **Enable entry** switch, and then select **Save**.



To permanently blacklist the certificate by deleting its enrollment group:

1. Sign in to the Azure portal and select **All resources** from the left menu.
2. In the list of resources, select the provisioning service that you want to blacklist your device from.
3. In your provisioning service, select **Manage enrollments**, and then select the **Enrollment Groups** tab.
4. Select the check box next to the enrollment group for the certificate that you want to blacklist.
5. Select **Delete** at the top of the window, and then select **Yes** to confirm that you want to remove the enrollment group.

The screenshot shows the 'test-dps-docs - Manage enrollments' page. On the left, there's a sidebar with various settings like Overview, Activity log, Access control (IAM), Tags, Properties, Locks, Automation script, Quick Start, Shared access policies, Linked IoT hubs, Certificates, and Manage enrollments (which is highlighted with a red box). The main content area has a search bar and buttons for Add enrollment group, Add individual enrollment, Refresh, and Delete (also highlighted with a red box). It includes a note about adding or removing device enrollments and groups. Below that, there are tabs for Enrollment Groups and Individual Enrollments, with 'Enrollment Groups' selected. A 'Filter enrollments' search bar is present. Under 'GROUP NAME', there's a list with 'example' checked (highlighted with a red box). Other items in the list include 'example2' and 'example3'.

After you finish the procedure, you should see your entry removed from the list of enrollment groups.

#### NOTE

If you delete an enrollment group for a certificate, devices that have the certificate in their certificate chain might still be able to enroll if an enabled enrollment group for the root certificate or another intermediate certificate higher up in their certificate chain exists.

## Blacklist specific devices in an enrollment group

Devices that implement the X.509 attestation mechanism use the device's certificate chain and private key to authenticate. When a device connects and authenticates with Device Provisioning Service, the service first looks for an individual enrollment that matches the device's credentials. The service then searches enrollment groups to determine whether the device can be provisioned. If the service finds a disabled individual enrollment for the device, it prevents the device from connecting. The service prevents the connection even if an enabled enrollment group for an intermediate or root CA in the device's certificate chain exists.

To blacklist an individual device in an enrollment group, follow these steps:

1. Sign in to the Azure portal and select **All resources** from the left menu.
2. From the list of resources, select the provisioning service that contains the enrollment group for the device that you want to blacklist.
3. In your provisioning service, select **Manage enrollments**, and then select the **Individual Enrollments** tab.
4. Select the **Add individual enrollment** button at the top.
5. On the **Add Enrollment** page, select **X.509** as the **Attestation Mechanism** for the device.

Upload the device certificate, and enter the device ID of the device to be blacklisted. For the certificate, use the signed end-entity certificate installed on the device. The device uses the signed end-entity certificate for

authentication.

Add Enrollment

Save

\* Mechanism X.509

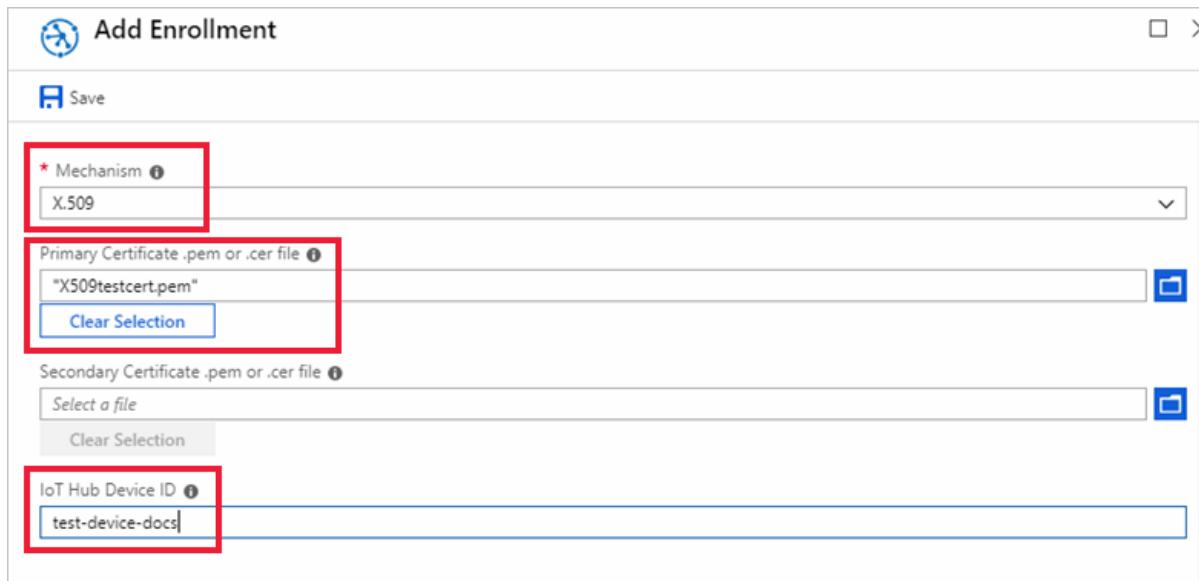
Primary Certificate .pem or .cer file X509testcert.pem

Secondary Certificate .pem or .cer file Select a file

Clear Selection

IoT Hub Device ID test-device-docs

Clear Selection



6. Scroll to the bottom of the Add Enrollment page and select Disable on the Enable entry switch, and then select Save.

Add Enrollment

Save

Select how you want to assign devices to hubs (preview for enrollment)

Lowest latency

Evenly weighted distribution

Static configuration (via enrollment list only)

Select the IoT hubs this group can be assigned to: (preview) 2 selected

Link a new IoT hub

\* Select how you want device data to be handled on re-provisioning (preview)

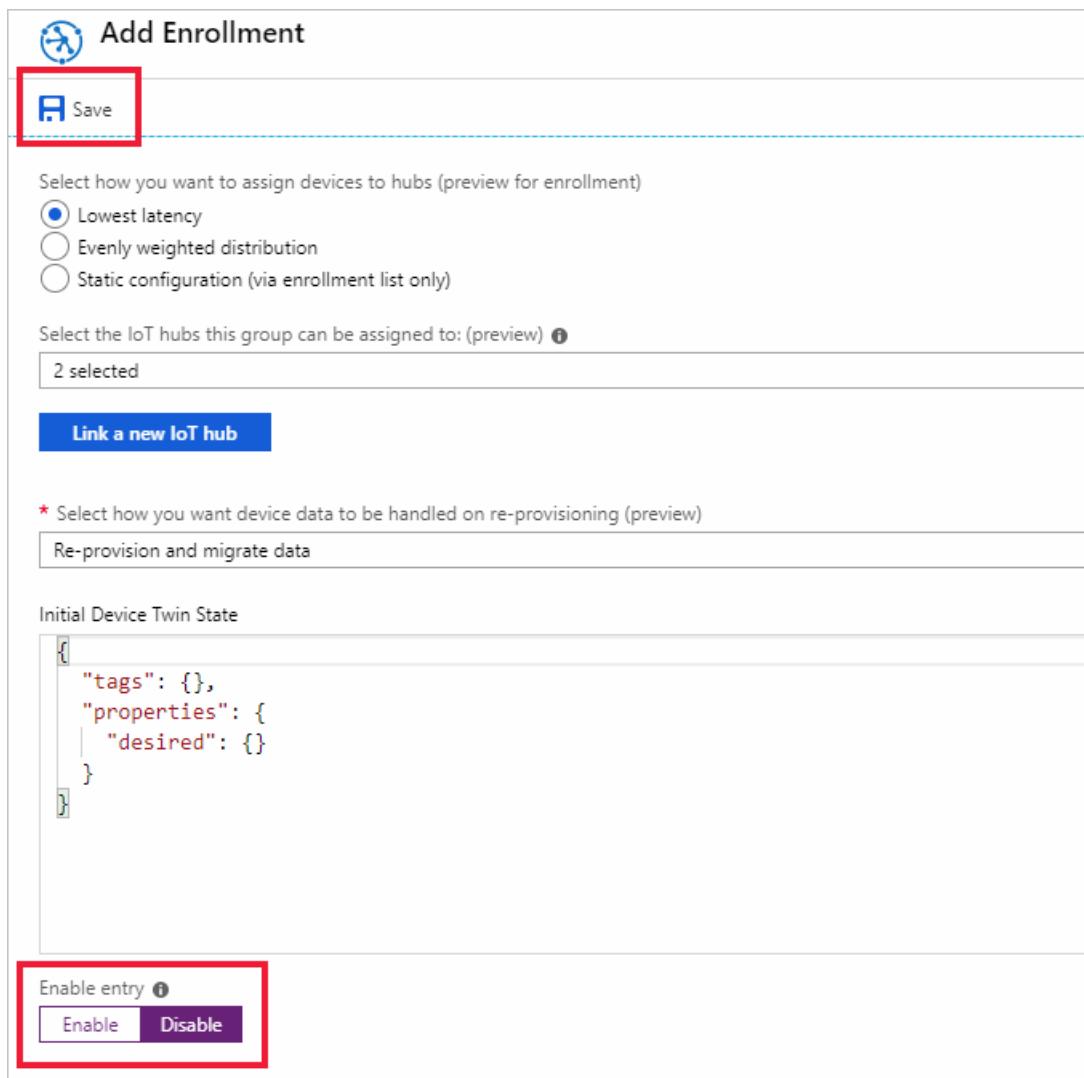
Re-provision and migrate data

Initial Device Twin State

```
{  
  "tags": {},  
  "properties": {  
    "desired": {}  
  }  
}
```

Enable entry

Enable Disable



When you successfully create your enrollment, you should see your disabled device enrollment listed on the Individual Enrollments tab.

## Next steps

Disenrollment is also part of the larger deprovisioning process. Deprovisioning a device includes both disenrollment from the provisioning service, and deregistering from IoT hub. To learn about the full process, see [How to deprovision devices that were previously auto-provisioned](#)

# How to deprovision devices that were previously auto-provisioned

12/10/2019 • 3 minutes to read • [Edit Online](#)

You may find it necessary to deprovision devices that were previously auto-provisioned through the Device Provisioning Service. For example, a device may be sold or moved to a different IoT hub, or it may be lost, stolen, or otherwise compromised.

In general, deprovisioning a device involves two steps:

1. Disenroll the device from your provisioning service, to prevent future auto-provisioning. Depending on whether you want to revoke access temporarily or permanently, you may want to either disable or delete an enrollment entry. For devices that use X.509 attestation, you may want to disable/delete an entry in the hierarchy of your existing enrollment groups.
  - To learn how to disenroll a device, see [How to disenroll a device from Azure IoT Hub Device Provisioning Service](#).
  - To learn how to disenroll a device programmatically using one of the provisioning service SDKs, see [Manage device enrollments with service SDKs](#).
2. Deregister the device from your IoT Hub, to prevent future communications and data transfer. Again, you can temporarily disable or permanently delete the device's entry in the identity registry for the IoT Hub where it was provisioned. See [Disable devices](#) to learn more about disablement. See "Device Management / IoT Devices" for your IoT Hub resource, in the [Azure portal](#).

The exact steps you take to deprovision a device depend on its attestation mechanism and its applicable enrollment entry with your provisioning service. The following sections provide an overview of the process, based on the enrollment and attestation type.

## Individual enrollments

Devices that use TPM attestation or X.509 attestation with a leaf certificate are provisioned through an individual enrollment entry.

To deprovision a device that has an individual enrollment:

1. Disenroll the device from your provisioning service:
  - For devices that use TPM attestation, delete the individual enrollment entry to permanently revoke the device's access to the provisioning service, or disable the entry to temporarily revoke its access.
  - For devices that use X.509 attestation, you can either delete or disable the entry. Be aware, though, if you delete an individual enrollment for a device that uses X.509 and an enabled enrollment group exists for a signing certificate in that device's certificate chain, the device can re-enroll. For such devices, it may be safer to disable the enrollment entry. Doing so prevents the device from re-enrolling, regardless of whether an enabled enrollment group exists for one of its signing certificates.
2. Disable or delete the device in the identity registry of the IoT hub that it was provisioned to.

## Enrollment groups

With X.509 attestation, devices can also be provisioned through an enrollment group. Enrollment groups are configured with a signing certificate, either an intermediate or root CA certificate, and control access to the

provisioning service for devices with that certificate in their certificate chain. To learn more about enrollment groups and X.509 certificates with the provisioning service, see [X.509 certificates](#).

To see a list of devices that have been provisioned through an enrollment group, you can view the enrollment group's details. This is an easy way to understand which IoT hub each device has been provisioned to. To view the device list:

1. Log in to the Azure portal and click **All resources** on the left-hand menu.
2. Click your provisioning service in the list of resources.
3. In your provisioning service, click **Manage enrollments**, then select **Enrollment Groups** tab.
4. Click the enrollment group to open it.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar includes the URL <https://ms.portal.azure.com/#resource/subscriptions/>. The main content area is titled "Microsoft Azure" and shows a sub-navigation path: Home > sample-provisioning-service - Manage enrollments > testgroup. On the left, there is a sidebar with various icons. The main pane is titled "testgroup" and contains sections for "Identity attestation information" (X509, Certificate name (or thumbprint), RootCA) and "Desired IoT hub" (Assign automatically, Provisioning status: Enabled). Below this is a JSON configuration block:

```
"tags": {},  
"desiredproperties": {}
```

At the bottom, there is a table titled "Search devices in this enrollment group" with columns: DEVICE ID, ASSIGNED IOT HUB, and REGISTRATION DATE. The table lists seven devices, all assigned to "iotsdk-node-win-gate.azure-devices.net" and registered between December 2017 and January 2018. The entire table is highlighted with a red border.

DEVICE ID	ASSIGNED IOT HUB	REGISTRATION DATE
mg-5340485c-2f9e-4395-8929-0403474c0002	iotsdk-node-win-gate.azure-devices.net	11/30/2017 11:06:18 PM
mg-173ca2d5-4a67-4add-8bc4-0f4e0a4f55ad	iotsdk-node-win-gate.azure-devices.net	12/5/2017 6:11:38 AM
mg-af92a2a2-ef22-4d64-a885-094a674c0001	iotsdk-node-win-gate.azure-devices.net	12/5/2017 11:21:27 PM
mg-340a9f9e-af5a-4480-8c47-e4a5f779e10e	iotsdk-node-win-gate.azure-devices.net	12/6/2017 3:48:57 AM
mg-291a2089-2259-4829-9987-de13cb3f6e01	iotsdk-node-win-gate.azure-devices.net	12/7/2017 11:44:59 PM
mg-410a44ec-0544-4891-aae7-2100e4a43c38	iotsdk-node-win-gate.azure-devices.net	12/12/2017 6:34:45 PM
mg-4734a6a2-0253-4f5c-94fc-00f0eab71705	iotsdk-node-win-gate.azure-devices.net	12/19/2017 9:39:51 PM

With enrollment groups, there are two scenarios to consider:

- To deprovision all of the devices that have been provisioned through an enrollment group:
  1. Disable the enrollment group to blacklist its signing certificate.
  2. Use the list of provisioned devices for that enrollment group to disable or delete each device from the identity registry of its respective IoT hub.
  3. After disabling or deleting all devices from their respective IoT hubs, you can optionally delete the enrollment group. Be aware, though, that, if you delete the enrollment group and there is an enabled enrollment group for a signing certificate higher up in the certificate chain of one or more of the devices, those devices can re-enroll.
- To deprovision a single device from an enrollment group:
  1. Create a disabled individual enrollment for its leaf (device) certificate. This revokes access to the provisioning service for that device while still permitting access for other devices that have the

enrollment group's signing certificate in their chain. Do not delete the disabled individual enrollment for the device. Doing so will allow the device to re-enroll through the enrollment group.

2. Use the list of provisioned devices for that enrollment group to find the IoT hub that the device was provisioned to and disable or delete it from that hub's identity registry.

# How to use Azure CLI and the IoT extension to manage the IoT Hub Device Provisioning Service

7/30/2020 • 2 minutes to read • [Edit Online](#)

Azure CLI is an open-source cross platform command-line tool for managing Azure resources such as IoT Edge. Azure CLI is available on Windows, Linux, and MacOS. Azure CLI enables you to manage Azure IoT Hub resources, Device Provisioning service instances, and linked-hubs out of the box.

The IoT extension enriches Azure CLI with features such as device management and full IoT Edge capability.

In this tutorial, you first complete the steps to setup Azure CLI and the IoT extension. Then you learn how to run CLI commands to perform basic Device Provisioning Service operations.

## NOTE

This article uses the newest version of the Azure IoT extension, called `azure-iot`. The legacy version is called `azure-cli-iot-ext`. You should only have one version installed at a time. You can use the command `az extension list` to validate the currently installed extensions.

Use `az extension remove --name azure-cli-iot-ext` to remove the legacy version of the extension.

Use `az extension add --name azure-iot` to add the new version of the extension.

To see what extensions you have installed, use `az extension list`.

## Installation

### Install Python

[Python 2.7x or Python 3.x](#) is required.

### Install the Azure CLI

Follow the [installation instruction](#) to setup Azure CLI in your environment. At a minimum, your Azure CLI version must be 2.0.70 or above. Use `az --version` to validate. This version supports az extension commands and introduces the Knack command framework. One simple way to install on Windows is to download and install the [MSI](#).

### Install IoT extension

The [IoT extension readme](#) describes several ways to install the extension. The simplest way is to run `az extension add --name azure-iot`. After installation, you can use `az extension list` to validate the currently installed extensions or `az extension show --name azure-iot` to see details about the IoT extension. To remove the extension, you can use `az extension remove --name azure-iot`.

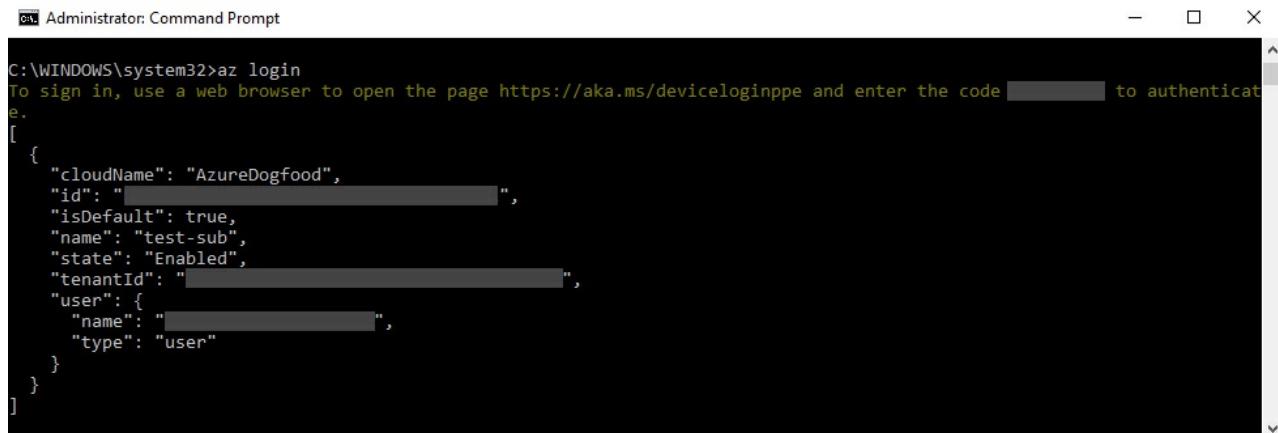
## Basic Device Provisioning Service operations

The example shows you how to log in to your Azure account, create an Azure Resource Group (a container that holds related resources for an Azure solution), create an IoT Hub, create a Device Provisioning service, list the existing Device Provisioning services and create a linked IoT hub with CLI commands.

Complete the installation steps described previously before you begin. If you don't have an Azure account yet, you can [create a free account](#) today.

## 1. Log in to the Azure account

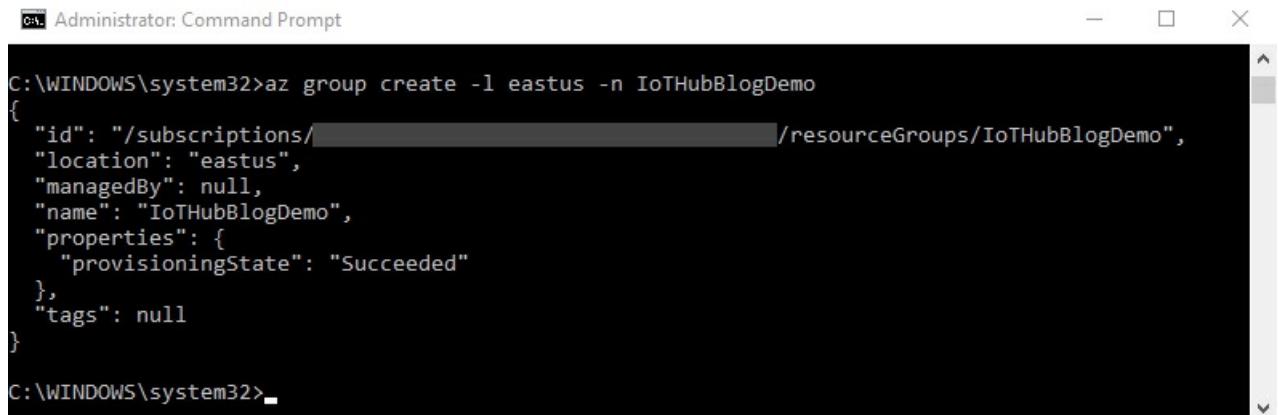
```
az login
```



```
C:\WINDOWS\system32>az login
To sign in, use a web browser to open the page https://aka.ms/deviceloginppe and enter the code [REDACTED] to authenticate.
[{"cloudName": "AzureDogfood", "id": "[REDACTED]", "isDefault": true, "name": "test-sub", "state": "Enabled", "tenantId": "[REDACTED]", "user": {"name": "[REDACTED]", "type": "user"}}]
```

## 2. Create a resource group IoTHubBlogDemo in eastus

```
az group create -l eastus -n IoTHubBlogDemo
```



```
C:\WINDOWS\system32>az group create -l eastus -n IoTHubBlogDemo
{
  "id": "/subscriptions/[REDACTED]/resourceGroups/IoTHubBlogDemo",
  "location": "eastus",
  "managedBy": null,
  "name": "IoTHubBlogDemo",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}

C:\WINDOWS\system32>
```

## 3. Create two Device Provisioning services

```
az iot dps create --resource-group IoTHubBlogDemo --name demodps
```



```
C:\WINDOWS\system32>az iot dps create --resource-group IoTHubBlogDemo --name demodps
[- Finished ..
{
  "additionalProperties": {
    "resourcegroup": "IoTHubBlogDemo",
    "subscriptionid": "[REDACTED]"
  },
  "etag": "AAAAAAAActr8=",
  "id": "/subscriptions/[REDACTED]/resourceGroups/IoTHubBlogDemo/providers/Microsoft.Devices/provisioningServices/demodps",
  "location": "eastus",
  "name": "demodps",
  "properties": {
    "additionalProperties": {},
    "allocationPolicy": "Hashed",
    "authorizationPolicies": null,
    "deviceProvisioningHostName": "global.df.azure-devices-provisioning-int.net",
    "idScope": "0ne0000234E",
    "iotHubs": [],
    "provisioningState": null,
    "serviceOperationsHostName": "demodps.df.azure-devices-provisioning-int.net",
    "state": "Active"
  },
  "resourceGroup": "IoTHubBlogDemo",
  "sku": {
    "additionalProperties": {},
    "capacity": 1,
    "name": "S1",
    "tier": "Standard"
  }
}
```

```
az iot dps create --resource-group IoTHubBlogDemo --name demodps2
```

**4. List all the existing Device Provisioning services under this resource group**

```
az iot dps list --resource-group IoTHubBlogDemo
```

```
C:\Windows\system32>az iot dps list --resource-group IoTHubBlogDemo
[
{
  "additionalProperties": {
    "resourcegroup": "IoTHubBlogDemo",
    "subscriptionid": "XXXXXXXXXXXXXX"
  },
  "etag": "AAAAAAAActr8=",
  "id": "/subscriptions/XXXXXXXXXXXXXX/resourceGroups/IoTHubBlogDemo/providers/Microsoft.Devices/provisioningServices/demodps",
  "location": "eastus",
  "name": "demodps",
  "properties": {
    "additionalProperties": {},
    "allocationPolicy": "Hashed",
    "authorizationPolicies": null,
    "deviceProvisioningHostName": "global.df.azure-devices-provisioning-int.net",
    "idScope": "0ne0000234E",
    "iotHubs": [],
    "provisioningState": null,
    "serviceOperationsHostName": "demodps.df.azure-devices-provisioning-int.net",
    "state": "Active"
  },
  "resourceGroup": "IoTHubBlogDemo",
  "sku": {
    "additionalProperties": {},
    "capacity": 1,
    "name": "S1",
    "tier": "Standard"
  },
  "tags": {},
  "type": "Microsoft.Devices/provisioningServices"
},
{
  "additionalProperties": {
    "resourcegroup": "IoTHubBlogDemo",
    "subscriptionid": "XXXXXXXXXXXXXX"
  },
  "etag": "AAAAAAAActso=",
  "id": "/subscriptions/XXXXXXXXXXXXXX/resourceGroups/IoTHubBlogDemo/providers/Microsoft.Devices/provisioningServices/demodps2",
  "location": "eastus",
  "name": "demodps2",
  "properties": {
```

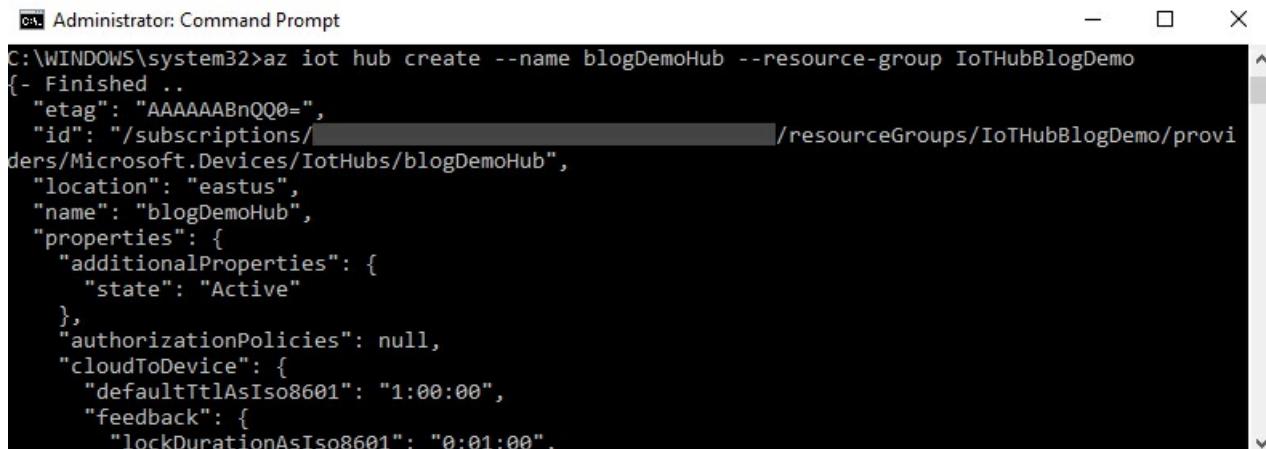
## 5. Create an IoT Hub blogDemoHub under the newly created resource group

```
az iot hub create --name blogDemoHub --resource-group IoTHubBlogDemo
```

```
C:\WINDOWS\system32>az iot hub create --name blogDemoHub --resource-group IoTHubBlogDemo
{- Finished ..
  "etag": "AAAAAAABnQQ0=",
  "id": "/subscriptions/[REDACTED]/resourceGroups/IoTHubBlogDemo/providers/Microsoft.Devices/IotHubs/blogDemoHub",
  "location": "eastus",
  "name": "blogDemoHub",
  "properties": {
    "additionalProperties": {
      "state": "Active"
    },
    "authorizationPolicies": null,
    "cloudToDevice": {
      "defaultTtlAsIso8601": "1:00:00",
      "feedback": {
        "lockDurationAsIso8601": "0:01:00",
        "maxDeliveryCount": 10,
        "maxLockDuration": "PT1H",
        "maxTtl": "PT1H"
      }
    },
    "deviceToCloud": {
      "maxTtl": "PT1H"
    }
  }
}
```

## 6. Link one existing IoT Hub to a Device Provisioning service

```
az iot dps linked-hub create --resource-group IoTHubBlogDemo --dps-name demodps --connection-string <connection string> -l westus
```



```
C:\WINDOWS\system32>az iot hub create --name blogDemoHub --resource-group IoTHubBlogDemo
{- Finished ..
  "etag": "AAAAAAABnQQ0=",
  "id": "/subscriptions/[REDACTED]/resourceGroups/IoTHubBlogDemo/providers/Microsoft.Devices/IotHubs/blogDemoHub",
  "location": "eastus",
  "name": "blogDemoHub",
  "properties": {
    "additionalProperties": {
      "state": "Active"
    },
    "authorizationPolicies": null,
    "cloudToDevice": {
      "defaultTtlAsIso8601": "1:00:00",
      "feedback": {
        "lockDurationAsIso8601": "0:01:00",
      }
    }
  }
}
```

## Next steps

In this tutorial, you learned how to:

- Enroll the device
- Start the device
- Verify the device is registered

Advance to the next tutorial to learn how to provision multiple devices across load-balanced hubs.

[Provision devices across load-balanced IoT hubs](#)

# Control access to Azure IoT Hub Device Provisioning Service

7/30/2020 • 5 minutes to read • [Edit Online](#)

This article describes the options for securing your IoT Device Provisioning service. The provisioning service uses *permissions* to grant access to each endpoint. Permissions limit the access to a service instance based on functionality.

This article describes:

- The different permissions that you can grant to a backend app to access your provisioning service.
- The authentication process and the tokens it uses to verify permissions.

## When to use

You must have appropriate permissions to access any of the provisioning service endpoints. For example, a backend app must include a token containing security credentials along with every message it sends to the service.

## Access control and permissions

You can grant [permissions](#) in the following ways:

- **Shared access authorization policies.** Shared access policies can grant any combination of [permissions](#). You can define policies in the [Azure portal](#), or programmatically by using the [Device Provisioning Service REST APIs](#). A newly created provisioning service has the following default policy:
  - **provisioningserviceowner:** Policy with all permissions.

### NOTE

See [permissions](#) for detailed information.

## Authentication

Azure IoT Hub Device Provisioning Service grants access to endpoints by verifying a token against the shared access policies. Security credentials, such as symmetric keys, are never sent over the wire.

### NOTE

The Device Provisioning Service resource provider is secured through your Azure subscription, as are all providers in the [Azure Resource Manager](#).

For more information about how to construct and use security tokens, see the next section.

HTTP is the only supported protocol, and it implements authentication by including a valid token in the **Authorization** request header.

### Example

```
SharedAccessSignature sr =  
    mydps.azure-devices-  
    provisioning.net&sig=kPszxZZZZZZZZZZZZZZAhLT%2bV7o%3d&se=1487709501&skn=provisioningserviceowner`\\
```

#### NOTE

The [Azure IoT Device Provisioning Service SDKs](#) automatically generate tokens when connecting to the service.

## Security tokens

The Device Provisioning Service uses security tokens to authenticate services to avoid sending keys on the wire. Additionally, security tokens are limited in time validity and scope. [Azure IoT Device Provisioning Service SDKs](#) automatically generate tokens without requiring any special configuration. Some scenarios do require you to generate and use security tokens directly. Such scenarios include the direct use of the HTTP surface.

### Security token structure

You use security tokens to grant time-bounded access for services to specific functionality in IoT Device Provisioning Service. To get authorization to connect to the provisioning service, services must send security tokens signed with either a shared access or symmetric key.

A token signed with a shared access key grants access to all the functionality associated with the shared access policy permissions.

The security token has the following format:

```
SharedAccessSignature sig={signature}&se={expiry}&skn={policyName}&sr={URL-encoded-resourceURI}
```

Here are the expected values:

VALUE	DESCRIPTION
{signature}	An HMAC-SHA256 signature string of the form: <code>{URL-encoded-resourceURI} + "\n" + expiry</code> . <b>Important:</b> The key is decoded from base64 and used as key to perform the HMAC-SHA256 computation.
{expiry}	UTF8 strings for number of seconds since the epoch 00:00:00 UTC on 1 January 1970.
{URL-encoded-resourceURI}	Lower case URL-encoding of the lower case resource URI. URI prefix (by segment) of the endpoints that can be accessed with this token, starting with host name of the IoT Device Provisioning Service (no protocol). For example, <code>mydps.azure-devices-provisioning.net</code> .
{policyName}	The name of the shared access policy to which this token refers.

**Note on prefix:** The URI prefix is computed by segment and not by character. For example `/a/b` is a prefix for `/a/b/c` but not for `/a/bc`.

The following Node.js snippet shows a function called `generateSasToken` that computes the token from the inputs `resourceUri, signingKey, policyName, expiresInMins`. The next sections detail how to initialize the different inputs for the different token use cases.

```

var generateSasToken = function(resourceUri, signingKey, policyName, expiresInMins) {
    resourceUri = encodeURIComponent(resourceUri);

    // Set expiration in seconds
    var expires = (Date.now() / 1000) + expiresInMins * 60;
    expires = Math.ceil(expires);
    var toSign = resourceUri + '\n' + expires;

    // Use crypto
    var hmac = crypto.createHmac('sha256', new Buffer(signingKey, 'base64'));
    hmac.update(toSign);
    var base64UriEncoded = encodeURIComponent(hmac.digest('base64'));

    // Construct authorization string
    var token = "SharedAccessSignature sr=" + resourceUri + "&sig=";
    + base64UriEncoded + "&se=" + expires + "&skn=" + policyName;
    return token;
};

}

```

As a comparison, the equivalent Python code to generate a security token is:

```

from base64 import b64encode, b64decode
from hashlib import sha256
from time import time
from urllib import quote_plus, urlencode
from hmac import HMAC

def generate_sas_token(uri, key, policy_name, expiry=3600):
    ttl = time() + expiry
    sign_key = "%s\n%d" % ((quote_plus(uri)), int(ttl))
    print sign_key
    signature = b64encode(HMAC(b64decode(key), sign_key, sha256).digest())

    rawtoken = {
        'sr' : uri,
        'sig': signature,
        'se' : str(int(ttl)),
        'skn' : policy_name
    }

    return 'SharedAccessSignature ' + urlencode(rawtoken)

```

#### **NOTE**

Since the time validity of the token is validated on IoT Device Provisioning Service machines, the drift on the clock of the machine that generates the token must be minimal.

### **Use security tokens from service components**

Service components can only generate security tokens using shared access policies granting the appropriate permissions as explained previously.

Here are the service functions exposed on the endpoints:

ENDPOINT	FUNCTIONALITY
{your-service}.azure-devices-provisioning.net/enrollments	Provides device enrollment operations with the Device Provisioning Service.

ENDPOINT	FUNCTIONALITY
{your-service}.azure-devices-provisioning.net/enrollmentGroups	Provides operations for managing device enrollment groups.
{your-service}.azure-devices-provisioning.net/registrations/{id}	Provides operations for retrieving and managing the status of device registrations.

As an example, a service generated using a pre-created shared access policy called **enrollmentread** would create a token with the following parameters:

- resource URI: {mydps}.azure-devices-provisioning.net ,
- signing key: one of the keys of the **enrollmentread** policy,
- policy name: **enrollmentread** ,
- any expiration time.backn

The screenshot shows the Azure Device Provisioning Service interface. On the left, under 'Shared access policies', the 'enrollmentread' policy is selected. In the center, the 'Add Access Policy' dialog box is open, showing the configuration for the 'enrollmentread' policy. The 'Name' field is set to 'enrollmentread'. Under 'Permissions', 'Enrollment read' and 'Registration status read' are checked. The 'Save' button at the bottom right of the dialog box is highlighted with a red box.

```
var endpoint ="mydps.azure-devices-provisioning.net";
var policyName = 'enrollmentread';
var policyKey = '...';

var token = generateSasToken(endpoint, policyKey, policyName, 60);
```

The result, which would grant access to read all enrollment records, would be:

```
SharedAccessSignature sr=mydps.azure-devices-provisioning.net&sig=JdyscqTpXdEJs49e1IUccohw2D1FDR3zfH5KqGJo4r4%3D&se=1456973447&skn=enrollmentread
```

## Reference topics:

The following reference topics provide you with more information about controlling access to your IoT Device Provisioning Service.

### Device Provisioning Service permissions

The following table lists the permissions you can use to control access to your IoT Device Provisioning Service.

PERMISSION	NOTES
ServiceConfig	Grants access to change the service configurations. This permission is used by backend cloud services.

PERMISSION	NOTES
<b>EnrollmentRead</b>	Grants read access to the device enrollments and enrollment groups. This permission is used by backend cloud services.
<b>EnrollmentWrite</b>	Grants write access to the device enrollments and enrollment groups. This permission is used by backend cloud services.
<b>RegistrationStatusRead</b>	Grants read access to the device registration status. This permission is used by backend cloud services.
<b>RegistrationStatusWrite</b>	Grants delete access to the device registration status. This permission is used by backend cloud services.

# Use Azure IoT DPS IP connection filters

7/30/2020 • 5 minutes to read • [Edit Online](#)

Security is an important aspect of any IoT solution. Sometimes you need to explicitly specify the IP addresses from which devices can connect as part of your security configuration. The *IP filter* feature for an Azure IoT Hub Device Provisioning Service (DPS) enables you to configure rules for rejecting or accepting traffic from specific IPv4 addresses.

## When to use

There are two specific use-cases where it is useful to block connections to a DPS endpoint from certain IP addresses:

- Your DPS should receive traffic only from a specified range of IP addresses and reject everything else. For example, you are using your DPS with [Azure Express Route](#) to create private connections between a DPS and your devices.
- You need to reject traffic from IP addresses that have been identified as suspicious by the DPS administrator.

## How filter rules are applied

The IP filter rules are applied at the DPS instance level. Therefore the IP filter rules apply to all connections from devices and back-end apps using any supported protocol.

Any connection attempt from an IP address that matches a rejecting IP rule in your DPS instance receives an unauthorized 401 status code and description. The response message does not mention the IP rule.

## Default setting

By default, the **IP Filter** grid in the portal for DPS is empty. This default setting means that your DPS accepts connections from any IP address. This default setting is equivalent to a rule that accepts the 0.0.0.0/0 IP address range.

The screenshot shows the 'ContosoTest - IP Filter' page in the Azure IoT DPS portal. The left sidebar has a red box around the 'IP Filter' item under the 'Settings' section. The main area has a heading 'IP filters block or allow specific IP address ranges. Your filters are applied in order.' and a blue button '+ Add IP Filter Rule'.

## Add or edit an IP filter rule

To add an IP filter rule, select + Add IP Filter Rule.

The screenshot shows the 'IP Filter' interface. At the top, there are three buttons: 'Save' (with a disk icon), 'Revert' (with a circular arrow icon), and 'Test' (with a checkmark icon). Below these buttons is a descriptive text: 'IP filters block or allow specific IP address ranges. Your filters are applied in order.' At the bottom left, there is a blue button labeled '+ Add IP Filter Rule' which is highlighted with a red rectangular border.

After selecting Add IP Filter Rule, fill in the fields.

The screenshot shows the 'IP Filter' interface after adding a rule. The 'Save' button is highlighted with a red rectangular border. The table below shows one rule:

NAME	ADDRESS RANGE	ACTION
MaliciousIP	6.6.6/6	<input checked="" type="radio"/> Block <span style="color: #ccc;">Delete</span>

At the bottom left, there is a blue button labeled '+ Add IP Filter Rule'.

- Provide a **name** for the IP Filter rule. This must be a unique, case-insensitive, alphanumeric string up to 128 characters long. Only the ASCII 7-bit alphanumeric characters plus `{'-', ':', '/', '\', '.', '+', '%', '_', '#', '*', '?', '!', '(', ')', ',', '=', '@', ';', '''}` are accepted.
- Provide a single IPv4 address or a block of IP addresses in CIDR notation. For example, in CIDR notation 192.168.100.0/22 represents the 1024 IPv4 addresses from 192.168.100.0 to 192.168.103.255.
- Select **Allow** or **Block** as the **action** for the IP filter rule.

After filling in the fields, select **Save** to save the rule. You see an alert notifying you that the update is in progress.

**IP Filter**

Save Revert Test

Updating DPS

IP filters block or allow specific IP address ranges. Your filters are applied in order. Drag and drop a filter to change its priority in the list.

NAME	ADDRESS RANGE	ACTION
MaliciousIP	6.6.6.6/6	Block

+ Add IP Filter Rule

The **Add** option is disabled when you reach the maximum of 10 IP filter rules.

To edit an existing rule, select the data you want to change, make the change, then select **Save** to save your edit.

**NOTE**

Rejecting IP addresses can prevent other Azure Services from interacting with the DPS instance.

## Delete an IP filter rule

To delete an IP filter rule, select the trash can icon on that row and then select **Save**. The rule is removed and the change is saved.

**IP Filter**

Save Revert Test

IP filters block or allow specific IP address ranges. Your filters are applied in order. Drag and drop a filter to change its priority in the list.

NAME	ADDRESS RANGE	ACTION
MaliciousIP	6.6.6.6/6	Block

+ Add IP Filter Rule

## Update IP filter rules in code

You may retrieve and modify your DPS IP filter using Azure resource Provider's REST endpoint. See `properties.ipFilterRules` in [createorupdate method](#).

Updating DPS IP filter rules is not currently supported with Azure CLI or Azure PowerShell but, can be accomplished with Azure Resource Manager templates. See, [Azure Resource Manager templates](#) for guidance on using Resource Manager templates. The template examples that follow show how to create, edit, and delete DPS IP filter rules.

### Add an IP filter rule

The following template example creates a new IP filter rule named "AllowAll" that accepts all traffic.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "iotDpsName": {
            "type": "string",
            "defaultValue": "[resourceGroup().name]",
            "minLength": 3,
            "metadata": {
                "description": "Specifies the name of the IoT DPS service."
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Location for IoT DPS resource."
            }
        }
    },
    "variables": {
        "iotDpsApiVersion": "2020-01-01"
    },
    "resources": [
        {
            "type": "Microsoft.Devices/provisioningServices",
            "apiVersion": "[variables('iotDpsApiVersion')]",
            "name": "[parameters('iotDpsName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "S1",
                "tier": "Standard",
                "capacity": 1
            },
            "properties": {
                "IpFilterRules": [
                    {
                        "FilterName": "AllowAll",
                        "Action": "Accept",
                        "ipMask": "0.0.0.0/0"
                    }
                ]
            }
        }
    ]
}
```

Update the IP filter rule attributes of the template based on your requirements.

ATTRIBUTE	DESCRIPTION
<b>FilterName</b>	Provide a name for the IP Filter rule. This must be a unique, case-insensitive, alphanumeric string up to 128 characters long. Only the ASCII 7-bit alphanumeric characters plus '-' , ':' , '/' , ',' , ';' , '+' , '%' , '_' , '#' , '*' , '?' , '!' , '(' , ')' , '  ' , '=' , '@' , ';;' are accepted.
<b>Action</b>	Accepted values are <b>Accept</b> or <b>Reject</b> as the action for the IP filter rule.

ATTRIBUTE	DESCRIPTION
ipMask	Provide a single IPv4 address or a block of IP addresses in CIDR notation. For example, in CIDR notation 192.168.100.0/22 represents the 1024 IPv4 addresses from 192.168.100.0 to 192.168.103.255.

## Update an IP filter rule

The following template example updates the IP filter rule named "AllowAll", shown previously, to reject all traffic.

```
{
"$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
"contentVersion": "1.0.0.0",
"parameters": {
"iotDpsName": {
"type": "string",
"defaultValue": "[resourceGroup().name]",
"minLength": 3,
"metadata": {
"description": "Specifies the name of the IoT DPS service."}
},
"location": {
"type": "string",
"defaultValue": "[resourceGroup().location]",
"metadata": {
"description": "Location for IoT DPS resource."}
},
"variables": {
"iotDpsApiVersion": "2020-01-01"
},
"resources": [
{
"type": "Microsoft.Devices/provisioningServices",
"apiVersion": "[variables('iotDpsApiVersion')]",
"name": "[parameters('iotDpsName')]",
"location": "[parameters('location')]",
"sku": {
"name": "S1",
"tier": "Standard",
"capacity": 1
},
"properties": {
"ipFilterRules": [
{
"FilterName": "AllowAll",
"Action": "Reject",
"ipMask": "0.0.0.0/0"
}
]
}
]
}
}
```

## Delete an IP filter rule

The following template example deletes all IP filter rules for the DPS instance.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "iotDpsName": {
      "type": "string",
      "defaultValue": "[resourceGroup().name]",
      "minLength": 3,
      "metadata": {
        "description": "Specifies the name of the IoT DPS service."
      }
    },
    "location": {
      "type": "string",
      "defaultValue": "[resourceGroup().location]",
      "metadata": {
        "description": "Location for IoT DPS resource."
      }
    }
  },
  "variables": {
    "iotDpsApiVersion": "2020-01-01"
  },
  "resources": [
    {
      "type": "Microsoft.Devices/provisioningServices",
      "apiVersion": "[variables('iotDpsApiVersion')]",
      "name": "[parameters('iotDpsName')]",
      "location": "[parameters('location')]",
      "sku": {
        "name": "S1",
        "tier": "Standard",
        "capacity": 1
      },
      "properties": {}
    }
  ]
}
```

## IP filter rule evaluation

IP filter rules are applied in order and the first rule that matches the IP address determines the accept or reject action.

For example, if you want to accept addresses in the range 192.168.100.0/22 and reject everything else, the first rule in the grid should accept the address range 192.168.100.0/22. The next rule should reject all addresses by using the range 0.0.0.0/0.

You can change the order of your IP filter rules in the grid by clicking the three vertical dots at the start of a row and using drag and drop.

To save your new IP filter rule order, click **Save**.

## IP Filter

 Save  Revert  Test

IP filters block or allow specific IP address ranges. Your filters are applied in order. Drag and drop a filter to change its priority in the list.

NAME	ADDRESS RANGE	ACTION
AcceptMyRange	* 192.168.100.0/22	<input checked="" type="checkbox"/> Allow 
MaliciousIP	* 0.0.0.0/0	<input type="checkbox"/> Block 
<a href="#">+ Add IP Filter Rule</a>		

## Next steps

To further explore the managing DPS, see:

- [Understanding IoT DPS IP addresses](#)
- [Configure DPS using the Azure CLI](#)
- [Control access to DPS](#)

# Create and provision an IoT Edge device with a TPM on Linux

7/30/2020 • 9 minutes to read • [Edit Online](#)

This article shows how to test auto-provisioning on a Linux IoT Edge device using a Trusted Platform Module (TPM). You can automatically provision Azure IoT Edge devices with the [Device Provisioning Service](#). If you're unfamiliar with the process of auto-provisioning, review the [auto-provisioning concepts](#) before continuing.

The tasks are as follows:

1. Create a Linux virtual machine (VM) in Hyper-V with a simulated Trusted Platform Module (TPM) for hardware security.
2. Create an instance of IoT Hub Device Provisioning Service (DPS).
3. Create an individual enrollment for the device.
4. Install the IoT Edge runtime and connect the device to IoT Hub.

## TIP

This article describes how to test DPS provisioning using a TPM simulator, but much of it applies to physical TPM hardware such as the [Infineon OPTIGA™ TPM](#), an Azure Certified for IoT device.

If you're using a physical device, you can skip ahead to the [Retrieve provisioning information from a physical device](#) section in this article.

## Prerequisites

- A Windows development machine with [Hyper-V enabled](#). This article uses Windows 10 running an Ubuntu Server VM.
- An active IoT Hub.

## NOTE

TPM 2.0 is required when using TPM attestation with DPS and can only be used to create individual, not group, enrollments.

## Create a Linux virtual machine with a virtual TPM

In this section, you create a new Linux virtual machine on Hyper-V. You configure this virtual machine with a simulated TPM for testing how automatic provisioning works with IoT Edge.

### Create a virtual switch

A virtual switch enables your virtual machine to connect to a physical network.

1. Open Hyper-V Manager on your Windows machine.
2. In the Actions menu, select **Virtual Switch Manager**.
3. Choose an **External** virtual switch, then select **Create Virtual Switch**.
4. Give your new virtual switch a name, for example **EdgeSwitch**. Make sure that the connection type is set to **External network**, then select **Ok**.

5. A pop-up warns you that network connectivity may be disrupted. Select **Yes** to continue.

If you see errors while creating the new virtual switch, ensure that no other switches are using the ethernet adaptor, and that no other switches use the same name.

### Create virtual machine

1. Download a disk image file to use for your virtual machine and save it locally. For example, [Ubuntu server 18.04](#). For information about supported operating systems for IoT Edge devices, see [Azure IoT Edge supported systems](#).
2. In Hyper-V Manager again, select **Action > New > Virtual Machine** in the **Actions** menu.
3. Complete the **New Virtual Machine Wizard** with the following specific configurations:
  - a. **Specify Generation:** Select **Generation 2**. Generation 2 virtual machines have nested virtualization enabled, which is required to run IoT Edge on a virtual machine.
  - b. **Configure Networking:** Set the value of **Connection** to the virtual switch that you created in the previous section.
  - c. **Installation Options:** Select **Install an operating system from a bootable image file** and browse to the disk image file that you saved locally.
4. Select **Finish** in the wizard to create the virtual machine.

It may take a few minutes to create the new VM.

### Enable virtual TPM

Once your VM is created, open its settings to enable the virtual trusted platform module (TPM) that lets you auto-provision the device.

1. In Hyper-V Manager, right-click on the VM and select **Settings**.
2. Navigate to **Security**.
3. Uncheck **Enable Secure Boot**.
4. Check **Enable Trusted Platform Module**.
5. Click **OK**.

### Start the virtual machine and collect TPM data

In the virtual machine, build a tool that you can use to retrieve the device's **Registration ID** and **Endorsement key**.

1. In Hyper-V Manager, start your VM and connect to it.
2. Follow the prompts within the virtual machine to finish the installation process and reboot the machine.
3. Sign in to your VM, then follow the steps in [Set up a Linux development environment](#) to install and build the Azure IoT device SDK for C.

#### TIP

In the course of this article, you'll copy and paste on the virtual machine, which is not easy through the Hyper-V Manager connection application. You may want to connect to the virtual machine through Hyper-V Manager once to retrieve its IP address. First run `sudo apt install net-tools` and then `hostname -I`. Then, you can use the IP address to connect through SSH: `ssh <username>@<ipaddress>`.

4. Run the following commands to build the SDK tool that retrieves your device provisioning information from the TPM.

```
cd azure-iot-sdk-c/cmake  
cmake -Duse_prov_client:BOOL=ON ..  
cd provisioning_client/tools/tpm_device_provision  
make  
sudo ./tpm_device_provision
```

5. The output window displays the device's **Registration ID** and the **Endorsement key**. Copy these values for use later when you create an individual enrollment for your device.

Once you have your registration ID and endorsement key, continue to the section [Set up the IoT Hub Device Provisioning Service](#)

## Retrieve provisioning information from a physical device

If you are using a physical IoT Edge device instead of a VM, build a tool that you can use to retrieve the device's provisioning information.

1. Follow the steps in [Set up a Linux development environment](#) to install and build the Azure IoT device SDK for C.
2. Run the following commands to build the SDK tool that retrieves your device provisioning information from the TPM device.

```
cd azure-iot-sdk-c/cmake  
cmake -Duse_prov_client:BOOL=ON ..  
cd provisioning_client/tools/tpm_device_provision  
make  
sudo ./tpm_device_provision
```

3. Copy the values for **Registration ID** and **Endorsement key**. You use these values to create an individual enrollment for your device in DPS.

## Set up the IoT Hub Device Provisioning Service

Create a new instance of the IoT Hub Device Provisioning Service in Azure, and link it to your IoT hub. You can follow the instructions in [Set up the IoT Hub DPS](#).

After you have the Device Provisioning Service running, copy the value of **ID Scope** from the overview page. You use this value when you configure the IoT Edge runtime.

## Create a DPS enrollment

Retrieve the provisioning information from your virtual machine, and use that to create an individual enrollment in Device Provisioning Service.

When you create an enrollment in DPS, you have the opportunity to declare an **Initial Device Twin State**. In the device twin, you can set tags to group devices by any metric you need in your solution, like region, environment, location, or device type. These tags are used to create [automatic deployments](#).

### TIP

In the Azure CLI, you can create an [enrollment](#) and use the `edge-enabled` flag to specify that a device is an IoT Edge device.

1. In the [Azure portal](#), navigate to your instance of IoT Hub Device Provisioning Service.
2. Under **Settings**, select **Manage enrollments**.

3. Select **Add individual enrollment** then complete the following steps to configure the enrollment:

- a. For **Mechanism**, select **TPM**.
- b. Provide the **Endorsement key** and **Registration ID** that you copied from your virtual machine.

**TIP**

If you're using a physical TPM device, you need to determine the **Endorsement key**, which is unique to each TPM chip and is obtained from the TPM chip manufacturer associated with it. You can derive a unique **Registration ID** for your TPM device by, for example, creating an SHA-256 hash of the endorsement key.

- c. Provide an ID for your device if you'd like. If you don't provide a device ID, the registration ID is used.
- d. Select **True** to declare that this virtual machine is an IoT Edge device.
- e. Choose the linked IoT Hub that you want to connect your device to, or select **Link to new IoT Hub**. You can choose multiple hubs, and the device will be assigned to one of them according to the selected assignment policy.
- f. Add a tag value to the **Initial Device Twin State** if you'd like. You can use tags to target groups of devices for module deployment. For more information, see [Deploy IoT Edge modules at scale](#).
- g. Select **Save**.

Now that an enrollment exists for this device, the IoT Edge runtime can automatically provision the device during installation.

## Install the IoT Edge runtime

The IoT Edge runtime is deployed on all IoT Edge devices. Its components run in containers, and allow you to deploy additional containers to the device so that you can run code at the edge. Install the IoT Edge runtime on your virtual machine.

Know your DPS ID Scope and device **Registration ID** before beginning the article that matches your device type. If you installed the example Ubuntu server, use the **x64** instructions. Make sure to configure the IoT Edge runtime for automatic, not manual, provisioning.

When you get to the step to configure the security daemon, be sure and choose [Option 2 Automatic Provisioning](#) and configure for TPM attestation.

### [Install the Azure IoT Edge runtime on Linux](#)

## Give IoT Edge access to the TPM

The IoT Edge runtime needs to access the TPM to automatically provision your device.

You can give TPM access to the IoT Edge runtime by overriding the systemd settings so that the `iotedge` service has root privileges. If you don't want to elevate the service privileges, you can also use the following steps to manually provide TPM access.

1. Find the file path to the TPM hardware module on your device and save it as a local variable.

```
tpm=$(sudo find /sys -name dev -print | grep tpm | sed 's/.\\{4\\}$//')
```

2. Create a new rule that will give the IoT Edge runtime access to tpm0.

```
sudo touch /etc/udev/rules.d/tpmaccess.rules
```

3. Open the rules file.

```
sudo nano /etc/udev/rules.d/tpmaccess.rules
```

4. Copy the following access information into the rules file.

```
# allow iotedge access to tpm0
KERNEL=="tpm0", SUBSYSTEM=="tpm", OWNER="iotedge", MODE="0600"
```

5. Save and exit the file.

6. Trigger the udev system to evaluate the new rule.

```
/bin/udevadm trigger $tpm
```

7. Verify that the rule was successfully applied.

```
ls -l /dev/tpm0
```

Successful output appears as follows:

```
crw-rw---- 1 root iotedge 10, 224 Jul 20 16:27 /dev/tpm0
```

If you don't see that the correct permissions have been applied, try rebooting your machine to refresh udev.

## Restart the IoT Edge runtime

Restart the IoT Edge runtime so that it picks up all the configuration changes that you made on the device.

```
sudo systemctl restart iotedge
```

Check to see that the IoT Edge runtime is running.

```
sudo systemctl status iotedge
```

If you see provisioning errors, it may be that the configuration changes haven't taken effect yet. Try restarting the IoT Edge daemon again.

```
sudo systemctl daemon-reload
```

Or, try restarting your virtual machine to see if the changes take effect on a fresh start.

## Verify successful installation

If the runtime started successfully, you can go into your IoT Hub and see that your new device was automatically provisioned. Now your device is ready to run IoT Edge modules.

Check the status of the IoT Edge Daemon.

```
systemctl status iotedge
```

Examine daemon logs.

```
journalctl -u iotedge --no-pager --no-full
```

List running modules.

```
iotedge list
```

You can verify that the individual enrollment that you created in Device Provisioning Service was used. Navigate to your Device Provisioning Service instance in the Azure portal. Open the enrollment details for the individual enrollment that you created. Notice that the status of the enrollment is **assigned** and the device ID is listed.

## Next steps

The DPS enrollment process lets you set the device ID and device twin tags at the same time as you provision the new device. You can use those values to target individual devices or groups of devices using automatic device management. Learn how to [Deploy and monitor IoT Edge modules at scale using the Azure portal](#) or [using Azure CLI](#).

# Create and provision a simulated IoT Edge device with a virtual TPM on Windows

4/24/2020 • 5 minutes to read • [Edit Online](#)

Azure IoT Edge devices can be auto-provisioned using the [Device Provisioning Service](#) just like devices that are not edge-enabled. If you're unfamiliar with the process of auto-provisioning, review the [auto-provisioning concepts](#) before continuing.

DPS supports symmetric key attestation for IoT Edge devices in both individual enrollment and group enrollment. For group enrollment, if you check "is IoT Edge device" option to be true in symmetric key attestation, all the devices that are registered under that enrollment group will be marked as IoT Edge devices.

This article shows you how to test auto-provisioning on a simulated IoT Edge device with the following steps:

- Create an instance of IoT Hub Device Provisioning Service (DPS).
- Create a simulated device on your Windows machine with a simulated Trusted Platform Module (TPM) for hardware security.
- Create an individual enrollment for the device.
- Install the IoT Edge runtime and connect the device to IoT Hub.

## TIP

This article describes testing auto-provisioning by using TPM attestation on virtual devices, but much of it applies when using physical TPM hardware as well.

## Prerequisites

- A Windows development machine. This article uses Windows 10.
- An active IoT Hub.

## NOTE

TPM 2.0 is required when using TPM attestation with DPS and can only be used to create individual, not group, enrollments.

## Set up the IoT Hub Device Provisioning Service

Create a new instance of the IoT Hub Device Provisioning Service in Azure, and link it to your IoT hub. You can follow the instructions in [Set up the IoT Hub DPS](#).

After you have the Device Provisioning Service running, copy the value of **ID Scope** from the overview page. You use this value when you configure the IoT Edge runtime.

**TIP**

If you're using a physical TPM device, you need to determine the **Endorsement key**, which is unique to each TPM chip and is obtained from the TPM chip manufacturer associated with it. You can derive a unique **Registration ID** for your TPM device by, for example, creating an SHA-256 hash of the endorsement key.

Follow the instructions in the article [How to manage device enrollments with Azure Portal](#) to create your enrollment in DPS and then proceed with the [Install the IoT Edge runtime](#) section in this article to continue.

## Simulate a TPM device

Create a simulated TPM device on your Windows development machine. Retrieve the **Registration ID** and **Endorsement key** for your device, and use them to create an individual enrollment entry in DPS.

When you create an enrollment in DPS, you have the opportunity to declare an **Initial Device Twin State**. In the device twin you can set tags to group devices by any metric you need in your solution, like region, environment, location, or device type. These tags are used to create [automatic deployments](#).

Choose the SDK language that you want to use to create the simulated device, and follow the steps until you create the individual enrollment.

When you create the individual enrollment, select **True** to declare that the simulated TPM device on your Windows development machine is an **IoT Edge device**.

**TIP**

In the Azure CLI, you can create an [enrollment](#) or an [enrollment group](#) and use the **edge-enabled** flag to specify that a device, or group of devices, is an IoT Edge device.

Simulated device and individual enrollment guides:

- [C](#)
- [Java](#)
- [C#](#)
- [Node.js](#)
- [Python](#)

After creating the individual enrollment, save the value of the **Registration ID**. You use this value when you configure the IoT Edge runtime.

## Install the IoT Edge runtime

The IoT Edge runtime is deployed on all IoT Edge devices. Its components run in containers, and allow you to deploy additional containers to the device so that you can run code at the edge.

You'll need the following information when provisioning your device:

- The DPS **ID Scope** value
- The device **Registration ID** you created

Install the IoT Edge runtime on the device that is running the simulated TPM. You'll configure the IoT Edge runtime for automatic, not manual, provisioning.

**TIP**

Keep the window that's running the TPM simulator open during your installation and testing.

For more detailed information about installing IoT Edge on Windows, including prerequisites and instructions for tasks like managing containers and updating IoT Edge, see [Install the Azure IoT Edge runtime on Windows](#).

1. Open a PowerShell window in administrator mode. Be sure to use an AMD64 session of PowerShell when installing IoT Edge, not PowerShell (x86).
2. The **Deploy-IoTEdge** command checks that your Windows machine is on a supported version, turns on the containers feature, and then downloads the moby runtime and the IoT Edge runtime. The command defaults to using Windows containers.

```
. {Invoke-WebRequest -useb https://aka.ms/iotedge-win} | Invoke-Expression; `  
Deploy-IoTEdge
```

3. At this point, IoT Core devices may restart automatically. Other Windows 10 or Windows Server devices may prompt you to restart. If so, restart your device now. Once your device is ready, run PowerShell as an administrator again.
4. The **Initialize-IoTEdge** command configures the IoT Edge runtime on your machine. The command defaults to manual provisioning with Windows containers. Use the **-Dps** flag to use the Device Provisioning Service instead of manual provisioning.

Replace the placeholder values for `{scope_id}` and `{registration_id}` with the data you collected earlier.

```
. {Invoke-WebRequest -useb https://aka.ms/iotedge-win} | Invoke-Expression; `  
Initialize-IoTEdge -Dps -ScopeId {scope ID} -RegistrationId {registration ID}
```

## Verify successful installation

If the runtime started successfully, you can go into your IoT Hub and start deploying IoT Edge modules to your device. Use the following commands on your device to verify that the runtime installed and started successfully.

Check the status of the IoT Edge service.

```
Get-Service iotedge
```

Examine service logs from the last 5 minutes.

```
. {Invoke-WebRequest -useb aka.ms/iotedge-win} | Invoke-Expression; Get-IoTEdgeLog
```

List running modules.

```
iotedge list
```

## Next steps

The Device Provisioning Service enrollment process lets you set the device ID and device twin tags at the same time as you provision the new device. You can use those values to target individual devices or groups of devices using

automatic device management. Learn how to [Deploy and monitor IoT Edge modules at scale using the Azure portal](#) or [using Azure CLI](#)

# How to use different attestation mechanisms with Device Provisioning Service Client SDK for C

4/20/2020 • 6 minutes to read • [Edit Online](#)

This article shows you how to use different [attestation mechanisms](#) with the Device Provisioning Service Client SDK for C. You can use either a physical device or a simulator. The provisioning service supports authentication for two types of attestation mechanisms: X.509 and Trusted Platform Module (TPM).

## Prerequisites

Prepare your development environment according to the section titled "Prepare the development environment" in the [Create and provision simulated device](#) guide.

### Choose an attestation mechanism

As a device manufacturer, you first need to choose an attestation mechanism based on one of the supported types. Currently, the [Device Provisioning Service client SDK for C](#) provides support for the following attestation mechanisms:

- **Trusted Platform Module (TPM)**: TPM is an established standard for most Windows-based device platforms, as well as a few Linux/Ubuntu based devices. As a device manufacturer, you may choose this attestation mechanism if you have either of these OSes running on your devices, and you are looking for an established standard. With TPM chips, you can only enroll each device individually to the Device Provisioning Service. For development purposes, you can use the TPM simulator on your Windows or Linux development machine.
- **X.509**: X.509 certificates can be stored in relatively newer chips called [Hardware Security Modules \(HSM\)](#). Work is also progressing within Microsoft, on RIOT or DICE chips, which implement the X.509 certificates. With X.509 chips, you can do bulk device enrollment in the portal. It also supports certain non-Windows OSes like embedOS. For development purpose, the Device Provisioning Service client SDK supports an X.509 device simulator.

For more information, see IoT Hub Device Provisioning Service [security concepts](#) and [auto-provisioning concepts](#).

## Enable authentication for supported attestation mechanisms

The SDK authentication mode (X.509 or TPM) must be enabled for the physical device or simulator before they can be enrolled in the Azure portal. First, navigate to the root folder for azure-iot-sdk-c. Then run the specified command, depending on the authentication mode you choose:

### Use X.509 with simulator

The provisioning service ships with a Device Identity Composition Engine (DICE) emulator that generates an X.509 certificate for authenticating the device. To enable X.509 authentication, run the following command:

```
cmake -Ddps_auth_type=x509 ..
```

Information regarding hardware with DICE can be found [here](#).

### Use X.509 with hardware

The provisioning service can be used with X.509 on other hardware. An interface between hardware and the SDK is needed to establish connection. Talk to your HSM manufacturer for information on the interface.

## Use TPM

The provisioning service can connect to Windows and Linux hardware TPM chips with SAS Token. To enable TPM authentication, run the following command:

```
cmake -Ddps_auth_type=tpm ..
```

## Use TPM with simulator

If you don't have a device with TPM chips, you can use a simulator for development purpose on Windows OS. To enable TPM authentication and run the TPM simulator, run the following command:

```
cmake -Ddps_auth_type=tpm_simulator ..
```

## Build the SDK

Build the SDK prior to creating device enrollment.

### Linux

- To build the SDK in Linux:

```
cd azure-iot-sdk-c  
mkdir cmake  
cd cmake  
cmake ..  
cmake --build . # append '-- -j <n>' to run <n> jobs in parallel
```

- To build Debug binaries, add the corresponding CMake option to the project generation command, for example:

```
cmake -DCMAKE_BUILD_TYPE=Debug ..
```

- There are many [CMake configuration options](#) available for building the SDK. For example, you can disable one of the available protocol stacks by adding an argument to the CMake project generation command:

```
cmake -Duse_amqp=OFF ..
```

- You can also build and run unit test:

```
cmake -Drun_unittests=ON ..  
cmake --build .  
ctest -C "debug" -V
```

### Windows

- To build the SDK in Windows, take the following steps to generate project files:
  - Open a "Developer Command Prompt for VS2015"
  - Run the following CMake commands from the root of the repository:

```
cd azure-iot-sdk-c  
mkdir cmake  
cd cmake  
cmake -G "Visual Studio 14 2015" ..
```

This command builds x86 libraries. To build for x64, modify the cmake generator argument:

```
cmake .. -G "Visual Studio 14 2015 Win64"
```

- If project generation completes successfully, you should see a Visual Studio solution file (.sln) under the `cmake` folder. To build the SDK:

- Open `cmake\azure_iot_sdks.sln` in Visual Studio and build it, OR
- Run the following command in the command prompt you used to generate the project files:

```
cmake --build . -- /m /p:Configuration=Release
```

- To build Debug binaries, use the corresponding MSBuild argument:

```
cmake --build . -- /m /p:Configuration=Debug`
```

- There are many CMake configuration options available for building the SDK. For example, you can disable one of the available protocol stacks by adding an argument to the CMake project generation command:

```
cmake -G "Visual Studio 14 2015" -Duse_amqp=OFF ..
```

- Also, you can build and run unit tests:

```
cmake -G "Visual Studio 14 2015" -Drun_unittests=ON ..  
cmake --build . -- /m /p:Configuration=Debug  
ctest -C "debug" -V
```

## Libraries to include

- These libraries should be included in your SDK:
  - The provisioning service: `dps_http_transport`, `dps_client`, `dps_security_client`
  - IoTHub Security: `iothub_security_client`

## Create a device enrollment entry in Device Provisioning Services

### TPM

If you are using TPM, follow instructions in "[Create and provision a simulated device using IoT Hub Device Provisioning Service](#)" to create a device enrollment entry in your Device Provisioning Service and simulate first boot.

### X.509

1. To enroll a device in the provisioning service, you need note down the Endorsement Key and Registration ID for each device, which are displayed in the Provisioning Tool provided by Client SDK. Run the following command to print out the root CA certificate (for enrollment groups) and the leaf certificate (for individual enrollment):

```
./azure-iot-sdk-c/dps_client/tools/x509_device_provision/x509_device_provision.exe
```

2. Sign in to the Azure portal, click on the **All resources** button on the left-hand menu and open your Device Provisioning service.

- **X.509 Individual Enrollment:** On the provisioning service summary blade, select **Manage enrollments**. Select **Individual Enrollments** tab and click the **Add** button at the top. Select **X.509** as the identity attestation *Mechanism*, upload the leaf certificate as required by the blade. Once complete, click the **Save** button.
- **X.509 Group Enrollment:** On the provisioning service summary blade, select **Manage enrollments**. Select **Group Enrollments** tab and click the **Add** button at the top. Select **X.509** as the identity attestation *Mechanism*, enter a group name and certification name, upload the CA/Intermediate certificate as required by the blade. Once complete, click the **Save** button.

## Enable authentication for devices using a custom attestation mechanism (optional)

### NOTE

This section is only applicable to devices that require support for a custom platform or attestation mechanisms, not currently supported by the Device Provisioning Service Client SDK for C. Also note, the SDK frequently uses the term "HSM" as a generic substitute in place of "attestation mechanism."

First you need to develop a repository and library for your custom attestation mechanism:

1. Develop a library to access your attestation mechanism. This project needs to produce a static library for the Device Provisioning SDK to consume.
2. Implement the functions defined in the following header file, in your library:
  - For a custom TPM: implement the functions defined under [HSM TPM API](#).
  - For a custom X.509: implement the functions defined under [HSM X509 API](#).

Once your library successfully builds on its own, you need to integrate it with the Device Provisioning Service Client SDK, by linking against your library.:

1. Supply the custom GitHub repository and the library in the following `cmake` command:

```
cmake -Duse_prov_client:BOOL=ON -Dhsm_custom_lib=<path_and_name_of_library> <PATH_TO_AZURE_IOT_SDK>
```

2. Open the Visual Studio solution file built by CMake (`\azure-iot-sdk-c\cmake\azure_iot_sdks.sln`), and build it.
  - The build process compiles the SDK library.
  - The SDK attempts to link against the custom library defined in the `cmake` command.
3. To verify that your custom attestation mechanism is implemented correctly, run the "prov\_dev\_client\_ll\_sample" sample app under "Provision\_Samples" (under `\azure-iot-sdk-c\cmake\provisioning_client\samples\prov_dev_client_ll_sample`).

## Connecting to IoT Hub after provisioning

Once the device has been provisioned with the provisioning service, this API uses the specified authentication mode (X.509 or TPM) to connect with IoT Hub:

```
IOTHUB_CLIENT_LL_HANDLE handle = IoTHubClient_LL_CreateFromDeviceAuth(iothub_uri, device_id, iothub_transport);
```

# How to provision for multitenancy

12/19/2019 • 12 minutes to read • [Edit Online](#)

The allocation policies defined by the provisioning service support a variety of allocation scenarios. Two common scenarios are:

- **Geolocation / GeoLatency:** As a device moves between locations, network latency is improved by having the device provisioned to the IoT hub closest to each location. In this scenario, a group of IoT hubs, which span across regions, are selected for enrollments. The **Lowest latency** allocation policy is selected for these enrollments. This policy causes the Device Provisioning Service to evaluate device latency and determine the closest IoT hub out of the group of IoT hubs.
- **Multi-tenancy:** Devices used within an IoT solution may need to be assigned to a specific IoT hub or group of IoT hubs. The solution may require all devices for a particular tenant to communicate with a specific group of IoT hubs. In some cases, a tenant may own IoT hubs and require devices to be assigned to their IoT hubs.

It is common to combine these two scenarios. For example, a multitenant IoT solution will commonly assign tenant devices using a group of IoT hubs that are scattered across regions. These tenant devices can be assigned to the IoT hub in that group, which has the lowest latency based on geographic location.

This article uses a simulated device sample from the [Azure IoT C SDK](#) to demonstrate how to provision devices in a multitenant scenario across regions. You will perform the following steps in this article:

- Use the Azure CLI to create two regional IoT hubs (**West US** and **East US**)
- Create a multitenant enrollment
- Use the Azure CLI to create two regional Linux VMs to act as devices in the same regions (**West US** and **East US**)
- Set up the development environment for the Azure IoT C SDK on both Linux VMs
- Simulate the devices to see that they are provisioned for the same tenant in the closest region.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Completion of the [Set up IoT Hub Device Provisioning Service with the Azure portal](#) quickstart.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	

OPTION	EXAMPLE/LINK
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the <b>Launch Cloud Shell</b> button to open Cloud Shell in your browser.	
Select the <b>Cloud Shell</b> button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

## Create two regional IoT hubs

In this section, you will use the Azure Cloud Shell to create two new regional IoT hubs in the **West US** and **East US** regions for a tenant.

1. Use the Azure Cloud Shell to create a resource group with the [az group create](#) command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named *contoso-us-resource-group* in the *eastus* region. It is recommended that you use this group for all resources created in this article. This will make clean up easier after you are finished.

```
az group create --name contoso-us-resource-group --location eastus
```

2. Use the Azure Cloud Shell to create an IoT hub in the **eastus** region with the [az iot hub create](#) command. The IoT hub will be added to the *contoso-us-resource-group*.

The following example creates an IoT hub named *contoso-east-hub* in the *eastus* location. You must use your own unique hub name instead of **contoso-east-hub**.

```
az iot hub create --name contoso-east-hub --resource-group contoso-us-resource-group --location eastus
--sku S1
```

This command may take a few minutes to complete.

3. Use the Azure Cloud Shell to create an IoT hub in the **westus** region with the [az iot hub create](#) command. This IoT hub will also be added to the *contoso-us-resource-group*.

The following example creates an IoT hub named *contoso-west-hub* in the *westus* location. You must use your own unique hub name instead of **contoso-west-hub**.

```
az iot hub create --name contoso-west-hub --resource-group contoso-us-resource-group --location westus
--sku S1
```

This command may take a few minutes to complete.

# Create the multitenant enrollment

In this section, you will create a new enrollment group for the tenant devices.

For simplicity, this article uses [Symmetric key attestation](#) with the enrollment. For a more secure solution, consider using [X.509 certificate attestation](#) with a chain of trust.

1. Sign in to the [Azure portal](#), and open your Device Provisioning Service instance.
2. Select the **Manage enrollments** tab, and then click the **Add enrollment group** button at the top of the page.
3. On **Add Enrollment Group**, enter the following information, and click the **Save** button.

**Group name:** Enter **contoso-us-devices**.

**Attestation Type:** Select **Symmetric Key**.

**Auto Generate Keys:** This checkbox should already be checked.

**Select how you want to assign devices to hubs:** Select **Lowest latency**.

The screenshot shows the 'Add Enrollment Group' dialog box. The 'Group name' field is filled with 'contoso-us-devices'. The 'Attestation Type' dropdown has 'Symmetric Key (preview)' selected. The 'Auto-generate keys' checkbox is checked. The 'Device Twin' note states it's only supported for standard tier IoT hubs. The 'Select how you want to assign devices to hubs' dropdown is set to 'Lowest latency'. The 'Select the IoT hubs this group can be assigned to' dropdown shows '0 selected'.

4. On **Add Enrollment Group**, click **Link a new IoT hub** to link both of your regional hubs.

**Subscription:** If you have multiple subscriptions, choose the subscription where you created the regional IoT hubs.

**IoT hub:** Select one of the regional hubs you created.

**Access Policy:** Choose **iothubowner**.

**Add link to IoT hub**

Learn more about linking IoT hubs.

\* Subscription

\* IoT hub

\* Access Policy

Hostname

Status

Pricing Tier

Location

**Save**

- Once both regional IoT hubs have been linked, you must select them for the enrollment group and click **Save** to create the regional IoT hub group for the enrollment.

**Add Enrollment Group**

**Save**

\* Group name

Attestation Type  Certificate  Symmetric Key (preview)

Auto-generate keys

\* Primary Key

\* Secondary Key

Device Twin is only supported for standard tier IoT hubs. Learn more about standard tier.

Select how you want to assign devices to hubs (preview for enrollment)  Lowest latency

Select the IoT hubs this group can be assigned to: (preview)  2 selected

- contoso-east-hub.azure-devices.net
- contoso-west-hub.azure-devices.net

- After saving the enrollment, reopen it and make a note of the **Primary Key**. You must save the enrollment first to have the keys generated. This key will be used to generate unique device keys for both simulated devices later.

## Create regional Linux VMs

In this section, you will create two regional Linux virtual machines (VMs). These VMs will run a device simulation sample from each region to demonstrate device provisioning for tenant devices from both regions.

To make clean-up easier, these VMs will be added to the same resource group that contains the IoT hubs that were created, *contoso-us-resource-group*. However, the VMs will run in separate regions (**West US** and **East US**).

1. In the Azure Cloud Shell, execute the following command to create an **East US** region VM after making the following parameter changes in the command:

--name: Enter a unique name for your **East US** regional device VM.

--admin-username: Use your own admin user name.

--admin-password: Use your own admin password.

```
az vm create \
--resource-group contoso-us-resource-group \
--name ContosoSimDeviceEast \
--location eastus \
--image Canonical:UbuntuServer:18.04-LTS:18.04.201809110 \
--admin-username contosoadmin \
--admin-password myContosoPassword2018 \
--authentication-type password
```

This command will take a few minutes to complete. Once the command has completed, make a note of the **publicIpAddress** value for your East US region VM.

2. In the Azure Cloud Shell, execute the command to create a **West US** region VM after making the following parameter changes in the command:

--name: Enter a unique name for your **West US** regional device VM.

--admin-username: Use your own admin user name.

--admin-password: Use your own admin password.

```
az vm create \
--resource-group contoso-us-resource-group \
--name ContosoSimDeviceWest \
--location westus \
--image Canonical:UbuntuServer:18.04-LTS:18.04.201809110 \
--admin-username contosoadmin \
--admin-password myContosoPassword2018 \
--authentication-type password
```

This command will take a few minutes to complete. Once the command has completed, make a note of the **publicIpAddress** value for your West US region VM.

3. Open two command-line shells. Connect to one of the regional VMs in each shell using SSH.

Pass your admin username, and the public IP address you noted for the VM as parameters to SSH. Enter the admin password when prompted.

```
ssh contosoadmin@1.2.3.4
contosoadmin@ContosoSimDeviceEast:~$
```

```
ssh contosoadmin@5.6.7.8  
contosoadmin@ContosoSimDeviceWest:~$
```

## Prepare the Azure IoT C SDK development environment

In this section, you will clone the Azure IoT C SDK on each VM. The SDK contains a sample that will simulate a tenant's device provisioning from each region.

1. For each VM, install **CMake**, **g++**, **gcc**, and **Git** using the following commands:

```
sudo apt-get update  
sudo apt-get install cmake build-essential libssl-dev libcurl4-openssl-dev uuid-dev git-all
```

2. Find the tag name for the [latest release](#) of the SDK.
3. Clone the [Azure IoT C SDK](#) on both VMs. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git  
cd azure-iot-sdk-c  
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. For both VMs, create a new **cmake** folder inside the repository and change to that folder.

```
mkdir ~/azure-iot-sdk-c/cmake  
cd ~/azure-iot-sdk-c/cmake
```

5. For both VMs, run the following command, which builds a version of the SDK specific to your development client platform.

```
cmake -Dhsms_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
```

Once the build succeeds, the last few output lines will look similar to the following output:

```
-- IoT Client SDK Version = 1.2.9  
-- Provisioning client ON  
-- Provisioning SDK Version = 1.2.9  
-- target architecture: x86_64  
-- Checking for module 'libcurl'  
--   Found libcurl, version 7.58.0  
-- Found CURL: curl  
-- target architecture: x86_64  
-- target architecture: x86_64  
-- target architecture: x86_64  
-- target architecture: x86_64  
-- iothub architecture: x86_64  
-- target architecture: x86_64  
-- Configuring done  
-- Generating done  
-- Build files have been written to: /home/contosoadmin/azure-iot-sdk-c/cmake
```

## Derive unique device keys

When using symmetric key attestation with group enrollments, you don't use the enrollment group keys directly. Instead you create a unique derived key for each device and mentioned in [Group Enrollments with symmetric keys](#).

To generate the device key, use the group master key to compute an [HMAC-SHA256](#) of the unique registration ID for the device and convert the result into Base64 format.

Do not include your group master key in your device code.

Use the Bash shell example to create a derived device key for each device using `openssl`.

- Replace the value for `KEY` with the **Primary Key** you noted earlier for your enrollment.
- Replace the value for `REG_ID` with your own unique registration ID for each device. Use lowercase alphanumeric and dash ('-') characters to define both IDs.

Example device key generation for *contoso-simdevice-east*.

```
KEY=rLuyBPpIJ+h0re2SFIP9Ajvdty3j0EwSP/WvTVH9eZAw5HpDuEmf13nziHy5RRXmuTy84FCLpOnhhBPASSbHYg==  
REG_ID=contoso-simdevice-east  
  
keybytes=$(echo $KEY | base64 --decode | xxd -p -u -c 1000)  
echo -n $REG_ID | openssl sha256 -mac HMAC -macopt hexkey:$keybytes -binary | base64
```

```
p3w2DQr9WqEGBLUS1Fi1jPQ7UWQL4siAGy75HFTFbf8=
```

Example device key generation for *contoso-simdevice-west*.

```
KEY=rLuyBPpIJ+h0re2SFIP9Ajvdty3j0EwSP/WvTVH9eZAw5HpDuEmf13nziHy5RRXmuTy84FCLpOnhhBPASSbHYg==  
REG_ID=contoso-simdevice-west  
  
keybytes=$(echo $KEY | base64 --decode | xxd -p -u -c 1000)  
echo -n $REG_ID | openssl sha256 -mac HMAC -macopt hexkey:$keybytes -binary | base64
```

```
J5n4NY2GiBYy7Mp4lDDa5CbEe6zDU/c62rhjCuFWxnc=
```

The tenant devices will each use their derived device key and unique registration ID to perform symmetric key attestation with the enrollment group during provisioning to the tenant IoT hubs.

## Simulate the devices from each region

In this section, you will update a provisioning sample in the Azure IoT C SDK for both of the regional VMs.

The sample code simulates a device boot sequence that sends the provisioning request to your Device Provisioning Service instance. The boot sequence will cause the device to be recognized and assigned to the IoT hub that is closest based on latency.

1. In the Azure portal, select the **Overview** tab for your Device Provisioning service and note down the **ID Scope** value.

2. Open `~/azure-iot-sdk-c/provisioning_client/samples/prov_dev_client_sample/prov_dev_client_sample.c` for editing on both VMs.

```
vi ~/azure-iot-sdk-c/provisioning_client/samples/prov_dev_client_sample/prov_dev_client_sample.c
```

3. Find the `id_scope` constant, and replace the value with your **ID Scope** value that you copied earlier.

```
static const char* id_scope = "0ne00002193";
```

4. Find the definition for the `main()` function in the same file. Make sure the `hsm_type` variable is set to `SECURE_DEVICE_TYPE_SYMMETRIC_KEY` as shown below to match the enrollment group attestation method.

Save your changes to the files on both VMs.

```
SECURE_DEVICE_TYPE hsm_type;
//hsm_type = SECURE_DEVICE_TYPE_TPM;
//hsm_type = SECURE_DEVICE_TYPE_X509;
hsm_type = SECURE_DEVICE_TYPE_SYMMETRIC_KEY;
```

5. On both VMs, find the call to `prov_dev_set_symmetric_key_info()` in `prov_dev_client_sample.c` which is commented out.

```
// Set the symmetric key if using they auth type
//prov_dev_set_symmetric_key_info("<symm_registration_id>", "<symmetric_Key>");
```

Uncomment the function calls, and replace the placeholder values (including the angle brackets) with the unique registration IDs and derived device keys for each device. The keys shown below are for example purposes only. Use the keys you generated earlier.

East US:

```
// Set the symmetric key if using they auth type
prov_dev_set_symmetric_key_info("contoso-simdevice-east",
"p3w2DQr9WqEGBLUS1Fi1jPQ7UWQL4siAGy75HFTFbf8=");
```

West US:

```
// Set the symmetric key if using they auth type
prov_dev_set_symmetric_key_info("contoso-simdevice-west",
"J5n4NY2GiBYy7Mp4lDDa5CbEe6zDU/c62rhjCuFWxnc=");
```

Save the files.

6. On both VMs, navigate to the sample folder shown below, and build the sample.

```
cd ~/azure-iot-sdk-c/cmake/provisioning_client/samples/prov_dev_client_sample/
cmake --build . --target prov_dev_client_sample --config Debug
```

7. Once the build succeeds, run **prov\_dev\_client\_sample.exe** on both VMs to simulate a tenant device from each region. Notice that each device is allocated to the tenant IoT hub closest to the simulated device's regions.

Run the simulation:

```
~/azure-iot-sdk-c/cmake/provisioning_client/samples/prov_dev_client_sample/prov_dev_client_sample
```

Example output from the East US VM:

```
contosoadmin@ContosoSimDeviceEast:~/azure-iot-sdk-
c/cmake/provisioning_client/samples/prov_dev_client_sample$ ./prov_dev_client_sample
Provisioning API Version: 1.2.9

Registering Device

Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING

Registration Information received from service: contoso-east-hub.azure-devices.net, deviceId: contoso-
simdevice-east
Press enter key to exit:
```

Example output from the West US VM:

```
contosoadmin@ContosoSimDeviceWest:~/azure-iot-sdk-
c/cmake/provisioning_client/samples/prov_dev_client_sample$ ./prov_dev_client_sample
Provisioning API Version: 1.2.9

Registering Device

Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING

Registration Information received from service: contoso-west-hub.azure-devices.net, deviceId: contoso-
simdevice-west
Press enter key to exit:
```

## Clean up resources

If you plan to continue working with resources created in this article, you can leave them. If you do not plan to continue using the resource, use the following steps to delete all resources created by this article to avoid unnecessary charges.

The steps here assume you created all resources in this article as instructed in the same resource group named **contoso-us-resource-group**.

### IMPORTANT

Deleting a resource group is irreversible. The resource group and all the resources contained in it are permanently deleted. Make sure that you do not accidentally delete the wrong resource group or resources. If you created the IoT Hub inside an existing resource group that contains resources you want to keep, only delete the IoT Hub resource itself instead of deleting the resource group.

To delete the resource group by name:

1. Sign in to the [Azure portal](#) and click **Resource groups**.
2. In the **Filter by name...** textbox, type the name of the resource group containing your resources, **contoso-us-resource-group**.
3. To the right of your resource group in the result list, click ... then **Delete resource group**.
4. You will be asked to confirm the deletion of the resource group. Type the name of your resource group again to confirm, and then click **Delete**. After a few moments, the resource group and all of its contained resources are deleted.

## Next steps

- To learn more Reprovisioning, see [IoT Hub Device reprovisioning concepts](#)
- To learn more Deprovisioning, see [How to deprovision devices that were previously auto-provisioned](#)

# How to use custom allocation policies

12/19/2019 • 19 minutes to read • [Edit Online](#)

A custom allocation policy gives you more control over how devices are assigned to an IoT hub. This is accomplished by using custom code in an [Azure Function](#) to assign devices to an IoT hub. The device provisioning service calls your Azure Function code providing all relevant information about the device and the enrollment. Your function code is executed and returns the IoT hub information used to provisioning the device.

By using custom allocation policies, you define your own allocation policies when the policies provided by the Device Provisioning Service don't meet the requirements of your scenario.

For example, maybe you want to examine the certificate a device is using during provisioning and assign the device to an IoT hub based on a certificate property. Or, maybe you have information stored in a database for your devices and need to query the database to determine which IoT hub a device should be assigned to.

This article demonstrates a custom allocation policy using an Azure Function written in C#. Two new IoT hubs are created representing a *Contoso Toasters Division* and a *Contoso Heat Pumps Division*. Devices requesting provisioning must have a registration ID with one of the following suffixes to be accepted for provisioning:

- **-contoso-tstrsd-007:** Contoso Toasters Division
- **-contoso-hpsd-088:** Contoso Heat Pumps Division

The devices will be provisioned based on one of these required suffixes on the registration ID. These devices will be simulated using a provisioning sample included in the [Azure IoT C SDK](#).

You perform the following steps in this article:

- Use the Azure CLI to create two Contoso division IoT hubs ([Contoso Toasters Division](#) and [Contoso Heat Pumps Division](#))
- Create a new group enrollment using an Azure Function for the custom allocation policy
- Create device keys for two device simulations.
- Set up the development environment for the Azure IoT C SDK
- Simulate the devices and verify that they are provisioned according to the example code in the custom allocation policy

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

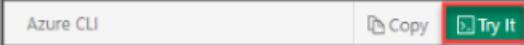
The following prerequisites are for a Windows development environment. For Linux or macOS, see the appropriate section in [Prepare your development environment](#) in the SDK documentation.

- [Visual Studio 2019](#) with the '[Desktop development with C++](#)' workload enabled. Visual Studio 2015 and Visual Studio 2017 are also supported.
- Latest version of [Git](#) installed.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the Copy button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

## Create the provisioning service and two divisional IoT hubs

In this section, you use the Azure Cloud Shell to create a provisioning service and two IoT hubs representing the **Contoso Toasters Division** and the **Contoso Heat Pumps division**.

### TIP

The commands used in this article create the provisioning service and other resources in the West US location. We recommend that you create your resources in the region nearest you that supports Device Provisioning Service. You can view a list of available locations by running the command

```
az provider show --namespace Microsoft.Devices --query "resourceTypes[?resourceType=='ProvisioningServices'].locations | [0]" --out table
```

or by going to the [Azure Status](#) page and searching for "Device Provisioning Service". In commands, locations can be specified either in one word or multi-word format; for example: westus, West US, WEST US, etc. The value is not case sensitive. If you use multi-word format to specify location, enclose the value in quotes; for example,

```
-- location "West US"
```

1. Use the Azure Cloud Shell to create a resource group with the [az group create](#) command. An Azure resource group is a logical container into which Azure resources are deployed and managed.

The following example creates a resource group named *contoso-us-resource-group* in the *westus* region. It is recommended that you use this group for all resources created in this article. This approach will make clean up easier after you're finished.

```
az group create --name contoso-us-resource-group --location westus
```

2. Use the Azure Cloud Shell to create a device provisioning service with the [az iot dps create](#) command. The provisioning service will be added to *contoso-us-resource-group*.

The following example creates a provisioning service named *contoso-provisioning-service-1098* in the *westus* location. You must use a unique service name. Make up your own suffix in the service name in place

of 1098.

```
az iot dps create --name contoso-provisioning-service-1098 --resource-group contoso-us-resource-group --location westus
```

This command may take a few minutes to complete.

3. Use the Azure Cloud Shell to create the **Contoso Toasters Division** IoT hub with the [az iot hub create](#) command. The IoT hub will be added to *contoso-us-resource-group*.

The following example creates an IoT hub named *contoso-toasters-hub-1098* in the *westus* location. You must use a unique hub name. Make up your own suffix in the hub name in place of **1098**. The example code for the custom allocation policy requires `-toasters-` in the hub name.

```
az iot hub create --name contoso-toasters-hub-1098 --resource-group contoso-us-resource-group --location westus --sku S1
```

This command may take a few minutes to complete.

4. Use the Azure Cloud Shell to create the **Contoso Heat Pumps Division** IoT hub with the [az iot hub create](#) command. This IoT hub will also be added to *contoso-us-resource-group*.

The following example creates an IoT hub named *contoso-heatpumps-hub-1098* in the *westus* location. You must use a unique hub name. Make up your own suffix in the hub name in place of **1098**. The example code for the custom allocation policy requires `-heatpumps-` in the hub name.

```
az iot hub create --name contoso-heatpumps-hub-1098 --resource-group contoso-us-resource-group --location westus --sku S1
```

This command may take a few minutes to complete.

## Create the custom allocation function

In this section, you create an Azure function that implements your custom allocation policy. This function decides which divisional IoT hub a device should be registered to based on whether its registration ID contains the string `-contoso-tstrsd-007` or `-contoso-hpsd-088`. It also sets the initial state of the device twin based on whether the device is a toaster or a heat pump.

1. Sign in to the [Azure portal](#). From your home page, select **+ Create a resource**.
2. In the *Search the Marketplace* search box, type "Function App". From the drop-down list select **Function App**, and then select **Create**.
3. On **Function App** create page, under the **Basics** tab, enter the following settings for your new function app and select **Review + create**:

**Resource Group:** Select the **contoso-us-resource-group** to keep all resources created in this article together.

**Function App name:** Enter a unique function app name. This example uses **contoso-function-app-1098**.

**Publish:** Verify that **Code** is selected.

**Runtime Stack:** Select **.NET Core** from the drop-down.

**Region:** Select the same region as your resource group. This example uses **West US**.

#### NOTE

By default, Application Insights is enabled. Application Insights is not necessary for this article, but it might help you understand and investigate any issues you encounter with the custom allocation. If you prefer, you can disable Application Insights by selecting the **Monitoring** tab and then selecting **No** for **Enable Application Insights**.

Home > New > Function App > Function App

## Function App

Looking for the classic Function App create experience? →

**Basics**   **Hosting**   **Monitoring**   **Tags**   **Review + create**

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

**Project Details**

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ Internal use

Resource Group \* ⓘ contoso-us-resource-group

Create new

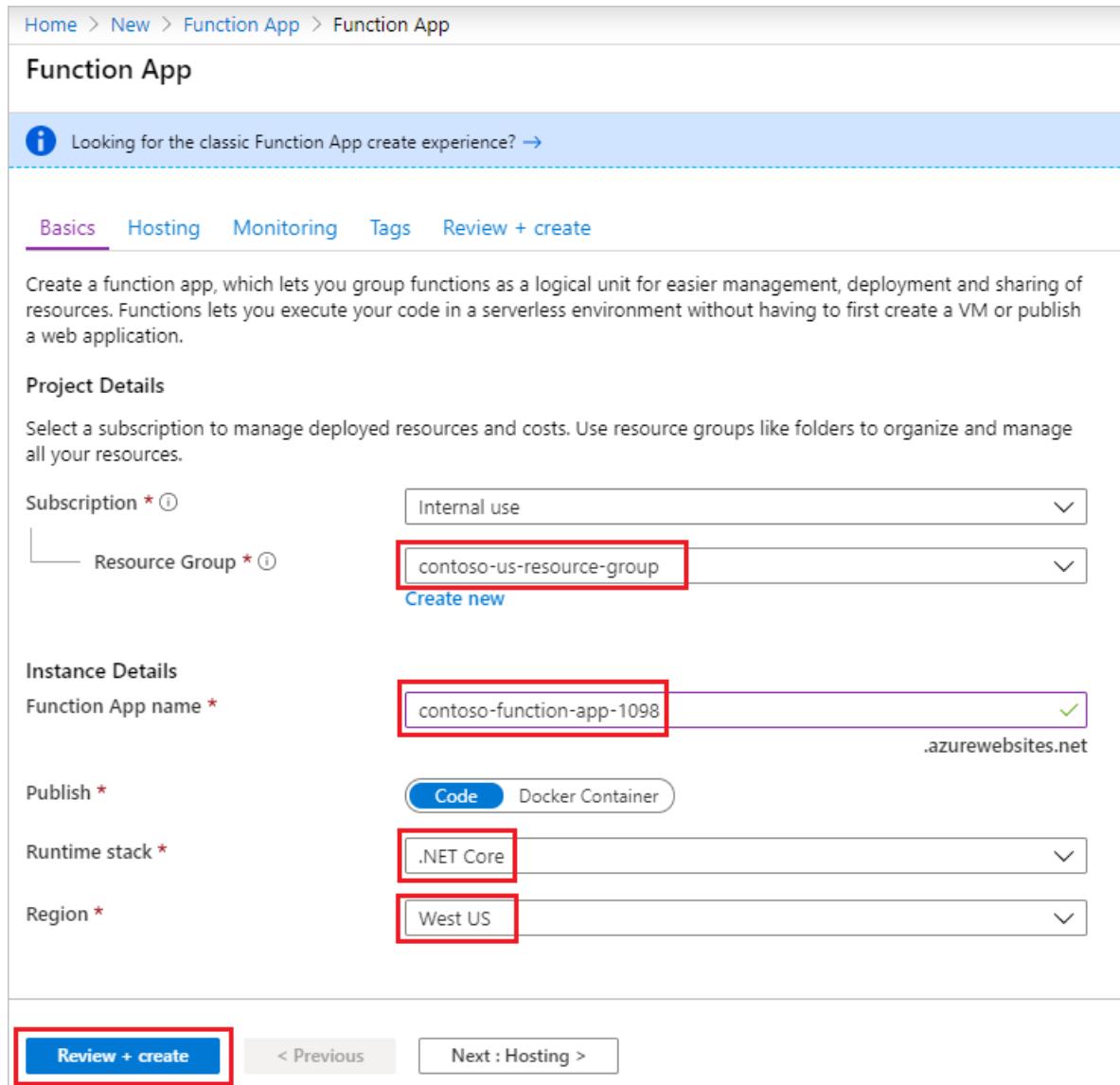
Function App name \* contoso-function-app-1098 .azurewebsites.net

Publish \* Code Docker Container

Runtime stack \* .NET Core

Region \* West US

**Review + create** < Previous Next : Hosting >



- On the **Summary** page, select **Create** to create the function app. Deployment may take several minutes. When it completes, select **Go to resource**.
- On the left pane of the function app **Overview** page, select **+** next to **Functions** to add a new function.

contoso-function-app-1098

Function Apps

Internal use

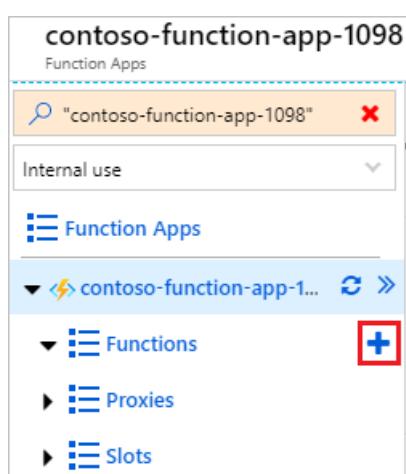
Function Apps

contoso-function-app-1098

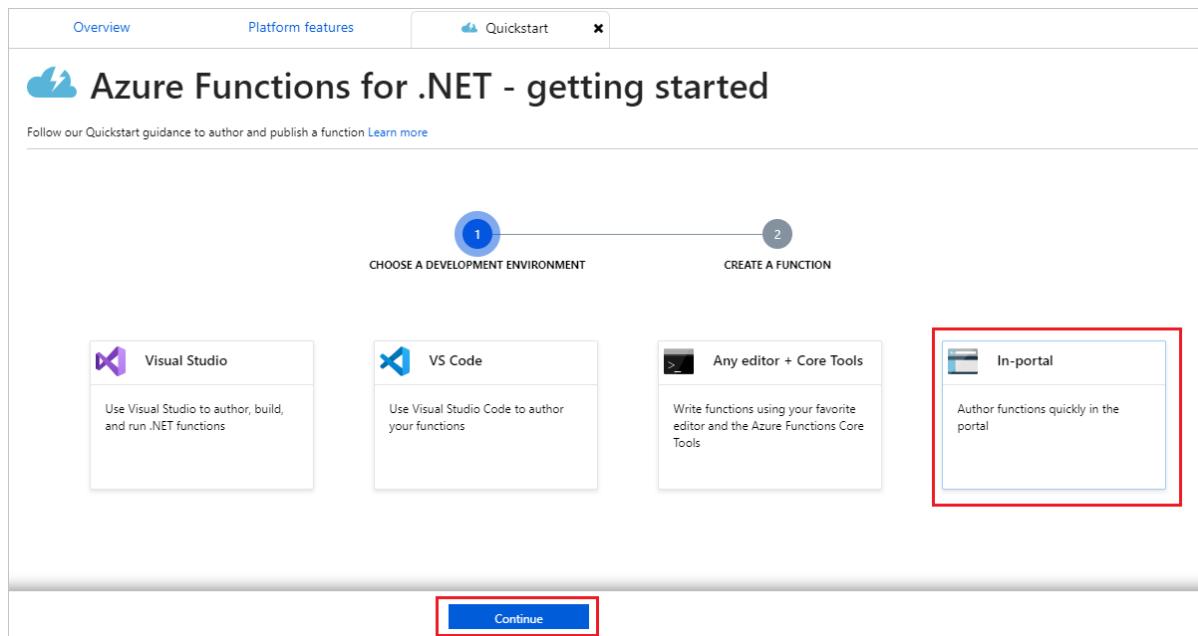
Functions

Proxies

Slots



6. On the Azure Functions for .NET - getting started page, for the CHOOSE A DEPLOYMENT ENVIRONMENT step, select the In-portal tile, then select Continue.



7. On the next page, for the CREATE A FUNCTION step, select the Webhook + API tile, then select Create. A function named **HttpTrigger1** is created, and the portal displays the contents of the `run.csx` code file.
8. Reference required Nuget packages. To create the initial device twin, the custom allocation function uses classes that are defined in two Nuget packages that must be loaded into the hosting environment. With Azure Functions, Nuget packages are referenced using a `function.host` file. In this step, you save and upload a `function.host` file.

- a. Copy the following lines into your favorite editor and save the file on your computer as `function.host`.

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>netstandard2.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
    <PackageReference Include="Microsoft.Azure.Devices.Provisioning.Service" Version="1.5.0" />
    <PackageReference Include="Microsoft.Azure.Devices.Shared" Version="1.16.0" />
  </ItemGroup>
</Project>
```

- b. On the **HttpTrigger1** function, expand the **View Files** tab on the right side of the window.

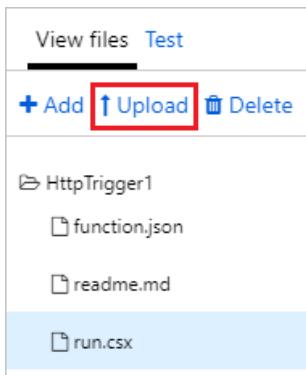


```

run.csx
Save ▶ Run </> Get function URL
1 #r "Newtonsoft.Json"
2
3 using System.Net;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task<IActionResult> Run(HttpContext req, ILogger log)
9 {
10     log.LogInformation("C# HTTP trigger function processed a request.");
11
12     string name = req.Query["name"];
13
14     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
15     dynamic data = JsonConvert.DeserializeObject(requestBody);
16     name = name ?? data?.name;
17
18     return name != null
19     ? (ActionResult)new OkObjectResult($"Hello, {name}")
20     : new BadRequestObjectResult("Please pass a name on the query string or in the request body");
21 }
22

```

- c. Select **Upload**, browse to the function.proj file, and select **Open** to upload the file.



9. Replace the code for the **HttpTrigger1** function with the following code and select **Save**:

```

#r "Newtonsoft.Json"

using System.Net;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

using Microsoft.Azure.Devices.Shared; // For TwinCollection
using Microsoft.Azure.Devices.Provisioning.Service; // For TwinState

public static async Task<IActionResult> Run(HttpContext req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");

    // Get request body
    string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
    dynamic data = JsonConvert.DeserializeObject(requestBody);

    log.LogInformation("Request.Body:....");
    log.LogInformation(requestBody);

    // Get registration ID of the device
    string regId = data?.deviceRuntimeContext?.registrationId;

    string message = "Uncaught error";
    bool fail = false;
    ResponseObj obj = new ResponseObj();

    if (regId == null)
    {
        message = "Registration ID not provided for the device.";
    }
}

```

```

        log.LogInformation("Registration ID : NULL");
        fail = true;
    }
    else
    {
        string[] hubs = data?.linkedHubs.ToObject<string[]>();

        // Must have hubs selected on the enrollment
        if (hubs == null)
        {
            message = "No hub group defined for the enrollment.";
            log.LogInformation("linkedHubs : NULL");
            fail = true;
        }
        else
        {
            // This is a Contoso Toaster Model 007
            if (regId.Contains("-contoso-tstrsd-007"))
            {
                //Find the "-toasters-" IoT hub configured on the enrollment
                foreach(string hubString in hubs)
                {
                    if (hubString.Contains("-toasters-"))
                        obj.iotHubHostName = hubString;
                }

                if (obj.iotHubHostName == null)
                {
                    message = "No toasters hub found for the enrollment.";
                    log.LogInformation(message);
                    fail = true;
                }
                else
                {
                    // Specify the initial tags for the device.
                    TwinCollection tags = new TwinCollection();
                    tags["deviceType"] = "toaster";

                    // Specify the initial desired properties for the device.
                    TwinCollection properties = new TwinCollection();
                    properties["state"] = "ready";
                    properties["darknessSetting"] = "medium";

                    // Add the initial twin state to the response.
                    TwinState twinState = new TwinState(tags, properties);
                    obj.initialTwin = twinState;
                }
            }
            // This is a Contoso Heat pump Model 008
            else if (regId.Contains("-contoso-hpsd-088"))
            {
                //Find the "-heatpumps-" IoT hub configured on the enrollment
                foreach(string hubString in hubs)
                {
                    if (hubString.Contains("-heatpumps-"))
                        obj.iotHubHostName = hubString;
                }

                if (obj.iotHubHostName == null)
                {
                    message = "No heat pumps hub found for the enrollment.";
                    log.LogInformation(message);
                    fail = true;
                }
                else
                {
                    // Specify the initial tags for the device.
                    TwinCollection tags = new TwinCollection();
                    tags["deviceType"] = "heatpump";
                }
            }
        }
    }
}

```

```

        // Specify the initial desired properties for the device.
        TwinCollection properties = new TwinCollection();
        properties["state"] = "on";
        properties["temperatureSetting"] = "65";

        // Add the initial twin state to the response.
        TwinState twinState = new TwinState(tags, properties);
        obj.initialTwin = twinState;
    }
}

// Unrecognized device.
else
{
    fail = true;
    message = "Unrecognized device registration.";
    log.LogInformation("Unknown device registration");
}
}

log.LogInformation("\nResponse");
log.LogInformation((obj.iothubHostName != null) ? JsonConvert.SerializeObject(obj) : message);

return (fail)
? new BadRequestObjectResult(message)
: (ActionResult)new OkObjectResult(obj);
}

public class ResponseObj
{
    public string iothubHostName {get; set;}
    public TwinState initialTwin {get; set;}
}

```

## Create the enrollment

In this section, you'll create a new enrollment group that uses the custom allocation policy. For simplicity, this article uses [Symmetric key attestation](#) with the enrollment. For a more secure solution, consider using [X.509 certificate attestation](#) with a chain of trust.

1. Still on the [Azure portal](#), open your provisioning service.
2. Select **Manage enrollments** on the left pane, and then select the **Add enrollment group** button at the top of the page.
3. On **Add Enrollment Group**, enter the following information, and select the **Save** button.

**Group name:** Enter **contoso-custom-allocated-devices**.

**Attestation Type:** Select **Symmetric Key**.

**Auto Generate Keys:** This checkbox should already be checked.

**Select how you want to assign devices to hubs:** Select **Custom (Use Azure Function)**.

 Save

Group name \*  
contoso-custom-allocated-devices

Attestation Type ⓘ  
 Certificate  Symmetric Key

Auto-generate keys ⓘ

Primary Key \* ⓘ  
Enter your primary key

Secondary Key \* ⓘ  
Enter your secondary key

IoT Edge device ⓘ  
 True  False

Select how you want to assign devices to hubs ⓘ  
Custom (Use Azure Function)

Select the IoT hubs this group can be assigned to: ⓘ  
0 selected

[Link a new IoT hub](#)

4. On Add Enrollment Group, select **Link a new IoT hub** to link both of your new divisional IoT hubs.

Execute this step for both of your divisional IoT hubs.

**Subscription:** If you have multiple subscriptions, choose the subscription where you created the divisional IoT hubs.

**IoT hub:** Select one of the divisional hubs you created.

**Access Policy:** Choose **iothubowner**.

**Add link to IoT hub**

Learn more about linking IoT hubs.

**Subscription \*** ⓘ  
Internal use

**IoT hub \***  
contoso-toasters-hub-1098

**Access Policy \*** ⓘ  
iothubowner

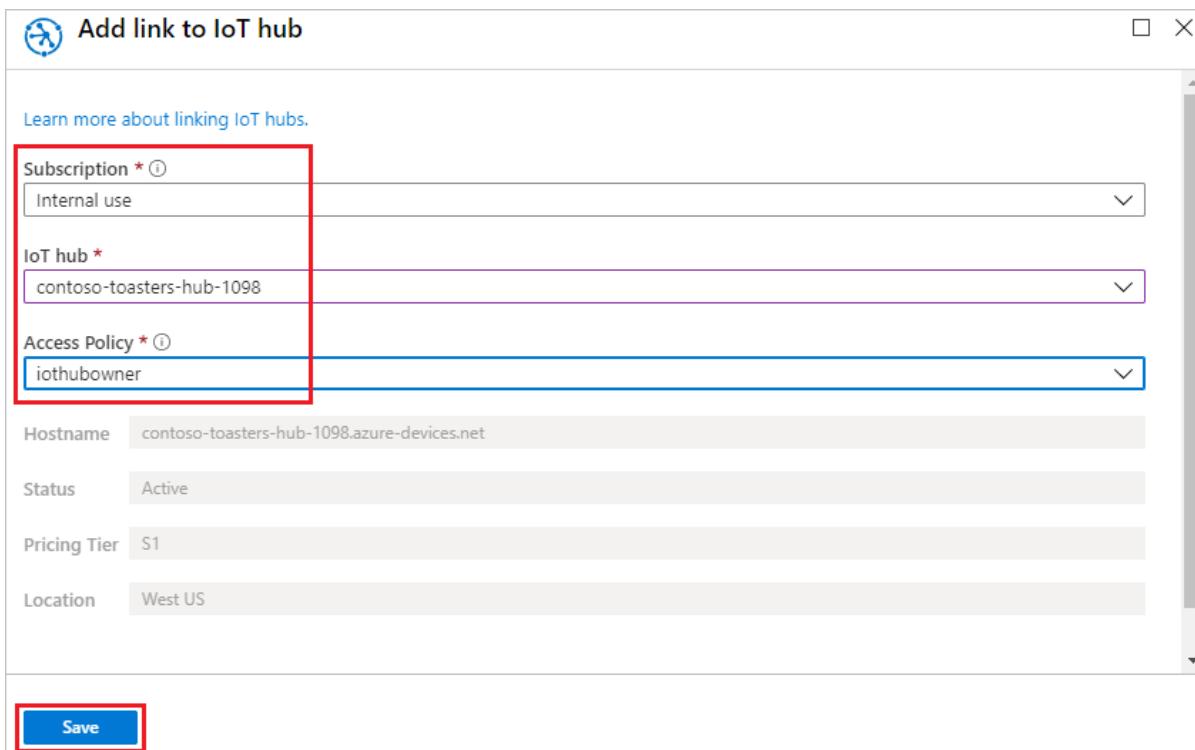
Hostname: contoso-toasters-hub-1098.azure-devices.net

Status: Active

Pricing Tier: S1

Location: West US

**Save**



5. On **Add Enrollment Group**, once both divisional IoT hubs have been linked, you must select them as the IoT Hub group for the enrollment group as shown below:

**Add Enrollment Group**

**Save**

**Group name \***  
contoso-custom-allocated-devices

**Attestation Type** ⓘ  
 Certificate  Symmetric Key

**Auto-generate keys** ⓘ

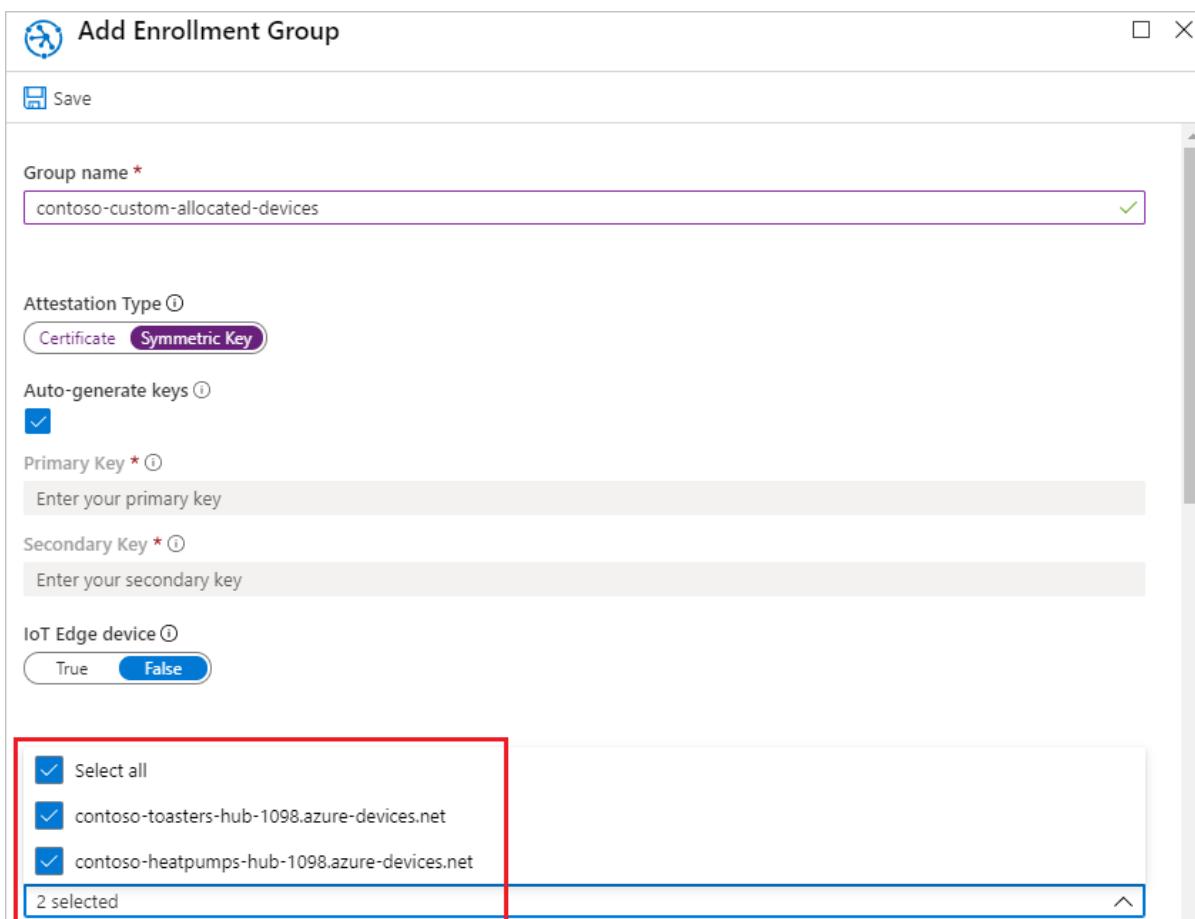
**Primary Key** \* ⓘ  
Enter your primary key

**Secondary Key** \* ⓘ  
Enter your secondary key

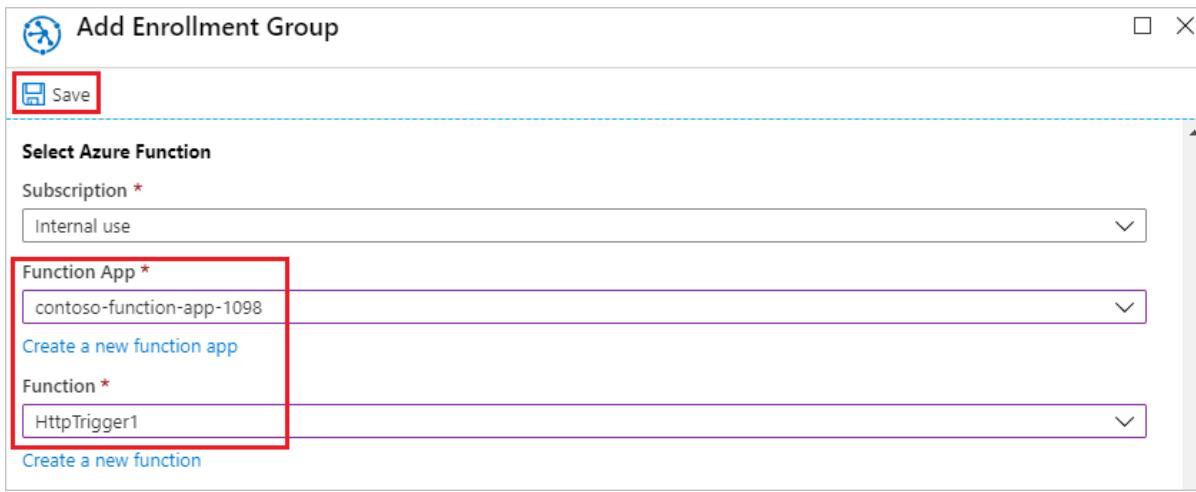
**IoT Edge device** ⓘ  
 True  False

Select all  
 contoso-toasters-hub-1098.azure-devices.net  
 contoso-heatpumps-hub-1098.azure-devices.net

2 selected



6. On **Add Enrollment Group**, scroll down to the **Select Azure Function** section, select the Function app you created in the previous section. Then select the function you created and select Save to save the enrollment group.



7. After saving the enrollment, reopen it and make a note of the **Primary Key**. You must save the enrollment first to have the keys generated. This key will be used to generate unique device keys for simulated devices later.

## Derive unique device keys

In this section, you create two unique device keys. One key will be used for a simulated toaster device. The other key will be used for a simulated heat pump device.

To generate the device key, you use the **Primary Key** you noted earlier to compute the [HMAC-SHA256](#) of the device registration ID for each device and convert the result into Base64 format. For more information on creating derived device keys with enrollment groups, see the group enrollments section of [Symmetric key attestation](#).

For the example in this article, use the following two device registration IDs and compute a device key for both devices. Both registration IDs have a valid suffix to work with the example code for the custom allocation policy:

- breakroom499-contoso-tstrsd-007
- mainbuilding167-contoso-hpsd-088

### Linux workstations

If you're using a Linux workstation, you can use openssl to generate your derived device keys as shown in the following example.

1. Replace the value of **KEY** with the **Primary Key** you noted earlier.

```
KEY=oIK770y7rBw8YB6IS6ukRChAw+Yq6GC61RMrPLSTI00tdI+XDu0LmLuNm11p+qv2I+adqGUdZHm46zXAQdZo0A==

REG_ID1=breakroom499-contoso-tstrsd-007
REG_ID2=mainbuilding167-contoso-hpsd-088

keybytes=$(echo $KEY | base64 --decode | xxd -p -u -c 1000)
devkey1=$(echo -n $REG_ID1 | openssl sha256 -mac HMAC -macopt hexkey:$keybytes -binary | base64)
devkey2=$(echo -n $REG_ID2 | openssl sha256 -mac HMAC -macopt hexkey:$keybytes -binary | base64)

echo -e "$\n$REG_ID1 : $devkey1$\n$REG_ID2 : $devkey2$\n"
```

```
breakroom499-contoso-tstrsd-007 : JC8F96eayuQwwz+PKE7IzjH2lIAjCUnAa61tDigBnSs=
mainbuilding167-contoso-hpsd-088 : 6uejA9PfkQgmYylj8Zerp3kcbeVrGZ172YLa7VSnJzg=
```

### Windows-based workstations

If you're using a Windows-based workstation, you can use PowerShell to generate your derived device key as shown in the following example.

1. Replace the value of **KEY** with the **Primary Key** you noted earlier.

```
$KEY='oiK770y7rBw8YB6IS6ukRChAw+Yq6GC61RMrPLSTi00tdI+XDu0LmLuNm11p+qv2I+adqGUdZHm46zXAQdZoOA=='  
  
$REG_ID1='breakroom499-contoso-tstrsd-007'  
$REG_ID2='mainbuilding167-contoso-hpsd-088'  
  
$hmacsha256 = New-Object System.Security.Cryptography.HMACSHA256  
$hmacsha256.key = [Convert]::FromBase64String($key)  
$sig1 = $hmacsha256.ComputeHash([Text.Encoding]::ASCII.GetBytes($REG_ID1))  
$sig2 = $hmacsha256.ComputeHash([Text.Encoding]::ASCII.GetBytes($REG_ID2))  
$derivedkey1 = [Convert]::ToBase64String($sig1)  
$derivedkey2 = [Convert]::ToBase64String($sig2)  
  
echo "`n`n$REG_ID1 : $derivedkey1`n$REG_ID2 : $derivedkey2`n`n"
```

```
breakroom499-contoso-tstrsd-007 : JC8F96eayuQwwz+PkE7IzjH2lIAjCUnAa61tDigBnSs=  
mainbuilding167-contoso-hpsd-088 : 6uejA9PfkQgmYylj8Zerp3kcbeVrGZ172YLa7VSnJzg=
```

The simulated devices will use the derived device keys with each registration ID to perform symmetric key attestation.

## Prepare an Azure IoT C SDK development environment

In this section, you prepare the development environment used to build the [Azure IoT C SDK](#). The SDK includes the sample code for the simulated device. This simulated device will attempt provisioning during the device's boot sequence.

This section is oriented toward a Windows-based workstation. For a Linux example, see the set-up of the VMs in [How to provision for multitenancy](#).

1. Download the [CMake build system](#).

It is important that the Visual Studio prerequisites (Visual Studio and the 'Desktop development with C++' workload) are installed on your machine, **before** starting the `cmake` installation. Once the prerequisites are in place, and the download is verified, install the CMake build system.

2. Find the tag name for the [latest release](#) of the SDK.
3. Open a command prompt or Git Bash shell. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git  
cd azure-iot-sdk-c  
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. Create a `cmake` subdirectory in the root directory of the git repository, and navigate to that folder. Run the following commands from the `azure-iot-sdk-c` directory:

```
mkdir cmake  
cd cmake
```

5. Run the following command, which builds a version of the SDK specific to your development client platform.

A Visual Studio solution for the simulated device will be generated in the `cmake` directory.

```
cmake -Dhsm_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
```

If `cmake` doesn't find your C++ compiler, you might get build errors while running the command. If that happens, try running the command in the [Visual Studio command prompt](#).

Once the build succeeds, the last few output lines will look similar to the following output:

```
$ cmake -Dhsm_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
-- Building for: Visual Studio 15 2017
-- Selecting Windows SDK version 10.0.16299.0 to target Windows 10.0.17134.
-- The C compiler identification is MSVC 19.12.25835.0
-- The CXX compiler identification is MSVC 19.12.25835.0

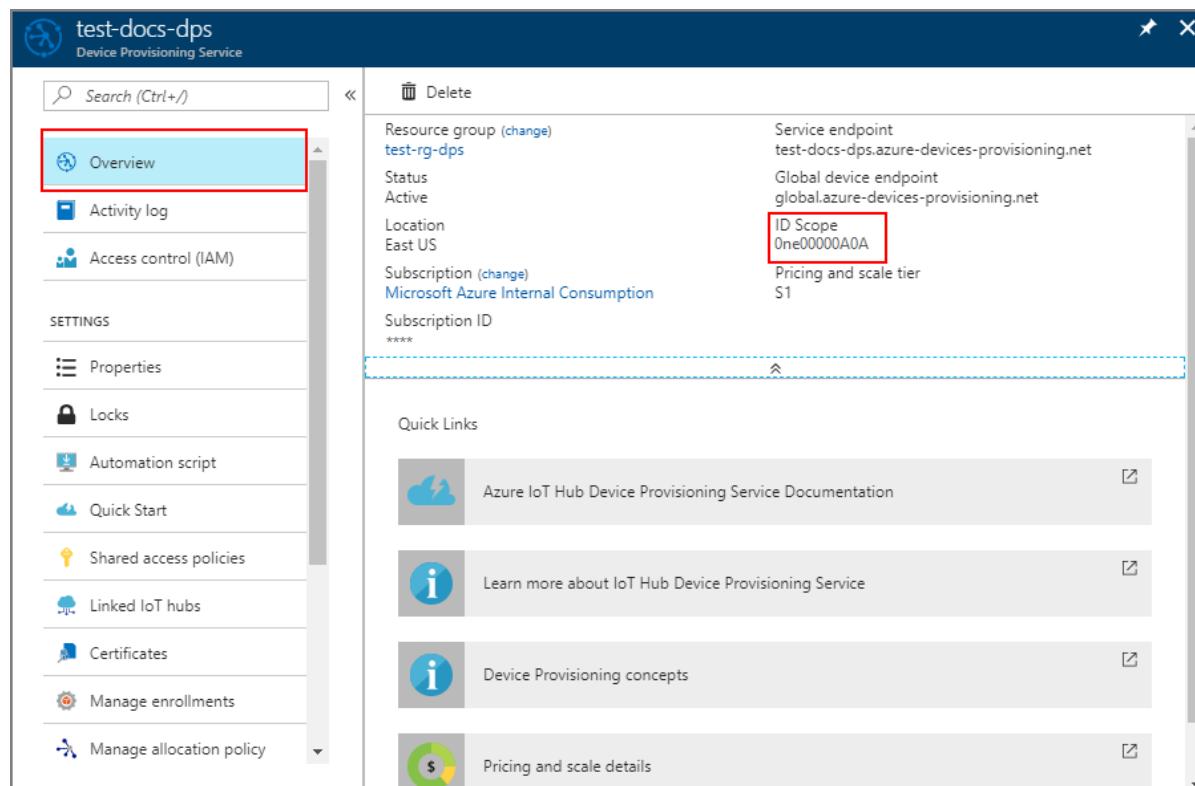
...
-- Configuring done
-- Generating done
-- Build files have been written to: E:/IoT Testing/azure-iot-sdk-c/cmake
```

## Simulate the devices

In this section, you update a provisioning sample named `prov_dev_client_sample` located in the Azure IoT C SDK you set up previously.

This sample code simulates a device boot sequence that sends the provisioning request to your Device Provisioning Service instance. The boot sequence will cause the toaster device to be recognized and assigned to the IoT hub using the custom allocation policy.

1. In the Azure portal, select the **Overview** tab for your Device Provisioning service and note down the **ID Scope** value.



2. In Visual Studio, open the `azure_iot_sdks.sln` solution file that was generated by running CMake earlier.

The solution file should be in the following location:

```
azure-iot-sdk-c\cmake\azure_iot_sdks.sln
```

3. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Samples** folder. Expand the sample project named **prov\_dev\_client\_sample**. Expand **Source Files**, and open **prov\_dev\_client\_sample.c**.
4. Find the `id_scope` constant, and replace the value with your **ID Scope** value that you copied earlier.

```
static const char* id_scope = "0ne00002193";
```

5. Find the definition for the `main()` function in the same file. Make sure the `hsm_type` variable is set to `SECURE_DEVICE_TYPE_SYMMETRIC_KEY` as shown below:

```
SECURE_DEVICE_TYPE hsm_type;
//hsm_type = SECURE_DEVICE_TYPE TPM;
//hsm_type = SECURE_DEVICE_TYPE_X509;
hsm_type = SECURE_DEVICE_TYPE_SYMMETRIC_KEY;
```

6. Right-click the **prov\_dev\_client\_sample** project and select **Set as Startup Project**.

### Simulate the Contoso toaster device

1. To simulate the toaster device, find the call to `prov_dev_set_symmetric_key_info()` in **prov\_dev\_client\_sample.c** which is commented out.

```
// Set the symmetric key if using they auth type
//prov_dev_set_symmetric_key_info("<symm_registration_id>", "<symmetric_Key>");
```

Uncomment the function call and replace the placeholder values (including the angle brackets) with the toaster registration ID and derived device key you generated previously. The key value `JC8F96eayuQwwz+PkE7IzjH2lIAjCUnAa61tDigBnSs=` shown below is only given as an example.

```
// Set the symmetric key if using they auth type
prov_dev_set_symmetric_key_info("breakroom499-contoso-tstrsd-007",
"JC8F96eayuQwwz+PkE7IzjH2lIAjCUnAa61tDigBnSs=');
```

Save the file.

2. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the prompt to rebuild the project, select **Yes**, to rebuild the project before running.

The following output is an example of the simulated toaster device successfully booting up and connecting to the provisioning service instance to be assigned to the toasters IoT hub by the custom allocation policy:

```
Provisioning API Version: 1.3.6
```

```
Registering Device
```

```
Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED  
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING  
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
```

```
Registration Information received from service: contoso-toasters-hub-1098.azure-devices.net, deviceId:  
breakroom499-contoso-tstrsd-007
```

```
Press enter key to exit:
```

## Simulate the Contoso heat pump device

1. To simulate the heat pump device, update the call to `prov_dev_set_symmetric_key_info()` in `prov_dev_client_sample.c` again with the heat pump registration ID and derived device key you generated earlier. The key value `6uejA9PfkQgmYylj8Zerp3kcbeVrGZ172YLa7VSnJzg=` shown below is also only given as an example.

```
// Set the symmetric key if using they auth type  
prov_dev_set_symmetric_key_info("mainbuilding167-contoso-hpsd-088",  
"6uejA9PfkQgmYylj8Zerp3kcbeVrGZ172YLa7VSnJzg=");
```

Save the file.

2. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the prompt to rebuild the project, select **Yes** to rebuild the project before running.

The following output is an example of the simulated heat pump device successfully booting up and connecting to the provisioning service instance to be assigned to the Contoso heat pumps IoT hub by the custom allocation policy:

```
Provisioning API Version: 1.3.6
```

```
Registering Device
```

```
Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED  
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING  
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
```

```
Registration Information received from service: contoso-heatpumps-hub-1098.azure-devices.net, deviceId:  
mainbuilding167-contoso-hpsd-088
```

```
Press enter key to exit:
```

## Troubleshooting custom allocation policies

The following table shows expected scenarios and the results error codes you might receive. Use this table to help troubleshoot custom allocation policy failures with your Azure Functions.

SCENARIO	REGISTRATION RESULT FROM PROVISIONING SERVICE	PROVISIONING SDK RESULTS
The webhook returns 200 OK with 'iotHubHostName' set to a valid IoT hub host name	Result status: Assigned	SDK returns PROV_DEVICE_RESULT_OK along with hub information

SCENARIO	REGISTRATION RESULT FROM PROVISIONING SERVICE	PROVISIONING SDK RESULTS
The webhook returns 200 OK with 'iotHubHostName' present in the response, but set to an empty string or null	Result status: Failed Error code: CustomAllocationIoTHubNotSpecified (400208)	SDK returns PROV_DEVICE_RESULT_HUB_NOT_SPECIFIED
The webhook returns 401 Unauthorized	Result status: Failed Error code: CustomAllocationUnauthorizedAccess (400209)	SDK returns PROV_DEVICE_RESULT_UNAUTHORIZED
An Individual Enrollment was created to disable the device	Result status: Disabled	SDK returns PROV_DEVICE_RESULT_DISABLED
The webhook returns error code >= 429	DPS' orchestration will retry a number of times. The retry policy is currently: <ul style="list-style-type: none"> <li>- Retry count: 10</li> <li>- Initial interval: 1s</li> <li>- Increment: 9s</li> </ul>	SDK will ignore error and submit another get status message in the specified time
The webhook returns any other status code	Result status: Failed Error code: CustomAllocationFailed (400207)	SDK returns PROV_DEVICE_RESULT_DEV_AUTH_ERROR

## Clean up resources

If you plan to continue working with the resources created in this article, you can leave them. If you don't plan to continue using the resources, use the following steps to delete all of the resources created in this article to avoid unnecessary charges.

The steps here assume you created all resources in this article as instructed in the same resource group named **contoso-us-resource-group**.

### IMPORTANT

Deleting a resource group is irreversible. The resource group and all the resources contained in it are permanently deleted. Make sure that you don't accidentally delete the wrong resource group or resources. If you created the IoT Hub inside an existing resource group that contains resources you want to keep, only delete the IoT Hub resource itself instead of deleting the resource group.

To delete the resource group by name:

1. Sign in to the [Azure portal](#) and select **Resource groups**.
2. In the **Filter by name...** textbox, type the name of the resource group containing your resources, **contoso-us-resource-group**.
3. To the right of your resource group in the result list, select ... then **Delete resource group**.
4. You'll be asked to confirm the deletion of the resource group. Type the name of your resource group again to confirm, and then select **Delete**. After a few moments, the resource group and all of its contained resources are deleted.

## Next steps

- To learn more Reprovisioning, see [IoT Hub Device reprovisioning concepts](#)
- To learn more Deprovisioning, see [How to deprovision devices that were previously autoprovisioned](#)

# How to provision legacy devices using symmetric keys

12/19/2019 • 9 minutes to read • [Edit Online](#)

A common problem with many legacy devices is that they often have an identity that is composed of a single piece of information. This identity information is usually a MAC address or a serial number. Legacy devices may not have a certificate, TPM, or any other security feature that can be used to securely identify the device. The Device Provisioning Service for IoT hub includes symmetric key attestation. Symmetric key attestation can be used to identify a device based off information like the MAC address or a serial number.

If you can easily install a [hardware security module \(HSM\)](#) and a certificate, then that may be a better approach for identifying and provisioning your devices. Since that approach may allow you to bypass updating the code deployed to all your devices, and you would not have a secret key embedded in your device image.

This article assumes that neither an HSM or a certificate is a viable option. However, it is assumed that you do have some method of updating device code to use the Device Provisioning Service to provision these devices.

This article also assumes that the device update takes place in a secure environment to prevent unauthorized access to the master group key or the derived device key.

This article is oriented toward a Windows-based workstation. However, you can perform the procedures on Linux. For a Linux example, see [How to provision for multitenancy](#).

## NOTE

The sample used in this article is written in C. There is also a [C# device provisioning symmetric key sample](#) available. To use this sample, download or clone the [azure-iot-samples-csharp](#) repository and follow the in-line instructions in the sample code. You can follow the instructions in this article to create a symmetric key enrollment group using the portal and to find the ID Scope and enrollment group primary and secondary keys needed to run the sample. You can also create individual enrollments using the sample.

## Overview

A unique registration ID will be defined for each device based on information that identifies that device. For example, the MAC address or a serial number.

An enrollment group that uses [symmetric key attestation](#) will be created with the Device Provisioning Service. The enrollment group will include a group master key. That master key will be used to hash each unique registration ID to produce a unique device key for each device. The device will use that derived device key with its unique registration ID to attest with the Device Provisioning Service and be assigned to an IoT hub.

The device code demonstrated in this article will follow the same pattern as the [Quickstart: Provision a simulated device with symmetric keys](#). The code will simulate a device using a sample from the [Azure IoT C SDK](#). The simulated device will attest with an enrollment group instead of an individual enrollment as demonstrated in the quickstart.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Completion of the [Set up IoT Hub Device Provisioning Service with the Azure portal](#) quickstart.

The following prerequisites are for a Windows development environment. For Linux or macOS, see the appropriate section in [Prepare your development environment](#) in the SDK documentation.

- [Visual Studio 2019](#) with the 'Desktop development with C++' workload enabled. Visual Studio 2015 and Visual Studio 2017 are also supported.
- Latest version of [Git](#) installed.

## Prepare an Azure IoT C SDK development environment

In this section, you will prepare a development environment used to build the [Azure IoT C SDK](#).

The SDK includes the sample code for the simulated device. This simulated device will attempt provisioning during the device's boot sequence.

1. Download the [CMake build system](#).

It is important that the Visual Studio prerequisites (Visual Studio and the 'Desktop development with C++' workload) are installed on your machine, **before** starting the `cmake` installation. Once the prerequisites are in place, and the download is verified, install the CMake build system.

2. Find the tag name for the [latest release](#) of the SDK.
3. Open a command prompt or Git Bash shell. Run the following commands to clone the latest release of the [Azure IoT C SDK](#) GitHub repository. Use the tag you found in the previous step as the value for the `-b` parameter:

```
git clone -b <release-tag> https://github.com/Azure/azure-iot-sdk-c.git
cd azure-iot-sdk-c
git submodule update --init
```

You should expect this operation to take several minutes to complete.

4. Create a `cmake` subdirectory in the root directory of the git repository, and navigate to that folder. Run the following commands from the `azure-iot-sdk-c` directory:

```
mkdir cmake
cd cmake
```

5. Run the following command, which builds a version of the SDK specific to your development client platform. A Visual Studio solution for the simulated device will be generated in the `cmake` directory.

```
cmake -Dhsm_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
```

If `cmake` does not find your C++ compiler, you might get build errors while running the above command. If that happens, try running this command in the [Visual Studio command prompt](#).

Once the build succeeds, the last few output lines will look similar to the following output:

```

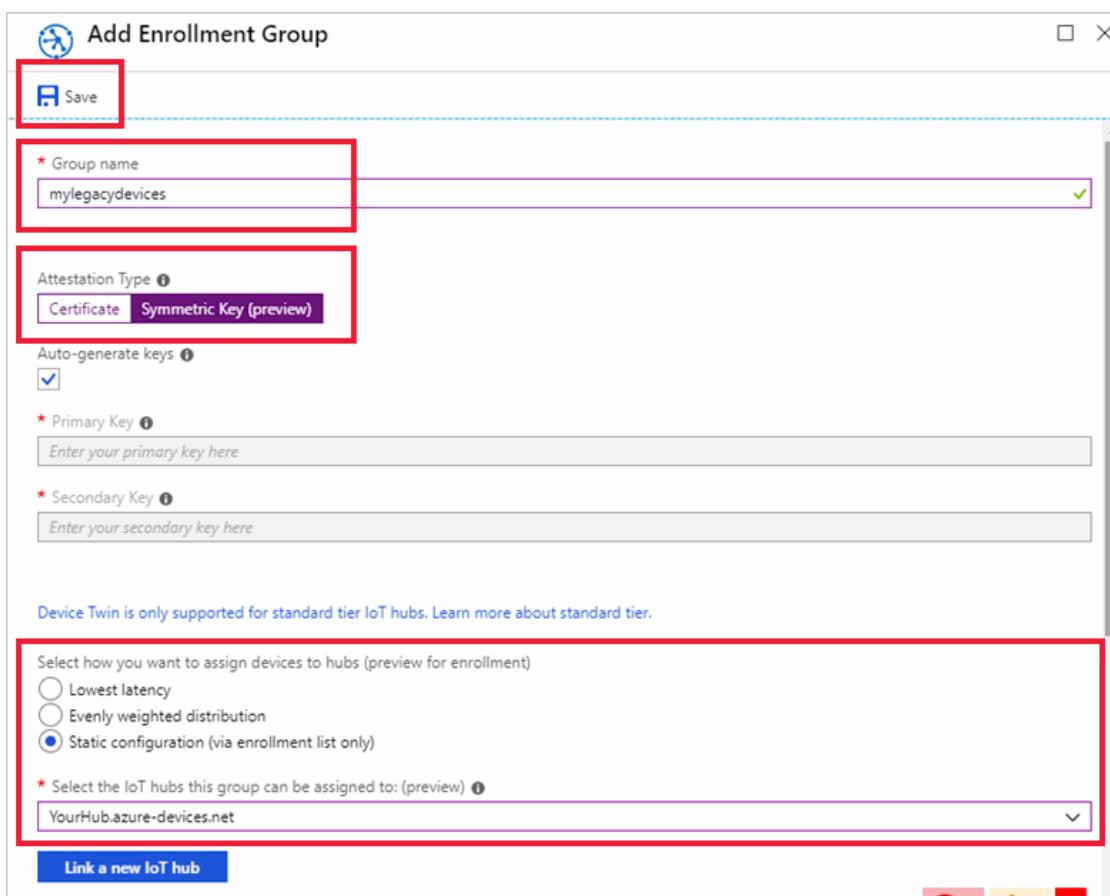
$ cmake -Dhsm_type_symm_key:BOOL=ON -Duse_prov_client:BOOL=ON ..
-- Building for: Visual Studio 15 2017
-- Selecting Windows SDK version 10.0.16299.0 to target Windows 10.0.17134.
-- The C compiler identification is MSVC 19.12.25835.0
-- The CXX compiler identification is MSVC 19.12.25835.0

...
-- Configuring done
-- Generating done
-- Build files have been written to: E:/IoT Testing/azure-iot-sdk-c/cmake

```

## Create a symmetric key enrollment group

1. Sign in to the [Azure portal](#), and open your Device Provisioning Service instance.
2. Select the **Manage enrollments** tab, and then click the **Add enrollment group** button at the top of the page.
3. On **Add Enrollment Group**, enter the following information, and click the **Save** button.
  - **Group name:** Enter **mylegacydevices**.
  - **Attestation Type:** Select **Symmetric Key**.
  - **Auto Generate Keys:** Check this box.
  - **Select how you want to assign devices to hubs:** Select **Static configuration** so you can assign to a specific hub.
  - **Select the IoT hubs this group can be assigned to:** Select one of your hubs.



4. Once you saved your enrollment, the **Primary Key** and **Secondary Key** will be generated and added to the enrollment entry. Your symmetric key enrollment group appears as **mylegacydevices** under the

*Group Name* column in the *Enrollment Groups* tab.

Open the enrollment and copy the value of your generated **Primary Key**. This key is your master group key.

## Choose a unique registration ID for the device

A unique registration ID must be defined to identify each device. You can use the MAC address, serial number, or any unique information from the device.

In this example, we use a combination of a MAC address and serial number forming the following string for a registration ID.

```
sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6
```

Create a unique registration ID for your device. Valid characters are lowercase alphanumeric and dash ('-').

## Derive a device key

To generate the device key, use the group master key to compute an [HMAC-SHA256](#) of the unique registration ID for the device and convert the result into Base64 format.

Do not include your group master key in your device code.

### Linux workstations

If you are using a Linux workstation, you can use openssl to generate your derived device key as shown in the following example.

Replace the value of **KEY** with the **Primary Key** you noted earlier.

Replace the value of **REG\_ID** with your registration ID.

```
KEY=8isrFI1sGsIlvvFSSFRiMfCNzv21fjbE/+ah/1Sh3lF8e2YG1Te7w1KpZhJFFXJrqYKi9yegxkqIChbqOS9Egw==  
REG_ID=sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6  
  
keybytes=$(echo $KEY | base64 --decode | xxd -p -u -c 1000)  
echo -n $REG_ID | openssl sha256 -mac HMAC -macopt hexkey:$keybytes -binary | base64
```

```
Jsm0lyGpjaVYVP2g3FnmmG9dI/9qU24wNoykUmermc=
```

### Windows-based workstations

If you are using a Windows-based workstation, you can use PowerShell to generate your derived device key as shown in the following example.

Replace the value of **KEY** with the **Primary Key** you noted earlier.

Replace the value of **REG\_ID** with your registration ID.

```
$KEY='8isrFI1sGsIlvvFSSFRiMfCNzv21fjbE/+ah/1Sh3lF8e2YG1Te7w1KpZhJFFXJrqYKi9yegxkqIChbqOS9Egw=='  
$REG_ID='sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6'  
  
$hmacsha256 = New-Object System.Security.Cryptography.HMACSHA256  
$hmacsha256.key = [Convert]::FromBase64String($KEY)  
$sig = $hmacsha256.ComputeHash([Text.Encoding]::ASCII.GetBytes($REG_ID))  
$derivedkey = [Convert]::ToBase64String($sig)  
echo "`n$derivedkey`n"
```

Jsm01yGpjaVYVP2g3FnmmG9dI/9qU24wNoykUmermc=

Your device will use the derived device key with your unique registration ID to perform symmetric key attestation with the enrollment group during provisioning.

## Create a device image to provision

In this section, you will update a provisioning sample named `prov_dev_client_sample` located in the Azure IoT C SDK you set up earlier.

This sample code simulates a device boot sequence that sends the provisioning request to your Device Provisioning Service instance. The boot sequence will cause the device to be recognized and assigned to the IoT hub you configured on the enrollment group.

1. In the Azure portal, select the **Overview** tab for your Device Provisioning service and note down the **ID Scope** value.

The screenshot shows the Azure Device Provisioning Service overview page. The left sidebar has a red box around the 'Overview' tab. The main content area shows the following details:

Resource group (change) test-rg-dps	Service endpoint test-docs-dps.azure-devices-provisioning.net
Status Active	Global device endpoint global.azure-devices-provisioning.net
Location East US	
Subscription (change) Microsoft Azure Internal Consumption	ID Scope One00000A0A
Subscription ID *****	Pricing and scale tier S1

Below this is a 'Quick Links' section with four items:

- Azure IoT Hub Device Provisioning Service Documentation
- Learn more about IoT Hub Device Provisioning Service
- Device Provisioning concepts
- Pricing and scale details

2. In Visual Studio, open the `azure_iot_sdks.sln` solution file that was generated by running CMake earlier. The solution file should be in the following location:

```
\azure-iot-sdk-c\cmake\azure_iot_sdks.sln
```

3. In Visual Studio's *Solution Explorer* window, navigate to the **Provision\_Samples** folder. Expand the sample project named `prov_dev_client_sample`. Expand **Source Files**, and open `prov_dev_client_sample.c`.
4. Find the `id_scope` constant, and replace the value with your **ID Scope** value that you copied earlier.

```
static const char* id_scope = "One00002193";
```

5. Find the definition for the `main()` function in the same file. Make sure the `hsm_type` variable is set to `SECURE_DEVICE_TYPE_SYMMETRIC_KEY` as shown below:

```
SECURE_DEVICE_TYPE hsm_type;
//hsm_type = SECURE_DEVICE_TYPE_TPM;
//hsm_type = SECURE_DEVICE_TYPE_X509;
hsm_type = SECURE_DEVICE_TYPE_SYMMETRIC_KEY;
```

6. Find the call to `prov_dev_set_symmetric_key_info()` in `prov_dev_client_sample.c` which is commented out.

```
// Set the symmetric key if using they auth type
//prov_dev_set_symmetric_key_info("<symm_registration_id>", "<symmetric_Key>");
```

Uncomment the function call, and replace the placeholder values (including the angle brackets) with the unique registration ID for your device and the derived device key you generated.

```
// Set the symmetric key if using they auth type
prov_dev_set_symmetric_key_info("sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6",
"Jsm0lyGpjVVVP2g3FnmnmG9dI/9qU24wNoykUmermc=");
```

Save the file.

7. Right-click the `prov_dev_client_sample` project and select **Set as Startup Project**.
8. On the Visual Studio menu, select **Debug > Start without debugging** to run the solution. In the prompt to rebuild the project, click **Yes**, to rebuild the project before running.

The following output is an example of the simulated device successfully booting up, and connecting to the provisioning Service instance to be assigned to an IoT hub:

```
Provisioning API Version: 1.2.8

Registering Device

Provisioning Status: PROV_DEVICE_REG_STATUS_CONNECTED
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING
Provisioning Status: PROV_DEVICE_REG_STATUS_ASSIGNING

Registration Information received from service:
test-docs-hub.azure-devices.net, deviceId: sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6

Press enter key to exit:
```

9. In the portal, navigate to the IoT hub your simulated device was assigned to and click the **IoT Devices** tab. On successful provisioning of the simulated to the hub, its device ID appears on the **IoT Devices** blade, with **STATUS** as **enabled**. You might need to click the **Refresh** button at the top.

The screenshot shows the Azure IoT Hub interface for the 'test-docs-hub' resource. The left sidebar contains a navigation menu with various options like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Explorers, and IoT devices. The 'IoT devices' option is selected and highlighted with a red box. The main content area has a search bar at the top. Below it is a message: 'You can use this tool to view, create, update, and delete devices on your IoT Hub.' There is a 'Query' section with a query editor containing a sample SQL query: 'SELECT \* FROM devices WHERE optional (e.g. tags.location='US')'. A blue 'Execute' button is below the query editor. At the bottom is a table with columns: DEVICE ID, STATUS, and AUTHENTICATION TYPE. One row in the table is highlighted with a red box, showing the device ID 'sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6', status 'Enabled', and authentication type 'Sas'.

DEVICE ID	STATUS	AUTHENTICATION TYPE
sn-007-888-abc-mac-a1-b2-c3-d4-e5-f6	Enabled	Sas

## Security concerns

Be aware that this leaves the derived device key included as part of the image, which is not a recommended security best practice. This is one reason why security and ease-of-use are tradeoffs.

## Next steps

- To learn more Reprovisioning, see [IoT Hub Device reprovisioning concepts](#)
- [Quickstart: Provision a simulated device with symmetric keys](#)
- To learn more Deprovisioning, see [How to deprovision devices that were previously auto-provisioned](#)

# How to use tools provided in the SDKs to simplify development for provisioning

12/10/2019 • 3 minutes to read • [Edit Online](#)

The IoT Hub Device Provisioning Service simplifies the provisioning process with zero-touch, just-in-time [auto-provisioning](#) in a secure and scalable manner. Security attestation in the form of X.509 certificate or Trusted Platform Module (TPM) is required. Microsoft is also partnering with [other security hardware partners](#) to improve confidence in securing IoT deployment. Understanding the hardware security requirement can be quite challenging for developers. A set of Azure IoT Provisioning Service SDKs are provided so that developers can use a convenience layer for writing clients that talk to the provisioning service. The SDKs also provide samples for common scenarios as well as a set of tools to simplify security attestation in development.

## Trusted Platform Module (TPM) simulator

[TPM](#) can refer to a standard for securely storing keys to authenticate the platform, or it can refer to the I/O interface used to interact with the modules implementing the standard. TPMs can exist as discrete hardware, integrated hardware, firmware-based, or software-based. In production, TPM is located on the device, either as discrete hardware, integrated hardware, or firmware-based. In the testing phase, a software-based TPM simulator is provided to developers. This simulator is only available for developing on Windows platform for now.

Steps for using the TPM simulator are:

1. [Prepare the development environment](#) and clone the GitHub repository:

```
git clone https://github.com/Azure/azure-iot-sdk-java.git
```

2. Navigate to the TPM simulator folder under `azure-iot-sdk-java/provisioning/provisioning-tool/tpm-simulator/`.
3. Run `Simulator.exe` prior to running any client application for provisioning device.
4. Let the simulator run in the background throughout the provisioning process to obtain registration ID and Endorsement Key. Both values are only valid for one instance of the run.

## X.509 certificate generator

[X.509 certificates](#) can be used as an attestation mechanism to scale production and simplify device provisioning. There are [several ways](#) to obtain an X.509 certificate:

- For production environment, we recommend purchasing an X.509 CA certificate from a public root certificate authority.
- For testing environment, you can generate an X.509 root certificate or X.509 certificate chain using:
  - OpenSSL: You can use scripts for certificate generation:
    - [Node.js](#)
    - [PowerShell or Bash](#)
  - Device Identity Composition Engine (DICE) Emulator: DICE can be used for cryptographic device identity and attestation based on TLS protocol and X.509 client certificates. [Learn](#) more about device identity with DICE.

### Using X.509 certificate generator with DICE emulator

The SDKs provide an X.509 certificate generator with DICE emulator, located in the [Java SDK](#). This generator works

cross-platform. The generated certificate can be used for development in other languages.

Currently, while the DICE Emulator outputs a root certificate, an intermediate certificate, a leaf certificate, and associated private key. However, the root certificate or intermediate certificate cannot be used to sign a separate leaf certificate. If you intend to test group enrollment scenario where one signing certificate is used to sign the leaf certificates of multiple devices, you can use OpenSSL to produce a chain of certificates.

To generate X.509 certificate using this generator:

1. [Prepare the development environment](#) and clone the GitHub repository:

```
git clone https://github.com/Azure/azure-iot-sdk-java.git
```

2. Change the root to `azure-iot-sdk-java`.
3. Run `mvn install -DskipTests=true` to download all required packages and compile the SDK
4. Navigate to the root for X.509 Certificate Generator in `azure-iot-sdk-java/provisioning/provisioning-tools/provisioning-x509-cert-generator`.
5. Build with `mvn clean install`
6. Run the tool using the following commands:

```
cd target  
java -jar ./provisioning-x509-cert-generator-{version}-with-deps.jar
```

7. When prompted, you may optionally enter a *Common Name* for your certificates.
8. The tool locally generates a **Client Cert**, the **Client Cert Private Key**, **Intermediate Cert**, and the **Root Cert**.

**Client Cert** is the leaf certificate on the device. **Client Cert** and the associated **Client Cert Private Key** are needed in device client. Depending on what language you choose, the mechanism to put this in the client application may be different. For more information, see the [Quickstarts](#) on create simulated device using X.509 for more information.

The root certificate or intermediate can be used to create an enrollment group or individual enrollment [programmatically](#) or using the [portal](#).

## Next steps

- Develop using the [Azure IoT SDK](#) for Azure IoT Hub and Azure IoT Hub Device Provisioning Service

# How to transfer a payload between device and DPS

3/25/2020 • 2 minutes to read • [Edit Online](#)

Sometimes DPS needs more data from devices to properly provision them to the right IoT Hub, and that data needs to be provided by the device. Vice versa, DPS can return data to the device to facilitate client side logics.

## When to use it

This feature can be used as an enhancement for [custom allocation](#). For instance, you want to allocate your devices based on the device model without human intervention. In this case, you will use [custom allocation](#). You can configure the device to report the model information as part of the [register device call](#). DPS will pass the device's payload into to the custom allocation webhook. And your function can decide which IoT Hub this device will go to when it receives device model information. Similarly, if the webhook wishes to return some data to the device, it will pass the data back as a string in the webhook response.

## Device sends data payload to DPS

When your device is sending a [register device call](#) to DPS, The register call can be enhanced to take other fields in the body. The body looks like the following:

```
{  
    "registrationId": "mydevice",  
    "tpm":  
    {  
        "endorsementKey": "stuff",  
        "storageRootKey": "things"  
    },  
    "payload": "your additional data goes here. It can be nested JSON."  
}
```

## DPS returns data to the device

If the custom allocation policy webhook wishes to return some data to the device, it will pass the data back as a string in the webhook response. The change is in the payload section below.

```
{  
    "iotHubHostName": "sample-iot-hub-1.azure-devices.net",  
    "initialTwin": {  
        "tags": {  
            "tag1": true  
        },  
        "properties": {  
            "desired": {  
                "tag2": true  
            }  
        }  
    },  
    "payload": "whatever is returned by the webhook"  
}
```

## SDK support

This feature is available in C, C#, JAVA and Node.js [client SDKs](#).

## Next steps

- Develop using the [Azure IoT SDK](#) for Azure IoT Hub and Azure IoT Hub Device Provisioning Service

# Troubleshooting with Azure IoT Hub Device Provisioning Service

1/3/2020 • 2 minutes to read • [Edit Online](#)

Connectivity issues for IoT devices can be difficult to troubleshoot because there are many possible points of failures such as attestation failures, registration failures etc. This article provides guidance on how to detect and troubleshoot device connectivity issues via [Azure Monitor](#).

## Using Azure Monitor to view metrics and set up alerts

The following procedure describes how to view and set up alert on IoT Hub Device Provisioning Service metric.

1. Sign in to the [Azure portal](#).
2. Browse to your IoT Hub Device Provisioning Service.
3. Select **Metrics**.
4. Select the desired metric.

Currently there are three metrics for DPS:

METRIC NAME	DESCRIPTION
Attestation attempts	Number of devices that attempted to authenticate with Device Provisioning Service
Registration attempts	Number of devices that attempted to register to IoT Hub after successful authentication
Device assigned	Number of devices that successfully assigned to IoT Hub

5. Select desired aggregation method to create a visual view of the metric.
6. To set up an alert of a metric, select **New alert rules** from the top right of the metric blade, similarly you can go to **Alert** blade and select **New alert rules**.
7. Select **Add condition**, then select the desired metric and threshold by following prompts.

To learn more, see [What are classic alerts in Microsoft Azure?](#)

## Using Log Analytic to view and resolve errors

1. Sign in to the [Azure portal](#).
2. Browse to your IoT hub.
3. Select **Diagnostics settings**.
4. Select **Turn on diagnostics**.
5. Enable the desired logs to be collected.

LOG NAME	DESCRIPTION
DeviceOperations	Logs related to device connection events
ServiceOperations	Event logs related to using service SDK (e.g. Creating or updating enrollment groups)

6. Turn on **Send to Log Analytics** ([see pricing](#)).
7. Go to **Logs** tab in the Azure portal under Device Provisioning Service resource.
8. Click **Run** to view recent events.
9. If there are results, look for `OperationName`, `ResultType`, `ResultSignature`, and `ResultDescription` (error message) to get more detail on the error.

## Common error codes

Use this table to understand and resolve common errors.

ERROR CODE	DESCRIPTION	HTTP STATUS CODE
400	The body of the request is not valid; for example, it cannot be parsed, or the object cannot be validated.	400 Bad format
401	The authorization token cannot be validated; for example, it is expired or does not apply to the request's URI. This error code is also returned to devices as part of the TPM attestation flow.	401 Unauthorized
404	The Device Provisioning Service instance, or a resource (e.g. an enrollment) does not exist.	404 Not Found
412	The ETag in the request does not match the ETag of the existing resource, as per RFC7232.	412 Precondition failed
429	Operations are being throttled by the service. For specific service limits, see <a href="#">IoT Hub Device Provisioning Service limits</a> .	429 Too many requests
500	An internal error occurred.	500 Internal Server Error

# Communicate with your DPS using the MQTT protocol

4/20/2020 • 3 minutes to read • [Edit Online](#)

DPS enables devices to communicate with the DPS device endpoint using:

- [MQTT v3.1.1](#) on port 8883
- [MQTT v3.1.1](#) over WebSocket on port 443.

DPS is not a full-featured MQTT broker and does not support all the behaviors specified in the MQTT v3.1.1 standard. This article describes how devices can use supported MQTT behaviors to communicate with DPS.

All device communication with DPS must be secured using TLS/SSL. Therefore, DPS doesn't support non-secure connections over port 1883.

## NOTE

DPS does not currently support devices using TPM [attestation mechanism](#) over the MQTT protocol.

## Connecting to DPS

A device can use the MQTT protocol to connect to a DPS using any of the following options.

- Libraries in the [Azure IoT Provisioning SDKs](#).
- The MQTT protocol directly.

## Using the MQTT protocol directly (as a device)

If a device cannot use the device SDKs, it can still connect to the public device endpoints using the MQTT protocol on port 8883. In the CONNECT packet, the device should use the following values:

- For the **ClientId** field, use **registrationId**.
- For the **Username** field, use `{idScope}/registrations/{registration_id}/api-version=2019-03-31`, where `{idScope}` is the [idScope](#) of the DPS.
- For the **Password** field, use a SAS token. The format of the SAS token is the same as for both the HTTPS and AMQP protocols:

`SharedAccessSignature sr={URL-encoded-resourceURI}&sig={signature-string}&se={expiry}&skn=registration`

The resourceURI should be in the format `{idScope}/registrations/{registration_id}`. The policy name should be `registration`.

## NOTE

If you use X.509 certificate authentication, SAS token passwords are not required.

For more information about how to generate SAS tokens, see the security tokens section of [Control access to DPS](#).

The following is a list of DPS implementation-specific behaviors:

- DPS does not support the functionality of **CleanSession** flag being set to 0.
- When a device app subscribes to a topic with **QoS 2**, DPS grants maximum QoS level 1 in the **SUBACK** packet. After that, DPS delivers messages to the device using QoS 1.

## TLS/SSL configuration

To use the MQTT protocol directly, your client *must* connect over TLS 1.2. Attempts to skip this step fail with connection errors.

## Registering a device

To register a device through DPS, a device should subscribe using `$dps/registrations/res/#` as a **Topic Filter**. The multi-level wildcard `#` in the Topic Filter is used only to allow the device to receive additional properties in the topic name. DPS does not allow the usage of the `#` or `?` wildcards for filtering of subtopics. Since DPS is not a general-purpose pub-sub messaging broker, it only supports the documented topic names and topic filters.

The device should publish a register message to DPS using

`$dps/registrations/PUT/iotdps-register/?$rid={request_id}` as a **Topic Name**. The payload should contain the [Device Registration](#) object in JSON format. In a successful scenario, the device will receive a response on the `$dps/registrations/res/202/?$rid={request_id}&retry-after=x` topic name where x is the retry-after value in seconds. The payload of the response will contain the [RegistrationOperationStatus](#) object in JSON format.

## Polling for registration operation status

The device must poll the service periodically to receive the result of the device registration operation. Assuming that the device has already subscribed to the `$dps/registrations/res/#` topic as indicated above, it can publish a get operationstatus message to the

`$dps/registrations/GET/iotdps-get-operationstatus/?$rid={request_id}&operationId={operationId}` topic name. The operation ID in this message should be the value received in the [RegistrationOperationStatus](#) response message in the previous step. In the successful case, the service will respond on the `$dps/registrations/res/200/?$rid={request_id}` topic. The payload of the response will contain the [RegistrationOperationStatus](#) object. The device should keep polling the service if the response code is 202 after a delay equal to the retry-after period. The device registration operation is successful if the service returns a 200 status code.

## Connecting over Websocket

When connecting over Websocket, specify the subprotocol as `mqtt`. Follow [RFC 6455](#).

## Next steps

To learn more about the MQTT protocol, see the [MQTT documentation](#).

To further explore the capabilities of DPS, see:

- [About IoT DPS](#)

# Summary of customer data request features

11/8/2019 • 2 minutes to read • [Edit Online](#)

The Azure IoT Hub Device Provisioning Service is a REST API-based cloud service targeted at enterprise customers that enables seamless, automated zero-touch provisioning of devices to Azure IoT Hub with security that begins at the device and ends with the cloud.

## NOTE

This article provides steps for how to delete personal data from the device or service and can be used to support your obligations under the GDPR. If you're looking for general info about GDPR, see the [GDPR section of the Service Trust portal](#).

Individual devices are assigned a registration ID and device ID by a tenant administrator. Data from and about these devices is based on these IDs. Microsoft maintains no information and has no access to data that would allow correlation of these devices to an individual.

Many of the devices managed in Device Provisioning Service are not personal devices, for example an office thermostat or factory robot. Customers may, however, consider some devices to be personally identifiable and at their discretion may maintain their own asset or inventory tracking methods that tie devices to individuals. Device Provisioning Service manages and stores all data associated with devices as if it were personal data.

Tenant administrators can use either the Azure portal or the service's REST APIs to fulfill information requests by exporting or deleting data associated with a device ID or registration ID.

## NOTE

Devices that have been provisioned in Azure IoT Hub through Device Provisioning Service have additional data stored in the Azure IoT Hub service. See the [Azure IoT Hub reference documentation](#) in order to complete a full request for a given device.

## Deleting customer data

Device Provisioning Service stores enrollments and registration records. Enrollments contain information about devices that are allowed to be provisioned, and registration records show which devices have already gone through the provisioning process.

Tenant administrators may remove enrollments from the Azure portal, and this removes any associated registration records as well.

For more information, see [How to manage device enrollments](#).

It is also possible to perform delete operations for enrollments and registration records using REST APIs:

- To delete enrollment information for a single device, you can use [Device Enrollment - Delete](#).
- To delete enrollment information for a group of devices, you can use [Device Enrollment Group - Delete](#).
- To delete information about devices that have been provisioned, you can use [Registration State - Delete Registration State](#).

## Exporting customer data

Device Provisioning Service stores enrollments and registration records. Enrollments contain information about devices that are allowed to be provisioned, and registration records show which devices have already gone through

the provisioning process.

Tenant administrators can view enrollments and registration records through the Azure portal and export them using copy and paste.

For more information on how to manage enrollments, see [How to manage device enrollments](#).

It is also possible to perform export operations for enrollments and registration records using REST APIs:

- To export enrollment information for a single device, you can use [Device Enrollment - Get](#).
- To export enrollment information for a group of devices, you can use [Device Enrollment Group - Get](#).
- To export information about devices that have already been provisioned, you can use [Registration State - Get Registration State](#).

**NOTE**

When you use Microsoft's enterprise services, Microsoft generates some information, known as system-generated logs. Some Device Provisioning Service system-generated logs are not accessible or exportable by tenant administrators. These logs constitute factual actions conducted within the service and diagnostic data related to individual devices.

## Links to additional documentation

Full documentation for Device Provisioning Service APIs is located at <https://docs.microsoft.com/rest/api/iot-dps>.

Azure IoT Hub [customer data request features](#).