**Video Module Introduction - Edge Analytics**

**Module Introduction - Analytics on the Edge**

In this module, you will learn about

* Characteristics of IoT Edge devices
* Provisioning IoT Edge devices
* Creating and running IoT Edge modules

This module introduces you to a different aspect of an IoT architecture – edge devices. Edge devices host and run the Edge runtime, which allows them to run modules with custom functionality. The goal is to gather telemetry data, gain insights and take action – all within the scope of one device. You will work on configuring and running an edge device. You will explore different types of edge modules, and you will create a custom module for your device.

During this module, you will complete the following hands-on labs:

* Lab 1: Getting Started with IoT Edge
* Lab 2: Implementing Analytics on the Edge
* Lab 3: Deploying an Azure Function to the IoT Edge

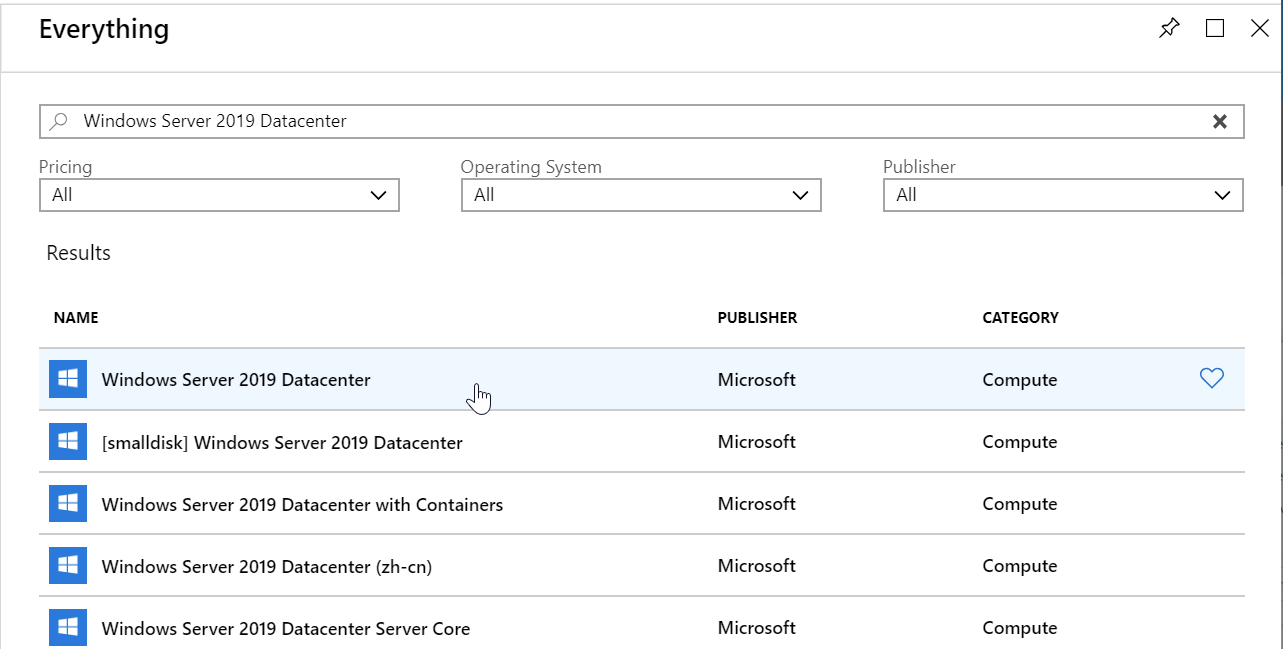
**Provision a virtual machine with nested virtualization**

**Note:** In an earlier version of this course, the VM created was based upon Windows Server 2016. However, the latest version of the IoT Edge runtime has a dependency on the enhancements made to container support in later versions of Windows, there for this task has been updated to use a Windows Server 2019 image.

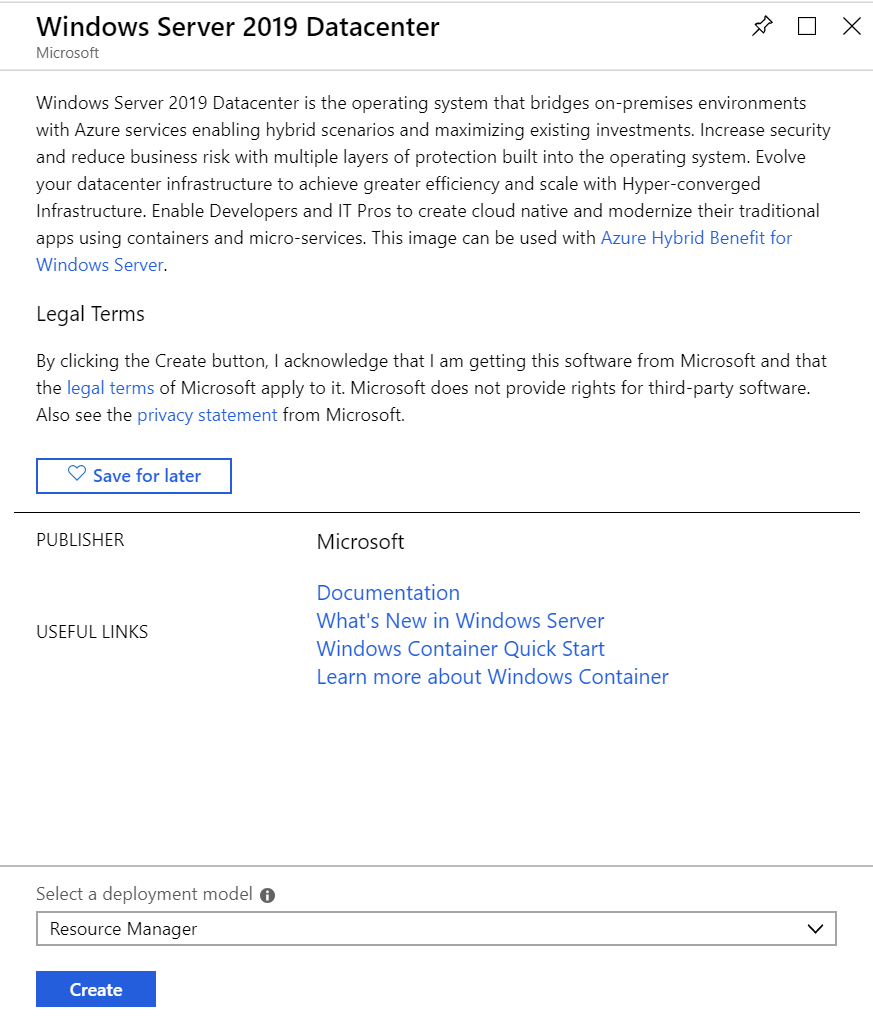
In this lesson, you will provision a Windows server that will take on the role of an IoT Edge device. You will do so by provisioning an Azure virtual machine and enabling nested virtualization. Nested virtualization means having the ability to run hypervisor within an virtual machine, and hypervisor is a requirement for installing Docker for Windows. Not all Azure virtual machines support nested virtualization, so the process requires some careful planning.

The end goal is to have an IoT Edge device that you can use to simulate a wind turbine. You could theoretically start with your own Windows server, install the IoT Edge runtime on it, and skip lesson 2, but it’s recommended you follow this script.

1. On the Azure Portal, click the **Create a resource** link in the upper left.
2. In the search box, enter **Windows Server 2019 Datacenter**



1. Select **Windows Server 2019 Datacenter**
2. Click the **Create** button.



1. Under **Resource group**, select **Create new**
2. Name the resource group **iotedge\_group**
3. Under **Virtual machine name** enter a unique name, such as **edgeturbineXX**, where XX are your initials.

**Note:** The VM name **must not** start with a number - although it will be accepted as a valid name for the VM, TLS encryption requires that hostnames start with a character. Using a number to start the name will cause certificate errors when trying to send telemetry to an IoT Hub.

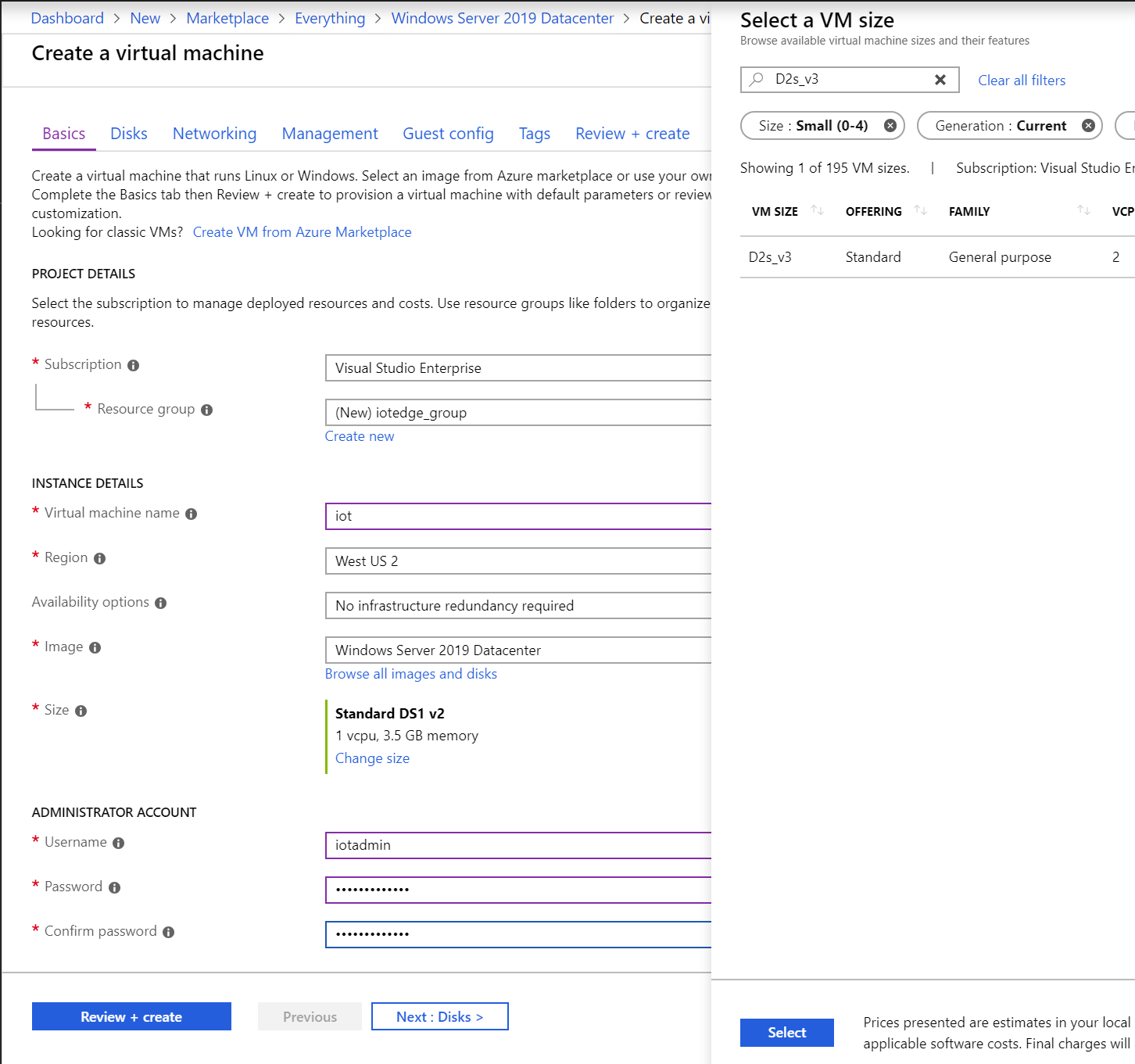
The *Region* value takes some extra care for this exercise. Only a certain subset of Azure virtual machines support nested virtualization, and they are only available in certain areas. We are going to use the *D2s\_v3* VM size. This web page indicates the areas where it is available – make certain to use one of these regions as your location: https://azure.microsoft.com/en-us/blog/introducing-the-new-dv3-and-ev3-vm-sizes/

The v3 vms are becoming increasingly more available, so they may be available in your preferred region when you do this lab. Just check before proceeding.

1. Under **Size**, select **Change size**
2. On the **Select a VM size** page, enter **D2s\_v3** in the search box

If you do not see that as an option, you may not have selected an eligible region. Close the **Select a VM size** window and select a different region, then try selecting **D2s\_v3** again.

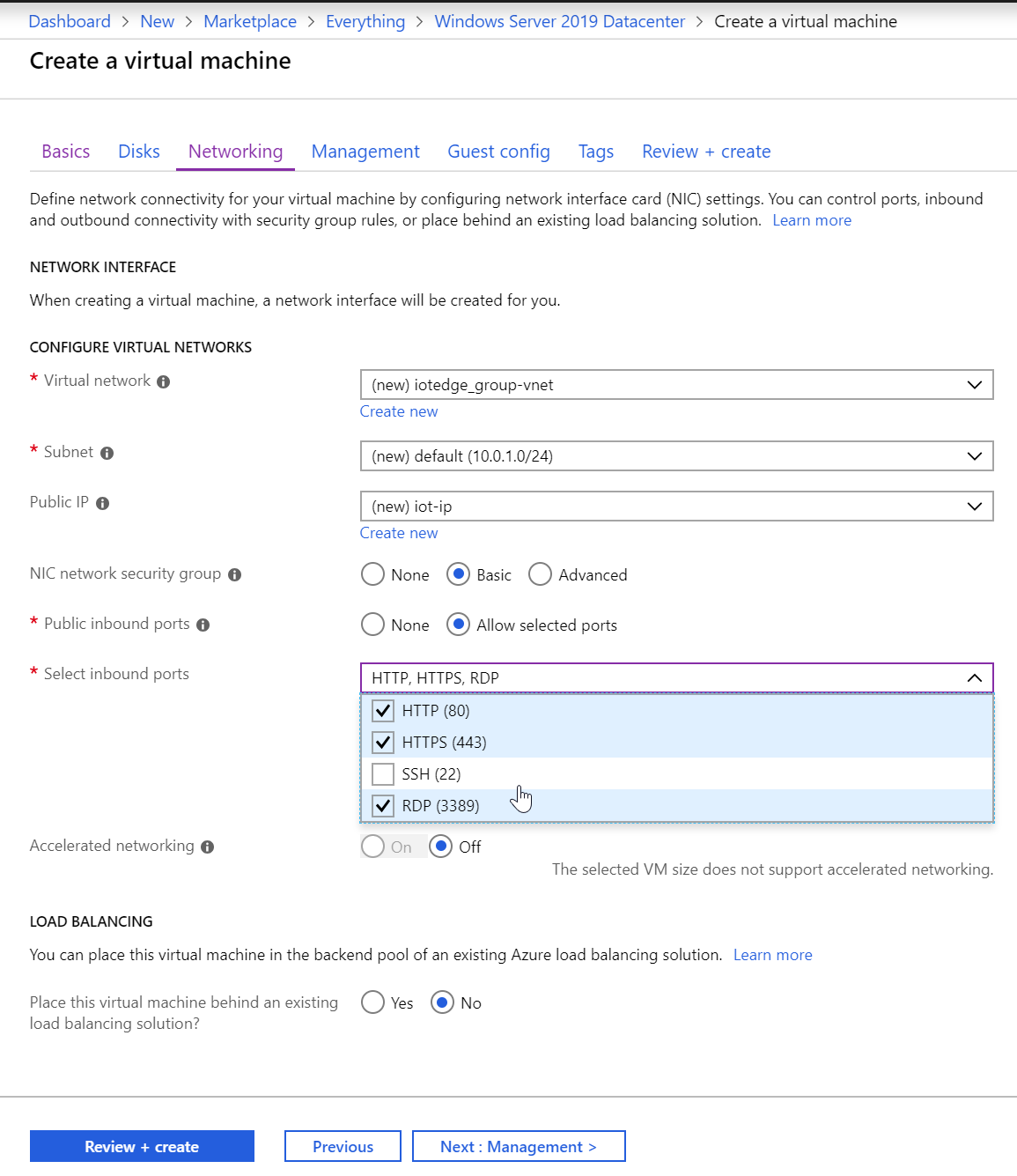
1. Under **Username**, enter **iotadmin**
2. Under **Password**, enter a cryptographically strong password.



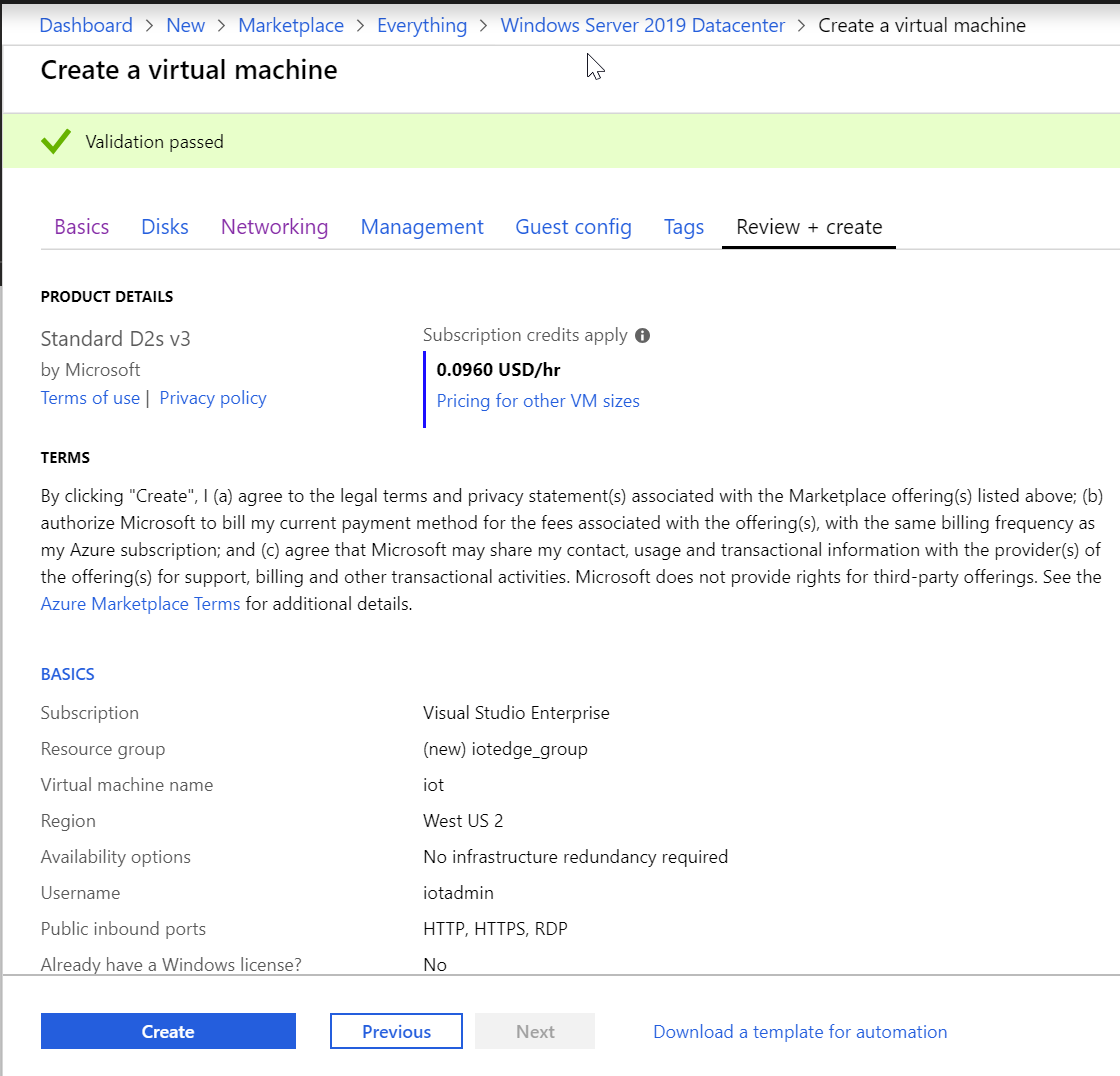
1. Select the **Networking** tab at the top of the window.
2. Expand **Select public inbound ports**.

Select HTTP, HTTPS and RDP. You may get a warning about exposure to the internet.

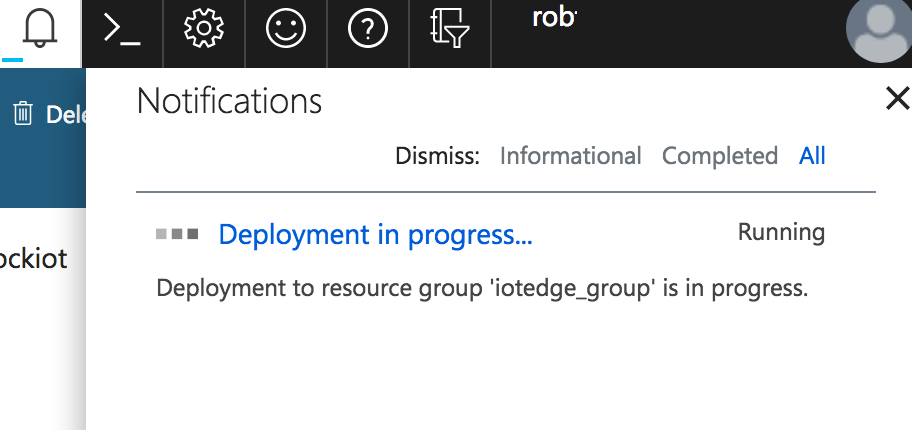
1. Select **Review + create**



1. If validation passes, click **Create**.



1. Check your notifications to monitor the progress of the vm provisioning. It will take a few minutes.



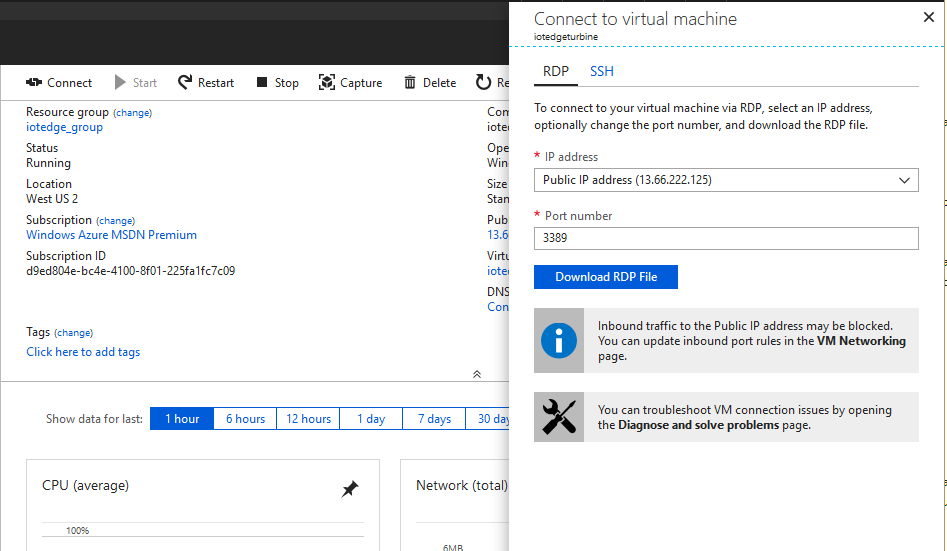
1. Navigate to the new virtual machine’s **Overview** blade.

The new machine should already be started, as a result of the provisioning.

1. Click **Connect**.
2. Under **IP Address**, select **Public IP address**.
3. Select **Download RDP File**.

This will download a file that you can use to connect to your virtual machine. Note its location.

1. Use your favorite RDP client to open the RDP file and remote into the virtual machine.



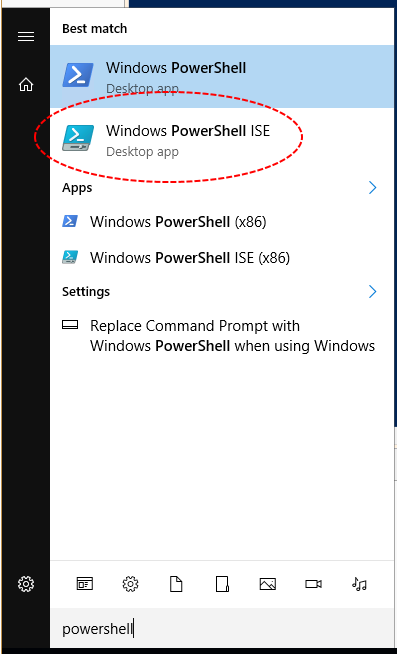
1. Save the following PowerShell file to your local machine.

[BootstrapNestedVirtualization.ps1](https://prod-edxapp.edx-cdn.org/assets/courseware/v1/7871211a2729d6e09066c82894359906/asset-v1:Microsoft+DEV326x+1T2019+type@asset+block/BootstrapNestedVirtualization.ps1)

You will need to run the PowerShell to enable virtualization on this VM.

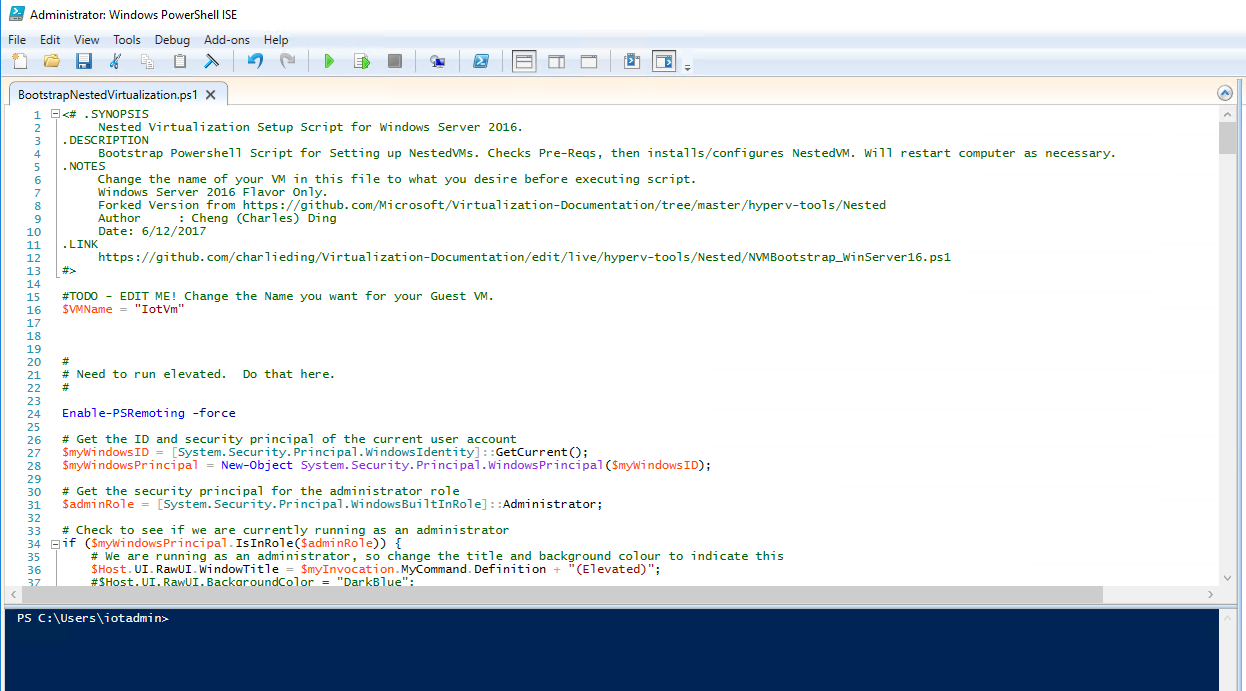
1. Copy the file from your local machine to the virtual machine
2. On the virtual machine's desktop, click the start icon and start typing **powershell**
3. Right-click the icon named **Windows PowerShell ISE**
4. Select **Run as administrator**.

This will open an integrated scripting environment with administrative rights.



1. In the ISE, select **File -> Open**
2. Locate the **BootstrapNestedVirtualization.ps1** file and open it.
3. Select **Debug -> Run/Continue** or click F5.

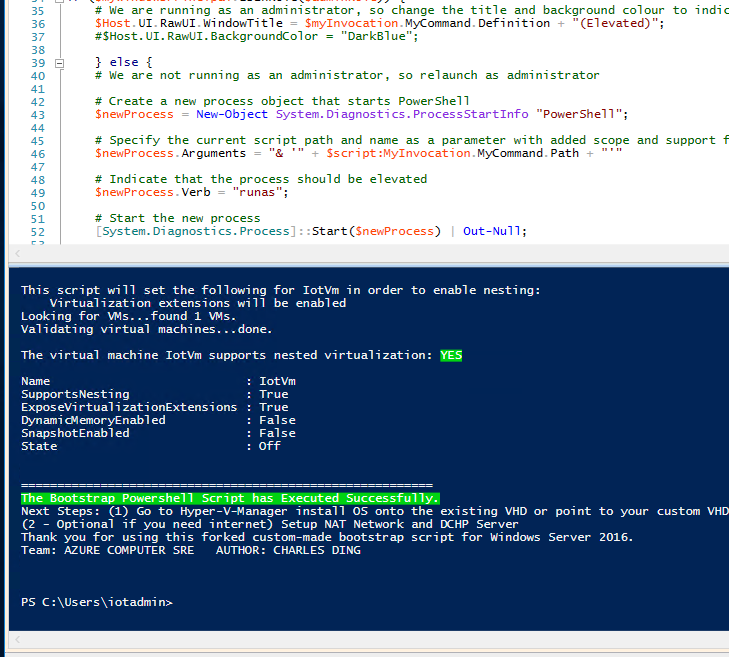
The PowerShell script will start running. It will take a moment as it will need to add Windows Features such as Hyper-V. You will see status messages as it proceeds. The virtual machine will almost certainly restart as part of the process. Give it a moment to restart, then reconnect with your RDP client.



**Note:** Despite being output stating Windows Server 2016, this script functions correctly on Windows Server 2019.

1. Reopen **Windows PowerShell ISE** as an administrator again, and run the **BootstrapNestedVirtualization.ps1** file again, to make certain it is finished.

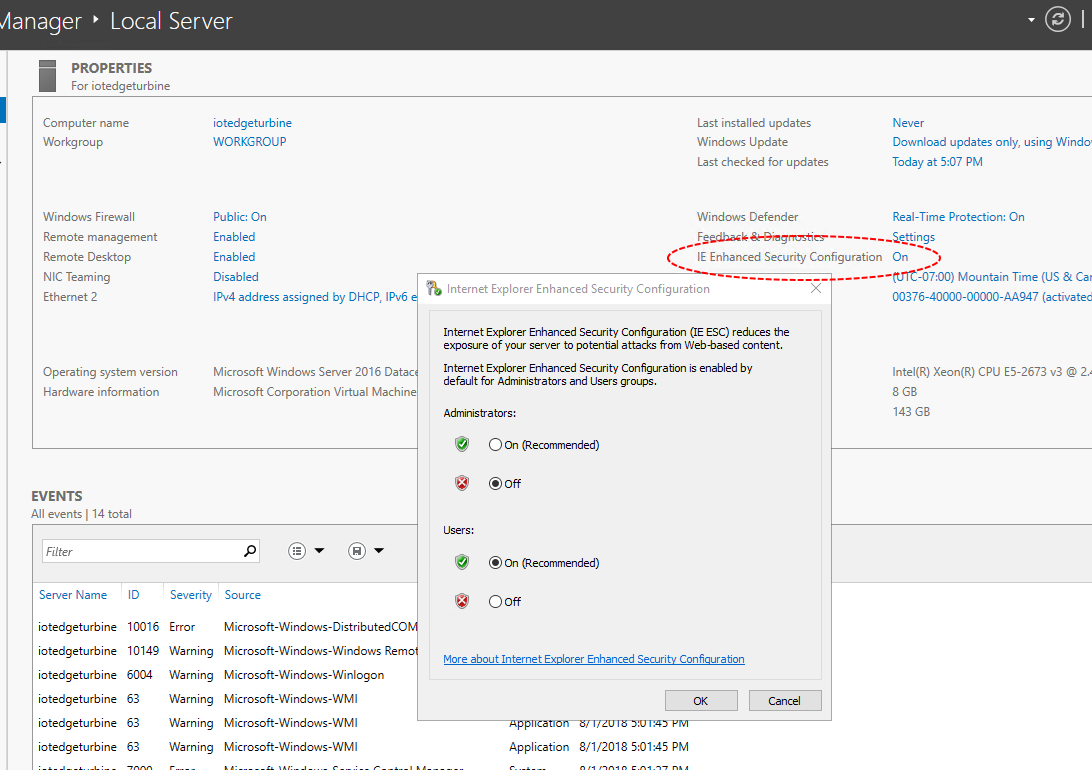
You should get a success message.



Next, you will turn off enhanced security configuration in order to use the stock web browser (Internet Explorer).

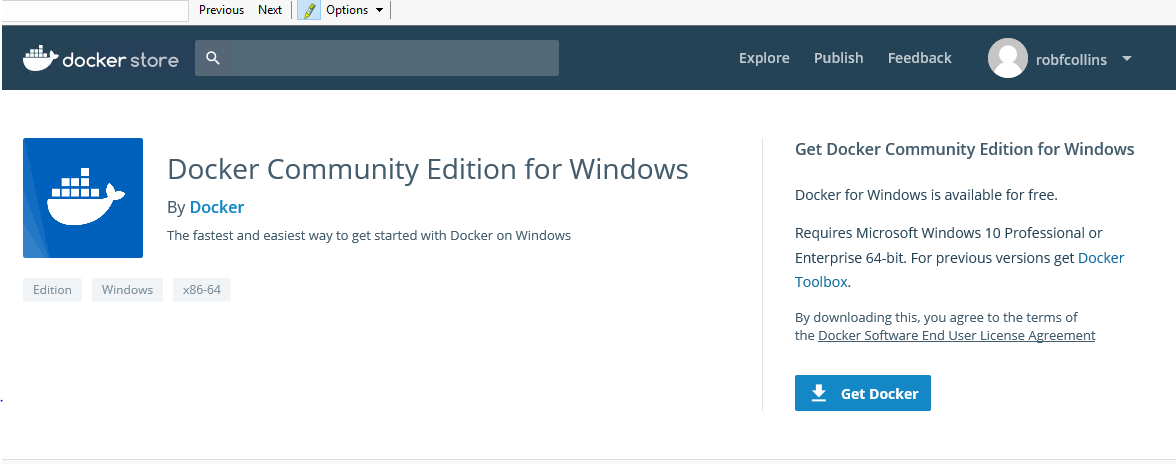
Alternately, you can download your favorite browser to your virtual machine and use that.

1. In the **Server Manager** window (which should be open by default on your VM), select the **IE Enhanced Security Configuration** link that says **On**
2. Under **Administrators**, select the **Off** radio button and click **OK**

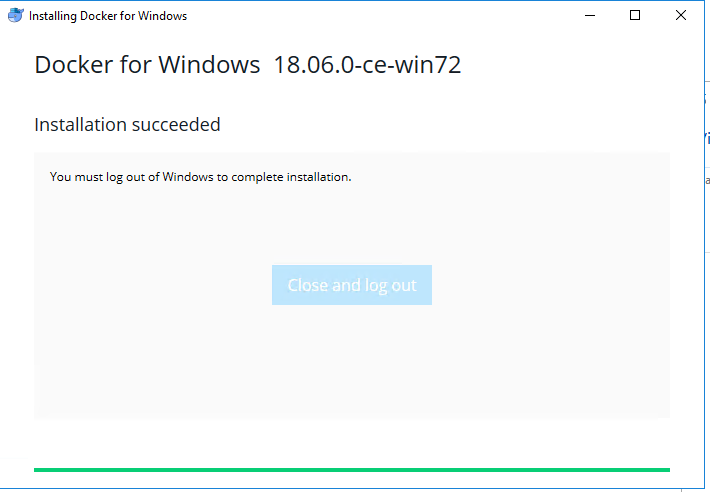


1. Launch IE and navigate to https://docs.docker.com/docker-for-windows/install/ in order to get the Docker for Windows package.

You may need to set up a docker account (or use an existing one), to get access to the installer.

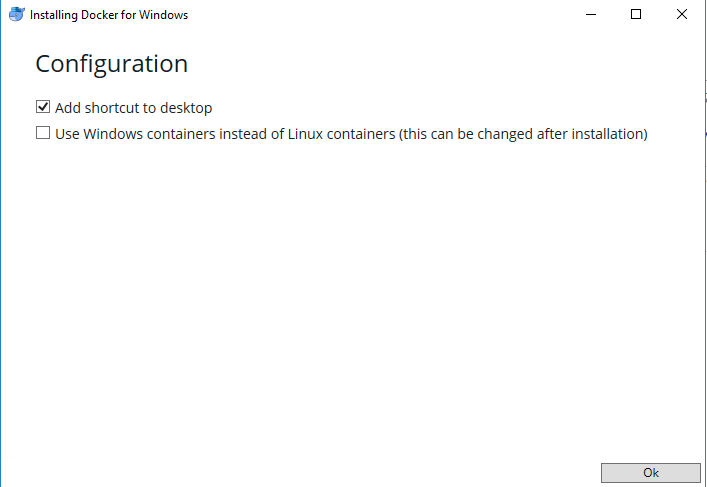


1. Download the msi x64 installer and follow the installation wizard.



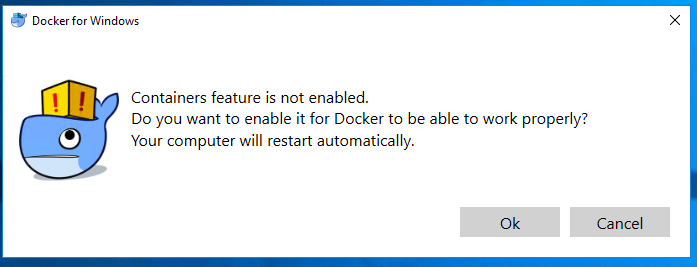
**Note:** the version may differ when you install Docker.

Make certain the **use Windows containers instead of Linux containers** checkbox remains unchecked because you will want to use Linux containers.

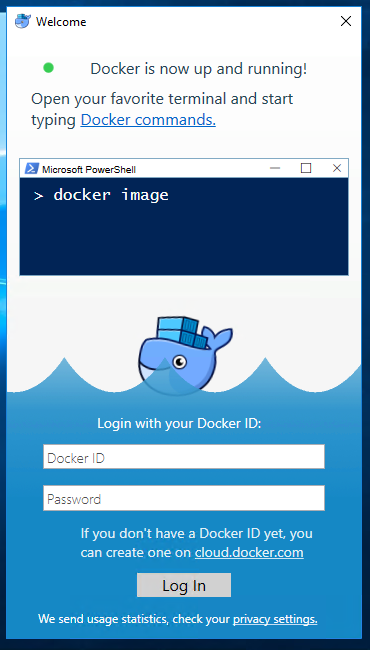


You may be requested to logout so that Docker can start - reconnect your remote desktop session and log back in. After log in, it may take a few minutes for Docker to start, and will then display the following:

1. After installation, restart to get Docker working.



You should get an indicator that Docker is running on your start bar.



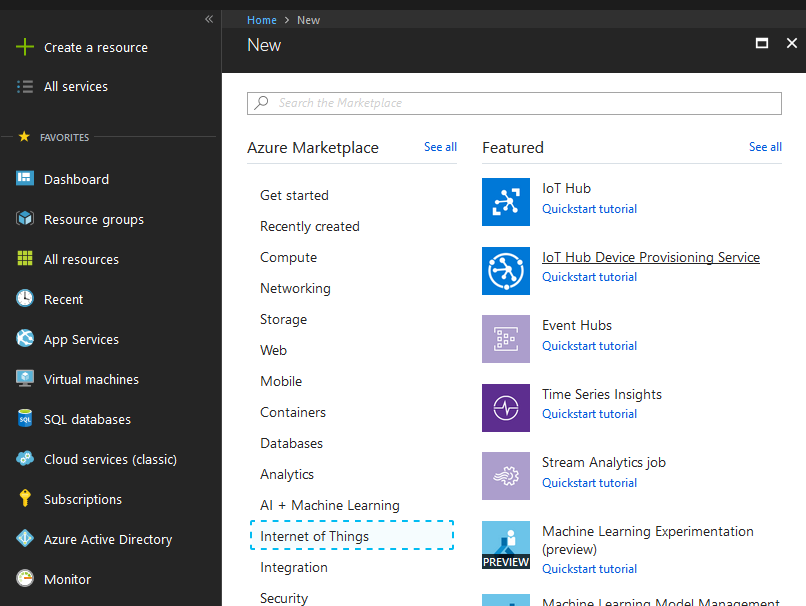
**Summary**

That’s it. If you want to install the IoT Edge runtime on a PC, it needs to be running Docker for Windows. And Docker for Windows require virtualization to be enabled. So you provisioned an Azure virtual machine, enabled nested virtualization and installed Docker.

**Create an IoT Hub**

In this lesson, you will provision an IoT Hub specifically for managing and interacting with your IoT Edge device. You will first provision an IoT Edge device on the Azure Portal. In subsequent lessons, you will provision the IoT Edge runtime on your virtual machine and hook it up to the IoT Hub, for communication and management.

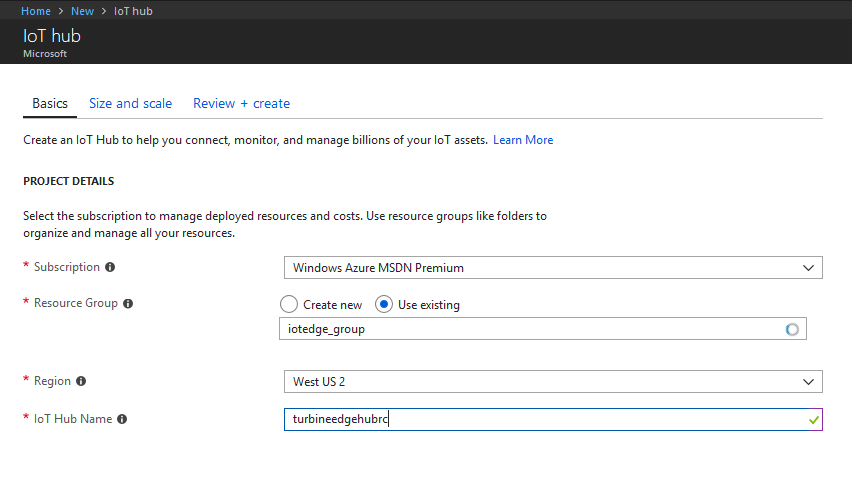
1. On the Azure Portal, select **Create a resource**
2. In the search search box, search for **IoT Hub**
3. Select the **IoT Hub** resource.



1. On the **Basics** configuration page, choose your subscription and select the **iotedge\_group** Resource Group that you created for the last step.

Make certain to choose the same Region that you did for your virtual machine in the last step.

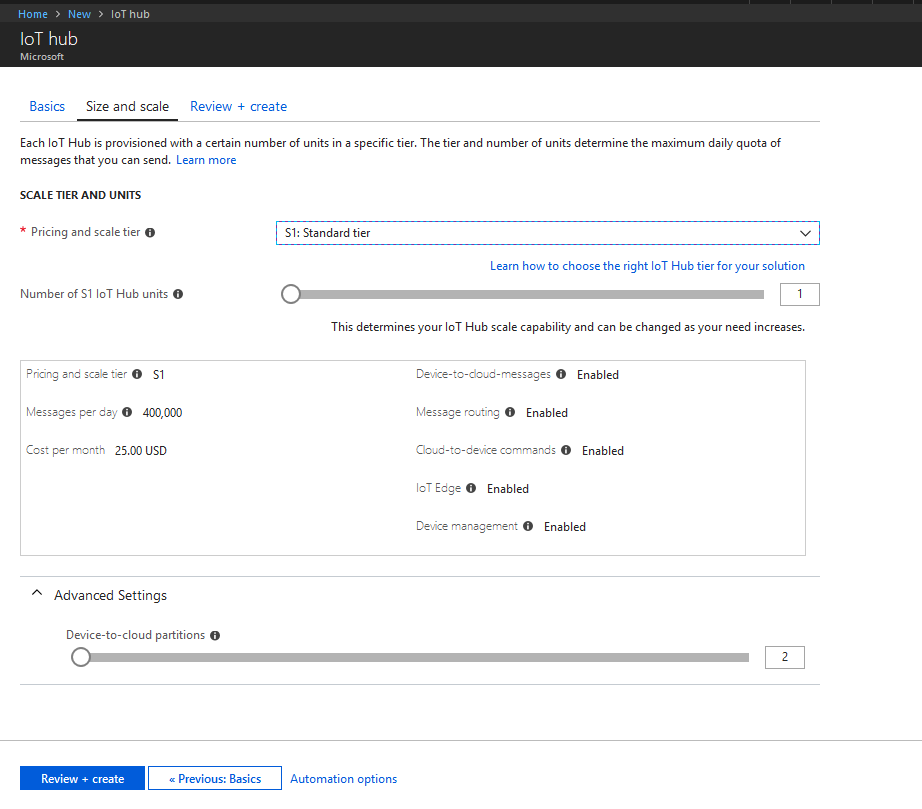
1. Under **IoT Hub Name**, enter a unique name, such as **turbineedgehubXX**, where XX is your initials.
2. Select **Next**



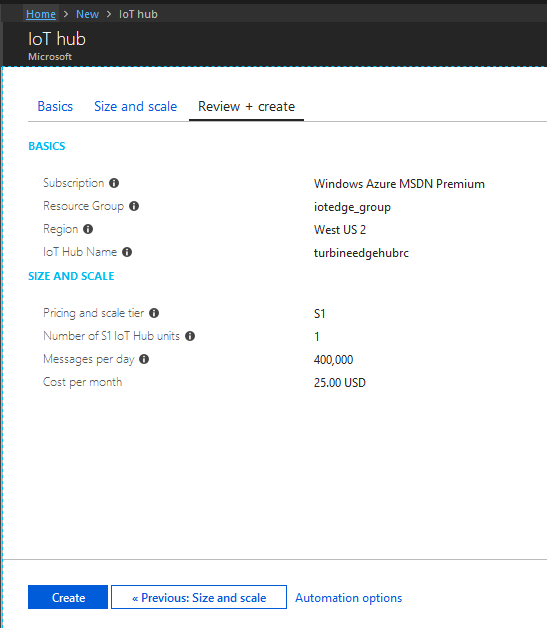
1. Under the **Pricing and Scale tier**, select **S1: Standard tier**.

The default values for the other parameters are fine.

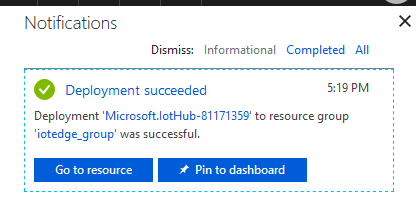
1. Select **Review + create**



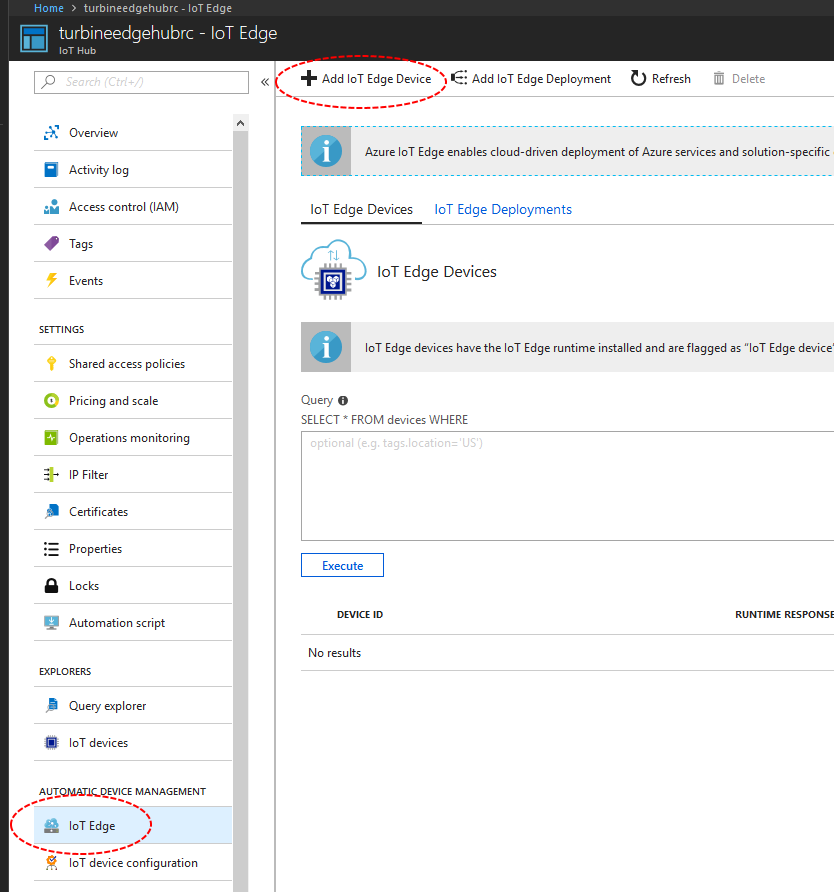
1. Under **Review + create**, select **Create**



It will take a few moments to provision. Keep an eye on your Notifications. When finished, go to the IoT Hub Resource page.



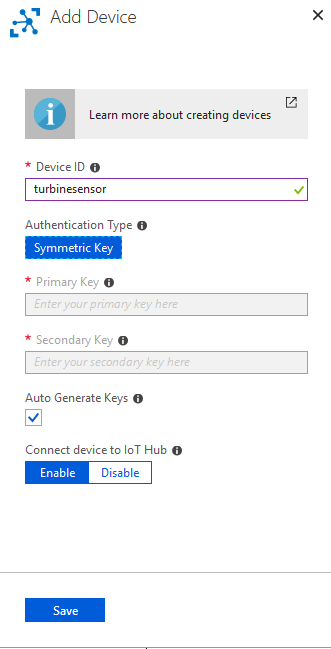
1. On the left of the **IoT Hub** page, select **IoT Edge**.
2. Select **Add IoT Edge Device**.



1. On the **Add Device** page, under **Device ID**, enter **turbinesensor**.

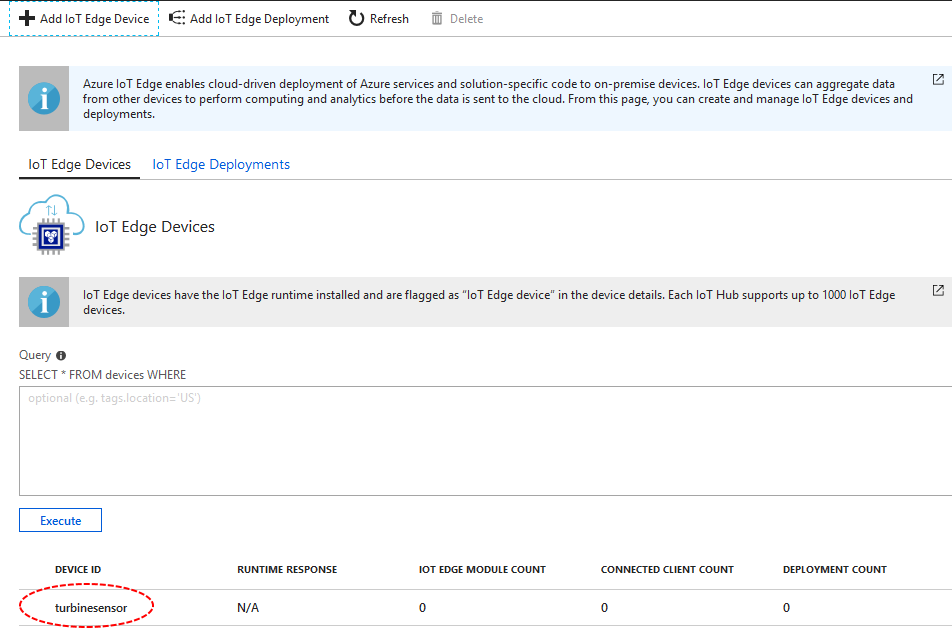
Leave the **Auto Generate Keys** check box selected. This will allow the IoT to generate cryptographic authentication keys.

1. Select **Save**



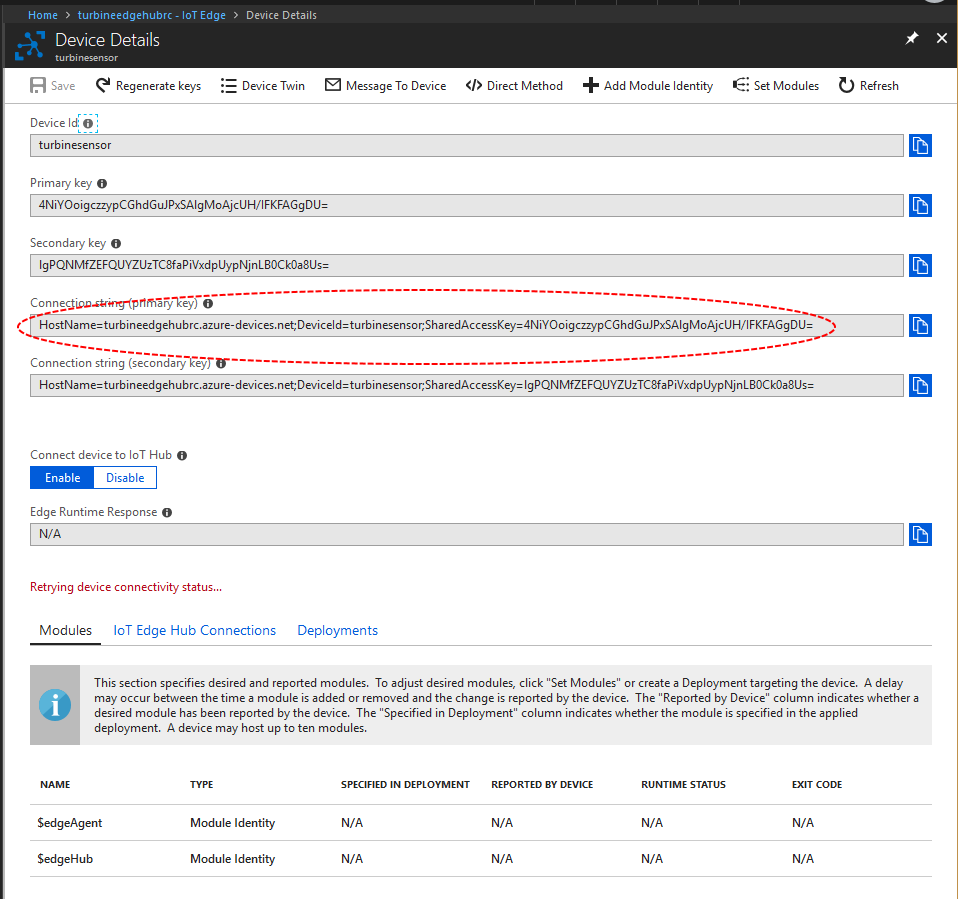
On the **IoT Edge** page, you will now see the **turbinesensor** device.

1. Select its Device ID to navigate to the **Device Details** page.



You will need one of the two **Connection string** values that you see here after you provision your IoT Edge device. The values allow the edge device to connect to this IoT hub. Copy the value or keep this page ready.

Notice that the **Edge Runtime Response** value is currently **N/A**. The physical edge device has not been provisioned yet, so that only makes sense.



**Summary**

In this lesson, you provisioned an IoT Hub on the Azure Portal and created a virtual representation of an IoT edge device.

**Install IoT Edge Runtime**

**Note:** A new version of the IoT Edge runtime has been released that has a reliance on Windows features that are present within Windows Server 2019 - if you have not created a Windows 2019 VM, please return to the **Provision a VM** task and create a new VM. You may shutdown and delete your 2016 VM.

**Note:** The latest version of the IoT Edge runtime has simplified the installation process, so this task has changed significantly.

In this lesson, you will install the IoT Edge runtime on your virtual machine. The IoT Edge runtime allows you to run Azure services such as Functions, Stream Analytics, and Machine Learning on individual devices. You can leverage code from the community, such as AI modules, to extend your edge devices even further. In our case, our Windows virtual machine will run the IoT Edge runtime, and we will use it to represent a wind turbine. In the real world, edge devices can range from small device gateways to full-powered servers.

In this lesson, you will need to execute a series of steps on your virtual machine. A PowerShell script downloads and installs the Azure IoT Edge security daemon. The security daemon then starts the first of two runtime modules, the IoT Edge agent, which enables remote deployments of other modules.

1. If your administrative instance of **Windows PowerShell ISE** is not still open on your virtual machine, re-launch it.
2. In **Windows PowerShell ISE**, create a new script file and paste the following:
3. . {Invoke-WebRequest -useb aka.ms/iotedge-win} | Invoke-Expression; `
4. Install-SecurityDaemon -Manual -ContainerOs Linux -DeviceConnectionString '<connection-string>'

**Note:** The backtick character (`) at the end of the first line and the -ContainerOs Linux setting.

Update <connection-string> with the connection string that you noted in lesson 3, step 11.

1. Press F5 to run it.

The PowerShell script downloads and installs the Azure IoT Edge security daemon. The security daemon then starts the first of two runtime modules, the IoT Edge agent, which enables remote deployments of other modules.

**Note:** In the version of the installer available at the time of writing, an error is thrown that **does not** impact the installation and you may safely disregard (this error may be addressed when you run it):

PS C:\Users\iotadmin> . {Invoke-WebRequest -useb aka.ms/iotedge-win} | Invoke-Expression; `

Install-SecurityDaemon -Manual -ContainerOs Linux -DeviceConnectionString 'HostName=turbineedgehubdm.azure-devices.net;DeviceId=turbinesensor;SharedAccessKey=XavRKI0d1fRTps+4X3kaYrQJHAePFVWJUGd5wfczOYY='

Exception setting "OutputEncoding": "The handle is invalid.

"

At line:3 char:1

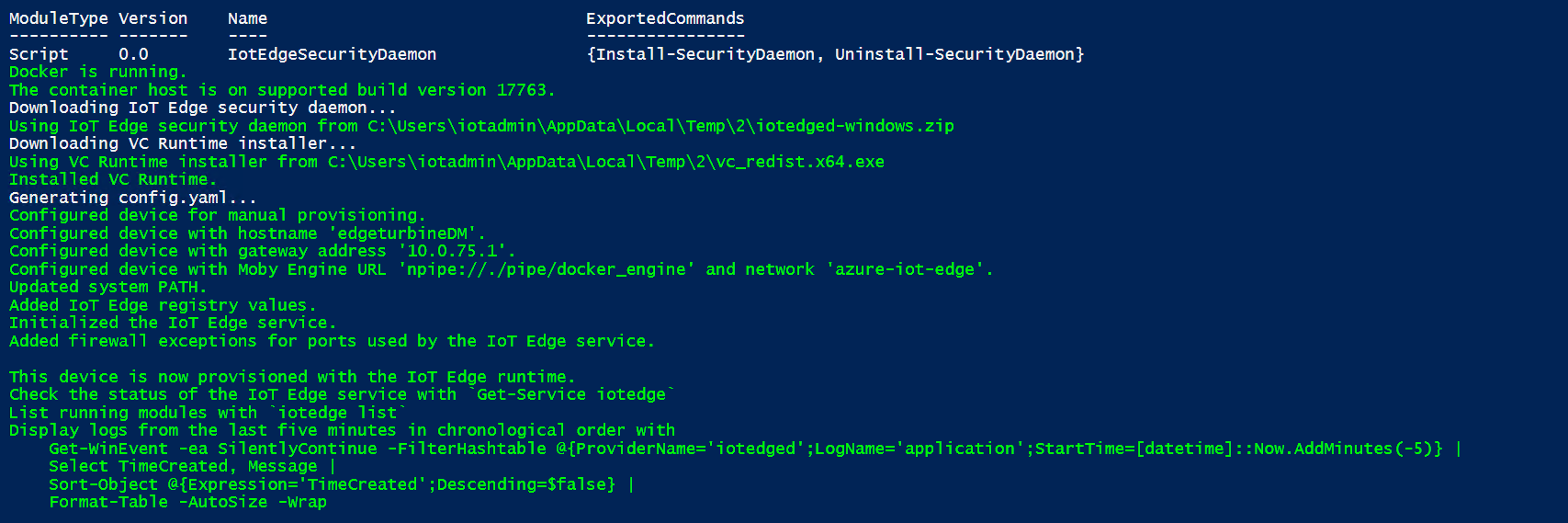
+ [Console]::OutputEncoding = New-Object -typename System.Text.ASCIIEnc ...

+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

+ CategoryInfo : NotSpecified: (:) [], SetValueInvocationException

+ FullyQualifiedErrorId : ExceptionWhenSetting

The script displays a log of the installation process:



1. Once installation has completed, run the following script to verify the Windows Service has started:
2. Get-Service iotedge

You should see the following:

PS C:\Users\iotadmin> Get-Service iotedge

Status Name DisplayName

------ ---- -----------

Running iotedge iotedge

If the service is not running correctly, perform some [troubleshooting](https://docs.microsoft.com/en-us/azure/iot-edge/troubleshoot).

1. Lastly, in the PowerShell window, run the following command:
2. iotedge list

You should see something similar to:

PS C:\Users\iotadmin> iotedge list

NAME STATUS DESCRIPTION CONFIG

edgeAgent running Up 26 minutes mcr.microsoft.com/azureiotedge-agent:1.0

**Note:** If you get an error message, you need to restart your virtual machine.

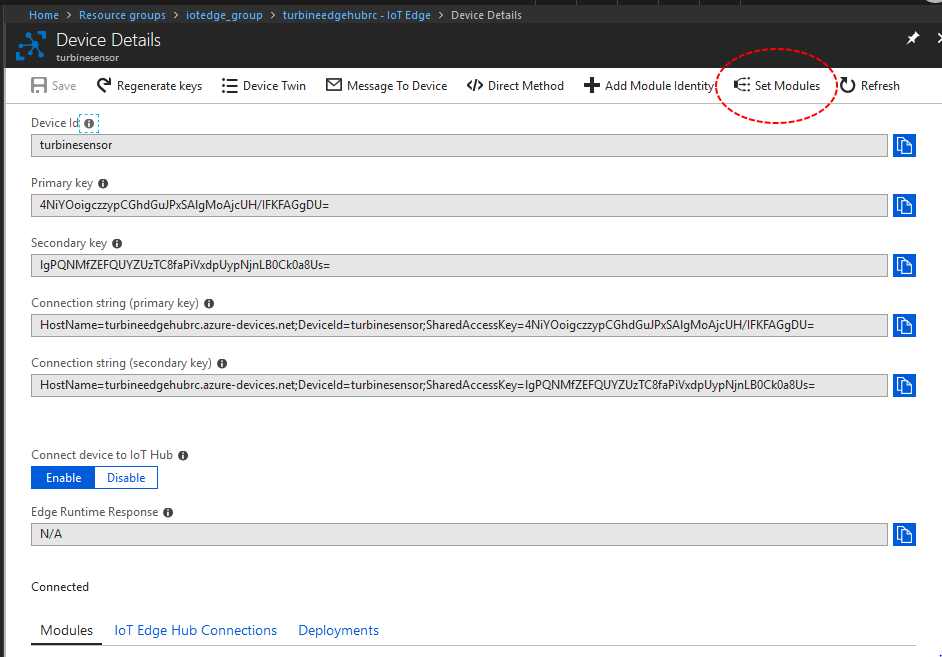
**Summary**

In this module, you went through several steps to install the IoT Edge runtime onto your virtual machine. (Note that some IoT Edge devices come with the Edge Runtime pre-installed). You downloaded the software package and its prerequisites. You set infrastructure values and set yaml configuration values, then started the Edge Runtime Windows Service.

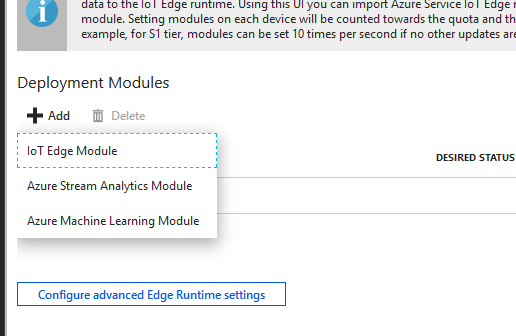
**Deploy a module to IoT edge**

In this lesson, you will deploy an IoT Edge module onto your IoT Edge device. This is the primary benefit of Azure IoT Edge. It can run sophisticated code on IoT Edge devices, using the Edge Runtime. Modules are packaged as Docker Containers and can run certain Azure services, or any variety of custom code. You can create custom modules for your own logic and make them available to the community.

1. Under the Azure Portal, navigate to your IoT Hub, then to the **Device Details** page of the IoT Edge device that you provisioned.
2. At the top of the window, click **Set Modules**.



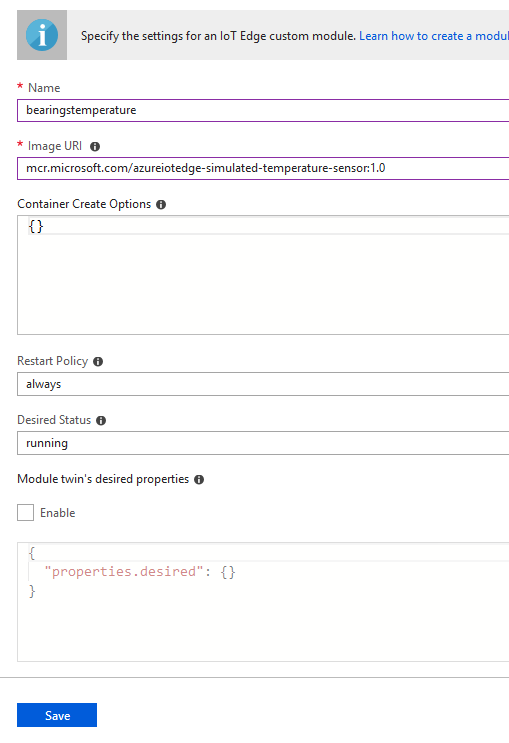
1. Under the **Deployment Modules** section, select **Add**.
2. Select **IoT Edge Module** from the list.



1. On the settings page for the IoT Edge custom module, under **Name**, enter **bearingstemperature**.
2. Under **Image URI** section, add **mcr.microsoft.com/azureiotedge-simulated-temperature-sensor:1.0**.

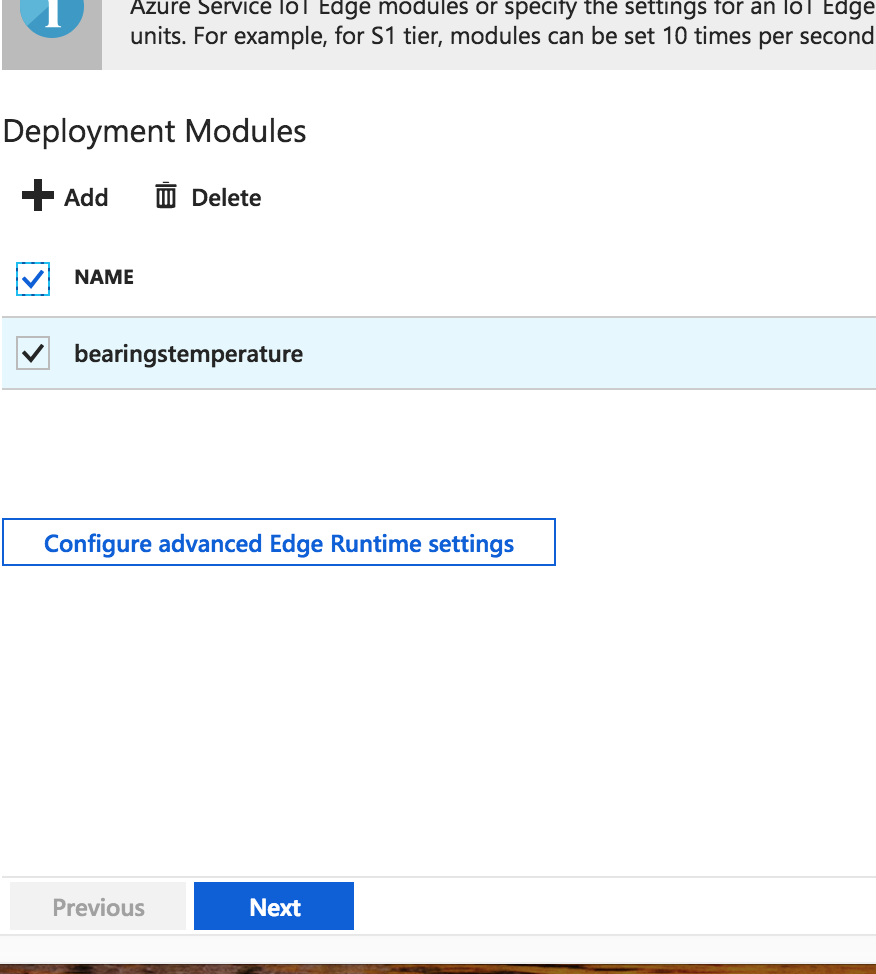
This is a Microsoft-developed IoT module that simulates an IoT temperature sensor. It is hosted in a Microsoft container registry. For our scenario, we will use it to simulate a wind turbine’s bearing temperature sensor.

1. Select **Save**



This will take you back to the **Modules** page.

1. Click the **Name** checkbox, so that the checkbox next to **bearingstemperature** is selected.
2. Then click **Next**



On the **Set Modules** page, you have the opportunity to set routes. You can route messages from the IoT device to the hub or between modules. You can write queries to route messages based on content.

In our case, we will send all messages from **bearingstemperature** to the IoT Hub. To do so, use the following json:

{

"routes" : {

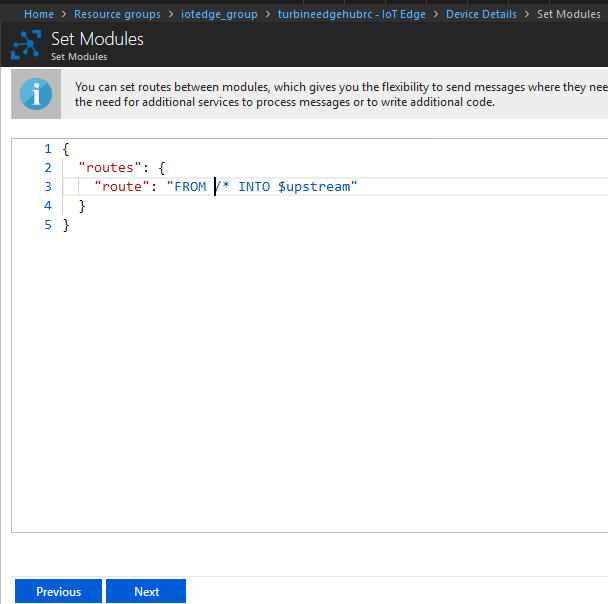
"route" : "FROM /\* INTO $upstream"

}

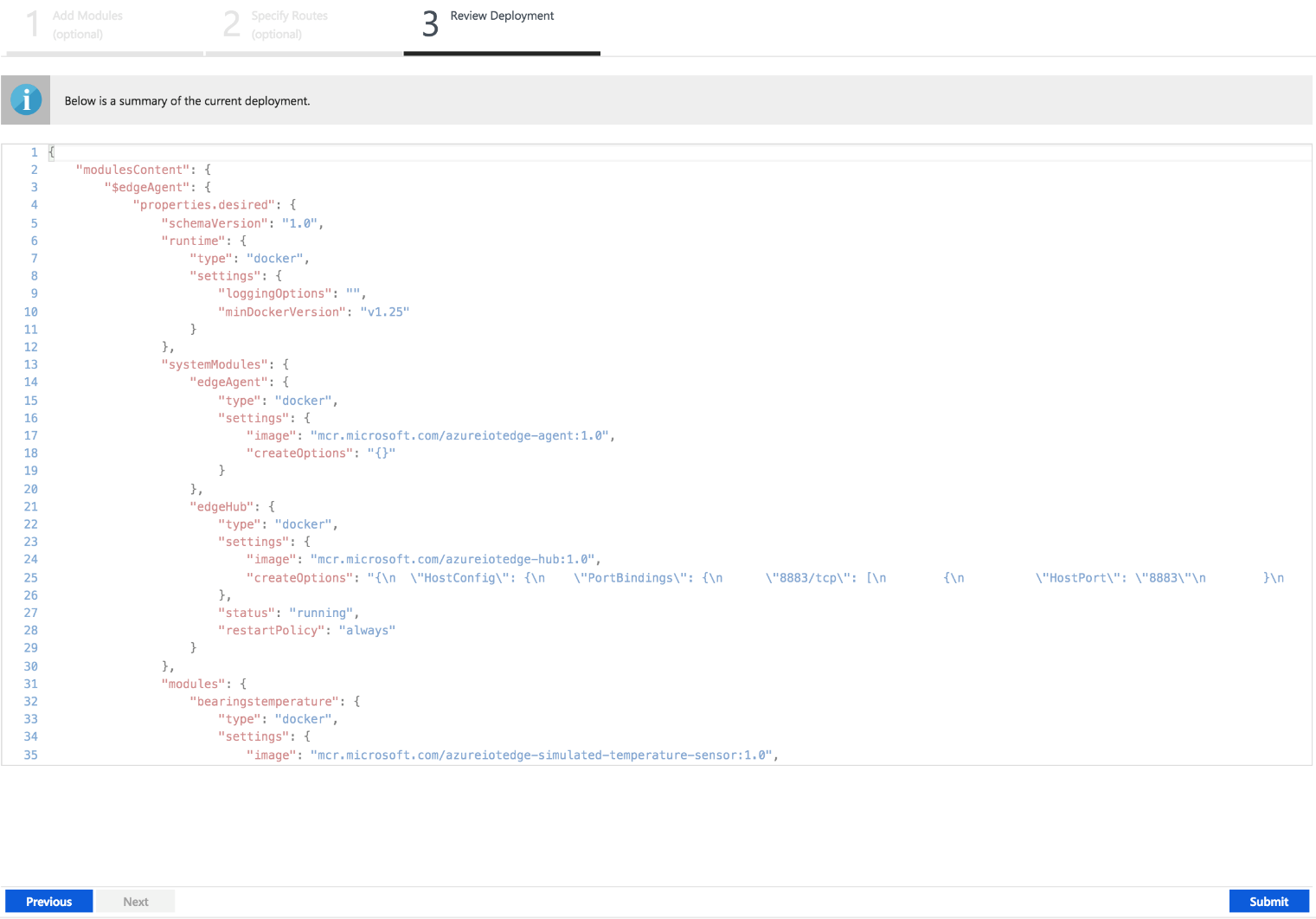
}

The **/** symbol simply means that every message generated by the IoT device will be sent to the IoT hub. The **$upstream** token represents our IoT Hub.

1. Select **Next**

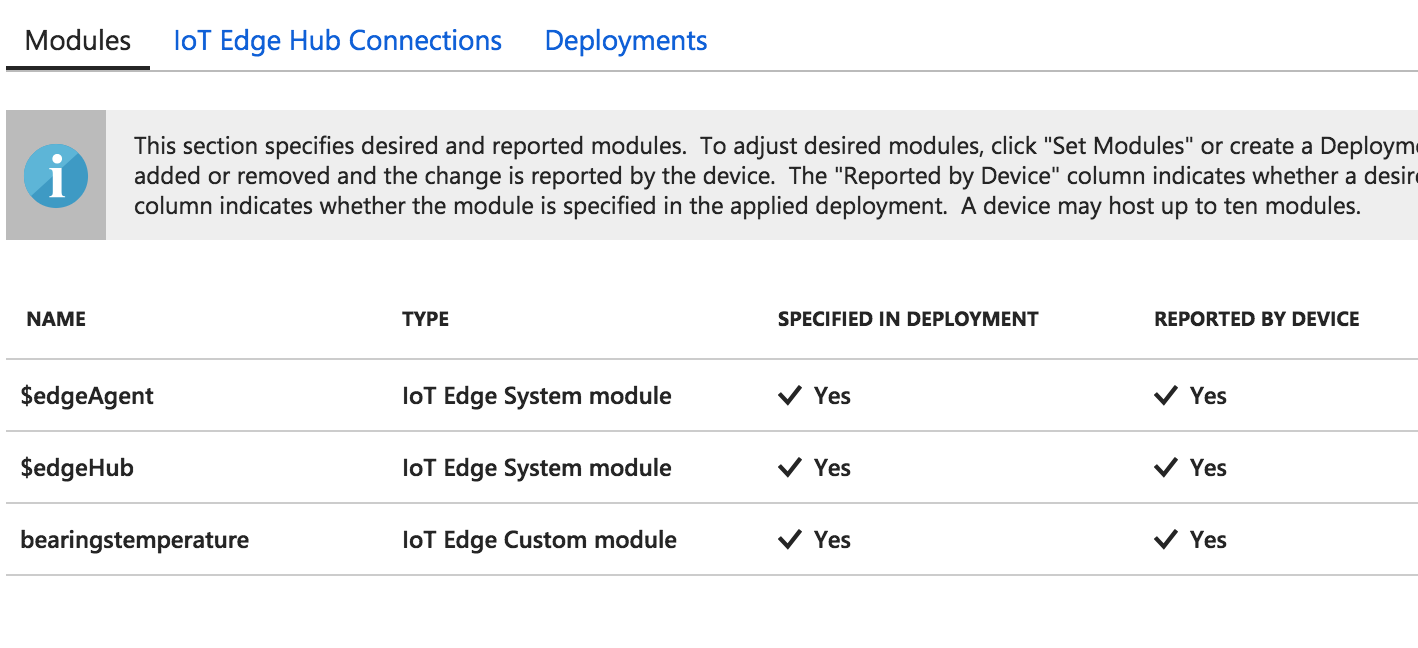


1. Under the **Review Deployment** page, examine the json representation of the module.
2. Select **Submit**



1. Navigate back to the **Device Details** page for the **turbinesensor** device.

Notice in the Modules Section that **bearingstemperature** is listed. It may take a moment for **Reported By Device** to be checked and **Runtime Status** to update to **running**, but once device communication happens, it will update.



**Summary**

In this lesson, you deployed a custom module to your Edge device. You used a publicly available module from Microsoft. In the next lesson, you will examine the module’s messages and output.

**Examine Module Data with IoT Hub Explorer**

In this lesson, you will examine the flow of data from the custom temperature module to the IoT Hub. In previous lessons, simulated devices populated the IoT Hub with data. This lesson will show you how IoT Edge devices can do the same. You will use the “standard” logging mechanisms to view the data, and you will use a the Azure CLI to dig deeper.

1. Log back into your virtual machine.
2. Launch **Windows PowerShell ISE** as administrator.
3. Select **File -> New** to make a new PowerShell script.
4. In the Script window, type **iotedge list**
5. Press F5 to run the script.

In the output window, you will get a list of modules that your IoT Edge runtime is running:

PS C:\Users\iotadmin> iotedge list

NAME STATUS DESCRIPTION CONFIG

bearingstemperature running Up 26 minutes mcr.microsoft.com/azureiotedge-simulated-temperature-sensor:1.0

edgeAgent running Up 26 minutes mcr.microsoft.com/azureiotedge-agent:1.0

edgeHub running Up 26 minutes mcr.microsoft.com/azureiotedge-hub:1.0

Notice that *bearingstemperature* is one of the active modules. That means it is currently sending temperature sensor data to the IoT Hub. You can sample that data.

1. In your PowerShell script window, type **iotedge logs bearingstemperature -f**
2. Press F5 to run the script.

You will get a rolling output of data, such as:

08/04/2018 02:34:38> Sending message: 1861, Body: [{"machine":{"temperature":105.81115236753186,"pressure":10.662030016554262},"ambient":{"temperature":20.951062803832379,"humidity":25},"tim

eCreated":"2018-08-04T02:34:38.3798528Z"}]

08/04/2018 02:34:43> Sending message: 1862, Body: [{"machine":{"temperature":105.9858169266888,"pressure":10.681928510635432},"ambient":{"temperature":20.826914300828665,"humidity":24},"time

Created":"2018-08-04T02:34:43.3844132Z"}]

08/04/2018 02:34:48> Sending message: 1863, Body: [{"machine":{"temperature":106.1146789205329,"pressure":10.696608990946785},"ambient":{"temperature":21.342549229433086,"humidity":25},"time

Created":"2018-08-04T02:34:48.3988291Z"}]

08/04/2018 02:34:53> Sending message: 1864, Body: [{"machine":{"temperature":106.11277206992395,"pressure":10.696391754801462},"ambient":{"temperature":21.37284144799823,"humidity":24},"time

Created":"2018-08-04T02:34:53.4023552Z"}]

08/04/2018 02:34:58> Sending message: 1865, Body: [{"machine":{"temperature":106.38074290839991,"pressure":10.726920078172141},"ambient":{"temperature":20.897229757344924,"humidity":24},"tim

eCreated":"2018-08-04T02:34:58.4035992Z"}]

08/04/2018 02:35:03> Sending message: 1866, Body: [{"machine":{"temperature":106.26097877708311,"pressure":10.713276063212},"ambient":{"temperature":20.540671407264039,"humidity":25},"timeCr

eated":"2018-08-04T02:35:03.4000468Z"}]

08/04/2018 02:35:08> Sending message: 1867, Body: [{"machine":{"temperature":106.58006448628386,"pressure":10.749627599703224},"ambient":{"temperature":21.436372271243656,"humidity":26},"tim

eCreated":"2018-08-04T02:35:08.3985937Z"}]

08/04/2018 02:35:13> Sending message: 1868, Body: [{"machine":{"temperature":107.05641953090972,"pressure":10.803895895926424},"ambient":{"temperature":20.794603404726182,"humidity":24},"tim

eCreated":"2018-08-04T02:35:13.4070742Z"}]

08/04/2018 02:35:18> Sending message: 1869, Body: [{"machine":{"temperature":107.05738977322235,"pressure":10.804006429860774},"ambient":{"temperature":20.831729589184619,"humidity":25},"tim

eCreated":"2018-08-04T02:35:18.4059063Z"}]

08/04/2018 02:35:23> Sending message: 1870, Body: [{"machine":{"temperature":106.99890379188534,"pressure":10.797343469961621},"ambient":{"temperature":20.713722480094862,"humidity":26},"tim

eCreated":"2018-08-04T02:35:23.4084118Z"}]

Note the structure of the data – it’s typical IoT time-series data, with a “timeCreated” property and temperature properties.

[

{

"machine":{

"temperature":106.26097877708311,

"pressure":10.713276063212

},

"ambient":{

"temperature":20.540671407264039,

"humidity":25

},

"timeCreated":"2018-08-04T02:35:03.4000468Z"

}

]

Next, you will use the Azure CLI to monitor the messages sent to the hub. The Azure CLI is a command-line tool providing a great experience for managing Azure resources. The CLI is designed to make scripting easy, query data, support long-running operations, and more.

1. On your virtual machine, navigate to the [Install the Azure CLI website](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest)
2. Click [Install on Windows](https://docs.microsoft.com/en-us/cli/azure/install-azure-cli-windows?view=azure-cli-latest) and then click on **Download the MSI installer**.

Once the installer has downloaded, run it and complete the installer wizard.

In order to access IoT features with the Azure CLI, an extension must be installed. The extension is hosted on GitHub [here](https://github.com/azure/azure-iot-cli-extension) and you can view the available commands [here](https://docs.microsoft.com/en-us/cli/azure/ext/azure-cli-iot-ext/?view=azure-cli-latest).

1. To add the extension, launch a Windows Command prompt (you need to launch a new one if you have one open, to pick up the changes to PATH) and run the following command:
2. az extension add --name azure-cli-iot-ext

The installation completes silently. To verify, enter the following command:

az extension show --name azure-cli-iot-ext

You should see something similar to:

C:\Users\iotadmin>az extension show --name azure-cli-iot-ext

{

"extensionType": "whl",

"metadata": null,

"name": "azure-cli-iot-ext",

"version": null

}

1. Once installed, you must connect to Azure. run the following script:
2. az login

This will launch a browser where you will login with your Azure subscription account.

**Note:** You may be prompted to select the browser to use if you have installed more than one - chose your desired browser and proceed.

After you have logged in, you may close the browser when prompted to do so. The command window will show the details of the subscription(s) associated with your account as JSON.

**Note:** If you have more than one subscription associated with the account, ensure the correct subscription is set as default. You are looking for a JSON property "isDefault": true to indicate the subscription that will be used. If you need to change the default, use the following command:

az account set --subscription <MyIdSubscription>

Replace the **<MyIdSubscription>** with the GUID from the "id" field listed when you logged on.

1. Next we need to generate a Shared Access Signatures (SAS) token to use when connecting to the IoT hub. Enter the following command:
2. az iot hub generate-sas-token --duration 3600 -n <yourhubname>

Replace **<yourhubname>** with the name of the IoT Hub you created in the **Create an IoT Hub** task.

You will see something similar to:

C:\Users\iotadmin>az iot hub generate-sas-token --duration 3600 -n <yourhubname>

{

"sas": "SharedAccessSignature sr=XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"

}

1. Now we have our SAS Token, we can start monitoring the hub messages by running the following command:
2. az iot hub monitor-events --hub-name \*\*\<yourhubname\>\*\*

**Note:** When you run this for the first time, you may see a prompt stating that a dependency must be updated, enter **Y** to update and then run the command.

You will see output similar to the following:

Starting event monitor, use ctrl-c to stop...

{

"event": {

"origin": "turbinesensor",

"payload": "{\"machine\":{\"temperature\":20.776163996209466,\"pressure\":0.97449969577069862},\"ambient\":{\"temperature\":21.487957738799956,\"humidity\":24},\"timeCreated\":\"2019-02-02T21:37:10.7021568Z\"}"

}

}

{

"event": {

"origin": "turbinesensor",

"payload": "{\"machine\":{\"temperature\":21.563345134753426,\"pressure\":1.0641785596554536},\"ambient\":{\"temperature\":21.331771905455632,\"humidity\":26},\"timeCreated\":\"2019-02-02T21:37:15.8467527Z\"}"

}

}

**Note:** If you **do not** see a series of events, then it means the **bearingstemperature** edge module has already sent 500 messages and has stopped sending messages. To restart the module, open another command prompt and enter the following:

iotedge restart bearingstemperature

1. When you have finished monitoring the events, enter **CTRL-C** followed by **Y** to exit the monitoring.

**Summary**

In this module, you worked through techniques for monitoring IoT Hub messages and for interfacing with you IoT Hub. You did so with built-in Azure tools, as well as the Azure CLI and the IoT extension.

**Introduction to Azure Streaming Analytics as IoT Edge modules**

Contoso Wind Power has used its cold and warm storage analytics to gain insights about its operations. It is now ready to put the insights into action and use analytics to help manage its wind turbines.

In the previous lesson, you created an IoT Edge device on a virtual machine then deployed a custom IoT Edge module to send simulated temperature sensor data to your IoT Hub. This lesson will expand upon that infrastructure. You will create an Azure Stream Analytics job – similar to the ones you used previously for warm storage and cold storage. But this time, instead of running the job in the Azure cloud, you will deploy it to your Azure IoT Edge device.

That is one of the prime benefits of IoT Edge platform. It can run the same types of applications and code on the edge that it can in the cloud. Using the Azure Portal, the details of Edge Modules that run Azure services are largely abstracted away. You configure a module, deploy a module to an Azure IoT Edge device and then manage that device through your IoT Hub. We will walk through that process in this lesson.

We will also take a closer look at routing with IoT Edge. Whereas cloud Azure Stream Analytics jobs have inputs and outputs to endpoints such as IoT Hubs and Cosmos DB databases, Edge jobs have endpoints that are *Routes* – communication channels between modules, or with the IoT Edge device, or upstream to the cloud.

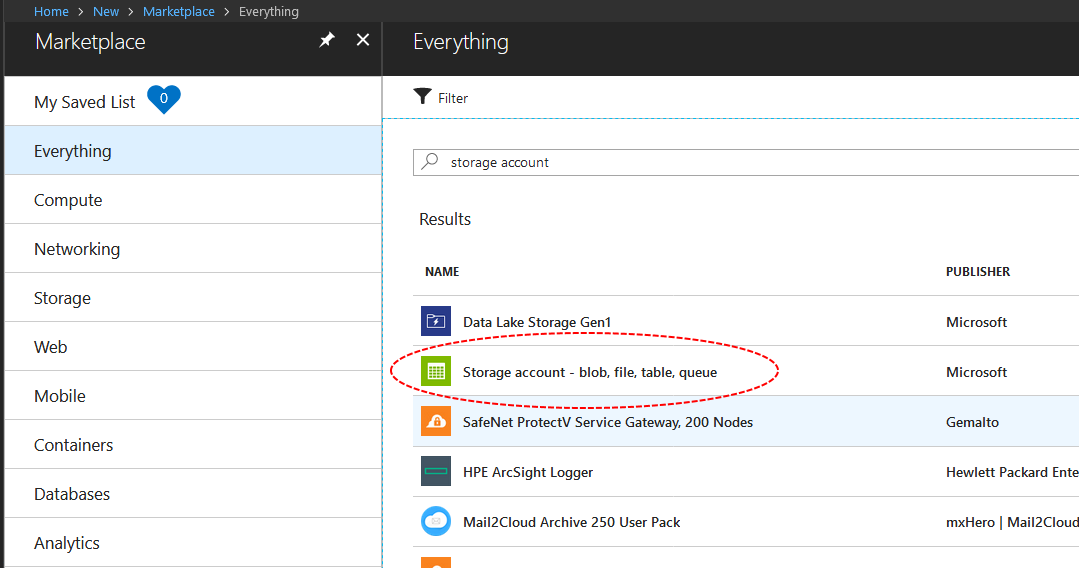
**Create an Azure Streaming Analytics job for IoT Edge**

Azure Stream Analytics jobs deployed to IoT Edge are very similar to jobs deployed to the cloud, but there are some important differences. With Edge jobs, the inputs and outputs are a combination of aliases and routes.

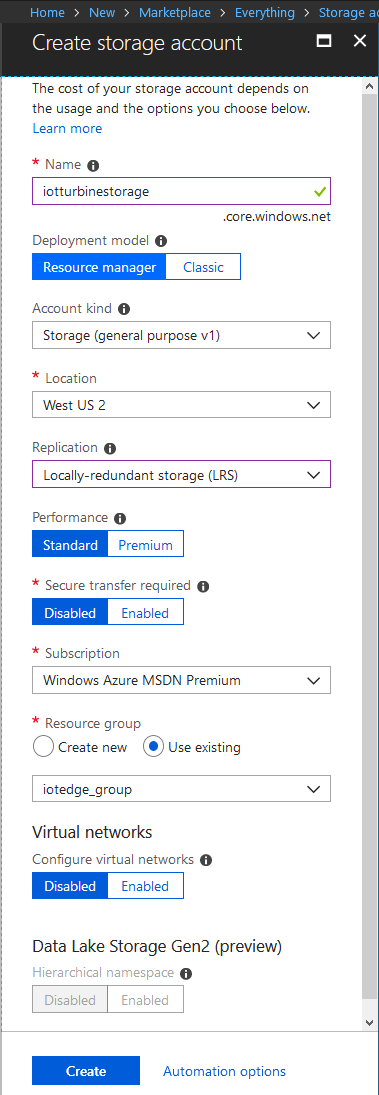
Azure Stream Analytics jobs that run on the edge have a dependency on Azure Storage accounts, which act as endpoints for job output.

Your first step will be to create a storage account.

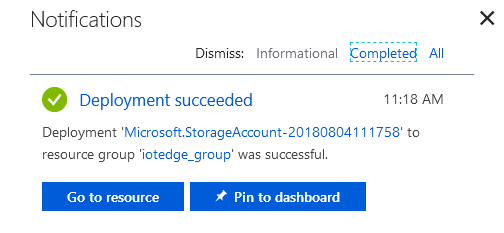
1. On the Azure Portal, search for **storage account**
2. Select **Storage account - blob, file, table, queue**
3. Select **Create**



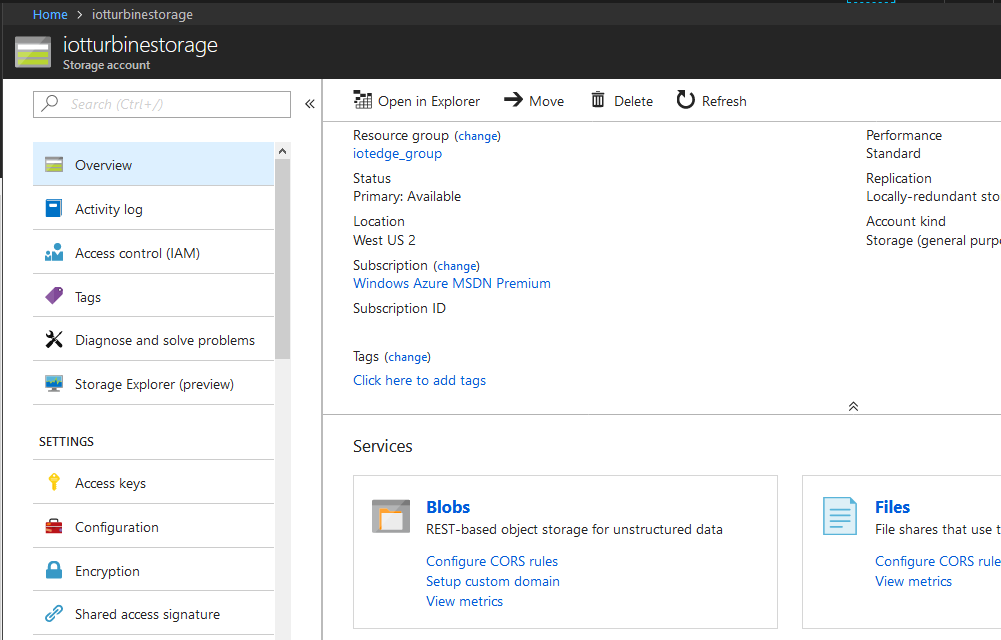
1. On the **Create storage account** page, assign a unique name like **iotturbinestorageXX**, where XX is your initials.
2. Choose **Locally-redundant storage** for replication.
3. Use the same **Resource group** and **Location** you’ve used for previous modules. The rest of the defaults are satisfactory.



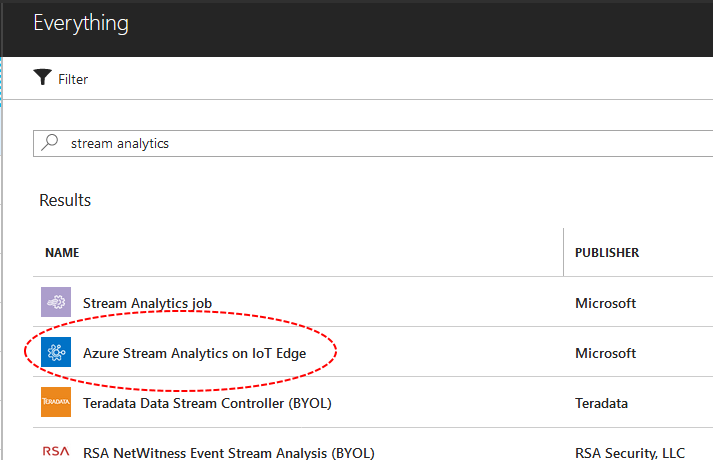
1. It may take a moment to provision, so watch your notifications.



1. Navigate to the storage account's **Overview** blade.

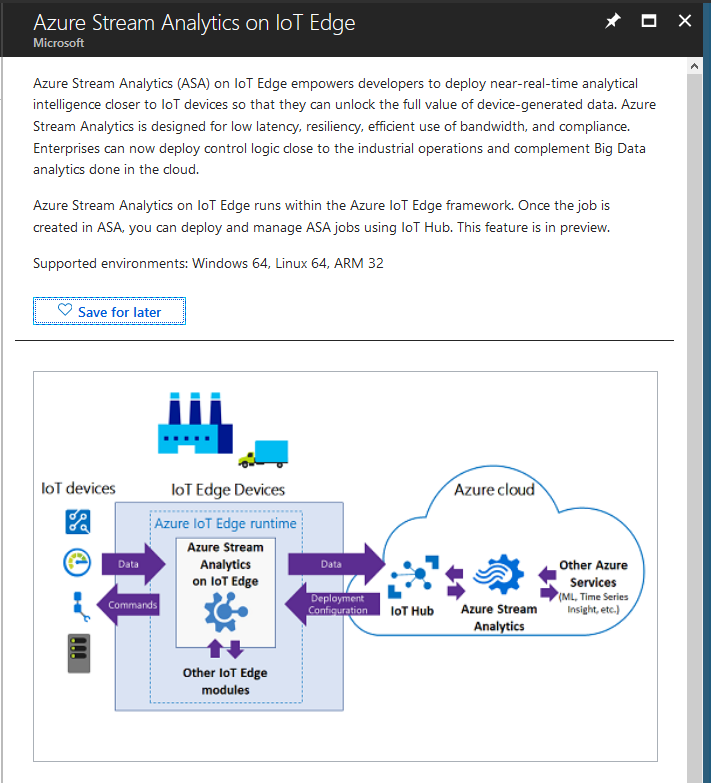


1. Select **Create a Resource** again.
2. Search for **stream analytics**
3. Choose **Azure Stream Analytics on IoT Edge**



You can leverage the knowledge you have gained using Azure Streaming Analytics in the cloud for use on the edge. ASA jobs for IoT Edge are structured the same way.

1. Select **Create**



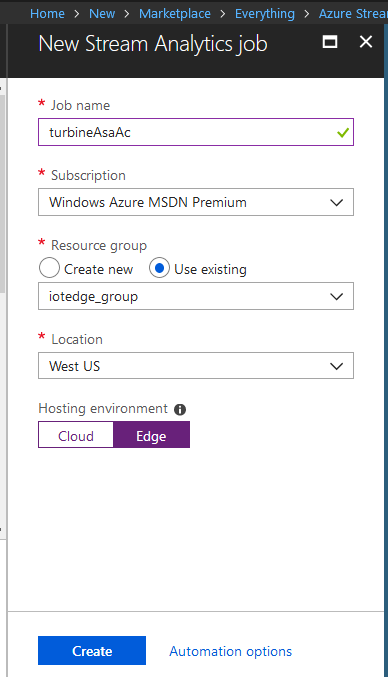
1. On the **New Stream Analytics job** page, under **Job name**, enter a name such as **turbineAsaXX**.

Use the same resource group and location you have been using throughout this lesson.

1. Under **Hosting environment**, select Edge.

Notice that you do not need to scope the throughput for Edge ASA jobs, like you do for cloud jobs, since they run on your hardware / IoT Edge device.

1. Select **Create**



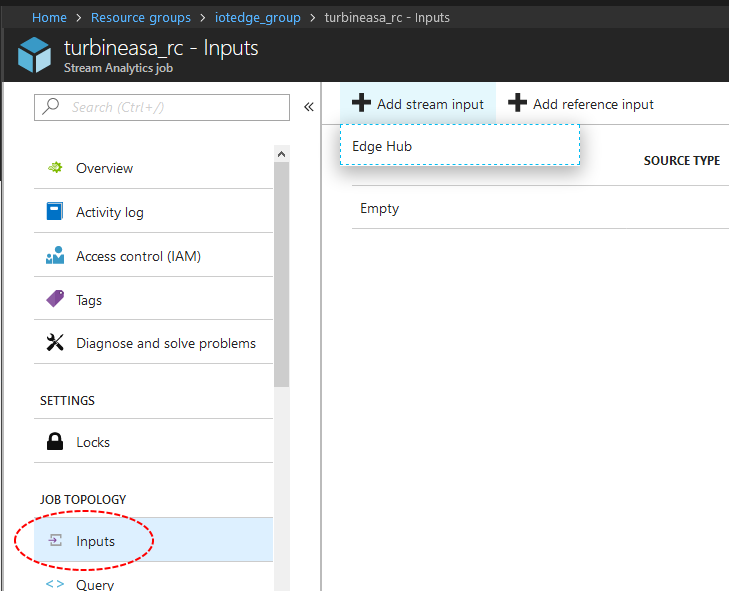
1. Navigate to the Azure Stream Analytics job.

Notice that this IoT Edge job looks exactly the same as the jobs you have created for the cloud.

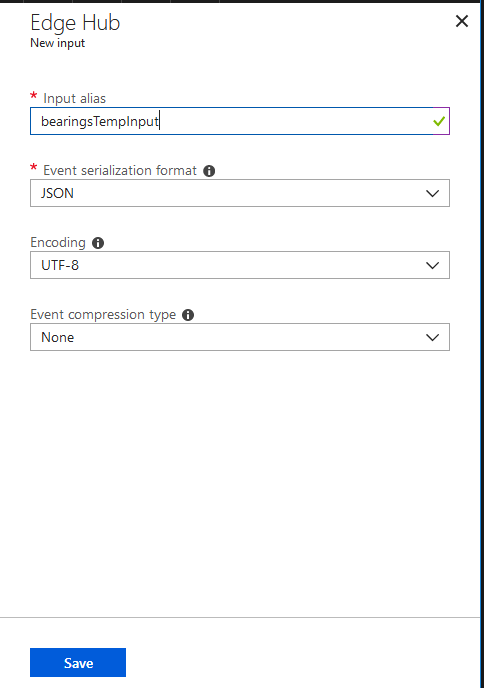
1. Select **Inputs**
2. Select **+Add stream input**

Since this is an IoT Edge job, the only option that will appear is **Edge Hub**

1. Select **Edge Hub**

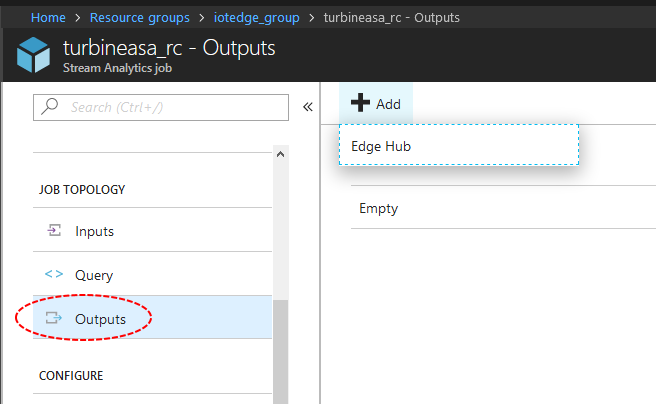


1. Under **Input alias**, enter **bearingsTempInput**
2. Select **Save**

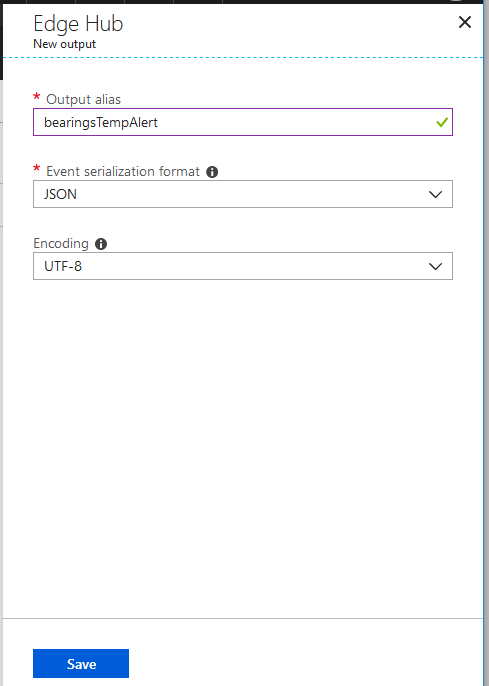


You can navigate directly to the outputs section by clicking **Outputs** in the **Job Topology** section.

1. Select **+Add**
2. Select **Edge Hub** from the list.



1. Under **Output alias**, enter **bearingsTempAlert**
2. Select **Save**



1. Navigate to the **Query** page.

The query that you will write will reference your input and output. Both the input and output for edge jobs are aliases. In the next lesson, you will fulfill those aliases with message routing. Replace the query with the following text:

SELECT

'reset' AS command

INTO

bearingsTempAlert

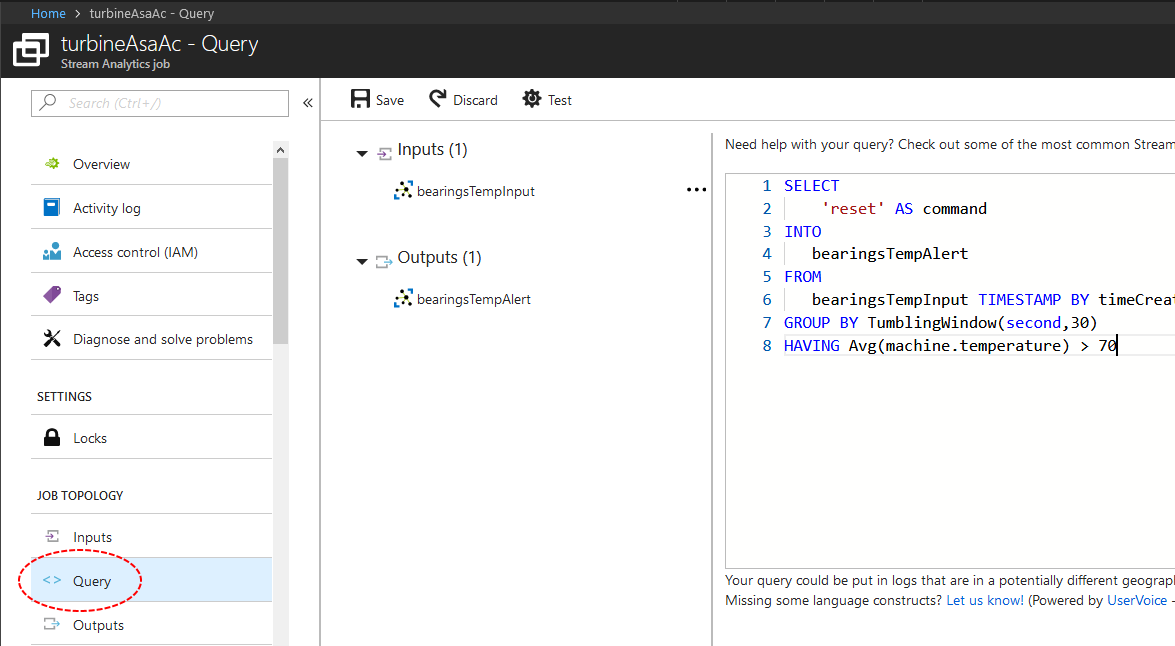
FROM

bearingsTempInput TIMESTAMP BY timeCreated

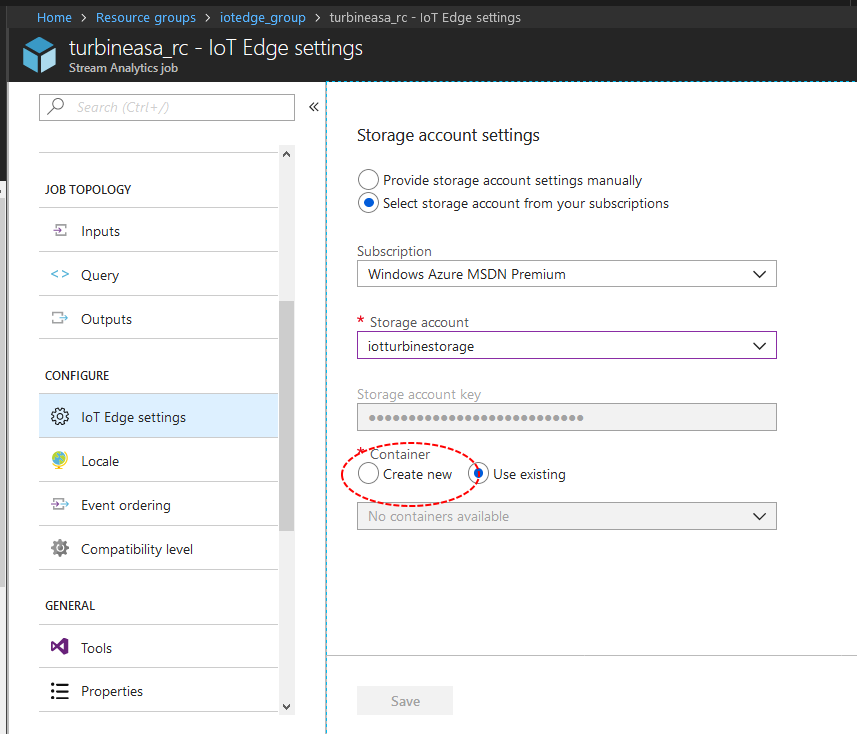
GROUP BY TumblingWindow(second,30)

HAVING Avg(machine.temperature) > 70

1. Select **Save**



1. Go to **IoT Edge settings** page.
2. Select the **Storage account** you set up earlier.
3. Under **Container**, select **Create new**
4. Name the container **bearingstempcontainer**
5. Select **Save**



**Summary**

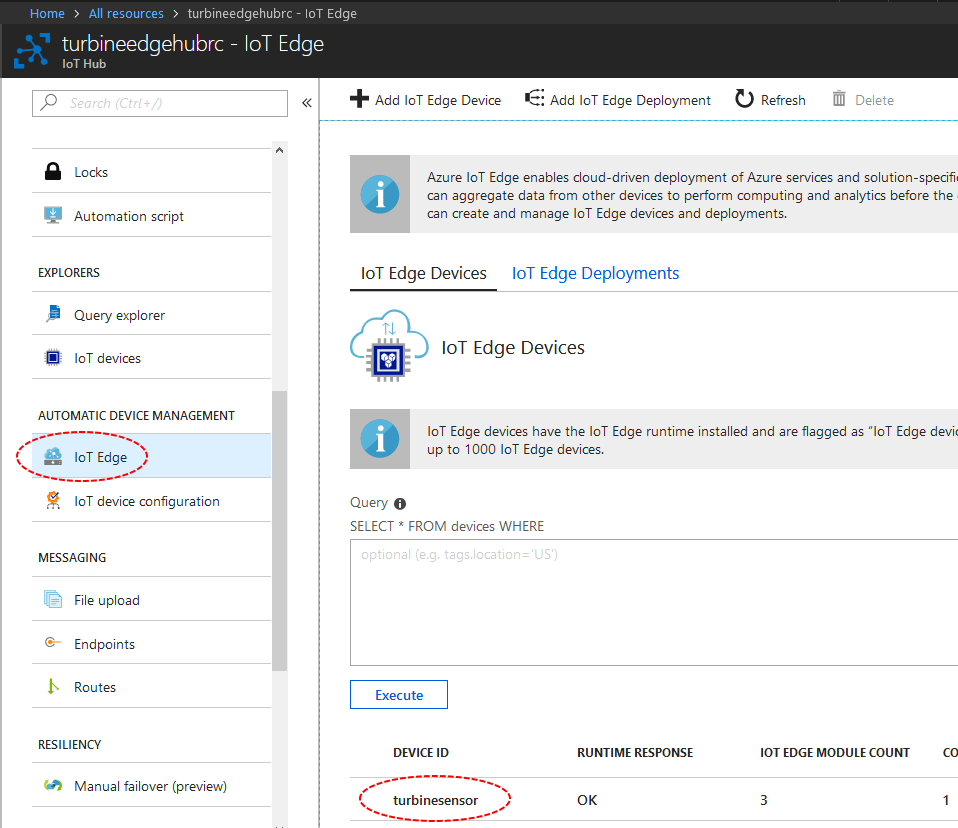
In this lesson, you set up a storage account to act as an endpoint for an Azure Stream Analytics job. You also created the stream analytics job to be deployed on Azure IoT Edge, set its inputs, outputs and query. You then linked the storage account to the analytics job.

**Deploy an Azure Streaming Analytics job for IoT Edge**

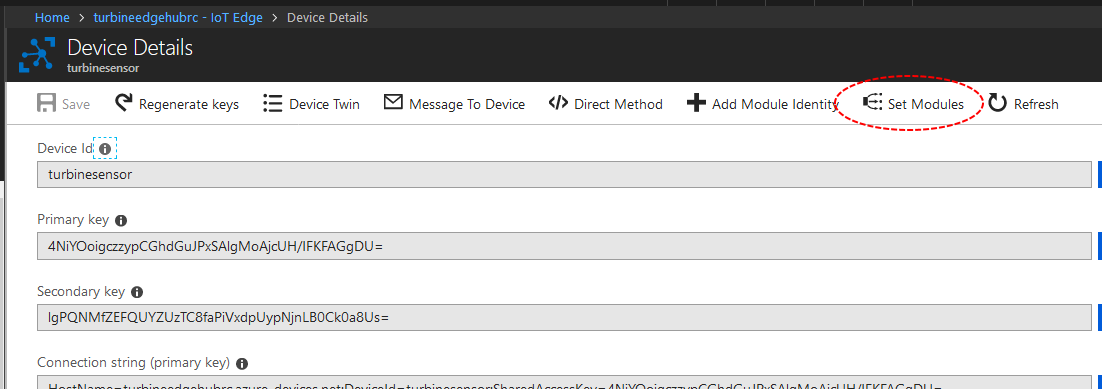
In this lesson, you will deploy your analytics job to your IoT Edge device. It will interact with the custom module you deployed in the first lesson. The two modules will communicate via routes that you set up in the IoT Hub.

The first module will simulate a bearings temperature sensor on a wind turbine whose temperature gradually increases. The Stream Analytics module will receive the message, and when the temperature becomes greater than 70 degrees, it will send a *reset* command to the bearings temperature sensor. That will set the temperature back to 20 degrees.

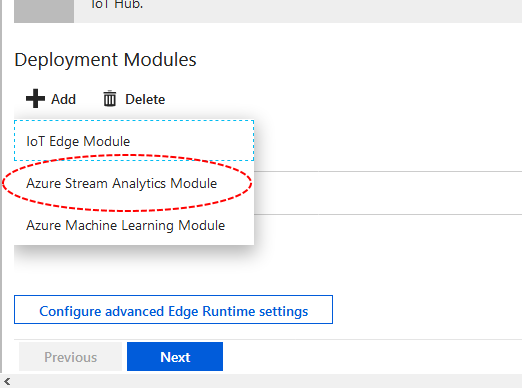
1. Navigate to the IoT Hub that you have been using for the past two lessons.
2. Select the **Iot Edge** blade.
3. Under **DEVICE ID**, select **turbinesensor**.



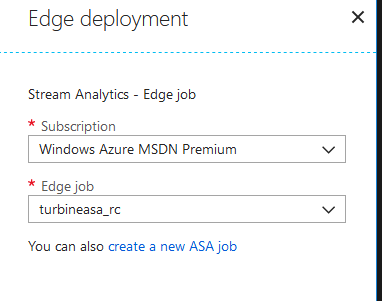
1. At the top of the **Device Details** page, select **Set Modules**.



1. Under **Deployment Modules**, select **+ Add**
2. From the list, select **Azure Stream Analytics Module**.

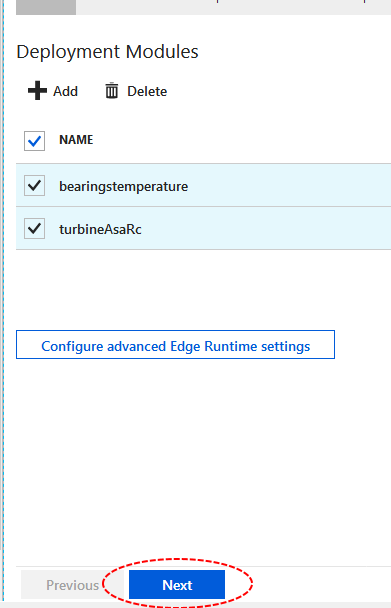


1. In the **Edge job** list, select the ASA job that you created in the last lesson.
2. Select **Save**



You will navigate back to the **Device Details** page.

1. In the **Deployment Modules** section, select **Next** to configure your routes.



1. In the **Specify Routes** box, replace the text with the following.

(Be certain to replace the module name in the sections that indicate {REPLACE WITH YOUR MODULE NAME}. For instance: **turbineAsaXX**).

{

"routes": {

"telemetryToCloud": "FROM /messages/modules/bearingstemperature/\* INTO $upstream",

"alertsToCloud": "FROM /messages/modules/{REPLACE WITH YOUR MODULE NAME}/\* INTO $upstream",

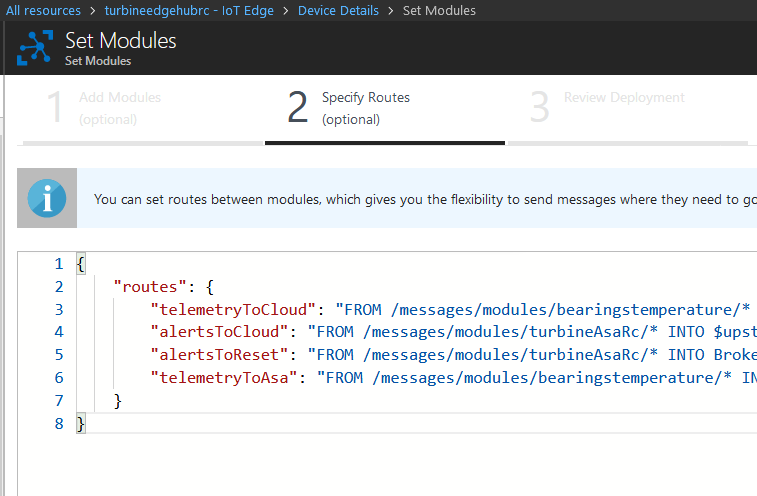
"alertsToReset": "FROM /messages/modules/{REPLACE WITH YOUR MODULE NAME}/\* INTO BrokeredEndpoint(\"/modules/bearingstemperature/inputs/control\")",

"telemetryToAsa": "FROM /messages/modules/bearingstemperature/\* INTO BrokeredEndpoint(\"/modules/{REPLACE WITH YOUR MODULE NAME}/inputs/bearingsTempInput\")"

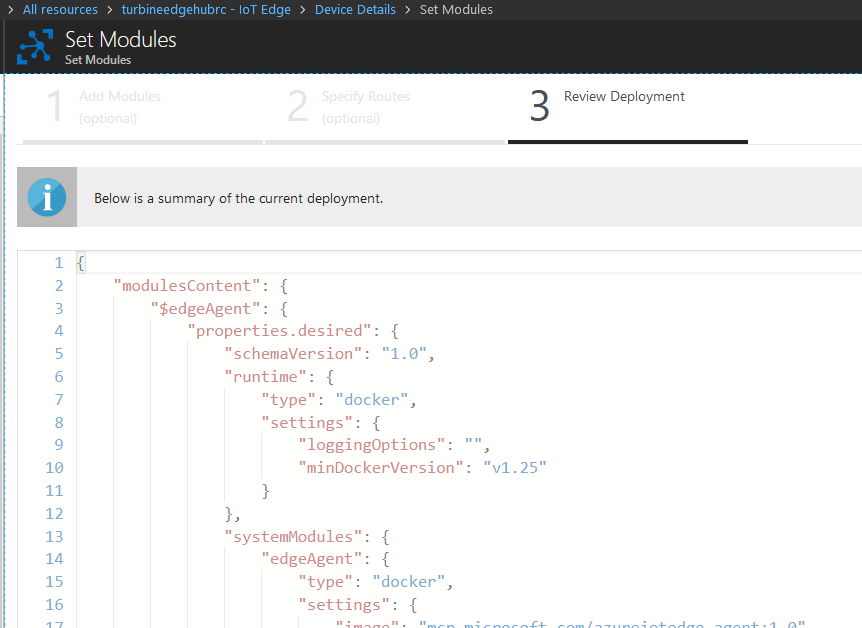
}

}

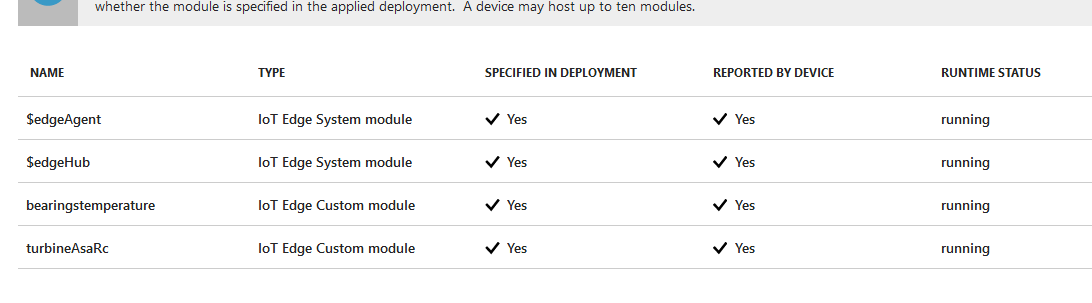
Notice that there are four separate routes specified here, based on the modules you deployed and the inputs and outputs you created. *telemetryToCloud* and *alertsToCloud* send messages to the *$upstream* variable, which represents your IoT Hub. *alertsToReset* and *telemetryToAsa* send messages between the two modules.



1. Select **Next** to **Review Deployment**
2. Submit the module configuration.



Your module will shows on the list of deployed modules. It will take a moment before its *Runtime status* shows as *running*.



1. Remote back onto your IoT Edge virtual machine.
2. On a command window, monitor events by entering the following:
3. iotedge logs bearingstemperature -f

Notice that when the temperature rises above 70, a message is sent from the ASA job to the bearings temperature module, which resets the temperature.

08/05/2018 16:16:32> Sending message: 655, Body: [{"machine":{"temperature":100.95502763006597,"pressure":10.108800616083466},"ambient":{"temperature":20.830930906502033,"humidity":26},"timeCreated":"2018-08-05T16:16:32.5514109Z"}]

08/05/2018 16:16:37> Sending message: 656, Body: [{"machine":{"temperature":100.6630404745987,"pressure":10.075536256599852},"ambient":{"temperature":20.701828793250876,"humidity":26},"timeCreated":"2018-08-05T16:16:37.5487233Z"}]

Received message Body: [[{"command":"reset"}]]

Resetting temperature sensor..

08/05/2018 16:16:42> Sending message: 657, Body: [{"machine":{"temperature":21.163029644830633,"pressure":1.0185729975123505},"ambient":{"temperature":21.251867014333545,"humidity":24},"timeCreated":"2018-08-05T16:16:42.5476109Z"}]

08/05/2018 16:16:47> Sending message: 658, Body: [{"machine":{"temperature":21.66830868235245,"pressure":1.0761364321667348},"ambient":{"temperature":21.252839506488684,"humidity":25},"timeCreated":"2018-08-05T16:16:47.5477222Z"}]

This is an example of setting up code to manage a device on the device itself – using the IoT Edge runtime, a custom module and an Azure Stream Analytics module.

**Summary**

In this lesson, you deployed the Azure Stream Analytics module that you created. You set up routes so that it could communicate with the custom module you deployed previously. Finally, you monitored the messages that the modules were sending, to examine the interaction.

**Introduction to developing custom IoT Edge modules**

**Introduction to developing custom IoT Edge modules**

In this lesson, you will learn about

* The different types of IoT edge modules
* The process of hosting and deploying IoT edge modules
* How to develop and deploy custom IoT edge modules

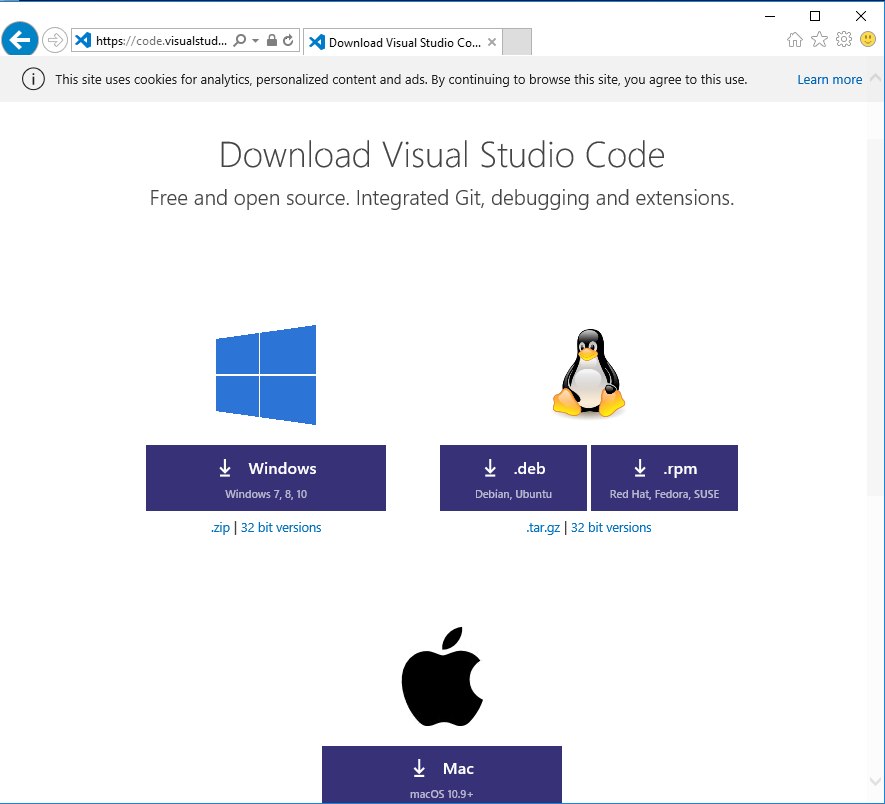
This lesson introduces you to the various types of modules that can run on an IoT Edge device. You will go through the entire lifecycle of creating a custom module, hosting it in a registry, building it on an IoT Edge device and running it.

**Intro and setup – Developing and deploying an IoT Edge Module**

In the following lessons, you will be using Visual Studio Code and its associated extensions to create an Azure IoT Edge module, send it to your own registry, then deploy it to your IoT Edge device.

You can use your own dev machine to complete this exercise, but there is a dependency on having Docker installed. Since you already have installed docker on the virtual machine that you have been using as an IoT Edge device, it’s recommended that you simply use your virtual machine for the following lessons. We’ll start by installing VS.NET Code

1. Use your favorite remote desktop client to log on to the virtual machine you created as your IoT Edge device for lesson 1.
2. Open the browser on your vm and navigate to <https://code.visualstudio.com/download>.
3. Choose the Windows installer.
4. Save the installer .exe file.
5. Launch the installer .exe file.

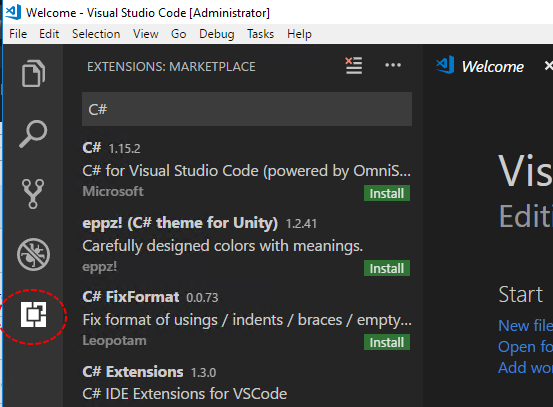


The installer for Visual Studio Code is pretty straightforward.

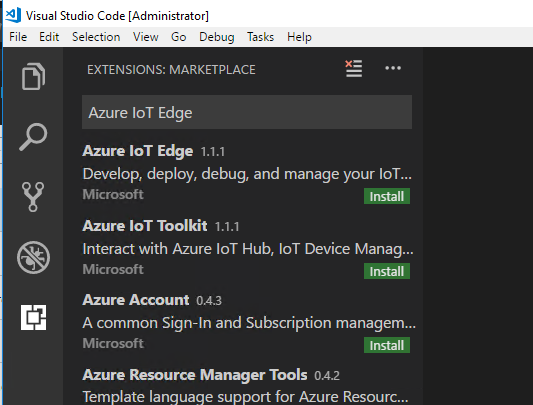
1. Accept the defaults for the onscreen prompts.
2. Allow **Visual Studio Code** to launch automatically.



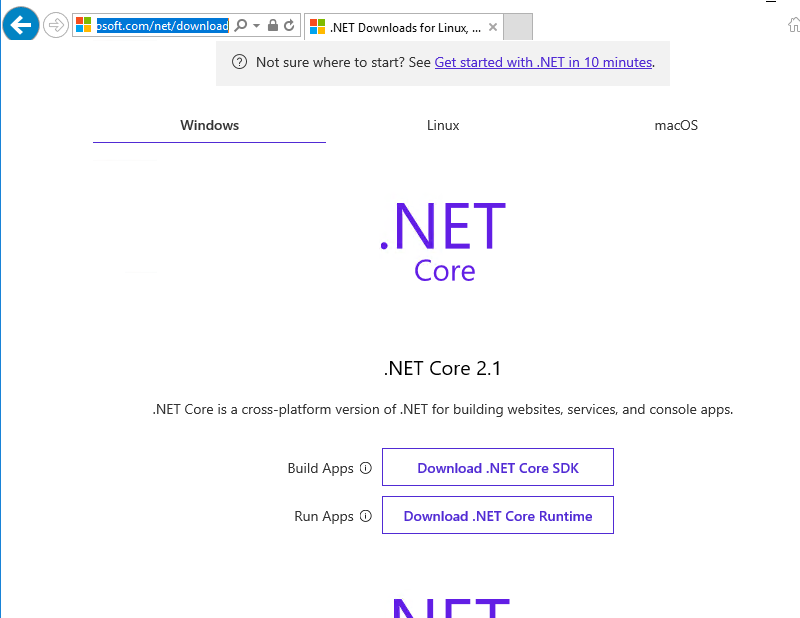
1. In the Visual Studio Code application, select the **Extensions** icon in the lower left corner.
2. Alternately, press the **Ctrl + Shift + X** key combination.
3. In the **Search Extension** box, type **C#**
4. Select the Microsoft C# extension and select **Install**



1. In the **Search Extension** box, type **Azure IoT Edge**
2. Select Microsoft’s **Azure IoT Edge** extension.
3. Click **Install**
4. Close Visual Studio Code



1. In your browser window, navigate to https://www.microsoft.com/net/download.
2. Download the latest version of the **.NET Core SDK**
3. Install it with default parameters.



**Summary**

In this lesson, you installed Visual Studio Code on to your Azure virtual machine, then installed the extensions required to develop IoT Edge modules. Finally, you installed the .NET Core SDK, which is also required for developing IoT Edge modules with Visual Studio Code.

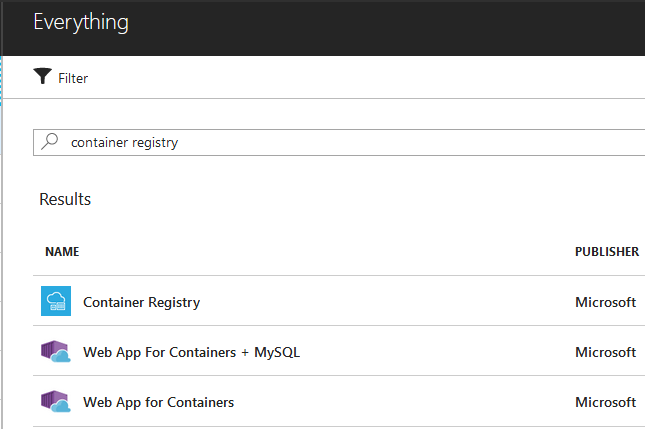
## Build Azure Function and package it as a custom IoT Edge Module

This lesson aims to give you deeper insight into the aspects of the Azure IoT Edge architecture. You will create your own registry for IoT edge modules, and you will use Visual Studio Code to create a custom module that leverages custom code.

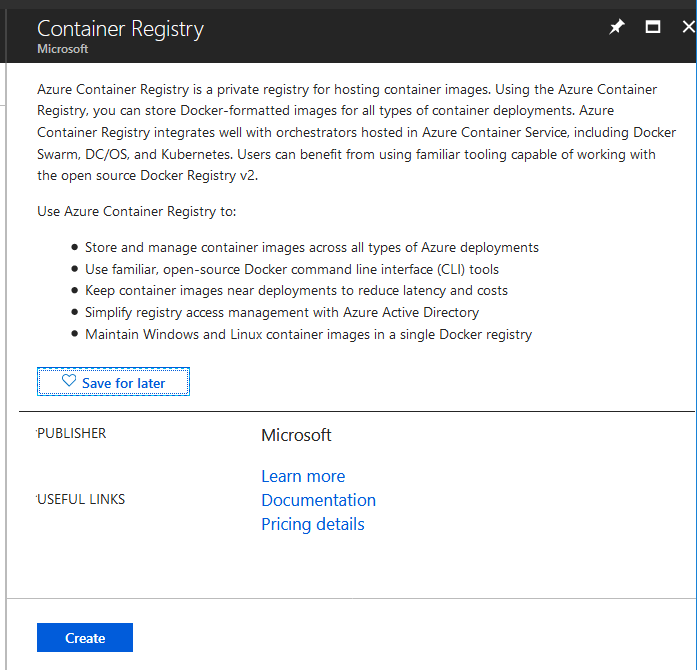
You have several programming language options for creating custom modules: C#, Python, Node.js, Java, C… and more are continually being added. For this module, you will use C#.

The module will raise an alert when a turbine sensor is above a certain threshold, similar to the module you created in the last lesson.

1. On your Azure Portal, go to **Create a resource**
2. In the search box, type **container registry**



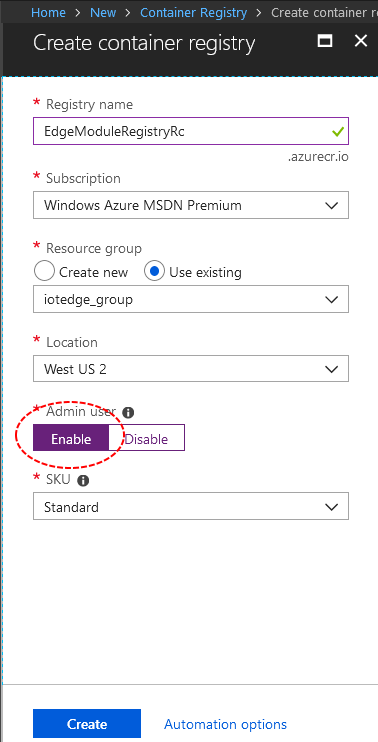
An Azure Container registry can not only host Azure IoT modules. It can also store nearly any type of container, such as Docker containers.



1. Under **Container Name**, give your container registry a unique name.

A good example would be **EdgeModuleRegistryXX**, where XX are your initials.

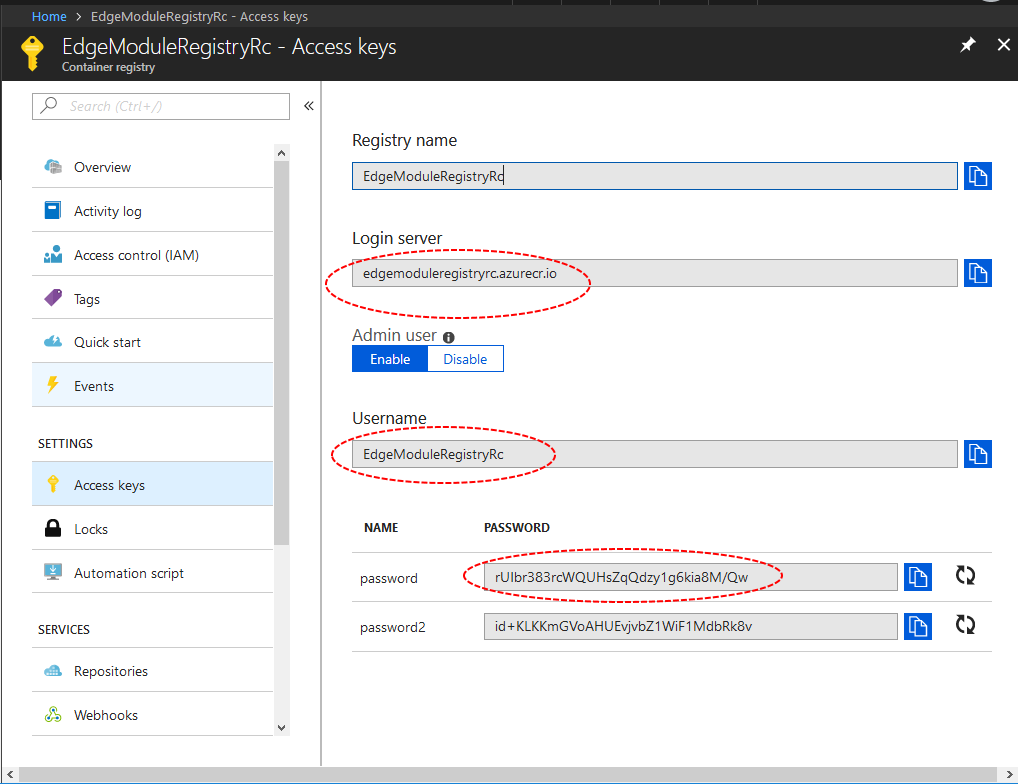
1. Use the same **Resource Group** you have been using for previous lessons.
2. Under **Location**, select the same region as your IoT Hub.
3. Under **Admin user**, select **Enable**
4. Select **Create**



Once the registry is provisioned, navigate to the overview blade.

1. Navigate to the **Access Keys** blade.
2. Copy the values for **Login Server**, **username**, and **password**

You will use the values later in the lesson.



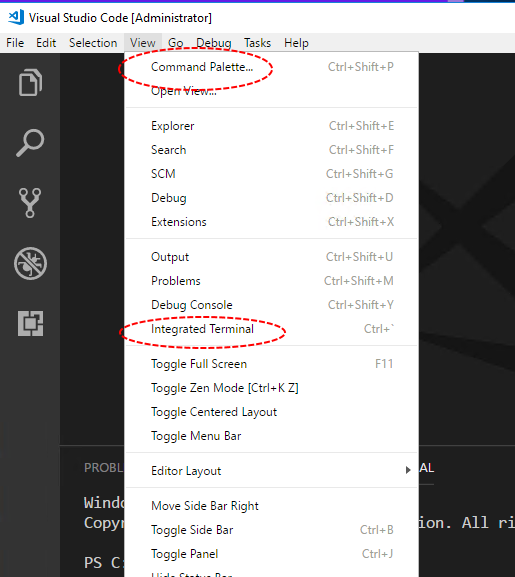
1. Open **Visual Studio Code**
2. Open the VS Code integrated terminal by selecting **View -> Integrated Terminal**

This is a PowerShell terminal, built directly into Visual Studio Code.

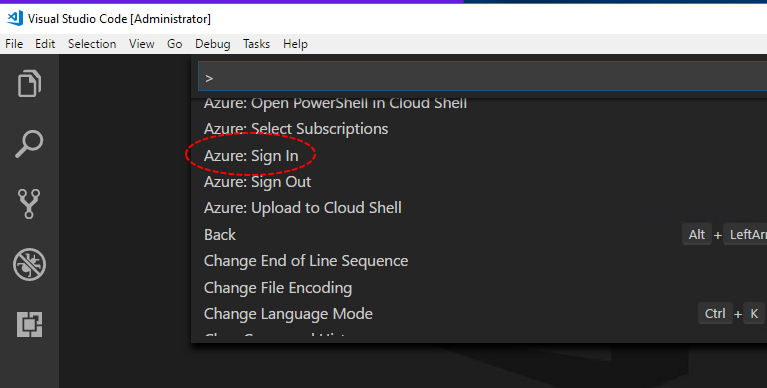
1. Open the Visual Studio Code command palette by selecting **View -> Command Palette**.

The Command Palette provides access to many commands.

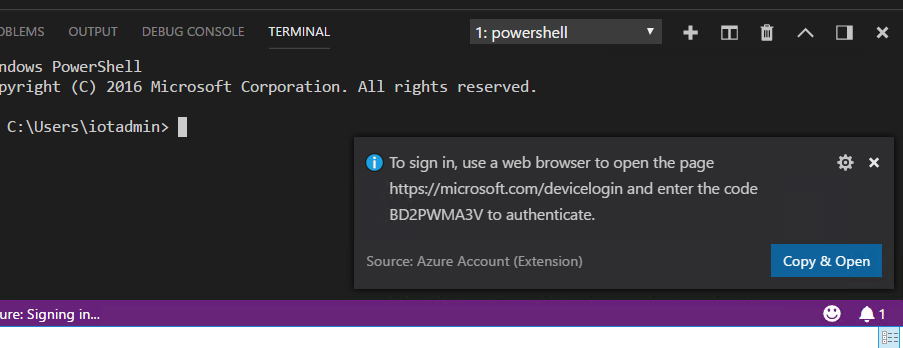
You can execute editor commands, open files, search for symbols, and see a quick outline of a file, all using the same interactive window.



1. Under the command palette, enter and run the command **Azure: Sign in**

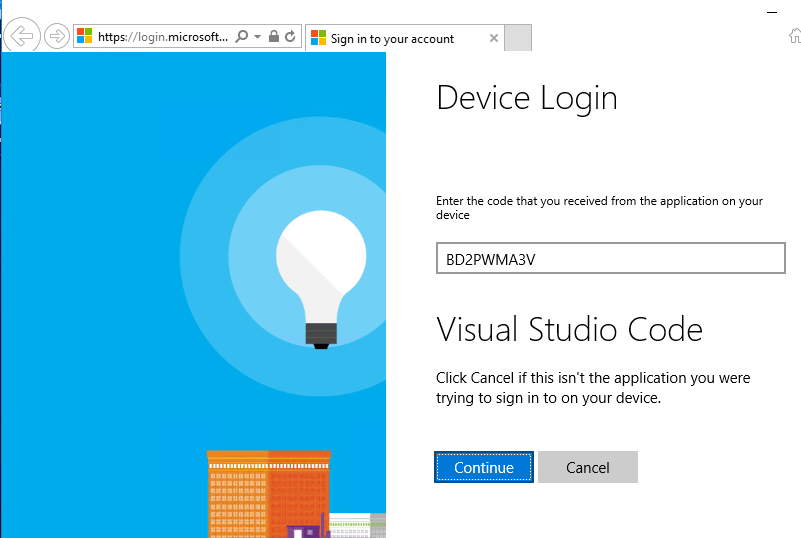


You’ll receive a device code, to allow your instance of Visual Studio Code to log in to your Azure account



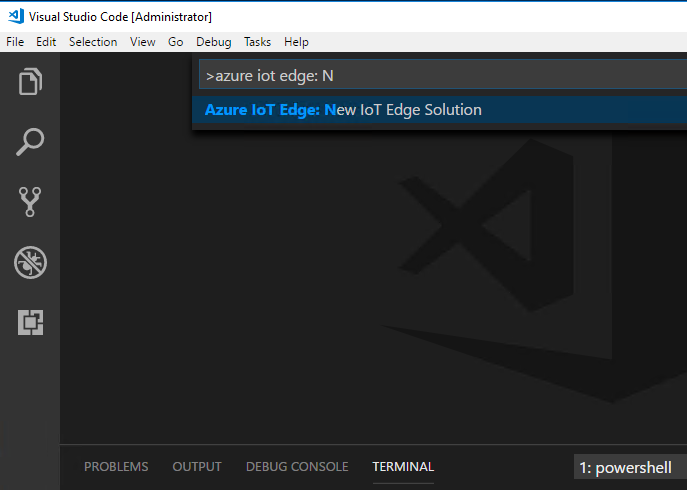
1. Paste your code Into the Device Login page’s box.
2. Click **Continue**.

You should get a success message. You are now signed in and have access to your Azure resources.



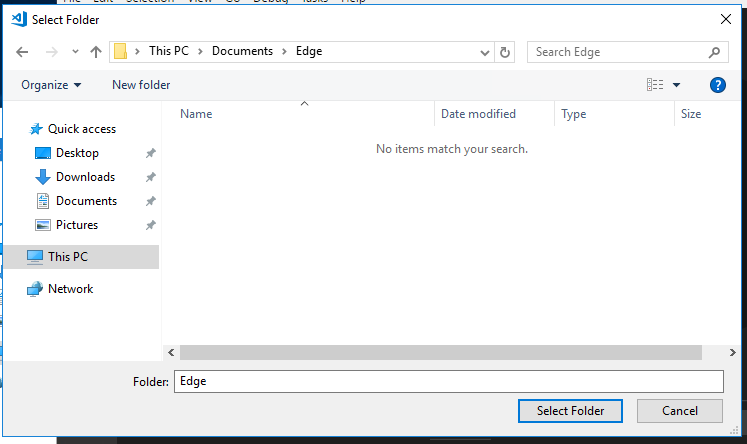
1. Under the command palette, enter and run the command **Azure IoT Edge: New IoT Edge Solution**.

This will start a series of steps that will end with a complete Edge module.

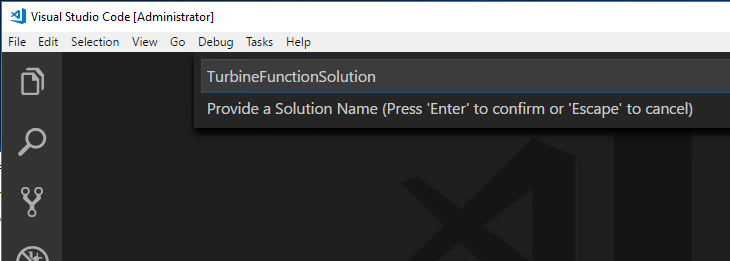


You will be asked for a folder for your solution.

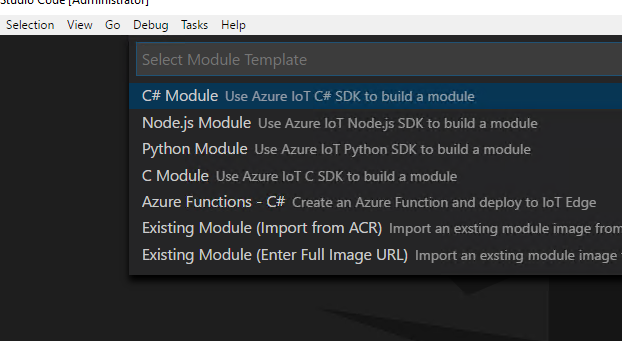
1. Choose a folder.



1. Provide a solution name: **TurbineFunctionSolution**

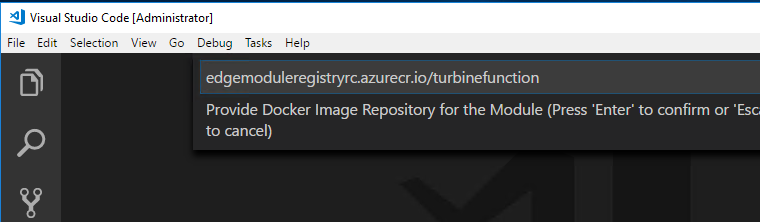


1. In the next box for **Select Module Template**, Choose **Azure Functions - C#** as the module template.



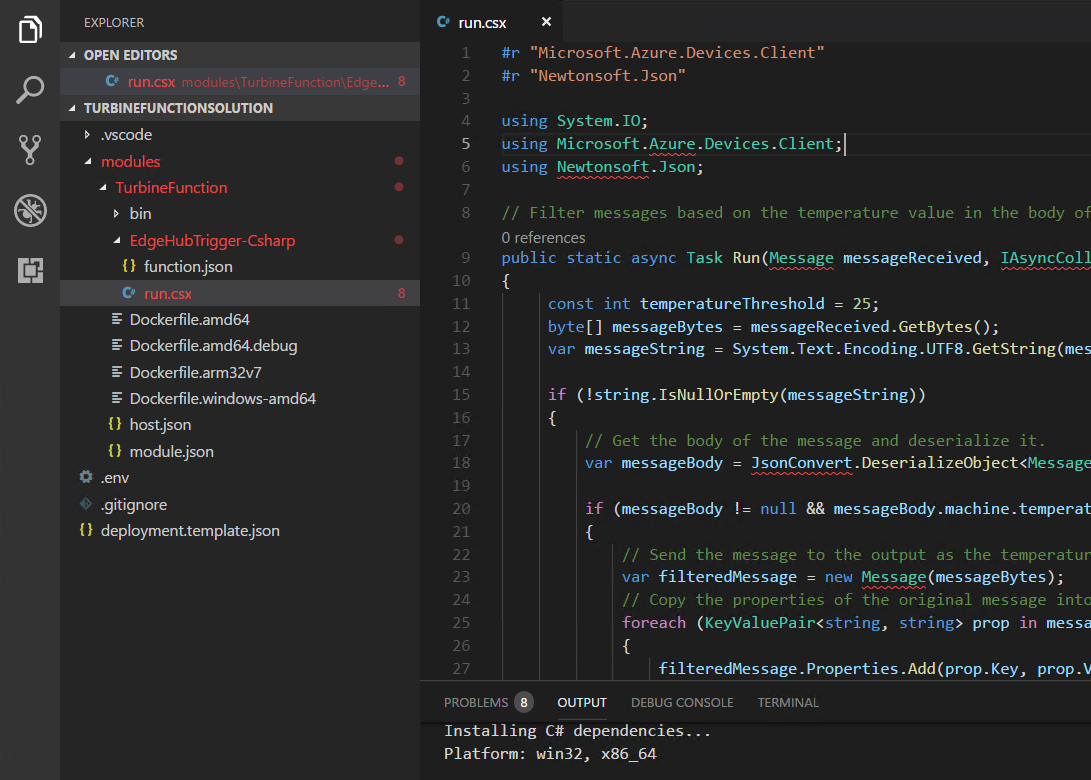
1. Name your module **TurbineFunction**
2. Specify the Azure container registry that you created in the previous section as the image repository for your first module.
3. Replace **localhost:5000** with the **login server** value that you copied.

The final string looks like **<registry name>.azurecr.io/turbinefunction**.



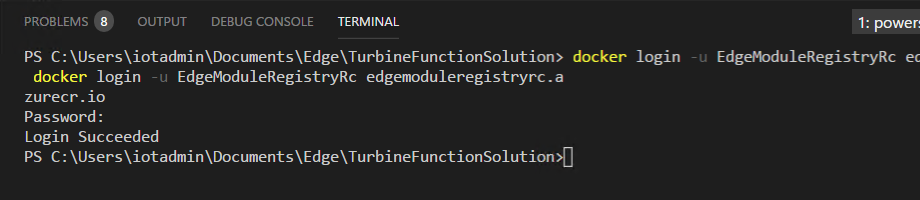
The VS Code window loads your IoT Edge solution workspace: a **.vscode** folder, a **modules** folder, a deployment manifest template file and a **.env** file.

1. In the VS Code explorer, open the file **modules > TurbineFunction > EdgeHubTrigger-Csharp > run.csx**
2. Replace the contents of the file with the following code:
3. #r "Microsoft.Azure.Devices.Client"
4. #r "Newtonsoft.Json"
5. using System.IO;
6. using Microsoft.Azure.Devices.Client;
7. using Newtonsoft.Json;
8. // Filter messages based on the temperature value in the body of the message and the temperature threshold value.
9. public static async Task Run(Message messageReceived, IAsyncCollector<Message> output, TraceWriter log)
10. {
11. const int temperatureThreshold = 25;
12. byte[] messageBytes = messageReceived.GetBytes();
13. var messageString = System.Text.Encoding.UTF8.GetString(messageBytes);
14. if (!string.IsNullOrEmpty(messageString))
15. {
16. // Get the body of the message and deserialize it.
17. var messageBody = JsonConvert.DeserializeObject<MessageBody>(messageString);
18. if (messageBody != null && messageBody.machine.temperature > temperatureThreshold)
19. {
20. // Send the message to the output as the temperature value is greater than the threashold.
21. var filteredMessage = new Message(messageBytes);
22. // Copy the properties of the original message into the new Message object.
23. foreach (KeyValuePair<string, string> prop in messageReceived.Properties)
24. {
25. filteredMessage.Properties.Add(prop.Key, prop.Value); }
26. // Add a new property to the message to indicate it is an alert.
27. filteredMessage.Properties.Add("MessageType", "Alert");
28. // Send the message.
29. await output.AddAsync(filteredMessage);
30. log.Info("Received and transferred a message with temperature above the threshold");
31. }
32. }
33. }
34. //Define the expected schema for the body of incoming messages.
35. class MessageBody
36. {
37. public Machine machine {get; set;}
38. public Ambient ambient {get; set;}
39. public string timeCreated {get; set;}
40. }
41. class Machine
42. {
43. public double temperature {get; set;}
44. public double pressure {get; set;}
45. }
46. class Ambient
47. {
48. public double temperature {get; set;}
49. public int humidity {get; set;}
50. }
51. Save the file.



1. Sign in to Docker by entering the following command in the Visual Studio Code integrated terminal.
2. docker login -u <ACR username> <ACR login server>

Then you can push your module image to your Azure container registry:



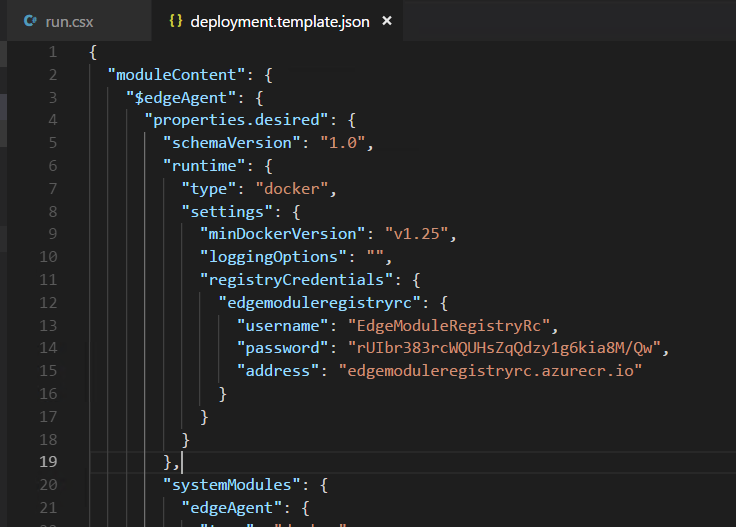
1. In the VS Code explorer, open the **deployment.template.json** file in your IoT Edge solution workspace.

This file tells the IoT Edge runtime which modules to deploy to a device.

1. Locate the **registryCredentials** section in the deployment manifest.
2. Update the **username**, **password**, and **address** with the credentials from your container registry.

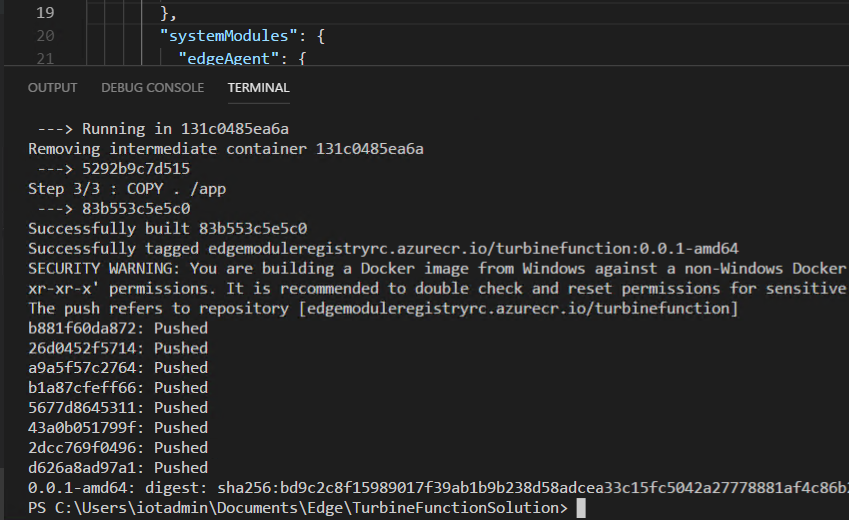
This section gives the IoT Edge runtime on your device permission to pull the container images that you store in your private registry. The actual username and password pairs are stored in the .env file, which is ignored by source control providers.

1. Save this file.

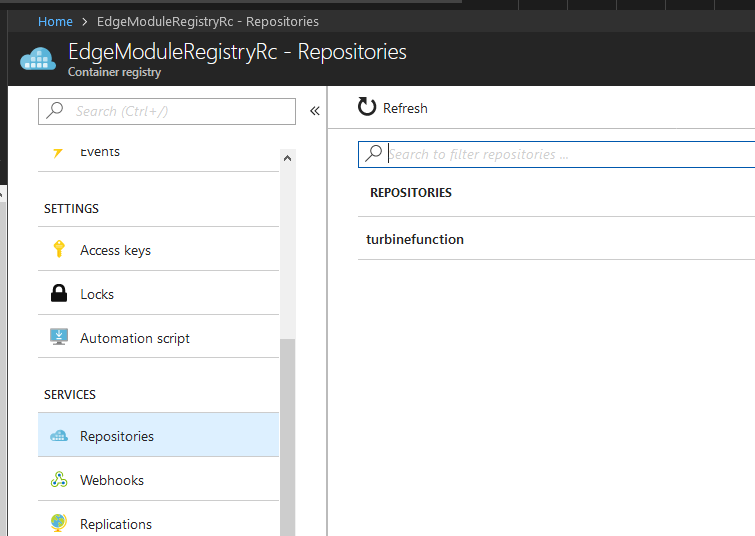


1. In the VS Code explorer, right-click the **deployment.template.json** file and select **Build IoT Edge solution**.

When you tell Visual Studio Code to build your solution, it first takes the information in the deployment template and generates a deployment.json file in a new folder named config. Then it runs two commands in the integrated terminal: docker build and docker push. These two commands build your code, containerize the functions, and then push the code to the container registry that you specified when you initialized the solution.

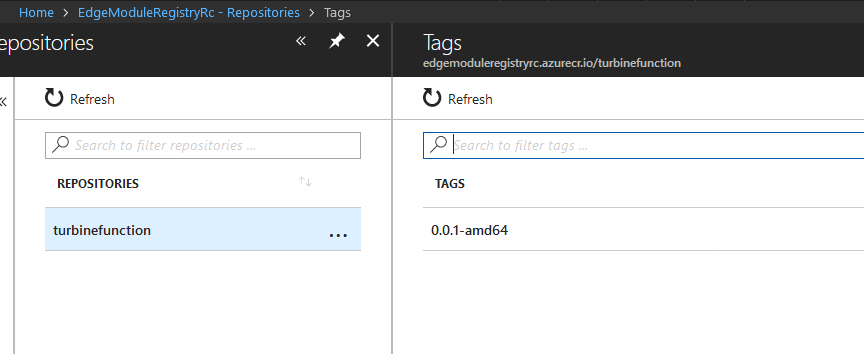


1. In the Azure portal, browse to your Azure container registry.
2. Select **Repositories**



You should see the **turbinefunction** repository in the list.

1. Select this repository to see more details.

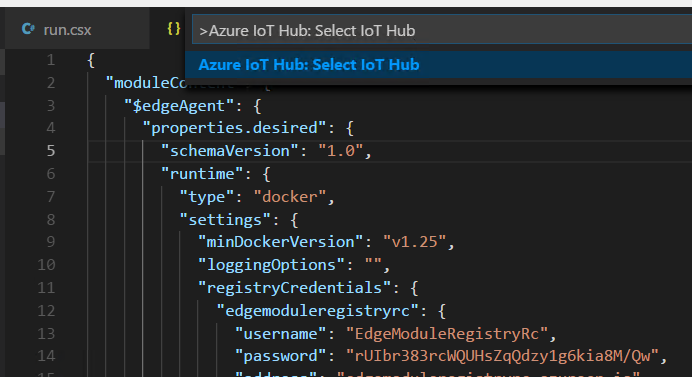


In the Tags section, you should see the **0.0.1-amd64** tag. This tag indicates the version and platform of the image that you built. These values are set in the module.json file in the TurbineFunction folder.

## Deploy IoT Edge module

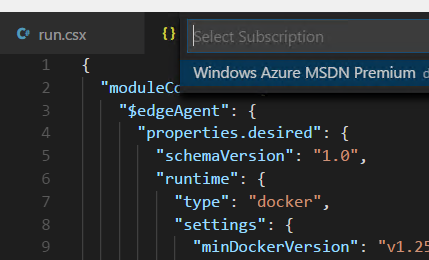
In this lesson, you will deploy the IoT Edge module you just built. We will try a different method of deployment – using the Azure IoT Edge plugin from Visual Studio Code.

1. Under the command palette, search for and run the command **Azure IoT Hub: Select IoT Hub**

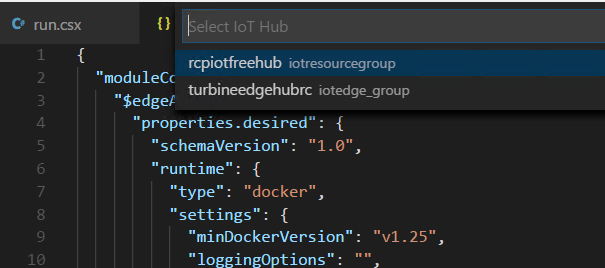


1. Select the subscription that has your IoT hub.
2. Select the IoT hub that you want to access.

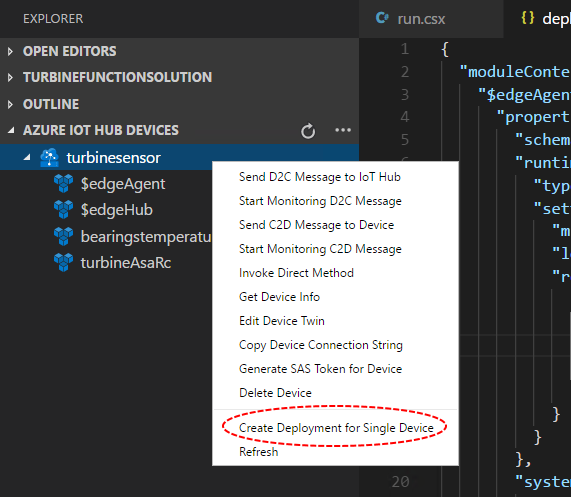
(Since you logged Visual Studio into your identity earlier, it will have access to your resources).



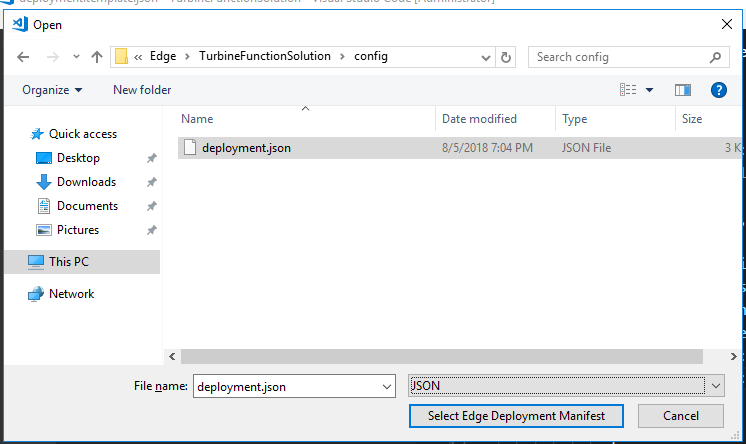
1. Select the IoT Hub from your subscription.



1. In the VS Code explorer, expand the **Azure IoT Hub Devices** section.
2. Right-click the name of your IoT Edge device.
3. Select **Create Deployment for Single device**

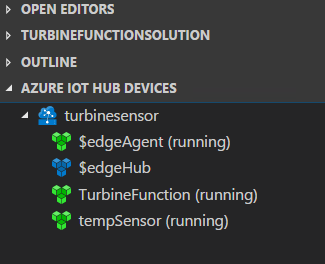


1. Browse to the solution folder that contains the **TurbineFunction**
2. Open the config folder.
3. Select the **deployment.json** file.
4. Click **Select Edge Deployment Manifest**



1. Refresh the **Azure IoT Hub Devices** section.

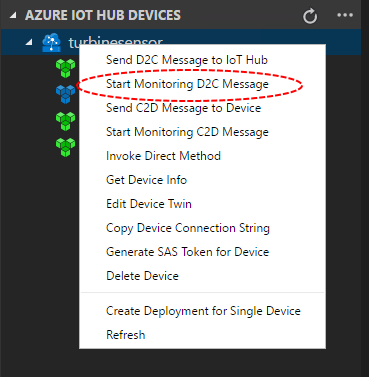
You should see the new TurbineFunction running along with the tempSensor module, and the $edgeAgent and $edgeHub.



You can see all of the messages that arrive at your IoT hub by running **Azure IoT Hub: Start Monitoring D2C Message** in the command palette. (D2C = Device to Cloud)

You can also filter the view to see all of the messages that arrive at your IoT hub from a specific device.

1. Right-click the device under **Azure IoT Hub Devices**
2. Click **Start Monitoring D2C Messages**



1. To stop monitoring messages, run the command **Azure IoT Hub: Stop monitoring D2C messages** in the command palette.

### Summary

In this module, you used Visual Studio Code’s IoT Edge tools to deploy and monitor your custom edge module.

### Summary

In this lesson, you created a module repository, built a custom IoT Edge module and uploaded it to the repository.