Consolidated Scenario

This is a work-in-progress document that contains the complete scenario to assist with editing.

> **Note**: This will be removed once the complete scenario has been rolled into the individual labs.

LAB_AK_01

Lab Scenario

You are an Azure IoT Developer for a leading gourmet cheese company named Contoso. Contoso evaluated the business opportunity for IoT and concluded that there are significant benefits that they can achieve. They selected Microsoft Azure IoT based on their evaluations.

To get started, you need to become familiar with the Azure tools.

Exercise 1: Explore the Azure Portal and Dashboard

Before you begin working with the Azure IoT services, it's good to be familiar with how Azure itself works.

Although we commonly refer to Azure as a 'cloud', it is actually a web portal that is designed to make Azure resources accessible from a single web site. All of Azure is accessible through the Azure portal.

Exercise 2: Create an Azure Dashboard and Resource Group

On the Azure portal, dashboards are used to present a customized view of your resources. Information is displayed through the use of tiles which can be arranged and sized to help you organize your resources in useful ways. You can create many different dashboards that provide different views and serve different purposes.

Each tile that you place on your dashboard exposes one or more of your resources. In addition to tiles that expose the data of an individual resource, you can create a tile for something called a resource group.

A resource group is a logical group that contains related resources for a project or application. The resource group can include all the resources for the solution, or only those resources that you want to manage as a group. You decide how you want to allocate resources to resource groups based on what makes the most sense for your organization. Generally, add resources that share the same lifecycle to the same resource group so you can easily deploy, update, and delete them as a group.

In the following tasks, you will:

- create a custom dashboard that you can use during this course
- create a Resource Group and add a Resource Group tile to your dashboard

LAB_AK_02

Lab Scenario

You are an Azure IoT Developer working for Contoso, a company that crafts and distributes gourmet cheeses. You have been tasked with exploring Azure and the Azure IoT services that will be used in the IoT solution that the company is planning to implement. You have become familiar with the Azure portal and created a resource group for your project. Now you need to begin your investigation of Azure IoT services.

Exercise 1: Naming Resources with Unique Names

Throughout this course you will be creating resources. To ensure consistency across the labs and to help in tidying up resources whenever you have finished with them, we will be providing you with the names you should use. However, many of these resources expose services that can be consumed across the web, which means they must have globally unique names. To achieve this, you will be using a unique identifier that will be added to the end of the resource name. Let's create your unique ID.

Exercise 2: Create an IoT Hub using the Azure portal

The Azure IoT Hub is a fully managed service that enables reliable and secure bidirectional communications between IoT devices and Azure. The Azure IoT Hub service provides the following:

- Establish bidirectional communication with billions of IoT devices
- Multiple device-to-cloud and cloud-to-device communication options, including one-way messaging, file transfer, and request-reply methods.
- Built-in declarative message routing to other Azure services.
- A queryable store for device metadata and synchronized state information.
- Secure communications and access control using per-device security keys or X.509 certificates.
- Extensive monitoring for device connectivity and device identity management events.
- SDK device libraries for the most popular languages and platforms.

There are several methods that you can use to create an IoT Hub. For example, you can create an IoT Hub resource using the Azure portal, which is what you will do in ths task. But you can also create an IoT Hub (and other resources) programmatically. During this course we will be investigating additional methods that can be used to to create and manage Azure resources, including Azure CLI and PowerShell.

Exercise 3: Examine the IoT Hub Service

As we have already noted, the IoT Hub is a managed service, hosted in the cloud, that acts as a central message hub for bi-directional communication between your IoT application and the devices it manages.

IoT Hub's capabilities help you build scalable, full-featured IoT solutions such as managing industrial equipment used in manufacturing, tracking valuable assets in healthcare, monitoring office building usage, and many more scenarios. IoT Hub monitoring helps you maintain the health of your solution by tracking events such as device creation, device failures, and device connections.

Exercise 4: Create a Device Provisioning Service using the Azure portal

The Azure IoT Hub Device Provisioning Service is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention. The Device Provisioning Service provides the following:

- Zero-touch provisioning to a single IoT solution without hardcoding IoT Hub connection information at the factory (initial setup)

- Load balancing devices across multiple hubs
- Connecting devices to their owner's IoT solution based on sales transaction data (multitenancy)
- Connecting devices to a particular IoT solution depending on use-case (solution isolation)
- Connecting a device to the IoT hub with the lowest latency (geo-sharding)
- Reprovisioning based on a change in the device
- Rolling the keys used by the device to connect to IoT Hub (when not using X.509 certificates to connect)

There are several methods that you can use to create an instance of the IoT Hub Device Provisioning Service. For example, you can use the Azure portal, which is what you will do in ths task. But you can also create a DPS instance using Azure CLI or an Azure Resource Manager Template.

Exercise 5: Examine the Device Provisioning Service

The IoT Hub Device Provisioning Service is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention, enabling customers to provision millions of devices in a secure and scalable manner.

LAB_AK_03

Lab Scenario

As one of the developers at Contoso, you know that setting up your development environment is an important step before starting to build your Azure IoT solution. You know that Microsoft provides a number of tools that can be used to develop and support your IoT solutions and that some decisions should be made about which tools your team will use. You will prepare a work environment that the team can use to develop your IoT solution, both on the Azure cloud side and for your local work environment.

After some discussion, your team has made the following high-level decisions about the dev environment:

- Operating System: Windows 10 will be used as the OS. Windows is used by most of your team, so it was a logical choice. You note that Azure IoT services support other operating systems (such as Mac OS and Linux), and Microsoft provides supporting documentation for those members of the team who choose one of these alternatives.
- General Coding Tools: Visual Studio Code and Azure CLI will be used as the primary coding tools. Both of these tools support extensions for IoT that leverage the Azure IoT SDKs.
- IoT Edge Tools: Docker Desktop Community and Python will be used to support custom IoT Edge module development.

In support of these decisions, you will be setting up the following environment:

- Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later). Including
    - 4GB – 8GB system RAM (higher the better for Docker)
    - Hyper-V and Containers features of Windows must be enabled.
    - BIOS-level hardware virtualization support must be enabled in the BIOS settings.

> **Note**: When setting up the development environment on a virtual machine, the VM environment must support nested virtualization - nested virtualization

- Azure CLI (current/latest)
- .NET Core 3.0.100 (or later) SDK
- VS Code (latest)
- Python 3.7 (not 3.8)
- Docker Desktop Community 2.1.0.5 (or later) set to Linux Containers
- Power BI Desktop (for data visualization)
- IoT Extensions for VS Code and Azure CLI

> **Note**: A virtual machine has been created for this course that provides a majority of the tools specified above. The instructions below support using the prepared VM or setting up the development environment locally using your PC.

Exercise 1: Install Developer Tools and Products

> **Note**: The tools and products associated with this Exercise are pre-installed on the virtual machine created for this course. Before continuing, check with your course Instructor to understand if you will be completing labs using the hosted lab VM environment or setting up the dev environment locally on your PC.

Exercise 2: Install Dev Tool Extensions

The Visual Studio Code and Azure CLI tools both support extension that help developers to create their solutions more efficiently. Extensions for IoT have been developed by Microsoft that leverage the IoT SDKs and reduce development time.

Exercise 3: Set Up Course Lab Files and Alternative Tools

A number of the labs in this course rely on pre-built resources, such as a code project that can be used as a starting point for the lab activity. We provide access to these lab resources through the use of a GitHub project. In addition to resources that directly support the course labs (the resources contained in the GitHub project), there are tools that can be used to support learning opportunities outside the actual course. The instructions below lead you through the configuration of both these resource types.

LAB_AK_04

Lab Scenario

Contoso is known for producing high quality cheeses. Due to the company's rapid growth in both popularity and sales, they want to take steps to ensure that their cheeses stay at the same high level of quality that their customers expect.

In the past, temperature and humidity data was collected by factory floor workers during each work shift. The company is concerned that the factory expansions will require increased monitoring as the new facilities come online and that a manual process for collecting data won't scale.

Contoso has decided to launch an automated system that uses IoT devices to monitor temperature and humidity. The rate at which telemetry data is communicated will be adjustable to help ensure that their manufacturing process is under control as batches of cheese proceed through environmentally sensitive processes.

To evaluate this asset monitoring solution prior to full scale implementation, you will be connecting an IoT device (that includes temperature and humidity sensors) to IoT Hub.

Exercise 1: Verify Lab Prerequisites

This lab assumes that the following Azure resources are available:

Resource Type	Resource Name
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*

Exercise 2: Create Azure IoT Hub Device ID using Azure CLI

The `iot` Azure CLI modules includes several commands for managing IoT Devices within Azure IoT Hub under the `az iot hub device-identity` command group. These commands can be used to manage IoT Devices within scripts or directly from the command-line / terminal.

Exercise 3: Configure and Test a Simulated Device (C#)

In this exercise you will configure a simulated device written in C# to connect to Azure IoT Hub using the Device ID and Shared Access Key created in the previous exercise. You will then test the device and ensure that IoT Hub is receiving telemetry from the device as expected.

LAB_AK_05

Lab Scenario

Contoso management is pushing for an an update to their Asset Monitoring and Tracking Solution that will use IoT devices to reduce the manual data entry work that is required under the current system and provide more advanced monitoring during the shipping process. The solution relies on the ability to provision and de-provision IoT devices. The best option for managing the provisioning requirements appears to be DSP.

The proposed system will use IoT devices with integrated sensors for tracking the location, temperature, pressure of shipping containers during transit. The devices will be placed within the existing shipping containers that Contoso uses to transport their cheese, and will connect to Azure IoT Hub using vehicle provided WiFi. The new system will provide continuous monitoring of the product environment and enable a variety of notification scenarios when issues are detected.

In Contoso's cheese packaging facility, when an empty container enters the system it will be equipped with the new IoT device and then loaded with packaged cheese products. The IoT device needs to be auto-provisioned to IoT hub using Device Provisioning Service. When the container arrives at the destination, the

IoT device will be retrieved and then "decommissioned" through DPS. The device will be re-used for future shipments.

You have been tasked with validating the device provisioning and de-provisioning process using DPS. For the initial phase of the process you will use an Individual Enrollment approach.

Exercise 1: Verify Lab Prerequisites

This lab assumes that the following Azure resources are available:

Resource Type	**Resource Name**
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*
Device Provisioning Service	AZ-220-DPS-*{YOUR-ID}*

Exercise 2: Create new individual enrollment (Symmetric keys) in DPS

In this exercise, you will create a new individual enrollment for a device within the Device Provisioning Service (DPS) using *symmetric key attestation*.

Exercise 3: Configure Simulated Device

In this exercise, you will configure a Simulated Device written in C# to connect to Azure IoT using the individual enrollment created in the previous unit. You will also add code to the Simulated Device that will read and update device configuration based on the device twin within Azure IoT Hub.

The simulated device that you create in this exercise represents an IoT Device that will be located within a shipping container/box, and will be used to monitor Contoso products while they are in transit. The sensor telemetry from the device that will be sent to Azure IoT Hub includes Temperature, Humidity, Pressure, and Latitude/Longitude coordinates of the container. The device is part of the overall asset tracking solution.

This is different than the earlier lab where a simulated device connected to Azure because in that lab, you used a shared access key to authenticate, which does not require device provisioning, but also does not give the provisioning management benefits (such as device twins), and requires fairly large distribution and management of a shared key. In this lab, you are provisioning a unique device through the Device Provisioning Service.

Exercise 4: Test the Simulated Device

In this exercise, you will run the Simulated Device and verify it's sending sensor telemetry to Azure IoT Hub. You will also update the delay at which telemetry is sent to Azure IoT Hub by updating the device twin for the simulated device within Azure IoT Hub.

Exercise 5: Retire the Device

In this unit you will perform the necessary tasks to retire the device from both the Device Provisioning Service (DPS) and Azure IoT Hub. To fully retire an IoT Device from an Azure IoT solution it must be removed from both of these services. When the transport box arrives at it's final destination, then sensor will be removed

from the box, and needs to be "decommissioned". Complete device retirement is an important step in the life cycle of IoT devices within an IoT solution.

LAB_AK_06

Lab Scenario

Your work to-date on Contoso's Asset Monitoring and Tracking Solution has enabled you to validate the device provisioning and de-provisioning process using an Individual Enrollment. The management team has now asked you to begin rolling out the process on a larger scale.

To keep the project moving forward you need to demonstrate that the Device Provisioning Service can be used to enroll larger numbers of devices automatically and securely using X.509 certificate authentication.

Exercise 1: Verify Lab Prerequisites

This lab assumes that the following Azure resources are available:

Resource Type	**Resource Name**
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*
Device Provisioning Service	AZ-220-DPS-*{YOUR-ID}*

Exercise 2: Generate and Configure X.509 CA Certificates using OpenSSL

In this exercise, you will generate an X.509 CA Certificate using OpenSSL within the Azure Cloud Shell. This certificate will be used to configure the Group Enrollment within the Device Provisioning Service (DPS).

Exercise 3: Create Group Enrollment (X.509 Certificate) in DPS

In this exercise, you will create a new individual enrollment for a device within the Device Provisioning Service (DPS) using *certificate attestation*.

Exercise 4: Configure simulated device with X.509 certificate

In this exercise, you will configure a simulated device written in C# to connect to your Azure IoT Hub via your Device Provisioning Service (DPS) using an X.509 certificate. This exercise will also introduce you to the workflow within the **simulated device** source code within the `/LabFiles` directory, and how it works to authenticate with DPS and send messages to IoT Hub.

Exercise 5: Handle device twin desired property Changes

In this exercise, you will modify the simulated device source code to include an event handler to update device configurations based on device twin desired property changes sent to the device from Azure IoT Hub.

Device twins are JSON documents that store device state information including metadata, configurations, and conditions. Azure IoT Hub maintains a device twin for each device that you connect to IoT Hub.

The Device Provisioning Service (DPS) contains the initial device twin desired properties for devices that are registered using Group Enrollment. Once the devices are registered they are created within IoT Hub using this initial device twin configuration from DPS. After registration, the Azure IoT Hub maintains a device twin (and its properties) for each device within the IoT Hub Device Registry.

When the device twin desired properties are updated for a device within Azure IoT Hub, the desired changes are sent to the IoT Device using the `DesiredPropertyUpdateCallback` event using the C# SDK. Handling this event within device code enables the devices configuration and properties to be updated as desired by easily managing the Device Twin state for the device within Azure IoT Hub.

This set of steps will be very similar to steps in earlier labs for working with a simulated device, because the concepts and processes are the same. The method used for authentication of the provisioning process doesn't change the handing of device twin property changes once the device is provisioned.

Exercise 6: Test the Simulated Device

In this exercise, you will run the simulated device. When the device is started for the first time, it will connect to the Device Provisioning Service (DPS) and automatically be enrolled using the configured group enrollment. Once enrolled into the DPS group enrollment, the device will be automatically registered within the Azure IoT Hub device registry. Once enrolled and registered, the device will begin communicating with Azure IoT Hub securely using the configured X.509 certificate authentication.

Exercise 7: Retire Group Enrollment

In this exercise, you will retire the enrollment group and its devices from both the Device Provisioning Service and Azure IoT Hub.

LAB_AK_07

Lab Scenario

Contoso Management is impressed with the implementation of automatic device enrollment and is now ready to explore a scenario that can deliver business value.

A key part of any cheese-making business is packaging and shipping the cheese to customers. To maximize cost efficiency, Contoso operates an on-premises packaging facility. The workflow is straightforward - packages are assembled for shipping, then placed on a conveyor belt that takes the packages and drops them off in mailing bins. The metric for success is the number of packages leaving the conveyor belt.

The conveyor belt is a critical link in your process, and is monitored for vibration. The conveyor belt has three speeds: stopped, slow, and fast. The number of packages being delivered at slow speed is less than at the faster speed, though the vibration is also less at the slower speed. If the vibration becomes excessive, the conveyor belt has to be stopped and inspected.

There are a number of different types of vibration. Forced vibration is vibration caused by an external force. Such a force as the broken wheel example, or a weighty package placed improperly on the conveyor belt. There's also increasing vibration, which might happen if a design limit is exceeded.

Vibration is typically measured as an acceleration (meters per second squared, m/s^2).

The ultimate goal here is preventive maintenance. Detect that something is wrong, before any damage is caused.

> **Note**: **Preventive maintenance** (sometimes called preventative maintenance) is an equipment maintenance program that schedules maintenance activities to be performed while the equipment is operating normally. The intent of this approach is to avoid unexpected breakdowns that often incur costly disruptions.

It's not always easy to detect abnormal vibration levels. For this reason, you are looking to Azure IoT Hub to detect data anomalies. You plan to have a vibration detection sensor on the conveyor belt, sending continuous telemetry to an IoT Hub. The IoT Hub will use Azure Stream Analytics, and a built-in Machine Learning (ML) model, to give you advance warning of vibration anomalies. You also plan to archive all the telemetry data, just in case it's ever needed.

You decide to build a prototype of the planned system, initially using simulated telemetry.

Exercise 1: Verify Lab Prerequisites

This lab assumes the following resources are available:

Resource Type	Resource Name
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*
Device ID	VibrationSensorId

Exercise 2: Write Code for Vibration Telemetry

The key to monitoring our conveyor belt is the output of vibration telemetry. Vibration is usually measured as an acceleration (m/s^2), although sometimes it's measured in g-forces, where 1 g = 9.81 m/s^2. There are three types of vibration.

- Natural vibration, which is just the frequency a structure tends to oscillate.
- Free vibration, which occurs when the structure is impacted, but then left to oscillate without interference.
- Forced vibration, which occurs when the structure is under some stress.

Forced vibration is the dangerous one for our conveyor belt. Even if it starts at a low level this vibration can build so that the structure fails prematurely. There's less of a case for free vibration in conveyor belt operation. Most machines, as we all know, have a natural vibration.

The code sample that you will build simulates a conveyor belt running at a range of speeds (stopped, slow, fast). The faster the belt is running, the more packages are delivered, but the greater the effects of vibration. We'll add natural vibration, based on a sine wave with some randomization. It's possible our anomaly detection system will falsely identify a spike or dip in this sine wave as an anomaly. We'll then add two forms of forced vibration. The first has the effect of a cyclic increase in vibration (see the images below). And secondly, an increasing vibration, where an additional sine wave is added, starting small but growing.

We assume that our conveyor belt has just one sensor device (our simulated IoT Device). In addition to communicating vibration data, the sensor also pumps out some other data (packages delivered, ambient temperature, and similar metrics). For this lab, the additional values will be sent to a storage archive.

Almost all the coding in this lab will be completed during this exercise. You will be using Visual Studio Code to build the simulator code in C#.

In this exercise, you will:

- build the conveyor belt simulator
- send telemetry messages to the IoT Hub created in the previous unit

Later in this lab you will complete a small amount of SQL coding.

Exercise 3: Create a Message Route to Azure Blob Storage

The architecture of our vibration monitoring system requires data be sent to two destinations: storage and analysis. Azure IoT provides a great method of directing data to the right service, through *message routing*.

In our scenario, we need to create two routes:

- the first route will be to storage for achiving data
- the second route will to an Event Hub for anomaly detection

Since message routes are best built and tested one at a time, this exercise will focus on the storage route. We'll call this route the "logging" route, and it involves digging a few levels deep into the creation of Azure resources. All the features required to build this route are available in the Azure portal.

We will keep the storage route simple, and use Azure Blob storage (though Data Lake storage is also available). The key feature of message routing is the filtering of incoming data. The filter, written in SQL, streams output down the route only when certain conditions are met.

One of the easiest ways to filter data is on a message property, which is why we added these two lines to our code:

```
...
telemetryMessage.Properties.Add("sensorID", "VSTel");
...
loggingMessage.Properties.Add("sensorID", "VSLog");
```

An SQL query embedded into our message route can test the `sensorID` value.

In this exercise, you will create and test the logging route.

Exercise 4: Logging Route Azure Stream Analytics Job

To verify that the logging route is working as expected, we will create a Stream Analytics job that routes logging messages to Blob storage, which can then be validated using Storage Explorer in the Azure Portal.

This will enable us to verify that our route includes the following settings:

- **Name** - vibrationLoggingRoute
- **Data Source** - DeviceMessages
- **Routing query** - sensorID = "VSLog"
- **Endpoint** - vibrationLogEndpoint
- **Enabled** - true

> **Note**: It may seem odd to be routing data to storage, then again sending it to storage through Azure Stream Analytics. In a production scenario, you wouldn't have both paths long-term. Instead, the second path that we're creating here would not exist. We're using it here simply as a way to demonstrate Azure Stream Analytics in an easy-to-validate way in a lab environment.

LAB_AK_08

Lab Scenario

You have developed a device simulator that generates vibration data and other telemetry outputs for a conveyor belt system that takes packages and drops them off in mailing bins. You have built and tested a logging route that sends data to Azure Blob storage.

The second route will be to an Event Hub, because an Event Hub is a convenient input to Stream Analytics. Stream Analytics is a convenient way of handling anomaly detection, such as the excessive vibration we're looking for in our scenario.

This route will be created for the IoT Hub, then added as an input to the Azure Stream Analytics job.

We need to update the job to handle two inputs and two outputs, and a more complex query.

The process of creating the second route follows a similar process to the first, though it diverges at the creation of an endpoint. An Event Hub is chosen as the endpoint for the telemetry route.

In this exercise, you will create an Event Hubs *namespace*. You then have to create an *instance* of the namespace to complete the setting up of an Event Hub. You can then use this instance as the destination for the new message route.

After the route is created, we move on to updating the query.

Make a Call to a Built-in Machine Learning Model

The built-in Machine Learning (ML) function we're going to call is `AnomalyDetection_SpikeAndDip`.

The `AnomalyDetection_SpikeAndDip` function takes a sliding window of data, and examines it for anomalies. The sliding window could be, say, the most recent two minutes of telemetry data. This sliding window keeps up with the flow of telemetry in close to real time. If the size of the sliding window is increased, generally the accuracy of anomaly detection will increase too. As will the latency.

As the flow of data continues, the algorithm establishes a normal range of values, then compares new values against those norms. The result is a score for each value, a percentage that determines the confidence level that the given value is anomalous. Low confidence levels are ignored, the question is what percentage confidence value is acceptable? In our query, we're going to set this tipping point at 95%.

There are always complications, like when there are gaps in the data (the conveyor belt stops for a while, perhaps). The algorithm handles voids in the data by imputing values.

> **Note**: In statistics, imputation is the process of replacing missing data with substituted values. You can learn more about about imputations here.

Spikes and dips in telemetry data are temporary anomalies. However, as we're dealing with sine waves for vibration, we can expect a short period of "normal" values follow a high or low value that triggers an anomaly alert. The operator is looking for a cluster of anomalies occurring in a short time span. Such a cluster indicates something is wrong.

There are other built-in ML models, such as a model for detecting trends. We don't include these models as part of this module, but the student is encouraged to investigate further.

Visualize data using Power BI

Visualizing numerical data, especially volumes of it, is a challenge in itself. How can we alert a human operator of the sequence of anomalies that infer something is wrong?

The solution we use in this module is to use some built-in functionality of Power BI along with the ability of Azure Stream Analytics to send data in a real-time format that Power BI can ingest.

We use the dashboard feature of Power BI to create a number of tiles. One tile contains the actual vibration measurement. Another tile is a gauge, showing from 0.0 to 1.0 the confidence level that the value is an anomaly. A third tile indicates if the 95% confidence level is reached. Finally, the forth tile shows the number of anomalies detected over the past hour. By including time as the x-axis, this tile makes it clear if a clutch of anomalies were detected in short succession as they will be clustered together horizontally.

The fourth tile allows you to compare the anomalies with the red text in the telemetry console window. Is there a cluster of anomalies being detected when forced, or increasing, or both, vibrations are in action?

Let's create the Event Hub, create the second route, update the SQL query, create a Power BI dashboard, and let it all run!

Exercise 1: Sign Up For PowerBI

Power BI can be your personal data analysis and visualization tool, and can also serve as the analytics and decision engine behind group projects, divisions, or entire corporations. Later on in this lab, you will build a dashboard and visualize data using PowerBI. This exercise explains how to sign up for Power BI as an individual.

> **Note:** If you already have a PowerBI subscription, you can skip to the next step.

Exercise 2: Verify Lab Prerequisites

In order to visualize data in a dashboard, we need some real-time telemetry. In this exercise you will ensure the Device Simulator app from the previous lab is running.

Exercise 3: Add Azure Event Hub Route and Anomaly Query

Now that we have telemetry data streaming into the IoT Hub, we're going to add an Azure Event Hub Namespace and Azure Event Hub instance to our solution. Azure Event hubs are ideal for processing streaming data and support live dashboarding scenarios - perfect for passing our vibration data to Power BI.

Exercise 4: Create Real-Time Message Route

Now that we have an Event Hubs Namespace and an Event Hub, we need to pass the telemetry data from the IoT hub to the Event Hub.

Exercise 5: Add Telemetry Route

With this new IoT Hub route in place, and the telemetry data streaming into the Event Hub, we need to update our Stream Analytics job. This job will need to consume the data from the Event Hub, perform analysis using the **AnomalyDetection_SpikeAndDip** ML model and then output the results to Power BI.

Exercise 6: Create a Power BI Dashboard

Now for the final part of the scenario - the actual data visualization. We have updated our job to process the vibration telemetry via the ML model and output the results to Power BI. Within Power BI, we need to create a dashboard with a number of tiles to visualize the results and provide decision support for the operator.

LAB_AK_09

Lab Scenario

Contoso is installing new connected Thermostats to be able to monitor temperature across different cheese caves. You will create an alert to notify facilities manager when a new thermostat has been created.

To create an alert, you will push device created event type to Event Grid when a new thermostat is created in IoT Hub. You will have a Logic Apps instance that will react on this event (on Event Grid) and will send an email to alert a facilities manager device a new device has been created, device ID, and connection state.

Exercise 1: Verify Lab Prerequisites

This lab assumes the following resources are available:

Resource Type	Resource Name
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*

If the resources are unavailable, please execute the **lab-setup.azcli** script before starting the lab.

Exercise 2: Create HTTP Web Hook Logic App that sends an email

Azure Logic Apps is a cloud service that helps you schedule, automate, and orchestrate tasks, business processes, and workflows when you need to integrate apps, data, systems, and services across enterprises or organizations.

In this exercise, you will create a new Azure Logic App that will be triggered via an HTTP Web Hook, then send an email using an Outlook.com email address.

Exercise 3: Configure Azure IoT Hub Event Subscription

Azure IoT Hub integrates with Azure Event Grid so that you can send event notifications to other services and trigger downstream processes. You can configure business applications to listen for IoT Hub events so that you can react to critical events in a reliable, scalable, and secure manner. For example, build an application that updates a database, creates a work ticket, and delivers an email notification every time a new IoT device is registered to your IoT hub.

In this exercise, you will create an Event Subscription within Azure IoT Hub to setup Event Grid integration that will trigger a Logic App to send an alert email.

LAB_AK_10

Lab Scenario

Contoso's asset condition tracking revealed a spike in temperature for a specific asset, and we want to find the root cause. We want to understand what happened: we correlate the IoT device's sensor data from transportation trucks and planes sensors. The temperature in one of the trucks rose and created the heat spike in one of the transport box equipped with an IoT device with temperature and humidity sensors. Your team will need to be able to explore data in near real time and perform root-cause analysis to help with improving delivery processes to ensure products are kept in the best condition.

You will add Time Series Insights to the solution to quickly store, visualize, and query large amounts of time series data, that are generated by the IoT devices in the trucks, planes, and the transport boxes themselves to visualize changes over time.

Exercise 1: Verify Lab Prerequisites

This lab assumes the following resources are available:

Resource Type	Resource Name
Resource Group	`AZ-220-RG`
IoT Hub	`AZ-220-HUB-{YOUR-ID}`
Device ID	`TruckDevice`
Device ID	`AirplaneDevice`
Device ID	`ContainerDevice`

Exercise 2: Setup Time Series Insights

Azure Time Series Insights (TSI) is an end-to-end platform-as-a-service (PaaS) offering used to collect, process, store, analyze, and query data from Internet of Things (IoT) solutions at scale. TSI is designed for ad hoc data exploration and operational analysis of data that's highly contextualized and optimized for time series.

In this exercise, you will setup Time Series Insights (TSI) integration with Azure IoT Hub.

Exercise 3: Run Simulated IoT Devices

In this exercise, you will run the simulated devices so they starts sending telemetry events to Azure IoT Hub.

Exercise 4: View Time Series Insights Explorer

In this exercise, you will be introduced to working with time series data using the Time Series Insights (TSI) Explorer.

LAB_AK_11

Lab Scenario

Contoso has cheese producing factories worldwide. Factories are equipped with production lines have multiple machines to create their cheeses. At the moment they have IoT devices connected to each machine that streams sensor data to Azure and processes all data in the cloud. Due to the large amount of data being collected and urgent time response needed on some of the machines, Contoso's wants to add a gateway device to bring the intelligence to the edge for processing data to the only send important data to the cloud. Plus, be able to process data and react quickly even if the local network is poor.

You will be setting up a new IoT Edge device that can monitor temperature of one of the machines and deploying Stream Analytics module to calculate the average temperature and send an alert to the device to act quickly.

Exercise 1: Verify Lab Prerequisites

[tbd]

Exercise 2: Deploy Azure IoT Edge enabled Linux VM

In this exercise, you will deploy an Ubuntu Server VM with Azure IoT Edge runtime support from the Azure Marketplace.

Exercise 3: Create IoT Edge Device Identity in IoT Hub using Azure CLI

In this exercise, you will create a new IoT Edge Device Identity within Azure IoT Hub using the Azure CLI.

Exercise 4: Connect IoT Edge Device to IoT Hub

In this exercise, you will connect the IoT Edge Device to Azure IoT Hub.

Exercise 5: Add Edge Module to Edge Device

In this exercise, you will add a Simulated Temperature Sensor as a custom IoT Edge Module, and deploy it to run on the IoT Edge Device.

Exercise 6: Deploy Azure Stream Analytics as IoT Edge Module

Now that the tempSensor module is deployed and running on the IoT Edge device, we can add a Stream Analytics module that can process messages on the IoT Edge device before sending them on to the IoT Hub.

LAB_AK_12

Lab Scenario

This lab is theoretical and will walk you through how an IoT Edge device can be used as a gateway.

There are three patterns for using an IoT Edge device as a gateway: transparent, protocol translation, and identity translation:

Transparent – Devices that theoretically could connect to IoT Hub can connect to a gateway device instead. The downstream devices have their own IoT Hub identities and are using any of the MQTT, AMQP, or HTTP protocols. The gateway simply passes communications between the devices and IoT Hub. The devices are unaware that they are communicating with the cloud via a gateway, and a user interacting with the devices in IoT Hub is unaware of the intermediate gateway device. Thus, the gateway is transparent. Refer to Create a transparent gateway for specifics on using an IoT Edge device as a transparent gateway.

Protocol translation – Also known as an opaque gateway pattern, devices that do not support MQTT, AMQP, or HTTP can use a gateway device to send data to IoT Hub on their behalf. The gateway understands the protocol used by the downstream devices, and is the only device that has an identity in IoT Hub. All information looks like it is coming from one device, the gateway. Downstream devices must embed additional identifying information in their messages if cloud applications want to analyze the data on a per-device basis. Additionally, IoT Hub primitives like twins and methods are only available for the gateway device, not downstream devices.

Identity translation - Devices that cannot connect to IoT Hub can connect to a gateway device, instead. The gateway provides IoT Hub identity and protocol translation on behalf of the downstream devices. The gateway is smart enough to understand the protocol used by the downstream devices, provide them identity, and translate IoT Hub primitives. Downstream devices appear in IoT Hub as first-class devices with twins and methods. A user can interact with the devices in IoT Hub and is unaware of the intermediate gateway device.

Exercise 1: Verify Lab Prerequisites

[tbd]

Exercise 2: Deploy Azure IoT Edge enabled Linux VM

In this exercise, you will deploy an Ubuntu Server VM with Azure IoT Edge runtime support from the Azure Marketplace.

Exercise 3: Generate and Configure IoT Edge Device CA Certificates

In this exercise, you will generate test certificates using Linux. You will do this on the `AZ-220-VM-EDGEGW-{YOUR-ID}` Virtual Machine using a helper script contained within the `Azure/IoTEdge` GitHub project.

Exercise 4: Create IoT Edge Device Identity in IoT Hub using Azure Portal

In this exercise, you will create a new IoT Edge Device identity in Azure IoT Hub for the IoT Edge Transparent Gateway.

Exercise 5: Setup IoT Edge Gateway Hostname

In this exercise, you will configure the DNS name for Public IP Address of the **AZ-220-VM-EDGEGW-*{YOUR-ID}***, and configure that DNS name as the `hostname` of the IoT Edge Gateway device.

Exercise 6: Connect IoT Edge Gateway Device to IoT Hub

In this exercise, you will connect the IoT Edge Device to Azure IoT Hub.

Exercise 7: Open IoT Edge Gateway Device Ports for Communication

In this exercise, you will configure the Network Security Group (NSG) that secures access to the Azure IoT Edge Gateway from the Internet. The necessary ports for MQTT, AMQP, and HTTPS communications need to be opened so the downstream IoT device(s) can communicate with the gateway.

For the Azure IoT Edge Gateway to function, at least one of the IoT Edge hub's supported protocols must be open for inbound traffic from downstream devices. The supported protocols are MQTT, AMQP, and HTTPS.

The IoT communication protocols supported by Azure IoT Edge have the following port mappings:

Protocol	Port Number
MQTT	8883
AMQP	5671
HTTPS MQTT + WS (Websocket) AMQP + WS (Websocket)	443

The IoT communication protocol chosen for your devices will need to have the corresponding port opened for the firewall that secures the IoT Edge Gateway device. In the case of this lab, an Azure Network Security Group (NSG) is used to secure the IoT Edge Gateway, so Inbound security rules for the NSG will be opened on these ports.

In a production scenario, you will only want to open the minimum number of ports for your devices to communicate. If you are using MQTT, then only open port 8883 for inbound communications. Opening additional ports will introduce addition security attack vectors that attackers could take exploit. It is a security best practice to only open the minimum number of ports necessary for your solution.

Exercise 8: Create Downstream Device Identity in IoT Hub

In this exercise, you will create a new IoT Device identity in Azure IoT Hub for the downstream IoT device. This device identity will be configured so that the Azure IoT Edge Gateway is a parent device for this downstream device.

Exercise 9: Connect Downstream Device to IoT Edge Gateway

In this exercise, you will configure a pre-built Downstream Device to connect to the IoT Edge Gateway.

Exercise 10: Verify Event Flow

In this exercise, you will use the Azure CLI to monitor the events being sent to Azure IoT Hub from the downstream IoT Device through the IoT Edge Gateway. This will validate that everything is working correctly.

LAB_AK_13

Lab Scenario

Contoso's warehouse moves inventory that is ready to be packed for delivery on a conveyor belt.

In order to make sure the correct amount of products have been packed, you will add a simple module to count objects detected on the belt by another object detection module (simulated) on the same IoT Edge device. We will show how to create a custom module that does object counting.

This lab includes the following prerequisites for the development machine (lab host environment - VM or PC):

- Visual Studio Code with the following extensions installed:
  - Azure IoT Tools by Microsoft
  - C# by Microsoft
  - Docker
- Docker Community Edition installed on development machine, with Docker Client version 18.03.0 or later
  - Download Docker Desktop for Mac and Windows

    > **Important**: Because of removal of Azure Container Registry support for TLS before TLS version 1.2 on January 13, 2020, you must be on Docker Client 18.03.0 or later.

Exercise 1: Verify Lab Prerequisites

[tbd]

Exercise 2: Install Azure IoT EdgeHub Dev Tool

In this exercise, you will will install the Azure IoT EdgeHub Dev Tool.

Exercise 3: Create Azure Container Registry

Azure Container Registry provides storage of private Docker images for container deployments. The service is a managed, private Docker registry service based on the open-source Docker Registry 2.0. Azure Container Registry is used to store and manage your private Docker container images.

In this exercise, you will use the Azure portal to create a new Azure Container Registry resource.

Exercise 4: Create Custom Edge Module in C#

In this exercise, you will create an Azure IoT Edge Solution that contains a custom Azure IoT Edge Module written in C#.

Exercise 5: Debug in Attach Mode with IoT Edge Simulator

In this exercise, you will build and run a custom IoT Edge Module solution using the IoT Edge Simulator from within Visual Studio Code.

Exercise 6: Deploy IoT Edge Solution

In this exercise, you will build and publish the custom IoT Edge Module into the Azure Container Registry (ACR) service. Once published to ACR, the custom module will then be made available to be deployed to any IoT Edge Device.

LAB_AK_14

Lab Scenario

The conveyor belt system monitors vibrations, telemetry, and counts objects. We want our system to be resilient to network outages and also optimize the bulk upload of telemetry data at specific times in the day (load balancing network usage). We will configure IoT Edge to support offline in case network drops and we will look into storing telemetry from sensors locally and configure for regular syncs at given times.

You will learn the different scenarios where IoT Edge device is on an enterprise network (needs proxy settings) or needs extended offline capabilities.

Exercise 1: Verify Lab Prerequisites

This lab assumes the following resources are available:

Resource Type	**Resource Name**
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*
IoT Device	SimulatedThermostat

Exercise 2: Deploy Azure IoT Edge enabled Linux VM

In this unit you will deploy an Ubuntu Server VM with Azure IoT Edge runtime support from the Azure Marketplace. In previous labs you have created the VM using the Azure Portal. This time around, we will create the VM using the Azure CLI.

Exercise 3: Setup IoT Edge Parent with Child IoT Devices

In this exercise, you will register an IoT Edge Device within Azure IoT Hub, and an IoT Device that is configured in a child relationship. This will enable the scenario to the the child IoT Device send messages through the IoT Edge Device as a gateway before communicating to the Azure IoT Hub in the cloud.

The use of Parent / Child relationships including an IoT Edge Gateway (the parent) and other IoT Devices (the child or leaf devices) enables the use of Offline capabilities within an Azure IoT solution.

The following diagram shows the relationship between the Azure IoT Edge Device as the parent, and a child device:

 IoT Edge Parent with Child Device Diagram

In this scenario, the child device connects to, and authenticate against the parent IoT Edge Device using their Azure IoT Hub credentials. Once authenticated, the child IoT Device sends messages to the Edge Hub (`$edgeHub`) on the IoT Edge Device. Once messages reach the parent IoT Edge Device, the IoT Edge Modules and Routing will handle the messages as configured; including sending the messages to the Azure IoT Hub when connected.

When the parent IoT Edge Device is disconnected (or loses connection to the Azure IoT Hub) it will automatically store all device messages to the IoT Edge Device. Once the connection is restored, the IoT Edge Device will resume connectivity and send any stored messages to Azure IoT Hub. Messages stored on the IoT Edge Device may expire according to the Time-to-Live (TTL) configurations for the device; which defaults to store messages for up to `7200` seconds (two hours).

Exercise 4: Configure IoT Edge Device as Gateway

In this exercise, you will configure the Azure IoT Edge on Ubuntu virtual machine that was created previously to be an IoT Edge Transparent Gateway device. The configuration will be handled by a helper script that is part of this unit to make the process quicker.

Exercise 5: Open IoT Edge Gateway Device Inbound Ports using Azure CLI

In this exercise, you will use the Azure CLI to configure the Network Security Group (NSG) that secures access to the Azure IoT Edge Gateway from the Internet. The necessary ports for MQTT, AMQP, and HTTPS communications need to be opened so the downstream IoT device(s) can communicate with the gateway.

For the Azure IoT Edge Gateway to communicate with Child IoT Devices, the TCP/IP port for the devices protocol must be open for **Inbound** communication. The device could use one of three supported protocols to communicate with the IoT Gateway.

These are the TCP/IP port numbers for the supported protocols:

Protocol	Port Number
MQTT	8883
AMQP	5671
HTTPS MQTT + WS (Websocket) AMQP + WS (Websocket)	443

Exercise 6: Configure IoT Edge Device Time-to-Live and Message Storage

In this exercise, you will configure the message Time-to-Live (TTL) of the Edge Hub module on the Azure IoT Edge Gateway device to be longer than the default. You will also configure the storage location on the IoT Edge Device where the messages are to be stored.

The default value of `7200` (2 hours) is not long enough for a device or solution that may need to function in Offline mode for extended periods of time. For the device and solution to operate for more extended periods

of being disconnected, you will configure the Time-to-Live (TTL) property of the IoT Edge Hub module to the value of 1,209,600 seconds, for a 2 week TTL period.

The Module Twin for the IoT Edge Hub is called `$edgeHub` and is used to coordinate communications between the IoT Edge Hub running on the device and the Azure IoT Hub service. Within the Desired Properties for the Module Twin, the `storeAndForwardConfiguration.timeToLiveSecs` property specifies the time in seconds that IoT Edge Hub keeps messages when in a state disconnected from routing endpoints, like Azure IoT Hub.

The `timeToLiveSecs` property for the Edge Hub can be specified in the Deployment Manifest on a specific device as part of a single-device or at-scale deployment. In this unit, you will use the Azure Portal user interface for Azure IoT Hub to modify the `timeToLiveSecs` property for the Edge Hub (`$edgeHub`) module on the single IoT Edge Gateway device.

Exercise 7: Connect Child IoT Device to IoT Edge Gateway

In this exercise, you will configure the downstream, child IoT Devices to connect to IoT Hub using their configured Symmetric Keys. The devices will be configured to connect to IoT Hub and the parent IoT Edge Device using a Connection String that contains the Symmetric Key; in addition to the Gateway Hostname for the Parent IoT Edge Device.

The process to authenticate regular IoT devices to IoT Hub with symmetric keys also applies to downstream (or child / leaf) devices. The only difference is that you need to add a pointer to the Gateway Device to route the connection or, in offline scenarios, to handle the authentication on behalf of IoT Hub.

In a previous unit, you created the IoT Device Identities in Azure IoT Hub. You copied the **Connection String** for the IoT Device. Alternatively, the Connection String can be accessed with the Azure portal for the Device ID of the device within Azure IoT Hub.

Exercise 8: Test Device Connectivity and Offline Support

In this exercise, you will monitor events from the **ChildIoTDevice** are being sent to Azure IoT Hub through the **IoTEdgeGateway** IoT Edge Transparent Gateway. You will then interrupt connectivity between the **IoTEdgeGateway** and Azure IoT Hub to see that telemetry is still sent from the child IoT Device to the IoT Edge Gateway. After this, you will resume connectivity with Azure IoT Hub and monitor that the IoT Edge Gateway resumes sending telemetry to Azure IoT Hub.

LAB_AK_15

Lab Scenario

Suppose you manage a gourmet cheese making company in a southern location. The company is proud of its cheese, and is careful to maintain the perfect temperature and humidity of a natural cave that is used to age the cheese. There are sensors in the cave that report on the temperature and humidity. A remote operator can set a fan to new settings if needed, to maintain the perfect environment for the aging cheese. The fan can heat and cool, and humidify and de-humidify.

Caves are used to mature cheese, their constant temperature, humidity, and air flow make them nearly ideal for the process. Not to mention the cachet of having your cheese products mature in a natural cave, instead

of a constructed cellar. Something to put on your product labels!

The accepted ideal temperature for aging cheese is 50 degrees fahrenheit (10 degrees centigrade), with up to 5 degrees (2.78 degrees C) either side of this being acceptable. Humidity is also important. Measured in percentage of maximum saturation, a humidity of between 75 and 95 percent is considered fine. We'll set 85 percent as the ideal, with a 10 percent variation as acceptable. These values apply to most cheeses. To achieve specific results, such as a certain condition of the rind, cheese makers will adjust these values for some of the time during aging.

In a southern location, a natural cave near the surface might have an ambient temperature of around 70 degrees. The cave might also have a relative humidity of close to 100 percent, because of water seeping through the roof. These high numbers aren't perfect conditions for aging cheese. At a more northerly location, the ambient temperature of a natural cave can be the ideal of 50 degrees. Because of our location, we need some Azure IoT intervention!

Sending and Receiving Telemetry

The frequency of telemetry output is an important factor. A temperature sensor in a refrigeration unit may only have to report every minute, or less. An acceleration sensor on an aircraft may have to report at least every second.

An IoT device may contain one or more sensors, and have some computational power. There may be LED lights, and even a small screen, on the IoT device. However, the device isn't intended for direct use by a human operator. An IoT device is designed to receive its instructions from the cloud.

Control a Cheese Cave Device

In this module, we assume the IoT cheese cave monitoring device has temperature and humidity sensors. The device has a fan capable of both cooling or heating, and humidifying or de-humidifying. Every few seconds, the device sends current temperature and humidity values to the IoT Hub. This rapid frequency is unrealistic for a cheese cave (maybe every 15 minutes, or less, would be granular enough), except during code development when we want rapid activity!

For this lab, we assume that the fan can be in one of three states: on, off, and failed. The fan is initialized to the off state. In a later unit, the fan is turned on by use of a direct method.

Another feature of our IoT device is that it can accept desired values from the IoT Hub. The device can then adjust its fan to target these desired values. These values are coded in this module using a feature called device twins. Desired values will override any default settings for the device.

Coding the Sample

The coding in this module is broken down into three parts: sending and receiving telemetry, sending and receiving a direct method, and managing device twins.

Let's start by writing two apps: one for the device to send telemetry, and one back-end service to run in the cloud, to receive the telemetry. You'll be able to select your preferred language (Node.js or C#), and development environment (Visual Studio Code, or Visual Studio).

Exercise 1: Create a custom Azure IoT Hub, using the IoT Hub portal

This lab assumes the following resources are available:

Resource Type	Resource Name
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*
IoT Device	CheeseCaveID

Exercise 2: Write Code to Send and Receive Telemetry

At the end of this unit, you'll be sending and receiving telemetry.

Exercise 2: Create a Second App to Receive Telemetry

Now we have a device pumping out telemetry, we need to listen for that telemetry with a back-end app, also connected to our IoT Hub.

Exercise 3: Write Code to Invoke a Direct Method

In this unit, we'll add code to the device app for a direct method to turn on the fan. Next, we add code to the back-end service app to invoke this direct method.

Calls from the back-end app to invoke direct methods can include multiple parameters as part of the payload. Direct methods are typically used to turn features of the device off and on, or specify settings for the device.

Exercise 4: Write Code for Device Twins

In this exercise, we'll add some code to both the device app and back-end service app, to show device twin synchronization in operation.

As a reminder, a device twin contains four types of information:

- **Tags**: information on the device that isn't visible to the device.
- **Desired properties**: the desired settings specified by the back-end app.
- **Reported properties**: the reported values of the settings on the device.
- **Device identity properties**: read-only information identifying the device.

Device twins are designed for querying, and automatically synchronizing, with the real IoT Hub device. The device twin can be queried, at any time, by the back-end app. This query can return the current state information for the device. Getting this data doesn't involve a call to the device, as the device and twin will have synchronized automatically. Much of the functionality of device twins is provided by Azure IoT, so not much code needs to be written to make use of them.

There is some overlap between the functionality of device twins and direct methods. We could set desired properties using direct methods, which might seem an intuitive way of doing things. However, using direct methods would require the back-end app to record those settings explicitly, if they ever needed to be accessed. Using device twins, this information is stored and maintained by default.

LAB_AK_16

Lab Scenario

Suppose you manage a company that offers a solution to maintain and monitor cheese caves' temperature and humidity at optimal levels. You have been working with gourmet cheese making companies for a long time and established long term trust with these customers who value the quality of your product.

Your solution consists in sensors and a climate system installed in the cave that report in real time on the temperature and humidity and an online portal customers can use to monitor and remotely operate their devices to adapt the temperature and humidity to the type of cheese they stored in their cave or to fine tune the environment for perfectly aging their cheese.

Your company is always enhancing the software running on the devices to better adapt to your customers different cheeses and diverse types of rooms they use to store their cheese. In addition to the features updates, you also want to make sure the devices deployed at customers locations have the latest security patches to ensure privacy and prevent hackers to take control of the system. In order to do this, you need to keep the devices up to date by remotely updating their firmware.

Exercise 1: Create an Azure IoT Hub and a Device ID

This lab assumes the following resources are available:

Resource Type	**Resource Name**
Resource Group	AZ-220-RG
IoT Hub	AZ-220-HUB-*{YOUR-ID}*
IoT Device	SimulatedSolutionThermostat

Exercise 2: Write code to simulate device that implements firmware update

At the end of this task, you'll have a device simulator awaiting for a firmware update request from IoT Hub.

Before getting started with your first firmware update on an IoT device, take a minute to review what it actually means to implement such an operation and how Azure IoT Hub helps making the process.

What does updating an IoT device's firmware imply?

IoT devices most often are powered by optimized operating systems or even sometimes running code directly on the silicon (without the need for an actual operating system). In order to update the software running on this kind of devices the most common method is to flash a new version of the entire software package, including the OS as well as the apps running on it (called firmware).

Because each device has a specific purpose, its firmware is also very specific and optimized for the purpose of the device as well as the constrained resources available.

The process for updating a firmware is also something that can be very specific to the hardware itself and to the way the hardware manufacturer does things. This means that a part of the firmware update process is not generic and you will need to work with your device manufacturer to get the details of the firmware update process (unless you are developing your own hardware which means you probably know what the firmware update process).

While firmware updates can be and used to applied manually on devices, this is no longer possible considering the rapid growth in scale of IoT solutions. Firmware updates are now more commonly done over-the-air (OTA) with deployments of new firmware managed remotely from the cloud.

There is a set of common denominators to all over-the-air firmware updates for IoT devices:

1. Firmware versions are uniquely identified
2. Firmware comes in a binary file format that the device will need to acquire from an online source
3. Firmware is locally stored is some form of physical storage (ROM memory, hard drive,...)
4. Device manufacturer provide a description of the required operations on the device to update the firmware.

Azure IoT Hub Automatic Device Management

Azure IoT Hub offers advanced support for implementing device management operations on a single and on collections of devices. The Automatic Device Management feature allows to simply configure a set of operations, trigger them and then monitor their execution.

In this exercise, you will create a simple simulator that will manage the device twin desired properties changes and will trigger a local process simulating a firmware update. The overall process would be exactly the same for a real device with the exception of the actual steps for the local firmware update. You will then use the Azure Portal to configure and execute a firmware update for a single device. IoT Hub will use the device twin properties to transfer the configuration change request to the device and monitor the progress

Exercise 3: Test firmware update on a single device

In this exercise, we will use the Azure portal to create a new device management configuration and apply it to our single simulated device.

LAB_AK_17

Lab Scenario

Our asset tracking solution is getting bigger, and provisioning devices one by one (even through DPS) cannot scale. We want to use DPS to enroll many devices automatically and securely using x.509 certificate authentication. Within our solution, we will use sensors to track our assets being transported. Each time a sensor is added in a transportation box, it will auto provision through DPS. We want to have a metric for the warehouse manager of how many boxes were "tagged" and need to count the Device Connected events from IoT Hub.

In this lab, you will setup a Group Enrollment within Device Provisioning Service (DPS) using a Root CA x.509 certificate chain. You will configure the linked IoT Hub to using monitoring to track the number of connected devices and telemetry messages sent, as well as send connection events to a log. Additionally you will create an alert that will be triggered based upon the average number of devices connected. You will the configure 10 simulated IoT Devices that will authenticate with DPS using a Device CA Certificate generated on the Root CA Certificate chain. The IoT Devices will be configured to send telemetry to the the IoT Hub.

Exercise 1: Verify Lab Prerequisites

[tbd]

Exercise 2: Set Up and Use Metrics and Diagnostic Logs with an IoT Hub

If you have an IoT Hub solution running in production, you want to set up some metrics and enable diagnostic logs. Then if a problem occurs, you have data to look at that will help you diagnose the problem and fix it more quickly. In this lab, you'll see how to enable the diagnostic logs, and how to check them for errors. You'll also set up some metrics to watch, and alerts that fire when the metrics hit a certain boundary.

For example, you could have an e-mail sent to you when the number of connected devices exceed a certain threshold, or when the number of messages used gets close to the quota of messages allowed per day for the IoT Hub.

Exercise 3: Enable Logging

Azure Resource logs are platform logs emitted by Azure resources that describe their internal operation. All resource logs share a common top-level schema with the flexibility for each service to emit unique properties for their own events.

Exercise 4: Configure an Alert

Now let us create an alert. Alerts proactively notify you when important conditions are found in your monitoring data. They allow you to identify and address issues before the users of your system notice them. In our asset tracking scenario, we use sensors to track our assets being transported. Each time a sensor is added in a transportation box, it will auto provision through DPS. We want to have a metric for the warehouse manager of how many boxes were "tagged" and need to count the Device Connected events from IoT Hub.

In this task we are going to add an alert that will inform the warehouse manager when 5 or more devices have connected.

Exercise 5: Simulating the Sensors

As part of the asset-tracking scenario, we need to have devices that simulate the tags that will be used to track the assets during transportation. As each device is activated, it should use automatic device provisioning to connect to the Iot solution and start sending telemetry. In order to automatically connect, each device will need its own X509 certificate that is part of a chain to the root certificate used to create a group enrollment.

In this task, we will verify the existing environment, perform any necessary setup, generate 10 device certificates, and configure a console application that will simulate the 10 devices.

Exercise 6: Simulate Devices

In this task we will be generating X509 certificates from the root certifcate. We will then use these certificates in a console application that will simulate 10 devices connecting to DPS and sending telemetry to an IoT Hub.

Exercise 7: Review Metrics, Alerts and Archive

[TBD]

LAB_AK_18

MISING

LAB_AK_19

Lab Scenario

Contoso has built all their solutions with security in mind. However, they want to see how they can better get a unified view of security across all of their on-premises and cloud workloads, including their Azure IoT solutions. Plus, when onboarding new devices, we want to apply security policies across workloads (Leaf devices, Microsoft Edge devices, IoT Hub) to ensure compliance with security standards and improved security posture.

Contoso is adding a brand new assembly line outfitted with new IoT devices to help with the increasing shipping and packing demands for new orders. You want to ensure that any new devices are secured and also want to be able to see security recommendations to continue improving your solution's security in your full end-to-end IoT solution. You will start investigating using Azure IoT Center for IoT for your solution.

Exercise 1: Verify Lab Prerequisites

[tbd]

Exercise 2: Create an IoT Hub using the Azure portal

In this task, you will use the Azure portal to create an IoT Hub resource.

Exercise 3: Enable Azure Security Center for IoT Hub

The Azure Security Center for IoT Hub unifies security management and enables end-to-end threat detection and analysis across hybrid cloud workloads and your Azure IoT solution.

Exercise 4: Create and Register a New Device

[TBD]

Exercise 5: Create a Security Module Twin

Azure Security Center for IoT offers full integration with your existing IoT device management platform, enabling you to manage your device security status as well as make use of existing device control capabilities. Azure Security Center for IoT integration is achieved by making use of the IoT Hub twin mechanism.

Azure Security Center for IoT makes use of the module twin mechanism and maintains a security module twin named azureiotsecurity for each of your devices.

The security module twin holds all the information relevant to device security for each of your devices.

To make full use of Azure Security Center for IoT features, you'll need to create, configure and use these security module twins for your new IoT Edge device.

The security module twin **azureiotsecurity** can be created in two ways:

- Module batch script - automatically creates module twin for new devices or devices without a module twin using the default configuration.
- Manually editing each module twin individually with specific configurations for each device.

In this task, you will be creating a security module twin manually.

Exercise 6: Deploy Azure Security Center for IoT C# Security Agent

Azure Security Center for IoT provides reference architecture for security agents that log, process, aggregate, and send security data through IoT Hub. You will be adding a security agent for C# to deploy on your simulated device (Linux VM). There are C and C# based agents. C agents are recommended for devices with more restricted or minimal device resources.

Security agents support the following features:

- Collect raw security events from the underlying Operating System (Linux, Windows). To learn more about available security data collectors, see Azure Security Center for IoT agent configuration.
- Aggregate raw security events into messages sent through IoT Hub.
- Authenticate with existing device identity, or a dedicated module identity. See Security agent authentication methods to learn more.
- Configure remotely through use of the **azureiotsecurity** module twin. To learn more, see Configure an Azure Security Center for IoT agent.

Exercise 7: Configure Solution Management

Azure Security Center for IoT provides comprehensive end-to-end security for Azure-based IoT solutions.

With Azure Security Center for IoT, you can monitor your entire IoT solution in one dashboard, surfacing all of your IoT devices, IoT platforms and back-end resources in Azure.

Once enabled on your IoT Hub, Azure Security Center for IoT automatically identifies other Azure services, also connected to your IoT Hub and related to your IoT solution.

In addition to automatic relationship detection, you can also pick and choose which other Azure resource groups to tag as part of your IoT solution. Your selections allow you to add entire subscriptions, resource groups, or single resources.

LAB_AK_20

Lab Scenario

Azure IoT Central enables the easy monitoring and management of a fleet of remote devices.

Azure IoT Central encompasses a range of underlying technologies that work great, but can be complicated to implement when many technologies are needed. These technologies include Azure IoT Hub, the Azure Device Provisioning System (DPS), Azure Maps, Azure Time Series Insights, Azure IoT Edge, and others. It's only necessary to use these technologies directly, if more granularity is needed than available through IoT Central.

One of the purposes of this lab is to help you decide if there's enough features in IoT Central to support the scenarios you are likely to need. So, let's investigate what IoT Central can do with a fun and involved scenario.

Contoso operates a fleet of refrigerated trucks. You've a number of customers within a city, and a base that you operate from. You command each truck to take its contents and deliver it to any one customer. However, the cooling system may fail on any one of your trucks, and if the contents does start to melt, you'll need the option of instructing the truck to return to base, and then dump the contents. Alternatively, you can deliver the contents to another customer who might be nearer to the truck when you become aware the contents are melting.

In order to make these decisions, you'll need an up-to-date picture of all that is going on with your trucks. You'll need to know the location of each truck on a map, the state of the cooling system, and the state of the contents.

IoT Central provides all you need to handle this scenario.

Exercise 1: Create a Custom IoT Central app

[TBD]

Exercise 2: Create Device Template

The data communicated between a remote device, and IoT Central, is specified in a *device template*. The device template encapsulates all the details of the data, so that both the device and IoT Central have all they need to make sense of the communication.

In this Lab, you'll create a device template for a refrigerated truck.

Exercise 3: Monitor Simulated Device

You'll first create a dashboard showing all the capabilities of the device template. Next, you'll create a real device, and record the connection settings needed for the remote device app.

Exercise 4: Create a free Azure Maps account

If you do not already have an Azure Maps account, you'll need to create one.

Exercise 5: Create a Programming Project for a Real Device

In this task, you are going to create a programming project to simulate a sensor device in a refrigerated truck. This simulation enables you to test the code long before requiring a real truck!

IoT Central treats this simulation as "real" because the communication code between the device app and the IoT Central app is the same for a real truck. In other words, if you do run a refrigerated truck company, you would start with simulated code similar to the code in this task. After this code works to your satisfaction, the simulation-specific code would be replaced with code that receives sensor data. This limited update makes writing the following code a valuable experience.

Exercise 6: Test Your IoT Central Device

Working through this task is an exciting time in IoT Central development! Finally, you get to check whether all the moving parts you've created work together.

Exercise 7: Create multiple devices

In this task, we consider what steps would be necessary to add multiple trucks to our system.