Microsoft Azure SQL Edge with Lenovo ThinkSystem SE350 Solution Guide

Last update: **01 December 2021**

Version 1.1

Highlights the benefits of ThinkSystem SE350 Edge Server

Presents a use case for SQL Server on Lenovo Edge devices

Provides deployment information and best practices

Includes list of all software package download URLs

David West

Vinay Kulkarni

LENOVO PRESS

Table of Contents

1 Introduction

The Internet of Things (IoT) is the name coined for the billions of physical devices around the world that are connected to the internet, all collecting and sharing data. With cheap computer chips and the spread of wireless networks it's possible to turn any small or large thing into an IoT device. By adding sensors to these objects and connecting them to the internet converts them into intelligence devices. Data generated by these edge devices can be collected and mined to make customer lives better or improve profits for businesses. Data generated needed to be uploaded to the public cloud for processing which compromised security and increased latency. Azure SQL Edge is a very small footprint, robust IoT database product from Microsoft that can run on these edge devices to process the data locally for increased security and reduced latency.

The combination of Microsoft Azure SQL Edge and Lenovo’s innovative ThinkSystem SE350 Edge Server provides the ideal edge gateway solution for processing data aggregated from scores of IoT devices. Azure SQL Edge is a fully containerized solution compatible with most docker container engines. It can run on any Intel, AMD, or ARM64 device and includes native support for data streaming (the same engine that powers Azure Stream Analytics). It has many security features like data encryption, classification, and access controls. Azure SQL Edge also has Time Series Processing and ML inferencing capabilities. ML Models can be trained on premises and deployed to the edge where inferencing can be done on data that is being collected.

This white paper walks you through the steps required to set up and run Azure SQL Edge on the SE350. It also provides some performance data with Azure SQL Edge running on the SE350.

This solution can also be implemented on ThinkAgile MX1020 Integrated system or ThinkAgile MX1021 certified nodes for high availability.

2 Business value

Microsoft Azure SQL Edge extends the performance and security of the very popular Microsoft SQL Server engine to the intelligent edge. It is optimized for IoT gateways and devices, is available in an easy to deploy modern container format and has a very small footprint of around 500MB. Azure SQL Edge provides the RDBMS features of a full version of Microsoft SQL Server along with real-time insights, built-in streaming, storage, and support for AI. Azure SQL Edge helps customers with their application modernization journey so they can develop their application once and deploy anywhere across the edge, their datacenter and Azure.

Azure SQL Edge has simplified pricing that is well suited for IoT deployments. Azure SQL Edge follows a per device pricing model as listed below in table number 1. Purchasing one unit of Azure SQL Edge provides usage rights to run Azure SQL Edge on one device.

Product	Pay as you go	1 Year Reserved	3 Year Reserved
Azure SQL Edge	$10/device/month	$100/device/year	$60/device/year

Table 1. Azure SQL Edge pricing

Billing starts when an Azure SQL Edge is deployed to devices, irrespective of whether the SQL process is running/failed/stopped.

3 Architectural overview

Azure SQL Edge is a compact, containerized version of SQL Server for Linux that is designed to run on smaller edge devices at remote locations. The system can run disconnected from Azure or a data center and then reconnect later to sync data.

Ideal use cases include using SQL Edge as a local database for IoT devices in retail or similar environments.

With the Azure connected model, an Azure IoT Hub is used to connect to an Azure defined IoT Edge device. Then the Azure SQL Edge module, which is a container, is deployed to the edge device.

The following diagram, complements of Microsoft, provides a high-level view of it.

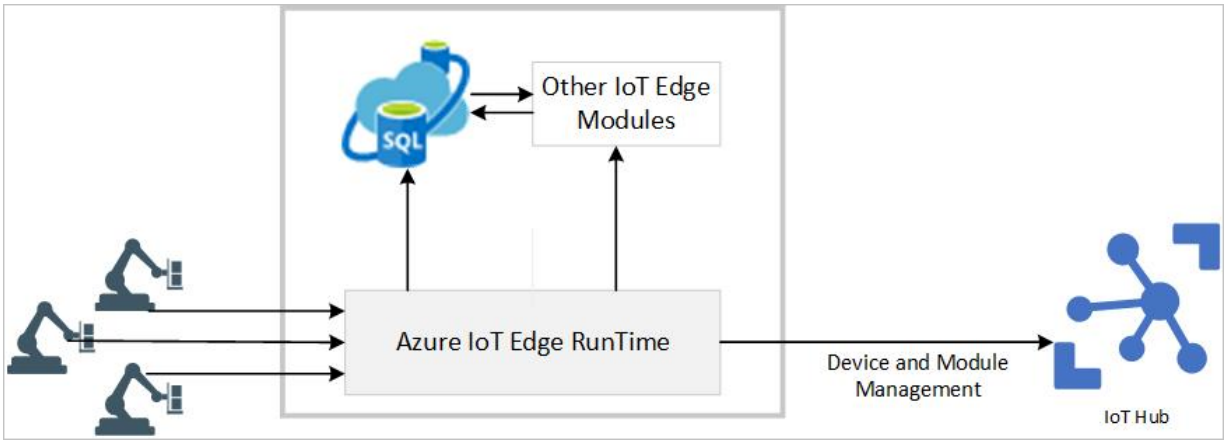

Figure 1 Overview of Azure SQL Edge architecture. Drawing credit, Microsoft

More information on the SQL Edge architecture can be found here:
https://docs.microsoft.com/en-us/azure/azure-sql-edge/overview

4 Deployment prerequisites

There are several concepts and pre-requisites to be aware of before deploying the solution. There are two deployment options. The more complex one is the Azure connected method, while the disconnected Docker container approach is rather straight forward.

For the connected option there is a requirement for an IoT Hub in Azure, along with a corresponding connected IoT Edge device which is where the SQL Edge container is deployed to. The IoT Edge device is a Linux system, which can be any size system or even a virtual machine running on an edge system.

Microsoft's documentation and packages are based on Ubuntu or Raspberry PI platforms. These 2 are the only Tier 1 supported platforms for the IoT Edge device (see the link below). There are several Tier 2 systems which should work fine but are not fully tested, supported, or as well documented by Microsoft. Lenovo's preference for this project was to use Red Hat Enterprise Linux (RHEL) for the IoT Edge device OS and the only Red Hat version on the Tier 2 list is RHEL 7.x. As a result, the deployment tested and detailed in this guide is based on RHEL 7.9 with specific RPM package links and dependency information for this OS. Setting this up on a Tier 2 platform requires more effort to find the right combination of supported packages and dependencies, which we have already worked through for you and provide in the sections below.

More information on the supported platforms for the IoT Edge device is available at:
https://docs.microsoft.com/en-us/azure/iot-edge/support?view=iotedge-2020-11#operating-systems

Specific package pre-requisites are covered in each component section that follows. Since most of the packages are not included in the usual Red Hat repositories, this guide includes the URLs to download the RPMs via wget and the commands to run the installations.

The deployment steps below assume access to an Azure account and an edge device has been provisioned running RHEL 7.x, with internet connectivity.

5 IoT Edge device setup

The IoT Edge devices are based on the Moby container engine, which is essentially open-source Docker. This is the container engine Microsoft requires. The packages below are the most recent at the time of writing. As time goes by, these may need updating, by looking in the same packages.microsoft location.

For each of the packages below, use the wget command to download the RPM packages.

Example: Sudo wget https://url_to_package.rpm

5.1 Create an IoT Hub in Azure

Follow the steps in this link to setup an IoT Hub in Azure, it is a simple wizard driven setup. The hub will be used later to register, connect, and manage the IoT Edge device.

https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-create-through-portal

5.2 Install Moby Engine prerequisites on edge device

There are 3 packages that must be installed as Moby engine dependencies. They must be installed **in the order provided below** to satisfy the dependencies. Use sudo wget <url> to download these.

Download links for the packages:

1. Container-selinux
   http://mirror.centos.org/centos/7/extras/x86_64/Packages/container-selinux-2.119.2-1.911c772.el7_8.noarch.rpm
2. Moby-runc
   https://packages.microsoft.com/centos/7/prod/moby-runc-1.0.0~rc95%2Bazure-1.x86_64.rpm
3. Moby-containerd
   https://packages.microsoft.com/centos/7/prod/moby-containerd-1.4.8%2Bazure-1.el7.x86_64.rpm

Install each of these one at a time, in order, as there is a y/n prompt on each one. Change to the directory where the packages were downloaded and install each one with the following yum command.

Sudo yum install <package name.rpm>

5.3 Install Moby Engine

After the 3 dependency packages are installed, proceed with downloading and installing the Moby container CLI and engine. These are two separate packages to install.

Download links for Moby Engine and CLI:

Moby-cli
https://packages.microsoft.com/centos/7/prod/moby-cli-20.10.7%2Bazure-1.x86_64.rpm

Moby-engine
https://packages.microsoft.com/centos/7/prod/moby-engine-20.10.7%2Bazure-1.el7.x86_64.rpm

Install each of these, one at a time, with the following command.

Sudo yum install <package name.rpm>

5.4 Install the IoT Edge components

After the Moby engine is installed, the two IoT Edge components can be installed. These must be installed **in the order listed below**, to meet dependencies.

Download links for the two IoT Edge files:

https://github.com/Azure/azure-iotedge/releases/download/1.1.4/libiothsm-std_1.1.4-1.el7.x86_64.rpm

https://github.com/Azure/azure-iotedge/releases/download/1.1.4/iotedge-1.1.4-1.el7.x86_64.rpm

Install each of these, one at a time, with the following command.

Note: These use the rpm command below, not the yum installer as previously used.

Sudo rpm -Uhv <package name.rpm>

5.5 Register the device with Azure IoTHub

The Edge device needs to be registered with an IoTHub to enable it as a connected Edge device. Follow the steps in the below link to register the device. For test or proof of concept configurations, the symmetrical key method is the simplest approach.

https://docs.microsoft.com/en-us/azure/iot-edge/how-to-register-device?view=iotedge-2020-11&tabs=azure-portal

After the device is registered, open the object and copy the connection strings. These are used to configure the IoTEdge connection in the next step.

5.6 Configure IoT Hub connection

The IoT Edge device needs connection strings configured to allow it to connect with the IoT Hub. Open the configuration file located at:

/etc/iotedge/config.yaml

Using vi or a similar editor, find the below section and add the connection string where indicated, then save the file. Note that this is a read-only file, so to save it in vi editor, use :wq! or the equivalent if using another editor.

Manual provisioning with an IoT Hub connection string (SharedAccessKey authentication only)
provisioning:

source: "manual"
device_connection_string: "<**ADD DEVICE CONNECTION STRING HERE**>"
dynamic_reprovisioning: false

Complete the remaining steps below to apply changes and verify the install.

1. To apply the new configuration, restart the IoTEdge service with the following command.

   Sudo systemctl restart iotedge

2. Use this command to verify that the service is running.

   Sudo systemctl status iotedge, shows the status of the service

It should show a status of active (running)

View the IoT Edge device in the Azure portal within the IoT Hub device. It should show running, however until the SQL Edge module is deployed there may be some errors showing. This is normal, proceed with the SQL Edge deployment below, and afterwards it will show connected and online.

6 Azure SQL Edge - Azure IoT Edge method

Now that the IoT Edge device is configured and connected to the Azure portal, the next step is to deploy SQL Edge to it from the Azure Marketplace. The Marketplace is an online listing of applications and services that are designed and optimized for Azure. Follow the steps below to deploy the SQL Edge container module.

1. Search within Azure Marketplace for the Azure SQL Edge offering.

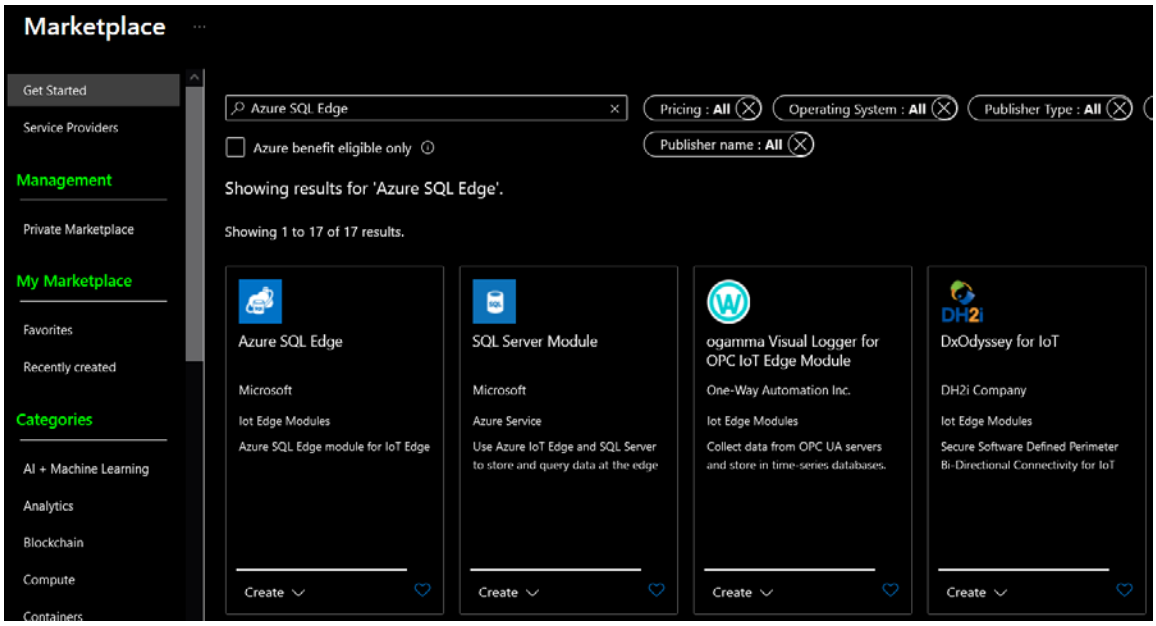

Figure 2 Azure Marketplace

2. From the list on the Azure SQL Edge page, choose developer or standard, and click Create

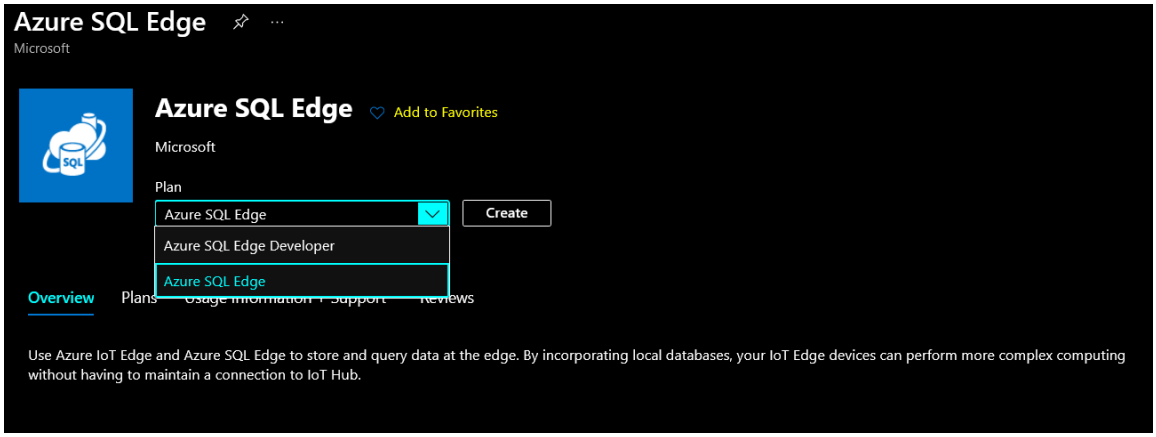

Figure 3 Create the SQL Edge module

3. On the Target Devices page, provide the following information where prompted:
    a. Azure subscription number
    b. IoT Hub name
    c. IoT Edge device name

Then click Create, at the bottom of the page.

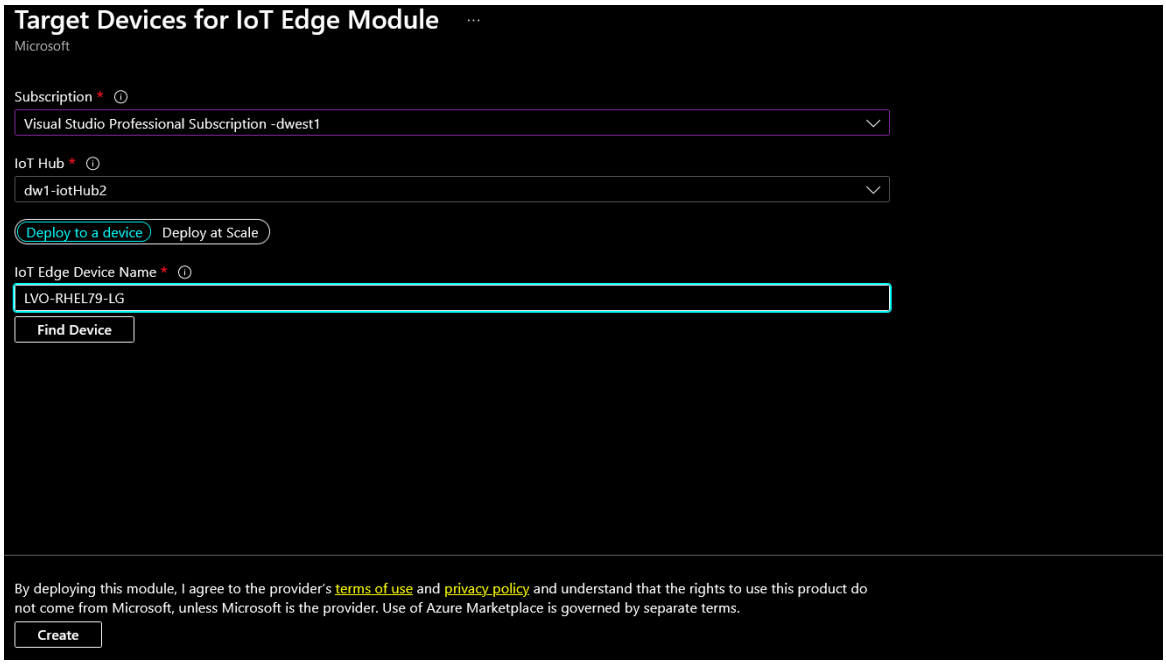

Figure 4 Specify the target device

4. On the Set Modules page, click on the Azure SQL Edge module. The default name is set to AzureSQLEdge, which is fine, but it can also be changed here.

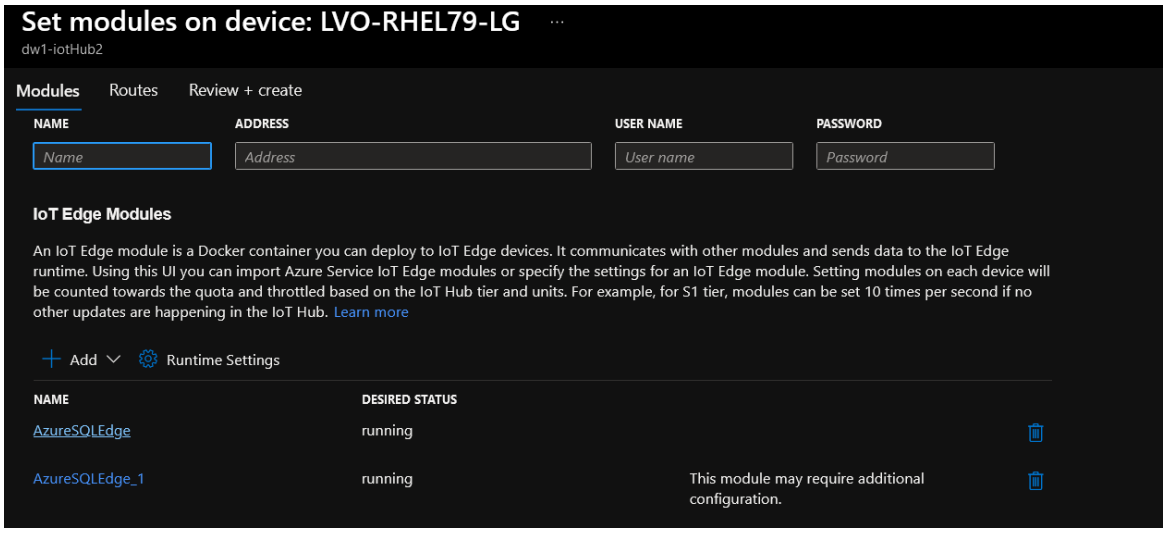

Figure 5 Select module to deploy

5. This opens the Module Settings tab of the Update IoT Edge Module page. Set the values for Module Name, Restart Policy and Desired Status per your requirements.

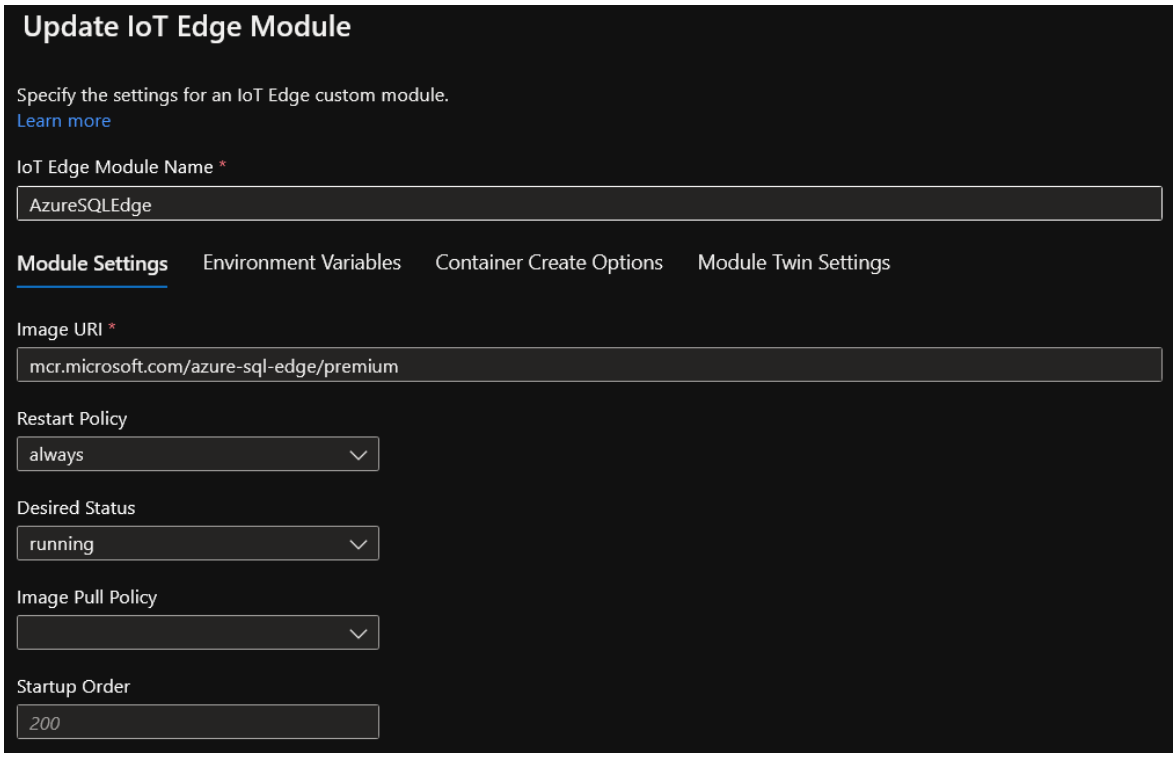

Figure 6 Set module parameters and policy

6. On the Environment Variables section, specify the SQL SA password, language code (leave default 1033 for English) and collation options.

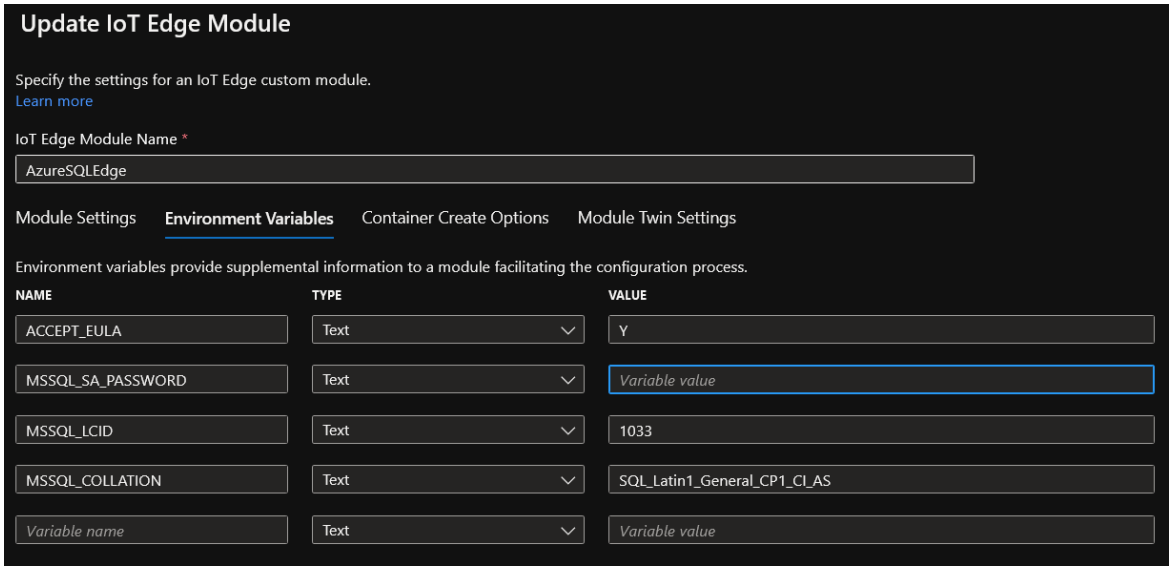

Figure 7 Set SQL password and language

7. For the Container Create Options section, there is a JSON file displayed. Here the module's volume bindings and SQL port number can be edited. Recommend leaving it at the default of 1433.

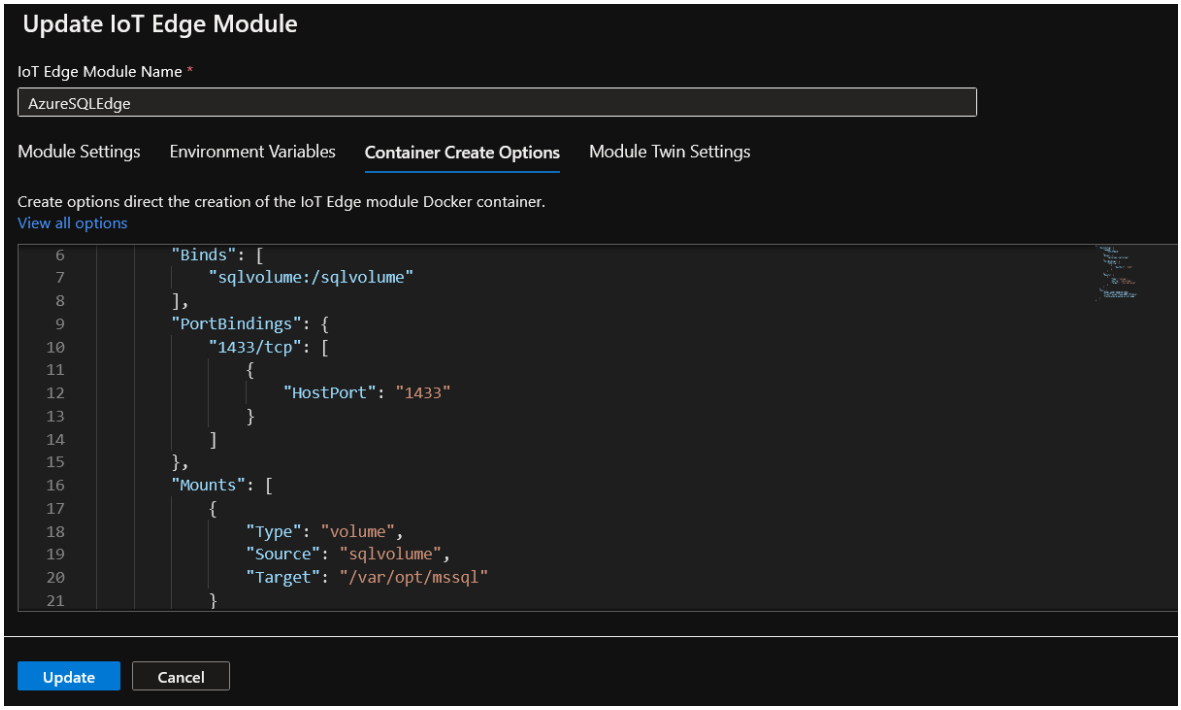

Figure 8 Edit deployment JSON file parameters

8. Click Update on the Update IoT Edge Module panel to save the changes.
9. Click Review + Create, then click Create. Open the IoTEdge device and verify the modules are up.

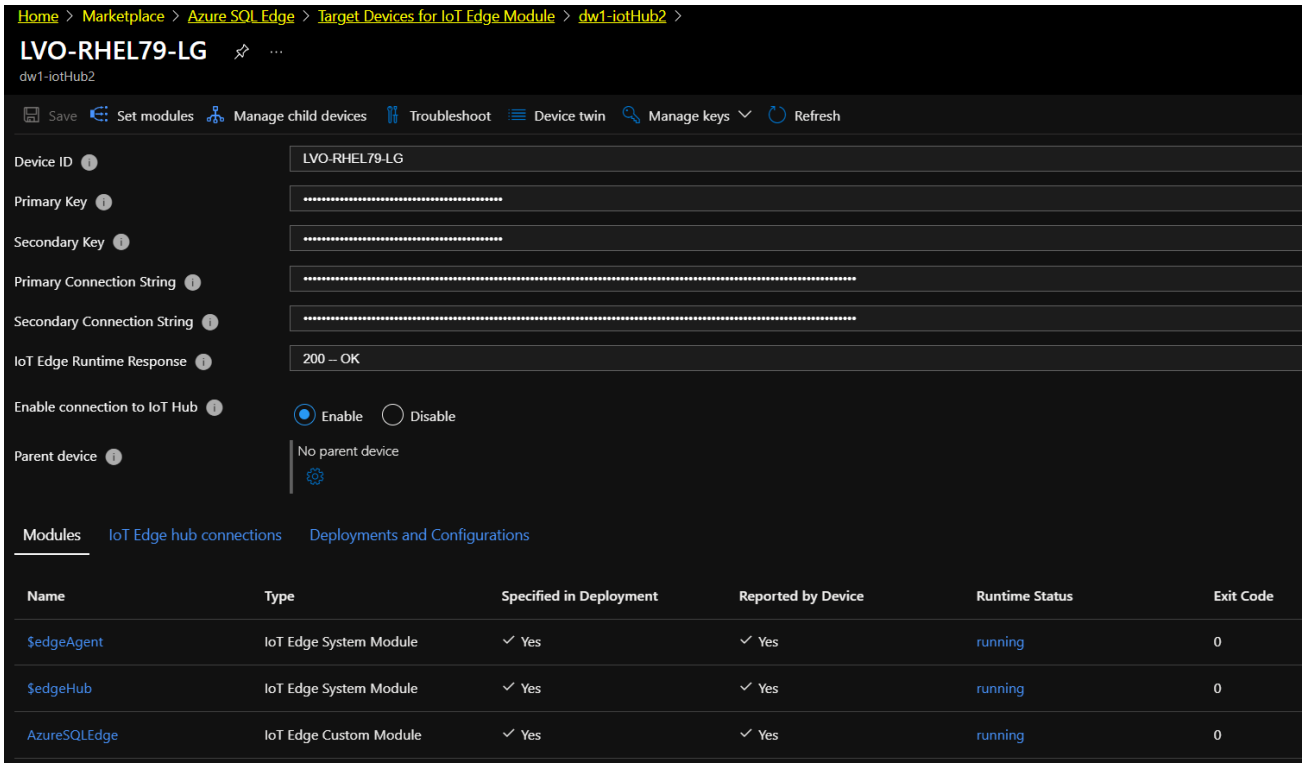

Figure 9 Verify module is running on IoT Edge device

7 Azure SQL Edge - Docker method

This section covers basic docker deployment method. It does not use the Azure IoT edge components and runs as a standalone SQL container on an edge device. This method is significantly easier, and a good option for development or testing SQL Edge. It involves pulling and running the Docker image Microsoft has provided.

1. Pull the image

   sudo docker pull mcr.microsoft.com/azure-sql-edge:latest

2. Run the image

   sudo docker run --cap-add SYS_PTRACE -e 'ACCEPT_EULA=1' -e 'MSSQL_SA_PASSWORD=yourPassword' -e 'MSSQL_PID=Premium' -p 1433:1433 --name azuresqledge -d mcr.microsoft.com/azure-sql-edge

3. View the container to verify its running

   sudo docker ps -a

At this point, use the methods in section 8 below to connect to the container or SQL instance.

8 Connecting to the SQL Edge container

After the SQL Edge container is deployed, there are several ways to connect and manage it. Connecting to the container's command line (CLI) enables viewing and configuring the file system and using other command-based tools. Using SQL Server Management Studio (SSMS) allows a direct connection to the SQL instance with a graphical interface to manage the databases.

8.1 Connect to the container CLI

Use the following Docker commands to connect to the container. Replace "azuresqledge" with whatever name was used for the SQL Edge container.

sudo docker exec -it azuresqledge "bash"

This opens a command line bash shell directly at the root file system on the container.

8.2 Connect to the SQL instance with SSMS

To connect with SSMS, enter the IP address assigned to the container. If the default SQL Server port number of 1433 was used during deployment, then SSMS should find the SQL instance and connect. Otherwise, use a colon after the IP address and the alternate port number.

9 Add data volumes to SQL Edge container

This section covers adding separate data volumes to a SQL Edge container. Note that this only works on the Docker deployment method (section 7). By default, the container and its SQL data directories all run on one volume, which the entire container uses. To improve performance, this mounts separate data volumes that are based on faster disks. This also makes the data persistent and separate, should the container be removed or deleted.

1. The first step is to mount the volumes to the container host's file system just like any other Linux volume. With Linux, the volumes get mounted to a directory. In the next step these directories (the volumes) are mounted as data volumes to the container.
2. Run this to get the image. sudo docker pull mcr.microsoft.com/azure-sql-edge:latest
3. Use the following command to create a SQL Edge container with the new data volumes mounted.

   sudo docker run --cap-add SYS_PTRACE -e 'ACCEPT_EULA=1' -e 'MSSQL_SA_PASSWORD=yourPassword' -e 'MSSQL_PID=Premium' -p 1433:1433 --name azuresqledge -d mcr.microsoft.com/azure-sql-edge -v <your host directory>/data:/var/opt/mssql/data -v <your host directory>/log:/var/opt/mssql/log

The -v specifies the volume parameter, which includes the volumes directory on the host.

In the example above, its mounting a data and log volume. The mount points on the container are within the default SQL installation directories on the container.

At this point, the SQL database and log files can be moved to these faster data volumes.

10 Performance Data

The performance tests are based on a 100 warehouse TPC-C database run with HammerDB. All tests used 64 GB RAM and 50 virtual users and varied the CPU count. The underlying storage for the VM's data VHDs were 6 disk RAID5 for the database and 2 disk RAID1 for the logs. The SE350 can be configured with up to 16 CPUs. By running performance tests up to 8 CPU there is remaining capacity for other applications, which is typical of an edge server.

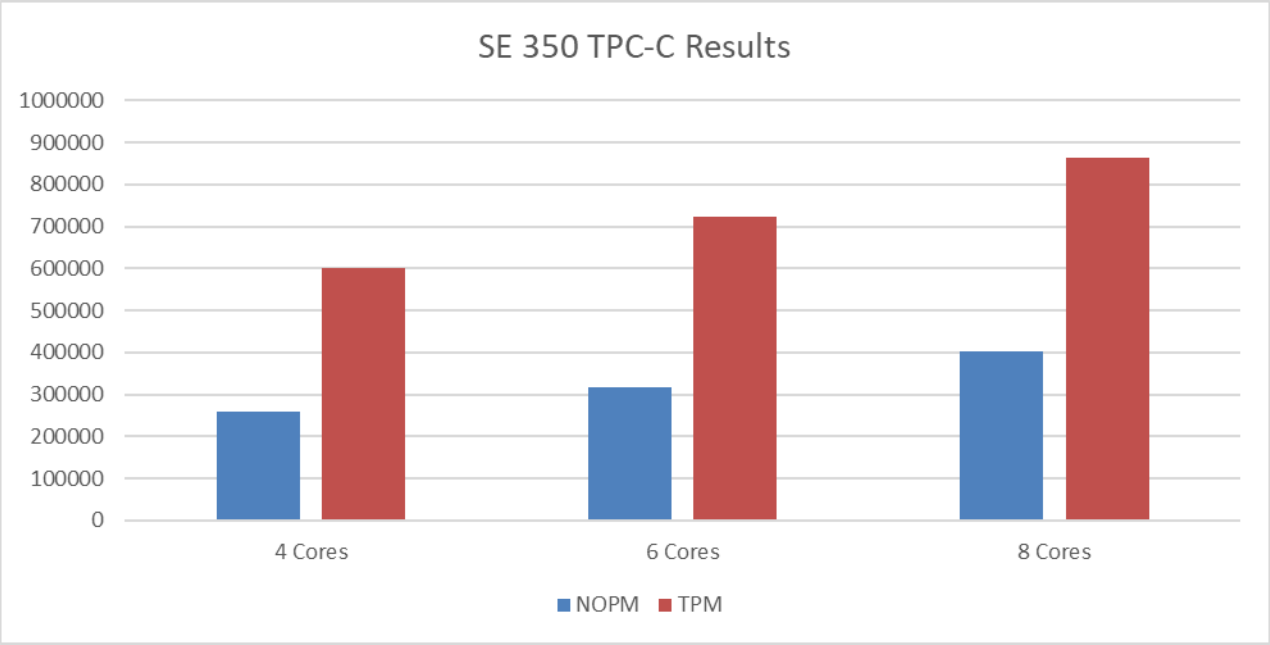

Figure 10 Performance testing

HammerDB is a database load testing and benchmarking tool. We used HammerDB to create a test schema, load it with data and simulate the workload of multiple virtual users against the database for both transactional and analytic scenarios. This workload can then be used to derive meaningful information about your environment such as hardware performance comparisons and software configurations. The benchmark involves a mix of five concurrent transactions of different types and complexity either executed online or queued for deferred execution. we have performed TPC-C benchmarking with Microsoft SQL 2019 using the widely accepted open-source benchmarking tool HammerDB. These results should not be compared actual TPC-C benchmarks results officially published.

11 Sample Use Case - IoT streaming data

Azure SQL Edge provides a local database at the edge for IoT devices. It features streaming live data and data retention policies to address limited storage on edge servers. This section covers deployment of an IoT Edge module that simulates several IoT sensor devices.

The code for this has been provided by Microsoft and is available on GitHub. The application gets uploaded to Azure as a docker image and deployed from there as a module to the IoT Edge device. Once deployed, it generates simulated telemetry data and streams it to the Azure SQL Edge database.

A summary of the steps to deploy this simulator include:

- Prepare the Visual Studio environment and load the project file
- Create a Container Registry in Azure
- Edit the code and push the build to the Azure registry
- Deploy the module from Azure IoT Hub to the IoT Edge device
- Create the database for the simulator data and set the routes

11.1 Download the code

Microsoft has made the code available at the following GitHub location. Search for microsoft/sqlsourabh in GitHub and download the entire repository.

11.2 Prepare Visual Studio for Azure IoT projects

There are a few prerequisites that need to be setup on the workstation that is running Visual Studio for this to work correctly.

1. Windows Subsystem for Linux v2 (WSL). Follow the steps at this link to install. Note that things will go smoother if the workstation is running a newer version of Windows 10, noted in the URL. After installing WSL, it will install a version of Linux.
   https://docs.microsoft.com/en-us/windows/wsl/install
2. Docker Desktop (Docker for Windows) It requires WSL v2, and the download and steps to install it are here https://www.docker.com/products/docker-desktop
3. Verify the sources are in Visual Studio correctly. Go to Tools - Nuget package manager - Package manager settings. Under Nuget Package Manager - Package sources, verify that Nuget.org is listed. If not, enter it. The source URL to enter is: https://api.nuget.org/v3/index.json
4. To build the simulator module, Visual Studio needs to have the supporting IoT Edge related extensions installed. Go to Extensions - Manage Extensions and search for IoT Edge. Look for Azure IoT Edge Tools for VS 2019 and add it. Once selected, close Visual Studio so it can install it.

11.3 Load the project file in Visual Studio

From Visual Studio we will open the project file in the GitHub download by navigating to: SQLEdgeSamples\IoTEdgeSamples\PredictiveMaintenance\PredictiveMaintenanceParent\PredictiveMaintenance and open the file named PredictiveMaintenance.sln. This is the overall solution file for the project. In the Visual Studio tree view there should be two main sections loaded. One called PredictiveMaintenance and another named TelemetryData.

11.4 Create a Container Registry in Azure

In the Azure portal search for container registries and open it. Click on Create and fill in the basic details on the page such as resource group and the registry name. Click review and create to finish.

Install the Azure CLI on the same system as Visual Studio and use Powershell to verify you can login to the registry. The login syntax in Powershell is: az acr login -n <nameOfRegistry> -u <username> -p <password> The login credentials can be found under the Access keys section of the container registry. Once the workstation is logged in, Visual Studio will be able to access it seamlessly for the next step below.

11.5 Edit the code and push the build to Azure

The code is mostly usable as-is from GitHub, with only one file that needs editing. The file is under Telemetry data, named module.json, and has a line that specifies the container registry. The existing file has a container registry name Microsoft was using. The line to edit is line 5 as shown below, and the part that needs updating is highlighted.

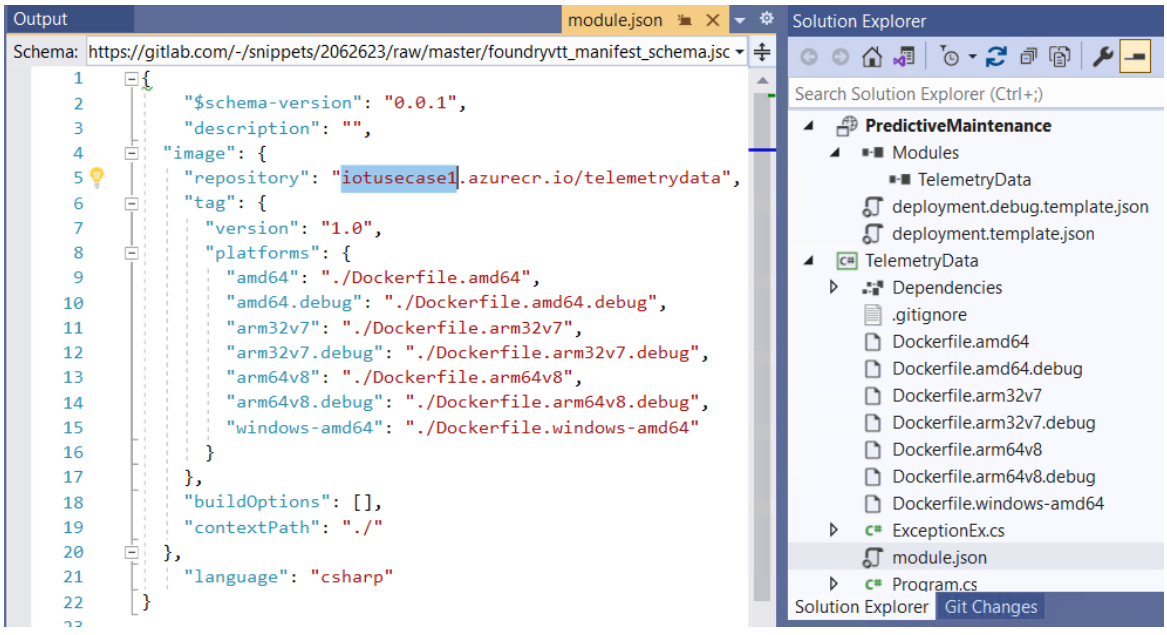

Figure 11 Editing the module.json file container registry name

Replace the highlighted part with the name of your container registry. This tells Visual Studio where to upload the image. Note that it puts it in the registry and a subfolder named Telemetrydata. Save the file.

At this point the solution can be built and pushed to the Azure container registry. In the Visual Studio solution explorer tree, right click on PredictiveMaintenance and select **Build and push IoT Edge module** as shown below. Wait for it to finish and verify the output panel is without errors.

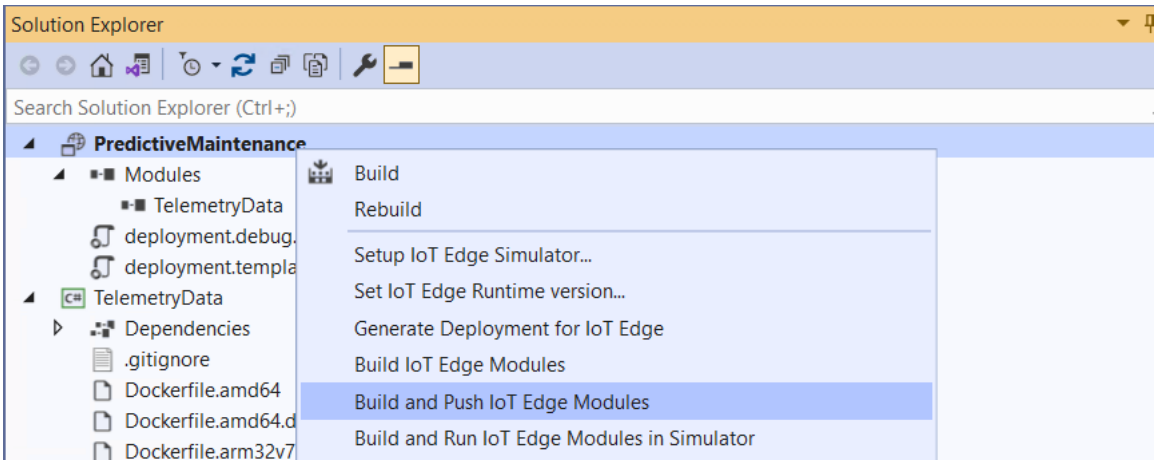

Figure 12 Build and Push IoT Edge Module in Visual Studio

11.6 Deploy the simulator module to the IoT Edge device

Deploying the module is done just like any other IoT Edge Hub module. In the Azure portal, go to your IoT Hub, and select the IoT Edge device. From the Set Modules blade, enter the container registry information at the top of the page, shown below. Then select Add, and choose IoT Edge Module.

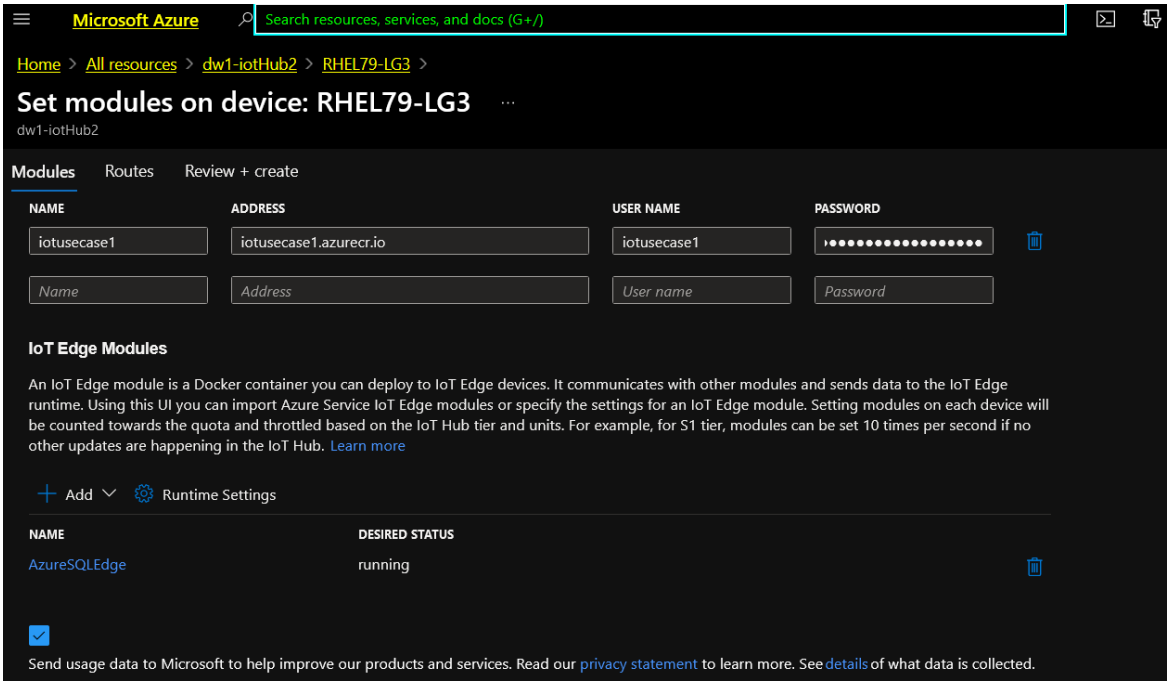

Figure 13 Enter Container Registry information and credentials

Enter a name and add the URI which is the container registry location plus the file name. Copy what is in the screen below except replace the first part with whatever name was used for your container registry.

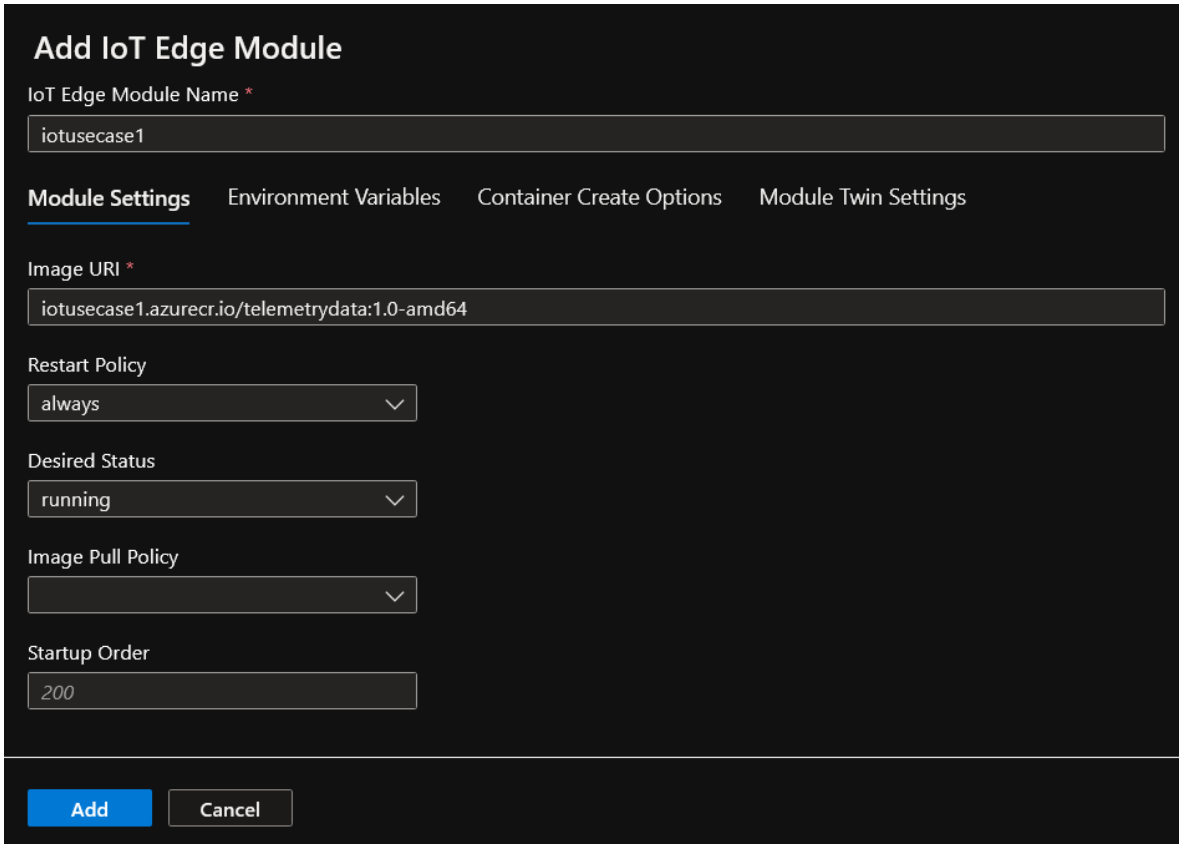

Figure 14 Adding new IoT Edge module details

Set desired state to running, no other settings are needed at this point. Click Add to finish. The module should show as running in the IoT Edge device's summary of modules, as shown below.

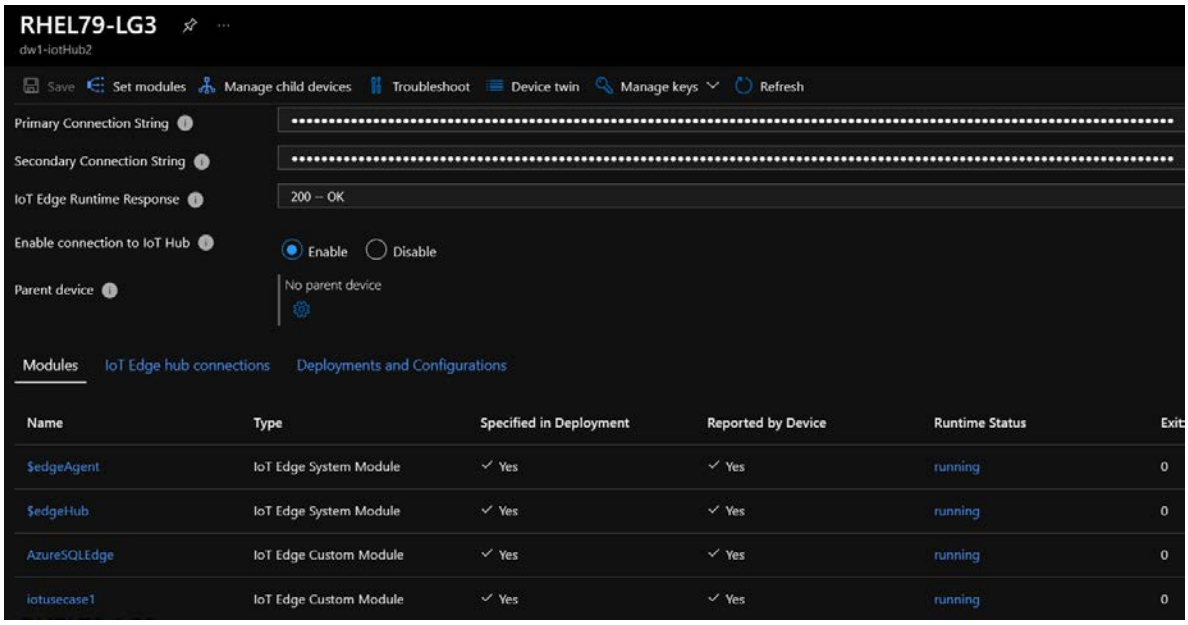

Figure 15 New IoT Edge module status of running

11.7 Create the database for the simulator data

Microsoft provides a SQL script to create the database for storing the streaming data. The script is in the GitHub repository, under SQLEdgeSamples\IoTEdgeSamples\PredictiveMaintenance\deployment scripts. Look for the script file named Machine_Telemetry_Database_Objects.sql.

1. Connect to the Azure SQL Edge server using Azure Data Studio (ADS). For ADS use the format of server name or IP address,1433 (or whatever your SQL port is). Note that ADS uses a comma not a semi-colon.
2. There are supporting files that need to be copied to the SQL data directory in the Azure SQL Edge container. To simplify this, just copy all the files in the deployment scripts folder to the sql data folder which is /var/opt/mssql/data. Since SQL Edge is a container, copy the files to the Linux host first (the IoT Edge device) then use the docker cp command to copy into the container. Here is an example:

   docker cp /tmpDir/*.* AzureSQLEdge:/var/opt/mssql/data/
3. Load the script into ADS and edit the passwords to match your SQL password. The password related lines are line 38 and 190 in the script.
4. Finally, run the script to create the database. Keep the script window open, so that the results panel can be viewed in later steps.

11.8 Add routes to IoT Edge Module

Go back to the IoT Edge module in the Azure portal. Follow the steps below to add a route.

1. Open the module and select Set Modules then click on routes.
2. In the route name box, enter Machine1.
3. In the value box enter the below string exactly. Note that the bold part needs to be changed to whatever name you gave the module in section 11.6. This is all one continuous string.

   FROM /messages/modules/**iotusecase1**/outputs/Machine1 INTO BrokeredEndpoint ("/modules/AzureSQLEdge/inputs/MachineTelemetry")
4. Leave the priority and time to live boxes at defaults. Click review and create to finish creating the route.
5. If you want to actually have data for 5 simulated sensors, then repeat the above steps and create four more routes, each named Machine2,3,4 and 5 and edit the Machine# value in the string on each one.

Note: Be aware that running even one of these simulators will generate a lot of messages between Azure IoT Hub and the IoT Edge Device and modules. If using the free tier for IoT message flow, it could exceed the free tier daily limits. If it exceeds the limit, Azure will shut down the IoT hub.

To avoid exceeding the message limits, stop the IoT Edge modules on the IoT Edge Device when you aren’t using it. Stop it by running this command:

Sudo systemctl stop iotedge

11.9 Verify streaming data flow and analyze results

After the route is added, the simulator data will be sending streaming data to the SQL Edge database. The data can be viewed in the results tab of the SQL script in Azure Data Studio.

Results Messages

	timestamp	var_machineid	var_voltate	var_rotate	var_pressure	var_vibration	var_error1	var_error2
1	2021-06-09 15:17:44.530	3	158.978900	269.343100	92.692600	35.082100	0.055300	0.014000
2	2021-06-09 15:17:44.530	5	155.902700	281.060900	104.754600	34.050800	0.028700	0.004400
3	2021-06-09 15:17:44.530	1	159.294100	265.480300	91.601500	37.506200	0.008100	0.002400
4	2021-06-09 15:17:44.530	4	160.587400	267.654400	93.084300	35.936900	0.000000	0.008900
5	2021-06-09 15:17:44.530	2	161.479600	347.671000	90.555400	35.420400	0.027100	0.001700
6	2021-06-09 15:17:39.413	4	156.900500	268.340900	91.489100	35.388700	0.014200	0.003200
7	2021-06-09 15:17:39.413	5	161.330200	269.154200	99.837000	36.738900	0.015500	0.002800

The data messages can also be viewed in the IoT edge module logs. Use this command on the IoT Edge device to view the data being generated by the sensors.

sudo docker logs -f AzureSqlEdge --tail 100

11.10 Data Retention settings

Azure SQL Edge administrators can set data retention policies on a SQL Edge database and the tables. After the retention policy is set background tasks purge old data. The setting is included in the sample database script Microsoft provided. The command below enables it on the database.

```
ALTER DATABASE [PredictiveMaintenance] SET DATA_RETENTION ON
```

This command example sets it on the specific table.

```
Create Table [Example]
(
    [timestamp] datetime,
    var_machineid smallint,
    var_voltate numeric(11,6),
    var_error2 numeric(11,6)
)With (DATA_DELETION = On (FILTER_COLUMN = [timestamp], RETENTION_PERIOD = 1 day))
```

12 Appendix: Bill of Materials

This appendix features the Bill of Materials (BOMs) for the SE350 server. The BOM listed is not meant to be exhaustive and must always be confirmed with the configuration tools and customer requirements.

12.1 ThinkSystem SE350 for Azure SQL Edge BOM

Part number	Product Description	Qty
7D1XCTO1WW	Node : ThinkSystem SE350 - 3yr Warranty	1
B6EQ	ThinkSystem SE350 Edge Server Chassis	1
B6F4	ThinkSystem SE350 10GbE SFP+ 2-Port, 10/100/1GbE RJ45 2-Port Intel i350	1
B8ZR	Standard Shock & Vibration (15G & 0.21Grms)	1
B8ZT	Operational Temperature 0-45C	1
BFYB	Operating mode selection for: "Maximum Performance Mode"	1
B93A	ThinkSystem SE350 Edge Server Intel Xeon D-2143IT 8C 65W 2.20 GHz	1
AUNC	ThinkSystem 16GB TruDDR4 2666 MHz (2Rx8 1.2V) RDIMM	4
B6FF	ThinkSystem SE350 M.2 SATA/NVMe 4-bay Data Drive Enablement Kit	1
5977	Select Storage devices - no configured RAID required	1
AVUX	On Board SATA AHCI Mode	1
B75A	ThinkSystem M.2 800GB Industrial A600i SATA SSD	4
B6FH	ThinkSystem SE350 M.2 Adapter SATA Cable	1
B88P	ThinkSystem SE350 M.2 Mirroring Enablement Kit	1
B758	ThinkSystem M.2 120GB Industrial A600i SATA SSD	2
B6FE	ThinkSystem SE350 M.2 Riser Cage Assembly	1
B6FU	ThinkSystem SE350 - 12V PDM	1
B6FW	ThinkSystem SE350 240W AC Adapter	2
6311	2.8m, 10A/100-250V, C13 to IEC 320-C14 Rack Power Cable	2
B6KT	ThinkSystem SE350 - Mini USB to USB Type A (F) Console Cable	1
B755	Desktop Mode	1
B6Q3	ThinkSystem SE350 Rubber Feet	1
AUPW	ThinkSystem XClarity Controller Standard to Enterprise Upgrade	1
B173	Companion Part for XClarity Controller Standard to Enterprise Upgrade in Factory	1
5PS7A34998	Premier Essential - 3Yr 24x7 4Hr Resp + YDYD SE350	1
5641PX3	XClarity Pro, Per Endpoint w/3 Yr SW S&S	1
1340	Lenovo XClarity Pro, Per Managed Endpoint w/3 Yr SW S&S	1

Conclusion

By deploying the Lenovo ThinkSystem SE350 edge server + Microsoft Azure SQL edge solution, you can quickly solve your database processing needs at the edge with high performance and resiliency at a reasonably low cost. This solution can also be implemented on ThinkAgile MX1020 integrated systems or ThinkAgile MX1021 certified nodes for high availability.

Change history

Changes in the December 1, 2021 update (version 1.1):

- Added use case for Azure SQL Edge on Lenovo SE350
- Added a section on adding data volumes to SQL Edge containers

Resources

For more information about the topics that are described in this document, see the following resources:

- IoT Edge on RHEL 7.x. Blog posting by Microsoft IoT Edge team.
  http://busbyland.com/install-iot-edge-on-red-hat-enterprise-linux-rhel-7-x/
- Azure SQL Edge documentation main page
  https://docs.microsoft.com/en-us/azure/azure-sql-edge/overview
- Azure IoT Edge device deployment - via Azure
  https://docs.microsoft.com/en-us/azure/azure-sql-edge/deploy-portal
- Azure IoT Edge device deployment - via Docker
  https://docs.microsoft.com/en-us/azure/azure-sql-edge/disconnected-deployment
- Lenovo ThinkSystem SE350 product overview
  https://www.lenovo.com/us/en/c/data-center/servers/edge
- Hammerdb toolkit

  https://www.hammerdb.com/

Trademarks and special notices

References in this document to Lenovo products or services do not imply that Lenovo intends to make them available in every country.

The following terms are trademarks of Lenovo in the United States, other countries, or both:

Lenovo®
ThinkSystem
TruDDR4
XClarity®

The following terms are trademarks of other companies:

Intel® and Xeon® are trademarks of Intel Corporation or its subsidiaries.

Linux® is the trademark of Linus Torvalds in the U.S. and other countries.

Azure®, Microsoft®, and SQL Server® are trademarks of Microsoft Corporation in the United States, other countries, or both.

TPC and TPC-C are trademarks of Transaction Processing Performance Council.

Other company, product, or service names may be trademarks or service marks of others.

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used Lenovo products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-Lenovo products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by Lenovo. Sources for non-Lenovo list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. Lenovo has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-Lenovo products. Questions on the capability of non-Lenovo products should be addressed to the supplier of those products.

All statements regarding Lenovo future direction and intent are subject to change or withdrawal without notice and represent goals and objectives only. Contact your local Lenovo office or Lenovo authorized reseller for the full text of the specific Statement of Direction.

Some information addresses anticipated future capabilities. Such information is not intended as a definitive statement of a commitment to specific levels of performance, function or delivery schedules with respect to any future products. Such commitments are only made in Lenovo product announcements. The information is presented here to communicate Lenovo’s current investment and development activities as a good faith effort to help with our customers' future planning.

Performance is based on measurements and projections using standard Lenovo benchmarks in a controlled environment. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Photographs shown are of engineering prototypes. Changes may be incorporated in production models.

Any references in this information to non-Lenovo websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this Lenovo product and use of those websites is at your own risk.