AZ-220T00-A - Microsoft Azure IoT Developer

### About this course

**Course Description**

This course provides students with the skills and knowledge required to successfully create and maintain the cloud and edge portions of an Azure IoT solution. The course includes full coverage of the core Azure IoT services such as IoT Hub, Device Provisioning Services, Azure Stream Analytics, Time Series Insights, and more. In addition to the focus on Azure PaaS services, the course includes sections on IoT Edge, device management, monitoring and troubleshooting, security concerns, and Azure IoT Central.

**Level**: Intermediate

**Audience**

An Azure IoT Developer is responsible for implementing and then maintaining the cloud and edge portions of an Azure IoT solution. In addition to configuring and maintaining devices by using Azure IoT services and other Microsoft tools, the IoT Developer also sets up the physical devices and is responsible for maintaining the devices throughout the life cycle.

The IoT Developer implements designs for IoT solutions, including device topology, connectivity, debugging and security. For Edge device scenarios, the IoT Developer also deploys compute/containers and configures device networking, which could include various edge gateway implementations. The IoT Developer implements designs for solutions to manage data pipelines, including monitoring and data transformation as it relates to IoT. The IoT Developer works with data engineers and other stakeholders to ensure successful business integration.

IoT Developers should have a good understanding of Azure services, including data storage options, data analysis, data processing, and the Azure IoT PaaS versus SaaS options. IoT Developers should have basic programming skills in at least one Azure-supported language, including C#, Node.js, C, Python, or Java.

**Prerequisites**

Software Development Experience: Software development experience is a prerequisite for this course, but no specific software language is required, and the experience does not need to be at a professional level.

Data Processing Experience: General understanding of data storage and data processing is a recommended but not required.

Cloud Solution Awareness: Students should have a basic understanding of PaaS, SaaS, and IaaS implementations. Microsoft Azure Fundamentals (AZ-900), or equivalent skills, is recommended.

**Expected learning**

* Create, configure, and manage an Azure IoT hub.
* Provision devices by using IoT Hub and DPS, including provisioning at scale.
* Establish secure 2-way communication between devices and IoT Hub.
* Implement message processing by using IoT Hub routing and Azure Stream Analytics.
* Configure the connection to Time Series Insights and support business integration requirements.
* Implement IoT Edge scenarios using marketplace modules and various edge gateway patterns.
* Implement IoT Edge scenarios that require developing and deploying custom modules and containers.
* Implement device management using device twins and direct methods.
* Implement solution monitoring, logging, and diagnostics testing.
* Recognize and address security concerns and implement Azure Security Center for IoT.
* Build an IoT Solution by using Azure IoT Central and recognize SaaS opportunities for IoT.

### Course syllabus

The course content includes a mix of content, demonstrations, hands-on labs, reference links, and module review questions.

M1: Introduction to IoT and Azure IoT Services

In this module, students will begin by examining the business considerations for various IoT implementations and reviewing how the Azure IoT Reference Architecture supports IoT solutions. This module also provides students with an overview of the Azure services commonly used in an IoT solution and provides an introduction to the Azure portal.

Lessons:

* Business Opportunities for IoT
* Introduction to IoT Solution Architecture
* IoT Hardware and Cloud Services
* Lab Scenarios for this Course

Labs:

* Getting Started with Azure
* Setting Started with Azure IoT Services

M2: Devices and Device Communication

In this module, students will take a closer look at the Azure IoT Hub service and will learn how to configure secure two-way communication between IoT hub and devices. Students will also be introduced to IoT Hub features such as Device Twins and IoT Hub Endpoints that will be explored in more depth as the course continues.

Lessons:

* IoT Hub and Devices
* IoT Developer Tools
* Device Configuration and Communication

Labs:

* Setup the Development Environment
* Connect IoT Device to Azure

M3: Device Provisioning at Scale

In this module, students will focus on device provisioning and how to configure and manage the Azure Device Provisioning Service. Students will learn about the enrollment process, auto-provisioning and re-provisioning, disenrollment, and how to implement various attestation mechanisms.

Lessons:

* Device Provisioning Service Terms and Concepts
* Configure and Manage the Device Provisioning Service
* Device Provisioning Tasks

Labs:

* Individual Enrollment of Devices in DPS
* Automatic Enrollment of Devices in DPS

M4: Message Processing and Analytics

In this module, students will examine how IoT Hub and other Azure services can be used to process messages. Students will begin with an investigation of how to configure message and event routing and how to implement routing to built-in and custom endpoints. Students will learn about some of the Azure storage options that are common for IoT solutions. To round out his module, students will implement Azure Stream Analytics and queries for a number of ASA patterns.

Lessons:

* Messages and Message Processing
* Data Storage Options
* Azure Stream Analytics

Labs:

* Device Message Routing
* Filtering and Aggregating Message Data

M5: Insights and Business Integration

In this module, students will learn about the Azure services and other Microsoft tools that can be used to generate business insights and enable business integration. Students will implement Azure Logic Apps and Event Grid, and they will configure the connection and data transformations for data visualization tools such as Time Series Insights and Power BI.

Lessons:

* Business Integration for IoT Solutions
* Data Visualization with Time Series Insights
* Data Visualization with Power BI

Labs:

* Integrate IoT Hub with Event Grid
* Explore and Analyze Time Stamped Data with Time Series Insights

M6: Azure IoT Edge Deployment Process

In this module, students will learn how to deploy a module to an Azure IoT Edge device. Students will also learn how to configure and use an IoT Edge device as a gateway device.

Lessons:

* Introduction to Azure IoT Edge
* Edge Deployment Process
* Edge Gateway Devices

Labs:

* Introduction to IoT Edge
* Set Up an IoT Edge Gateway

M7: Azure IoT Edge Modules and Containers

In this module, students will develop and deploy custom edge modules, and will implement support for an offline scenario that relies on local storage. Students will use Visual Studio Code to build custom modules as containers using a supported container engine.

Lessons:

* Develop Custom Edge Modules
* Offline and Local Storage

Labs:

* Develop, Deploy, and Debug a Custom Module on Azure IoT Edge
* Run an IoT Edge Device in Restricted Network and Offline

M8: Device Management

In this module, students will learn how to implement device management for their IoT solution. Students will develop device management solutions that use devoice twins and solutions that use direct methods.

Lessons:

* Introduction to IoT Device Management
* Manage IoT and IoT Edge Devices
* Device Management at Scale

Labs:

* Remotely Monitor and Control Devices with Azure IoT Hub
* Automatic Device Management

M9: Solution Testing, Diagnostics, and Logging

In this module, students will configure logging and diagnostic tools that help developers to test their IoT solution. Students will use IoT Hub and Azure Monitor to configure alerts and track conditions such as device connection state that can be used to troubleshoot issues.

Lessons:

* Monitoring and Logging
* Troubleshooting

Labs:

* Configure Metrics and Logs in Azure IoT Hub
* Monitor and Debug Connection Failures

M10: Azure Security Center and IoT Security Considerations

In this module, students will examine the security considerations that apply to an IoT solution. Students will begin by investigating security as it applies to the solution architecture and best practices, and then look at how Azure Security Center for IoT supports device deployment and IoT Hub integration. Students then use Azure Security Center for IoT Agents to enhance the security of their solution.

Lessons:

* Security Fundamentals for IoT Solutions
* Introduction to Azure Security Center for IoT
* Enhance Protection with Azure Security Center for IoT Agents

Labs:

* Implementing Azure Security Center for IoT

M11: Build an IoT Solution with IoT Central

In this module, students will learn how configure and implement Azure IoT Central as a SaaS solution for IoT. Students will begin with a high-level investigation of IoT Central and how it works. With a basic understanding of IoT central establish, students will move on to creating and managing device templates, and then managing devices in their IoT Central application.

Lessons:

* Introduction to IoT Central
* Create and Manage Device Templates
* Manage Devices in Azure IoT Central

Labs:

* Get Started with Azure IoT Central
* Implementing IoT Solutions with Azure IoT Central

### AZ-220 certification exam

The AZ-220, [Microsoft Azure IoT Developer](https://docs.microsoft.com/en-us/learn/certifications/exams/az-220), certification exam is geared towards the person who is responsible for the implementation and the coding required to create and maintain the cloud and edge portion of an IoT solution.

The exam includes six study areas. The percentages indicate the relative weight of each area on the exam. The higher the percentage, the more questions the exam will contain.

| **AZ-220 Study Areas** | **Weights** |
| --- | --- |
| Implement the IoT Solution Infrastructure | 15-20% |
| Provision and Manage Devices | 20-25% |
| Implement Edge | 15-20% |
| Process and Manage Data | 15-20% |
| Monitor, Troubleshoot, and Optimize IoT Solutions | 15-20% |
| Implement Security | 15-20% |

✔️ Learn more about the [certification](https://docs.microsoft.com/en-us/learn/certifications/azure-iot-developer-specialty).

### Resources

There are a lot of resources to help you and the student learn about Azure. We recommend you bookmark these pages.

* [Azure IoT Fundamentals](https://docs.microsoft.com/en-us/azure/iot-fundamentals/). The Azure IoT documentation page provides links to lots of resources.
* [Azure IoT Products](https://azure.microsoft.com/en-us/product-categories/iot/). The Azure IoT Products page provides links to product pages.
* [Azure IoT Reference Architecture](https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot/). This reference architecture shows a recommended architecture for IoT applications on Azure using PaaS (platform-as-a-service) components.
* [Building IoT solutions with Azure: a developer’s guide](https://discover.microsoft.com/azure-iot-building-solutions-dev-guide/). This guide provides an overview of Azure services that address key IoT solution requirements, as well as a step-by-step progression you can use to build proficiency and move toward a fully functioning solution quickly and easily.

### Obtaining and Redeeming Your Azure Pass

Your instructor will provide guidance on obtaining and redeeming your Azure pass.

Introduction to IoT and Azure IoT Services

### Subsystems of an IoT Architecture

During the past ten years, the Internet of Things has been defined in many ways by many people. One simple definition is as follows:

*The Internet of Things is a network of Internet connected devices that communicate embedded sensor data to the cloud for centralized processing*.

As this simple definition above points out, an IoT solution involves two essential components:

1. A device-side (made up of individual devices) that acts as a data source
2. A cloud-side that gathers data and provides resources for analyzing it

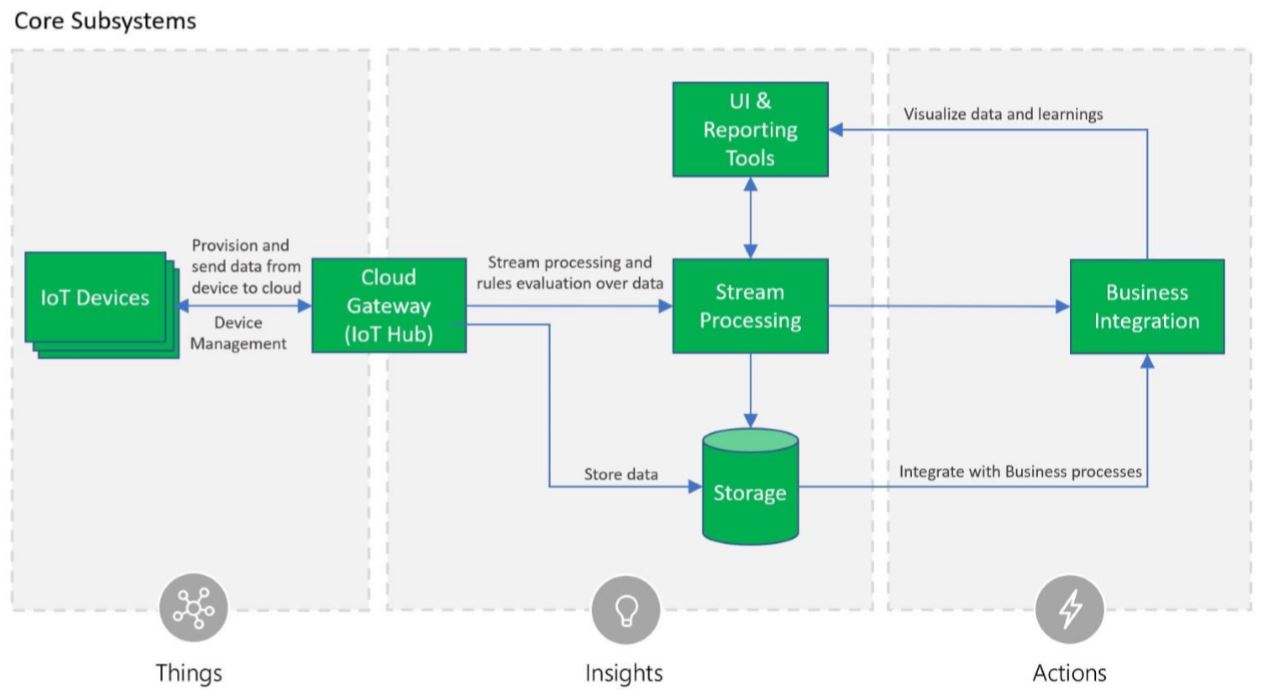
Of course, once your business begins planning an IoT solution you find that both the device-side and the cloud-side involve complex implementations that provide hundreds of required features, and even the communication between the device and cloud requires secure communication protocols.

#### Core Subsystems

At its core, an IoT solution architecture consists of the following subsystems:

1. Devices that have the ability to securely register with the cloud, and connectivity options for sending and receiving data (between devices and the cloud).
2. A cloud gateway service, or hub, to securely accept data from devices, and to provide device management capabilities.
3. Stream processors that consume that data, integrate with business processes, and place the data into storage.
4. A user interface to visualize telemetry data and facilitate device management.

These core subsystems can be aligned to a Things/Insights/Actions model, as shown below in a high-level view of the Azure IoT Reference Architecture.



**IoT Devices**: The physical devices where our data originates.

**Cloud Gateway**: The Cloud Gateway provides a cloud hub for secure connectivity, telemetry and event ingestion and device management (including command and control) capabilities.

**User Interface and Reporting**: The user interface for an IoT application can be delivered on a wide array of device types, in native applications, and browsers.

**Stream Processing**: Processes large streams of data records and evaluates rules for those streams.

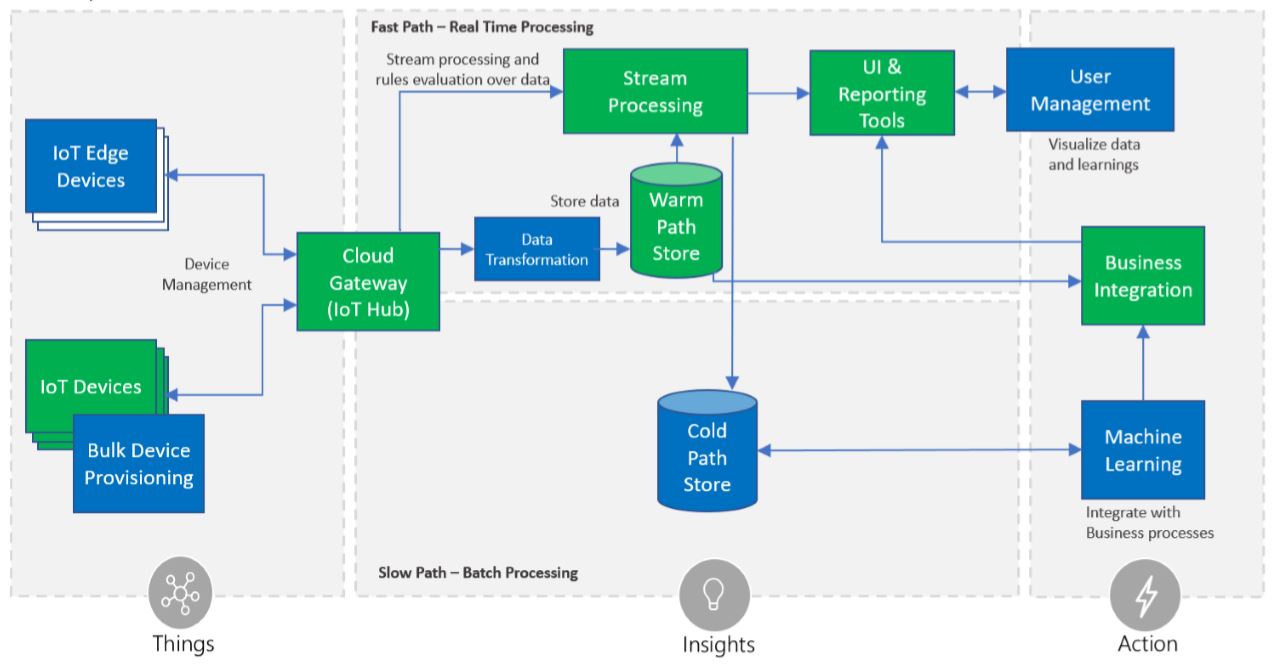
**Storage**: Storage can be divided into warm path (data that is required to be available for reporting and visualization immediately from devices), and cold path (data that is stored longer term and used for batch processing).

**Business Process Integration**: Facilitates executing actions based on insights garnered from device telemetry data during stream processing. Integration could include storage of informational messages, alarms, sending email or SMS, integration with CRM, and more.

#### Optional Subsystems

In addition to the core subsystems many IoT applications will include subsystems for:

1. telemetry data transformation which allows restructuring, combination, or transformation of telemetry data sent from devices
2. machine learning which allows predictive algorithms to be executed over historical telemetry data, enabling scenarios such as predictive maintenance, and
3. user management which allows splitting of functionality amongst different roles and users.



**Data transformation**: The manipulation or aggregation of the telemetry stream either before or after it is received by the cloud gateway service (the IoT Hub). Manipulation can include protocol transformation (e.g. converting binary streamed data to JSON), combining data points, and more.

**Machine Learning (ML) Subsystem**: Enables systems to learn from data and experiences and to act without being explicitly programmed. Scenarios such as predictive maintenance are enabled through ML.

**User Management Subsystem**: Allows specification of different capabilities for users and groups to perform actions on devices (e.g. command and control such as upgrading firmware for a device) and capabilities for users in applications.

**Edge Devices**: These devices serve an active role in managing access and information flow. They may assist in device provisioning, data filtering, batching and aggregation, buffering of data, protocol translation, event rules processing, and more.

**Bulk Provisioning**: Facilitates provisioning of large numbers of devices.

### Data Flow and Processing

As data is delivered to the IoT backend, it is important to understand how the flow of data processing may vary. Depending on scenarios and applications, data records can flow through different stages, combined in different order, and often processed by concurrent, parallel tasks.

These stages can be classified in four categories - storage, routing, analysis and action/display:

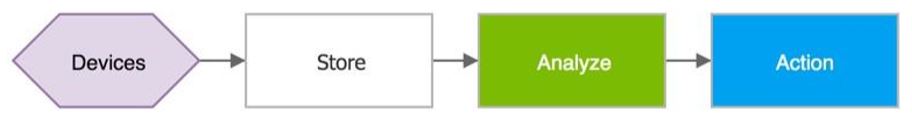
* Storage includes [in-memory caches](https://en.wikipedia.org/wiki/CPU_cache), temporary queues and permanent archives (e.g. a database).
* Routing allows sending data records to one or more storage endpoints, analysis processes, and actions. Routing makes decisions on what data should go which target and when.
* Analysis is used to run data records through a set of conditions and can produce different output data records. For instance, input telemetry data encoded in one format may return output telemetry [encoded](https://en.wikipedia.org/wiki/Code) in another format.
* Original input data records and analysis output records are typically stored and available to display, and may trigger actions such as emails, instant messages, incident tickets, CRM tasks, device commands, etc.

These processes can be combined in simple graphs, for instance to display raw telemetry received in real time, or more complex graphs executing multiple and advanced tasks, for example updating dashboards, triggering alarms, and starting business integration processes, etc.

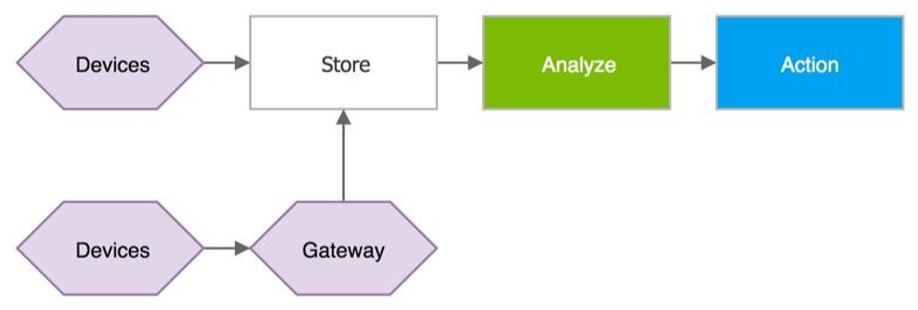
For example, the following graph represents a simple scenario in which devices send telemetry records which are temporarily stored in Azure IoT Hub, and then are immediately displayed on graph on screen for visualization:



The following graph represents another common scenario, in which devices send telemetry, store it short term in Azure IoT Hub, shortly after analyzing the data to detect anomalies, then trigger actions such as an email, SMS text, instant message, etc.:



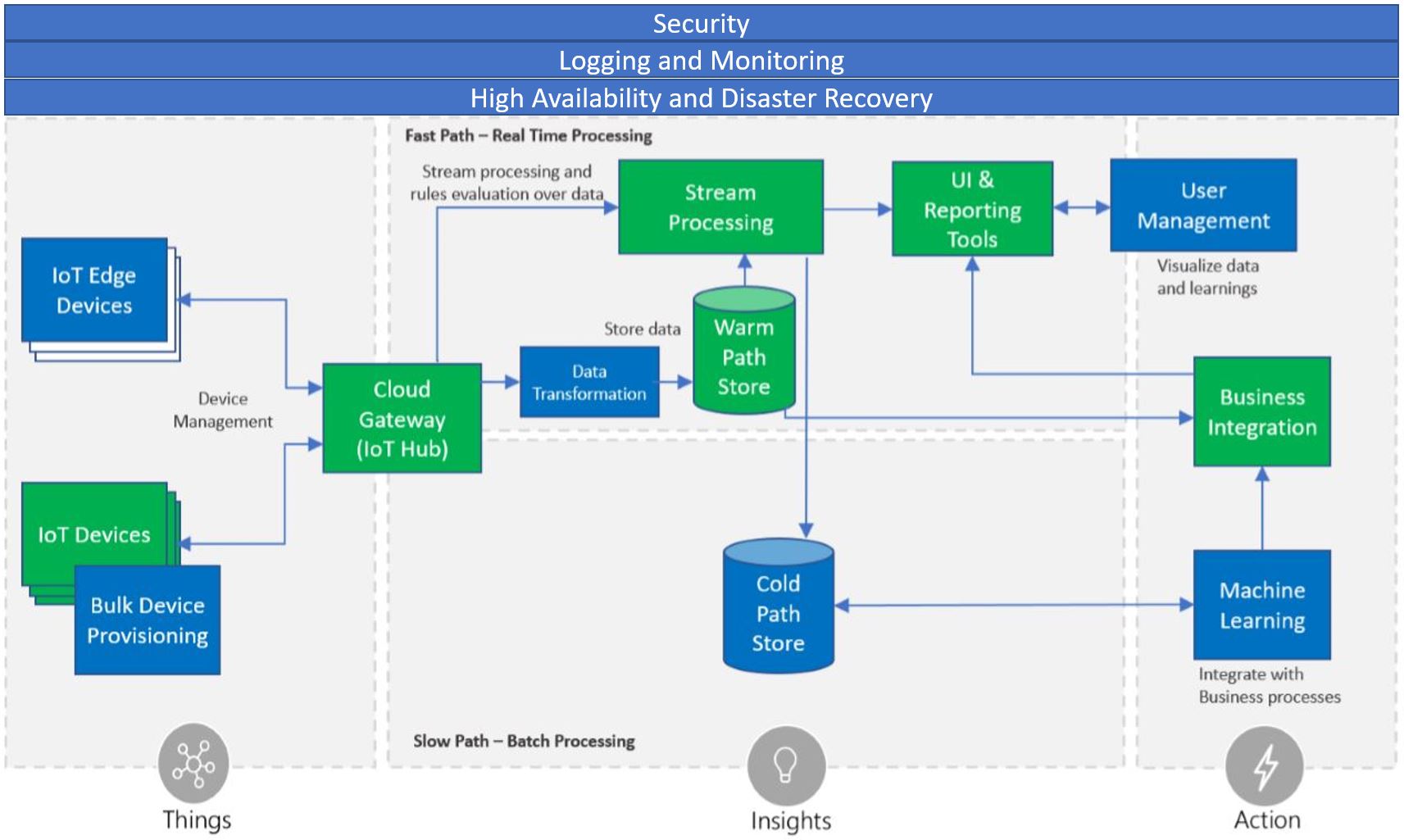
IoT architectures can also support multiple systems that can accept and do something with data. For instance, some telemetry storage and/or analysis may occur on premise, within devices and field/edge gateways. In other scenarios, [protocol translations](https://en.wikipedia.org/wiki/Protocol_converter) may be required to connect constrained devices to the cloud. While the resulting graph is more complex, the logical building blocks are the same:



### Cross-Cutting Architectural Needs

Cross-cutting needs, or concerns, are aspects of the solution that affect the other subsystem within a solution and which cannot be separated from these subsystems.

There are multiple cross-cutting needs for IoT solutions that are critical for success, including: 1) security requirements; including user management and auditing, device connectivity, in-transit telemetry, and at rest security, 2) logging and monitoring for an IoT cloud application is critical for determining health and for troubleshooting failures both for individual subsystems and the application as a whole, and 3) high availability and disaster recovery which is used to rapidly recover from systemic failures.



**Security**: Security is a critical consideration in each of the subsystems. Protecting IoT solutions requires secure provisioning of devices, secure connectivity between devices, edge devices, and the cloud, secure access to the backend solutions, and secure data protection in the cloud during processing and storage (encryption at rest).

**Logging and monitoring**: Logging actions and monitoring activity associated with your IoT solution is critical for determining system uptime and troubleshooting failures.

**High availability and disaster recovery**: High availability and disaster recovery (HA/DR) focuses on ensuring an IoT system is always available, including from failures resulting from disasters. The technology used in IoT subsystems have different failover and cross-region support characteristics. For IoT applications, this can result in requiring hosting of duplicate services and duplicating application data across regions depending on acceptable failover downtime and data loss.

### IoT Hardware Components

When thinking about building an IoT solution, perhaps the first area of consideration is what hardware you will need (or already have). This is partly driven by the fact that data is the main driver behind implementing many IoT solutions, so figuring out what data you want to collect and how you want to collect it has a primary place in your architecture.

The hardware implemented in an IoT solution includes the network infrastructure that is used to connect devices and provide secure data communication to the cloud as well as hardware that is used broker supporting communication with other hardware and cloud services.

#### IP-enabled IoT Devices

An IP-enabled device is, simply, a device that can establish a a connection to a network (for many IoT devices, this means the Internet) and have a unique identity on that network. “IP” stands for "[Internet protocol](https://en.wikipedia.org/wiki/Internet_Protocol)" and defines the way messages are delivered over a network. A message in networking terms is just a packet of information and single packet could deliver part of a text message or a video file. Most data that is transferred over the Internet uses this communication protocol.

In terms of IoT, an IP-enabled device is one that can connect directly to a network like the Internet, and use the connection to transmit and receive data. Consumer device examples we commonly think of are the home automation devices like doorbells and thermostats that use an Internet connection to communicate with a central server. But industrial-grade IoT devices can be IP-enabled as well. IP-enabled devices use specialized hardware to enable this functionality.

As you might expect, people deploy IP-enabled devices in scenarios where data needs to be collected, delivered, and analyzed in real-time, near real-time, or periodically.

#### Non-IP Enabled Devices

A device does not need to be directly IP-enabled in order to be a part of an IoT solution. Some devices don't use IP to connect to other parts of an IoT solution but can use other protocols. These devices don't connect to the Internet *per se* but their messages are routed to the Internet via other hardware like a field gateway (IoT Edge Device) which we'll discuss below. Non IP-enabled devices can use industry-specific protocols (such as CoAP5, OPC),and short-range communication technologies (such as Bluetooth, ZigBee) to connect to other hardware.

Continuing our discussion of consumer devices, when implementing a home security system that includes window sensors and a controllable door lock, the sensors and devices may be communicating locally within the home and use a centralized field gateway device (that is IP-enabled and connected to the Internet) in order to communicate with a cloud service.

Devices of this type can be useful in scenarios where data from a number of devices needs to be aggregated, cleaned-up, and possibly even analyzed before being sent to a cloud service. Since IP-enabled devices typically take more resources, low-powered or resource- (or space-) constrained devices can use protocols with lower resource consumption requirements that transmit to a device that doesn't have these constraints.

#### Sensors

A sensor is a circuit (or device) that collects a specific type of data about the physical environment. As IoT continues to evolve, the list of available sensors is likely to grow with it. In the meantime, if the sensor that you need doesn't exist, there are communities that will help you build your own sensors.

A **smart sensor** according to [the website](https://internetofthingsagenda.techtarget.com/definition/smart-sensor)*IoT Agenda* is “a device that takes input from the physical environment and uses built-in compute resources to perform predefined functions upon detection of specific input and then process data before passing it on.” That is, the device itself processes the data to some degree before sending it to the next node in the IoT architecture.

Sensors can either be directly embedded within an IoT device, or implemented as an external piece of hardware that connects to the IoT device through a defined interface. Examples of simple sensor measurements include: Temperature, Humidity, Distance, and Light

#### IoT Edge Devices and Field Gateways

A field gateway is a specialized device-appliance or general-purpose software that acts as a communication enabler and, potentially, as a local device control system and device data processing hub. A field gateway can perform local processing and control functions toward the devices; on the other side it can filter or aggregate the device telemetry and thus reduce the amount of data being transferred to the cloud backend.

A field gateway’s scope includes the field gateway itself and all devices that are attached to it. As the name implies, field gateways act outside dedicated data processing facilities and are usually collocated with the devices.

A field gateway is different from a mere traffic router in that it has an active role in managing access and information flow. It is an application-addressed entity and network connection or session terminal. For example, gateways in this context may assist in device provisioning, data filtering, batching and aggregation, buffering of data, protocol translation, and event rules processing. NAT devices or firewalls, in contrast, do not qualify as field gateways since they are not explicit connection or session terminals, but rather route (or deny) connections or sessions made through them.

#### Other Hardware

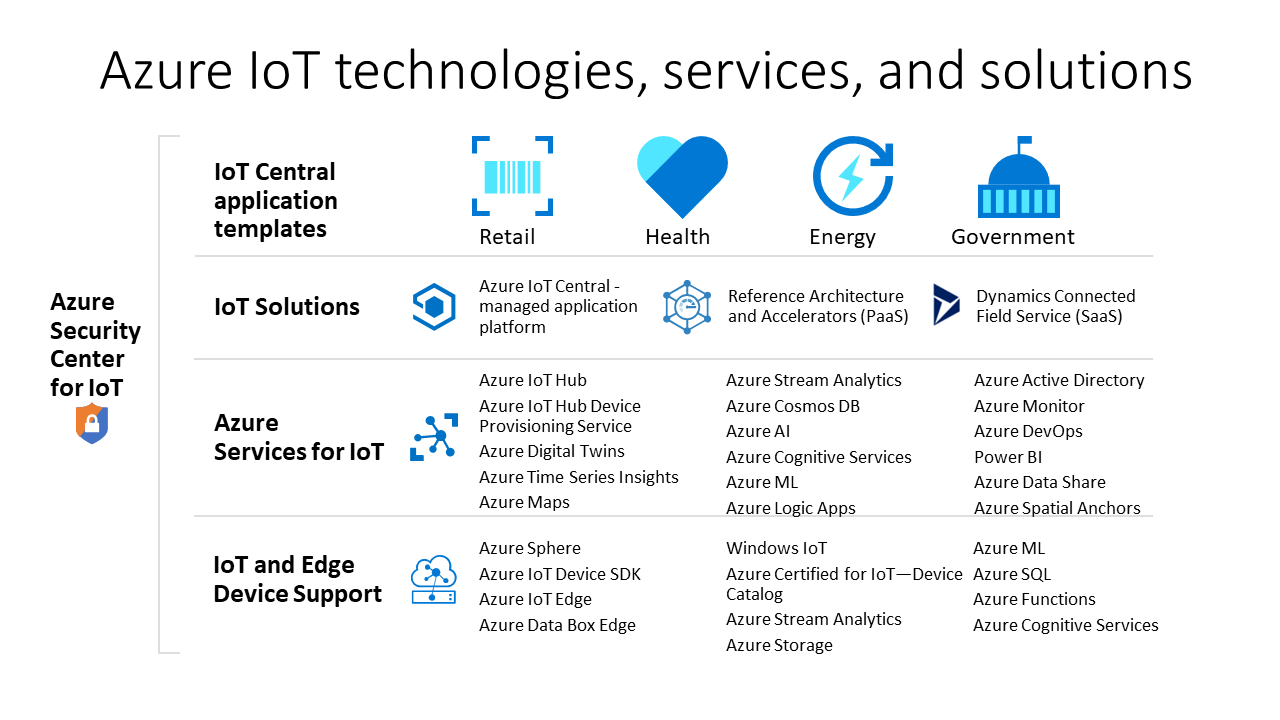
General networking hardware and specialized protocol gateway devices can also play a significant role in the device-side infrastructure.

### Azure IoT Services and Technologies

Microsoft has built a portfolio that supports the needs of all customers, enabling everyone to access the benefits of digital transformation.

Azure IoT technologies and services provide you with options to create a wide variety of IoT solutions that enable digital transformation for your organization. For example, you can:

* Use Azure IoT Central, a managed IoT application platform, to build and deploy a secure, enterprise-grade IoT solution. IoT Central features a collection of industry-specific application templates, such as retail and healthcare, to accelerate your solution development process.
* Extend the open-source code base for an Azure IoT solution accelerator to implement a common IoT scenario such as remote monitoring or predictive maintenance.
* Use Azure IoT platform services such as Azure IoT Hub and the Azure IoT device SDKs to build a custom IoT solution from scratch.



#### Managed and Prebuilt Solutions

Get started quickly with the solution accelerators or managed application platform offerings. Choose from preconfigured solution accelerators that enable common IoT scenarios, such as remote monitoring, predictive maintenance, and connected factory, to create a fully customizable solution. Or use Azure IoT Central, a fully managed, end-to-end solution that enables powerful IoT scenarios without requiring cloud-solution expertise.

##### Azure IoT Central (SaaS)

The IoT Central application platform reduces the burden and cost of developing, managing, and maintaining enterprise-grade IoT solutions. IoT Central's customizable web UI in lets you monitor device conditions, create rules, and manage millions of devices and their data throughout their life cycle. The API surface within IoT Central gives you programmatic access to configure and interact with your IoT solution.

Azure IoT Central is a fully managed application platform that you can use to create custom IoT solutions. IoT Central uses application templates to create solutions. There are templates for generic solutions and for specific industries such as energy, healthcare, government, and retail. IoT Central application templates let you deploy an IoT Central application in minutes that you can then customize with themes, dashboards, and views.

Choose devices from the Azure Certified for IoT device catalog to quickly connect to your solution. Use the IoT Central web UI to monitor and manage your devices to keep them healthy and connected. Use connectors and APIs to integrate your IoT Central application with other business applications.

As a fully managed application platform, IoT Central has a simple, predictable pricing model.

##### Azure IoT solution accelerators (PaaS)

Azure IoT solution accelerators are customizable PaaS solutions that provide a high level of control over your IoT solution. If your business is implementing IoT for connected operations or has specific customization requirements for connected products, Azure IoT solution accelerators provide the control you need.

Organizations with a large number of devices or device models, and manufacturers seeking connected factory solutions, are examples of companies that can benefit from IoT solution accelerators.

#### Custom Solutions (PaaS)

With the most comprehensive IoT portfolio of platform services, Platform-as-a-Service (PaaS) technologies that span the Azure platform enable you to easily create, customize, and control all aspects of your IoT solution. Establish bi-directional communications with billions of IoT devices and manage your IoT devices at scale. Then integrate your IoT device data with other platform services, such as Azure Cosmos DB and Azure Time Series Insights, to enhance insights across your solution.

##### Device support

Get started on your IoT project with confidence by leveraging Azure IoT Starter Kits or choosing from hundreds of Certified for IoT devices in the device catalog. All devices are platform-agnostic and tested to connect seamlessly to IoT Hub. Connect all your devices to Azure IoT using the open-source device SDKs. The SDKs support multiple operating systems, such as Linux, Windows, and real-time operating systems, as well as multiple programming languages, such as C, Node.js, Java, .NET, and Python.

##### IoT Hub

Azure IoT Hub is a fully managed service that enables reliable and secure bidirectional communications between millions of IoT devices and a solution back end. 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, enabling customers to provision millions of devices in a secure and scalable manner.

##### IoT Edge

Azure IoT Edge is an IoT service. This service is meant for customers who want to analyze data on devices, a.k.a. “at the edge.” By moving parts of your workload to the edge, you will experience reduced latency and have the option for off-line scenarios.

##### Spatial Intelligence

Azure Digital Twins is an IoT service that enables you to create a model of a physical environment. It provides a spatial intelligence graph to model the relationships between people, spaces, and devices. By corelating data across the digital and physical world you can create contextually aware solutions.

##### Data and analytics

Take advantage of an array of Azure data and analytics PaaS offerings in your IoT solution, from bringing cloud intelligence to the edge with Azure Machine Learning, to storing IoT device data in a cost-effective way with Azure Data Lake, to visualizing huge amount of data from IoT devices with Azure Time Series Insights.

### Introduction to IoT Device Software

Like any other piece of complex electronic hardware that we use today, IoT devices need the ability to run code in order to be useful. But because IoT devices tend to be small and resource-constrained, the environment in which code runs will vary in functionality, memory footprint, and feature set. Devices also need to be programmed–given the instructions they need to do the tasks that engineers need them to do. There are many vendors developing runtime environments, operating systems for resource constrained devices, and programming tools. The choice you make for any given solution will be the product of a number of factors including:

* Availability of the software you need
* Compatibility of the software with the devices you've chosen
* Compatibility with other software/cloud systems in your solution
* Reputation and longevity of the software provider
* The commitment of the software provider to update the operating system and tools to address security issues, bugs, and new features.
* Security and privacy requirements

Of course, your solution may involve devices with a variety of operating systems and development environments. But the more you add to your solution, the more complex development and maintenance becomes so it pays to be mindful of the software choices you make and the implications of each one during the architectural phase of the project.

#### Device Operating Systems

As we mentioned above, there are a lot of options for device operating systems. In this section, we'll survey a few of the more popular onces to get a sense of the features and options available on them.

| **OS** | **Type** | **Description** |
| --- | --- | --- |
| [Windows IoT Core](https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core) | Managed | Windows IoT Core is a version of Windows 10 that is optimized for smaller devices with or without a display that run on both ARM and x86/x64 devices. |
| [Ubuntu Core](https://www.ubuntu.com/core) | Open Source | "Ubuntu Core uses the same kernel, libraries and system software as classic Ubuntu. You can develop snaps on your Ubuntu PC just like any other application. The difference is that it’s been built for the Internet of Things." |
| [Riot](https://www.riot-os.org/) | Open Source | "RIOT supports most low-power IoT devices and microcontroller architectures (32-bit, 16-bit, 8-bit). RIOT aims to implement all relevant open standards supporting an Internet of Things that is connected, secure, durable & privacy-friendly." |
| [QNX](https://www.qnx.com/) | Managed | "Through sophisticated software solutions BlackBerry® QNX helps ensure the proper operation of embedded systems that require mission-critical and life-critical operations. Since 1980, BlackBerry QNX has established itself as a trusted partner for companies building vehicles, medical devices, heavy machinery, power and energy, robotics, and industrial automation systems that are required to be safety-certified, extremely reliable, and highly secure." |
| [Android Automotive](https://en.wikipedia.org/wiki/Android_Automotive) | Managed | "Android Automotive is a variation of Google's Android operating system, tailored for its use in vehicle dashboards." |

And there are many others. Which OS you choose will largely depend on what you need to accomplish, your architectural design, development tools and developer resources and similar considerations. Most vendors and organizations (even if they're not open source) provide free “trial” options so you can spend some time with the software and tools as you work through the options. Be sure to look at development tools as well as being able to write software for your devices should be as much of a consideration as the operating system itself. Let's look at the development environments next.

#### Programming Languages

When it comes to programming devices, the operating system running on the device may determine what languages can be used to program it. Many modern hardware devices can support multiple languages and board engineers may develop specific flavors of hardware to support various languages. Microsoft's IoT core, for example, supports most languages that Windows develop in general supports including C#, C++, and JavaScript. Ubuntu Core, on the other hand, supports Python, Ruby, and Node.js.

This makes choosing a programming platform complex and attempting to even outline the matrix of options here would not present an adequate picture. Instead, we can suggest how to approach the decision-making process when it comes to a programming platform. These suggestions build upon the strategies we've been seeing throughout this course so some items will be familiar and other items will be new.

1. **Determine what data you want to collect**. As we've seen throughout the course, your IoT architecture generally will begin by figuring out what problems you're solving and this, most times, will be characterized in terms of the data you want to collect. This relates to programming languages because the data you want to collect will impact the devices you choose and the programming language(s) you choose will have to work with the device infrastructure you deploy.
2. **Think about your development team**. When considering the programming languages you want to use in your solution, you will need to consider whether you want to use talent you already have at your disposal, bring on new resources, or use a blend of both. If your current software development team knows C# but doesn't know Python, choosing a platform that supports C# as a programming language will most likely enable you to get to market quicker than having either to train existing talent in another language or bring on new talent that knows an alternate language.
3. **Think about your broader software environment**. Similar to item 2 above, when you think about what software platform you want to use, it can be helpful to think about the development environment across your business group or enterprise. By using a language that already is deployed in other areas of your business can make tasks like resource balancing, code sharing, source control, hiring, and similar factors more efficient.
4. **Choose a device or devices platform**. Once you've figured out what data you want to collect and have thought about your larger ecosystem, you'll be better informed when it comes to choosing a device platform. As we've said in other lessons, you may need more than one device platform so choosing platforms that are the most compatible with items 1-4 above will give you a more efficient overall environment in which to develop your solution.

These are not the only factors to consider of course. Items like cost can have a big impact on choices but sometimes using a device platform that is slightly higher in cost per item can pay off in the long run if the platform supports a language platform that will mean more efficiency in the long run.

**What about the cloud?** It may go without saying, but we'll say it anyway: an essential component of the software platform when making a platform decision is the cloud services you'll use to support your software and hardware. We'll talk a bit more about this in the next topic but we think it's important to call out here as an essential aspect of the decision-making process.

#### Software Development Kits

Before we leave this topic, let's briefly discuss IoT software development kits (SDKs) as a means by which you can more quickly create an environment to build your solution. If you've never used an [SDK](https://en.wikipedia.org/wiki/Software_development_kit) before, these kits can accelerate the software development process by providing the developer with all the necessary tools, software packages, and integration software necessary to build a complete solution. While the degree to which any given SDK does this will vary with providers and companies, a good SDK will provide many if not most of the software-related tools needed to build a solution.

### Cloud Service Components of an IoT Solution

In this topic, you'll learn:

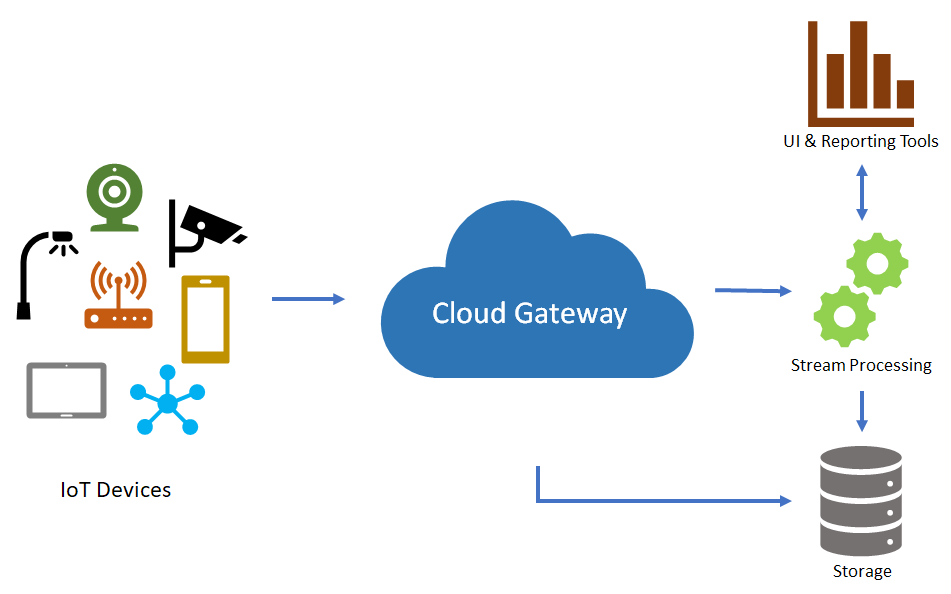
* About cloud-based gateways and storage options
* About cloud-based analytics and data visualization
* About how to use machine learning in IoT solutions

As we noted in the last topic, the cloud services you choose is an essential part of your overall solution. In fact, the cloud services used in your solution constitutes the ‘I’ in IoT. There are options from many of the larger companies participating in this space as well as offerings from startups and medium-sized businesses. You can explore the individual offerings on your own. In this topic, we'll look at categories of services these companies offer to give you an idea of how cloud services fit into an overall IoT architecture.

#### Cloud Gateways

In an earlier topic in this lesson, we looked at, briefly, the concept of a field gateway–a piece of hardware that brokers communication between IoT devices and cloud services. Cloud gateways do more than broker communication. They provide a set of services that devices can run either locally or in the cloud. Cloud gateways can provide workloads such as (among others):

* Authentication and authorization
* Message brokering
* Data storage and filtering
* Data analytics
* Functions (discrete code blocks that perform specific tasks)



#### Data Storage Options

Given the centrality of data in an IoT solution, figuring out the right cloud-based data storage and retrieval options ranks high on the list in terms of importance. IoT devices can generate enormous amounts of data very quickly and storing high volumes of data in the cloud can not become expensive but also unwieldy–you have to be able to do something with the data and too much of it can make analytics and decision-making harder.

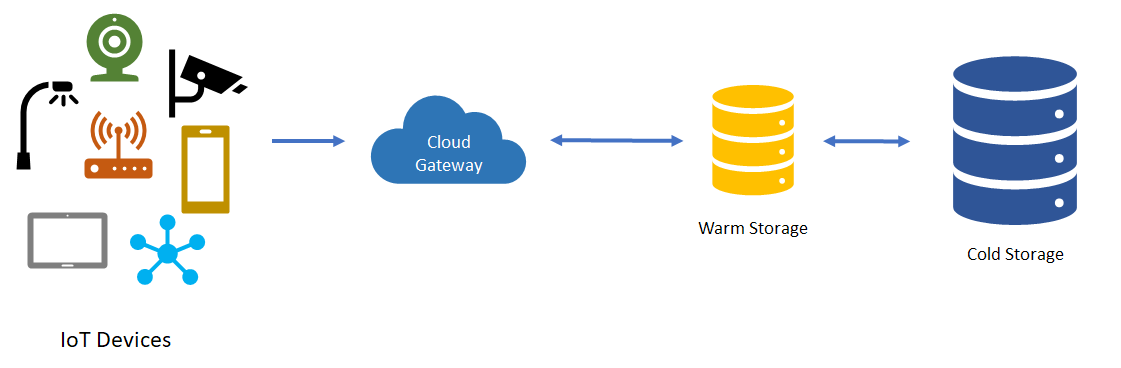
Cloud service providers are continually updating their data services to make it easier and more cost-effective for organizations to store, manage,and analyze data. Even so, a thorough analysis of cloud storage technical options and prices should be a fundamental part of any IoT architecture. For example, some architectures may demand a multi-tiered approach with some data being stored on the device, other stored in on-premise databases and other data stored in the cloud. Depending on the needed architecture, you should be sure the cloud services you choose supports your needs.

Here are some other concepts to be aware of when considering cloud storage.

Data is often time series data and is required to be stored where it can be used in visualization and reporting as well as later accessed for additional processing. It is common to have data split into “warm” and “cold” data stores. The **warm data store** holds recent data that needs to be accessed with low latency. Data stored in **cold storage** is typically historical data. Most often the cold storage database solution chosen will be cheaper in cost but offer fewer query and reporting features than the warm database solution.

**Note**: The Lambda architecture for data storage going to warm and cold storage is introduced later in this course.

A common implementation for storage is to keep a recent range (e.g. the last day, week, or month) of telemetry data in warm storage and to store historical data in cold storage. With this implementation, the application has access to the most recent data and can quickly observe recent telemetry data and trends. Retrieving historical information for devices can be accomplished using cold storage, generally with higher latency than if the data were in warm storage.



Cloud service providers may provide services to support both types of storage and make managing data across these types easier.[^1]

[^1]: You can read more about warm and cold storage different technologies Microsoft Azure provides for managing these storage options in section 3.5 of the [Azure Reference Architecure document](https://aka.ms/iotrefarchitecture).

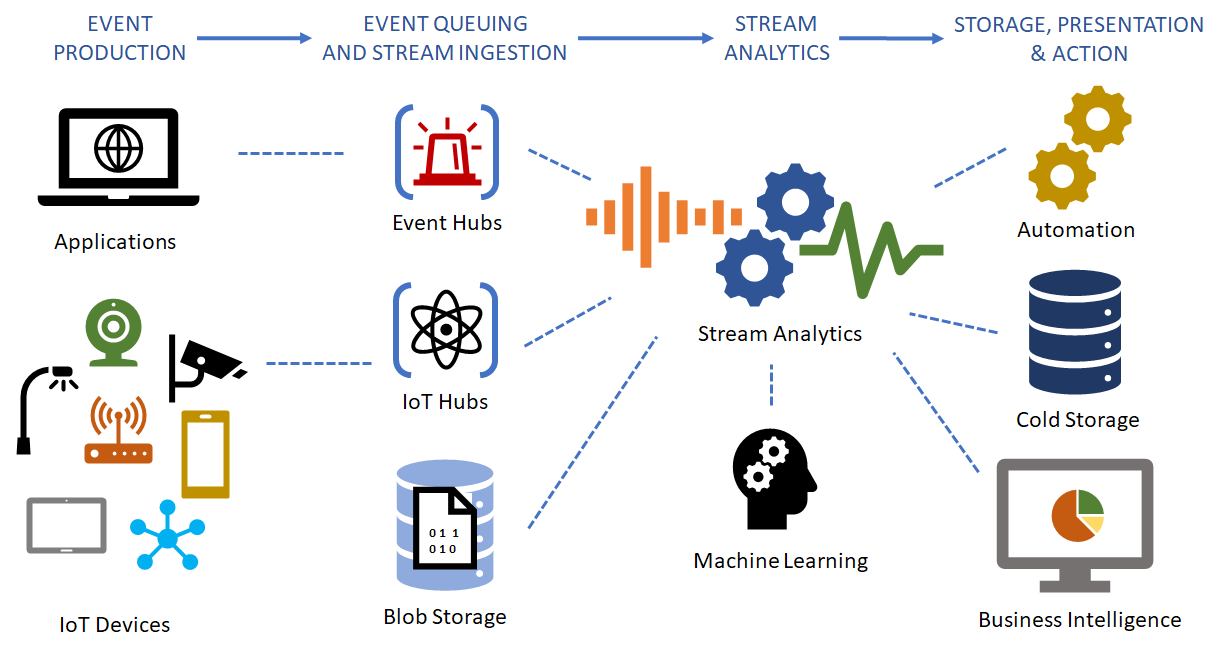
#### Analytics Services and Data Visualization

##### Analytics

Once data is captured and stored, it only becomes useful when it provides insights into the physical world from which your IoT devices have captured the data. This is where analytic services come into play.

Azure Analysis Services, for example, enable architects to use advanced mashup and modeling features to combine data from multiple data sources, define metrics, and secure data in a single, trusted tabular [semantic data model](https://en.wikipedia.org/wiki/Semantic_data_model). The data model provides an easier and faster way for users to browse massive amounts of data for ad-hoc data analysis.

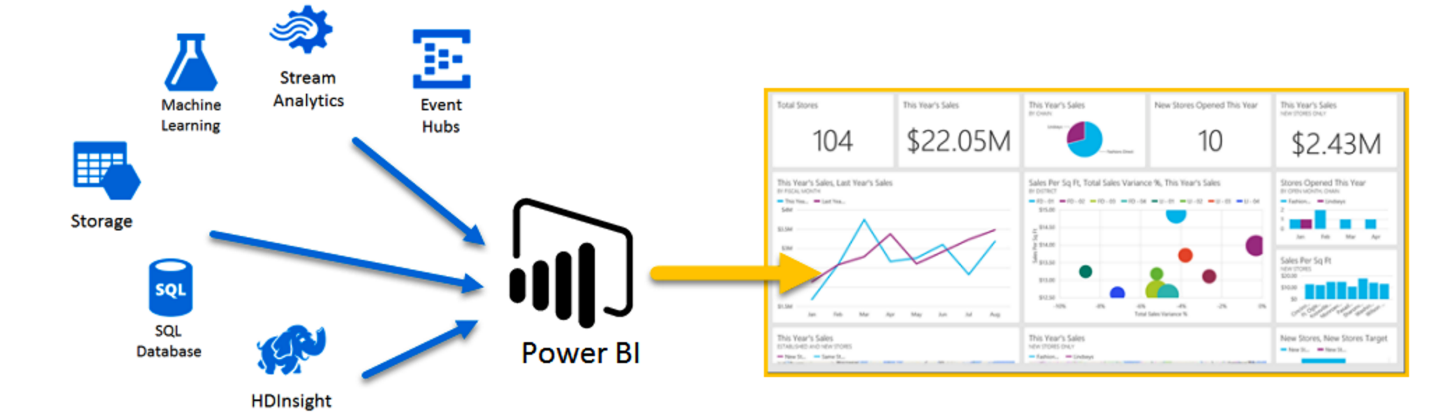
Without analytics, data collected from IoT would be too voluminous and unstructured to visualize or gain insights. Analytic services enable architects to build meaningful relationships between sets of data in order to make it easier to manage. For example, Azure Stream Analytics can take stream data from IoT devices and engineers can specify a transformation query that defines how to look for data, patterns, or relationships. The transformation query leverages a SQL-like query language that is used to filter, sort, aggregate, and join streaming data over a period of time.



##### Data Visualization

Stream analytics can help condition data so its easier to manage and provides models that give insight into what you need to understand or learn. Once the data is conditioned and you've created the right models, the data can be visualized using tools like Microsoft's Power BI or Tableau so it can be acted upon.

Data visualization tools can take input from various data streams and combine them into “dashboards” that can be used to tell a story about the data that was collected. Ultimately, this is the goal of IoT.



#### Machine Learning

Machine Learning (ML) is one of the more exiting developments in modern computer science. It's a complex field but one that is producing significant positive results with large datasets. As we've said throughout this course, IoT devices produces large large volumes of data. Analytic systems help engineers to model the existing data in meaningful ways. [Machine learning](https://en.wikipedia.org/wiki/Machine_learning) takes this a step further and can actually make predictions about what new data will show and provide insights that would not be possible without the machine learning algorithms.

As the name states, the technology gives computers the ability to “learn” (predict) from data by expressing trends or a direction future data will take. This can provide engineers with a powerful mechanism for enabling a wide variety of scenarios.

Using big data and machine learning to predict purchasing decisions is one simple example. Suppose a retailer has warehouse space in various cities and needs to determine which items to stock in those cities in order to be able to get products to customers in the most efficient and timely way. Using machine learning the retailer can predict, for example, that a given set of users that purchase a specific television tend to buy a particular type of cable and other accessories like tv stands and audio equipment. This would allow the retailer to keep those items in the warehouse near where those television sales are popular so that if a customer orders the cable or other accessory, the item can be shipped more and get to the customer more quickly.

Can you think of other, IoT-specific scenarios where machine learning would be help enable various scenarios that can make the IoT architecture more effective?

Because of the tremendous amount of computer power needed to perform the calculations needed to do this type of analysis, cloud-based ML technology tends to be the most effective at providing the type of insights machine learning promises.

#### Conclusion

In this topic, we've surveyed the various cloud-based services and technologies that make IoT possible. Below, you can try your hand at using the Azure cloud to model IoT scenarios. While these exercises are just an introduction to the space, they can give you a good feel for how the technology works together and how it can be used in an IoT architecture.

### Features of Azure IoT Hub

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. You can use Azure IoT Hub to build IoT solutions with reliable and secure communications between millions of IoT devices and a cloud-hosted solution backend. You can connect virtually any device to IoT Hub.

IoT Hub supports communications both from the device to the cloud and from the cloud to the device. IoT Hub supports multiple messaging patterns such as device-to-cloud telemetry, file upload from devices, and request-reply methods to control your devices from the cloud. IoT Hub monitoring helps you maintain the health of your solution by tracking events such as device creation, device failures, and device connections.

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, and monitoring office building usage.

#### Secure your communications

IoT Hub gives you a secure communication channel for your devices to send data.

* Per-device authentication enables each device to connect securely to IoT Hub and for each device to be managed securely.
* You have complete control over device access and can control connections at the per-device level.
* The IoT Hub Device Provisioning Service automatically provisions devices to the correct IoT hub when the device first boots up.
* Multiple authentication types support a variety of device capabilities:
  + SAS token-based authentication to quickly get started with your IoT solution.
  + Individual X.509 certificate authentication for secure, standards-based authentication.
  + X.509 CA authentication for simple, standards-based enrollment.

#### Scale your solution

IoT Hub scales to millions of simultaneously connected devices and millions of events per second to support your IoT workloads.

#### Route device data

Built-in message routing functionality gives you flexibility to set up automatic rules-based message fan-out:

* Use message routing to control where your hub sends device telemetry.
* There is no additional cost to route messages to multiple endpoints.
* No-code routing rules take the place of custom message dispatcher code.

#### Integrate with other services

You can integrate IoT Hub with other Azure services to build complete, end-to-end solutions. For example, use:

* Azure Event Grid to enable your business to react quickly to critical events in a reliable, scalable, and secure manner.
* Azure Logic Apps to automate business processes.
* Azure Machine Learning to add machine learning and AI models to your solution.
* Azure Stream Analytics to run real-time analytic computations on the data streaming from your devices.

#### Configure and control your devices

You can manage your devices connected to IoT Hub with an array of built-in functionality.

* Store, synchronize, and query device metadata and state information for all your devices.
* Set device state either per-device or based on common characteristics of devices.
* Automatically respond to a device-reported state change with message routing integration.

#### Make your solution highly available

There's a 99.9% Service Level Agreement for IoT Hub. The full Azure SLA explains the guaranteed availability of Azure as a whole.

#### Connect your devices

Use the Azure IoT device SDK libraries to build applications that run on your devices and interact with IoT Hub. Supported platforms include multiple Linux distributions, Windows, and real-time operating systems. Supported languages include C, C#, Java, Python, and Node.js. IoT Hub and the device SDKs support a wide range of protocols for connecting devices, including HTTPS, AMQP, AMQP over WebSockets, MQTT, and MQTT over WebSockets.

#### Quotas and limits

Each Azure subscription has default quota limits in place to prevent service abuse, and these limits could impact the scope of your IoT solution. The current limit on a per-subscription basis is 50 IoT hubs per subscription. You can request quota increases by contacting support.

### Features of Azure IoT Hub Device Provisioning Service

#### Registration and provisioning

Provisioning means various things depending on the industry in which the term is used. In the context of provisioning IoT devices to their cloud solution, provisioning is a two part process:

* The first part is establishing the initial connection between the device and the IoT solution by registering the device.
* The second part is applying the proper configuration to the device based on the specific requirements of the solution it was registered to.

Once both of those two steps have been completed, we can say that the device has been fully provisioned. Some cloud services only provide the first step of the provisioning process, registering devices to the IoT solution endpoint, but do not provide the initial configuration. The Device Provisioning Service automates both steps to provide a seamless provisioning experience for the device.

#### When to use Device Provisioning Service

There are many provisioning scenarios in which the Device Provisioning Service is an excellent choice for getting devices connected and configured to IoT Hub, such as:

* 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)

Many of the manual steps traditionally involved in provisioning are automated with the Device Provisioning Service to reduce the time to deploy IoT devices and lower the risk of manual error.

#### Features of the Device Provisioning Service

The Device Provisioning Service has many features, making it ideal for provisioning devices.

* Secure attestation support for both X.509 and TPM-based identities.
* Enrollment list containing the complete record of devices/groups of devices that may at some point register. The enrollment list contains information about the desired configuration of the device once it registers, and it can be updated at any time.
* Multiple allocation policies to control how the Device Provisioning Service assigns devices to IoT hubs in support of your scenarios.
* Monitoring and diagnostics logging to make sure everything is working properly.
* Multi-hub support allows the Device Provisioning Service to assign devices to more than one IoT hub. The Device Provisioning Service can talk to hubs across multiple Azure subscriptions.
* Cross-region support allows the Device Provisioning Service to assign devices to IoT hubs in other regions.

### Scenario Overview

In this course, you spend most of your time focused on the Azure Platform-as-a-Service tools for IoT, but you will also take a little time to become familiar with IoT Central, Microsoft's Software-as-a-Service alternative. Understanding how to create an IoT solution using each of these tools will help you to determine when to implement one over the other for your company and/or customers.

Your lab experience begins with an introduction to the Azure services for IoT. As the course progresses, you will be using Azure IoT platform services such as Azure IoT Hub and the IoT Hub Device Provisioning Service, in conjunction with Azure IoT SDKs and various other tools, to build a custom Azure IoT solution from scratch.

In the final lab, you will use IoT Central to build and deploy a second IoT solution.

#### Lab Scenario

During the hands-on labs, you will be acting in the role of an Azure IoT Developer helping to create IoT solutions. You are employed by Contoso, a global company that specializes in making and selling gourmet cheese. At Contoso, you are prepared to start your digital transformation journey, with the goal of improving processes throughout the company. This will include the factory floors where you produce your cheese, your fleet of vehicles used to transport your products, your warehouse facilities, and your facilities used for aging and storing cheese.

To get started, you will set up an Azure subscription and become familiar with the Azure IoT services. Getting your Azure portal set up and examining Azure IoT services will help you to work more efficiently when you start building your IoT solution. You will also work on setting up your development environment, another important step to take before starting work on your IoT solution.

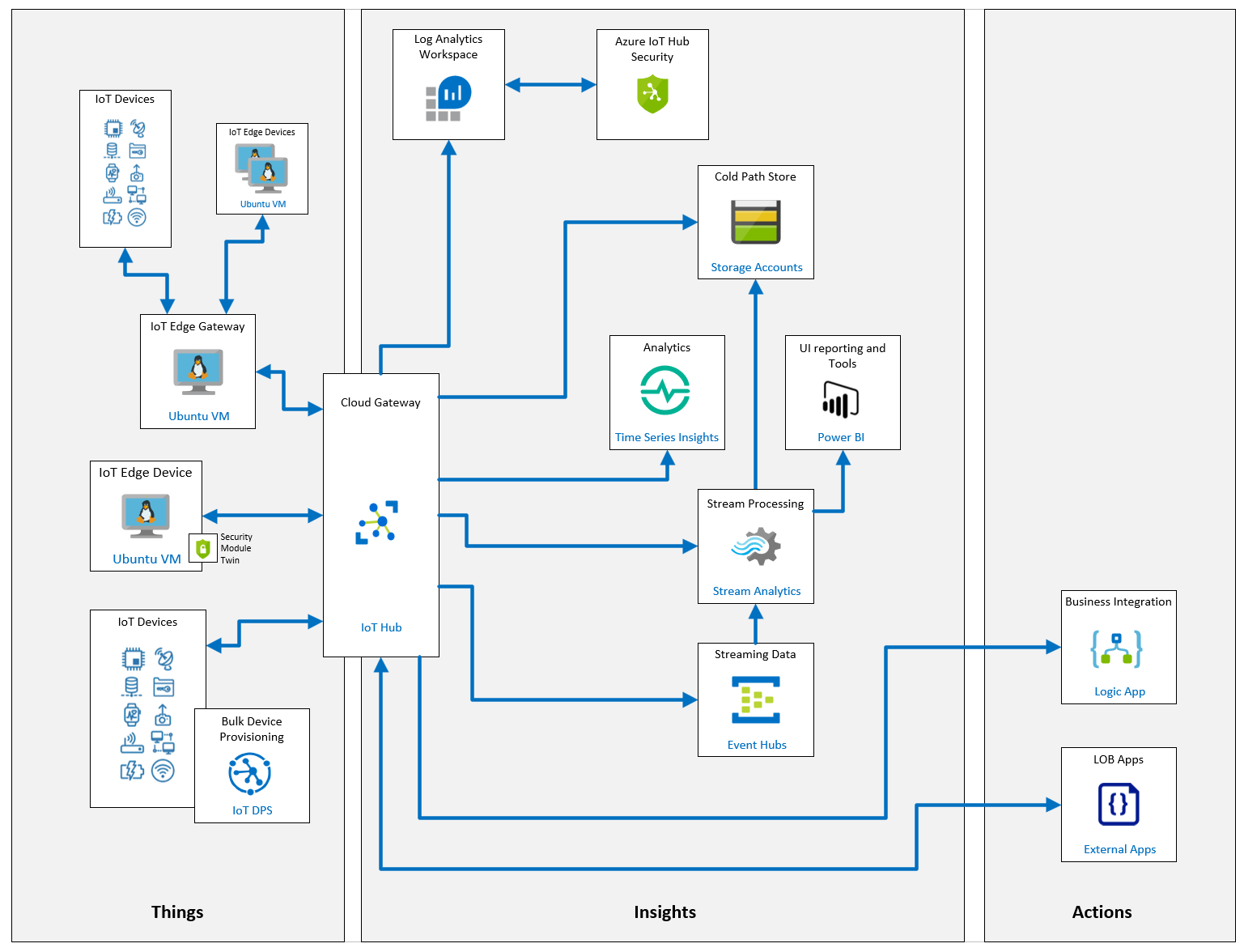
Contoso has become known for producing top quality cheeses, which has led to rapid growth in popularity and sales. You need to ensure that your cheese products stay at the same high level of quality. This means that the conditions at your factory floor, cheese aging facilities, and during transportation to customers all need to be strictly controlled as your business expands and production increases.

##### Architecture

As you move through each of the hands-on labs, you will be focusing on specific aspects of the overall scenario - reusing earlier resources, if they are still available, or creating new resources as required. However, it is often useful to understand these aspects within the context of a complete architectural view of the solution. An IoT architecture is usually discussed within the context of 3 component categories:

* **Things** - devices that send data.
* **Insights** - insights are generated from the data that the devices send.
* **Actions** - the insights generate actions to improve a business or process.

The architecture diagram below represents the components that you will implement during the labs in this course as you build Contoso's Azure IoT solution.



##### Things

Within the **Things** category, you should find the following components:

* **IoT Hub Device Provisioning Service (DPS)** - a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention, allowing customers to provision millions of devices in a secure and scalable manner. A number of hands-on labs use DPS to connect both regular devices and IoT Edge devices to IoT Hub. An ioT Hub is foundational to any IoT solution built with Azure - Azure IoT Central leverages IoT Hub behind the scenes.
* **IoT Devices** - these are internet-connected devices that send telemetry data to the cloud. A number of labs use C# .NET Core console applications to simulate devices equipped with temperature, humidity, GPS and vibration sensors. The simulated devices map to the Contoso scenarios for cheese making, packaging and shipping. Some of these devices will utilize the IoT DPS component to obtain connection configuration and register with the IoT Hub, whereas others will be manually registered and use symmetric keys to connect to an IoT Hub.
* **IoT Edge Devices** - Azure IoT Edge moves cloud analytics and custom business logic to devices so that your organization can focus on business insights instead of data management. During the hands-on labs you will deploy IoT Edge devices that run on Ubuntu Virtual Machines and utilize custom code modules to simulate sensors, intelligent-edge analytics to reduce the volume of data sent to the IoT hub and generate local alerts, and support offline operation scenarios.
* **IoT Edge Gateways** - IoT Edge Gateways are used to provide device connectivity and edge analytics to IoT devices that otherwise wouldn't have those capabilities. In a hands-on lab, you will configure an IoT Edge device to act as a **Transparent Gateway**. The gateway simply passes communications between the devices and IoT Hub. Both the devices and the users interacting with them through IoT Hub are unaware that a gateway is mediating their communications. The advantage of this approach is that all the device communication is multiplexed over a single, secure connection, rather than ensuring each individual device connection is secured appropriately.

##### Insights

Within the **Insights** category, you should find the following components:

* **IoT Hub** - the IoT hub acts as a central message hub for bi-directional communication between your IoT solution and the devices it manages. During the hands-on labs you will be registering IoT devices, IoT Edge devices, linking Device Provisioning Services and routing messages for the purposes of archiving logs to cold storage and passing telemetry onto Stream Analytics.
* **Storage Accounts** - Azure Storage is Microsoft's cloud storage solution for modern data storage scenarios. Azure Storage offers a massively scalable object store for data objects, a file system service for the cloud, a messaging store for reliable messaging, and a NoSQL store. The hands-on labs utilize blob storage to archive log data sent from IoT devices - cold data storage. Of course, there are many more scenarios for persistent storage of data than addressed by the hands-on labs, and additional storage options such as Data Lake, CosmosDB and so on.
* **Stream Analytics** - Azure Stream Analytics is a real-time analytics and complex event-processing engine that is designed to analyze and process high volumes of fast streaming data from multiple sources simultaneously. In the hands-on labs you will be building a preventative maintenance solution that utilizes a Machine Learning vibration analysis function within the Stream Analytics query to identify anomalies and send data to Power BI. Additionally, you will configure a Stream Analytics module that will be deployed to an IoT Edge device to reduce data volume sent to the IoT hub and provide offline capability.
* **Power BI** - Power BI is a business analytics tool with rich visualization capabilities that allows you to share insights and results across your organization. In the hands-on labs you will build a dashboard with multiple tiles that display device telemetry and the vibration analysis from Stream Analytics. This will enable an operator to identify clusters on anomalous readings that may indicate the need to service conveyors within the cheese packaging facility.
* **Time Series Insights** - Azure Time Series Insights (TSI) is an end-to-end platform-as-a-service offering used to collect, process, store, analyze, and query data from 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 the hands-on labs you will use TSI to correlate anomalies between temperature telemetry streams within a cheese shipping solution. This will help operators identify if specific trucks or aircraft have faulty temperature management solutions.
* **Event Hubs** - the primary difference between Azure IoT Hub and Azure Event Hubs is that Event Hubs designed for big data streaming, while IoT hub is optimized for an IoT solution. Both services support ingestion of data with low latency and high reliability. As Azure Event hubs are ideal for processing streaming data and support live dashboarding scenarios - perfect for passing our vibration data to Power BI. A hands-on lab explores how you can add an Event Hub to your solution.
* **Log Analytics Workspace** & **Azure IoT Hub Security** - Azure Security Center for IoT unifies security management and enables end-to-end threat detection and analysis across hybrid cloud workloads and your Azure IoT solution. The Log Analytics Workspace is a unique environment for Azure Monitor log data. Each workspace has its own data repository and configuration, and data sources and solutions are configured to store their data in a particular workspace. In the hands-on labs you will add security agents and security module twins to IoT edge devices and configure the Azure Security Center for IoT. You will explore the recommendations UI and learn how to configure Custom Alerts.

##### Actions

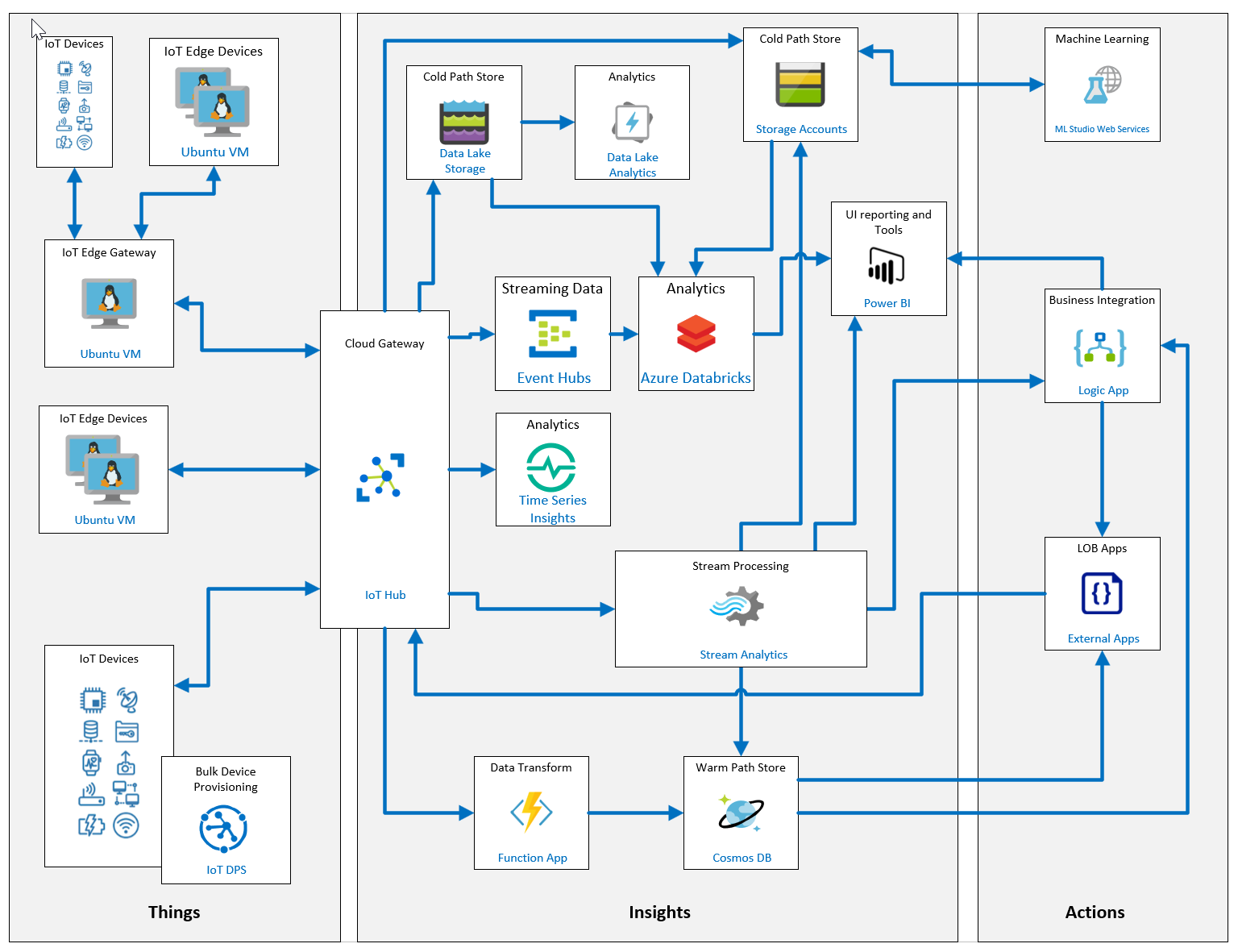
Within the **Actions** category, you should find the following components:

* **Logic Apps** - 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 the hands-on labs you will leverage a logic app to send an email to a solution operator whenever a temperature monitoring IoT device is added to the IoT Hub.
* **External Apps** - External apps are implemented in many languages and may connect either directly to an IoT Hub via service connections, or via Logic Apps and Azure functions. In the hands-on labs, you will create C# .NET Core console applications that will receive event information and invoke direct methods on IoT devices in response to operator intervention.

##### Next Steps

As you proceed through the hands-on labs, refer back to the architectural overview for the Contoso solution above, and consider how each individual lab fits into the overall scenario and architecture.

Finally, it's important to realize that there are many more capabilities and configurations available for constructing Azure IoT solutions than we can cover in one course. To help illustrate the larger breadth of services available, the architecture diagram below includes additional analytics options with Azure Databricks, warm path storage with CosmosDB, Machine Learning through analysis of the volumes of data held in cold storage, and so on.



### Resources and 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.

#### Unique ID

Your unique ID will be constructed using your initials and the current date using the following pattern:

YourInitialsYYMMDD

So, your initials followed by the last two digits of the year, the current numeric month, and the numeric day . Here are some examples:

GWB190123

BHO190504

CAH191216

DM190911

In some cases, you may be asked to use the lowercase version of your unique ID:

gwb190123

bho190504

cah191216

dm190911

Whenever you are expected to use your unique ID, you will see {YOUR-ID}. You will replace the entire string (including the {}) with your unique value.

Make a note of your unique ID now and **use the same value through the entire course** - don't update the date each day.

Let's review some examples of resources and the names associated with them.

#### Resource Groups

A Resource Group must have a unique name within a subscription, however it does not need to be globally unique. Therefore, throughout this course you will be using the resource group name: **AZ-220-RG**.

**Information:** Resource Group Name - **AZ-220-RG**

#### Publicly Visible Resources

Many of the resources that you create will have publicly-addressable (although secured) endpoints and therefore must have globally unique. Examples of such resources include IoT Hubs, Device Provisioning Services, and Azure Storage Accounts. For each of these you will be provided with a name template and expected to replace {YOUR-ID} with your unique ID. Here are some examples:

If your Unique ID is: **CAH191216**

| **Resource Type** | **Name Template** | **Example** |
| --- | --- | --- |
| IoT Hub | AZ-220-HUB-{YOUR-ID} | AZ-220-HUB-CAH191216 |
| Device Provisioning Service | AZ-220-DPS-{YOUR-ID} | AZ-220-DPS-CAH191216 |
| Azure Storage Account (name must be lower-case and no dashes) | az220storage{YOUR-ID} | az220storagecah191216 |

You may also be required to update values within bash scripts and C# source files as well as entering the names into the Azure Portal UI. Here are some examples:

**#!/bin/bash**

YourID="{YOUR-ID}"

RGName="AZ-220-RG"

IoTHubName="AZ-220-HUB-$YourID"

Notice that YourID="{YOUR-ID}" should be updated to YourID="CAH191216" - you do not change $YourID. Similarly, in C# you might see:

private string \_yourId = "{YOUR-ID}";

private string \_rgName = "AZ-220-RG";

private string \_iotHubName = $"AZ-220-HUB-{\_yourId}";

Again, private string \_yourId = "{YOUR-ID}"; should be updated to private string \_yourId = "{CAH191216}"; - you do not change {\_yourId}.

### Module Labs

Your course instructor will provide guidance on accessing the module lab instructions

* Lab 1: Getting Started with Azure
* Lab 2: Getting Started with Azure IoT Services

**WARNING**: Be prepared for UI changes

Given the dynamic nature of Microsoft cloud tools, you may experience user interface (UI) changes that were made following the development of this training content that do not match up with lab instructions presented in this lab manual.

The Microsoft Learning team will update this training course as soon as we can when any such changes are brought to our attention. However, given the dynamic nature of cloud updates, you may run into UI changes before this training content is updated. If this occurs, you will have to adapt to the changes and work through them in the labs as needed.

### Module Review Questions

#### Review Question 1.1

You have been asked to join a team that will be working on an IoT solution for your company. You begin by reviewing a high-level reference architecture diagram for the proposed IoT solution. You see “cloud gateway” listed on the diagram. What is the primary purpose of the cloud gateway? (choose one best answer)

* It facilitates the execution of actions based on insights garnered from device telemetry data during stream processing.
* It provides secure connectivity, telemetry and event ingestion, and device management capabilities.
* It provides operational information and displays reports through a browser interface.
* It gathers input values from connected sensors and sends telemetry to the cloud.
* It provides immediate and long-term access to telemetry data.

Check Answers

#### Review Question 1.2

You have been asked to join a team that will be working on an IoT solution for your company. You know that security is an important consideration. When you review the proposed architecture diagram, you don’t see security listed. Where should security be called out on the architecture diagram? (choose one best answer)

* Security is associated primarily with devices and should appear on the device side of the diagram.
* Security is associated primarily with communication between the Cloud Gateway and the other services/devices and should appear next to Cloud Gateway on the diagram.
* Security is associated primarily with data storage and retrieval and should appear next to storage on the diagram.
* Security is associated with all subsystems of the solution and should appear across the top of the diagram.
* Security is associated primarily with cloud services and should appear on the cloud side of the diagram.

Check Answers

#### Review Question 1.3

You have been asked to join a team that will be working on an IoT solution for your company. Your team is small and will need a fully managed solution that can be started quickly and operated with minimal IoT experience. You are asked to help choose one of the Microsoft Azure offerings for IoT. Which of the Microsoft offerings should you recommend? (choose one best answer)

* IoT Hub, IoT Hub Device Provisioning Service, Stream Analytics, and Cosmos DB.
* IoT Hub, Stream Analytics, and Cosmos DB.
* IoT Hub, Stream Analytics, and Azure Storage.
* IoT Central.

Check Answers

#### Review Question 1.4

You have joined a team that is developing an IoT solution for your company. You will be implementing IoT Hub, the IoT Hub Device Provisioning Service, Azure Stream Analytics, Azure Storage, and Time Series Insights. You need to understand the features and capabilities of IoT Hub. Which of the following are features of IoT Hub? (choose all correct answers)

* Provides control over device access by using resource groups.
* Provides individual X.509 certificate authentication for secure, standards-based authentication.
* Enables routing of messages to multiple endpoints at no additional cost.
* Enables storage, synchronization, and querying of device metadata for all your devices.

Check Answers

#### Review Question 1.5

You have joined a team that is developing an IoT solution for your company. You will be implementing IoT Hub, the IoT Hub Device Provisioning Service, Azure Stream Analytics, Azure Storage, and Time Series Insights. You need to understand the features and capabilities of the Device Provisioning Service. Which of the following are features of the Device Provisioning Service? (choose all correct answers)

* Secure attestation support for both X.509 and TPM-based identities.
* Multiple allocation policies to control how the Device Provisioning Service assigns devices to IoT hubs.
* Multi-hub support that allows the Device Provisioning Service to assign devices to more than one IoT hub.
* Cross-region support that allows the Device Provisioning Service to assign devices to IoT hubs in other regions.

Check Answers

Devices and Device Comunication

### IoT Hub Tiers

Every IoT solution is different, so Azure IoT Hub offers several options based on pricing and scale.

The decision on which IoT Hub tier is right for your solution is generally made by the solution architect, however, understanding the implications of this decision is important to the IoT developer role who will be called upon to support the solution.

To evaluate which IoT Hub tier is right for your solution, consider these two questions:

* What features do I plan to use?

Azure IoT Hub offers two tiers, basic and standard, that differ in the number of features they support. If your IoT solution is based around collecting data from devices and analyzing it centrally, then the basic tier is probably right for you. If you want to use more advanced configurations to control IoT devices remotely or distribute some of your workloads onto the devices themselves, then you should consider the standard tier. For a detailed breakdown of which features are included in each tier continue to Basic and standard tiers.

* How much data do I plan to move daily?

Each IoT Hub tier is available in three sizes, based around how much data throughput they can handle in any given day. These sizes are numerically identified as 1, 2, and 3. For example, each unit of a level 1 IoT hub can handle 400 thousand messages a day, while a level 3 unit can handle 300 million.

#### Basic and standard tiers

The standard tier of IoT Hub enables all features, and is required for any IoT solutions that want to make use of the bi-directional communication capabilities. The basic tier enables a subset of the features and is intended for IoT solutions that only need uni-directional communication from devices to the cloud. Both tiers offer the same security and authentication features.

**Note**: IoT Hub also offers a free tier that is meant for testing and evaluation. The free tier does not support upgrading to basic or standard. It enables you to transmit up to a total of 8,000 messages per day, and register up to 500 device identities. The device identity limit is only present for the Free Edition.

| **Capability** | **Basic tier** | **Free/Standard tier** |
| --- | --- | --- |
| Device-to-cloud telemetry | Yes | Yes |
| Per-device identity | Yes | Yes |
| Message routing, message enrichments, and Event Grid integration | Yes | Yes |
| HTTP, AMQP, and MQTT protocols | Yes | Yes |
| Device Provisioning Service | Yes | Yes |
| Monitoring and diagnostics | Yes | Yes |
| Cloud-to-device messaging |  | Yes |
| Device twins, Module twins, and Device management |  | Yes |
| Device streams (preview) |  | Yes |
| Azure IoT Edge |  | Yes |
| IoT Plug and Play Preview |  | Yes |

#### Message throughput

Message traffic is measured for your IoT hub on a per-unit basis. When you create an IoT hub, you choose its tier and edition, and set the number of units available. You can purchase up to 200 units for the B1, B2, S1, or S2 edition, or up to 10 units for the B3 or S3 edition. After your IoT hub is created, you can change the number of units available within its edition, upgrade or downgrade between editions within its tier (B1 to B2), or upgrade from the basic to the standard tier (B1 to S1) without interrupting your existing operations.

Only one type of edition within a tier can be chosen per IoT Hub. For example, you can create an IoT Hub with multiple units of S1, but not with a mix of units from different editions, such as S1 and S2.

As an example of each tier's traffic capabilities, device-to-cloud messages follow these sustained throughput guidelines:

| **Tier edition** | **Sustained throughput** | **Sustained send rate** |
| --- | --- | --- |
| B1, S1 | Up to 1111 KB/minute per unit (1.5 GB/day/unit) | Average of 278 messages/minute per unit (400,000 messages/day per unit) |
| B2, S2 | Up to 16 MB/minute per unit (22.8 GB/day/unit) | Average of 4,167 messages/minute per unit (6 million messages/day per unit) |
| B3, S3 | Up to 814 MB/minute per unit (1144.4 GB/day/unit) | Average of 208,333 messages/minute per unit (300 million messages/day per unit) |

#### Partitions

Azure IoT hubs contain many of the core components of Azure Event Hubs, including Partitions.

The event streams for IoT hubs are generally populated with incoming telemetry data that is reported by various IoT devices. The partitioning of the event stream is used to reduce contentions that could occur when concurrently reading and writing to event streams.

The partition limit is chosen when IoT Hub is created and cannot be changed, so long-term scale should be considered when setting partition count. The maximum partition limit is 32 for both the basic and standard tiers of IoT Hub, but most IoT hubs only need 4 partitions. The number of partitions is directly related to the number of concurrent readers you expect to have.

The decision on how many partitions are needed is made by the solution architect. The default value of 4 partitions should be used unless otherwise specified by the architect.

#### Tier upgrade

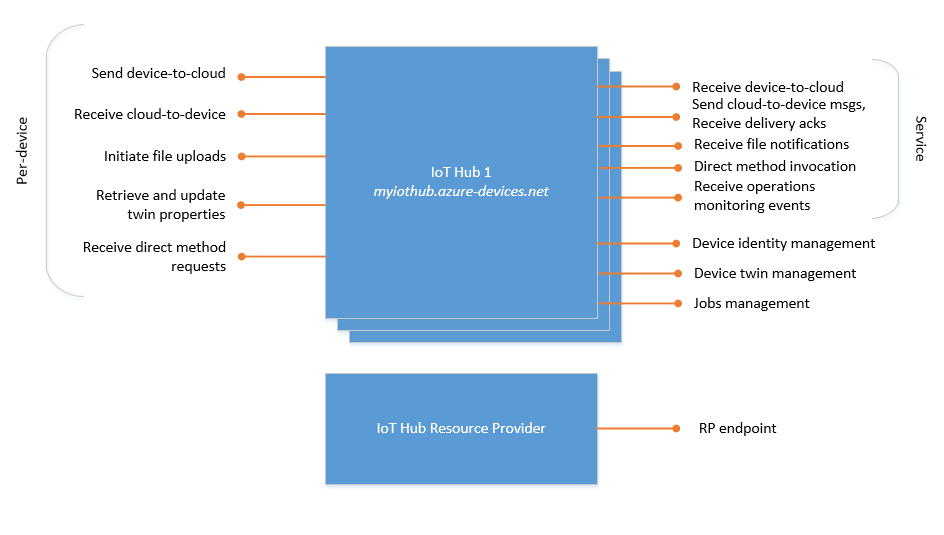
Once you create your IoT hub, you can upgrade from the basic tier to the standard tier without interrupting your existing operations. The partition configuration remains unchanged when you migrate from basic tier to standard tier.

**Note**: If you want to downgrade your IoT hub, you can remove units and reduce the size of the IoT hub but you cannot downgrade to a lower tier. For example, you can move from the S2 tier to the S1 tier, but not from the S2 tier to the B1 tier.

### IoT Hub Endpoints

Azure IoT Hub is a multi-tenant service that provides access to its functionality using a combination of built-in and custom endpoints.

#### Built-in Endpoints



The following list describes the endpoints:

* Resource provider. The IoT Hub resource provider exposes an Azure Resource Manager interface. This interface enables Azure subscription owners to create and delete IoT hubs, and to update IoT hub properties. IoT Hub properties govern hub-level security policies, as opposed to device-level access control, and functional options for cloud-to-device and device-to-cloud messaging. The IoT Hub resource provider also enables you to export device identities.
* Device identity management. Each IoT hub exposes a set of HTTPS REST endpoints to manage device identities (create, retrieve, update, and delete). Device identities are used for device authentication and access control.
* Device twin management. Each IoT hub exposes a set of service-facing HTTPS REST endpoints to query and update device twins (update tags and properties).
* Jobs management. Each IoT hub exposes a set of service-facing HTTPS REST endpoint to query and manage jobs.
* Device endpoints. For each device in the identity registry, IoT Hub exposes a set of endpoints:
  + Send device-to-cloud messages. A device uses this endpoint to send device-to-cloud messages.
  + Receive cloud-to-device messages. A device uses this endpoint to receive targeted cloud-to-device messages.
  + Initiate file uploads. A device uses this endpoint to receive an Azure Storage SAS URI from IoT Hub to upload a file.
  + Retrieve and update device twin properties. A device uses this endpoint to access its device twin's properties.
  + Receive direct method requests. A device uses this endpoint to listen for direct method's requests.

These endpoints are exposed using MQTT v3.1.1, HTTPS 1.1, and AMQP 1.0 protocols. AMQP is also available over WebSockets on port 443.

* Service endpoints. Each IoT hub exposes a set of endpoints for your solution back end to communicate with your devices. With one exception, these endpoints are only exposed using the AMQP protocol. The method invocation endpoint is exposed over the HTTPS protocol.
  + Receive device-to-cloud messages. This endpoint is compatible with Azure Event Hubs. A back-end service can use it to read the device-to-cloud messages sent by your devices. You can create custom endpoints on your IoT hub in addition to this built-in endpoint.
  + Send cloud-to-device messages and receive delivery acknowledgments. These endpoints enable your solution back end to send reliable cloud-to-device messages, and to receive the corresponding delivery or expiration acknowledgments.
  + Receive file notifications. This messaging endpoint allows you to receive notifications of when your devices successfully upload a file.
  + Direct method invocation. This endpoint allows a back-end service to invoke a direct method on a device.
  + Receive operations monitoring events. This endpoint allows you to receive operations monitoring events if your IoT hub has been configured to emit them. For more information, see IoT Hub operations monitoring.

#### Custom Endpoints

You can link existing Azure services in your subscription to your IoT hub to act as endpoints for message routing. These endpoints act as service endpoints and are used as sinks for message routes. Devices cannot write directly to the additional endpoints.

IoT Hub currently supports the following Azure services as additional endpoints:

* Azure Storage containers
* Event Hubs
* Service Bus Queues
* Service Bus Topics

### Introduction to IoT Hub Security Features

IoT Hub uses a combination of authentication and access control permissions to help protect your IoT solution.

#### Access control and permissions

IoT Hub uses permissions to grant access to each IoT hub endpoint. Permissions limit the access to an IoT hub based on functionality.

You can grant permissions in the following ways:

* IoT hub-level shared access policies. Shared access policies can grant any combination of permissions. You can define policies in the Azure portal, programmatically by using the IoT Hub Resource REST APIs, or using the az iot hub policy CLI. A newly created IoT hub has the following default policies:

| **Shared Access Policy** | **Permissions** |
| --- | --- |
| iothubowner | All permission |
| service | ServiceConnect permissions |
| device | DeviceConnect permissions |
| registryRead | RegistryRead permissions |
| registryReadWrite | RegistryRead and RegistryWrite permissions |

* Per-Device Security Credentials. Each IoT Hub contains an identity registry For each device in this identity registry, you can configure security credentials that grant DeviceConnect permissions scoped to the corresponding device endpoints.

For example, in a typical IoT solution:

* The device management component uses the registryReadWrite policy.
* The event processor component uses the service policy.
* The run-time device business logic component uses the service policy.
* Individual devices connect using credentials stored in the IoT hub's identity registry.

#### Authentication

Azure IoT Hub grants access to endpoints by verifying a token against the shared access policies and identity registry security credentials.

Security credentials, such as symmetric keys, are never sent over the wire.

#### Scope IoT hub-level credentials

You can scope IoT hub-level security policies by creating tokens with a restricted resource URI. For example, the endpoint to send device-to-cloud messages from a device is /devices/{deviceId}/messages/events. You can also use an IoT hub-level shared access policy with DeviceConnect permissions to sign a token whose resourceURI is /devices/{deviceId}. This approach creates a token that is only usable to send messages on behalf of device deviceId.

#### Security tokens

IoT Hub uses security tokens to authenticate devices and services to avoid sending keys on the wire. Additionally, security tokens are limited in time validity and scope. Azure IoT SDKs automatically generate tokens without requiring any special configuration. Some scenarios do require you to generate and use security tokens directly. Such scenarios include:

* The direct use of the MQTT, AMQP, or HTTPS surfaces.
* The implementation of the token service pattern, as explained in Custom device authentication.

IoT Hub also allows devices to authenticate with IoT Hub using X.509 certificates.

#### Supported X.509 certificates

You can use any X.509 certificate to authenticate a device with IoT Hub by uploading either a certificate thumbprint or a certificate authority (CA) to Azure IoT Hub. Authentication using certificate thumbprints only verifies that the presented thumbprint matches the configured thumbprint. Authentication using certificate authority validates the certificate chain.

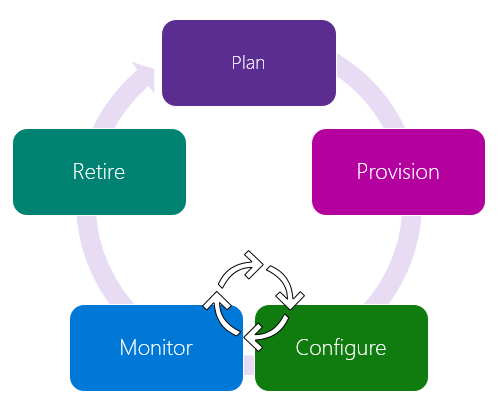
Supported certificates include:

* An existing X.509 certificate. A device may already have an X.509 certificate associated with it. The device can use this certificate to authenticate with IoT Hub. Works with either thumbprint or CA authentication.
* CA-signed X.509 certificate. To identify a device and authenticate it with IoT Hub, you can use an X.509 certificate generated and signed by a Certification Authority (CA). Works with either thumbprint or CA authentication.
* A self-generated and self-signed X-509 certificate. A device manufacturer or in-house deployer can generate these certificates and store the corresponding private key (and certificate) on the device. You can use tools such as OpenSSL and Windows SelfSignedCertificate utility for this purpose. Only works with thumbprint authentication.

A device may either use an X.509 certificate or a security token for authentication, but not both.

### Device Lifecycle Terms and Concepts

There is a set of general device management stages that are common to all enterprise IoT projects. In Azure IoT, there are five stages within the device lifecycle:



Each of the five stages includes goals and activities:

* Plan: Enable operators to create a device metadata scheme that enables them to easily and accurately query for, and target a group of devices for bulk management operations. You can use the device twin to store this device metadata in the form of tags and properties.
* Provision: Securely provision new devices to IoT Hub and enable operators to immediately discover device capabilities. Use the IoT Hub identity registry to create flexible device identities and credentials, and perform this operation in bulk by using a job. Build devices to report their capabilities and conditions through device properties in the device twin.
* Configure: Facilitate bulk configuration changes and firmware updates to devices while maintaining both health and security. Perform these device management operations in bulk by using desired properties or with direct methods and broadcast jobs.
* Monitor: Monitor overall device collection health, the status of ongoing operations, and alert operators to issues that might require their attention. Apply the device twin to allow devices to report realtime operating conditions and status of update operations. Build powerful dashboard reports that surface the most immediate issues by using device twin queries.
* Retire: Replace or decommission devices after a failure, upgrade cycle, or at the end of the service lifetime. Use the device twin to maintain device info if the physical device is being replaced, or archived if being retired. Use the IoT Hub identity registry for securely revoking device identities and credentials.

### Azure IoT Devices

The devices that you implement within an Azure IoT solution can range from constrained sensors and single purpose microcontrollers, to powerful gateways that route communications for groups of devices.

#### Azure IoT Device Types

For the purpose of this course, we will describe the devices that connect to IoT Hub as being one of the following; IoT Devices, IoT Edge Devices, or Simulated Devices.

* IoT Devices: An IoT device is typically a small-scale, standalone computing device that may collect data or control other devices. For example, a device might be an environmental monitoring device, or a controller for the watering and ventilation systems in a greenhouse.
* IoT Edge Devices: IoT Edge devices have the IoT Edge runtime installed and are flagged as IoT Edge device in the device details. An IoT Edge device can be used as a field gateway device, meaning that it is an IoT Edge device that connects downstream devices to the Azure IoT Hub (downstream devices can be either IoT devices or IoT Edge devices). IoT Edge will be discussed in detail later in the course.
* Simulated Devices: A simulated device is a software representation of a physical device that runs on your local machine or in the cloud. Simulated devices can be used in various stages during the rollout of an IoT solution to represent individual device behaviors or to generate a telemetry workload.

**Note**: Microsoft maintains an online device catalog that provides a list of hardware devices certified to work with IoT Hub - [Azure Certified for IoT Device Catalog](https://catalog.azureiotsolutions.com/alldevices).

### Device Identity and Registration

Every IoT hub has an identity registry that stores information about the devices permitted to connect to the IoT hub. Before a device can connect to an IoT hub, there must be an entry for that device in the IoT hub's identity registry. A device must also authenticate with the IoT hub based on credentials stored in the identity registry.

The device ID stored in the identity registry is case-sensitive.

At a high level, the identity registry is a REST-capable collection of device identity resources. When you add an entry in the identity registry, IoT Hub creates a set of per-device resources such as the queue that contains in-flight cloud-to-device messages.

Use the identity registry when you need to:

* Provision devices that connect to your IoT hub.
* Control per-device access to your hub's device-facing endpoints.

#### Module Identity

In IoT Hub, under each device identity, you can create up to 20 module identities. Each module identity implicitly generates a module twin. Similar to device twins, module twins are JSON documents that store module state information including metadata, configurations, and conditions. Azure IoT Hub maintains a module twin for each module that you connect to IoT Hub.

On the device side, the IoT Hub device SDKs enable you to create modules where each one opens an independent connection to IoT Hub. This functionality enables you to use separate namespaces for different components on your device. For example, you have a vending machine that has three different sensors. Each sensor is controlled by different departments in your company. You can create a module for each sensor. This way, each department is only able to send jobs or direct methods to the sensor that they control, avoiding conflicts and user errors.

Module identity and module twin provide the same capabilities as device identity and device twin but at a finer granularity. This finer granularity enables capable devices, such as operating system-based devices or firmware devices managing multiple components, to isolate configuration and conditions for each of those components. Module identity and module twins provide a management separation of concerns when working with IoT devices that have modular software components.

#### Identity registry operations

The IoT Hub identity registry exposes the following operations:

* Create device or module identity
* Update device or module identity
* Retrieve device or module identity by ID
* Delete device or module identity
* List up to 1000 identities
* Export device identities to Azure blob storage
* Import device identities from Azure blob storage

#### Create an IoT Device using the Azure Portal

When you create a new IoT device using the Azure portal, you will specify a device ID and a method for device authentication. IoT hub uses an identity registry to store information about the individual devices that are permitted to connect to it.

* Device ID: The device identity is a unique identifier that is assigned to a device.
* Authentication Type: Device authentication can be accomplished using either a Symmetric key pair or X.509 Certificate. The certificate can either be self-signed or come from a certificate authority.

### Introduction to Device Twins

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.

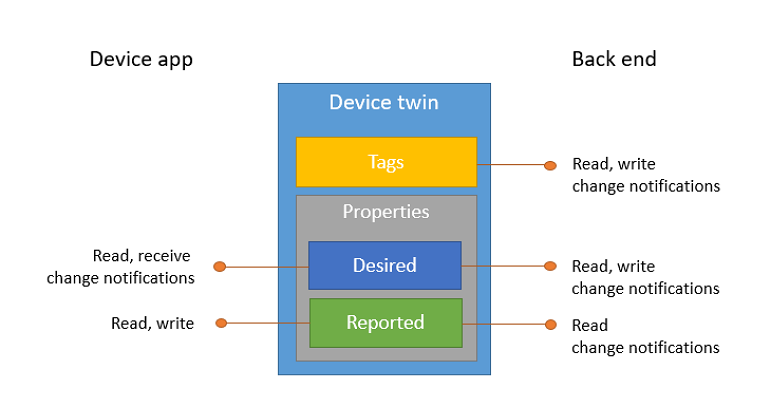
Device twins store device-related information that:

* Device and back ends can use to synchronize device conditions and configuration.
* The solution back end can use to query and target long-running operations.

The lifecycle of a device twin is linked to the corresponding device identity. Device twins are implicitly created and deleted when a device identity is created or deleted in IoT Hub.

A device twin is a JSON document that includes:

* Tags. A section of the JSON document that the solution back end can read from and write to. Tags are not visible to device apps.
* Desired properties. Used along with reported properties to synchronize device configuration or conditions. The solution back end can set desired properties, and the device app can read them. The device app can also receive notifications of changes in the desired properties.
* Reported properties. Used along with desired properties to synchronize device configuration or conditions. The device app can set reported properties, and the solution back end can read and query them.
* Device identity properties. The root of the device twin JSON document contains the read-only properties from the corresponding device identity stored in the identity registry.



#### Example

The following example shows a device twin JSON document:

{

"deviceId": "devA",

"etag": "AAAAAAAAAAc=",

"status": "enabled",

"statusReason": "provisioned",

"statusUpdateTime": "0001-01-01T00:00:00",

"connectionState": "connected",

"lastActivityTime": "2015-02-30T16:24:48.789Z",

"cloudToDeviceMessageCount": 0,

"authenticationType": "sas",

"x509Thumbprint": {

"primaryThumbprint": null,

"secondaryThumbprint": null

},

"version": 2,

"tags": {

"$etag": "123",

"deploymentLocation": {

"building": "43",

"floor": "1"

}

},

"properties": {

"desired": {

"telemetryConfig": {

"sendFrequency": "5m"

},

"$metadata" : {...},

"$version": 1

},

"reported": {

"telemetryConfig": {

"sendFrequency": "5m",

"status": "success"

},

"batteryLevel": 55,

"$metadata" : {...},

"$version": 4

}

}

}

#### Usage

Use device twins to:

* Store device-specific metadata in the cloud. For example, the deployment location of a vending machine.
* Report current state information such as available capabilities and conditions from your device app. For example, a device is connected to your IoT hub over cellular or WiFi.
* Synchronize the state of long-running workflows between device app and back-end app. For example, when the solution back end specifies the new firmware version to install, and the device app reports the various stages of the update process.
* Query your device metadata, configuration, or state.

#### Purpose and Best Practices

Device twins enable synchronizing desired configuration from the cloud and for reporting current configuration and device properties. The best way to implement device twins within cloud solutions applications is through the Azure IoT SDKs. Device twins are best suited for configuration because they:

* Support bi-directional communication.
* Allow for both connected and disconnected device states.
* Follow the principle of eventual consistency.
* Are fully queryable in the cloud.

### Introduction to Device Monitoring

Monitor overall device collection *health*, the *status* of ongoing operations, and alert operators to issues that might require their attention. Apply the device twin to allow devices to report realtime operating conditions and status of update operations. Build powerful dashboard reports that surface the most immediate issues by using device twin queries.

*Further reading*:

* How to use device twin properties
* IoT Hub query language for device twins, jobs, and message routing
* Configure and monitor IoT devices at scale
* Best practices for device configuration within an IoT solution

### Introduction to Device Retirement

Replace or decommission devices after a failure, upgrade cycle, or at the end of the service lifetime. Use the device twin to maintain device info if the physical device is being replaced, or archived if being retired. Use the IoT Hub identity registry for securely revoking device identities and credentials.

*Further reading*:

* How to use device twin properties
* Manage device identities

### IoT Developer Tools Overview

Microsoft offers a comprehensive set of development tools for any developer — using any platform or language — to deliver IoT and other cloud applications. You can develop your IoT solution with the language of your choice using a range of SDKs and take advantage of full-featured integrated development environments (IDEs) and editors with advanced debugging capabilities and built-in Azure support.

The primary developer tool categories for Azure IoT include:

* SDKs: Download and install language-specific SDKs and tools for your platform of choice, including .NET, Java, Node.js, and Python
* Visual Studio: Get all the power and capabilities you need to easily develop, debug, deploy, manage, and diagnose cloud-scale applications on Azure, using a full-featured IDE
* Visual Studio Code: Edit and debug code quickly with a lightweight code editor that runs on macOS, Linux, and Windows
* CLIs: Use a cross-platform command-line interface to create and manage services and automate everyday tasks in Azure

### Azure IoT Hub SDKs

Azure IoT provides a set of open-source Software Development Kits (SDKs) to simplify and accelerate the development of IoT solutions built with Azure IoT Hub. Using the SDKs in prototyping and production enables you to:

* Develop a “future-proof” solution with minimal code: While you can use protocol libraries to communicate with Azure IoT Hub, you may come back to this decision later and regret it. You will miss out on a lot of upcoming advanced features of IoT Hub and spend time redeveloping code and functionality that you could get for free. The SDKs support new features from IoT Hub, so you can incorporate them with minimal code and ensure your solution is up-to-date.
* Leverage features designed for a complete software solution and focus on your specific need: The SDKs contain many libraries that address key problems and needs of IoT solutions such as security, device management, reliability, etc. You can speed up time to market by leveraging these libraries directly and focus on developing for your specific IoT scenario.
* Develop with your preferred language for different platform: You can develop with C, C#, Java, Node.js, or Python without worrying about protocol specific intricacy. The SDKs provide out-of-box support for a range of platforms and the C SDK can be ported to new platforms.
* Benefit from the flexibility of open source with support from Microsoft and community: The SDKs are available open source on GitHub and Microsoft works in the open. You can modify, adapt, and contribute to the code that will run your devices and your applications.

There are two categories of software development kits (SDKs) for working with IoT Hub:

* IoT Hub Device SDKs enable you to build apps that run on your IoT devices using device client or module client. These apps send telemetry to your IoT hub, and optionally receive messages, job, method, or twin updates from your IoT hub. You can also use module client to author modules for Azure IoT Edge runtime.
* IoT Hub Service SDKs enable you to build backend applications to manage your IoT hub, and optionally send messages, schedule jobs, invoke direct methods, or send desired property updates to your IoT devices or modules.

#### Azure IoT Device SDKs Platform Support

Microsoft publishes open-source SDKs on GitHub for the following languages: C, .NET (C#), Node.js, Java, and Python.

Microsoft provides SDK support in the following ways:

* Continuously builds and runs end-to-end tests against the master branch of the relevant SDK in GitHub on several popular platforms. To provide test coverage across different compiler versions, Microsoft generally tests against the latest LTS (Long Term Support) version and the most popular version.
* Provides installation guidance or installation packages if applicable.
* Fully supports the SDKs on GitHub with open-source code, a path for customer contributions, and product team engagement with GitHub issues.

### Azure IoT Hub Device SDKs

The Microsoft Azure IoT device SDKs contain code that facilitates building applications that connect to and are managed by Azure IoT Hub services.

#### Coding Language Support

Depending on the IoT scenario and your developer experience, you may have a preferred language or platform. The SDKs got you covered. Broad language and platform support with protocol flexibility allows you to develop in your preferred environment without worrying about protocol specific intricacy. Five languages are currently supported: C, C#, Java, Node.js, and Python. Microsoft strives to maintain consistency of APIs across the five languages, as much as language specific constructs allow.

Each language is being maintained as a public repository on GitHub, including sample code and documentation. In addition, the SDKs are available as binary packages from Nuget for C#, Maven for Java, apt-get for some Linux Distributions, npm for Node.js and pip for Python.

#### Platform Testing

The SDKs are regularly tested on the following platforms (when languages apply):

* Linux (Ubuntu, Debian, Raspbian)
* Windows
* MBED
* Arduino (Huzzah, ThingDev, FeatherM0), FreeRTOS (ESP32, ESP8266)
* .NETFramework 4.5, UWP, PCL (Profile 7 – UWP, Xamarin.iOS, Xamarin.Android), .NetMicroFramework, .NetStandard 1.3
* Intel Edison

#### Supported Protocols

IoT Hub allows devices to use the following protocols for device-side communications:

* MQTT
* MQTT over WebSockets
* AMQP
* AMQP over WebSockets
* HTTPS

### Azure IoT Hub Service SDKs

The IoT Hub Service SDKs enable you to build backend applications that interact directly with IoT Hub. The service SDKs contain code that facilitates building applications to manage your IoT hub, send C2D messages, schedule jobs, invoke direct methods, or send desired property updates to your IoT devices or modules.

#### Coding Language Support

Five languages are currently supported: C, C#, Java, Node.js, and Python.

#### Backend App Scenarios

* Identity registry (CRUD): Use your backend app to perform CRUD operation for individual device or in bulk.
* Cloud-to-device messaging: Use your backend app to send cloud-to-device messages in AMQP and AMQP-WS, and set up cloud-to-device message receivers.
* Direct Methods operations: Use your backend app to invoke direct method on device.
* Device Twins operations: Use your backend app to perform twin operations. The .NET (C#) SDK only supports Get Twin at the moment.
* Query: Use your backend app to perform query for information.
* Jobs: Use your backend app to perform job operation.
* File Upload: Set up your backend app to send file upload notification receiver.

Here is a detailed comparison of the various cloud-to-device communication options.

|  | **Direct methods** | **Twin's desired properties** | **Cloud-to-device messages** |
| --- | --- | --- | --- |
| Scenario | Commands that require immediate confirmation, such as turning on a fan. | Long-running commands intended to put the device into a certain desired state. For example, set the telemetry send interval to 30 minutes. | One-way notifications to the device app. |
| Data flow | Two-way. The device app can respond to the method right away. The solution back end receives the outcome contextually to the request. | One-way. The device app receives a notification with the property change. | One-way. The device app receives the message |
| Durability | Disconnected devices are not contacted. The solution back end is notified that the device is not connected. | Property values are preserved in the device twin. Device will read it at next reconnection. Property values are retrievable with the IoT Hub query language. | Messages can be retained by IoT Hub for up to 48 hours. |
| Targets | Single device using deviceId, or multiple devices using jobs. | Single device using deviceId, or multiple devices using jobs. | Single device by deviceId. |
| Size | Maximum direct method payload size is 128 KB. | Maximum desired properties size is 8 KB. | Up to 64 KB messages. |
| Frequency | High. For more information, see IoT Hub limits. | Medium. For more information, see IoT Hub limits. | Low. For more information, see IoT Hub limits. |
| Protocol | Available using MQTT or AMQP. | Available using MQTT or AMQP. | Available on all protocols. Device must poll when using HTTPS. |

### Visual Studio Code Extensions

Microsoft Azure IoT support for Visual Studio Code is provided through a rich set of extensions that make it easy to discover and interact with Azure IoT Hub and other Azure IoT services that power your IoT Edge and device applications.

The Azure IoT Tools collection includes the following extensions:

* Azure loT Hub Toolkit
* Azure loT Edge
* Azure loT Device Workbench

#### Installation

By installing the Azure IoT Tools extension you will install all three of the extensions listed above. Some of these extensions will have a dependency on the Azure Account extension to provide a single Azure login and subscription filtering experience.

You can easily uninstall individual extensions if you are not interested in using them, without affecting other extensions provided by this pack. You can uninstall all of the extensions by uninstalling the Azure Tools extension.

#### Azure loT Hub Toolkit

Azure loT Hub Toolkit extension provides everything that you need to start building IoT applications:

* IoT Hub Management
* Device Management
* Module management
* Interact with IoT Hub
* Interact with Azure IoT Edge
* Endpoints management

For more information on the Azure loT Hub Toolkit for Visual Studio Code, see: <https://github.com/Microsoft/vscode-azure-iot-toolkit/wiki>

#### Azure loT Edge

Azure IoT Edge extension makes it easy to code, build, deploy, and debug your IoT Edge solutions in Visual Studio Code, by providing a rich set of functionalities:

* Create new IoT Edge solution
* Add new IoT Edge module to Edge solution
* Build and publish IoT Edge modules
* Debug IoT Edge modules locally and remotely
* IntelliSense and code snippets for the deployment manifest
* Manage IoT Edge devices and modules in IoT Hub (with Azure IoT Toolkit)
* Deploy IoT solutions to IoT Edge devices

For more information about the Azure IoT Edge for Visual Studio Code, see: <https://github.com/Microsoft/vscode-azure-iot-edge/blob/master/README.md>

#### Azure loT Device Workbench

The Azure IoT Device Workbench is a Visual Studio Code extension that provides an integrated environment to code, build, deploy, and debug your IoT device project with multiple Azure services supported. The extension also supports working with IoT Plug and Play by defining device capability model schemas and generating skeleton device code and projects.

For more information on the Azure IoT Device Workbench for Visual Studio Code, see: <https://github.com/Microsoft/vscode-iot-workbench/blob/master/README.md>

### Azure CLI Tools

The Azure command-line interface (CLI) is Microsoft's cross-platform command-line experience for managing Azure resources. The Azure CLI is designed to be easy to learn and get started with, but powerful enough to be a great tool for building custom automation to use Azure resources.

From a purely IoT perspective, Azure CLI enables you to manage Azure IoT Hub resources, Device Provisioning service instances, and linked-hubs out of the box. The IoT extension enriches Azure CLI with features such as device management and full IoT Edge capability.

You can access the full list of Azure CLI resources from the Microsoft Docs site here: <https://docs.microsoft.com/en-us/cli/azure/?view=azure-cli-latest>

#### Command-line Tools

Azure CLI commands can be run from within a large number of command-line environments available for Windows, Linux, and the MacOS.

**Note**: In scripts and on the Microsoft documentation site, Azure CLI examples are written for the bash shell. One-line examples will run on any platform. Longer examples which include line continuations (\) or variable assignment need to be modified to work on other shells, including PowerShell.

When running in a Windows environment, two convenient options for running Azure CLI commands are:

* Azure Cloud Shell
* Windows Command Prompt app

**Note**: You need administrator privileges to run Azure CLI commands from within a Windows Command Prompt window, so select **Run as administrator** when you open the Command Prompt app.

#### Install or run in Azure Cloud Shell

Instructions for installing Azure CLI on your local machine can be found here: <https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>

The Azure Cloud Shell environment, which is accessible through the Azure portal, is perhaps the easiest way to get started with Azure CLI. To learn about Cloud Shell, try out the following Quickstart activity: <https://docs.microsoft.com/en-us/azure/cloud-shell/quickstart>.

After installing the CLI for the first time, check that it's installed and you've got the correct version by running az –version.

#### Sign in

Before using any CLI commands with a local install, you need to sign in with the az login command. For security reasons, there are a couple steps that you will need to complete.

1. At the command prompt, enter the login command:

az login

If the CLI can open your default browser, it will do so and load an Azure sign-in page.

Otherwise, open a browser page at <https://aka.ms/devicelogin> and enter the authorization code displayed in your terminal.

1. In the browser, follow the on-screen instructions to sign in with your account credentials.

After logging in, you see a list of subscriptions associated with your Azure account. The subscription information with isDefault: true is the currently activated subscription after logging in.

To select a different subscription, use the az account set command with the subscription ID for the subscription that you want to switch to. For more information about subscription selection, see <https://docs.microsoft.com/en-us/cli/azure/manage-azure-subscriptions-azure-cli?view=azure-cli-latest>.

#### Extensions

The Azure CLI offers the capability to load extensions. Extensions are Python wheels that aren't shipped as part of the CLI but run as CLI commands. With extensions, you gain access to experimental and pre-release commands along with the ability to write your own CLI interfaces. This article covers how to manage extensions and answers common questions about their use.

##### Find extensions

To see the extensions provided and maintained by Microsoft, use the az extension list-available command.

az extension list-available --output table

##### Install extensions

Once you have found an extension to install, use az extension add to get it. If the extension is listed in az extension list-available, you can install the extension by name.

az extension add --name <extension-name>

To install the extension for IoT, use the following command:

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

##### Update extensions

If an extension was installed by name, update it using az extension update.

az extension update --name <extension-name>

##### Uninstall extensions

If you no longer need an extension, remove it with az extension remove.

### Azure CLI Support for IoT Hub

Azure CLI commands can be used to accomplish many of the tasks associated with managing devices and the Azure IoT Hub service.

#### Azure CLI Commands for IoT Hub

Azure CLI commands for the IoT Hub service include commands that work directly on the hub, as well as commands that work on a subgroup of the hub.

##### Hub Commands

The following commands can be used to complete a task associated with an IoT Hub.

| **IoT Hub Commands** | **Command Description** |
| --- | --- |
| create | Create an Azure IoT hub. |
| delete | Delete an IoT hub. |
| generate-sas-token | Generate a SAS token for a target IoT Hub, device or module. |
| invoke-device-method | Invoke a device method. |
| invoke-module-method | Invoke an Edge module method. |
| list | List IoT hubs. |
| list-skus | List available pricing tiers. |
| manual-failover | Initiate a manual failover for the IoT Hub to the geo-paired disaster recovery region. |
| monitor-events | Monitor device telemetry & messages sent to an IoT Hub. |
| monitor-feedback | Monitor feedback sent by devices to acknowledge cloud-to-device (C2D) messages. |
| query | Query an IoT Hub using a powerful SQL-like language. |
| show | Get the details of an IoT hub. |
| show-connection-string | Show the connection strings for an IoT hub. |
| show-quota-metrics | Get the quota metrics for an IoT hub. |
| show-stats | Get the statistics for an IoT hub. |
| update | Update metadata for an IoT hub. |

##### Command Usage

For usage and help content for any command, append the --help parameter to the name of the command as follows:

az iot hub <command name> --help

For example, to see the usage instruction for the create command, enter the following:

az iot hub create --help

When you run the above command, you will see a message displayed that is similar to the following:

Command

az iot hub create : Create an Azure IoT hub.

For an introduction to Azure IoT Hub, see https://docs.microsoft.com/azure/iot-hub/.

Arguments

--name -n [Required] : IoT Hub name.

--resource-group -g [Required] : Name of resource group. You can configure

the default group using `az configure

--defaults group=<name>`.

--c2d-max-delivery-count --cdd : The number of times the IoT hub will

attempt to deliver a cloud-to-device

message to a device, between 1 and 100.

Default: 10.

--c2d-ttl --ct : The amount of time a message is available

for the device to consume before it is

expired by IoT Hub, between 1 and 48 hours.

Default: 1.

--fc --fileupload-storage-container-name : The name of the root container where you

upload files. The container need not exist

but should be creatable using the

connectionString specified.

--fcs --fileupload-storage-connectionstring : The connection string for the Azure Storage

account to which files are uploaded.

--fd --feedback-max-delivery-count : The number of times the IoT hub attempts to

deliver a message on the feedback queue,

between 1 and 100. Default: 10.

--feedback-lock-duration --fld : The lock duration for the feedback queue,

between 5 and 300 seconds. Default: 5.

--feedback-ttl --ft : The period of time for which the IoT hub

will maintain the feedback for expiration

or delivery of cloud-to-device messages,

between 1 and 48 hours. Default: 1.

--fileupload-notification-max-delivery-count --fnd : The number of times the IoT hub will

attempt to deliver a file notification

message, between 1 and 100. Default: 10.

--fileupload-notification-ttl --fnt : The amount of time a file upload

notification is available for the service

to consume before it is expired by IoT Hub,

between 1 and 48 hours. Default: 1.

--fileupload-notifications --fn : A boolean indicating whether to log

information about uploaded files to the

messages/servicebound/filenotifications IoT

Hub endpoint. Allowed values: false, true.

--fileupload-sas-ttl --fst : The amount of time a SAS URI generated by

IoT Hub is valid before it expires, between

1 and 24 hours. Default: 1.

--location -l : Location of your IoT Hub. Default is the

location of target resource group.

--partition-count : The number of partitions of the backing

Event Hub for device-to-cloud messages.

Default: 2.

--rd --retention-day : Specifies how long this IoT hub will

maintain device-to-cloud events, between 1

and 7 days. Default: 1.

--sku : Pricing tier for Azure IoT Hub. Default

value is F1, which is free. Note that only

one free IoT hub instance is allowed in

each subscription. Exception will be thrown

if free instances exceed one. Allowed

values: B1, B2, B3, F1, S1, S2, S3.

Default: F1.

--unit : Units in your IoT Hub. Default: 1.

Global Arguments

--debug : Increase logging verbosity to show all

debug logs.

--help -h : Show this help message and exit.

--output -o : Output format. Allowed values: json,

jsonc, none, table, tsv, yaml. Default:

json.

--query : JMESPath query string. See

http://jmespath.org/ for more information

and examples.

--subscription : Name or ID of subscription. You can

configure the default subscription using

`az account set -s NAME\_OR\_ID`.

--verbose : Increase logging verbosity. Use --debug for

full debug logs.

Examples

Create an IoT Hub with the free pricing tier F1, in the region of the resource group.

az iot hub create --resource-group MyResourceGroup --name MyIotHub

Create an IoT Hub with the standard pricing tier S1 and 4 partitions, in the 'westus' region.

az iot hub create --resource-group MyResourceGroup --name MyIotHub --sku S1 --location

westus --partition-count 4

For more specific examples, use: az find "az iot hub create"

##### IoT Hub Subgroups

The following IoT Hub subgroups can be used to complete tasks that are associated with the subgroup.

| **IoT Hub Subgroups** | **Subgroup Description** |
| --- | --- |
| certificate | Manage IoT Hub certificates. |
| configuration | Manage IoT device configurations at scale. |
| consumer-group | Manage the event hub consumer groups of an IoT hub. |
| device-identity | Manage IoT devices. |
| device-twin | Manage IoT device twin configuration. |
| devicestream | Manage device streams of an IoT hub. |
| distributed-tracing [Preview] | Manage distributed settings per-device. |
| job | Manage jobs in an IoT hub. |
| message-enrichment | Manage message enrichments for endpoints of an IoT Hub. |
| module-identity | Manage IoT device modules. |
| module-twin | Manage IoT device module twin configuration. |
| policy | Manage shared access policies of an IoT hub. |
| route | Manage routes of an IoT hub. |
| routing-endpoint | Manage custom endpoints of an IoT hub. |

The commands available by accessing each of these subgroup categories can be viewed by running a command that appends --help to the subgroup name as follows:

az iot hub <subgroup name> --help

For example, if you run the command above for the device-identity subgroup, you will see a message displayed that is similar to the following:

az iot hub device-identity : Manage IoT devices.

Commands:

add-children : Add specified comma-separated list of non edge device ids as children

of specified edge device.

create : Create a device in an IoT Hub.

delete : Delete an IoT Hub device.

export : Export all device identities from an IoT Hub to an Azure Storage blob

container.

get-parent : Get the parent device of the specified device.

import : Import device identities to an IoT Hub from a blob.

list : List devices in an IoT Hub.

list-children : Print comma-separated list of assigned child devices.

remove-children : Remove non edge devices as children from specified edge device.

set-parent : Set the parent device of the specified non-edge device.

show : Get the details of an IoT Hub device.

show-connection-string : Show a given IoT Hub device connection string.

update : Update an IoT Hub device.

### Overview of Azure Cloud Shell

Azure Cloud Shell is an interactive, authenticated, browser-accessible shell for managing Azure resources. It provides the flexibility of choosing the shell experience that best suits the way you work, either Bash or PowerShell.

#### Features

##### Browser-based shell experience

Cloud Shell enables access to a browser-based command-line experience built with Azure management tasks in mind. Leverage Cloud Shell to work untethered from a local machine in a way only the cloud can provide.

##### Choice of preferred shell experience

Users can choose between Bash or PowerShell.

##### Authenticated and configured Azure workstation

Cloud Shell is managed by Microsoft so it comes with popular command-line tools and language support. Cloud Shell also securely authenticates automatically for instant access to your resources through the Azure CLI or Azure PowerShell cmdlets.

View the full list of tools installed in Cloud Shell.

##### Integrated Cloud Shell editor

Cloud Shell offers an integrated graphical text editor based on the open-source Monaco Editor. Simply create and edit configuration files by running code . for seamless deployment through Azure CLI or Azure PowerShell.

Learn more about the Cloud Shell editor.

##### Integrated with docs.microsoft.com

You can use Cloud Shell directly from documentation hosted on [docs.microsoft.com](https://docs.microsoft.com/). It is integrated in [Microsoft Learn](https://docs.microsoft.com/learn/), [Azure PowerShell](https://docs.microsoft.com/powershell/azure/overview) and [Azure CLI documentation](https://docs.microsoft.com/cli/azure) - click on the “Try It” button in a code snippet to open the immersive shell experience.

##### Multiple access points

Cloud Shell is a flexible tool that can be used from:

* [portal.azure.com](https://portal.azure.com/)
* [shell.azure.com](https://shell.azure.com/)
* [Azure CLI documentation](https://docs.microsoft.com/cli/azure)
* [Azure PowerShell documentation](https://docs.microsoft.com/powershell/azure/overview)
* [Azure mobile app](https://azure.microsoft.com/features/azure-portal/mobile-app/)
* [Visual Studio Code Azure Account extension](https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account)

##### Connect your Microsoft Azure Files storage

Cloud Shell machines are temporary, but your files are persisted in two ways: through a disk image, and through a mounted file share named clouddrive. On first launch, Cloud Shell prompts to create a resource group, storage account, and Azure Files share on your behalf. This is a one-time step and will be automatically attached for all sessions. A single file share can be mapped and will be used by both Bash and PowerShell in Cloud Shell.

Read more to learn how to mount a new or existing storage account or to learn about the persistence mechanisms used in Cloud Shell.

[!NOTE] Azure storage firewall is not supported for cloud shell storage accounts.

#### Concepts

* Cloud Shell runs on a temporary host provided on a per-session, per-user basis
* Cloud Shell times out after 20 minutes without interactive activity
* Cloud Shell requires an Azure file share to be mounted
* Cloud Shell uses the same Azure file share for both Bash and PowerShell
* Cloud Shell is assigned one machine per user account
* Cloud Shell persists $HOME using a 5-GB image held in your file share
* Permissions are set as a regular Linux user in Bash

Learn more about features in Bash in Cloud Shell and PowerShell in Cloud Shell.

#### Pricing

The machine hosting Cloud Shell is free, with a pre-requisite of a mounted Azure Files share. Regular storage costs apply.

### Device Communication

IoT Hub allows for bi-directional communication with your devices. Use IoT Hub messaging to communicate with your devices by sending messages from your devices to your solutions back end and sending commands from your IoT solutions back end to your devices. Learn more about the IoT Hub message format.

* Sending device-to-cloud messages to IoT Hub: IoT Hub has a built-in service endpoint that can be used by back-end services to read telemetry messages from your devices. This endpoint is compatible with Event Hubs and you can use standard IoT Hub SDKs to read from this built-in endpoint.
* Sending cloud-to-device messages from IoT Hub: You can send cloud-to-device messages from the solution back end to your devices. Cloud-to-device messaging is only available in the standard tier of IoT Hub.

Core properties of IoT Hub messaging functionality are the reliability and durability of messages. These properties enable resilience to intermittent connectivity on the device side, and to load spikes in event processing on the cloud side. IoT Hub implements at least once delivery guarantees for both device-to-cloud and cloud-to-device messaging.

#### Device-to-Cloud Communications

When sending information from the device app to the solution back end, IoT Hub exposes three options:

* Device-to-cloud messages for time series telemetry and alerts.
* Device twin's reported properties for reporting device state information such as available capabilities, conditions, or the state of long-running workflows. For example, configuration and software updates.
* File uploads for media files and large telemetry batches uploaded by intermittently connected devices or compressed to save bandwidth.

#### Cloud-to-Device Communications

IoT Hub provides three options for communicating to a device from a back-end app:

* Direct methods for communications that require immediate confirmation of the result. Direct methods are often used for interactive control of devices such as turning on a fan.
* Twin's desired properties for long-running commands intended to put the device into a certain desired state. For example, set the telemetry send interval to 30 minutes.
* Cloud-to-device messages for one-way notifications to the device app.

### Communication Protocols

The following table provides the high-level recommendations for your choice of protocol:

| **Protocol** | **When you should use this protocol** |
| --- | --- |
| MQTT MQTT over WebSocket | Use on all devices that do not require to connect multiple devices (each with its own per-device credentials) over the same TLS connection. |
| AMQP AMQP over Websocket | Use on field and cloud gateways to take advantage of connection multiplexing across devices. |
| HTTPS | Use for devices that cannot support other protocols. |

#### Port numbers

Devices can communicate with IoT Hub in Azure using various protocols. Typically, the choice of protocol is driven by the specific requirements of the solution. The following table lists the outbound ports that must be open for a device to be able to use a specific protocol:

| **Protocol** | **Port** |
| --- | --- |
| MQTT | 8883 |
| MQTT over WebSockets | 443 |
| AMQP | 5671 |
| AMQP over WebSockets | 443 |
| HTTPS | 443 |

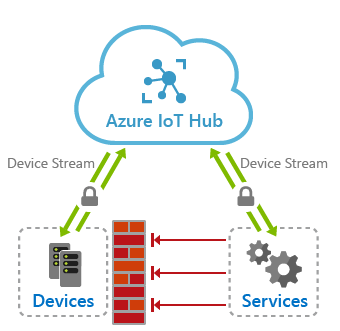
#### Considerations

Consider the following points when you choose your protocol for device-side communications:

* Cloud-to-device pattern. HTTPS does not have an efficient way to implement server push. As such, when you are using HTTPS, devices poll IoT Hub for cloud-to-device messages. This approach is inefficient for both the device and IoT Hub. Under current HTTPS guidelines, each device should poll for messages every 25 minutes or more. MQTT and AMQP support server push when receiving cloud-to-device messages. They enable immediate pushes of messages from IoT Hub to the device. If delivery latency is a concern, MQTT or AMQP are the best protocols to use. For rarely connected devices, HTTPS works as well.
* Field gateways. When using MQTT and HTTPS, you cannot connect multiple devices (each with its own per-device credentials) using the same TLS connection. For Field gateway scenarios that require one TLS connection between the field gateway and IoT Hub for each connected device, these protocols are suboptimal.
* Low resource devices. The MQTT and HTTPS libraries have a smaller footprint than the AMQP libraries. As such, if the device has limited resources (for example, less than 1-MB RAM), these protocols might be the only protocol implementation available.
* Network traversal. The standard AMQP protocol uses port 5671, and MQTT listens on port 8883. USe of these ports could cause problems in networks that are closed to non-HTTPS protocols. Use MQTT over WebSockets, AMQP over WebSockets, or HTTPS in this scenario.
* Payload size. MQTT and AMQP are binary protocols, which result in more compact payloads than HTTPS.

### IoT Hub Device Streams (Preview)

Azure IoT Hub device streams facilitate the creation of secure bi-directional TCP tunnels for a variety of cloud-to-device communication scenarios. A device stream is mediated by an IoT Hub streaming endpoint which acts as a proxy between your device and service endpoints. This setup, depicted in the diagram below, is especially useful when devices are behind a network firewall or reside inside of a private network. As such, IoT Hub device streams help address customers' need to reach IoT devices in a firewall-friendly manner and without the need to broadly opening up incoming or outgoing network firewall ports.



Using IoT Hub device streams, devices remain secure and will only need to open up outbound TCP connections to IoT hub's streaming endpoint over port 443. Once a stream is established, the service-side and device-side applications will each have programmatic access to a WebSocket client object to send and receive raw bytes to one another. The reliability and ordering guarantees provided by this tunnel is on par with TCP.

#### Benefits

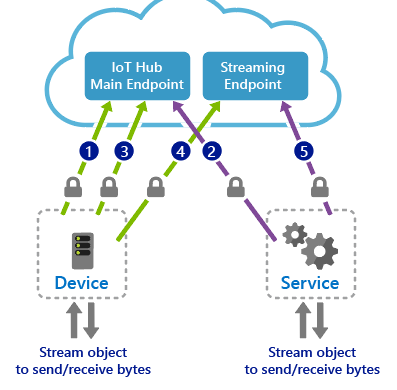
IoT Hub device streams provide the following benefits:

* Firewall-friendly secure connectivity: IoT devices can be reached from service endpoints without opening of inbound firewall port at the device or network perimeters (only outbound connectivity to IoT Hub is needed over port 443).
* Authentication: Both device and service sides of the tunnel need to authenticate with IoT Hub using their corresponding credentials.
* Encryption: By default, IoT Hub device streams use TLS-enabled connections. This ensures that the traffic is always encrypted regardless of whether the application uses encryption or not.
* Simplicity of connectivity: In many cases, the use of device streams eliminates the need for complex setup of Virtual Private Networks to enable connectivity to IoT devices.
* Compatibility with TCP/IP stack: IoT Hub device streams can accommodate TCP/IP application traffic. This means that a wide range of proprietary as well as standards-based protocols can leverage this feature.
* Ease of use in private network setups: Service can communicate with a device by referencing its device ID, rather than device's IP address. This is useful in situations where a device is located inside a private network and has a private IP address, or its IP address is assigned dynamically and is unknown to the service side.

#### Device stream workflows

A device stream is initiated when the service requests to connect to a device by providing its device ID. This workflow particularly fits into a client/server communication model, including SSH and RDP, where a user intends to remotely connect to the SSH or RDP server running on the device using an SSH or RDP client program.

The device stream creation process involves a negotiation between the device, service, IoT hub's main and streaming endpoints. While IoT hub's main endpoint orchestrates the creation of a device stream, the streaming endpoint handles the traffic that flows between the service and device.



1. The device application registers a callback in advance to be notified of when a new device stream is initiated to the device. This step typically takes place when the device boots up and connects to IoT Hub.
2. The service-side program initiates a device stream when needed by providing the device ID (not the IP address).
3. IoT hub notifies the device-side program by invoking the callback registered in step 1. The device may accept or reject the stream initiation request. This logic can be specific to your application scenario. If the stream request is rejected by the device, IoT Hub informs the service accordingly; otherwise, the steps below follow.
4. The device creates a secure outbound TCP connection to the streaming endpoint over port 443 and upgrades the connection to a WebSocket. The URL of the streaming endpoint as well as the credentials to use to authenticate are both provided to the device by IoT Hub as part of the request sent in step 3.
5. The service is notified of the result of device accepting the stream and proceeds to create its own WebSocket client to the streaming endpoint. Similarly, it receives the streaming endpoint URL and authentication information from IoT Hub.

#### Connectivity Requirements

Both the device and the service sides of a device stream must be capable of establishing TLS-enabled connections to IoT Hub and its streaming endpoint. This requires outbound connectivity over port 443 to these endpoints. The hostname associated with these endpoints can be found on the Overview tab of IoT Hub.

### Device-Side Code Implementation

Along with the Device SDK, Microsoft has developed sample code projects that illustrate common coding tasks.

The Sample Projects for C# include:

* DeviceStreamingSample
* FileUploadSample
* ImportExportDevicesSample
* KeysRolloverSample
* MessageSample
* MethodSample
* TwinSample
* XamarinSample

#### MessageSample project

The MessageSample project includes the following code files:

* MessageSample.cs
* MessageSample.csproj
* Program.cs

##### Examine Program.cs

The contents of the Program.cs file should be similar to the code shown below. Look for the following within this code:

* This class uses a device connection string
* The **DeviceClient** class is used for communication between the device and the **IoT Hub**

// Copyright (c) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE file in the project root for full license information.

using System;

namespace Microsoft.Azure.Devices.Client.Samples

{

public class Program

{

// String containing Hostname, Device Id & Device Key in one of the following formats:

// "HostName=<iothub\_host\_name>;DeviceId=<device\_id>;SharedAccessKey=<device\_key>"

// "HostName=<iothub\_host\_name>;CredentialType=SharedAccessSignature;DeviceId=<device\_id>;SharedAccessSignature=SharedAccessSignature sr=<iot\_host>/devices/<device\_id>&sig=<token>&se=<expiry\_time>";

// For this sample either

// - pass this value as a command-prompt argument

// - set the IOTHUB\_DEVICE\_CONN\_STRING environment variable

// - create a launchSettings.json (see launchSettings.json.template) containing the variable

private static string s\_deviceConnectionString = Environment.GetEnvironmentVariable("IOTHUB\_DEVICE\_CONN\_STRING");

// Select one of the following transports used by DeviceClient to connect to IoT Hub.

private static TransportType s\_transportType = TransportType.Amqp;

//private static TransportType s\_transportType = TransportType.Mqtt;

//private static TransportType s\_transportType = TransportType.Http1;

//private static TransportType s\_transportType = TransportType.Amqp\_WebSocket\_Only;

//private static TransportType s\_transportType = TransportType.Mqtt\_WebSocket\_Only;

public static int Main(string[] args)

{

if (string.IsNullOrEmpty(s\_deviceConnectionString) && args.Length > 0)

{

s\_deviceConnectionString = args[0];

}

DeviceClient deviceClient = DeviceClient.CreateFromConnectionString(s\_deviceConnectionString, s\_transportType);

if (deviceClient == null)

{

Console.WriteLine("Failed to create DeviceClient!");

return 1;

}

var sample = new MessageSample(deviceClient);

sample.RunSampleAsync().GetAwaiter().GetResult();

Console.WriteLine("Done.\n");

return 0;

}

}

}

The CreateFromConnectionString static method on the DeviceClient class accepts connection string and protocol parameters, and the return type is **DeviceClient**. If successful, the function will return an instance of the **DeviceClient** class. If it fails, it will return null. We use the generated IoT Hub client when invoking other functions.

Notice that we are passing the device connection string and that we also designate the communications protocol that we will be using (this example uses AMQP, but MQTT and HTTPS are also options). If the client is null, we log to the console, otherwise we continue as noted above.

**Tip:** Obviously, logging to the console is very poor error handling. In a production system, this should be logged to a storage location that is monitored, or has alerts delivered to support technicians.

You can obtain the device connection string from the Azure portal.

**Note**: Once you have a valid **DeviceClient** object, you can start calling the APIs to send and receive messages to and from IoT Hub.

##### Examine the MessageSample.cs file

The contents of the MessageSample.cs file should be similar to the code shown below.

// Copyright (c) Microsoft. All rights reserved.

// Licensed under the MIT license. See LICENSE file in the project root for full license information.

using System;

using System.Text;

using System.Threading.Tasks;

namespace Microsoft.Azure.Devices.Client.Samples

{

public class MessageSample

{

private const int MessageCount = 5;

private const int TemperatureThreshold = 30;

private static Random s\_randomGenerator = new Random();

private float \_temperature;

private float \_humidity;

private DeviceClient \_deviceClient;

public MessageSample(DeviceClient deviceClient)

{

\_deviceClient = deviceClient ?? throw new ArgumentNullException(nameof(deviceClient));

}

public async Task RunSampleAsync()

{

await SendEvent().ConfigureAwait(false);

await ReceiveCommands().ConfigureAwait(false);

}

private async Task SendEvent()

{

string dataBuffer;

Console.WriteLine("Device sending {0} messages to IoTHub...\n", MessageCount);

for (int count = 0; count < MessageCount; count++)

{

\_temperature = s\_randomGenerator.Next(20, 35);

\_humidity = s\_randomGenerator.Next(60, 80);

dataBuffer = $"{{\"messageId\":{count},\"temperature\":{\_temperature},\"humidity\":{\_humidity}}}";

Message eventMessage = new Message(Encoding.UTF8.GetBytes(dataBuffer));

eventMessage.Properties.Add("temperatureAlert", (\_temperature > TemperatureThreshold) ? "true" : "false");

Console.WriteLine("\t{0}> Sending message: {1}, Data: [{2}]", DateTime.Now.ToLocalTime(), count, dataBuffer);

await \_deviceClient.SendEventAsync(eventMessage).ConfigureAwait(false);

}

}

private async Task ReceiveCommands()

{

Console.WriteLine("\nDevice waiting for commands from IoTHub...\n");

Console.WriteLine("Use the IoT Hub Azure Portal to send a message to this device.\n");

Message receivedMessage;

string messageData;

receivedMessage = await \_deviceClient.ReceiveAsync(TimeSpan.FromSeconds(30)).ConfigureAwait(false);

if (receivedMessage != null)

{

messageData = Encoding.ASCII.GetString(receivedMessage.GetBytes());

Console.WriteLine("\t{0}> Received message: {1}", DateTime.Now.ToLocalTime(), messageData);

int propCount = 0;

foreach (var prop in receivedMessage.Properties)

{

Console.WriteLine("\t\tProperty[{0}> Key={1} : Value={2}", propCount++, prop.Key, prop.Value);

}

await \_deviceClient.CompleteAsync(receivedMessage).ConfigureAwait(false);

}

else

{

Console.WriteLine("\t{0}> Timed out", DateTime.Now.ToLocalTime());

}

}

}

}

Notice that the SendEvent method will generate random weather data, and send messages to the **IoT Hub**. The **MessageCount** property is set to **5**, which will send 5 messages in the **for** loop.

Inspecting the body of the **for** loop, we see that the **\_temperature**, and **\_humidity** variables are set to randomly generated numbers. The temperature between **20** and **35**, whereas the humidity is between **60** and **80**. Next, the **dataBuffer** variable is a **JSON** representation of the data. In the previous lab, you serialized a dynamic object into **JSON**, and this example manually creates the **JSON**. It does not matter which approach you take, but I have found that serializing an object is more accurate, and fewer issues arise.

In the next three lines the message is created, and a custom property is added to the message. Then, a message is written to the console that indicates a message is going out, the current date and time, the message number, and the message body.

The last line in the **for** loop sends the message to the **IoT Hub** by calling the SendEventAsync method.

**Tip:** One thing that is missing from this sample is any error handling. If communication issues arise, or the connection string is incorrect, this application will not fail gracefully. It is usually a good idea to include some basic error handling, even in prototypes applications.

### Module Labs

Your course instructor will provide guidance on accessing the module lab instructions

* Lab 3: Setup the Development Environment
* Lab 4: Connect IoT Device to Azure

**WARNING**: Be prepared for UI changes

Given the dynamic nature of Microsoft cloud tools, you may experience user interface (UI) changes that were made following the development of this training content that do not match up with lab instructions presented in this lab manual.

The Microsoft Learning team will update this training course as soon as we can when any such changes are brought to our attention. However, given the dynamic nature of cloud updates, you may run into UI changes before this training content is updated. If this occurs, you will have to adapt to the changes and work through them in the labs as needed.

#### Lab 3 - Scenario and Architecture

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](https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/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

#### Lab 4 - Scenario and Architecture

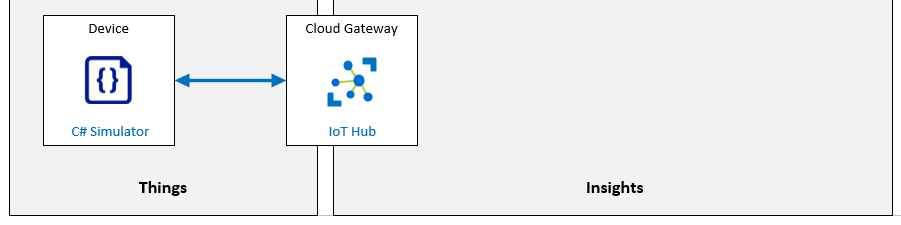
Contoso is known for producing high quality cheeses. Due to the company's rapid growth in both popularity and sales, they want to take steps to ensure that their cheeses stay at the same high level of quality that their customers expect.

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.

The following resources will be created:



### Module Review Questions

#### Review Question 2.1

You have joined a team that is developing an IoT solution for your company. You will be implementing IoT Hub as part of your solution. The planning docs that you reviewed indicate that the solution will need to process a little over 5 million messages per day and will include both IoT Edge and regular IoT devices. The plans call for using the Basic tier, specify the B1 tier edition, and call for two units. When you think about it, this doesn’t seem to be correct. What tier and edition of IoT hub will be needed and how many units? (choose one best answer)

* B1 with 8 units.
* B2 with 1 unit.
* S1 with 8 units.
* S2 with 1 unit.

Check Answers

#### Review Question 2.2

You have joined a team that is developing an IoT solution for your company. You will be implementing IoT Hub as part of your solution. You are told that Device Twins will be used to help configure and manage devices throughout the device lifecycle. Which of the following statements about Device Twins are true? (choose all correct answers)

* Device twins are JSON documents maintained by IoT Hub.
* Each device that is registered with IoT Hub has a Device Twin.
* Device Twins include Tags, Cloud properties, Device properties, and Communication properties.
* Device Twins include Tags, Desired properties, Reported properties, and Device identity properties.
* Device Twins include Tags, Desired properties, Reported properties, and Lifecycle properties.

Check Answers

#### Review Question 2.3

You have joined a team that is developing an IoT solution for your company. You are using the Azure Cloud Shell and Azure CLI commands to create some resources. You open the Cloud Shell and ensure that Bash is selected as the environment. You run the following two commands in the Azure Cloud Shell: "az group create --name MyAZ220RG --location westus" and "az iot hub create --resource-group MyAZ220RG --name MyIotHub". What will result when you run the commands? (choose all correct answers)

* The first command will fail to create a resource group because no subscription is provided.
* The first command will create a resource group named MyAZ220RG in the westus region.
* The second command will fail because no pricing tier is specified.
* The second command will fail because no region is specified.
* The second command will create an IoT Hub using the F1 pricing tier in the region of the resource group.

Check Answers

#### Review Question 2.4

You have joined a team that is developing an IoT solution for your company. You are investigating how to implement 2-way communication between devices and IoT Hub. Which of the following are common examples of sending information from devices to the cloud? (choose all correct answers)

* Device-to-cloud messages for scheduling firmware updates.
* Device-to-cloud messages for time series telemetry and alerts.
* Device twin's reported properties for reporting device state information.
* Device twin's proposed properties for requesting device state information.
* File uploads for media files and large telemetry batches.

Check Answers

#### Review Question 2.5

You have joined a team that is developing an IoT solution for your company. You are investigating how to implement 2-way communication between devices and IoT Hub. Which of the following are valid protocol options? (choose all correct answers)

* MQTT on port 8883.
* MQTT over WebSockets on port 443.
* AMQP on port 5671.
* AMQP over WebSockets on port 443.
* HTTPS on port 443.