**Module Introduction - Approaches to an IoT Implementation**

This module begins with ...

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

* Lab 1: Exploring the Components of an IoT Solution
* Lab 2: Exploring the Approaches to IoT Business Solutions

**Note**: Future versions of this course will include a video that introduces the instructors for the course and describes the module labs

**IoT Hardware**

In this topic, you'll learn:

* About the different types of IoT hardware used in IoT solutions
* How IoT hardware is being deployed and used in the real world

When thinking about building an IoT solution, perhaps the first area of consideration is what hardware you will need. This partly is 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 often includes a network infrastructure that is used to connect devices. Still, some devices could be stand-alone. How would this work? A sensor, for example, could collect temperature data or collect data about how a bridge is being stressed but not deliver those data immediately to a network database. A technician could come by on a regular schedule, collect the data from the device using an internet-connected tool which then delivers the data to the database.[^1]

[^1]: Though some argue that when a device no longer is able to connect to the internet, the “Internet” part of IoT no longer applies so the device shouldn't be considered an IoT device.

Other hardware involved in a broad IoT solution don't collect data at all but broker communication with other hardware and cloud services. In the following, we'll look at various types of hardware used in IoT solutions and talk about the role they might play.

**IP-enabled 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.

**Examples**

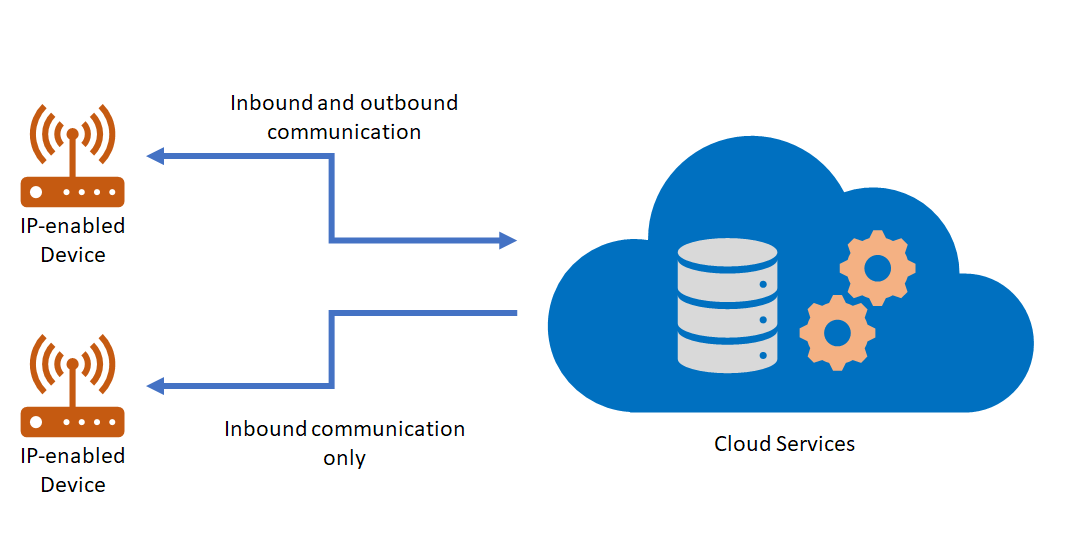
In terms of IoT, an IP-enabled device is one that can connect directly to a network like the internet and transmit or receive data. Examples we commonly think of are the home automation devices like doorbells and thermostats that use an internet connection to communicate with a central server. But industrial-grade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

**Usage**

As you might expect, people deploy IP-enabled devices in scenarios where data needs to be collected, delivered, and analyzed in real-time or near real-time. IP-enabled devices enable live connections to the other networked devices or services so data can be transmitted without interaction from a technician.

Data collection isn't the only reason to use an IP-enabled device. If transmitted data over the internet is particularly risky, your device may not have “outbound” ability enabled but the device still can receive messages (inbound) from the network so its firmware or software can be updated. Or it may have the ability only to send status messages but not any data it has collected.

While it's possible that a device may have only outbound communication abilities over IP, it's not typical since when delivering messages, it is important that the device knows if the receiving node has an error or if messages did not transmit for some reason. So some inbound communication typically is needed.



**Non-IP Enabled Devices**

As mentioned above, it's not necessary for a device to be 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 which we'll discuss below.

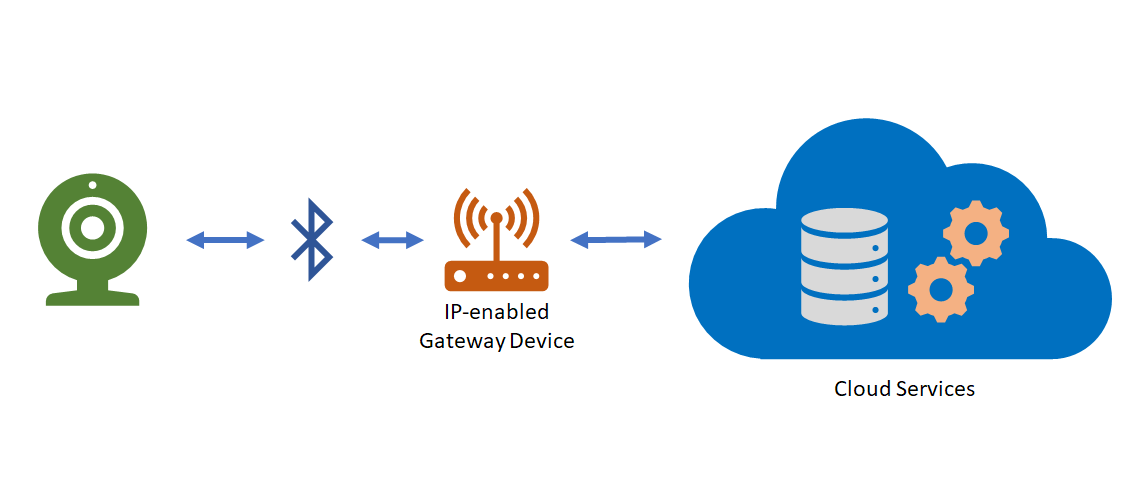
Devices can use industry-specific protocols (such as CoAP5, OPC),and short-range communication technologies (such as Bluetooth, ZigBee) to connect to other hardware.

**Examples**

For example, when setting up an internet connected lock, you may need first to connect the lock to your phone using Bluetooth in order to set up a relationship with a cloud service. While this is a temporary situation, you can imagine a scenario where the device can only connect to a local device using Bluetooth and the secondary device brokers all the communication with the cloud service.

**Usage**

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**

We can break this category into two subcategories: sensors and smart sensors.

The IEEE, in their [definition paper](https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf) on IoT, writes this about **sensors**:

Sensors are one of the key building blocks of IoT. As ubiquitous systems, they can be deployed everywhere – from military battlefields to vineyards and redwoods and on the Golden Gate Bridge. They can also be implanted under human skin, in a purse or on a t-shirt. Some can be as small as four millimeters in size, but the data they collect can be received hundreds of miles away. They complement human senses and have become indispensable in a large number of industries, from health care to construction. Sensors have a key advantage in that they can anticipate human needs based on information collected about their context. Their intelligence, “multiplied” by numerous networks, allows them not only to report about the external environment, but also to take action without human intervention.

Within an intelligent networked system, sensors perform the functions of input devices – they serve as “eyes,” collecting information about their environment.

We can define a sensor, then, as a device that collects a specific type of data about the physical environment. As IoT as a technology grows, the list of available sensors most likely will grow with it. There also are communities that will help you build your own sensors if the one you need doesn't exist.

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 of both types can be embedded on other devices which manages communication with a network or stand alone and handle all the necessary functions needed to collect and communicate data.

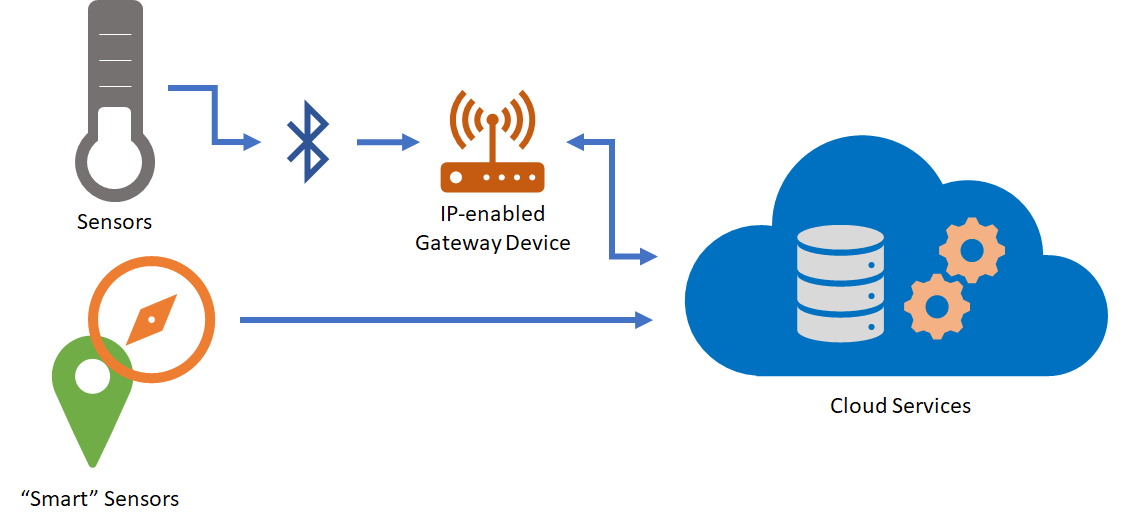
**Examples**

Sensors that can collect data on a wide variety of things actively are being developed. Intel has [compiled a list](https://software.intel.com/en-us/industrial-sensors-with-upm-support) of some sensors currently available. Examples include:

* Temperature
* Humidity
* Energy
* Compass
* Pressure
* Sonar
* Light and UV

**Usage**

As should be clear from the above, sensors are used whenever a discrete bit of information needs to be collected about something in the physical environment.



**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.

**The Growing IoT Device Ecosystem**

This is a brief introduction to the growing world of IoT devices. Which devices you'll need for a solution will depend largely on what data you're interested in collecting how you want to work with that data and what devices are available in the market (or that you have the capability of building).

**IoT Device Software**

In this topic, you'll learn:

* About device operating systems and configurations
* About IoT programming languages and environments
* About IoT software development kits (SDKs) and how to use them when programming IoT hardware.

Like any piece of complex hardware, IoT devices need an operating system in order to be useful. But because IoT devices tend to be small and resource-constrained, operating systems 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 operating systems and programming tools and 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.” |

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 we'll say it: 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.

There are many IoT SDKs you can review and we'll survey a couple here.

**Microsoft IoT SDKs**

Microsoft provides [a number of IoT SDKs](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks) that range across a variety of languages and devices and are designed to provide the software and services needed to accelerate your solution development. The company breaks down their SDKs into three categories:

* **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. The device SDK comes in the following platform flavors: **.NET**, **C**, **Java**, **Node.js**, **Python**, and **iOS**
* **Service SDKs** enable you 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.
* **Device Provisioning SDKs** enable you to provision devices to your IoT Hub using the Device Provisioning Service.

**AWS (Amazon Web Services) IoT Device SDKs**

Amazon's IoT device SDKs cover a variety of platforms and languages including, **Embedded C**, **JavaScript**, **Arduino Yun**, **Python**, **Java**, **C++**, and **iOS**.

The SDK provides developers with the ability to work with Amazon's “Thing Shadows” which are virtual versions of physical devices, as well as interacting with the [MQTT](https://en.wikipedia.org/wiki/MQTT) messaging protocol which is used by AWS services. Similar to Microsoft's offering, each of the various SDKs implement specific features not necessarily shared by other SDKs in the same family. As with all SDKs, architects and developers should examine the SDK they're interested in to ensure it supports the features they'll need for their solution.

There are other SDKs available and, many times, hardware vendors will provide their own SDKs to make development on their platform easier. Once you've chosen a hardware and software platform, investigate the SDKs available for those platforms. You may want to use the availability of an SDK as a part of the decision-making process when evaluating hardware and software platforms.

**Cloud Service Components of an IoT Solution**

In this topic, you'll learn:

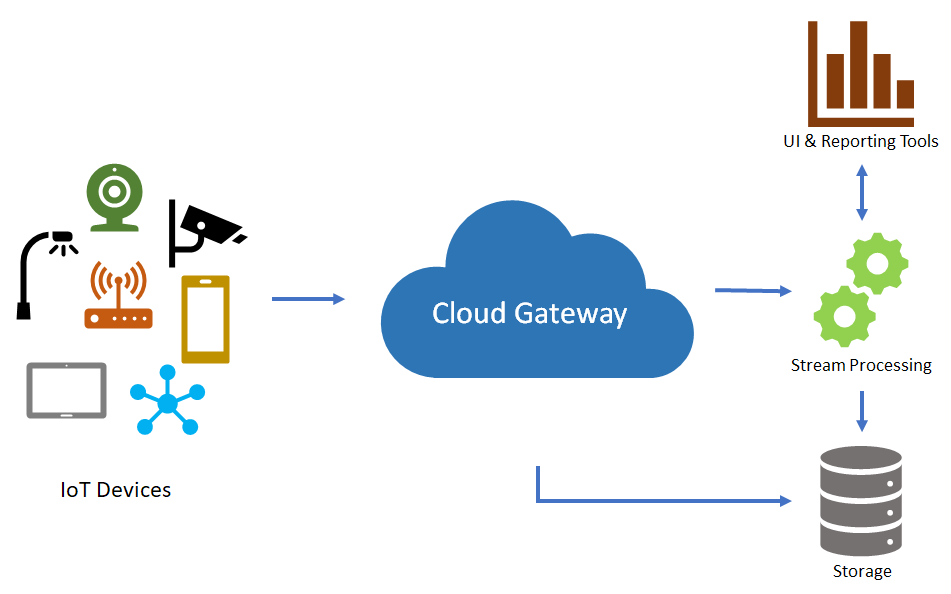
* About cloud-based gateways and storage options
* About cloud-based analytics and data visualization
* About how to use machine learning in IoT solutions

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**

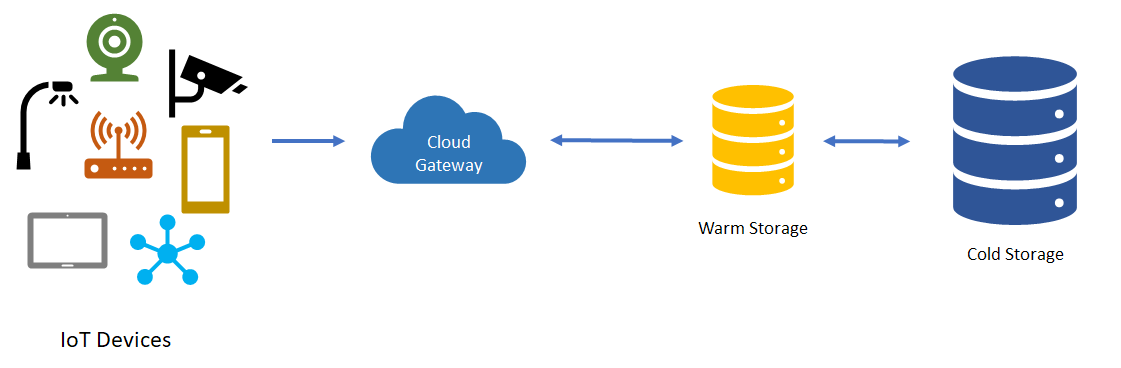
Given the centrality of data in an IoT solution, figuring out the right cloud-based data storage and retrieval options ranks high on the list in terms of importance. IoT devices can generate enormous amounts of data very quickly and storing high volumes of data in the cloud can not become expensive but 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.

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).

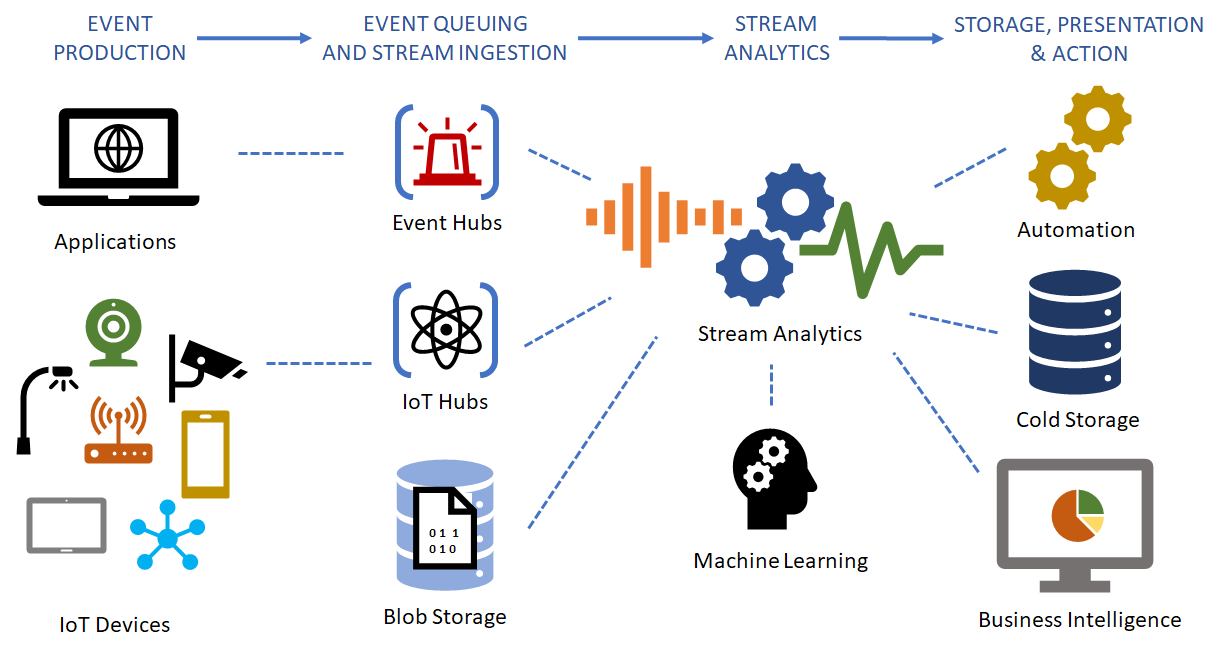
**Analytics Services and Data Visualization**

**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.

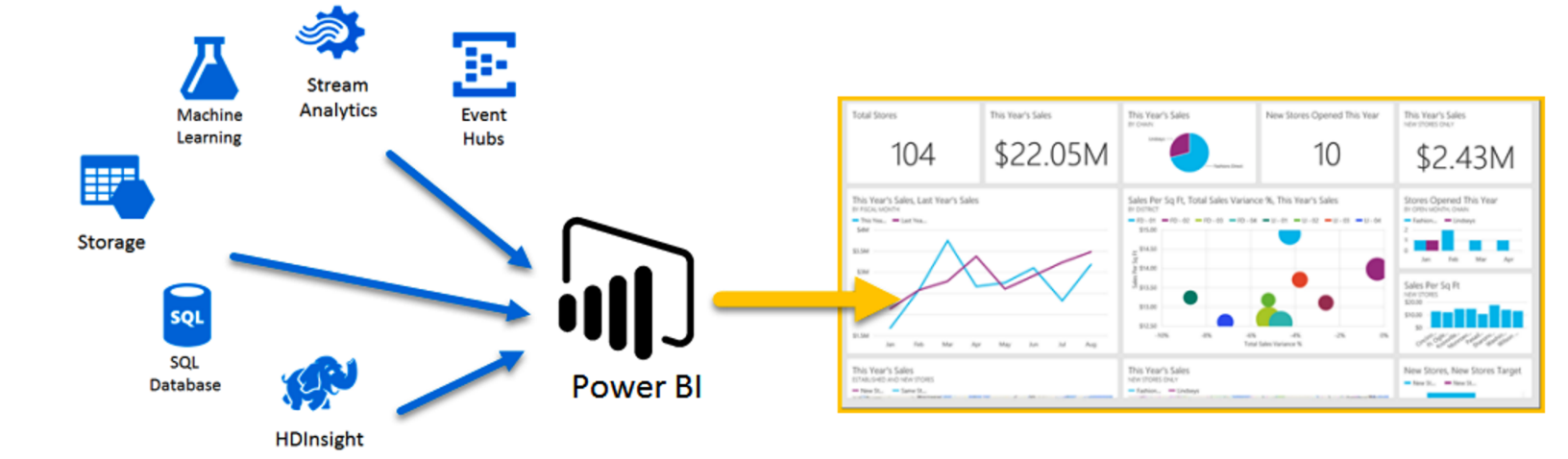
Without analytics, data collected from IoT would be too voluminous and unstructured to visualize or gain insights. Analytic services enable architects to build meaningful relationships between sets of data in order to make it easier to manage. For example, Azure Stream Analytics can take stream data 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 PowerBI 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.