IOT CENTRAL

1. **Introduction**
   1. Overview
2. **Benefits/Features:**

2.1)Azure IoT Central application Integration with Dynamics 365 Field Services.

2.2) Add an Azure IoT Edge device to your Azure IoT Central application.

2.3) Pricing.

2.4) Export IoT data to cloud destinations using data export.

2.5) Application Templates.

2.6) Device Templates.

2.6) Built-In Dashboard.

2.7) Telemetry Retention Period.

2.8) Rules.

2.9) Built-In Device Provisioning Service.

2.10) Objectives.

2.11) Adding Device Template for testing.

2.12) Jobs to manage device.

1. **How Tos:**

3.1) Add an Azure IoT Edge device to your Azure IoT Central application.

3.2) Export IoT data to cloud destinations using data export.

3.3) Create Alert Rules.

3.4) Create Application.

3.5) Run Job to Manage Device.

3.6) Create Dashboard

1. **Difference Between IoT Central & IoT Hub**
2. **OVERVIEW:**

IOT Central is one of the application of IOT platform. It functions as developing, managing, and maintaining enterprise-grade IoT solutions cost efficiently. It has a lot of ready to deploy templates of devices, gateways, edge devices ,etc. It let’s us deploy demo device using device template and let us to test it before deploying in actual environment or using real device.

Types of Application:

-Custom.

-Industrial Templates: -Retail, Energy, Government, Healthcare.

Types of Device Templates:

-IoT Device.

-IoT Edge Device.

-Gateway Device.

-Industry specific Devices.

1. **Features Of IoT Central:**
   1. **Azure IoT Central application Integration with Dynamics 365 Field Services**

The below integration processes can be easily implemented based on a pure configuration experience:

* Azure IoT Central can send information about device anomalies to be connected Field Service (as an IoT Alert) for diagnosis.
* Connected Field Service can create cases or work orders triggered from device anomalies.
* Connected Field Service can schedule technicians for inspection to prevent the downtime incidents.
* Azure IoT Central device dashboard can be updated with relevant service and scheduling information.
  1. **Add an Azure IoT Edge device to your Azure IoT Central application**
* 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

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 and downstream device.
  + Cloud properties that aren't stored on the IoT Edge device.
  + Customizations, dashboards, and forms that are part of your IoT Central application.
* 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.
  1. **Export IoT data to cloud destinations using data export:**
* Continuously export telemetry, property changes, device connectivity, device lifecycle, and device template lifecycle data in JSON format in near-real time.
* Filter the data streams to export data that matches custom conditions.
* Enrich the data streams with custom values and property values from the device.
* Send the data to destinations such as Azure Event Hubs, Azure Service Bus, Azure Blob Storage, and webhook endpoints.
  1. **What are application templates?**

These are pre-configured templates for IOT solution development.

Application templates consist of:

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

These are of 2 types:

**(I)Custom templates**

If you want to create your application from scratch, choose the **Custom application** template.

**(II)Industry focused templates**

Azure IoT Central is an industry agnostic application platform. Application templates are industry focused examples available for these industries: Retail, Energy, Government, Healthcare.

* 1. **Types of Device Templates:**

There are various Device Templates available to deploy and test:

* IoT Device.
* IoT Edge Device.
* Gateway Device.
* Industry specific Devices.
  1. **Built-In Dashboard.**

Built-In Dashboard supported and provides a customizable UI to monitor device health and telemetry.

* 1. **Telemetry Retention Period**

Telemetry Retention Period is for 30 days.

* 1. **Rule’s**

We can Create a rule and set up notifications in your Azure IoT Central application

* 1. **Built-In Device Provisioning Service.**
  2. **Offers to connect, manage & monitor devices at scale.**
  3. **We could add any device template to review it’s working before deploying on solution.**
  4. **Jobs**

In this we could work to execute some jobs on a connected device.

Types of jobs available are based on : Cloud property, Property and Command.

* 1. **Pricing:**

|  |  |  |  |
| --- | --- | --- | --- |
| Pricing tier | Standard Tier 0 | Standard Tier 1 | Standard Tier 2 |
| Use Case | For devices sending a few messages per day | For devices sending a few messages per hour | For devices sending a message every few minutes |
| Price per device per month | $0.08 per month | $0.40 per month | $0.70 per month |
| Monthly device message allocation\*1 | 400 messages | 5,000 messages | 30,000 messages |
| Included free quantities per application | 2 free devices (800 included messages) | 2 free devices (10,000 included messages) | 2 free devices (60,000 included messages) |
| Overage pricing per 1K messages\*2 | $0.07 per 1K messages | $0.015 per 1K messages | $0.015 per 1K messages |

\*1 Total message allocation is shared across all devices in an IoT Central application

\*2 The standard message size is 4 KB. For example, if the device sends a 4.5 KB message, it will be billed as 2 messages.

1. **How To’s:**
   1. **Add an Azure IoT Edge device to your Azure IoT Central application.**

**Steps:**

-Created Edge Runtime Template in IoT Central and published it.

-Registered Edge Device in IoT Central.

-Added custom telemetry fields.

-Deployed Vm and installed Edge Runtime on it.

-Registered Edge Device on it via symmetric key provisioning.

-Applied the changes in the configuration file.

-Restart iotedge

-Listed Modules.

-Modules deployed successfully.

-Device Status changes to Provisioned.

**Detailed Steps with Demo:**

* + 1. **Create device template**
       1. **Import manifest to create template**

To create a device template from an IoT Edge manifest:

* In your IoT Central application, navigate to **Device templates** and select **+ New**.
* On the **Select template type** page, select the **Azure IoT Edge** tile. Then select **Next: Customize**.
* 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**. Then select **Next: Review**.
* **EnvironmentalSensorManifest.json:**

{

"modulesContent": {

"$edgeAgent": {

"properties.desired": {

"schemaVersion": "1.0",

"runtime": {

"type": "docker",

"settings": {

"minDockerVersion": "v1.25",

"loggingOptions": "",

"registryCredentials": {}

}

},

"systemModules": {

"edgeAgent": {

"type": "docker",

"settings": {

"image": "mcr.microsoft.com/azureiotedge-agent:1.0",

"createOptions": "{}"

}

},

"edgeHub": {

"type": "docker",

"status": "running",

"restartPolicy": "always",

"settings": {

"image": "mcr.microsoft.com/azureiotedge-hub:1.0",

"createOptions": "{}"

}

}

},

"modules": {

"SimulatedTemperatureSensor": {

"version": "1.0",

"type": "docker",

"status": "running",

"restartPolicy": "always",

"settings": {

"image": "mcr.microsoft.com/azureiotedge-simulated-temperature-sensor:1.0",

"createOptions": "{}"

}

}

}

}

},

"$edgeHub": {

"properties.desired": {

"schemaVersion": "1.0",

"routes": {

"route": "FROM /\* INTO $upstream"

},

"storeAndForwardConfiguration": {

"timeToLiveSecs": 7200

}

}

},

"SimulatedTemperatureSensor": {

"properties.desired": {

"SendData": true,

"SendInterval": 10

}

}

}

}

* On the Review page, select Create.
* Select the Manage interface in the SimulatedTemperatureSensor module to view the two properties defined in the manifest:
  + 1. **Add telemetry to manifest**

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:

* Select the **Manage** interface in the **Environmental Sensor Edge Device** template.
* Select **+ Add capability**. Enter *machine* as the **Display name** and make sure that the **Capability type** is **Telemetry**.
* 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**.
* Select **+ Add capability**. Enter *ambient* as the **Display name** and make sure that the **Capability type** is **Telemetry**.
* 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**.
* Select **+ Add capability**. Enter *timeCreated* as the **Display name** and make sure that the **Capability type** is **Telemetry**.
* Select **DateTime** as the schema type.
* Select **Save** to update the template.
  + 1. **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:

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

* + 1. **Add IoT Edge device**

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

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

* + 1. **Get the device credentials**

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

* On the Device page, select the device you created.
* Select Connect.
* On the Device connection page, make a note of the ID Scope, the Device ID, and the Primary Key. You use these values later.
* Select Close.

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

* + 1. **Deploy an IoT Edge device**
* Create a ubuntu Vm.
* Enable IoT Edge on it.
  + 1. **Configure the IoT Edge VM**

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

* Run the following command to check the IoT Edge runtime version.

sudo iotedge –version

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

sudo nano /etc/iotedge/config.yaml

* 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"

* 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}"

* Replace {scope\_id} with the **ID Scope** you made a note of previously.
* Replace {registration\_id} with the **Device ID** you made a note of previously.
* Replace {symmetric\_key} with the **Primary key** you made a note of previously.
* Save the changes (**Ctrl-O**) and exit (**Ctrl-X**) the nano editor.
* Run the following command to restart the IoT Edge daemon:

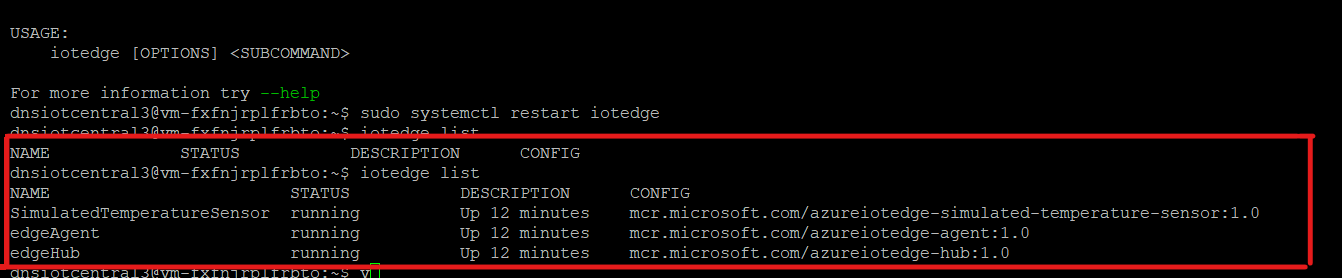
sudo systemctl restart iotedge

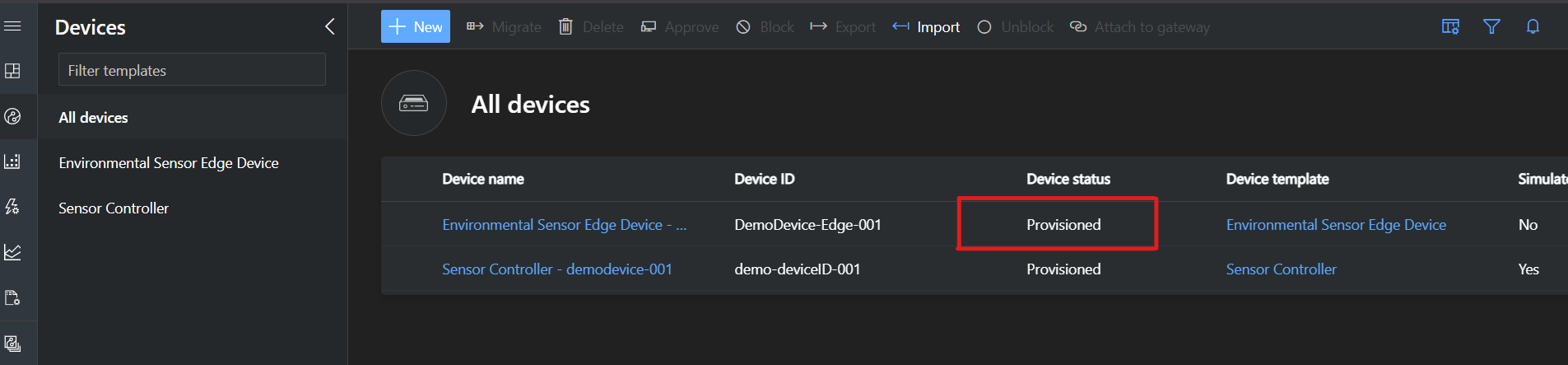
* To check the status of the IoT Edge modules, run the following command

iotedge list

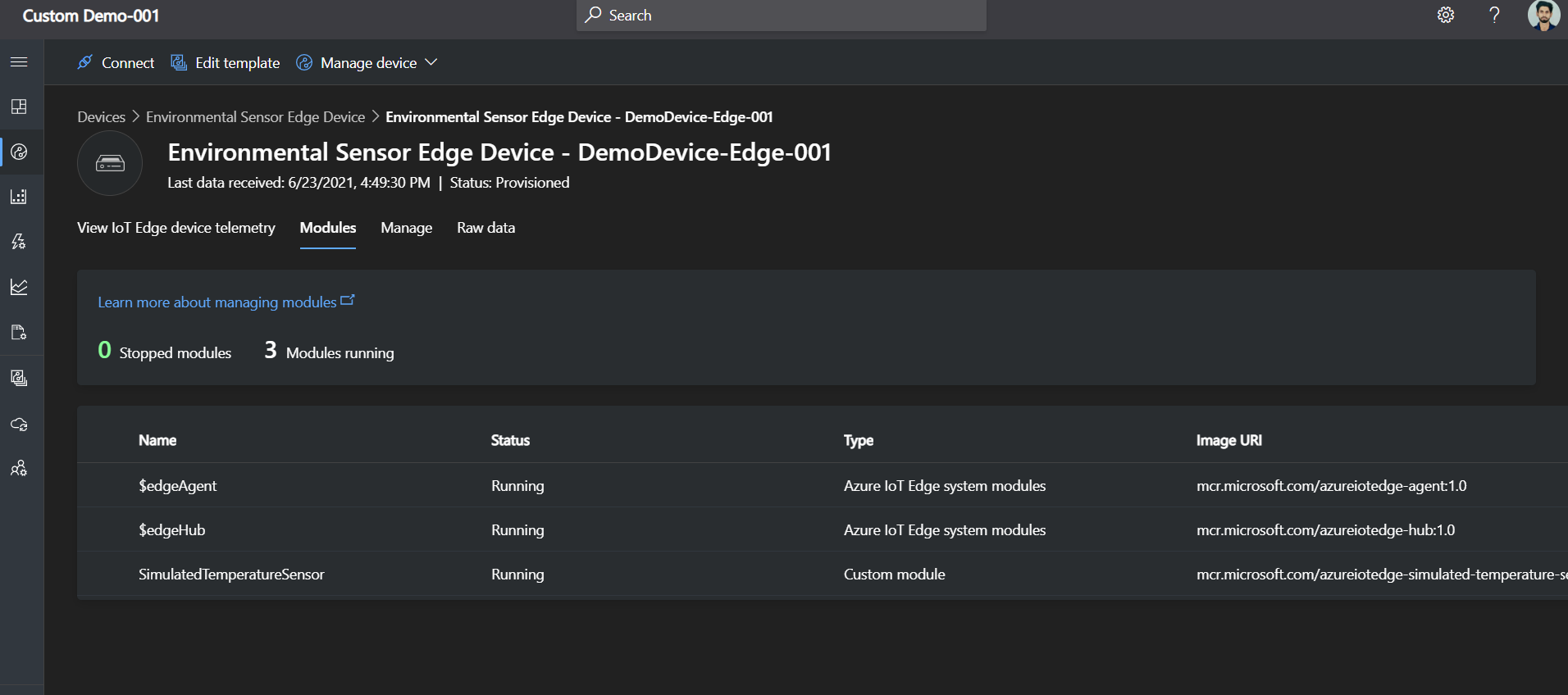
* + 1. **View the telemetry**
* The simulated IoT Edge device is now running in the VM. In the IoT Central application, the device status is now **Provisioned** on the **Devices** page.
* You can see the telemetry from the device on the **View IoT Edge device telemetry** page.
* The **Modules** page shows the status of the IoT Edge modules on the device.

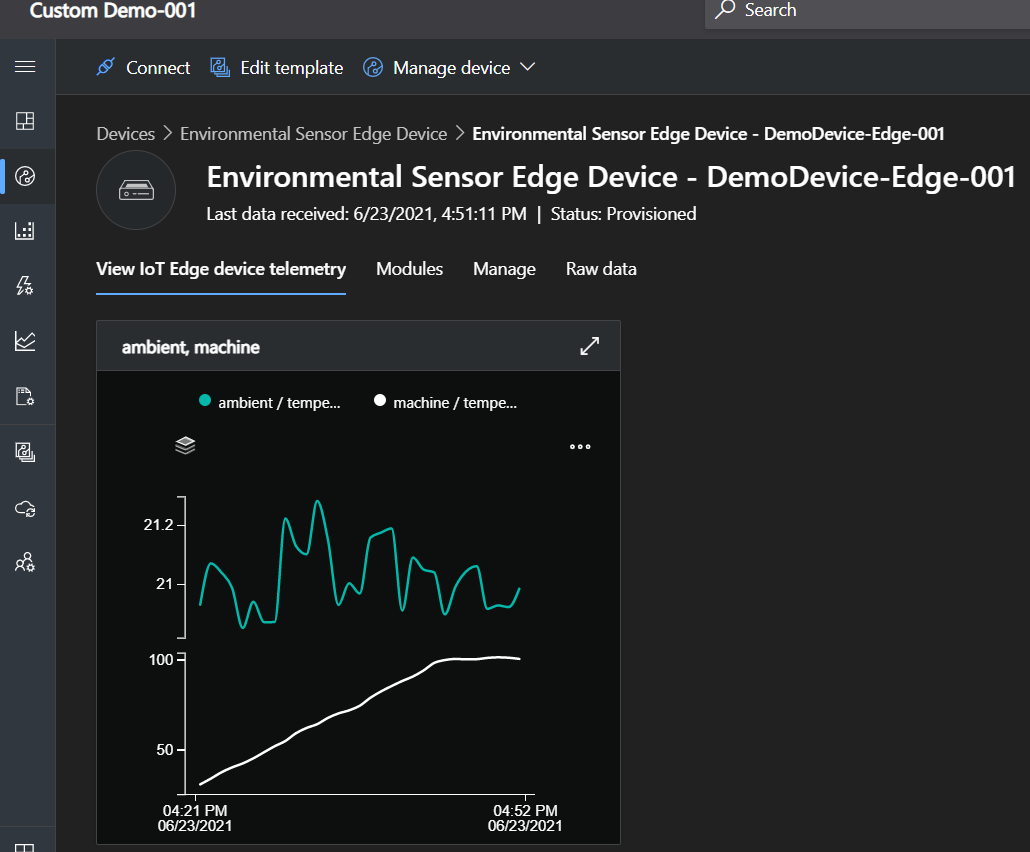
Pic 1)Modules running on Vm.



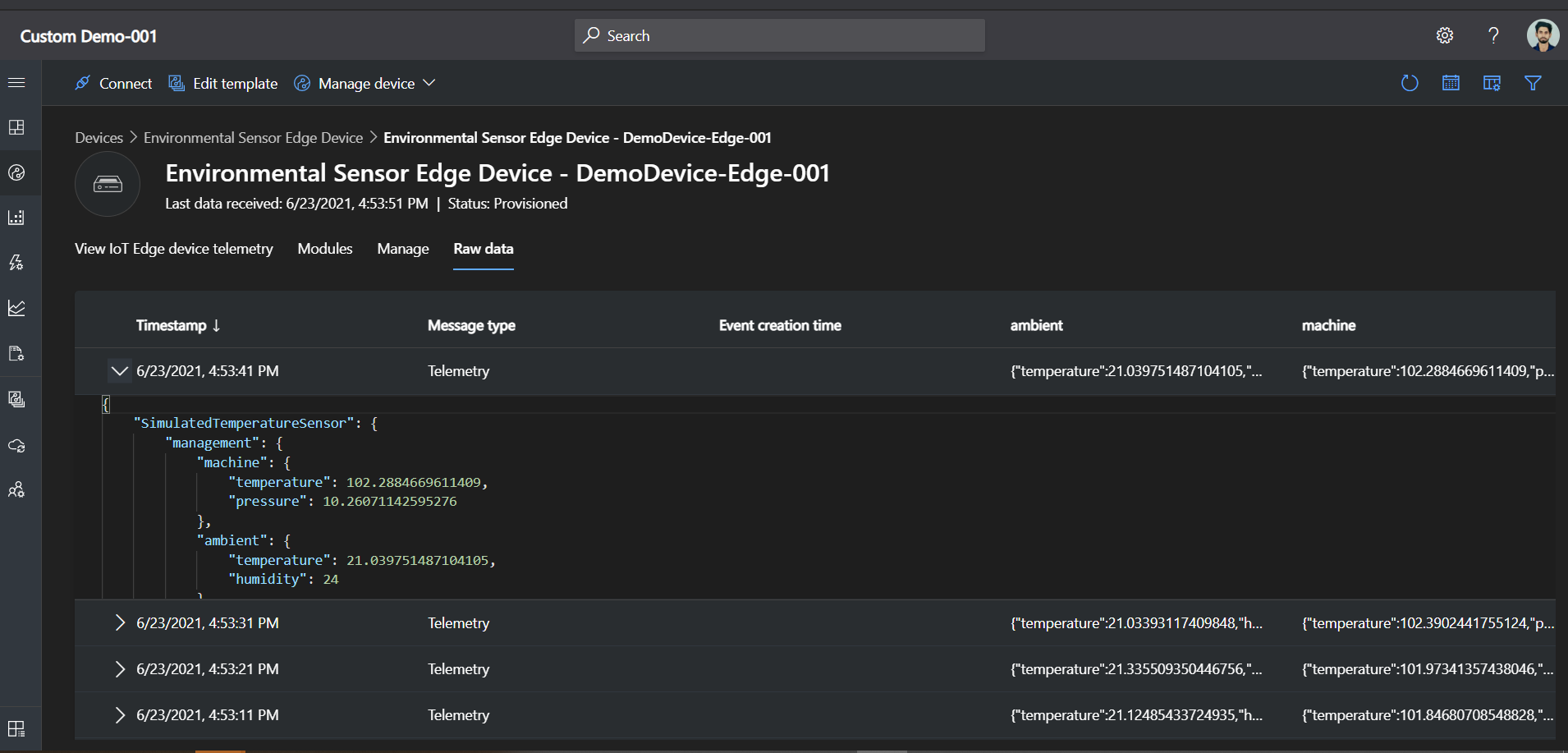
Pic 2) Device Status changed to Provisioned.

Pic 3)Modules and there Status listed on IoT Central.



Pic 4) Telemetry Using Dashboard Function.

Pic 5)Telemetry Using Raw JSON file we manifested during building template of device.



* 1. **Export IoT data to cloud destinations using data export.**

**Steps:**

Used To export IoT data.

**Set up export destination**

Your export destination must exist before you configure your data export. The following destination types are currently available:

* Azure Event Hubs
* Azure Service Bus queue
* Azure Service Bus topic
* Azure Blob Storage
* Webhook

**Set up data export**

Now that you have a destination to export your data to, set up data export in your IoT Central application:

* + 1. Sign in to your IoT Central application.
    2. In the left pane, select Data export.
    3. Select + New export.
    4. Enter a display name for your new export, and make sure the data export is Enabled.
    5. Choose the type of data to export. The following table lists the supported data export types:
* **Telemetry:** Export telemetry messages from devices in near-real time. Each exported message contains the full contents of the original device message, normalized.
* **Property changes:** Export changes to device and cloud properties in near-real time. For read-only device properties, changes to the reported values are exported. For read-write properties, both reported and desired values are exported.
* **Device connectivity:** Export device connected and disconnected events.
* **Device lifecycle:** Export device registered, deleted, provisioned, enabled, disabled, displayNameChanged, and deviceTemplateChanged events.
* **Device template lifecycle:** Export published device template changes including created, updated, and deleted.
  + 1. We could add filters to reduce the amount of data exported.
    2. Add a new destination or add a destination that you've already created. Select the **Create a new one** link and add the following information:
* **Destination name**: the display name of the destination in IoT Central.
* **Destination type**: choose the type of destination.
* For Azure Event Hubs, Azure Service Bus queue or topic, paste the connection string for your resource, and enter the case-sensitive event hub, queue, or topic name if necessary.
* For Azure Blob Storage, paste the connection string for your resource and enter the case-sensitive container name if necessary.
* For Webhook, paste the callback URL for your webhook endpoint. You can optionally configure webhook authorization (OAuth 2.0 and Authorization token) and add custom headers.
  + For OAuth 2.0, only the client credentials flow is supported. When the destination is saved, IoT Central will communicate with your OAuth provider to retrieve an authorization token. This token will be attached to the "Authorization" header for every message sent to this destination.
  + For Authorization token, you can specify a token value that will be directly attached to the "Authorization" header for every message sent to this destination.
* Select **Create**.
  + 1. Select + Destination and choose a destination from the dropdown. You can add up to five destinations to a single export.
    2. Save.
  1. **Create Alert Rules.**

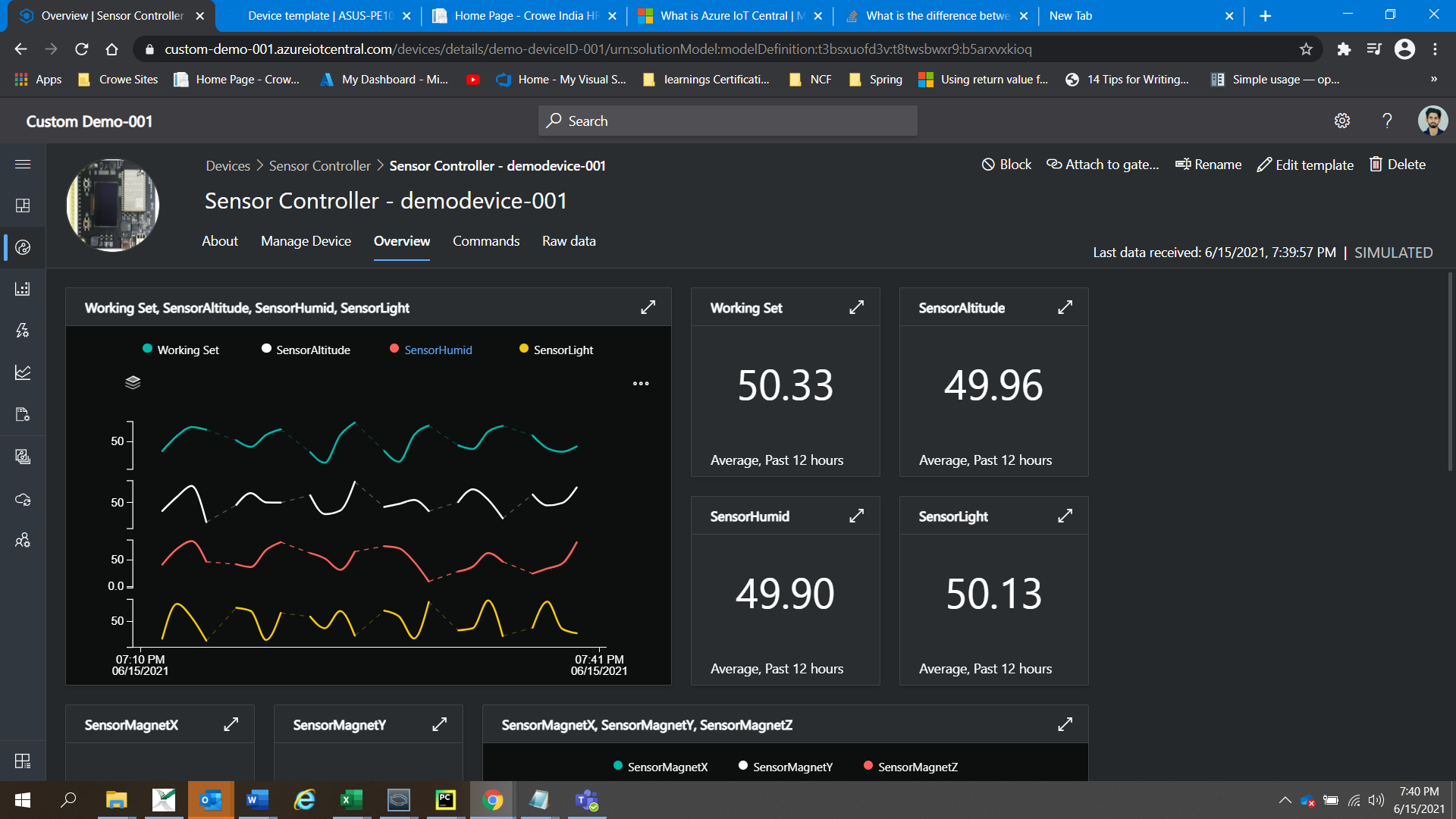
**Steps:**

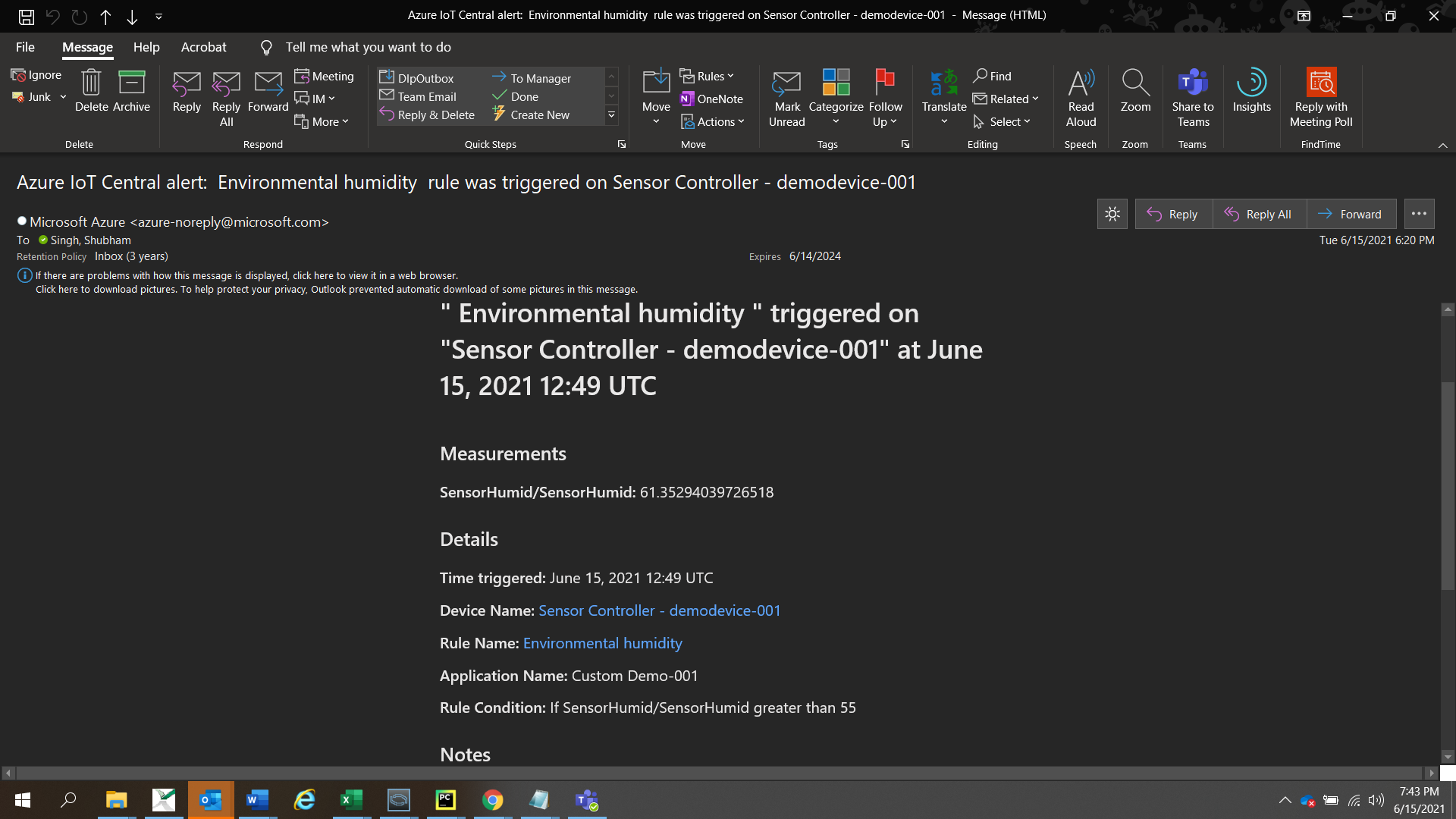
* To add a new telemetry-based rule to your application, in the left pane, select Rules.
* To create a new rule, select Create a rule.
* Enter the rule name.
* In the Target devices section, select desired Device.
* In the Conditions section, you define what triggers your rule. You can choose time and Telemetry.
* To add an email action to run when the rule triggers, in the Actions section, select + Email.
* Select Save. Your rule is now listed on the Rules page.

A sample demo with Creating Application and applying Rules on it.

* Created Application.
* Created Device custom Template.
* -Added Cloud Properties.
* -Added Views.
* -Publish Device Template.
* Added a Simulated Device.
* Created telemetry based rule.
* Added Email as a Alert.
* Monitoring

Pic 1) Device receiving telemetry in IOT Central.



Pic 2) Email Alert Received on temperature rise greater than a certain level.

* 1. **Create Application.**
  2. **Run Job to Manage Device.**

**Steps:**

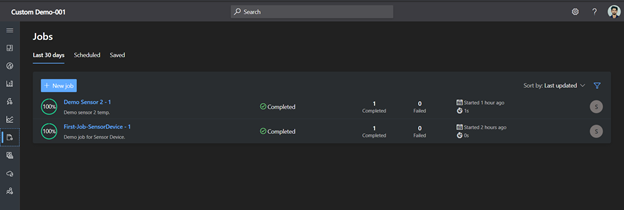
* Open Jobs Section in IoT Central.
* Select Create New Job.
* Enter Configuration: Name of Job , Description of job , Target Device , Job Properties.
* Click Save And Next.
* Enter Delivery Options: Batches and Cancelation Threshold.
* -Batches let stagger jobs for large numbers of devices.
* -The cancellation threshold lets automatically cancel a job if the number of errors exceeds the set limit.
* Click Save And Next.
* Select Schedule: Set the number of time we want to run the job and at what time of day we want to run the job.
* Click Save and Next.
* Review Settings.
* Job is Scheduled Now.

**Types of jobs available are based on : Cloud property, Property, or Command.**

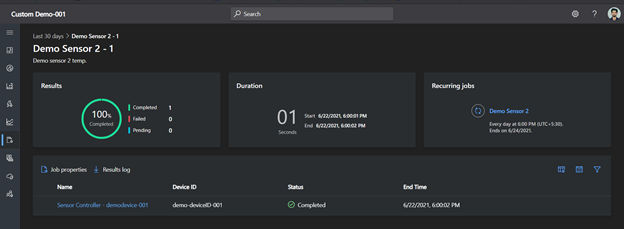
Demo on registering Jobs on Device from IoT Central:

* Purpose of Job was to Raise Target Temperature.
* Job run successfully.
* Target Temperature was raised.

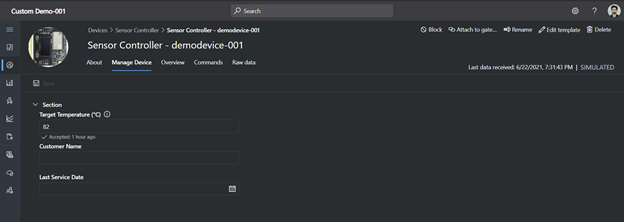
Pic 1)Scheduled Jobs.



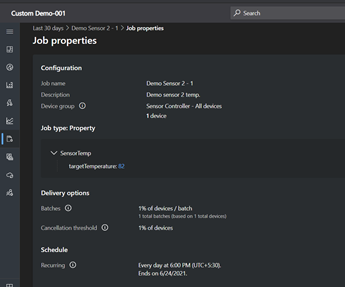
Pic 2)Details of job: Completed, Failed, Pending.



Pic 3)Job Output: Target Temperature was raised.



Pic 4) Job Properties.



* 1. **Create Dashboard.**

**Steps:**

* Click on Dashboard Section.
* Click on Create New.
* Select Dashboard name and Dashboard Type.
* Click on Create.
* Select Device Group.
* Select Device.
* Select telemetry, properties, and commands from the devices to display.
* Add Tile.
* Save.
* We can also add various conditions to get visuals accordingly.
* Now We can see it in our Dashboard Section.

**4. Difference Between IoT Central & IoT Hub**

|  |  |
| --- | --- |
| **IOT Central** | **IOT Hub** |
| SaaS | PaaS |
| Offers to connect, manage & monitor devices at scale. | Offers to connect millions of device securely and at a scale. |
| enterprise-grade IoT solutions | More used for custom IoT Solutions. |
| define and manage device message templates centrally based on device types. | Requires user to create own repository to define and manage device message template. |
| Automate/Code free or little code. | Require code. |
| Message Retention:30 days. | Message Retention:7 days. |
| Built-In Dashboard supported and provides a customizable UI to monitor device health and telemetry. | Built-In Dashboard not supported. |
| Supported SDK's to simulate device:Node.js,Python. | Supported SDK's to simulate device:Node.js,Python,C#,Java,C. |
| Provides data export to : blob storage, event hubs, service bus, Webhook. | Provides data export to : built-in event hub endpoint or can make use of message routing. |
| Built-In Device Provisioning Service. | Device Provisioning Service setup required separately. |