Contents

What is Azure IoT Central?

5/19/2020 • 7 minutes to read • Edit Online

IoT Central is an IoT application platform that reduces the burden and cost of developing, managing, and maintaining enterprise-grade IoT solutions. Choosing to build with IoT Central gives you the opportunity to focus time, money, and energy on transforming your business with IoT data, rather than just maintaining and updating a complex and continually evolving IoT infrastructure.

The web UI lets you monitor device conditions, create rules, and manage millions of devices and their data throughout their life cycle. Furthermore, it enables you to act on device insights by extending IoT intelligence into line-of-business applications.

This article outlines, for IoT Central:

- The typical personas associated with a project.
- How to create your application.
- How to connect your devices to your application
- How to manage your application.
- Azure IoT Edge capabilities in IoT Central.
- How to connect your Azure IoT Edge runtime powered devices to your application.

Personas

The IoT Central documentation refers to four personas who interact with an IoT Central application:

- A *solution builder* is responsible for defining the types of devices that connect to the application and customizing the application for the operator.
- An *operator* manages the devices connected to the application.
- An *administrator* is responsible for administrative tasks such as managing user roles and permissions within the application.
- A *device developer* creates the code that runs on a device or IoT Edge module connected to your application.

Create your IoT Central application

As a solution builder, you use IoT Central to create a custom, cloud-hosted IoT solution for your organization. A custom IoT solution typically consists of:

- A cloud-based application that receives telemetry from your devices and enables you to manage those devices.
- Multiple devices running custom code connected to your cloud-based application.

You can quickly deploy a new IoT Central application and then customize it to your specific requirements in your browser. You can start with a generic *application template* or with one of the industry-focused application templates for Retail, Energy, Government, or Healthcare.

As a solution builder, you use the web-based tools to create a *device template* for the devices that connect to your application. A device template is the blueprint that defines the characteristics and behavior of a type of device such as the:

- Telemetry it sends.
- Business properties that an operator can modify.
- Device properties that are set by a device and are read-only in the application.

- Properties, that an operator sets, that determine the behavior of the device.

This device template includes:

- A *device capability model* that describes the capabilities a device should implement such as the telemetry it sends and the properties it reports.
- Cloud properties that aren't stored on the device.
- Customizations, dashboards, and forms that are part of your IoT Central application.

Create device templates

IoT Plug and Play (preview) enables IoT Central to integrate devices without you writing any embedded device code. At the core of IoT Plug and Play (preview), is a device capability model schema that describes device capabilities. In an IoT Central application, device templates use these IoT Plug and Play (preview) device capability models.

As a solution builder, you have several options for creating device templates:

- Import a device capability model from the Azure Certified for IoT device catalog and then add any cloud properties, customizations, and dashboards your IoT Central application needs.
- Design the device template in IoT Central and then implement its device capability model in your device code.
- Create a device capability model using Visual Studio code and publish the model to a repository. Implement your device code from the model, and connect your device to your IoT Central application. IoT Central finds the device capability model from the repository and creates a simple device template for you.
- Create a device capability model using Visual Studio code. Implement your device code from the model. Manually import the device capability model into your IoT Central application and then add any cloud properties, customizations, and dashboards your IoT Central application needs.

As a solution builder, you can use IoT Central to generate code for test devices to validate your device templates.

If you're a device developer, see IoT Central device development overview for an introduction to implementing devices that use these device templates.

Customize the UI

As a solution builder, you can also customize the IoT Central application UI for the operators who are responsible for the day-to-day use of the application. Customizations that a solution builder can make include:

- Defining the layout of properties and settings on a device template.
- Configuring custom dashboards to help operators discover insights and resolve issues faster.
- Configuring custom analytics to explore time series data from your connected devices.

Manage your devices

As an operator, you use the IoT Central application to manage the devices in your IoT Central solution. Operators do tasks such as:

- Monitoring the devices connected to the application.
- Troubleshooting and remediating issues with devices.
- Provisioning new devices.

As a solution builder, you can define custom rules and actions that operate over data streaming from connected devices. An operator can enable or disable these rules at the device level to control and automate tasks within the application.

With any IoT solution designed to operate at scale, a structured approach to device management is important. It's not enough just to connect your devices to the cloud, you need to keep your devices connected and healthy. An operator can use the following IoT Central capabilities to manage your devices throughout the application life cycle:

Dashboards

Built-in dashboards provide a customizable UI to monitor device health and telemetry. Start with a pre-built dashboard in an application template or create your own dashboards tailored to the needs of your operators. You can share dashboards with all users in your application, or keep them private.

Rules and actions

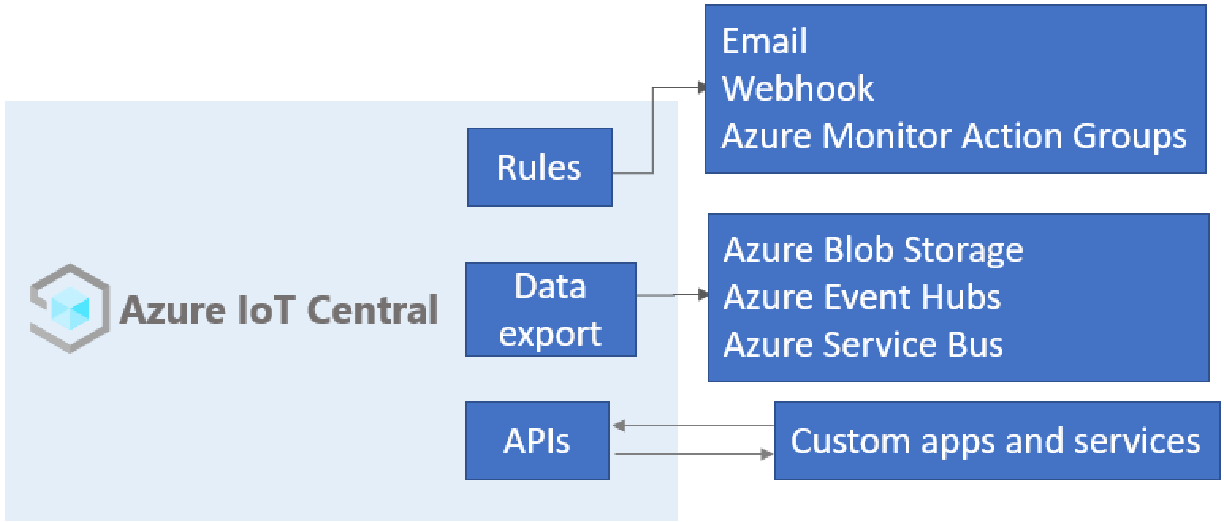
Build custom rules based on device state and telemetry to identify devices in need of attention. Configure actions to notify the right people and ensure corrective measures are taken in a timely fashion.

Jobs

Jobs let you apply single or bulk updates to devices by setting properties or calling commands.

Integrate with other services

As an application platform, IoT Central lets you transform your IoT data into the business insights that drive actionable outcomes. Rules, data export, and the public REST API are examples of how you can integrate IoT Central with line-of-business applications:



You can generate business insights, such as determining machine efficiency trends or predicting future energy usage on a factory floor, by building custom analytics pipelines to process telemetry from your devices and store the results. Configure data exports in your IoT Central application to export telemetry, device property changes, and device template changes to other services where you can analyze, store, and visualize the data with your preferred tools.

Build custom IoT solutions and integrations with the REST APIs

Build IoT solutions such as:

- Mobile companion apps that can remotely set up and control devices.
- Custom integrations that enable existing line-of-business applications to interact with your IoT devices and data.
- Device management applications for device modeling, onboarding, management, and data access.

Administer your application

IoT Central applications are fully hosted by Microsoft, which reduces the administration overhead of managing your applications. Administrators manage access to your application with user roles and permissions.

Pricing

You can create IoT Central application using a 7-day free trial, or use a standard pricing plan.

- Applications you create using the *free* plan are free for seven days and support up to five devices. You can convert them to use a standard pricing plan at any time before they expire.
- Applications you create using the *standard* plan are billed on a per device basis, you can choose either **Standard 1** or **Standard 2** pricing plan with the first two devices being free. Learn more about IoT Central pricing.

Quotas

Each Azure subscription has default quotas that could impact the scope of your IoT solution. Currently, IoT Central limits the number of applications you can deploy in a subscription to 10. If you need to increase this limit, contact Microsoft support.

Known issues

- Continuous data export doesn't support the Avro format (incompatibility).
- GeoJSON isn't currently supported.
- Map tile isn't currently supported.
- Array schema types aren't supported.
- Only the C device SDK and the Node.js device and service SDKs are supported.
- IoT Central is currently available in the United States, Europe, Asia Pacific, Australia, United Kingdom, and Japan locations.
- You cannot use the **Custom application (legacy)** application template in the United Kingdom and Japan locations.
- Device capability models must have all the interfaces defined inline in the same file.
- Support for IoT Plug and Play is in preview and is only supported only in selected regions.

Next steps

Now that you have an overview of IoT Central, here are some suggested next steps:

- Understand the available Azure technologies and services for creating IoT solutions.
- Familiarize yourself with the Azure IoT Central UI.
- Get started by creating an Azure IoT Central application.
- Learn more about IoT Plug and Play (preview).
- Learn how to Connect an Azure IoT Edge device.
- Learn more about Azure IoT technologies and services.

If you're a device developer and want to dive into some code, the suggested next step is to Create and connect a client application to your Azure IoT Central application.

Take a tour of the Azure IoT Central UI

7/21/2020 • 5 minutes to read • Edit Online

This article introduces you to the Microsoft Azure IoT Central UI. You can use the UI to create, manage, and use an Azure IoT Central solution and its connected devices.

As a *solution builder*, you use the Azure IoT Central UI to define your Azure IoT Central solution. You can use the UI to:

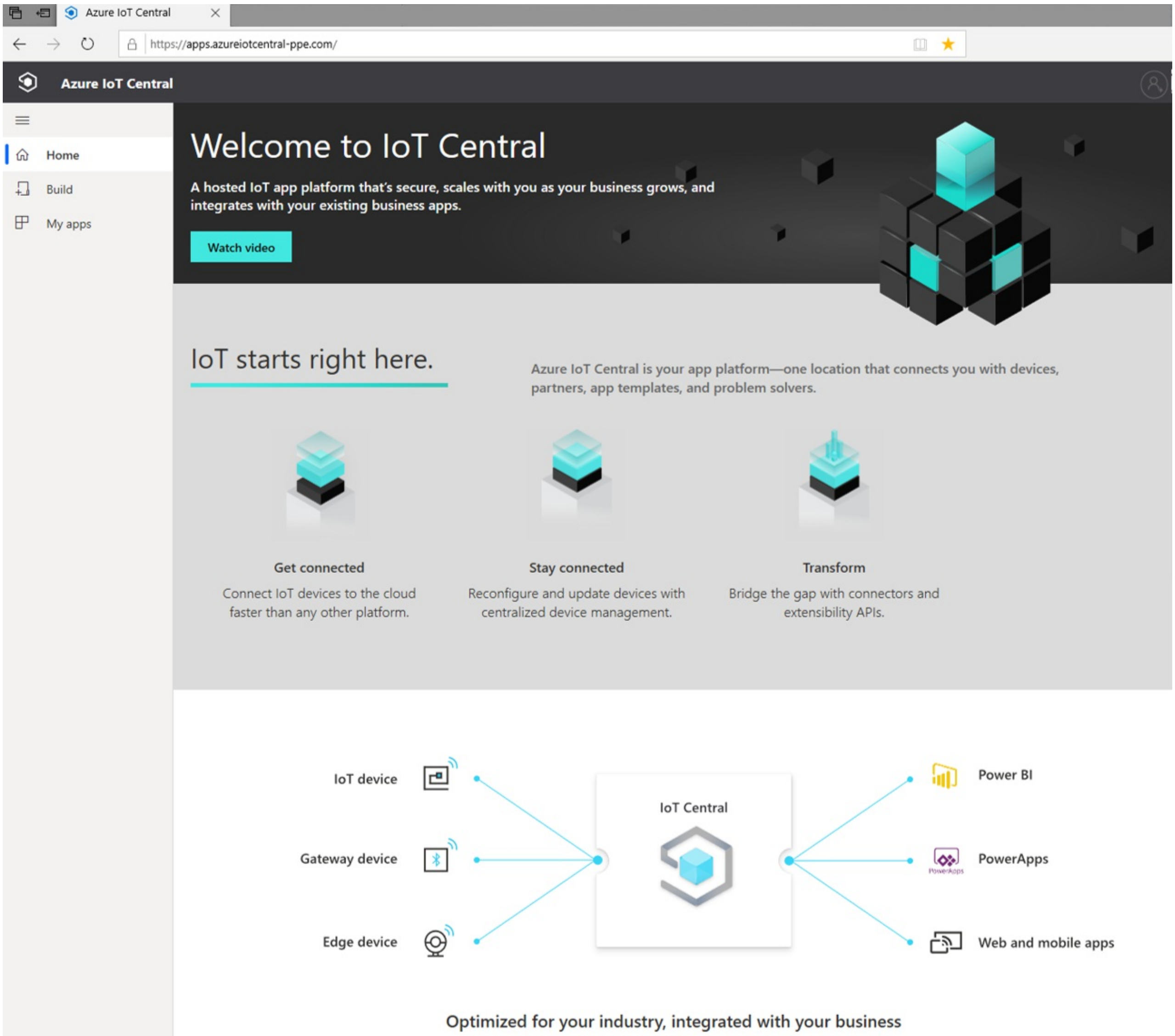
- Define the types of device that connect to your solution.
- Configure the rules and actions for your devices.
- Customize the UI for an *operator* who uses your solution.

As an *operator*, you use the Azure IoT Central UI to manage your Azure IoT Central solution. You can use the UI to:

- Monitor your devices.
- Configure your devices.
- Troubleshoot and remediate issues with your devices.
- Provision new devices.

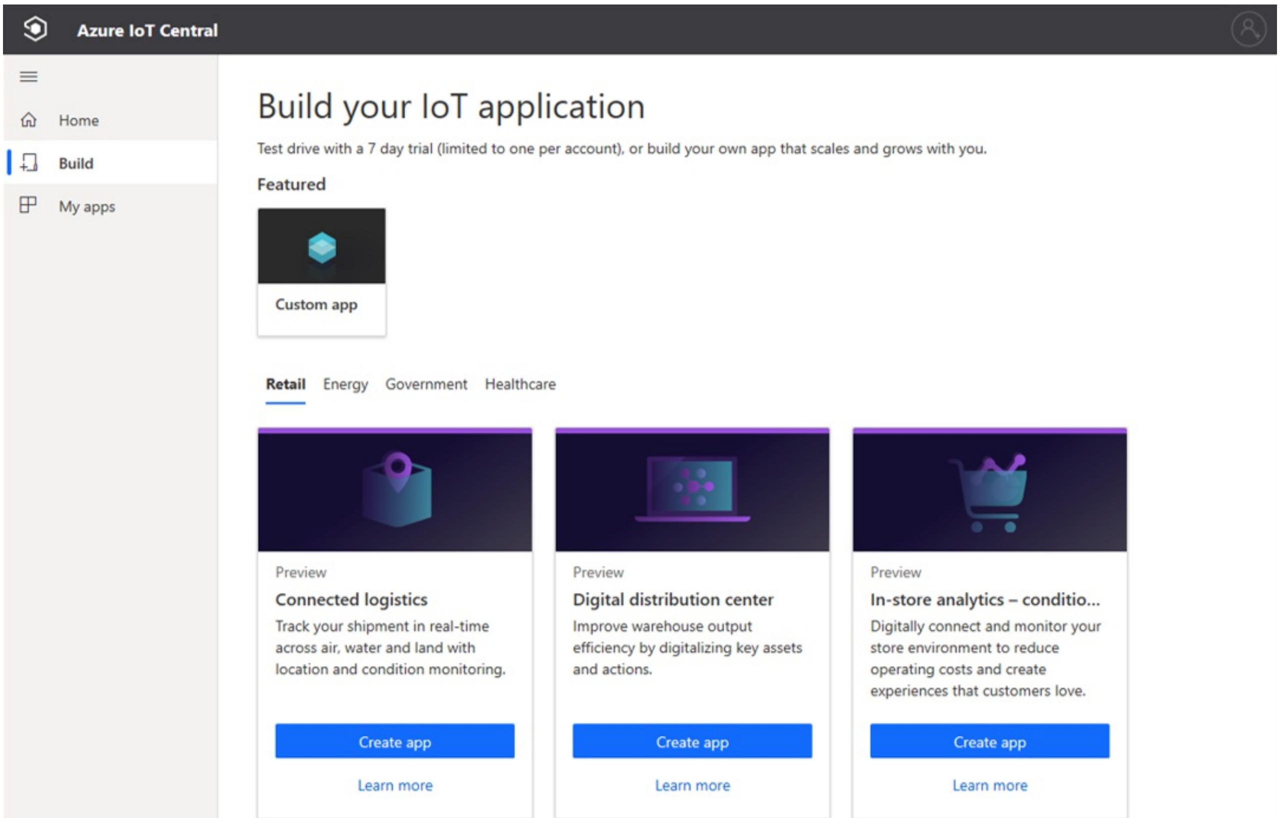
IoT Central homepage

The IoT Central homepage page is the place where you can learn more about the latest news and features available on IoT Central, create new applications, and see and launch your existing application.



Create an application

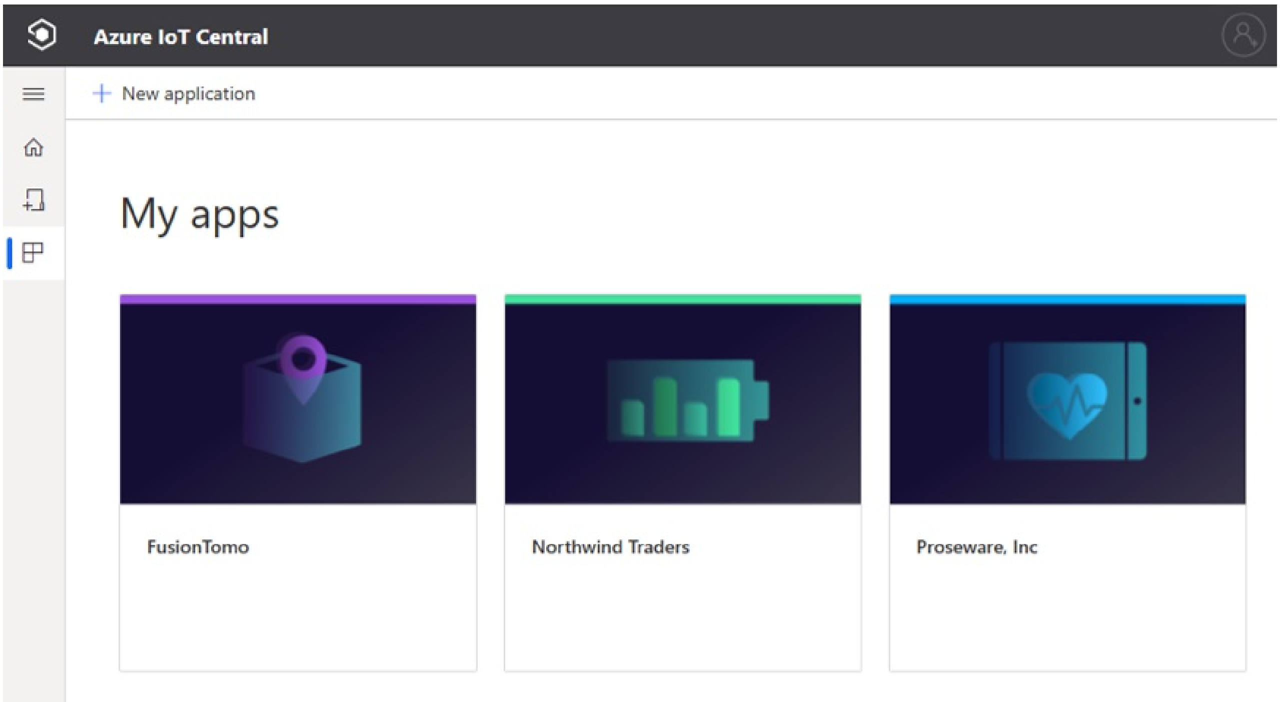
In the Build section you can browse the list of industry-relevant IoT Central templates to help you get started quickly, or start from scratch using a Custom app template.



To learn more, see the Create an Azure IoT Central application quickstart.

Launch your application

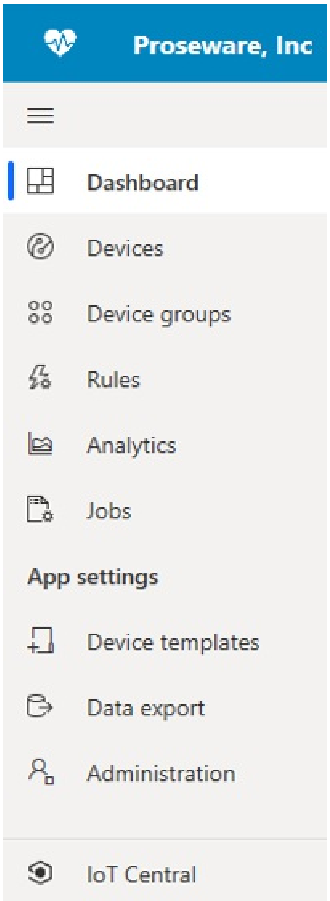
You can launch your IoT Central application by going to the URL that you or your solution builder choose during app creation. You can also see a list of all the applications you have access to in the IoT Central app manager.



Navigate your application

Once you're inside your IoT application, use the left pane to access the different areas. You can expand or collapse the left pane by selecting the three-lined icon on top of the pane:

> **NOTE**
> The items you see in the left pane depend on your user role. Learn more about managing users and roles.



Dashboard displays your application dashboard. As a *solution builder*, you can customize the global dashboard for your operators. Depending on their user role, operators can also create their own personal dashboards.

Devices enables you to manage your connected devices - real and simulated.

Device groups lets you view and create logical collections of devices specified by a query. You can save this query and use device groups through the application to perform bulk operations.

Rules enables you to create and edit rules to monitor your devices. Rules are evaluated based on device telemetry and trigger customizable actions.

Analytics lets you create custom views on top of device data to derive insights from your application.

Jobs enables you to manage your devices at scale by running bulk operations.

Device templates is where you create and manage the characteristics of the devices that connect to your application.

Data export enables you to configure a continuous export to external services - such as storage and queues.

Administration is where you can manage your application's settings, customization, billing, users, and roles.

IoT Central lets *administrators* to jump back to IoT Central's app manager.

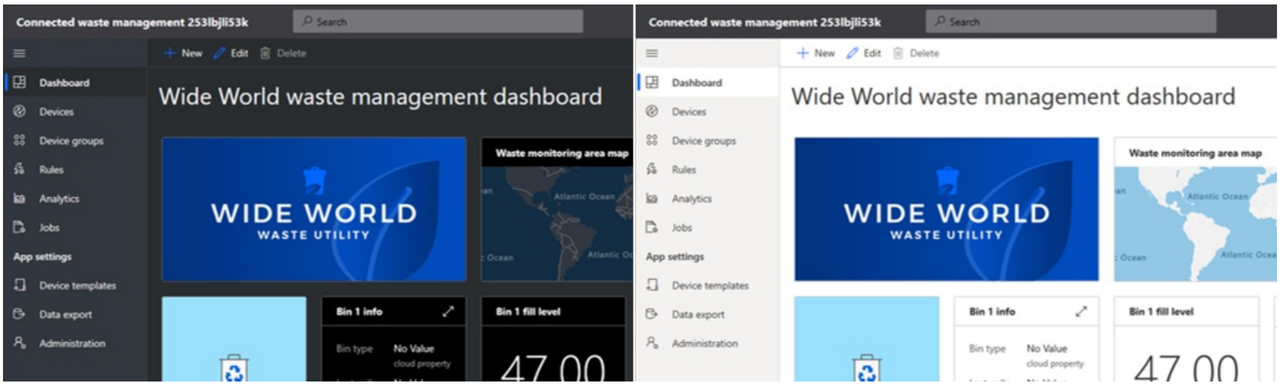
Search, help, theme, and support

The top menu appears on every page:

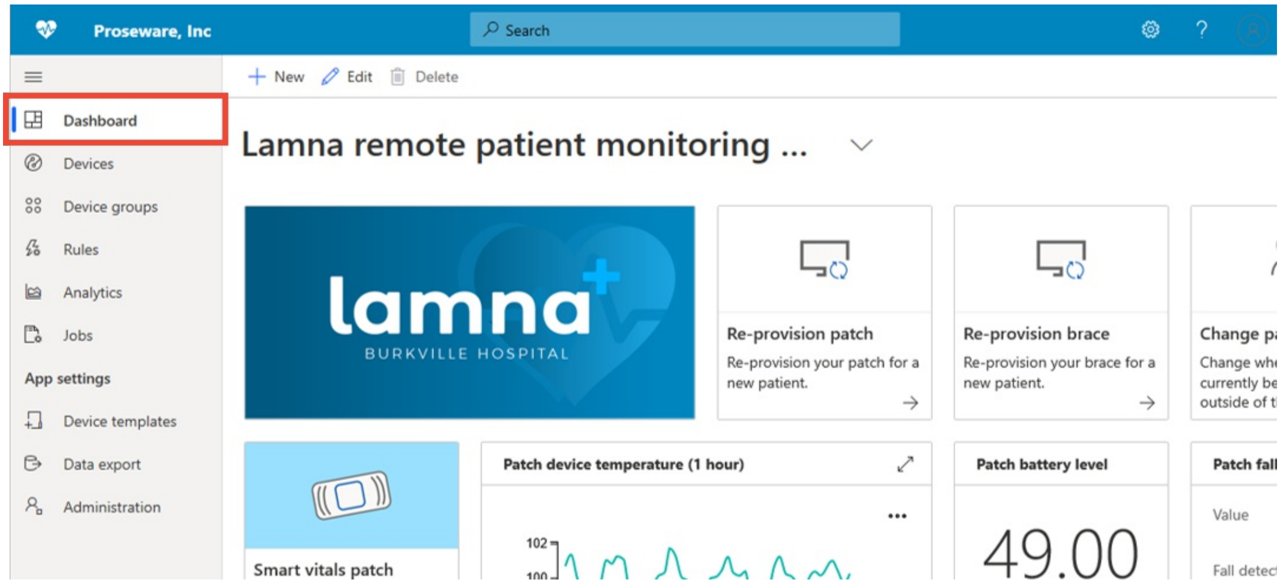
- To search for device templates and devices, enter a **Search** value.
- To change the UI language or theme, choose the **Settings** icon. Learn more about managing your application preferences
- To sign out of the application, choose the **Account** icon.
- To get help and support, choose the **Help** drop-down for a list of resources. You can get information about your application from the **About your app** link. In an application on the free pricing plan, the support resources include access to live chat.

You can choose between a light theme or a dark theme for the UI:

> **NOTE**
>
> The option to choose between light and dark themes isn't available if your administrator has configured a custom theme for the application.

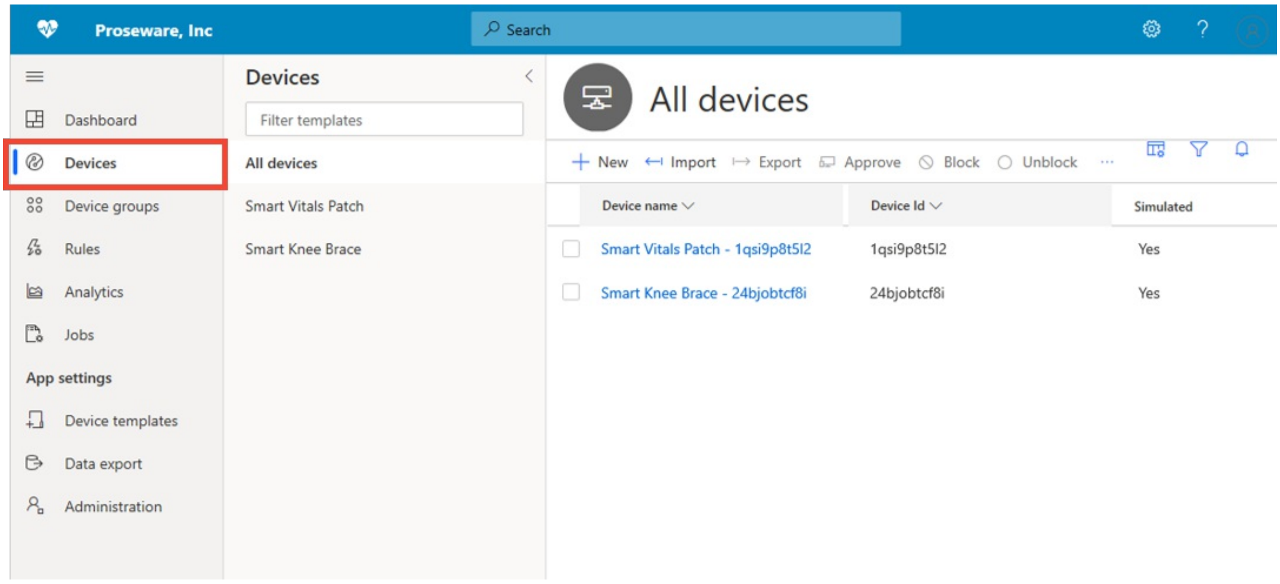


Dashboard



- The dashboard is the first page you see when you sign in to your Azure IoT Central application. As a *solution builder*, you can create and customize multiple global application dashboards for other users. Learn more about adding tiles to your dashboard
- As an *operator*, if your user role allows it, you can create personal dashboards to monitor what you care about. To learn more, see the Create Azure IoT Central personal dashboards how-to article.

Devices

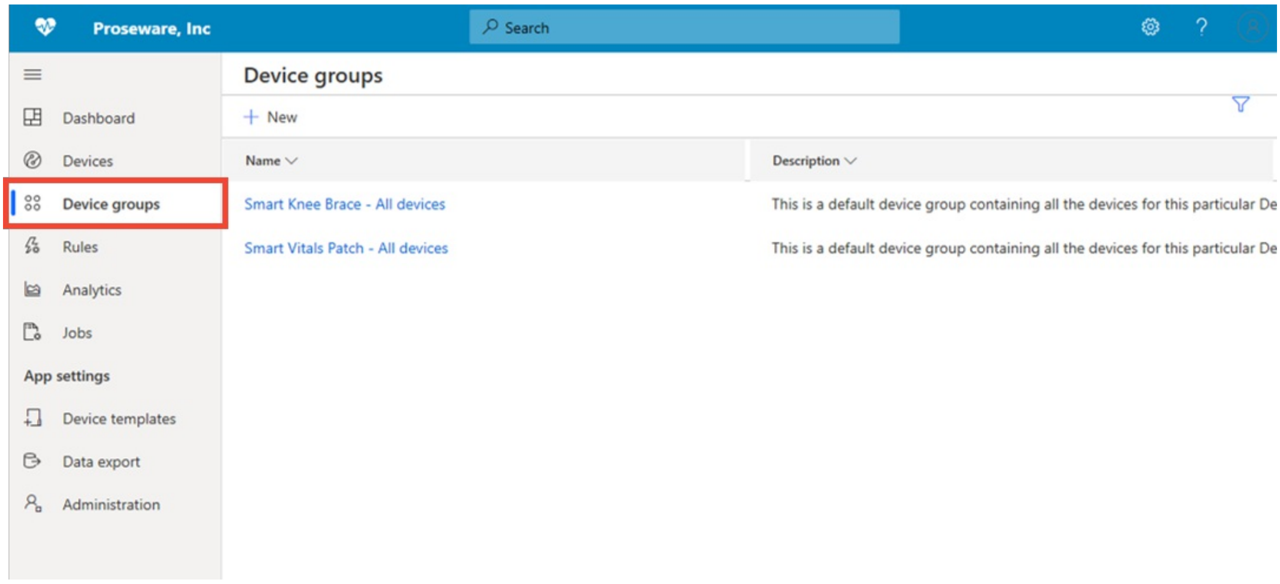


The explorer page shows the *devices* in your Azure IoT Central application grouped by *device template*.

- A device template defines a type of device that can connect to your application.
- A device represents either a real or simulated device in your application.

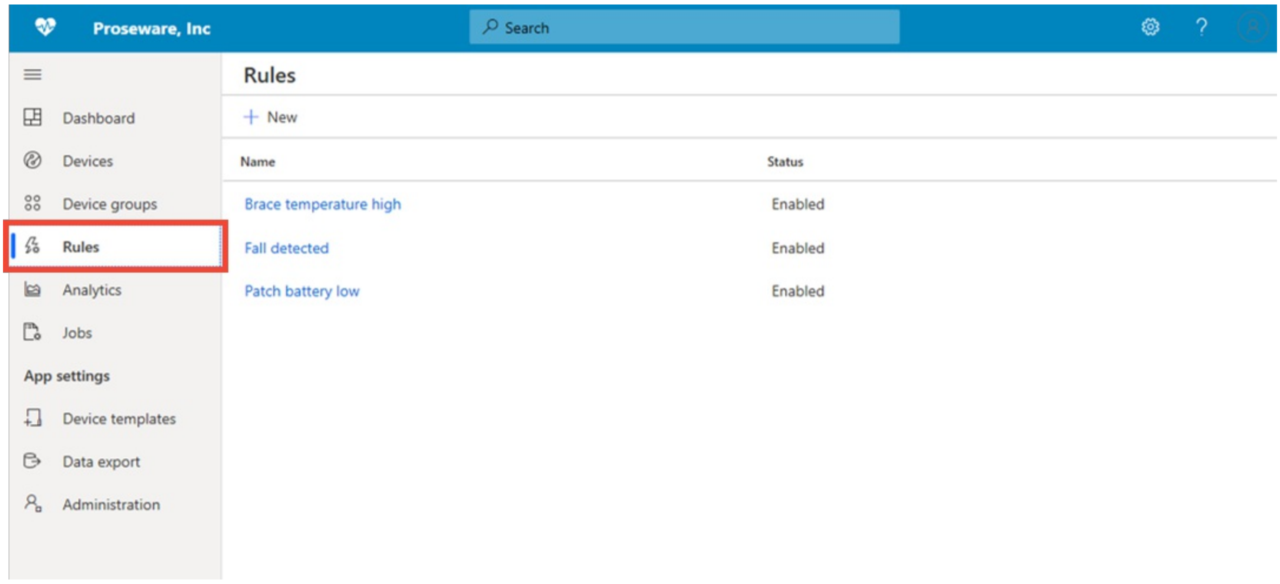
To learn more, see the Monitor your devices quickstart.

Device groups



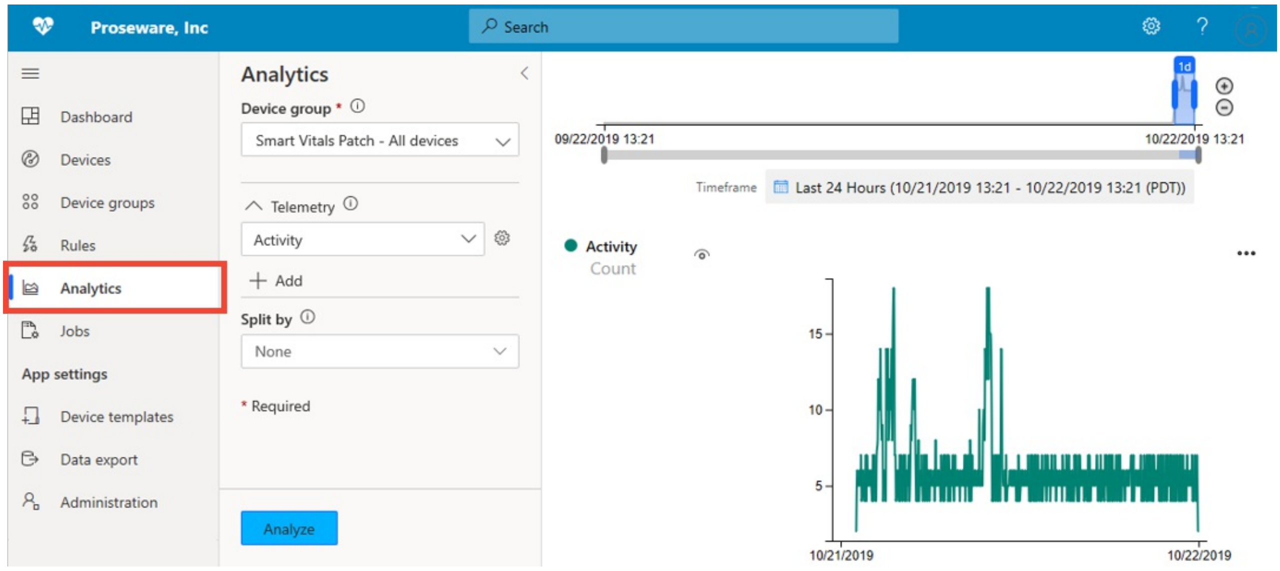
Device group are a collection of related devices. A *solution builder* defines a query to identify the devices that are included in a device group. You use device groups to perform bulk operations in your application. To learn more, see the Use device groups in your Azure IoT Central application article.

Rules



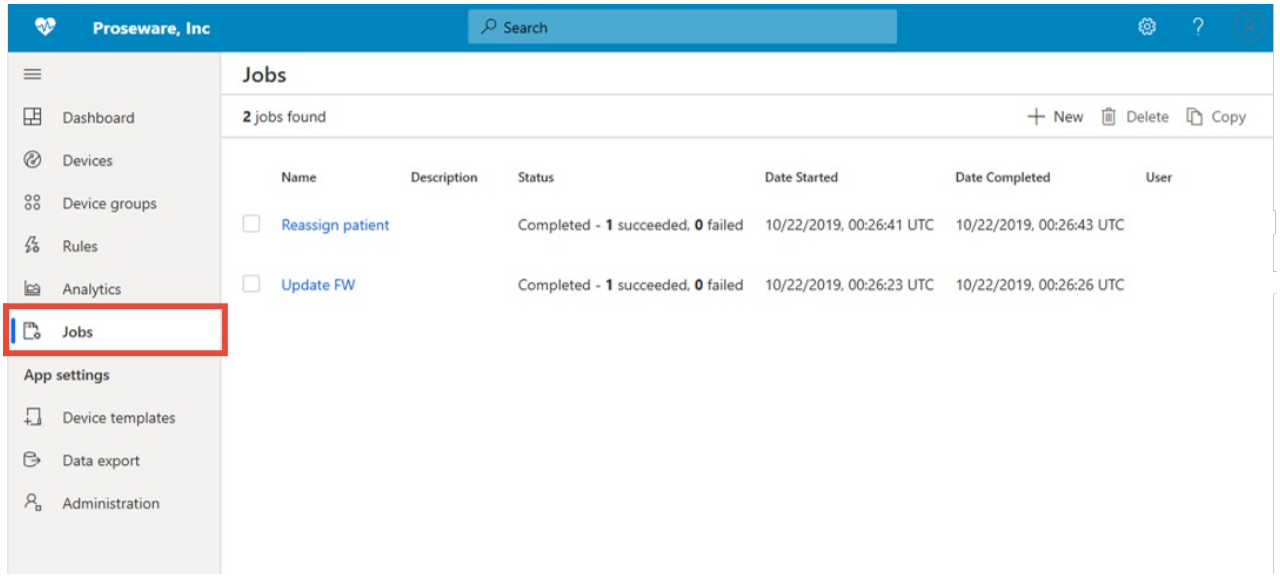
The rules page lets you define rules based on devices' telemetry, state, or events. When a rule fires, it can trigger one or more actions - such as sending an email, notify an external system via webhook alerts, etc. To learn, see the Configuring rules tutorial.

Analytics



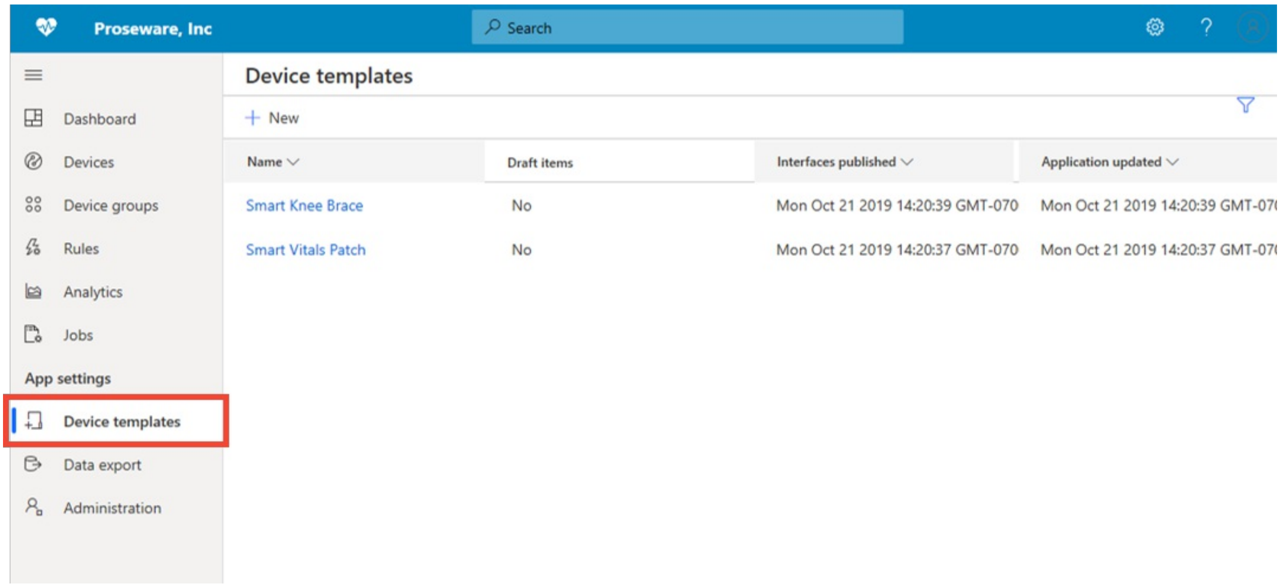
The analytics lets you create custom views on top of device data to derive insights from your application. To learn more, see the Create analytics for your Azure IoT Central application article.

Jobs



The jobs page lets you run bulk device management operations on your devices. You can update device properties, settings, and execute commands against device groups. To learn more, see the Run a job article.

Device templates



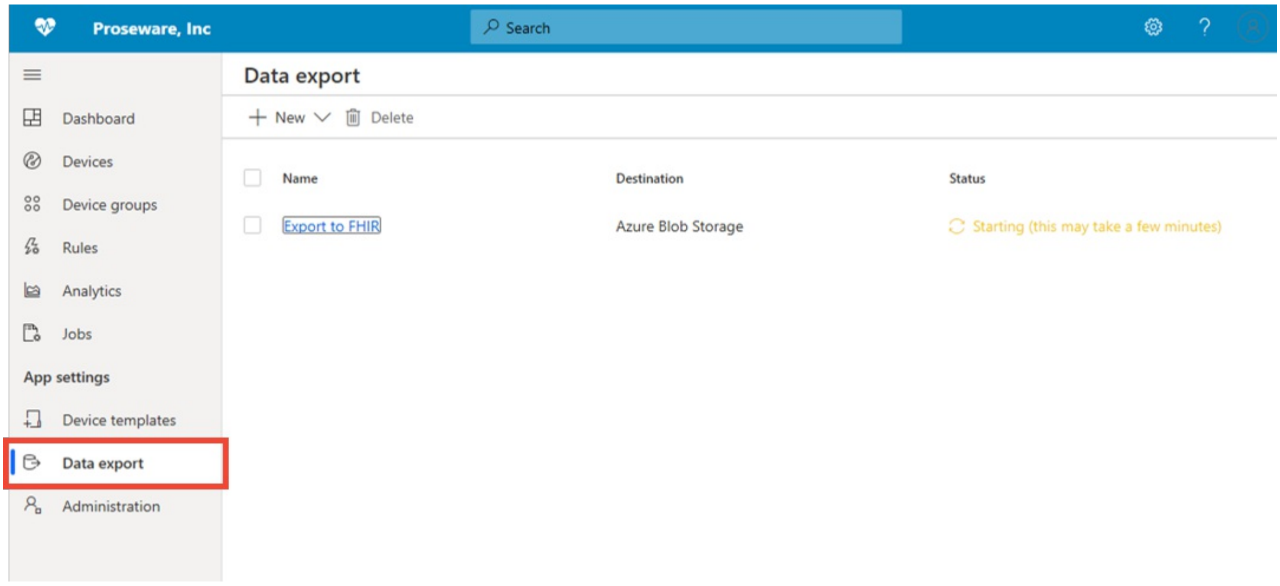
The device templates page is where a builder creates and manages the device templates in the application. A device template specifies devices characteristics such as:

- Telemetry, state, and event measurements
- Properties
- Commands
- Views

The *solution builder* can also create forms and dashboards for operators to use to manage devices.

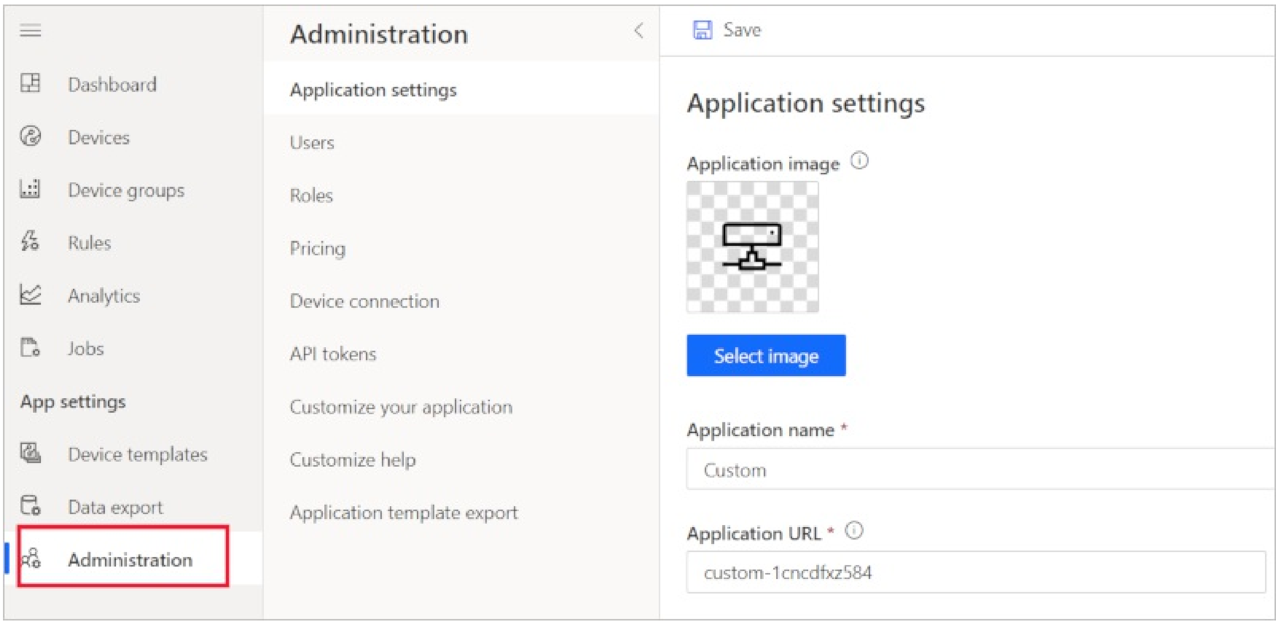
To learn more, see the Define a new device type in your Azure IoT Central application tutorial.

Data export



Data export enables you to set up streams of data, such as telemetry, from the application to external systems. To learn more, see the Export your data in Azure IoT Central article.

Administration



The administration page allows you to configure and customize your IoT Central application. Here you can change your application name, URL, theming, manage users and roles, create API tokens, and export your application. To learn more, see the Administer your Azure IoT Central application article.

Next steps

Now that you have an overview of Azure IoT Central and are familiar with the layout of the UI, the suggested next step is to complete the Create an Azure IoT Central application quickstart.

IoT Central device development overview

7/21/2020 • 4 minutes to read • Edit Online

This article applies to device developers.

An IoT Central application lets you monitor and manage millions of devices throughout their life cycle. This overview is intended for device developers who implement code to run on devices that connect to IoT Central.

Devices interact with an IoT Central application using the following primitives:

- *Telemetry* is data that a device sends to IoT Central. For example, a stream of temperature values from an onboard sensor.
- *Properties* are state values that a device reports to IoT Central. For example, the current firmware version of the device. You can also have writable properties that IoT Central can update on the device.
- *Commands* are called from IoT Central to control the behavior a device. For example, your IoT Central application might call a command to reboot a device.

A solution builder is responsible for configuring dashboards and views in the IoT Central web UI to visualize telemetry, manage properties, and call commands.

Types of device

The following sections describe the main types of device you can connect to an IoT Central application:

Standalone device

A standalone device connects directly to IoT Central. A standalone device typically sends telemetry from its onboard or connected sensors to your IoT Central application. Standalone devices can also report property values, receive writable property values, and respond to commands.

Gateway device

A gateway device manages one or more downstream devices that connect to your IoT Central application. You use IoT Central to configure the relationships between the downstream devices and the gateway device. To learn more, see Define a new IoT gateway device type in your Azure IoT Central application.

Edge device

An edge device connects directly to IoT Central, but acts as an intermediary for other devices known as *leaf devices*. An edge device is typically located close to the leaf devices for which it's acting as an intermediary. Scenarios that use edge devices include:

- Enable devices that can't connect directly to IoT Central to connect through the edge device. For example, a leaf device might use bluetooth to connect to the edge device, which then connects over the internet to IoT Central.
- Aggregate telemetry before it's sent to IoT Central. This approach can help to reduce the costs of sending data to IoT Central.
- Control leaf devices locally to avoid the latency associated with connecting to IoT Central over the internet.

An edge device can also send its own telemetry, report its properties, and respond to writable property updates and commands.

IoT Central only sees the edge device, not the leaf devices connected to the edge device.

To learn more, see Add an Azure IoT Edge device to your Azure IoT Central application.

Connect a device

Azure IoT Central uses the Azure IoT Hub Device Provisioning service (DPS) to manage all device registration and connection.

Using DPS enables:

- IoT Central to support onboarding and connecting devices at scale.
- You to generate device credentials and configure the devices offline without registering the devices through IoT Central UI.
- You to use your own device IDs to register devices in IoT Central. Using your own device IDs simplifies integration with existing back-office systems.
- A single, consistent way to connect devices to IoT Central.

To learn more, see Get connected to Azure IoT Central.

Security

The connection between a device and your IoT Central application is secured using either shared access signatures or industry-standard X.509 certificates.

Communication protocols

Communication protocols that a device can use to connect to IoT Central include MQTT, AMQP, and HTTPS. Internally, IoT Central uses an IoT hub to enable device connectivity. For more information about the communication protocols that IoT Hub supports for device connectivity, see Choose a communication protocol.

Implement the device

Use one of the Azure IoT device SDKs to implement the behavior of your device. The code should:

- Register the device with DPS and use the information from DPS to connect to the internal IoT hub in your IoT Central application.
- Send telemetry in the format that the device template in IoT Central specifies. IoT Central uses the device template to determine how to use the telemetry for visualizations and analysis.
- Synchronize property values between the device and IoT Central. The device template specifies the property names and data types so that IoT Central can display the information.
- Implement command handlers for the commands specifies in the device template. The device template specifies the command names and parameters that the device should use.

For more information about the role of device templates, see What are device templates?.

For some sample code, see Create and connect a Node.js client application or Create and connect a Python client application.

Languages and SDKs

For more information about the supported languages and SDKs, see Understand and use Azure IoT Hub device SDKs.

Next steps

If you're a device developer and want to dive into some code, the suggested next step is to Create and connect a client application to your Azure IoT Central application.

If you want to learn more about using IoT Central, the suggested next steps are to try the quickstarts, beginning with Create an Azure IoT Central application.

Create an Azure IoT Central application

4/21/2020 • 2 minutes to read • Edit Online

This quickstart shows you how to create an Azure IoT Central application.

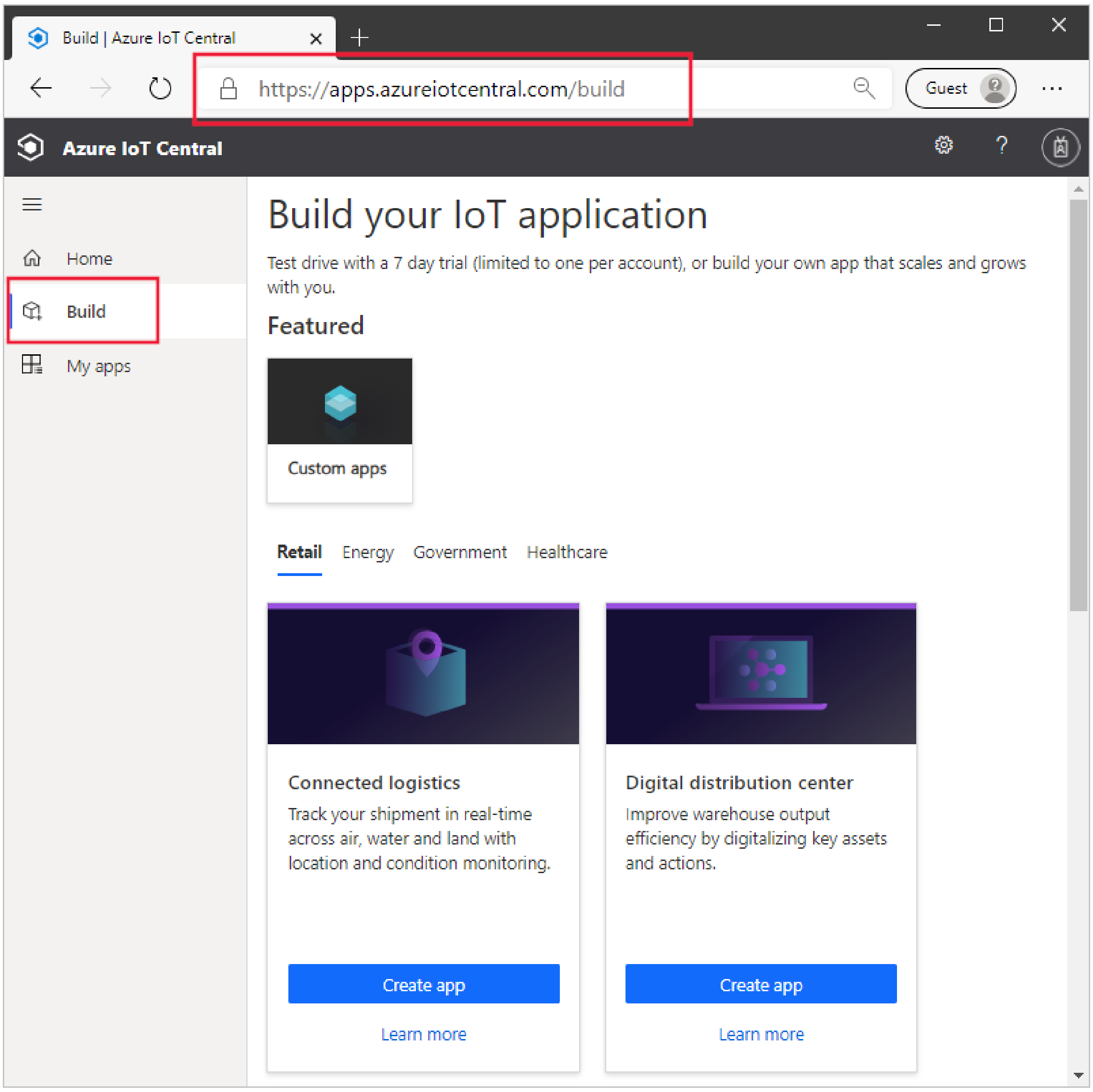
Create an application

Navigate to the Azure IoT Central Build site. Then sign in with a Microsoft personal, work, or school account.

You create a new application either from the list of industry-relevant IoT Central templates to help you get started quickly, or start from scratch using a **Custom apps** template. In this quickstart, you use the **Custom application** template.

To create a new Azure IoT Central application from the **Custom application** template:

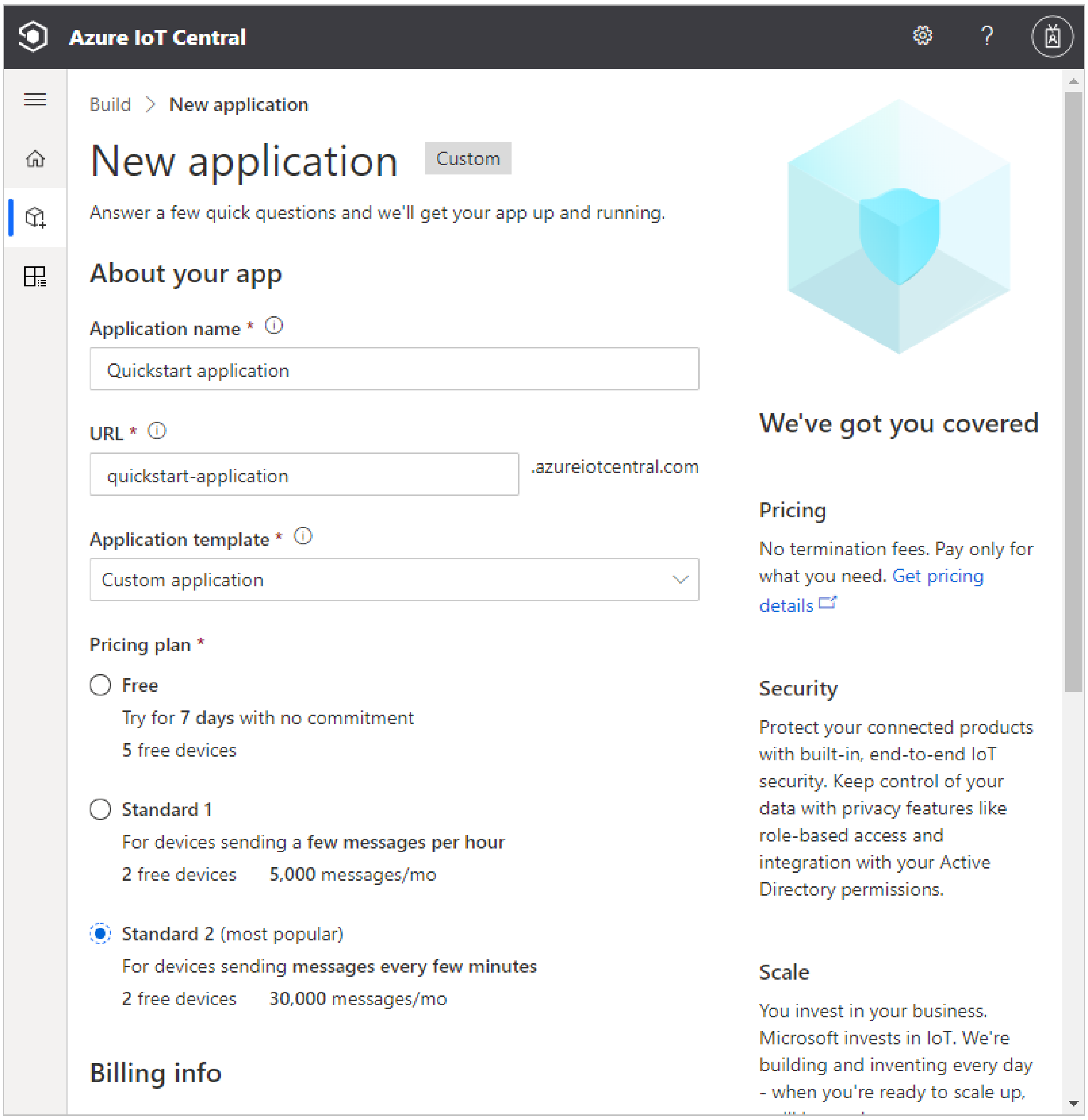
1. Navigate to the **Build** page:

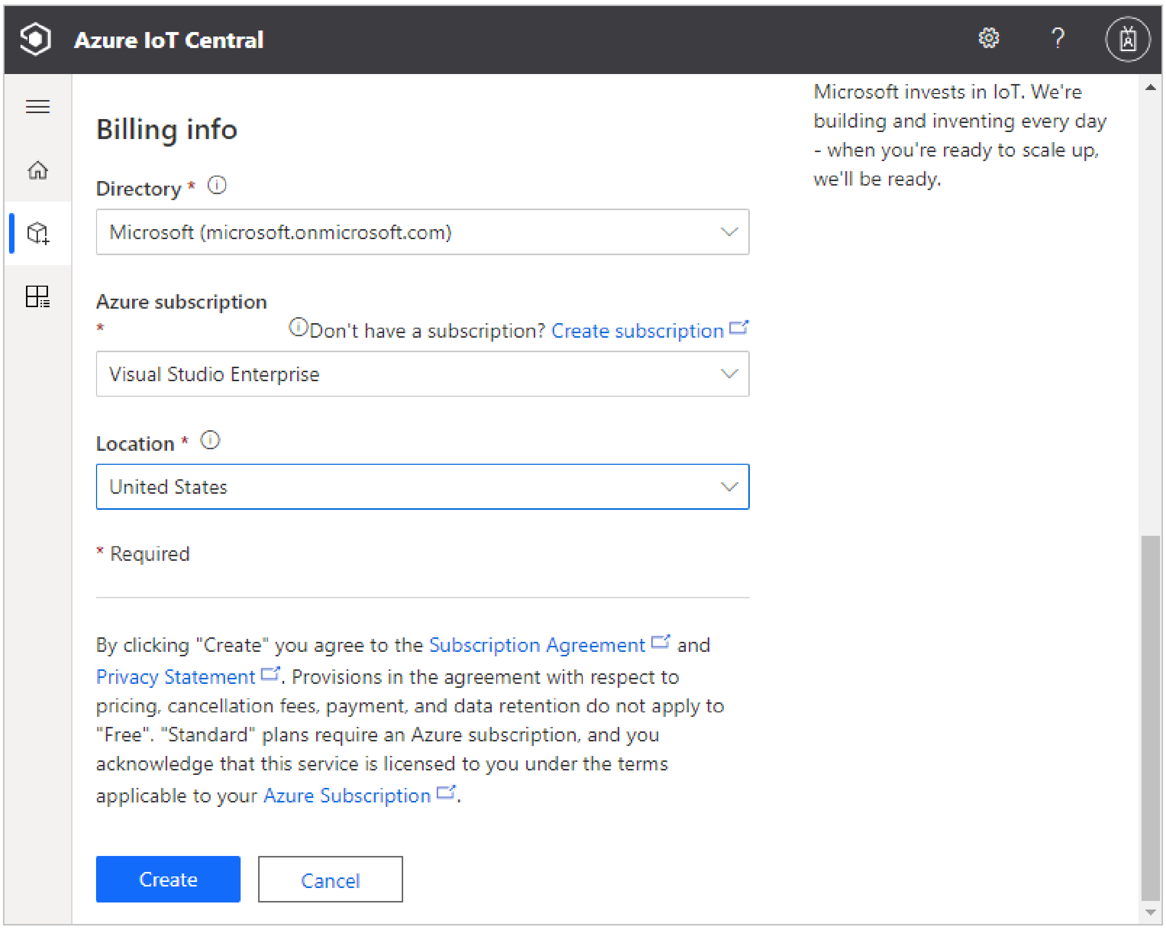


2. Choose **Custom apps** and make sure that the **Custom application** template is selected.

3. Azure IoT Central automatically suggests an **application name** based on the application template you've selected. You can use this name or enter your own friendly application name.

4. Azure IoT Central also generates a unique **application URL** prefix for you, based on the application name. You use this URL to access your application. Change this URL prefix to something more memorable if you'd like.



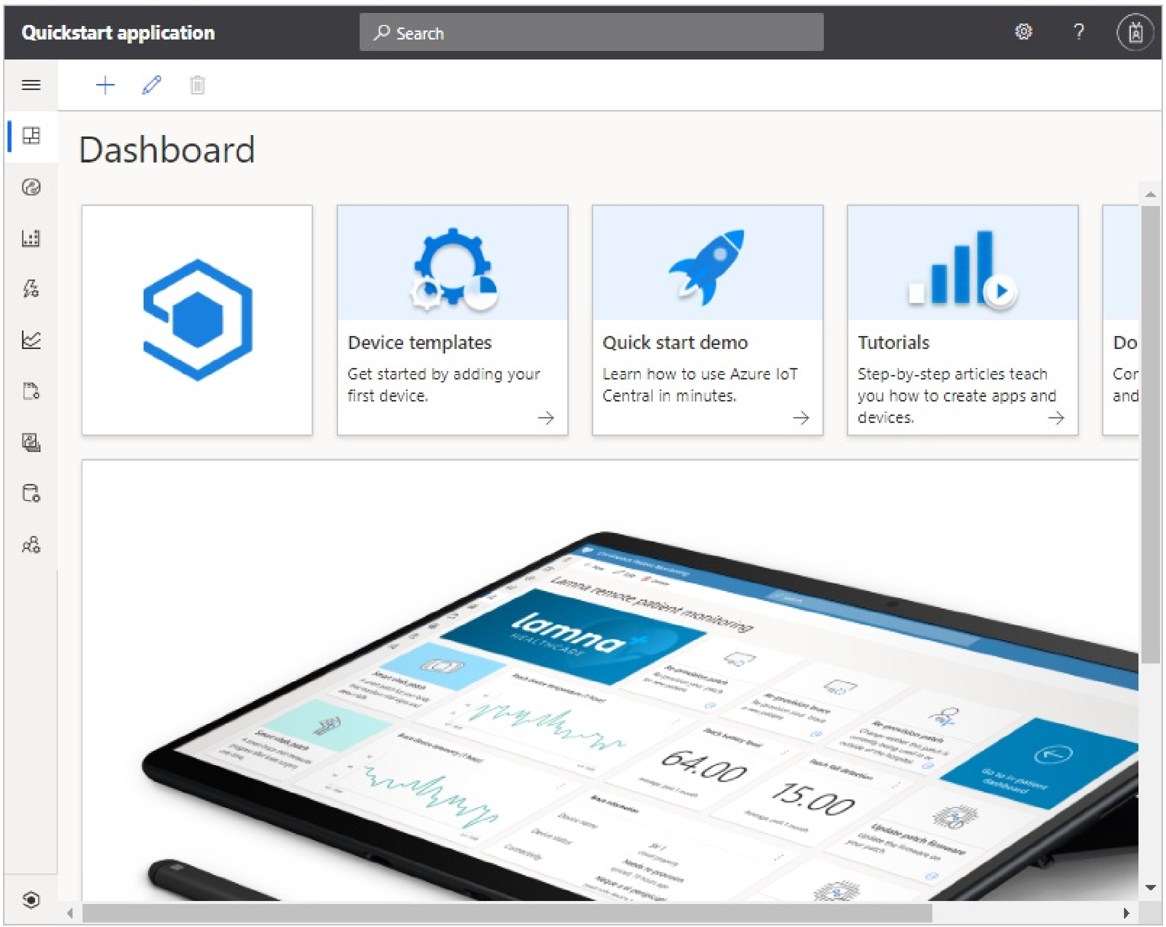


> **NOTE**
>
> If you chose **Custom app** on the previous page, you see an **Application template** dropdown. From here you can switch between custom and legacy templates. You might also see other templates that have been made available for your organization.

5. Choose to create this application using the 7-day free trial pricing plan, or one of the standard pricing plans:

- Applications you create using the *free* plan are free for seven days and support up to five devices. You can convert them to use a standard pricing plan at any time before they expire.
- Applications you create using a *standard* plan are billed on a per device basis, you can choose either **Standard 1** or **Standard 2** pricing plan with the first two devices being free. Learn more about the free and standard pricing plans on the Azure IoT Central pricing page. If you create an application using a standard pricing plan, you need to select your *Directory*, *Azure Subscription*, and *Location*:
  - *Directory* is the Azure Active Directory in which you create your application. An Azure Active Directory contains user identities, credentials, and other organizational information. If you don't have an Azure Active Directory, one is created for you when you create an Azure subscription.
  - An *Azure Subscription* enables you to create instances of Azure services. IoT Central provisions resources in your subscription. If you don't have an Azure subscription, you can create one for free on the Azure sign-up page. After you create the Azure subscription, navigate back to the **New application** page. Your new subscription now appears in the **Azure Subscription** drop-down.
  - *Location* is the geography where you'd like to create your application. Typically, you should choose the location that's physically closest to your devices to get optimal performance. Once you choose a location, you can't later move your application to a different location.

6. Review the Terms and Conditions, and select **Create** at the bottom of the page. After a few minutes, you IoT Central application is ready to use:



Next steps

In this quickstart, you created an IoT Central application. Here's the suggested next step to continue learning about IoT Central:

Add a simulated device to your IoT Central application

If you're a device developer and want to dive into some code, the suggested next step is to:

Create and connect a client application to your Azure IoT Central application

Quickstart: Add a simulated device to your IoT Central application

7/21/2020 • 6 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

A device template defines the capabilities of a device that connects to your IoT Central application. Capabilities include telemetry the device sends, device properties, and the commands a device responds to. From a device template, a builder or operator can add both real and simulated devices to an application. Simulated devices are useful for testing the behavior of your IoT Central application before you connect real devices.

In this quickstart, you add a device template for an *MXChip IoT DevKit* (DevKit) board and create a simulated device. To complete this quickstart you don't need a real device, you work with a simulation of the device. A DevKit device:

- Sends telemetry such as temperature.
- Reports device-specific properties such as brightness level.
- Responds to commands such as turn on and turn off.
- Reports generic device properties such as firmware version and serial number.

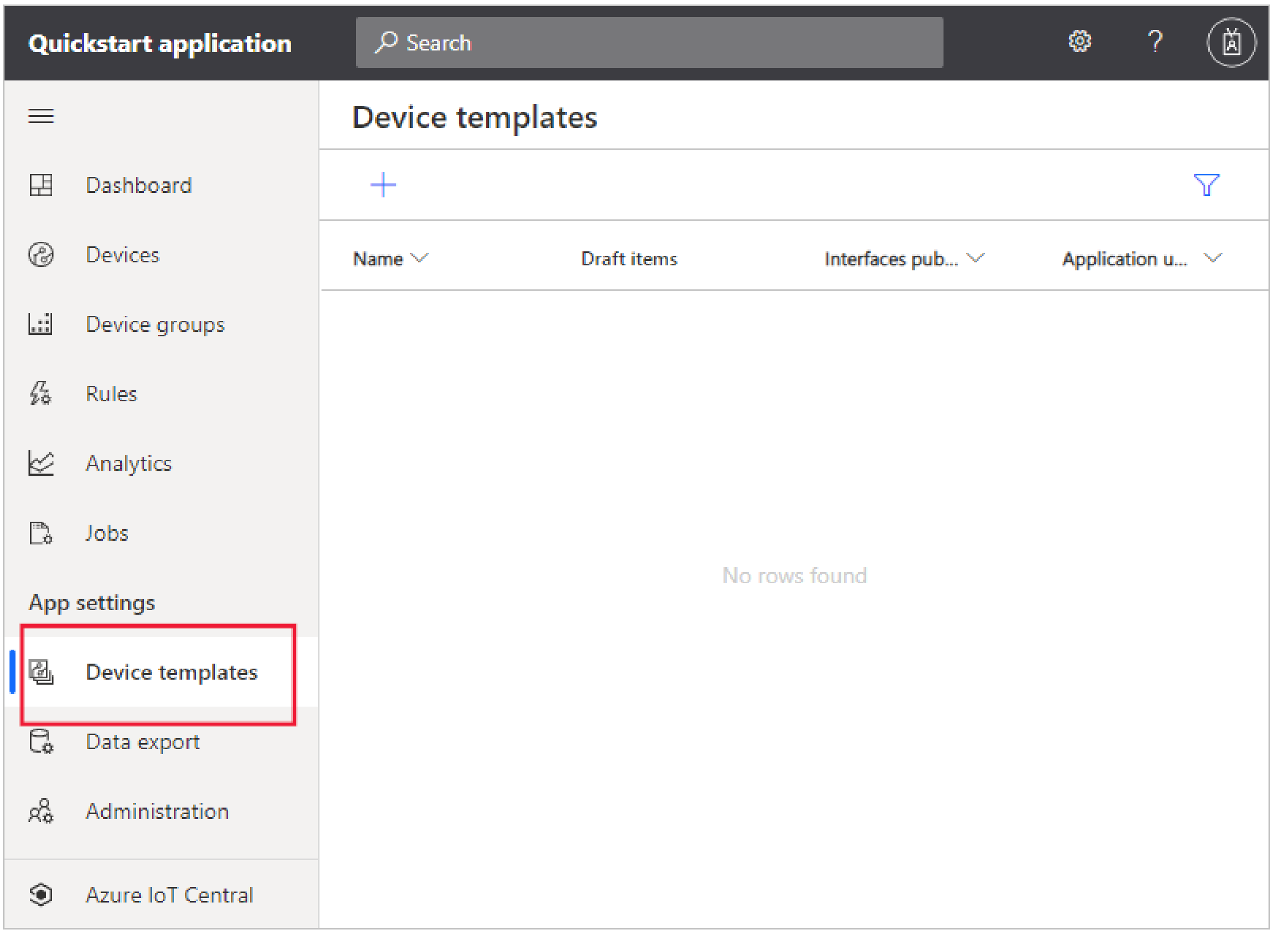
Prerequisites

Complete the Create an Azure IoT Central application quickstart to create an IoT Central application using the **Custom app > Custom application** template.

Create a template

As a builder, you can create and edit device templates in your IoT Central application. After you publish a device template, you can generate simulated device or connect real devices from the device template. Simulated devices let you test the behavior of your application before you connect a real device.

To add a new device template to your application, select the **Device Templates** tab in the left pane.



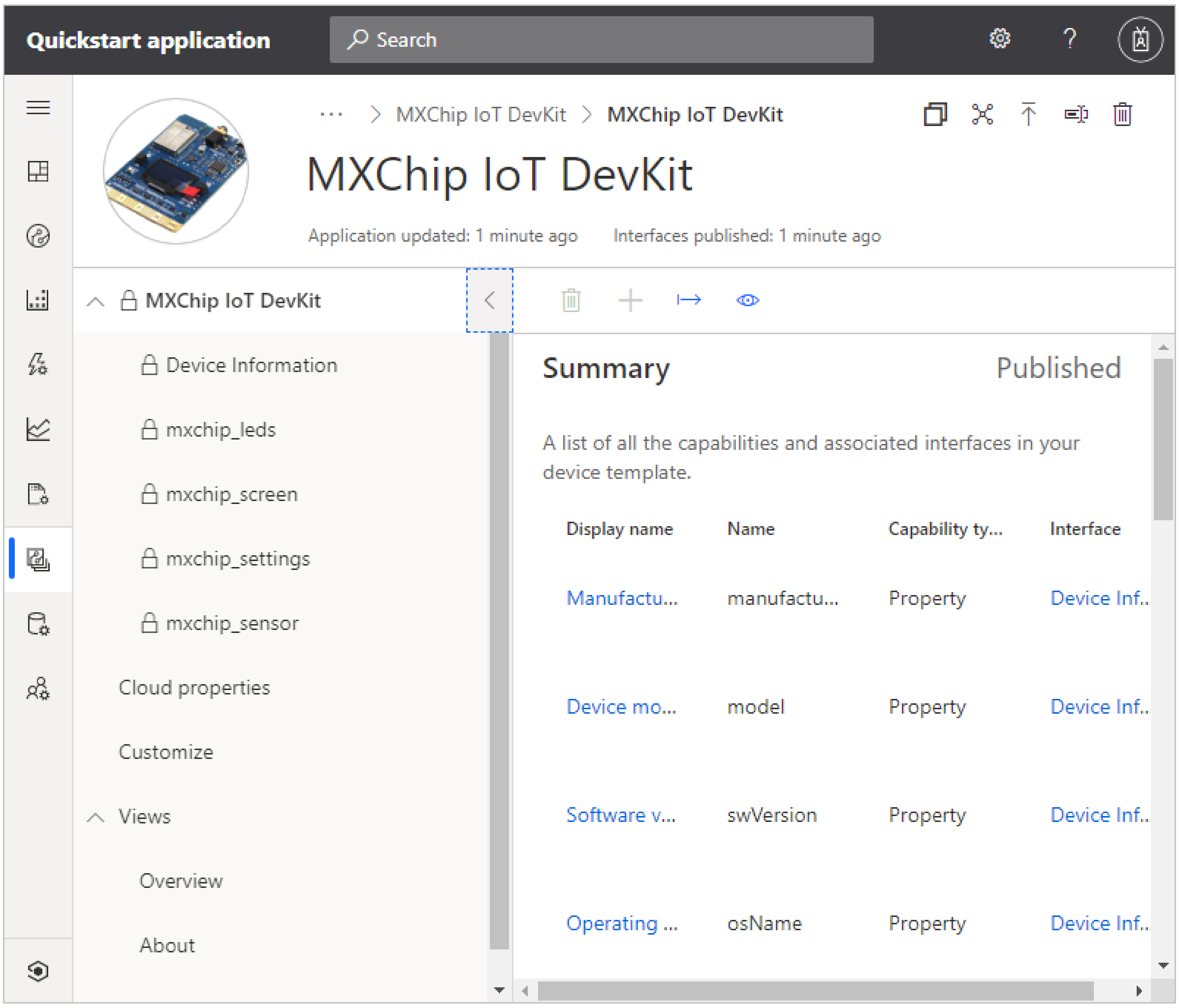
A device template includes a device capability model that defines the telemetry the device sends, device properties, and the commands the device responds to.

Add a device capability model

There are several options for adding a device capability model to your IoT Central application. You can create a model from scratch, import a model from a file, or select a device from the device catalog. IoT Central also supports a *device-first* approach where it automatically imports a model from a repository when a device connects for the first time. In this quickstart, you choose a device from the device catalog to import its device capability model.

The following steps show you how to use the device catalog to import the capability model for an **MXChip IoT DevKit** device. These devices send telemetry, such as temperature, to your application:

1. To add a new device template, select **+** on the **Device templates** page.
2. On the **Select template type** page, scroll down until you find the **MXChip IoT DevKit** tile.
3. Select the **MXChip IoT DevKit** tile, and then select **Next: Customize**.
4. On the **Review** page, select **Create**.
5. After a few seconds, you can see your new device template:



The MXChip IoT DevKit capability model includes interfaces such as **mxchip_sensor**, **mxchip_settings**, and **Device Information**. Interfaces define the capabilities of an MXChip IoT DevKit device. Capabilities include the telemetry a device sends, the properties a device reports, and the commands a device responds to.

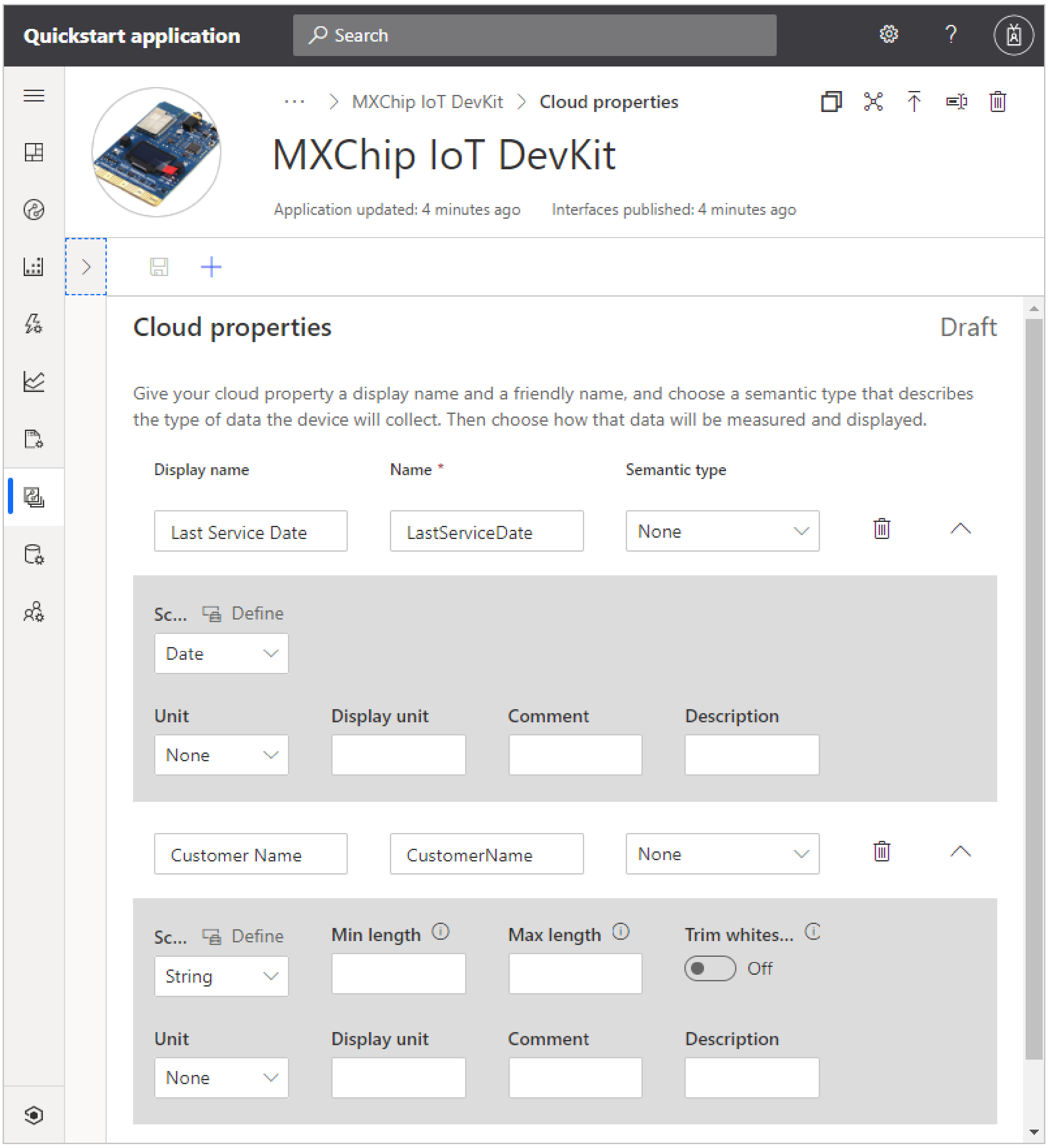
Add cloud properties

A device template can include cloud properties. Cloud properties only exist in the IoT Central application and are never sent to, or received from, a device. To add a cloud property:

1. Select **Cloud Properties** and then **+ Add cloud property**. Use the information in the following table to add two cloud properties to your device template:

DISPLAY NAME	SEMANTIC TYPE	SCHEMA
Last Service Date	None	Date
Customer Name	None	String

2. Select **Save** to save your changes:



Views

As a builder, you can customize the application to display relevant information about the device to an operator. Your customizations enable the operator to manage the devices connected to the application. You can create two types of views for an operator to use to interact with devices:

- Forms to view and edit device and cloud properties.
- Dashboards to visualize devices including the telemetry they send.

Default views

Default views are a quick way to get started with visualizing your important device information. You can have up to three default views generated for your device template:

- The **Commands** view lets your operator dispatch commands to your device.
- The **Overview** view uses charts and metrics to display device telemetry.
- The **About** view displays device properties.

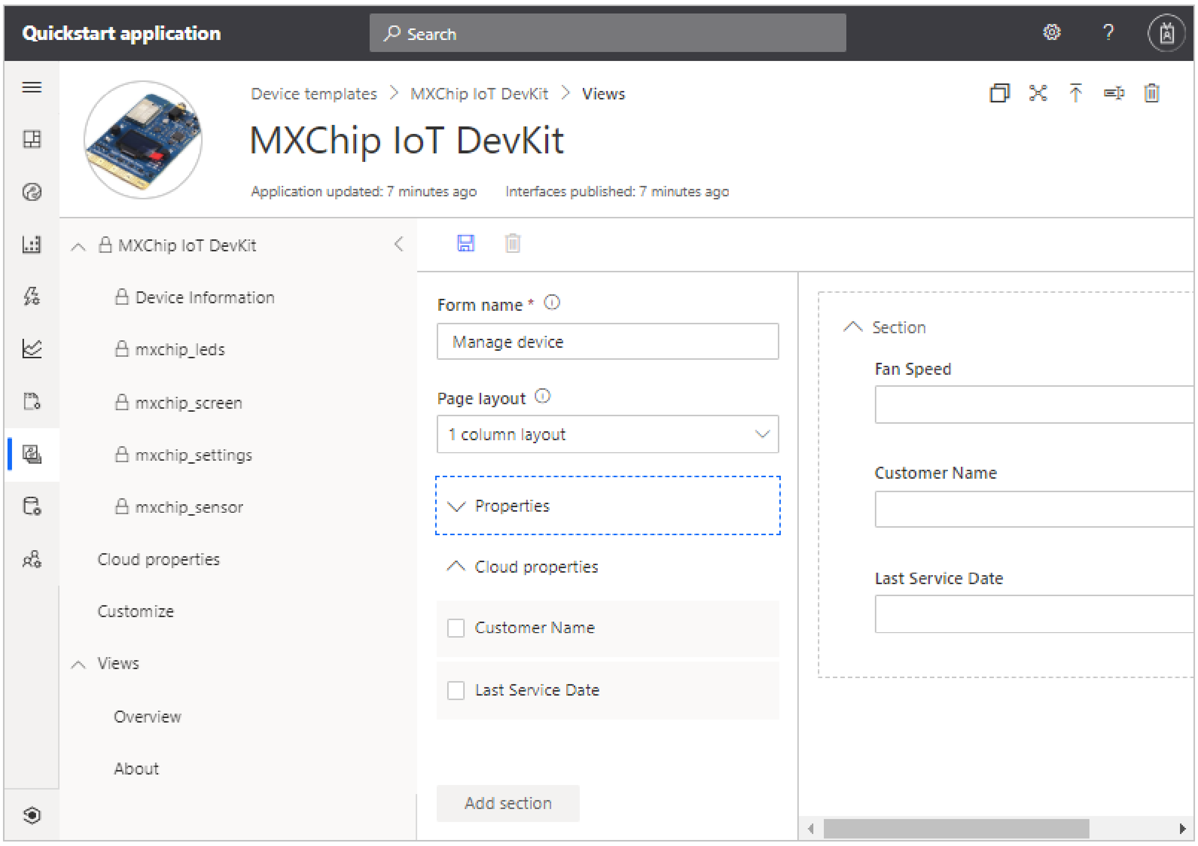
Select the **Views** node in the device template. You can see that IoT Central generated an **Overview** and an **About** view for you when you added the template.

To add a new **Manage device** form that an operator can use to manage the device:

1. Select the **Views** node, and then select the **Editing device and cloud data** tile to add a new view.

2. Change the form name to **Manage device**.

3. Select the **Customer Name** and **Last Service Date** cloud properties, and the **Fan Speed** property. Then select **Add section**:



4. Select **Save** to save your new form.

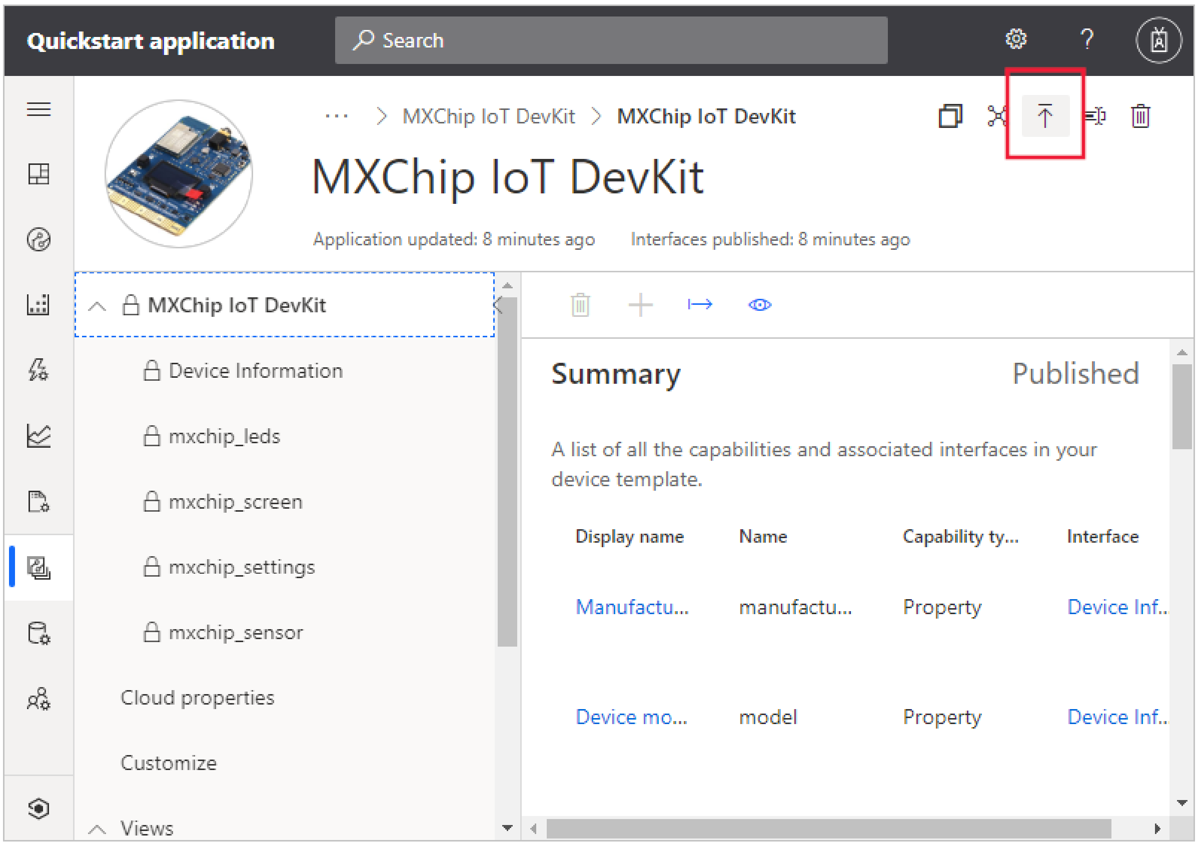
Publish device template

Before you can create a simulated device, or connect a real device, you need to publish your device template. Although IoT Central published the template when you first created it, you must publish the updated version.

To publish a device template:

1. Go to your device template from the **Device Templates** page.

2. Select **Publish**:



3. On the **Publish this device template to the application** dialog, select **Publish**.

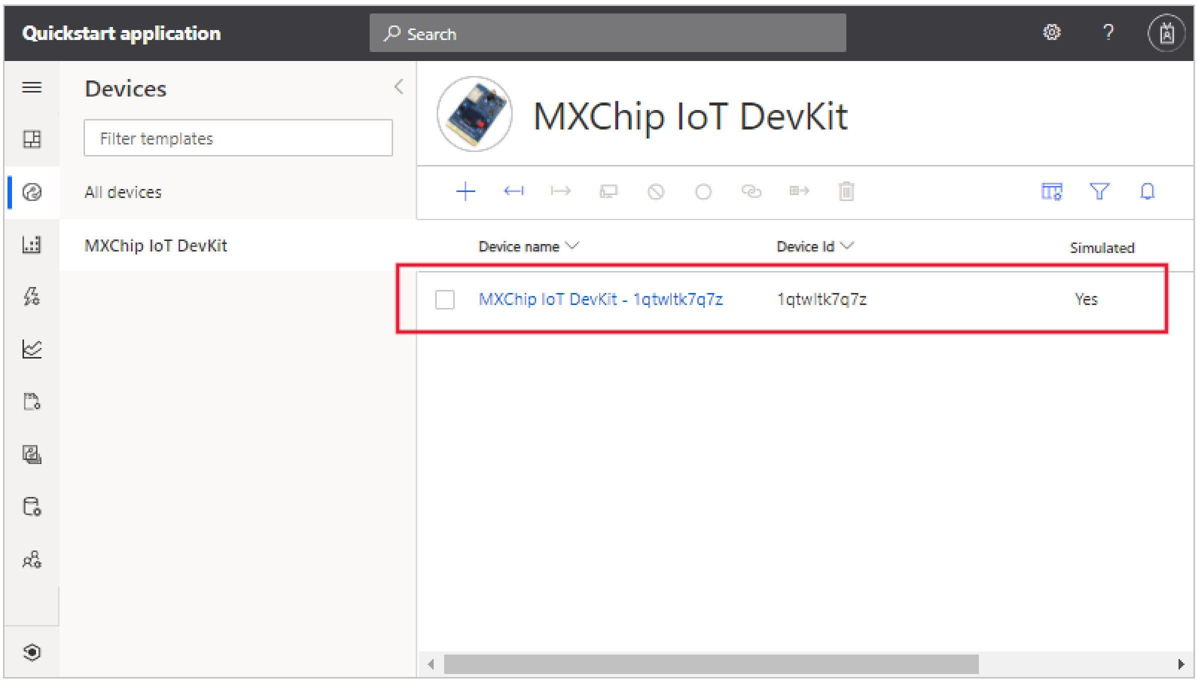
After you publish a device template, it's visible on the **Devices** page. In a published device template, you can't edit a device capability model without creating a new version. However, you can make updates to cloud properties, customizations, and views, in a published device template without versioning. After making any changes, select **Publish** to push those changes out to your operator.

Add a simulated device

To add a simulated device to your application, you use the **MXChip IoT DevKit** device template you created.

1. To add a new device as an operator choose **Devices** in the left pane. The **Devices** tab shows **All devices** and the **MXChip IoT DevKit** device template. Select **MXChip IoT DevKit**.

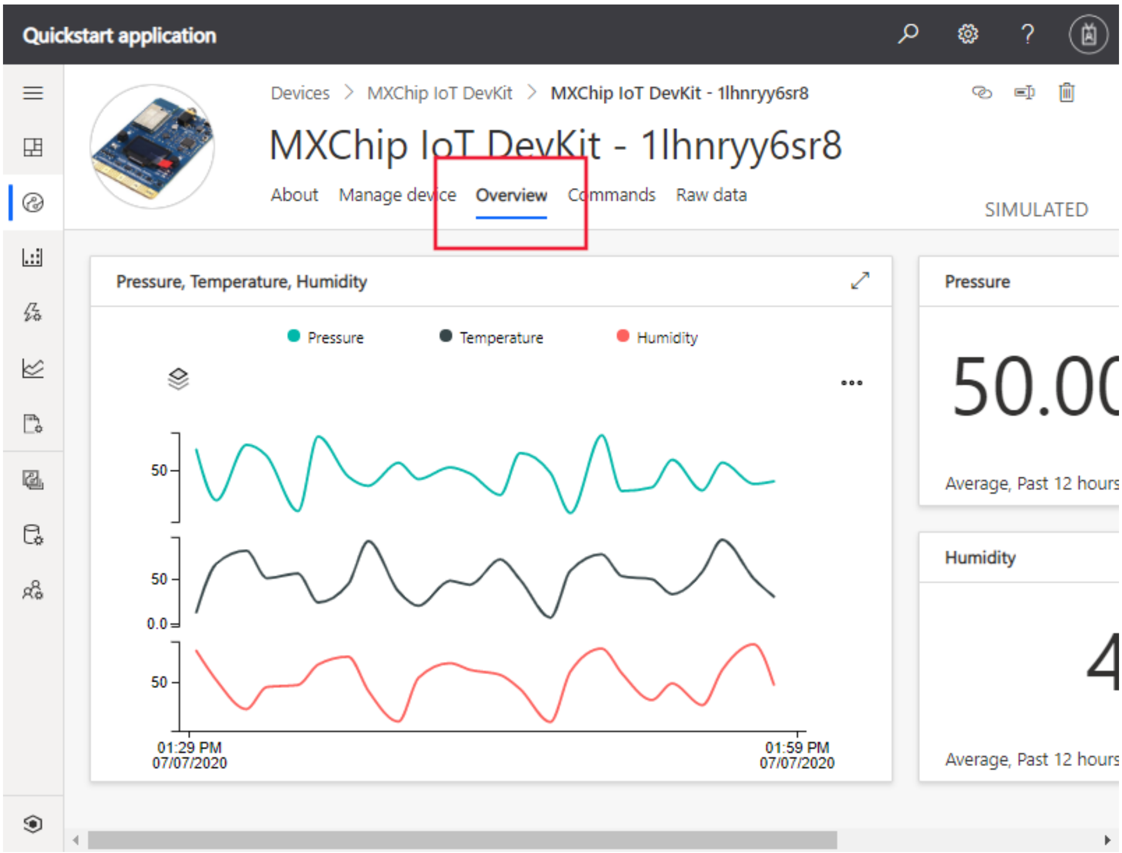
2. To add a simulated DevKit device, select **+**. Use the suggested **Device ID** or enter your own lowercase **Device ID**. You can also enter a name for your new device. Make sure the **Simulated** toggle is **On** and then select **Create**.



Now you can interact with the views that were created by the builder for the device template using simulated data:

1. Select your simulated device on the **Devices** page

- The **Overview** view shows a plot of the simulated telemetry:



- The **About** view shows property values, including the cloud properties you added to the view.
- The **Commands** view lets you run commands, such as **blink** on the device.
- The **Manage devices** view is the form you created for the operator to manage the device.
- The **Raw data** view lets you view the raw telemetry and property values sent by the device. This view is useful for debugging devices.

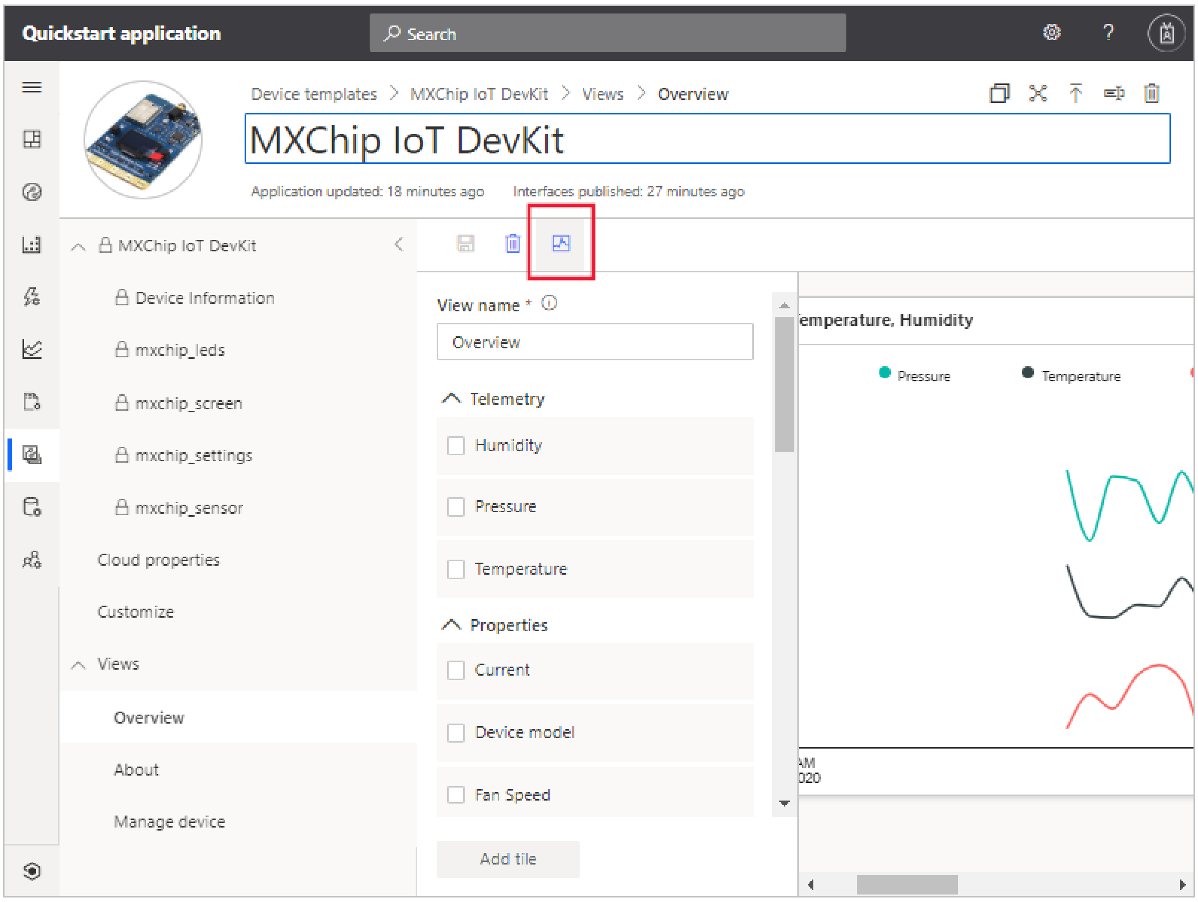
Use a simulated device to improve views

After you create a new simulated device, the builder can use this device to continue to improve and build upon the views for the device template.

1. Choose **Device templates** in the left pane and select the **MXChip IoT DevKit** template.

2. Select any of the views you would like to edit, or create a new view. Select **Configure preview device**, then **Select from a running device**. Here you can choose to have no preview device, a real device configured for testing, or an existing device you've added into IoT Central.

3. Choose your simulated device in the list. Then select **Apply**. Now you can see the same simulated device in your device template views building experience. This view is useful for charts and other visualizations.



Next steps

In this quickstart, you learned how to you create an **MXChip IoT DevKit** device template and add a simulated device to your application.

To learn more about monitoring devices connected to your application, continue to the quickstart:

Configure rules and actions

Quickstart: Configure rules and actions for your device in Azure IoT Central

4/9/2020 • 2 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

In this quickstart, you create a rule that sends an email when the temperature reported by a device sensor exceeds 90° F.

Prerequisites

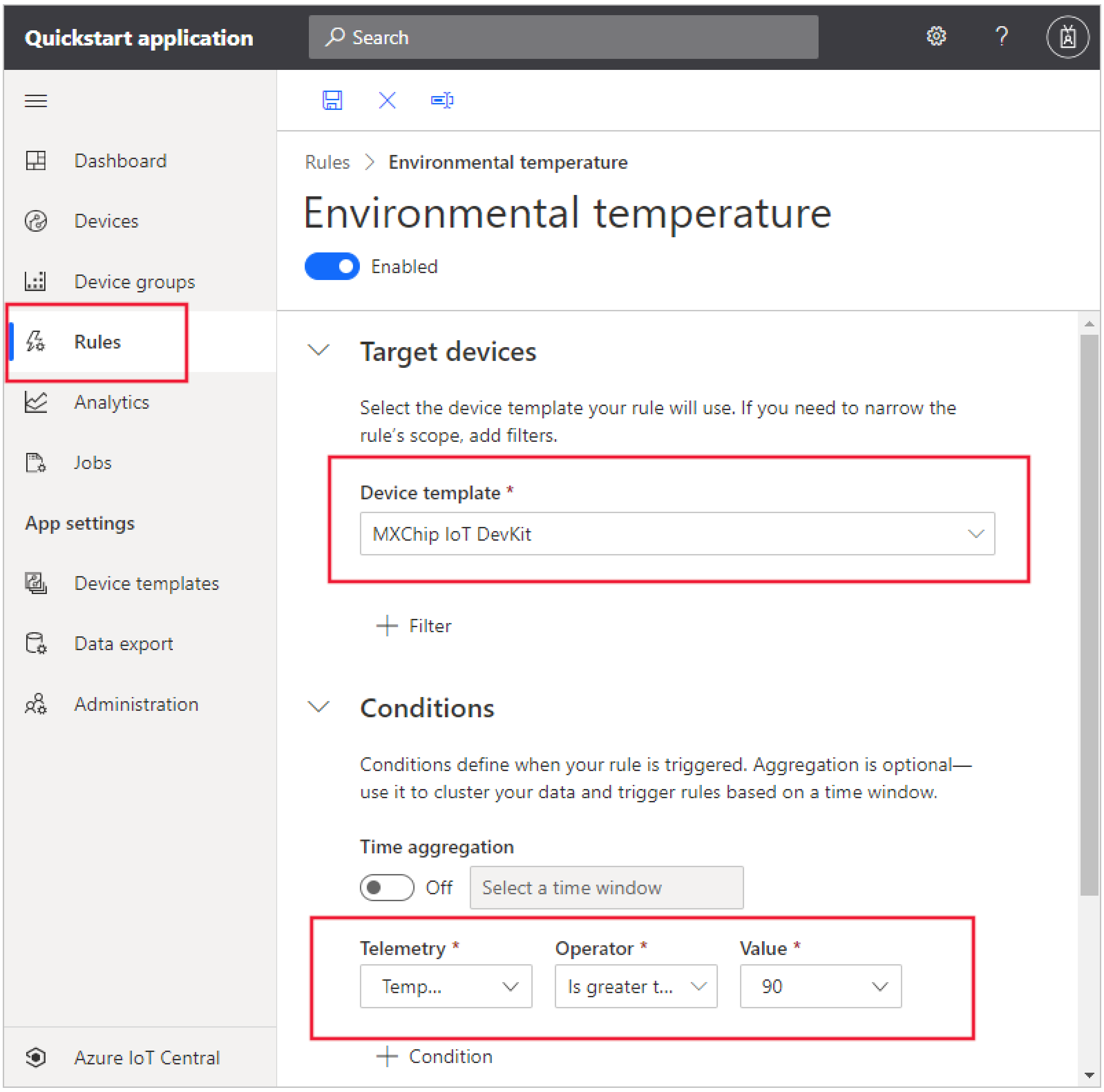
Before you begin, you should complete the two previous quickstarts Create an Azure IoT Central application and Add a simulated device to your IoT Central application to create the **MXChip IoT DevKit** device template to work with.

Create a telemetry-based rule

1. To add a new telemetry-based rule to your application, in the left pane, select **Rules**.
2. To create a new rule, select **+**.
3. Enter **Environmental temperature** as the rule name.
4. In the **Target devices** section, select **MXChip IoT DevKit** as the device template. This option filters the devices the rule applies to by device template type. You can add more filter criteria by selecting **+ Filter**.
5. In the **Conditions** section, you define what triggers your rule. Use the following information to define a condition based on temperature telemetry:

FIELD	VALUE
Measurement	Temperature
Operator	is greater than
Value	90

   To add more conditions, select **+ Condition**.



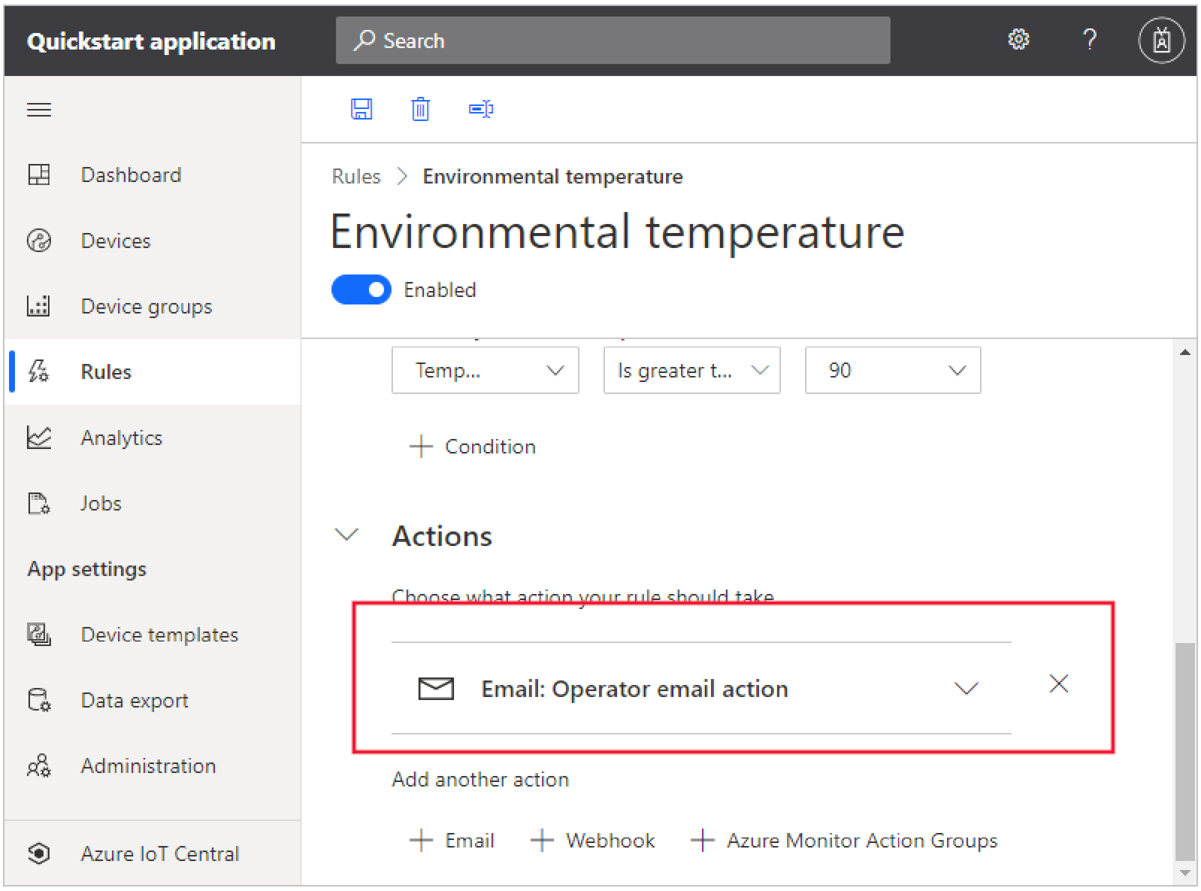
6. To add an email action to run when the rule triggers, select **+ Email**.

7. Use the information in the following table to define your action and then select **Done**:

SETTING	VALUE
Display name	Operator email action
To	Your email address
Notes	Environmental temperature exceeded the threshold.

NOTE

To receive an email notification, the email address must be a user ID in the application, and that user must have signed in to the application at least once.



8. Select **Save**. Your rule is listed on the **Rules** page.

Test the rule

Shortly after you save the rule, it becomes live. When the conditions defined in the rule are met, your application sends a message to the email address you specified in the action.

> **NOTE**
> After your testing is complete, turn off the rule to stop receiving alerts in your inbox.

Next steps

In this quickstart, you learned how to:

- Create a telemetry-based rule
- Add an action

To learn more about monitoring devices connected to your application, continue to the quickstart:

Use Azure IoT Central to monitor your devices.

Quickstart: Use Azure IoT Central to monitor your devices

4/9/2020 • 2 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

This quickstart shows you, as an operator, how to use your Microsoft Azure IoT Central application to monitor your devices and change settings.

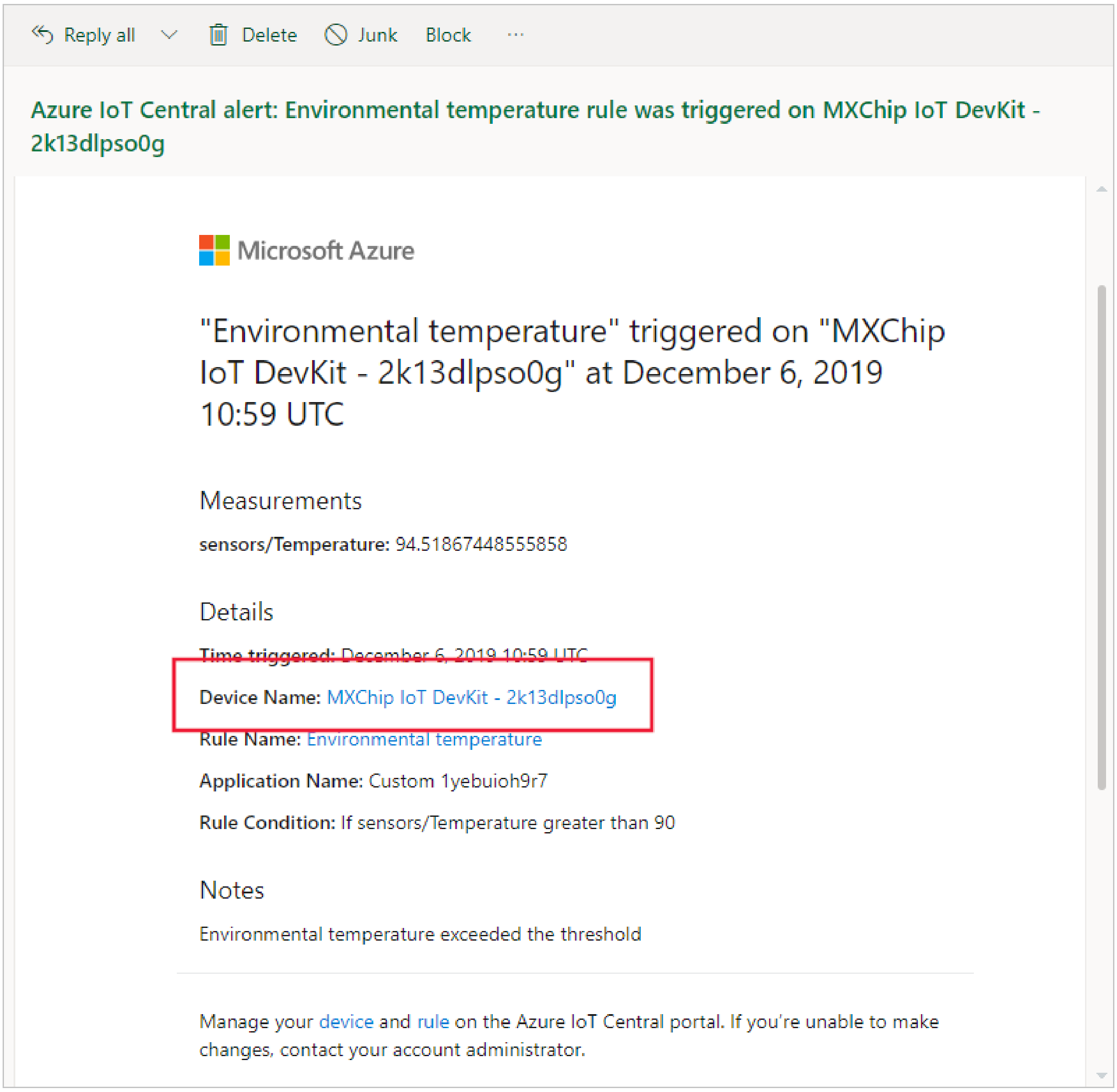
Prerequisites

Before you begin, you should complete the three previous quickstarts Create an Azure IoT Central application, Add a simulated device to your IoT Central application and Configure rules and actions for your device.

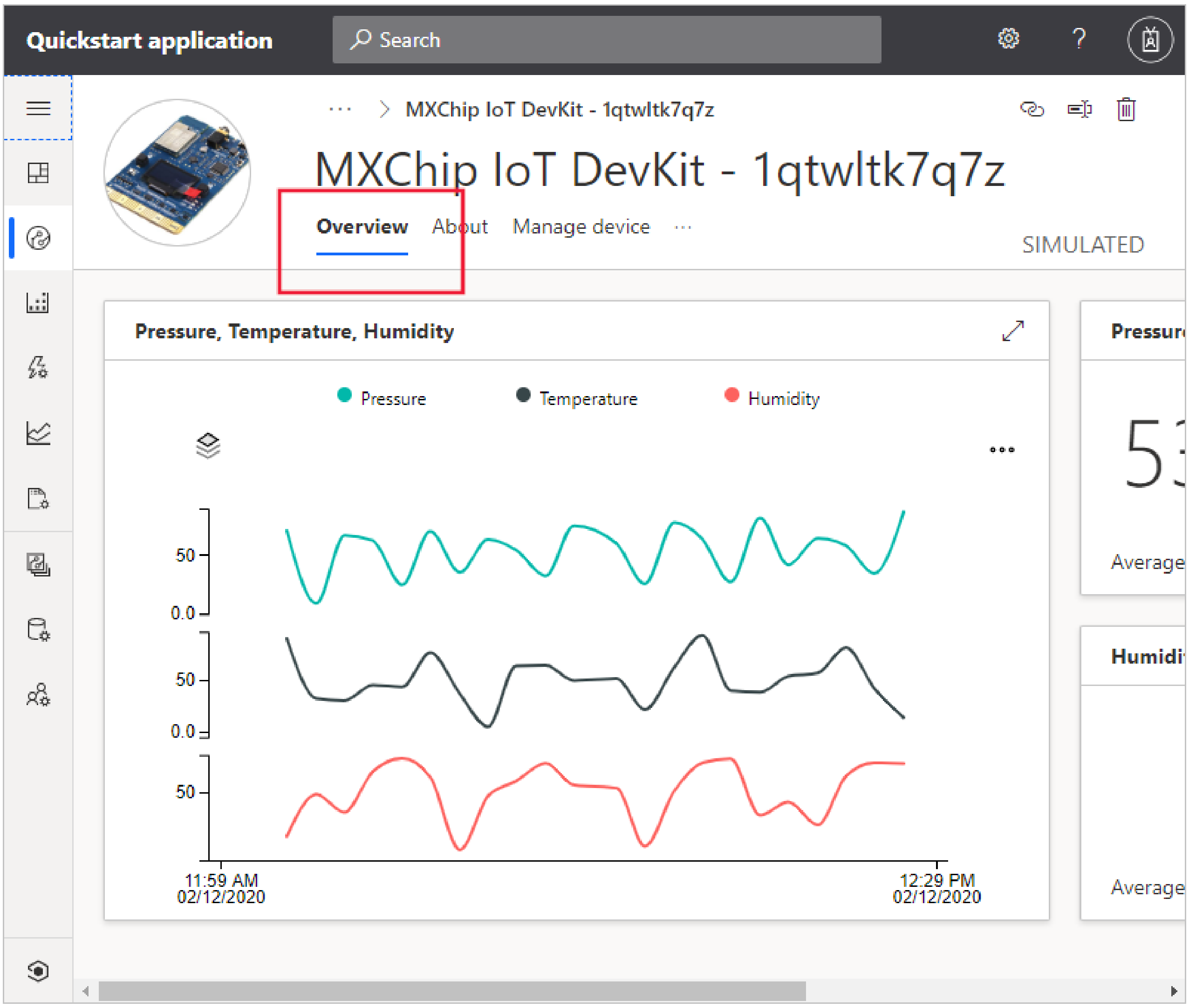
Receive a notification

Azure IoT Central sends notifications about devices as email messages. The builder added a rule to send a notification when the temperature in a connected device sensor exceeded a threshold. Check the emails sent to the account the builder chose to receive notifications.

Open the email message you received at the end of the Configure rules and actions for your device quickstart. In the email, select the link to the device:



The **Overview** view for the simulated device you created in the previous quickstarts opens in your browser:



Investigate an issue

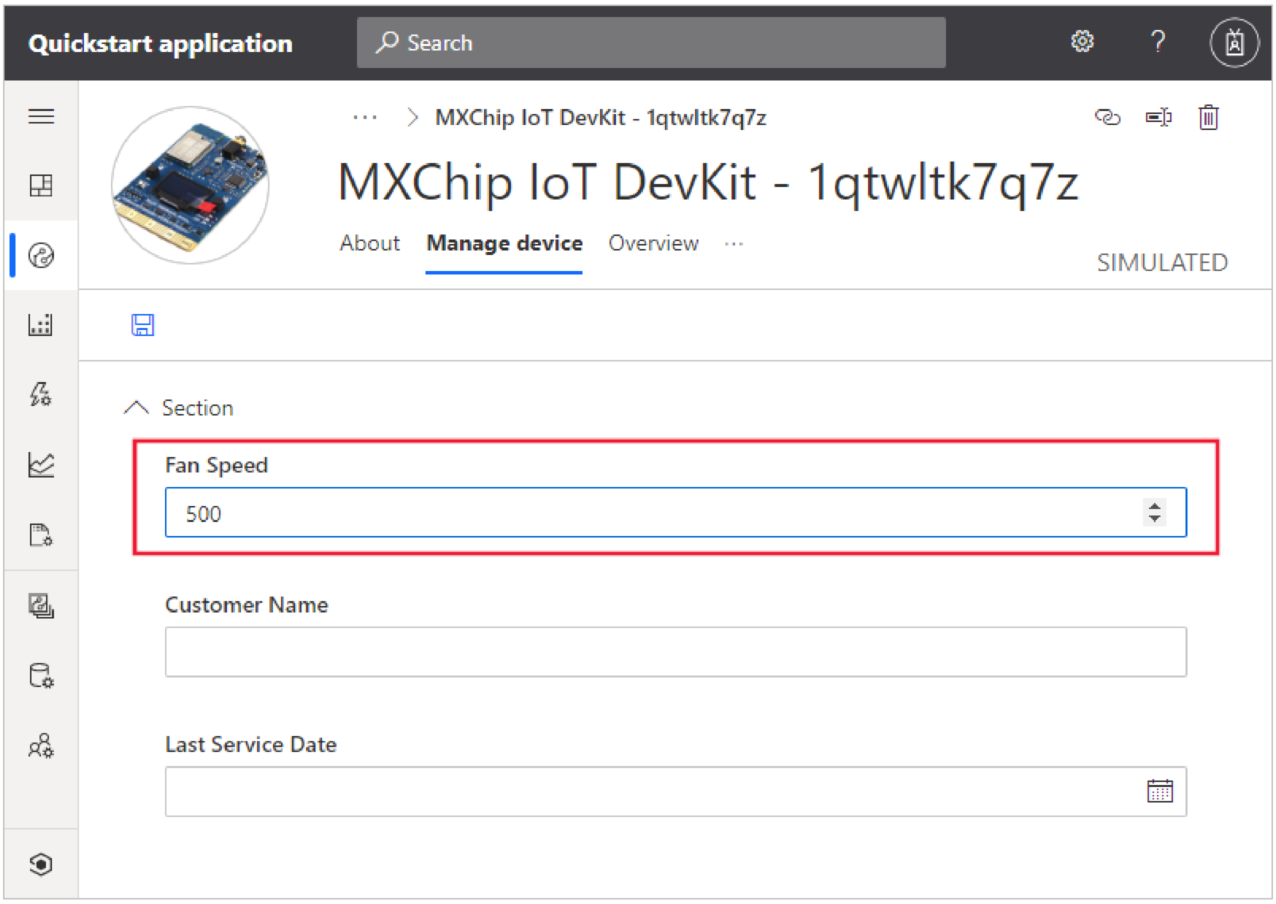
As an operator, you can view information about the device on the **Overview**, **About**, and **Commands** views. The builder created a **Manage device** view for you to edit device information and set device properties.

The chart on the dashboard shows a plot of the device temperature. You decide that the device temperature is too high.

Remediate an issue

To make a change to the device, use the **Manage device** page.

Change **Fan Speed** to 500 to cool the device. Choose **Save** to update the device. When the device confirms the settings change, the status of the property changes to **synced**:



Next steps

In this quickstart, you learned how to:

- Receive a notification
- Investigate an issue
- Remediate an issue

Now that you know now to monitor your device, the suggested next step is to:

Build and manage a device template.

Tutorial: Create and connect a client application to your Azure IoT Central application (Node.js)

7/21/2020 • 11 minutes to read • Edit Online

This article applies to solution builders and device developers.

This tutorial shows you how, as a device developer, to connect a Node.js client application to your Azure IoT Central application. The Node.js application simulates the behavior of an environmental sensor device. You use a sample *device capability model* to create a *device template* in IoT Central. You add views to the device template to enable an operator to interact with a device.

In this tutorial, you learn how to:

- Import a device capability model to create a device template.
- Add default and custom views to a device template.
- Publish a device template and add a real device to your IoT Central application.
- Create and run the Node.js device code and see it connect to your IoT Central application.
- View the simulated telemetry sent from the device.
- Use a view to manage device properties.
- Call synchronous and asynchronous commands to control the device.

Prerequisites

To complete the steps in this article, you need the following:

- An Azure IoT Central application created using the **Custom application** template. For more information, see the create an application quickstart. The application must have been created on or after 07/14/2020.
- A development machine with Node.js version 10.0.0 or later installed. You can run `node --version` in the command line to check your version. The instructions in this tutorial assume you're running the **node** command at the Windows command prompt. However, you can use Node.js on many other operating systems.

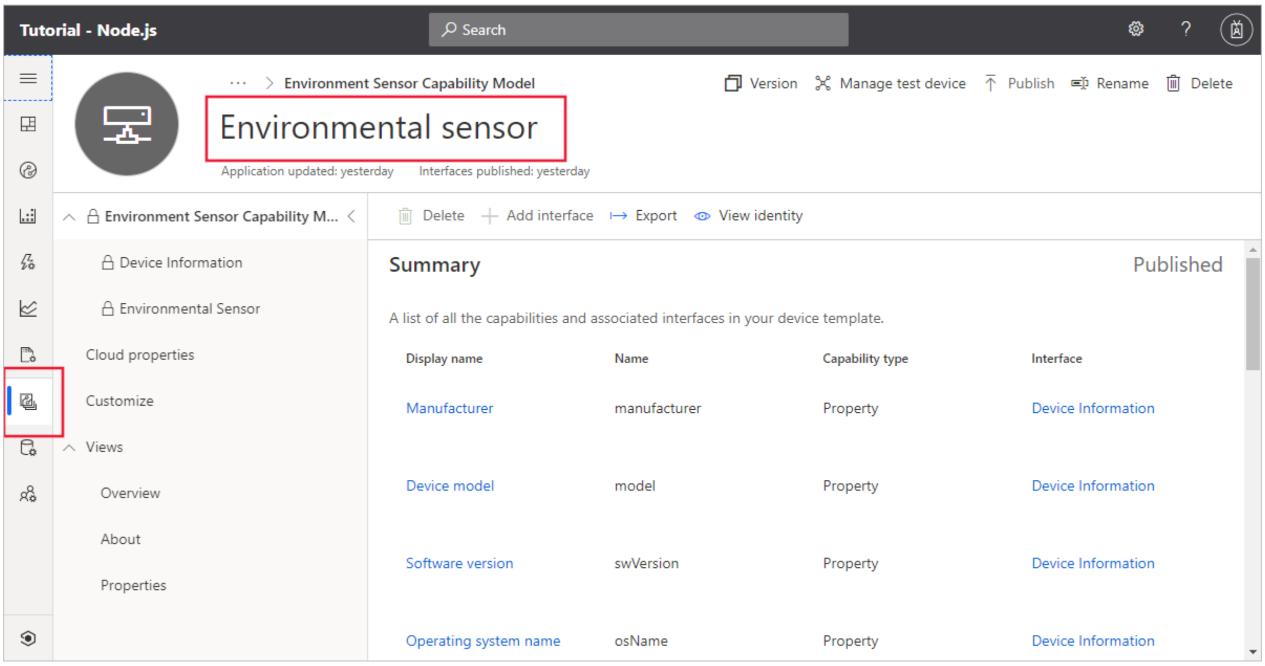
Create a device template

Create a folder called `environmental-sensor` on your local machine.

Download the Environmental sensor capability model JSON file and save it in the `environmental-sensor` folder.

Use a text editor to replace the two instances of `{YOUR_COMPANY_NAME_HERE}` with your company name in the `EnvironmentalSensorInline.capabilitymodel.json` file you downloaded. Use only the characters a-z, A-Z, 0-9, and underscore.

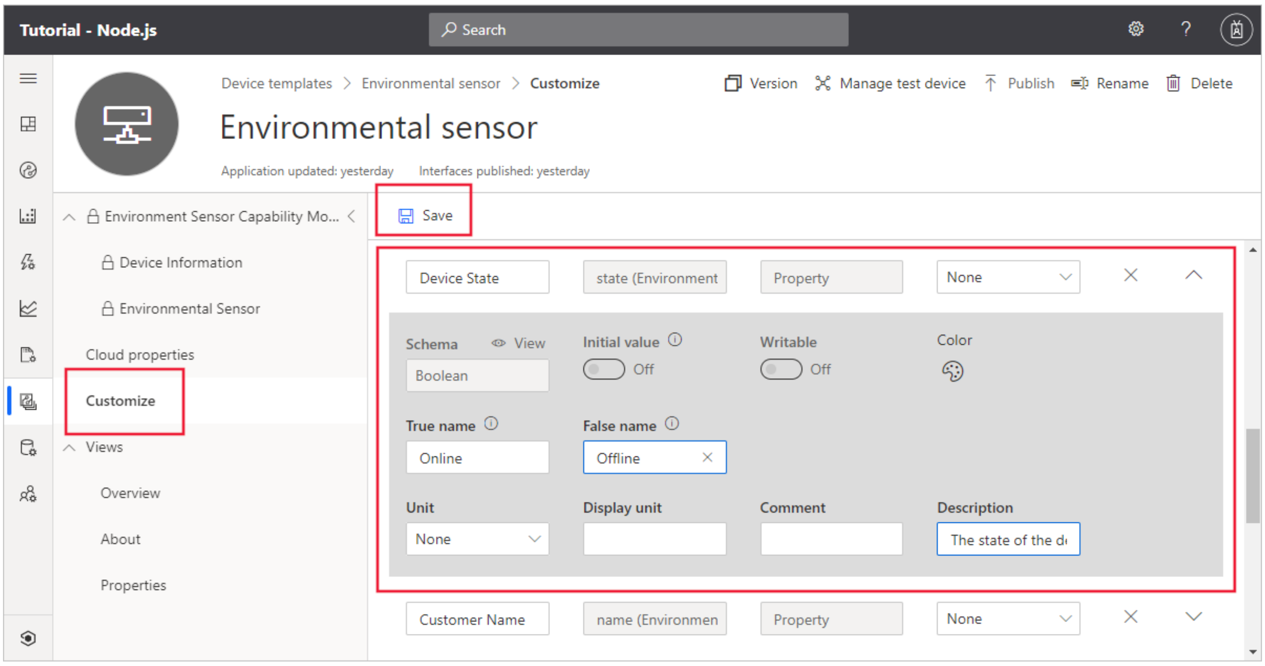
In your Azure IoT Central application, create a device template called *Environmental sensor* by importing the `EnvironmentalSensorInline.capabilitymodel.json` device capability model file:



The device capability model includes two interfaces: the standard **Device Information** interface and the custom **Environmental Sensor** interface. The **Environmental Sensor** interface defines the following capabilities:

TYPE	DISPLAY NAME	DESCRIPTION
Property	Device State	The state of the device. Two states online/offline are available.
Property (writeable)	Customer Name	The name of the customer currently operating the device.
Property (writeable)	Brightness Level	The brightness level for the light on the device. Can be specified as 1 (high), 2 (medium), 3 (low).
Telemetry	Temperature	Current temperature detected by the device.
Telemetry	Humidity	Current humidity detected by the device.
Command	blink	Begin blinking the LED on the device for given time interval.
Command	turnon	Turn on the LED on the device.
Command	turnoff	Turn off the LED on the device.
Command	rundiagnostics	This asynchronous command starts a diagnostics run on the device.

To customize how the **Device State** property displays in your IoT Central application, select **Customize** in the device template. Expand the **Device State** entry, enter *Online* as the **True name** and *Offline* as the **False name**. Then save the changes:



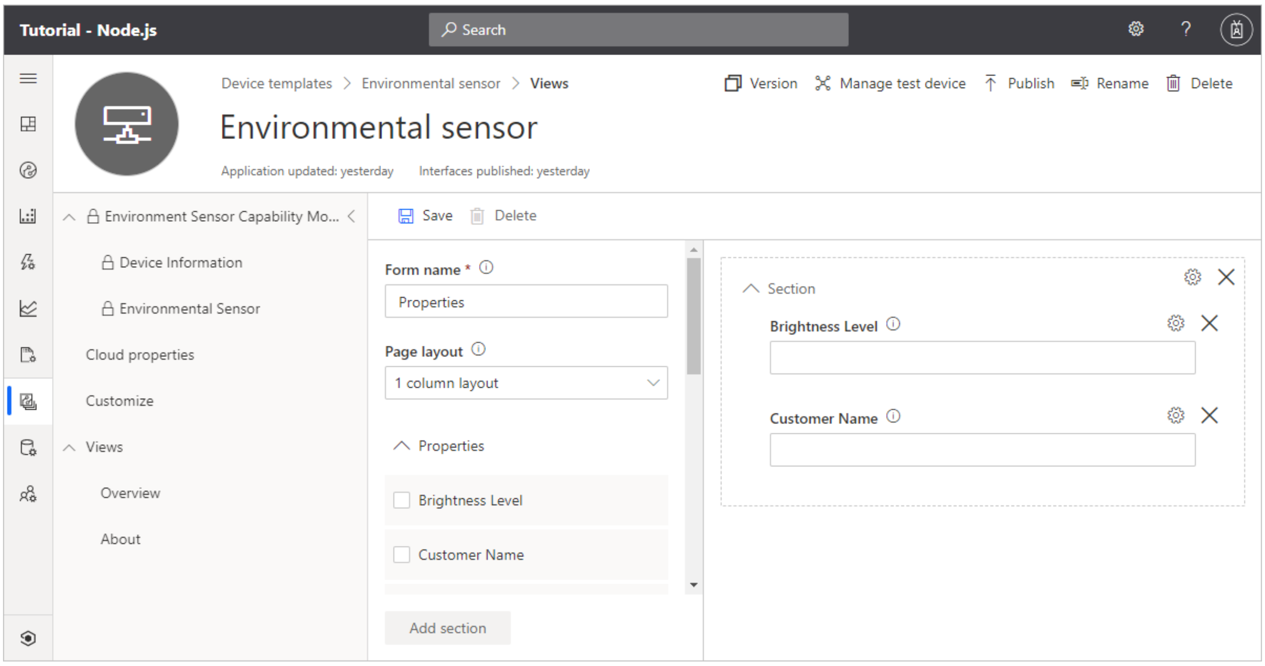
Create views

Views let you interact with devices connected to your IoT Central application. For example, you can have views that display telemetry, views that display properties, and views that let you edit writeable and cloud properties. Views are part of a device template.

To add some default views to your **Environmental sensor** device template, navigate to your device template, select **Views**, and select the **Generate Default views** tile. Make sure **Overview** and **About** are **On**, and then select **Generate default dashboard view(s)**. You now have two default views defined in your template.

The **Environmental Sensor** interface includes two writeable properties - **Customer Name** and **Brightness Level**. To create a view, you can use to edit these properties:

1. Select **Views** and then select the **Editing device and cloud data** tile.
2. Enter *Properties* as the form name.
3. Select the **Brightness Level** and **Customer Name** properties. Then select **Add section**.
4. Save your changes.

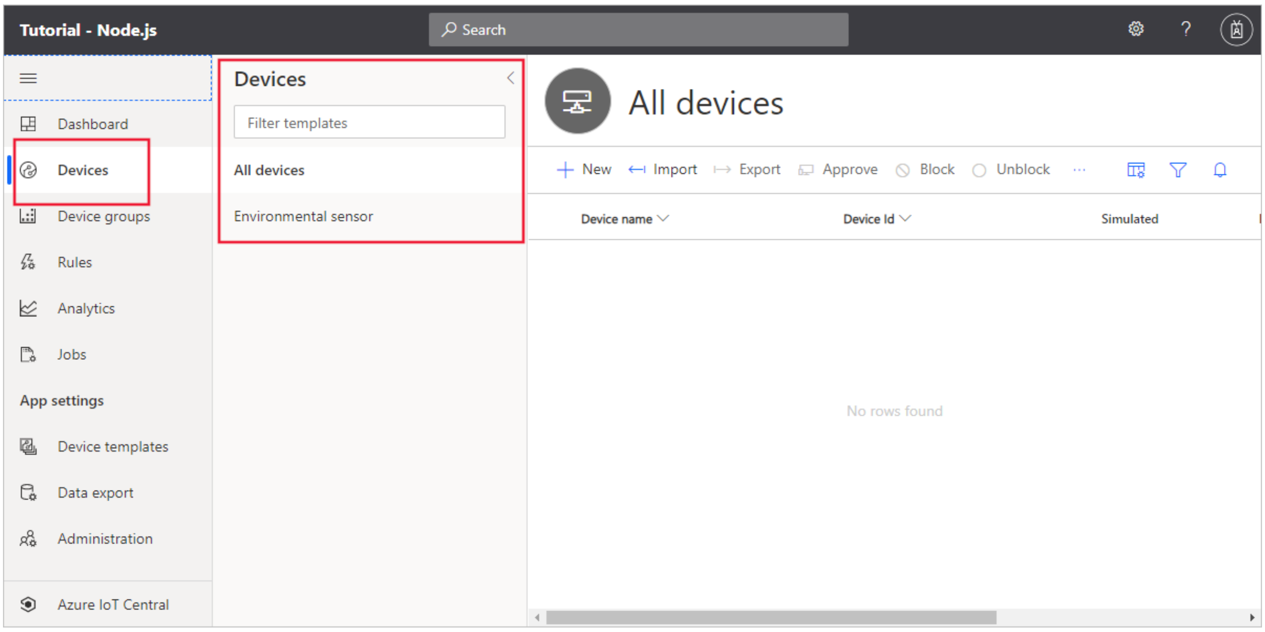


Publish the template

Before you can add a device that uses the **Environmental sensor** device template, you must publish it.

In the device template, select **Publish**. On the **Publish this device template to the application** panel, select **Publish**.

To check that the template is ready to use, navigate to the **Devices** page in your IoT Central application. The **Devices** section shows a list of the published devices in the application:

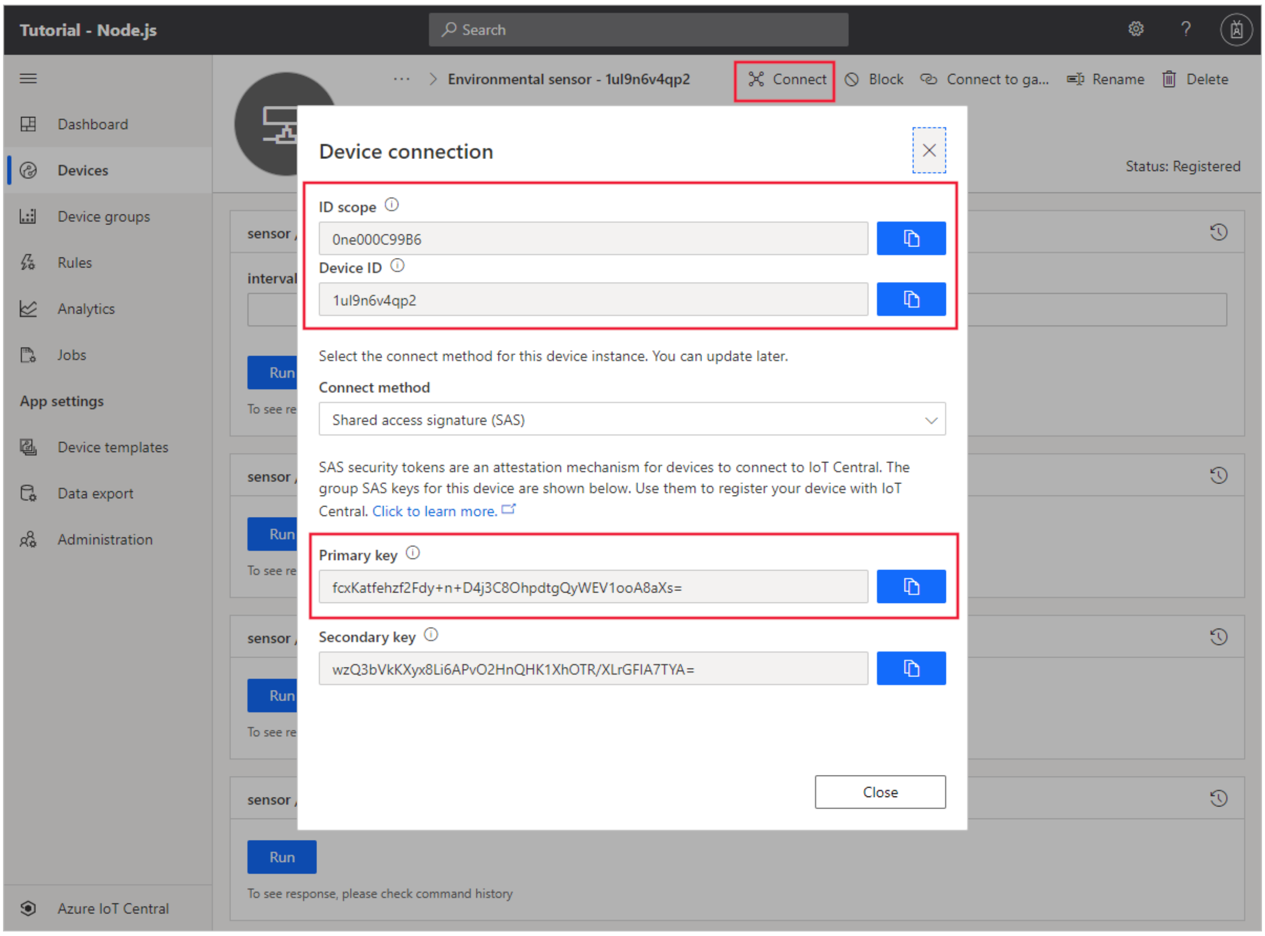


Add a real device

In your Azure IoT Central application, add a real device to the device template you created in the previous section:

1. On the **Devices** page, select the **Environmental sensor** device template.
2. Select **+ New**.
3. In the **Create a new device** dialog, make sure that **Environmental Sensor** is the template type and that **Simulate this device?** is set to **No**.
4. Then select **Create**.

Click on the device name, and then select **Connect**. Make a note of the device connection information on the **Device Connection** page - **ID scope**, **Device ID**, and **Primary key**. You need these values when you create your device code:



Create a Node.js application

The following steps show you how to create a Node.js client application that connects to the real device you added to the application. This Node.js application simulates the behavior of a real device.

1. In your command-line environment, navigate to the `environmental-sensor` folder you created previously.

2. To initialize your Node.js project and install the required dependencies, run the following commands - accept all the default options when you run `npm init`:

```
npm init
npm install azure-iot-device azure-iot-device-mqtt azure-iot-provisioning-device-mqtt azure-iot-
security-symmetric-key --save
```

3. Create a file called **environmentalSensor.js** in the `environmental-sensor` folder.

4. Add the following `require` statements at the start of the **environmentalSensor.js** file:

```
"use strict";

// Use the Azure IoT device SDK for devices that connect to Azure IoT Central.
var iotHubTransport = require('azure-iot-device-mqtt').Mqtt;
var Client = require('azure-iot-device').Client;
var Message = require('azure-iot-device').Message;
var ProvisioningTransport = require('azure-iot-provisioning-device-mqtt').Mqtt;
var SymmetricKeySecurityClient = require('azure-iot-security-symmetric-
key').SymmetricKeySecurityClient;
var ProvisioningDeviceClient = require('azure-iot-provisioning-device').ProvisioningDeviceClient;
```

5. Add the following variable declarations to the file:

```
var provisioningHost = 'global.azure-devices-provisioning.net';
var idScope = '{your Scope ID}';
var registrationId = '{your Device ID}';
var symmetricKey = '{your Primary Key}';
var provisioningSecurityClient = new SymmetricKeySecurityClient(registrationId, symmetricKey);
var provisioningClient = ProvisioningDeviceClient.create(provisioningHost, idScope, new
ProvisioningTransport(), provisioningSecurityClient);
var hubClient;

var targetTemperature = 0;
var ledOn = true;
```

Update the placeholders `{your Scope ID}`, `{your Device ID}`, and `{your Primary Key}` with the values you made a note of previously. In this sample, you initialize `targetTemperature` to zero, you could use the current reading from the device or a value from the device twin.

6. To send simulated telemetry to your Azure IoT Central application, add the following function to the file:

```
// Send simulated device telemetry.
function sendTelemetry() {
  var temp = targetTemperature + (Math.random() * 15);
  var humid = 70 + (Math.random() * 10);
  var data = JSON.stringify({
    temp: temp,
    humid: humid,
    });
  var message = new Message(data);
  hubClient.sendEvent(message, (err, res) => console.log(`Sent message: ${message.getData()}` +
    (err ? `; error: ${err.toString()}` : '') +
    (res ? `; status: ${res.constructor.name}` : '')));
}
```

The names of the telemetry items (`temp` and `humid`) must match the names used in the device template.

7. To send device twin properties to your Azure IoT Central application, add the following function to your file:

```
// Send device twin reported properties.
function sendDeviceProperties(twin, properties) {
  twin.properties.reported.update(properties, (err) => console.log(`Sent device properties:
${JSON.stringify(properties)}; ` +
    (err ? `error: ${err.toString()}` : `status: success`)));
}
```

IoT Central uses device twins to synchronize property values between the device and the IoT Central application. Device property values use device twin reported properties. Writeable properties use both device twin reported and desired properties.

8. To define and handle the writeable properties your device responds to, add the following code:

```
// Add any writeable properties your device supports,
// mapped to a function that's called when the writeable property
// is updated in the IoT Central application.
var writeableProperties = {
  'name': (newValue, callback) => {
      setTimeout(() => {
        callback(newValue, 'completed', 200);
      }, 1000);
  },
  'brightness': (newValue, callback) => {
    setTimeout(() => {
        callback(newValue, 'completed', 200);
    }, 5000);
  }
};

// Handle writeable property updates that come from IoT Central via the device twin.
function handleWriteablePropertyUpdates(twin) {
  twin.on('properties.desired', function (desiredChange) {
    for (let setting in desiredChange) {
      if (writeableProperties[setting]) {
        console.log(`Received setting: ${setting}: ${desiredChange[setting]}`);
        writeableProperties[setting](desiredChange[setting], (newValue, status, code) => {
          var patch = {
            [setting]: {
              value: newValue,
              ad: status,
              ac: code,
              av: desiredChange.$version
            }
          }
          sendDeviceProperties(twin, patch);
        });
      }
    }
  });
}
```

When the operator sets a writeable property in the IoT Central application, the application uses a device twin desired property to send the value to the device. The device then responds using a device twin reported property. When IoT Central receives the reported property value, it updates the property view with a status of **synced**.

The names of the properties (`name` and `brightness`) must match the names used in the device template.

9. Add the following code to handle the commands sent from the IoT Central application:

```
// Setup command handlers
function setupCommandHandlers(twin) {

  // Handle synchronous LED blink command with request and response payload.
  function onBlink(request, response) {
    console.log('Received synchronous call to blink');
    var responsePayload = {
      status: 'Blinking LED every ' + request.payload  + ' seconds'
    }
    response.send(200, responsePayload, (err) => {
      if (err) {
        console.error('Unable to send method response: ' + err.toString());
      } else {
        console.log('Blinking LED every ' + request.payload  + ' seconds');
      }
    });
  }
```

```
  // Handle synchronous LED turn on command
  function turnOn(request, response) {
    console.log('Received synchronous call to turn on LED');
    if(!ledOn){
      console.log('Turning on the LED');
      ledOn = true;
    }
    response.send(200, (err) => {
      if (err) {
        console.error('Unable to send method response: ' + err.toString());
      }
    });
  }

  // Handle synchronous LED turn off command
  function turnOff(request, response) {
    console.log('Received synchronous call to turn off LED');
    if(ledOn){
      console.log('Turning off the LED');
      ledOn = false;
    }
    response.send(200, (err) => {
      if (err) {
        console.error('Unable to send method response: ' + err.toString());
      }
    });
  }

  // Handle asynchronous sensor diagnostics command with response payload.
  function diagnostics(request, response) {
    console.log('Starting asynchronous diagnostics run...');
    response.send(202, (err) => {
      if (err) {
        console.error('Unable to send method response: ' + err.toString());
      } else {
        var repetitions = 3;
        var intervalID = setInterval(() => {
          console.log('Generating diagnostics...');
          if (--repetitions === 0) {
            clearInterval(intervalID);
            var properties = {
              rundiagnostics: {
                value: 'Diagnostics run complete at ' + new Date().toLocaleString()
              }
            };
            sendDeviceProperties(twin, properties);
          }
        }, 2000);
      }
    });
  }

  hubClient.onDeviceMethod('blink', onBlink);
  hubClient.onDeviceMethod('turnon', turnOn);
  hubClient.onDeviceMethod('turnoff', turnOff);
  hubClient.onDeviceMethod('rundiagnostics', diagnostics);
}
```

The names of the commands (`blink`, `turnon`, `turnoff`, and `rundiagnostics`) must match the names used in the device template.

Currently, IoT Central doesn't use the response schema defined in the device capability model. For a synchronous command, the response payload can be any valid JSON. For an asynchronous command, the device should return a 202 response immediately, followed by reported property update when the work is finished. The format of the reported property update is:

```
{
  [command name] : {
    value: 'response message'
  }
}
```

An operator can view the response payload in the command history.

10. Add the following code to complete the connection to Azure IoT Central and hook up the functions in the client code:

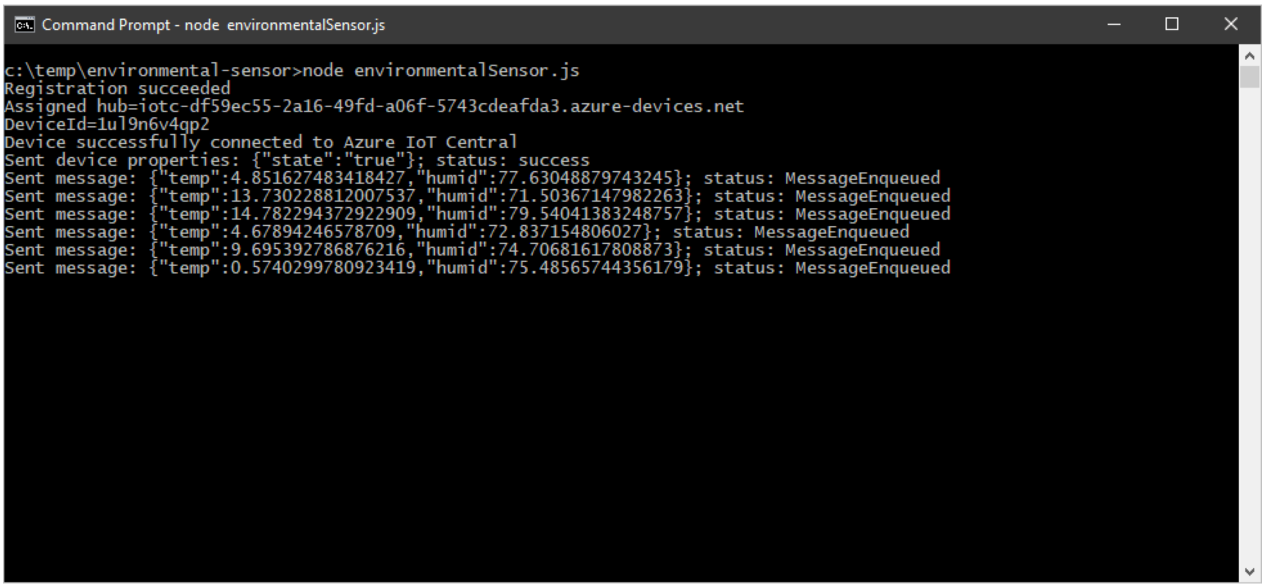
```
// Handle device connection to Azure IoT Central.
var connectCallback = (err) => {
  if (err) {
    console.log(`Device could not connect to Azure IoT Central: ${err.toString()}`);
  } else {
    console.log('Device successfully connected to Azure IoT Central');

    // Send telemetry to Azure IoT Central every 1 second.
    setInterval(sendTelemetry, 1000);

    // Get device twin from Azure IoT Central.
    hubClient.getTwin((err, twin) => {
      if (err) {
        console.log(`Error getting device twin: ${err.toString()}`);
      } else {
        // Send device properties once on device start up.
        var properties = {
          state: 'true',
          processorArchitecture: 'ARM',
          swVersion: '1.0.0'
        };
        sendDeviceProperties(twin, properties);

        handleWriteablePropertyUpdates(twin);

        setupCommandHandlers(twin);
      }
    });
  }
};

// Start the device (register and connect to Azure IoT Central).
provisioningClient.register((err, result) => {
  if (err) {
    console.log('Error registering device: ' + err);
  } else {
    console.log('Registration succeeded');
    console.log('Assigned hub=' + result.assignedHub);
    console.log('DeviceId=' + result.deviceId);
    var connectionString = 'HostName=' + result.assignedHub + ';DeviceId=' + result.deviceId +
';SharedAccessKey=' + symmetricKey;
    hubClient = Client.fromConnectionString(connectionString, iotHubTransport);

    hubClient.open(connectCallback);
  }
});
```

Run your Node.js application

To start the device client application, run the following command in your command-line environment:

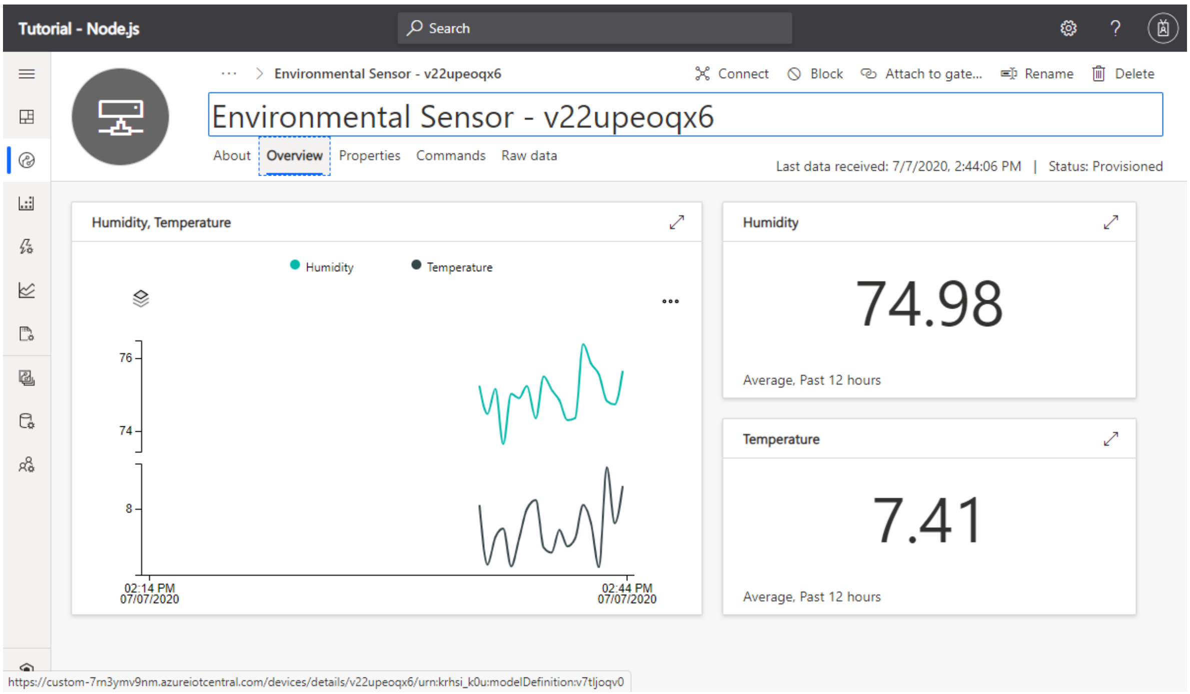
```
node environmentalSensor.js
```

You can see the device connects to your Azure IoT Central application and starts sending telemetry:

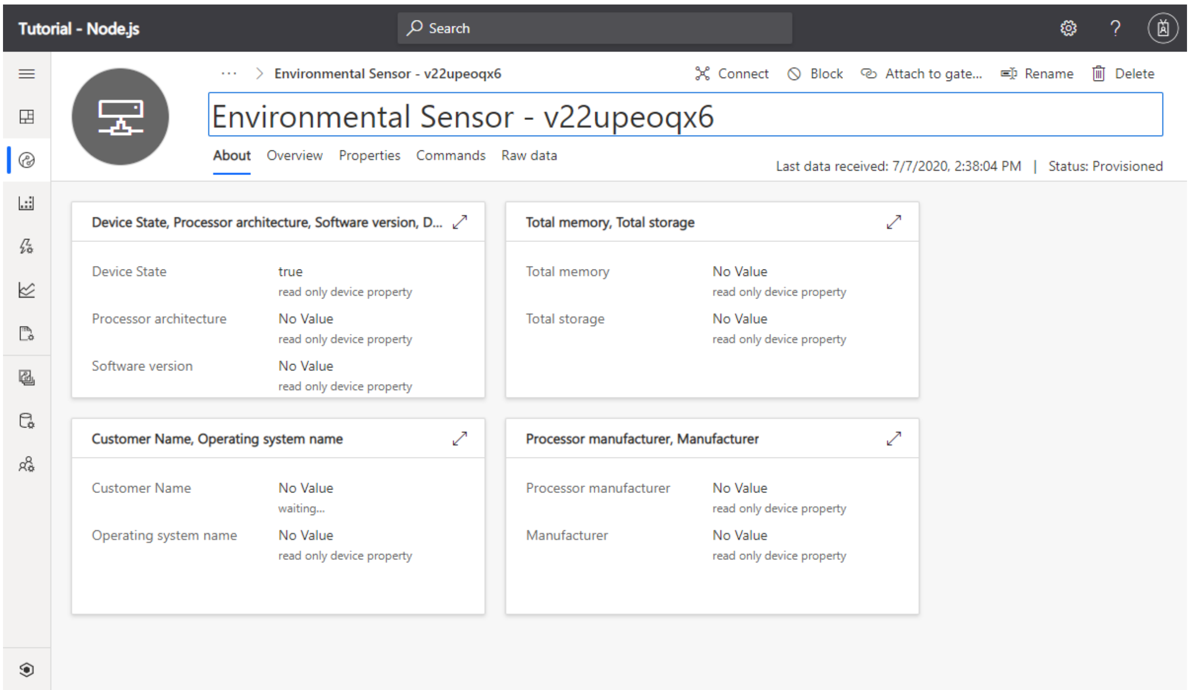


As an operator in your Azure IoT Central application, you can:

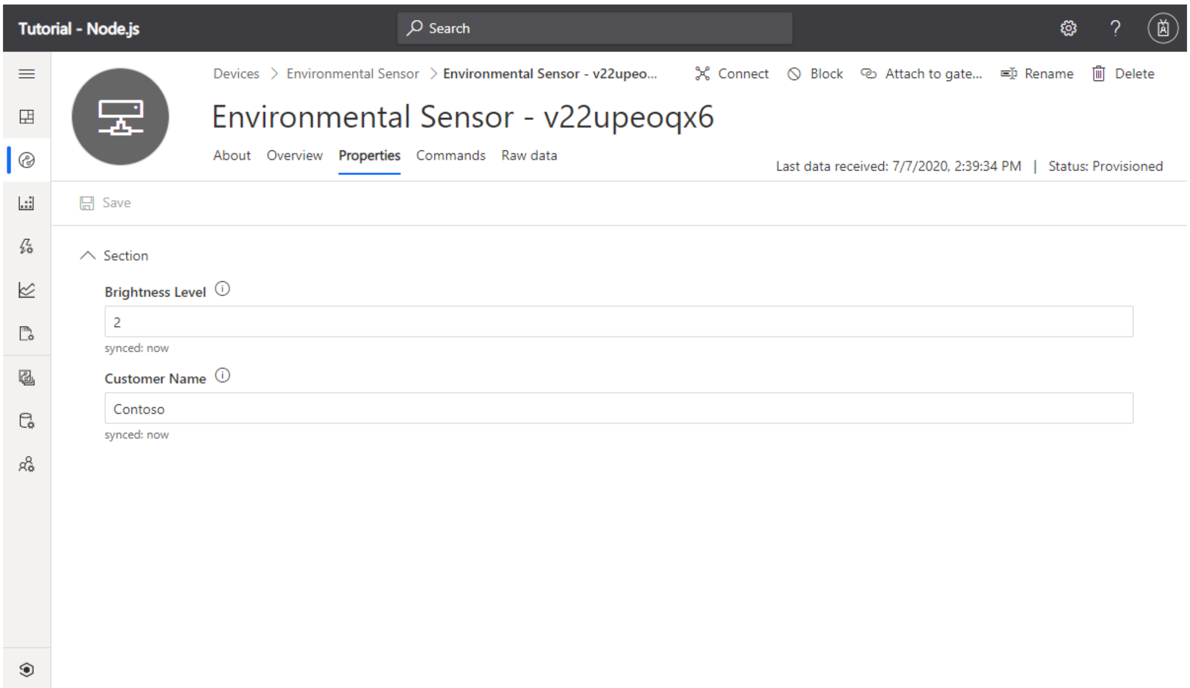
- View the telemetry sent by the device on the **Overview** page:



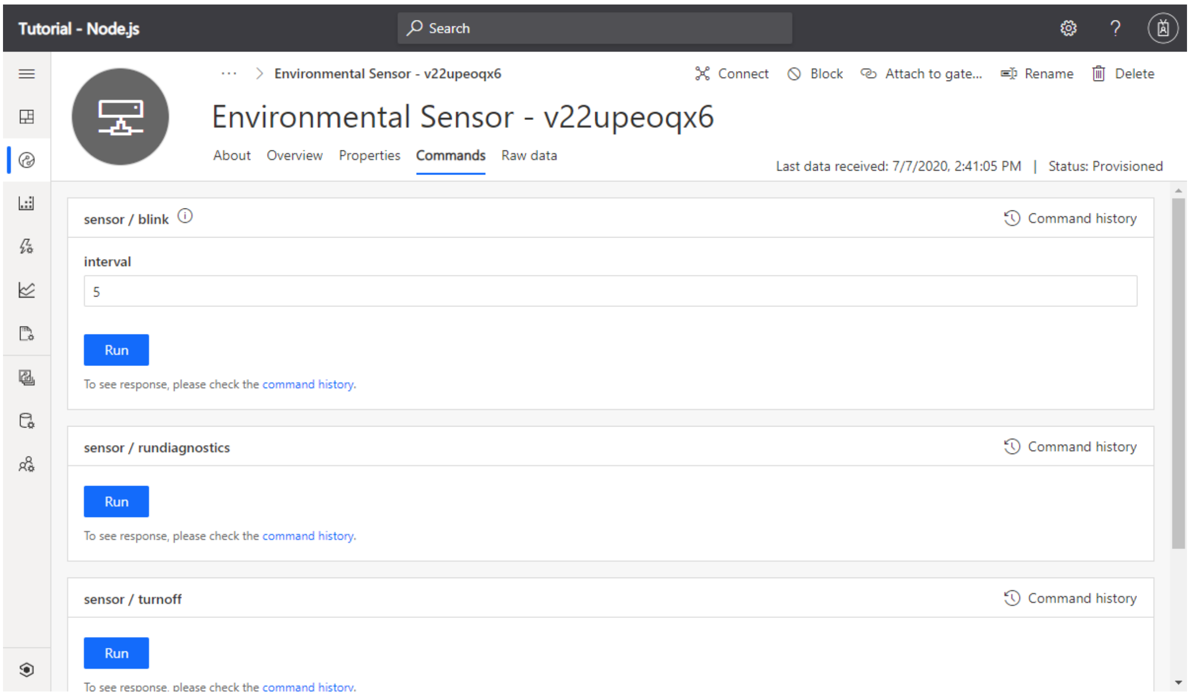
- View the device properties on the **About** page:

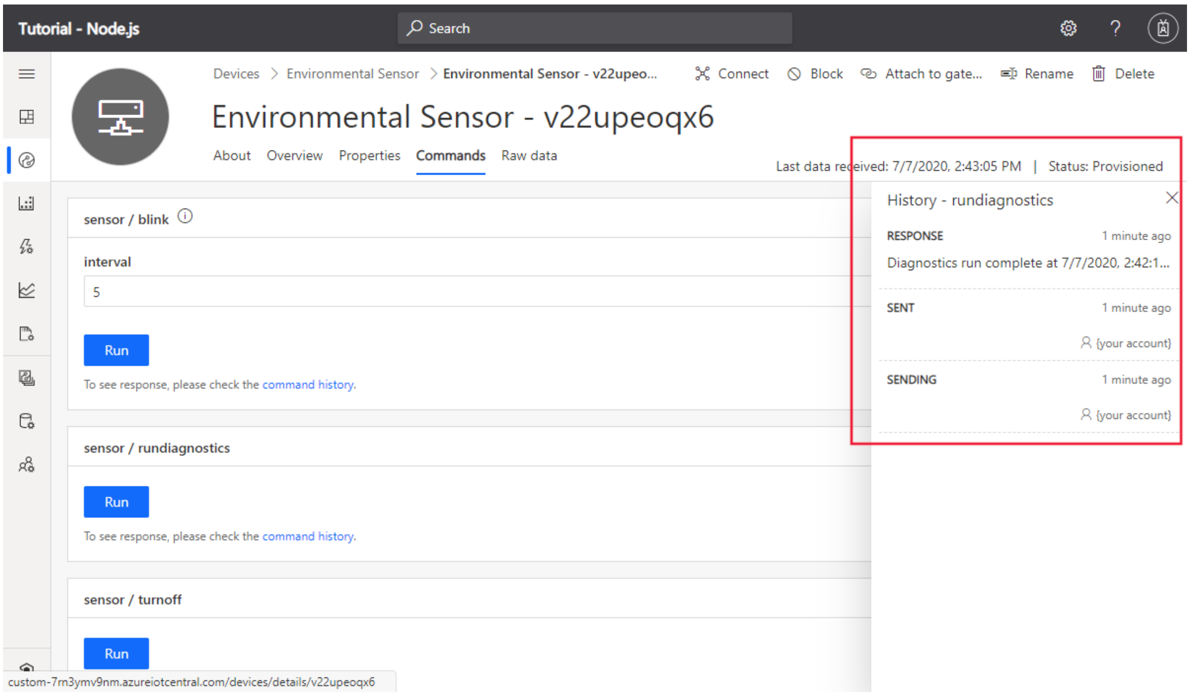


- Update writeable property values on the **Properties** page:

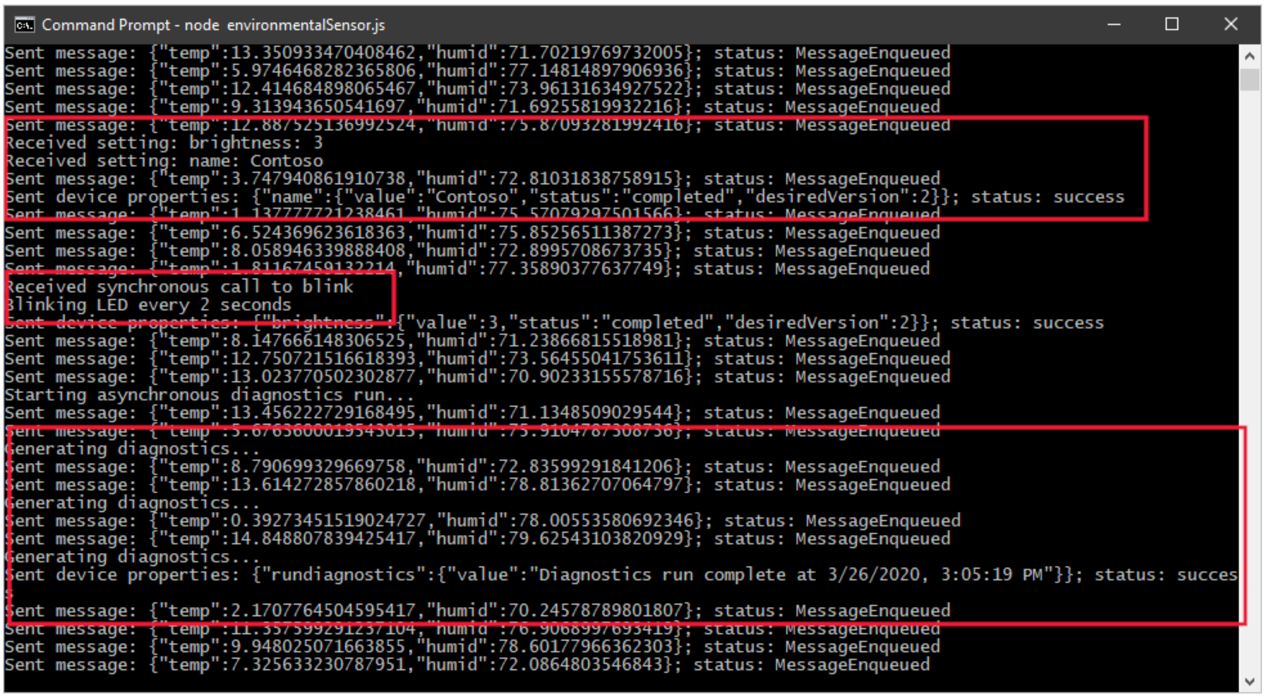


- Call the commands from the **Commands** page:



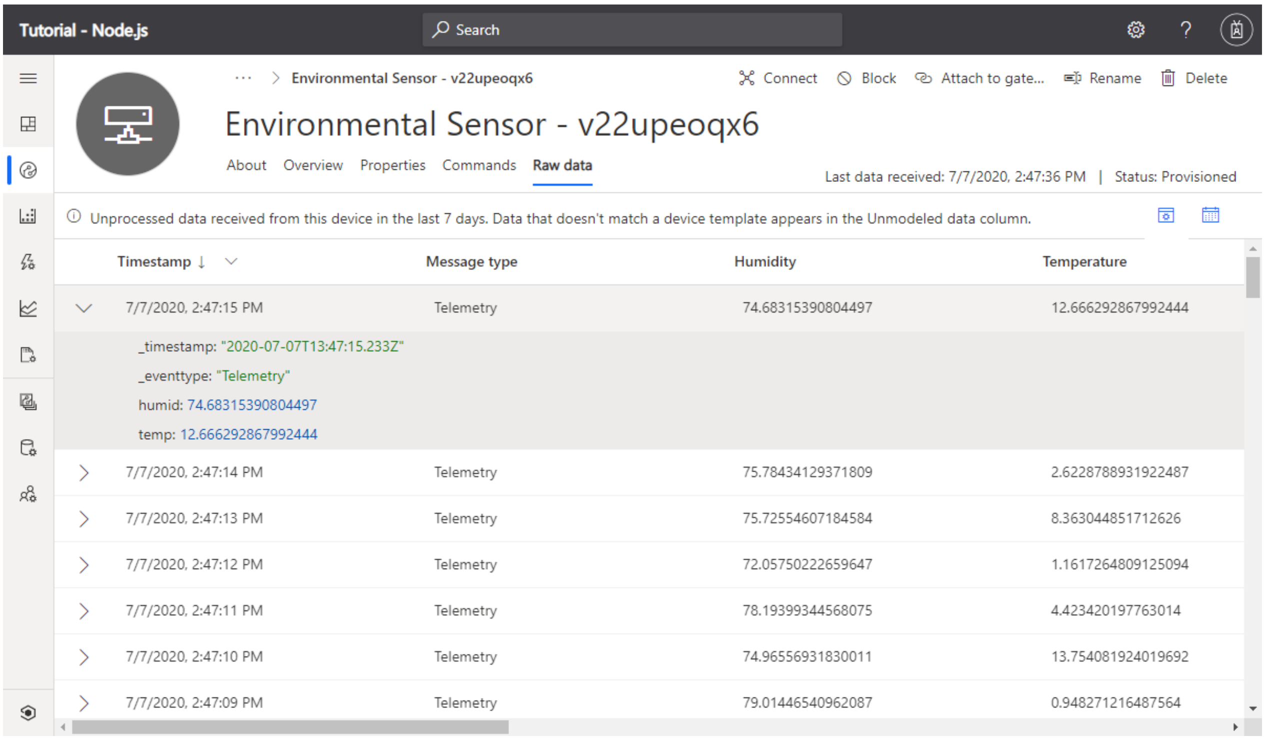


You can see how the device responds to commands and property updates:



View raw data

As a device developer, you can use the **Raw data** view to examine the raw data your device is sending to IoT Central:



On this view, you can select the columns to display and set a time range to view. The **Unmodeled data** column shows data from the device that doesn't match any property or telemetry definitions in the device template.

Next steps

As a device developer, now that you've learned the basics of how to create a device using Node.js, some suggested next steps are to:

- Learn how to connect a real device to IoT Central in the Connect an MXChip IoT DevKit device to your Azure IoT Central application how-to article.
- Read What are device templates? to learn more about the role of device templates when you're implementing

your device code.

- Read Get connected to Azure IoT Central to learn more about how to register devices with IoT Central and how IoT Central secures device connections.

If you'd prefer to continue through the set of IoT Central tutorials and learn more about building an IoT Central solution, see:

Create a gateway device template

Tutorial: Create and connect a client application to your Azure IoT Central application (Python)

7/21/2020 • 10 minutes to read • Edit Online

This article applies to solution builders and device developers.

This tutorial shows you how, as a device developer, to connect a Python client application to your Azure IoT Central application. The Python application simulates the behavior of an environmental sensor device. You use a sample *device capability model* to create a *device template* in IoT Central. You add views to the device template to enable an operator to interact with a device.

In this tutorial, you learn how to:

- Import a device capability model to create a device template.
- Add default and custom views to a device template.
- Publish a device template and add a real device to your IoT Central application.
- Create and run the Python device code and see it connect to your IoT Central application.
- View the simulated telemetry sent from the device.
- Use a view to manage device properties.
- Call synchronous and asynchronous commands to control the device.

Prerequisites

To complete the steps in this article, you need the following:

- An Azure IoT Central application created using the **Custom application** template. For more information, see the create an application quickstart. The application must have been created on or after 07/14/2020.
- A development machine with Python version 3.7 or later installed. You can run `python3 --version` at the command line to check your version. Python is available for a wide variety of operating systems. The instructions in this tutorial assume you're running the **python3** command at the Windows command prompt.

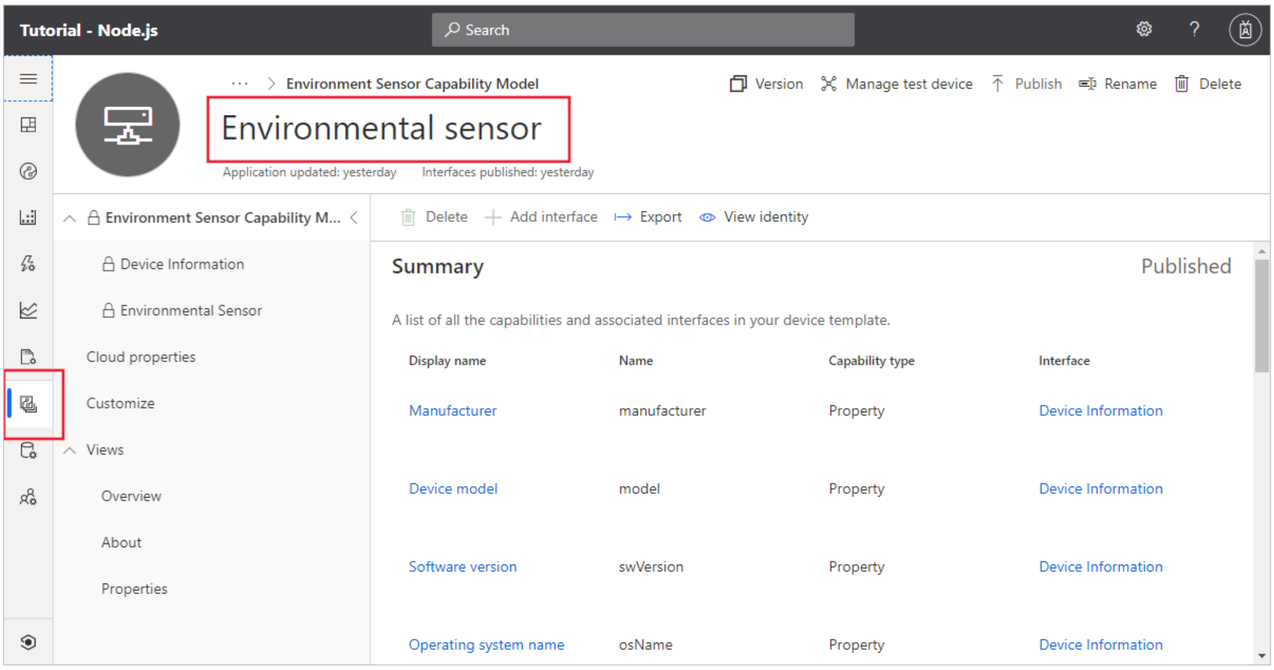
Create a device template

Create a folder called `environmental-sensor` on your local machine.

Download the Environmental sensor capability model JSON file and save it in the `environmental-sensor` folder.

Use a text editor to replace the two instances of `{YOUR_COMPANY_NAME_HERE}` with your company name in the `EnvironmentalSensorInline.capabilitymodel.json` file you downloaded. Use only the characters a-z, A-Z, 0-9, and underscore.

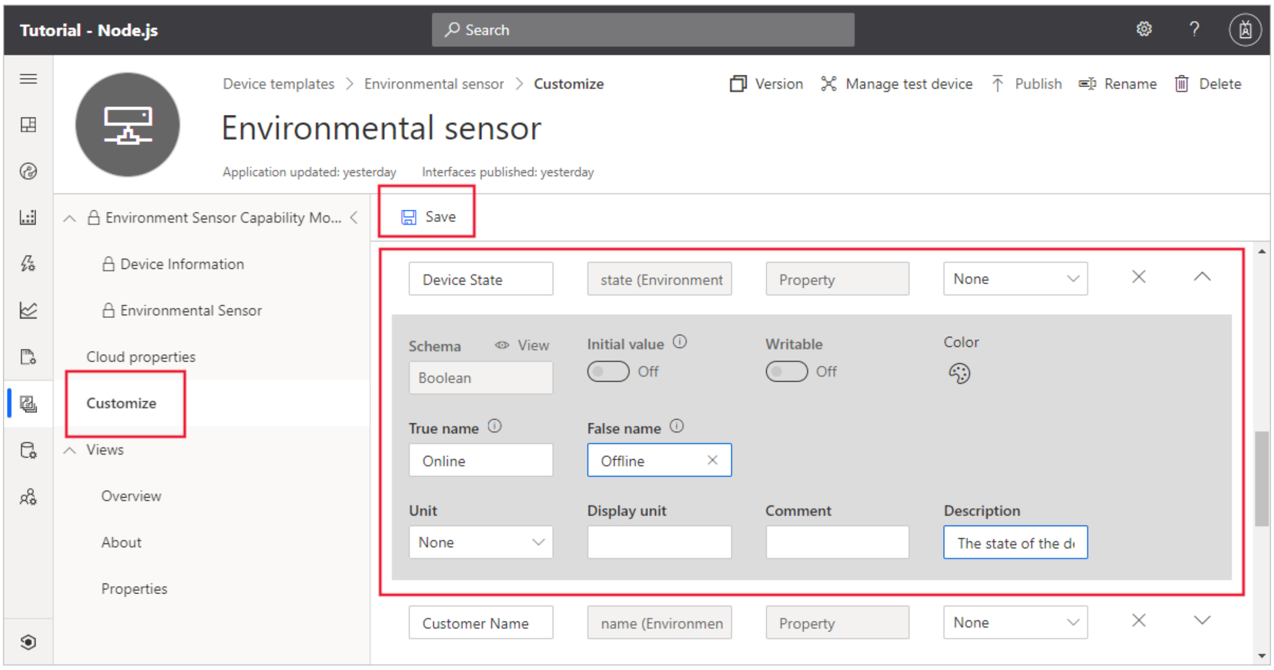
In your Azure IoT Central application, create a device template called *Environmental sensor* by importing the `EnvironmentalSensorInline.capabilitymodel.json` device capability model file:



The device capability model includes two interfaces: the standard **Device Information** interface and the custom **Environmental Sensor** interface. The **Environmental Sensor** interface defines the following capabilities:

TYPE	DISPLAY NAME	DESCRIPTION
Property	Device State	The state of the device. Two states online/offline are available.
Property (writeable)	Customer Name	The name of the customer currently operating the device.
Property (writeable)	Brightness Level	The brightness level for the light on the device. Can be specified as 1 (high), 2 (medium), 3 (low).
Telemetry	Temperature	Current temperature detected by the device.
Telemetry	Humidity	Current humidity detected by the device.
Command	blink	Begin blinking the LED on the device for given time interval.
Command	turnon	Turn on the LED on the device.
Command	turnoff	Turn off the LED on the device.
Command	rundiagnostics	This asynchronous command starts a diagnostics run on the device.

To customize how the **Device State** property displays in your IoT Central application, select **Customize** in the device template. Expand the **Device State** entry, enter *Online* as the **True name** and *Offline* as the **False name**. Then save the changes:



Create views

Views let you interact with devices connected to your IoT Central application. For example, you can have views that display telemetry, views that display properties, and views that let you edit writeable and cloud properties. Views are part of a device template.

To add some default views to your **Environmental sensor** device template, navigate to your device template, select **Views**, and select the **Generate Default views** tile. Make sure **Overview** and **About** are **On**, and then select **Generate default dashboard view(s)**. You now have two default views defined in your template.

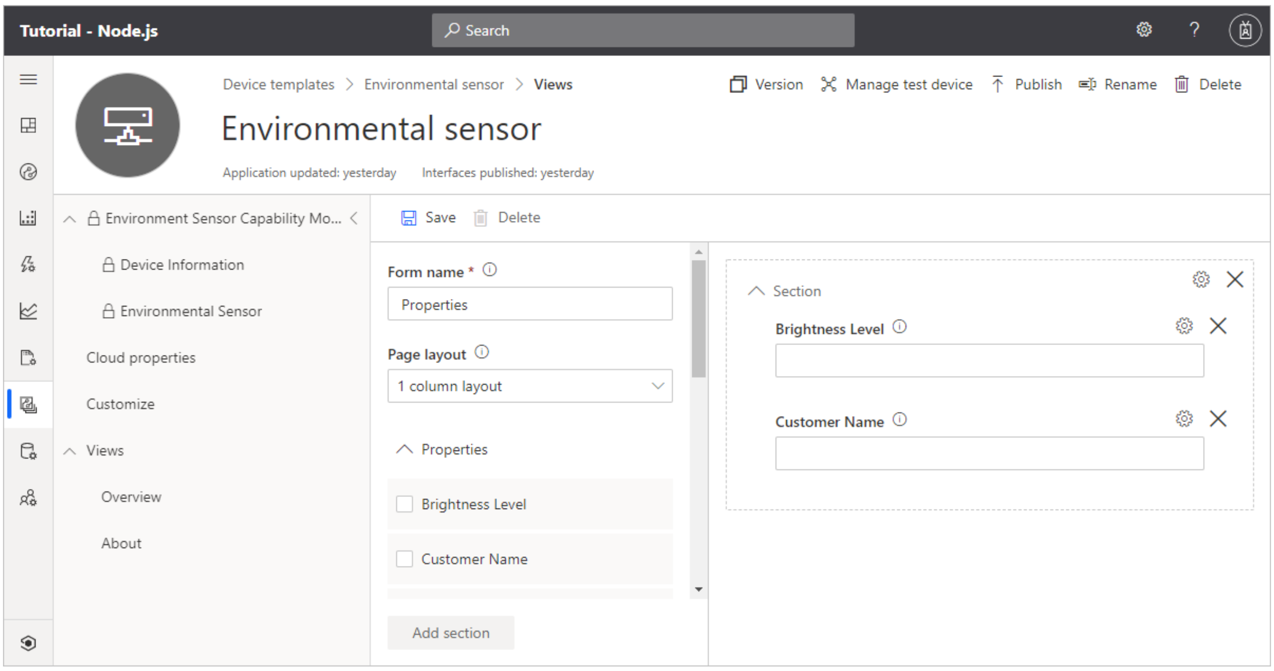
The **Environmental Sensor** interface includes two writeable properties - **Customer Name** and **Brightness Level**. To create a view, you can use to edit these properties:

1. Select **Views** and then select the **Editing device and cloud data** tile.

2. Enter *Properties* as the form name.

3. Select the **Brightness Level** and **Customer Name** properties. Then select **Add section**.

4. Save your changes.

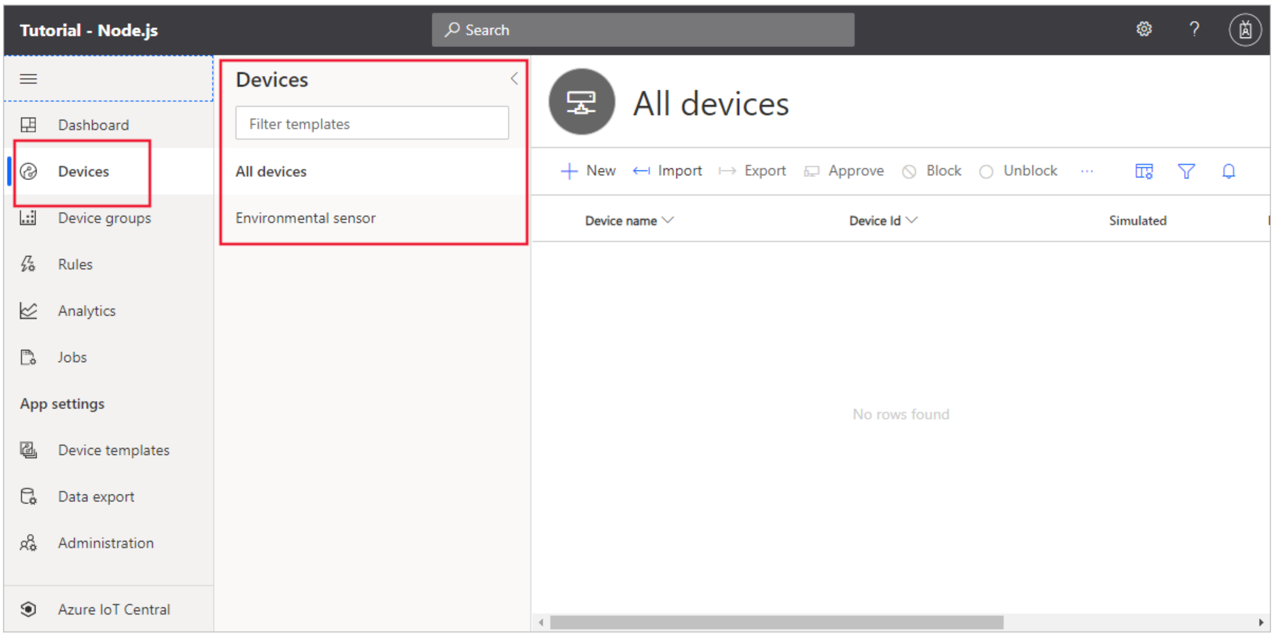


Publish the template

Before you can add a device that uses the **Environmental sensor** device template, you must publish it.

In the device template, select **Publish**. On the **Publish this device template to the application** panel, select **Publish**.

To check that the template is ready to use, navigate to the **Devices** page in your IoT Central application. The **Devices** section shows a list of the published devices in the application:

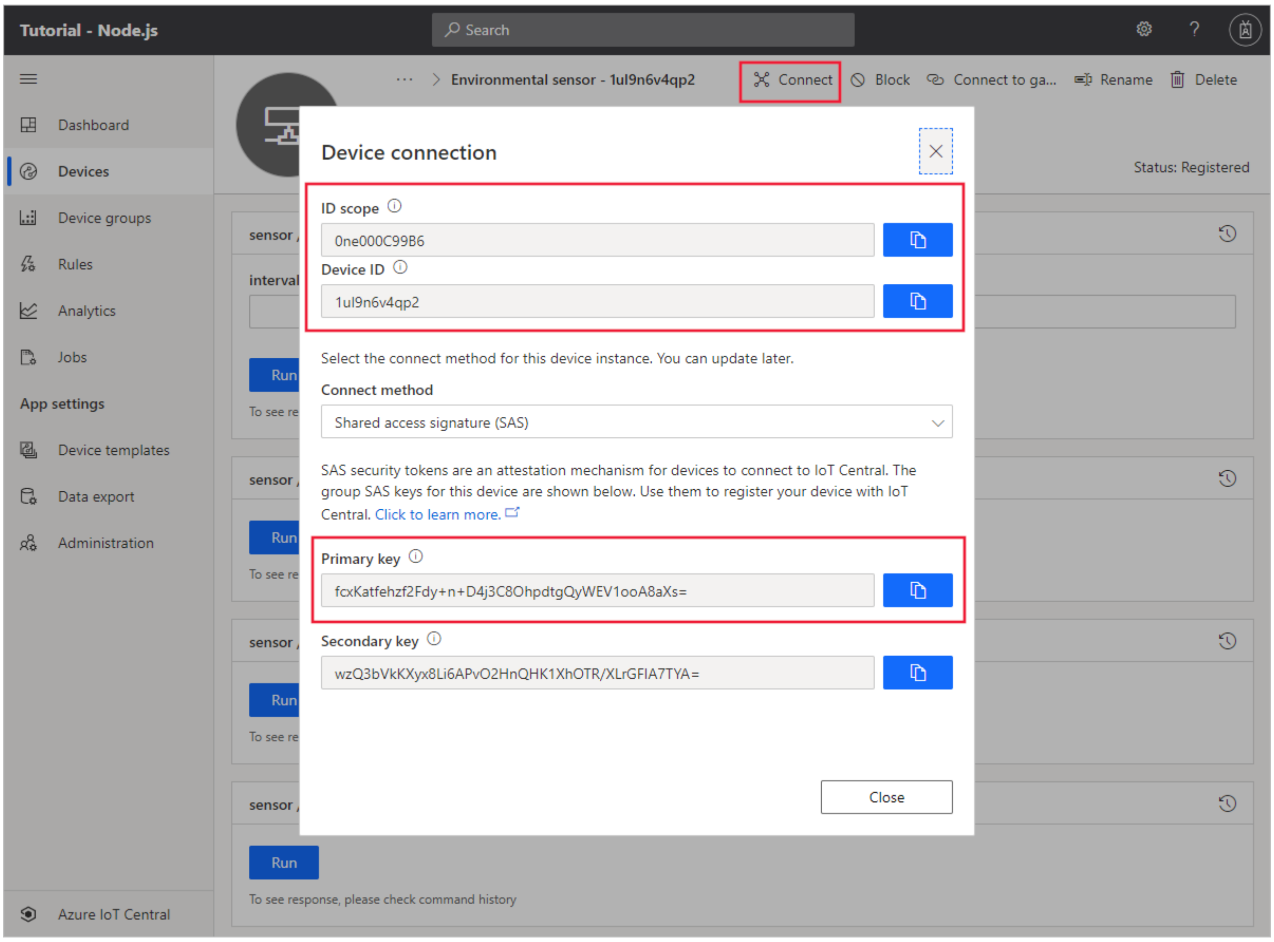


Add a real device

In your Azure IoT Central application, add a real device to the device template you created in the previous section:

1. On the **Devices** page, select the **Environmental sensor** device template.
2. Select **+ New**.
3. In the **Create a new device** dialog, make sure that **Environmental Sensor** is the template type and that **Simulate this device?** is set to **No**.
4. Then select **Create**.

Click on the device name, and then select **Connect**. Make a note of the device connection information on the **Device Connection** page - **ID scope**, **Device ID**, and **Primary key**. You need these values when you create your device code:



Create a Python application

The following steps show you how to create a Python client application that connects to the real device you added to the application. This Python application simulates the behavior of a real device.

1. In your command-line environment, navigate to the `environmental-sensor` folder you created previously.

2. To install the required libraries, run the following commands:

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

3. Create a file called **environmental_sensor.py** in the `environmental-sensor` folder.

4. Add the following `import` statements at the start of the **environmental_sensor.py** file:

```
import asyncio
import os
import json
import datetime
import random

from azure.iot.device.aio import ProvisioningDeviceClient
from azure.iot.device.aio import IoTHubDeviceClient
from azure.iot.device import MethodResponse
from azure.iot.device import Message
```

5. Add the following asynchronous `main` function and variable declarations to the file:

```
async def main():
  # In a production environment, don't store
  # connection information in the code.
  provisioning_host = 'global.azure-devices-provisioning.net'
  id_scope = '{your Scope ID}'
  registration_id = '{your Device ID}'
  symmetric_key = '{your Primary Key}'

  delay = 2

  # All the remaining code is nested within this main function

if __name__ == '__main__':
asyncio.run(main())
```

Update the placeholders `{your Scope ID}`, `{your Device ID}`, and `{your Primary Key}` with the values you made a note of previously. In a real application, don't hard code this information in the application.

All the following function definitions and code are nested within the `main` function.

6. Add the following two functions inside the `main` function to register the device and connect it to your IoT Central application. Registration uses the Azure Device Provisioning Service:

```
  async def register_device():
    provisioning_device_client = ProvisioningDeviceClient.create_from_symmetric_key(
      provisioning_host=provisioning_host,
      registration_id=registration_id,
      id_scope=id_scope,
      symmetric_key=symmetric_key,
    )

    registration_result = await provisioning_device_client.register()

    print(f'Registration result: {registration_result.status}')

    return registration_result

  async def connect_device():
    device_client = None
    try:
      registration_result = await register_device()
      if registration_result.status == 'assigned':
        device_client = IoTHubDeviceClient.create_from_symmetric_key(
          symmetric_key=symmetric_key,
          hostname=registration_result.registration_state.assigned_hub,
          device_id=registration_result.registration_state.device_id,
        )
        # Connect the client.
        await device_client.connect()
        print('Device connected successfully')
    finally:
      return device_client
```

7. Add the following function inside the `main` function to send telemetry to your IoT Central application:

```
async def send_telemetry():
  print(f'Sending telemetry from the provisioned device every {delay} seconds')
  while True:
    temp = random.randrange(1, 75)
    humid = random.randrange(30, 99)
    payload = json.dumps({'temp': temp, 'humid': humid})
    msg = Message(payload)
    await device_client.send_message(msg, )
    print(f'Sent message: {msg}')
    await asyncio.sleep(delay)
```

The names of the telemetry items (`temp` and `humid`) must match the names used in the device template.

8. Add the following functions inside the `main` function to handle commands called from your IoT Central application:

```
    async def blink_command(request):
      print('Received synchronous call to blink')
      response = MethodResponse.create_from_method_request(
        request, status = 200, payload = {'description': f'Blinking LED every {request.payload} seconds'}
      )
      await device_client.send_method_response(response)  # send response
      print(f'Blinking LED every {request.payload} seconds')

    async def diagnostics_command(request):
      print('Starting asynchronous diagnostics run...')
      response = MethodResponse.create_from_method_request(
        request, status = 202
      )
      await device_client.send_method_response(response)  # send response
      print('Generating diagnostics...')
      await asyncio.sleep(2)
      print('Generating diagnostics...')
      await asyncio.sleep(2)
      print('Generating diagnostics...')
      await asyncio.sleep(2)
      print('Sending property update to confirm command completion')
      await device_client.patch_twin_reported_properties({'rundiagnostics': {'value': f'Diagnostics run
  complete at {datetime.datetime.today()}.'}})

    async def turnon_command(request):
      print('Turning on the LED')
      response = MethodResponse.create_from_method_request(
        request, status = 200
      )
      await device_client.send_method_response(response)  # send response

    async def turnoff_command(request):
      print('Turning off the LED')
      response = MethodResponse.create_from_method_request(
        request, status = 200
      )
      await device_client.send_method_response(response)  # send response

    commands = {
      'blink': blink_command,
      'rundiagnostics': diagnostics_command,
      'turnon': turnon_command,
      'turnoff': turnoff_command,
    }

    # Define behavior for handling commands
    async def command_listener():
      while True:
        method_request = await device_client.receive_method_request()  # Wait for commands
        await commands[method_request.name](method_request)
```

The names of the commands (`blink`, `turnon`, `turnoff`, and `rundiagnostics`) must match the names used in the device template.

Currently, IoT Central doesn't use the response schema defined in the device capability model. For a synchronous command, the response payload can be any valid JSON. For an asynchronous command, the device should return a 202 response immediately, followed by reported property update when the work is finished. The format of the reported property update is:

```
{
  [command name] : {
    value: 'response message'
  }
}
```

An operator can view the response payload in the command history.

9. Add the following functions inside the `main` function to handle property updates sent from your IoT Central application:

```
  async def name_setting(value, version):
    await asyncio.sleep(1)
    print(f'Setting name value {value} - {version}')
    await device_client.patch_twin_reported_properties({'name' : {'value': value, 'ad': 'completed',
'ac': 200, 'av': version}})

  async def brightness_setting(value, version):
    await asyncio.sleep(5)
    print(f'Setting brightness value {value} - {version}')
    await device_client.patch_twin_reported_properties({'brightness' : {'value': value, 'ad':
'completed', 'ac': 200, 'av': version}})

  settings = {
    'name': name_setting,
    'brightness': brightness_setting
  }

  # define behavior for receiving a twin patch
  async def twin_patch_listener():
    while True:
      patch = await device_client.receive_twin_desired_properties_patch() # blocking
      to_update = patch.keys() & settings.keys()
      await asyncio.gather(
        *[settings[setting](patch[setting], patch['$version']) for setting in to_update]
      )
```

When the operator sets a writeable property in the IoT Central application, the application uses a device twin desired property to send the value to the device. The device then responds using a device twin reported property. When IoT Central receives the reported property value, it updates the property view with a status of **synced**.

The names of the properties (`name` and `brightness`) must match the names used in the device template.

10. Add the following functions inside the `main` function to control the application:

```
  # Define behavior for halting the application
  def stdin_listener():
    while True:
      selection = input('Press Q to quit\n')
      if selection == 'Q' or selection == 'q':
        print('Quitting...')
        break

  device_client = await connect_device()

  if device_client is not None and device_client.connected:
    print('Send reported properties on startup')
    await device_client.patch_twin_reported_properties({'state': 'true', 'processorArchitecture':
'ARM', 'swVersion': '1.0.0'})
    tasks = asyncio.gather(
      send_telemetry(),
      command_listener(),
      twin_patch_listener(),
    )

    # Run the stdin listener in the event loop
    loop = asyncio.get_running_loop()
    user_finished = loop.run_in_executor(None, stdin_listener)

    # Wait for user to indicate they are done listening for method calls
    await user_finished

    # Cancel tasks
    tasks.add_done_callback(lambda r: r.exception())
    tasks.cancel()
    await device_client.disconnect()

  else:
    print('Device could not connect')
```

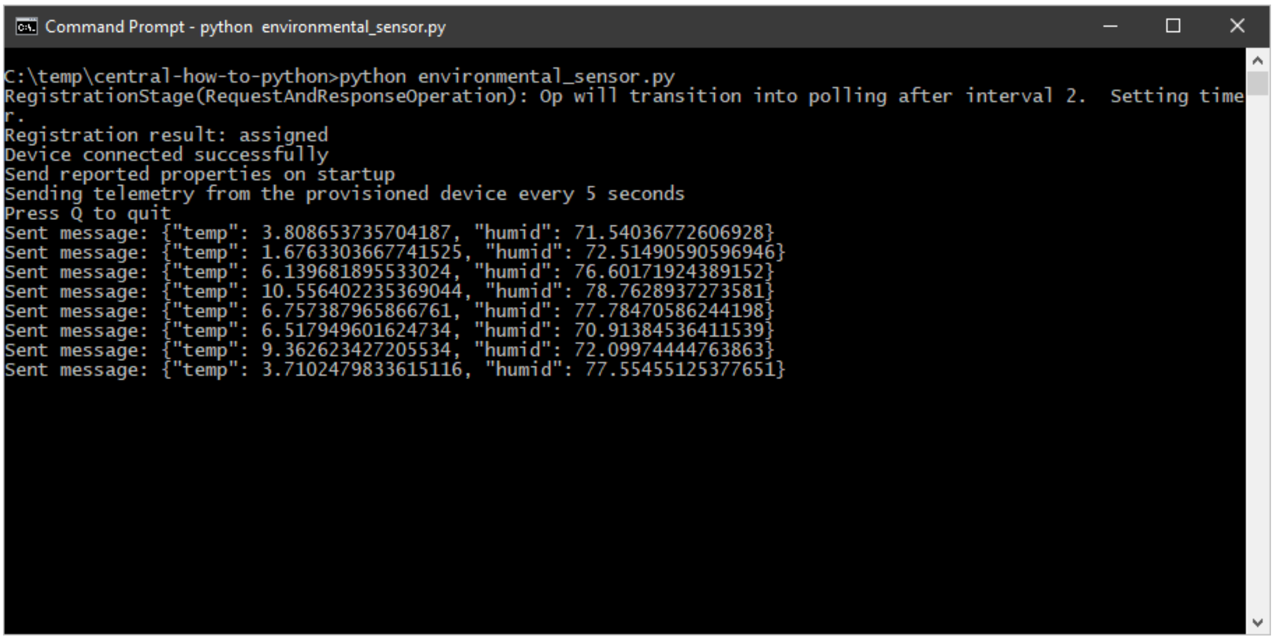
11. Save the the **environmental_sensor.py** file.

Run your Python application

To start the device client application, run the following command in your command-line environment:

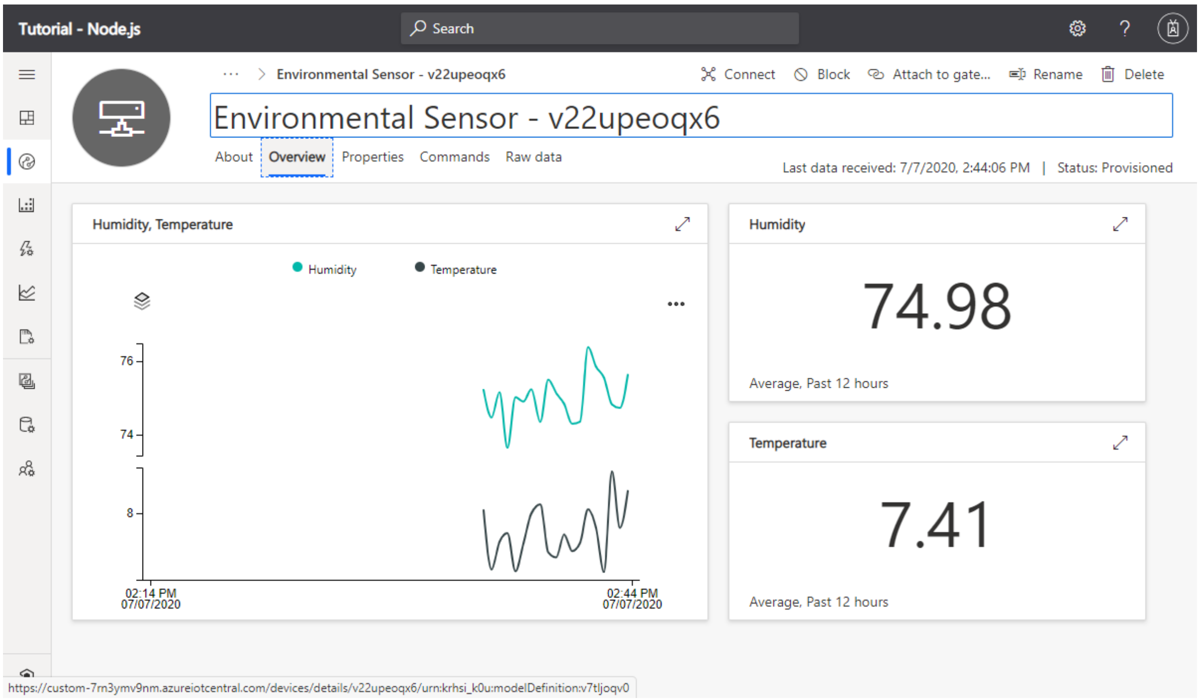
```
python3 environmental_sensor.py
```

You can see the device connects to your Azure IoT Central application and starts sending telemetry:

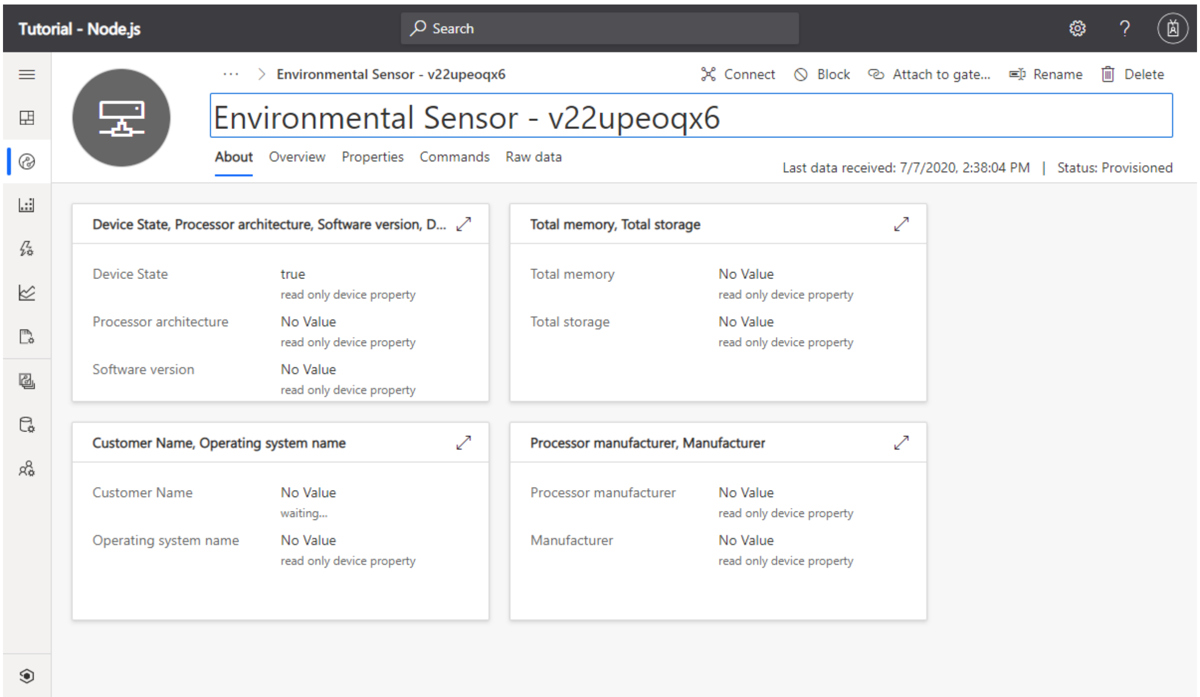


As an operator in your Azure IoT Central application, you can:

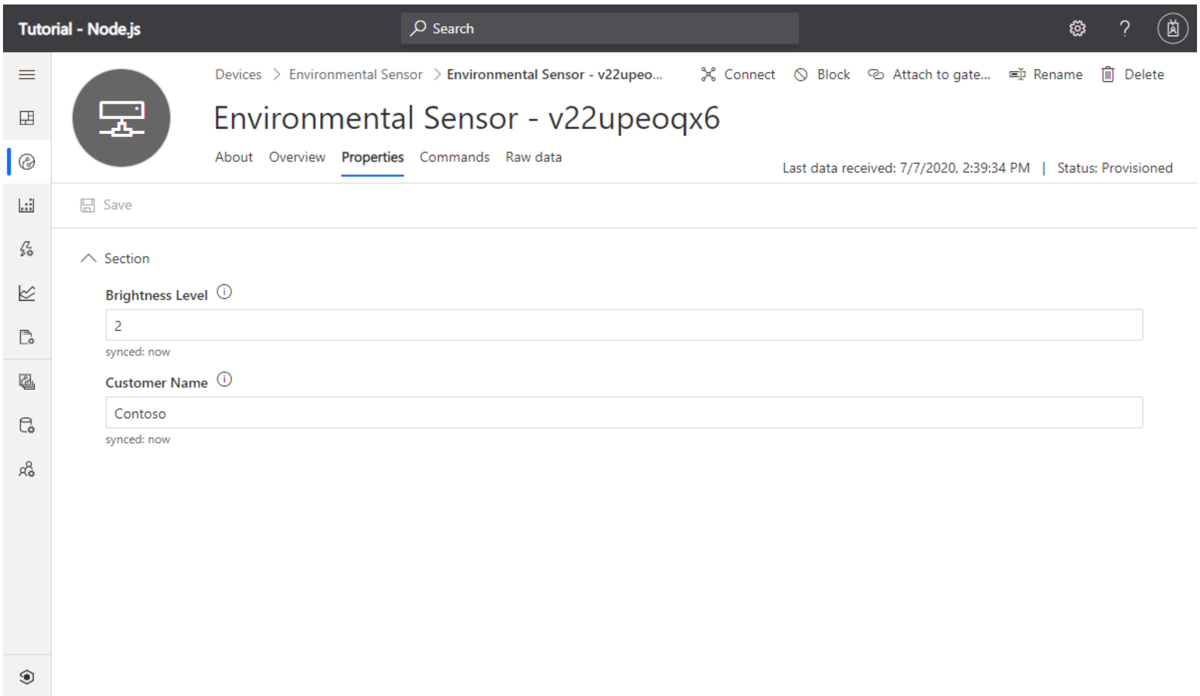
- View the telemetry sent by the device on the **Overview** page:



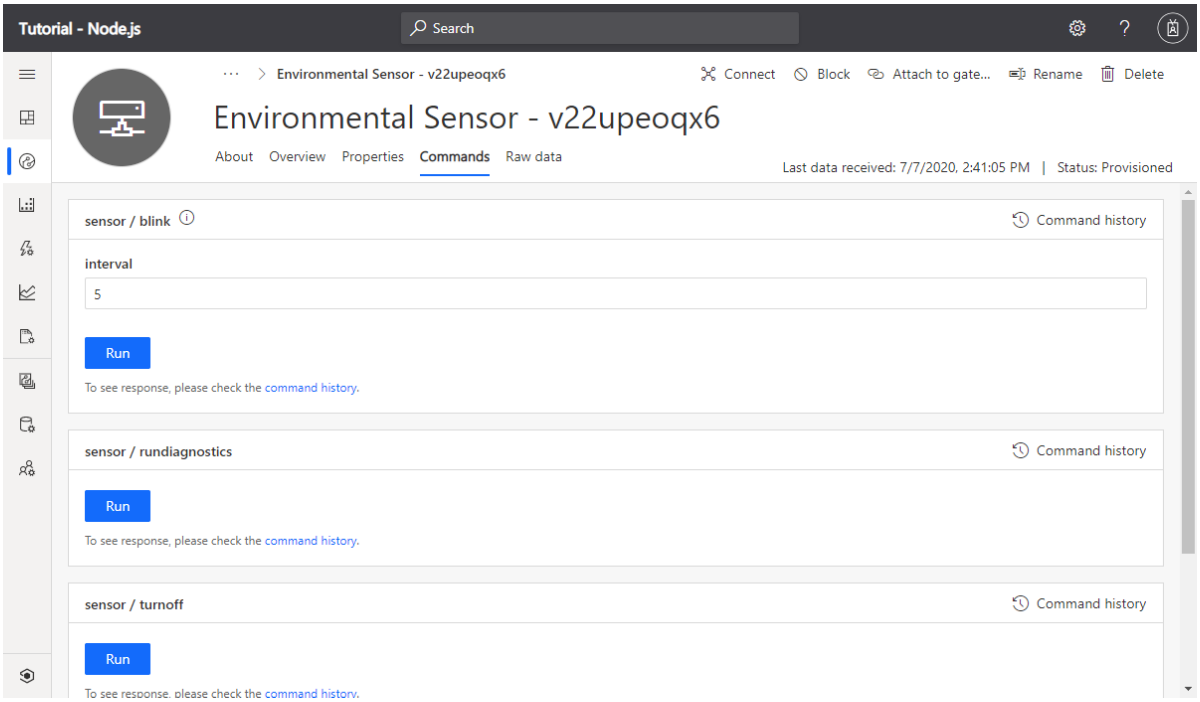
- View the device properties on the **About** page:

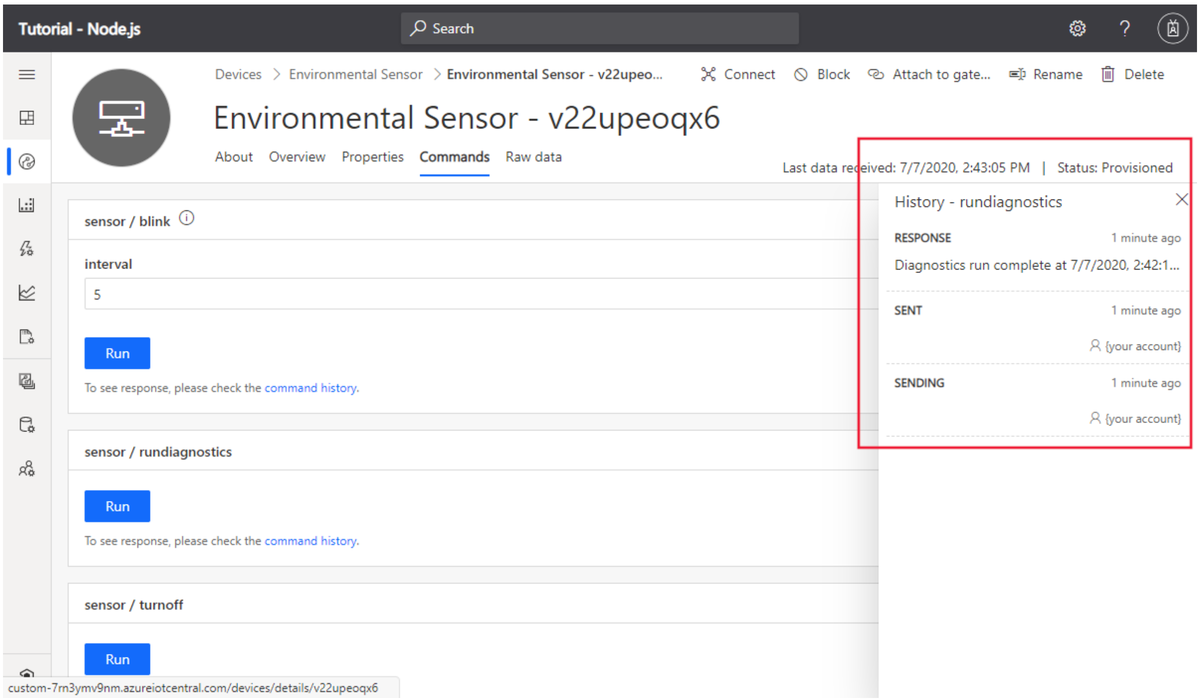


- Update writeable property values on the **Properties** page:

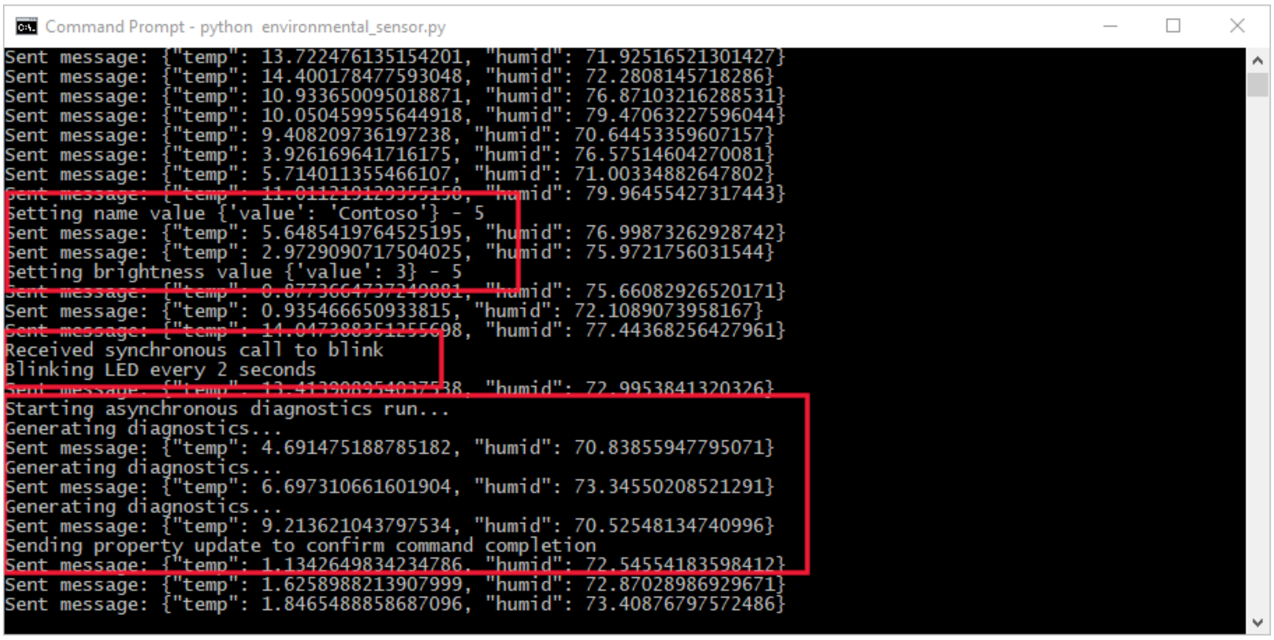


- Call the commands from the **Commands** page:



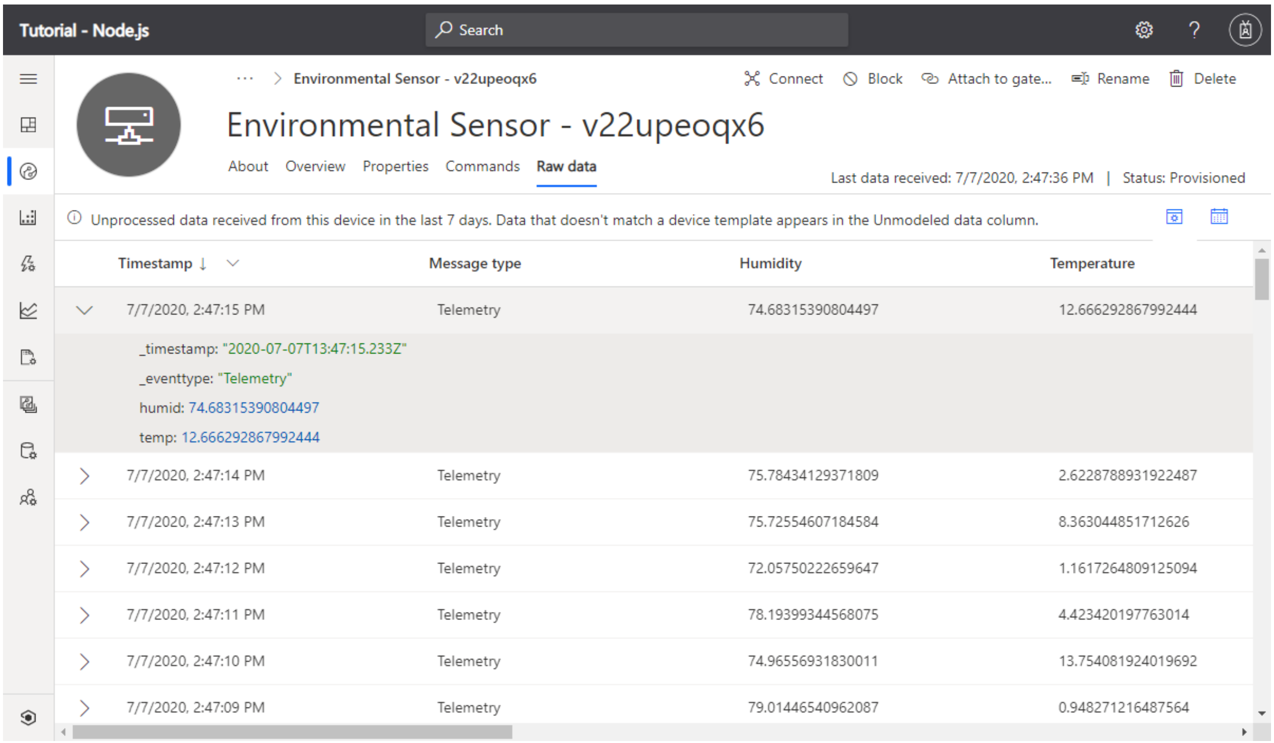


You can see how the device responds to commands and property updates:



View raw data

As a device developer, you can use the **Raw data** view to examine the raw data your device is sending to IoT Central:



On this view, you can select the columns to display and set a time range to view. The **Unmodeled data** column shows data from the device that doesn't match any property or telemetry definitions in the device template.

Next steps

As a device developer, now that you've learned the basics of how to create a device using Python, some suggested next steps are to:

- Learn how to connect a real device to IoT Central in the Connect an MXChip IoT DevKit device to your Azure IoT Central application how-to article.
- Read What are device templates? to learn more about the role of device templates when you're implementing your device code.

- Read Get connected to Azure IoT Central to learn more about how to register devices with IoT Central and how IoT Central secures device connections.

If you'd prefer to continue through the set of IoT Central tutorials and learn more about building an IoT Central solution, see:

Create a gateway device template

Tutorial: Use a device capability model to create an IoT Plug and Play (preview) device and connect it to your IoT Central application

7/21/2020 • 5 minutes to read • Edit Online

A *device capability model* (DCM) describes the capabilities of an IoT Plug and Play (preview) device. IoT Central can use a DCM to create a device template and visualizations for a device when the device connects for the first time.

Support for IoT Plug and Play is in preview and is only supported only in selected regions.

In this tutorial, you learn how to:

- Use Visual Studio Code to create an IoT Plug and Play (preview) device using a DCM.
- Run the device code in Windows and see it connect to your IoT Central application.
- View the simulated telemetry the device sends.

Prerequisites

Complete the Create an Azure IoT Central application quickstart to create an IoT Central application using the **Custom app > Custom application** template.

To complete this tutorial, you need to install the following software on your local machine:

- Build Tools for Visual Studio with **C++ build tools** and **Nuget package manager component** workloads. Or if you already have Visual Studio (Community, Professional, or Enterprise) 2019, 2017 or 2015 with same workloads installed.
- Git.
- CMake - when you install **CMake**, select the option **Add CMake to the system PATH**.
- Visual Studio Code.
- Node.js
- The `dps-keygen` utility:

  ```
  npm i -g dps-keygen
  ```

Install Azure IoT Tools

Use the following steps to install the Azure IoT Tools extension pack in VS Code:

1. In VS Code, select the **Extensions** tab.
2. Search for **Azure IoT Tools**.
3. Select **Install**.

Prepare the development environment

In this tutorial, you use the Vcpkg library manager to install the Azure IoT C device SDK in your development environment.

1. Open a command prompt. Execute the following command to install Vcpkg:

```
git clone https://github.com/Microsoft/vcpkg.git
cd vcpkg

.\bootstrap-vcpkg.bat
```

   Then, to hook up user-wide integration, run the following command. The first time you run this command it requires administrative rights:

```
.\vcpkg.exe integrate install
```

2. Install Azure IoT C device SDK Vcpkg:

```
.\vcpkg.exe install azure-iot-sdk-c[public-preview,use_prov_client]
```

Generate device key

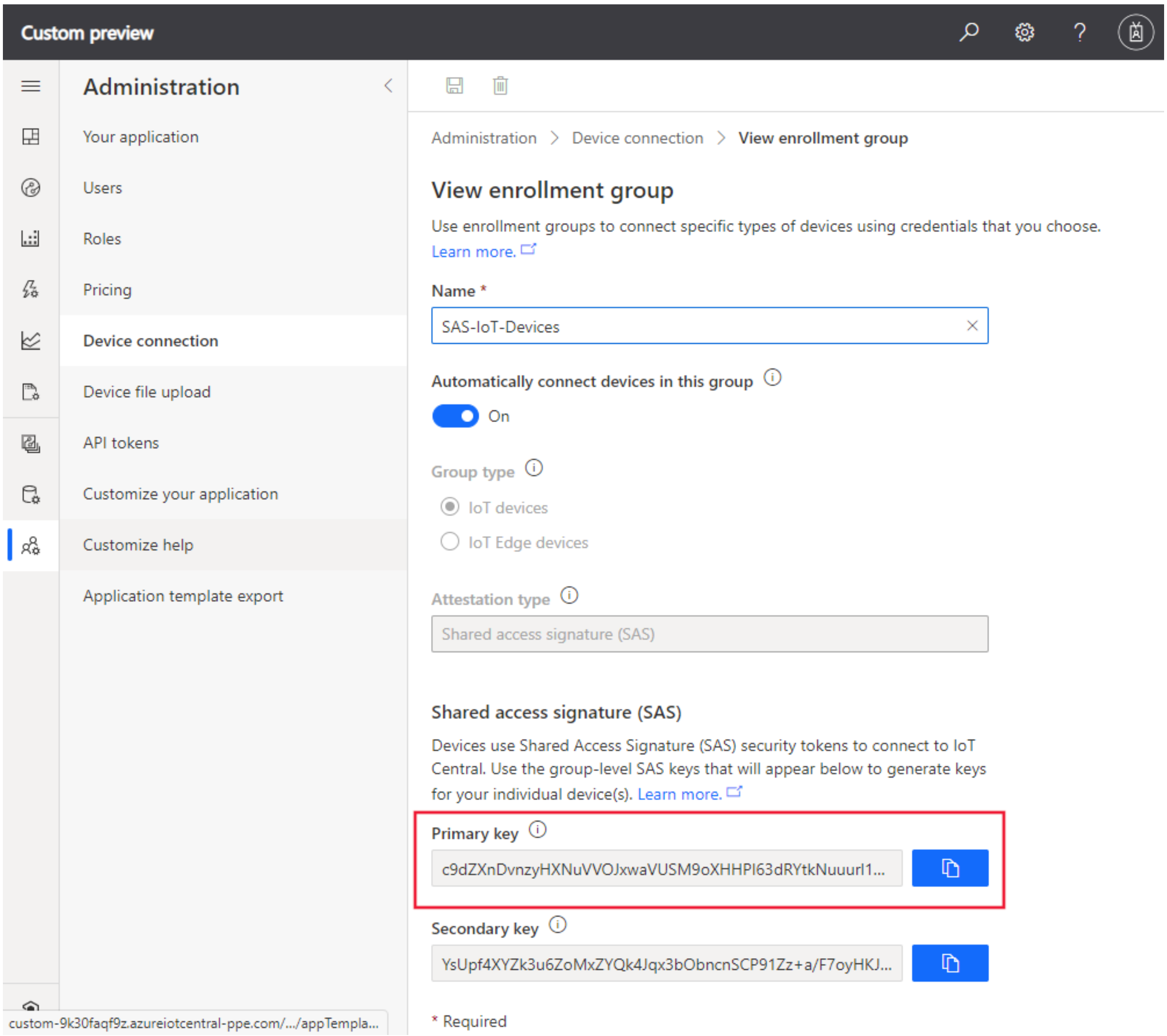
To connect a device to an IoT Central application, you need a device key. To generate a device key:

1. Sign in to the IoT Central application you created using the **Custom application** template in the Create an Azure IoT Central application quickstart.

2. Go to the **Administration** page and select **Device Connection**.

3. Make a note of the **ID Scope**. You use this value later in this tutorial.

4. Select the **SAS-IoT-Devices** enrollment group. Make a note of the **Primary Key**. You use this value later in this tutorial.



5. Open a command prompt and run the following command to generate a device key:

```
dps-keygen -di:mxchip-001 -mk:{Primary Key from previous step}
```

Make a note of the generated *device key*, you use this value in a later step in this tutorial.

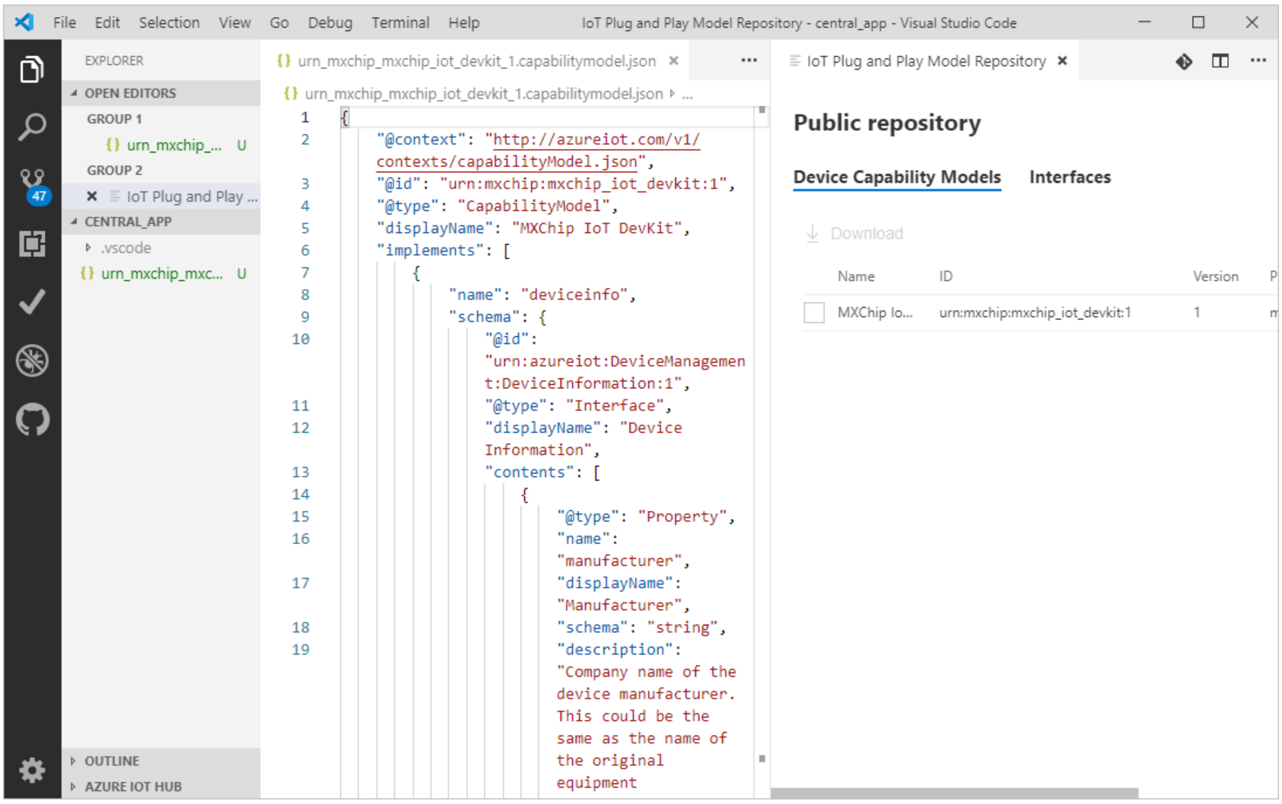
Download your model

In this tutorial, you use the public DCM for an MxChip IoT DevKit device. You don't need an actual DevKit device to run the code, in this tutorial you compile the code to run on Windows.

1. Create a folder called `central_app` and open it in VS Code.

2. Use **Ctrl+Shift+P** to open the command palette, enter **IoT Plug and Play**, and select **Open Model Repository**. Select **Public repository**. VS Code shows a list of the DCMs in the public model repository.

3. Select the **MXChip IoT DevKit** DCM with ID `urn:mxchip:mxchip_iot_devkit:1`. Then select **Download**. You now have a copy of the DCM in the `central_app` folder.



> **NOTE**
>
> To work with IoT Central, the device capability model must have all the interfaces defined inline in the same file.

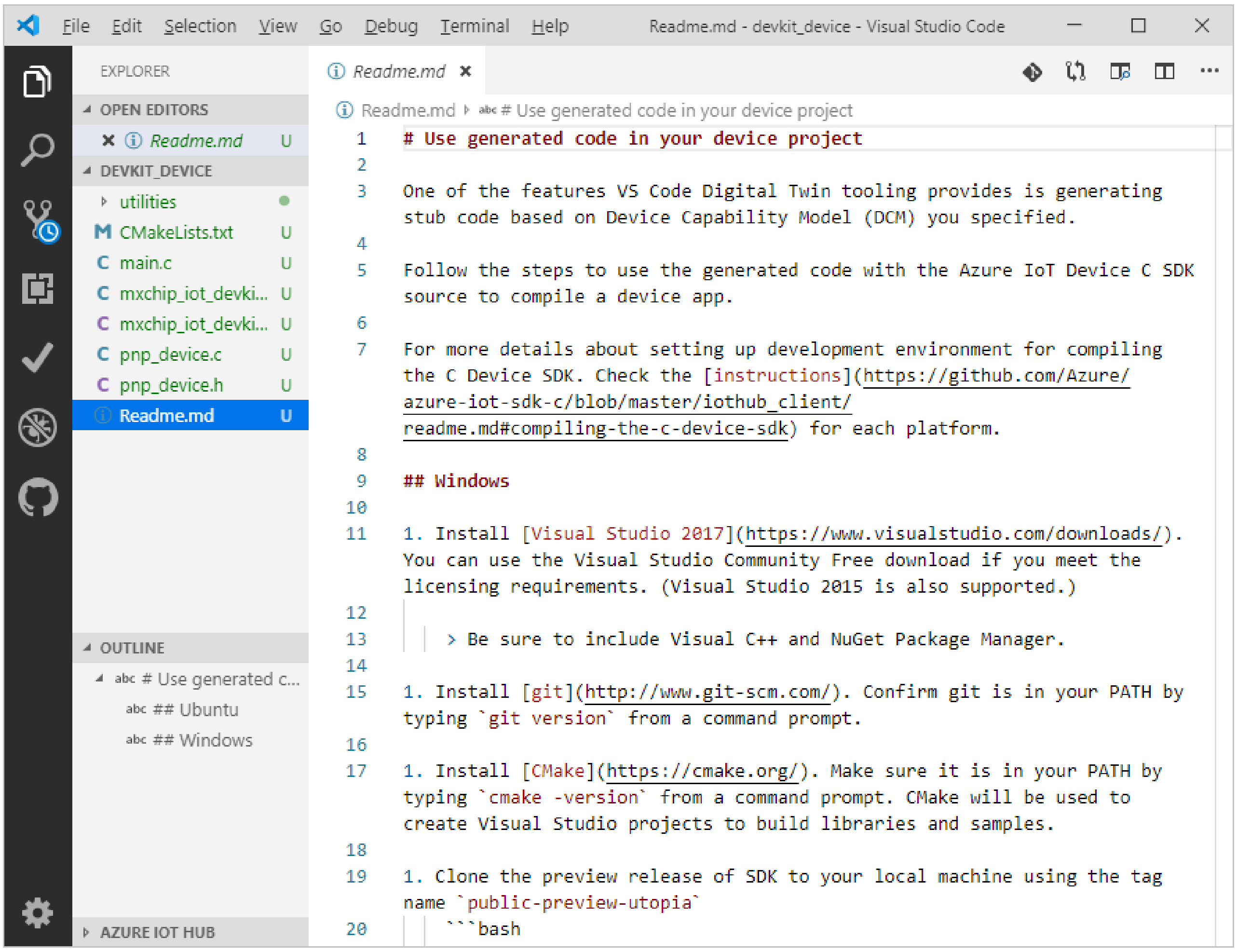
Generate the C code stub

Now you have the **MXChip IoT DevKit** DCM and its associated interfaces, you can generate the device code that implements the model. To generate the C code stub in VS code:

1. With the folder with DCM files open, use **Ctrl+Shift+P** to open the command palette, enter **IoT Plug and Play**, and select **Generate Device Code Stub**.

   > **NOTE**
   >
   > The first time you use the IoT Plug and Play Code Generator utility, it takes a few seconds to download.

2. Select the **MXChip IoT DevKit** DCM file you just downloaded.
3. Enter the project name **devkit_device**.
4. Choose **ANSI C** as your language.
5. Choose **Via DPS (Device Provisioning Service) symmetric key** as the connection method.
6. Choose **CMake Project on Windows** as your project type. Don't choose **MXChip IoT DevKit Project**, this option is for when you have a real DevKit device.
7. Choose **Via Vcpkg** as the way to include the SDK.
8. VS Code opens a new window with generated device code stub files in the `devkit_device` folder.



Build the code

You use the device SDK to build the generated device code stub. The application you build simulates an **MXChip IoT DevKit** device and connects to your IoT Central application. The application sends telemetry and properties, and receives commands.

1. At a command prompt, create a `cmake` subdirectory in the `devkit_device` folder, and navigate to that folder:

```
mkdir cmake
cd cmake
```

2. Run the following commands to build the generated code stub. Replace the `<directory of your Vcpkg repo>` placeholder with the path to your copy of the **Vcpkg** repository:

```
cmake .. -G "Visual Studio 16 2019" -A Win32 -Duse_prov_client=ON -Dhsm_type_symm_key:BOOL=ON -DCMAKE_TOOLCHAIN_FILE="<directory of your Vcpkg repo>\scripts\buildsystems\vcpkg.cmake"

cmake --build . -- /p:Configuration=Release
```

If you're using Visual Studio 2017 or 2015, you need to specify the CMake generator based on the build tools you're using:

```
# Either
cmake .. -G "Visual Studio 15 2017" -Duse_prov_client=ON -Dhsm_type_symm_key:BOOL=ON -
DCMAKE_TOOLCHAIN_FILE="<directory of your Vcpkg repo>\scripts\buildsystems\vcpkg.cmake"
# or
cmake .. -G "Visual Studio 14 2015" -Duse_prov_client=ON -Dhsm_type_symm_key:BOOL=ON -
DCMAKE_TOOLCHAIN_FILE="<directory of your Vcpkg repo>\scripts\buildsystems\vcpkg.cmake"
```

3. After the build completes successfully, at the same command prompt run your application. Replace `<scopeid>` and `<devicekey>` with the values you noted previously:

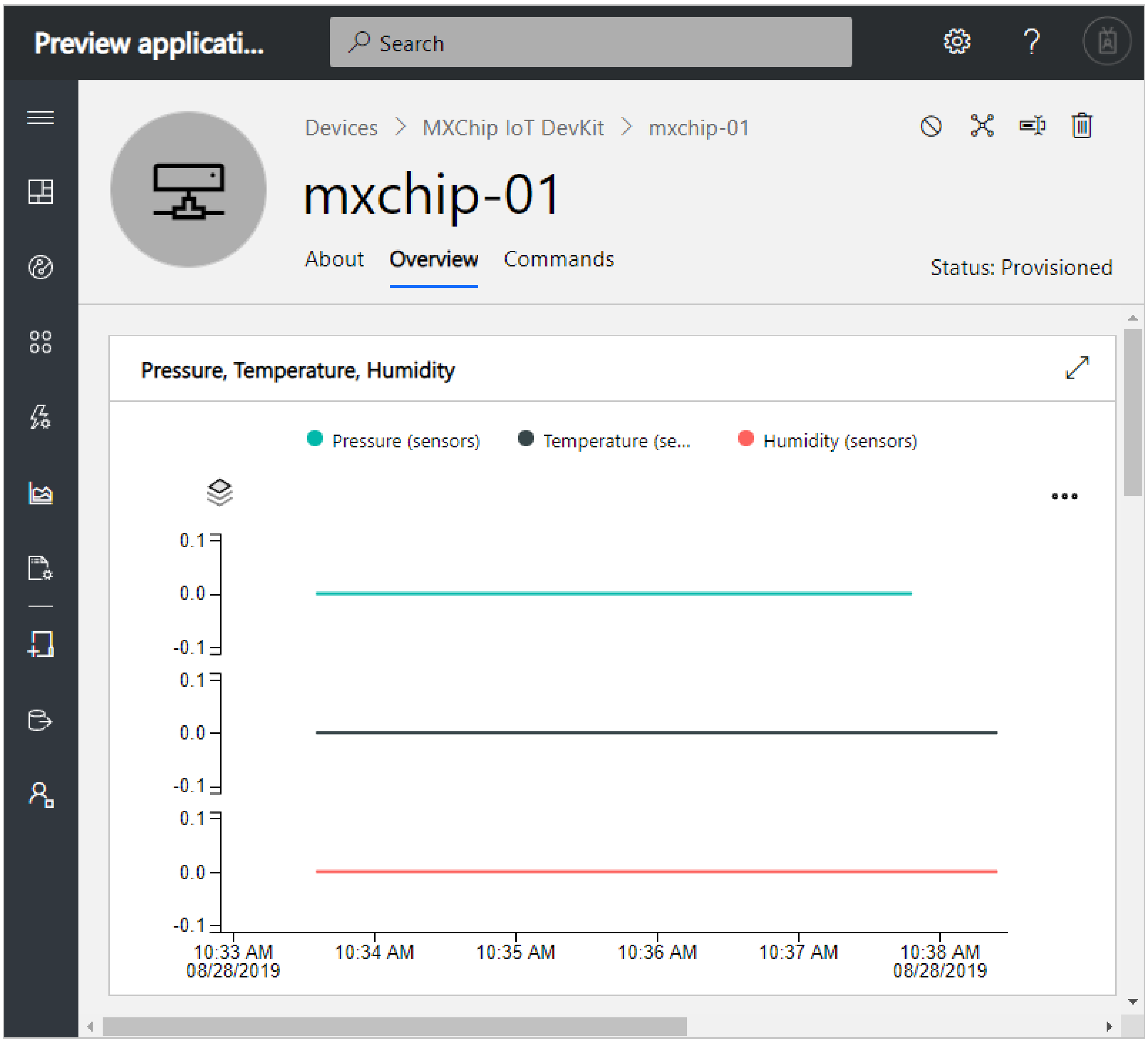
```
.\Release\devkit_device.exe mxchip-001 <scopeid> <devicekey>
```

4. The device application starts sending data to IoT Hub. Sometimes you see the error `Error registering device for DPS` the first time you run the previous command. If you see this error, retry the command.

View the device

After your device code connects to your IoT Central, you can view the properties and telemetry it sends:

1. In your IoT Central application, go to the **Devices** page and select the **mxchip-01** device. This device was automatically added when the device code connected:

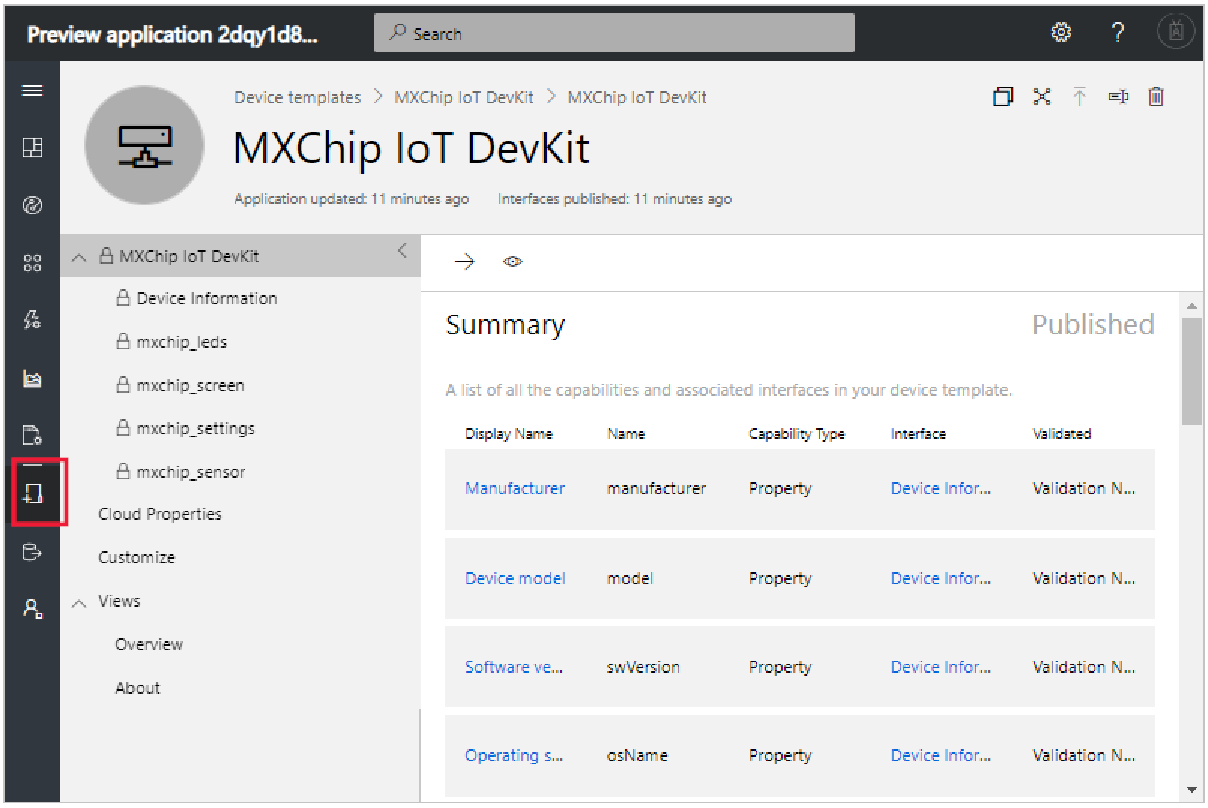


After a couple of minutes, this page shows charts of the telemetry the device is sending.

2. Select the **About** page to see the property values the device sent.

3. Select the **Commands** page to call commands on the device. You can see the device responding at the command prompt that's running the device code.

4. Go to the **Device templates** page to see the template that IoT Central created from the DCM in the public repository:



Next steps

In this tutorial, you learned how to connect an IoT Plug and Play (preview) device that was generated from a DCM in the public model repository.

To learn more about DCMs and how to create your own models, continue to the how-to guide:

Define a new IoT device type

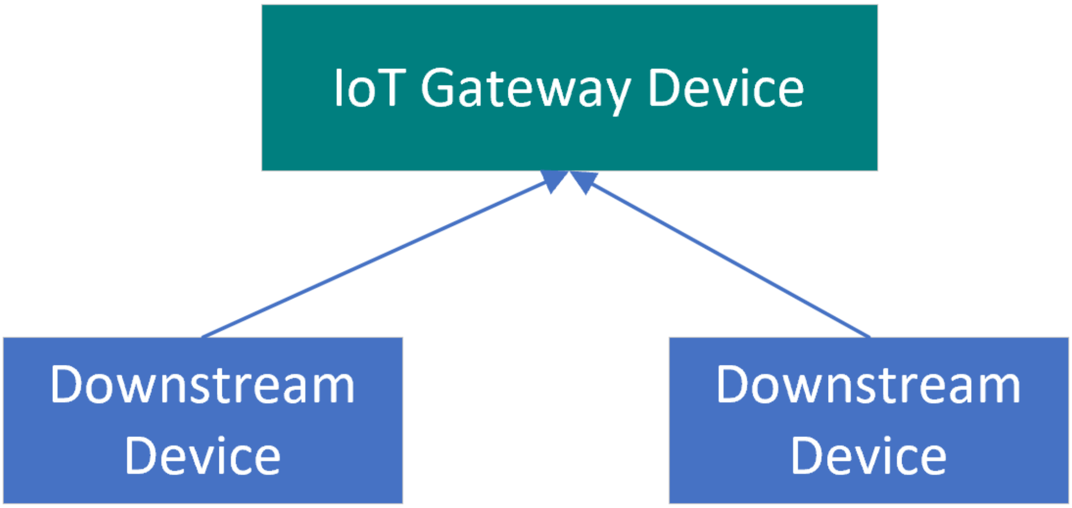
Define a new IoT gateway device type in your Azure IoT Central application

4/21/2020 • 6 minutes to read • Edit Online

This article applies to solution builders and device developers.

This tutorial shows you, as a solution builder, how to use a gateway device template to define a gateway device in your IoT Central application. You then configure several downstream devices that connect to your IoT Central application through the gateway device.

In this tutorial, you create a **Smart Building** gateway device template. A **Smart Building** gateway device has relationships with other downstream devices.



As well as enabling downstream devices to communicate with your IoT Central application, a gateway device can also:

- Send its own telemetry, such as temperature.
- Respond to writeable property updates made by an operator. For example, an operator could changes the telemetry send interval.
- Respond to commands, such as rebooting the device.

Prerequisites

To complete this tutorial, you need to Create an Azure IoT Central application.

Create downstream device templates

This tutorial uses device templates for an **S1 Sensor** device and an **RS40 Occupancy Sensor** device to generate simulated downstream devices.

To create a device template for an **S1 Sensor** device:

1. In the left pane, select **Device Templates**. Then select **+** to start adding the template.
2. Scroll down until you can see the tile for the **S1 Sensor** device. Select the tile and then select **Next:**

Customize.

3. On the **Review** page, select **Create** to add the device template to your application.

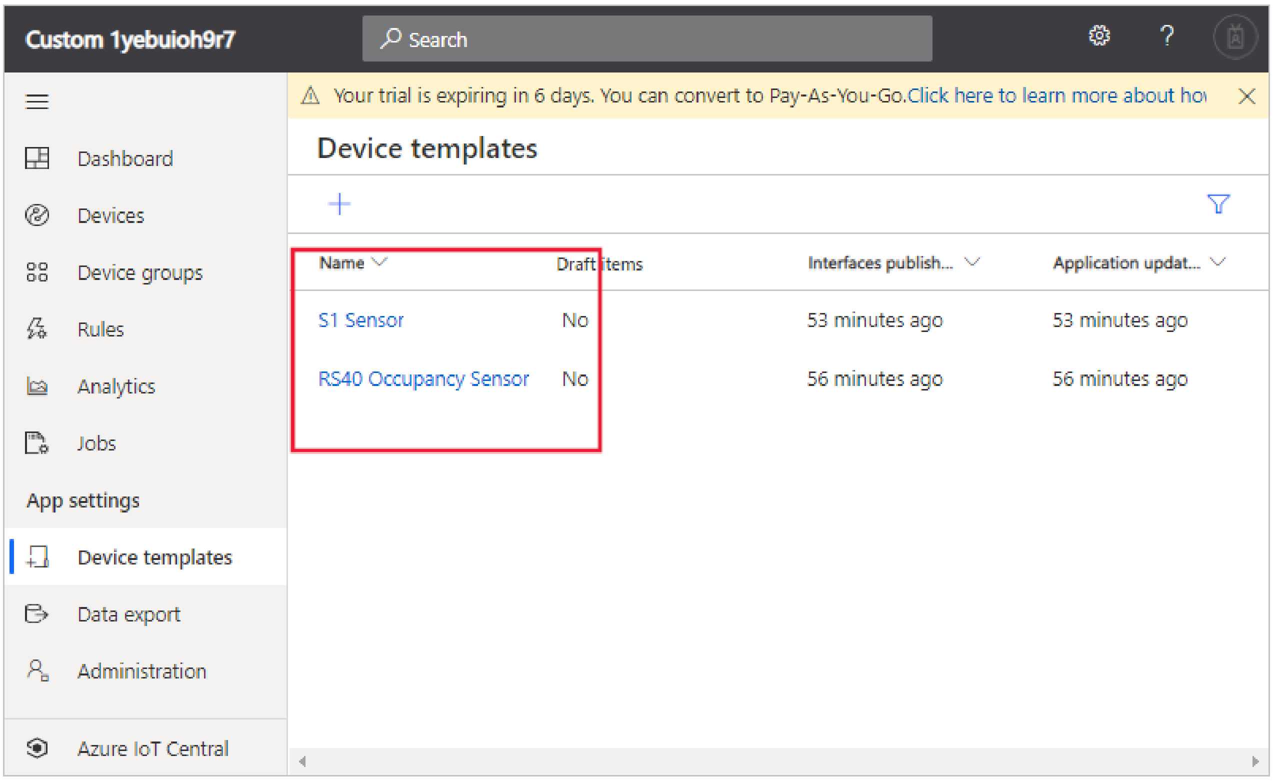
To create a device template for an ***RS40 Occupancy Sensor** device:

1. In the left pane, select **Device Templates**. Then select **+** to start adding the template.

2. Scroll down until you can see the tile for the ***RS40 Occupancy Sensor** device. Select the tile and then select **Next: Customize**.

3. On the **Review** page, select **Create** to add the device template to your application.

You now have device templates for the two downstream device types:



Create a gateway device template

In this tutorial you create a device template for a gateway device from scratch. You use this template later to create a simulated gateway device in your application.

To add a new gateway device template to your application:

1. In the left pane, select **Device Templates**. Then select **+** to start adding the template.

2. On the **Select template type** page, select the **IoT Device** tile, and then select **Next: Customize**.

3. On the **Customize device** page, select the **Gateway device** checkbox.

4. On the **Review** page, select **Create**.

5. Enter **Smart Building gateway device** as the template name.

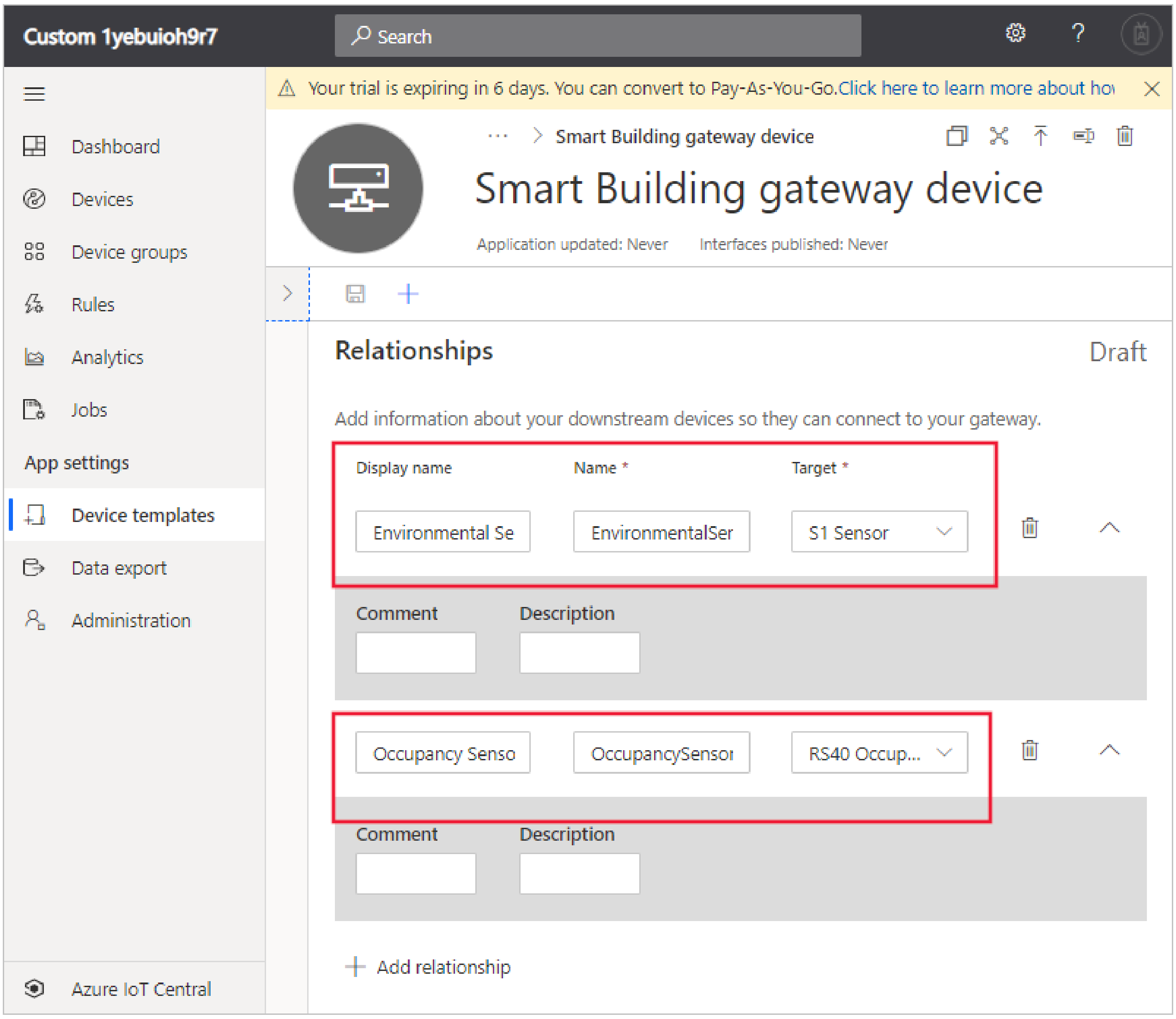
6. On the **Create a capability model** page, select the **Custom** tile.

7. Select **+** to add an interface. Choose the **Device Information** standard interface.

Add relationships

Next you add relationships to the templates for the downstream device templates:

1. In the **Smart Building gateway device** template, select **Relationships**.
2. Select **+ Add relationship**. Enter **Environmental Sensor** as the display name, and select **S1 Sensor** as the target.
3. Select **+ Add relationship** again. Enter **Occupancy Sensor** as the display name, and select **RS40 Occupancy Sensor** as the target.
4. Select **Save**.



Add cloud properties

A gateway device template can include cloud properties. Cloud properties only exist in the IoT Central application, and are never sent to, or received from, a device.

To add cloud properties to the **Smart Building gateway device** template.

1. In the **Smart Building gateway device** template, select **Cloud properties**.
2. Use the information in the following table to add two cloud properties to your gateway device template.

DISPLAY NAME	SEMANTIC TYPE	SCHEMA
Last Service Date	None	Date
Customer Name	None	String

3. Select **Save**.

Create views

As a builder, you can customize the application to display relevant information about the environmental sensor device to an operator. Your customizations enable the operator to manage the environmental sensor devices connected to the application. You can create two types of views for an operator to use to interact with devices:

- Forms to view and edit device and cloud properties.
- Dashboards to visualize devices.

To generate the default views for the **Smart Building gateway device** template:

1. In the **Smart Building gateway device** template, select **Views**.

2. Select **Generate default views** tile and make sure that all the options are selected.

3. Select **Generate default dashboard view(s)**.

Publish the device template

Before you can create a simulated gateway device, or connect a real gateway device, you need to publish your device template.

To publish the gateway device template:

1. Select the **Smart Building gateway device** template from the **Device templates** page.

2. Select **Publish**.

3. In the **Publish a Device Template** dialog box, choose **Publish**.

After a device template is published, it's visible on the **Devices** page and to the operator. In a published device template, you can't edit a device capability model without creating a new version. However, you can make updates to cloud properties, customizations, and views, in a published device template. These updates don't cause a new version to be created. After making any changes, select **Publish** to push those changes out to your operator.

Create the simulated devices

This tutorial uses simulated downstream devices and a simulated gateway device.

To create a simulated gateway device:

1. On the **Devices** page, select **Smart Building gateway device** in the list of device templates.

2. Select **+** to start adding a new device.

3. Keep the generated **Device ID** and **Device name**. Make sure that the **Simulated** switch is **On**. Select **Create**.

To create a simulated downstream devices:

1. On the **Devices** page, select **RS40 Occupancy Sensor** in the list of device templates.

2. Select **+** to start adding a new device.

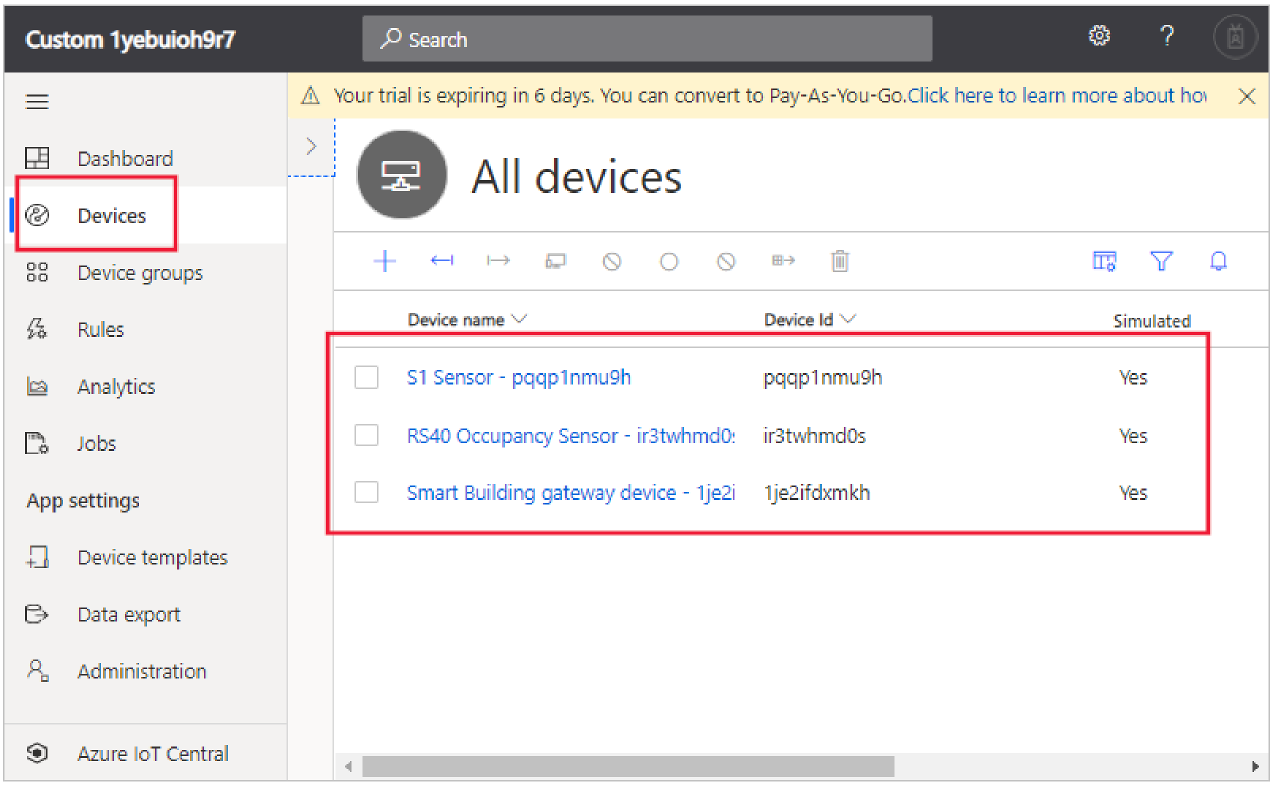
3. Keep the generated **Device ID** and **Device name**. Make sure that the **Simulated** switch is **On**. Select **Create**.

4. On the **Devices** page, select **S1 Sensor** in the list of device templates.

5. Select **+** to start adding a new device.

6. Keep the generated **Device ID** and **Device name**. Make sure that the **Simulated** switch is **On**. Select

Create.

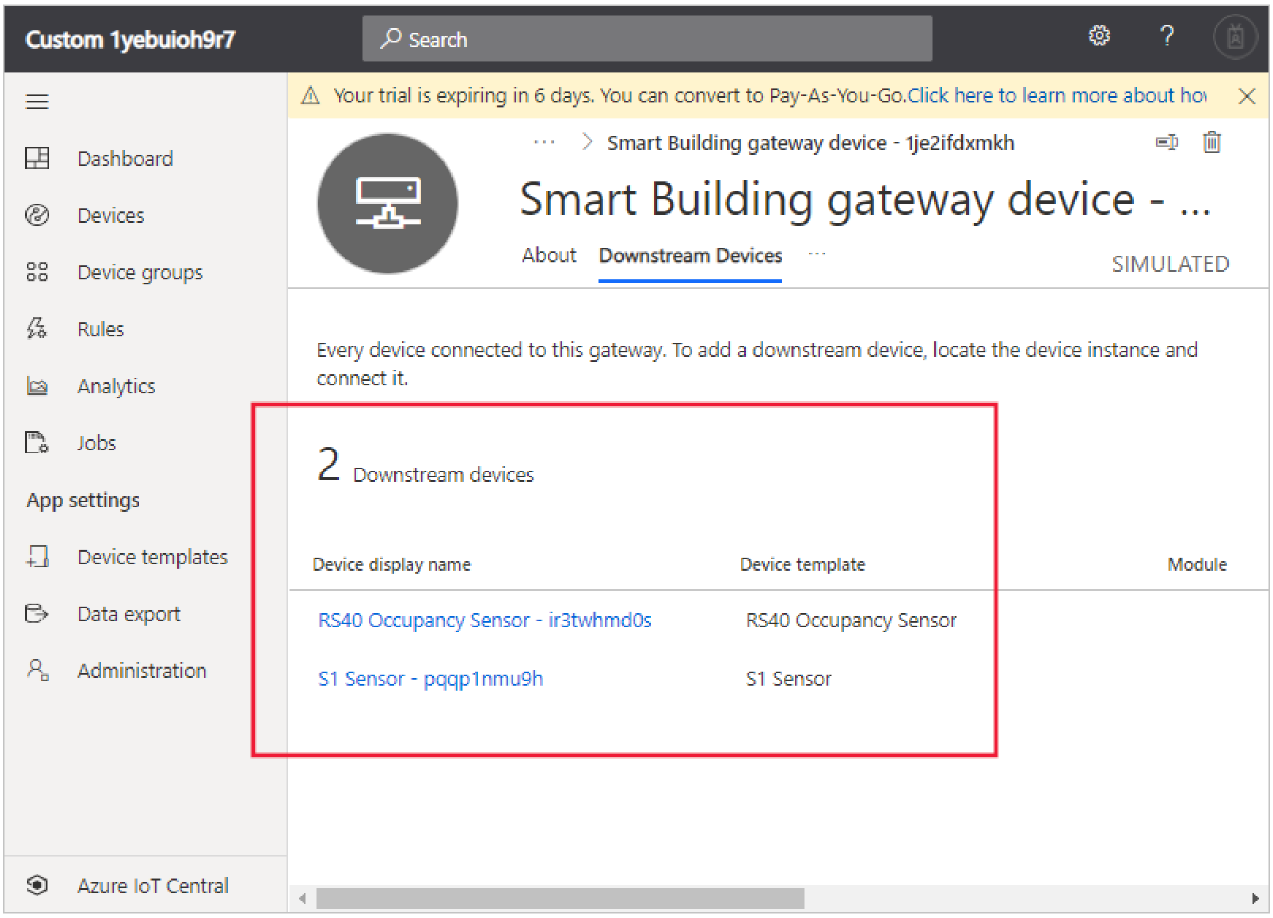


Add downstream device relationships to a gateway device

Now that you have the simulated devices in your application, you can create the relationships between the downstream devices and the gateway device:

1. On the **Devices** page, select **S1 Sensor** in the list of device templates, and then select your simulated **S1 Sensor** device.
2. Select **Connect to gateway**.
3. On the **Connect to a gateway** dialog, select the **Smart Building gateway device** template, and then select the simulated instance you created previously.
4. Select **Join**.
5. On the **Devices** page, select **RS40 Occupancy Sensor** in the list of device templates, and then select your simulated **RS40 Occupancy Sensor** device.
6. Select **Connect to gateway**.
7. On the **Connect to a gateway** dialog, select the **Smart Building gateway device** template, and then select the simulated instance you created previously.
8. Select **Join**.

Both your simulated downstream devices are now connected to your simulated gateway device. If you navigate to the **Downstream Devices** view for your gateway device, you can see the related downstream devices:



Select a gateway device template and gateway device instance, and select **Join**.

Next steps

In this tutorial, you learned how to:

- Create a new IoT gateway as a device template.
- Create cloud properties.
- Create customizations.
- Define a visualization for the device telemetry.
- Add relationships.
- Publish your device template.

> **NOTE**
> VS Code based code generation is currently not supported for gateway devices modeled in IoT Central.

Next, as a device developer, you can learn how to:

[Add an Azure IoT Edge device to your Azure IoT Central application]()

Tutorial: Add an Azure IoT Edge device to your Azure IoT Central application

7/21/2020 • 7 minutes to read • Edit Online

This article applies to operators, solution builders, and device developers.

This tutorial shows you how to configure and add an Azure IoT Edge device to your Azure IoT Central application. The tutorial uses an IoT Edge-enabled Linux virtual machine (VM) to simulate an IoT Edge device. The IoT Edge device uses a module that generates simulated environmental telemetry. You view the telemetry on a dashboard in your IoT Central application.

In this tutorial, you learn how to:

- Create a device template for an IoT Edge device
- Create an IoT Edge device in IoT Central
- Deploy a simulated IoT Edge device to a Linux VM

Prerequisites

Complete the Create an Azure IoT Central application quickstart to create an IoT Central application using the **Custom app > Custom application** template.

To complete the steps in this tutorial, you need an active Azure subscription.

If you don't have an Azure subscription, create a free account before you begin.

Download the IoT Edge manifest file from GitHub. Right-click on the following link and then select **Save link as**: EnvironmentalSensorManifest.json

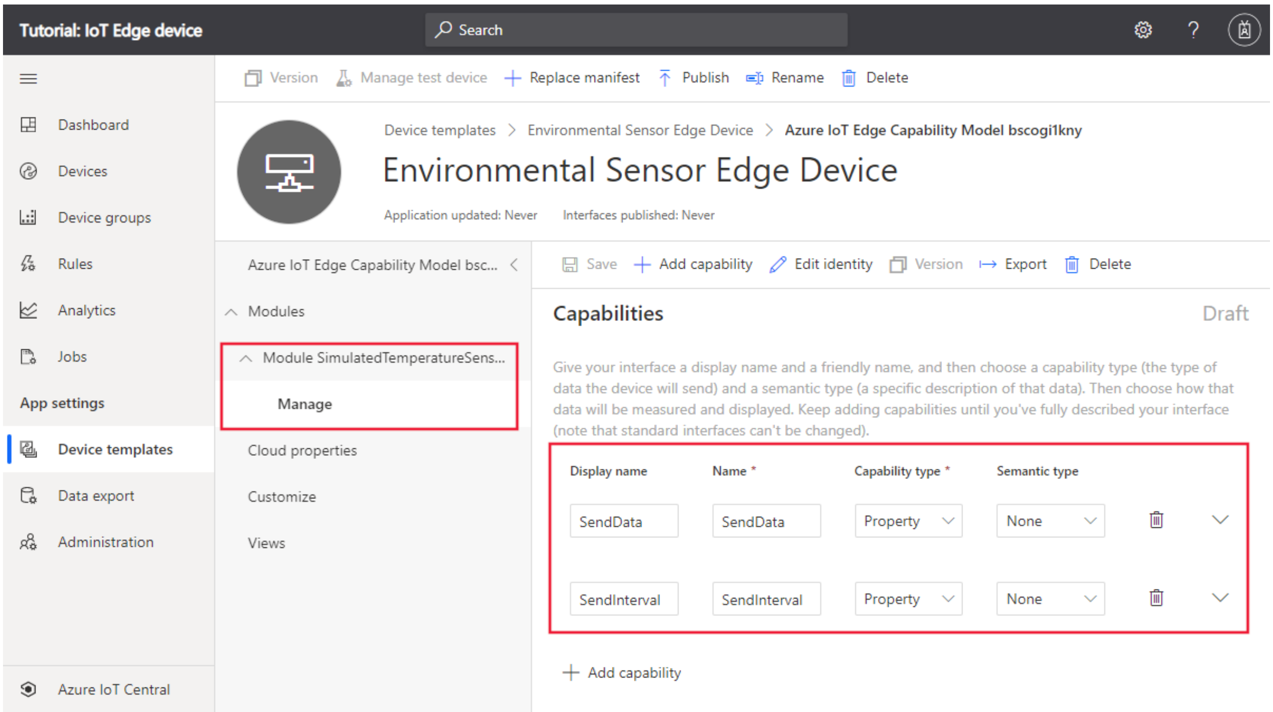
Create device template

In this section, you create an IoT Central device template for an IoT Edge device. You import an IoT Edge manifest to get started, and then modify the template to add telemetry definitions and views:

Import manifest to create template

To create a device template from an IoT Edge manifest:

1. In your IoT Central application, navigate to **Device templates** and select **+ New**.
2. On the **Select template type** page, select the **Azure IoT Edge** tile. Then select **Next: Customize**.
3. On the **Upload an Azure IoT Edge deployment manifest** page, enter *Environmental Sensor Edge Device* as the device template name. Then select **Browse** to upload the **EnvironmentalSensorManifest.json** you downloaded previously. Then select **Next: Review**.
4. On the **Review** page, select **Create**.
5. Select the **Manage** interface in the **SimulatedTemperatureSensor** module to view the two properties defined in the manifest:



Add telemetry to manifest

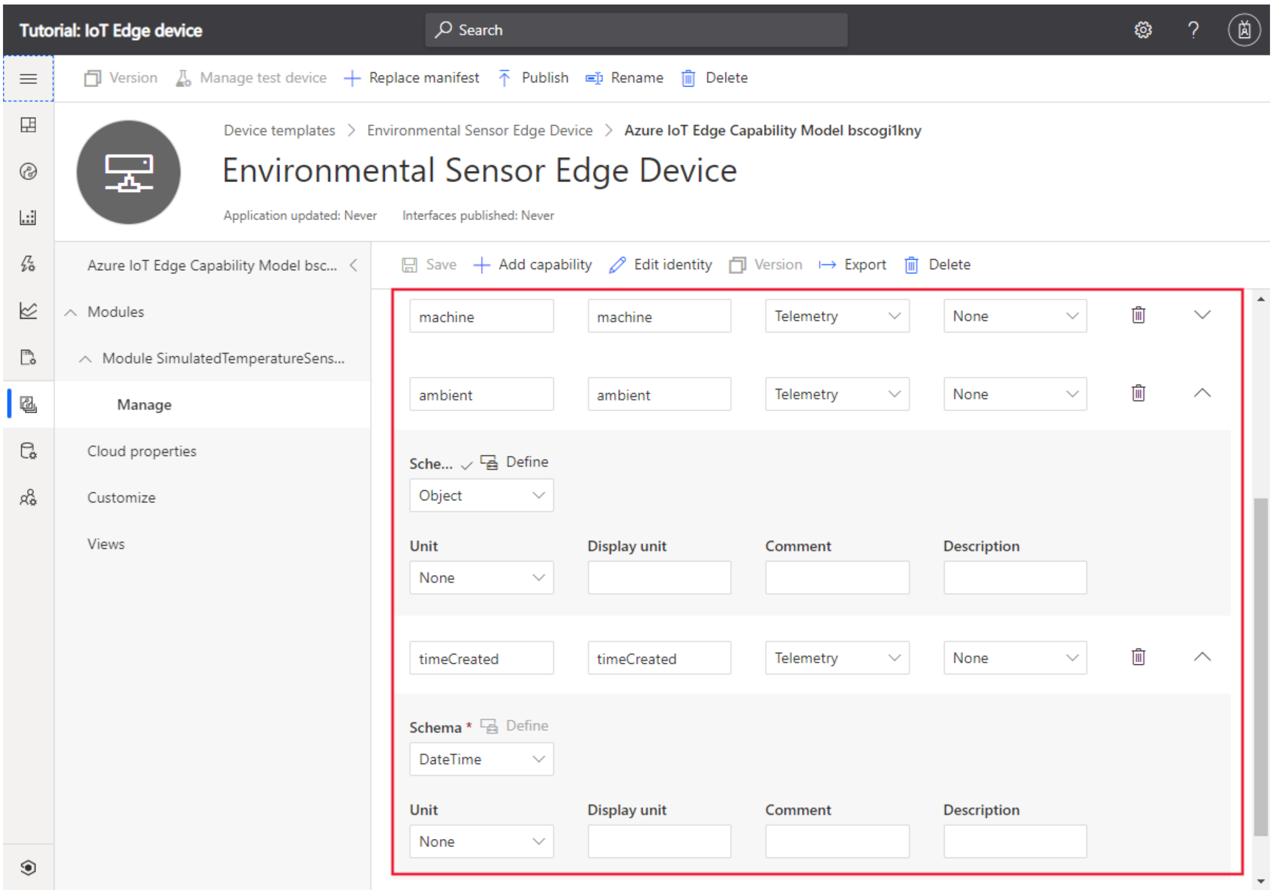
An IoT Edge manifest doesn't define the telemetry a module sends. You add the telemetry definitions to the device template in IoT Central. The **SimulatedTemperatureSensor** module sends telemetry messages that look like the following JSON:

```
{
  "machine": {
    "temperature": 75.0,
    "pressure": 40.2
  },
  "ambient": {
    "temperature": 23.0,
    "humidity": 30.0
  },
  "timeCreated": ""
}
```

To add the telemetry definitions to the device template:

1. Select the **Manage** interface in the **Environmental Sensor Edge Device** template.
2. Select **+ Add capability**. Enter *machine* as the **Display name** and make sure that the **Capability type** is **Telemetry**.
3. Select **Object** as the schema type, and then select **Define**. On the object definition page, add *temperature* and *pressure* as attributes of type **Double** and then select **Apply**.
4. Select **+ Add capability**. Enter *ambient* as the **Display name** and make sure that the **Capability type** is **Telemetry**.
5. Select **Object** as the schema type, and then select **Define**. On the object definition page, add *temperature* and *humidity* as attributes of type **Double** and then select **Apply**.
6. Select **+ Add capability**. Enter *timeCreated* as the **Display name** and make sure that the **Capability type** is **Telemetry**.
7. Select **DateTime** as the schema type.
8. Select **Save** to update the template.

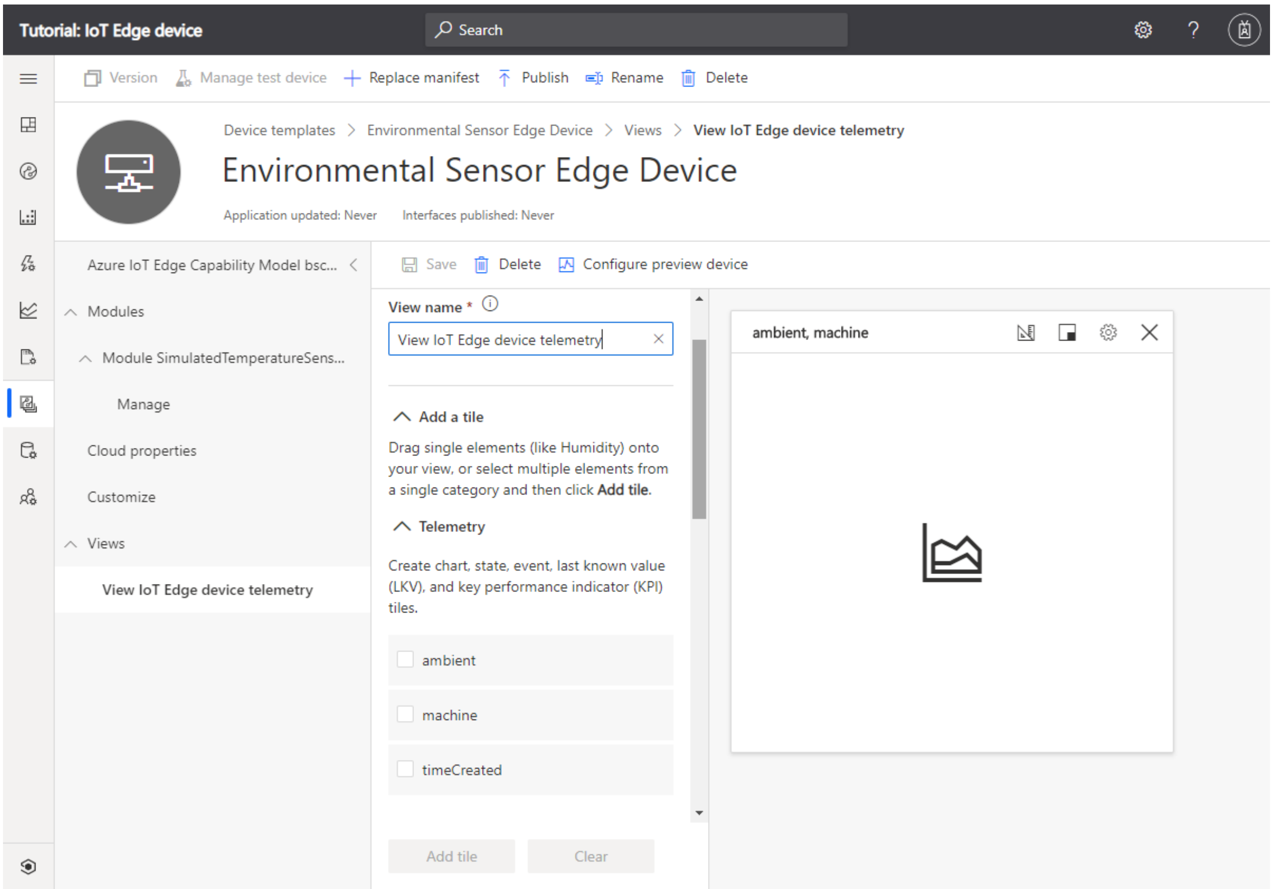
The **Manage** interface now includes the **machine**, **ambient**, and **timeCreated** telemetry types:



Add views to template

The device template doesn't yet have a view that lets an operator see the telemetry from the IoT Edge device. To add a view to the device template:

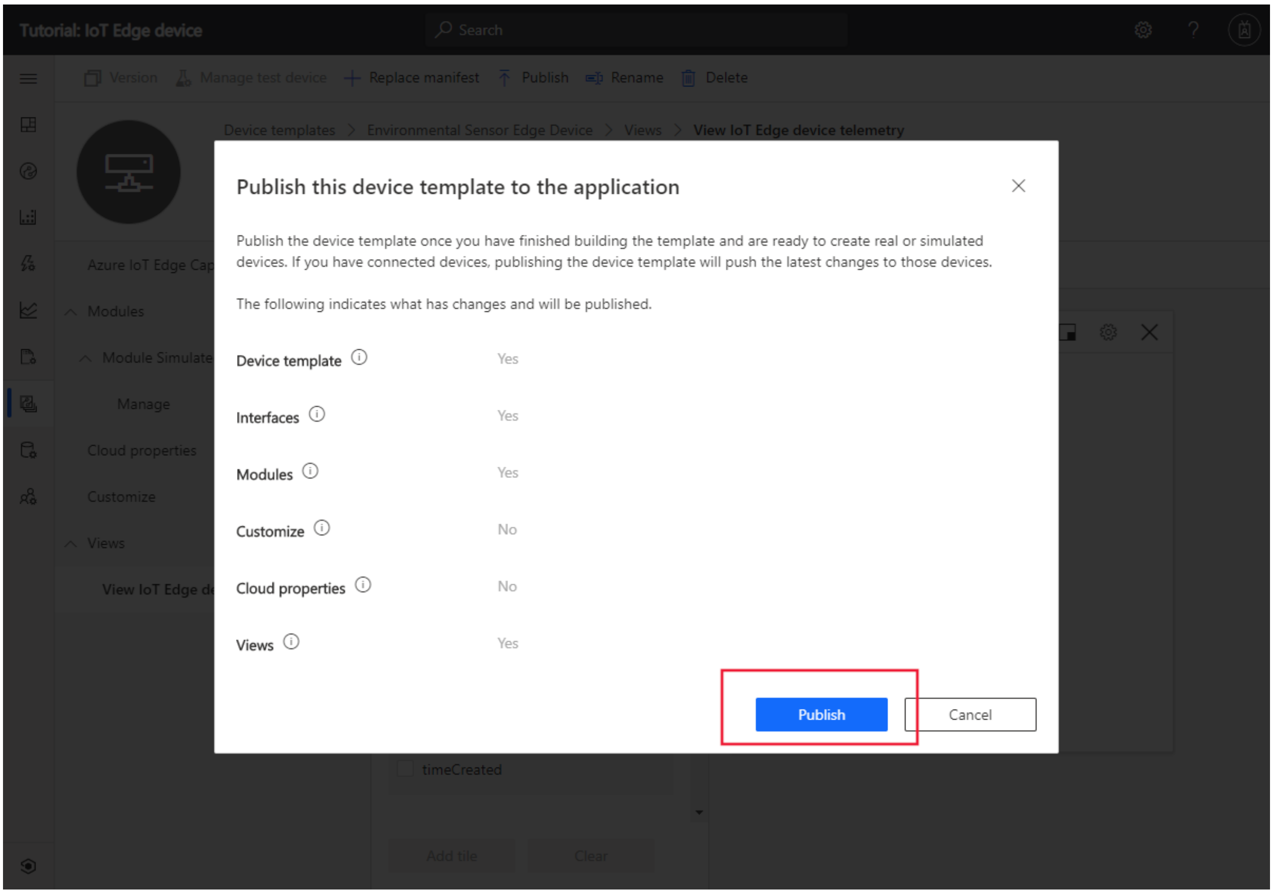
1. Select **Views** in the **Environmental Sensor Edge Device** template.
2. On the **Select to add a new view** page, select the **Visualizing the device** tile.
3. Change the view name to *View IoT Edge device telemetry*.
4. Select the **ambient** and **machine** telemetry types. Then select **Add tile**.
5. Select **Save** to save the **View IoT Edge device telemetry** view.



Publish the template

Before you can add a device that uses the **Environmental Sensor Edge Device** template, you must publish the template.

Navigate to the **Environmental Sensor Edge Device** template and select **Publish**. On the **Publish this device template to the application** panel, select **Publish** to publish the template:



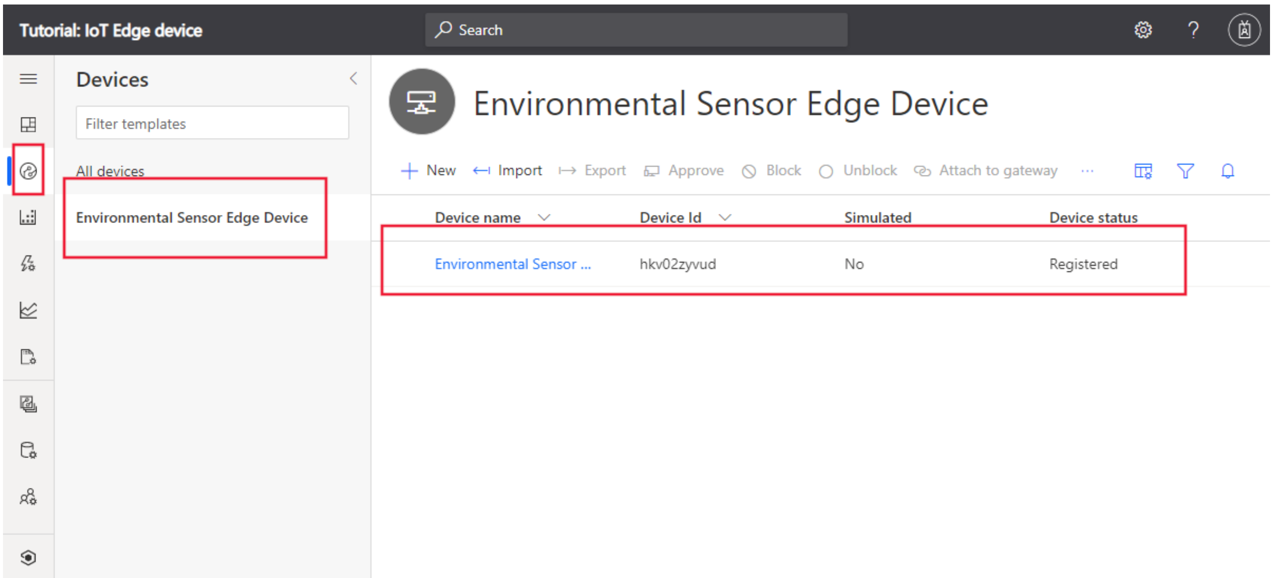
Add IoT Edge device

Now you've published the **Environmental Sensor Edge Device** template, you can add a device to your IoT Central application:

1. In your IoT Central application, navigate to the **Devices** page and select **Environmental Sensor Edge Device** in the list of available templates.

2. Select **+ New** to add a new device from the template. On the **Create new device** page, select **Create**.

You now have a new device with the status **Registered**:



Get the device credentials

When you deploy the IoT Edge device later in this tutorial, you need the credentials that allow the device to connect to your IoT Central application. The get the device credentials:

1. On the **Device** page, select the device you created.

2. Select **Connect**.

3. On the **Device connection** page, make a note of the **ID Scope**, the **Device ID**, and the **Primary Key**. You use these values later.

4. Select **Close**.

You've now finished configuring your IoT Central application to enable an IoT Edge device to connect.

Deploy an IoT Edge device

In this tutorial, you use an Azure IoT Edge-enabled Linux VM, created on Azure to simulate an IoT Edge device. To create the IoT Edge-enabled VM in your Azure subscription, click:

Deploy to Azure

On the **Custom deployment** page:

1. Select your Azure subscription.

2. Select **Create new** to create a new resource group called *central-edge-rg*.

3. Choose a region close to you.

4. Add a unique **DNS Label Prefix** such as *contoso-central-edge*.

5. Choose an admin user name for the virtual machine.

6. Enter *temp* as the connection string. Later, you configure the device to connect using DPS.

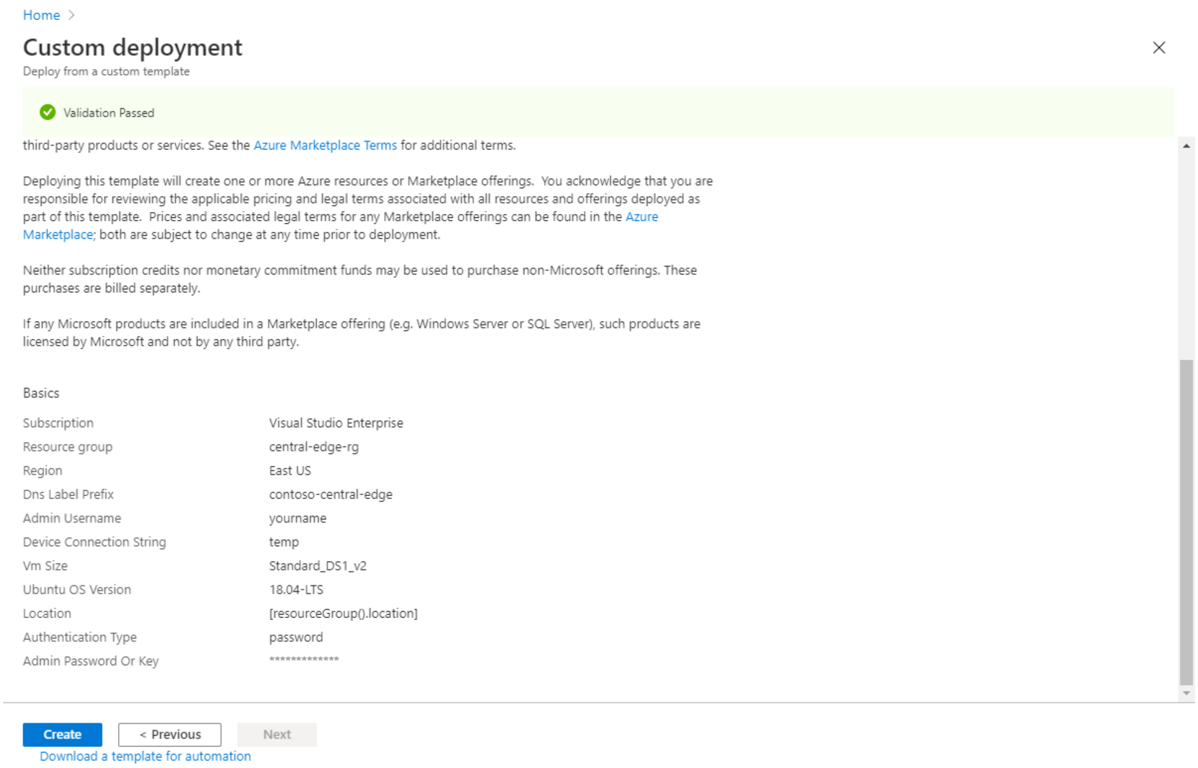
7. Accept the default values for the VM size, Ubuntu version, and location.

8. Select **password** as the authentication type.

9. Enter a password for the VM.

10. Then select **Review + Create**.

11. Review your choices and then select **Create**:



The deployment takes a couple of minutes to complete. When the deployment is complete, navigate to the **central-edge-rg** resource group in the Azure portal.

Configure the IoT Edge VM

To configure IoT Edge in the VM to use DPS to register and connect to your IoT Central application:

1. In the **contoso-edge-rg** resource group, select the virtual machine instance.

2. In the **Support + troubleshooting** section, select **Serial console**. If you're prompted to configure boot diagnostics, follow the instructions in the portal.

3. Press **Enter** to see the `login:` prompt. Enter your username and password to sign in.

4. Run the following command to check the IoT Edge runtime version. At the time of writing, the version is 1.0.9.1:

```
sudo iotedge --version
```

5. Use the `nano` editor to open the IoT Edge config.yaml file:

```
sudo nano /etc/iotedge/config.yaml
```

6. Scroll down until you see `# Manual provisioning configuration`. Comment out the next three lines as shown in the following snippet:

```
# Manual provisioning configuration
#provisioning:
#  source: "manual"
#  device_connection_string: "temp"
```

7. Scroll down until you see `# DPS symmetric key provisioning configuration`. Uncomment the next eight lines as shown in the following snippet:

```
# DPS symmetric key provisioning configuration
provisioning:
  source: "dps"
  global_endpoint: "https://global.azure-devices-provisioning.net"
  scope_id: "{scope_id}"
  attestation:
    method: "symmetric_key"
    registration_id: "{registration_id}"
    symmetric_key: "{symmetric_key}"
```

> **TIP**
>
> Make sure there's no space left in front of `provisioning:`

8. Replace `{scope_id}` with the **ID Scope** you made a note of previously.
9. Replace `{registration_id}` with the **Device ID** you made a note of previously.
10. Replace `{symmetric_key}` with the **Primary key** you made a note of previously.
11. Save the changes (**Ctrl-O**) and exit (**Ctrl-X**) the `nano` editor.
12. Run the following command to restart the IoT Edge daemon:

```
sudo systemctl restart iotedge
```

13. To check the status of the IoT Edge modules, run the following command:

```
iotedge list
```

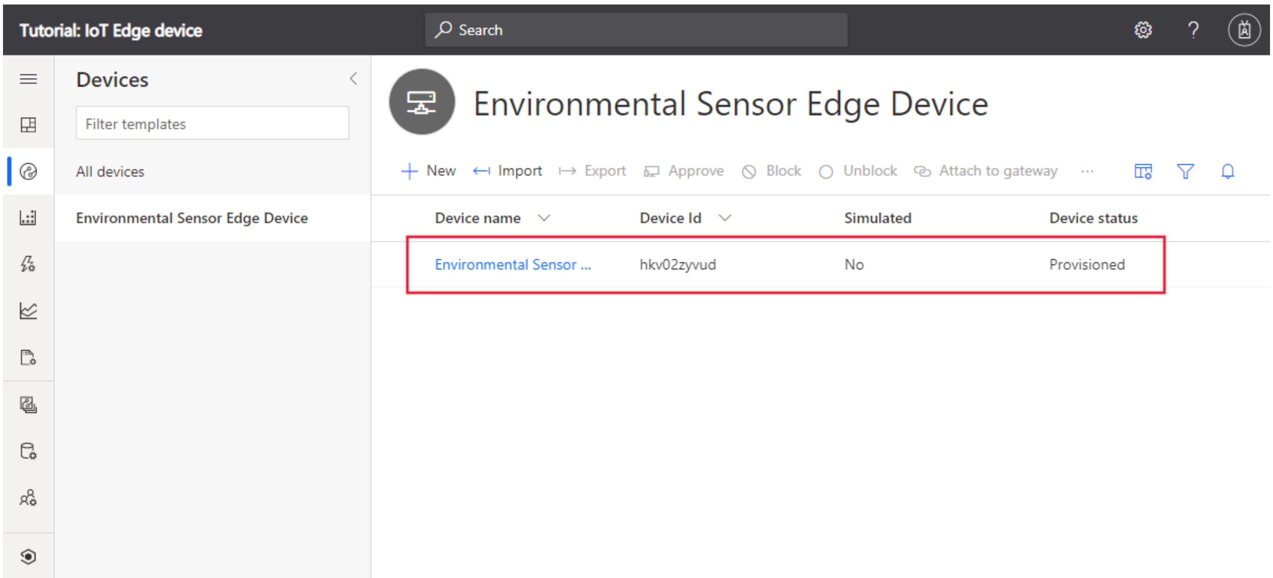
The following sample output shows the running modules:

```
NAME                        STATUS           DESCRIPTION      CONFIG
SimulatedTemperatureSensor  running          Up 20 seconds    mcr.microsoft.com/azureiotedge-simulated-
temperature-sensor:1.0
edgeAgent                   running          Up 27 seconds    mcr.microsoft.com/azureiotedge-agent:1.0
edgeHub                     running          Up 22 seconds    mcr.microsoft.com/azureiotedge-hub:1.0
```

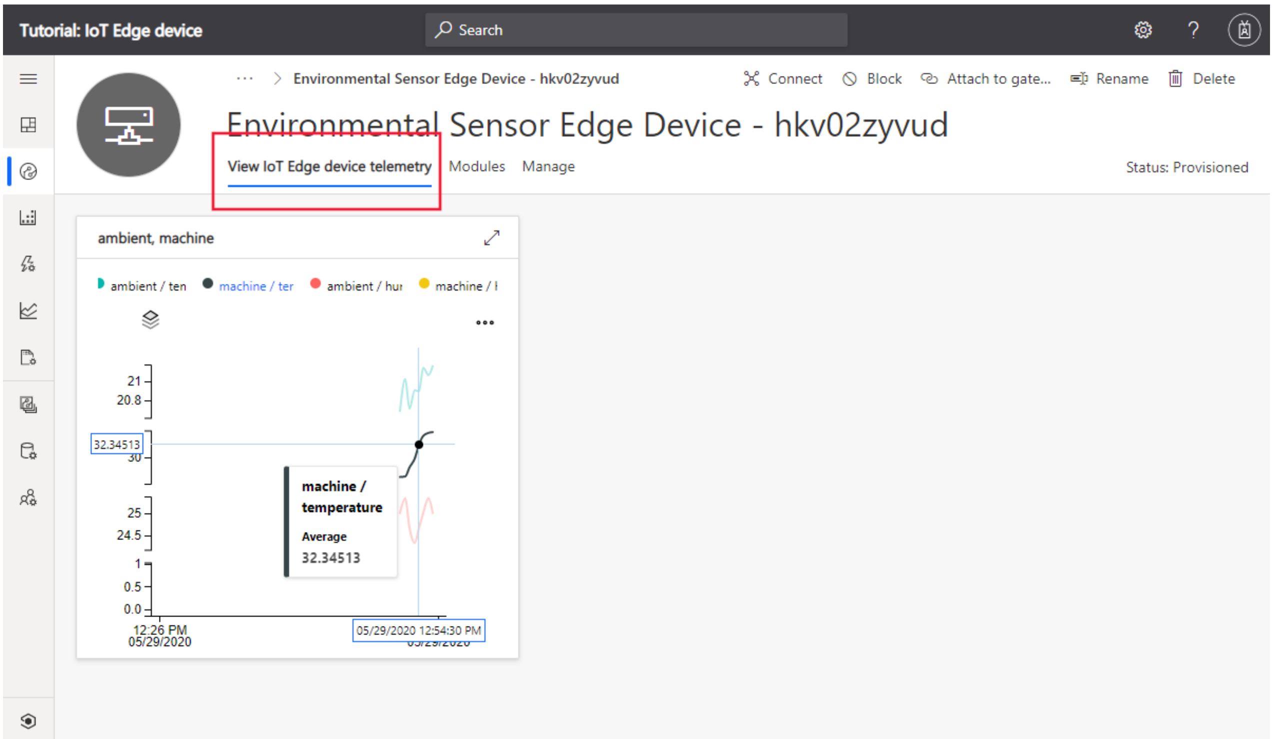
> **TIP**
>
> You may need to wait for all the modules to start running.

View the telemetry

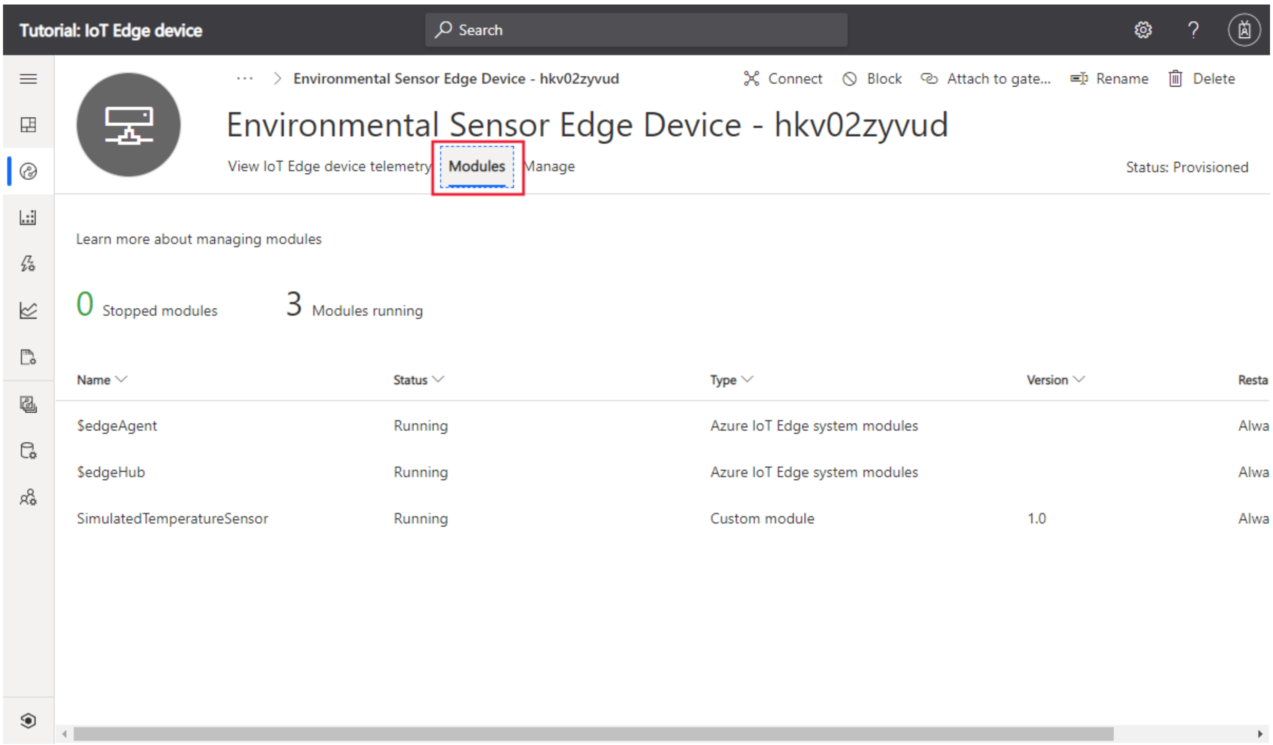
The simulated IoT Edge device is now running in the VM. In your IoT Central application, the device status is now **Provisioned** on the **Devices** page:



You can see the telemetry from the device on the **View IoT Edge device telemetry** page:



The **Modules** page shows the status of the IoT Edge modules on the device:



Clean up resources

If you plan to continue working with the IoT Edge VM, you can keep and reuse the resources you used in this tutorial. Otherwise, you can delete the resources you created in this tutorial to avoid additional charges:

- To delete the IoT Edge VM and its associated resources, delete the the **contoso-edge-rg** resource group in the Azure portal.
- To delete the IoT Central application, navigate to the **Your application** page in the **Administration** section of the application and select **Delete**.

Next steps

As a device developer, now that you've learned how to work with and manage IoT Edge devices in IoT Central, a suggested next step is to read:

Develop IoT Edge modules

As a solution developer or operator, now that you've learned how to work with and manage IoT Edge devices in IoT Central, a suggested next step is to:

Use device groups to analyze device telemetry

Tutorial: Use device groups to analyze device telemetry

4/9/2020 • 2 minutes to read • Edit Online

This article describes how, as an operator, to use device groups to analyze device telemetry in your Azure IoT Central application.

A device group is a list of devices that are grouped together because they match some specified criteria. Device groups help you manage, visualize, and analyze devices at scale by grouping devices into smaller, logical groups. For example, you can create a device group to list all the air conditioner devices in Seattle to enable a technician to find the devices for which they're responsible.

In this tutorial, you learn how to:

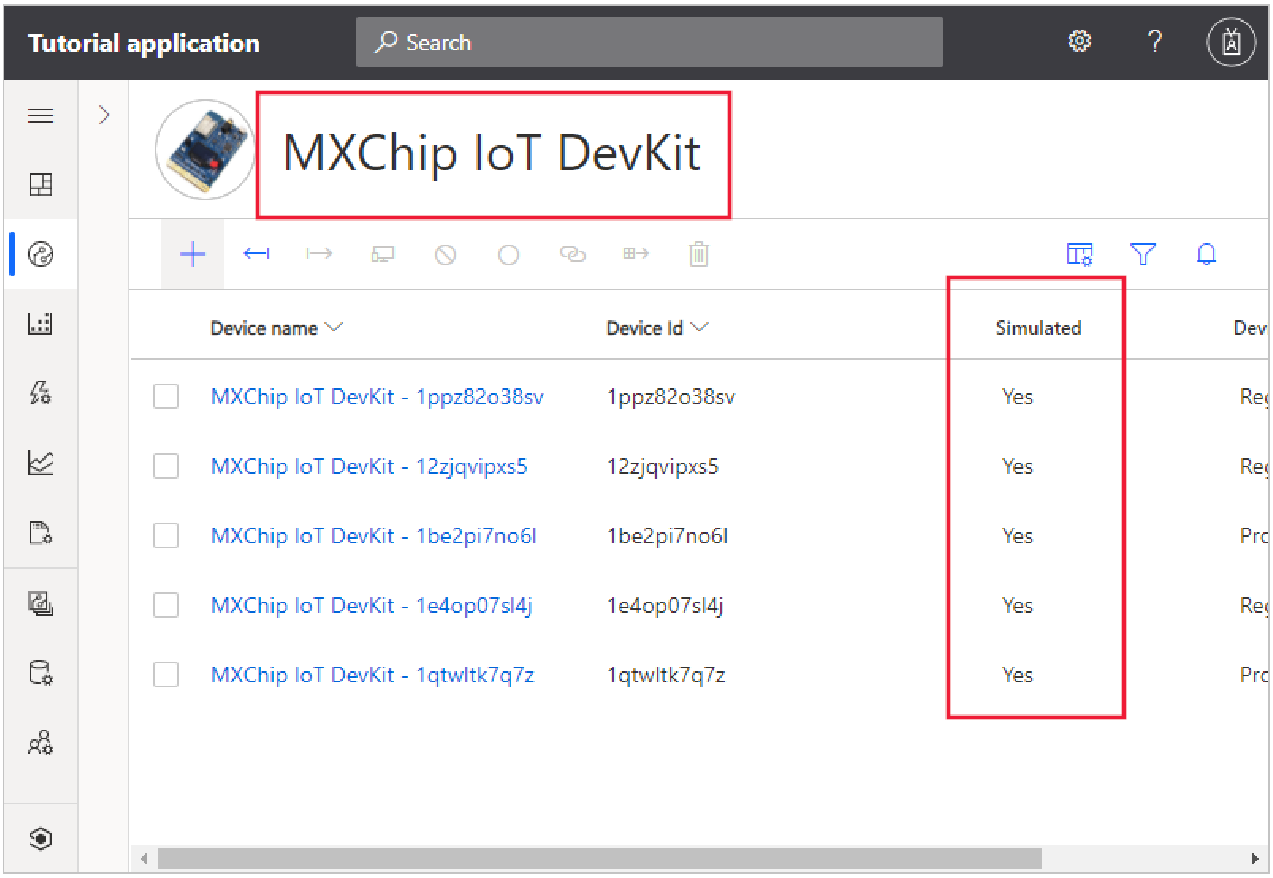
- Create a device group
- Use a device group to analyze device telemetry

Prerequisites

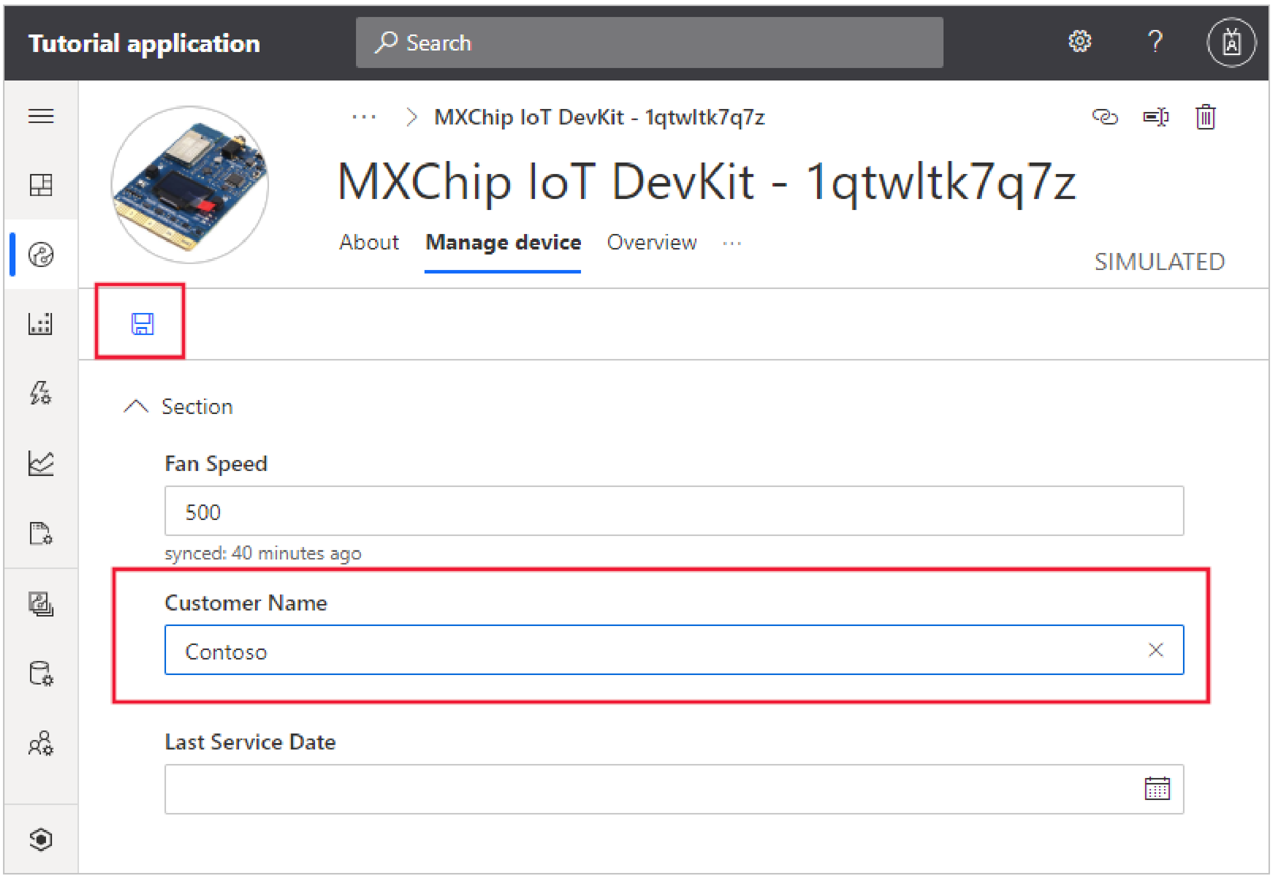
Before you begin, you should complete the Create an Azure IoT Central application and Add a simulated device to your IoT Central application quickstarts to create the **MXChip IoT DevKit** device template to work with.

Create simulated devices

Before you create a device group, add at least five simulated devices from the **MXChip IoT DevKit** device template to use in this tutorial:



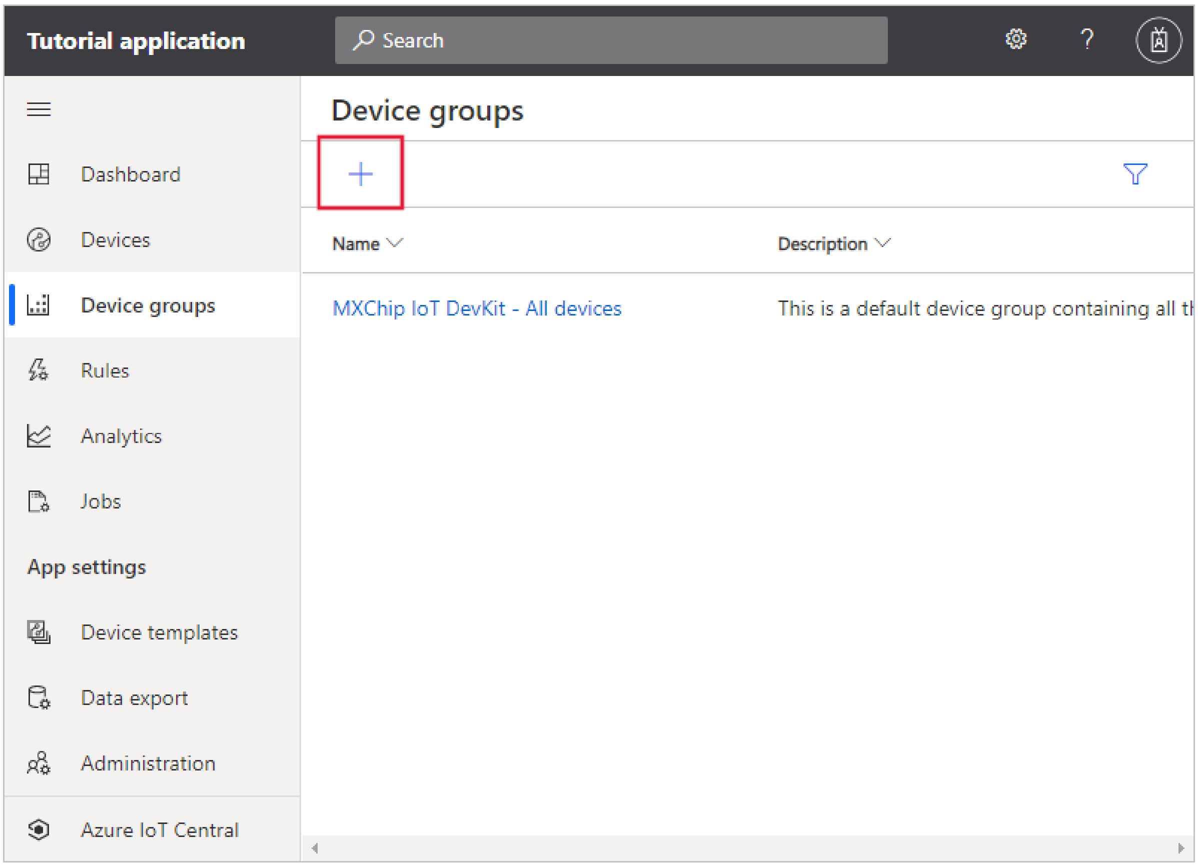
For four of the simulated sensor devices, use the **Manage device** view to set the customer name to *Contoso*:



Create a device group

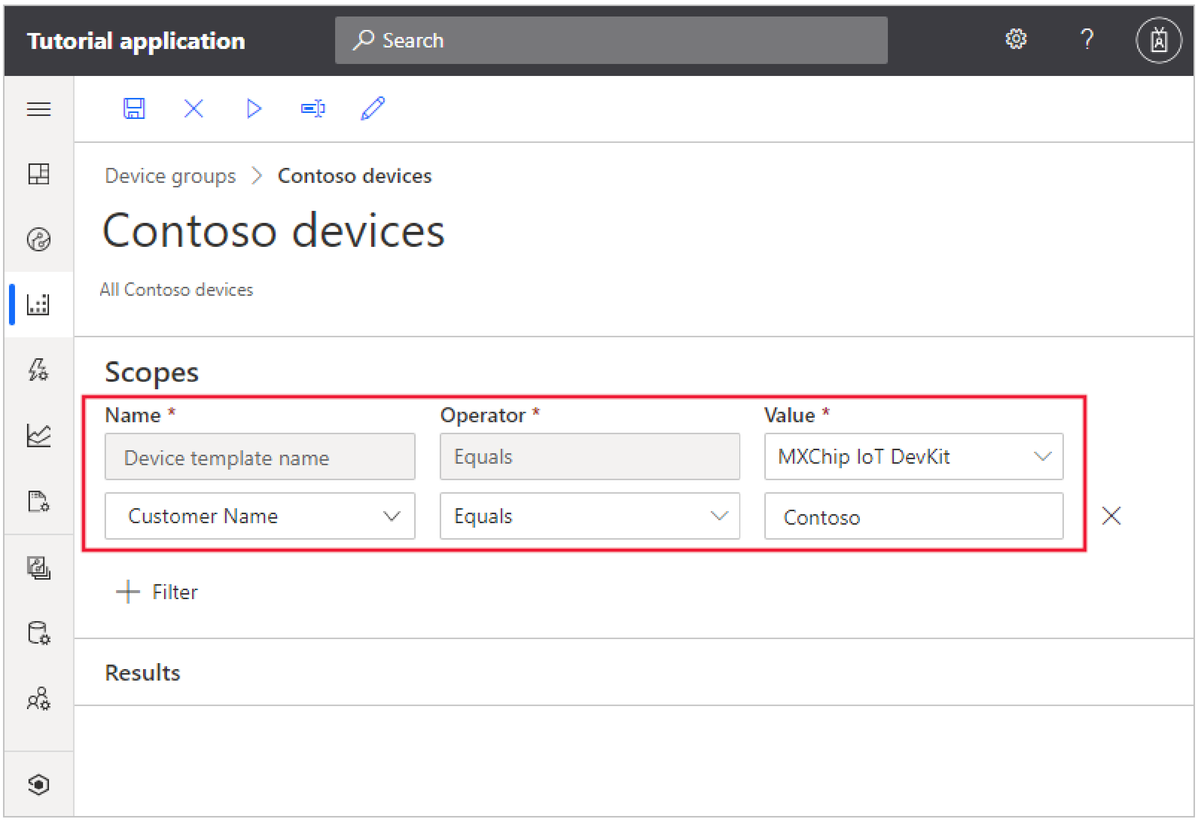
To create a device group:

1. Choose **Device groups** on the left pane.
2. Select **+**:



3. Give your device group the name *Contoso devices*. You can also add a description. A device group can only contain devices from a single device template. Choose the **MXChip IoT DevKit** device template to use for this group.

4. To customize the device group to include only the devices belonging to **Contoso**, select **+ Filter**. Select the **Customer Name** property, the **Equals** comparison operator, and **Contoso** as the value. You can add multiple filters and devices that meet **all** the filter criteria are placed in the device group. The device group you create is accessible to anyone who has access to the application, so anyone can view, modify, or delete the device group:



> **TIP**
>
> The device group is a dynamic query. Every time you view the list of devices, there may be different devices in the list. The list depends on which devices currently meet the criteria of the query.

5. Choose **Save**.

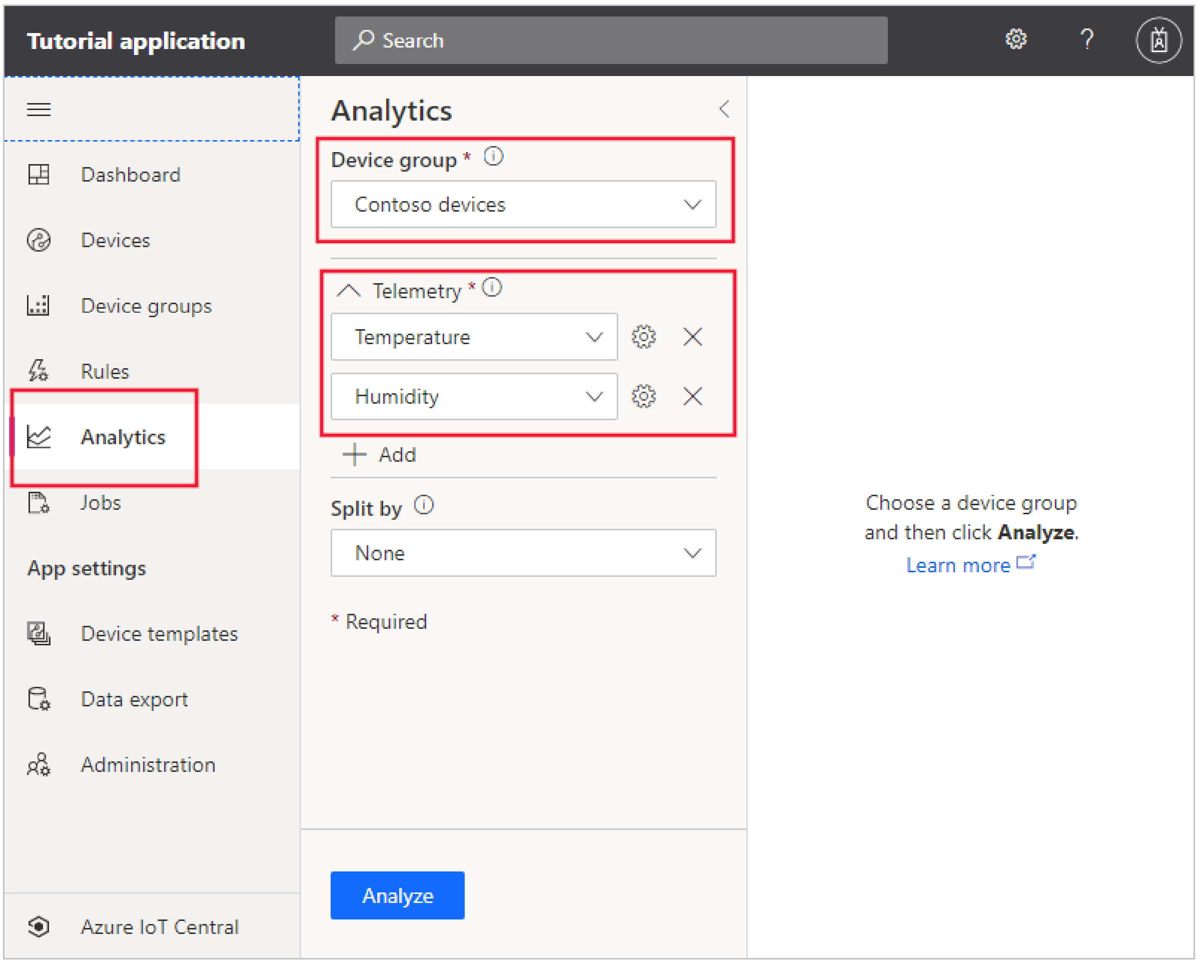
> **NOTE**
>
> For Azure IoT Edge devices, select Azure IoT Edge templates to create a device group.

Analytics

You can use **Analytics** with a device group to analyze the telemetry from the devices in the group. For example, you can plot the average temperature reported by all the Contoso environmental sensors.

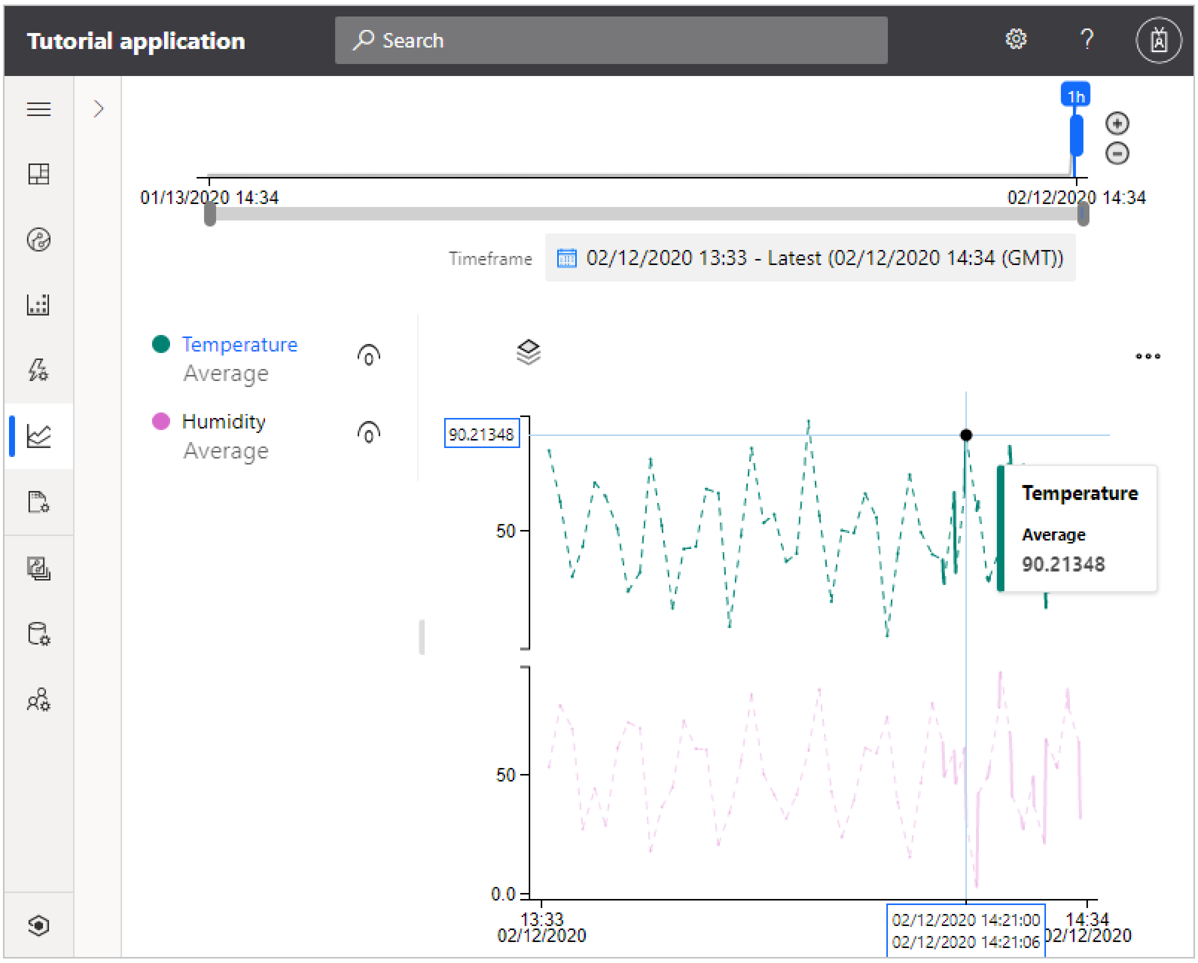
To analyze the telemetry for a device group:

1. Choose **Analytics** on the left pane.
2. Select the **Contoso devices** device group you created. Then add both the **Temperature** and **Humidity** telemetry types:



   Use the gear-wheel icons next to the telemetry types to select an aggregation type. The default is **Average**. Use **Split by** to change how the aggregate data is shown. For example, if you split by device ID you see a plot for each device when you select **Analyze**.

3. Select **Analyze** to view the average telemetry values:



You can customize the view, change the time period shown, and export the data.

Next steps

Now that you've learned how to use device groups in your Azure IoT Central application, here is the suggested next step:

[How to create telemetry rules]

Tutorial: Create a rule and set up notifications in your Azure IoT Central application

4/9/2020 • 4 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

You can use Azure IoT Central to remotely monitor your connected devices. Azure IoT Central rules let you monitor your devices in near real time and automatically invoke actions, such as sending an email. This article explains how to create rules to monitor the telemetry your devices send.

Devices use telemetry to send numerical data from the device. A rule triggers when the selected device telemetry crosses a specified threshold.

In this tutorial, you create a rule to send an email when the temperature in a simulated environmental sensor device exceeds 70° F.

In this tutorial, you learn how to:

- Create a rule
- Add an email action

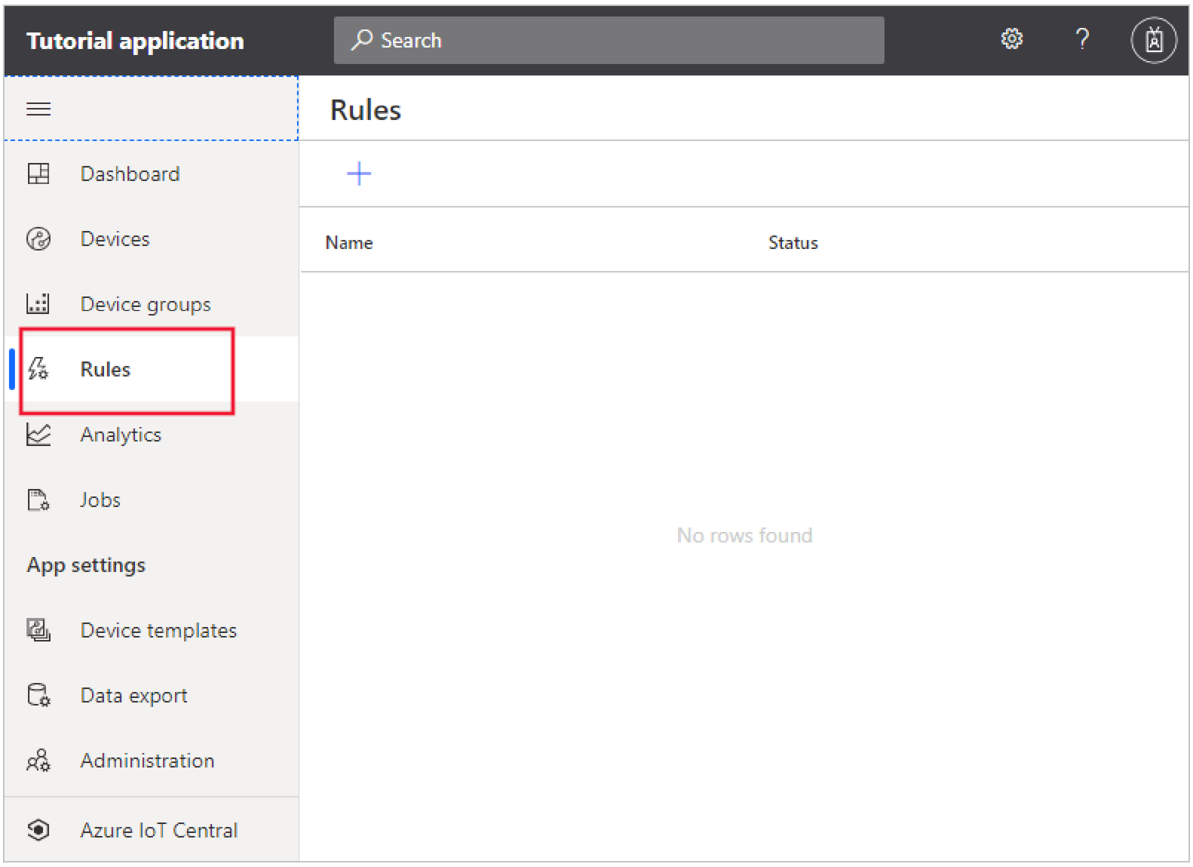
Prerequisites

Before you begin, complete the Create an Azure IoT Central application and Add a simulated device to your IoT Central application quickstarts to create the **MXChip IoT DevKit** device template to work with.

Create a rule

To create a telemetry rule, the device template must include at least one telemetry value. This tutorial uses a simulated **MXChip IoT DevKit** device that sends temperature and humidity telemetry. You added this device template and created a simulated device in the Add a simulated device to your IoT Central application quickstart. The rule monitors the temperature reported by the device and sends an email when it goes above 70 degrees.

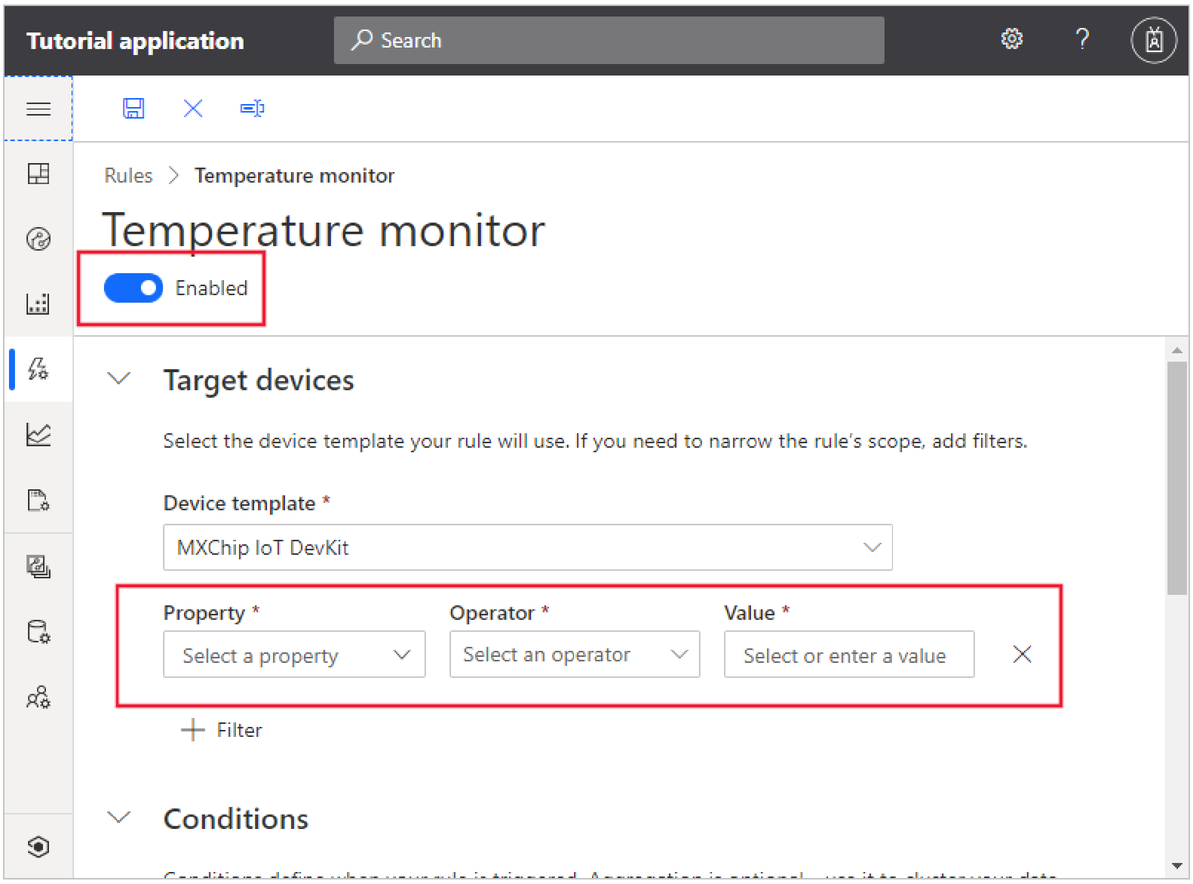
1. In the left pane, select **Rules**.
2. If you haven't created any rules yet, you see the following screen:



3. Select + to add a new rule.

4. Enter the name *Temperature monitor* to identify the rule and press Enter.

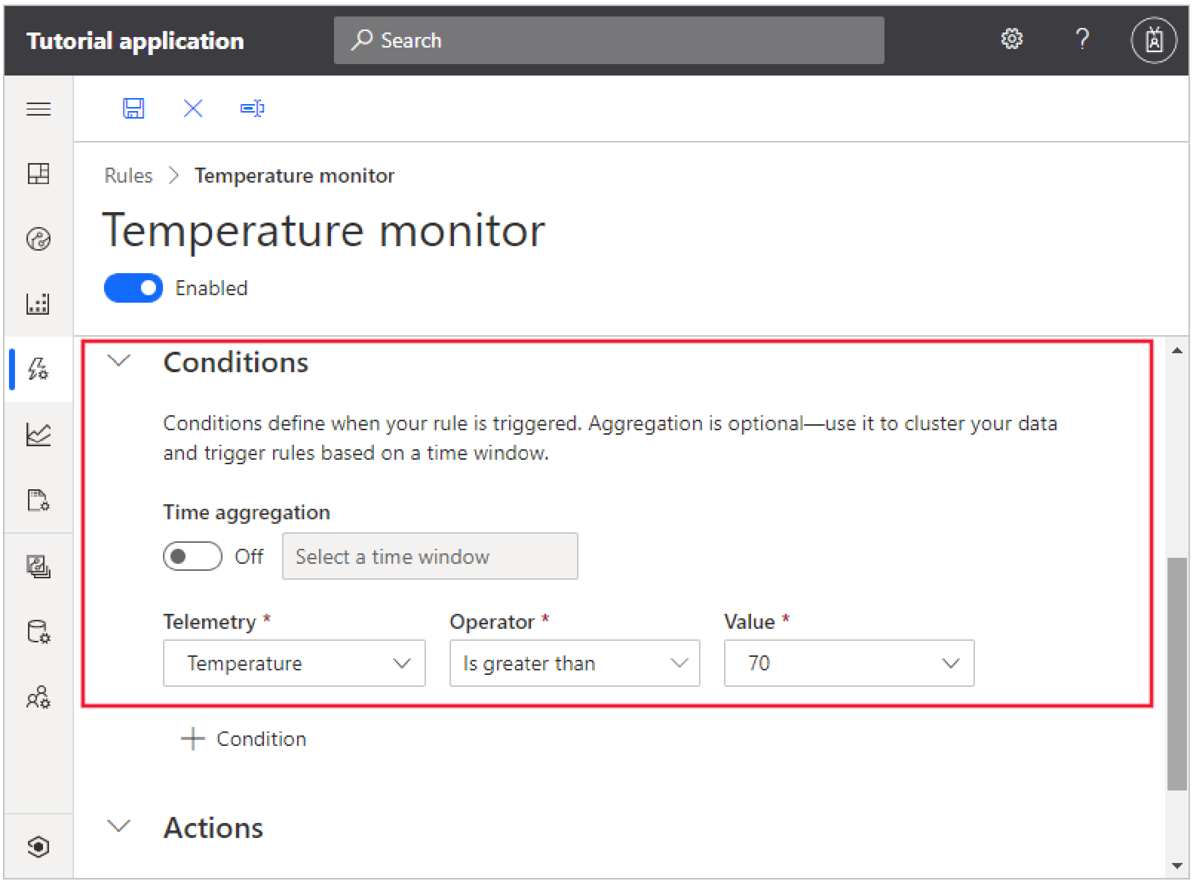
5. Select the **MXChip IoT DevKit** device template. By default, the rule automatically applies to all the devices associated with the device template. To filter for a subset of the devices, select **+ Filter** and use device properties to identify the devices. To disable the rule, toggle the **Enabled/Disabled** button in the rule header:



Configure the rule conditions

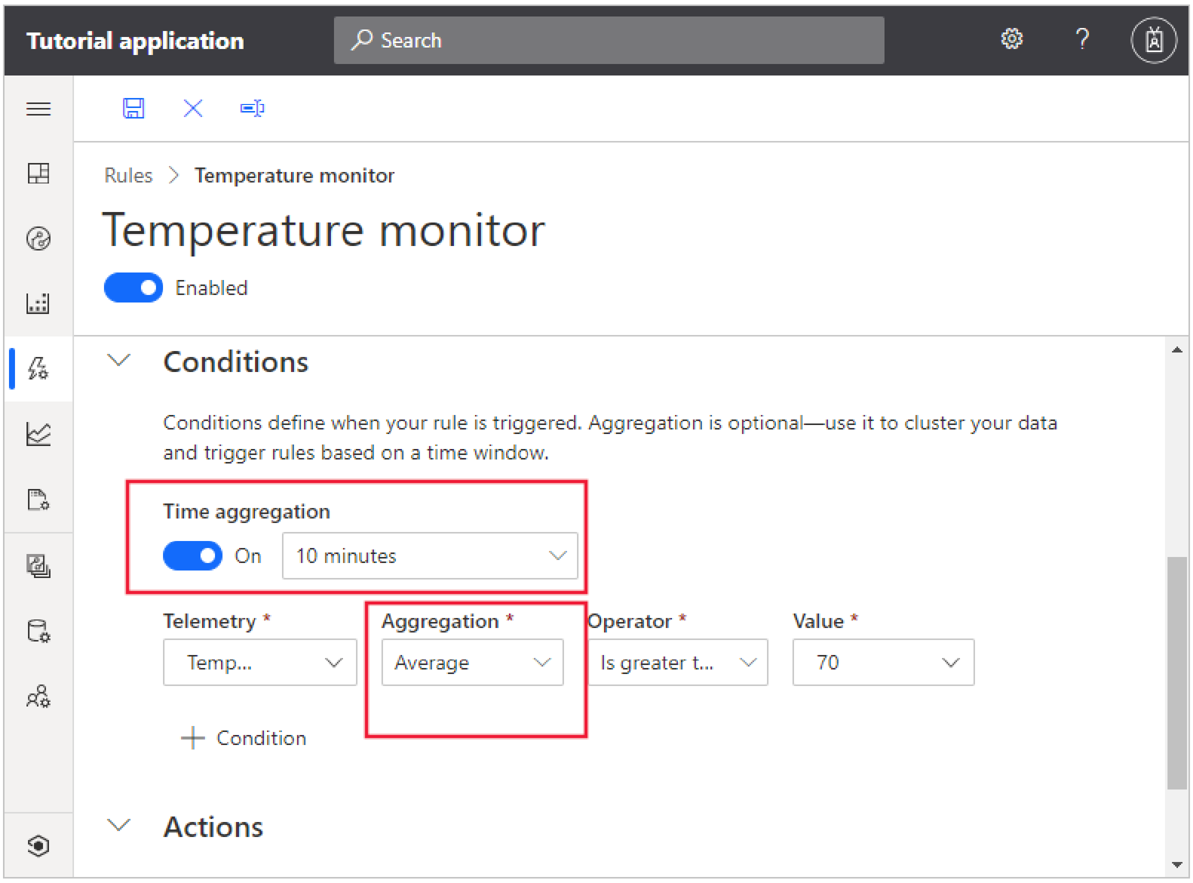
Conditions define the criteria that the rule monitors. In this tutorial, you configure the rule to fire when the temperature exceeds 70° F.

1. Select **Temperature** in the **Telemetry** dropdown.
2. Next, choose **Is greater than** as the **Operator** and enter *70* as the **Value**.



3. Optionally, you can set a **Time aggregation**. When you select a time aggregation, you must also select an aggregation type, such as average or sum from the aggregation drop-down.

- Without aggregation, the rule triggers for each telemetry data point that meets the condition. For example, if you configure the rule to trigger when temperature is above 70 then the rule triggers almost instantly when the device temperature exceeds this value.
- With aggregation, the rule triggers if the aggregate value of the telemetry data points in the time window meets the condition. For example, if you configure the rule to trigger when temperature is above 70 and with an average time aggregation of 10 minutes, then the rule triggers when the device reports an average temperature greater than 70, calculated over a 10-minute interval.



You can add multiple conditions to a rule by selecting **+ Condition**. When multiple conditions are specified, all the conditions must be met for the rule to trigger. Each condition is joined by an implicit `AND` clause. If you're using time aggregation with multiple conditions, all the telemetry values must be aggregated.

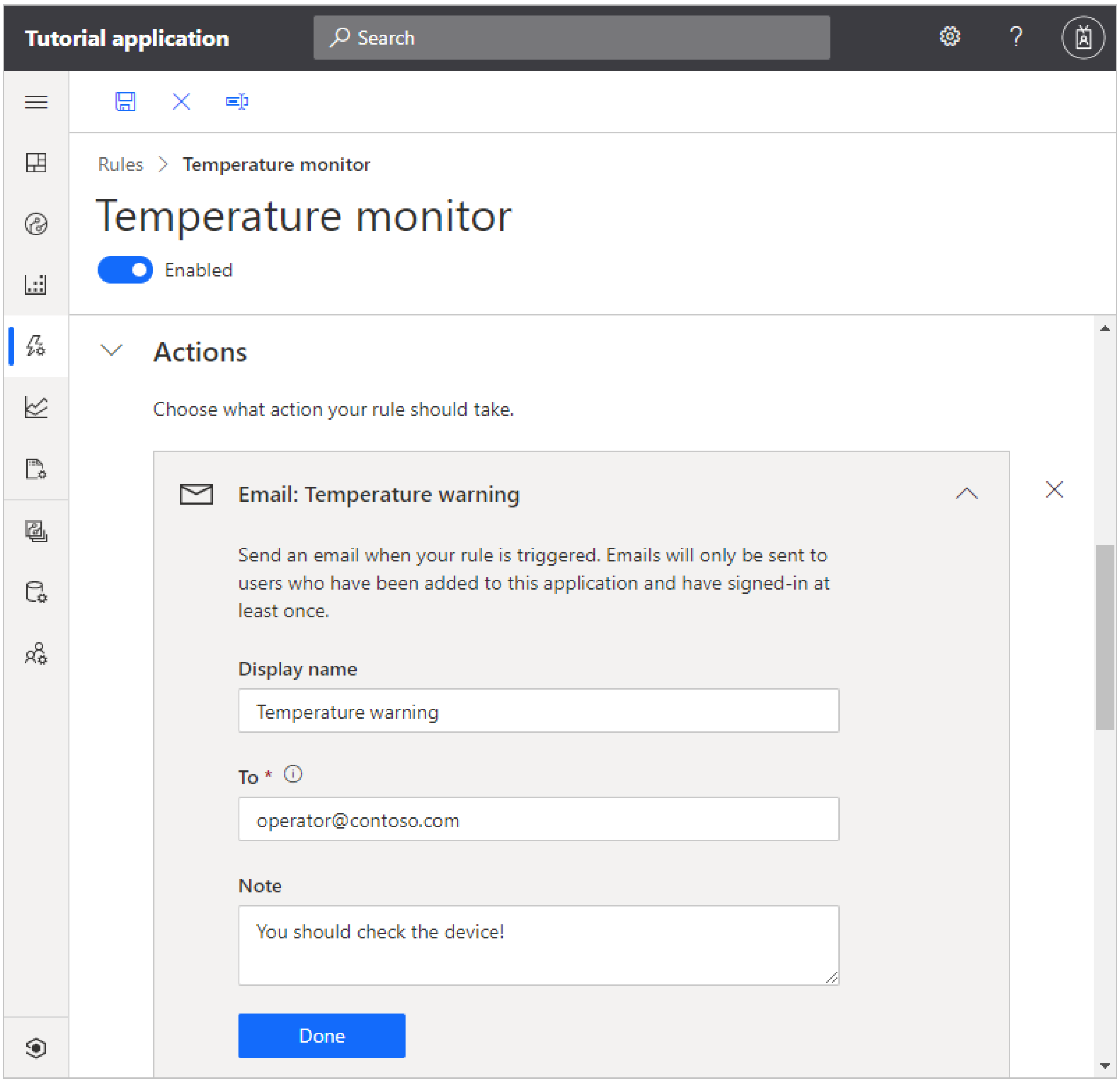
Configure actions

After you define the condition, you set up the actions to take when the rule fires. Actions are invoked when all the conditions specified in the rule evaluate to true.

1. Select **+ Email** in the **Actions** section.

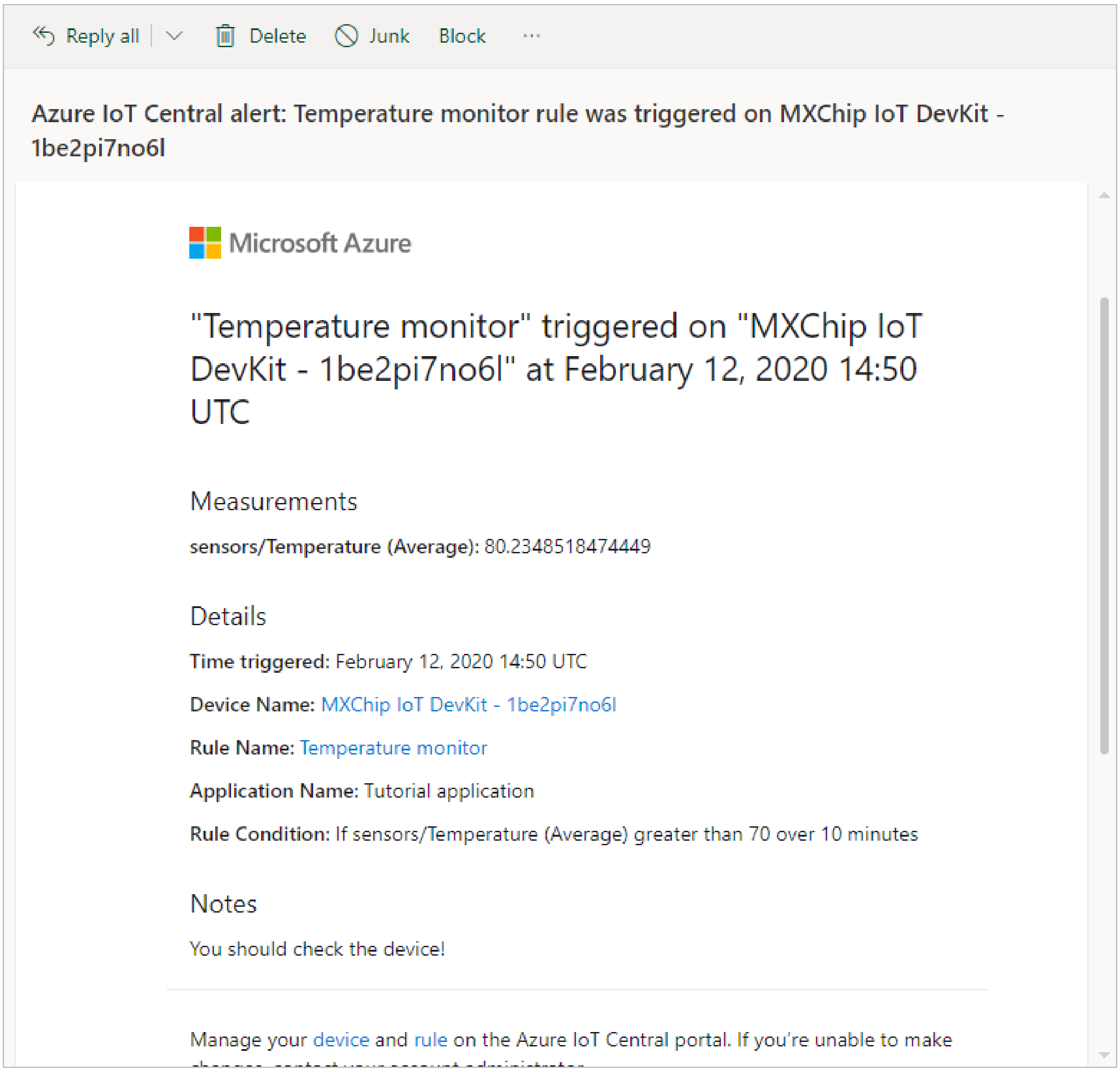
2. Enter *Temperature warning* as the display name for the action, your email address in the **To** field, and *You should check the device!* as a note to appear in the body of the email.

> **NOTE**
>
> Emails are only sent to the users that have been added to the application and have logged in at least once. Learn more about user management in Azure IoT Central.



3. To save the action, choose **Done**. You can add multiple actions to a rule.
4. To save the rule, choose **Save**. The rule goes live within a few minutes and starts monitoring telemetry being sent to your application. When the condition specified in the rule is met, the rule triggers the configured email action.

After a while, you receive an email message when the rule fires:



Delete a rule

If you no longer need a rule, delete it by opening the rule and choosing **Delete**.

Enable or disable a rule

Choose the rule you want to enable or disable. Toggle the **Enabled/Disabled** button in the rule to enable or disable the rule for all devices that are scoped in the rule.

Enable or disable a rule for specific devices

Choose the rule you want to customize. Use one or more filters in the **Target devices** section to narrow the scope of the rule to the devices you want to monitor.

Next steps

In this tutorial, you learned how to:

- Create a telemetry-based rule
- Add an action

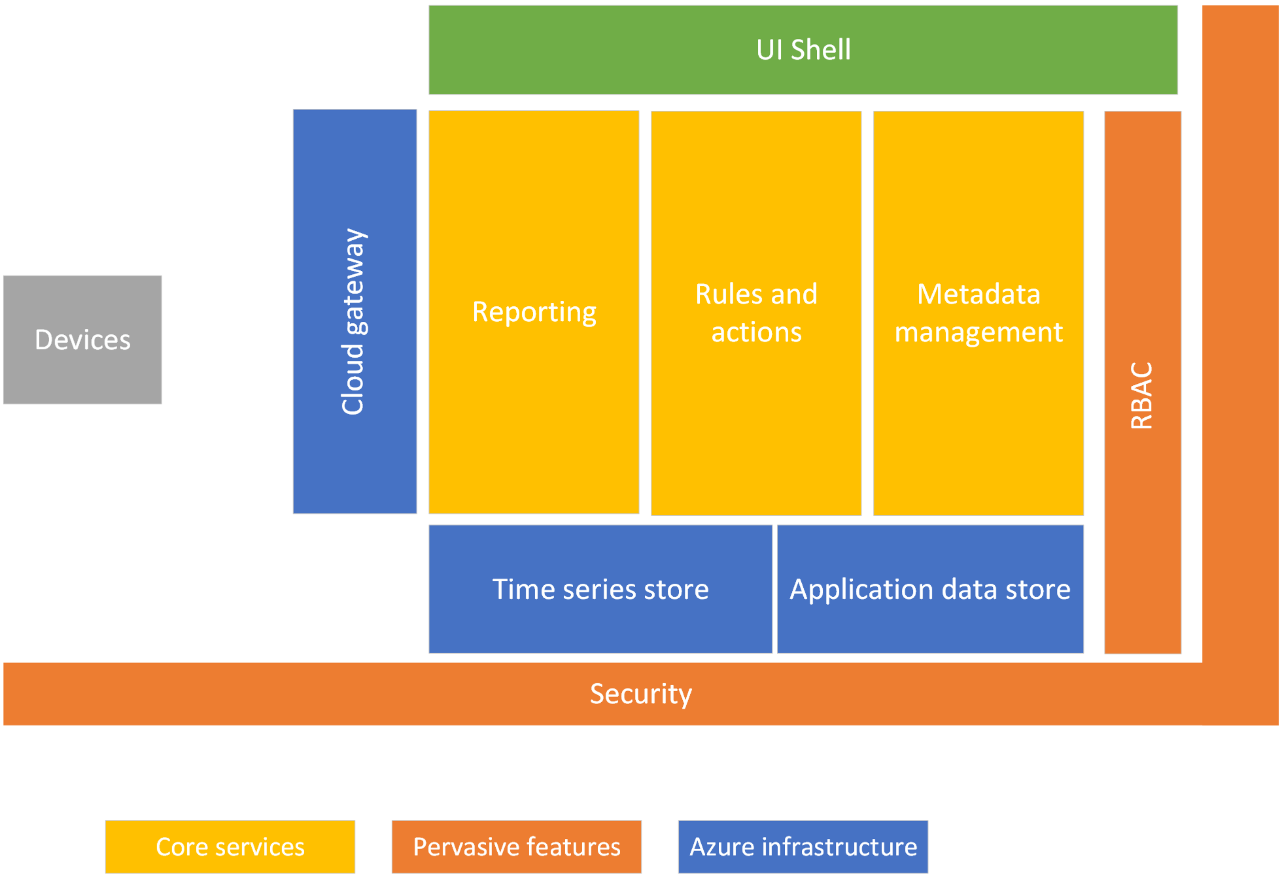
Now that you've defined a threshold-based rule the suggested next step is to learn how to:

Configure continuous data export.

Azure IoT Central architecture

2/4/2020 • 6 minutes to read • Edit Online

This article provides an overview of the Microsoft Azure IoT Central architecture.



Devices

Devices exchange data with your Azure IoT Central application. A device can:

- Send measurements such as telemetry.
- Synchronize settings with your application.

In Azure IoT Central, the data that a device can exchange with your application is specified in a device template. For more information about device templates, see Metadata management.

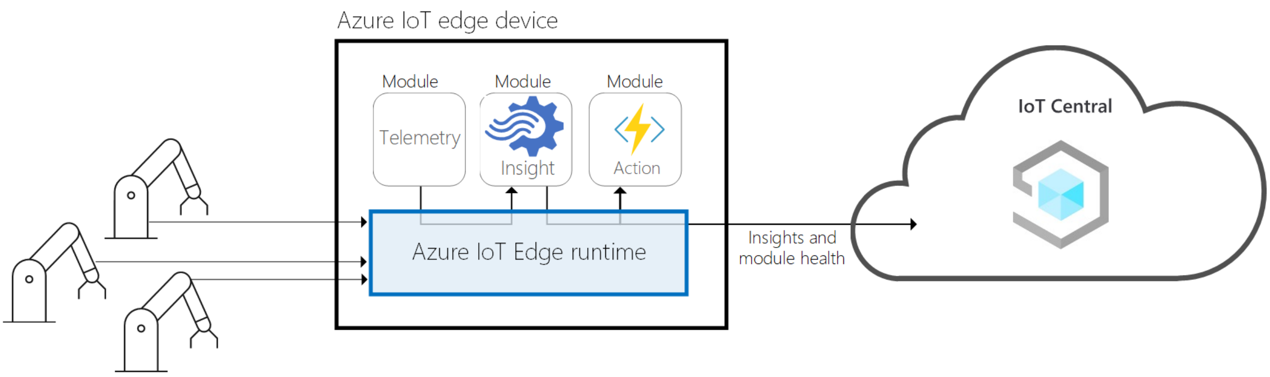
To learn more about how devices connect to your Azure IoT Central application, see Device connectivity.

Azure IoT Edge devices

As well as devices created using the Azure IoT SDKs, you can also connect Azure IoT Edge devices to an IoT Central application. IoT Edge lets you run cloud intelligence and custom logic directly on IoT devices managed by IoT Central. The IoT Edge runtime enables you to:

- Install and update workloads on the device.
- Maintain IoT Edge security standards on the device.
- Ensure that IoT Edge modules are always running.
- Report module health to the cloud for remote monitoring.

- Manage communication between downstream leaf devices and an IoT Edge device, between modules on an IoT Edge device, and between an IoT Edge device and the cloud.



IoT Central enables the following capabilities to for IoT Edge devices:

- Device templates to describe the capabilities of an IoT Edge device, such as:
  - Deployment manifest upload capability, which helps you manage a manifest for a fleet of devices.
  - Modules that run on the IoT Edge device.
  - The telemetry each module sends.
  - The properties each module reports.
  - The commands each module responds to.
  - The relationships between an IoT Edge gateway device capability model and downstream device capability model.
  - Cloud properties that aren't stored on the IoT Edge device.
  - Customizations, dashboards, and forms that are part of your IoT Central application.

  For more information, see the Connect Azure IoT Edge devices to an Azure IoT Central application article.

- The ability to provision IoT Edge devices at scale using Azure IoT device provisioning service

- Rules and actions.

- Custom dashboards and analytics.

- Continuous data export of telemetry from IoT Edge devices.

IoT Edge device types

IoT Central classifies IoT Edge device types as follows:

- Leaf devices. An IoT Edge device can have downstream leaf devices, but these devices aren't provisioned in IoT Central.
- Gateway devices with downstream devices. Both gateway device and downstream devices are provisioned in IoT Central

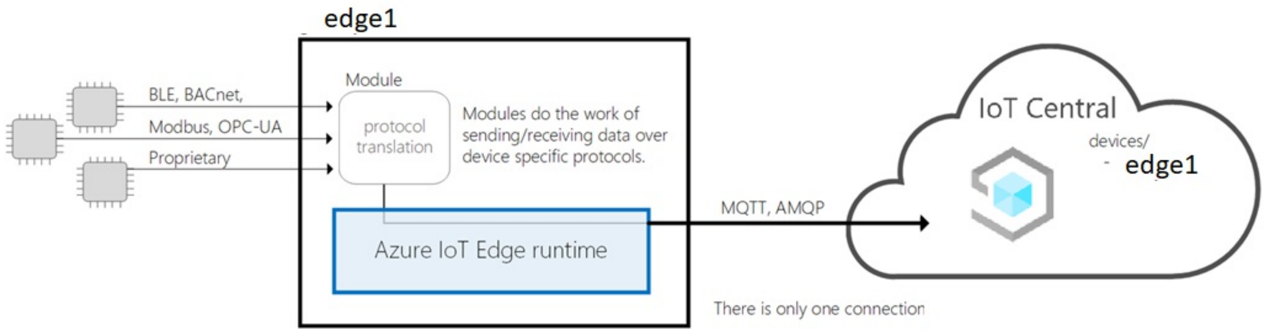
Gateway Device?

Azure IoT Edge?	Yes	No
Yes	Azure IoT Edge Gateway Device ✓ Module Management ✓ Child devices provisioned in Central	Standalone Azure IoT Edge Device ✓ Module Management × Child Devices (if exists will not be provisioned in Central)

IoT Edge patterns

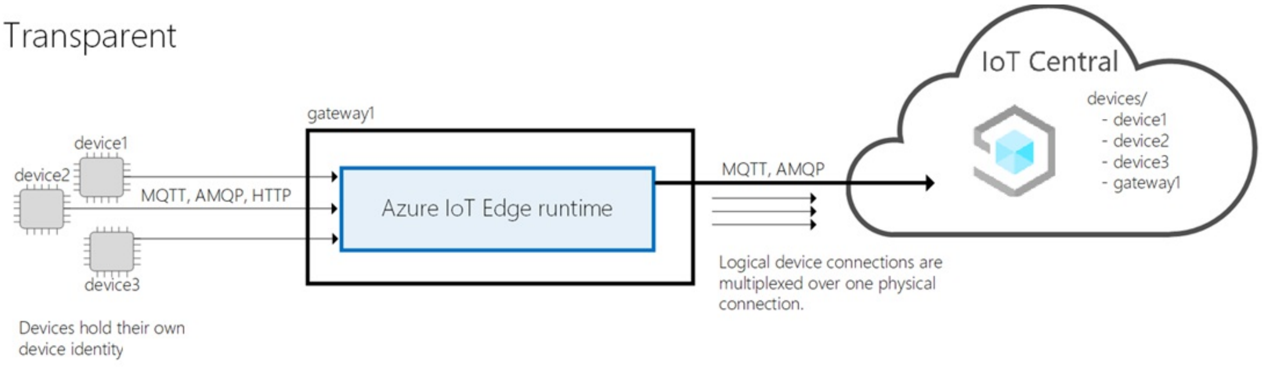
IoT Central supports the following IoT Edge device patterns:

IoT Edge as leaf device



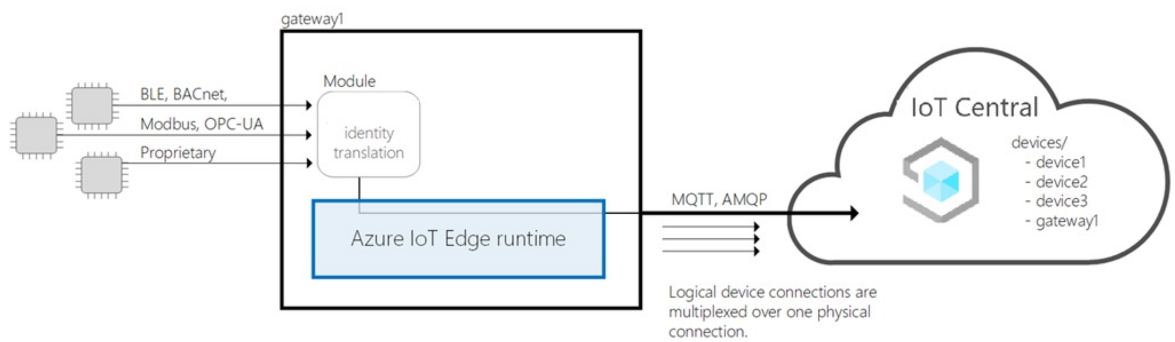
The IoT Edge device is provisioned in IoT Central and any downstream devices and their telemetry is represented as coming from the IoT Edge device. Downstream devices connected to the IoT Edge device aren't provisioned in IoT Central.

IoT Edge gateway device connected to downstream devices with identity



The IoT Edge device is provisioned in IoT Central along with the downstream devices connected to the IoT Edge device. Runtime support for provisioning downstream devices through the gateway isn't currently supported.

IoT Edge gateway device connected to downstream devices with identity provided by the IoT Edge gateway



The IoT Edge device is provisioned in IoT Central along with the downstream devices connected to the IoT Edge device. Runtime support of gateway providing identity to downstream devices and provisioning of downstream devices isn't currently supported. If you bring your own identity translation module, IoT Central can support this pattern.

Cloud gateway

Azure IoT Central uses Azure IoT Hub as a cloud gateway that enables device connectivity. IoT Hub enables:

- Data ingestion at scale in the cloud.
- Device management.
- Secure device connectivity.

To learn more about IoT Hub, see Azure IoT Hub.

To learn more about device connectivity in Azure IoT Central, see Device connectivity.

Data stores

Azure IoT Central stores application data in the cloud. Application data stored includes:

- Device templates.
- Device identities.
- Device metadata.
- User and role data.

Azure IoT Central uses a time series store for the measurement data sent from your devices. Time series data from devices used by the analytics service.

Analytics

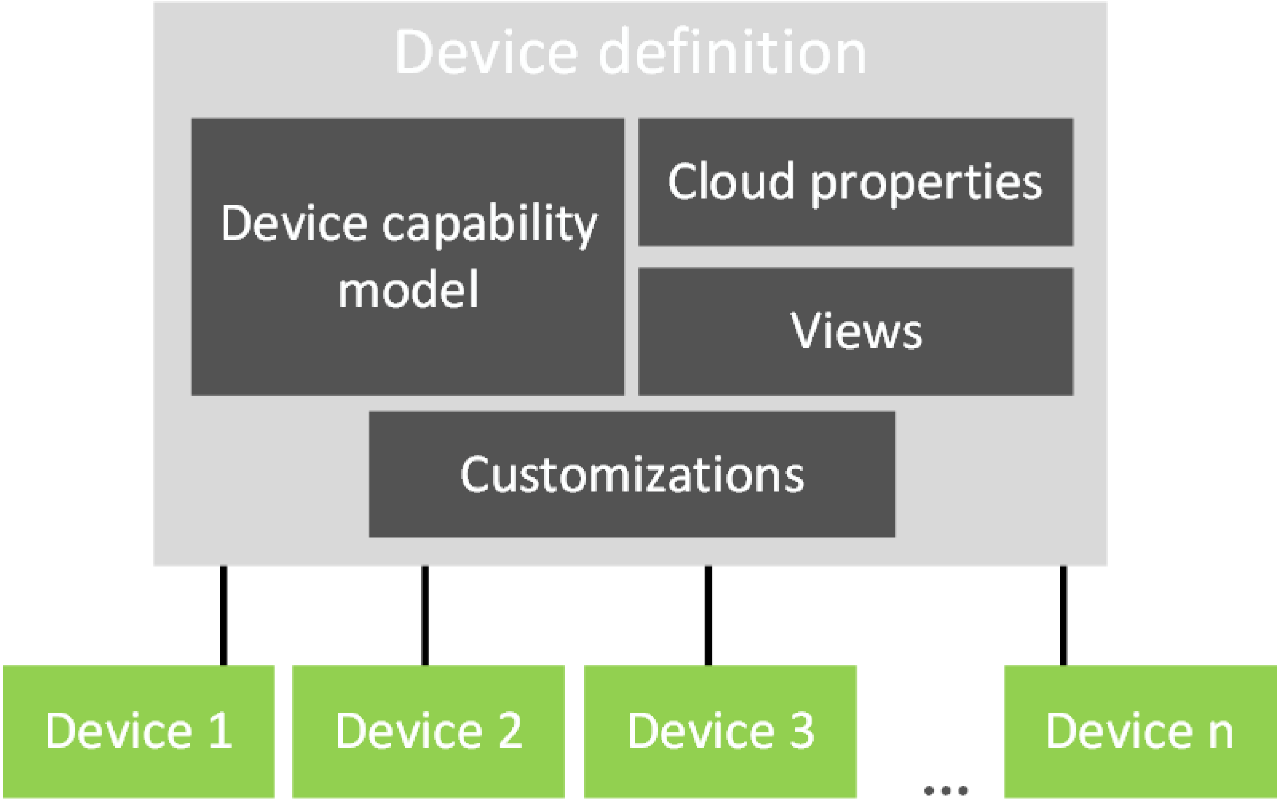
The analytics service is responsible for generating the custom reporting data that the application displays. An operator can customize the analytics displayed in the application. The analytics service is built on top of Azure Time Series Insights and processes the measurement data sent from your devices.

Rules and actions

Rules and actions work closely together to automate tasks within the application. A builder can define rules based on device telemetry such as the temperature exceeding a defined threshold. Azure IoT Central uses a stream processor to determine when the rule conditions are met. When a rule condition is met, it triggers an action defined by the builder. For example, an action can send an email to notify an engineer that the temperature in a device is too high.

Metadata management

In an Azure IoT Central application, device templates define the behavior and capability of types of device. For example, a refrigerator device template specifies the telemetry a refrigerator sends to your application.



In an IoT Central application device template contains:

- **Device capability models** specify the capabilities of a device such as the telemetry it sends, the properties that define the device state, and the commands the device responds to. Device capabilities are organized into one or more interfaces. For more information about device capability models, see the IoT Plug and Play (preview) documentation.
- **Cloud properties** specify the properties IoT Central stores for a device. These properties are only stored in IoT Central and are never sent to a device.
- **Views** specify the dashboards and forms the builder creates to let the operator monitor and manage the devices.
- **Customizations** let the builder override some of the definitions in the device capability model to make them more relevant to the IoT Central application.

An application can have one or more simulated and real devices based on each device template.

Data export

In an Azure IoT Central application, you can continuously export your data to your own Azure Event Hubs and Azure Service Bus instances. You can also periodically export your data to your Azure Blob storage account. IoT Central can export measurements, devices, and device templates.

Batch device updates

In an Azure IoT Central application, you can create and run jobs to manage connected devices. These jobs let you do bulk updates to device properties or settings, or run commands. For example, you can create a job to increase the fan speed for multiple refrigerated vending machines.

Role-based access control (RBAC)

An administrator can define access rules for an Azure IoT Central application using one of the predefined roles or by creating a custom role. Roles determine what areas of the application a user has access to and what actions they can perform.

Security

Security features within Azure IoT Central include:

- Data is encrypted in transit and at rest.
- Authentication is provided either by Azure Active Directory or Microsoft Account. Two-factor authentication is supported.
- Full tenant isolation.
- Device level security.

UI shell

The UI shell is a modern, responsive, HTML5 browser-based application. An administrator can customize the UI of the application by applying custom themes and modifying the help links to point to your own custom help resources. To learn more about UI customization, see Customize the Azure IoT Central UI article.

An operator can create personalized application dashboards. You can have several dashboards that display different data and switch between them.

Next steps

Now that you've learned about the architecture of Azure IoT Central, the suggested next step is to learn about device connectivity in Azure IoT Central.

What are application templates?

7/21/2020 • 2 minutes to read • Edit Online

Application templates in Azure IoT Central are a tool to help solution builders kickstart their IoT solution development. You can use app templates for everything from getting a feel for what is possible, to fully customizing and your application for resale to your customers.

Application templates consist of:

- Sample operator dashboards
- Sample device templates
- Simulated devices producing real-time data
- Pre-configured rules and jobs
- Rich documentation including tutorials and how-tos

You choose the application template when you create your application. You can't change the template after the application is created.

Custom templates

If you want to create your application from scratch, choose one of the two custom application templates:

- Custom application
- Custom application (legacy)

Choose the **Custom application** template unless you have a specific reason to use the legacy template.

Industry focused templates

Azure IoT Central is an industry agnostic application platform. Application templates are industry focused examples available for these industries today, with more to come in the future:

- Retail
  - Connected logistics
  - Digital distribution center
  - In-store analytics - condition monitoring
  - In-store analytics - checkout
  - Smart Inventory Management
- Energy
  - Smart meter monitoring
  - Solar panel monitoring
- Government
  - Connected waste management
  - Water consumption monitoring
  - Water quality monitoring
- Healthcare.
  - Continuous patient monitoring

Application versions

Templates are associated with specific IoT Central application versions. You can find the version of an application on the About your app page from the **Help** link.

Next steps

Now that you know what IoT Central application templates are, get started by creating an IoT Central Application.

What are device templates?

7/21/2020 • 9 minutes to read • Edit Online

This article applies to device developers and solution builders.

A device template in Azure IoT Central is a blueprint that defines the characteristics and behaviors of a type of device that connects to your application. For example, the device template defines the telemetry that a device sends so that IoT Central can create visualizations that use the correct units and data types.

A solution builder adds device templates to an IoT Central application. A device developer writes the device code that implements the behaviors defined in the device template.

A device template includes the following sections:

- *A device capability model (DCM)*. This part of the device template defines how the device interacts with your application. A device developer implements the behaviors defined in the DCM.
- *Cloud properties*. This part of the device template lets the solution developer specify any device metadata to store. Cloud properties are never synchronized with devices and only exist in the application. Cloud properties don't affect the code that a device developer writes to implement the DCM.
- *Customizations*. This part of the device template lets the solution developer override some of the definitions in the DCM. Customizations are useful if the solution developer wants to refine how the application handles a value, such as changing the display name for a property or the color used to display a telemetry value. Customizations don't affect the code that a device developer writes to implement the DCM.
- *Views*. This part of the device template lets the solution developer define visualizations to view data from the device, and forms to manage and control a device. The views use the DCM, cloud properties, and customizations. Views don't affect the code that a device developer writes to implement the DCM.

Device capability models

A DCM defines how a device interacts with your IoT Central application. The device developer must make sure that the device implements the behaviors defined in the DCM so that IoT Central can monitor and manage the device. A DCM is made up of one or more *interfaces*, and each interface can define a collection of *telemetry* types, *device properties*, and *commands*. A solution developer can import a JSON file that defines the DCM into a device template, or use the web UI in IoT Central to create or edit a DCM. Changes to a DCM made using the Web UI require the device template to be versioned.

A solution developer can also export a JSON file that contains the DCM. A device developer can use this JSON document to understand how the device should communicate with the IoT Central application.

The JSON file that defines the DCM uses the Digital Twin Definition Language (DTDL) V1. IoT Central expects the JSON file to contain the DCM with the interfaces defined inline, rather than in separate files.

A typical IoT device is made up of:

- Custom parts, which are the things that make your device unique.
- Standard parts, which are things that are common to all devices.

These parts are called *interfaces* in a DCM. Interfaces define the details of each part your device implements. Interfaces are reusable across DCMs.

The following example shows the outline of device capability model for an environmental sensor device with two interfaces:

```json
{
  "@id": "urn:contoso:sensor_device:1",
  "@type": "CapabilityModel",
  "displayName": "Environment Sensor Capability Model",
  "implements": [
    {
      "@type": "InterfaceInstance",
      "name": "deviceinfo",
      "schema": {
        "@id": "urn:azureiot:DeviceManagement:DeviceInformation:1",
        "@type": "Interface",
        "displayName": "Device Information",
        "@context": "http://azureiot.com/v1/contexts/IoTModel.json",
        "contents": [
          ...
        ]
      }
    },
    {
      "@type": "InterfaceInstance",
      "name": "sensor",
      "schema": {
        "@id": "urn:contoso:EnvironmentalSensor:1",
        "@type": "Interface",
        "displayName": "Environmental Sensor",
        "@context": "http://azureiot.com/v1/contexts/IoTModel.json",
        "contents": [
          ...
        ]
      }
    }
  ],
  "@context": "http://azureiot.com/v1/contexts/IoTModel.json"
}
```

A capability model has some required fields:

- `@id`: a unique ID in the form of a simple Uniform Resource Name.
- `@type`: declares that this object is a capability model.
- `@context`: specifies the DTDL version used for the capability model.
- `implements`: lists the interfaces that your device implements.

Each entry in the list of interfaces in the implements section has a:

- `name`: the programming name of the interface.
- `schema`: the interface the capability model implements.

An interface has some required fields:

- `@id`: a unique ID in the form of a simple Uniform Resource Name.
- `@type`: declares that this object is an interface.
- `@context`: specifies the DTDL version used for the interface.
- `contents`: lists the properties, telemetry, and commands that make up your device.

There are some optional fields you can use to add more details to the capability model, such as display name and description.

Interface

The DTDL lets you describe the capabilities of your device. Related capabilities are grouped into interfaces. Interfaces describe the properties, telemetry, and commands a part of your device implements:

- `Properties`. Properties are data fields that represent the state of your device. Use properties to represent the durable state of the device, such as the on-off state of a coolant pump. Properties can also represent basic device properties, such as the firmware version of the device. You can declare properties as read-only or writable.
- `Telemetry`. Telemetry fields represent measurements from sensors. Whenever your device takes a sensor measurement, it should send a telemetry event containing the sensor data.
- `Commands`. Commands represent methods that users of your device can execute on the device. For example, a reset command or a command to switch a fan on or off.

The following example shows the environmental sensor interface definition:

```
{
  "@type": "Property",
  "displayName": "Device State",
  "description": "The state of the device. Two states online/offline are available.",
  "name": "state",
  "schema": "boolean"
},
{
  "@type": "Property",
  "displayName": "Customer Name",
  "description": "The name of the customer currently operating the device.",
  "name": "name",
  "schema": "string",
  "writable": true
},
{
  "@type": [
    "Telemetry",
    "SemanticType/Temperature"
  ],
  "description": "Current temperature on the device",
  "displayName": "Temperature",
  "name": "temp",
  "schema": "double",
  "unit": "Units/Temperature/fahrenheit"
},
{
  "@type": "Command",
  "name": "turnon",
  "comment": "This Commands will turn-on the LED light on the device.",
  "commandType": "synchronous"
},
{
  "@type": "Command",
  "name": "turnoff",
  "comment": "This Commands will turn-off the LED light on the device.",
  "commandType": "synchronous"
}
```

This example shows two properties, a telemetry type, and two commands. A minimal field description has a:

- `@type` to specify the type of capability: `Telemetry`, `Property`, or `Command`. In some cases, the type includes a semantic type to enable IoT Central to make some assumptions about how to handle the value.
- `name` for the telemetry value.
- `schema` to specify the data type for the telemetry or the property. This value can be a primitive type, such as double, integer, boolean, or string. Complex object types, arrays, and maps are also supported.
- `commandType` to specify how the command should be handled.

Optional fields, such as display name and description, let you add more details to the interface and capabilities.

Properties

By default, properties are read-only. Read-only properties mean that the device reports property value updates to your IoT Central application. Your IoT Central application can't set the value of a read-only property.

You can also mark a property as writeable on an interface. A device can receive an update to a writeable property from your IoT Central application as well as reporting property value updates to your application.

Devices don't need to be connected to set property values. The updated values are transferred when the device next connects to the application. This behavior applies to both read-only and writeable properties.

Don't use properties to send telemetry from your device. For example, a readonly property such as `temperatureSetting=80` should mean that the device temperature has been set to 80, and the device is trying to get to, or stay at, this temperature.

For writable properties, the device application returns a desired state status code, version, and description to indicate whether it received and applied the property value.

Telemetry

IoT Central lets you view telemetry on dashboards and charts, and use rules to trigger actions when thresholds are reached. IoT Central uses the information in the DCM, such as data types, units and display names, to determine how to display telemetry values.

You can use the IoT Central data export feature to stream telemetry to other destinations such as storage or Event Hubs.

Commands

Commands are either synchronous or asynchronous. A synchronous command must execute within 30 seconds by default, and the device must be connected when the command arrives. If the device does respond in time, or the device isn't connected, then the command fails.

Use asynchronous commands for long-running operations. The device sends progress information using telemetry messages. These progress messages have the following header properties:

- `iothub-command-name`: the command name, for example `UpdateFirmware`.
- `iothub-command-request-id`: the request ID generated on the server side and sent to the device in the initial call.
- `iothub-interface-id`: The ID of the interface this command is defined on, for example `urn:example:AssetTracker:1`. `iothub-interface-name`: the instance name of this interface, for example `myAssetTracker`.
- `iothub-command-statuscode`: the status code returned from the device, for example `202`.

Cloud properties

Cloud properties are part of the device template, but aren't part of the DCM. Cloud properties let the solution developer specify any device metadata to store in the IoT Central application. Cloud properties don't affect the code that a device developer writes to implement the DCM.

A solution developer can add cloud properties to dashboards and views alongside device properties to enable an operator to manage the devices connected to the application. A solution developer can also use cloud properties as part of a rule definition to make a threshold value editable by an operator.

Customizations

Customizations are part of the device template, but aren't part of the DCM. Customizations let the solution developer enhance or override some of the definitions in the DCM. For example, a solution developer can change the display name for a telemetry type or property. A solution developer can also use customizations to add validation such as a minimum or maximum length for a string device property.

Customizations may affect the code that a device developer writes to implement the DCM. For example, a customization could set minimum and maximum string lengths or minimum and maximum numeric values for telemetry.

Views

A solution developer creates views that let operators monitor and manage connected devices. Views are part of the device template, so a view is associated with a specific device type. A view can include:

- Charts to plot telemetry.
- Tiles to display read-only device properties.
- Tiles to let the operator edit writable device properties.
- Tiles to let the operator edit cloud properties.
- Tiles to let the operator call commands, including commands that expect a payload.
- Tiles to display labels, images, or markdown text.

The telemetry, properties, and commands that you can add to a view are determined by the DCM, cloud properties, and customizations in the device template.

Next steps

As a device developer, now that you've learned about device templates, a suggested next steps is to read Get connected to Azure IoT Central to learn more about how to register devices with IoT Central and how IoT Central secures device connections.

As a solution developer, a suggested next step is to read Define a new IoT device type in your Azure IoT Central application to learn more about how to create a device template.

Get connected to Azure IoT Central

7/21/2020 • 15 minutes to read • Edit Online

This article applies to operators and device developers.

This article describes the options for connecting your devices to an Azure IoT Central application.

Typically, you must register a device in your application before it can connect. However, IoT Central does support scenarios where devices can connect without first being registered.

IoT Central uses the Azure IoT Hub Device Provisioning service (DPS) to manage the connection process. A device first connects to a DPS endpoint to retrieve the information it needs to connect to your application. Internally, your IoT Central application uses an IoT hub to handle device connectivity. Using DPS enables:

- IoT Central to support onboarding and connecting devices at scale.
- You to generate device credentials and configure the devices offline without registering the devices through IoT Central UI.
- You to use your own device IDs to register devices in IoT Central. Using your own device IDs simplifies integration with existing back-office systems.
- A single, consistent way to connect devices to IoT Central.

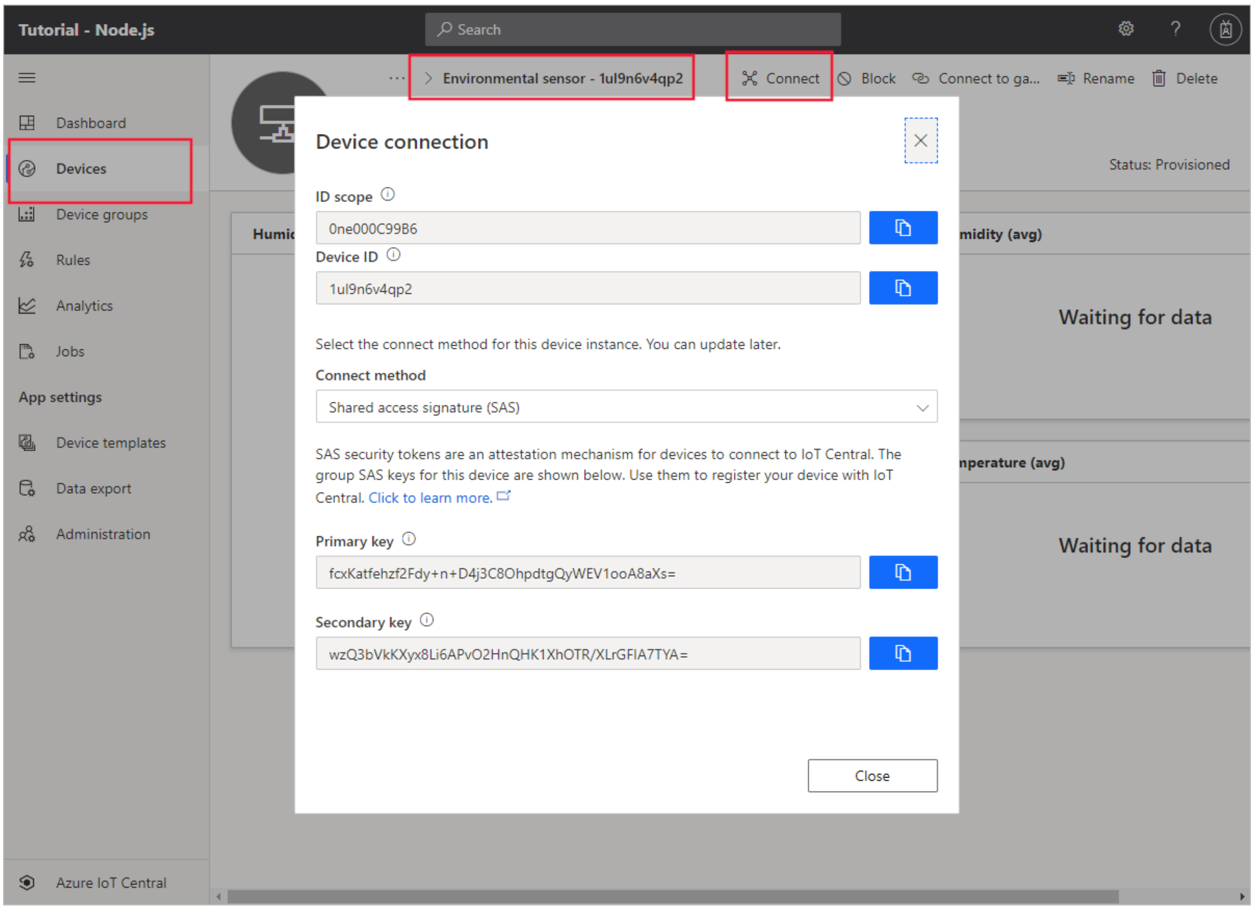
To secure the communication between a device and your application, IoT Central supports both shared access signatures (SAS) and X.509 certificates. X.509 certificates are recommended in production environments.

This article describes the following use cases:

- Connect a single device using SAS
- Connect devices at scale using SAS
- Connect devices at scale using X.509 certificates - the recommended approach for production environments.
- Connect devices without first registering them
- Connect devices that use DPS individual enrollments
- Automatically associate a device with a device template

Connect a single device

This approach is useful when you're experimenting with IoT Central or testing devices. You can use the device connection SAS keys from your IoT Central application to connect a device to your IoT Central application. Copy the *device SAS key* from the connection information for a registered device:



To learn more, see the Create and connect a Node.js client application to your Azure IoT Central application tutorial.

Connect devices at scale using SAS

To connect devices to IoT Central at scale using SAS keys, you need to register and then set up the devices:

Register devices in bulk

To register a large number of devices with your IoT Central application, use a CSV file to import device IDs and device names.

To retrieve the connection information for the imported devices, export a CSV file from your IoT Central application. The exported CSV file includes the device IDs and the SAS keys.

Set up your devices

Use the connection information from the export file in your device code to enable your devices to connect and send data to IoT to your IoT Central application. You also need the DPS **ID scope** for your application. You can find this value in **Administration > Device connection**.

> **NOTE**
> To learn how you can connect devices without first registering them in IoT Central, see Connect without first registering devices.

Connect devices using X.509 certificates

In a production environment, using X.509 certificates is the recommended device authentication mechanism for IoT Central. To learn more, see Device Authentication using X.509 CA Certificates.

To connect a device with an X.509 certificate to your application:

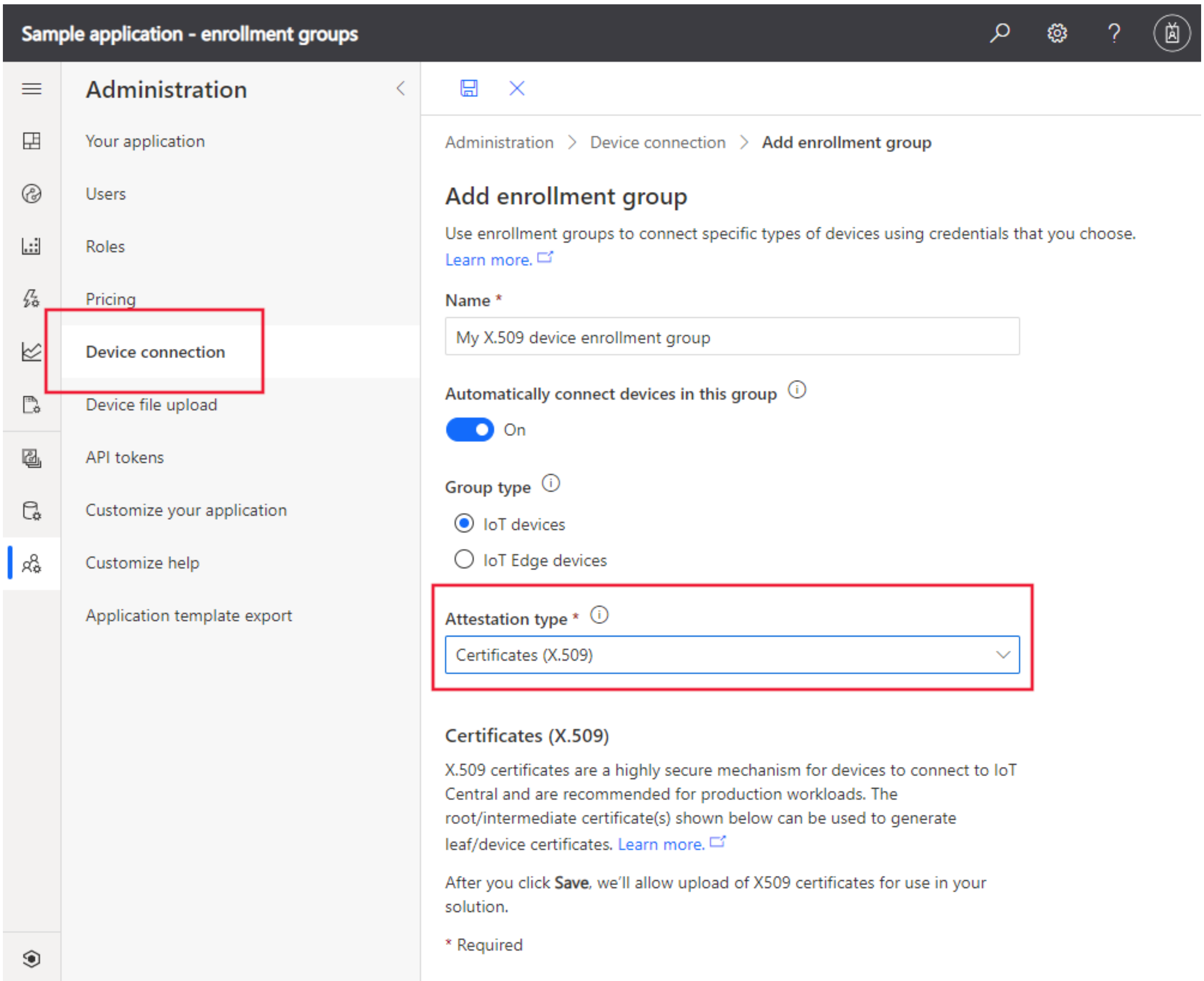
1. Create an *enrollment group* that uses the **Certificates (X.509)** attestation type.
2. Add and verify an intermediate or root X.509 certificate in the enrollment group.
3. Register and connect devices that use leaf X.509 certificates generated from the root or intermediate certificate in the enrollment group.

Create an enrollment group

An enrollment group is a group of devices that share the same attestation type. The two supported attestation types are X.509 certificates and SAS:

- In an X.509 enrollment group, all the devices that connect to IoT Central use leaf X.509 certificates generated from the root or intermediate certificate in the enrollment group.
- In a SAS enrollment group, all the devices that connect to IoT Central use a SAS token generated from the SAS token in the enrollment group.

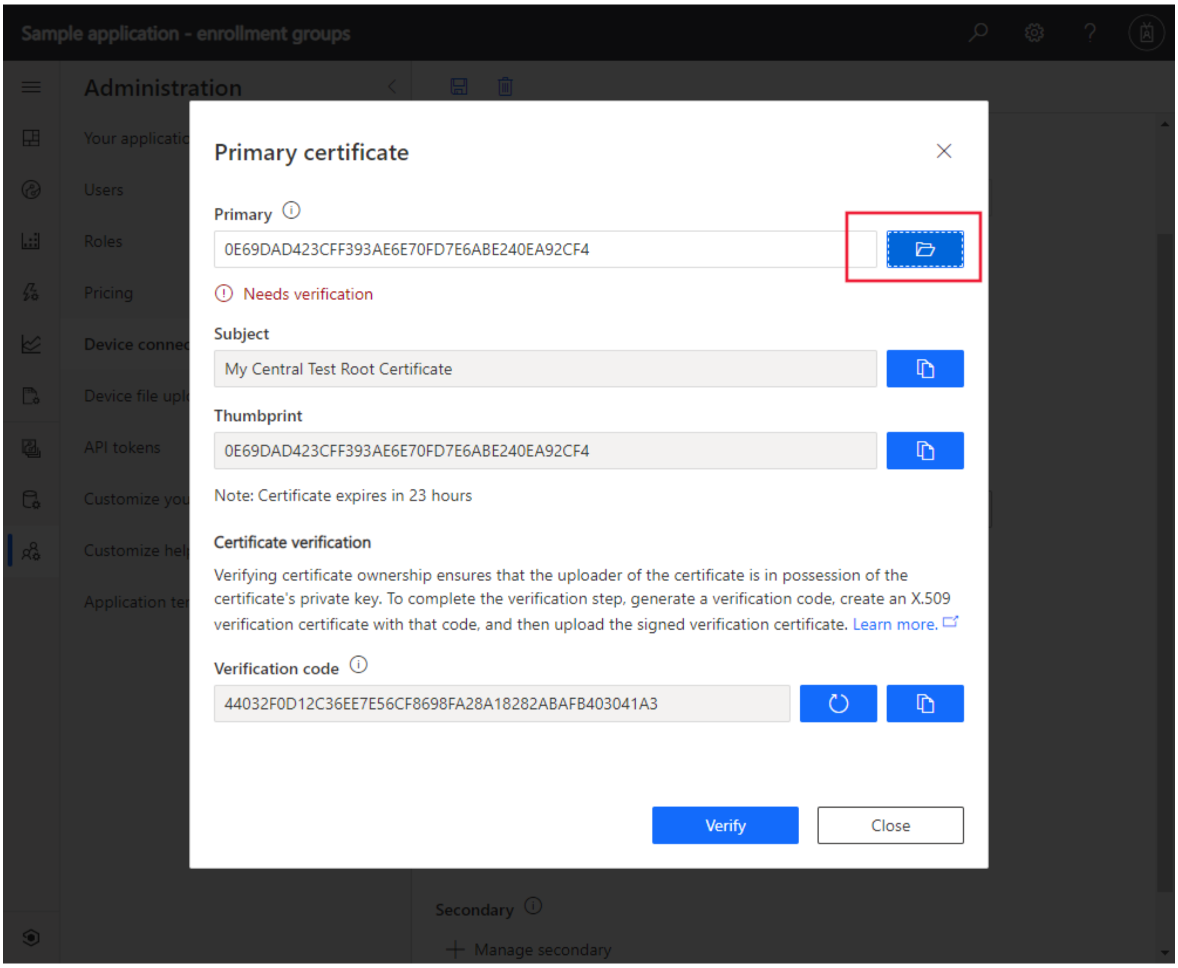
The two default enrollment groups in every IoT Central application are SAS enrollment groups - one for IoT devices, and one for Azure IoT Edge devices. To create an X.509 enrollment group, navigate to the **Device connection** page and select **+ Add enrollment group**:



Add and verify a root or intermediate X.509 certificate

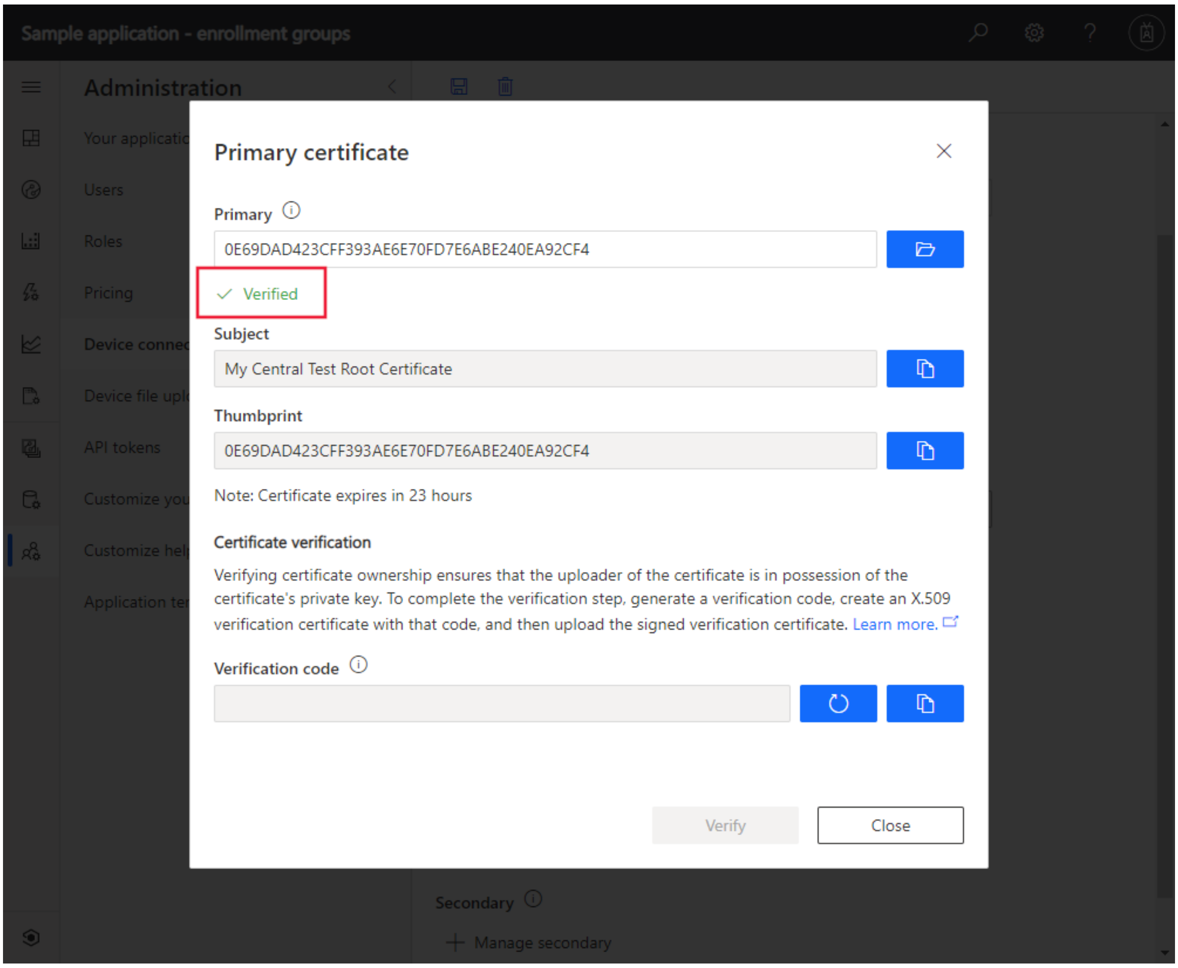
To add and verify a root or intermediate certificate to your enrollment group:

1. Navigate to the X.509 enrollment group you just created. You have the option to add both primary and secondary X.509 certificates. Select **+ Manage primary**.
2. On the **Primary certificate page**, upload your primary X.509 certificate. This is your root or intermediate certificate:



3. Use the **Verification code** to generate a verification code in the tool you're using. Then select **Verify** to upload the verification certificate.

4. When the verification is successful, you see the following confirmation:



Verifying certificate ownership ensures that the person uploading the certificate has the certificate's private key.

If you have a security breach or your primary certificate is set to expire, use the secondary certificate to reduce downtime. You can continue to provision devices using the secondary certificate while you update the primary certificate.

Register and connect devices

To bulk connect devices using X.509 certificates, first register the devices in your application by using a CSV file to import the device IDs and device names. The device IDs should all be in lower case.

Generate X.509 leaf certificates for your devices using the root or intermediate certificate you uploaded to your X.509 enrollment group. Use the **Device ID** as the `CNAME` value in the leaf certificates. Your device code needs the **ID scope** value for your application, the **device ID**, and the corresponding device certificate.

Sample device code

The following sample from the Azure IoT NodeJS SDK shows how a Node.js device client uses an X.509 leaf certificate and DPS to register with an IoT Central application:

```
// Copyright (c) Microsoft. All rights reserved.
// Licensed under the MIT license. See LICENSE file in the project root for full license information.

'use strict';

var iotHubTransport = require('azure-iot-device-mqtt').Mqtt;
var Client = require('azure-iot-device').Client;
var Message = require('azure-iot-device').Message;

var fs = require('fs');
// You can change the following using statement if you would like to try another protocol.
var Transport = require('azure-iot-provisioning-device-mqtt').Mqtt;
// var Transport = require('azure-iot-provisioning-device-amqp').Amqp;
// var Transport = require('azure-iot-provisioning-device-amqp').AmqpWs;
// var Transport = require('azure-iot-provisioning-device-http').Http;
// var Transport = require('azure-iot-provisioning-device-mqtt').MqttWs;

var X509Security = require('azure-iot-security-x509').X509Security;
var ProvisioningDeviceClient = require('azure-iot-provisioning-device').ProvisioningDeviceClient;

var provisioningHost = process.env.PROVISIONING_HOST;
var idScope = process.env.PROVISIONING_IDSCOPE;
var registrationId = process.env.PROVISIONING_REGISTRATION_ID;
var deviceCert = {
  cert: fs.readFileSync(process.env.CERTIFICATE_FILE).toString(),
  key: fs.readFileSync(process.env.KEY_FILE).toString()
};

var transport = new Transport();
var securityClient = new X509Security(registrationId, deviceCert);
var deviceClient = ProvisioningDeviceClient.create(provisioningHost, idScope, transport, securityClient);

// Register the device.  Do not force a re-registration.
deviceClient.register(function(err, result) {
  if (err) {
    console.log("error registering device: " + err);
  } else {
    console.log('registration succeeded');
    console.log('assigned hub=' + result.assignedHub);
    console.log('deviceId=' + result.deviceId);
    var connectionString = 'HostName=' + result.assignedHub + ';DeviceId=' + result.deviceId + ';x509=true';
    var hubClient = Client.fromConnectionString(connectionString, iotHubTransport);
    hubClient.setOptions(deviceCert);
    hubClient.open(function(err) {
      if (err) {
        console.error('Failure opening iothub connection: ' + err.message);
      } else {
        console.log('Client connected');
        var message = new Message('Hello world');
        hubClient.sendEvent(message, function(err, res) {
          if (err) console.log('send error: ' + err.toString());
          if (res) console.log('send status: ' + res.constructor.name);
          process.exit(1);
        });
      }
    });
  }
});
```

For an equivalent C sample, see prov_dev_client_sample.c in the Azure IoT C Provisioning Device Client SDK.

For testing purposes only

For testing only, you can use the following utilities to generate root, intermediate, and device certificates:

- Tools for the Azure IoT Device Provisioning Device SDK: a collection of Node.js tools that you can use to generate and verify X.509 certificates and keys.

- If you're using a DevKit device, this command-line tool generates a CA certificate that you can add to your IoT Central application to verify the certificates.
- Manage test CA certificates for samples and tutorials: a collection of PowerShell and Bash scripts to:
  - Create a certificate chain.
  - Save the certificates as .cer files to upload to your IoT Central application.
  - Use the verification code from the IoT Central application to generate the verification certificate.
  - Create leaf certificates for your devices using your device IDs as a parameter to the tool.

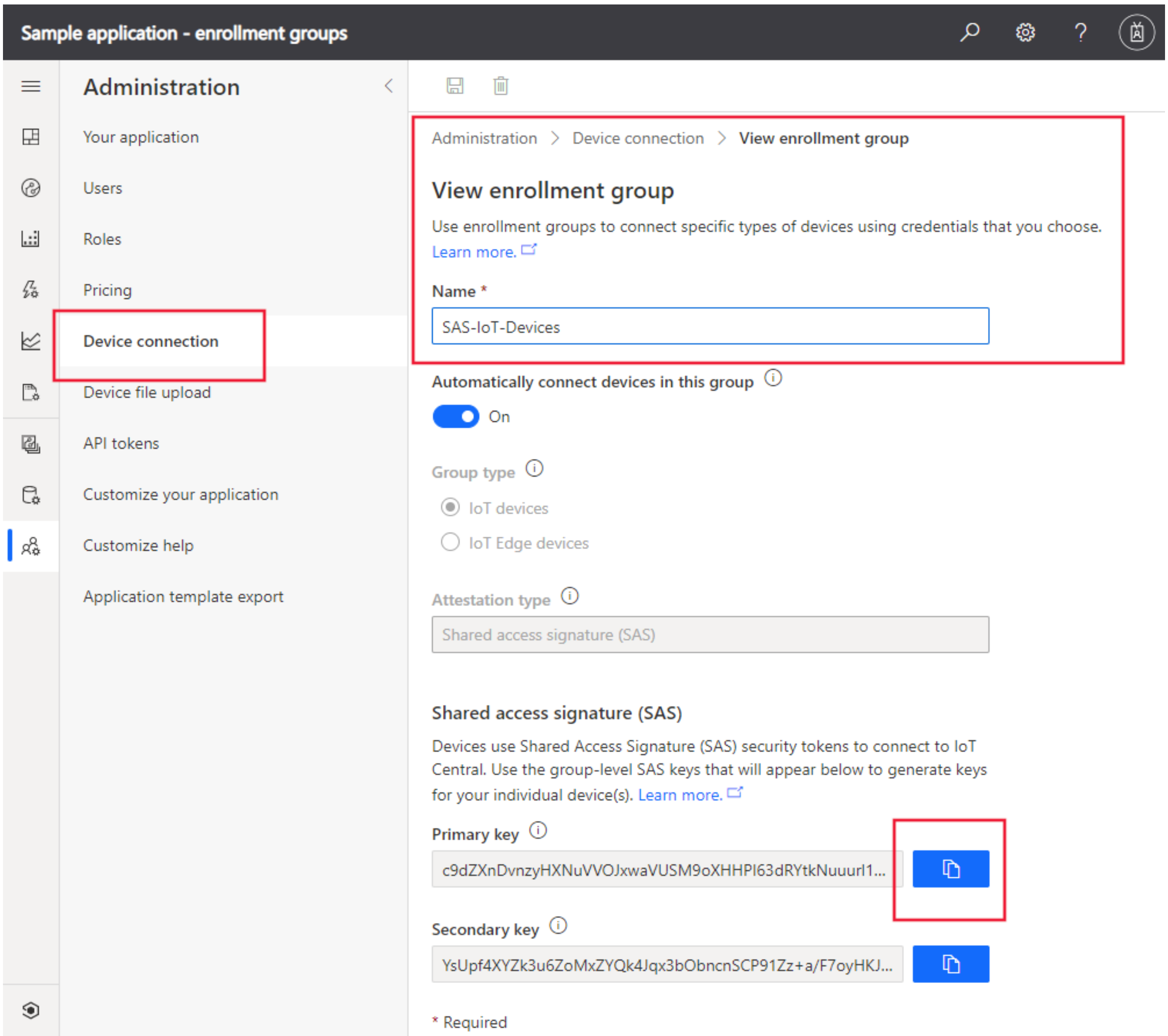
Connect without registering devices

The previously described scenarios all require you to register devices in your application before they connect. IoT Central also enables OEMs to mass manufacture devices that can connect without first being registered. An OEM generates suitable device credentials, and configures the devices in the factory. When a customer turns on a device for the first time, it connects to DPS, which then automatically connects the device to the correct IoT Central application. An IoT Central operator must approve the device before it starts sending data to the application.

The flow is slightly different depending on whether the devices use SAS tokens or X.509 certificates:

Connect devices that use SAS tokens without registering

1. Copy the group primary key from the **SAS-IoT-Devices** enrollment group:



2. Use the dps-keygen tool to generate the device SAS keys. Use the group primary key from the previous step. The device IDs must be lower-case:

```
dps-keygen -mk:<group primary key> -di:<device ID>
```

3. The OEM flashes each device with a device ID, a generated device SAS key, and the application **ID scope** value.

4. When you switch on a device, it first connects to DPS to retrieve its IoT Central registration information.

   The device initially has a device status **Unassociated** on the **Devices** page and isn't assigned to a device template. On the **Devices** page, **Migrate** the device to the appropriate device template. Device provisioning is now complete, the device status is now **Provisioned**, and the device can start sending data.

   On the **Administration** > **Device connection** page, the **Auto approve** option controls whether you need to manually approve the device before it can start sending data.

   > **NOTE**
   >
   > To learn how automatically associate a device with a device template, see Automatically associate a device with a device template.

Connect devices that use X.509 certificates without registering

1. Create an enrollment group and then Add and verify a root or intermediate X.509 certificate to your IoT Central application.

2. Generate the leaf-certificates for your devices using the root or intermediate certificate you added to your IoT Central application. Use lower-case device IDs as the `CNAME` in the leaf certificates.

3. The OEM flashes each device with a device ID, a generated leaf X.509 certificate, and the application **ID scope** value.

4. When you switch on a device, it first connects to DPS to retrieve its IoT Central registration information.

   The device initially has a device status **Unassociated** on the **Devices** page and isn't assigned to a device template. On the **Devices** page, **Migrate** the device to the appropriate device template. Device provisioning is now complete, the device status is now **Provisioned**, and the device can start sending data.

   On the **Administration** > **Device connection** page, the **Auto approve** option controls whether you need to manually approve the device before it can start sending data.

   > **NOTE**
   >
   > To learn how automatically associate a device with a device template, see Automatically associate a device with a device template.

Individual enrollment-based device connectivity

For customers connecting devices that each have their own authentication credentials, use individual enrollments. An individual enrollment is an entry for a single device that is allowed to connect. Individual enrollments can use either X.509 leaf certificates or SAS tokens (from a physical or virtual trusted platform module) as attestation mechanisms. The device ID (also known as registration ID) in an individual enrollment is alphanumeric, lowercase, and may contain hyphens. For more information, see DPS individual enrollment.

> **NOTE**
>
> When you create an individual enrollment for a device, it takes precedence over the default group enrollment options in your IoT Central application.

Create individual enrollments

IoT Central supports the following attestation mechanisms for individual enrollments:

- **Symmetric key attestation:** Symmetric key attestation is a simple approach to authenticating a device with the DPS instance. To create an individual enrollment that uses symmetric keys, open the **Device Connection** page, select **Individual enrollment** as the connection method, and **Shared access signature (SAS)** as the mechanism. Enter base64 encoded primary and secondary keys, and save your changes. Use the **ID scope**, **Device ID**, and either the primary or secondary key to connect your device.

  > **TIP**
  >
  > For testing, you can use **OpenSSL** to generate base64 encoded keys: `openssl rand -base64 64`

- **X.509 certificates:** To create an individual enrollment with X.509 certificates, open the **Device Connection** page, select **Individual enrollment** as the connection method, and **Certificates (X.509)** as the mechanism. Device certificates used with an individual enrollment entry have a requirement that the issuer and subject CN are set to the device ID.

  > **TIP**
  >
  > For testing, you can use Tools for the Azure IoT Device Provisioning Device SDK for Node.js to generate a self-signed certificate: `node create_test_cert.js device "mytestdevice"`

- **Trusted Platform Module (TPM) attestation:** A TPM is a type of hardware security module. Using a TPM is one of the most secure ways to connect a device. This article assumes you're using a discrete, firmware, or integrated TPM. Software emulated TPMs are well suited for prototyping or testing, but they don't provide the same level of security as discrete, firmware, or integrated TPMs. Don't use software TPMs in production. To create an individual enrollment that uses a TPM, open the **Device Connection** page, select **Individual enrollment** as the connection method, and **TPM** as the mechanism. Enter the TPM endorsement key and save the device connection information.

Automatically associate with a device template

One of the key features of IoT Central is the ability to associate device templates automatically on device connection. Along with device credentials, devices can send a **CapabilityModelId** as part of the device registration call. The **CapabilityModelID** is a URN that identifies the capability model the device implements. The IoT Central application can use the **CapabilityModelID** to identify the device template to use and then automatically associate the device with the device template. The discovery process works as follows:

1. If the device template is already published in the IoT Central application, the device is associated with the device template.
2. For pre-certified IoT Plug and Play devices, if the device template is not already published in the IoT Central application, the device template is fetched from the public repository.

The following snippets show the format of the additional payload the device must send during the DPS registration call for automatic association to work.

This is the format for devices that use the generally available device SDK that doesn't support IoT Plug and Play:

```
    iotcModelId: '< this is the URN for the capability model>';
```

This is the format for devices using public preview device SDK that does support IoT Plug and Play:

```
'__iot:interfaces': {
    CapabilityModelId: <this is the URN for the capability model>
}
```

> **NOTE**
>
> The **Auto approve** option on **Administration** > **Device connection** must be enabled for devices to automatically connect, discover the device template, and start sending data.

Device status values

When a real device connects to your IoT Central application, its device status changes as follows:

1. The device status is first **Registered**. This status means the device is created in IoT Central, and has a device ID. A device is registered when:
   - A new real device is added on the **Devices** page.
   - A set of devices is added using **Import** on the **Devices** page.
2. The device status changes to **Provisioned** when the device that connected to your IoT Central application with valid credentials completes the provisioning step. In this step, the device uses DPS to automatically retrieve a connection string from the IoT Hub used by your IoT Central application. The device can now connect to IoT Central and start sending data.
3. An operator can block a device. When a device is blocked, it can't send data to your IoT Central application. Blocked devices have a status of **Blocked**. An operator must reset the device before it can resume sending data. When an operator unblocks a device the status returns to its previous value, **Registered** or **Provisioned**.
4. If the device status is **Waiting for Approval**, it means the **Auto approve** option is disabled. An operator must explicitly approve a device before it starts sending data. Devices not registered manually on the **Devices** page, but connected with valid credentials will have the device status **Waiting for Approval**. Operators can approve these devices from the **Devices** page using the **Approve** button.
5. If the device status is **Unassociated**, it means the device connecting to IoT Central doesn't have an associated device template. This situation typically happens in the following scenarios:
   - A set of devices is added using **Import** on the **Devices** page without specifying the device template.
   - A device was registered manually on the **Devices** page without specifying the device template. The device then connected with valid credentials.

   The Operator can associate a device to a device template from the **Devices** page using the **Migrate** button.

Best practices

Don't persist or cache the device connection string that DPS returns when you first connect the device. To reconnect a device, go through the standard device registration flow to get the correct device connection string. If the device caches the connection string, the device software runs into the risk of having a stale connection string if IoT Central updates the underlying Azure IoT hub it uses.

SDK support

The Azure Device SDKs offer the easiest way for you implement your device code. The following device SDKs are available:

- 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

SDK features and IoT Hub connectivity

All device communication with IoT Hub uses the following IoT Hub connectivity options:

- Device-to-cloud messaging
- Device twins

The following table summarizes how Azure IoT Central device features map on to IoT Hub features:

AZURE IOT CENTRAL	AZURE IOT HUB
Telemetry	Device-to-cloud messaging
Property	Device twin reported properties
Property (writeable)	Device twin desired and reported properties
Command	Direct methods

To learn more about using the Device SDKs, see Connect an MXChip IoT DevKit device to your Azure IoT Central application for example code.

Protocols

The Device SDKs support the following network protocols for connecting to an IoT hub:

- MQTT
- AMQP
- HTTPS

For information about these difference protocols and guidance on choosing one, see Choose a communication protocol.

If your device can't use any of the supported protocols, you can use Azure IoT Edge to do protocol conversion. IoT Edge supports other intelligence-on-the-edge scenarios to offload processing to the edge from the Azure IoT Central application.

Security

All data exchanged between devices and your Azure IoT Central is encrypted. IoT Hub authenticates every request from a device that connects to any of the device-facing IoT Hub endpoints. To avoid exchanging credentials over the wire, a device uses signed tokens to authenticate. For more information, see, Control access to IoT Hub.

Next steps

If you're a device developer, some suggested next steps are to:

- Learn how to Monitor device connectivity using Azure CLI
- Learn how to Define a new IoT device type in your Azure IoT Central application
- Read about Azure IoT Edge devices and Azure IoT Central

Connect Azure IoT Edge devices to an Azure IoT Central application

4/21/2020 • 2 minutes to read • Edit Online

This article applies to solution builders and device developers.

IoT Edge is made up of three components:

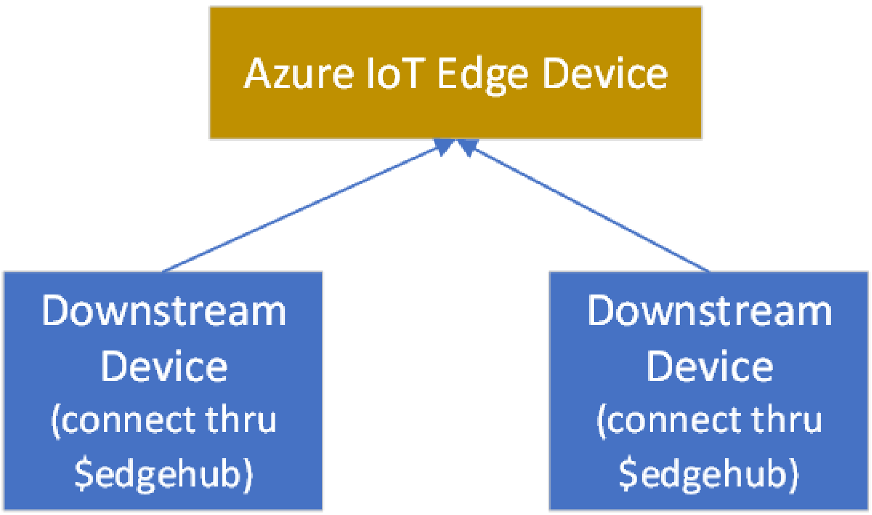
- **IoT Edge modules** are containers that run Azure services, partner services, or your own code. Modules are deployed to IoT Edge devices, and run locally on those devices.
- The **IoT Edge runtime** runs on each IoT Edge device, and manages the modules deployed to each device.
- A **cloud-based interface** enables you to remotely monitor and manage IoT Edge devices. IoT Central is the cloud interface.

An **Azure IoT Edge** device can be a gateway device, with downstream devices connecting into the IoT Edge device. This article shares more information about downstream device connectivity patterns.

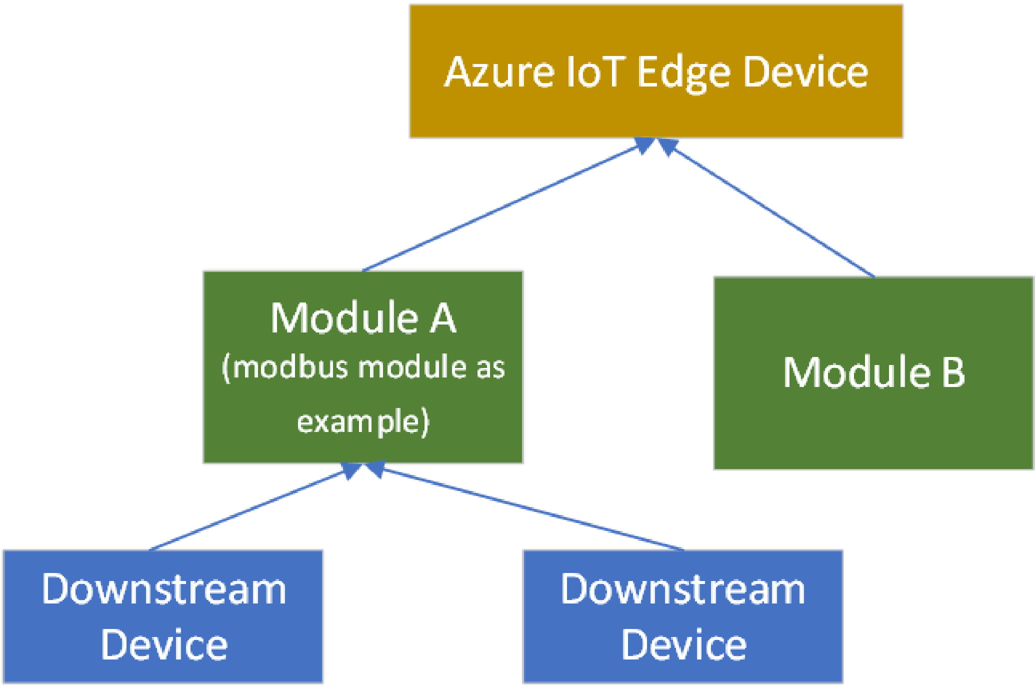
A **device template** defines the capabilities of your device and IoT Edge modules. Capabilities include telemetry the module sends, module properties, and the commands a module responds to.

Downstream device relationships with a gateway and modules

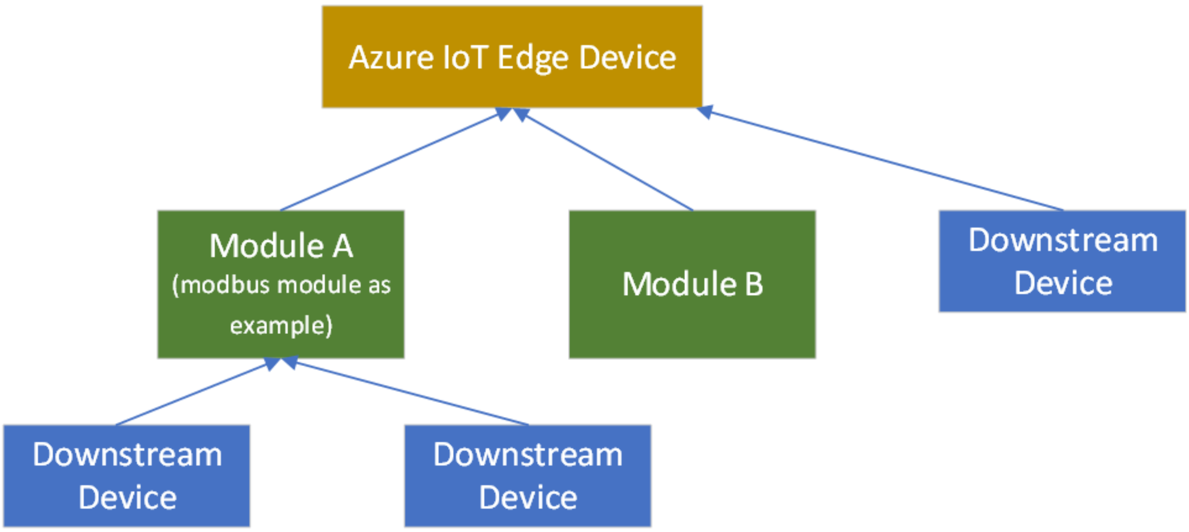
Downstream devices can connect to an IoT Edge gateway device through the `$edgeHub` module. This IoT Edge device becomes a transparent gateway in this scenario.



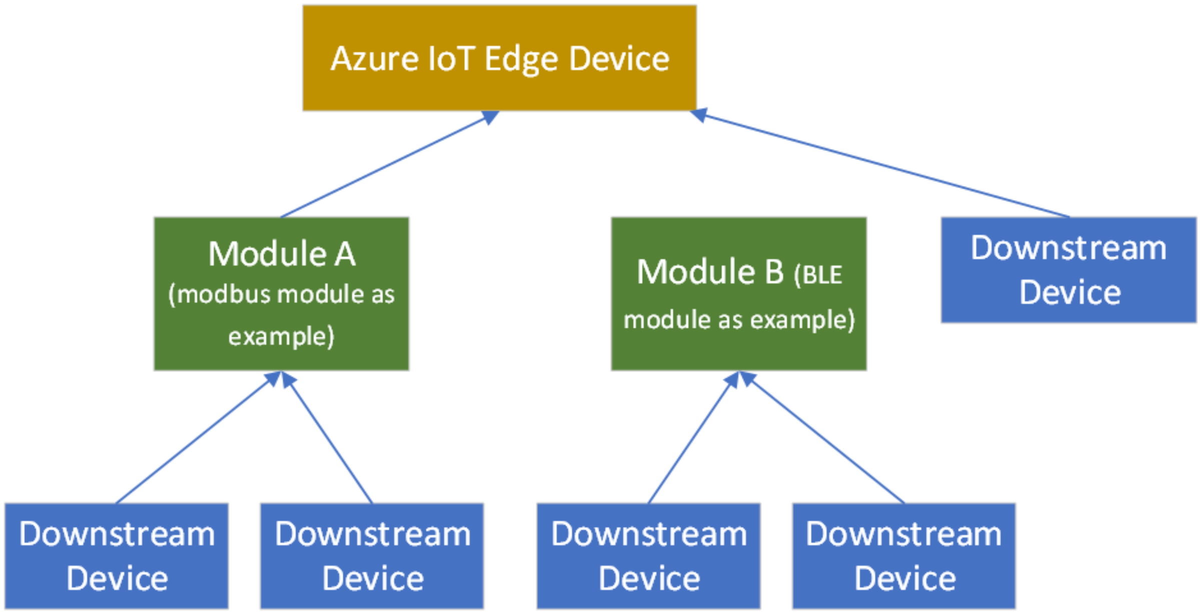
Downstream devices can also connect to an IoT Edge gateway device through a custom module. In the following scenario, downstream devices connect through a Modbus custom module.



The following diagram shows connection to an IoT Edge gateway device through both types of modules (custom and `$edgeHub`).



Finally, downstream devices can connect to an IoT Edge gateway device through multiple custom modules. The following diagram shows downstream devices connecting through a Modbus custom module, a BLE custom module, and the `$edgeHub` module.



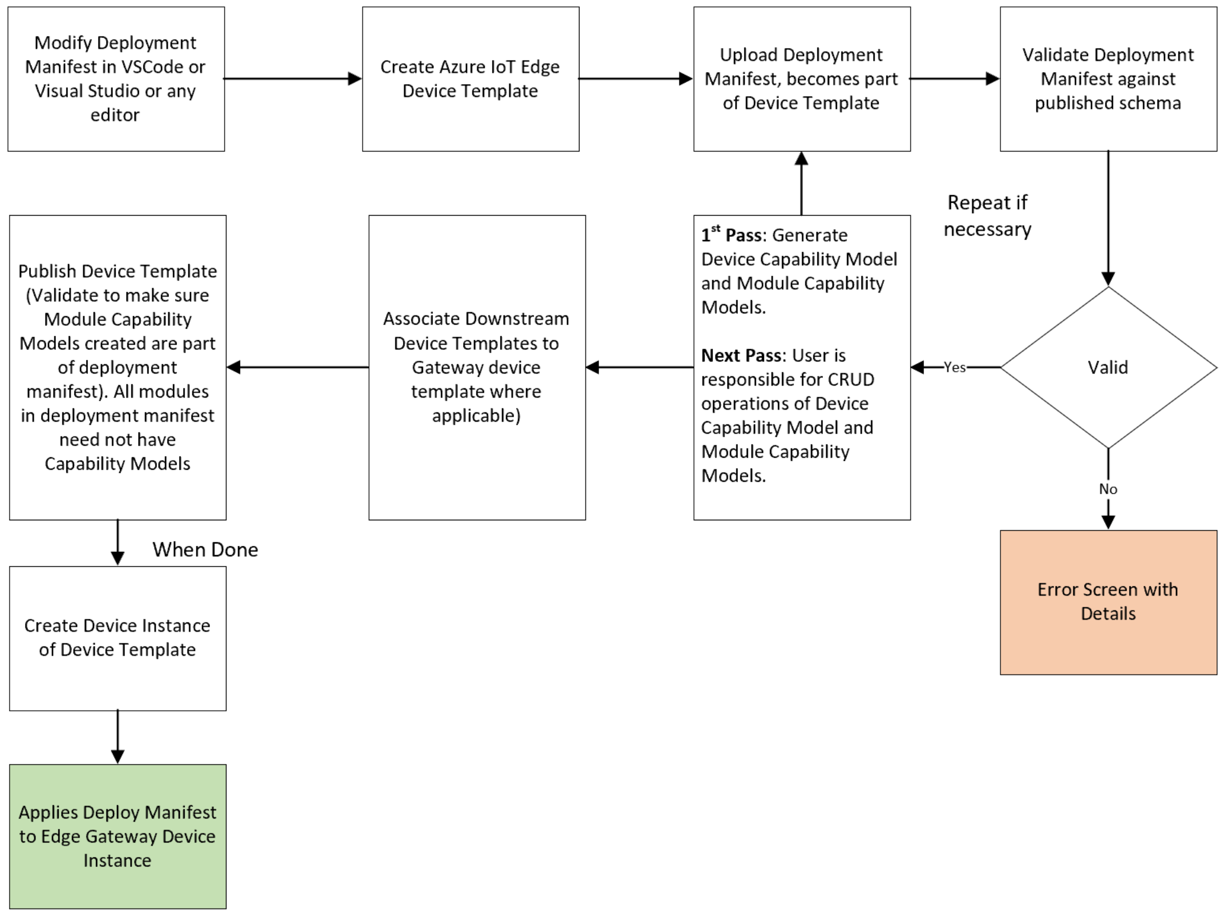
Deployment manifests and device templates

In IoT Edge, you can deploy and manage business logic in the form of modules. IoT Edge modules are the smallest unit of computation managed by IoT Edge, and can contain Azure services (such as Azure Stream Analytics), or your own solution-specific code. To understand how modules are developed, deployed, and maintained, see IoT Edge modules.

At a high level, a deployment manifest is a list of module twins that are configured with their desired properties. A deployment manifest tells an IoT Edge device (or a group of devices) which modules to install, and how to configure them. Deployment manifests include the desired properties for each module twin. IoT Edge devices report back the reported properties for each module.

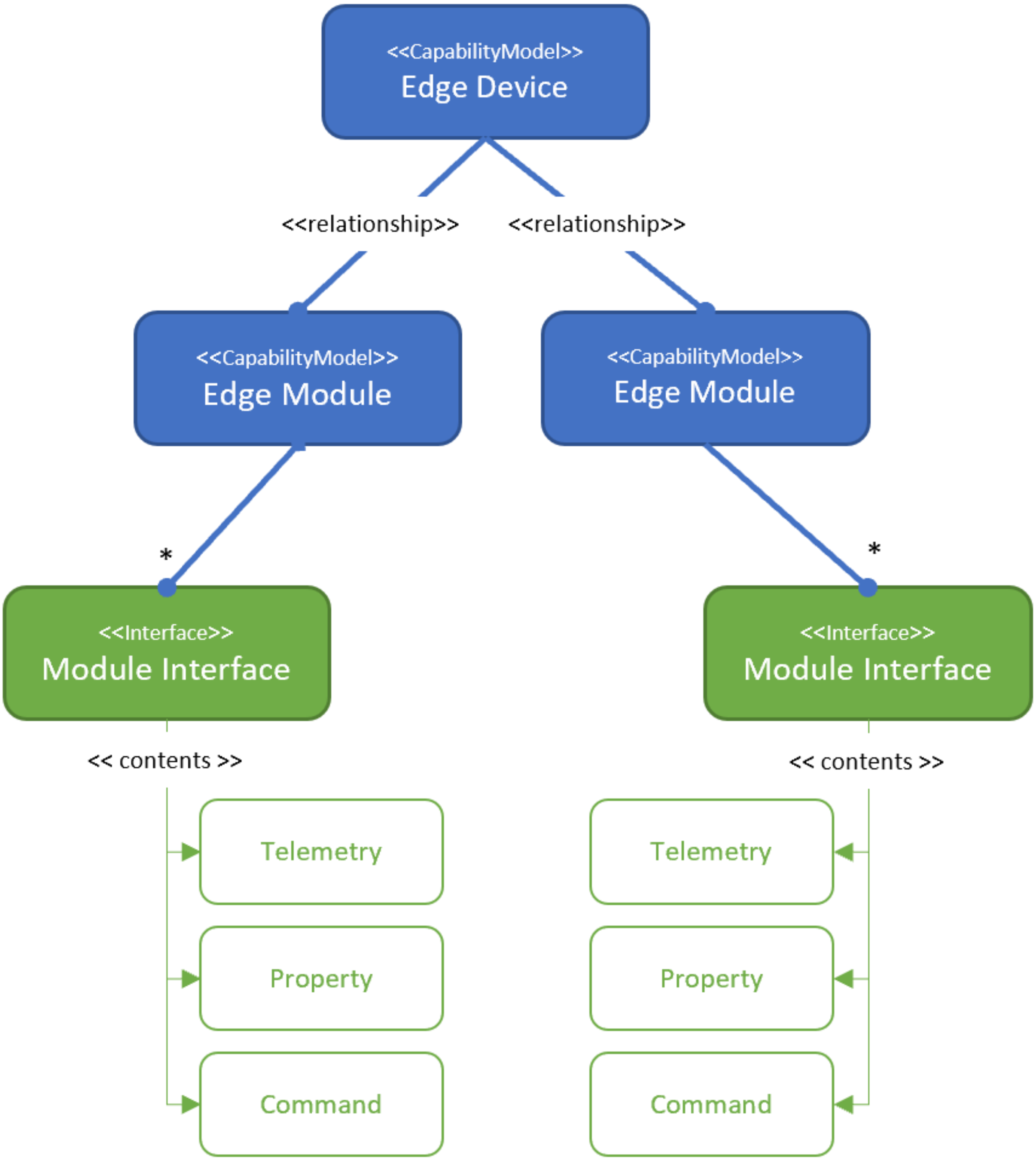
Use Visual Studio Code to create a deployment manifest. To learn more, see Azure IoT Edge for Visual Studio Code.

In Azure IoT Central, you can import a deployment manifest to create a device template. The following flowchart shows a deployment manifest life cycle in IoT Central.



IoT Plug and Play (preview) models an IoT Edge device as follows:

- Every IoT Edge device template has a device capability model.
- For every custom module listed in the deployment manifest, a module capability model is generated.
- A relationship is established between each module capability model and a device capability model.
- A module capability model implements module interfaces.
- Each module interface contains telemetry, properties, and commands.



IoT Edge gateway devices

If you selected an IoT Edge device to be a gateway device, you can add downstream relationships to device capability models for devices you want to connect to the gateway device.

Next steps

If you're a device developer, a suggested next step is to learn about gateway device types in IoT Central.

Define a new IoT device type in your Azure IoT Central application

7/21/2020 • 12 minutes to read • Edit Online

This article applies to solution builders and device developers.

A device template is a blueprint that defines the characteristics and behaviors of a type of device that connects to an Azure IoT Central application.

For example, a builder can create a device template for a connected fan that has the following characteristics:

- Sends temperature telemetry
- Sends location property
- Sends fan motor error events
- Sends fan operating state
- Provides a writeable fan speed property
- Provides a command to restart the device
- Gives you an overall view of the device via a dashboard

From this device template, an operator can create and connect real fan devices. All these fans have measurements, properties, and commands that operators use to monitor and manage them. Operators use the device dashboards and forms to interact with the fan devices.

> **NOTE**
> Only builders and administrators can create, edit, and delete device templates. Any user can create devices on the **Devices** page from existing device templates.

IoT Plug and Play (preview) enables IoT Central to integrate devices, without you writing any embedded device code. At the core of IoT Plug and Play (preview) is a device capability model schema that describes device capabilities. In an IoT Central application, device templates use these IoT Plug and Play (preview) device capability models.

As a builder, you have several options for creating device templates:

- Design the device template in IoT Central, and then implement its device capability model in your device code.
- Import a device capability model from the Azure Certified for IoT device catalog. Then add any cloud properties, customizations, and dashboards your IoT Central application needs.
- Create a device capability model by using Visual Studio Code. Implement your device code from the model. Manually import the device capability model into your IoT Central application, and then add any cloud properties, customizations, and dashboards your IoT Central application needs.
- Create a device capability model by using Visual Studio Code. Implement your device code from the model, and connect your real device to your IoT Central application by using a device-first connection. IoT Central finds and imports the device capability model from the public repository for you. You can then add any cloud properties, customizations, and dashboards your IoT Central application needs to the device template.

Create a device template from the device catalog

As a builder, you can quickly start building out your solution by using an IoT Plug and Play (preview) certified device. See the list in the Azure IoT Device Catalog. IoT Central integrates with the device catalog so you can import a device capability model from any of these IoT Plug and Play (preview) certified devices. To create a device template from one of these devices in IoT Central:

1. Go to the **Device Templates** page in your IoT Central application.
2. Select **+ New**, and then select any of the IoT Plug and Play (preview) certified devices from the catalog. IoT Central creates a device template based on this device capability model.
3. Add any cloud properties, customizations, or views to your device template.
4. Select **Publish** to make the template available for operators to view and connect devices.

Create a device template from scratch

A device template contains:

- A *device capability model* that specifies the telemetry, properties, and commands that the device implements. These capabilities are organized into one or more interfaces.
- *Cloud properties* that define information that your IoT Central application stores about your devices. For example, a cloud property might record the date a device was last serviced. This information is never shared with the device.
- *Customizations* let the builder override some of the definitions in the device capability model. For example, the builder can override the name of a device property. Property names appear in IoT Central dashboards and forms.
- *Dashboards and forms* let the builder create a UI that lets operators monitor and manage the devices connected to your application.

To create a device template in IoT Central:

1. Go to the **Device Templates** page in your IoT Central application.
2. Select **+ New** > **Custom**.
3. Enter a name for your template, such as **Environmental Sensor**.
4. Press **Enter**. IoT Central creates an empty device template.

Manage a device template

You can rename or delete a template from the template's home page.

After you've added a device capability model to your template, you can publish it. Until you've published the template, you can't connect a device based on this template for your operators to see in the **Devices** page.

Create a capability model

To create a device capability model, you can:

- Use IoT Central to create a custom model from scratch.
- Import a model from a JSON file. A device builder might have used Visual Studio Code to author a device capability model for your application.
- Select one of the devices from the Device Catalog. This option imports the device capability model that the manufacturer has published for this device. A device capability model imported like this is automatically published.

Manage a capability model

After you create a device capability model, you can:

- Add interfaces to the model. A model must have at least one interface.
- Edit model metadata, such as its ID, namespace, and name.
- Delete the model.

Create an interface

A device capability must have at least one interface. An interface is a reusable collection of capabilities.

To create an interface:

1. Go to your device capability model, and choose **+ Add Interface**.
2. On the **Select an Interface** page, you can:
    - Create a custom interface from scratch.
    - Import an existing interface from a file. A device builder might have used Visual Studio Code to author an interface for your device.
    - Choose one of the standard interfaces, such as the **Device Information** interface. Standard interfaces specify the capabilities common to many devices. These standard interfaces are published by Azure IoT, and can't be versioned or edited.
3. After you create an interface, choose **Edit Identity** to change the display name of the interface.
4. If you choose to create a custom interface from scratch, you can add your device's capabilities. Device capabilities are telemetry, properties, and commands.

Telemetry

Telemetry is a stream of values sent from the device, typically from a sensor. For example, a sensor might report the ambient temperature.

The following table shows the configuration settings for a telemetry capability:

FIELD	DESCRIPTION
Display Name	The display name for the telemetry value used on dashboards and forms.
Name	The name of the field in the telemetry message. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary. This field needs to be alphanumeric.
Capability Type	Telemetry.
Semantic Type	The semantic type of the telemetry, such as temperature, state, or event. The choice of semantic type determines which of the following fields are available.
Schema	The telemetry data type, such as double, string, or vector. The available choices are determined by the semantic type. Schema isn't available for the event and state semantic types.
Severity	Only available for the event semantic type. The severities are **Error**, **Information**, or **Warning**.

FIELD	DESCRIPTION
State Values	Only available for the state semantic type. Define the possible state values, each of which has display name, name, enumeration type, and value.
Unit	A unit for the telemetry value, such as **mph**, **%**, or **°C**.
Display Unit	A display unit for use on dashboards and forms.
Comment	Any comments about the telemetry capability.
Description	A description of the telemetry capability.

Properties

Properties represent point-in-time values. For example, a device can use a property to report the target temperature it's trying to reach. You can set writeable properties from IoT Central.

The following table shows the configuration settings for a property capability:

FIELD	DESCRIPTION
Display Name	The display name for the property value used on dashboards and forms.
Name	The name of the property. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary. This field needs to be alphanumeric.
Capability Type	Property.
Semantic Type	The semantic type of the property, such as temperature, state, or event. The choice of semantic type determines which of the following fields are available.
Schema	The property data type, such as double, string, or vector. The available choices are determined by the semantic type. Schema isn't available for the event and state semantic types.
Writeable	If the property isn't writeable, the device can report property values to IoT Central. If the property is writeable, the device can report property values to IoT Central and IoT Central can send property updates to the device.
Severity	Only available for the event semantic type. The severities are **Error**, **Information**, or **Warning**.
State Values	Only available for the state semantic type. Define the possible state values, each of which has display name, name, enumeration type, and value.
Unit	A unit for the property value, such as **mph**, **%**, or **°C**.
Display Unit	A display unit for use on dashboards and forms.

FIELD	DESCRIPTION
Comment	Any comments about the property capability.
Description	A description of the property capability.

Commands

You can call device commands from IoT Central. Commands optionally pass parameters to the device and receive a response from the device. For example, you can call a command to reboot a device in 10 seconds.

The following table shows the configuration settings for a command capability:

FIELD	DESCRIPTION
Display Name	The display name for the command used on dashboards and forms.
Name	The name of the command. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary. This field needs to be alphanumeric.
Capability Type	Command.
Command	`SynchronousExecutionType`.
Comment	Any comments about the command capability.
Description	A description of the command capability.
Request	If enabled, a definition of the request parameter, including: name, display name, schema, unit, and display unit.
Response	If enabled, a definition of the command response, including: name, display name, schema, unit, and display unit.

Manage an interface

If you haven't published the interface, you can edit the capabilities defined by the interface. After you publish the interface, if you want to make any changes, you'll need to create a new version of the device template and version the interface. You can make changes that don't require versioning, such as display names or units, in the **Customize** section.

You can also export the interface as a JSON file if you want to reuse it in another capability model.

Add cloud properties

Use cloud properties to store information about devices in IoT Central. Cloud properties are never sent to a device. For example, you can use cloud properties to store the name of the customer who has installed the device, or the device's last service date.

The following table shows the configuration settings for a cloud property:

FIELD	DESCRIPTION
Display Name	The display name for the cloud property value used on dashboards and forms.
Name	The name of the cloud property. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary.
Semantic Type	The semantic type of the property, such as temperature, state, or event. The choice of semantic type determines which of the following fields are available.
Schema	The cloud property data type, such as double, string, or vector. The available choices are determined by the semantic type.

Add customizations

Use customizations when you need to modify an imported interface or add IoT Central-specific features to a capability. You can only customize fields that don't break interface compatibility. For example, you can:

- Customize the display name and units of a capability.
- Add a default color to use when the value appears on a chart.
- Specify initial, minimum, and maximum values for a property.

You can't customize the capability name or capability type. If there are changes you can't make in the **Customize** section, you'll need to version your device template and interface to modify the capability.

Generate default views

Generating default views is a quick way to visualize your important device information. You have up to three default views generated for your device template:

- **Commands** provides a view with device commands, and allows your operator to dispatch them to your device.
- **Overview** provides a view with device telemetry, displaying charts and metrics.
- **About** provides a view with device information, displaying device properties.

After you've selected **Generate default views**, you see that they have been automatically added under the **Views** section of your device template.

Add dashboards

Add dashboards to a device template to enable operators to visualize a device by using charts and metrics. You can have multiple dashboards for a device template.

To add a dashboard to a device template:

1. Go to your device template, and select **Views**.
2. Choose **Visualizing the Device**.
3. Enter a name for your dashboard in **Dashboard Name**.
4. Add tiles to your dashboard from the list of static, property, cloud property, telemetry, and command tiles. Drag and drop the tiles you want to add to your dashboard.
5. To plot multiple telemetry values on a single chart tile, select the telemetry values, and then select **Combine**.
6. Configure each tile you add to customize how it displays data. You can do this by selecting the gear icon, or by

selecting **Change configuration** on your chart tile.
7. Arrange and resize the tiles on your dashboard.
8. Save the changes.

Configure preview device to view dashboard

To view and test your dashboard, select **Configure preview device**. This enables you to see the dashboard as your operator sees it after it's published. Use this option to validate that your views show the correct data. You can choose from the following:

- No preview device.
- The real test device you've configured for your device template.
- An existing device in your application, by using the device ID.

Add forms

Add forms to a device template to enable operators to manage a device by viewing and setting properties. Operators can only edit cloud properties and writeable device properties. You can have multiple forms for a device template.

To add a form to a device template:

1. Go to your device template, and select **Views**.
2. Choose **Editing Device and Cloud data**.
3. Enter a name for your form in **Form Name**.
4. Select the number of columns to use to lay out your form.
5. Add properties to an existing section on your form, or select properties and choose **Add Section**. Use sections to group properties on your form. You can add a title to a section.
6. Configure each property on the form to customize its behavior.
7. Arrange the properties on your form.
8. Save the changes.

Publish a device template

Before you can connect a device that implements your device capability model, you must publish your device template.

After you publish a device template, you can only make limited changes to the device capability model. To modify an interface, you need to create and publish a new version.

To publish a device template, go to you your device template, and select **Publish**.

After you publish a device template, an operator can go to the **Devices** page, and add either real or simulated devices that use your device template. You can continue to modify and save your device template as you're making changes. When you want to push these changes out to the operator to view under the **Devices** page, you must select **Publish** each time.

Next steps

If you're a device developer, a suggested next step is to read about device template versioning.

Connect an MXChip IoT DevKit device to your Azure IoT Central application

4/21/2020 • 3 minutes to read • Edit Online

This article applies to device developers.

This article shows you how to connect an MXChip IoT DevKit (DevKit) device to an Azure IoT Central application. The device uses the certified IoT Plug and Play (preview) model for the DevKit device to configure its connection to IoT Central.

In this how-to article, you:

- Get the connection details from your IoT Central application.
- Prepare the device and connect it to your IoT Central application.
- View the telemetry and properties from the device in IoT Central.

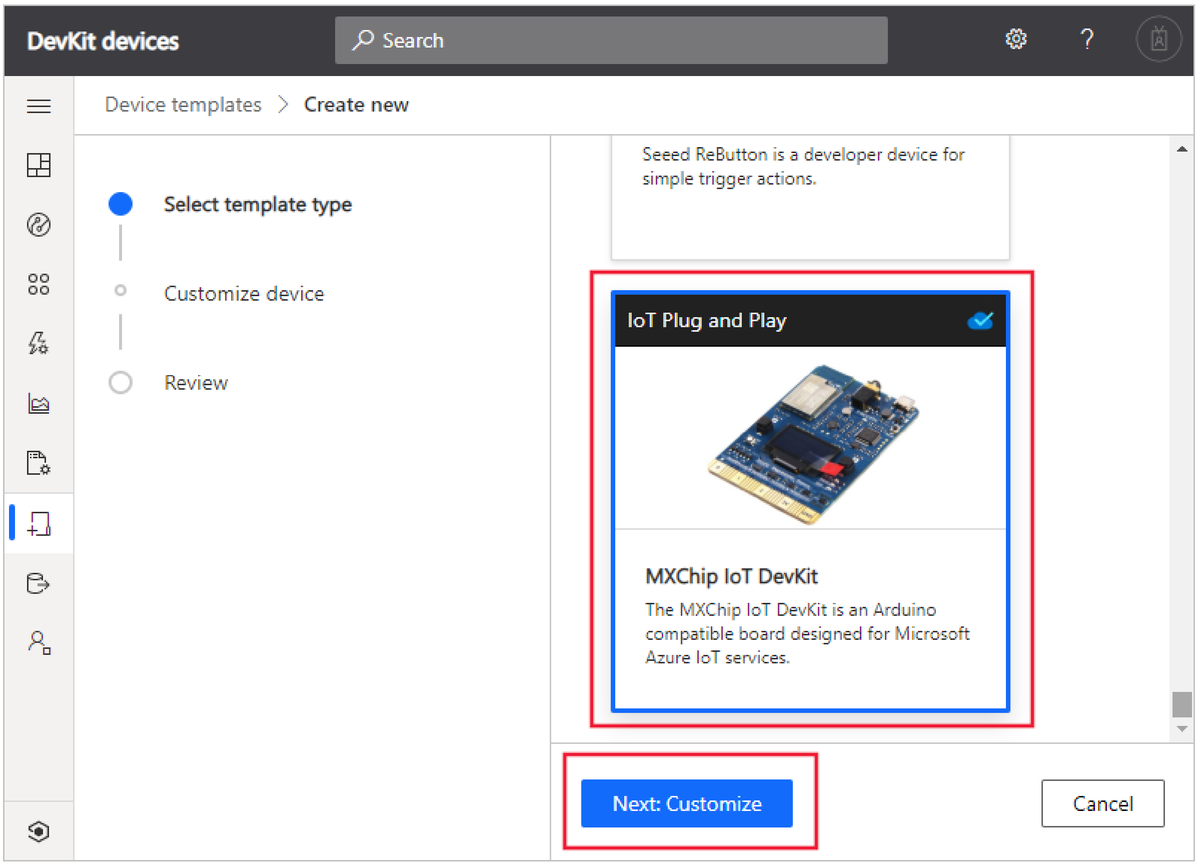
Prerequisites

To complete the steps in this article, you need the following resources:

- A DevKit device.
- An IoT Central application. You can follow the steps in Create an IoT Central application.

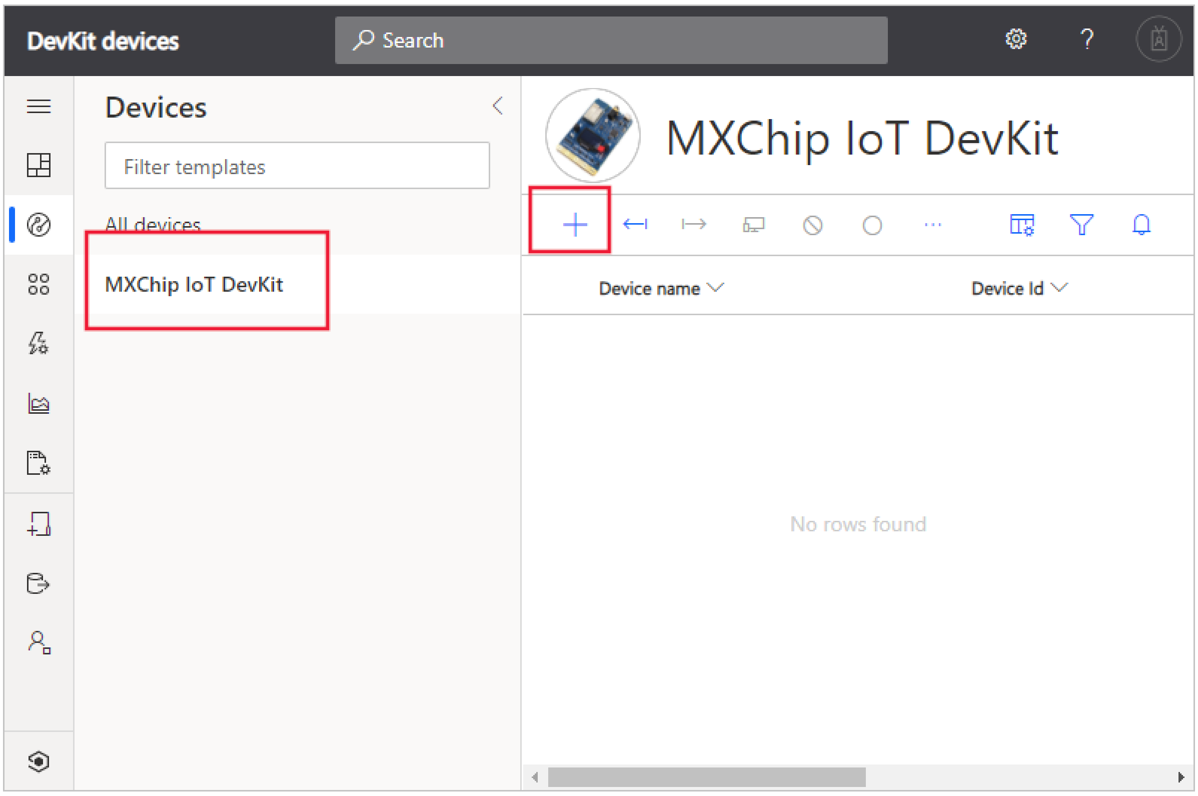
Get device connection details

1. In your Azure IoT Central application, select the **Device Templates** tab and select **+ New**. In the section **Use a preconfigured device template**, select **MXChip IoT DevKit**.

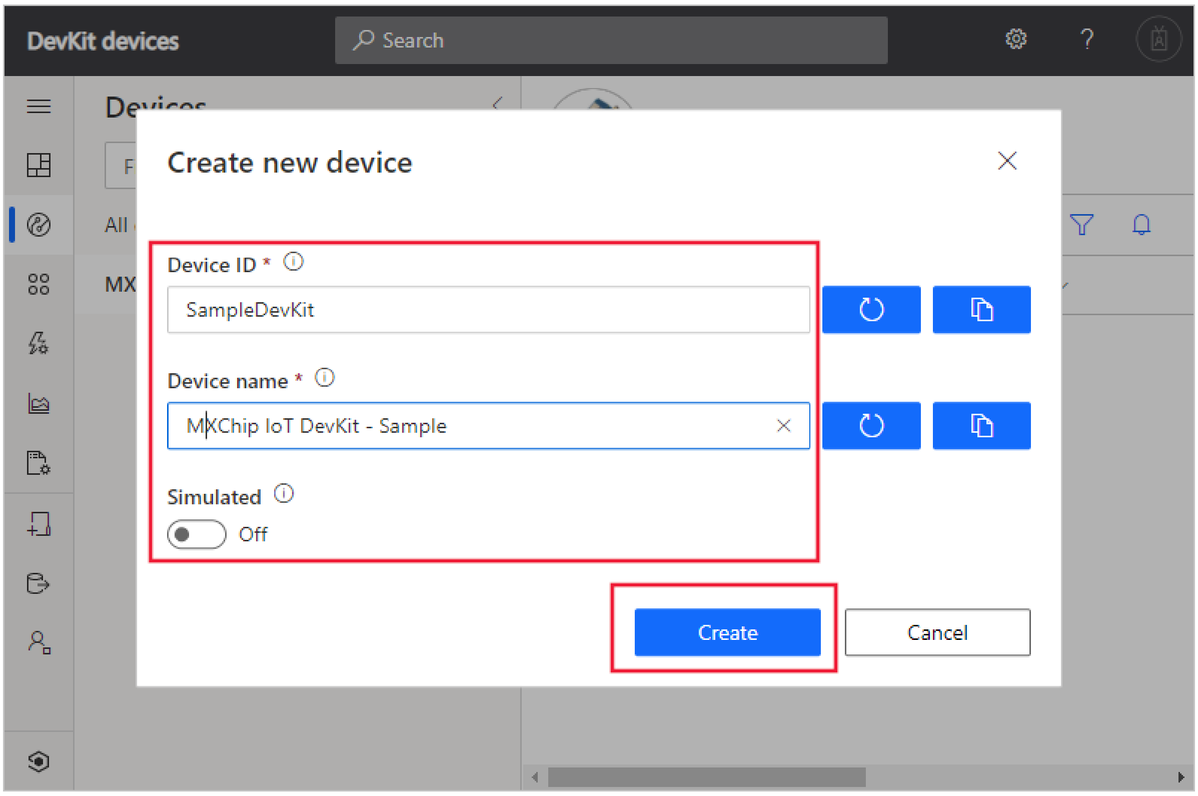


2. Select **Next: Customize** and then **Create**.

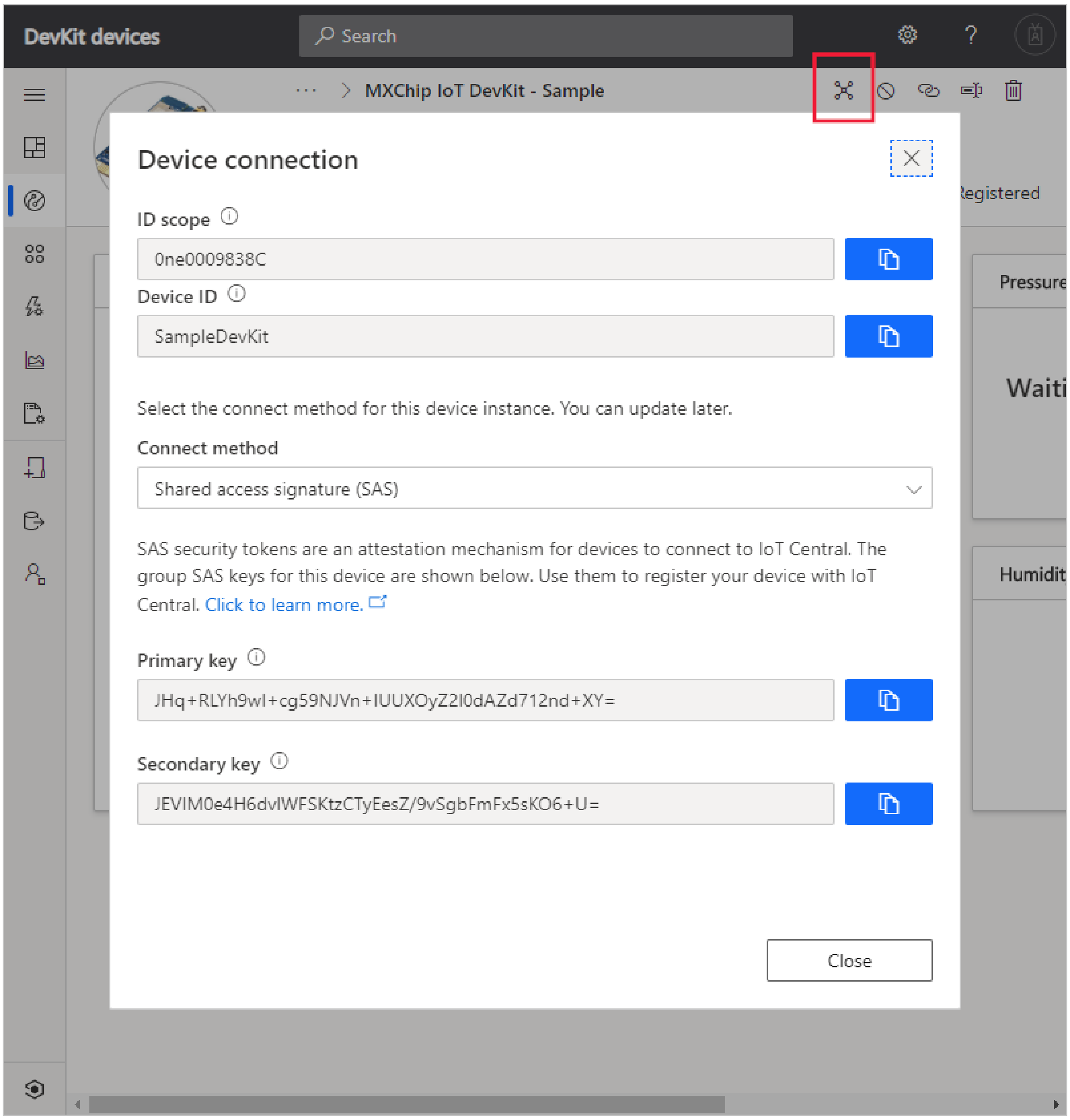
3. Select **Devices** tab. In the devices list, select **MXChip IoT DevKit** and select **+ New** to create a new device from the template.



4. In the pop-up window, enter the **Device ID** as `SampleDevKit` and **Device Name** as `MXChip IoT DevKit - Sample`. Make sure the **Simulated** option is off. Then select **Create**.



5. Select the device you created and then select **Connect**. Make a note of the **ID Scope**, **Device ID**, and **Primary key**. You need these values later in this how-to article.



Prepare the device

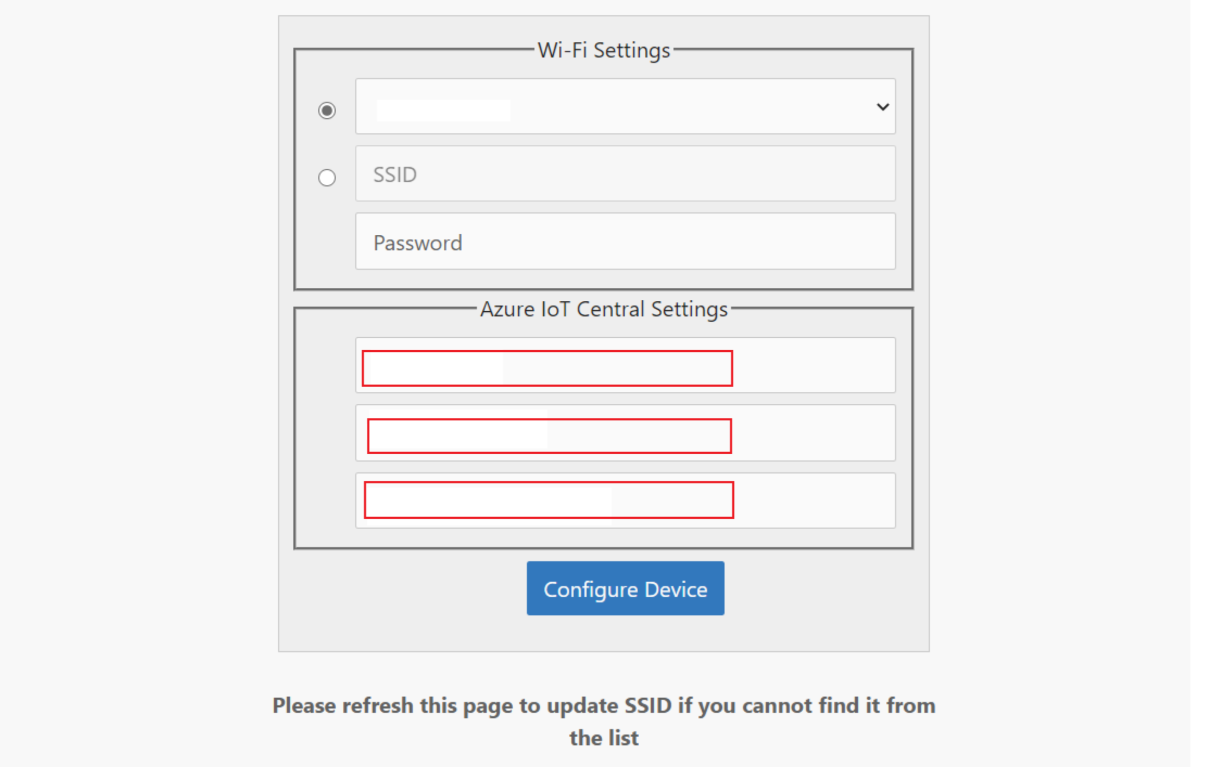
1. Download the latest pre-built Azure IoT Central Plug and Play (preview) firmware for the DevKit device from GitHub.
2. Connect the DevKit device to your development machine using a USB cable. In Windows, a file explorer window opens on a drive mapped to the storage on the DevKit device. For example, the drive might be called **AZ3166 (D:)**.
3. Drag the **iotc_devkit.bin** file onto the drive window. When the copying is complete, the device reboots with the new firmware.

> **NOTE**
>
> If you see errors on the screen such as **No Wi-Fi**, this is because the DevKit has not yet been connected to WiFi.

4. On the DevKit, hold down **button B**, push and release the **Reset** button, and then release **button B**. The device is now in access point mode. To confirm, the screen displays "IoT DevKit - AP" and the configuration portal IP address.
5. On your computer or tablet, connect to the WiFi network name shown on the screen of the device. The WiFi

network starts with **AZ-** followed by the MAC address. When you connect to this network, you don't have internet access. This state is expected, and you only connect to this network for a short time while you configure the device.

6. Open your web browser and navigate to http://192.168.0.1/. The following web page displays:

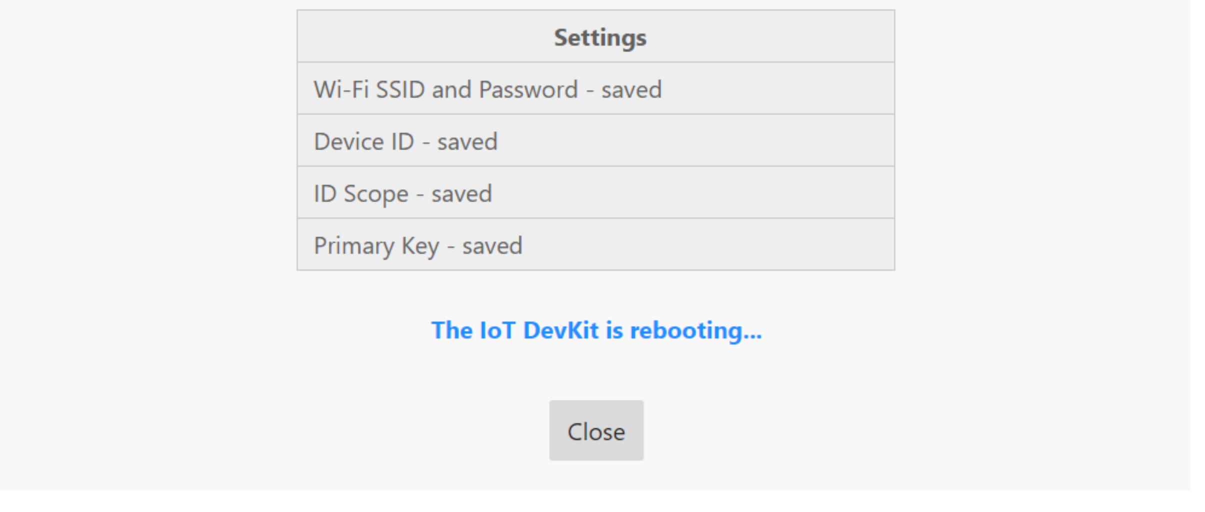


On the web page, enter:

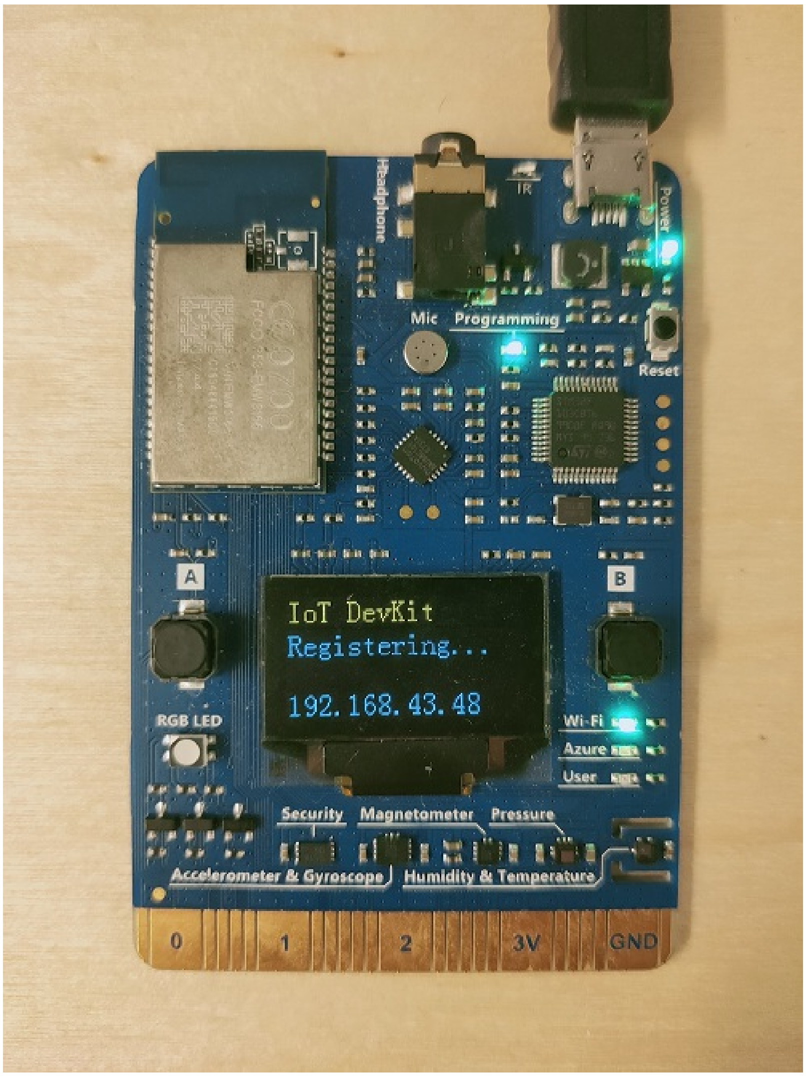
- The name of your WiFi network (SSID).
- Your WiFi network password.
- The connection details: enter the **Device ID**, **ID Scope**, and **SAS Primary Key** you made a note of previously.

> **NOTE**
>
> Currently, the IoT DevKit only can connect to 2.4 GHz Wi-Fi, 5 GHz is not supported due to hardware restrictions.

7. Choose **Configure Device**, the DevKit device reboots and runs the application:



The DevKit screen displays a confirmation that the application is running:

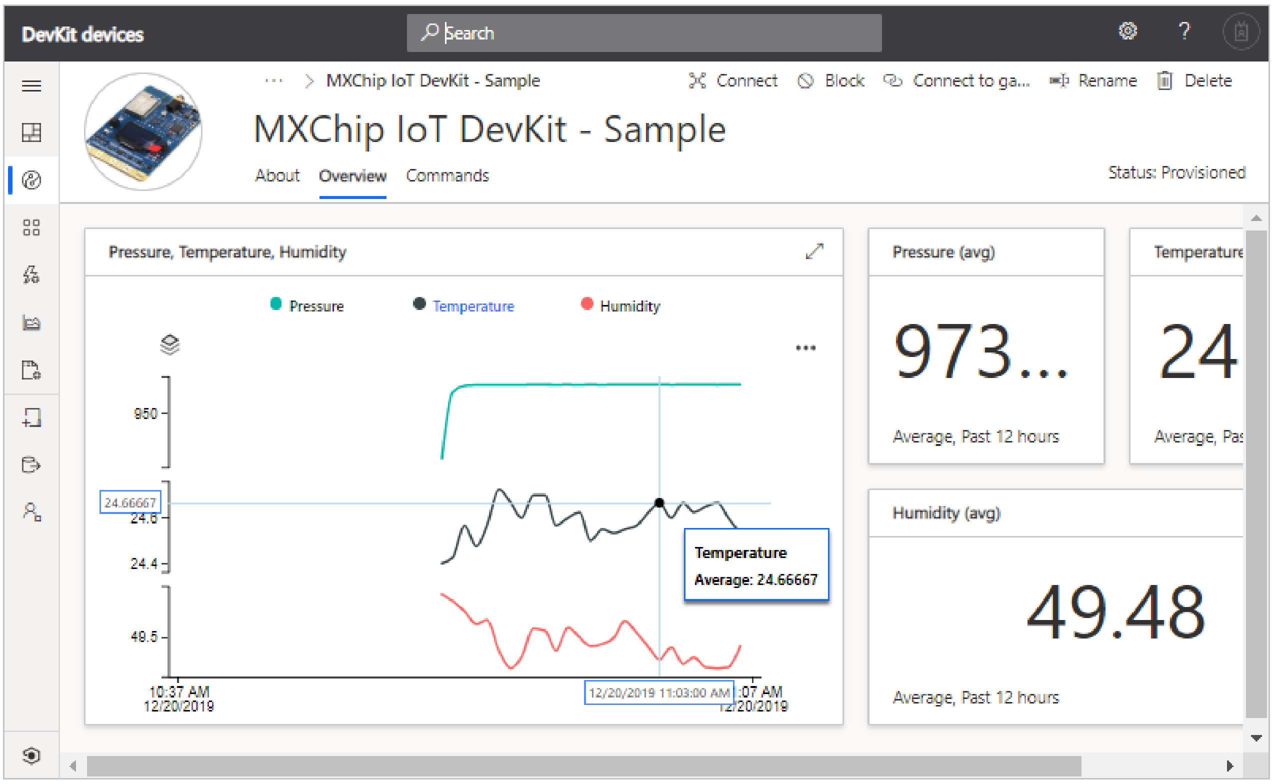


The DevKit first registers a new device in IoT Central application and then starts sending data.

View the telemetry

In this step, you view the telemetry in your Azure IoT Central application.

In your IoT Central application, select **Devices** tab, select the device you added. In the **Overview** tab, you can see the telemetry from the DevKit device:



Review the code

To review the code or modify and compile it, go to the Code Samples.

Next steps

If you're a device developer, some suggested next steps are to:

- Read about Device connectivity in Azure IoT Central
- Learn how to Monitor device connectivity using Azure CLI

Connect an Azure Sphere device to your Azure IoT Central application

5/20/2020 • 2 minutes to read • Edit Online

This article applies to device developers.

This article shows you how to connect an Azure Sphere (DevKit) device to an Azure IoT Central application.

Azure Sphere is a secured, high-level application platform with built-in communication and security features for internet-connected devices. It includes a secured, connected, crossover microcontroller unit (MCU), a custom high-level Linux-based operating system (OS), and a cloud-based security service that provides continuous, renewable security. For more information, see What is Azure Sphere?.

Azure Sphere development kits provide everything you need to start prototyping and developing Azure Sphere applications. Azure IoT Central with Azure Sphere enables an end-to-end stack for an IoT Solution. Azure Sphere provides the device support and IoT Central as a zero-code, managed IoT application platform.

In this how-to article, you:

- Create an Azure Sphere device in IoT Central using the Azure Sphere DevKit device template from the library.
- Prepare Azure Sphere DevKit device for Azure IoT.
- Connect Azure Sphere DevKit to Azure IoT Central.
- View the telemetry from the device in IoT Central.

Prerequisites

To complete the steps in this article, you need the following resources:

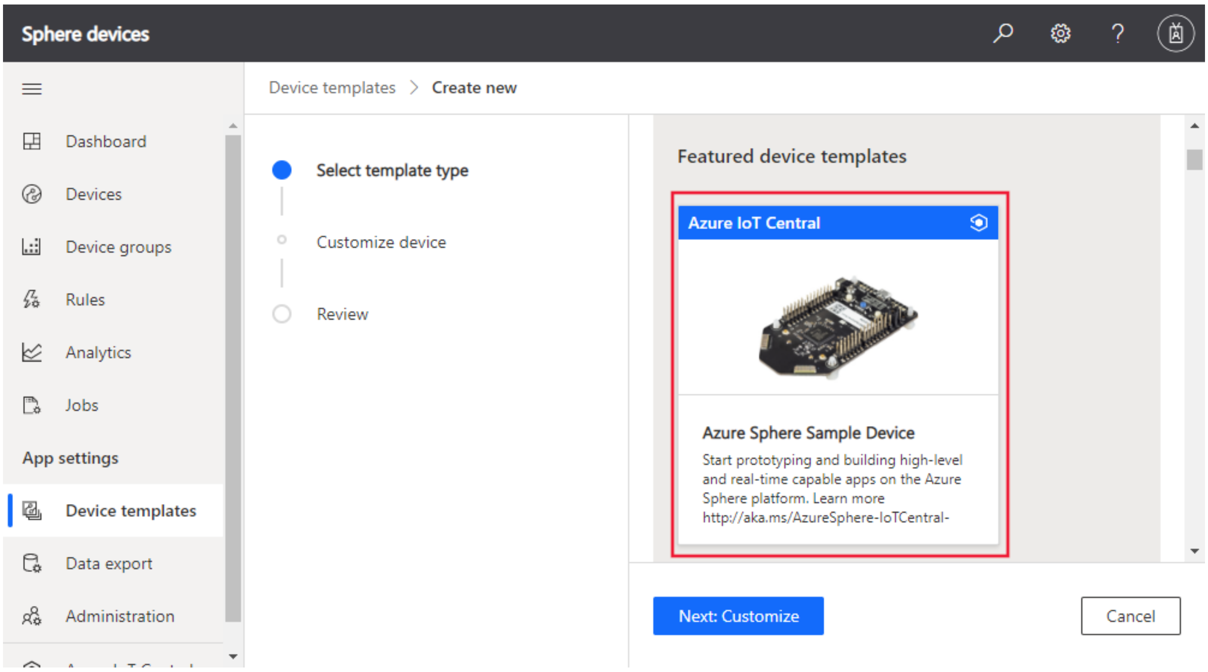
- An Azure IoT Central application.
- Visual Studio 2019, version 16.4 or later.
- An Azure Sphere MT3620 development kit from Seeed Studios.

> **NOTE**
> If you don't have a physical device, then after the first step step skip to the last section to try a simulated device.

Create the device in IoT Central

To create an Azure Sphere device in IoT Central:

1. In your Azure IoT Central application, select the **Device Templates** tab and select **+ New**. In the section **Use a featured device template**, select **Azure Sphere Sample Device**.



2. In the device template, edit the view called **Overview** to show **Temperature** and **Button Press**.
3. Select the **Editing Device and Cloud Data** view type to add another view that shows the read/write property **Status LED**. Drag the **Status LED** property to the empty, dotted rectangle on the right-side of the form. Select **Save**.

Prepare the device

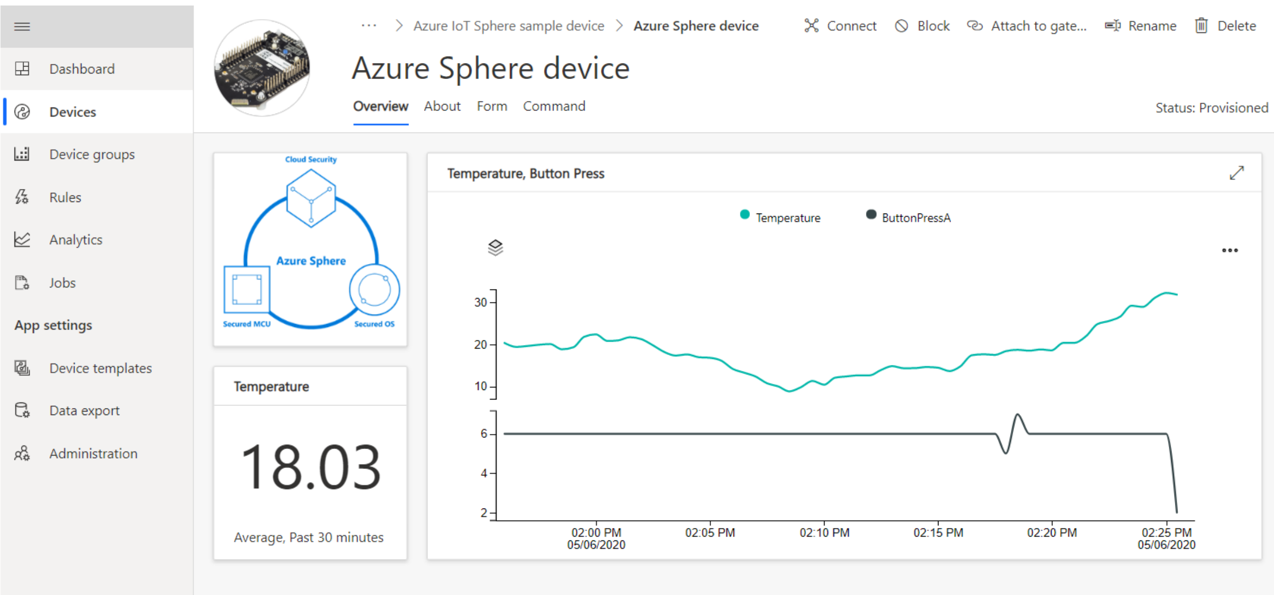
Before you can connect the Azure Sphere DevKit device to IoT Central, you need to setup the device and development environment.

Connect the device

To enable the sample to connect to IoT Central, you must configure an Azure IoT Central application and then modify the sample's application manifest.

View the telemetry from the device

When the device is connected to IoT Central, you can see the telemetry on the dashboard.



Create a simulated device

If you don't have a physical Azure Sphere DevKit device, you can create a simulated device to try Azure IoT Central application.

To create a simulated device:

- Select **Devices > Azure IoT Sphere**
- Select **+ New**.
- Enter a unique **Device ID** and a friendly **Device name**.
- Enable the **Simulated** setting.
- Select **Create**.

Next steps

If you're a device developer, some suggested next steps are to:

- Read about Device connectivity in Azure IoT Central
- Learn how to Monitor device connectivity using Azure CLI

Connect a RuuviTag sensor to your Azure IoT Central application

4/21/2020 • 2 minutes to read • Edit Online

This article applies to solution builders and device developers.

This article describes how, as a solution builder, you can connect a RuuviTag sensor to your Microsoft Azure IoT Central application.

What is a Ruuvi tag?

RuuviTag is an advanced open-source sensor beacon platform designed to fulfill the needs of business customers, developers, makers, students, and hobbyists. The device is set up to work as soon as you take it out of its box and is ready for you to deploy it where you need it. It's a Bluetooth LE beacon with an environment sensor and accelerometer built in.

RuuviTag communicates over BLE (Bluetooth Low Energy) and requires a gateway device to talk to Azure IoT Central. Make sure you have a gateway device, such as the Rigado Cascade 500, setup to enable a RuuviTag to connect to IoT Central.

Please follow the instructions here if you'd like to set up a Rigado Cascade 500 gateway device.

Prerequisites

To connect RuuviTag sensors, you need the following resources:

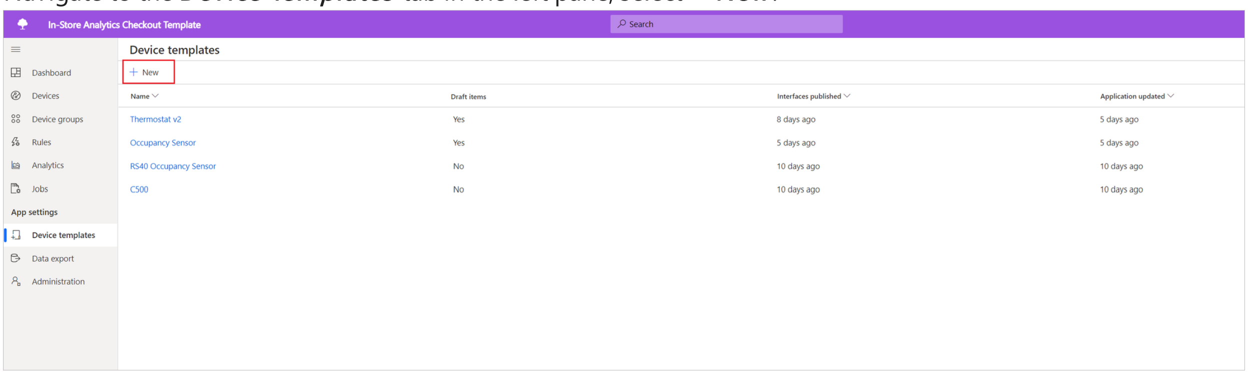
- A RuuviTag sensor. For more information, please visit RuuviTag.
- A Rigado Cascade 500 device or another BLE gateway. For more information, please visit Rigado.
- An Azure IoT Central application. For more information, see the create a new application.

Add a RuuviTag device template

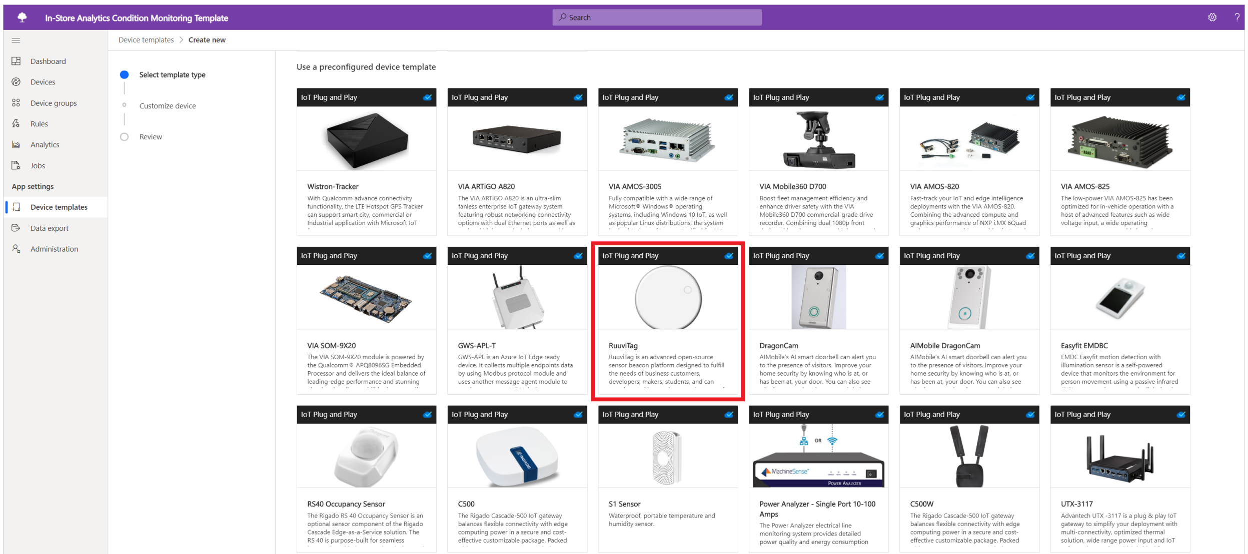
To onboard a RuuviTag sensor into your Azure IoT Central application instance, you need to configure a corresponding device template within your application.

To add a RuuviTag device template:

1. Navigate to the ***Device Templates*** tab in the left pane, select **+ New**:



   The page gives you an option to ***Create a custom template*** or ***Use a preconfigured device template***
2. Select the RuuviTag device template from the list of preconfigured device templates as shown below:

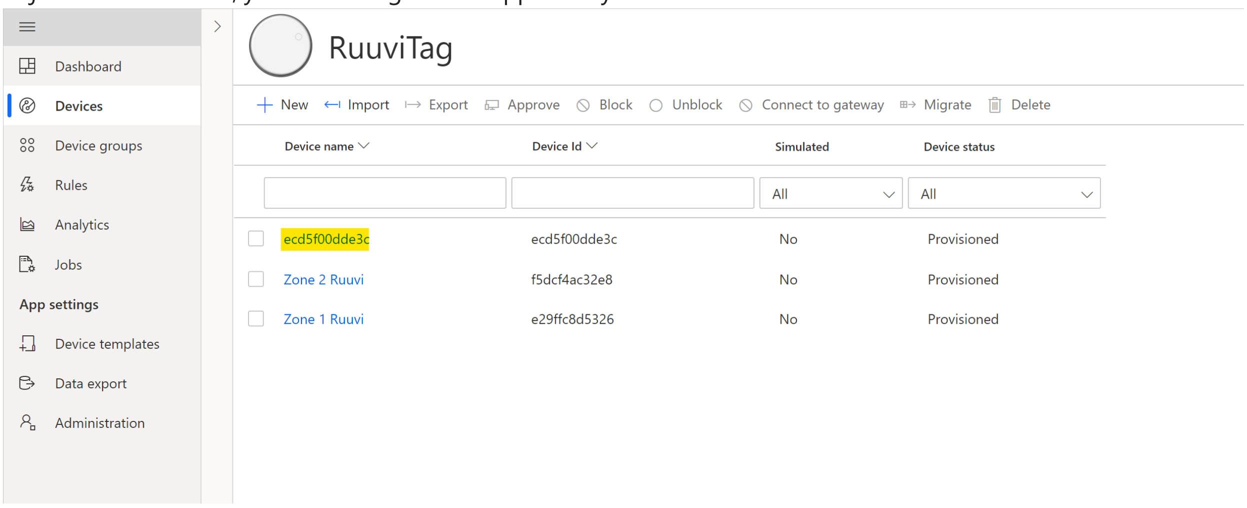


3. Select ***Next: Customize*** to continue to the next step.
4. On the next screen, select ***Create*** to onboard the C500 device template into your IoT Central application.

Connect a RuuviTag sensor

As mentioned previously, to connect the RuuviTag with your IoT Central application, you need to set up a gateway device. The steps below assume that you've set up a Rigado Cascade 500 gateway device.

1. Power on your Rigado Cascade 500 device and connect it to your network connection (via Ethernet or wireless)
2. Pop the cover off of the RuuviTag and pull the plastic tab to secure the connection with the battery.
3. Place the RuuviTag close to a Rigado Cascade 500 gateway that's already configured in your IoT Central application.
4. In just a few seconds, your RuuviTag should appear in your list of devices within IoT Central.



You can now use this RuuviTag within your IoT Central application.

Create a simulated RuuviTag

If you don't have a physical RuuviTag device, you can create a simulated RuuviTag sensor to use for testing within your Azure IoT Central application.

To create a simulated RuuviTag:

1. Select **Devices** > **RuuviTag**.
2. Select **+ New**.
3. Specify a unique **Device ID** and a friendly **Device name**.

4. Enable the **Simulated** setting.
5. Select **Create**.

Next Steps

If you're a device developer, some suggested next steps are to:

- Read about Device connectivity in Azure IoT Central
- Learn how to Monitor device connectivity using Azure CLI

Connect a Rigado Cascade 500 gateway device to your Azure IoT Central application

4/21/2020 • 2 minutes to read • Edit Online

This article applies to solution builders and device developers.

This article describes how, as a solution builder, you can connect a Rigado Cascade 500 gateway device to your Microsoft Azure IoT Central application.

What is Cascade 500?

Cascade 500 IoT gateway is a hardware offering from Rigado that is included as part of their Cascade Edge-as-a-Service solution. It provides commercial IoT project and product teams with flexible edge computing power, a robust containerized application environment, and a wide variety of wireless device connectivity options, including Bluetooth 5, LTE, & Wi-Fi.

Cascade 500 is pre-certified for Azure IoT Plug and Play (preview) allowing our solution builders to easily onboard the device into their end to end solutions. The Cascade gateway allows you to wirelessly connect to a variety of condition monitoring sensors that are in proximity to the gateway device. These sensors can be onboarded into IoT Central via the gateway device.

Prerequisites

To step through this how-to guide, you need the following resources:

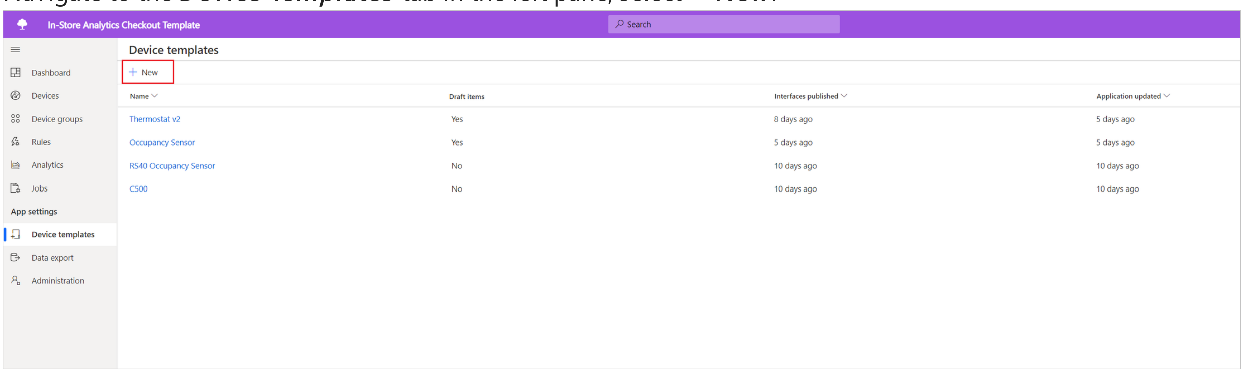
- A Rigado Cascade 500 device. For more information, please visit Rigado.
- An Azure IoT Central application. For more information, see the create a new application.

Add a device template

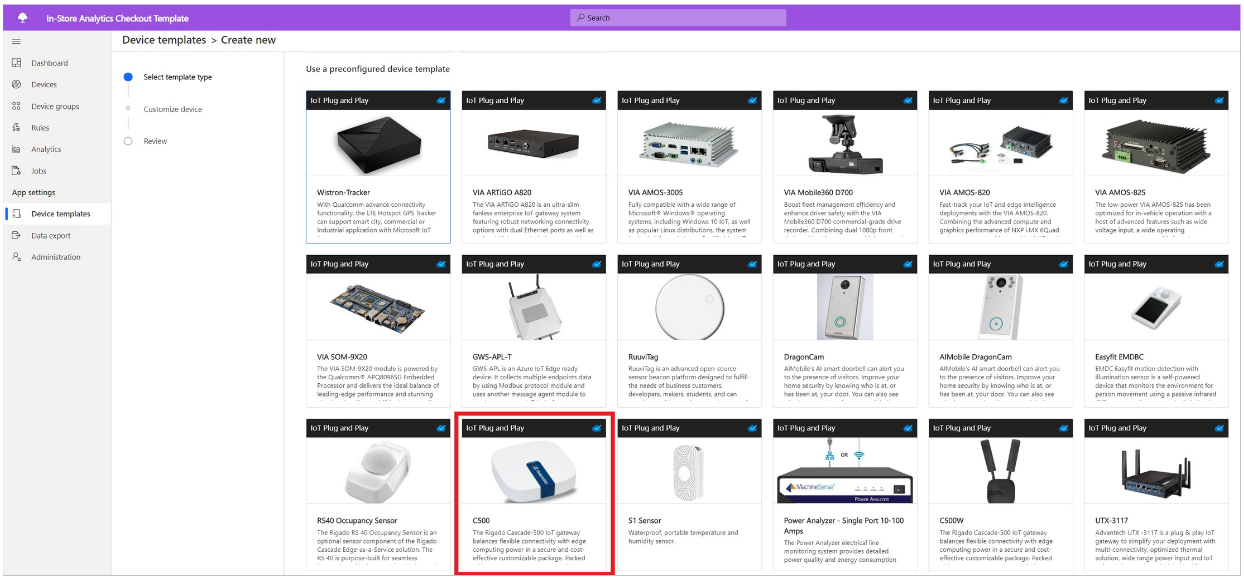
In order to onboard a Cascade 500 gateway device into your Azure IoT Central application instance, you will need to configure a corresponding device template within your application.

To add a Cascade 500 device template:

1. Navigate to the *Device Templates* tab in the left pane, select **+ New**:



2. The page gives you an option to *Create a custom template* or *Use a preconfigured device template*
3. Select the C500 device template from the list of preconfigured device templates as shown below:

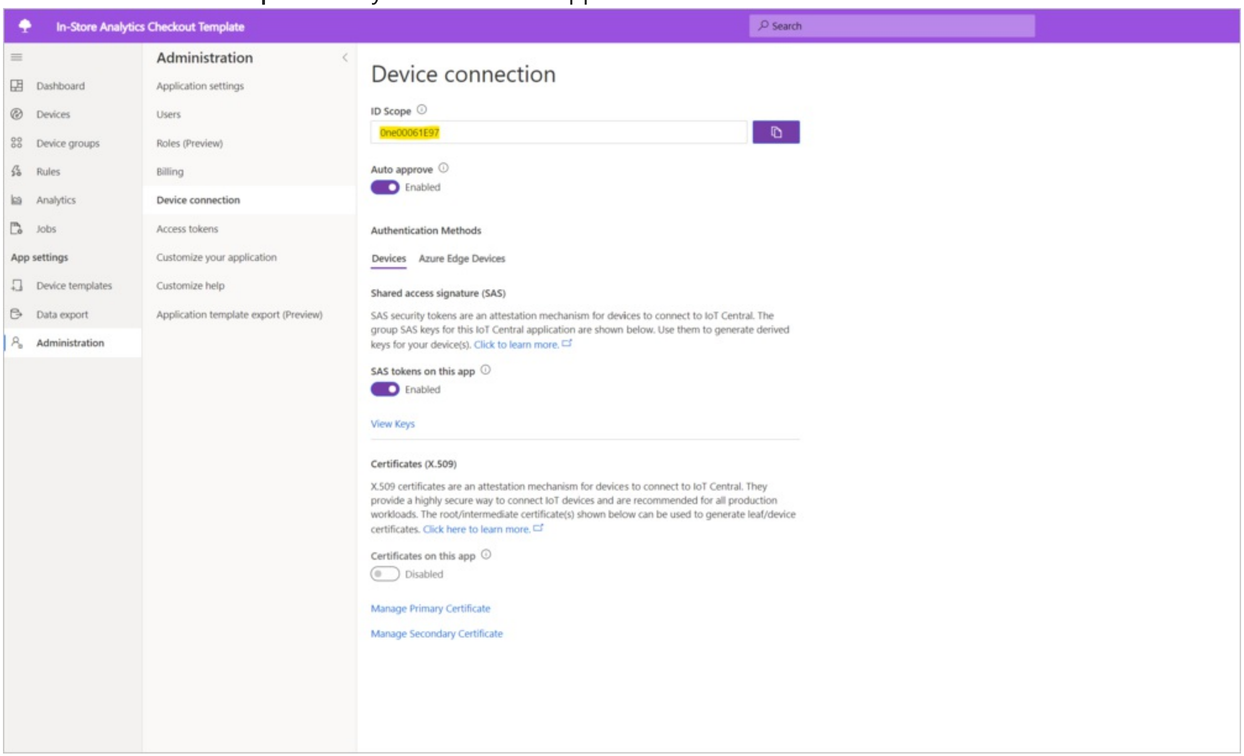


4. Select ***Next: Customize*** to continue to the next step.
5. On the next screen, select ***Create*** to onboard the C500 device template into your IoT Central application.

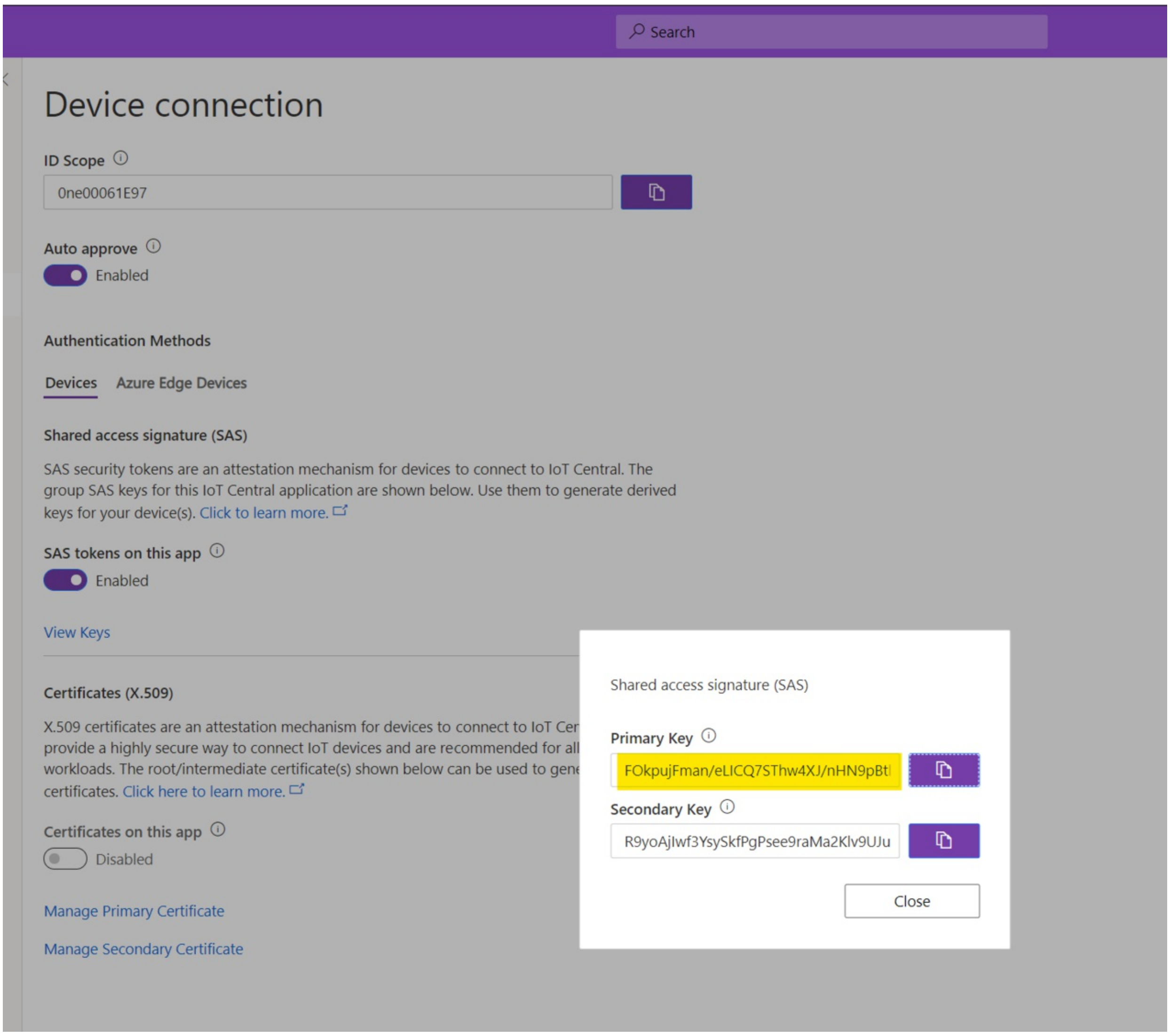
Retrieve application connection details

You will now need to retrieve the **Scope ID** and **Primary key** for your Azure IoT Central application in order to connect the Cascade 500 device.

1. Navigate to **Administration** in the left pane and click on **Device connection**.
2. Make a note of the **Scope ID** for your IoT Central application.



3. Now click on **View Keys** and make a note of the **Primary key**

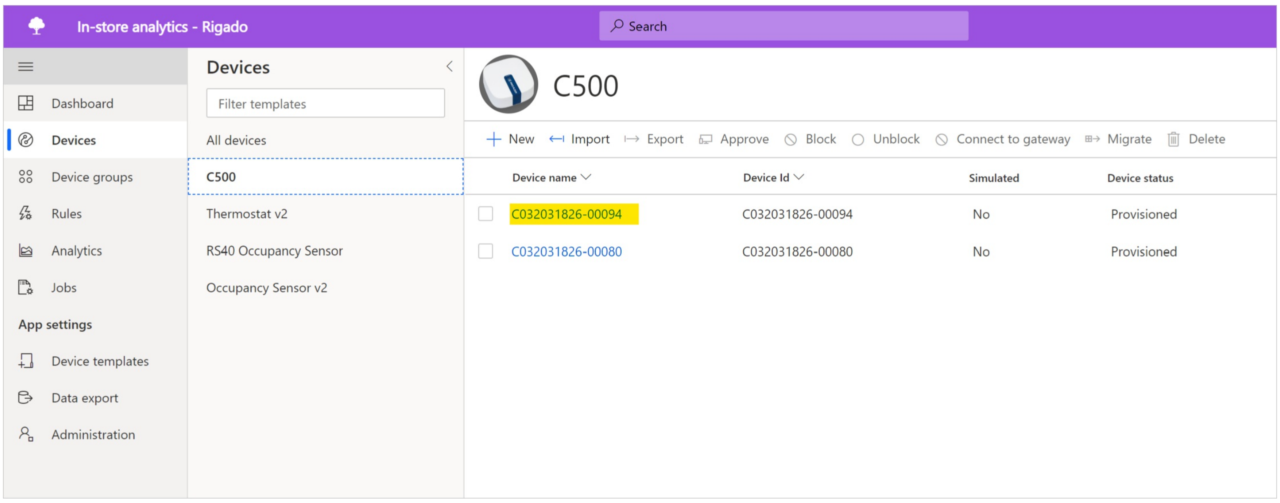


Contact Rigado to connect the gateway

In order to connect the Cascade 500 device to your IoT Central application, you will need to contact Rigado and provide them with the application connection details from the above steps.

Once the device is connected to the internet, Rigado will be able to push down a configuration update down to the Cascade 500 gateway device through a secure channel.

This update will apply the IoT Central connection details on the Cascade 500 device and it will appear in your devices list.



You are now ready to use your C500 device in your IoT Central application!

Next steps

If you're a device developer, some suggested next steps are to:

- Read about Device connectivity in Azure IoT Central
- Learn how to Monitor device connectivity using Azure CLI

Build the IoT Central device bridge to connect other IoT clouds to IoT Central

3/24/2020 • 2 minutes to read • Edit Online

This topic applies to administrators.

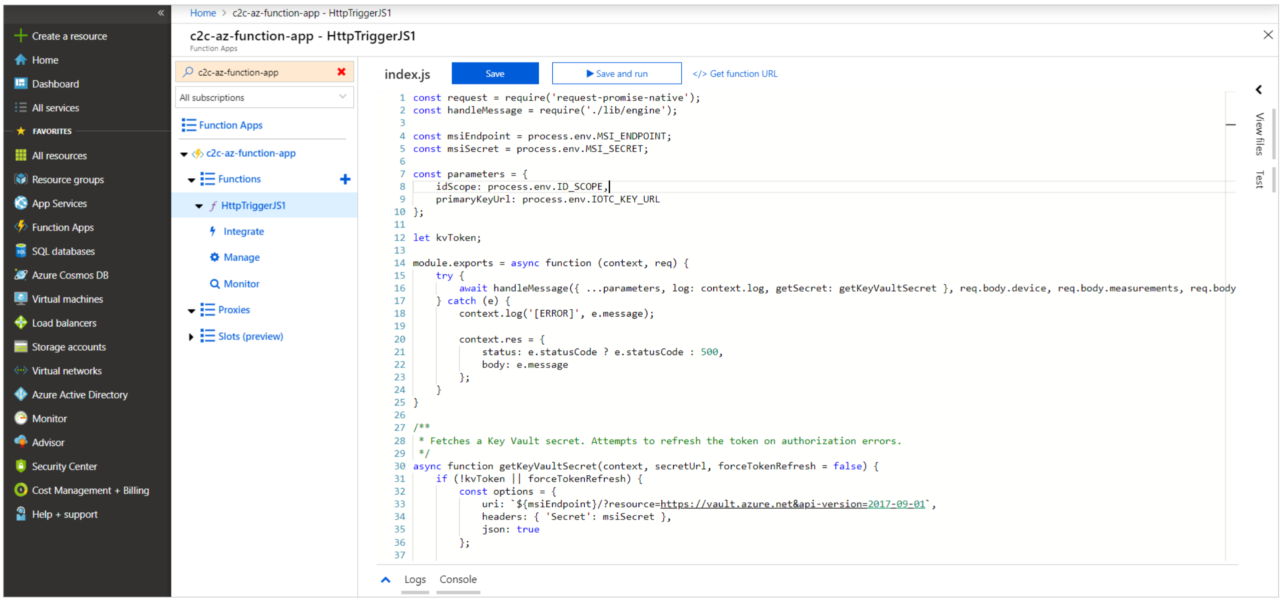
The IoT Central device bridge is an open-source solution that connects your Sigfox, Particle, The Things Network, and other clouds to your IoT Central app. Whether you are using asset tracking devices connected to Sigfox's Low-Power-Wide Area Network, or using air quality monitoring devices on the Particle Device Cloud, or using soil moisture monitoring devices on TTN, you can directly leverage the power of IoT Central using the IoT Central device bridge. The device bridge connects other IoT clouds with IoT Central by forwarding the data your devices send to the other clouds through to your IoT Central app. In your IoT Central app, you can build rules and run analytics on that data, create workflows in Microsoft Flow and Azure Logic apps, export that data, and much more. Get the IoT Central device bridge from GitHub

What is it and how does it work?

The IoT Central device bridge is an open-source solution in GitHub. It is ready to go with a "Deploy to Azure" button that deploys a custom Azure Resource Manager template with several Azure resources into your Azure subscription. The resources include:

- Azure Function app
- Azure Storage Account
- Consumption Plan
- Azure Key Vault

The function app is the critical piece of the device bridge. It receives HTTP POST requests from other IoT platforms or any custom platforms via a simple webhook integration. We have provided examples that show how to connect to Sigfox, Particle, and TTN clouds. You can easily extend this solution to connect to your custom IoT cloud if your platform can send HTTP POST requests to your function app. The Function app transforms the data into a format accepted by IoT Central and forwards it along via DPS APIs.



If your IoT Central app recognizes the device by device ID in the forwarded message, a new measurement will appear for that device. If the device ID has never been seen by your IoT Central app, your function app will attempt to register a new device with that device ID, and it will appear as an "Unassociated device" in your IoT Central app.

How do I set it up?

The instructions are listed in detail in the README file in the GitHub repo.

Pricing

The Azure resources will be hosted in your Azure subscription. You can learn more about pricing in the README file.

Next steps

Now that you've learned how to build the IoT Central device bridge, here is the suggested next step:

Manage your devices

Monitor device connectivity using Azure CLI

4/21/2020 • 2 minutes to read • Edit Online

This topic applies to device developers and solution builders.

Use the Azure CLI IoT extension to see messages your devices are sending to IoT Central and observe changes in the device twin. You can use this tool to debug and observe device connectivity and diagnose issues of device messages not reaching the cloud or devices not responding to twin changes.

Visit the Azure CLI extensions reference for more details

Prerequisites

- Azure CLI installed and is version 2.0.7 or higher. Check the version of your Azure CLI by running `az --version`. Learn how to install and update from the Azure CLI docs
- A work or school account in Azure, added as a user in an IoT Central application.

Install the IoT Central extension

Run the following command from your command line to install:

```
az extension add --name azure-iot
```

Check the version of the extension by running:

```
az --version
```

You should see the azure-iot extension is 0.8.1 or higher. If it is not, run:

```
az extension update --name azure-iot
```

Using the extension

The following sections describe common commands and options that you can use when you run `az iot central`. To view the full set of commands and options, pass `--help` to `az iot central` or any of its subcommands.

Login

Start by signing into the Azure CLI.

```
az login
```

Get the Application ID of your IoT Central app

In **Administration/Application Settings**, copy the **Application ID**. You use this value in later steps.

Monitor messages

Monitor the messages that are being sent to your IoT Central app from your devices. The output includes all headers and annotations.

```
az iot central app monitor-events --app-id <app-id> --properties all
```

View device properties

View the current read and read/write device properties for a given device.

```
az iot central device-twin show --app-id <app-id> --device-id <device-id>
```

Next steps

If you're a device developer, a suggested next step is to read about Device connectivity in Azure IoT Central.

Create a new device template version

4/30/2020 • 5 minutes to read • Edit Online

This article applies to solution builders and device developers.

A device template includes a schema that describes how a device interacts with IoT Central. These interactions include telemetry, properties, and commands. Both the device and the IoT Central application rely on a shared understanding of this schema to exchange information. You can only make limited changes to the schema without breaking the contract, that's why most schema changes require a new version of the device template. Versioning the device template lets older devices continue with the schema version they understand, while newer or updated devices use a later schema version.

The schema in a device template is defined in the device capability model (DCM) and its interfaces. Device templates include other information, such as cloud properties, display customizations, and views. If you make changes to those parts of the device template that don't define how the device exchanges data with IoT Central, you don't need to version the template.

You must publish any device template changes, whether or not they require a version update, before an operator can use it. IoT Central stops you from publishing breaking changes to a device template without first versioning the template.

> **NOTE**
> To learn more about how to create a device template see Set up and manage a device template

Versioning rules

This section summarizes the versioning rules that apply to device templates. Both DCMs and interfaces have version numbers. The following snippet shows the DCM for an environmental sensor device. The DCM has two interfaces: **DeviceInformation** and **EnvironmentalSensor**. You can see the version numbers at the end of the `@id` fields. To view this information in the IoT Central UI, select **View identity** in the device template editor.

```json
{
  "@id": "urn:contoso:sample_device:1",
  "@type": "CapabilityModel",
  "implements": [
    {
      "@id": "urn:contoso:sample_device:deviceinfo:1",
      "@type": "InterfaceInstance",
      "name": "deviceinfo",
      "schema": {
        "@id": "urn:azureiot:DeviceManagement:DeviceInformation:1",
        "@type": "Interface",
        "displayName": {
          "en": "Device Information"
        },
        "contents": [...
        ]
      }
    },
    {
      "@id": "urn:contoso:sample_device:sensor:1",
      "@type": "InterfaceInstance",
      "name": "sensor",
      "schema": {
        "@id": "urn:contoso:EnvironmentalSensor:2",
        "@type": "Interface",
        "displayName": {
          "en": "Environmental Sensor"
        },
        "contents": [...
        ]
      }
    }
  ],
  "displayName": {
    "en": "Environment Sensor Capability Model"
  },
  "@context": [
    "http://azureiot.com/v1/contexts/IoTModel.json"
  ]
}
```

- After a DCM is published, you can't remove any interfaces, even in a new version of the device template.
- After a DCM is published, you can add an interface if you create a new version of the device template.
- After a DCM is published, you can replace an interface with a newer version if you create a new version of the device template. For example, if the sensor v1 device template uses the EnvironmentalSensor v1 interface, you can create a sensor v2 device template that uses the EnvironmentalSensor v2 interface.
- After an interface is published, you can't remove any of the interface contents, even in a new version of the device template.
- After an interface is published, you can add items to the contents of an interface if you create a new version of the interface and device template. Items that you can add to the interface include telemetry, properties, and commands.
- After an interface is published, you can make non-schema changes to existing items in the interface if you create a new version of the interface and device template. Non-schema parts of an interface item include the display name and the semantic type. The schema parts of an interface item that you can't change are name, capability type, and schema.

The following sections walk you through some examples of modifying device templates in IoT Central.

Customize the device template without versioning

Certain elements of your device capabilities can be edited without needing to version your device template and interfaces. For example, some of these fields include display name, semantic type, minimum value, maximum value, decimal places, color, unit, display unit, comment, and description. To add one of these customizations:

1. Go to the **Device Templates** page.
2. Select the device template you wish to customize.
3. Choose the **Customize** tab.
4. All the capabilities defined in your device capability model are listed here. You can edit, save, and use all of these fields without the need to version your device template. If there are fields you wish to edit that are read-only, you must version your device template to change them. Select a field you wish to edit and enter in any new values.
5. Click **Save**. Now these values override anything that was initially saved in your device template and are used across the application.

Version a device template

Creating a new version of your device template creates a draft version of the template where the device capability model can be edited. Any published interfaces remain published until they're individually versioned. To modify a published interface, first create a new device template version.

Only version the device template when you're trying to edit a part of the device capability model that you can't edit in the customizations section.

To version a device template:

1. Go to the **Device Templates** page.
2. Select the device template you're trying to version.
3. Click the **Version** button at the top of the page and give the template a new name. IoT Central suggests a new name, which you can edit.
4. Click **Create**.
5. Now your device template is in draft mode. You can see your interfaces are still locked. Version any interfaces you want to modify.

Version an interface

Versioning an interface allows you to add, update, and remove the capabilities inside the interface you had already created.

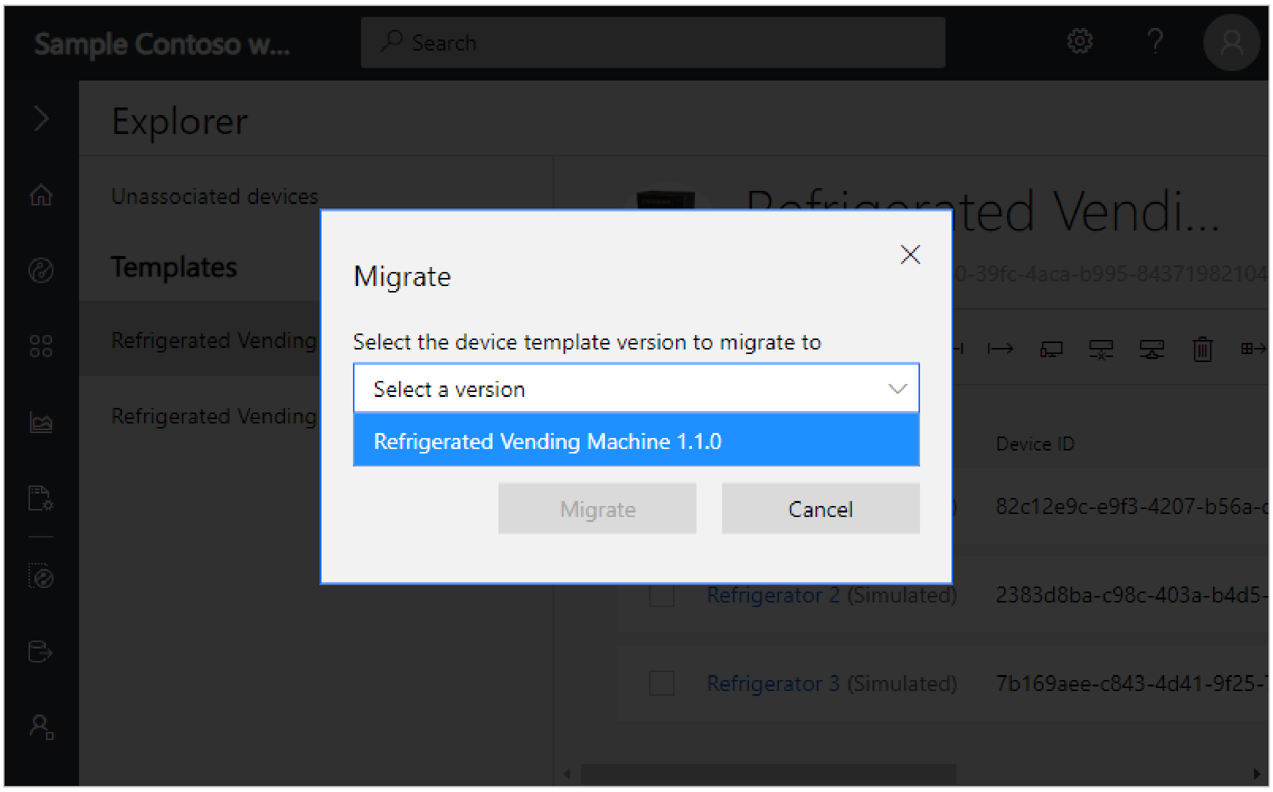
To version an interface:

1. Go to the **Device Templates** page.
2. Select the device template you have in a draft mode.
3. Select the interface that is in published mode that you wish to version and edit.
4. Click the **Version** button at the top of the interface page.
5. Click **Create**.
6. Now your interface is in draft mode. You can add or edit capabilities to your interface without breaking existing customizations and views.

Migrate a device across versions

You can create multiple versions of the device template. Over time, you'll have multiple connected devices using these device templates. You can migrate devices from one version of your device template to another. The following steps describe how to migrate a device:

1. Go to the **Device Explorer** page.
2. Select the device you need to migrate to another version.
3. Choose **Migrate**.
4. Select the device template with the version number you want to migrate the device to and choose **Migrate**.



Next steps

If you're an operator or solution builder, a suggested next step is to learn how to manage your devices.

If you're a device developer, a suggested next step is to read about Azure IoT Edge devices and Azure IoT Central.

Manage devices in your Azure IoT Central application

3/24/2020 • 5 minutes to read • Edit Online

This article describes how, as an operator, to manage devices in your Azure IoT Central application. As an operator, you can:

- Use the **Devices** page to view, add, and delete devices connected to your Azure IoT Central application.
- Maintain an up-to-date inventory of your devices.
- Keep your device metadata up-to-date by changing the values stored in the device properties from your views.
- Control the behavior of your devices by updating a setting on a specific device from your views.

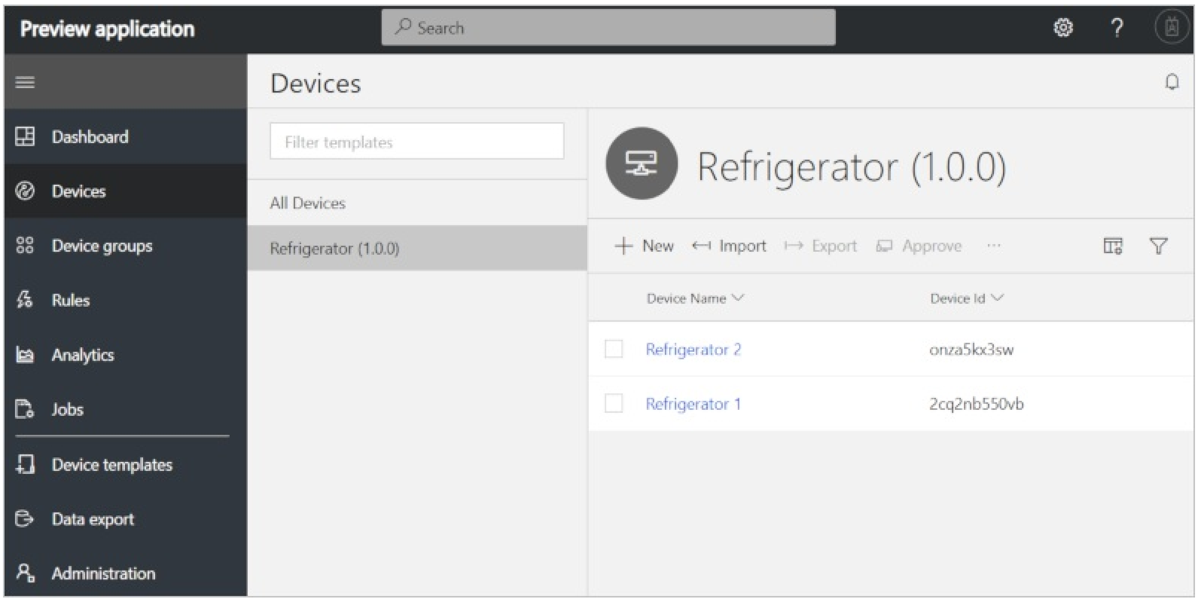
View your devices

To view an individual device:

1. Choose **Devices** on the left pane. Here you see a list of all devices and of your device templates.

2. Choose a device template.

3. In the right-hand pane of the **Devices** page, you see a list of devices created from that device template. Choose an individual device to see the device details page for that device:



Add a device

To add a device to your Azure IoT Central application:

1. Choose **Devices** on the left pane.

2. Choose the device template from which you want to create a device.

3. Choose + **New**.

4. Turn the **Simulated** toggle to **On** or **Off**. A real device is for a physical device that you connect to your Azure IoT Central application. A simulated device has sample data generated for you by Azure IoT Central.

5. Click **Create**.

6. This device now appears in your device list for this template. Select the device to see the device details page that contains all views for the device.

Import devices

To connect large number of devices to your application, you can bulk import devices from a CSV file. The CSV file should have the following columns and headers:

- **IOTC_DeviceID** - the device ID should be all lowercase.
- **IOTC_DeviceName** - this column is optional.

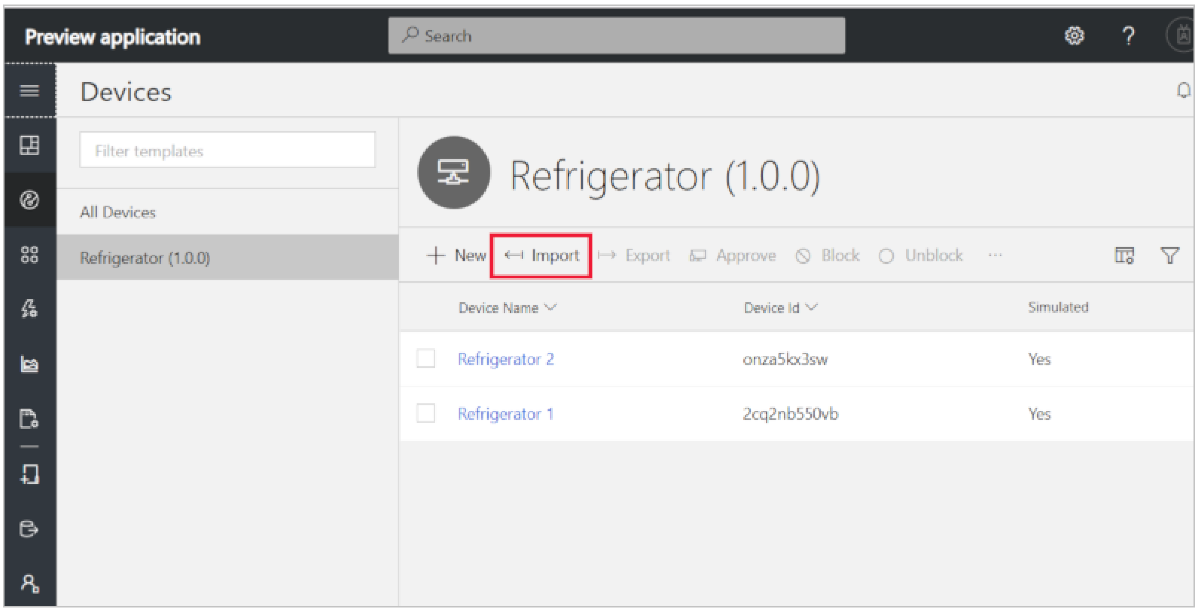
To bulk-register devices in your application:

1. Choose **Devices** on the left pane.

2. On the left panel, choose the device template for which you want to bulk create the devices.

> **NOTE**
>
> If you don't have a device template yet then you can import devices under **All devices** and register them without a template. After devices have been imported, you can then migrate them to a template.

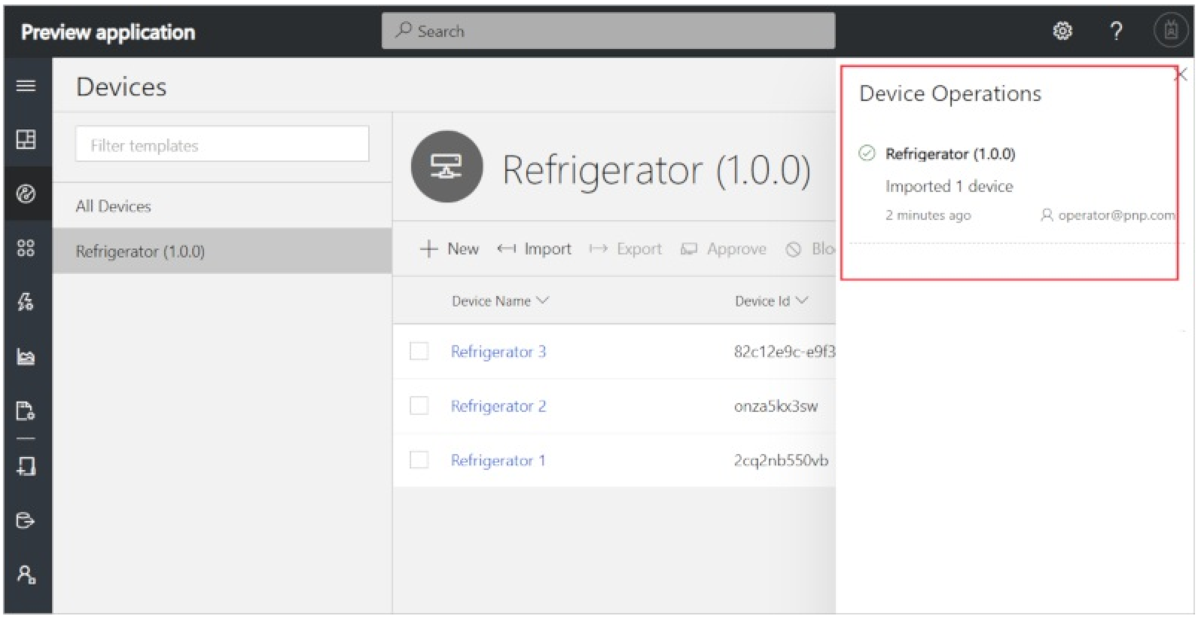
3. Select **Import**.



4. Select the CSV file that has the list of Device IDs to be imported.

5. Device import starts once the file has been uploaded. You can track the import status in the Device Operations panel. This panel appears automatically after the import starts or you can access it through the bell icon in the top right-hand corner.

6. Once the import completes, a success message is shown in the Device Operations panel.

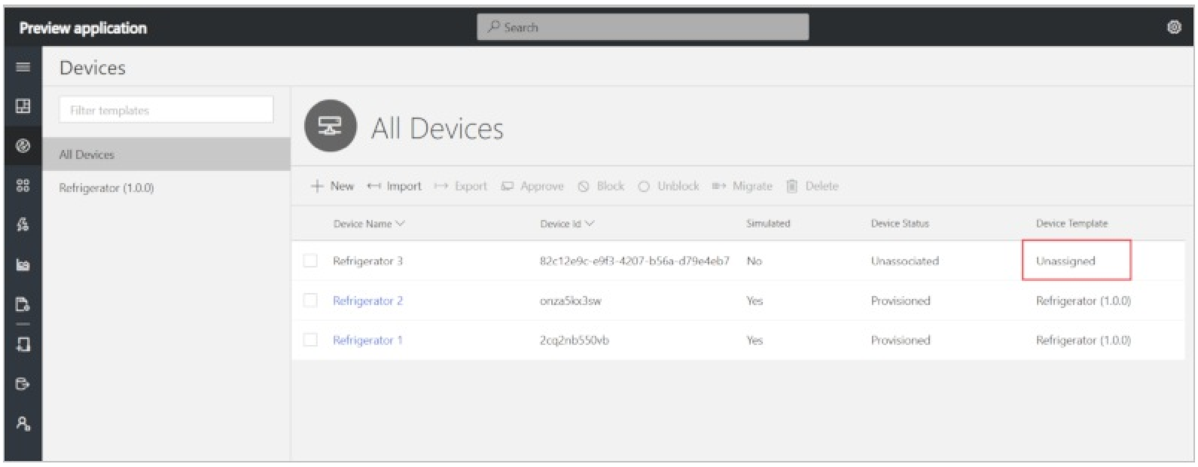


If the device import operation fails, you see an error message on the Device Operations panel. A log file capturing all the errors is generated that you can download.

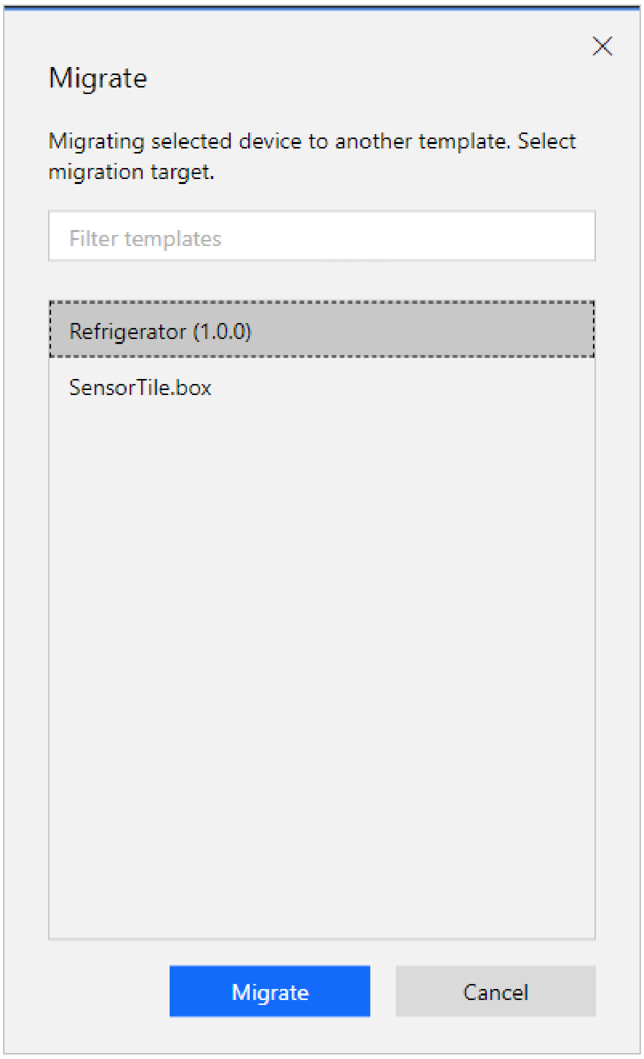
Migrating devices to a template

If you register devices by starting the import under **All devices**, then the devices are created without any device template association. Devices must be associated with a template to explore the data and other details about the device. Follow these steps to associate devices with a template:

1. Choose **Devices** on the left pane.
2. On the left panel, choose **All devices**:



3. Use the filter on the grid to determine if the value in the **Device Template** column is "Unassociated" for any of your devices.
4. Select the devices you want to associate with a template:
5. Select **Migrate**:



6. Choose the template from the list of available templates and select **Migrate**.

7. The selected devices are associated with the device template you chose.

Export devices

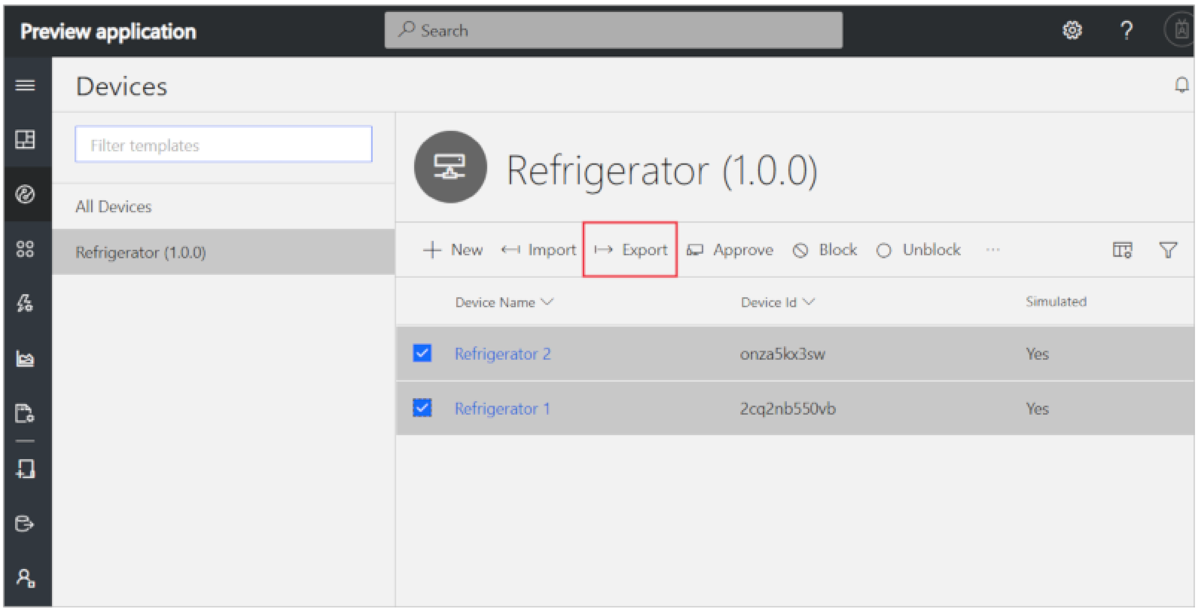
To connect a real device to IoT Central, you need its connection string. You can export device details in bulk to get the information you need to create device connection strings. The export process creates a CSV file with the device identity, device name, and keys for all the selected devices.

To bulk export devices from your application:

1. Choose **Devices** on the left pane.

2. On the left pane, choose the device template from which you want to export the devices.

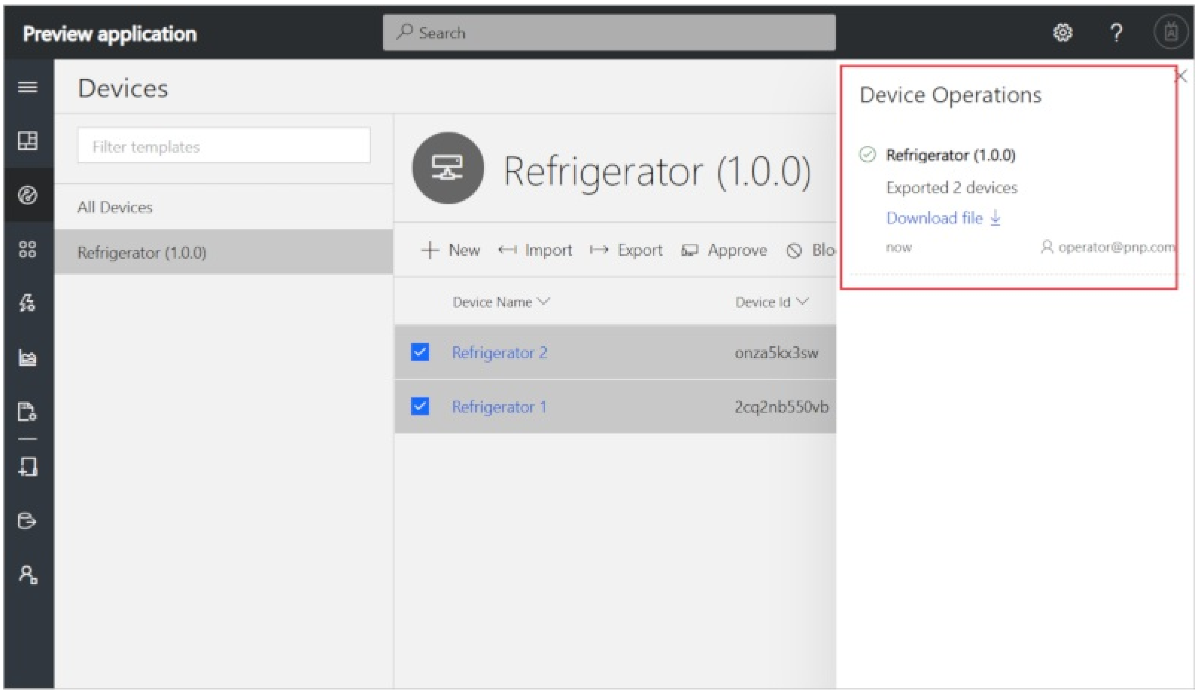
3. Select the devices that you want to export and then select the **Export** action.



4. The export process starts. You can track the status using the Device Operations panel.

5. When the export completes, a success message is shown along with a link to download the generated file.

6. Select the **Download File** link to download the file to a local folder on the disk.



7. The exported CSV file contains the following columns: device ID, device name, device keys, and X509 certificate thumbprints:

   - IOTC_DEVICEID
   - IOTC_DEVICENAME
   - IOTC_SASKEY_PRIMARY
   - IOTC_SASKEY_SECONDARY
   - IOTC_X509THUMBPRINT_PRIMARY
   - IOTC_X509THUMBPRINT_SECONDARY

For more information about connection strings and connecting real devices to your IoT Central application, see Device connectivity in Azure IoT Central.

Delete a device

To delete either a real or simulated device from your Azure IoT Central application:

1. Choose **Devices** on the left pane.

2. Choose the device template of the device you want to delete.

3. Use the filter tools to filter and search for your devices. Check the box next to the devices to delete.

4. Choose **Delete**. You can track the status of this deletion in your Device Operations panel.

Change a property

Cloud properties are the device metadata associated with the device, such as city and serial number. Writeable properties control the behavior of a device. In other words, they enable you to provide inputs to your device. Device properties are set by the device and are read-only within IoT Central. You can view and update properties on the **Device Details** views for your device.

1. Choose **Devices** on the left pane.

2. Choose the device template of the device whose properties you want to change and select the target device.

3. Choose the view that contains properties for your device, this view enables you to input values and select **Save** at the top of the page. Here you see the properties your device has and their current values. Cloud properties and writeable properties have editable fields, while device properties are read-only. For writeable properties, you can see their sync status at the bottom of the field.

4. Modify the properties to the values you need. You can modify multiple properties at a time and update them all at the same time.

5. Choose **Save**. If you saved writeable properties, the values are sent to your device. When the device confirms the change for the writeable property, the status returns back to **synced**. If you saved a cloud property, the value is updated.

Next steps

Now that you've learned how to manage devices in your Azure IoT Central application, here is the suggested next step:

How to use device groups

Configure rules

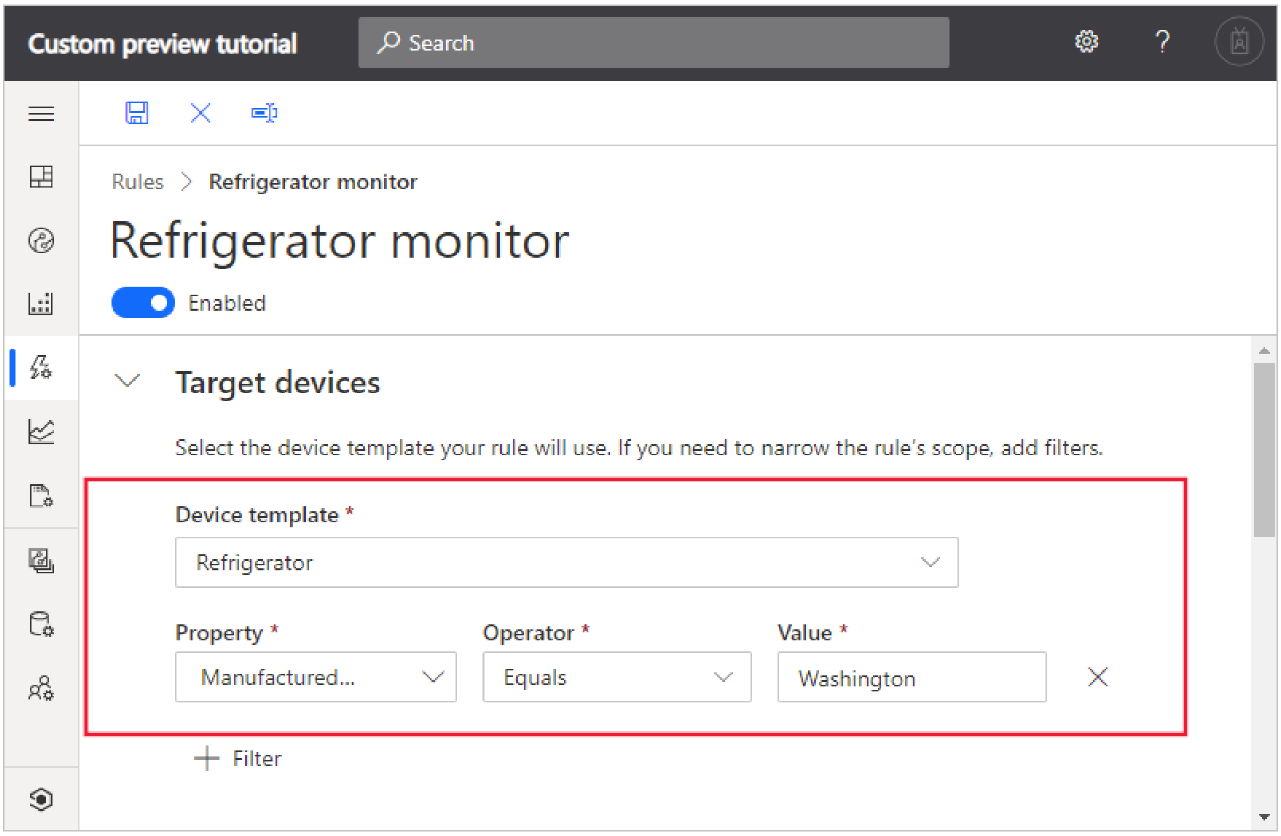
5/19/2020 • 2 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

Rules in IoT Central serve as a customizable response tool that trigger on actively monitored events from connected devices. The following sections describe how rules are evaluated.

Select target devices

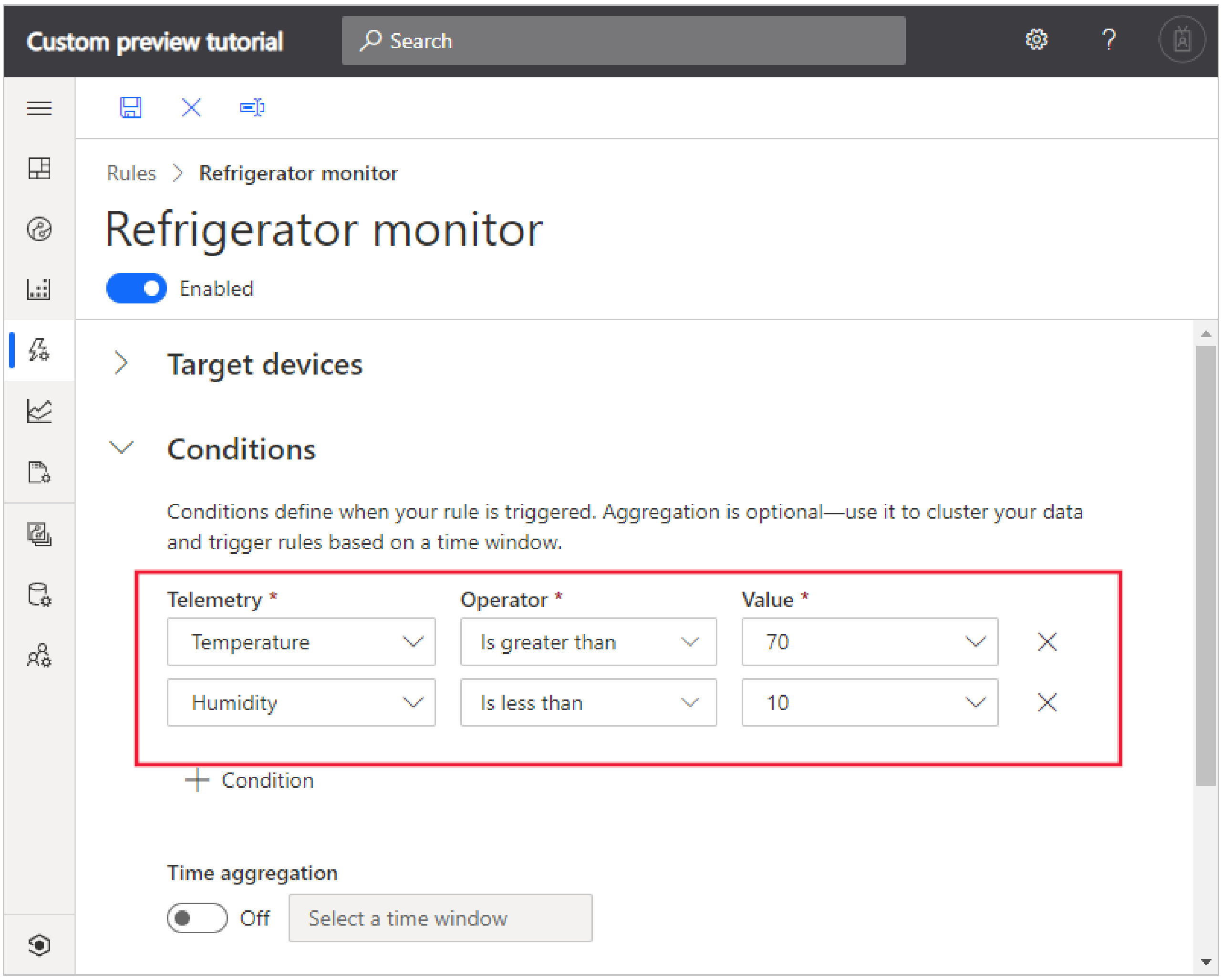
Use the target devices section to select on what kind of devices this rule will be applied. Filters allow you to further refine what devices should be included. The filters use properties on the device template to filter down the set of devices. Filters themselves don't trigger an action. In the following screenshot, the devices that are being targeted are of device template type **Refrigerator**. The filter states that the rule should only include **Refrigerators** where the **Manufactured State** property equals **Washington**.



Use multiple conditions

Conditions are what rules trigger on. Currently, when you add multiple conditions to a rule, they're logically AND'd together. In other words, all conditions must be met for the rule to evaluate as true.

In the following screenshot, the conditions check when the temperature is greater than 70° F and the humidity is less than 10. When both of these statements are true, the rule evaluates to true and triggers an action.



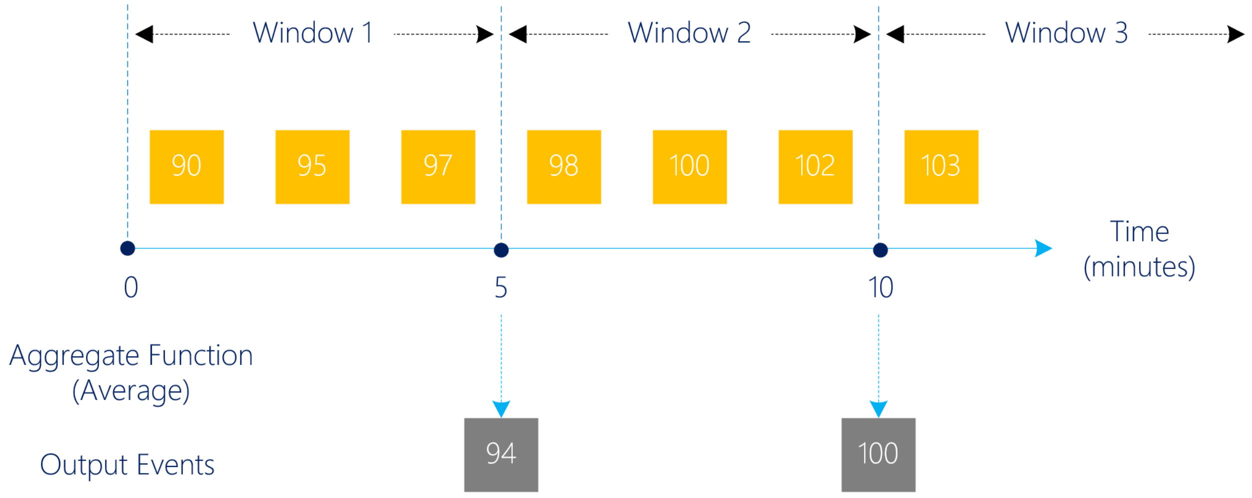
Use a cloud property in a value field

You can reference a cloud property from the device template in the **Value** field for a condition. The cloud property and telemetry value must have similar types. For example, if **Temperature** is a double, then only cloud properties of type double show as options in the **Value** drop-down.

If you choose an event type telemetry value, the **Value** drop-down includes the option **Any**. The **Any** option means the rule fires when your application receives an event of that type, whatever the payload.

Use aggregate windowing

Rules evaluate aggregate time windows as tumbling windows. In the screenshot below, the time window is five minutes. Every five minutes, the rule evaluates on the last five minutes of data. The data is only evaluated once in the window to which it corresponds.



Use rules with IoT Edge modules

A restriction applies to rules that are applied to IoT Edge modules. Rules on telemetry from different modules aren't evaluated as valid rules. Take the following as an example. The first condition of the rule is on a temperature telemetry from Module A. The second condition of the rule is on a humidity telemetry on Module B. Since the two conditions are from different modules, this is an invalid set of conditions. The rule isn't valid and will throw an error on trying to save the rule.

Next steps

Now that you've learned how to configure a rule in your Azure IoT Central application, you can learn how to Configure advanced rules using Power Automate or Azure Logic Apps.

How to use analytics to analyze device data

7/21/2020 • 4 minutes to read • Edit Online

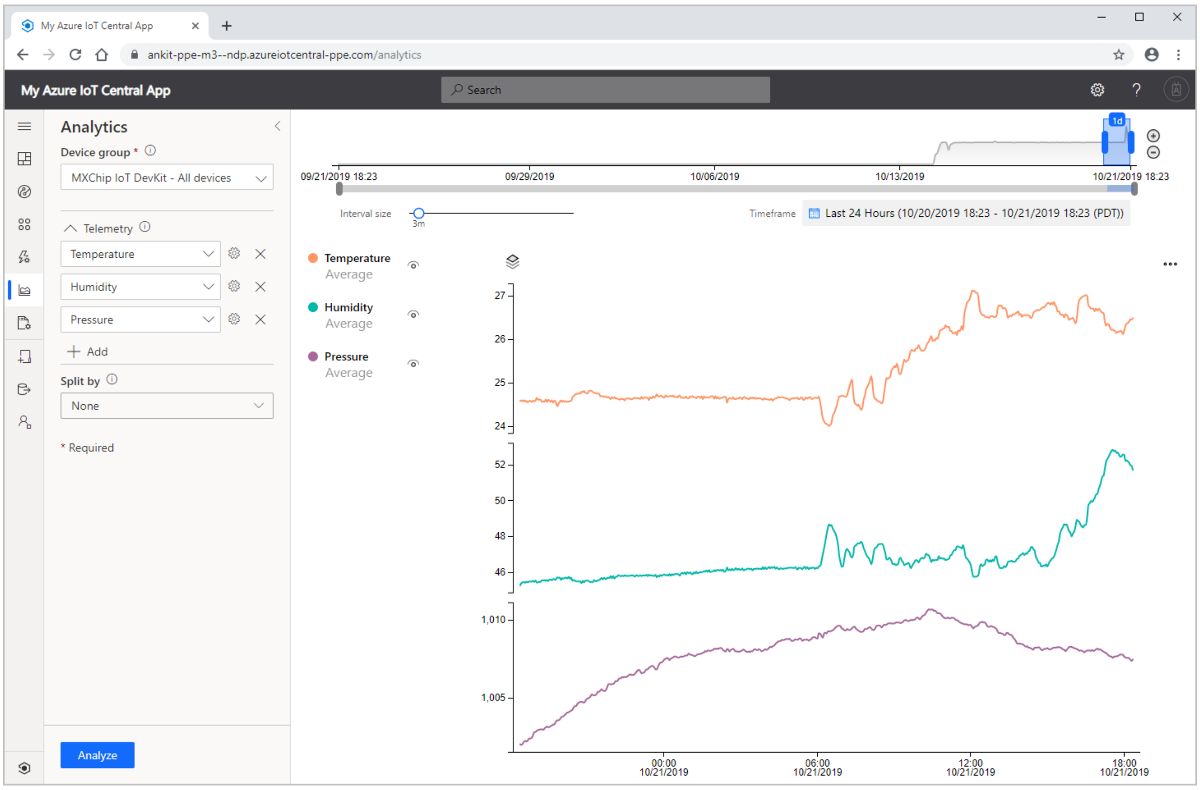
This article applies to operators, builders, and administrators.

Azure IoT Central provides rich analytics capabilities to analyze historical trends and correlate various telemetries from your devices. To get started, visit **Analytics** on the left pane.

Understanding the Analytics UI

Analytics user interface is made of three main components:

- **Data configuration panel:** On the configuration panel, start by selecting the device group for which you want to analyze the data. Next, select the telemetry that you want to analyze and select the aggregation method for each telemetry. **Split By** control helps to group the data by using the device properties as dimensions.
- **Time control:** Time control is used to select the duration for which you want to analyze the data. You can drag either end of the time slider to select the time span. Time control also has an **Interval size** slider that controls the bucket or the interval size used to aggregate the data.
- **Chart control:** Chart control visualizes the data as a line chart. You can toggle the visibility of specific lines by interacting with the chart legend.



Querying your data

You'll need to start by choosing a **device group**, and the telemetry that you want to analyze. Once you're done, select **Analyze** to start visualizing your data.

- **Device group:** A device group is a user-defined group of your devices. For example, all Refrigerators in Oakland, or All version 2.0 wind turbines.

- **Telemetry:** Select the telemetry that you want to analyze and explore. You can select multiple telemetries to analyze together. Default aggregation method is set to Average for numerical and Count for string data-type respectively. Supported aggregation methods for Numeric data types are Average, Maximum, Minimum, Count and, Sum. Supported aggregation methods for string data type are count.

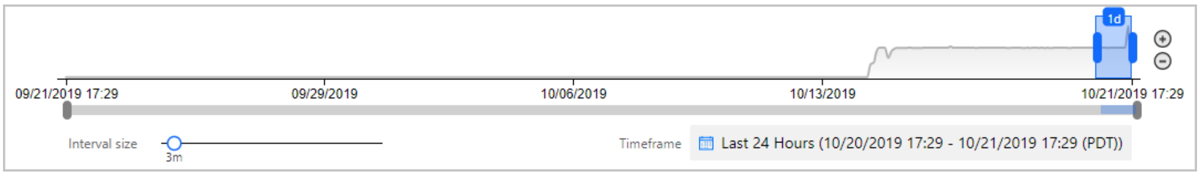
- **Split by:** 'Split by' control helps to group the data by using the device properties as dimensions. Values of the device and cloud properties are joined along with the telemetry as and when it is sent by the device. If the cloud or device property has been updated, then you will see the telemetry grouped by different values on the chart.

  > **TIP**
  >
  > To view data for each device separately, select Device Id in the 'Split by' control.

Interacting with your data

Once you've queried your data, you can start visualizing it on the line chart. You can show/hide telemetry, change the time duration, view telemetry in a data grid.

- **Time editor panel:** By default we'll retrieve data from the past one day. You can drag either end of the time slider to change the time duration. You can also use the calendar control to select one of the predefined time buckets or select a custom time range. Time control also has an **Interval size** slider that controls the bucket or the interval size used to aggregate the data.

  - **Inner date range slider tool**: Use the two endpoint controls by dragging them over the time span you want. This inner date range is constrained by the outer date range slider control.

  - **Outer date range slider control**: Use the endpoint controls to select the outer date range, which will be available for your inner date range control.

  - **Increase and decrease date range buttons**: Increase or decrease your time span by selecting either button for the interval you want.

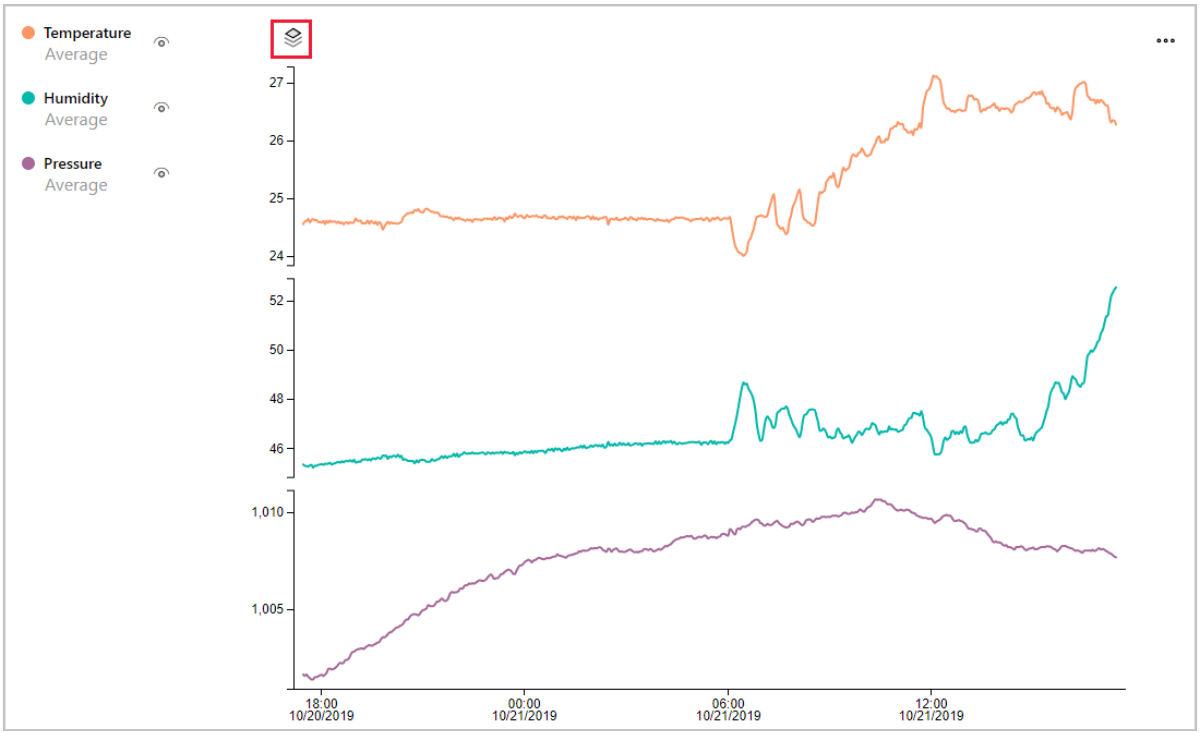
  - **Interval-size slider**: Use it to zoom in and out of intervals over the same time span. This action provides more precise control of movement between large slices of time. You can use it to see granular, high-resolution views of your data, even down to milliseconds. The slider's default starting point is set as the most optimal view of the data from your selection, which balances resolution, query speed, and granularity.

  - **Date range picker**: With this web control, you can easily select the date and time ranges you want. You can also use the control to switch between different time zones. After you make the changes to apply to your current workspace, select Save.

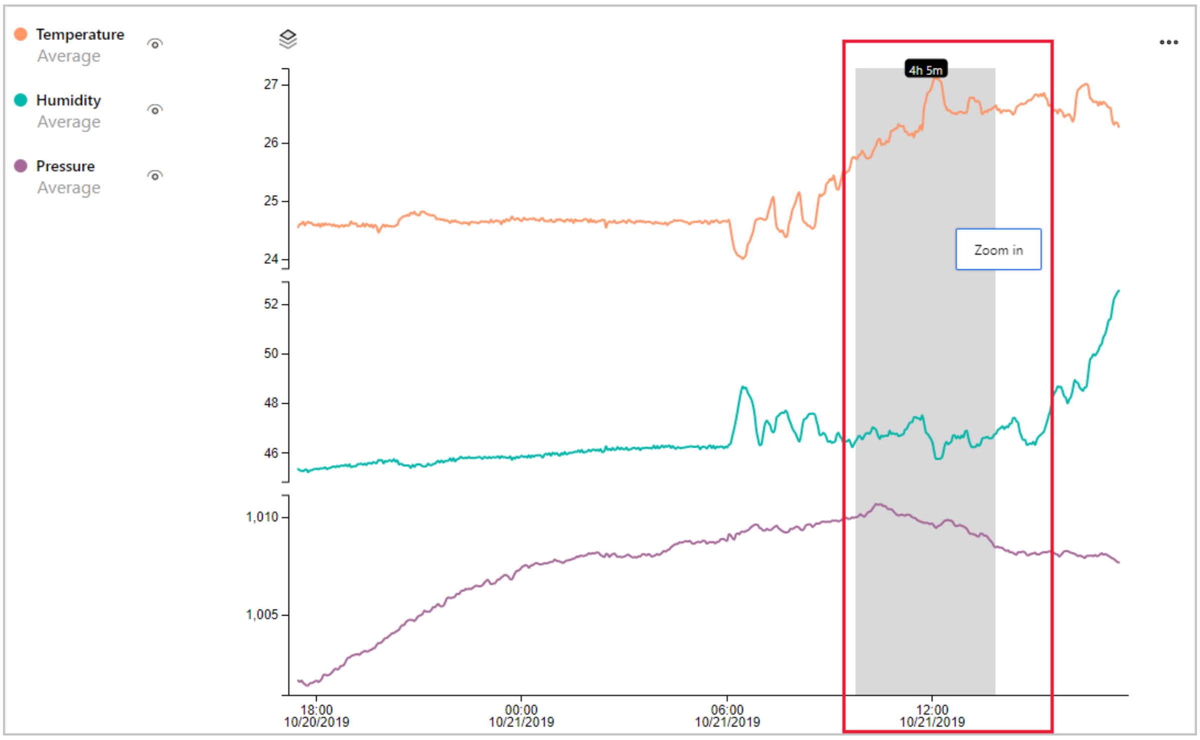
  > **TIP**
  >
  > Interval size is determined dynamically based on the selected time span. Smaller time spans will enable aggregating the data into very granular intervals of up to a few seconds.

- **Chart Legend:** Chart legend shows the selected telemetry on the chart. You can hover over each item on the legend to bring it into focus on the chart. When using 'Split By', the telemetry is grouped by the respective values of the selected dimension. You can toggle the visibility of each specific telemetry or the whole group by clicking on the group name.

- **Y-axis format control:** y-axis mode cycles through the available y-axis view options. This control is available only when different telemetries are being visualized. You can set the y-axis by choosing from one of three modes:
  - **Stacked:** A graph for every telemetry is stacked and each of the graphs have their own y-axis. This mode is set as default.
  - **Shared:** A graph for every telemetry is plotted against the same y-axis.
  - **Overlap:** Use it to stack multiple lines on the same y-axis, with the y-axis data changing based on the selected line.

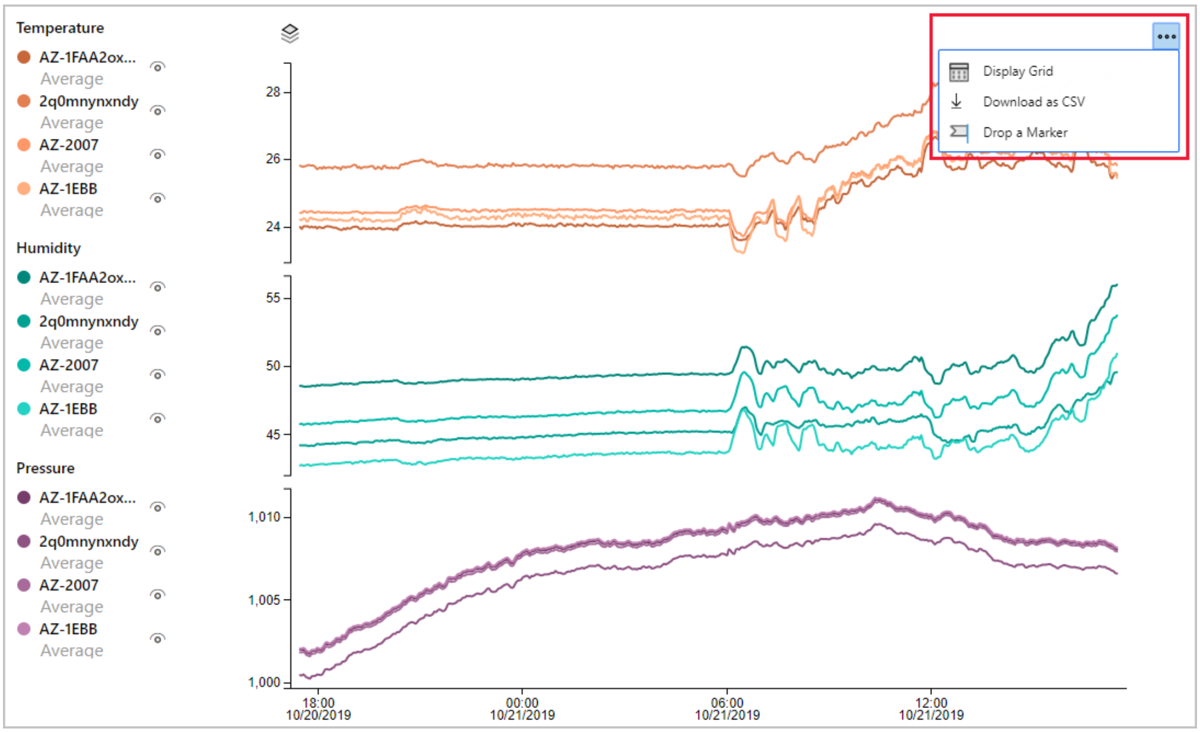


- **Zoom control:** Zoom lets you drill further into your data. If you find a time period you'd like to focus on within your result set, use your mouse pointer to grab the area and then drag it to the endpoint of your choice. Then right click on the selected area and click Zoom.



Under the ellipsis, there are more chart controls to interact with the data:

- **Display Grid**: Your results are available in a table format, enabling you to view the specific value for each data point.
- **Download as CSV**: Your results are available to export as a comma-separated values (CSV) file. The CSV file contains data for each device. Results are exported by using the interval and timeframe specified.
- **Drop a Marker**: The 'Drop Marker' control provides a way to anchor certain data points on the chart. It is useful when you are trying to compare data for multiple lines across different time periods.



Configure the application dashboard

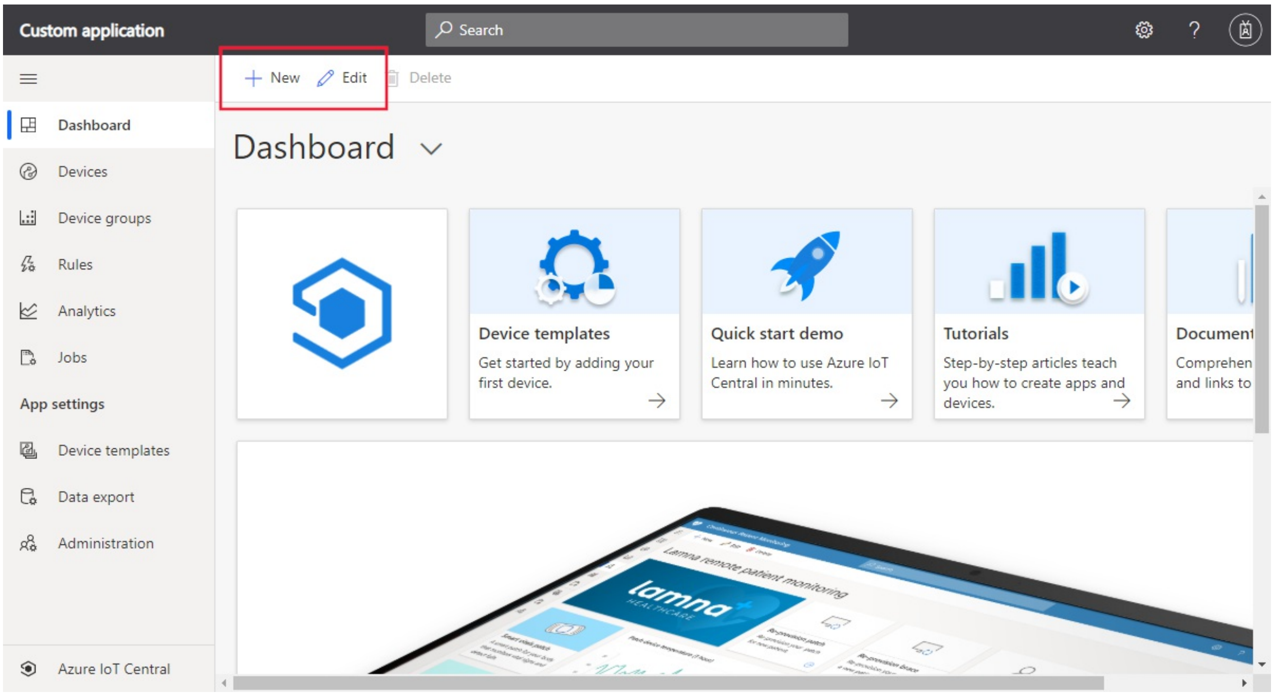
7/21/2020 • 4 minutes to read • Edit Online

The **Dashboard** is the first page you see when you connect to an IoT Central application. If you create your application from one of the industry-focused application templates, your application has a pre-defined dashboard to start. If you create your application from a custom application template, your dashboard shows some tips to get started.

> **TIP**
> Users can create multiple dashboards in addition to the default application dashboard. These dashboards can be personal to the user only, or shared across all users of the application.

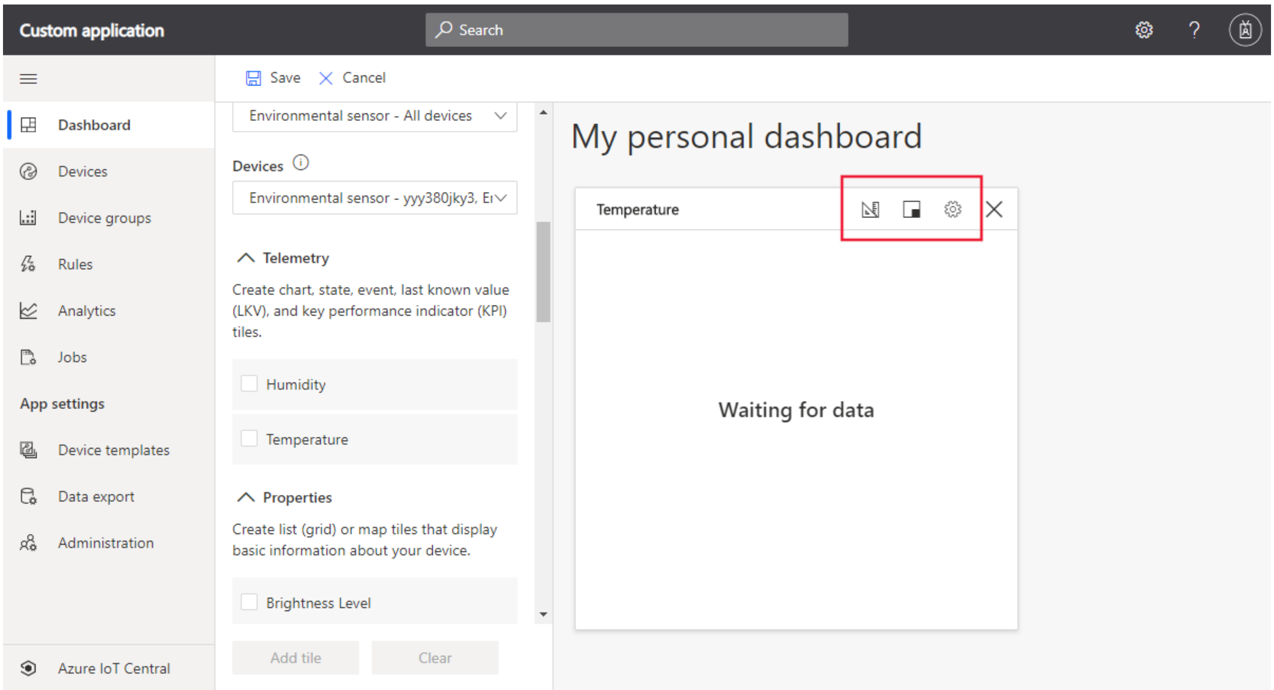
Add tiles

The following screenshot shows the dashboard in an application created from the **Custom application** template. To customize the current dashboard, select **Edit**, to add a custom personal or shared dashboard, select **New**:



After you select **Edit** or **New**, the dashboard is in *edit* mode. You can use the tools in the **Edit dashboard** panel to add tiles to the dashboard, and customize and remove tiles on the dashboard itself. For example, to add a **Telemetry** tile to show current temperature reported by one or more devices:

1. In the **Edit dashboard** panel, select a **Device group**.
2. Select one or more devices in the **Devices** dropdown to show on the tile. You now see the available telemetry, properties, and commands from the devices.
3. Select **Temperature** in the telemetry section, and then select **Add tile**. The tile now shows on the dashboard where you can change the visualization, resize the tile, and configure it:



When you've finished adding and customizing tiles on the dashboard, select **Save**.

Customize tiles

To customize a tile on the dashboard, the dashboard must be in edit mode. The available customization options depend on the tile type:

- The ruler icon on a tile lets you change the visualization. Visualizations include line charts, last known values, and heat maps.
- The square icon lets you resize the tile.
- The gear icon lets you configure the visualization. For example, for a line chart visualization you can choose to show the legend and axes, and choose the time range to plot.

Tile types

The following table describes the different types of tile you can add to a dashboard:

TILE	DESCRIPTION
Markdown	Markdown tiles are clickable tiles that display a heading and description text formatted using markdown. The URL can be a relative link to another page in the application, or an absolute link to an external site.
Image	Image tiles display a custom image and can be clickable. The URL can be a relative link to another page in the application, or an absolute link to an external site.
Label	Label tiles display custom text on a dashboard. You can choose the size of the text. Use a label tile to add relevant information to the dashboard such descriptions, contact details, or help.
Count	Count tiles display the number of devices in a device group.

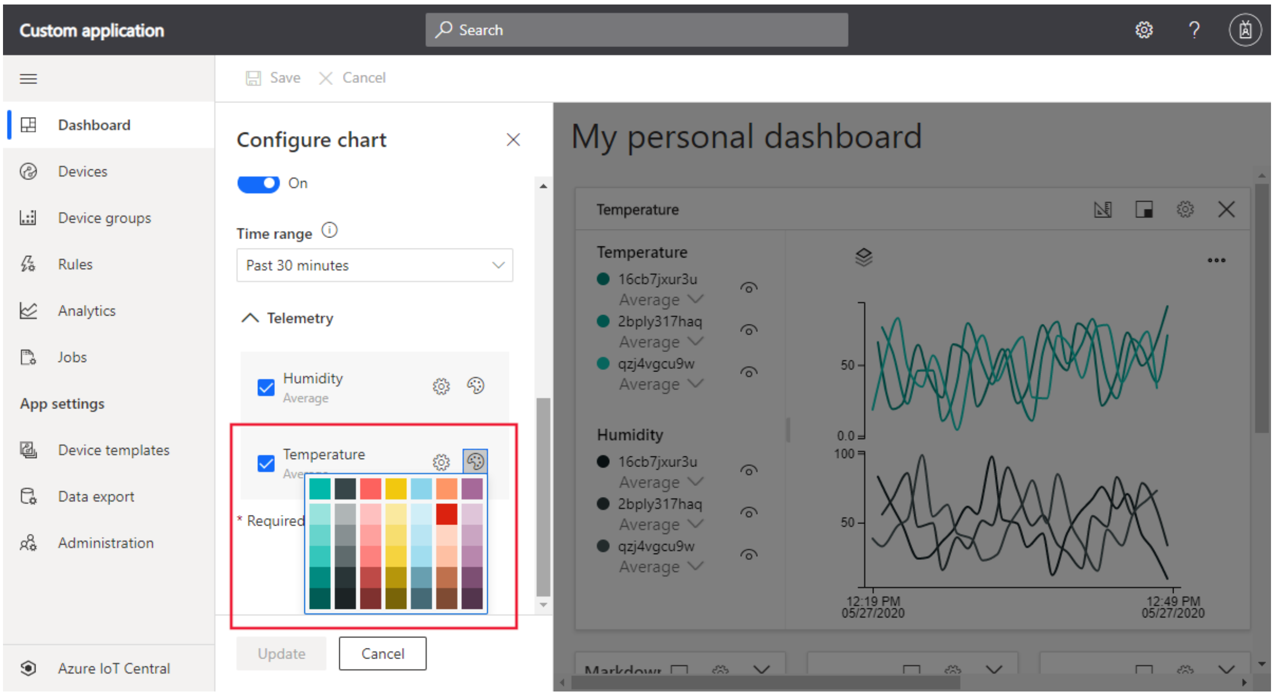
TILE	DESCRIPTION
Map	Map tiles display the location of one or more devices on a map. You can also display up to 100 points of a device's location history. For example, you can display sampled route of where a device has been on the past week.
KPI	KPI tiles display aggregate telemetry values for one or more devices over a time period. For example, you can use it to show the maximum temperature and pressure reached for one or more devices during the last hour.
Line chart	Line chart tiles plot one or more aggregate telemetry values for one or more devices for a time period. For example, you can display a line chart to plot the average temperature and pressure of one or more devices for the last hour.
Bar chart	Bar chart tiles plot one or more aggregate telemetry values for one or more devices for a time period. For example, you can display a bar chart to show the average temperature and pressure of one or more devices over the last hour.
Pie chart	Pie chart tiles display one or more aggregate telemetry values for one or more devices for a time period.
Heat map	Heat map tiles display information about one or more devices, represented as colors.
Last Known Value	Last known value tiles display the latest telemetry values for one or more devices. For example, you can use this tile to display the most recent temperature, pressure, and humidity values for one or more devices.
Event History	Event History tiles display the events for a device over a time period. For example, you can use it to show all the valve open and close events for one or more devices during the last hour.
Property	Property tiles display the current value for properties and cloud properties of one or more devices. For example, you can use this tile to display device properties such as the manufacturer or firmware version for a device.

Currently, you can add up to 10 devices to tiles that support multiple devices.

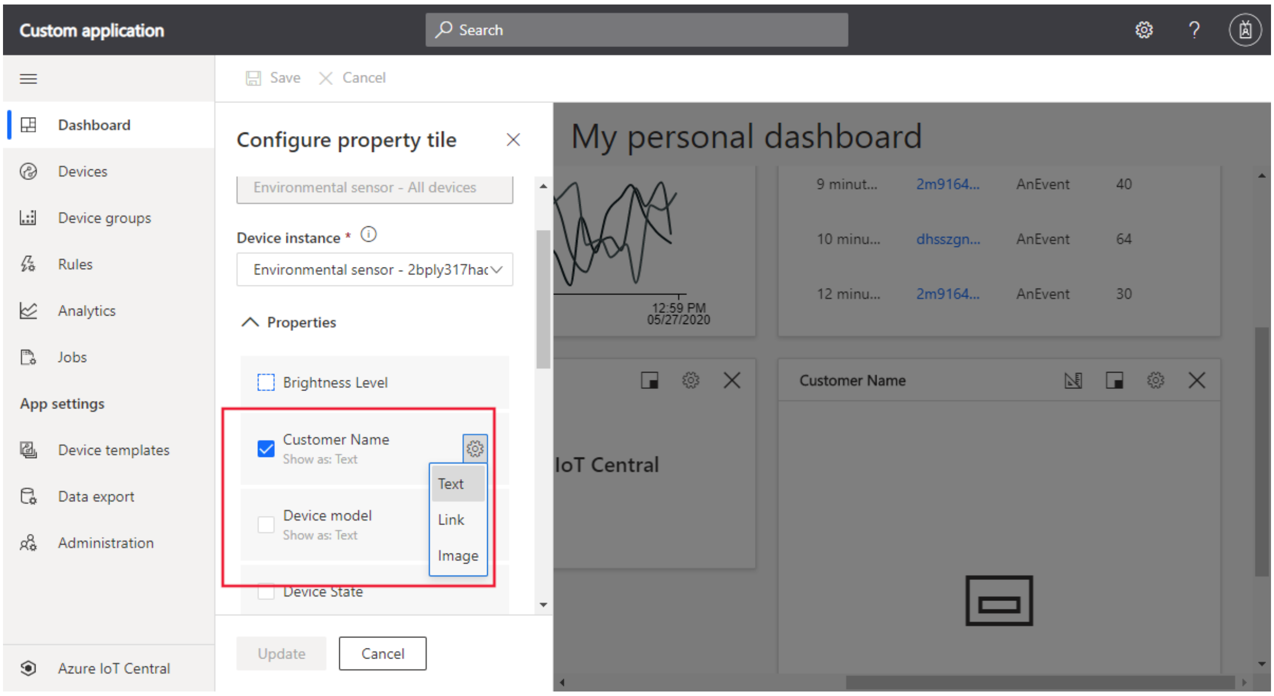
Customizing visualizations

For tiles that display aggregate values, select the gear icon next to the telemetry type in the **Configure chart** panel to choose the aggregation. You can choose from average, sum, maximum, minimum, and count.

For line charts, bar charts, and pie charts, you can customize the color of the different telemetry values. Select the palette icon next to the telemetry you want to customize:



For tiles that show string properties or telemetry values, you can choose how to display the text. For example, if the device stores a URL in a string property, you can display it as a clickable link. If the URL references an image, you can render the image in a last known value or property tile. To change how a string displays, in the tile configuration select the gear icon next to the telemetry type or property:



Next steps

Now that you've learned how to configure your Azure IoT Central default application dashboard, you can Learn how to create a personal dashboard.

Create and manage multiple dashboards

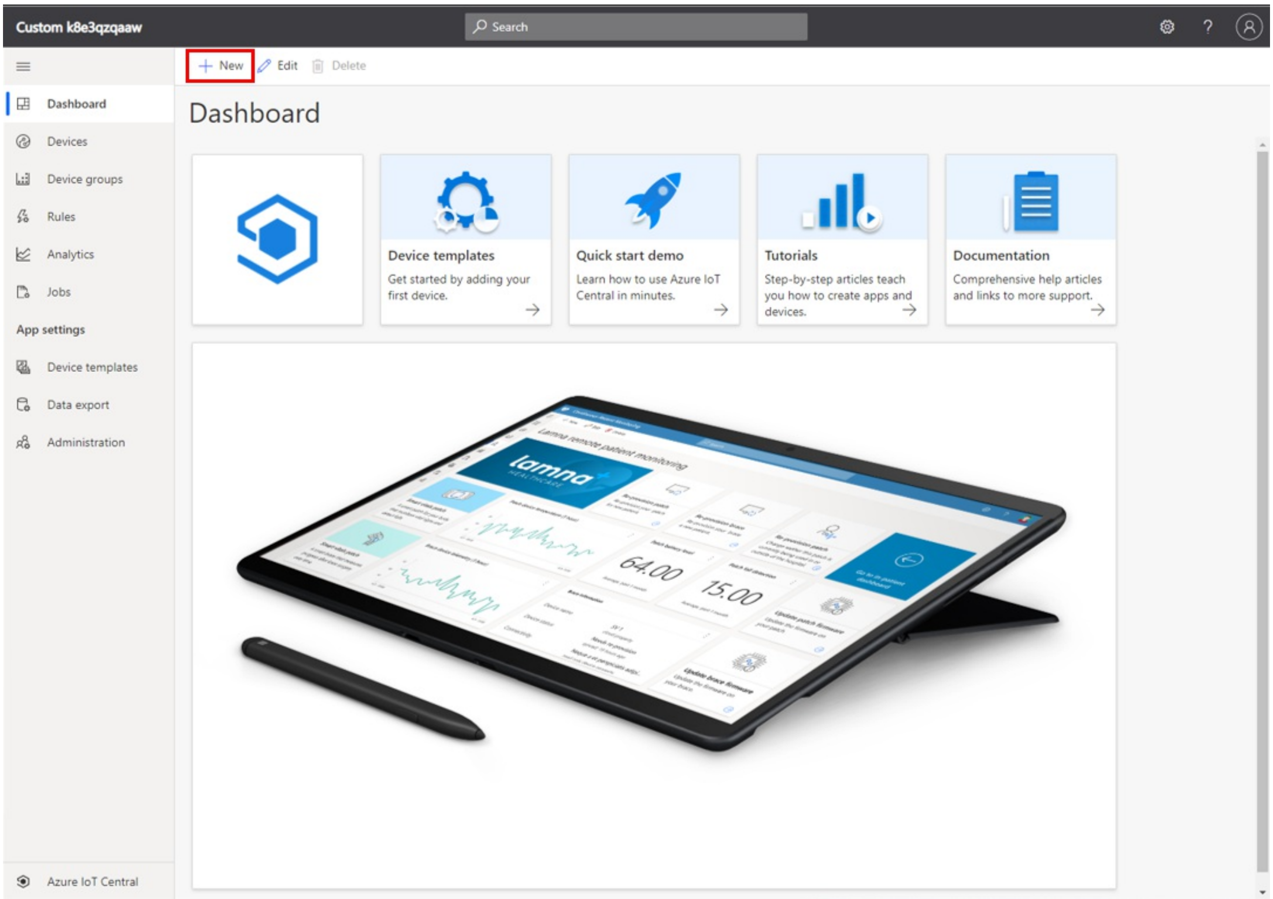
5/19/2020 • 2 minutes to read • Edit Online

The **Dashboard** is the page that loads when you first navigate to your application. An **builder** in your application defines the default application dashboard for all users. You can additionally create your own, personalized application dashboard. You can have several dashboards that display different data and switch between them.

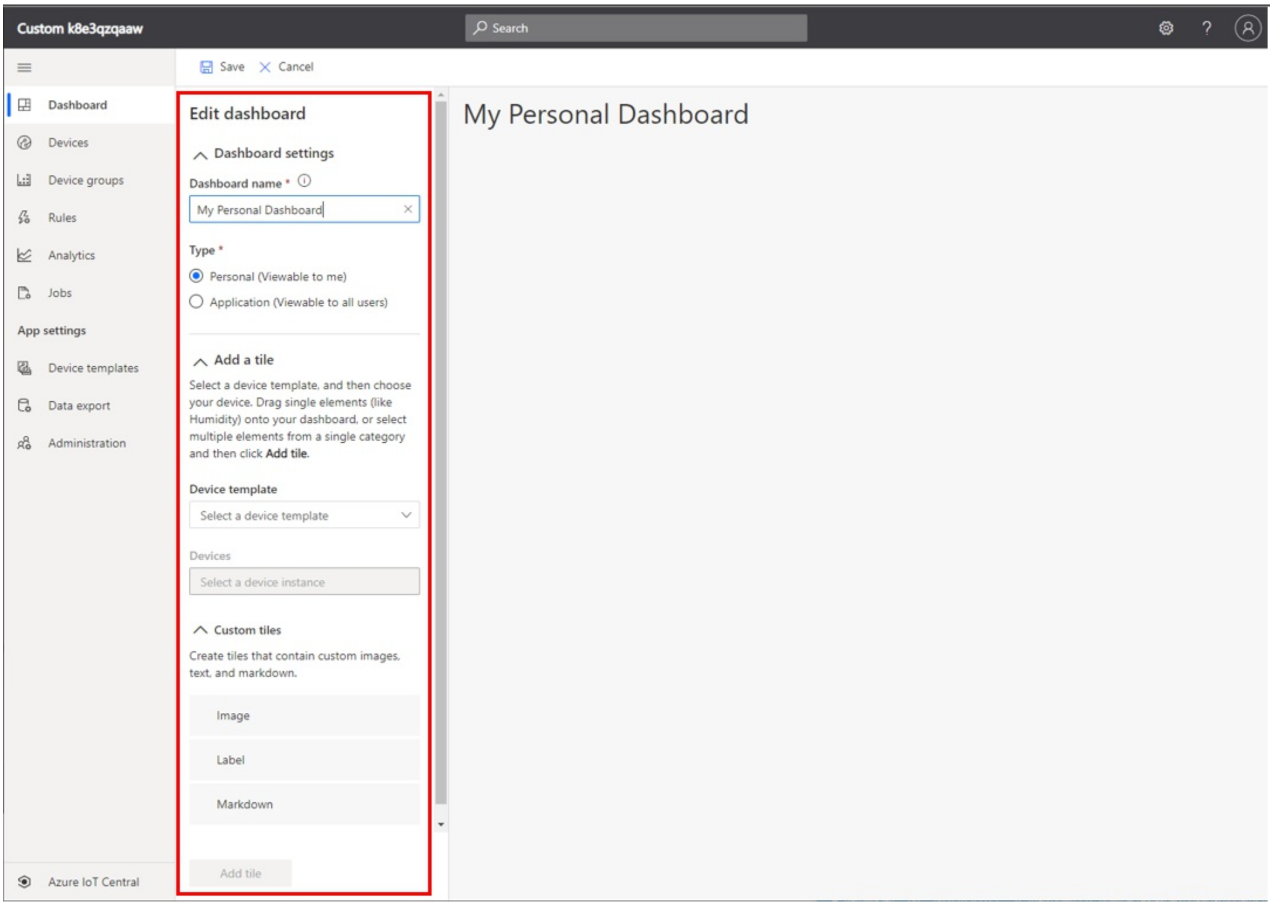
If you are an **admin** of the application, you also can create up to 10 application level dashboards to share with other users of the application. Only **admins** have the ability to create, edit, and delete application level dashboards.

Create dashboard

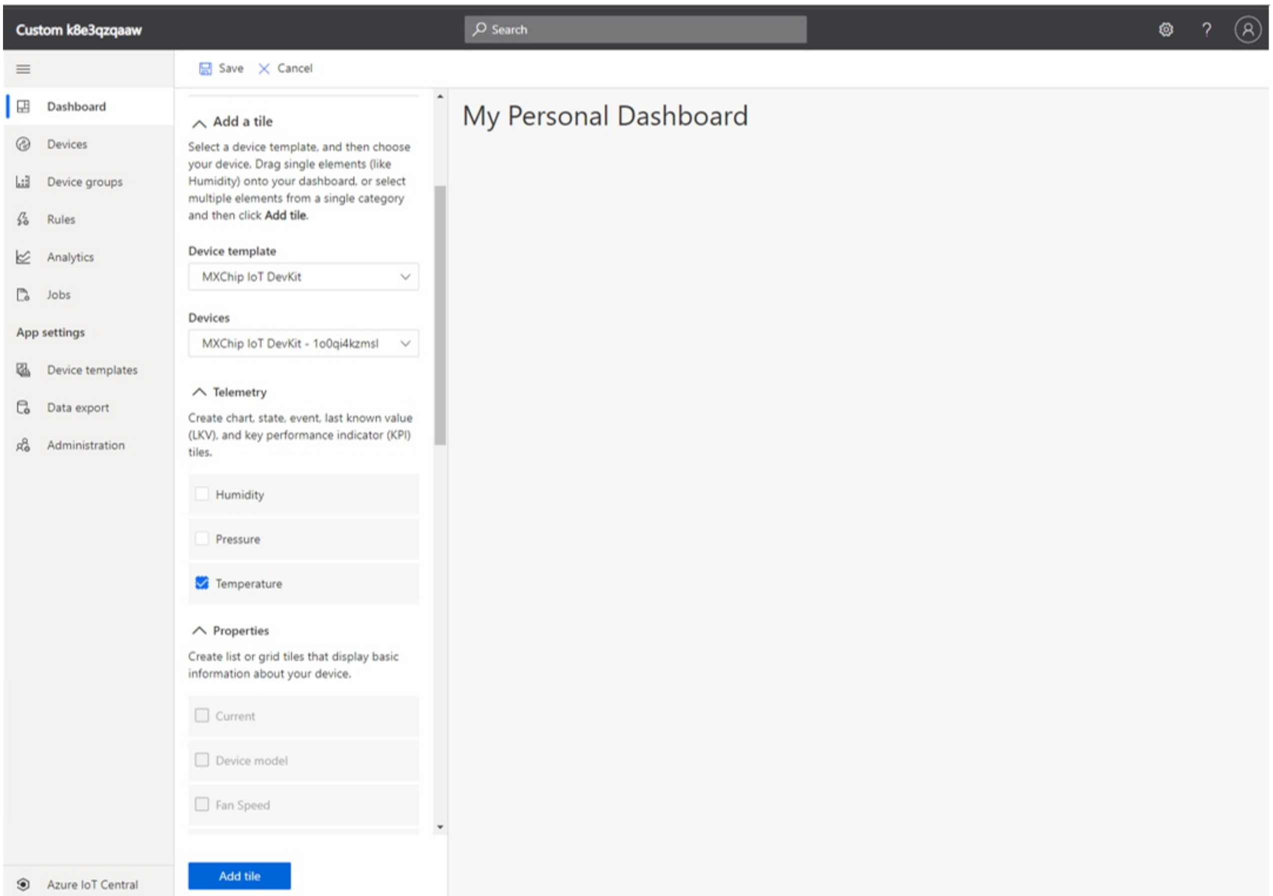
The following screenshot shows the dashboard in an application created from the **Custom Application** template. You can replace the default application dashboard with a personal dashboard, or if you are an admin, another application level dashboard. To do so, select **+ New** at the top left of the page.



Selecting **+ New** opens the dashboard editor. In the editor, you can give your dashboard a name and chose items from the library. The library contains the tiles and dashboard primitives you can use to customize the dashboard.

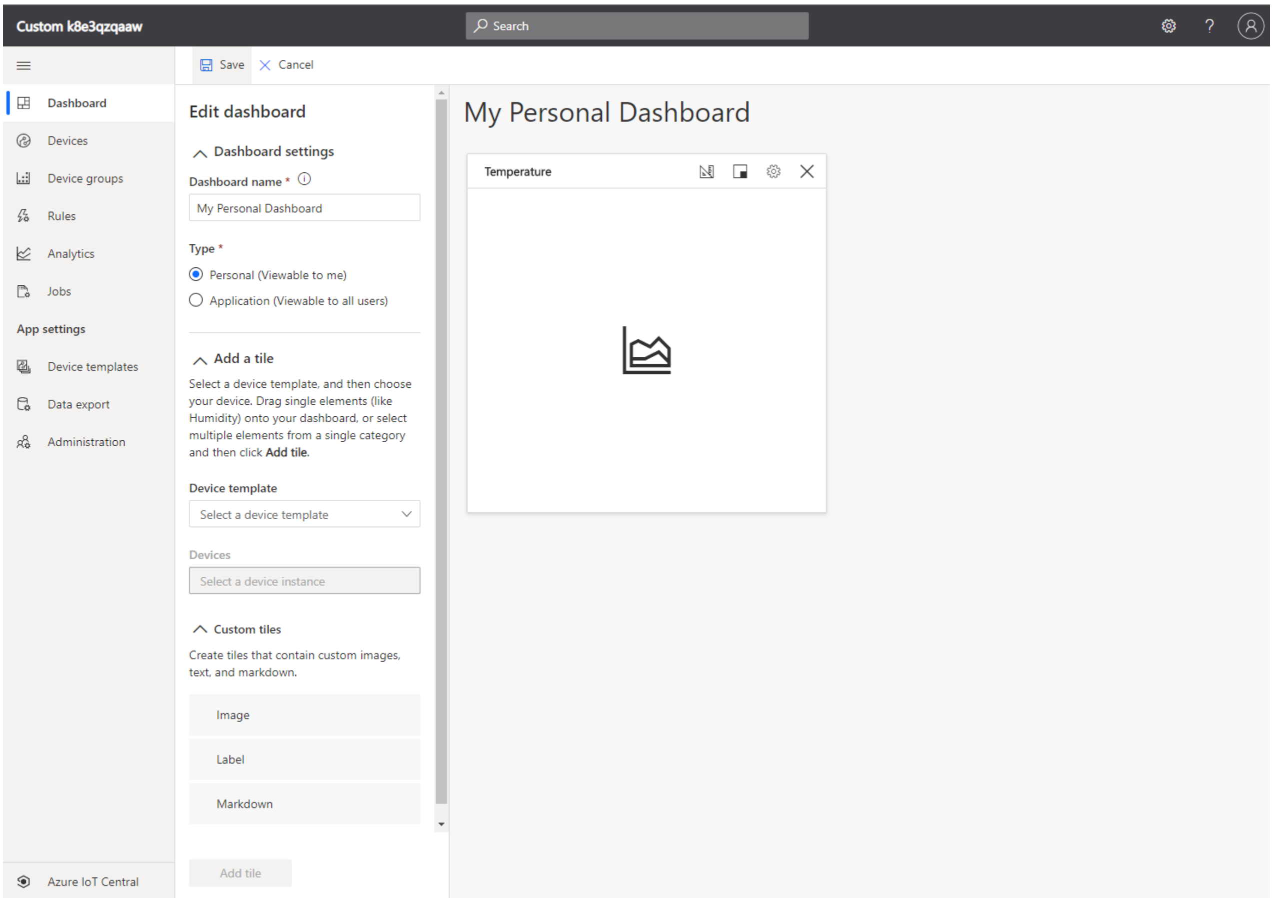


If you are an **admin** of the application, you will be given the option to create a personal level dashboard or an application level dashboard. If you create a personal level dashboard, only you will be able to see it. If you create an application level dashboard, every user of the application will be able to see it. After entering a title and selecting the type of dashboard you want to create, you can save and add tiles later. Or if you are ready now and have added a device template and device instance, you can go ahead and create your first tile.

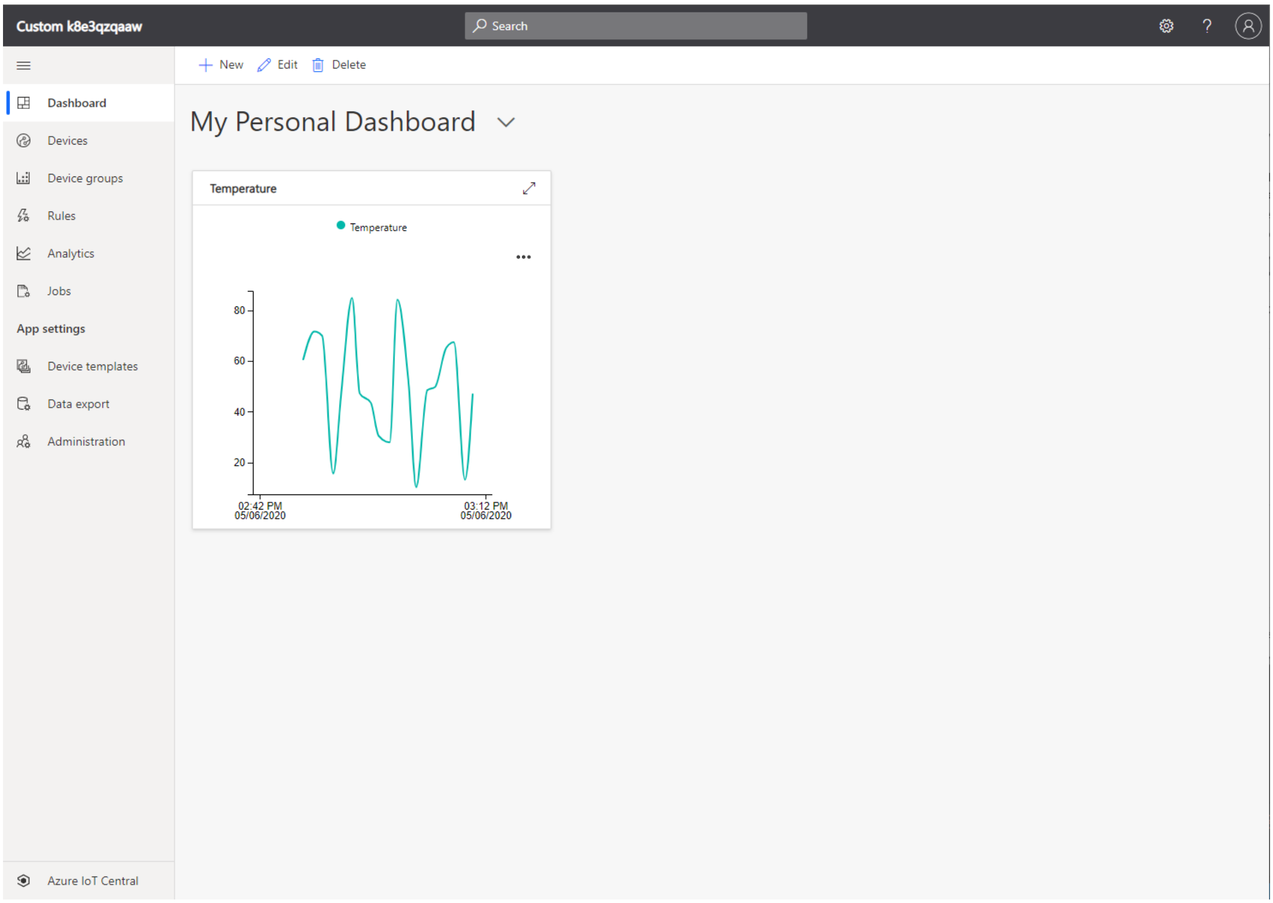


For example, you can add a **Telemetry** tile for the current temperature of the device. To do so:

1. Select a **Device template**
2. Select a device from **Devices** for the device you want to see on a dashboard tile. Then you will see a list of the device's properties that can be used on the tile.
3. To create the tile on the dashboard, click on **Temperature** and drag it to the dashboard area. You can also click the checkbox next to **Temperature** and click **Add tile**. The following screenshot shows selecting a Device Template and device then creating a Temperature Telemetry tile on the dashboard.
4. Select **Save** in the top left to save your changes to the dashboard.



Now when you view your personal dashboard, you see the new tile with the **Temperature** setting for the device:

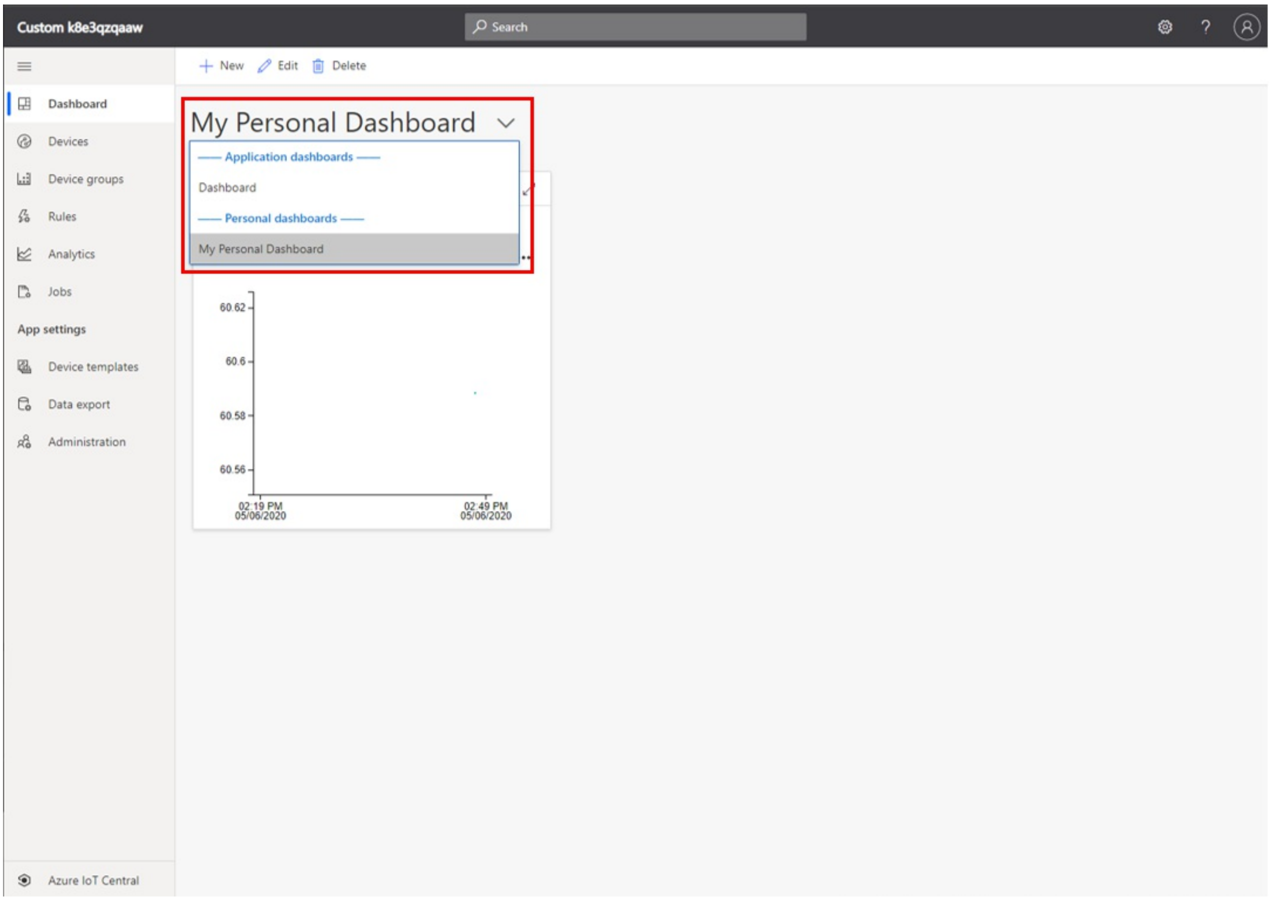


You can explore other tile types in the library to discover how to further customize your personal dashboards.

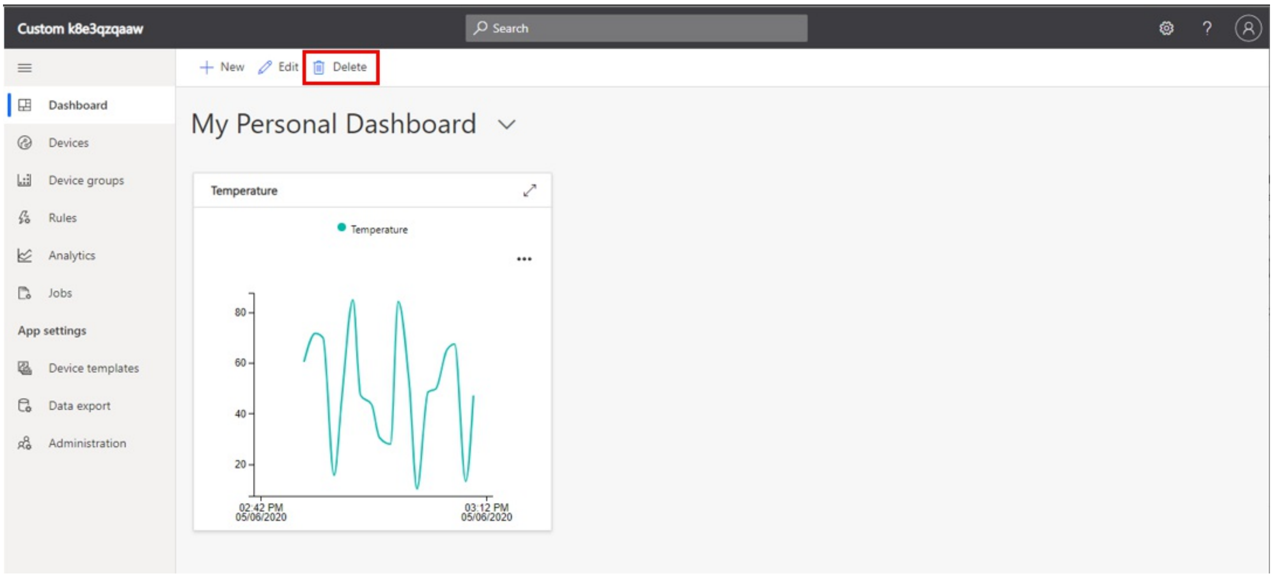
To learn more about how to use tiles in Azure IoT Central, see Add Tiles to your Dashboard.

Manage dashboards

You can have several personal dashboards and switch between them or choose from one of the default application dashboards:



You can edit your personal dashboards and delete any dashboards you no longer need. If you are an **admin**, you also have the ability to edit or delete application level dashboards as well.



Next steps

Now that you've learned how to create and manage personal dashboards, you can Learn how to manage your application preferences.

Create and run a job in your Azure IoT Central application

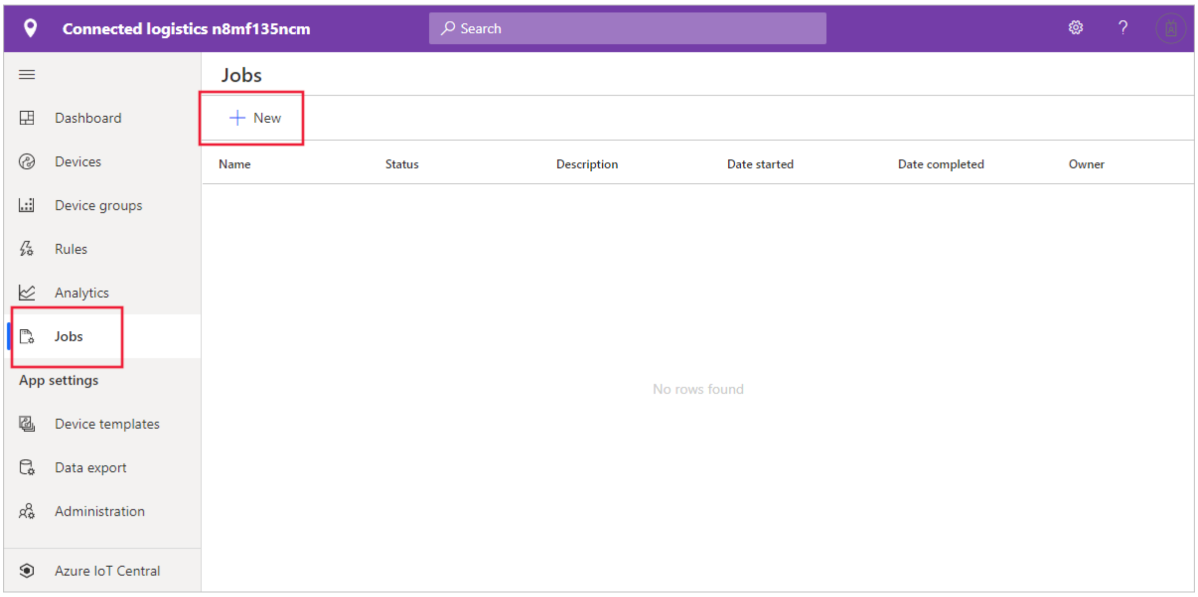
7/21/2020 • 4 minutes to read • Edit Online

You can use Microsoft Azure IoT Central to manage your connected devices at scale using jobs. Jobs let you do bulk updates to device properties and run commands. This article shows you how to get started using jobs in your own application.

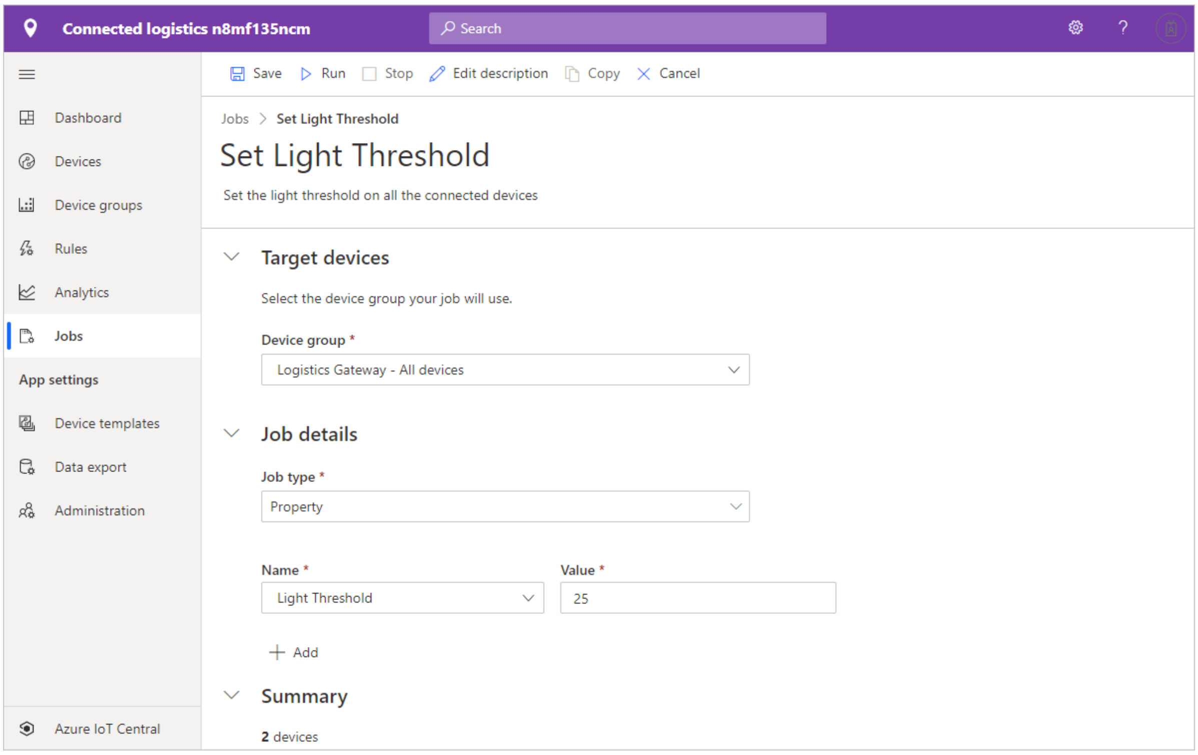
Create and run a job

This section shows you how to create and run a job. It shows you how to set the light threshold for a group of logistic gateway devices.

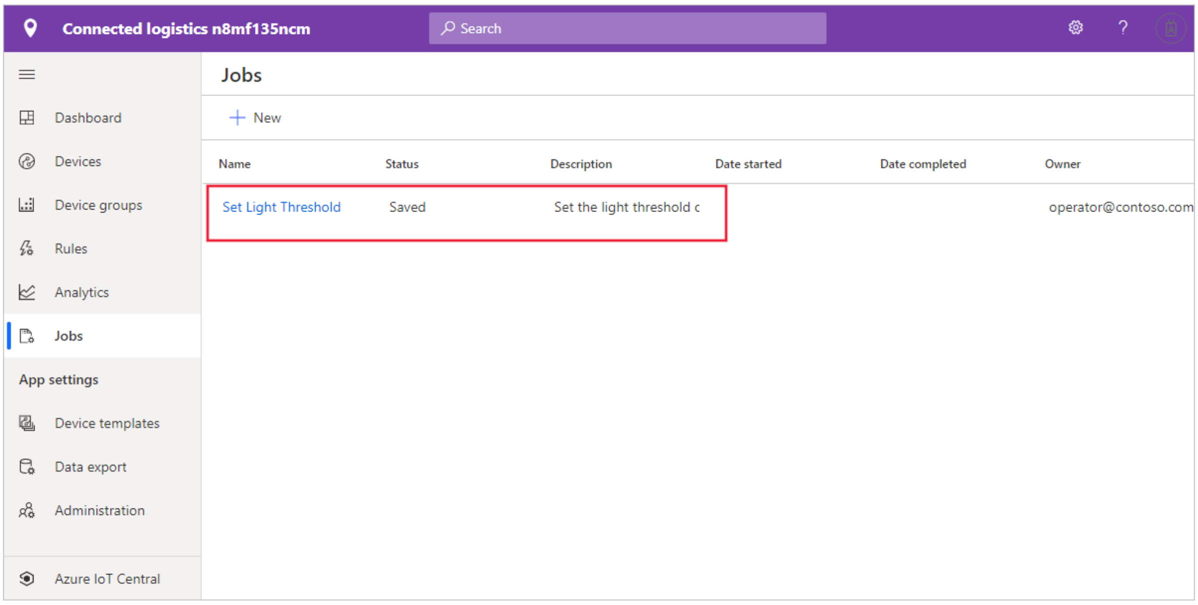
1. Navigate to **Jobs** from the left pane.
2. Select **+ New** to create a new job:



3. Enter a name and description to identify the job you're creating.
4. Select the target device group you want your job to apply to. You can see how many devices your job configuration applies to in the **Summary** section.
5. Next, choose either **Cloud property**, **Property**, or **Command** as the type of job to configure. To set up a **Property** job configuration, select a property and set its new value. To set up a **Command**, choose the command to run. A property job can set multiple properties:

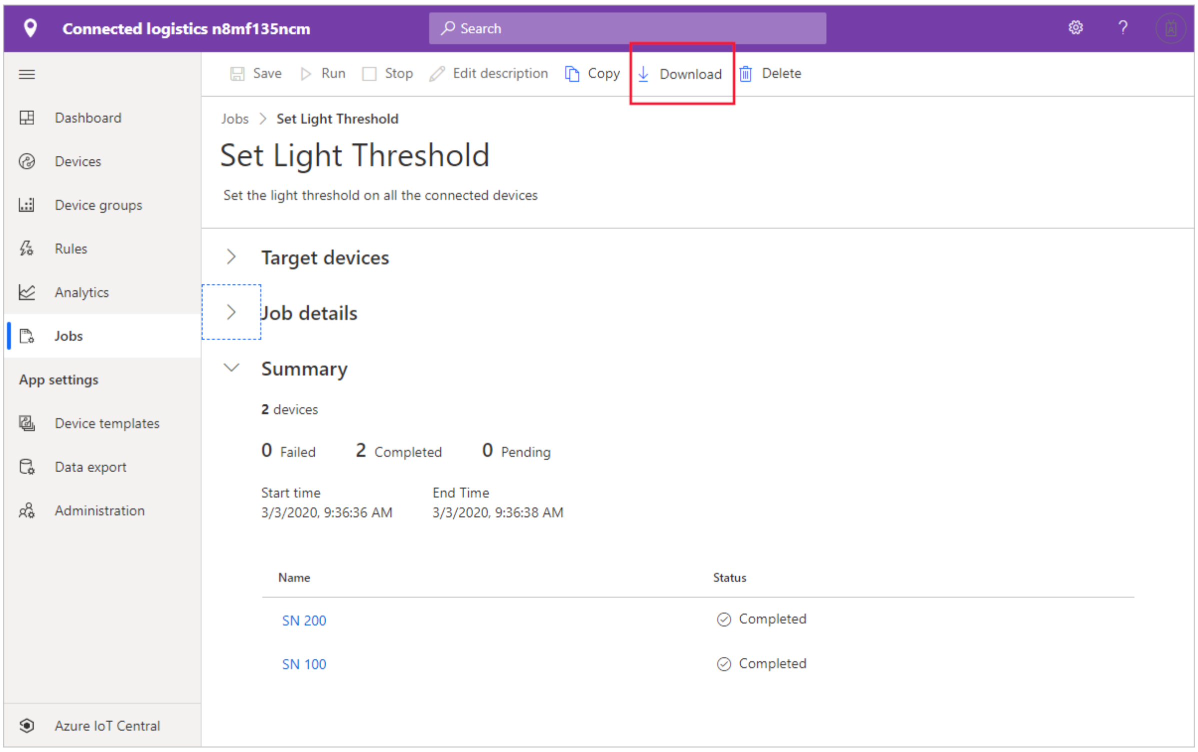


6. After creating your job, choose **Run** or **Save**. The job now appears on your main **Jobs** page. On this page, you can see your currently running job and the history of any previously run or saved jobs. Your saved job can be opened again at any time to continue editing it or to run it:



> **NOTE**
>
> You can view up 30 days of history for your previously run jobs.

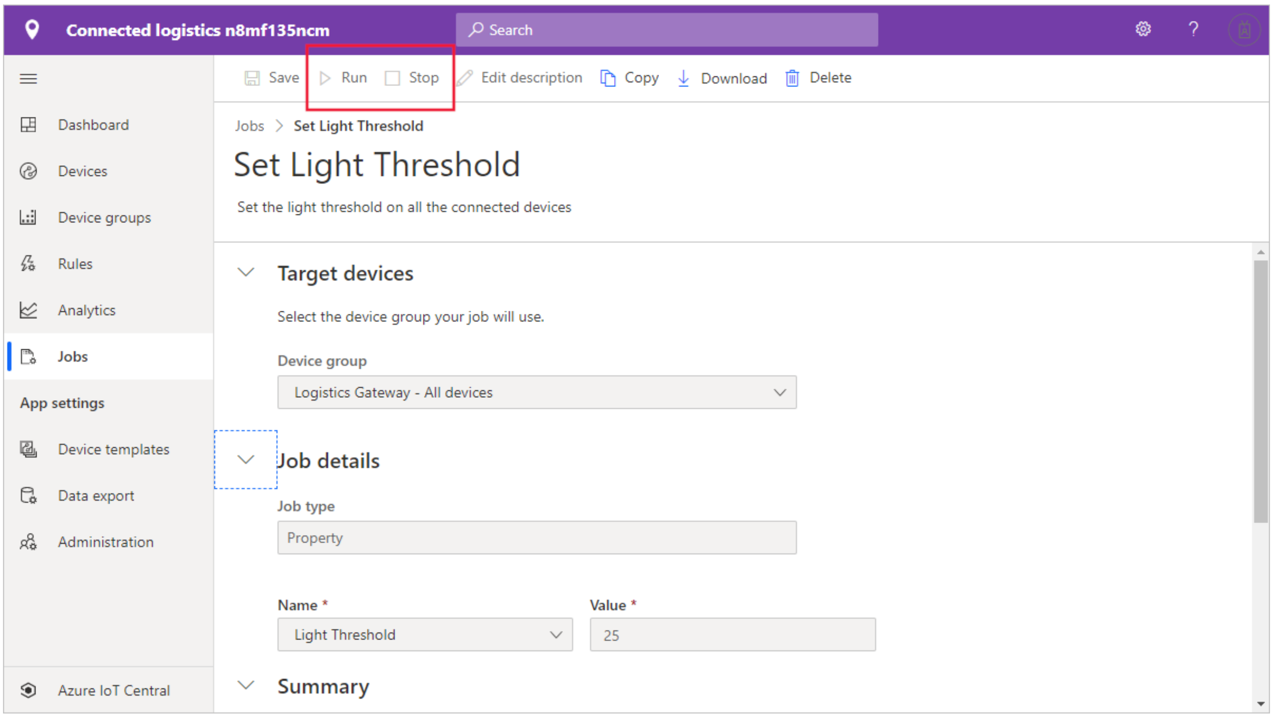
7. To get an overview of your job, select the job to view from the list. This overview contains the job details, devices, and device status values. From this overview, you can also select **Download Job Details** to download a CSV file of your job details, including the devices and their status values. This information can be useful for troubleshooting:



Manage jobs

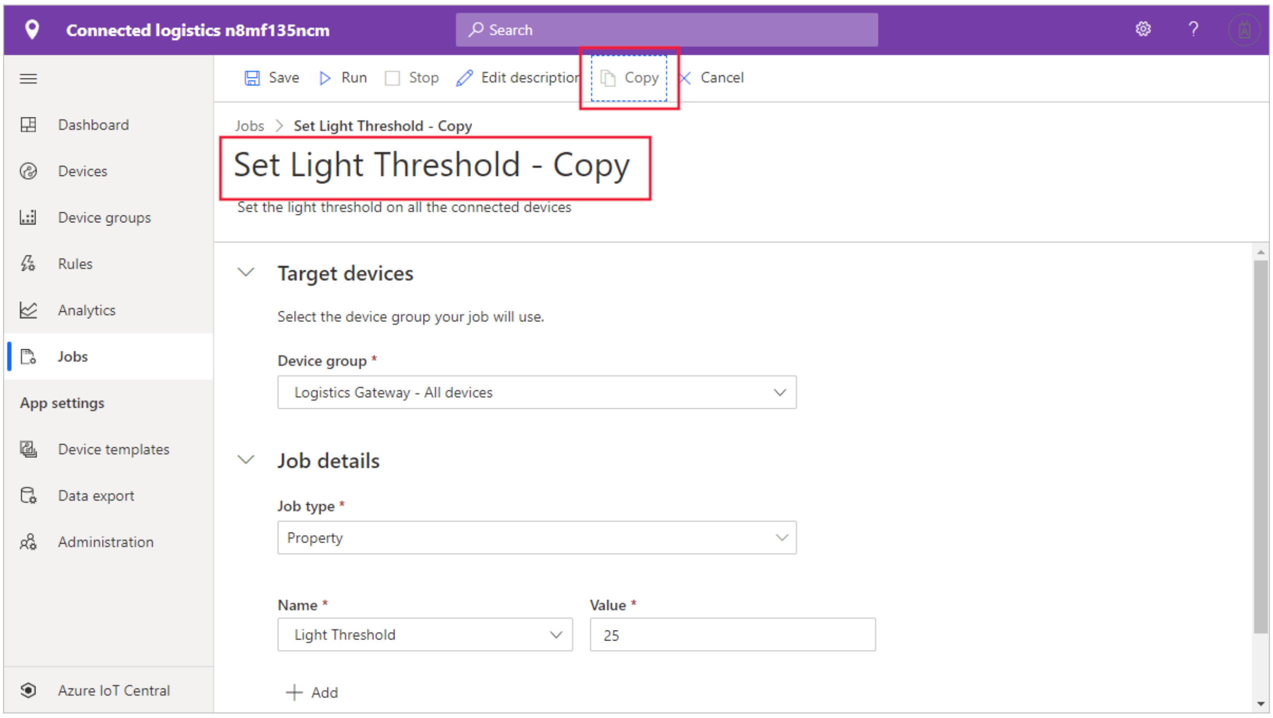
To stop one of your running jobs, open it and select **Stop**. The job status changes to reflect the job is stopped. The **Summary** section shows which devices have completed, failed, or are still pending.

To run a job that's currently stopped, select it, and then select **Run**. The job status changes to reflect the job is now running again. The **Summary** section continues to update with the latest progress.



Copy a job

To copy one of your existing jobs, select it on the **Jobs** page and select **Copy**. A copy of the job configuration opens for you to edit, and **Copy** is appended to the job name. You can save or run the new job:



View job status

After a job is created, the **Status** column updates with the latest status message of the job. The following table lists the possible status values:

STATUS MESSAGE	STATUS MEANING
Completed	This job has been executed on all devices.
Failed	This job has failed and not fully executed on devices.
Pending	This job hasn't yet begun executing on devices.
Running	This job is currently executing on devices.
Stopped	This job has been manually stopped by a user.

The status message is followed by an overview of the devices in the job. The following table lists the possible device status values:

STATUS MESSAGE	STATUS MEANING
Succeeded	The number of devices that the job successfully executed on.
Failed	The number of devices that the job has failed to execute on.

View the device status values

To view the status of the job and all the affected devices, open the job. Next to each device name, you see one of the following status messages:

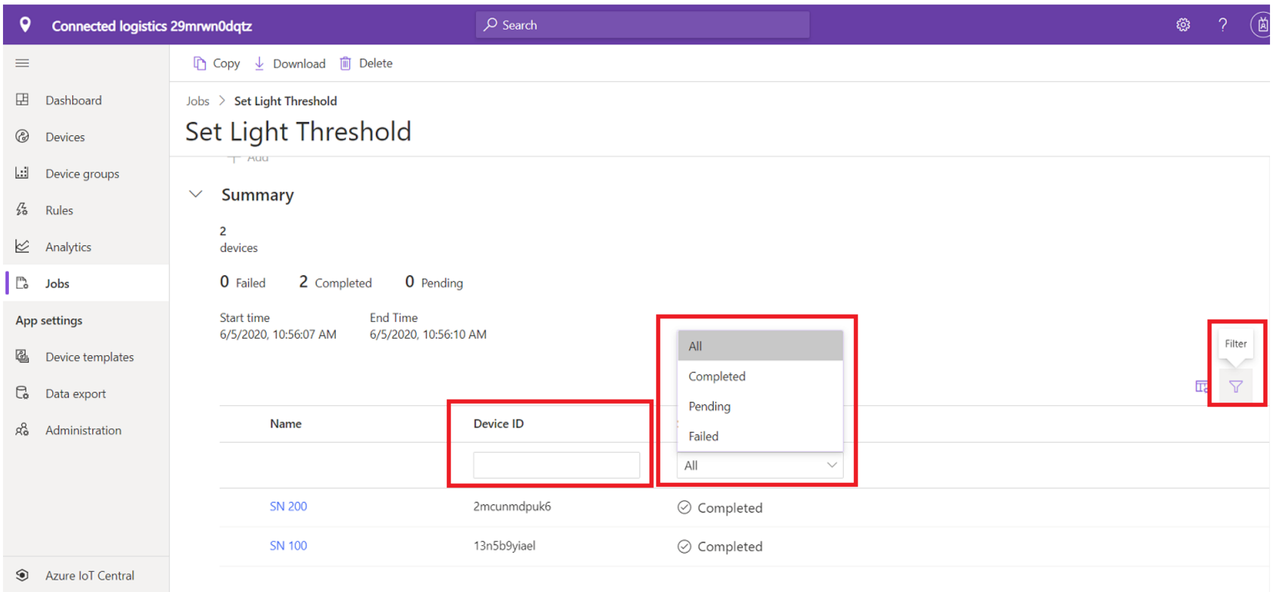
STATUS MESSAGE	STATUS MEANING
Completed	The job executed on this device.

STATUS MESSAGE	STATUS MEANING
Failed	The job failed to execute on this device. The error message shows more information.
Pending	The job hasn't yet executed on this device.

To download a CSV file that includes the job details and the list of devices and their status values, select **Download**.

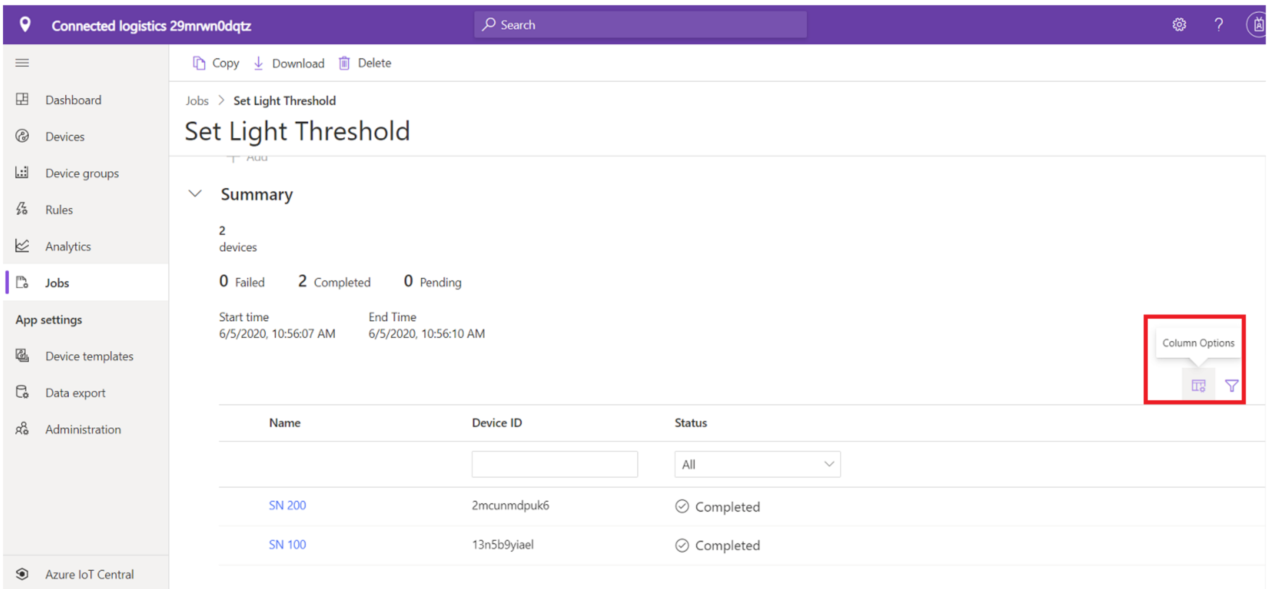
Filter the list of devices

You can filter the device list on the job details page by selecting the filter icon. You can filter on the **Device ID** or **Status** fields:

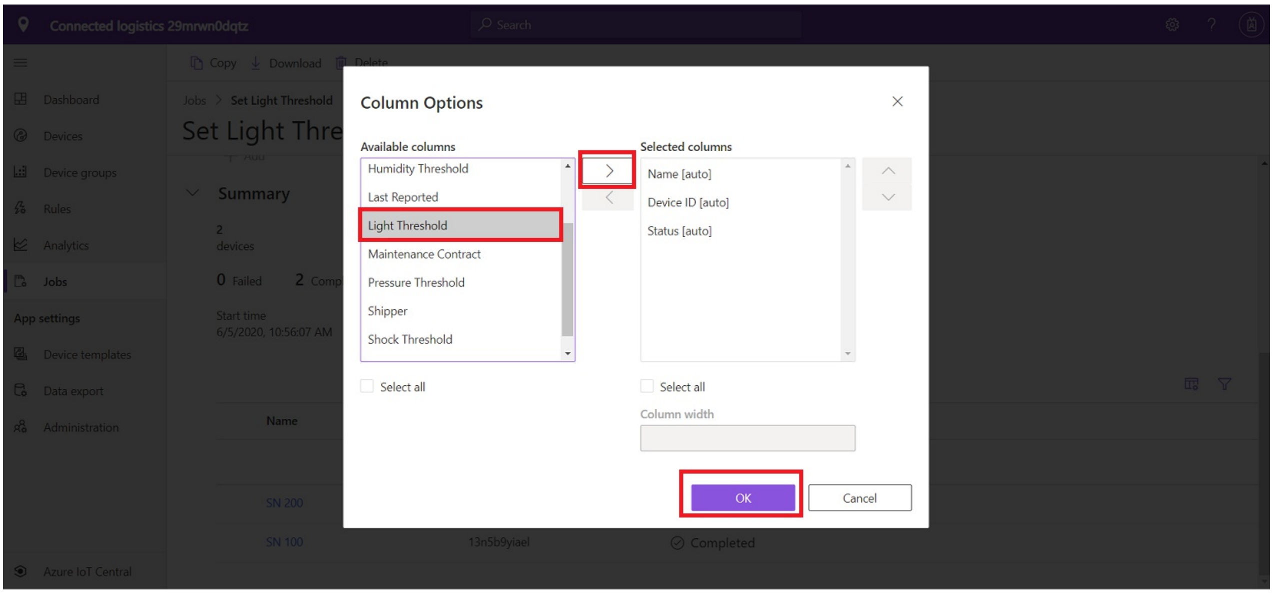


Customize columns in the device list

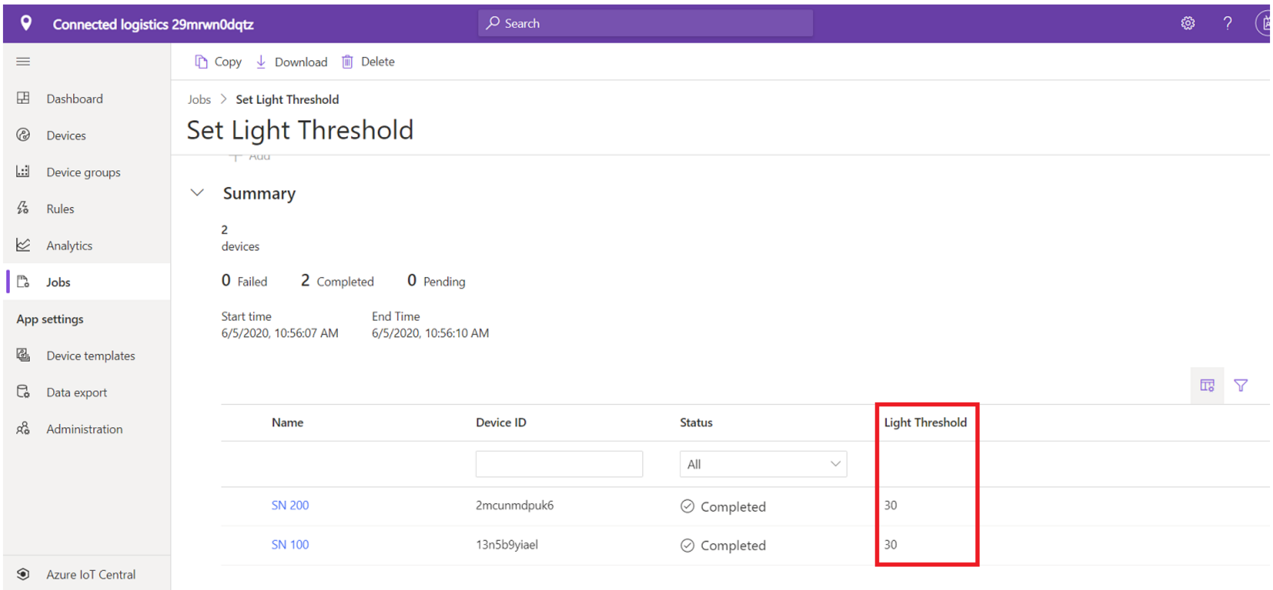
You can choose additional columns to display in the device list by selecting the column options icon:



You see a dialog that lets you choose the columns to display in the device list. Select the columns you want to display, select the right arrow icon, and the select **OK**. To select all the available columns, check **Select all**:



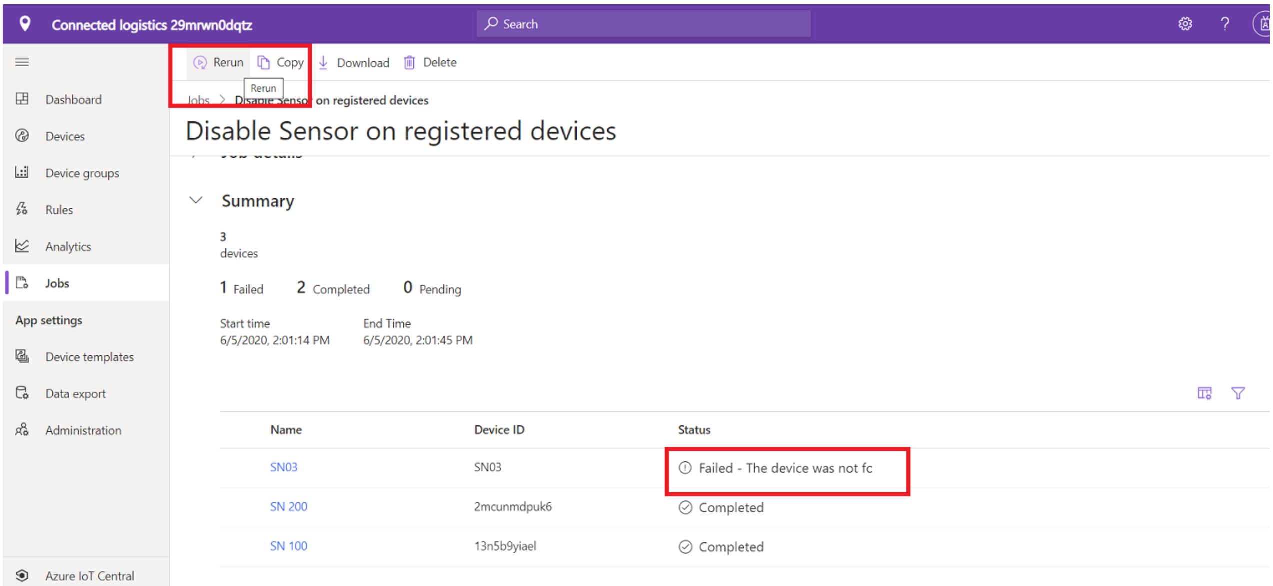
The selected columns display on the device list:



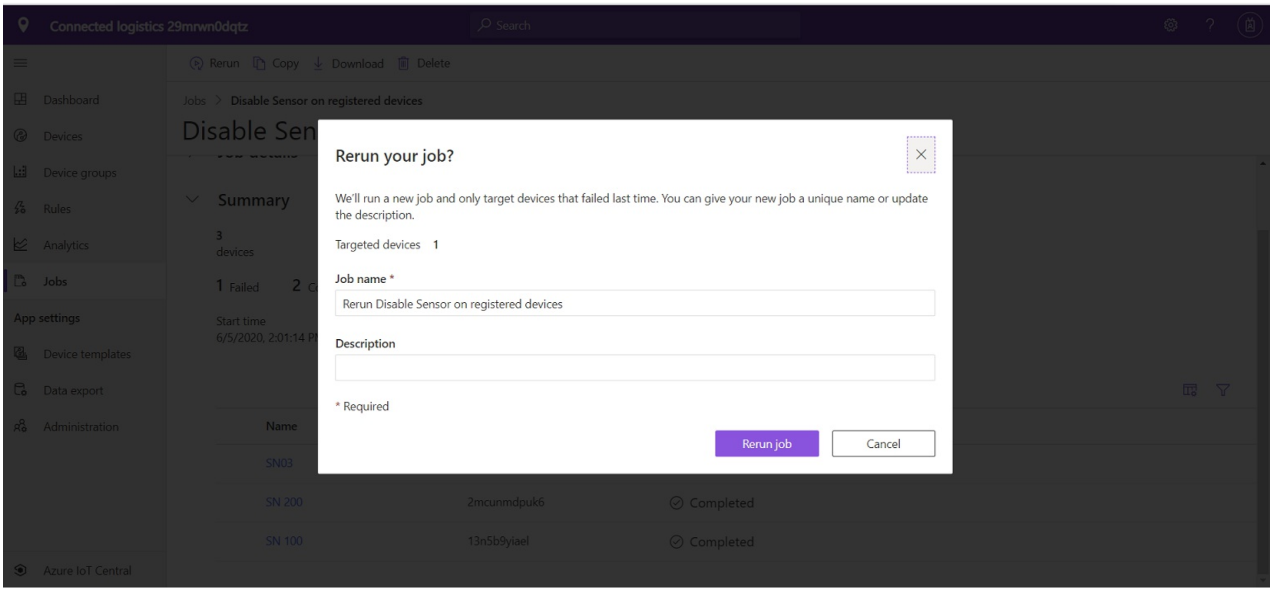
Selected columns are persisted across a user session or across user sessions that have access to the application.

Rerun jobs

You can rerun a job that has failed devices. Select **Rerun**:



Enter a job name and description, then select **Rerun job**. A new job is submitted to retry the action on failed devices:



NOTE

You can't execute more than five jobs at the same time from an IoT Central application.

NOTE

When a job is complete and you delete a device that's in the job's device list, the device entry shows as deleted in the device name and device details link isn't available for the deleted device.

Next steps

Now that you've learned how to create jobs in your Azure IoT Central application, here are some next steps:

- Manage your devices
- Version your device template

Export IoT data to destinations in Azure

7/21/2020 • 14 minutes to read • Edit Online

This topic applies to administrators.

This article describes how to use the data export feature in Azure IoT Central. This feature lets you export your data continuously to **Azure Event Hubs**, **Azure Service Bus**, or **Azure Blob storage** instances. Data export uses the JSON format and can include telemetry, device information, and device template information. Use the exported data for:

- Warm-path insights and analytics. This option includes triggering custom rules in Azure Stream Analytics, triggering custom workflows in Azure Logic Apps, or passing it through Azure Functions to be transformed.
- Cold-path analytics such as training models in Azure Machine Learning or long-term trend analysis in Microsoft Power BI.

> **NOTE**
> When you turn on data export, you get only the data from that moment onward. Currently, data can't be retrieved for a time when data export was off. To retain more historical data, turn on data export early.

Prerequisites

You must be an administrator in your IoT Central application, or have Data export permissions.

Set up export destination

Your export destination must exist before you configure your data export.

Create Event Hubs namespace

If you don't have an existing Event Hubs namespace to export to, follow these steps:

1. Create a new Event Hubs namespace in the Azure portal. You can learn more in Azure Event Hubs docs.
2. Choose a subscription. You can export data to other subscriptions that aren't in the same subscription as your IoT Central application. You connect using a connection string in this case.
3. Create an event hub in your Event Hubs namespace. Go to your namespace, and select **+ Event Hub** at the top to create an event hub instance.

Create Service Bus namespace

If you don't have an existing Service Bus namespace to export to, follow these steps:

1. Create a new Service Bus namespace in the Azure portal. You can learn more in Azure Service Bus docs.
2. Choose a subscription. You can export data to other subscriptions that aren't in the same subscription as your IoT Central application. You connect using a connection string in this case.
3. To create a queue or topic to export to, go to your Service Bus namespace, and select **+ Queue** or **+ Topic**.

When you choose Service Bus as an export destination, the queues and topics must not have Sessions or Duplicate Detection enabled. If either of those options are enabled, some messages won't arrive in your queue or topic.

Create storage account

If you don't have an existing Azure storage account to export to, follow these steps:

1. Create a new storage account in the Azure portal. You can learn more about creating new Azure Blob storage accounts or Azure Data Lake Storage v2 storage accounts. Data export can only write data to storage accounts that support block blobs. The following list shows the known compatible storage account types:

PERFORMANCE TIER	ACCOUNT TYPE
Standard	General Purpose V2
Standard	General Purpose V1
Standard	Blob storage
Premium	Block Blob storage

2. Create a container in your storage account. Go to your storage account. Under **Blob Service**, select **Browse Blobs**. Select **+ Container** at the top to create a new container.

Set up data export

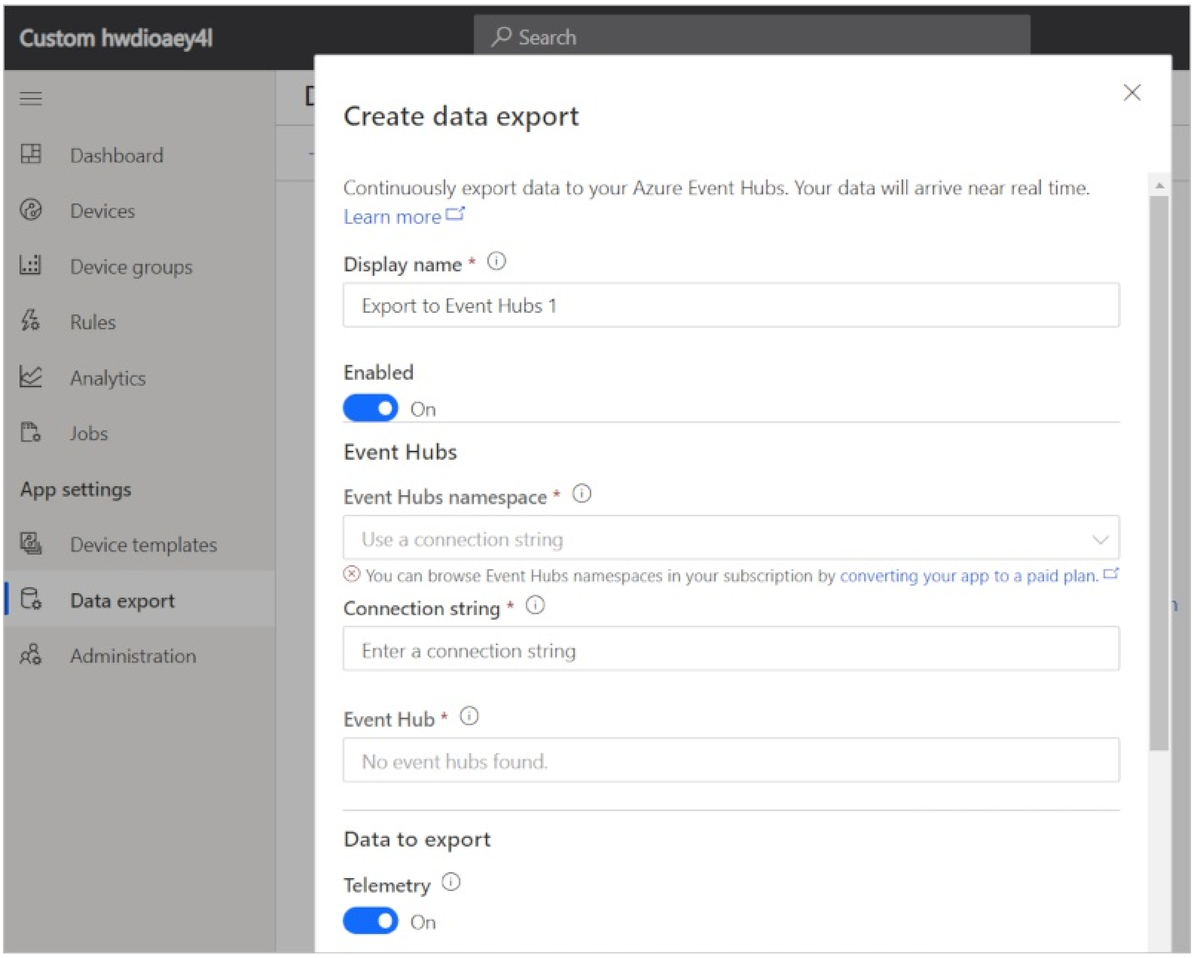
Now that you have a destination to export data to, follow these steps to set up data export.

1. Sign in to your IoT Central application.
2. In the left pane, select **Data export**.

> **TIP**
>
> If you don't see **Data export** in the left pane, then you don't have permissions to configure data export in your app. Talk to an administrator to set up data export.

3. Select the **+ New** button. Choose one of **Azure Blob Storage**, **Azure Event Hubs**, **Azure Service Bus Queue**, or **Azure Service Bus Topic** as the destination of your export. The maximum number of exports per application is five.
4. Enter a name for the export. In the drop-down list box, select your **namespace**, or **Enter a connection string**.

   - You only see storage accounts, Event Hubs namespaces, and Service Bus namespaces in the same subscription as your IoT Central application. If you want to export to a destination outside of this subscription, choose **Enter a connection string** and see step 6.
   - For apps created using the free pricing plan, the only way to configure data export is through a connection string. Apps on the free pricing plan don't have an associated Azure subscription.



5. Choose an event hub, queue, topic, or container from the drop-down list box.

6. (Optional) If you chose **Enter a connection string**, a new box appears for you to paste your connection string. To get the connection string for your:

- Event Hubs or Service Bus, go to the namespace in the Azure portal:
  - To use a connection string for the entire namespace:
    a. Under **Settings**, select **Shared Access Policies**
    b. Create a new key or choose an existing key that has **Send** permissions.
    c. Copy either the primary or secondary connection string
  - To use connection string for a specific event hub instance or Service Bus queue or topic, go to **Entities > Event Hubs** or **Entities > Queues** or **Entities > Topics**. Choose a specific instance, and follow the same steps above to get a connection string.
- Storage account, go to the storage account in the Azure portal:
  - Only connection strings for the entire storage account are supported. Connection strings scoped to a single container are not supported.
    a. Under **Settings**, select **Access keys**
    b. Copy either the key1 connection string or the key2 connection string

Paste in the connection string. Type in the instance or **container name**, keeping in mind this is case-sensitive.

7. Under **Data to export**, choose the types of data to export by setting the type to **On**.

8. To turn on data export, make sure the **Enabled** toggle is **On**. Select **Save**.

9. After a few minutes, your data appears in your chosen destination.

Export contents and format

Exported telemetry data contains the entirety of the message your devices sent to IoT Central, not just the telemetry values themselves. Exported devices data contains changes to properties and metadata of all devices, and exported device templates contains changes to all device templates.

For Event Hubs and Service Bus, data is exported in near-realtime. The data is in the `body` property and is in JSON format. See below for examples.

For Blob storage, data is exported once per minute, with each file containing the batch of changes since the last exported file. Exported data is placed in three folders in JSON format. The default paths in your storage account are:

- Telemetry: *{container}/{app-id}/telemetry/{YYYY}/{MM}/{dd}/{hh}/{mm}/{filename}*
- Devices: *{container}/{app-id}/devices/{YYYY}/{MM}/{dd}/{hh}/{mm}/{filename}*
- Device templates: *{container}/{app-id}/deviceTemplates/{YYYY}/{MM}/{dd}/{hh}/{mm}/{filename}*

To browse the exported files in the Azure portal, navigate to the file and select the **Edit blob** tab.

Telemetry

For Event Hubs and Service Bus, IoT Central exports a new message quickly after it receives the message from a device. Each exported message contains the full message the device sent in the body property in JSON format.

For Blob storage, messages are batched and exported once per minute. The exported files use the same format as the message files exported by IoT Hub message routing to blob storage.

> **NOTE**
>
> For Blob storage, ensure that your devices are sending messages that have `contentType: application/JSON` and `contentEncoding:utf-8` (or `utf-16`, `utf-32`). See the IoT Hub documentation for an example.

The device that sent the telemetry is represented by the device ID (see the following sections). To get the names of the devices, export device data and correlate each message by using the **connectionDeviceId** that matches the **deviceId** of the device message.

The following example shows a message received from an event hub or Service Bus queue or topic:

```
{
  "temp":81.129693132351775,
  "humid":59.488071477541247,
  "EventProcessedUtcTime":"2020-04-07T09:41:15.2877981Z",
  "PartitionId":0,
  "EventEnqueuedUtcTime":"2020-04-07T09:38:32.7380000Z"
}
```

This message doesn't include the device ID of the sending device.

To retrieve the device ID from the message data in an Azure Stream Analytics query, use the GetMetadataPropertyValue function. For an example, see the query in Extend Azure IoT Central with custom rules using Stream Analytics, Azure Functions, and SendGrid.

To retrieve the device ID in an Azure Databricks or Apache Spark workspace, use systemProperties. For an example, see the Databricks workspace in Extend Azure IoT Central with custom analytics using Azure Databricks.

The following example shows a record exported to blob storage:

```
{
  "EnqueuedTimeUtc":"2019-09-26T17:46:09.8870000Z",
  "Properties":{

  },
  "SystemProperties":{
    "connectionDeviceId":"<deviceid>",
    "connectionAuthMethod":"
{\"scope\":\"device\",\"type\":\"sas\",\"issuer\":\"iothub\",\"acceptingIpFilterRule\":null}",
    "connectionDeviceGenerationId":"637051167384630591",
    "contentType":"application/json",
    "contentEncoding":"utf-8",
    "enqueuedTime":"2019-09-26T17:46:09.8870000Z"
  },
  "Body":{
    "temp":49.91322758395974,
    "humid":49.61214852573155,
    "pm25":25.87332214661367
  }
}
```


Devices

Each message or record in a snapshot represents one or more changes to a device and its device and cloud properties since the last exported message. The message includes the:

- `id` of the device in IoT Central
- `displayName` of the device
- Device template ID in `instanceOf`
- `simulated` flag, true if the device is a simulated device
- `provisioned` flag, true if the device has been provisioned
- `approved` flag, true if the device has been approved to send data
- Property values
- `properties` including device and cloud properties values

Deleted devices aren't exported. Currently, there are no indicators in exported messages for deleted devices.

For Event Hubs and Service Bus, IoT Central sends messages containing device data to your event hub or Service Bus queue or topic in near real time.

For Blob storage, a new snapshot containing all the changes since the last one written is exported once per minute.

The following example message shows information about devices and properties data in an event hub or Service Bus queue or topic:

```
{
  "body":{
    "id": "<device Id>",
    "etag": "<etag>",
    "displayName": "Sensor 1",
    "instanceOf": "<device template Id>",
    "simulated": false,
    "provisioned": true,
    "approved": true,
    "properties": {
        "sensorComponent": {
            "setTemp": "30",
            "fwVersion": "2.0.1",
            "status": { "first": "first", "second": "second" },
            "$metadata": {
                "setTemp": {
                    "desiredValue": "30",
                    "desiredVersion": 3,
                    "desiredTimestamp": "2020-02-01T17:15:08.9284049Z",
                    "ackVersion": 3
                },
                "fwVersion": { "ackVersion": 3 },
                "status": {
                    "desiredValue": {
                        "first": "first",
                        "second": "second"
                    },
                    "desiredVersion": 2,
                    "desiredTimestamp": "2020-02-01T17:15:08.9284049Z",
                    "ackVersion": 2
                }
            },

        }
    },
    "installDate": { "installDate": "2020-02-01" }
},
  "annotations":{
    "iotcentral-message-source":"devices",
    "x-opt-partition-key":"<partitionKey>",
    "x-opt-sequence-number":39740,
    "x-opt-offset":"<offset>",
    "x-opt-enqueued-time":1539274959654
  },
  "partitionKey":"<partitionKey>",
  "sequenceNumber":39740,
  "enqueuedTimeUtc":"2020-02-01T18:14:49.3820326Z",
  "offset":"<offset>"
}
```


This snapshot is an example message that shows devices and properties data in Blob storage. Exported files contain a single line per record.

```json
{
  "id": "<device Id>",
  "etag": "<etag>",
  "displayName": "Sensor 1",
  "instanceOf": "<device template Id>",
  "simulated": false,
  "provisioned": true,
  "approved": true,
  "properties": {
      "sensorComponent": {
          "setTemp": "30",
          "fwVersion": "2.0.1",
          "status": { "first": "first", "second": "second" },
          "$metadata": {
              "setTemp": {
                  "desiredValue": "30",
                  "desiredVersion": 3,
                  "desiredTimestamp": "2020-02-01T17:15:08.9284049Z",
                  "ackVersion": 3
              },
              "fwVersion": { "ackVersion": 3 },
              "status": {
                  "desiredValue": {
                      "first": "first",
                      "second": "second"
                  },
                  "desiredVersion": 2,
                  "desiredTimestamp": "2020-02-01T17:15:08.9284049Z",
                  "ackVersion": 2
              }
          },

      }
  },
  "installDate": { "installDate": "2020-02-01" }
}
```

Device templates

Each message or snapshot record represents one or more changes to a published device template since the last exported message. Information sent in each message or record includes:

- `id` of the device template that matches the `instanceOf` of the devices stream above
- `displayName` of the device template
- The device `capabilityModel` including its `interfaces`, and the telemetry, properties, and commands definitions
- `cloudProperties` definitions
- Overrides and initial values, inline with the `capabilityModel`

Deleted device templates aren't exported. Currently, there are no indicators in exported messages for deleted device templates.

For Event Hubs and Service Bus, IoT Central sends messages containing device template data to your event hub or Service Bus queue or topic in near real time.

For Blob storage, a new snapshot containing all the changes since the last one written is exported once per minute.

This example shows a message about device templates data in event hub or Service Bus queue or topic:

```json
{
  "body":{
```

```
        "id": "<device template id>",
        "etag": "<etag>",
        "types": ["DeviceModel"],
        "displayName": "Sensor template",
        "capabilityModel": {
            "@id": "<capability model id>",
            "@type": ["CapabilityModel"],
            "contents": [],
            "implements": [
                {
                    "@id": "<component Id>",
                    "@type": ["InterfaceInstance"],
                    "name": "sensorComponent",
                    "schema": {
                        "@id": "<interface Id>",
                        "@type": ["Interface"],
                        "displayName": "Sensor interface",
                        "contents": [
                            {
                                "@id": "<id>",
                                "@type": ["Telemetry"],
                                "displayName": "Humidity",
                                "name": "humidity",
                                "schema": "double"
                            },
                            {
                                "@id": "<id>",
                                "@type": ["Telemetry", "SemanticType/Event"],
                                "displayName": "Error event",
                                "name": "error",
                                "schema": "integer"
                            },
                            {
                                "@id": "<id>",
                                "@type": ["Property"],
                                "displayName": "Set temperature",
                                "name": "setTemp",
                                "writable": true,
                                "schema": "integer",
                                "unit": "Units/Temperature/fahrenheit",
                                "initialValue": "30"
                            },
                            {
                                "@id": "<id>",
                                "@type": ["Property"],
                                "displayName": "Firmware version read only",
                                "name": "fwversion",
                                "schema": "string"
                            },
                            {
                                "@id": "<id>",
                                "@type": ["Property"],
                                "displayName": "Display status",
                                "name": "status",
                                "writable": true,
                                "schema": {
                                    "@id": "urn:testInterface:status:obj:ka8iw8wka:1",
                                    "@type": ["Object"]
                                }
                            },
                            {
                                "@id": "<id>",
                                "@type": ["Command"],
                                "commandType": "synchronous",
                                "request": {
                                    "@id": "<id>",
                                    "@type": ["SchemaField"],
                                    "displayName": "Configuration",
                                    "name": "config",
```

```json
                                "schema": "string"
                            },
                            "response": {
                                "@id": "<id>",
                                "@type": ["SchemaField"],
                                "displayName": "Response",
                                "name": "response",
                                "schema": "string"
                            },
                            "displayName": "Configure sensor",
                            "name": "sensorConfig"
                        }
                    ]
                }
            }
        ],
        "displayName": "Sensor capability model"
    },
    "solutionModel": {
        "@id": "<id>",
        "@type": ["SolutionModel"],
        "cloudProperties": [
            {
                "@id": "<id>",
                "@type": ["CloudProperty"],
                "displayName": "Install date",
                "name": "installDate",
                "schema": "dateTime",
                "valueDetail": {
                    "@id": "<id>",
                    "@type": ["ValueDetail/DateTimeValueDetail"]
                }
            }
        ]
    }
  },
    "annotations":{
      "iotcentral-message-source":"deviceTemplates",
      "x-opt-partition-key":"<partitionKey>",
      "x-opt-sequence-number":25315,
      "x-opt-offset":"<offset>",
      "x-opt-enqueued-time":1539274985085
    },
    "partitionKey":"<partitionKey>",
    "sequenceNumber":25315,
    "enqueuedTimeUtc":"2019-10-02T16:23:05.085Z",
    "offset":"<offset>"
  }
}
```

This example snapshot shows a message that contains device and properties data in Blob storage. Exported files contain a single line per record.

```json
{
      "id": "<device template id>",
      "etag": "<etag>",
      "types": ["DeviceModel"],
      "displayName": "Sensor template",
      "capabilityModel": {
          "@id": "<capability model id>",
          "@type": ["CapabilityModel"],
          "contents": [],
          "implements": [
              {
                  "@id": "<component Id>",
                  "@type": ["InterfaceInstance"],
                  "name": "Sensor component",
```

```
"schema": {
    "@id": "<interface Id>",
    "@type": ["Interface"],
    "displayName": "Sensor interface",
    "contents": [
        {
            "@id": "<id>",
            "@type": ["Telemetry"],
            "displayName": "Humidity",
            "name": "humidity",
            "schema": "double"
        },
        {
            "@id": "<id>",
            "@type": ["Telemetry", "SemanticType/Event"],
            "displayName": "Error event",
            "name": "error",
            "schema": "integer"
        },
        {
            "@id": "<id>",
            "@type": ["Property"],
            "displayName": "Set temperature",
            "name": "setTemp",
            "writable": true,
            "schema": "integer",
            "unit": "Units/Temperature/fahrenheit",
            "initialValue": "30"
        },
        {
            "@id": "<id>",
            "@type": ["Property"],
            "displayName": "Firmware version read only",
            "name": "fwversion",
            "schema": "string"
        },
        {
            "@id": "<id>",
            "@type": ["Property"],
            "displayName": "Display status",
            "name": "status",
            "writable": true,
            "schema": {
                "@id": "urn:testInterface:status:obj:ka8iw8wka:1",
                "@type": ["Object"]
            }
        },
        {
            "@id": "<id>",
            "@type": ["Command"],
            "commandType": "synchronous",
            "request": {
                "@id": "<id>",
                "@type": ["SchemaField"],
                "displayName": "Configuration",
                "name": "config",
                "schema": "string"
            },
            "response": {
                "@id": "<id>",
                "@type": ["SchemaField"],
                "displayName": "Response",
                "name": "response",
                "schema": "string"
            },
            "displayName": "Configure sensor",
            "name": "sensorconfig"
        }
    ]
```

```
                    ]
                }
            }
        ],
        "displayName": "Sensor capability model"
    },
    "solutionModel": {
        "@id": "<id>",
        "@type": ["SolutionModel"],
        "cloudProperties": [
            {
                "@id": "<id>",
                "@type": ["CloudProperty"],
                "displayName": "Install date",
                "name": "installDate",
                "schema": "dateTime",
                "valueDetail": {
                    "@id": "<id>",
                    "@type": ["ValueDetail/DateTimeValueDetail"]
                }
            }
        ]
    }
}
```

Data format change notice

> **NOTE**
>
> The telemetry stream data format is unaffected by this change. Only the devices and device templates streams of data are affected.

If you have an existing data export in your preview application with the *Devices* and *Device templates* streams turned on, update your export by **30 June 2020**. This requirement applies to exports to Azure Blob storage, Azure Event Hubs, and Azure Service Bus.

Starting 3 February 2020, all new exports in applications with Devices and Device templates enabled will have the data format described above. All exports created before this date remain on the old data format until 30 June 2020, at which time these exports will automatically be migrated to the new data format. The new data format matches the device, device property, device cloud property, and device template objects in the IoT Central public API.

For **Devices**, notable differences between the old data format and the new data format include:

- `@id` for device is removed, `deviceId` is renamed to `id`
- `provisioned` flag is added to describe the provisioning status of the device
- `approved` flag is added to describe the approval state of the device
- `properties` including device and cloud properties, matches entities in the public API

For **Device templates**, notable differences between the old data format and the new data format include:

- `@id` for device template is renamed to `id`
- `@type` for the device template is renamed to `types`, and is now an array

Devices (format deprecated as of 3 February 2020)

```
{
  "@id":"<id-value>",
  "@type":"Device",
  "displayName":"Airbox",
  "data":{
    "$cloudProperties":{
        "Color":"blue"
    },
    "EnvironmentalSensor":{
      "thsensormodel":{
        "reported":{
          "value":"Neque quia et voluptatem veritatis assumenda consequuntur quod.",
          "$lastUpdatedTimestamp":"2019-09-30T20:35:43.8478978Z"
        }
      },
      "pm25sensormodel":{
        "reported":{
          "value":"Aut alias odio.",
          "$lastUpdatedTimestamp":"2019-09-30T20:35:43.8478978Z"
        }
      }
    },
    "urn_azureiot_DeviceManagement_DeviceInformation":{
      "totalStorage":{
        "reported":{
          "value":27900.9730905171,
          "$lastUpdatedTimestamp":"2019-09-30T20:35:43.8478978Z"
        }
      },
      "totalMemory":{
        "reported":{
          "value":4667.82916715811,
          "$lastUpdatedTimestamp":"2019-09-30T20:35:43.8478978Z"
        }
      }
    }
  },
  "instanceOf":"<template-id>",
  "deviceId":"<device-id>",
  "simulated":true
}
```

Device templates (format deprecated as of 3 February 2020)

```
{
  "@id":"<template-id>",
  "@type":"DeviceModelDefinition",
  "displayName":"Airbox",
  "capabilityModel":{
    "@id":"<id>",
    "@type":"CapabilityModel",
    "implements":[
      {
        "@id":"<id>",
        "@type":"InterfaceInstance",
        "name":"EnvironmentalSensor",
        "schema":{
          "@id":"<id>",
          "@type":"Interface",
          "comment":"Requires temperature and humidity sensors.",
          "description":"Provides functionality to report temperature, humidity. Provides telemetry, commands
and read-write properties",
          "displayName":"Environmental Sensor",
          "contents":[
            {
              "@id":"<id>",
```

```
        "@type":"Telemetry",
        "description":"Current temperature on the device",
        "displayName":"Temperature",
        "name":"temp",
        "schema":"double",
        "unit":"Units/Temperature/celsius",
        "valueDetail":{
          "@id":"<id>",
          "@type":"ValueDetail/NumberValueDetail",
          "minValue":{
            "@value":"50"
          }
        },
        "visualizationDetail":{
          "@id":"<id>",
          "@type":"VisualizationDetail"
        }
      },
      {
        "@id":"<id>",
        "@type":"Telemetry",
        "description":"Current humidity on the device",
        "displayName":"Humidity",
        "name":"humid",
        "schema":"integer"
      },
      {
        "@id":"<id>",
        "@type":"Telemetry",
        "description":"Current PM2.5 on the device",
        "displayName":"PM2.5",
        "name":"pm25",
        "schema":"integer"
      },
      {
        "@id":"<id>",
        "@type":"Property",
        "description":"T&H Sensor Model Name",
        "displayName":"T&H Sensor Model",
        "name":"thsensormodel",
        "schema":"string"
      },
      {
        "@id":"<id>",
        "@type":"Property",
        "description":"PM2.5 Sensor Model Name",
        "displayName":"PM2.5 Sensor Model",
        "name":"pm25sensormodel",
        "schema":"string"
      }
    ]
  }
},
{
  "@id":"<id>",
  "@type":"InterfaceInstance",
  "name":"urn_azureiot_DeviceManagement_DeviceInformation",
  "schema":{
    "@id":"<id>",
    "@type":"Interface",
    "displayName":"Device information",
    "contents":[
      {
        "@id":"<id>",
        "@type":"Property",
        "comment":"Total available storage on the device in kilobytes. Ex. 20480000 kilobytes.",
        "displayName":"Total storage",
        "name":"totalStorage",
        "displayUnit":"kilobytes",
```

```
            "schema":"long"
          },
          {
            "@id":"<id>",
            "@type":"Property",
            "comment":"Total available memory on the device in kilobytes. Ex. 256000 kilobytes.",
            "displayName":"Total memory",
            "name":"totalMemory",
            "displayUnit":"kilobytes",
            "schema":"long"
          }
        ]
      }
    }
  ],
  "displayName":"AAEONAirbox52"
},
"solutionModel":{
  "@id":"<id>",
  "@type":"SolutionModel",
  "cloudProperties":[
    {
      "@id":"<id>",
      "@type":"CloudProperty",
      "displayName":"Color",
      "name":"Color",
      "schema":"string",
      "valueDetail":{
        "@id":"<id>",
        "@type":"ValueDetail/StringValueDetail"
      },
      "visualizationDetail":{
        "@id":"<id>",
        "@type":"VisualizationDetail"
      }
    }
  ]
}
}
```

Next steps

Now that you know how to export your data to Azure Event Hubs, Azure Service Bus, and Azure Blob storage, continue to the next step:

How to run custom analytics with Databricks

Create webhook actions on rules in Azure IoT Central

4/23/2020 • 2 minutes to read • Edit Online

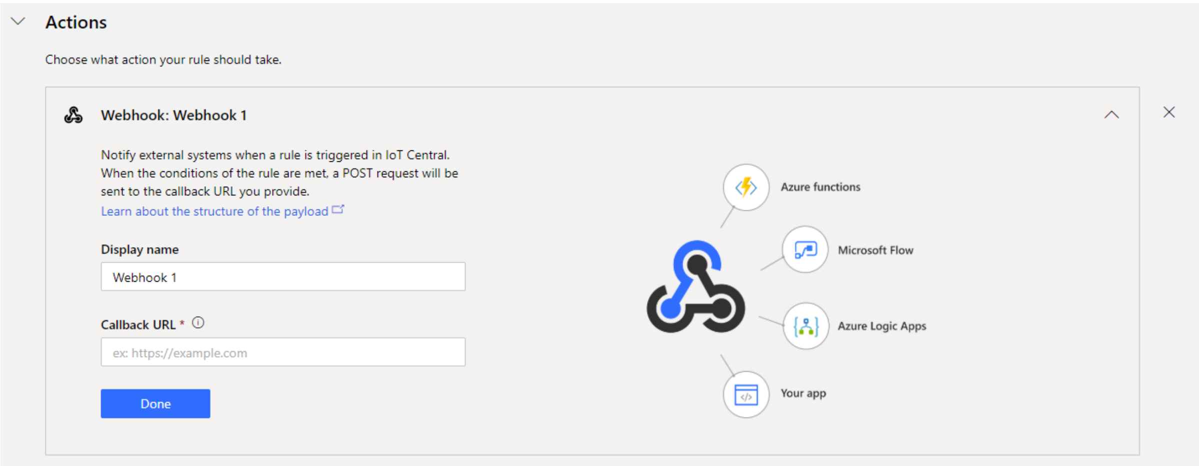
This topic applies to builders and administrators.

Webhooks enable you to connect your IoT Central app to other applications and services for remote monitoring and notifications. Webhooks automatically notify other applications and services you connect whenever a rule is triggered in your IoT Central app. Your IoT Central app sends a POST request to the other application's HTTP endpoint whenever a rule is triggered. The payload contains device details and rule trigger details.

Set up the webhook

In this example, you connect to RequestBin to get notified when rules fire using webhooks.

1. Open RequestBin.
2. Create a new RequestBin and copy the **Bin URL**.
3. Create a telemetry rule. Save the rule and add a new action.



4. Choose the webhook action and provide a display name and paste the Bin URL as the Callback URL.
5. Save the rule.

Now when the rule is triggered, you see a new request appear in RequestBin.

Payload

When a rule is triggered, an HTTP POST request is made to the callback URL containing a json payload with the telemetry, device, rule, and application details. The payload could look like the following:

```json
{
    "timestamp": "2020-04-06T00:20:15.06Z",
    "action": {
        "id": "<id>",
        "type": "WebhookAction",
        "rules": [
            "<rule_id>"
        ],
        "displayName": "Webhook 1",
        "url": "<callback_url>"
    },
    "application": {
        "id": "<application_id>",
        "displayName": "Contoso",
        "subdomain": "contoso",
        "host": "contoso.azureiotcentral.com"
    },
    "device": {
        "id": "<device_id>",
        "etag": "<etag>",
        "displayName": "MXChip IoT DevKit - 1yl6vvhax6c",
        "instanceOf": "<device_template_id>",
        "simulated": true,
        "provisioned": true,
        "approved": true,
        "cloudProperties": {
            "City": {
                "value": "Seattle"
            }
        },
        "properties": {
            "deviceinfo": {
                "firmwareVersion": {
                    "value": "1.0.0"
                }
            }
        },
        "telemetry": {
            "<interface_instance_name>": {
                "humidity": {
                    "value": 47.33228889360127
                }
            }
        }
    },
    "rule": {
        "id": "<rule_id>",
        "displayName": "Humidity monitor"
    }
}
```


If the rule monitors aggregated telemetry over a period of time, the payload will contain a different telemetry section.


```json
{
    "telemetry": {
        "<interface_instance_name>": {
            "Humidity": {
                "avg": 39.5
            }
        }
    }
}
```

Data format change notice

If you have one or more webhooks created and saved before **3 April 2020**, you will need to delete the webhook and create a new webhook. This is because older webhooks use an older payload format that will be deprecated in the future.

Webhook payload (format deprecated as of 3 April 2020)


```
{
    "id": "<id>",
    "displayName": "Webhook 1",
    "timestamp": "2019-10-24T18:27:13.538Z",
    "rule": {
        "id": "<id>",
        "displayName": "High temp alert",
        "enabled": true
    },
    "device": {
        "id": "mx1",
        "displayName": "MXChip IoT DevKit - mx1",
        "instanceOf": "<device-template-id>",
        "simulated": true,
        "provisioned": true,
        "approved": true
    },
    "data": [{
        "@id": "<id>",
        "@type": ["Telemetry"],
        "name": "temperature",
        "displayName": "Temperature",
        "value": 66.27310467496761,
        "interfaceInstanceName": "sensors"
    }],
    "application": {
        "id": "<id>",
        "displayName": "x - Store Analytics Checkout---PnP",
        "subdomain": "<subdomain>",
        "host": "<host>"
    }
}
```


Known limitations

Currently there is no programmatic way of subscribing/unsubscribing from these webhooks through an API.

If you have ideas for how to improve this feature, post your suggestions to our User voice forum.

Next steps

Now that you've learned how to set up and use webhooks, the suggested next step is to explore configuring Azure Monitor Action Groups.

Use workflows to integrate your Azure IoT Central application with other cloud services

7/21/2020 • 6 minutes to read • Edit Online

This article applies to solution builders.

You can create rules in IoT Central that trigger actions, such as sending an email, in response to telemetry-based conditions, such as device temperature exceeding a threshold.

The Azure IoT Central V3 connector for Power Automate and Azure Logic Apps lets you create more advanced rules to automate operations in IoT Central:

- When a rule fires in your Azure IoT Central app, it can trigger a workflow in Power Automate or Azure Logic Apps. These workflows can run actions in other cloud services, such as Office 365, or a third-party service.
- An event in another cloud service, such as Office 365, can trigger a workflow in Power Automate or Azure Logic Apps. These workflows can run actions or retrieve data from your IoT Central application.

Prerequisites

To complete the steps in this how-to guide, you need an active Azure subscription. If you don't have an Azure subscription, create a free account before you begin.

Setting up the solution requires a version 3 IoT Central application. To learn how to check your application version, see About your application. To learn how to create an IoT Central application, see Create an Azure IoT Central application.

> **NOTE**
> If you're using a version 2 IoT Central application, see Build workflows with the IoT Central connector in Azure Logic Apps on the previous versions documentation site and use the Azure IoT Central V2 connector

Trigger a workflow from a rule

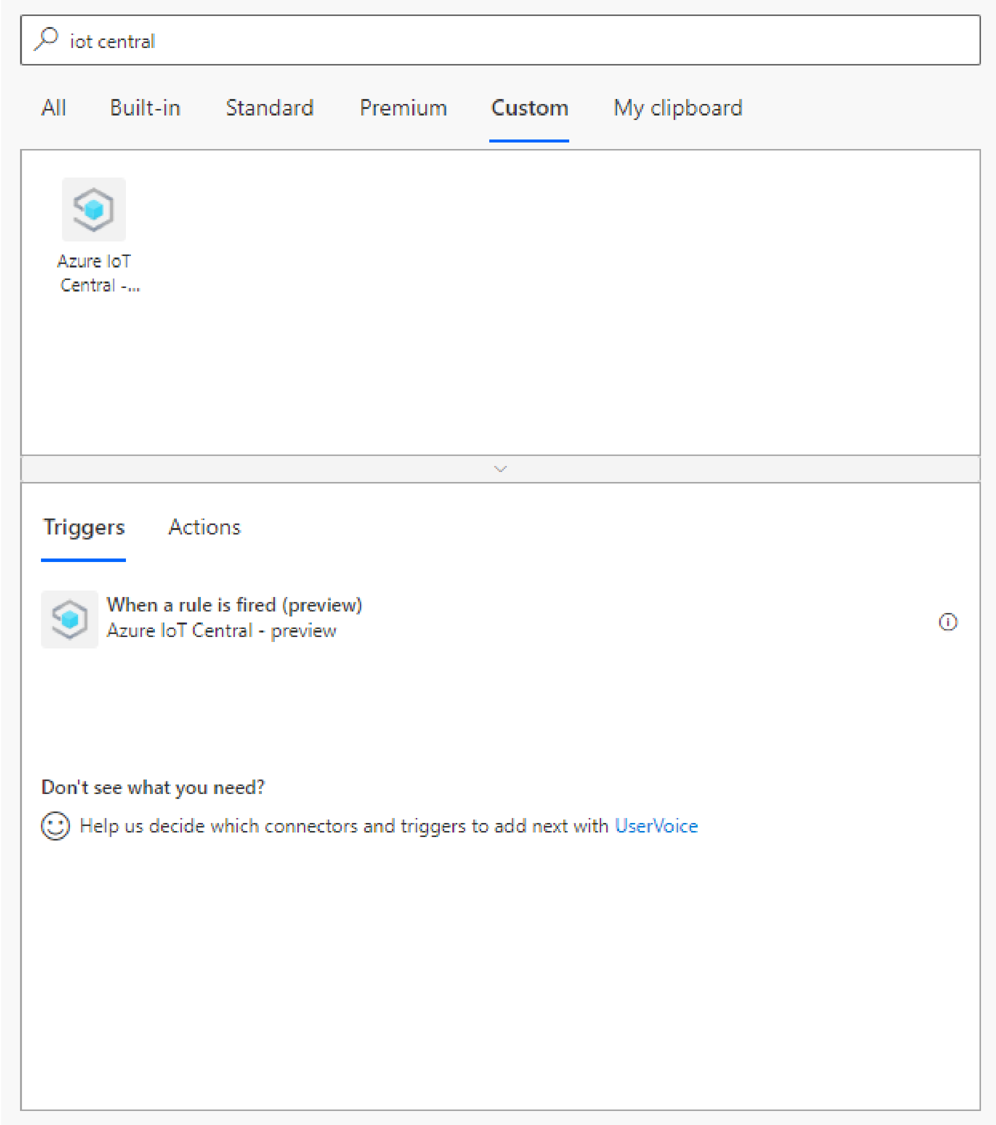
Before you can trigger a workflow in Power Automate or Azure Logic Apps, you need a rule in your IoT Central application. To learn more, see Configure rules and actions in Azure IoT Central.

To add the **Azure IoT Central V3 - preview** connector as a trigger in Power Automate:

1. In Power Automate, select **+ Create**, select the **Custom** tab.
2. Search for *IoT Central*, and select the **Azure IoT Central V3 - preview** connector.
3. In the list of triggers, select **When a rule is fired (preview)**.
4. In the **When a rule is fired** step, select your IoT Central application and the rule you're using.

To add the **Azure IoT Central V3 - preview** connector as a trigger in Azure Logic Apps:

1. In **Logic Apps Designer**, select the **Blank Logic App** template.
2. In the designer, select the **Custom** tab.
3. Search for *IoT Central*, and select the **Azure IoT Central V3 - preview** connector.
4. In the list of triggers, select **When a rule is fired (preview)**.
5. In the **When a rule is fired** step, select your IoT Central application and the rule you're using.



You can now add more steps to your workflow to build out your integration scenario.

Run an action

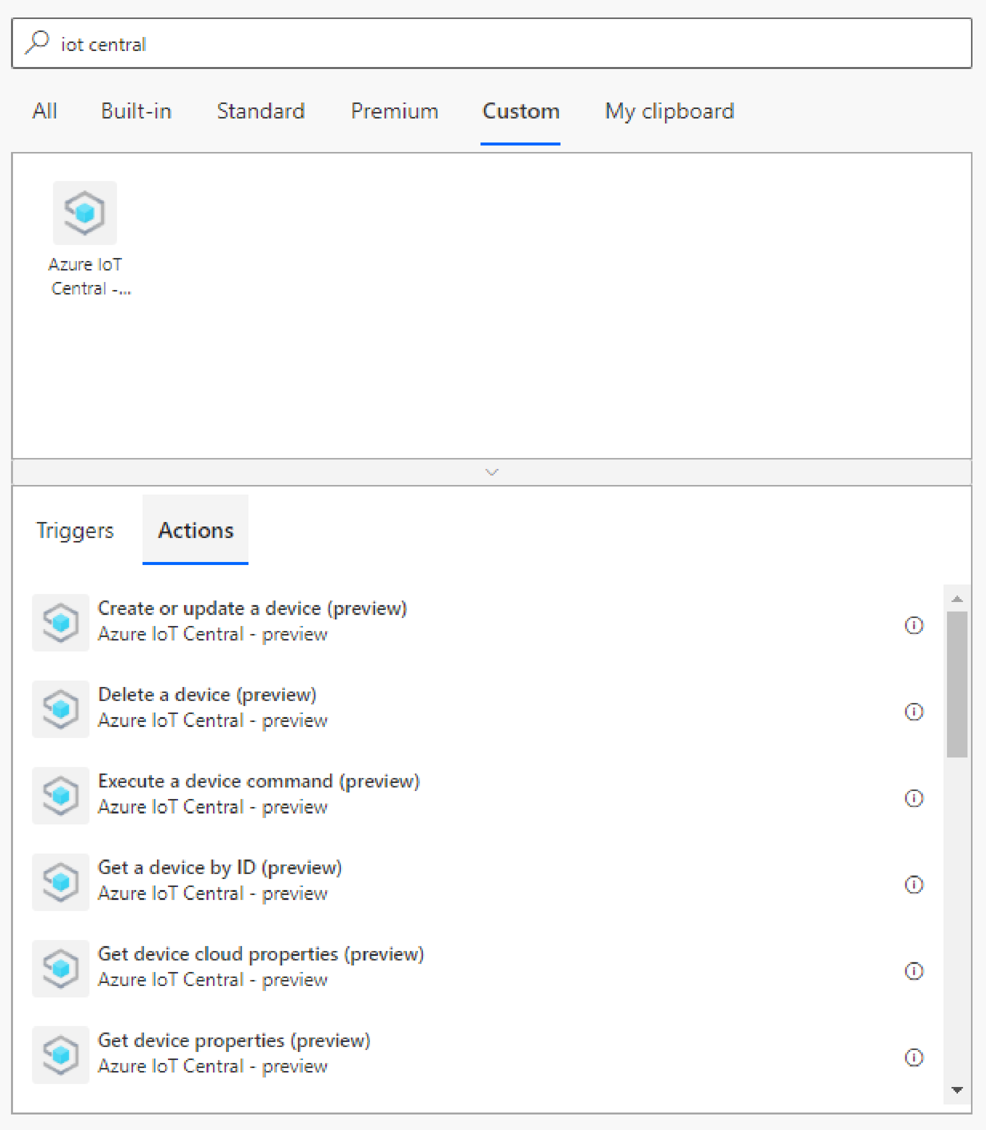
You can run actions in an IoT Central application from Power Automate and Azure Logic Apps workflows. First, create your workflow and use a connector to define a trigger to start the workflow. Then use the **Azure IoT Central V3 - preview** connector as an action.

To add the **Azure IoT Central V3 - preview** connector as an action in Power Automate:

1. In Power Automate, in the **Choose an action** panel, select the **Custom** tab.
2. Search for *IoT Central* and select the **Azure IoT Central V3 - preview** connector.
3. In the list of actions, select the IoT Central action you want to use.
4. In the action step, complete the configuration for the action you chose. Then select **Save**.

To add the **Azure IoT Central V3- preview** connector as an action in Azure Logic Apps:

1. In **Logic Apps Designer**, in the **Choose an action** panel, select the **Custom** tab.
2. Search for *IoT Central*, and select the **Azure IoT Central V3- preview** connector.
3. In the list of actions, select the IoT Central action you want to use.
4. In the action step, complete the configuration for the action you chose. Then select **Save**.



List of actions

The following list shows all the available IoT Central actions in the **Azure IoT Central V3 - preview** connector and their configuration options. Many of the fields can have dynamically generated content. For example, a previous step could determine the device ID that the current step acts on.

Create or update a device

Use this action to create or update a device in your IoT Central application.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to create or update.
Approved	Choose whether the device has been approved to connect to IoT Central.
Device Description	A detailed description of the device.
Device Name	The display name of the device.

FIELD	DESCRIPTION
Device Template	Choose from the list of device templates in your IoT Central application.
Simulated	Choose whether the device is simulated.

Delete a device

Use this action to delete a device from your IoT Central application.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.

Execute a device command

Use this action to execute a command defined in one of the device's interfaces.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.
Device Component	The interface in the device template that contains the command.
Device Command	Choose one of the commands on the selected interface.
Device Template	Choose from the list of device templates in your IoT Central application.
Device Command Request Payload	If the command requires a request payload, add it here.

> **NOTE**
> You can't choose a device component until you've chosen a device template.

Get a device by ID

Use this action to retrieve the device's details.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.

You can use the returned details in the dynamic expressions in other actions. The device details returned include: `Approved`, `body`, `Device Description`, `Device Name`, `Device Template`, `Provisioned`, and `Simulated`.

Get device cloud properties

Use this action to retrieve the cloud property values for a specific device.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.
Device Template	Choose from the list of device templates in your IoT Central application.

You can use the returned cloud property values in the dynamic expressions in other actions.

Get device properties

Use this action to retrieve the property values for a specific device.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.
Device Template	Choose from the list of device templates in your IoT Central application.

You can use the returned property values in the dynamic expressions in other actions.

Get device telemetry value

Use this action to retrieve the telemetry values for a specific device.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.
Device Template	Choose from the list of device templates in your IoT Central application.

You can use the returned telemetry values in the dynamic expressions in other actions.

Update device cloud properties

Use this action to update cloud property values for a specific device.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.
Device Template	Choose from the list of device templates in your IoT Central application.

FIELD	DESCRIPTION
Cloud properties	After you choose a device template, a field is added for each cloud property defined in the template.

Update device properties

Use this action to update writeable property values for a specific device.

FIELD	DESCRIPTION
Application	Choose from your list of IoT Central applications.
Device	The unique ID of the device to delete.
Device Template	Choose from the list of device templates in your IoT Central application.
Writeable properties	After you choose a device template, a field is added for each writeable property defined in the template.

Next steps

Now that you've learned how to create an advanced rule in your Azure IoT Central application, you can learn how to Analyze device data in your Azure IoT Central application

Group multiple actions to run from one or more rules

3/24/2020 • 2 minutes to read • Edit Online

This article applies to builders and administrators.

In Azure IoT Central, you create rules to run actions when a condition is met. Rules are based on device telemetry or events. For example, you can notify an operator when the temperature of a device exceeds a threshold. This article describes how to use Azure Monitor *action groups* to attach multiple actions to an IoT Central rule. You can attach an action group to multiple rules. An action group is a collection of notification preferences defined by the owner of an Azure subscription.

Prerequisites

- An application created using a standard pricing plan
- An Azure account and subscription to create and manage Azure Monitor action groups

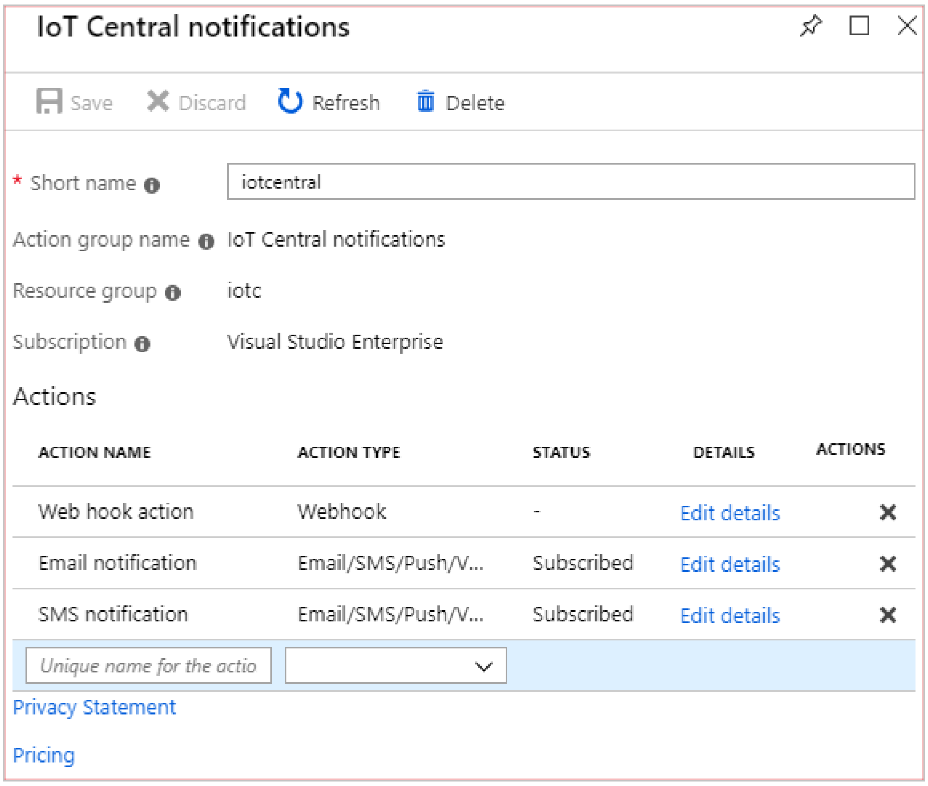
Create action groups

You can create and manage action groups in the Azure portal or with an Azure Resource Manager template.

An action group can:

- Send notifications such as an email, an SMS, or make a voice call.
- Run an action such as calling a webhook.

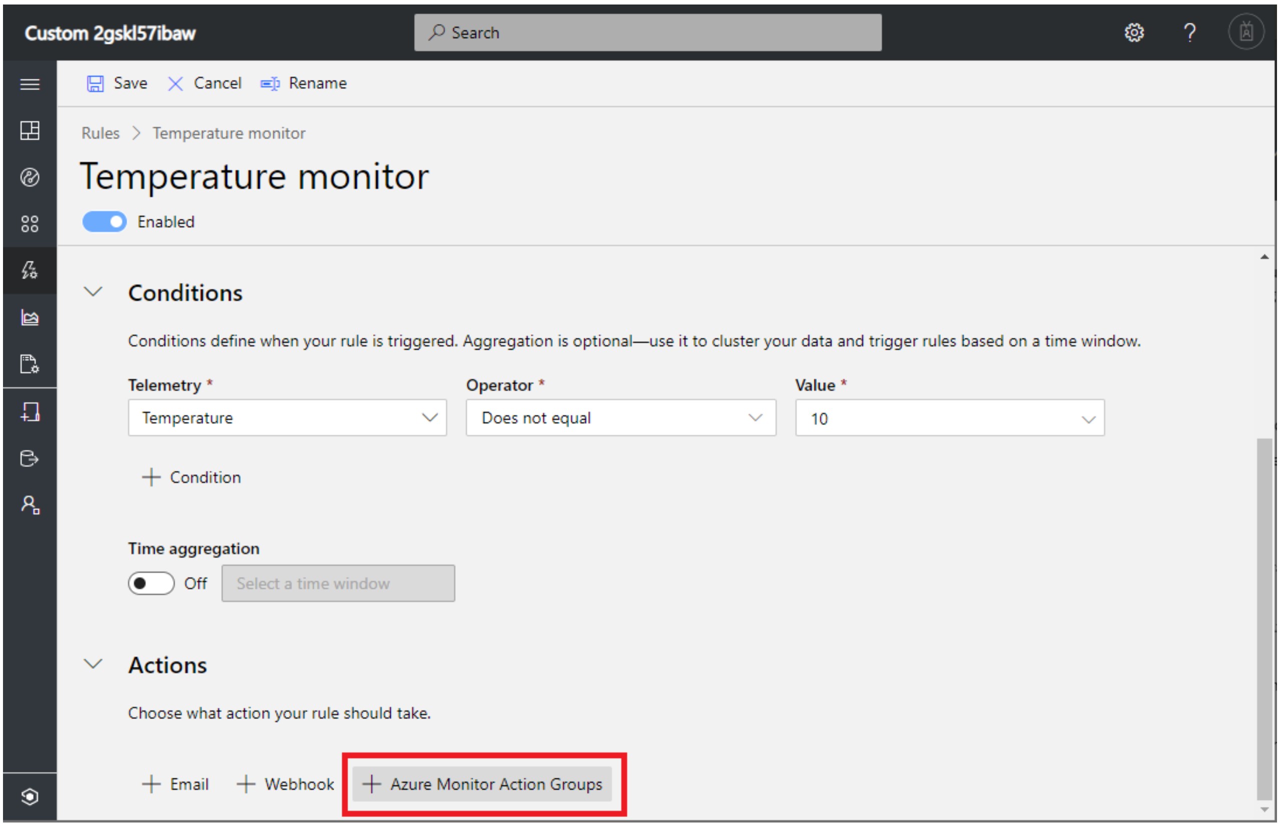
The following screenshot shows an action group that sends email and SMS notifications and calls a webhook:



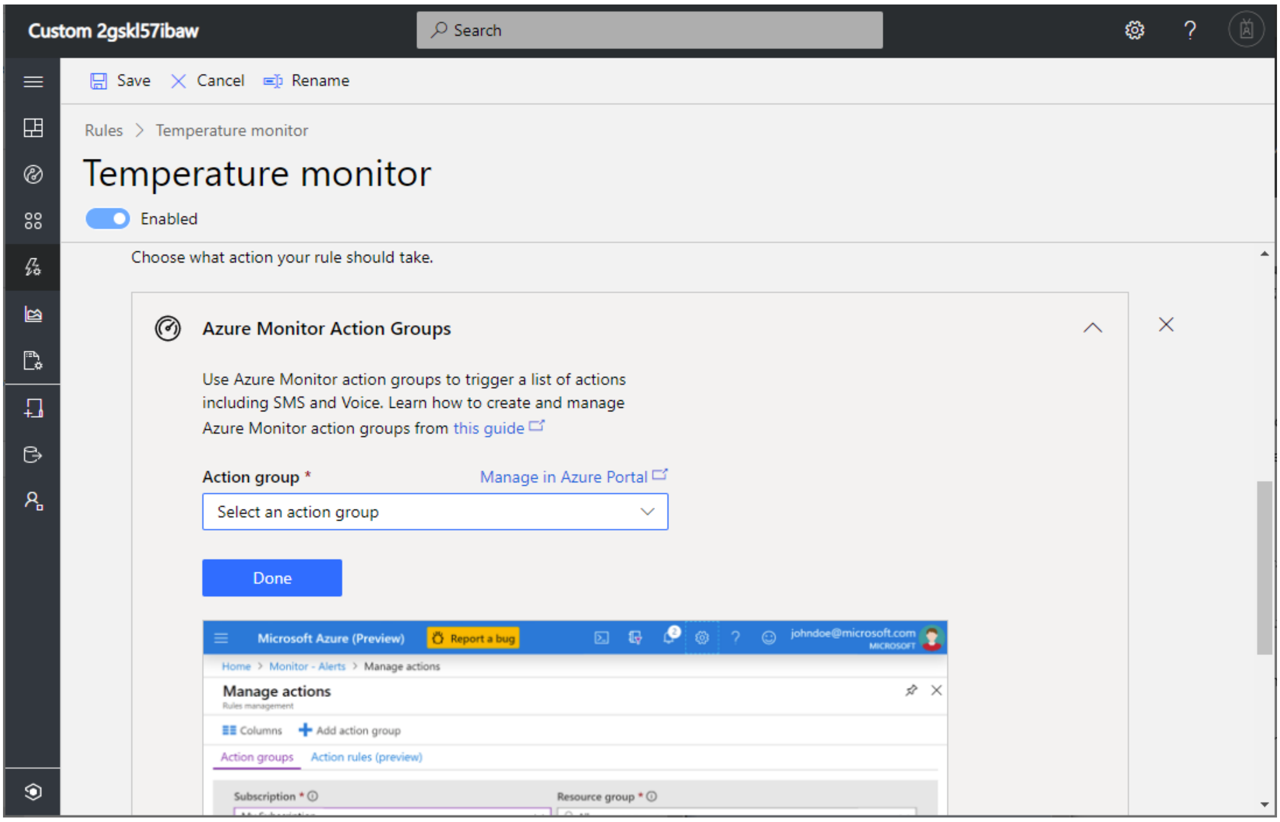
To use an action group in an IoT Central rule, the action group must be in the same Azure subscription as the IoT Central application.

Use an action group

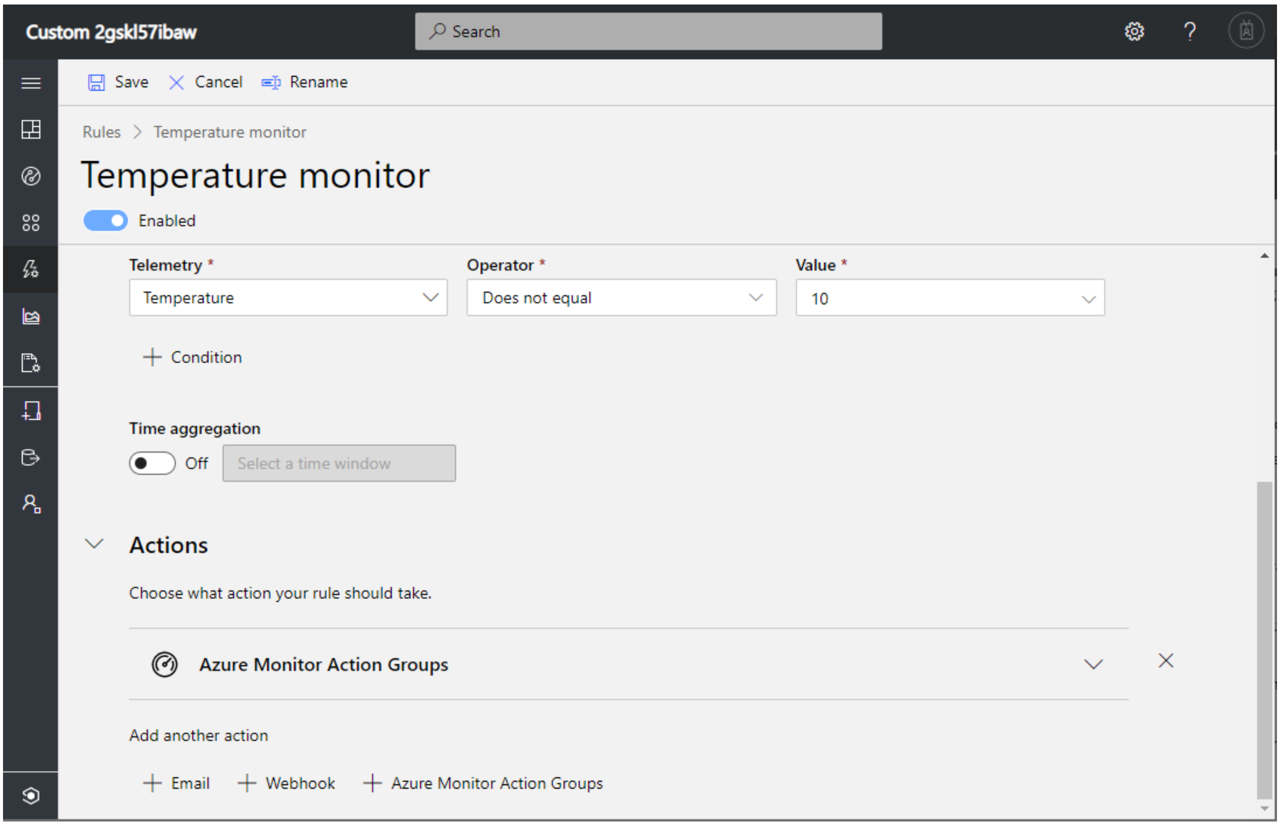
To use an action group in your IoT Central application, first create a rule. When you add an action to the rule, select **Azure Monitor Action Groups**:



Choose an action group from your Azure subscription:



Select **Save**. The action group now appears in the list of actions to run when the rule is triggered:



The following table summarizes the information sent to the supported action types:

ACTION TYPE	OUTPUT FORMAT
Email	Standard IoT Central email template
SMS	Azure IoT Central alert: ${applicationName} - "${ruleName}" triggered on "${deviceName}" at ${triggerDate} ${triggerTime}
Voice	Azure I.O.T Central alert: rule "${ruleName}" triggered on device "${deviceName}" at ${triggerDate} ${triggerTime}, in application ${applicationName}
Webhook	{ "schemaId" : "AzureIoTCentralRuleWebhook", "data": {regular webhook payload}}

The following text is an example SMS message from an action group:

```
iotcentral: Azure IoT Central alert: Contoso - "Low pressure alert" triggered on "Motion sensor 2" at March 20,
2019 10:12 UTC
```

Next steps

Now that you've learned how to use action groups with rules, the suggested next step is to learn how to manage your devices.

Extend Azure IoT Central with custom rules using Stream Analytics, Azure Functions, and SendGrid

3/24/2020 • 8 minutes to read • Edit Online

This how-to guide shows you, as a solution developer, how to extend your IoT Central application with custom rules and notifications. The example shows sending a notification to an operator when a device stops sending telemetry. The solution uses an Azure Stream Analytics query to detect when a device has stopped sending telemetry. The Stream Analytics job uses Azure Functions to send notification emails using SendGrid.

This how-to guide shows you how to extend IoT Central beyond what it can already do with the built-in rules and actions.

In this how-to guide, you learn how to:

- Stream telemetry from an IoT Central application using *continuous data export*.
- Create a Stream Analytics query that detects when a device has stopped sending data.
- Send an email notification using the Azure Functions and SendGrid services.

Prerequisites

To complete the steps in this how-to guide, you need an active Azure subscription.

If you don't have an Azure subscription, create a free account before you begin.

IoT Central application

Create an IoT Central application on the Azure IoT Central application manager website with the following settings:

SETTING	VALUE
Pricing plan	Standard
Application template	In-store analytics – condition monitoring
Application name	Accept the default or choose your own name
URL	Accept the default or choose your own unique URL prefix
Directory	Your Azure Active Directory tenant
Azure subscription	Your Azure subscription
Region	Your nearest region

The examples and screenshots in this article use the **United States** region. Choose a location close to you and make sure you create all your resources in the same region.

This application template includes two simulated thermostat devices that send telemetry.

Resource group

Use the Azure portal to create a resource group called **DetectStoppedDevices** to contain the other resources you create. Create your Azure resources in the same location as your IoT Central application.

Event Hubs namespace

Use the Azure portal to create an Event Hubs namespace with the following settings:

SETTING	VALUE
Name	Choose your namespace name
Pricing tier	Basic
Subscription	Your subscription
Resource group	DetectStoppedDevices
Location	East US
Throughput Units	1

Stream Analytics job

Use the Azure portal to create a Stream Analytics job with the following settings:

SETTING	VALUE
Name	Choose your job name
Subscription	Your subscription
Resource group	DetectStoppedDevices
Location	East US
Hosting environment	Cloud
Streaming units	3

Function app

Use the Azure portal to create a function app with the following settings:

SETTING	VALUE
App name	Choose your function app name
Subscription	Your subscription
Resource group	DetectStoppedDevices
OS	Windows
Hosting Plan	Consumption Plan
Location	East US
Runtime Stack	.NET

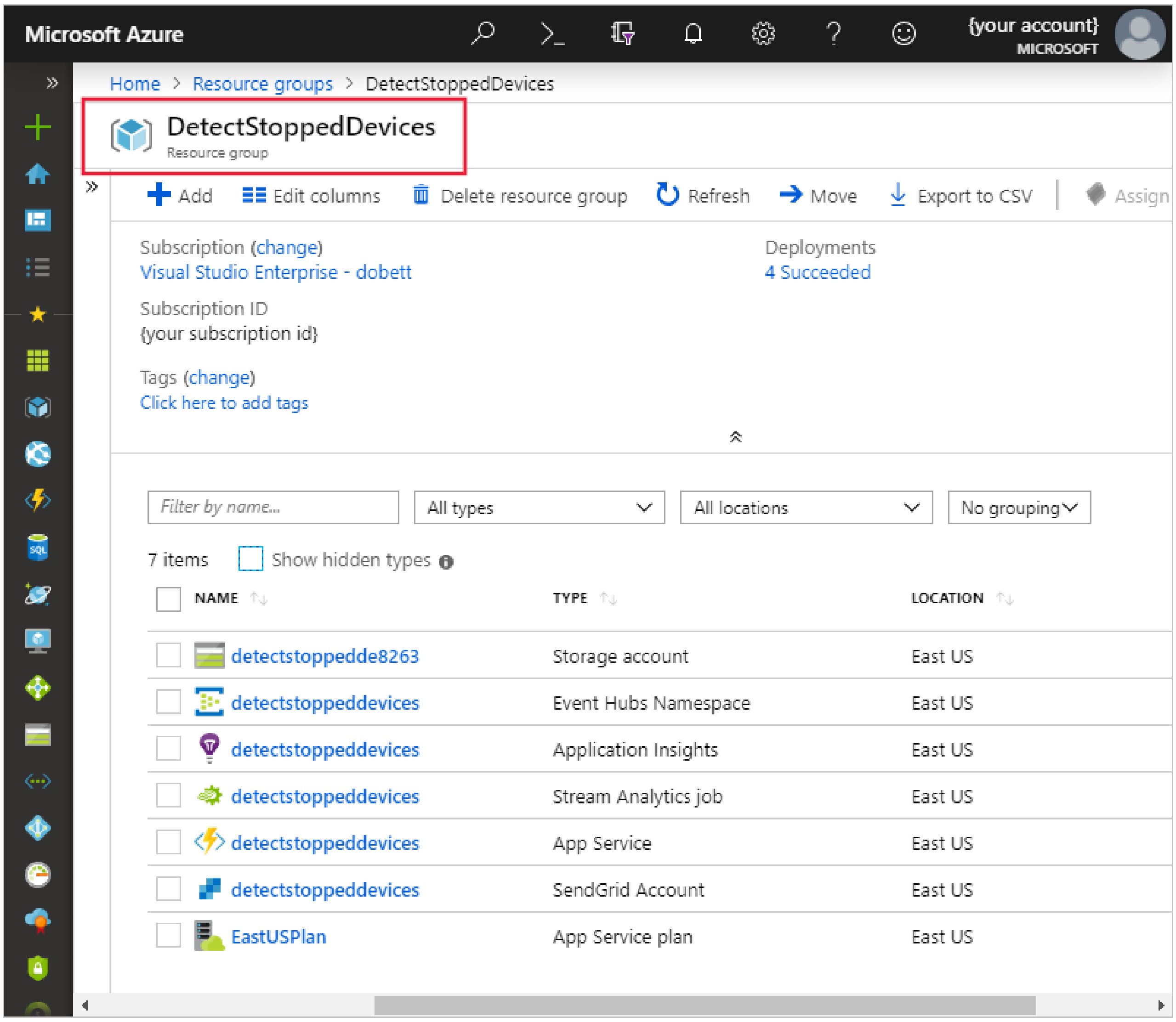
SETTING	VALUE
Storage	Create new

SendGrid account

Use the Azure portal to create a SendGrid account with the following settings:

SETTING	VALUE
Name	Choose your SendGrid account name
Password	Create a password
Subscription	Your subscription
Resource group	DetectStoppedDevices
Pricing tier	F1 Free
Contact information	Fill out required information

When you've created all the required resources, your **DetectStoppedDevices** resource group looks like the following screenshot:

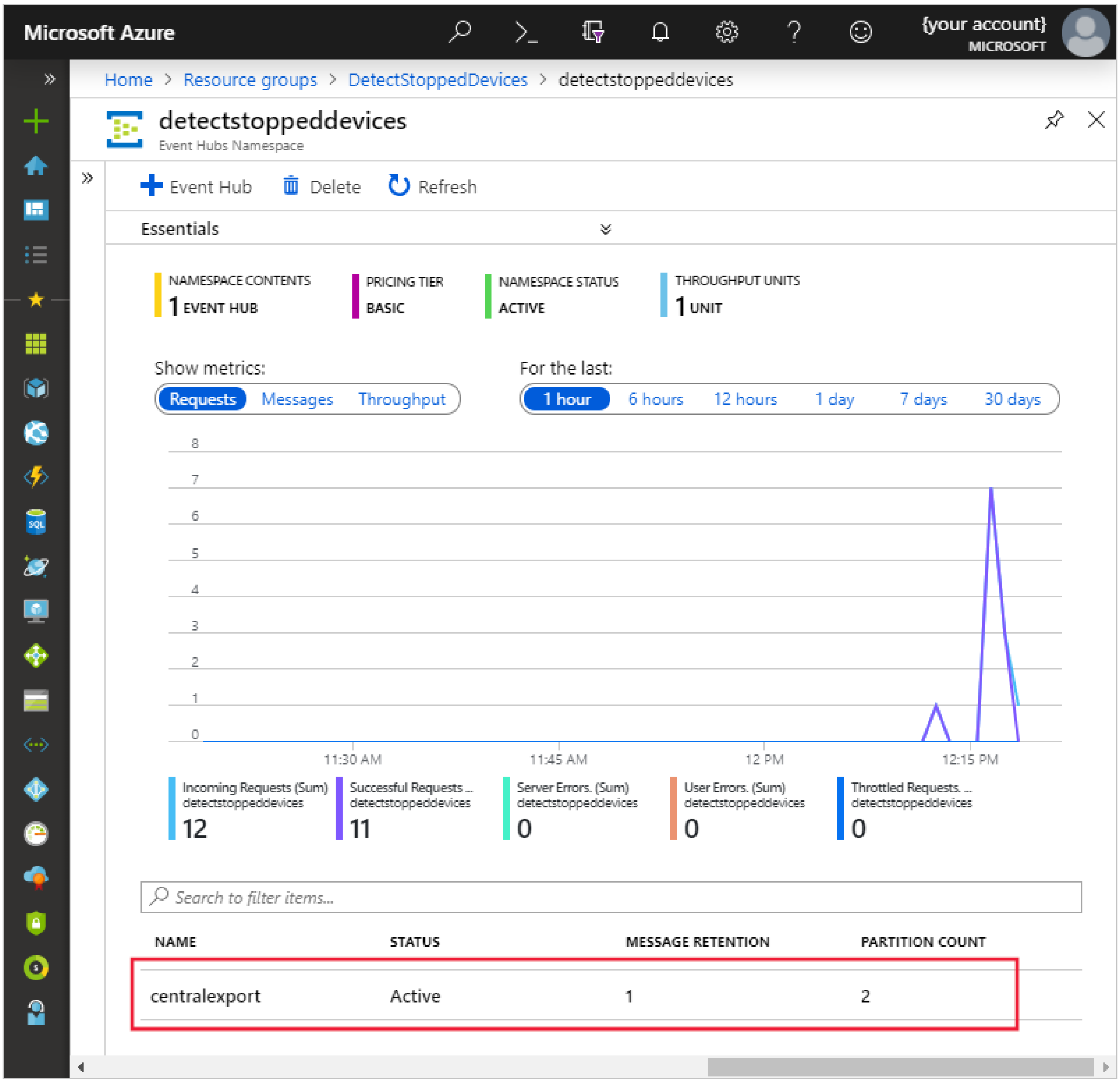


Create an event hub

You can configure an IoT Central application to continuously export telemetry to an event hub. In this section, you create an event hub to receive telemetry from your IoT Central application. The event hub delivers the telemetry to your Stream Analytics job for processing.

1. In the Azure portal, navigate to your Event Hubs namespace and select **+ Event Hub**.
2. Name your event hub **centralexport**, and select **Create**.

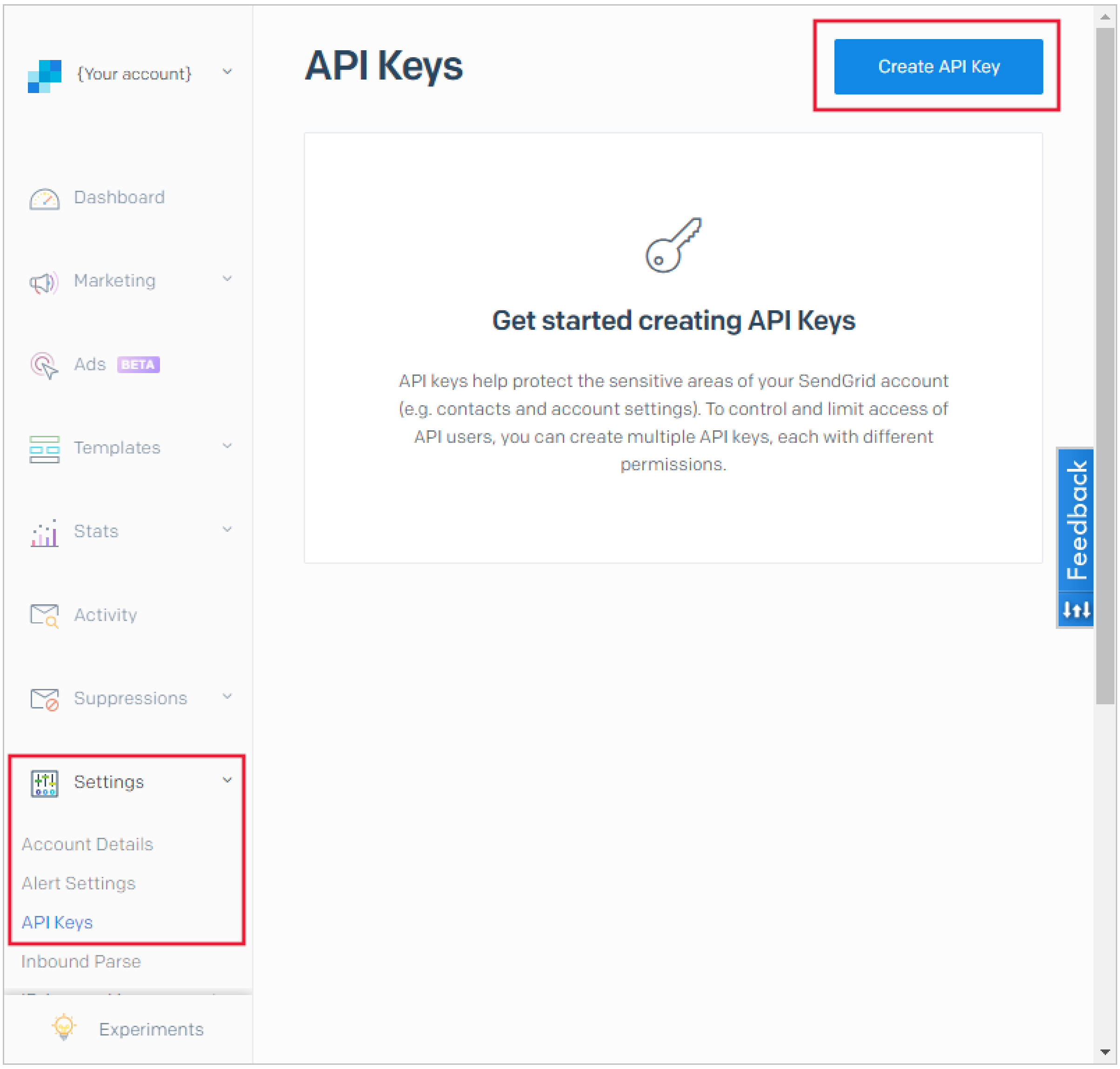
Your Event Hubs namespace looks like the following screenshot:



Get SendGrid API key

Your function app needs a SendGrid API key to send email messages. To create a SendGrid API key:

1. In the Azure portal, navigate to your SendGrid account. Then choose **Manage** to access your SendGrid account.
2. In your SendGrid account, choose **Settings**, then **API Keys**. Choose **Create API Key**:



3. On the **Create API Key** page, create a key named **AzureFunctionAccess** with **Full Access** permissions.

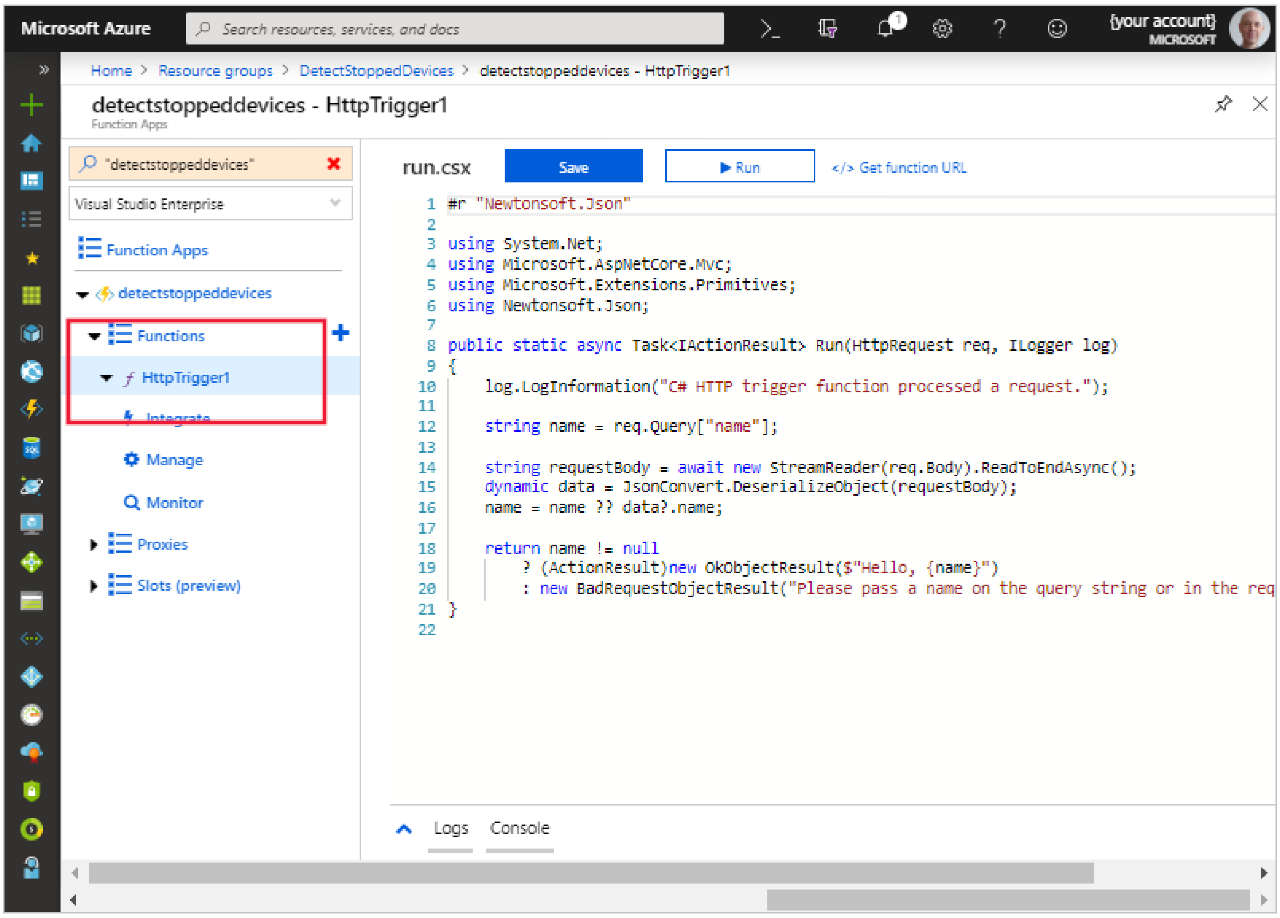
4. Make a note of the API Key, you need it when you configure your function app.

Define the function

This solution uses an Azure Functions app to send an email notification when the Stream Analytics job detects a stopped device. To create your function app:

1. In the Azure portal, navigate to the **App Service** instance in the **DetectStoppedDevices** resource group.
2. Select **+** to create a new function.
3. On the **CHOOSE A DEVELOPMENT ENVIRONMENT** page, choose **In-portal** and then select **Continue**.
4. On the **CREATE A FUNCTION** page, choose **Webhook + API** and then select **Create**.

The portal creates a default function called **HttpTrigger1**:

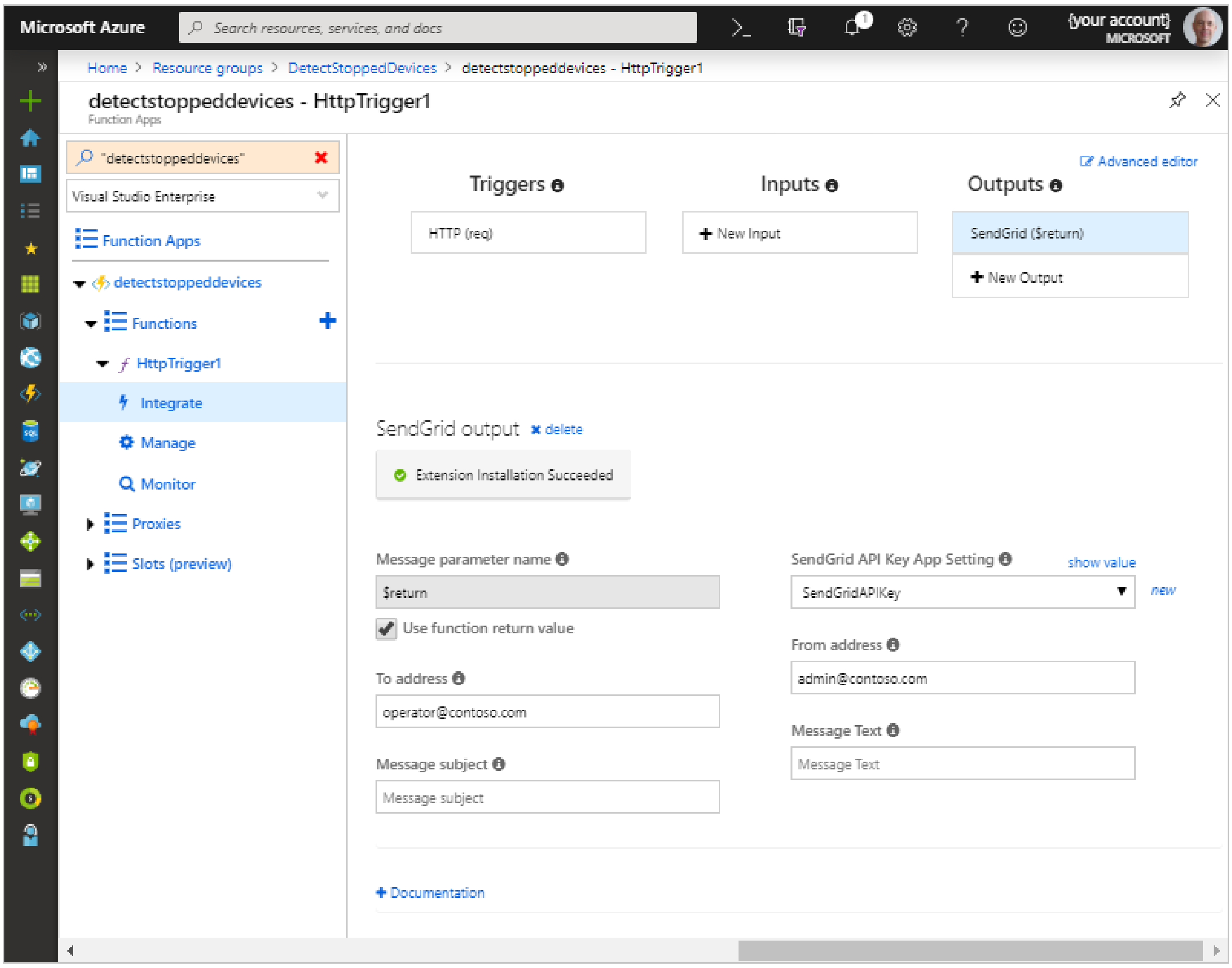


Configure function bindings

To send emails with SendGrid, you need to configure the bindings for your function as follows:

1. Select **Integrate**, choose the output **HTTP ($return)**, and then select **delete**.
2. Choose **+ New Output**, then choose **SendGrid**, and then choose **Select**. Choose **Install** to install the SendGrid extension.
3. When the installation completes, select **Use function return value**. Add a valid **To address** to receive email notifications. Add a valid **From address** to use as the email sender.
4. Select **new** next to **SendGrid API Key App Setting**. Enter **SendGridAPIKey** as the key, and the SendGrid API key you noted previously as the value. Then select **Create**.
5. Choose **Save** to save the SendGrid bindings for your function.

The integrate settings look like the following screenshot:



Add the function code

To implement your function, add the C# code to parse the incoming HTTP request and send the emails as follows:

1. Choose the **HttpTrigger1** function in your function app and replace the C# code with the following code:

```
#r "Newtonsoft.Json"
#r "..\bin\SendGrid.dll"

using System;
using SendGrid.Helpers.Mail;
using Microsoft.Azure.WebJobs.Host;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Primitives;
using Newtonsoft.Json;

public static SendGridMessage Run(HttpRequest req, ILogger log)
{
    string requestBody = new StreamReader(req.Body).ReadToEnd();
    log.LogInformation(requestBody);
    var notifications = JsonConvert.DeserializeObject<IList<Notification>>(requestBody);

    SendGridMessage message = new SendGridMessage();
    message.Subject = "Contoso device notification";

    var content = "The following device(s) have stopped sending telemetry:<br/><br/><table><tr>
<th>Device ID</th><th>Time</th></tr>";
    foreach(var notification in notifications) {
        log.LogInformation($"No message received - Device: {notification.deviceid}, Time:
{notification.time}");
        content += $"<tr><td>{notification.deviceid}</td><td>{notification.time}</td></tr>";
    }
    content += "</table>";
    message.AddContent("text/html", content);

    return message;
}

public class Notification
{
    public string deviceid { get; set; }
    public string time { get; set; }
}
```

You may see an error message until you save the new code.

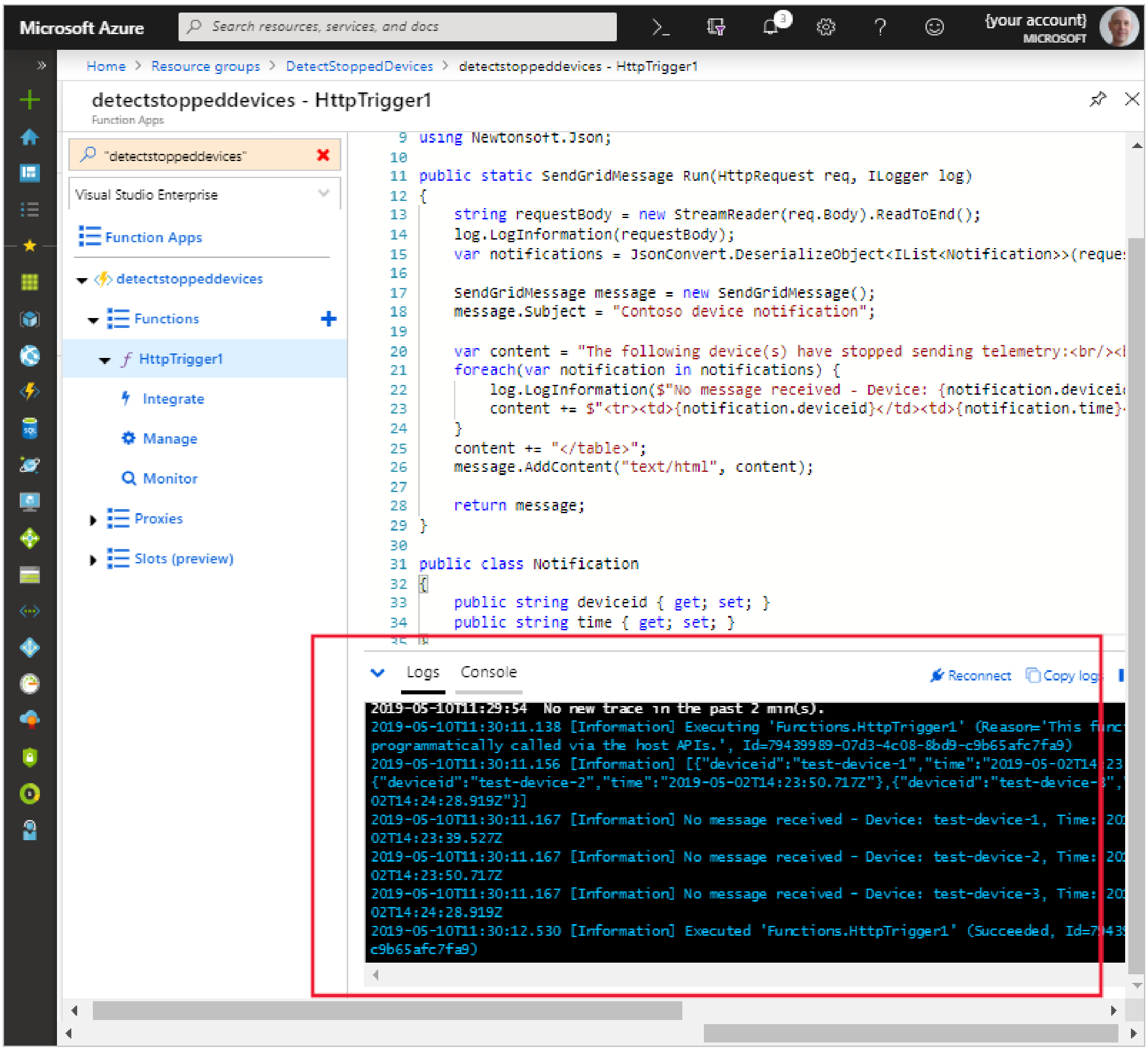
2. Select **Save** to save the function.

Test the function works

To test the function in the portal, first choose **Logs** at the bottom of the code editor. Then choose **Test** to the right of the code editor. Use the following JSON as the **Request body**:


```
[{"deviceid":"test-device-1","time":"2019-05-02T14:23:39.527Z"},{"deviceid":"test-device-2","time":"2019-05-
02T14:23:50.717Z"},{"deviceid":"test-device-3","time":"2019-05-02T14:24:28.919Z"}]
```


The function log messages appear in the **Logs** panel:



After a few minutes, the **To** email address receives an email with the following content:

```
The following device(s) have stopped sending telemetry:

Device ID    Time
test-device-1    2019-05-02T14:23:39.527Z
test-device-2    2019-05-02T14:23:50.717Z
test-device-3    2019-05-02T14:24:28.919Z
```

Add Stream Analytics query

This solution uses a Stream Analytics query to detect when a device stops sending telemetry for more than 120 seconds. The query uses the telemetry from the event hub as its input. The job sends the query results to the function app. In this section, you configure the Stream Analytics job:

1. In the Azure portal, navigate to your Stream Analytics job, under **Jobs topology** select **Inputs**, choose **+ Add stream input**, and then choose **Event Hub**.

2. Use the information in the following table to configure the input using the event hub you created previously, then choose **Save**:

SETTING	VALUE
Input alias	centraltelemetry
Subscription	Your subscription
Event Hub namespace	Your Event Hub namespace
Event Hub name	Use existing - **centralexport**

3. Under **Jobs topology**, select **Outputs**, choose **+ Add**, and then choose **Azure function**.
4. Use the information in the following table to configure the output, then choose **Save**:

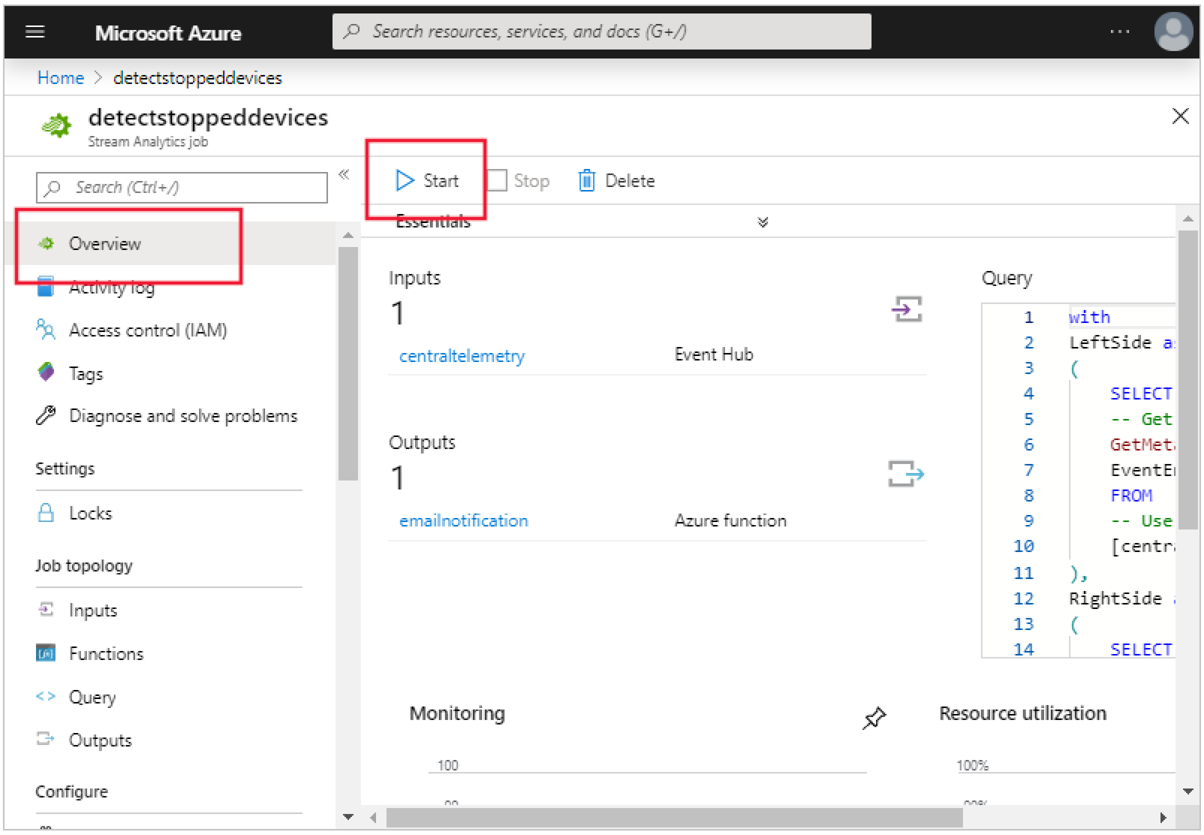
SETTING	VALUE
Output alias	emailnotification
Subscription	Your subscription
Function app	Your function app
Function	HttpTrigger1

5. Under **Jobs topology**, select **Query** and replace the existing query with the following SQL:

```
with
LeftSide as
(
    SELECT
    -- Get the device ID from the message metadata and create a column
    GetMetadataPropertyValue([centraltelemetry], '[EventHub].[IoTConnectionDeviceId]') as deviceid1,
    EventEnqueuedUtcTime AS time1
    FROM
    -- Use the event enqueued time for time-based operations
    [centraltelemetry] TIMESTAMP BY EventEnqueuedUtcTime
),
RightSide as
(
    SELECT
    -- Get the device ID from the message metadata and create a column
    GetMetadataPropertyValue([centraltelemetry], '[EventHub].[IoTConnectionDeviceId]') as deviceid2,
    EventEnqueuedUtcTime AS time2
    FROM
    -- Use the event enqueued time for time-based operations
    [centraltelemetry] TIMESTAMP BY EventEnqueuedUtcTime
)

SELECT
    LeftSide.deviceid1 as deviceid,
    LeftSide.time1 as time
INTO
    [emailnotification]
FROM
    LeftSide
    LEFT OUTER JOIN
    RightSide
    ON
    LeftSide.deviceid1=RightSide.deviceid2 AND DATEDIFF(second,LeftSide,RightSide) BETWEEN 1 AND 120
    where
    -- Find records where a device didn't send a message 120 seconds
    RightSide.deviceid2 is NULL
```

6. Select **Save**.

7. To start the Stream Analytics job, choose **Overview**, then **Start**, then **Now**, and then **Start**:

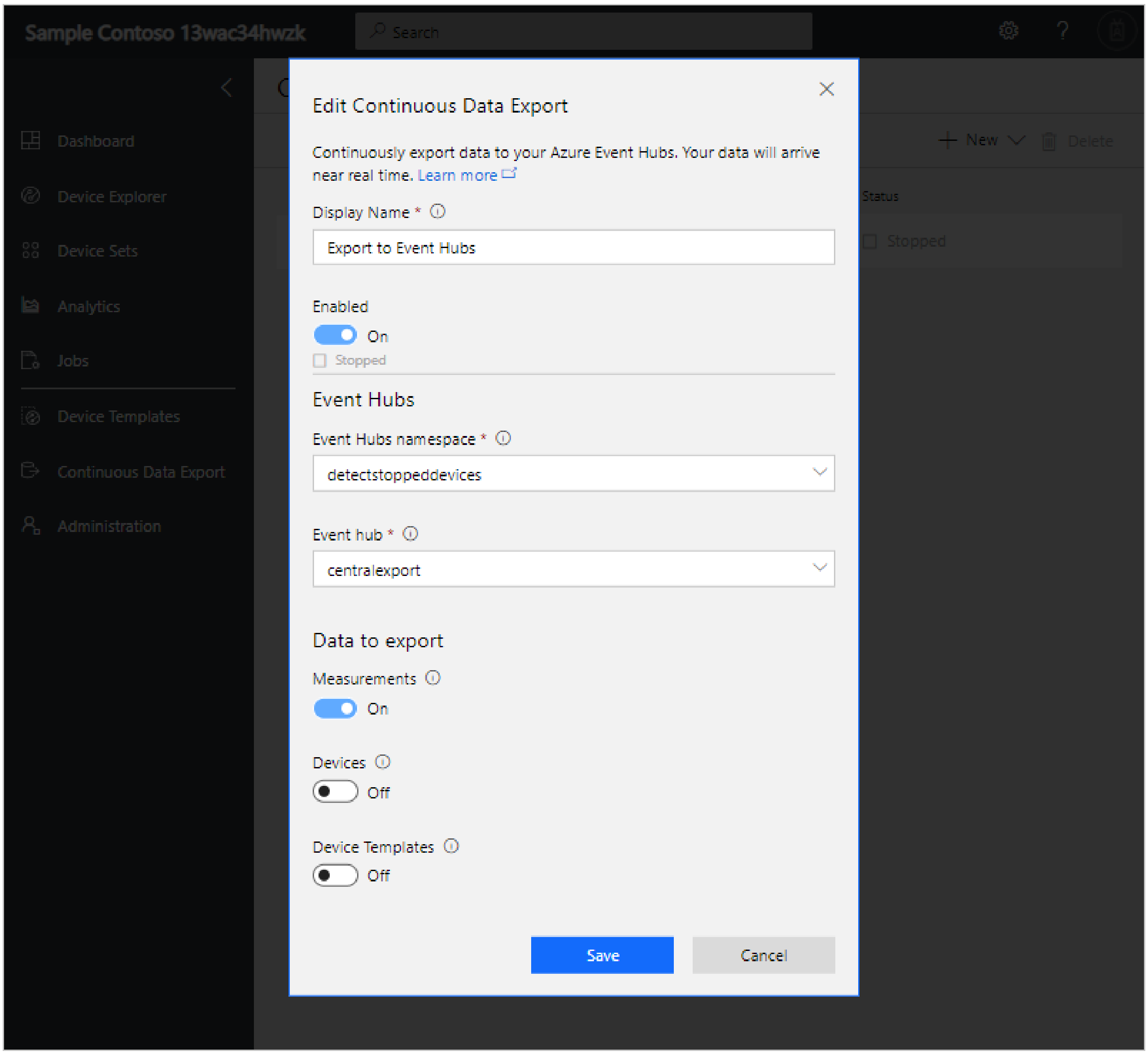


Configure export in IoT Central

On the Azure IoT Central application manager website, navigate to the IoT Central application you created from the Contoso template. In this section, you configure the application to stream the telemetry from its simulated devices to your event hub. To configure the export:

1. Navigate to the **Data Export** page, select **+ New**, and then **Azure Event Hubs**.
2. Use the following settings to configure the export, then select **Save**:

SETTING	VALUE
Display Name	Export to Event Hubs
Enabled	On
Event Hubs namespace	Your Event Hubs namespace name
Event hub	centralexport
Measurements	On
Devices	Off
Device Templates	Off



Wait until the export status is **Running** before you continue.

Test

To test the solution, you can disable the continuous data export from IoT Central to simulated stopped devices:

1. In your IoT Central application, navigate to the **Data Export** page and select the **Export to Event Hubs** export configuration.
2. Set **Enabled** to **Off** and choose **Save**.
3. After at least two minutes, the **To** email address receives one or more emails that look like the following example content:

```
The following device(s) have stopped sending telemetry:

Device ID         Time
Thermostat-Zone1  2019-11-01T12:45:14.686Z
```

Tidy up

To tidy up after this how-to and avoid unnecessary costs, delete the **DetectStoppedDevices** resource group in the Azure portal.

You can delete the IoT Central application from the **Management** page within the application.

Next steps

In this how-to guide, you learned how to:

- Stream telemetry from an IoT Central application using *continuous data export*.
- Create a Stream Analytics query that detects when a device has stopped sending data.
- Send an email notification using the Azure Functions and SendGrid services.

Now that you know how to create custom rules and notifications, the suggested next step is to learn how to Extend Azure IoT Central with custom analytics.

Extend Azure IoT Central with custom analytics using Azure Databricks

7/21/2020 • 5 minutes to read • Edit Online

This how-to guide shows you, as a solution developer, how to extend your IoT Central application with custom analytics and visualizations. The example uses an Azure Databricks workspace to analyze the IoT Central telemetry stream and to generate visualizations such as box plots.

This how-to guide shows you how to extend IoT Central beyond what it can already do with the built-in analytics tools.

In this how-to guide, you learn how to:

- Stream telemetry from an IoT Central application using *continuous data export*.
- Create an Azure Databricks environment to analyze and plot device telemetry.

Prerequisites

To complete the steps in this how-to guide, you need an active Azure subscription.

If you don't have an Azure subscription, create a free account before you begin.

IoT Central application

Create an IoT Central application on the Azure IoT Central application manager website with the following settings:

SETTING	VALUE
Pricing plan	Standard
Application template	In-store analytics – condition monitoring
Application name	Accept the default or choose your own name
URL	Accept the default or choose your own unique URL prefix
Directory	Your Azure Active Directory tenant
Azure subscription	Your Azure subscription
Region	Your nearest region

The examples and screenshots in this article use the **United States** region. Choose a location close to you and make sure you create all your resources in the same region.

This application template includes two simulated thermostat devices that send telemetry.

Resource group

Use the Azure portal to create a resource group called **IoTCentralAnalysis** to contain the other resources you create. Create your Azure resources in the same location as your IoT Central application.

Event Hubs namespace

Use the Azure portal to create an Event Hubs namespace with the following settings:

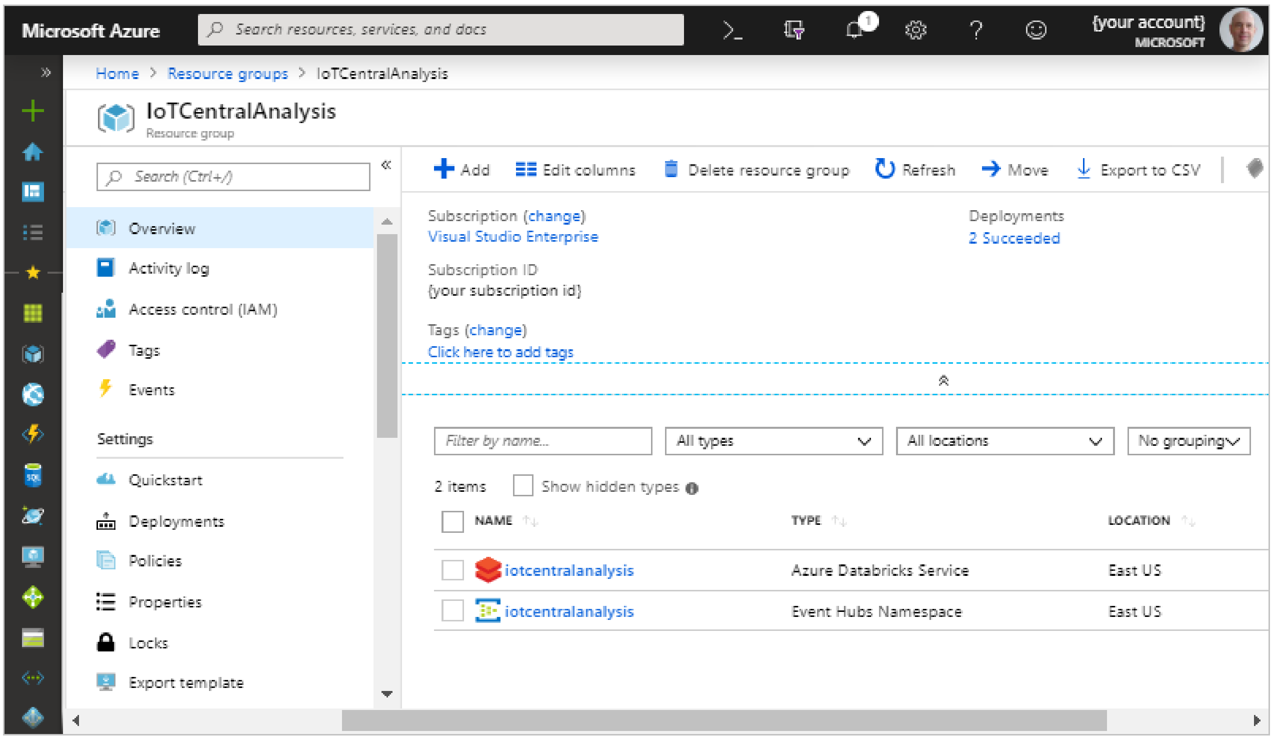
SETTING	VALUE
Name	Choose your namespace name
Pricing tier	Basic
Subscription	Your subscription
Resource group	IoTCentralAnalysis
Location	East US
Throughput Units	1

Azure Databricks workspace

Use the Azure portal to create an Azure Databricks Service with the following settings:

SETTING	VALUE
Workspace name	Choose your workspace name
Subscription	Your subscription
Resource group	IoTCentralAnalysis
Location	East US
Pricing Tier	Standard

When you've created the required resources, your **IoTCentralAnalysis** resource group looks like the following screenshot:

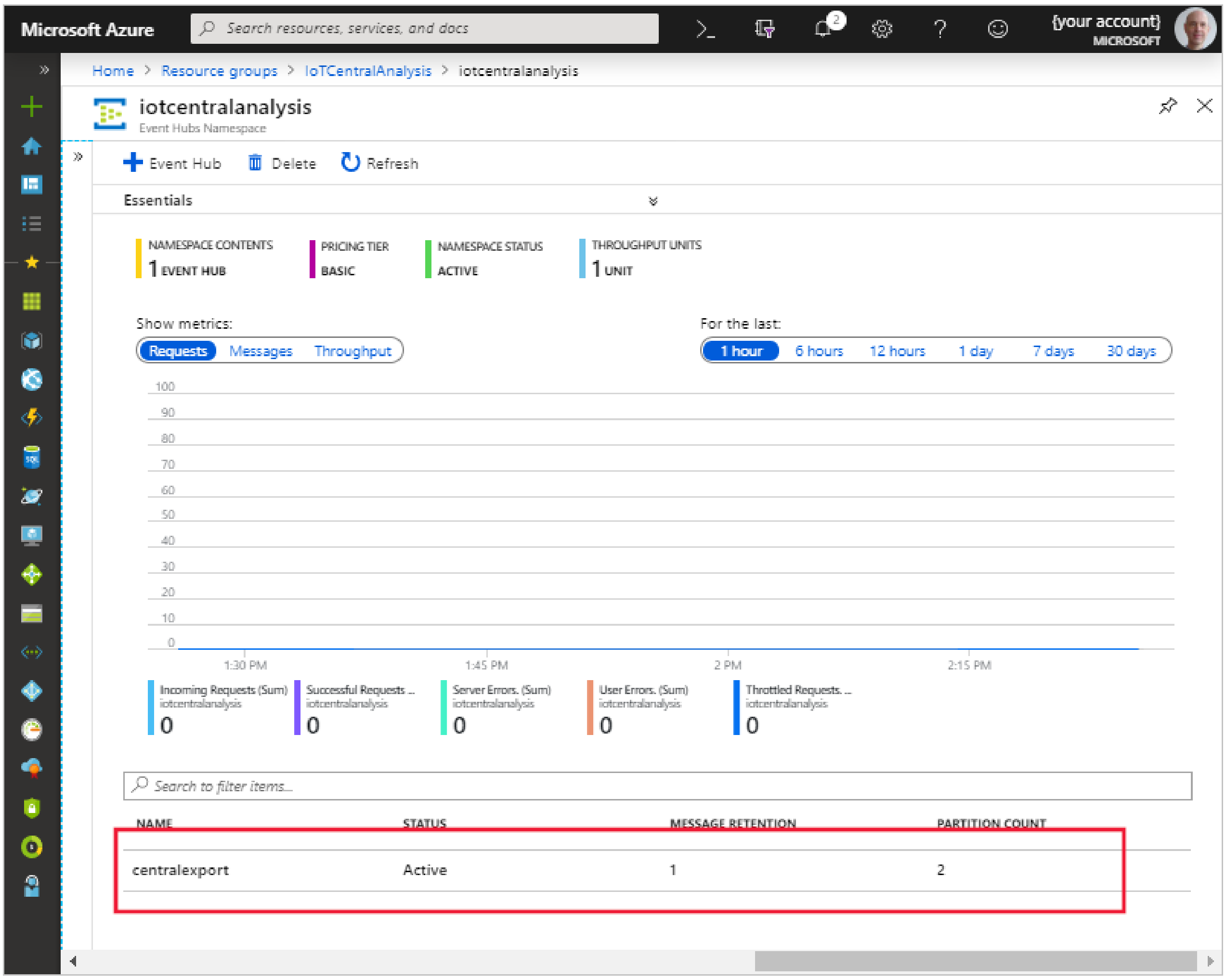


Create an event hub

You can configure an IoT Central application to continuously export telemetry to an event hub. In this section, you create an event hub to receive telemetry from your IoT Central application. The event hub delivers the telemetry to your Stream Analytics job for processing.

1. In the Azure portal, navigate to your Event Hubs namespace and select **+ Event Hub**.
2. Name your event hub **centralexport**, and select **Create**.
3. In the list of event hubs in your namespace, select **centralexport**. Then choose **Shared access policies**.
4. Select **+ Add**. Create a policy named **Listen** with the **Listen** claim.
5. When the policy is ready, select it in the list, and then copy the **Connection string-primary key** value.
6. Make a note of this connection string, you use it later when you configure your Databricks notebook to read from the event hub.

Your Event Hubs namespace looks like the following screenshot:

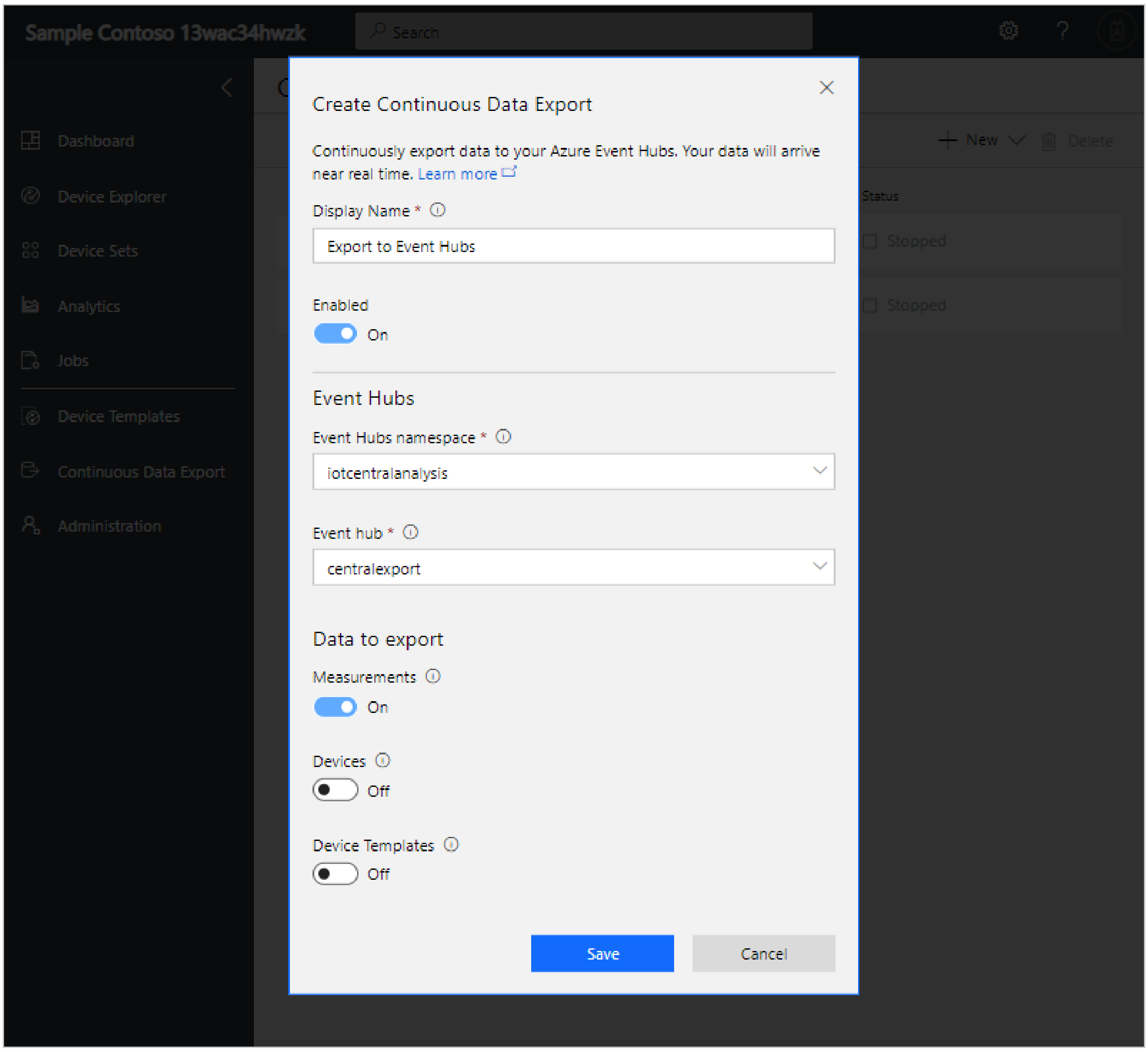


Configure export in IoT Central

On the Azure IoT Central application manager website, navigate to the IoT Central application you created from the Contoso template. In this section, you configure the application to stream the telemetry from its simulated devices to your event hub. To configure the export:

1. Navigate to the **Data Export** page, select **+ New**, and then **Azure Event Hubs**.
2. Use the following settings to configure the export, then select **Save**:

SETTING	VALUE
Display Name	Export to Event Hubs
Enabled	On
Event Hubs namespace	Your Event Hubs namespace name
Event hub	centralexport
Measurements	On
Devices	Off
Device Templates	Off



Wait until the export status is **Running** before you continue.

Configure Databricks workspace

In the Azure portal, navigate to your Azure Databricks service and select **Launch Workspace**. A new tab opens in your browser and signs you in to your workspace.

Create a cluster

On the **Azure Databricks** page, under the list of common tasks, select **New Cluster**.

Use the information in the following table to create your cluster:

SETTING	VALUE
Cluster Name	centralanalysis
Cluster Mode	Standard
Databricks Runtime Version	5.5 LTS (Scala 2.11, Spark 2.4.3)
Python Version	3
Enable Autoscaling	No
Terminate after minutes of inactivity	30
Worker Type	Standard_DS3_v2
Workers	1
Driver Type	Same as worker

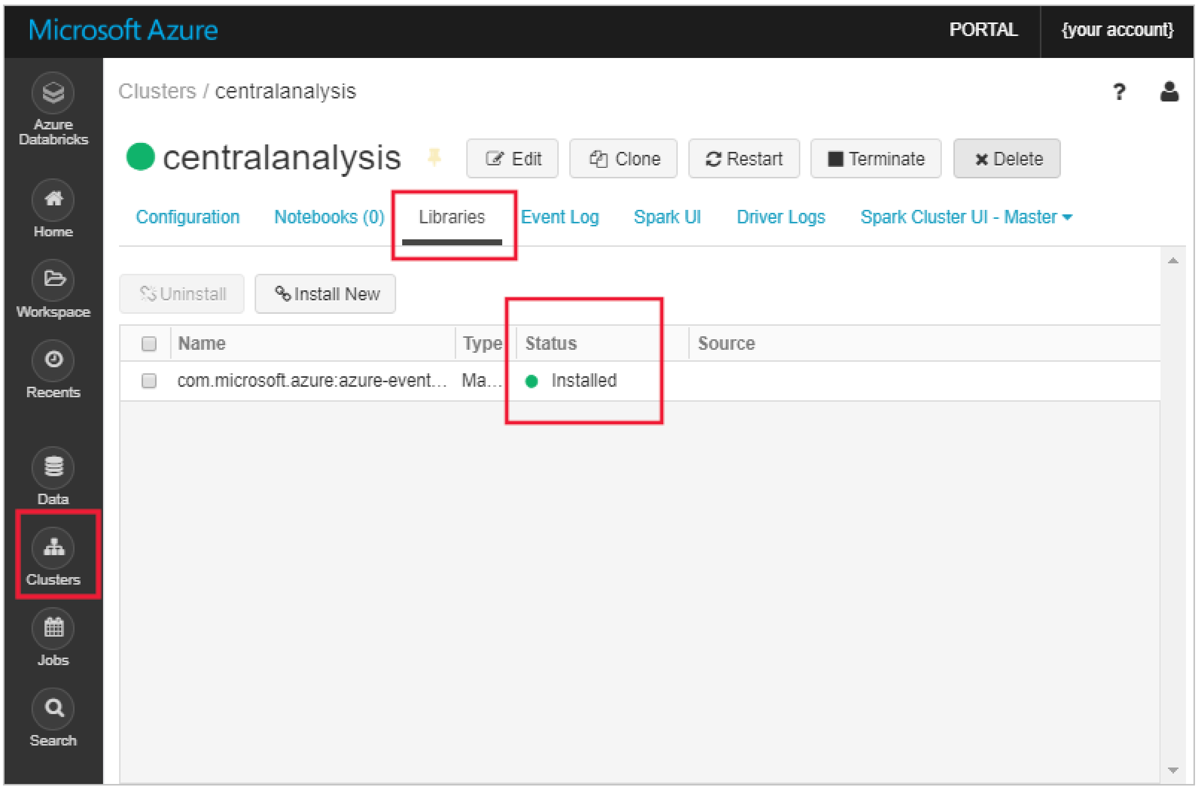
Creating a cluster may take several minutes, wait for the cluster creation to complete before you continue.

Install libraries

On the **Clusters** page, wait until the cluster state is **Running**.

The following steps show you how to import the library your sample needs into the cluster:

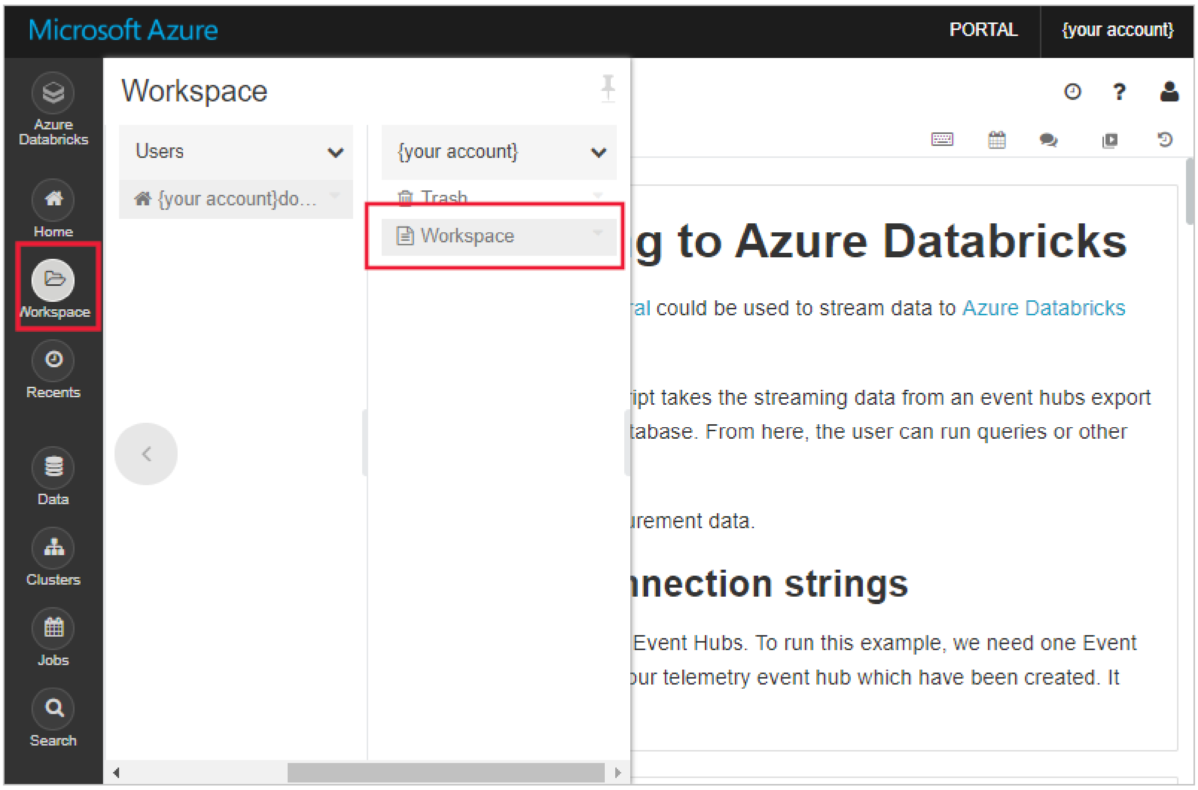
1. On the **Clusters** page, wait until the state of the **centralanalysis** interactive cluster is **Running**.
2. Select the cluster and then choose the **Libraries** tab.
3. On the **Libraries** tab, choose **Install New**.
4. On the **Install Library** page, choose **Maven** as the library source.
5. In the **Coordinates** textbox, enter the following value: `com.microsoft.azure:azure-eventhubs-spark_2.11:2.3.10`
6. Choose **Install** to install the library on the cluster.
7. The library status is now **Installed**:



Import a Databricks notebook

Use the following steps to import a Databricks notebook that contains the Python code to analyze and visualize your IoT Central telemetry:

1. Navigate to the **Workspace** page in your Databricks environment. Select the dropdown next to your account name and then choose **Import**.
2. Choose to import from a URL and enter the following address: https://github.com/Azure-Samples/iot-central-docs-samples/blob/master/databricks/IoT%20Central%20Analysis.dbc?raw=true
3. To import the notebook, choose **Import**.
4. Select the **Workspace** to view the imported notebook:



5. Edit the code in the first Python cell to add the Event Hubs connection string you saved previously:

```
from pyspark.sql.functions import *
from pyspark.sql.types import *

###### Event Hub Connection strings ######
telementryEventHubConfig = {
  'eventhubs.connectionString' : '{your Event Hubs connection string}'
}
```

Run analysis

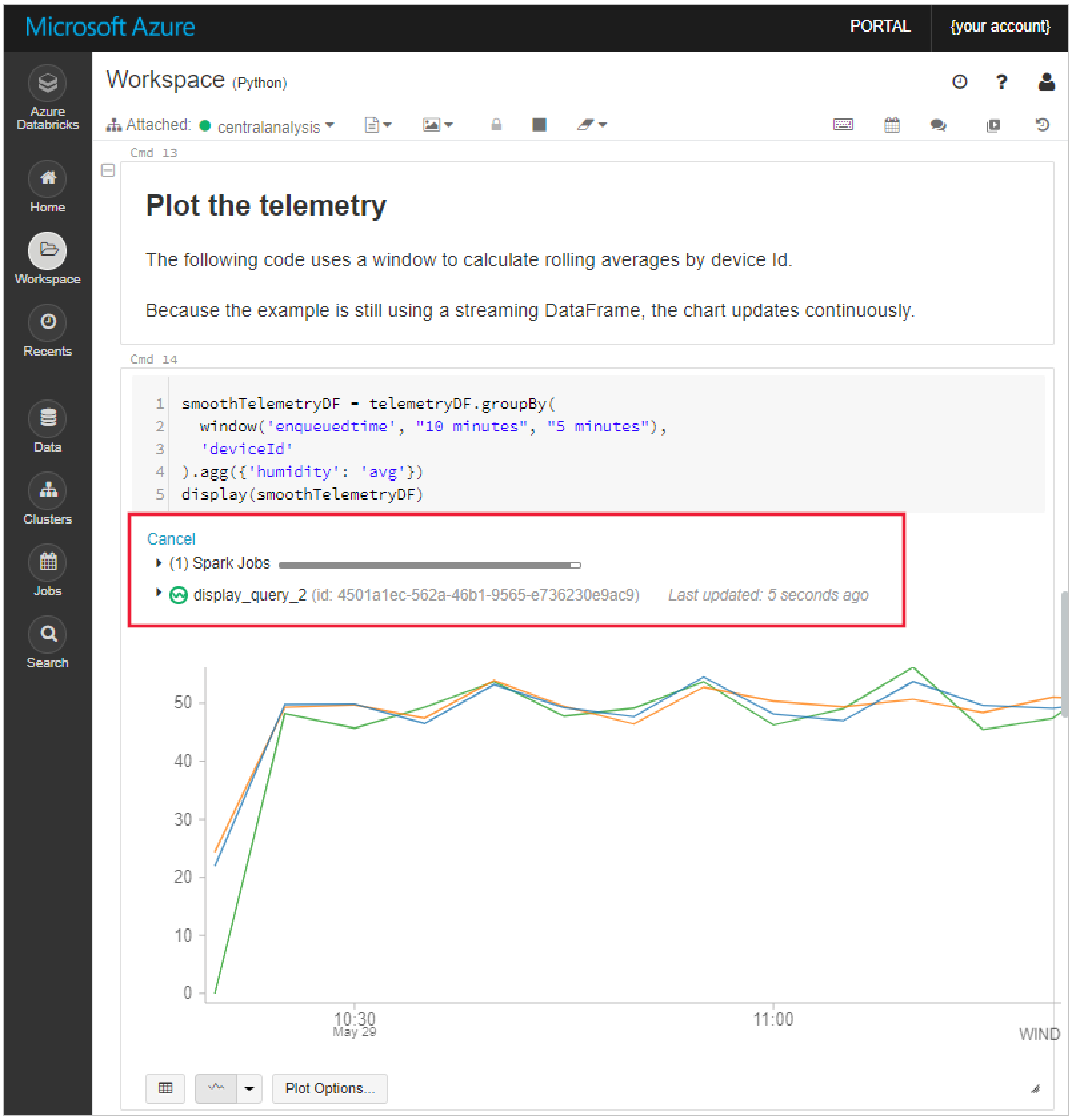
To run the analysis, you must attach the notebook to the cluster:

1. Select **Detached** and then select the **centralanalysis** cluster.
2. If the cluster isn't running, start it.
3. To start the notebook, select the run button.

You may see an error in the last cell. If so, check the previous cells are running, wait a minute for some data to be written to storage, and then run the last cell again.

View smoothed data

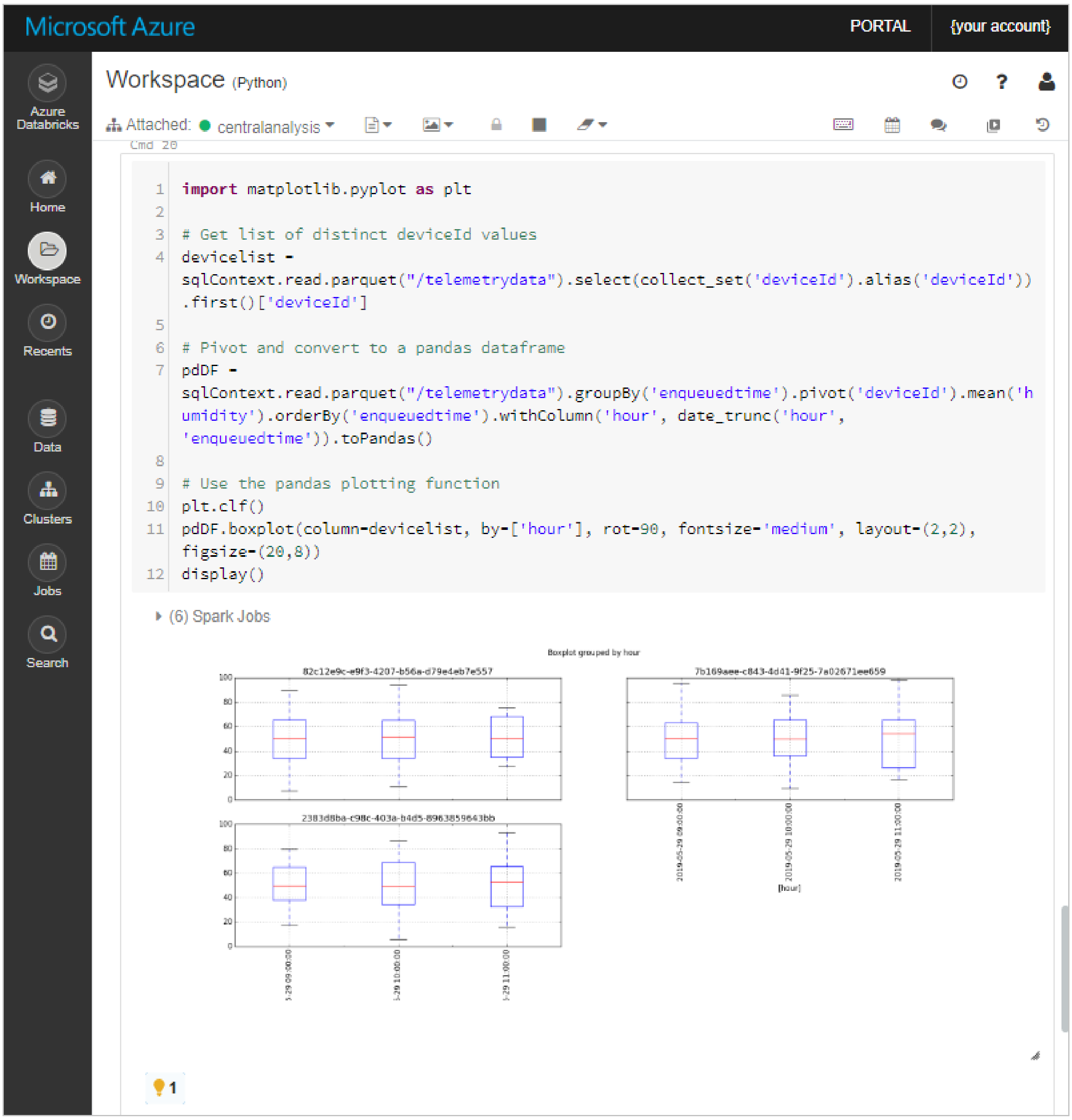
In the notebook, scroll down to cell 14 to see a plot of the rolling average humidity by device type. This plot continuously updates as streaming telemetry arrives:



You can resize the chart in the notebook.

View box plots

In the notebook, scroll down to cell 20 to see the box plots. The box plots are based on static data so to update them you must rerun the cell:



You can resize the plots in the notebook.

Tidy up

To tidy up after this how-to and avoid unnecessary costs, delete the **IoTCentralAnalysis** resource group in the Azure portal.

You can delete the IoT Central application from the **Management** page within the application.

Next steps

In this how-to guide, you learned how to:

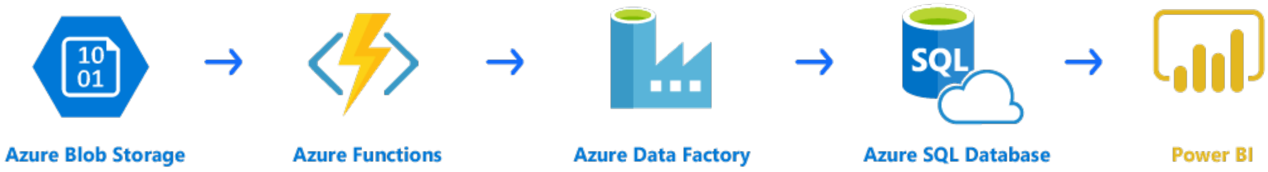
- Stream telemetry from an IoT Central application using *continuous data export*.
- Create an Azure Databricks environment to analyze and plot telemetry data.

Now that you know how to create custom analytics, the suggested next step is to learn how to Visualize and analyze your Azure IoT Central data in a Power BI dashboard.

Visualize and analyze your Azure IoT Central data in a Power BI dashboard

7/21/2020 • 3 minutes to read • Edit Online

This topic applies to administrators and solution developers.



Use the Power BI Solution for Azure IoT Central V3 to create a powerful Power BI dashboard to monitor the performance of your IoT devices. In your Power BI dashboard, you can:

- Track how much data your devices are sending over time
- Compare data volumes between different telemetry streams
- Filter down to data sent by specific devices
- View the most recent telemetry data in a table

This solution sets up a pipeline that reads data from your Continuous Data Export Azure Blob storage account. The pipeline uses Azure Functions, Azure Data Factory, and Azure SQL Database to process and transform the data. you can visualize and analyze the data in a Power BI report that you download as a PBIX file. All of the resources are created in your Azure subscription, so you can customize each component to suit your needs.

Prerequisites

To complete the steps in this how-to guide, you need an active Azure subscription. If you don't have an Azure subscription, create a free account before you begin.

Setting up the solution requires the following resources:

- A version 3 IoT Central application. To learn how to check your application version, see About your application. To learn how to create an IoT Central application, see Create an Azure IoT Central application.
- Continuous data export configured to export telemetry, devices, and device templates to Azure Blob storage. To learn more, see How to export IoT data to destinations in Azure.
  - Make sure that only your IoT Central application is exporting data to the blob container.
  - Your devices must send JSON encoded messages. Devices must specify `contentType:application/JSON` and `contentEncoding:utf-8` or `contentEncoding:utf-16` or `contentEncoding:utf-32` in the message system properties.
- Power BI Desktop (latest version). See Power BI downloads.
- Power BI Pro (if you want to share the dashboard with others).

> **NOTE**
>
> If you're using a version 2 IoT Central application, see Visualize and analyze your Azure IoT Central data in a Power BI dashboard on the previous versions documentation site.

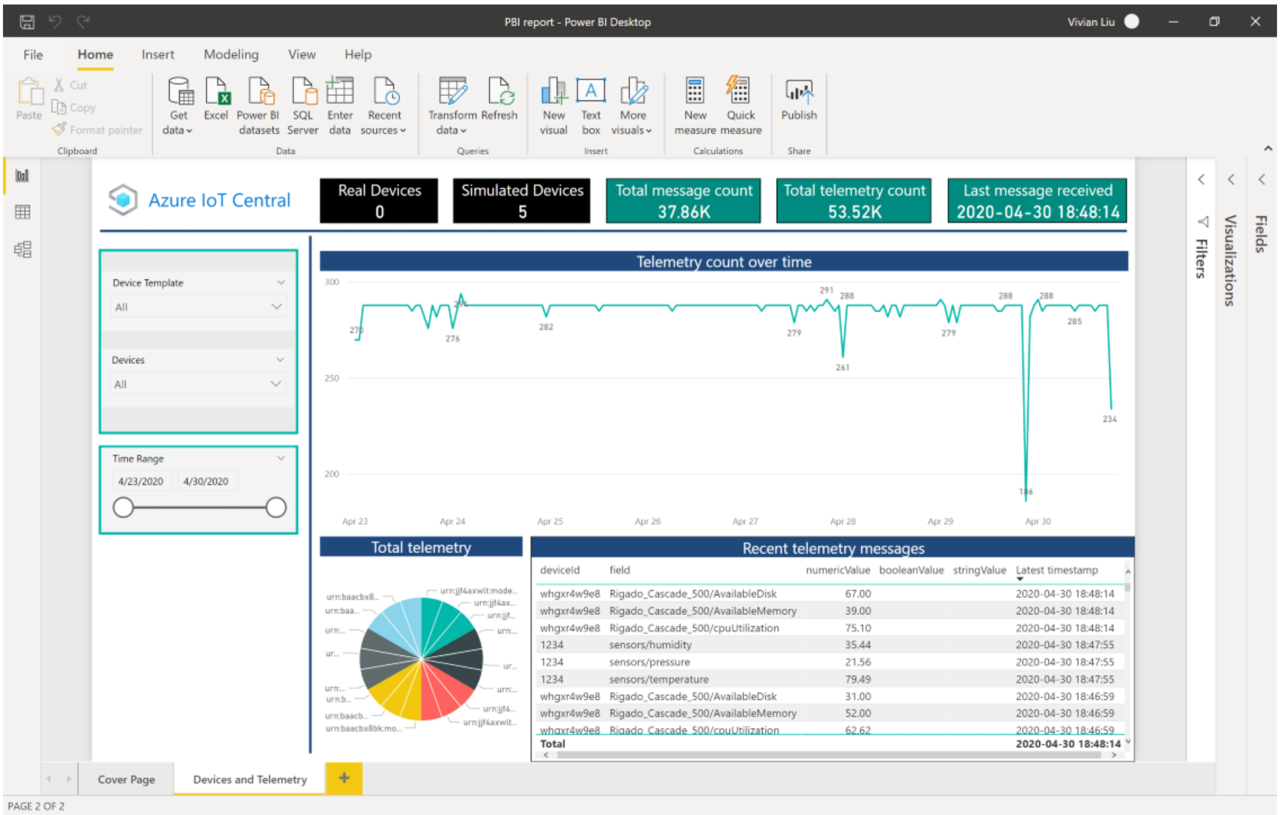
Install

To set up the pipeline, navigate to the Power BI Solution for Azure IoT Central V3 page on the **Microsoft AppSource** site. Select **Get it now**, and follow the instructions.

When you open the PBIX file, be sure the read and follow the instructions on the cover page. These instructions describe how to connect your report to your SQL database.

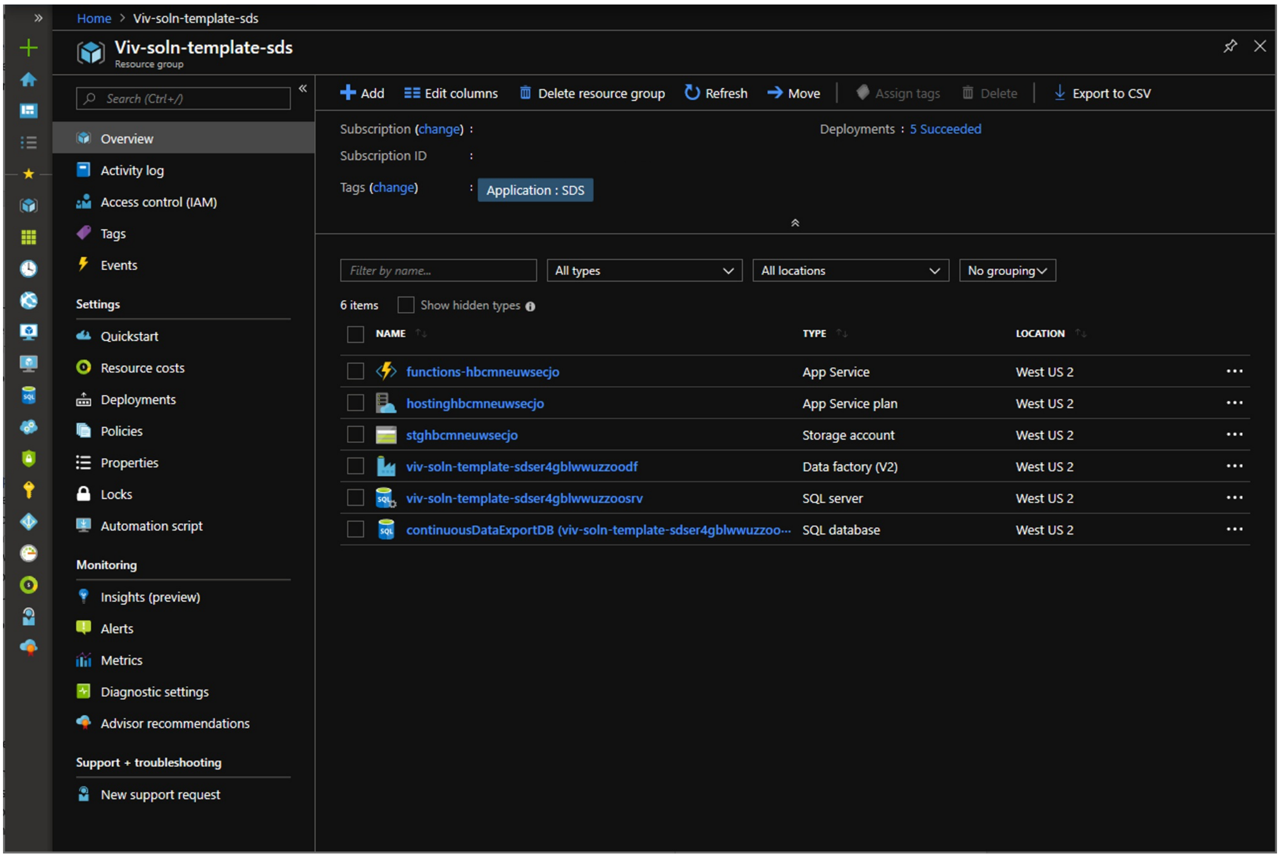
Report

The PBIX file contains the **Devices and Telemetry** report shows a historical view of the telemetry that has been sent by devices. It provides a breakdown of the different types of telemetry, and also shows the most recent telemetry sent by devices.



Pipeline resources

You can access all the Azure resources that make up the pipeline in the Azure portal. All the resources are in the resource group you created when you set up the pipeline.



The following list describes the role of each resource in the pipeline:

Azure Functions

The Azure Function app triggers each time IoT Central writes a new file to Blob storage. The functions extract data from the telemetry, devices, and device templates blobs to populate the intermediate SQL tables that Azure Data Factory uses.

Azure Data Factory

Azure Data Factory connects to SQL Database as a linked service. It runs stored procedures to process the data and store it in the analysis tables.

Azure Data Factory runs every 15 minutes to transform the latest batch of data to load into the SQL tables (which is the current minimal number for the **Tumbling Window Trigger**).

Azure SQL Database

Azure Data Factory generates a set of analysis tables for Power BI. You can explore these schemas in Power BI and use them to build your own visualizations.

Estimated costs

The Power BI Solution for Azure IoT Central V3 page on the Microsoft AppSource site includes a link to a cost estimator for the resources you deploy.

Next steps

Now that you've learned how to visualize your data in Power BI, the suggested next step is to learn How to manage devices.

Change IoT Central application settings

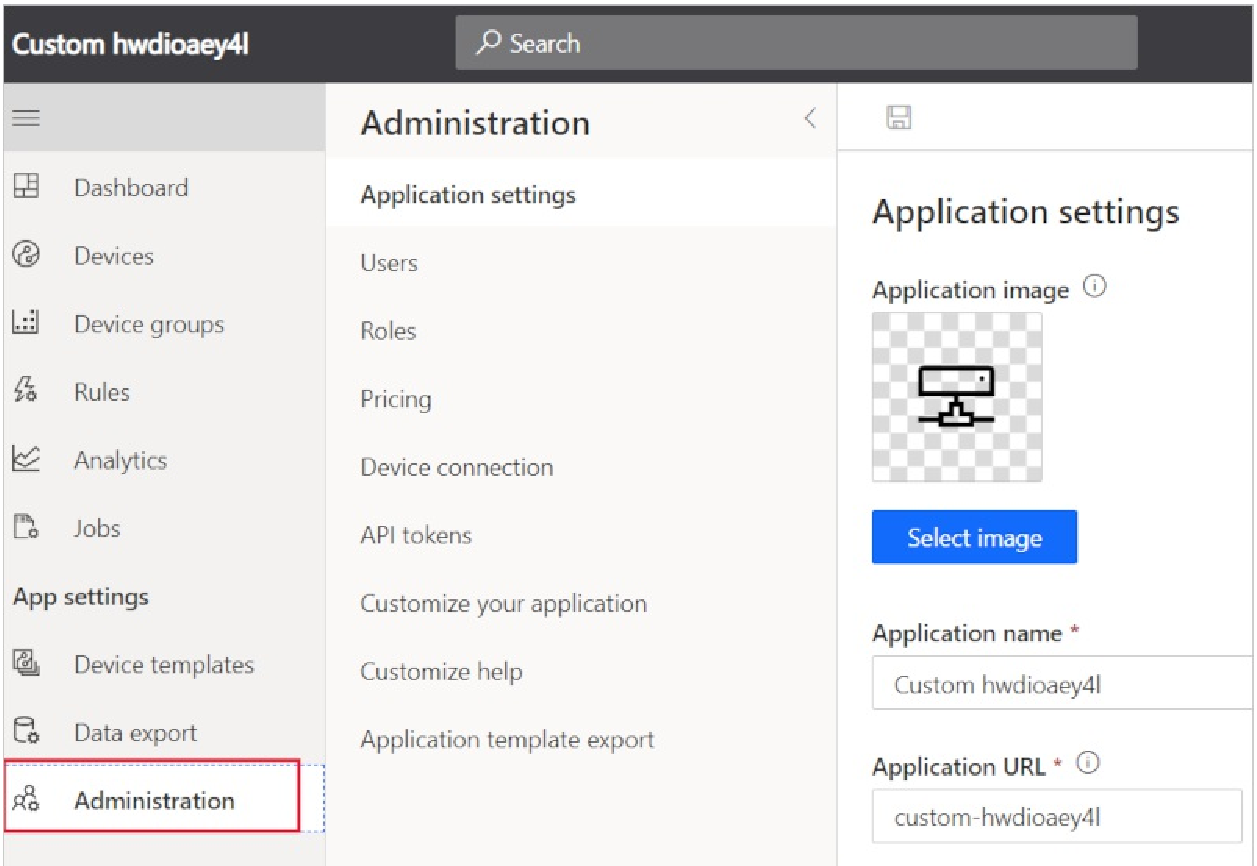
3/24/2020 • 2 minutes to read • Edit Online

This article describes how, as an administrator, you can manage application by changing application name and URL, uploading image, and delete an application in your Azure IoT Central application.

To access and use the **Administration** section, you must be in the **Administrator** role for an Azure IoT Central application. If you create an Azure IoT Central application, you're automatically assigned to the **Administrator** role for that application.

Change application name and URL

In the **Application Settings** page, you can change the name and URL of your application, then select **Save**.



If your administrator creates a custom theme for your application, this page includes an option to hide the **Application Name** in the UI. This option is useful if the application logo in the custom theme includes the application name. For more information, see Customize the Azure IoT Central UI.

> **NOTE**
> If you change your URL, your old URL can be taken by another Azure IoT Central customer. If that happens, it is no longer available for you to use. When you change your URL, the old URL no longer works, and you need to notify your users about the new URL to use.

Delete an application

Use the **Delete** button to permanently delete your IoT Central application. This action permanently deletes all data that's associated with the application.

> **NOTE**
> To delete an application, you must also have permissions to delete resources in the Azure subscription you chose when you created the application. To learn more, see Use role-based access control to manage access to your Azure subscription resources.

Manage programmatically

IoT Central Azure Resource Manager SDK packages are available for Node, Python, C#, Ruby, Java, and Go. You can use these packages to create, list, update, or delete IoT Central applications. The packages include helpers to manage authentication and error handling.

You can find examples of how to use the Azure Resource Manager SDKs at https://github.com/emgarten/iotcentral-arm-sdk-examples.

To learn more, see the following GitHub repositories and packages:

LANGUAGE	REPOSITORY	PACKAGE
Node	https://github.com/Azure/azure-sdk-for-node	https://www.npmjs.com/package/azure-arm-iotcentral
Python	https://github.com/Azure/azure-sdk-for-python	https://pypi.org/project/azure-mgmt-iotcentral
C#	https://github.com/Azure/azure-sdk-for-net	https://www.nuget.org/packages/Microsoft.Azure.Management.IotCentral
Ruby	https://github.com/Azure/azure-sdk-for-ruby	https://rubygems.org/gems/azure_mgmt_iot_central
Java	https://github.com/Azure/azure-sdk-for-java	https://search.maven.org/search?q=a:azure-mgmt-iotcentral
Go	https://github.com/Azure/azure-sdk-for-go	https://github.com/Azure/azure-sdk-for-go

Next steps

Now that you've learned about how to administer your Azure IoT Central application, the suggested next step is to learn about Manage users and roles in Azure IoT Central.

Manage users and roles in your IoT Central application

3/27/2020 • 6 minutes to read • Edit Online

This article describes how, as an administrator, you can add, edit, and delete users in your Azure IoT Central application. The article also describes how to manage roles in your Azure IoT Central application.

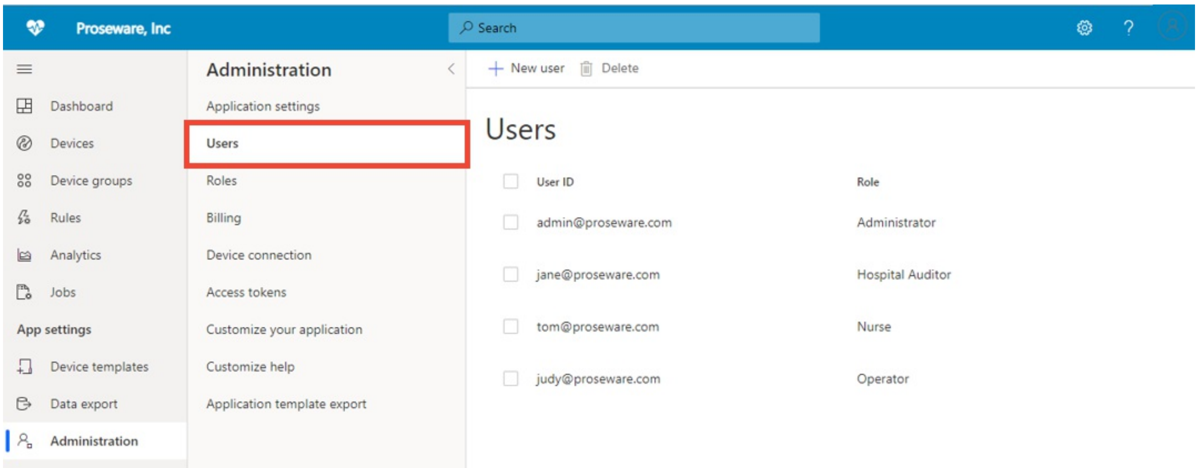
To access and use the **Administration** section, you must be in the **Administrator** role for an Azure IoT Central application. If you create an Azure IoT Central application, you're automatically added to the **Administrator** role for that application.

Add users

Every user must have a user account before they can sign in and access an Azure IoT Central application. Microsoft Accounts and Azure Active Directory accounts are supported in Azure IoT Central. Azure Active Directory groups aren't currently supported in Azure IoT Central.

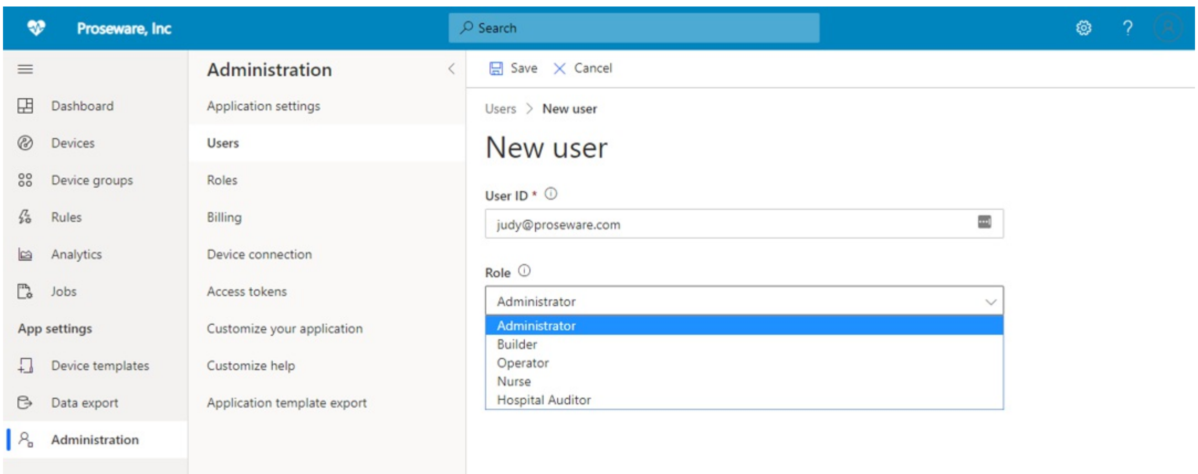
For more information, see Microsoft account help and Quickstart: Add new users to Azure Active Directory.

1. To add a user to an IoT Central application, go to the **Users** page in the **Administration** section.



2. To add a user, on the **Users** page, choose **+ Add user**.

3. Choose a role for the user from the **Role** drop-down menu. Learn more about roles in the Manage roles section of this article.



> **NOTE**
>
> A user who is in a custom role that grants them the permission to add other users, can only add users to a role with same or fewer permissions than their own role.

If an IoT Central user ID is deleted from Azure Active Directory and then readded, the user won't be able to sign in the IoT Central application. To re-enable access, the IoT Central administrator should delete and readd the user in the application.

Edit the roles that are assigned to users

Roles can't be changed after they're assigned. To change the role that's assigned to a user, delete the user, and then add the user again with a different role.

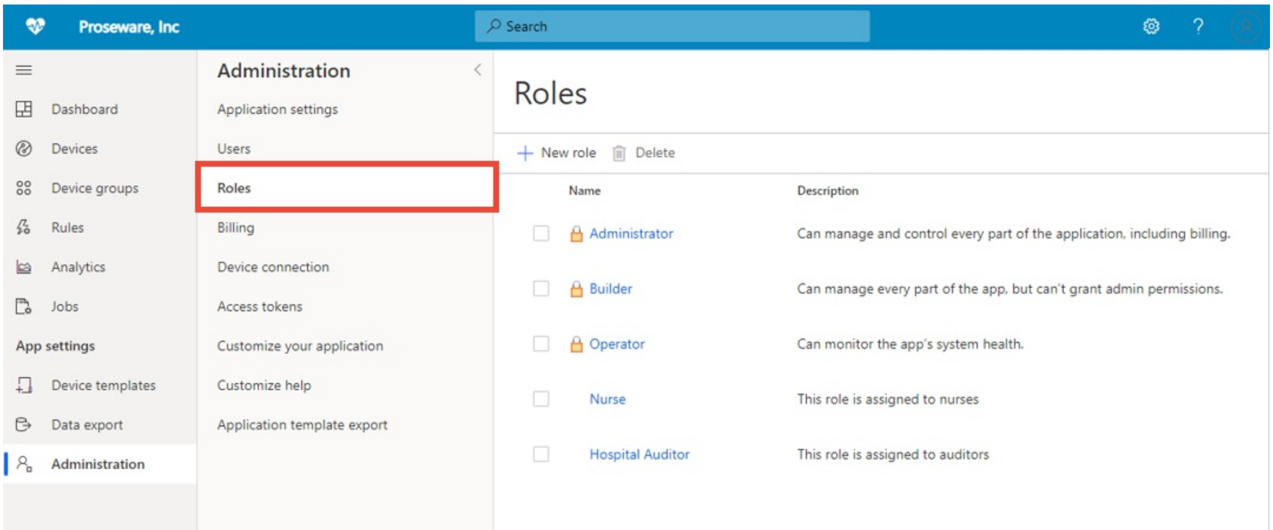
> **NOTE**
>
> The roles assigned are specific to IoT Central application and cannot be managed from the Azure Portal.

Delete users

To delete users, select one or more check boxes on the **Users** page. Then select **Delete**.

Manage roles

Roles enable you to control who within your organization is allowed to do various tasks in IoT Central. There are three built-in roles you can assign to users of your application. You can also create custom roles if you require finer-grained control.



Administrator

Users in the **Administrator** role can manage and control every part of the application, including billing.

The user who creates an application is automatically assigned to the **Administrator** role. There must always be at least one user in the **Administrator** role.

Builder

Users in the **Builder** role can manage every part of the app, but can't make changes on the Administration or Continuous Data Export tabs.

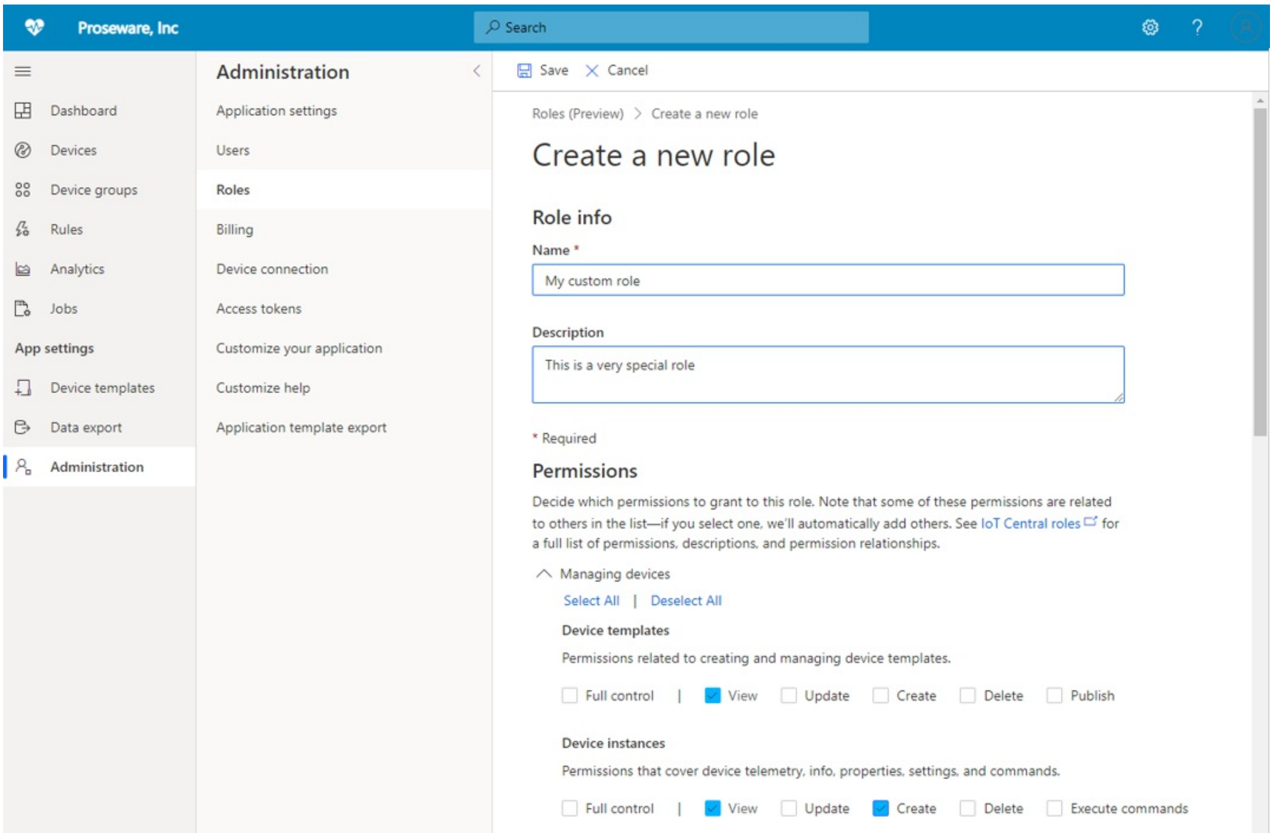
Operator

Users in the **Operator** role can monitor device health and status. They aren't allowed to make changes to device templates or to administer the application. Operators can add and delete devices, manage device sets, and run analytics and jobs.

Create a custom role

If your solution requires finer-grained access controls, you can create custom roles with custom sets of permissions. To create a custom role, navigate to the **Roles** page in the **Administration** section of your application. Then select **+ New role**, and add a name and description for your role. Select the permissions your role requires and then select **Save**.

You can add users to your custom role in the same way that you add users to a built-in role.



Custom role options

When you define a custom role, you choose the set of permissions that a user is granted if they're a member of the role. Some permissions are dependent on others. For example, if you add the **Update application dashboards** permission to a role, the **View application dashboards** permission is automatically added. The following tables summarize the available permissions, and their dependencies, you can use when creating custom roles.

Managing devices

Device template permissions

NAME	DEPENDENCIES
View	None
Manage	View Other dependencies: View device instances
Full Control	View, Manage Other dependencies: View device instances

Device instance permissions

NAME	DEPENDENCIES
View	None Other dependencies: View device templates and device groups
Update	View Other dependencies: View device templates and device groups
Create	View Other dependencies: View device templates and device groups
Delete	View Other dependencies: View device templates and device groups
Execute Commands	Update, View Other dependencies: View device templates and device groups
Full Control	View, Update, Create, Delete, Execute Commands Other dependencies: View device templates and device groups

Device groups permissions

NAME	DEPENDENCIES
View	None Other dependencies: View device templates and device instances
Update	View Other dependencies: View device templates and device instances
Create	View, Update Other dependencies: View device templates and device instances
Delete	View Other dependencies: View device templates and device instances
Full Control	View, Update, Create, Delete Other dependencies: View device templates and device instances

Device connectivity management permissions

NAME	DEPENDENCIES
Read instance	None Other dependencies: View device templates, device groups, device instances

NAME	DEPENDENCIES
Manage instance	None
Read global	None
Manage global	Read Global
Full Control	Read instance, Manage instance, Read global, Manage global. Other dependencies: View device templates, device groups, device instances

Jobs permissions

NAME	DEPENDENCIES
View	None Other dependencies: View device templates, device instances, and device groups
Update	View Other dependencies: View device templates, device instances, and device groups
Create	View, Update Other dependencies: View device templates, device instances, and device groups
Delete	View Other dependencies: View device templates, device instances, and device groups
Execute	View Other dependencies: View device templates, device instances, and device groups; Update device instances; Execute commands on device instances
Full Control	View, Update, Create, Delete, Execute Other dependencies: View device templates, device instances, and device groups; Update device instances; Execute commands on device instances

Rules permissions

NAME	DEPENDENCIES
View	None Other dependencies: View device templates
Update	View Other dependencies: View device templates
Create	View, Update Other dependencies: View device templates

NAME	DEPENDENCIES
Delete	View Other dependencies: View device templates
Full Control	View, Update, Create, Delete Other dependencies: View device templates

Managing the app

Application settings permissions

NAME	DEPENDENCIES
View	None
Update	View
Copy	View Other dependencies: View device templates, device instances, device groups, dashboards, data export, branding, help links, custom roles, rules
Delete	View
Full Control	View, Update, Copy, Delete Other dependencies: View device templates, device groups, application dashboards, data export, branding, help links, custom roles, rules

Application template export permissions

NAME	DEPENDENCIES
View	None
Export	View Other dependencies: View device templates, device instances, device groups, dashboards, data export, branding, help links, custom roles, rules
Full Control	View, Export Other dependencies: View device templates, device groups, application dashboards, data export, branding, help links, custom roles, rules

Billing permissions

NAME	DEPENDENCIES
Manage	None
Full Control	Manage

Managing users and roles

Custom roles permissions

NAME	DEPENDENCIES
View	None
Update	View
Create	View, Update
Delete	View
Full Control	View, Update, Create, Delete

User management permissions

NAME	DEPENDENCIES
View	None Other dependencies: View custom roles
Add	View Other dependencies: View custom roles
Delete	View Other dependencies: View custom roles
Full Control	View, Add, Delete Other dependencies: View custom roles

> **NOTE**
>
> A user who is in a custom role that grants them the permission to add other users, can only add users to a role with same or fewer permissions than their own role.

Customizing the app

Application dashboard permissions

NAME	DEPENDENCIES
View	None
Update	View
Create	View, Update
Delete	View
Full Control	View, Update, Create, Delete

Personal dashboards permissions

NAME	DEPENDENCIES
View	None

NAME	DEPENDENCIES
Update	View
Create	View, Update
Delete	View
Full Control	View, Update, Create, Delete

Branding, favicon, and colors permissions

NAME	DEPENDENCIES
View	None
Update	View
Full Control	View, Update

Help links permissions

NAME	DEPENDENCIES
View	None
Update	View
Full Control	View, Update

Extending the app

Data export permissions

NAME	DEPENDENCIES
View	None
Update	View
Create	View, Update
Delete	View
Full Control	View, Update, Create, Delete

API token permissions

NAME	DEPENDENCIES
View	None
Create	View

NAME	DEPENDENCIES
Delete	View
Full Control	View, Create, Delete

Next steps

Now that you've learned about how to manage users and roles in your Azure IoT Central application, the suggested next step is to learn how to Manage your bill.

Manage your bill in an IoT Central application

3/24/2020 • 2 minutes to read • Edit Online

This article describes how, as an administrator, you can manage your bill in Azure IoT Central application in the administration section. You will learn how you can move your application from the free pricing plan to a standard pricing plan, and also how to upgrade or downgrade your pricing plan.

To access and use the **Administration** section, you must be in the *Administrator* role or have a *custom user role* that allows you to view billing for an Azure IoT Central application. If you create an Azure IoT Central application, you're automatically assigned to the **Administrator** role for that application.

Move from free to standard pricing plan

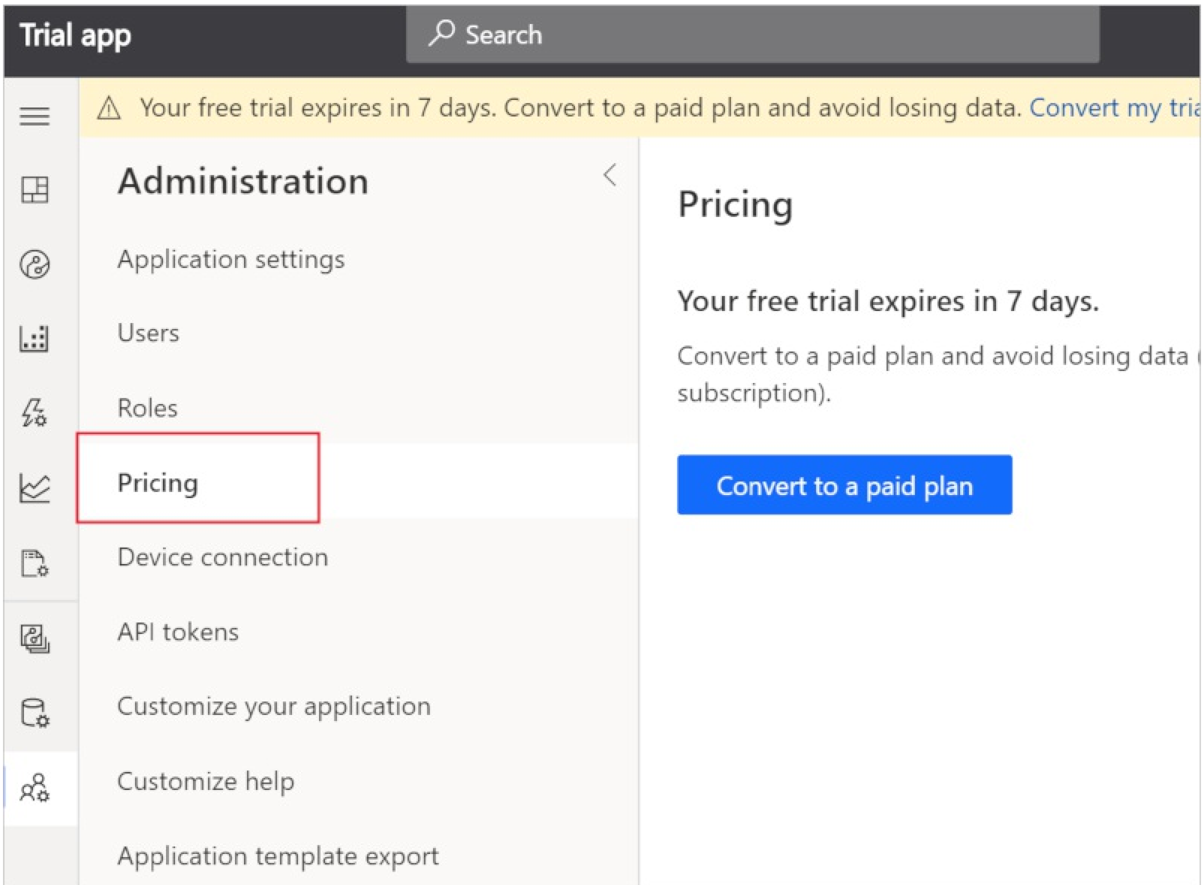
- Applications that use the free pricing plan are free for seven days before they expire. In order to avoid losing data you can move them to a standard pricing plan at any time before they expire.
- Applications that use a standard pricing plan are charged per device, with the first two devices free, per application.

Learn more about pricing on the Azure IoT Central pricing page.

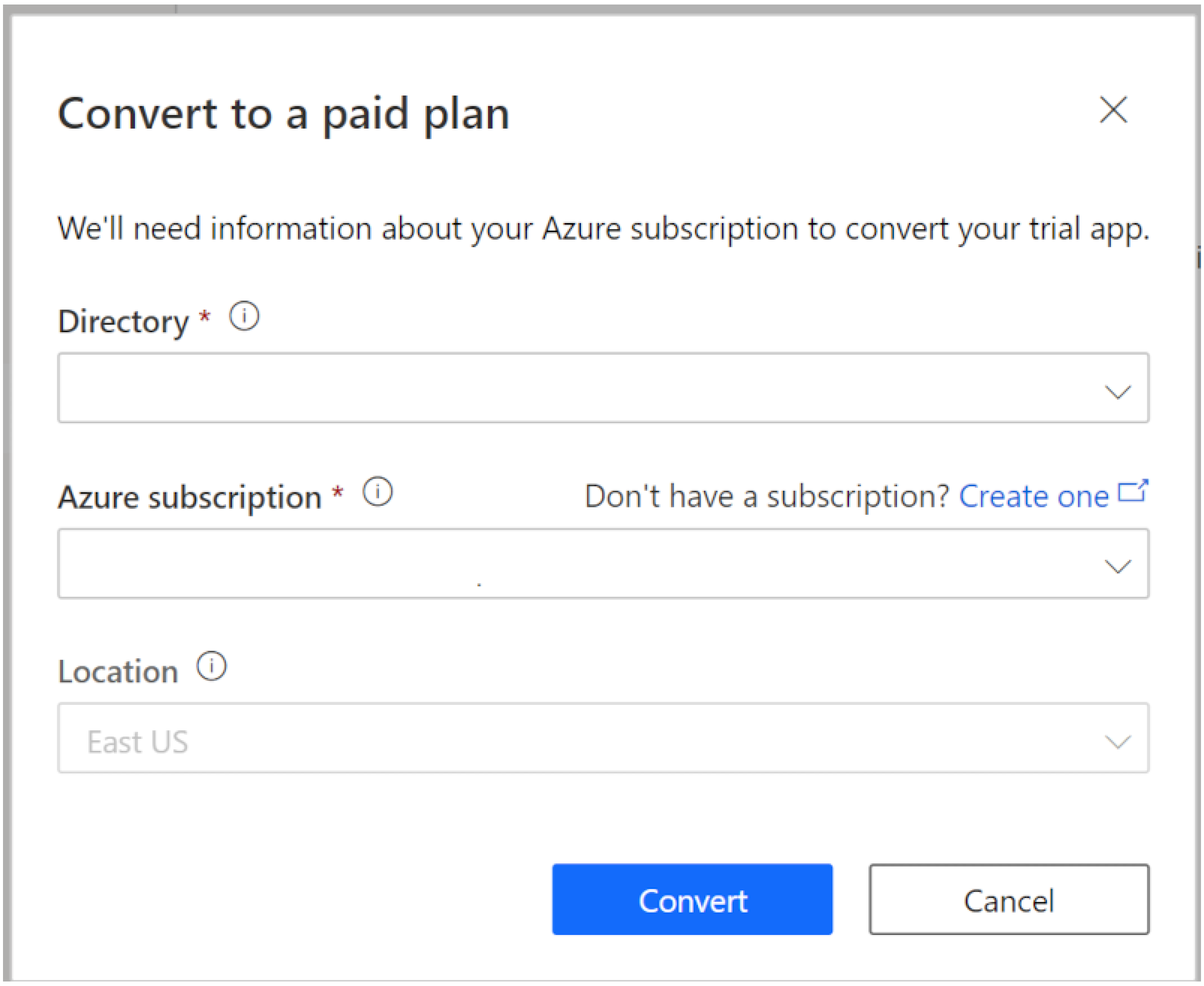
In the pricing section, you can move your application from the free to a standard pricing plan.

To complete this self-service process, follow these steps:

1. Go to the **Pricing** page in the **Administration** section.



2. Select **Convert to a paid plan**.



3. Select the appropriate Azure Active Directory, and then the Azure subscription to use for your application that uses a paid plan.

4. After you select **Convert**, your application now uses a paid plan and you start getting billed.

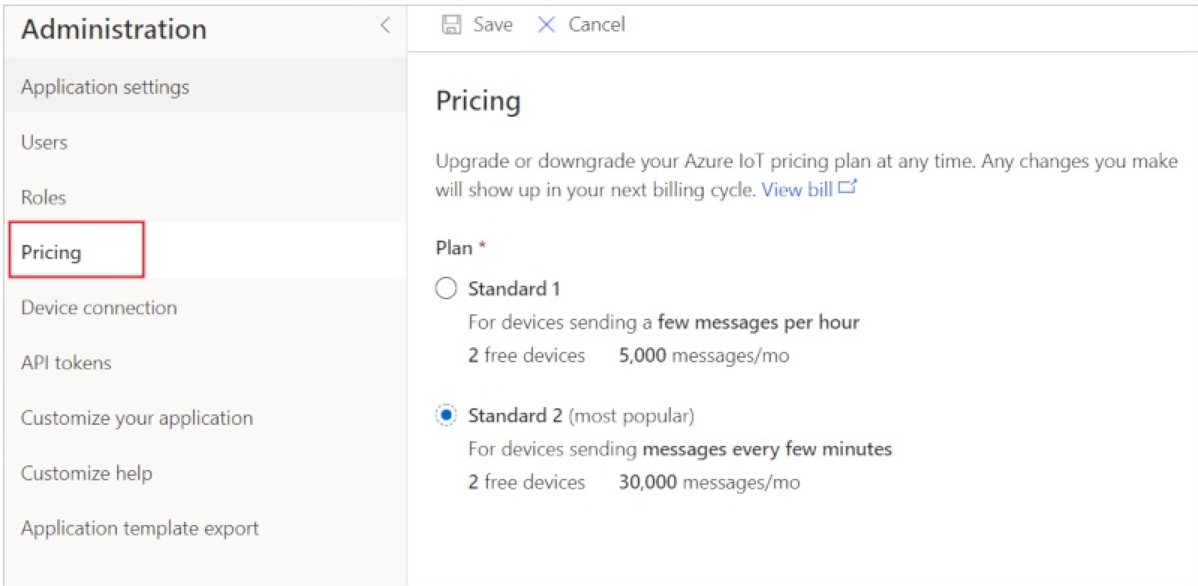
> **NOTE**
> By default, you are converted to a *Standard 2* pricing plan.

How to change your application pricing plan

Applications that use a standard pricing plan are charged per device, with the first two devices free, per application.

In the pricing section, you can upgrade or downgrade your Azure IoT pricing plan at any time.

1. Go to the **Pricing** page in the **Administration** section.



2. Select the **Plan** and click **Save** to upgrade or downgrade.

View your bill

1. Select the appropriate Azure Active Directory, and then the Azure subscription to use for your application that uses a paid plan.

2. After you select **Convert**, your application now uses a paid plan and you start getting billed.

> **NOTE**
> By default, you are converted to a *Standard 2* pricing plan.

Next steps

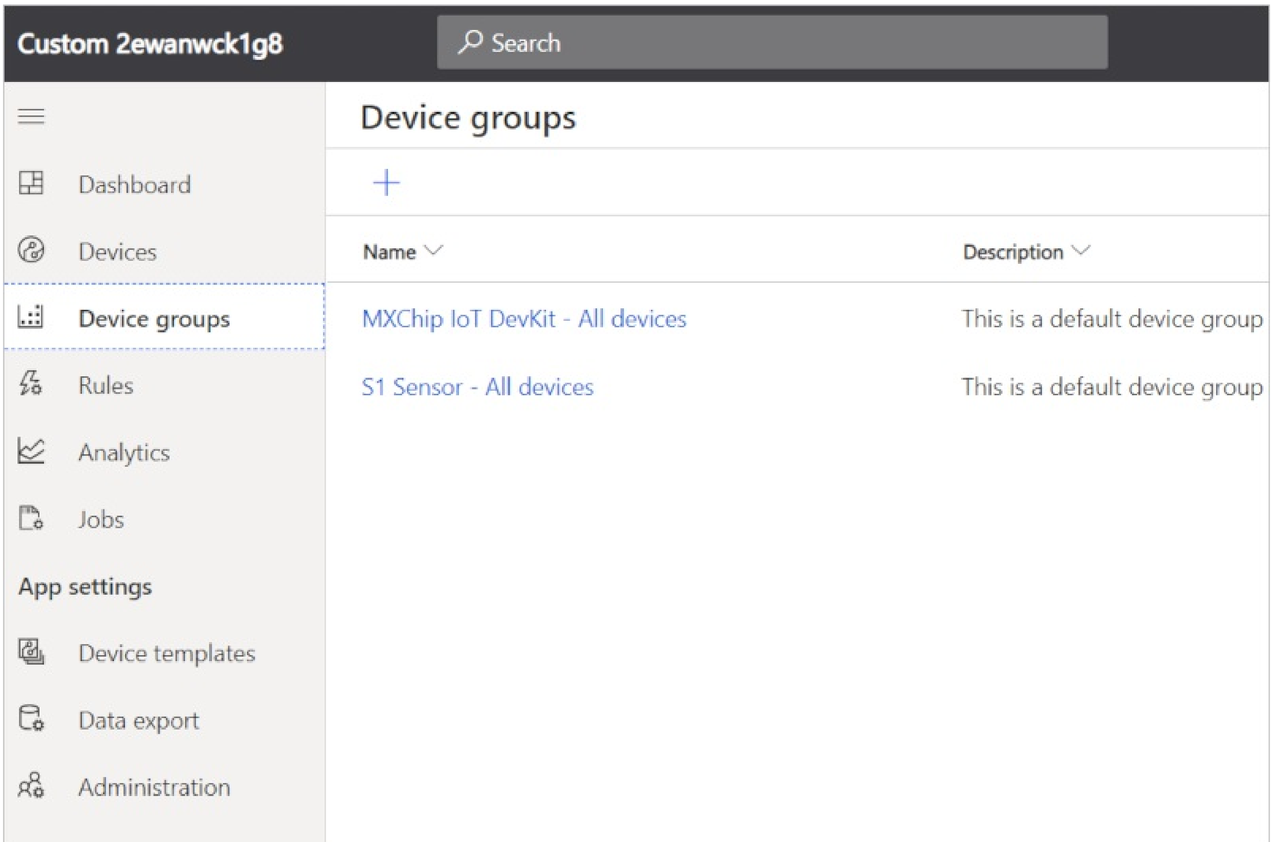
Now that you've learned about how to manage your bill in Azure IoT Central application, the suggested next step is to learn about Customize application UI in Azure IoT Central.

Customize the Azure IoT Central UI

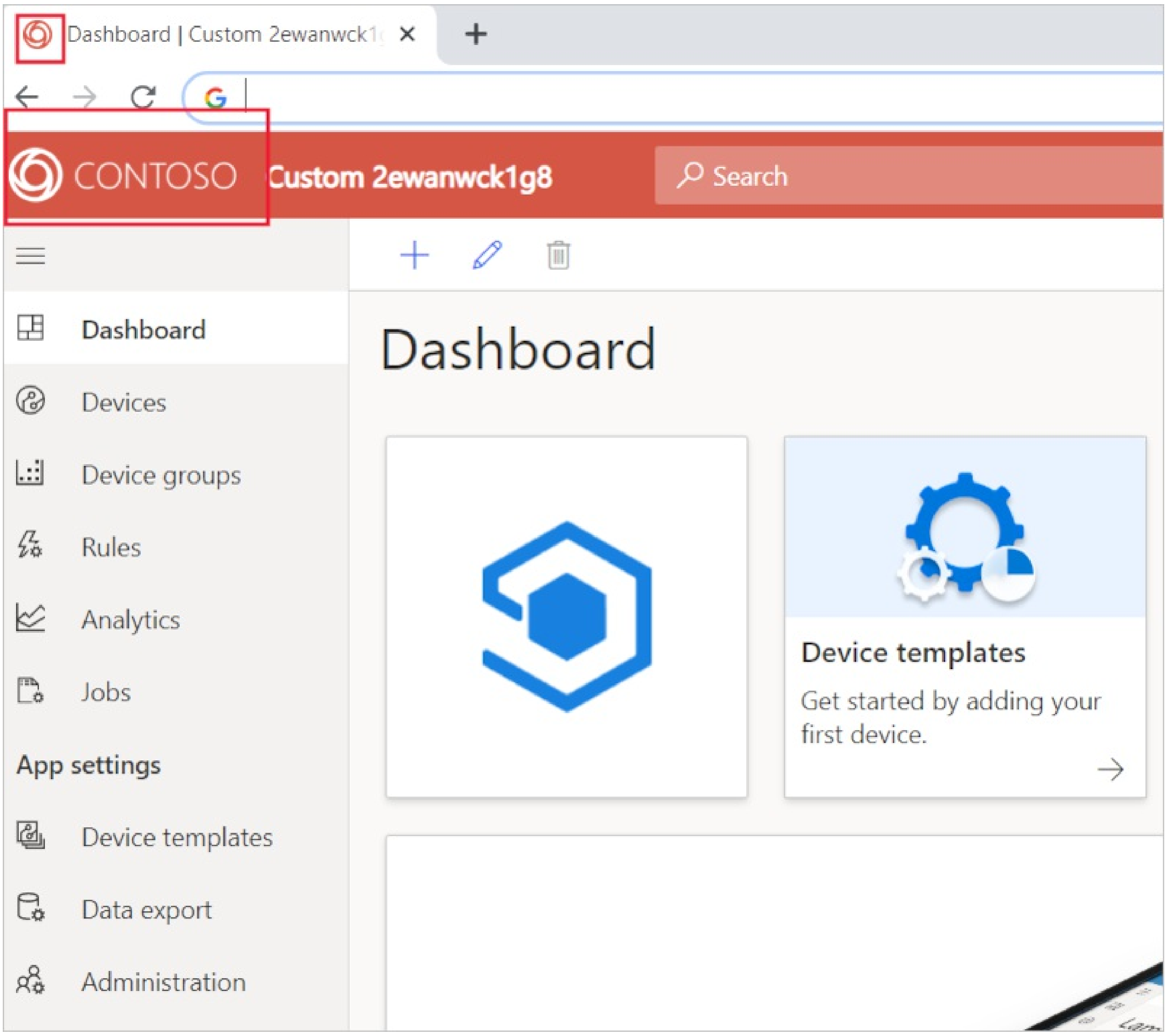
3/24/2020 • 2 minutes to read • Edit Online

This article describes how, as an administrator, you can customize the UI of your application by applying custom themes and modifying the help links to point to your own custom help resources.

The following screenshot shows a page using the standard theme:

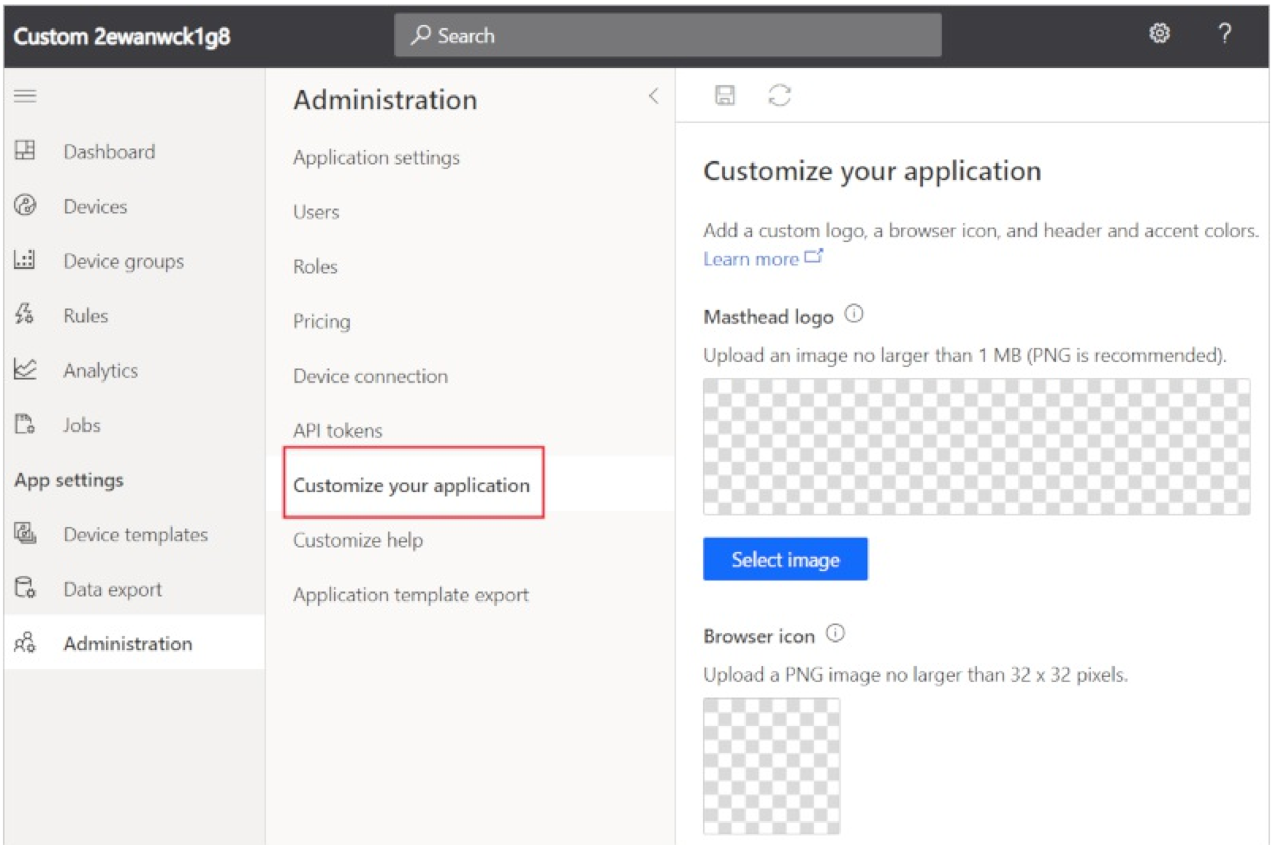


The following screenshot shows a page using a custom screenshot with the customized UI elements highlighted:



Create theme

To create a custom theme, navigate to the **Customize your application** page in the **Administration** section:



On this page, you can customize the following aspects of your application:

Application logo

A PNG image, no larger than 1 MB, with a transparent background. This logo displays to the left on the IoT Central application title bar.

If your logo image includes the name of your application, you can hide the application name text. For more information, see Manage your application.

Browser icon (favicon)

A PNG image, no larger than 32 x 32 pixels, with a transparent background. A web browser can use this image in the address bar, history, bookmarks, and browser tab.

Browser colors

You can change the color of the page header and the color used for accenting buttons and other highlights. Use a six character hex color value in the format `##ff6347`. For more information about **HEX Value** color notation, see HTML Colors.

> **NOTE**
>
> You can always revert back to the default options on the **Customize your application** page.

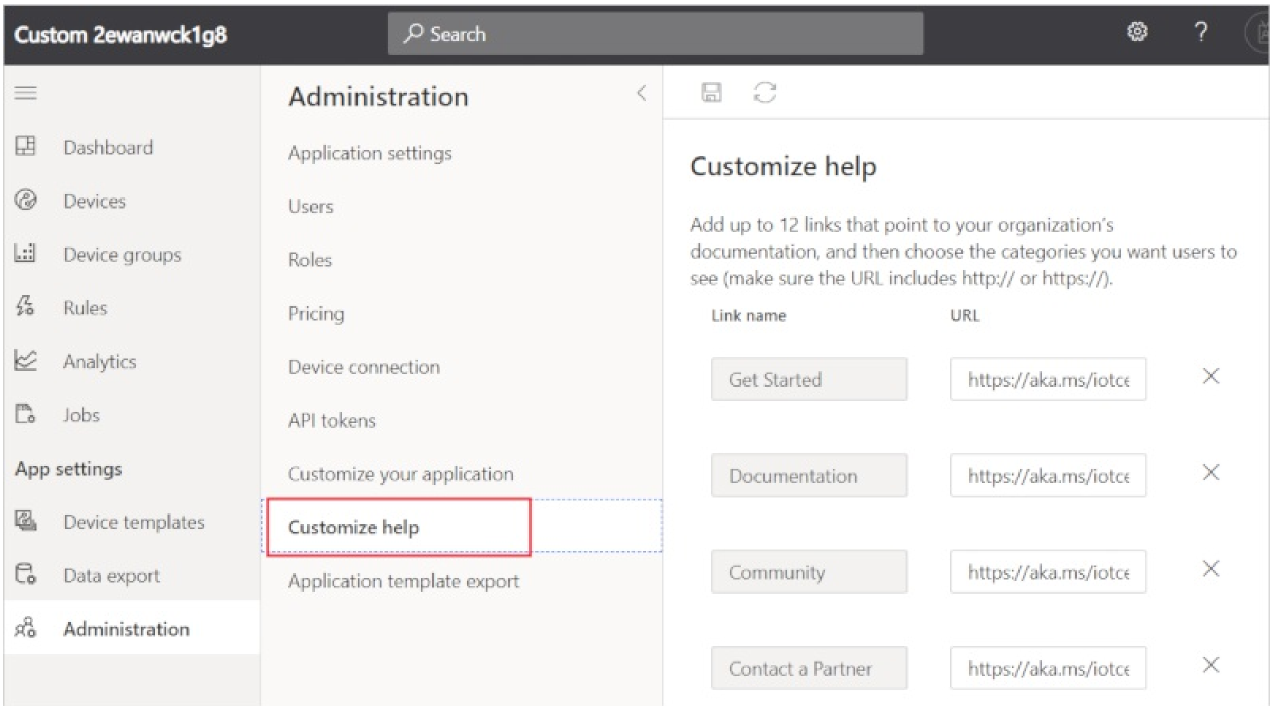
Changes for operators

If an administrator creates a custom theme, then operators and other users of your application can no longer choose a theme in **Settings**.

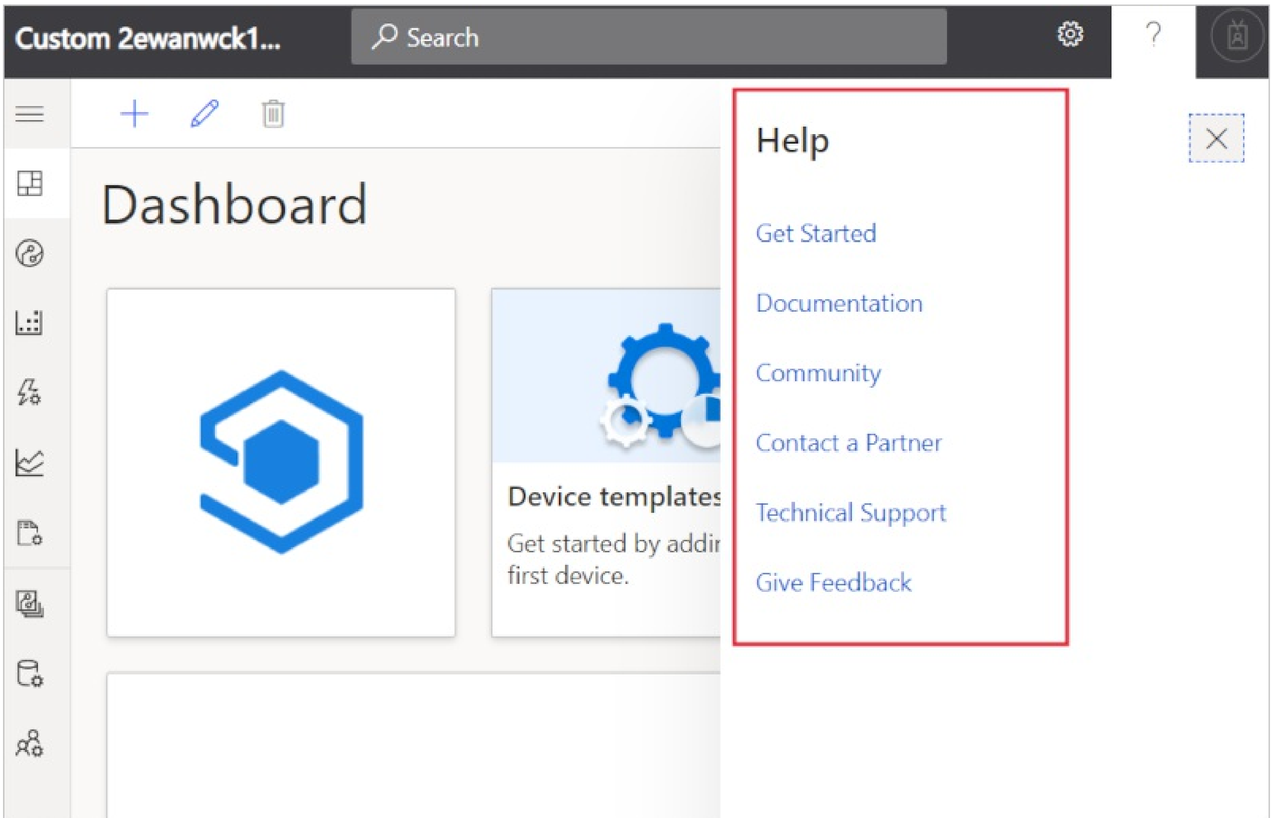
Replace help links

To provide custom help information to your operators and other users, you can modify the links on the application **Help** menu.

To modify the help links, navigate to the **Customize help** page in the **Administration** section:



You can also add new entries to the help menu and remove default entries:



> **NOTE**
>
> You can always revert back to the default help links on the **Customize help** page.

Next steps

Now that you've learned how to customize the UI in your IoT Central application, here are some suggested next steps:

- Administer your application

- Add tiles to your dashboard

Export your application

7/21/2020 • 3 minutes to read • Edit Online

This article describes how, as a solution manager, to export an IoT Central application to be able to reuse it.

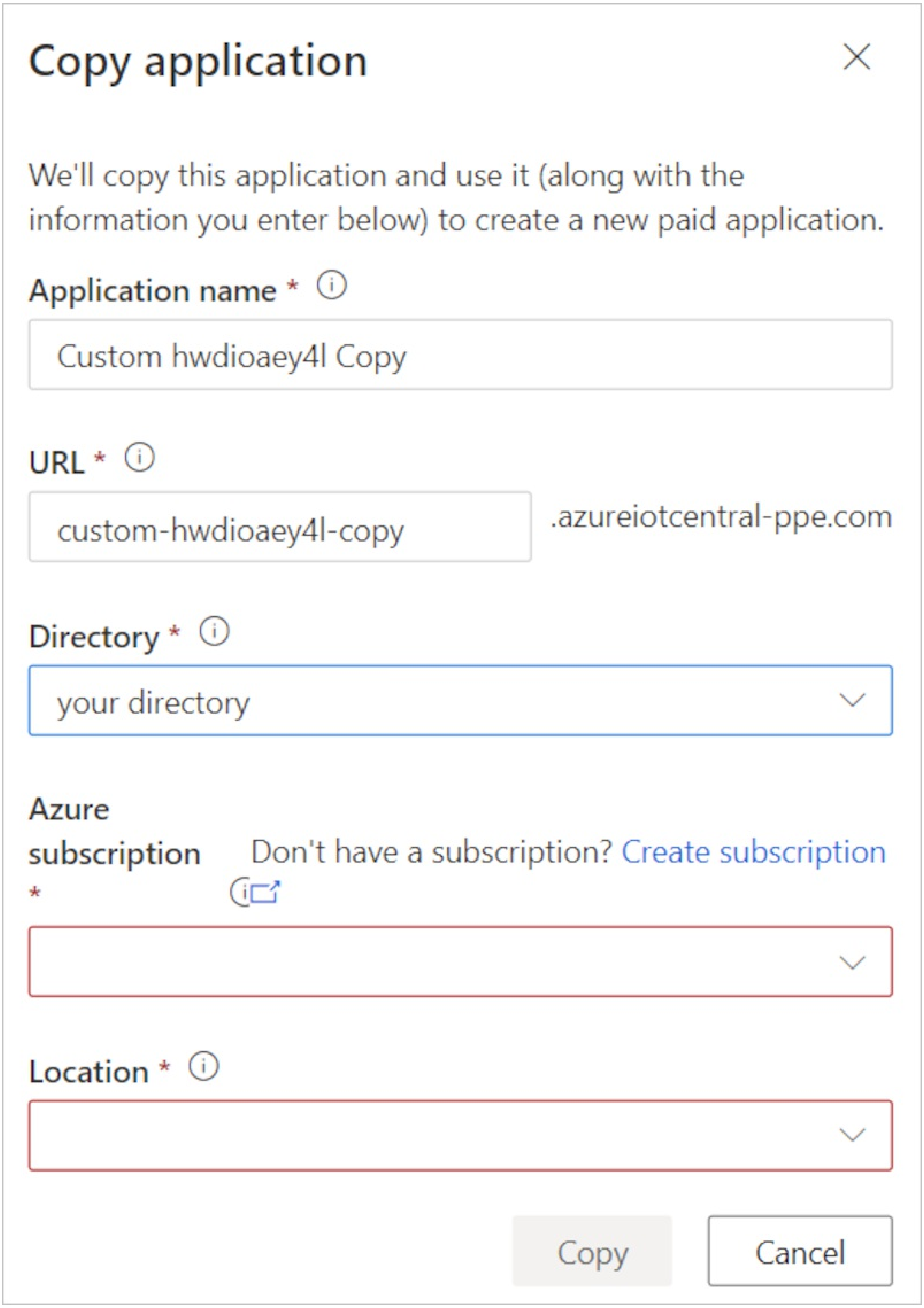
You have two options:

- You can create a copy of your application if you just need to create a duplicate copy of your application.
- You can create an application template from your application if you plan to create multiple copies.

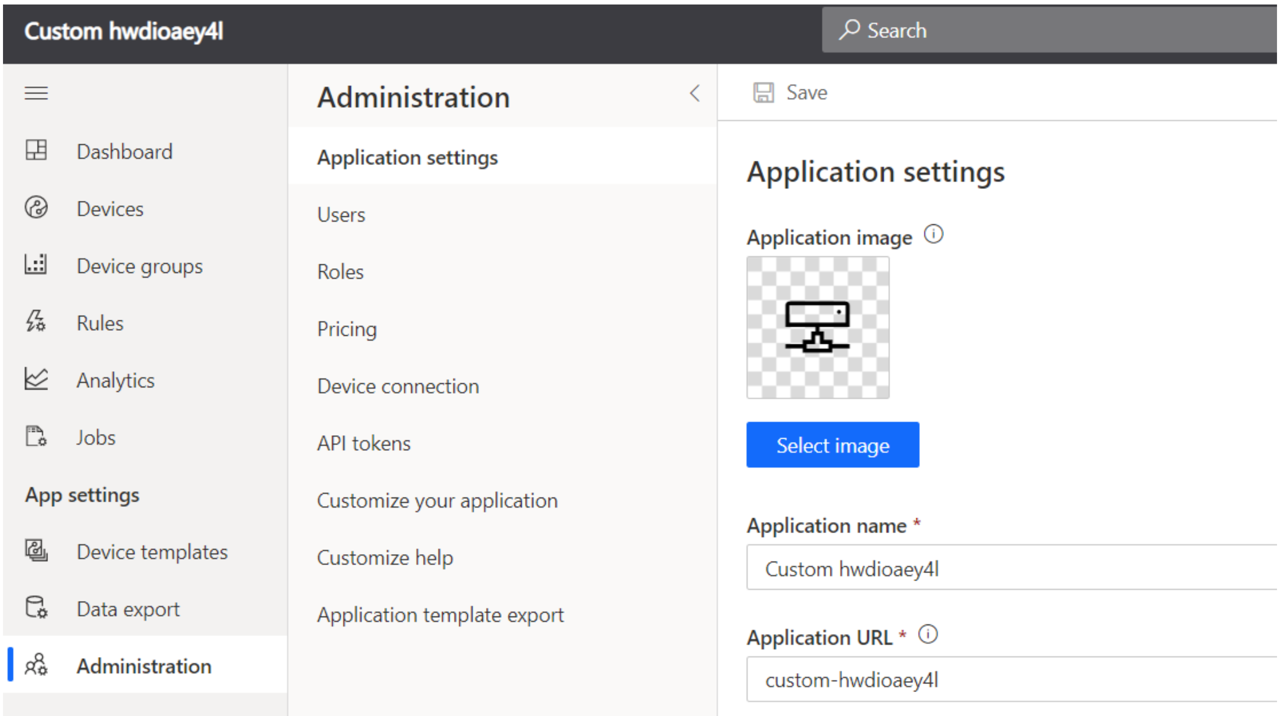
Copy your application

You can create a copy of any application, minus any device instances, device data history, and user data. The copy uses a standard pricing plan that you'll be billed for. You can't create an application that uses the free pricing plan by copying an application.

Select **Copy**. In the dialog box, enter the details for the new application. Then select **Copy** to confirm that you want to continue. To learn more about the fields in the form, see the Create an application quickstart.



After the app copy operation succeeds, you can navigate to the new application using the link.



Copying an application also copies the definition of rules and email action. Some actions, such as Flow and Logic Apps, are tied to specific rules via the Rule ID. When a rule is copied to a different application, it gets its own Rule ID. In this case, users will have to create a new action and then associate the new rule with it. In general, it's a good idea to check the rules and actions to make sure they're up-to-date in the new app.

> **WARNING**
> If a dashboard includes tiles that display information about specific devices, then those tiles show `The requested resource was not found` in the new application. You must reconfigure these tiles to display information about devices in your new application.

Create an application template

When you create an Azure IoT Central application, you have a choice of built-in sample templates. You can also create your own application templates from existing IoT Central applications. You can then use your own application templates when you create new applications.

When you create an application template, it includes the following items from your existing application:

- The default application dashboard, including the dashboard layout and all the tiles you've defined.
- Device templates, including measurements, settings, properties, commands, and dashboard.
- Rules. All rule definitions are included. However actions, except for email actions, aren't included.
- Device sets, including their conditions and dashboards.

> **WARNING**
> If a dashboard includes tiles that display information about specific devices, then those tiles show `The requested resource was not found` in the new application. You must reconfigure these tiles to display information about devices in your new application.

When you create an application template, it doesn't include the following items:

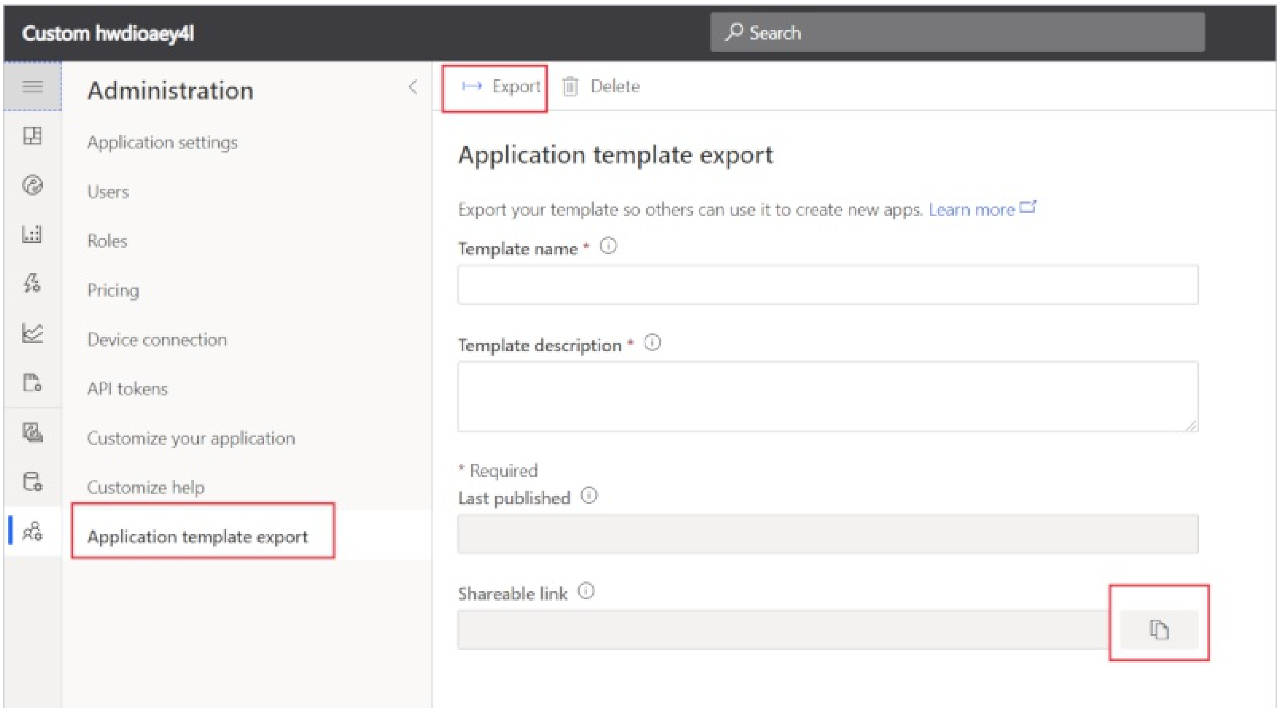
- Devices
- Users

- Continuous data export definitions

Add these items manually to any applications created from an application template.

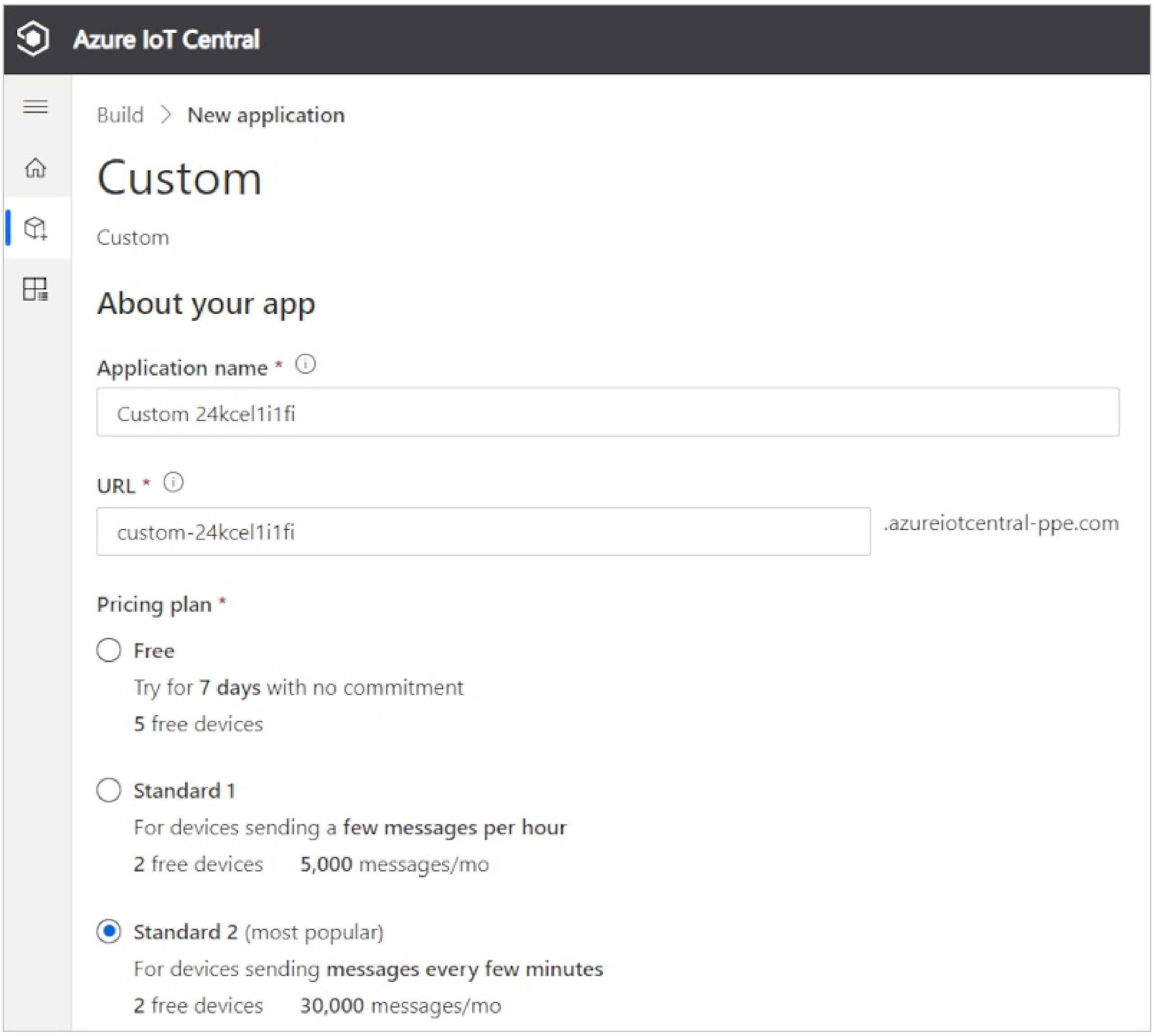
To create an application template from an existing IoT Central application:

1. Go to the **Administration** section in your application.
2. Select **Application Template Export**.
3. On the **Application Template Export** page, enter a name and description for your template.
4. Select the **Export** button to create the application template. You can now copy the **Shareable Link** that enables someone to create a new application from the template:



Use an application template

To use an application template to create a new IoT Central application, you need a previously created **Shareable Link**. Paste the **Shareable Link** into your browser's address bar. The **Create an application** page displays with your custom application template selected:



Select your pricing plan and fill out the other fields on the form. Then select **Create** to create a new IoT Central application from the application template.

Manage application templates

On the **Application Template Export** page, you can delete or update the application template.

If you delete an application template, you can no longer use the previously generated shareable link to create new applications.

To update your application template, change the template name or description on the **Application Template Export** page. Then select the **Export** button again. This action generates a new **Shareable link** and invalidates any previous **Shareable link** URL.

Next steps

Now that you've learned how to use application templates, the suggested next step is to learn how to [Monitor the overall health of the devices connected to an IoT Central application](#)

Monitor the overall health of the devices connected to an IoT Central application

7/21/2020 • 3 minutes to read • Edit Online

This article applies to operators and administrators.

In this article, you learn how to use the set of metrics provided by IoT Central to assess the overall health of the devices connected to your IoT Central application.

Metrics are enabled by default for your IoT Central application and you access them from the Azure portal. The Azure Monitor data platform exposes these metrics and provides several ways for you to interact with them. For example, you can use charts in the Azure portal, a REST API, or queries in PowerShell or the Azure CLI.

Trial applications

Applications that use the free trial plan don't have an associated Azure subscription and so don't support Azure Monitor metrics. You can convert an application to a standard pricing plan and get access to these metrics.

View metrics in the Azure portal

The following steps assume you have an IoT Central application with some connected devices.

To view IoT Central metrics in the portal:

1. Navigate to your IoT Central application resource in the portal. By default, IoT Central resources are located in a resource group called **IOTC**.
2. To create a chart from your application's metrics, select **Metrics** in the **Monitoring** section.

Azure portal permissions

Access to metrics in the Azure portal is managed by Azure role based access control. Use the Azure portal to add users to the IoT Central application/resource group/subscription to grant them access. You must add a user in the portal even they're already added to the IoT Central application. Use Azure built-in roles for finer grained access control.

IoT Central metrics

The following table describes the metrics that are currently available for IoT Central:

METRIC	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
connectedDeviceCount	Total Connected Devices	Count	Total	Number of devices connected to IoT Central
c2d.property.read.success	Successful Device Property Reads from IoT Central	Count	Total	The count of all successful property reads initiated from IoT Central

METRIC	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION
c2d.property.read.failure	Failed Device Property Reads from IoT Central	Count	Total	The count of all failed property reads initiated from IoT Central
d2c.property.read.success	Successful Device Property Reads from Devices	Count	Total	The count of all successful property reads initiated from devices
d2c.property.read.failure	Failed Device Property Reads from Devices	Count	Total	The count of all failed property reads initiated from devices
c2d.property.update.success	Successful Device Property Updates from IoT Central	Count	Total	The count of all successful property updates initiated from IoT Central
c2d.property.update.failure	Failed Device Property Updates from IoT Central	Count	Total	The count of all failed property updates initiated from IoT Central
d2c.property.update.success	Successful Device Property Updates from Devices	Count	Total	The count of all successful property updates initiated from devices
d2c.property.update.failure	Failed Device Property Updates from Devices	Count	Total	The count of all failed property updates initiated from devices

Metrics and invoices

Metrics may differ from the numbers shown on your Azure IoT Central invoice. This situation occurs for a number of reasons such as:

- IoT Central standard pricing plans include two devices and varying message quotas for free. While the free items are excluded from billing, they're still counted in the metrics.
- IoT Central autogenerates one test device ID for each device template in the application. This device ID is visible on the **Manage test device** page for a device template. Solution builders may choose to validate their device templates before publishing them by generating code that uses these test device IDs. While these devices are excluded from billing, they're still counted in the metrics.
- While metrics may show a subset of device-to-cloud communication, all communication between the device and the cloud counts as a message for billing.

Next steps

Now that you've learned how to use application templates, the suggested next step is to learn how to Manage IoT Central from the Azure portal

About your application

7/21/2020 • 2 minutes to read • Edit Online

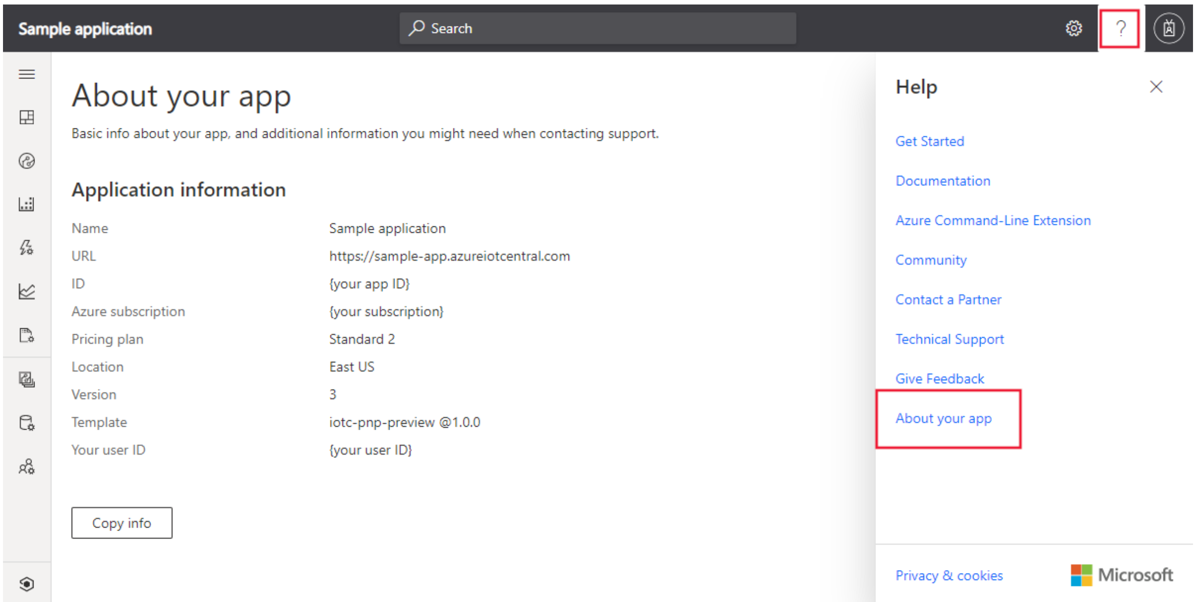
This article shows you how to get information about your IoT Central application. You may need:

- This information if you contact support.
- The Azure subscription your application uses to locate billing information in the Azure portal.
- The application's ID when you're working with the REST API.
- The application's version to complete tasks such as adding a connector.

Get information about your application

To get information about your IoT Central application:

1. Select the **Help** link on the top menu.
2. Select **About your app**.
3. The **About your app** page shows information about your application:



Use the **Copy info** button to copy the information to the clipboard.

Next steps

Now that you know how to find the version of your IoT Central application, a suggested next step is to continue exploring the how-to articles for administrators: Change IoT Central application settings.

Manage IoT Central from the Azure portal

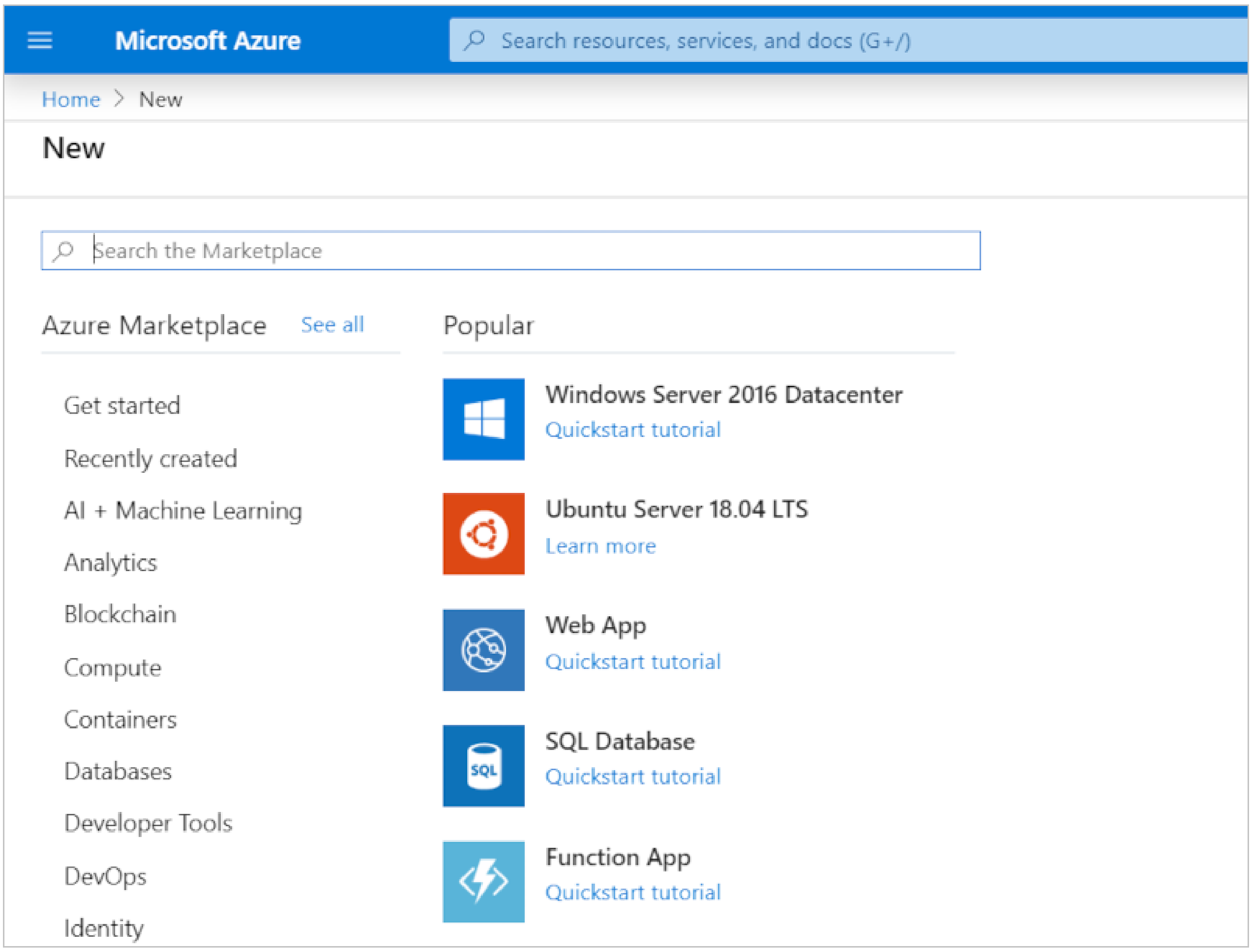
3/24/2020 • 2 minutes to read • Edit Online

Instead of creating and managing IoT Central applications on the Azure IoT Central application manager website, you can use the Azure portal to manage your applications.

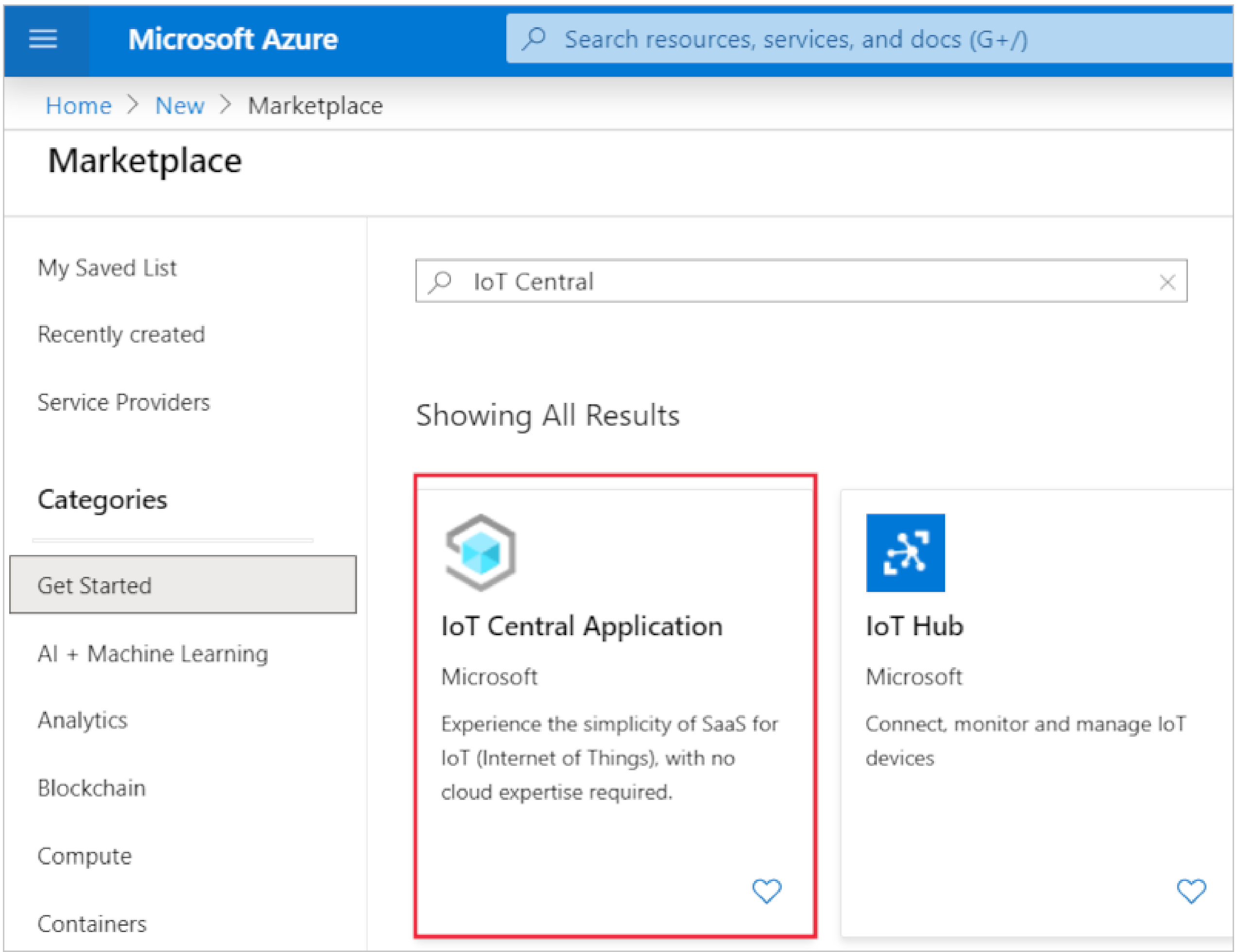
Create IoT Central applications

To create an application, navigate to the Azure portal and select **Create a resource**.

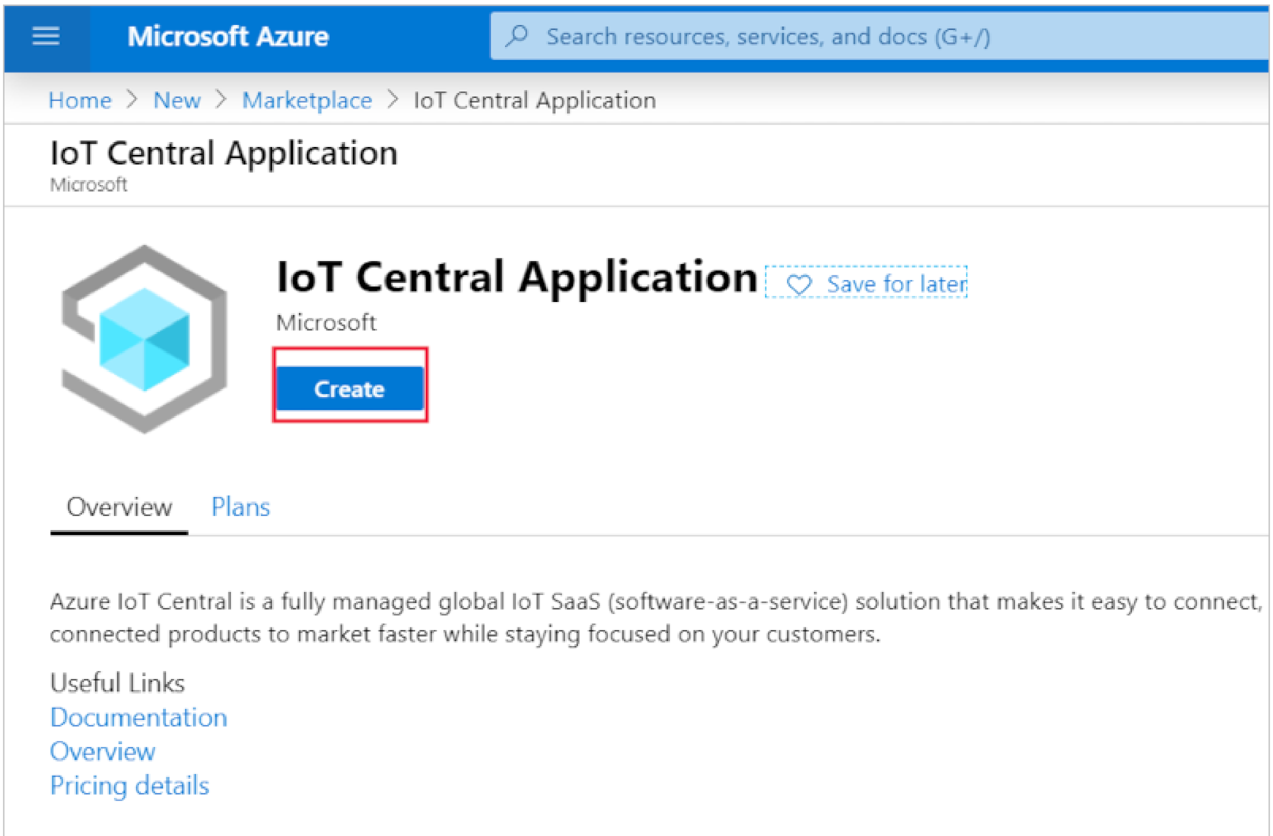
In **Search the Marketplace** bar, type *IoT Central*:



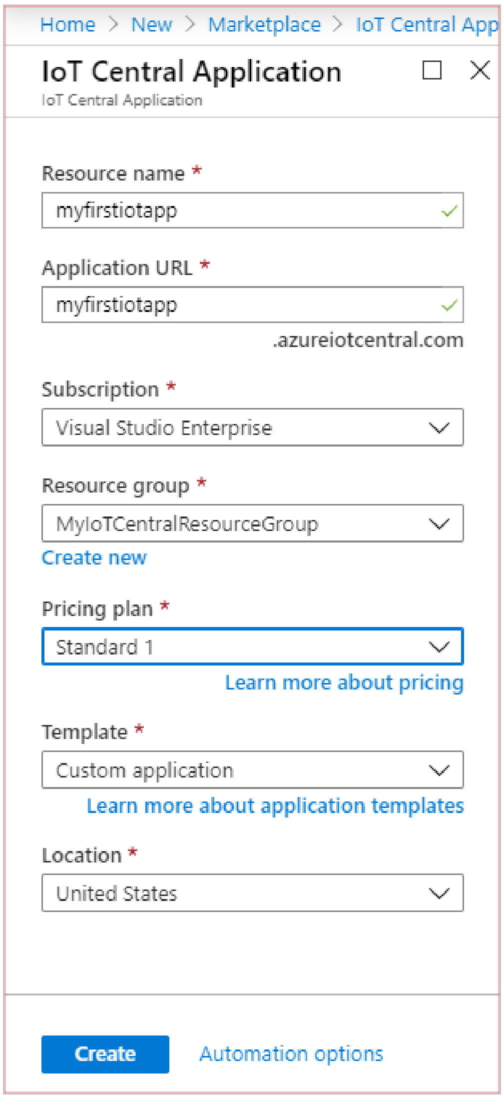
Select the **IoT Central Application** tile in the search results:



Now, select **Create**:



Fill in all the fields in the form. This form is similar to the form you fill out to create applications on the Azure IoT Central application manager website. For more information, see the Create an IoT Central application quickstart.



Location is the geography where you'd like to create your application. Typically, you should choose the location that's physically closest to your devices to get optimal performance. Azure IoT Central is currently available in the **Australia**, **Asia Pacific**, **Europe**, **United States**, **United Kingdom**, and **Japan** geographies. Once you choose a location, you can't move your application to a different location later.

After filling out all fields, select **Create**.

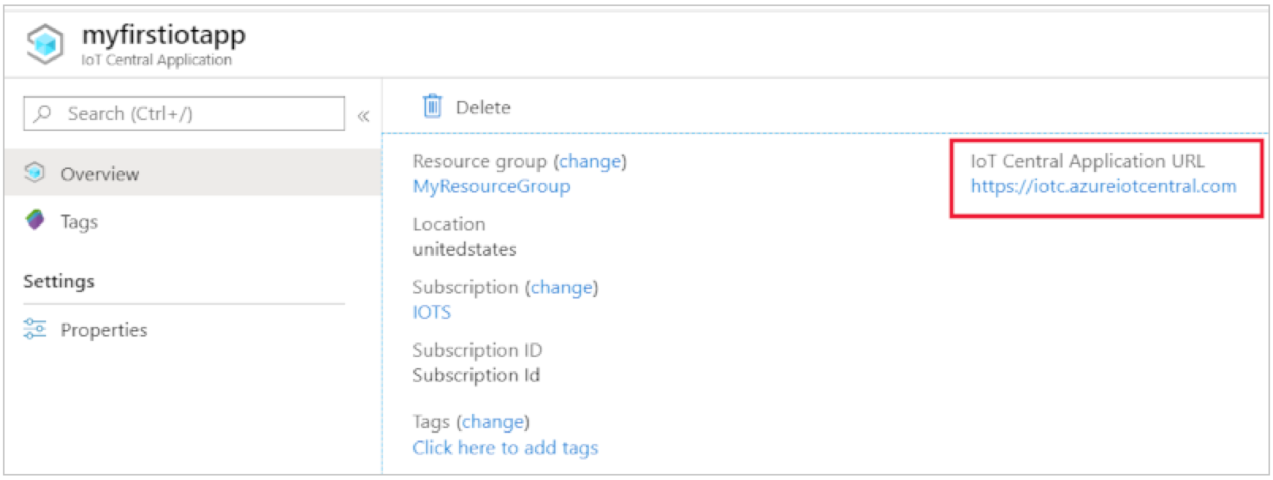
Manage existing IoT Central applications

If you already have an Azure IoT Central application you can delete it, or move it to a different subscription or resource group in the Azure portal.

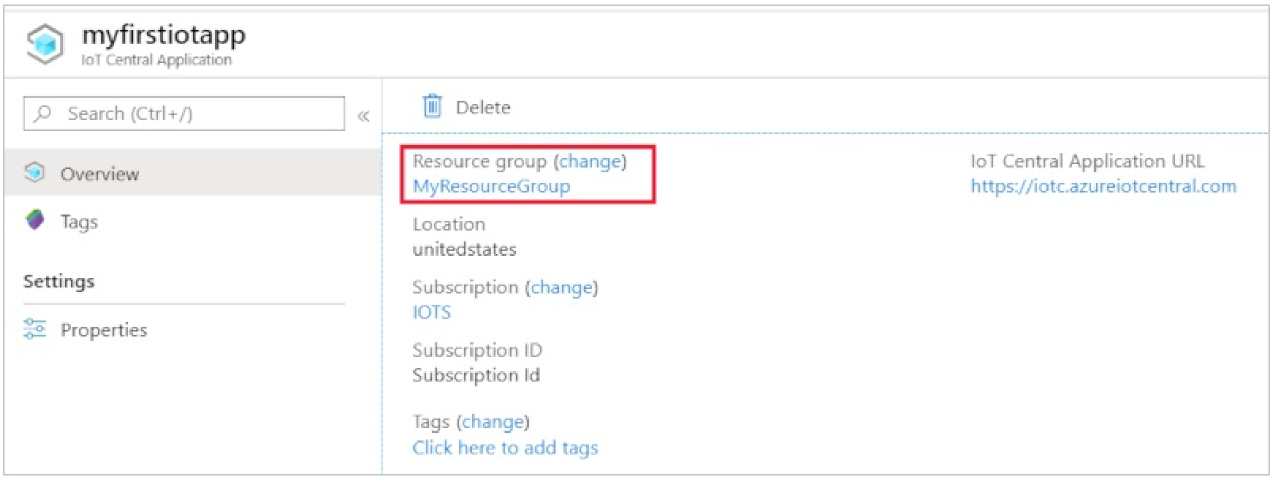
> **NOTE**
>
> You can't see applications created on the free pricing plan in the Azure portal because they are not associated with your subscription.

To get started, select **All resources** in the portal. Select **Show hidden types** and start typing the name of your application in **Filter by name** to find it. Then select the IoT Central application you'd like to manage.

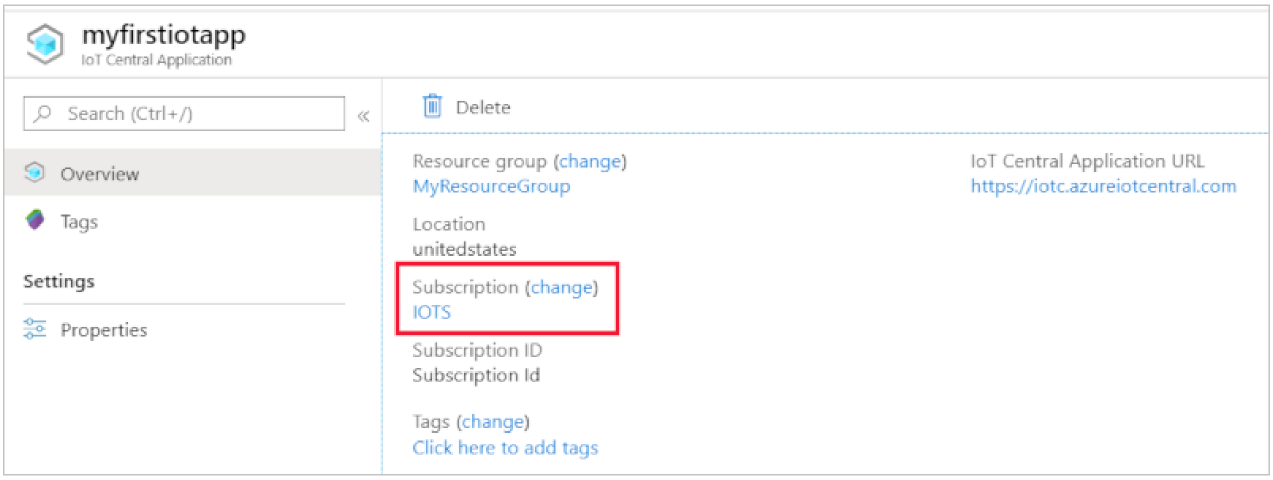
To navigate to the application, select the **IoT Central Application URL**:



To move the application to a different resource group, select **change** beside the resource group. On the **Move resources** page, choose the resource group you'd like to move this application to:



To move the application to a different subscription, select **change** beside the subscription. On the **Move resources** page, choose the subscription you'd like to move this application to:



Next steps

Now that you've learned how to manage Azure IoT Central applications from the Azure portal, here is the suggested next step:

Administer your application

Manage IoT Central from Azure CLI

5/5/2020 • 4 minutes to read • Edit Online

Instead of creating and managing IoT Central applications on the Azure IoT Central application manager website, you can use Azure CLI to manage your applications.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

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 https://shell.azure.com, 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 Azure portal.	

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.

If you prefer to run Azure CLI on your local machine, see Install the Azure CLI. When you run Azure CLI locally, use the **az login** command to sign in to Azure before you try the commands in this article.

> **TIP**
> If you need to run your CLI commands in a different Azure subscription, see Change the active subscription.

Install the extension

The commands in this article are part of the **azure-iot** CLI extension. Run the following command to install the extension:

```
az extension add --name azure-iot
```

Create an application

Use the az iot central app create command to create an IoT Central application in your Azure subscription. For example:

```
# Create a resource group for the IoT Central application
az group create --location "East US" \
    --name "MyIoTCentralResourceGroup"
```

```
# Create an IoT Central application
az iot central app create \
  --resource-group "MyIoTCentralResourceGroup" \
  --name "myiotcentralapp" --subdomain "mysubdomain" \
  --sku ST1 --template "iotc-pnp-preview" \
  --display-name "My Custom Display Name"
```

These commands first create a resource group in the east US region for the application. The following table describes the parameters used with the **az iot central app create** command:

PARAMETER	DESCRIPTION
resource-group	The resource group that contains the application. This resource group must already exist in your subscription.
location	By default, this command uses the location from the resource group. Currently, you can create an IoT Central application in the **Australia**, **Asia Pacific**, **Europe**, **United States**, **United Kingdom**, and **Japan** geographies.
name	The name of the application in the Azure portal.
subdomain	The subdomain in the URL of the application. In the example, the application URL is `https://mysubdomain.azureiotcentral.com`.
sku	Currently, you can use either **ST1** or **ST2**. See Azure IoT Central pricing.
template	The application template to use. For more information, see the following table.
display-name	The name of the application as displayed in the UI.

Application templates

TEMPLATE	NAME	DESCRIPTION
iotc-pnp-preview	Custom application	Creates an empty application for you to populate with your own device templates and devices.

TEMPLATE	NAME	DESCRIPTION
iotc-default	Custom application (legacy)	Creates an empty legacy application for you to populate with your own device templates and devices.
iotc-condition	In-store Analytics – Condition Monitoring	Creates an application to connect and monitor a store environment.
iotc-consumption	Water Consumption Monitoring	Creates an application to monitor and control water flow.
iotc-distribution	Digital Distribution Center	Creates an application to improve warehouse output efficiency by digitizing key assets and actions.
iotc-inventory	Smart Inventory Management	Creates an application to automate receiving, product movement, cycle counting, and tracking.
iotc-logistics	Connected Logistics	Creates an application to track your shipments in real time across air, water, and land with location and condition monitoring.
iotc-meter	Smart Meter Analytics	Creates an application to monitor energy consumption, network status, and identify trends to improve customer support and smart meter management.
iotc-mfc	Micro-fulfillment Center	Creates an application to digitally connect and manage a fully automated fulfillment center.
iotc-patient	Continuous Patient Monitoring	Creates an application to extend patient care, reduce readmissions, and manage diseases.
iotc-power	Solar Power Monitoring	Creates an application to monitor solar panel status and energy generation trends.
iotc-quality	Water Quality Monitoring	Creates an application to digitally monitor water quality.
iotc-store	In-store Analytics – Checkout	Creates an application to monitor and manage the checkout flow inside your store.
iotc-waste	Connected Waste Management	Creates an application to monitor waste bins and dispatch field operators.

View your applications

Use the az iot central app list command to list your IoT Central applications and view metadata.

Modify an application

Use the az iot central app update command to update the metadata of an IoT Central application. For example, to change the display name of your application:

```
az iot central app update --name myiotcentralapp \
  --resource-group MyIoTCentralResourceGroup \
  --set displayName="My new display name"
```

Remove an application

Use the az iot central app delete command to delete an IoT Central application. For example:

```
az iot central app delete --name myiotcentralapp \
  --resource-group MyIoTCentralResourceGroup
```

Next steps

Now that you've learned how to manage Azure IoT Central applications from Azure CLI, here is the suggested next step:

Administer your application

Manage IoT Central from Azure PowerShell

5/5/2020 • 4 minutes to read • Edit Online

Instead of creating and managing IoT Central applications on the Azure IoT Central application manager website, you can use Azure PowerShell to manage your applications.

Prerequisites

If you don't have an Azure subscription, create a free account before you begin.

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 https://shell.azure.com, 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 Azure portal.	

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.

If you prefer to run Azure PowerShell on your local machine, see Install the Azure PowerShell module. When you run Azure PowerShell locally, use the **Connect-AzAccount** cmdlet to sign in to Azure before you try the cmdlets in this article.

> **TIP**
>
> If you need to run your PowerShell commands in a different Azure subscription, see Change the active subscription.

Install the IoT Central module

Run the following command to check the IoT Central module is installed in your PowerShell environment:

```
Get-InstalledModule -name Az.I*
```

If the list of installed modules doesn't include **Az.IotCentral**, run the following command:

```
Install-Module Az.IotCentral
```

Create an application

Use the New-AzIotCentralApp cmdlet to create an IoT Central application in your Azure subscription. For example:

```
# Create a resource group for the IoT Central application
New-AzResourceGroup -ResourceGroupName "MyIoTCentralResourceGroup" `
  -Location "East US"
```

```
# Create an IoT Central application
New-AzIotCentralApp -ResourceGroupName "MyIoTCentralResourceGroup" `
  -Name "myiotcentralapp" -Subdomain "mysubdomain" `
  -Sku "ST1" -Template "iotc-pnp-preview" `
  -DisplayName "My Custom Display Name"
```

The script first creates a resource group in the east US region for the application. The following table describes the parameters used with the **New-AzIotCentralApp** command:

PARAMETER	DESCRIPTION
ResourceGroupName	The resource group that contains the application. This resource group must already exist in your subscription.
Location	By default, this cmdlet uses the location from the resource group. Currently, you can create an IoT Central application in the **Australia**, **Asia Pacific**, **Europe**, **United States**, **United Kingdom**, and **Japan** geographies.
Name	The name of the application in the Azure portal.
Subdomain	The subdomain in the URL of the application. In the example, the application URL is `https://mysubdomain.azureiotcentral.com`.
Sku	Currently, you can use either **ST1** or **ST2**. See Azure IoT Central pricing.
Template	The application template to use. For more information, see the following table.
DisplayName	The name of the application as displayed in the UI.

Application templates

TEMPLATE	NAME	DESCRIPTION

TEMPLATE	NAME	DESCRIPTION
iotc-pnp-preview	Custom application	Creates an empty application for you to populate with your own device templates and devices.
iotc-default	Custom application (legacy)	Creates an empty legacy application for you to populate with your own device templates and devices.
iotc-condition	In-store Analytics – Condition Monitoring	Creates an application to connect and monitor a store environment.
iotc-consumption	Water Consumption Monitoring	Creates an application to monitor and control water flow.
iotc-distribution	Digital Distribution Center	Creates an application to improve warehouse output efficiency by digitizing key assets and actions.
iotc-inventory	Smart Inventory Management	Creates an application to automate receiving, product movement, cycle counting, and tracking.
iotc-logistics	Connected Logistics	Creates an application to track your shipments in real time across air, water, and land with location and condition monitoring.
iotc-meter	Smart Meter Analytics	Creates an application to monitor energy consumption, network status, and identify trends to improve customer support and smart meter management.
iotc-mfc	Micro-fulfillment Center	Creates an application to digitally connect and manage a fully automated fulfillment center.
iotc-patient	Continuous Patient Monitoring	Creates an application to extend patient care, reduce readmissions, and manage diseases.
iotc-power	Solar Power Monitoring	Creates an application to monitor solar panel status and energy generation trends.
iotc-quality	Water Quality Monitoring	Creates an application to digitally monitor water quality.
iotc-store	In-store Analytics – Checkout	Creates an application to monitor and manage the checkout flow inside your store.
iotc-waste	Connected Waste Management	Creates an application to monitor waste bins and dispatch field operators.

View your IoT Central applications

Use the Get-AzIotCentralApp cmdlet to list your IoT Central applications and view metadata.

Modify an application

Use the Set-AzIotCentralApp cmdlet to update the metadata of an IoT Central application. For example, to change the display name of your application:

```
Set-AzIotCentralApp -Name "myiotcentralapp" `
  -ResourceGroupName "MyIoTCentralResourceGroup" `
  -DisplayName "My new display name"
```

Remove an application

Use the Remove-AzIotCentralApp cmdlet to delete an IoT Central application. For example:

```
Remove-AzIotCentralApp -ResourceGroupName "MyIoTCentralResourceGroup" `
 -Name "myiotcentralapp"
```

Next steps

Now that you've learned how to manage Azure IoT Central applications from Azure PowerShell, here is the suggested next step:

Administer your application

Manage IoT Central programmatically

5/21/2020 • 2 minutes to read • Edit Online

Instead of creating and managing IoT Central applications on the Azure IoT Central application manager website, you can manage your applications programmatically using the Azure SDKs. Supported languages include JavaScript, Python, C#, Ruby, and Go.

Install the SDK

The following table lists the SDK repositories and package installation commands:

SDK REPOSITORY	PACKAGE INSTALL
Azure IotCentralClient SDK for JavaScript	`npm install @azure/arm-iotcentral`
Microsoft Azure SDK for Python	`pip install azure-mgmt-iotcentral`
Azure SDK for .NET	`dotnet add package Microsoft.Azure.Management.IotCentral`
Microsoft Azure SDK for Ruby - Resource Management (preview)	`gem install azure_mgmt_iot_central`
Azure SDK for Java	Maven package
Azure SDK for Go	Package releases

Samples

The Azure IoT Central ARM SDK samples repository has code samples for multiple programming languages that show you how to create, update, list, and delete Azure IoT Central applications.

Next steps

Now that you've learned how to manage Azure IoT Central applications programmatically, a suggested next step is to learn more about the Azure Resource Manager service.

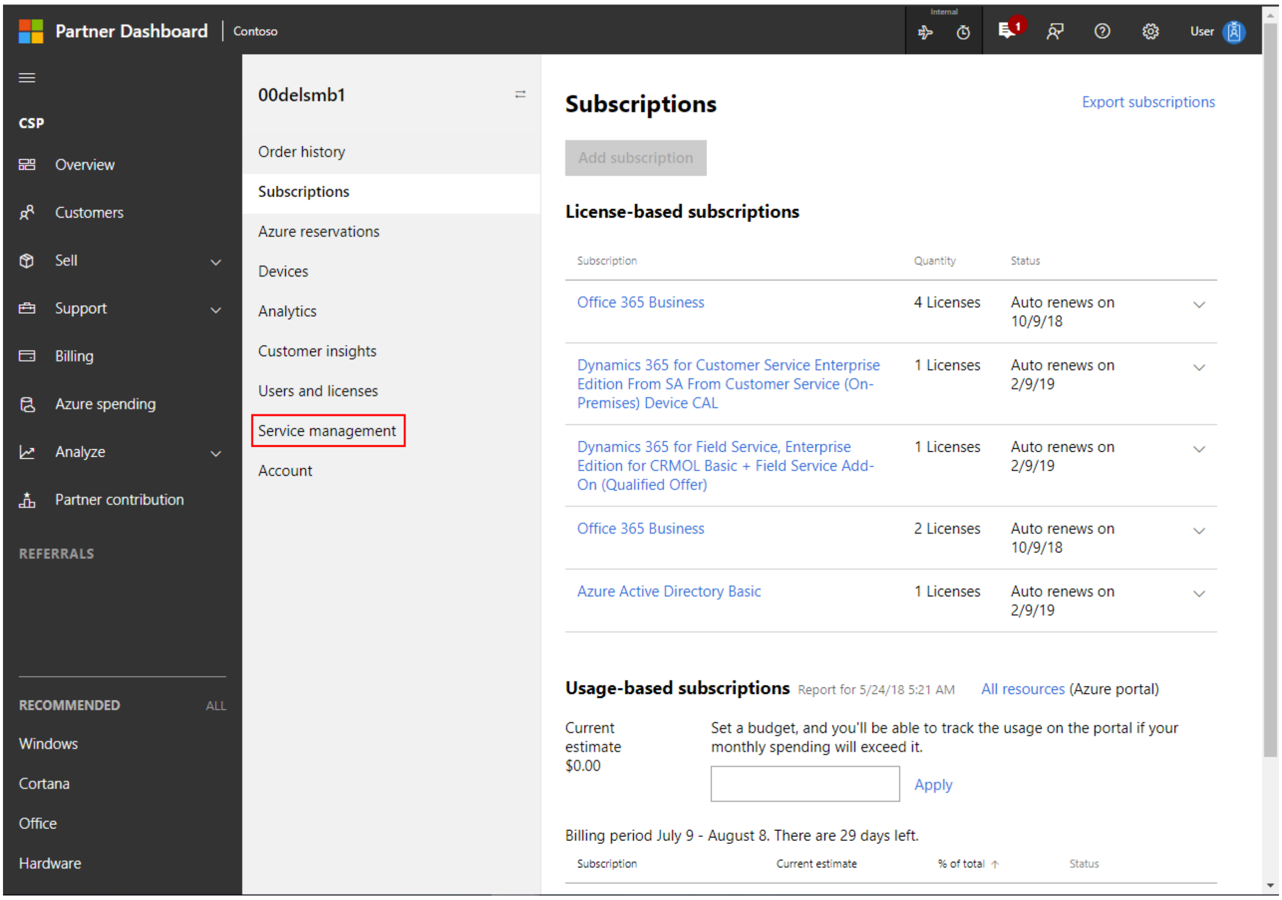
Create and manage an Azure IoT Central application from the CSP portal

4/8/2020 • 3 minutes to read • Edit Online

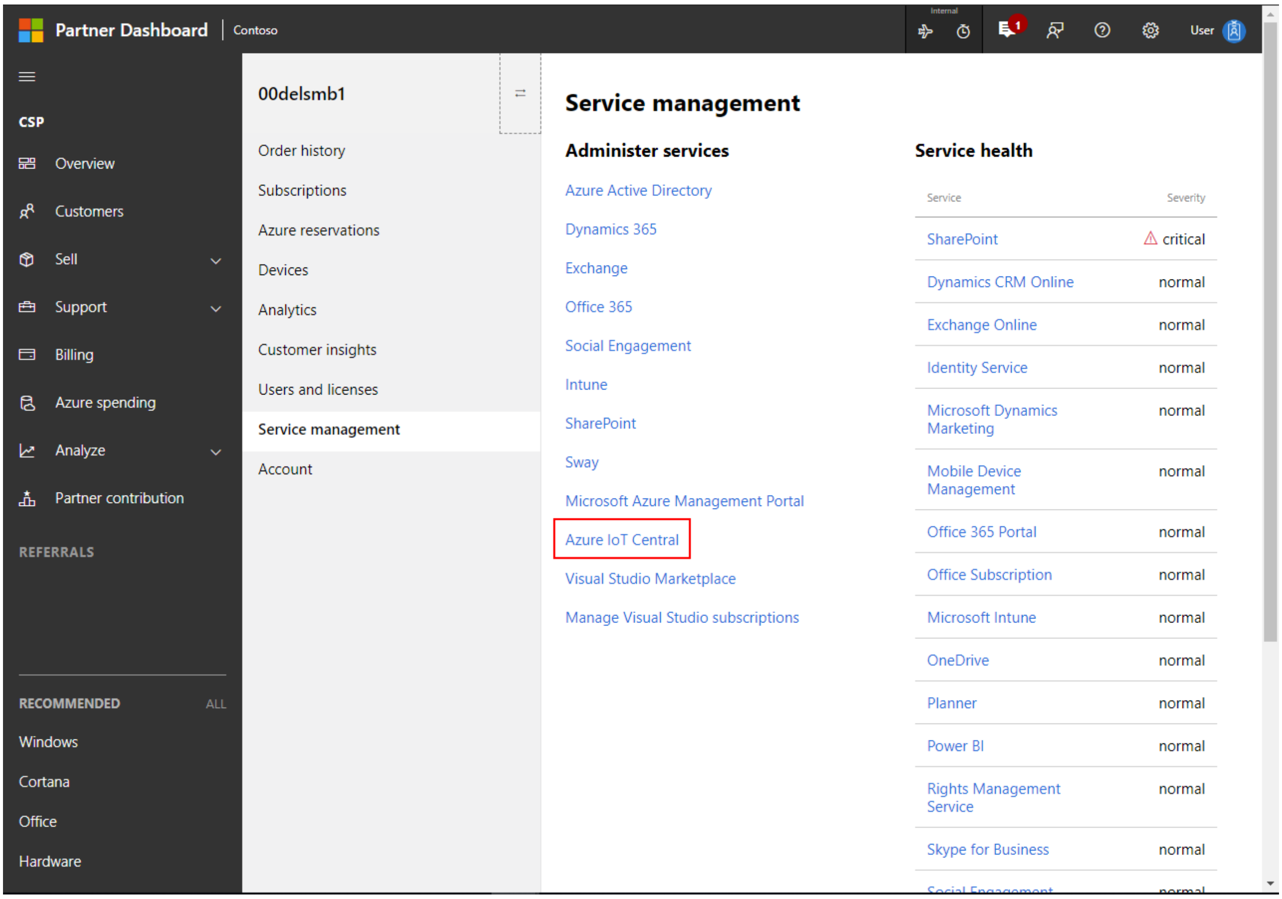
The Microsoft Cloud Solution Provider (CSP) program is a Microsoft Reseller program. Its intent is to provide our channel partners with a one-stop program to resell all Microsoft Commercial Online Services. Learn more about the Cloud Solution Provider program.

As a CSP, you can create and manage Microsoft Azure IoT Central applications on behalf of your customers through the Microsoft Partner Center. When Azure IoT Central applications are created on behalf of customers by CSPs, just like with other CSP managed Azure services, CSPs manage billing for customers. A charge for Azure IoT Central will appear in your total bill in the Microsoft Partner Center.

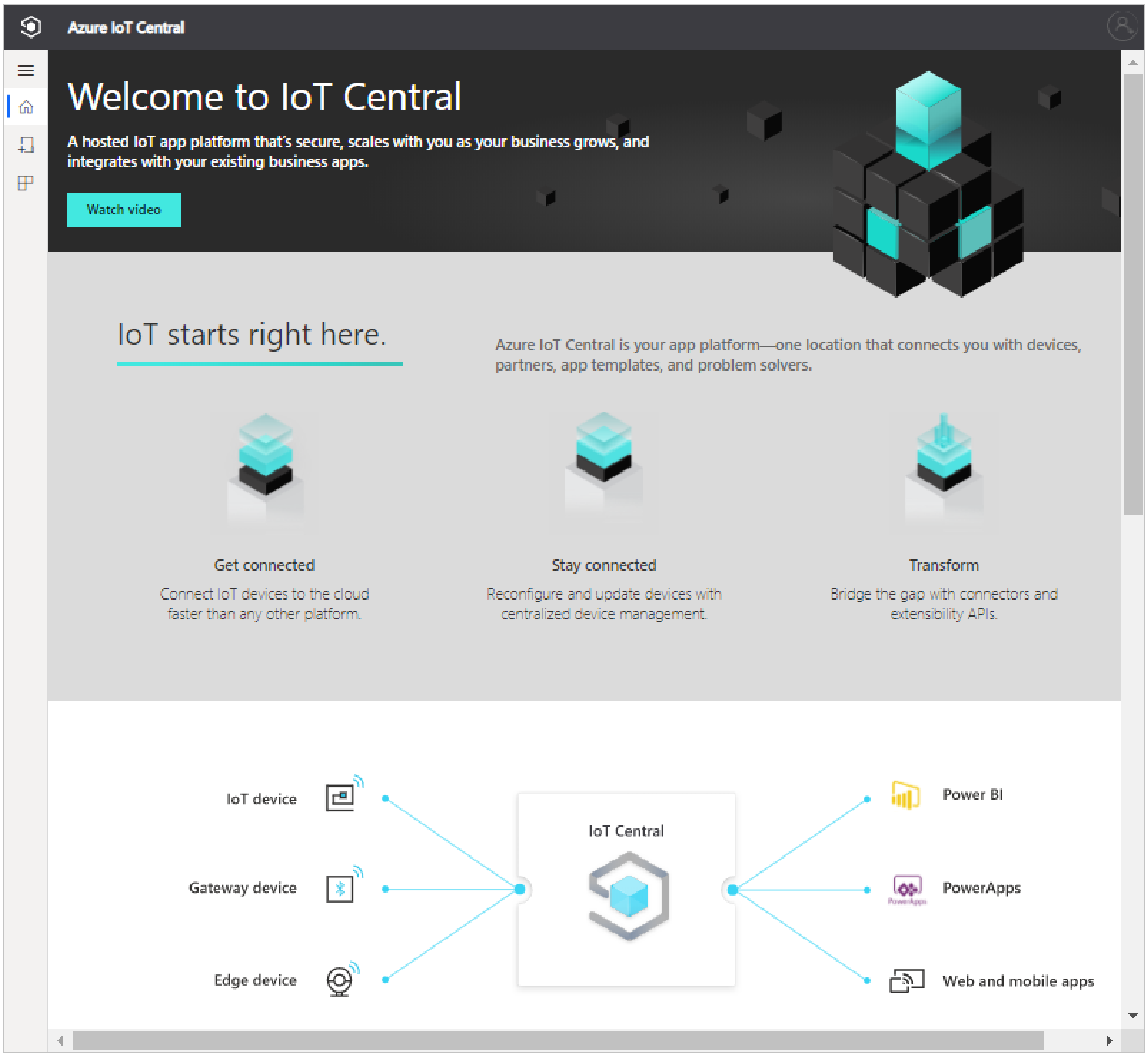
To get started, sign-in to your account on the Microsoft Partner Portal and select a customer for whom you want to create an Azure IoT Central application. Navigate to Service Management for the customer from the left nav.



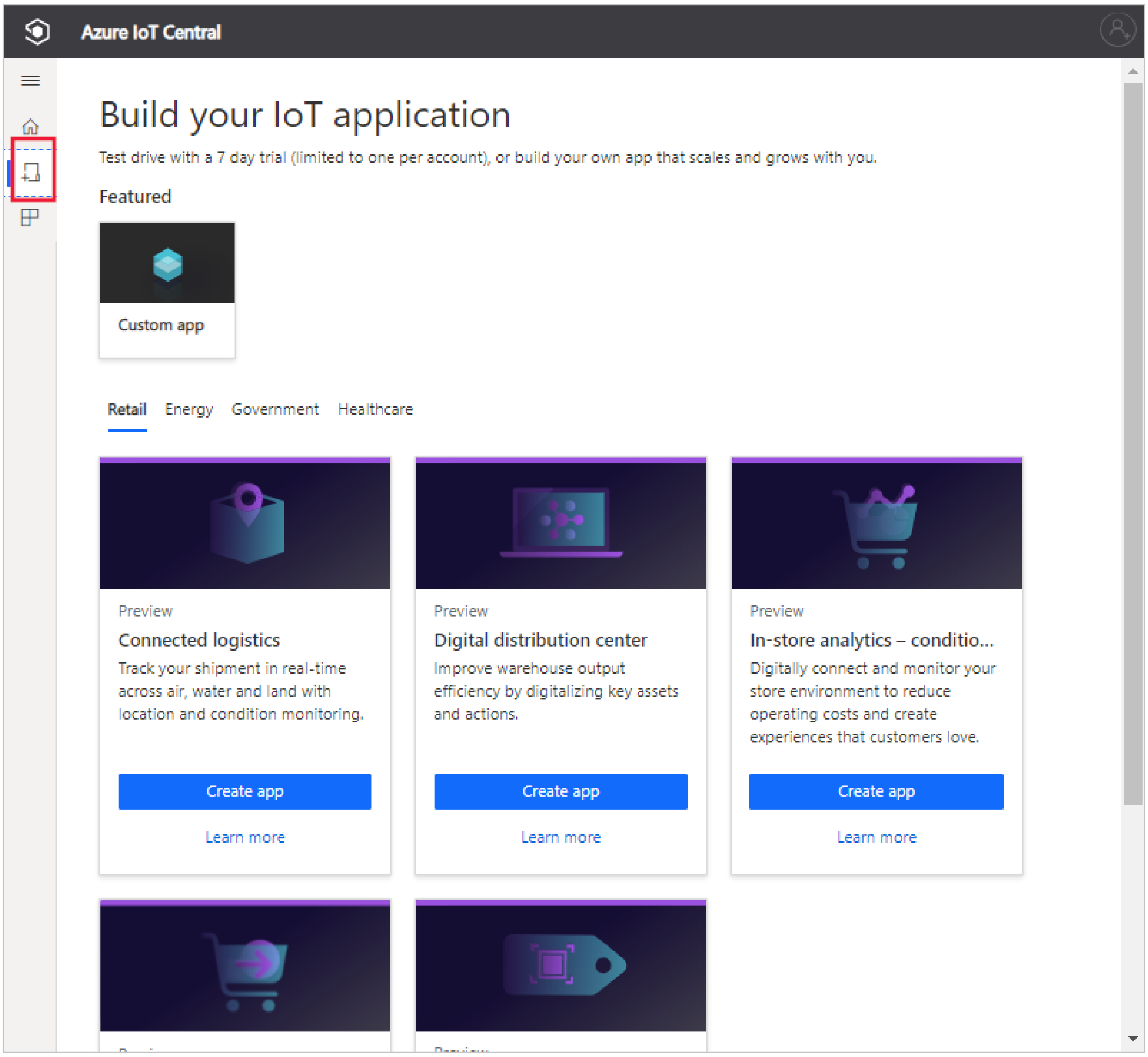
Azure IoT Central is listed as a service available to administer. Select the Azure IoT Central link on the page to create new applications or manage existing applications for this customer.

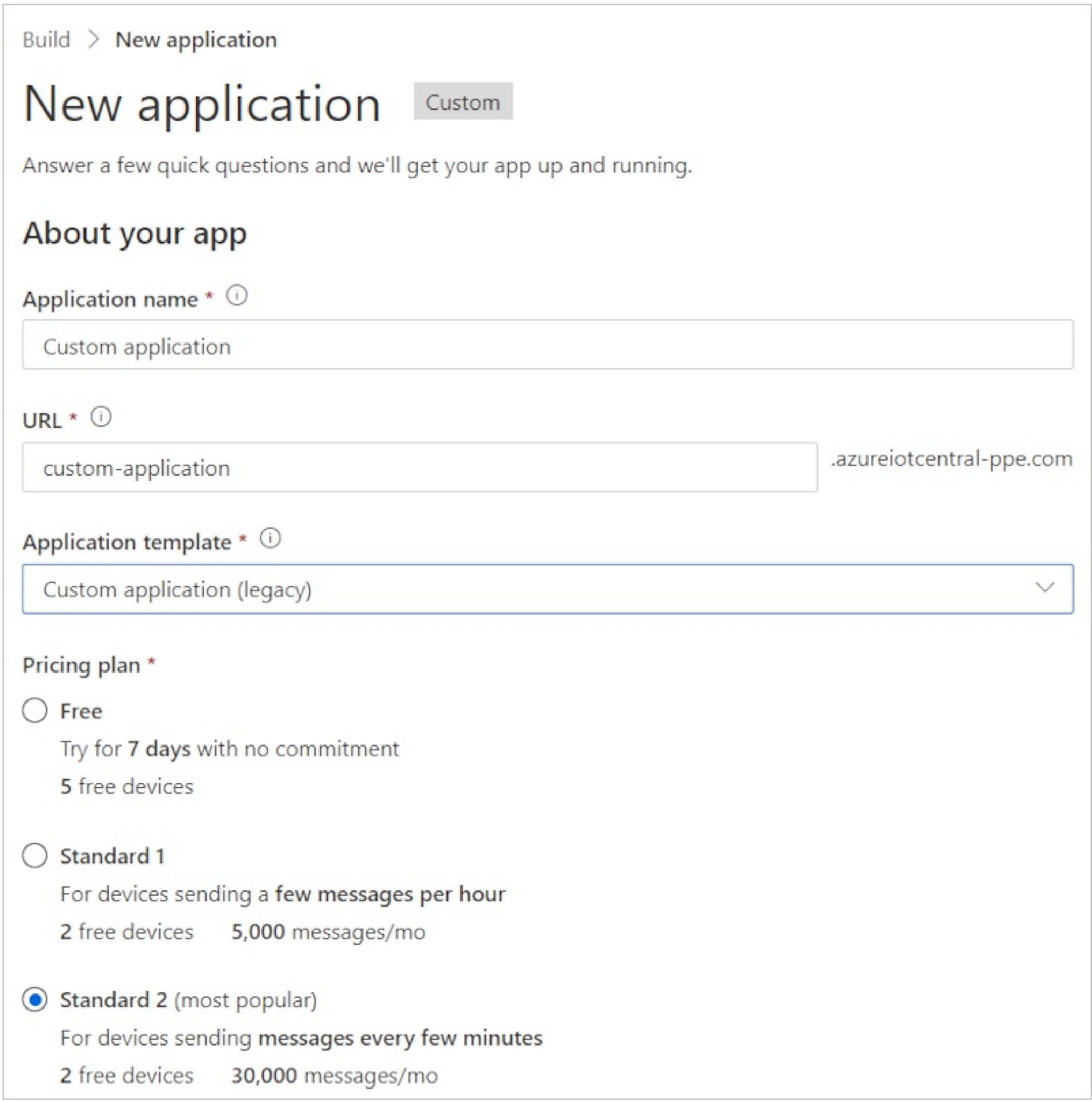


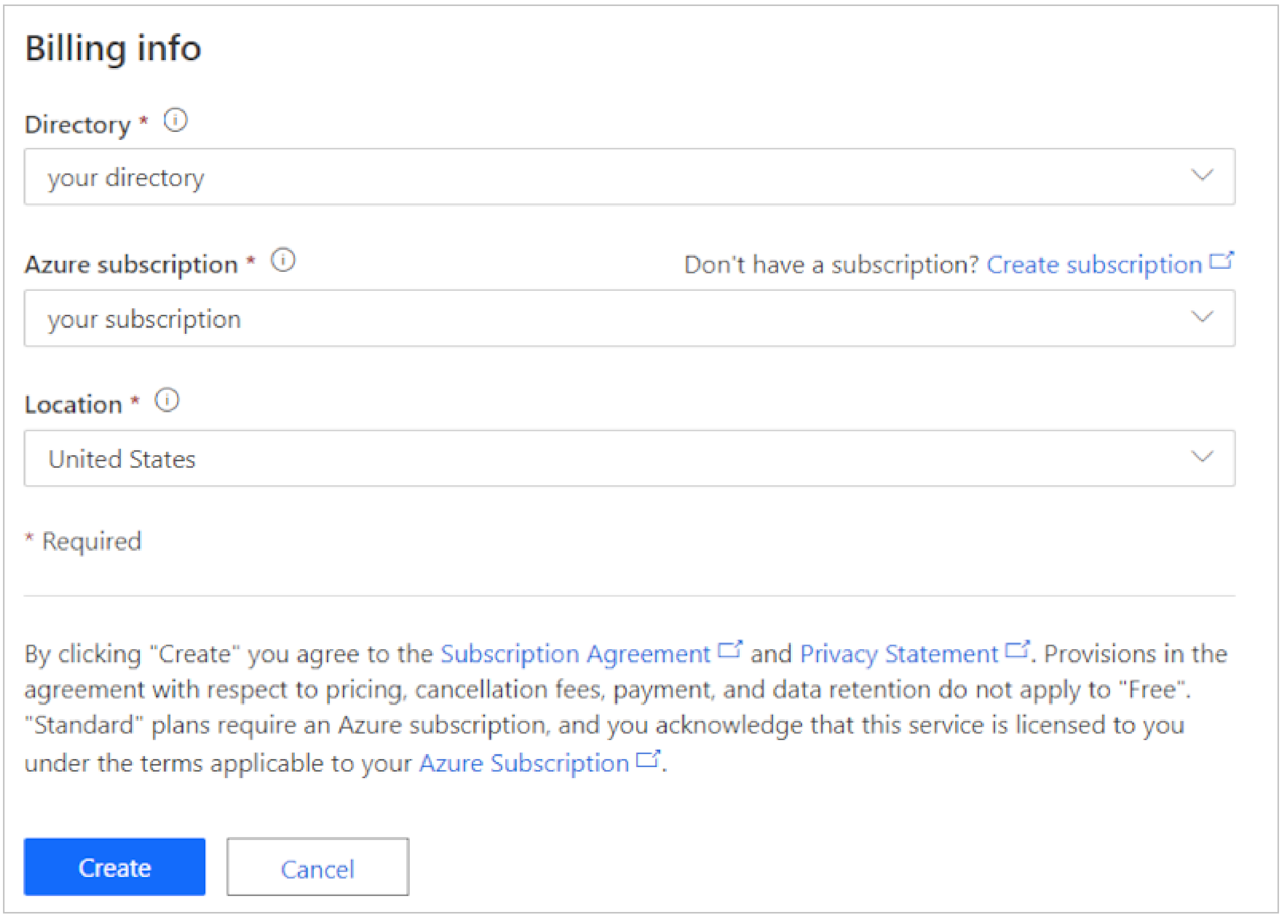
You land on the Azure IoT Central Application Manager page. Azure IoT Central keeps context that you came from the Microsoft Partner Center and that you came to manage that particular customer. You see this acknowledged in the header of the Application Manager page. From here, you can either navigate to an existing application you had created earlier for this customer to manage or create a new application for the customer.



To create an Azure IoT Central application, select **Build** in the left menu. Choose one of the industry templates, or choose **Custom app** to create an application from scratch. This will load the Application Creation page. You must complete all the fields on this page and then choose **Create**. You find more information about each of the fields below.


Azure IoT Central
Build your IoT application
Test drive with a 7 day trial (limited to one per account), or build your own app that scales and grows with you.
Featured
Custom app
Retail
Energy
Government
Healthcare
Preview
Connected logistics
Track your shipment in real-time across air, water and land with location and condition monitoring.
Create app
Learn more
Preview
Digital distribution center
Improve warehouse output efficiency by digitalizing key assets and actions.
Create app
Learn more
Preview
In-store analytics – conditio...
Digitally connect and monitor your store environment to reduce operating costs and create experiences that customers love.
Create app
Learn more


Build > New application
New application
Custom
Answer a few quick questions and we'll get your app up and running.
About your app
Application name *
Custom application
URL *
custom-application
.azureiotcentral-ppe.com
Application template *
Custom application (legacy)
Pricing plan *
Free
Try for 7 days with no commitment
5 free devices
Standard 1
For devices sending a few messages per hour
2 free devices
5,000 messages/mo
Standard 2 (most popular)
For devices sending messages every few minutes
2 free devices
30,000 messages/mo



Pricing plan

You can only create applications that use a standard pricing plan as a CSP. To showcase Azure IoT Central to your customer, you can create an application that uses the free pricing plan separately. Learn more about the free and standard pricing plans on the Azure IoT Central pricing page.

You can only create applications that use a standard pricing plan as a CSP. To showcase Azure IoT Central to your customer, you can create an application that uses the free pricing plan separately. Learn more about the free and standard pricing plans on the Azure IoT Central pricing page.

Application name

The name of your application is displayed on the **Application Manager** page and within each Azure IoT Central application. You can choose any name for your Azure IoT Central application. Choose a name that makes sense to you and to others in your organization.

Application URL

The application URL is the link to your application. You can save a bookmark to it in your browser or share it with others.

When you enter the name for your application, your application URL is autogenerated. If you prefer, you can choose a different URL for your application. Each Azure IoT Central URL must be unique within Azure IoT Central. You see an error message if the URL you choose has already been taken.

Directory

Since Azure IoT Central has context that you came to manage the customer you selected in the Microsoft Partner Portal, you see just the Azure Active Directory tenant for that customer in the Directory field.

An Azure Active Directory tenant contains user identities, credentials, and other organizational information. Multiple Azure subscriptions can be associated with a single Azure Active Directory tenant.

To learn more, see Azure Active Directory.

Azure subscription

An Azure subscription enables you to create instances of Azure services. Azure IoT Central automatically finds all Azure Subscriptions of the customer to which you have access, and displays them in a dropdown on the **Create Application** page. Choose an Azure subscription to create a new Azure IoT Central Application.

If you don't have an Azure subscription, you can create one in the Microsoft Partner Center. After you create the Azure subscription, navigate back to the **Create Application** page. Your new subscription appears in the **Azure Subscription** drop-down.

To learn more, see Azure subscriptions.

Location

Location is the geography where you'd like to create the application. Typically, you should choose the location that's physically closest to your devices to get optimal performance. Currently, you can create an IoT Central application in the **Australia**, **Asia Pacific**, **Europe**, **United States**, **United Kingdom**, and **Japan** geographies. Once you choose a location, you can't later move your application to a different location.

Application template

Choose the application template you want to use for your application.

Next steps

Now that you have learned how to create an Azure IoT Central application as a CSP, here is the suggested next step:

Administer your application

Manage your personal application preferences

3/24/2020 • 2 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

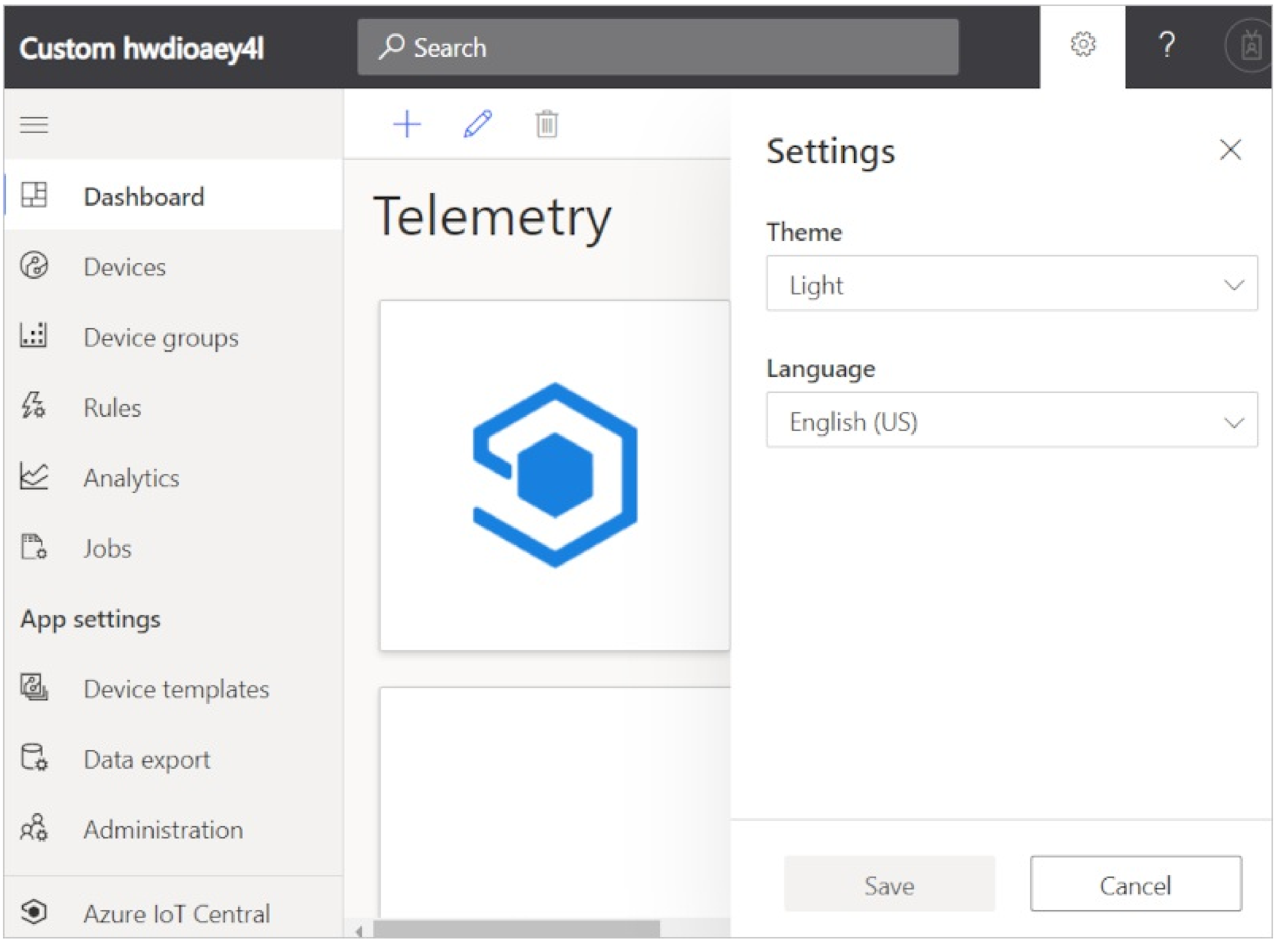
IoT Central provides the flexibility to customize your applications to fit your need. We also provide some flexibility on a per-user basis to customize your own view. This article describes the various customization options that a user can apply to their profile.

Changing language

IoT Central is supported in multiple languages. You can switch your preferred language by using the **language picker** on the settings icon on the top navigation bar. Once you've changed your language, IoT Central remembers your selection and applies it across all your applications. Customization within the application such dashboard images aren't localized.

Changing theme

We have support for both dark theme and light theme. While the light theme is the default, you can change the theme by selecting the settings icon on the top navigation bar.



NOTE

The option to choose between light and dark themes isn't available if your administrator has configured a custom theme for the application.

Next steps

Now that you've learned how to manage your profile in Azure IoT Central, here is the suggested next step:

Toggle live chat

Toggle live chat

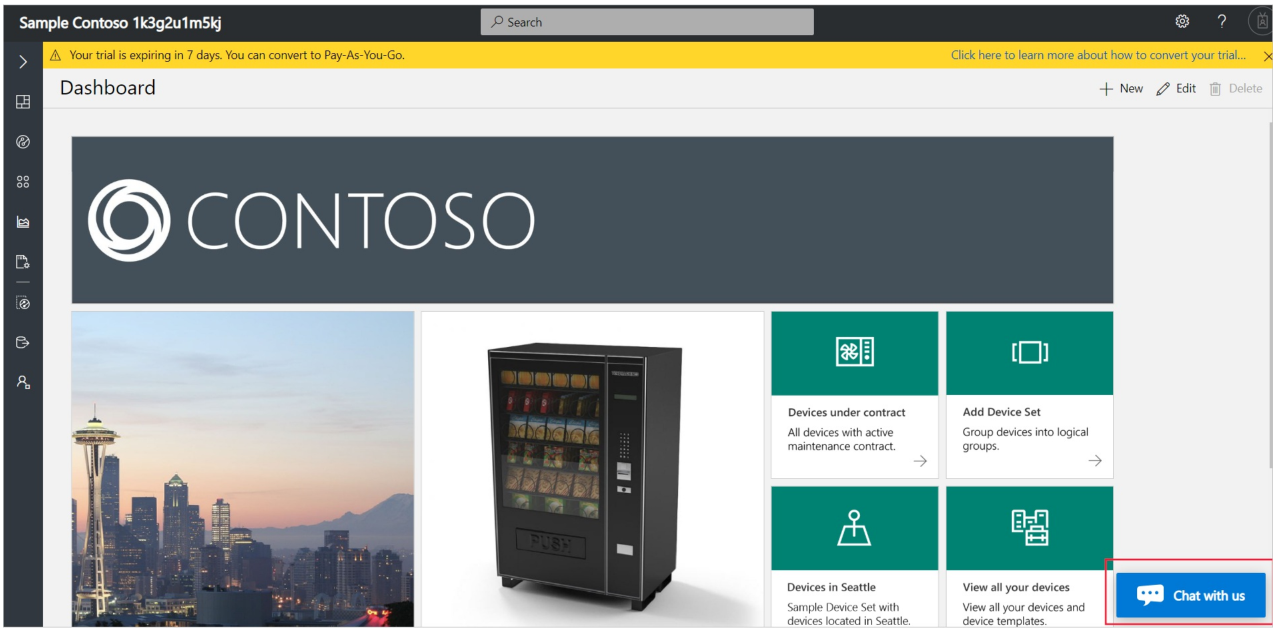
3/24/2020 • 2 minutes to read • Edit Online

This how-to article shows you how to toggle the live chat in your IoT Central application. You can use live chat to access technical support.

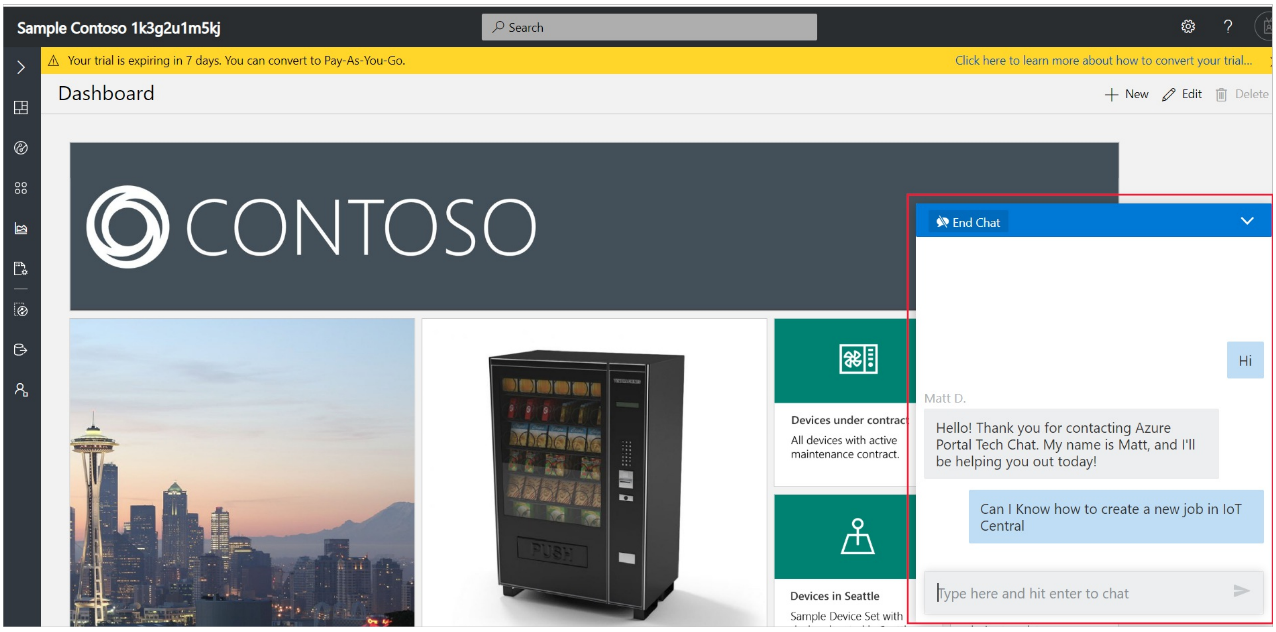
> **NOTE**
> The chat option is available only for applications created using the free pricing plan.

Chat with us

To get technical support, open your IoT Central application and select **Chat with us**.

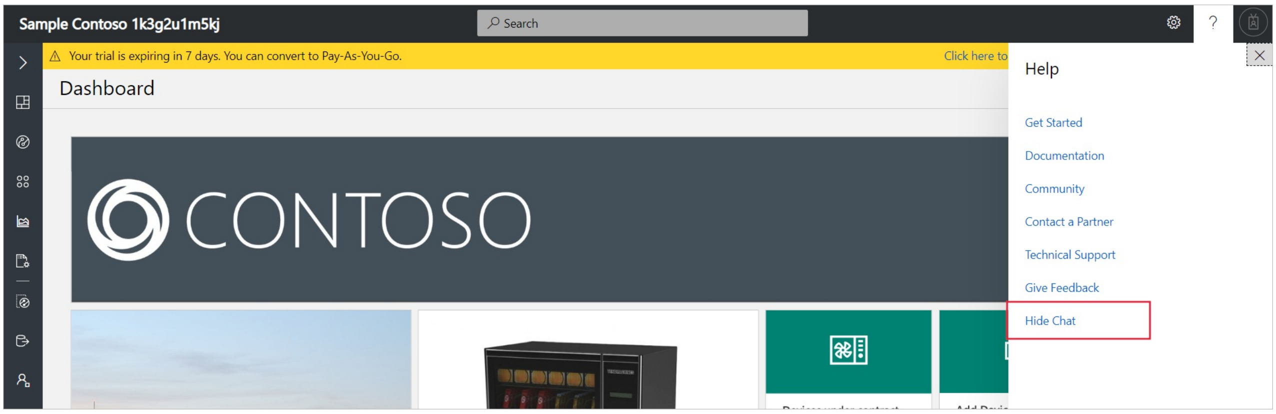


You can type a question as shown in the following screenshot:



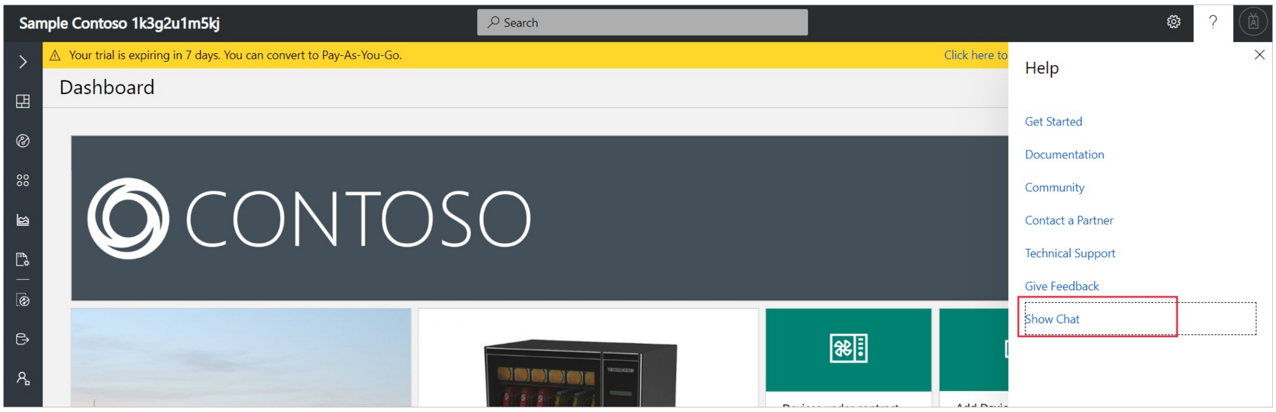
Hide chat

To hide the chat, choose **Hide Chat** in the **Help** panel:



Enable chat

To show the chat, choose **Show Chat** in the **Help** panel:



Next steps

Now that you've learned how to toggle live chat in Azure IoT Central, here is the suggested next step:

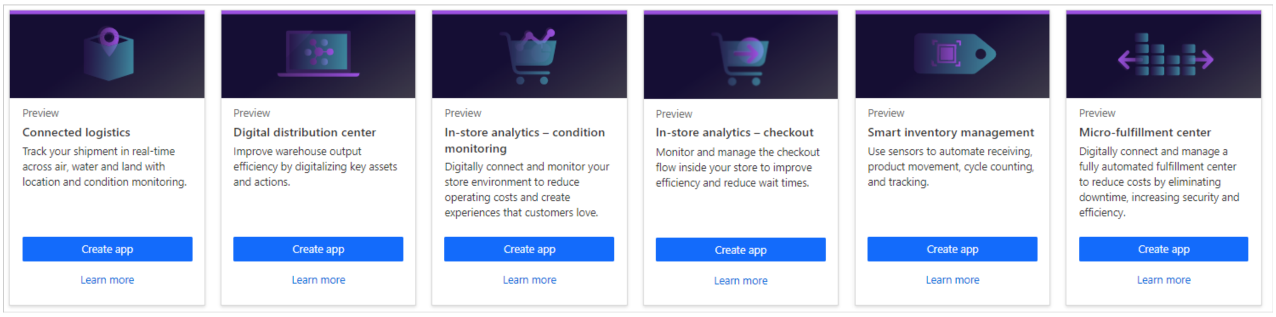
[Add tiles to your dashboard]

Building retail solutions with Azure IoT Central

4/9/2020 • 7 minutes to read • Edit Online

Azure IoT Central is an IoT app platform that reduces the burden and cost associated with developing, managing, and maintaining enterprise-grade IoT solutions. Choosing to build with Azure IoT Central gives you the opportunity to focus your time, money, and energy on transforming your business with IoT data, rather than just maintaining and updating a complex and continually evolving IoT infrastructure.

This article, describes several retail-specific IoT Central application templates. As a solution builder, you can use these templates to build IoT solutions that optimize supply chains, improve in-store experiences for customers, and track inventory more efficiently.



The following sections describe the capabilities of these application templates:

Connected logistics

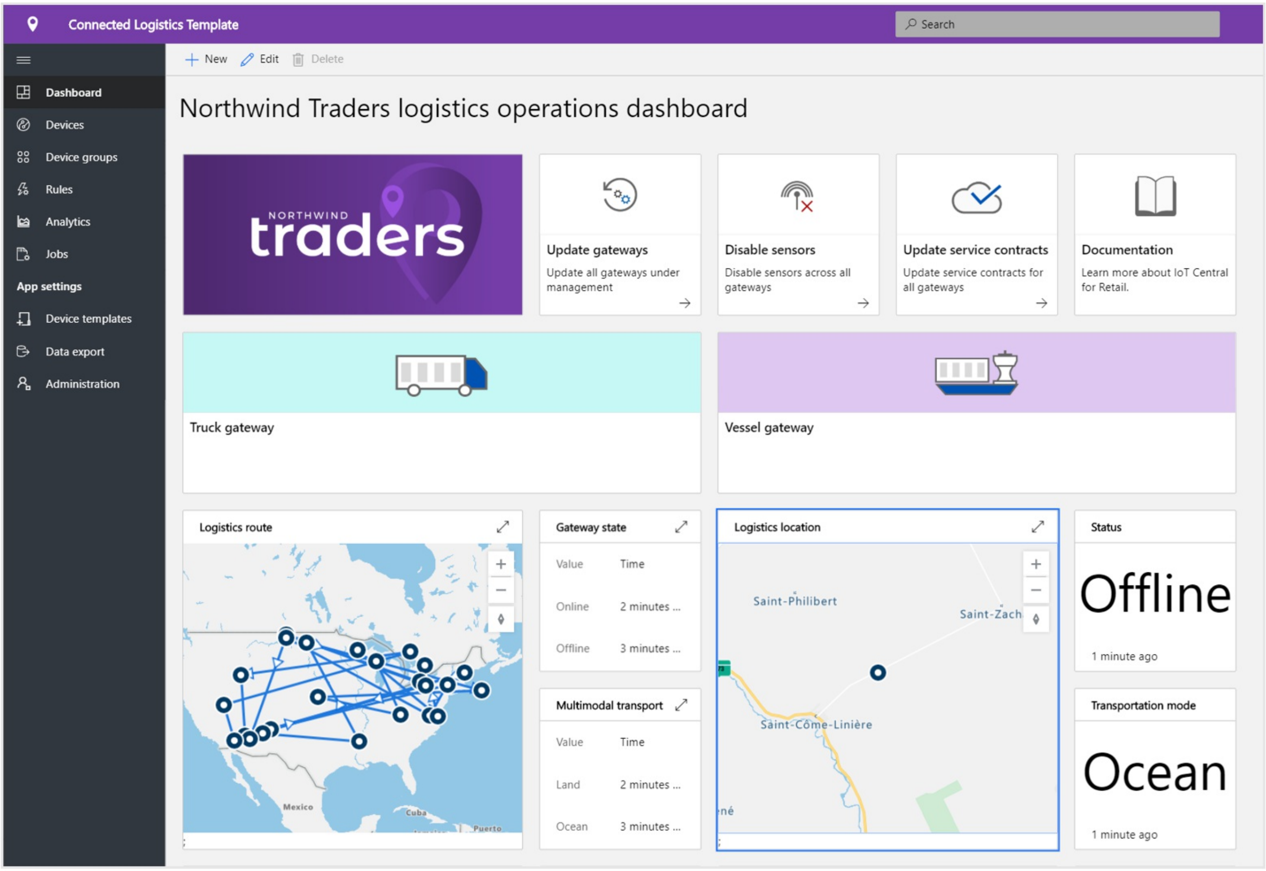
Global logistics spending is expected to reach $10.6 trillion in 2020. Transportation of goods accounts for the majority of this spending and shipping providers are under intense competitive pressure and constraints.

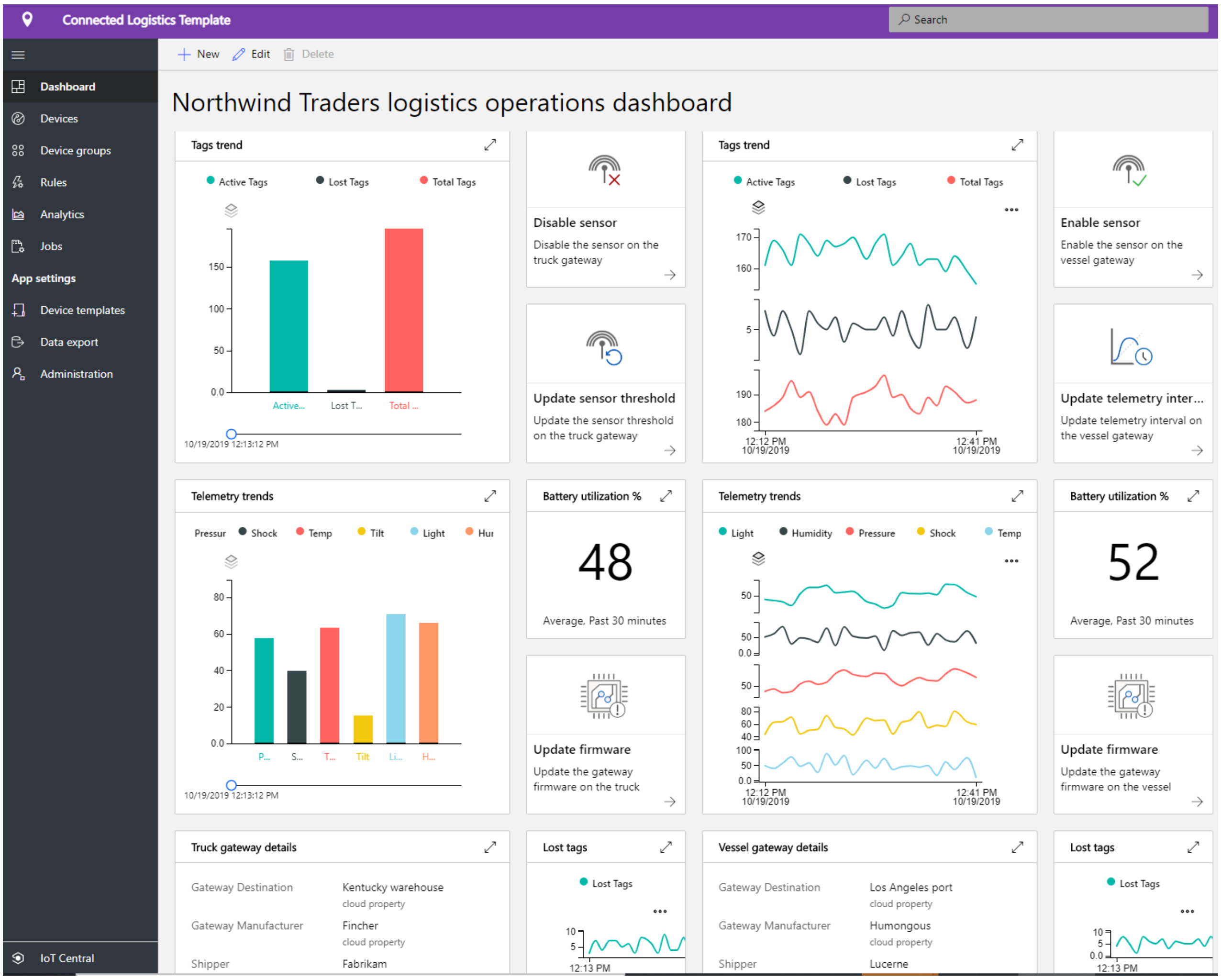
You can use IoT sensors to collect and monitor ambient conditions such as temperature, humidity, tilt, shock, light, and the location of a shipment. You can combine telemetry gathered from IoT sensors and devices with other data sources such as weather and traffic information in cloud-based business intelligence systems.

The benefits of a connected logistics solution include:

- Shipment monitoring with real-time tracing and tracking.
- Shipment integrity with real-time ambient condition monitoring.
- Security from theft, loss, or damage of shipments.
- Geo-fencing, route optimization, fleet management, and vehicle analytics.
- Forecasting for predictable departure and arrival of shipments.

The following screenshots show the out-of-the-box dashboard in the application template. The dashboard is fully customizable to meet your specific solution requirements:





To learn more, see the Deploy and walk through a connected logistics application template tutorial.

Digital distribution center

As manufacturers and retailers establish worldwide presences, their supply chains branch out and become more complex. Consumers now expect large selections of products to be available, and for those goods to arrive within one or two days of purchase. Distribution centers must adapt to these trends while overcoming existing inefficiencies.

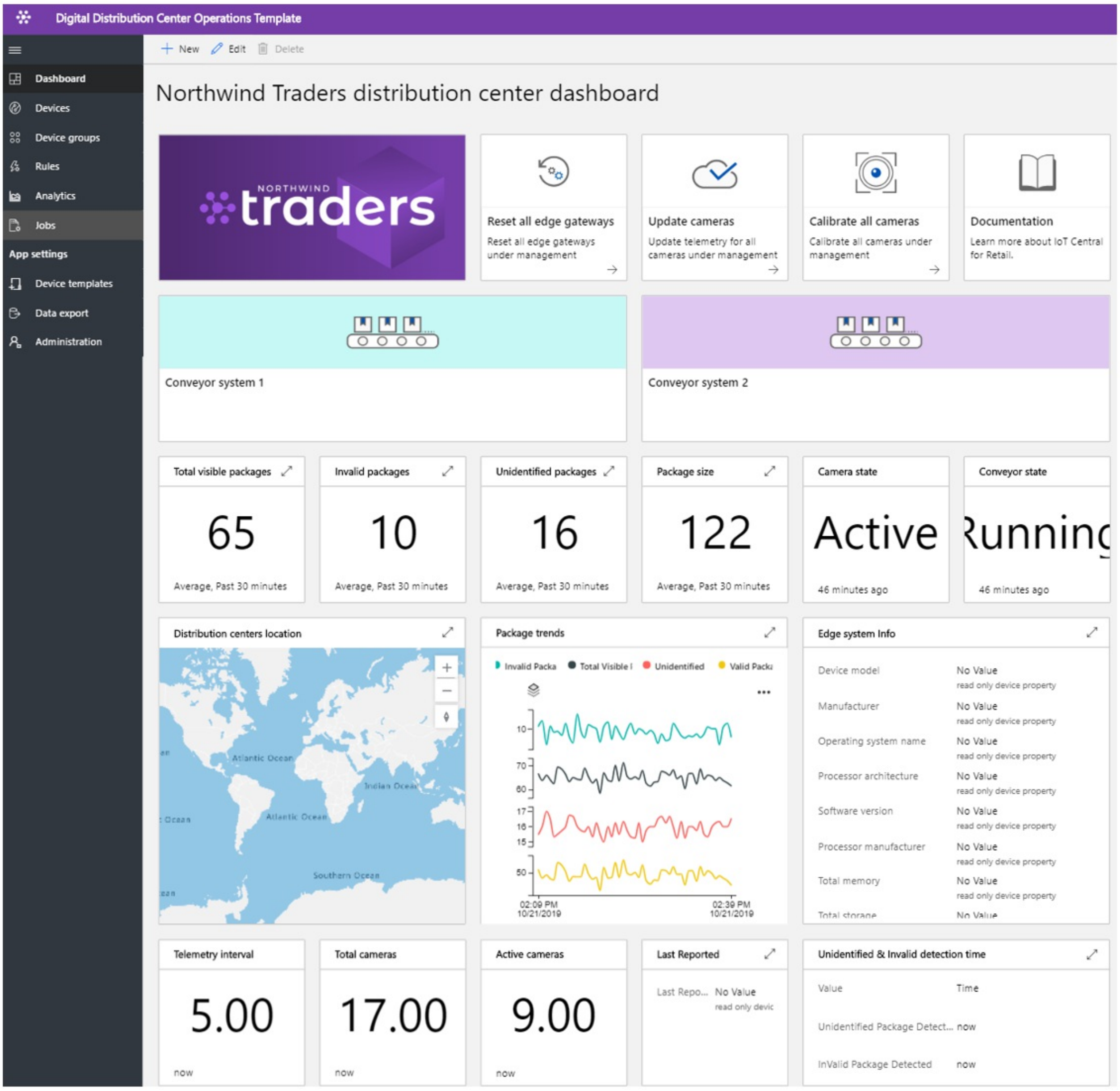
Today, a reliance on manual labor means that picking and packing accounts for 55-65% of distribution center costs. Manual picking and packing are also typically slower than automated systems, and rapidly fluctuating staffing needs make it even harder to meet shipping volumes. This seasonal fluctuation results in high staff turnover and increase the likelihood of costly errors.

Solutions based on IoT enabled cameras can deliver transformational benefits by enabling a digital feedback loop. Data from across the distribution center leads to actionable insights that, in turn, results in better data.

The benefits of a digital distribution center include:

- Cameras monitor goods as they arrive and move through the conveyor system.
- Automatic identification of faulty goods.
- Efficient order tracking.
- Reduced costs, improved productivity, and optimized usage.

The following screenshot shows the out-of-the-box dashboard in the application template. The dashboard is fully customizable to meet your specific solution requirements:



To learn more, see the Deploy and walk through a digital distribution center application template tutorial.

In-store analytics - condition monitoring

For many retailers, environmental conditions within their stores are a key differentiator from their competitors. Retailers want to maintain pleasant conditions within their stores for the benefit of their customers.

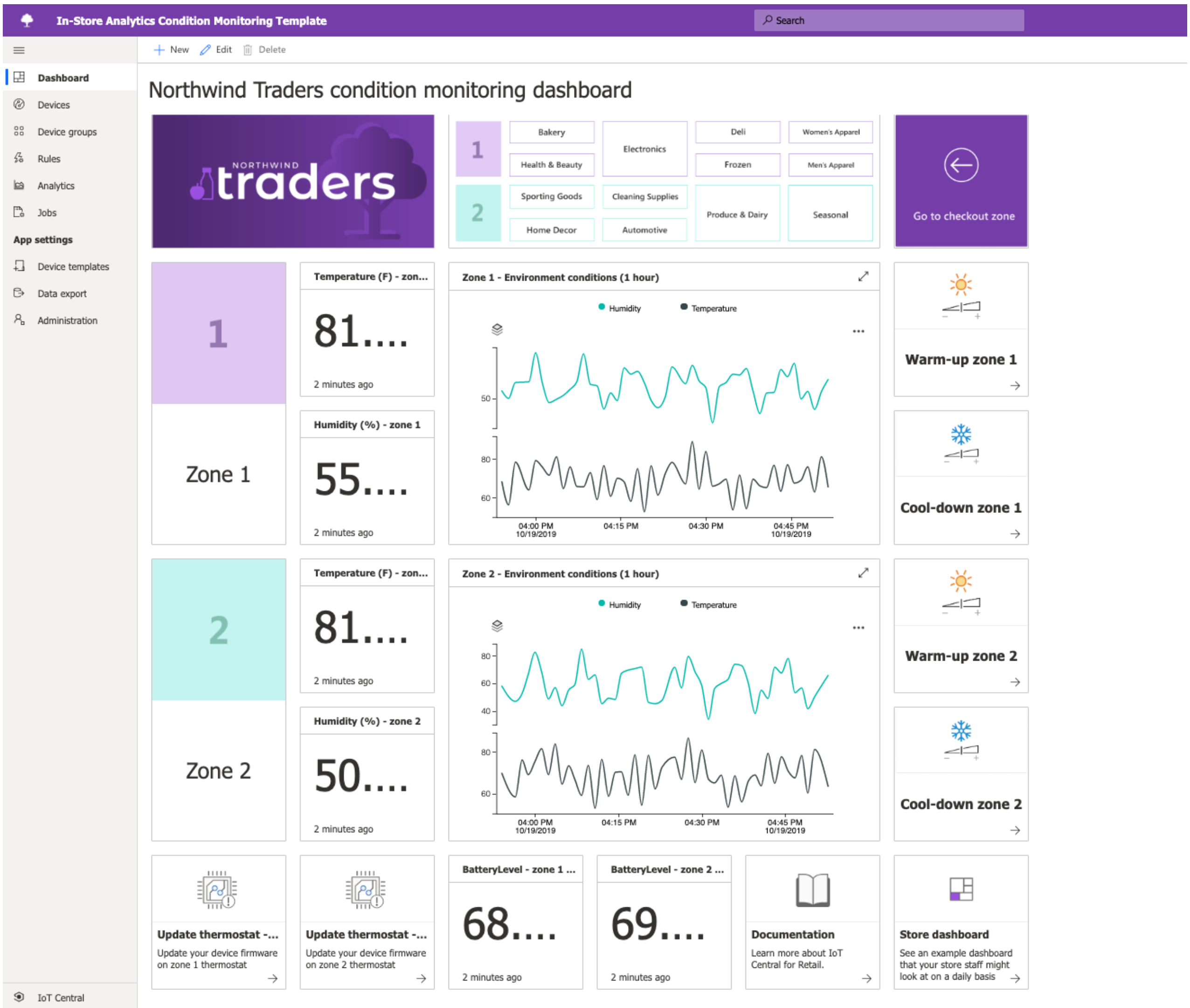
As a solution builder, you can use the IoT Central in-store analytics condition monitoring application template to build an end-to-end solution. The application template lets you digitally connect to and monitor a retail store environment using of different kinds of sensor devices. These sensor devices generate telemetry that you can convert into business insights helping the retailer to reduce operating costs and create a great experience for their customers.

Use the application template to:

- Connect a variety of IoT sensors to an IoT Central application instance.
- Monitor and manage the health of the sensor network as well as any gateway devices in the environment.
- Create custom rules around the environmental conditions within a store to trigger alerts for store managers.
- Transform the environmental conditions within your store into insights that the retail store team can use to improve the customer experience.
- Export the aggregated insights into existing or new business applications to provide useful and timely information to retail staff.

The application template comes with a set of device templates and uses a set of simulated devices to populate the dashboard.

The following screenshot shows the out-of-the-box dashboard in the application template. The dashboard is fully customizable to meet your specific solution requirements:



To learn more, see the Create an in-store analytics application in Azure IoT Central tutorial.

In-store analytics - checkout

For some retailers, the checkout experience within their stores is a key differentiator from their competitors. Retailers want to deliver a smooth checkout experience within their stores to encourage customers to return.

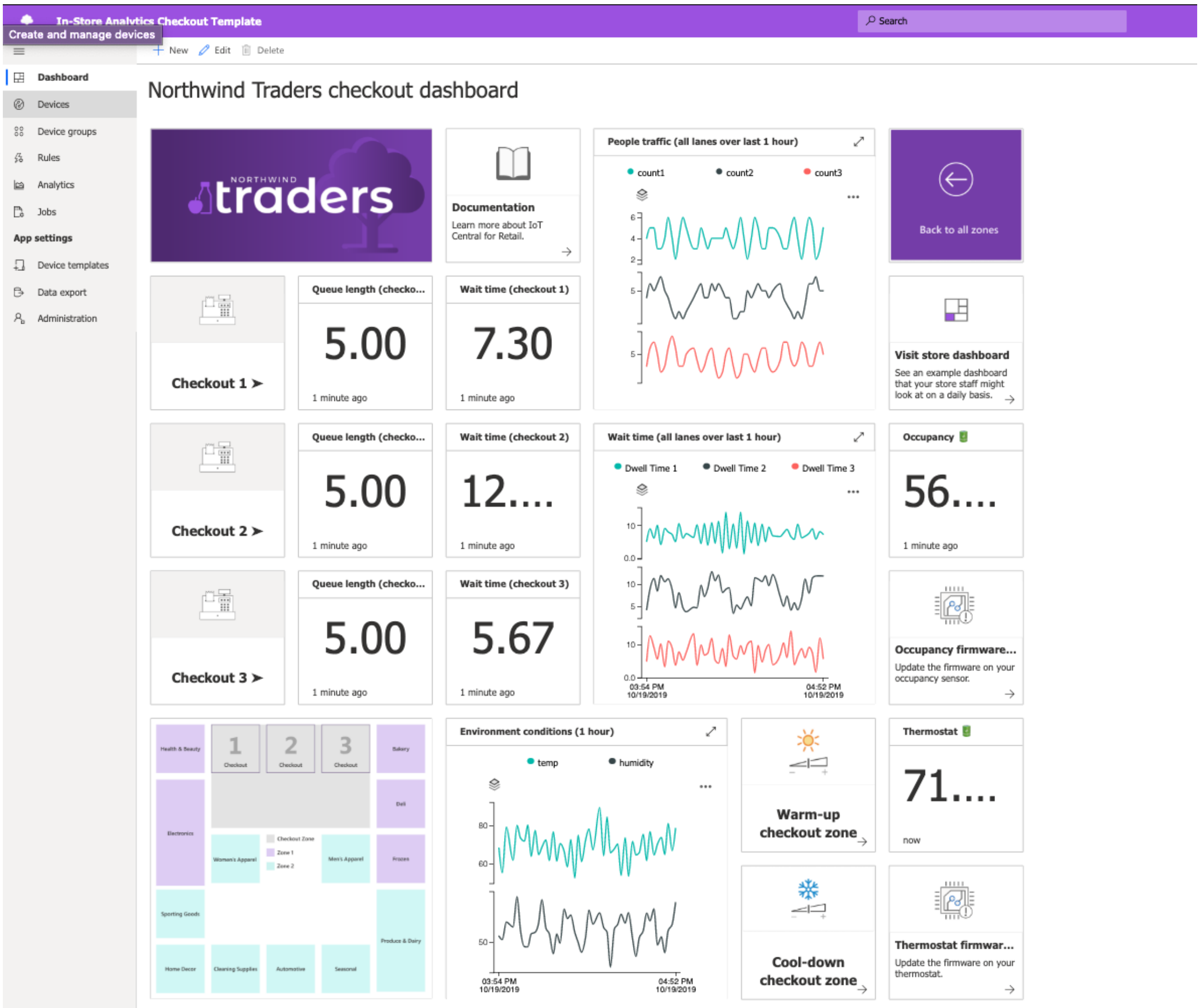
As a solution builder, you can use the IoT Central in-store analytics checkout application template to build a solution that delivers insights from around the checkout zone of a store to retail staff. For example, sensors can provide information about queue lengths and average wait times for each checkout lane.

Use the application template to:

- Connect a variety of IoT sensors to an IoT Central application instance.
- Monitor and manage the health of the sensor network as well as any gateway devices in the environment.
- Create custom rules around the checkout condition within a store to trigger alerts for retail staff.
- Transform the checkout conditions within the store into insights that the retail store team can use to improve the customer experience.
- Export the aggregated insights into existing or new business applications to provide useful and timely information to retail staff.

The application template comes with a set of device templates and uses a set of simulated devices to populate the dashboard with lane occupancy data.

The following screenshot shows the out-of-the-box dashboard in the application template. The dashboard is fully customizable to meet your specific solution requirements:



To learn more, see the Create an in-store analytics application in Azure IoT Central tutorial.

Smart inventory management

Inventory is the stock of goods a retailer holds. Inventory management is critical to ensure the right product is in the right place at the right time. A retailer must balance the costs of storing too much inventory against the costs of not having sufficient items in stock to meet demand.

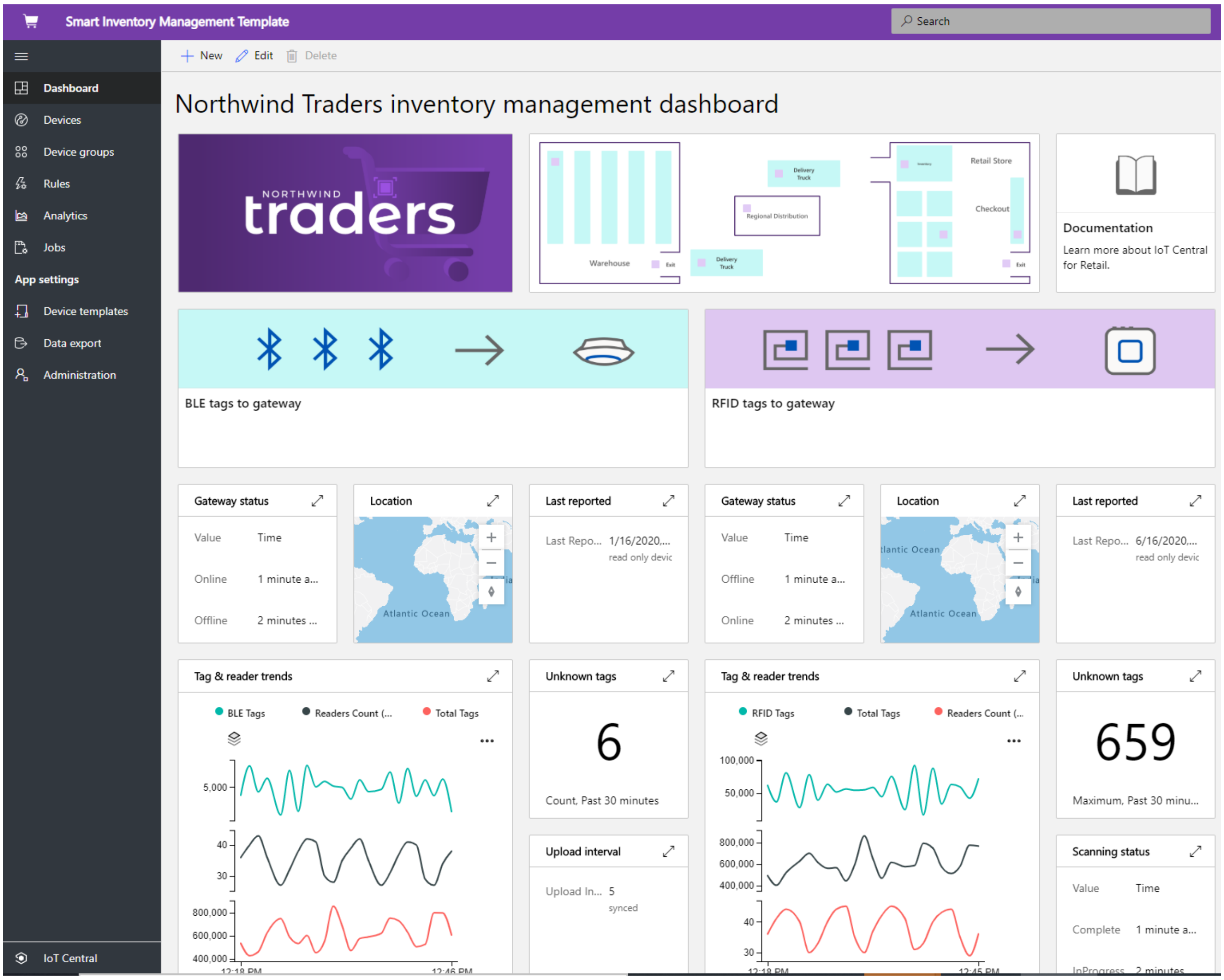
IoT data generated from radio-frequency identification (RFID) tags, beacons, and cameras provide opportunities to improve inventory management processes. You can combine telemetry gathered from IoT sensors and devices with other data sources such as weather and traffic information in cloud-based business intelligence systems.

The benefits of smart inventory management include:

- Reducing the risk of items being out of stock and ensuring the desired customer service level.
- In-depth analysis and insights into inventory accuracy in near real time.
- Tools to help decide on the right amount of inventory to hold to meet customer orders.

This application template focuses on device connectivity, and the configuration and management of RFID and Bluetooth low energy (BLE) reader devices.

The following screenshot shows the out-of-the-box dashboard in the application template. The dashboard is fully customizable to meet your specific solution requirements:



To learn more, see the Deploy and walk through a smart inventory management application template tutorial.

Micro-fulfillment center

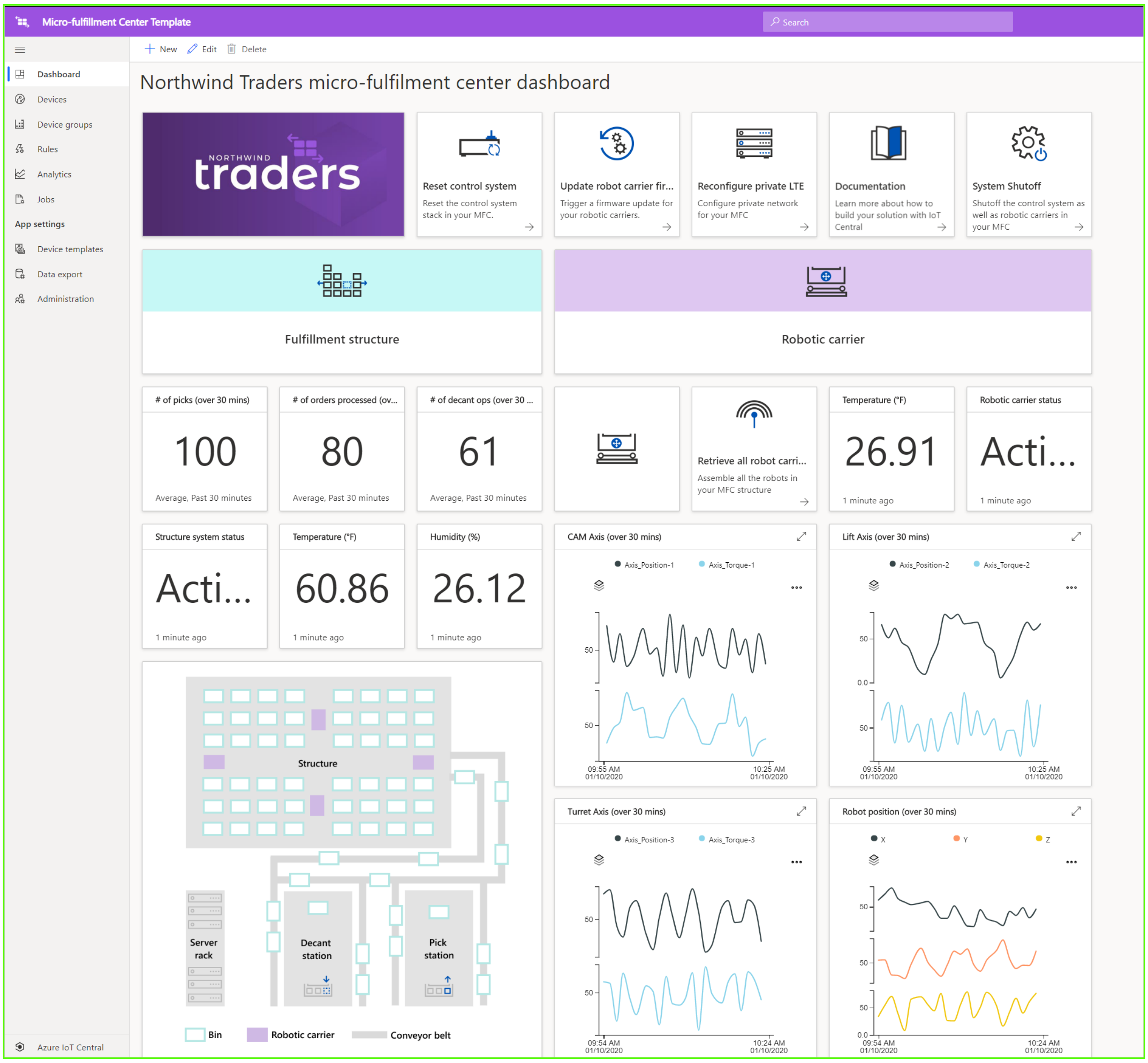
In the increasingly competitive retail landscape, retailers constantly face pressure to close the gap between demand and fulfillment. A new trend that has emerged to address the growing consumer demand is to house inventory near the end customers and the stores they visit.

The IoT Central micro-fulfillment center application template enables solution builders to monitor and manage all aspects of their fully automated fulfillment centers. The template includes a set of simulated condition monitoring sensors and robotic carriers to accelerate the solution development process. These sensor devices capture meaningful signals that can be converted into business insights allowing retailers to reduce their operating costs and create experiences for their customers.

The application template enables you to:

- Seamlessly connect different kinds of IoT sensors such as robots or condition monitoring sensors to an IoT Central application instance.
- Monitor and manage the health of the sensor network, and any gateway devices in the environment.
- Create custom rules around the environmental conditions within a fulfillment center to trigger appropriate alerts.
- Transform the environmental conditions within your fulfillment center into insights that can be leveraged by the retail warehouse team.
- Export the aggregated insights into existing or new business applications for the benefit of the retail staff members.

The following screenshot shows the out-of-the-box dashboard in the application template. The dashboard is fully customizable to meet your specific solution requirements:



To learn more, see the Deploy and walk through the micro-fulfillment center application template tutorial.

Next steps

To get started building a retail solution:

- Get started with the Create an in-store analytics application in Azure IoT Central tutorial that walks you through how to build a solution with one of the in-store analytics application templates.
- Deploy and walk through a connected logistics application template.
- Deploy and walk through a digital distribution center application template.
- Deploy and walk through a smart inventory management application template.
- Deploy and walk through the micro-fulfillment center application template.
- Learn more about IoT Central in the IoT Central overview.

Build energy solutions with IoT Central

2/4/2020 • 2 minutes to read • Edit Online

Smart meters and solar panels are playing an important role in the energy industry transformation. The smart meters give more controls and real-time insights about energy consumptions and solar panels growth is driving breakthrough in renewable energy generation. The smart meter and solar panel monitoring apps are sample templates to show the various capabilities. Partners can leverage these templates to build energy solutions with IoT Central for their specific needs. No new coding and no additional cost are required to deploy and use these applications. Learn more about energy application templates and their capabilities.

What is the smart meter monitoring application?

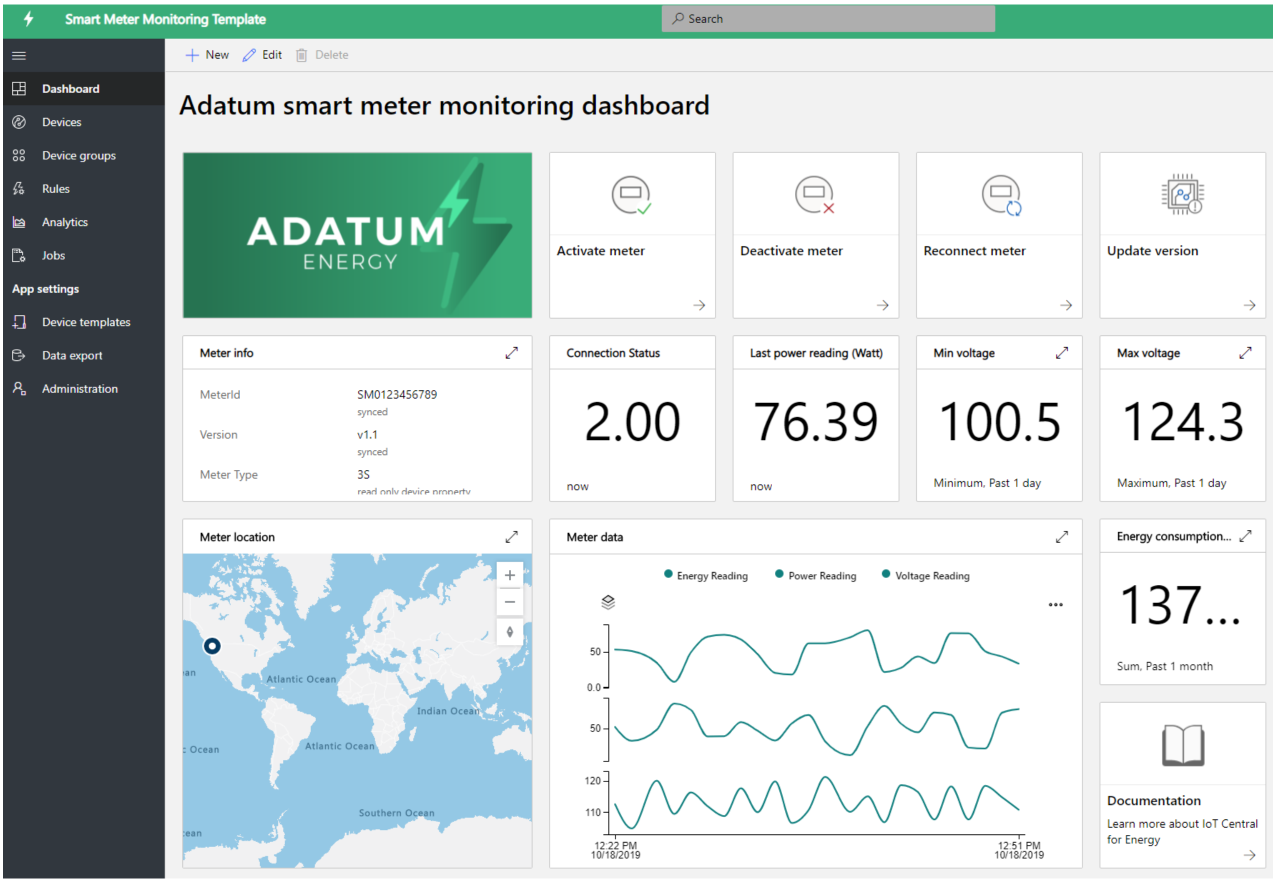
The smart meters not only enable automated billing, but also advanced metering use cases such as real-time readings and bi-directional communication. The smart meter app template enables utilities and partners to monitor smart meters status and data, define alarms and notifications. It provides sample commands, such as disconnect meter and update software. The meter data can be set up to egress to other business applications and to develop custom solutions.

App's key functionalities:

- Meter sample device model
- Meter info and live status
- Meter readings such as energy, power, and voltages
- Meter command samples
- Built-in visualization and dashboards
- Extensibility for custom solution development

You can try the smart meter monitoring app for free without an Azure subscription, and any commitments.

After you deploy the app, you'll see the simulated meter data on the dashboard, as shown in the figure below. This template is a sample app that you can easily extend and customize for your specific use cases.



What is the solar panel monitoring application?

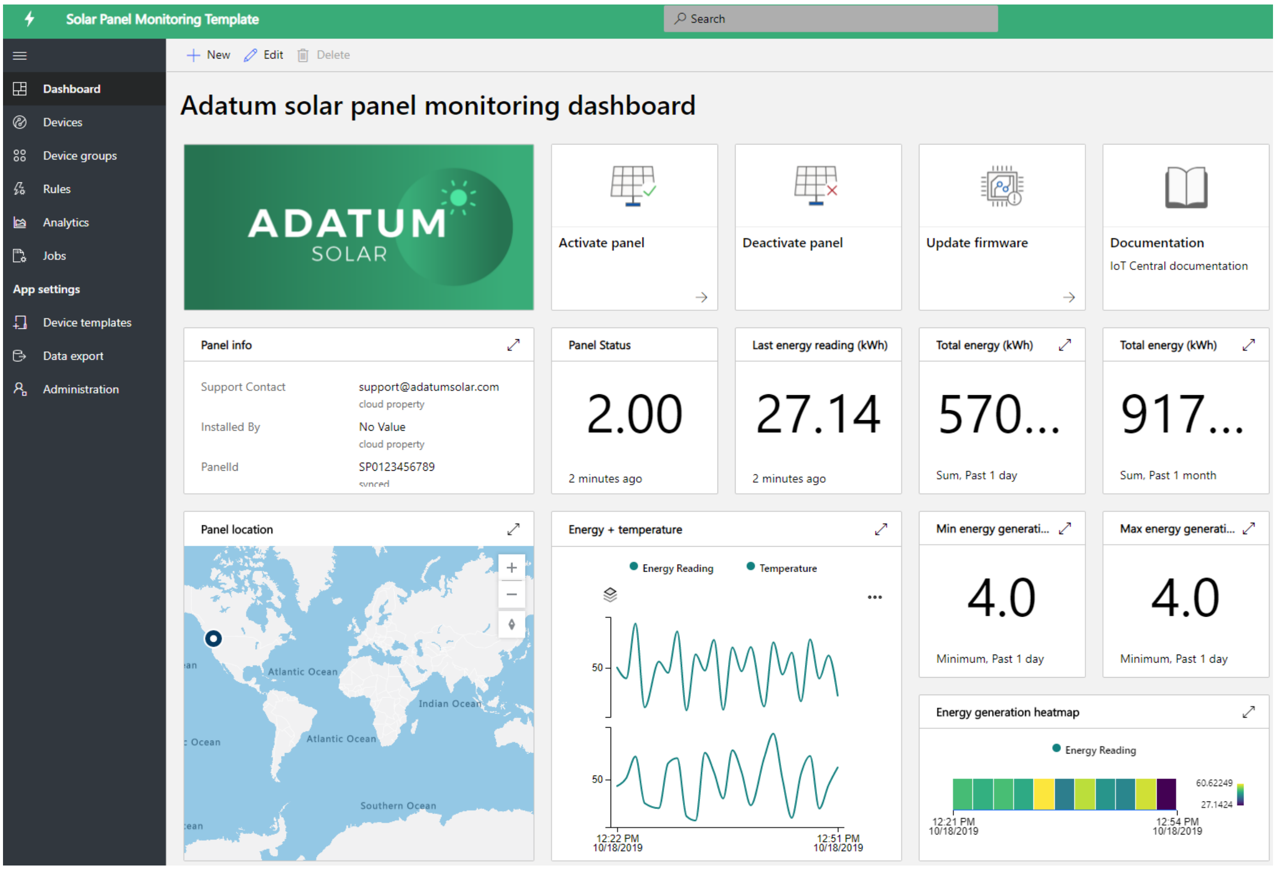
The solar panel monitoring app enables utilities and partners to monitor solar panels, such as their energy generation and connection status in near real time. It can send notifications based on defined threshold criteria. It provides sample commands, such as update firmware and other properties. The solar panel data can be set up to egress to other business applications and to develop custom solutions.

App's key functionalities:

- Solar panel sample device model
- Solar Panel info and live status
- Solar energy generation and other readings
- Command and control samples
- Built-in visualization and dashboards
- Extensibility for custom solution development

You can try the solar panel monitoring app for free without an Azure subscription and any commitments.

After you deploy the app, you'll see the simulated solar panel data within 1-2 minutes, as shown in the dashboard below. This template is a sample app that you can easily extend and customize for your specific use cases.



Next steps

To get started building an energy solution:

- Create application templates for free: smart meter app, solar panel app
- Learn about smart meter monitoring app concepts
- Learn about solar panel monitoring app concepts
- Learn about IoT Central platform

Building government solutions with Azure IoT Central

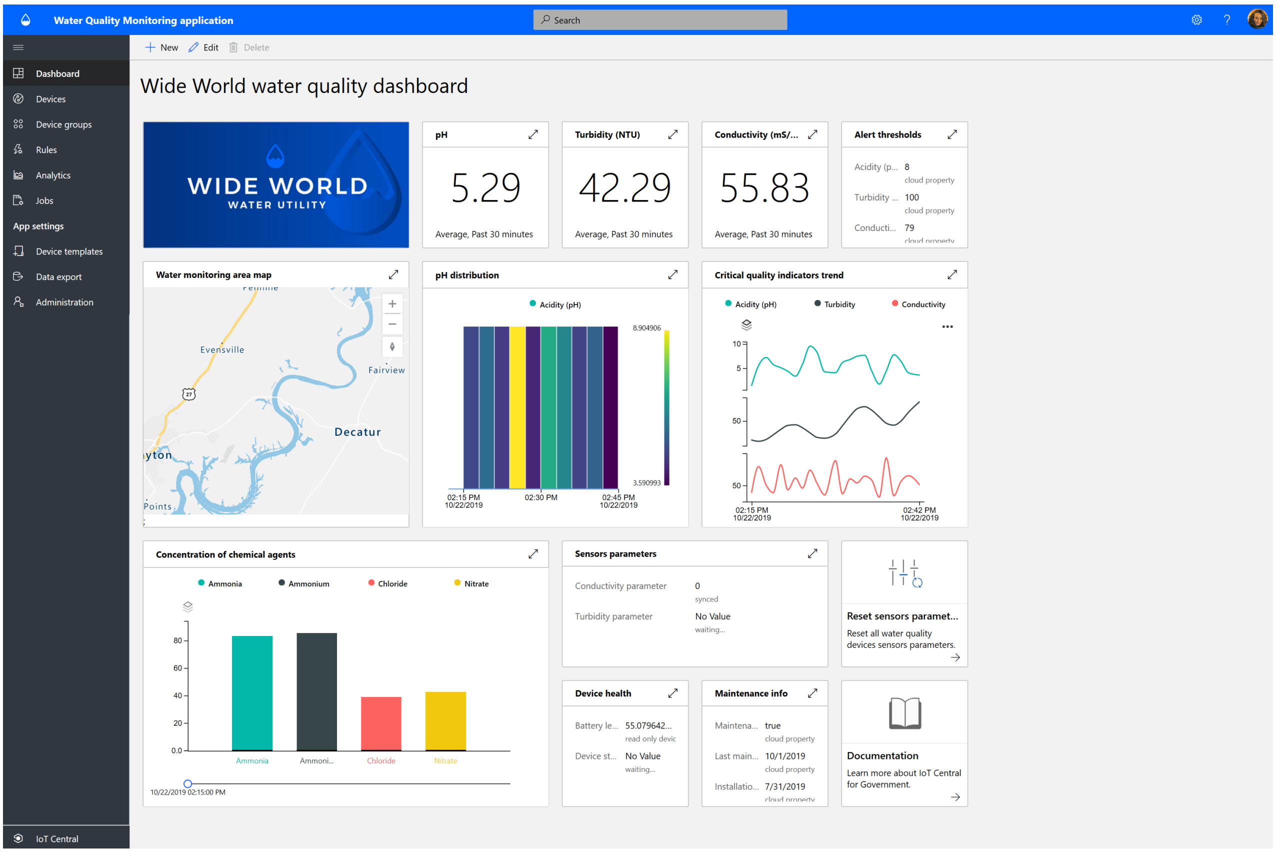
2/4/2020 • 2 minutes to read • Edit Online

Get started with building smart city solutions using Azure IoT Central application templates. Start now with **water quality monitoring**, **water consumption monitoring**, and **connected waste management**.

What is Water Quality Monitoring application template?

Traditional water quality monitoring relies on manual sampling techniques and field laboratory analysis, which is time consuming and costly. By remotely monitoring water quality in real-time, water quality issues can be managed before citizens are affected. Moreover, with advanced analytics, water utilities, and environmental agencies can act on early warnings on potential water quality issues and plan on water treatment in advance.

Water Quality Monitoring app is an IoT Central app template to help you kickstart your IoT solution development and enable water utilities to digitally monitor water quality in smart cities.



The App template consists of:

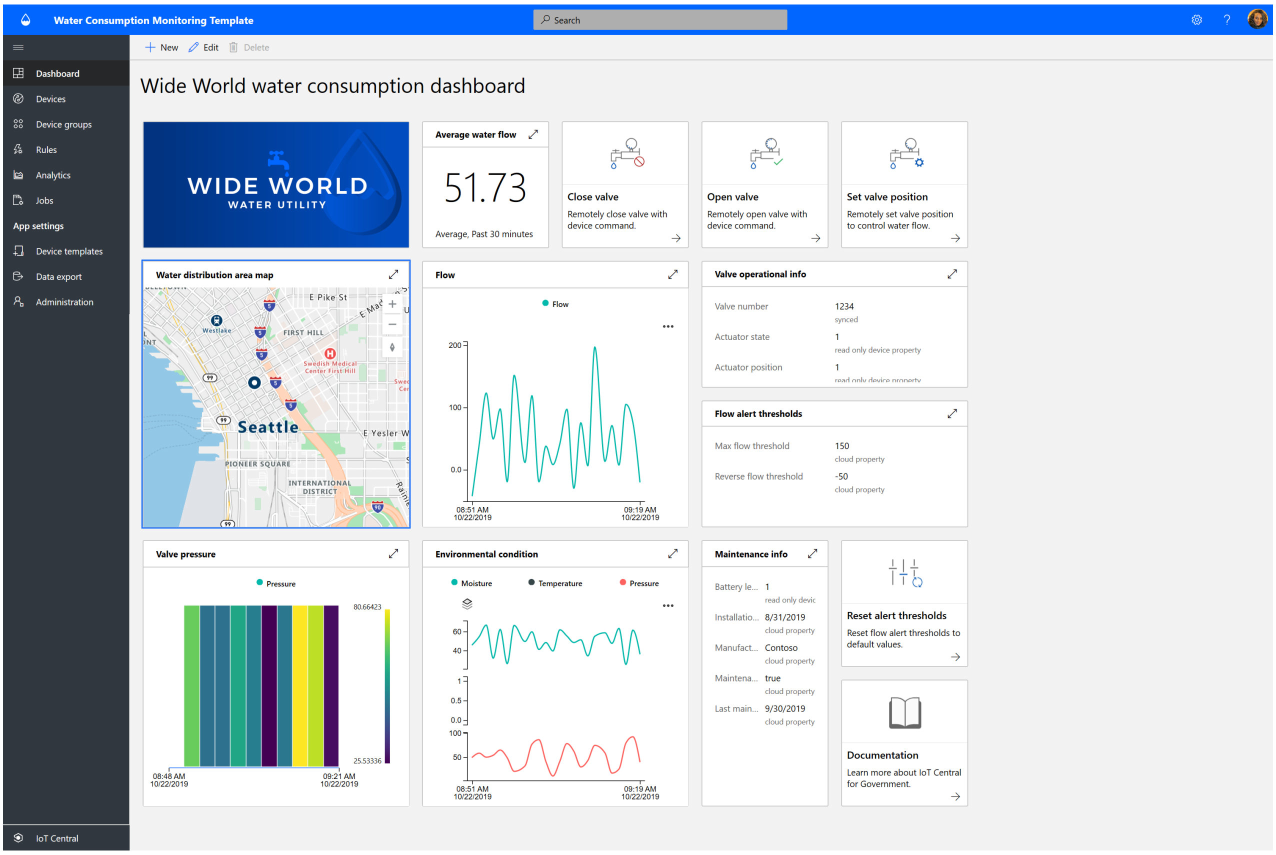
- Sample operator dashboards
- Sample water quality monitor device templates
- Simulated water quality monitor devices
- Pre-configured rules and jobs
- Branding using white labeling

Get started with the Water Quality Monitoring application tutorial.

What is Water Consumption Monitoring application template?

Traditional water consumption tracking relies on water operators manually reading water consumption meters at the meter sites. More and more cities are replacing traditional meters with advanced smart meters enabling remote monitoring of consumption and remotely controlling valves to control water flow. Water consumption monitoring coupled with digital feedback message to the citizen can increase awareness and reduce water consumption.

Water Consumption Monitoring app is an IoT Central app template to help you kickstart your IoT solution development to enable water utilities and cities to remotely monitor and control water flow to reduce consumption.



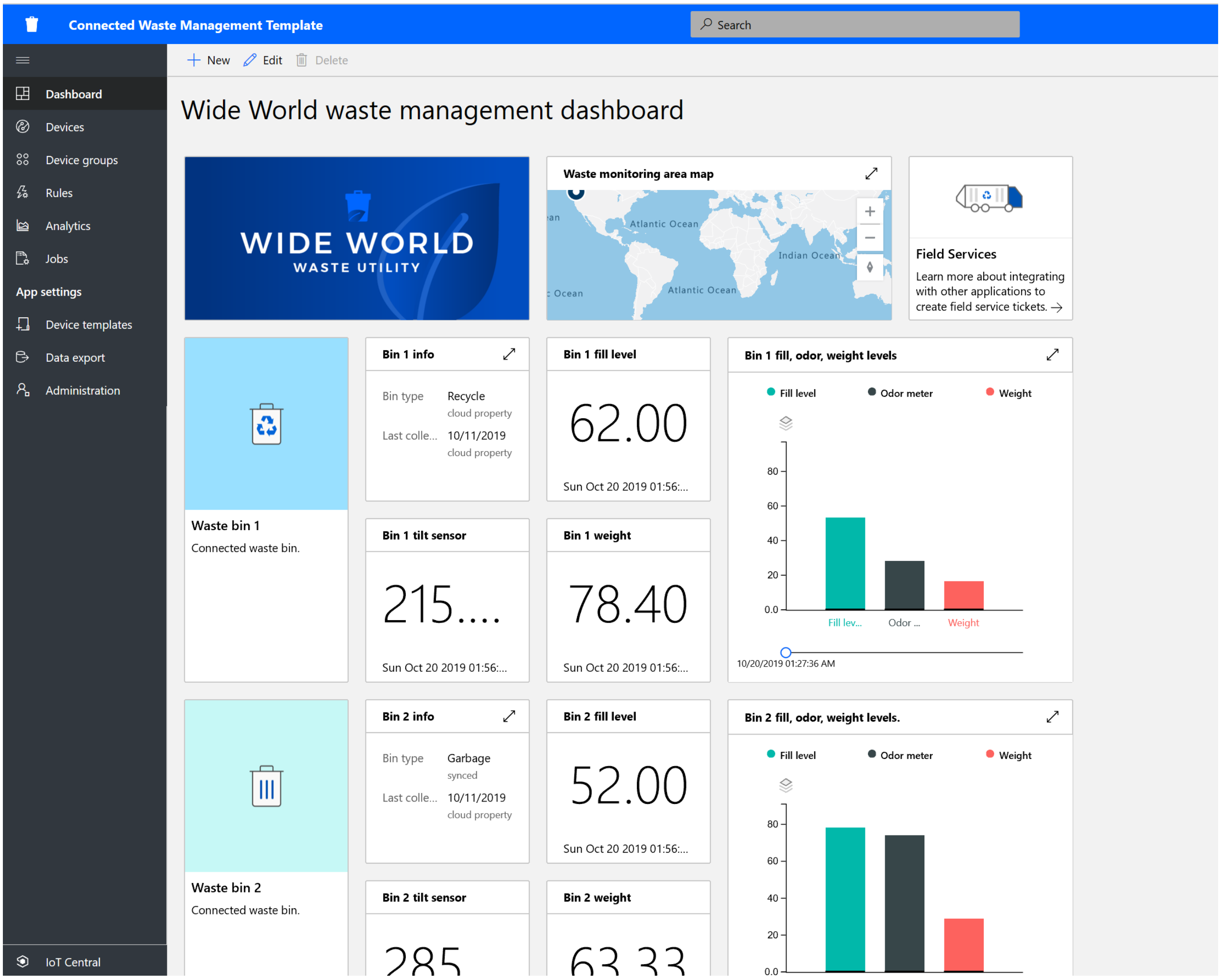
The Water Consumption Monitoring app template consists of pre-configured:

- Sample operator dashboards
- Sample water quality monitor device templates
- Simulated water quality monitor devices
- Pre-configured rules and jobs
- Branding using white labeling

Get started with the Water Consumption Monitoring application tutorial.

What is Connected Waste Management application template?

Connected Waste Management app is an IoT Central app template to help you kickstart your IoT solution development to enable smart cities to remotely monitor to maximize efficient waste collection.



The Connected Waste Management app template consist of pre-configured:

- Sample operator dashboards
- Sample connected waste bin device templates
- Simulated connected waste bin devices
- Pre-configured rules and jobs
- Branding using white labeling

Get started with the Connected Waste Management application tutorial.

Next steps

- Try any of the Government application templates in IoT Central for free create app
- Learn about Water Quality Monitoring concepts
- Learn about Water Consumption Monitoring concepts
- Learn about Connected Waste Management concepts
- Learn about IoT Central, see IoT Central overview

Building healthcare solutions with Azure IoT Central

2/4/2020 • 2 minutes to read • Edit Online

Learn to build healthcare solutions with Azure IoT Central using application templates.

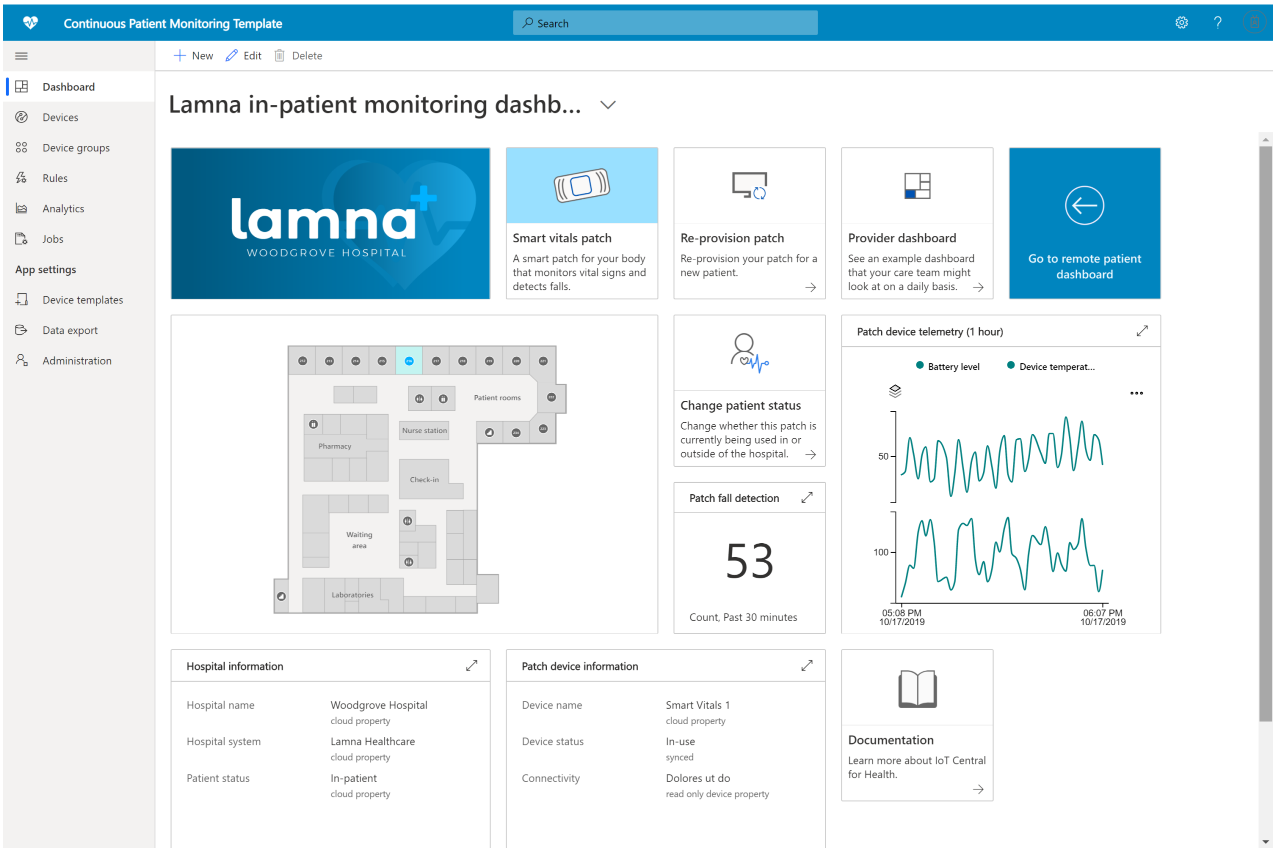
What is continuous patient monitoring template?

In the healthcare IoT space, Continuous Patient Monitoring is one of the key enablers of reducing the risk of readmissions, managing chronic diseases more effectively, and improving patient outcomes. Continuous Patient Monitoring can be split into two major categories:

1. **In-patient monitoring**: Using medical wearables and other devices in the hospital, care teams can monitor patient vital signs and medical conditions without having to send a nurse to check up on a patient multiple times a day. Care teams can understand the moment that a patient needs critical attention through notifications and prioritizes their time effectively.
2. **Remote patient monitoring**: By using medical wearables and patient reported outcomes (PROs) to monitor patients outside of the hospital, the risk of readmission can be lowered. Data from chronic disease patients and rehabilitation patients can be collected to ensure that patients are adhering to care plans and that alerts of patient deterioration can be surfaced to care teams before they become critical.

This application template can be used to build solutions for both categories of Continuous Patient Monitoring. The benefits include:

- Seamlessly connect different kinds of medical wearables to an IoT Central instance.
- Monitor and manage the devices to ensure they remain healthy.
- Create custom rules around device data to trigger appropriate alerts.
- Export your patient health data to the Azure API for FHIR, a compliant data store.
- Export the aggregated insights into existing or new business applications.



Next steps

To get started building a Continuous Patient monitoring solution:

- Deploy the application template
- See an example architecture

Azure IoT Central customer data request features

7/21/2020 • 2 minutes to read • Edit Online

Azure IoT Central is a fully managed Internet of Things (IoT) software-as-a-service solution that makes it easy to connect, monitor, and manage your IoT assets at scale, create deep insights from your IoT data, and take informed action.

> **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.

Identifying customer data

Azure Active Directory Object-IDs are used to identify users and assign roles. The Azure IoT Central portal displays user email addresses for role assignments but only the Azure Active Directory Object-ID is stored, the email address is dynamically queried from Azure Active Directory. Azure IoT Central administrators can view, export, and delete application users in the user administration section of an Azure IoT Central application.

Within the application, email addresses can be configured to receive alerts. In this case, email addresses are stored within IoT Central and must be managed from the in-app account administration page.

Regarding devices, Microsoft maintains no information and has no access to data that enables device to user correlation. Many of the devices managed in Azure IoT Central are not personal devices, for example a vending machine or coffee maker. Customers may, however, consider some devices to be personally identifiable and at their discretion may maintain their own asset or inventory tracking systems that tie devices to individuals. Azure IoT Central manages and stores all data associated with devices as if it were personal data.

When you use Microsoft enterprise services, Microsoft generates some information, known as system-generated logs. These logs constitute factual actions conducted within the service and diagnostic data related to individual devices, and are not related to user activity. Azure IoT Central system-generated logs are not accessible or exportable by application administrators.

Deleting customer data

The ability to delete user data is only provided through the IoT Central administration page. Application administrators can select the user to be deleted and select **Delete** in the upper right corner of the application to delete the record. Application administrators can also remove individual accounts that are no longer associated with the application in question.

After a user is deleted, no further alerts are emailed to them. However, their email address must be individually removed from each configured alert.

Exporting customer data

The ability to export data is only provided through the IoT Central administration page. Customer data, including assigned roles, can be selected, copied, and pasted by an application administrator.

Links to additional documentation

For more information about account administration, including role definitions, see How to administer your application.

Supported browsers for Azure IoT Central

10/27/2019 • 2 minutes to read • Edit Online

This article applies to operators, builders, and administrators.

Azure IoT Central can be accessed across most modern desktops, tablets, and browsers. The following article outlines the list of supported browsers and required connectivity.

Supported browsers

We recommend that you use the most up-to-date browser that's compatible with your operating system. The following browsers are supported:

- Microsoft Edge (latest version)
- Safari (latest version, Mac only)
- Chrome (latest version)
- Firefox (latest version)

Required protocols

Azure IoT Central requires that your network supports both the HTTPS and WebSocket protocols for outbound connectivity.