**IoT Implementation Patterns**

In this lesson, we're going to look at examples of specific IoT architectural patterns[^1] and technologies. In this lesson, more than the others in this course, we'll be focusing specifically on Microsoft technology, specifically Azure-based IoT services. While we'll be focusing on Azure in this lesson, other IoT cloud providers offer similar services though their implementations may differ from the way Azure works.

Specifically in this lesson, we'll introduce you to:

* Using Azure IoT Solution Accelerators to get started with your IoT implementation
* Device and cloud implementations focusing on Azure IoT Hub
* Messaging in IoT Hub
* How to configure devices using IoT Hub
* Data Analytics services in Azure
* Using business intelligence and machine learning services in Azure to enhance reporting for your IoT solution

While this material will be introductory, you can explore these concepts more deeply in other courses in this series. These topics will give you a good overview of how these services work and what options are available to you using Microsoft's robust and growing IoT platform.

[^1]: A pattern in software and hardware design is a model or template for a way to build a software architecture or hardware component in part or in whole. A pattern can describe a specific way to do something or can be as general as a best practice. Take software security as an example. A pattern can describe all the component pieces of a secure system like the *type of* hardware that should be in place, the protocols that should be used to send and receive data. This would be an implementation-specific example of a pattern. A pattern also can describe best practices like a software design should include security considerations as core part of the software architecture. See [this Wikipedia Article](https://en.wikipedia.org/wiki/Software_design_pattern) for more information.

**IoT Solution Accelerators**

In this topic, you'll learn:

* What Solution Accelerators are
* How Solution Accelerators are used
* About the limitations of Solution Accelerators

**What are Solution Accelerators?**

The [IoT solution accelerators](https://azure.microsoft.com/en-us/features/iot-accelerators/) are a collection of complete, ready-to-deploy, IoT solutions that implement common IoT scenarios such as remote monitoring, connected factory, predictive maintenance, and device simulation. When you deploy a solution accelerator, the deployment includes all the required cloud-based services along with any required application code. Think of Solution Accelerators as fully scaffolded starter solutions that you can use either to learn about various IoT scenarios or as templates upon which you can build your own.[^1]

[^1]: Note: In order to use the solution accelerators, you will need to have or sign up for an Azure account and deploying an accelerator will incur charges on Azure.

These accelerators have been developed my Microsoft and are built on Azure IoT services. As of this writing, the following Solution Accelerators are available:

1. **Remote Monitoring**: Use this solution accelerator to collect telemetry from multiple remote devices and to control them. Example devices include cooling systems installed on your customers' premises or valves installed in remote pump stations.
2. **Connected Factory**: Use this solution accelerator to collect telemetry from industrial assets with an OPC Unified Architecture (a standards-based, service-oriented framework) interface and to control them. Industrial assets might include assembly and test stations on a factory production line.
3. **Predictive Maintenance**: Use this solution accelerator to predict when a remote device is expected to fail so you can carry out maintenance before the predicted failure happens. This solution accelerator uses machine learning algorithms to predict failures from device telemetry. Example devices might be airplane engines or elevators.
4. **Device Simulation**: Use this solution accelerator to run multiple simulated devices that generate realistic telemetry. You can use this solution accelerator to test the behavior of the other solution accelerators or to test your own custom IoT solutions.

Each of these accelerators conform to Microsoft's reference architecture (the document we've been looking at throughout this course) so you know at least the basics of your solution starts out the way Microsoft recommends.

**How are Solution Accelerators used?**

Like any template, Solution Accelerators provide the basic framework needed for to model and then provide the foundation for specific aspects of an overall solution. Given the complexity of an IoT solution, the accelerators deploy all the necessary components needed to get essential parts of an overall solution up and running quickly.

This model is used in software development. For example, those familiar with Angular development may have used the Angular CLI (command line interface) command '[ng new](https://github.com/angular/angular-cli/wiki/new)'. This command will install all the files necessary needed to build an Angular application and can scaffold the application creating a basic Angular app. By using ‘ng new’, developers can avoid having to figure out which parts of the framework they need to install individually and get a basic app they can build on for their own solution. IoT Solution Accelerators work in a similar way for IoT development.

For example, the Remote Monitoring solution accelerator will provision the following services in Azure:

1 Azure Active Directory application 1 Virtual Machine 1 IoT Hub 1 Cosmos DB Account 1 Storage account 1 Web Application 1 Azure Maps account 1 Azure Stream Analytics 1 Azure Device Provisioning Service 1 Azure Time Series Insights

These service will provide you with all the basics you will need to to explore a remote monitoring solution. You can use this basic deployment as a basis for your own solution and the accelerator helps you get started by taking the guess work out of what you'll need. Of course you need to first understand how to use all these services and understand how to put together an architecture for your IoT solution. You can learn more about each of the architectural areas the four accelerators cover in other courses in this series.

**Limitations of Solution Accelerators**

As we stated above, the accelerators will provide only the basic framework for parts of an IoT solution. Using them assumes that you know how to put together an overall architecture and build upon them if your goal is to build a solution that you can deploy for a real-world scenario. Note also that the accelerators are limited in the following ways:

1. They are not a single solution but isolated scenarios designed to provide a primer for a specific aspect of what could become a solution. Connecting the individual solutions together to create an overall solution would require development work.
2. The accelerators are not comprehensive. That is, even if you deployed all the available accelerators, you still would not have everything you'd need for a complete IoT solution.
3. As the name of the last accelerator in the list indicates, devices are simulated in these accelerators so you would need to do the work of provisioning physical devices and managing them on your own. The accelerators do, however, give you the framework for working with physical devices and the simulated hardware is a good start for learning how to work with IoT devices.

Despite these limitations, Solution Accelerators are a great way to get started with IoT and can be a good solution to jump start your own development. they can also be used to develop cost models and proof-of-concept projects as you work through your own architectural models.

We encourage you to experiment with the accelerators as you work through the other courses in this series.

**Implementing IoT Devices**

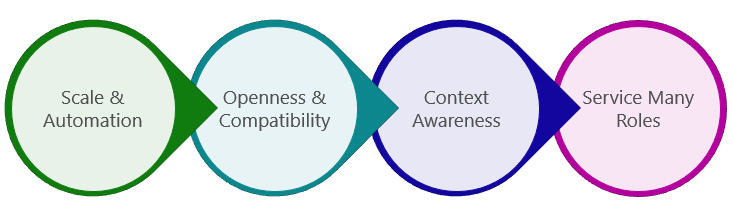
In this topic, you'll learn:

* Basic principles in Azure IoT Device Management
* About the foundation of the life cycle of devices in IoT
* About the limitations of Solution Accelerators

Azure IoT Hub provides the features that, along with an extensibility model, enable device and back-end developers to build robust device management solutions. Regardless of what cloud solution you choose, a solution must accommodate devices that range from constrained sensors and single purpose micro-controllers, to powerful gateways that route communications for groups of devices. In addition, the use cases and requirements for IoT operators vary significantly across industries. In this topic we explore how device management with IoT Hub provides the capabilities, patterns, and code libraries to cater to a diverse set of devices and end users.

**Device Management Principles**

IoT brings with it a unique set of device management challenges and every enterprise-class solution must address the following principles:



**Scale and automation**: IoT solutions require simple tools that enable operators to manage devices remotely and in bulk. Operators should only be alerted when issues arise that require their direct attention.

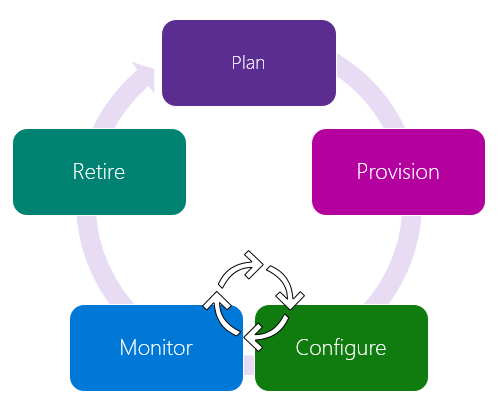
**Openness and compatibility**: Management tools must be tailored to accommodate a multitude of device classes, platforms, and protocols.

**Context awareness**: Device management operations must take into account various states a device can be in to ensure that maintenance downtime doesn't affect critical business operations or create dangerous conditions.

**Service many roles**: Operators must work within the constraints of internal IT department workflows and processes, and they must surface real-time device operations information to supervisors and other business roles.

**Device Life Cycle**

In Azure IoT, there are five device management stages within the device life cycle that are common to all enterprise IoT projects. Within each of these five stages, there are several device operator requirements that should be fulfilled to provide a complete solution:



**Plan**: Enable operators to create a device metadata scheme (a way of categorizing and organizing data across devices) 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. We will look more closely at how device twins can be used for this purpose in an upcoming topic later in this lesson.

**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 their health and security. Azure IoT Hub enables you to 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 real-time 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.

For more information on these ideas, see Microsoft's [device management overview documentation](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-device-management-overview).

**An Implementation Example**

Microsoft's *Transform* blog [describes a specific implementation](https://news.microsoft.com/transform/quicker-fixes-hands-free-when-thyssenkrupp-elevators-service-technicians-use-microsoft-hololens/#sm.00002newnyccectz11ougo72kvcis) of devices and Azure cloud services along with HoloLens technology that has enabled elevator manufacturer thyssenkrupp to monitor the health of elevators they've deployed and use HoloLens to empower engineers to better service those elevators without having to be on site. While the article focuses on the solution thyssenkrupp and Microsoft developed together, consider the implementation details needed to manage the IoT deployment described in the article. Use the material in the sections above to consider how you might use those services to manage the deployment and keep the solution running.

In the solution, Microsoft and thyssenkrupp developed a solution, “that securely connects thyssenkrupp’s thousands of sensors and systems in its elevators to the cloud. With the Microsoft Azure IoT Suite, thyssenkrupp captures elevator data – such as motor temperature, shaft alignment, cab speed and door functioning – and transmits it to a single dashboard.”

The implementation enables the company to monitor the health of the elevators and, using HoloLens allow technicians to remotely troubleshoot and plan for site visits by better understanding the problem before a truck is deployed. This reduces the on-site time needed to make repairs and helped them ensure that the right parts are taken to the site reducing the need for second visits.

While the solution focuses on how IoT helps the company manage their elevators, as an IoT architect, you would need to be responsible for the IoT devices themselves. Each of the sensors deployed to the field needs to be monitored, updated, and configured and your IoT solution would need to account for these activities. This is where Azure IoT hub helps. Any complex deployment would have the same requirements.

By using the services we describe above, device configuration information (for multiple devices) can be stored in the device twin, queried for information and updated using jobs or direct method calls. We'll explore how these services work in an upcoming topic later in this lesson.

**Architecting a Device Management Solution**

You can apply the same thought experiments to the other examples we've explored in previous lessons. As an exercise, while working through Module 4, pick one of the scenarios we discuss and mock up a device management plan for the devices used in that solution. You can ask yourself the following questions:

1. What devices might I need for this solution?
2. How can I use Microsoft's IoT Hub to provision these devices?
3. How can IoT Hub help me push updates or configuration changes to the devices in this solution?
4. How does IoT Hub help me monitor the health of these devices?

If you're not sure how to answer these questions, work through the rest of the topics in this lesson to learn more about how IoT Hub works and the services it provides to help with each of these tasks.

**IoT Cloud Services: The Azure IoT Hub**

In this topic, you'll learn:

* About the benefits of Azure IoT Hub as an IoT cloud service provider
* How Azure IoT Hub can help manage device identity
* About the Azure IoT Hub messaging and communication services
* How Azure IoT Hub keeps your devices connected to the cloud

Azure IoT Hub is a fully managed service that enables secure and reliable bidirectional communications between the solution back end and a wide variety of devices. In fact, a single IoT Hub is capable of connecting millions of IoT devices and ingesting high volumes of [telemetry](https://en.wikipedia.org/wiki/Telemetry).

**The IoT Hub service**

* Provides multiple device-to-cloud and cloud-to-device communication options, including one-way messaging, file transfer, and request-reply methods.
* Provides built-in declarative message routing (one-to-one messaging or being able to target specific endpoints with specific messages – contrast with a [broadcast](https://en.wikipedia.org/wiki/Broadcasting) message) to other Azure services.
* Provides a queryable store for device metadata and synchronized state information. This means you can request information about devices in your IoT network and get information about the state they're in.
* Enables secure communications and access control using per-device security keys or [X.509](https://en.wikipedia.org/wiki/X.509) certificates.
* Provides extensive monitoring for device connectivity and device identity management events.
* Includes device libraries for the most popular languages and platforms.

**Azure IoT Hub Identity Registry**

Every IoT hub has an identity registry that stores information about the devices that are 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.

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

**Messaging**

IoT Hub provides the following messaging primitives to communicate with a device:

* Device-to-cloud from a device to a back-end app.
* Cloud-to-device from a back-end app (service or cloud).

See the [IoT Hub Messaging](https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messaging) resource documentation for a more detailed look at Device-to-cloud and Cloud-to-device messaging.

**Service-Assisted Communication**

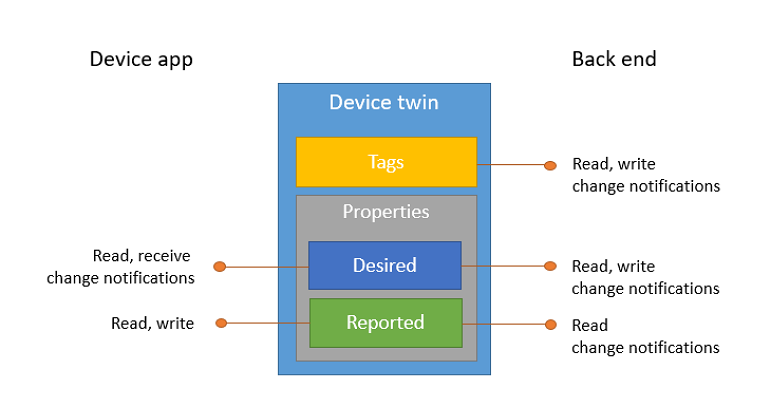
Azure IoT Hub implements the service-assisted communication pattern to mediate the interactions between your devices and your solution back end. The goal of service-assisted communication is to establish trustworthy, [bidirectional](https://en.wikipedia.org/wiki/Duplex_(telecommunications)) communication paths between a control system, such as IoT Hub, and special-purpose devices that are deployed in untrusted physical space. The pattern establishes the following principles:

* Security takes precedence over all other capabilities.
* Devices do not accept network information it has not specifically requested. A device establishes all connections and routes in an outbound-only fashion. For a device to receive a command from the solution back end, the device must regularly initiate a connection to check for any pending commands to process.
* Devices should only connect to or establish routes to well-known services they are paired with, such as IoT Hub.
* The communication path between device and service or between device and gateway is secured at the application protocol layer (that is, the technology that handles the actual communication between the devices and services).
* System-level authorization and authentication are based on per-device identities. They make access credentials and permissions nearly instantly revocable. This means you can prevent or allow access to a device nearly instantaneously.
* Bidirectional communication for devices that connect sporadically due to power or connectivity concerns is facilitated by holding commands and device notifications until a device connects to receive them. IoT Hub maintains device-specific queues for the commands it sends.
* Application data is secured separately for protected transmission through gateways to a particular service.

**Device Connectivity**

In addition to a rich set of device-to-cloud and cloud-to-device communication options, including messaging, file transfers, and request-reply methods, Azure IoT Hub addresses the device-connectivity challenges in the following ways:

* **Device twins**. Using Device twins, you can store, synchronize, and query device metadata and state information. Device twins are JSON documents that store device state information (metadata, configurations, and conditions). IoT Hub creates and maintains a device twin for each device that you connect to IoT Hub.



* **Per-device authentication and secure connectivity**. You can provision each device with its own security key to enable it to connect to IoT Hub. The IoT Hub identity registry stores device identities and keys in a solution. A solution back end can add individual devices to allow or deny lists to enable complete control over device access.
* **Route device-to-cloud messages to Azure services based on declarative rules**. IoT Hub enables you to define message routes based on routing rules to control where your hub sends device-to-cloud messages. Routing rules do not require you to write any code, and can take the place of custom post-ingestion message dispatchers.
* **Monitoring of device connectivity operations**. You can receive detailed operation logs about device identity management operations and device connectivity events. This monitoring capability enables your IoT solution to identify connectivity issues, such as devices that try to connect with wrong credentials, send messages too frequently, or reject all cloud-to-device messages.
* **An extensive set of device libraries**. Azure IoT device SDKs are available and supported for various languages and platforms–C for many Linux distributions, Windows, and real-time operating systems. Azure IoT device SDKs also support managed languages, such as C#, Java, and JavaScript.
* **IoT protocols and extensibility**. If your solution cannot use the device libraries, IoT Hub exposes a public protocol that enables devices to natively use the MQTT v3.1.1, HTTP 1.1, or AMQP 1.0 protocols. You can also extend IoT Hub to provide support for custom protocols by:
  + Creating a field gateway with the Azure IoT Gateway SDK that converts your custom protocol to one of the three protocols understood by IoT Hub.
  + Customizing the Azure IoT protocol gateway, an open source component that runs in the cloud.
* **Scale**. Azure IoT Hub scales to millions of simultaneously connected devices and millions of events per second.

**An Example Implementation**

The Microsoft Azure team has put together a demo project you can explore to better understand how to implement Azure IoT Hub services into your solution. [This demo](https://github.com/azure-cat-emea/servicefabriciothubdemo) shows how to use the Azure Service Fabric along with IoT Hub, Event Hubs, [OWIN](https://en.wikipedia.org/wiki/Open_Web_Interface_for_.NET), and Web API. The application ingest events from the input Event Hub, processes sensor readings and generates an alert whenever a value outside of the tolerance range is received.

Working with this demo will require an Azure subscription and some technical proficiency that may be beyond what you're ready for at this point in your coursework. However, it will be beneficial for you to review the implementation to see what is possible using Azure IoT Hub. As you move into more advanced topics in the rest of the courses in this series, you can revisit this demo to test your skills and further lock in the concepts you're learning. You can also check out a [more advanced IoT demo](https://github.com/azure-cat-emea/servicefabricobserver) that uses the Observer design pattern to show how changes in one part of an implementation can be broadcast and inform changes to other parts of an implementation.

**IoT Hub Messaging**

In this topic, you'll learn:

* How Azure IoT Hub manages messages going deeper on this topic
* How device-to-cloud and cloud-to-device messaging works
* How to manage the messaging life cycle using Azure IoT Hub

As we noted in the previous topic, IoT Hub provides the following messaging primitives to communicate with a device:

* Device-to-cloud from a device to a back-end app.
* Cloud-to-device from a back-end app (service or cloud).

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.

IoT Hub supports multiple device-facing protocols (such as MQTT, AMQP, and HTTP). To support seamless interoperability across protocols, IoT Hub defines a common message format that all device-facing protocols support.

IoT Hub exposes a built-in Event Hub-compatible endpoint to enable back-end apps to read the device-to-cloud messages received by the hub. You can also create custom endpoints in your IoT hub by linking other services in your subscription to the hub.

Use device-to-cloud messages for sending time series telemetry and alerts from your device app, and cloud-to-device messages for one-way notifications to the device app.

**Device-To-Cloud**

You send device-to-cloud messages through a device-facing endpoint (/devices//messages/events). Routing rules then route your messages to one of the service-facing endpoints on your IoT hub. Routing rules use the properties of the device-to-cloud messages flowing through your hub to determine where to route them. By default, messages are routed to the built-in service-facing endpoint (messages/events), that is compatible with Event Hubs. Therefore, you can use standard Event Hubs integration and SDKs to receive device-to-cloud messages.

IoT Hub implements device-to-cloud messaging using a streaming messaging pattern. IoT Hub's device-to-cloud messages are more like Event Hubs events than Service Bus messages in that there is a high volume of events passing through the service that can be read by multiple readers.

This implementation has the following implications:

* Like Event Hubs events, device-to-cloud messages are durable and retained in an IoT hub's default messages/events endpoint for up to seven days.
* Like Event Hubs events, device-to-cloud messages can be at most 256 KB, and can be grouped in batches to optimize sends. Batches can be at most 256 KB.

There are, however, a few important distinctions between IoT Hub device-to-cloud messaging and Event Hubs:

* IoT Hub allows per-device authentication and access control.
* IoT Hub allows you to create up to 10 custom endpoints. Messages are delivered to the endpoints based on routes configured on your IoT hub.
* IoT Hub allows millions of simultaneously connected devices (see Quotas and throttling), while Event Hubs is limited to 5000 AMQP connections per namespace.
* IoT Hub does not allow arbitrary partitioning using a PartitionKey. Device-to-cloud messages are partitioned based on their originating deviceId.
* Scaling IoT Hub is slightly different than scaling Event Hubs.
* Cloud-To-Device
* You send cloud-to-device messages through a service-facing endpoint (/messages/devicebound). A device receives them through a device-specific endpoint (/devices//messages/devicebound).

Each cloud-to-device message is targeted at a single device by setting the to property to /devices//messages/devicebound.

**Message Lifecycle**

To guarantee at least once message delivery, IoT Hub persists cloud-to-device messages in per-device queues. Devices must explicitly acknowledge completion for IoT Hub to remove them from the queue. This guarantees resiliency against connectivity and device failures.

The following diagram shows the lifecycle state graph for a cloud-to-device message.

CloudToDeviceMessaging

When the service sends a message, it is considered Enqueued. When a device wants to receive a message, IoT Hub locks the message (sets the state to Invisible) allowing other threads on the same device to start receiving other messages. When a device thread completes the processing of a message, it notifies IoT Hub by completing the message.

A device can also:

* Reject the message, which causes IoT Hub to set it to the Deadlettered state. Note: devices connecting with MQTT cannot reject cloud-to-device messages.
* Abandon the message, which causes IoT Hub to put the message back in the queue, with the state set to Enqueued.

A thread could fail to process a message without notifying IoT Hub. In this case, messages automatically transition from the Invisible state back to the Enqueued state after a visibility (or lock) timeout. The default value of this timeout is one minute.

A message can transition between the Enqueued and Invisible states for, at most, the number of times specified in the max delivery count property on IoT Hub. After that number of transitions, IoT Hub sets the state of the message to Deadlettered. Similarly, IoT Hub sets the state of a message to Deadlettered after its expiration time (see Time to live).

**IoT Device Configuration**

In this topic, you'll learn:

* How to use Azure IoT Hub to configure devices
* About using Azure IoT Hub to run programs on devices and run batch jobs
* About using device twins to manage device configuration information

We learned in an earlier topic in this lesson about the types of devices and device services that can be used in an IoT solution. We looked at the Azure IoT Hub can be used to *manage* IoT devices and in this topic we'll look more closely at how the IoT Hub can be used to *configure* devices in your solution and ensure that devices are kept up to date.

**Remote Operations**

In many (perhaps most) IoT solutions, devices are deployed “in the field” which means they are placed in areas distinct from those where the engineers and operators who need to manage those devices are located. Because of this, most of the operations that need to be performed on these devices will need to be done remotely. When a solution includes dozens or hundreds of devices, monitoring, updating, and managing the configuration of those devices can present significant operational and logistical challenges. Azure IoT Hub was designed with this scenario in mind and can make remote device operations much more manageable. Remote operations that target IoT devices can be divided into three implementation categories: Direct Methods, Device Twins, and Device Management. Let's look at each of these in turn.

**Direct Methods**

IoT Hub gives you ability to run programs on devices from the cloud. Direct methods represent a request-reply operation with a device similar to an HTTP call in that they succeed or fail immediately (after a user-specified timeout). This is useful for scenarios where the course of immediate action is different depending on whether the device was able to respond, such as sending an SMS wake-up to a device if a device is offline.

Each direct method targets a single device. Jobs provide a way to invoke direct methods on multiple devices, and schedule operations on disconnected devices.

For example, you may want to update the [firmware](https://en.wikipedia.org/wiki/Firmware) on a set of devices deployed to turbines in a wind farm. You can use the Jobs feature to push the update to all devices that are online and schedule the update for devices that currently are offline (perhaps some turbines go into a “sleep mode” for a period of time each day to reduce wear) so that the update is pushed the next time the device comes back online.

Each IoT hub has an identity registry that you can use to create per-device resources in the service, such as a queue that contains cloud-to-device messages. The identity registry also enables you to control access to the device-facing endpoints (like a field gateway).

Jobs manage import and export operations like transferring data collected from an IoT device (or set of devices) or pushing a settings file to the device. These enable you to execute bulk service operations using the IoT hub.

**What are Jobs?**

Identity registry operations use the Job system when the operation:

* Has a potentially long execution time compared to standard run-time operations.
* Returns a large amount of data to the user.

In these cases, instead of a single API call waiting or blocking on the result of the operation, the operation asynchronously creates a Job for that IoT hub. The operation then immediately returns a JobProperties object. So instead of your program waiting for the result of the operation (for example, requesting a large amount of data from the IoT device), you can use the JobProperties object as a pointer to the request and run other operations at the same time. It's like dropping your clothing at the dry cleaners and, instead of waiting for them to finish the job, you get a ticket which represents the job so you can go home and do other things. You can then call them or visit the shop again to see if the work is done.

**Device Twins**

Device twins are [JSON documents](https://en.wikipedia.org/wiki/JSON) that store device state information (metadata, configurations, and conditions). IoT Hub persists 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 created and deleted when a new device identity is created or deleted in IoT Hub.

Use device twins to:

* Store device-specific [metadata](https://en.wikipedia.org/wiki/Metadata) in the cloud. For example, the deployment location of a vending machine is metadata about the physical vending machine.
* Report current state information such as available capabilities and conditions from your device app. For example, whether a device is connected to your IoT hub over cellular or WiFi.
* Synchronize the state of long-running workflows (operations or running programs) 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.

**Device Management Patterns**

IoT Hub enables the device management patterns described below. If necessary, you can extend these patterns to fit your exact scenario, or you can design new patterns based on these core templates.

**Reboot** - The back-end app informs the device through a direct method that it has initiated a reboot. The device uses the reported properties to update the reboot status of the device.

<https://docs.microsoft.com/en-us/azure/iot-hub/media/iot-hub-device-management-overview/reboot-pattern.png>

**Factory Reset** - The back-end app informs the device through a direct method that it has initiated a factory reset. The device uses the reported properties to update the factory reset status of the device.

<https://docs.microsoft.com/en-us/azure/iot-hub/media/iot-hub-device-management-overview/facreset-pattern.png>

**Configuration** - The back-end app uses the desired properties to configure software running on the device. The device uses the reported properties to update configuration status of the device.

<https://docs.microsoft.com/en-us/azure/iot-hub/media/iot-hub-device-management-overview/configuration-pattern.png>

**Firmware Update** - The back-end app informs the device through a direct method that it has initiated a firmware update. The device initiates a multistep process to download the firmware image, apply the firmware image, and finally reconnect to the IoT Hub service. Throughout the multistep process, the device uses the reported properties to update the progress and status of the device.

<https://docs.microsoft.com/en-us/azure/iot-hub/media/iot-hub-device-management-overview/fwupdate-pattern.png>

**Reporting progress and status** - The solution back end runs device twin queries, across a set of devices, to report on the status and progress of actions running on the devices.

<https://docs.microsoft.com/en-us/azure/iot-hub/media/iot-hub-device-management-overview/report-progress-pattern.png>

**Introduction to IoT Data Analytics and Storage**

In this topic, you'll learn:

* About IoT data storage options
* The basics of IoT data analytics in Azure IoT Hub

**Introduction to Data Storage**

There are several cloud storage options to consider when planning a storage solution for your IoT data.

**Azure Cosmos DB**

Azure Cosmos DB is a multi-model (e.g. document, relational, key-value pair) storage option that includes a fully managed NoSQL database service that provides rich and familiar [SQL query](https://en.wikipedia.org/wiki/SQL) capabilities with consistent low latencies on JSON data. Cosmos DB is a great fit for IoT solutions and many other types of applications that need seamless scale and global replication.

**SQL Database**

SQL Database is a [relational database](https://en.wikipedia.org/wiki/Relational_database) service in the Microsoft cloud based on the Microsoft SQL Server engine and capable of handling mission-critical workloads. SQL Database delivers predictable performance at multiple service levels, dynamic scalability with no downtime, built-in business continuity, and data protection — all with near-zero administration. These capabilities allow you to focus on rapid app development and accelerating your time to market, rather than allocating precious time and resources to managing virtual machines and infrastructure. Because SQL Database is based on the SQL Server engine, SQL Database supports existing SQL Server tools, libraries, and APIs.

For more information see [SQL Database Documentation](https://docs.microsoft.com/en-us/azure/sql-database/)

**Azure Storage**

Azure storage provides the following services that can be used in your IoT solutions: Blob storage, Table storage, Queue storage.

**Blob Storage** stores unstructured object data. A blob can be any type of text or binary data, such as a document, media file, or application installer. Blob storage is also referred to as Object storage.

**Table Storage** stores structured datasets. Table storage is a [NoSQL](https://en.wikipedia.org/wiki/NoSQL) key-attribute data store, which allows for rapid development and fast access to large quantities of data.

**Queue Storage** provides reliable messaging for workflow processing and for communication between components of cloud services.

For more information see [Azure Storage Documentation](https://docs.microsoft.com/en-us/azure/storage/)

**Azure Data Lake Store**

Azure Data Lake Store is an enterprise-wide [hyper-scale](https://en.wikipedia.org/wiki/Hyperscale) repository for big data analytic workloads. Azure Data Lake enables you to capture data of any size, type, and ingestion speed in one single place for operational and exploratory analytics.

Azure Data Lake Store provides unlimited storage and is suitable for storing a variety of data for analytics. It does not impose any limits on account sizes, file sizes, or the amount of data that can be stored in a data lake. Individual files can range from kilobyte to petabytes in size making it a great choice to store any type of data. Data is stored durably by making multiple copies and there is no limit on the duration of time for which the data can be stored in the data lake.

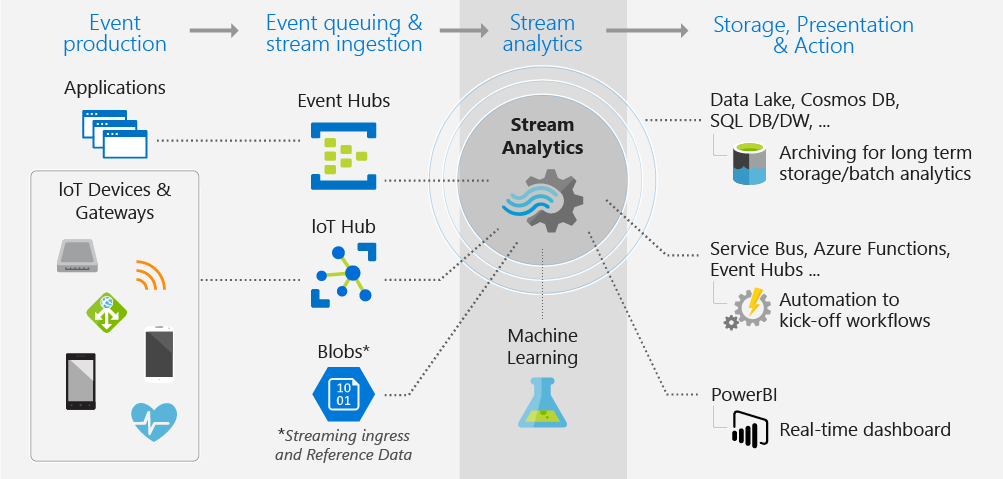
For more information see [Data Lake Store Documentation](https://docs.microsoft.com/en-us/azure/data-lake-store/)

**Data Analytics and IoT**

Being able to run analytics on data in real time and generate alerts is a key component of most IoT solutions.

**Stream Analytics Job**

Azure Stream Analytics is a fully managed, real-time event processing engine that helps you to unlock deep insights from your data. Stream Analytics enables you to set up real-time analytic computations on data streaming from devices, sensors, applications, and more.



The Azure portal enables you to create a Stream Analytics job using the same methods that you would use to add any other service. Once the service is deployed to your resource group, you are presented with a blade that can be used to specify the input source of the streaming data, the output sink for the results of your job, and a SQL-like query expression that can be modified to transform your data. You can monitor and adjust the scale/speed of your job in the Azure portal to scale from a few kilobytes to a gigabyte or more of events processed per second. Your Stream Analytics jobs are backed by highly tuned streaming engines for time-sensitive processing.

Scenarios of real-time streaming analytics can be found across all industries: personalized, real-time stock-trading analysis and alerts offered by financial services companies; real-time fraud detection; data and identity protection services; reliable ingestion and analysis of data generated by sensors and actuators embedded in physical objects (Internet of Things, or IoT); web clickstream analytics; and customer relationship management (CRM) applications issuing alerts when customer experience within a time frame is degraded.

**Configuring Inputs**

The data connection to Stream Analytics is a data stream of events from a data source. This is called an “input.” Stream Analytics has first-class integration with Azure data stream sources Event Hub, IoT Hub, and Blob storage that can be from the same or different Azure subscription as your analytics job.

As data is pushed to a data source, it is consumed by the Stream Analytics job and processed in real time. Inputs are divided into two distinct types: data stream inputs and reference data inputs.

* **Data stream inputs**: A data stream is unbounded sequence of events coming over time. Stream Analytics jobs must include at least one data stream input to be consumed and transformed by the job. Blob storage, Event Hubs, and IoT Hubs are supported as data stream input sources. Event Hubs are used to collect event streams from multiple devices and services, such as social media activity feeds, stock trade information or data from sensors. IoT Hubs are optimized to collect data from connected devices in Internet of Things (IoT) scenarios. Blob storage can be used as an input source for ingesting bulk data as a stream.
* **Reference data**: Stream Analytics supports a second type of input known as reference data. This is auxiliary data which is either static or slowly changing over time and is typically used for performing correlation and look-ups. Azure Blob storage is currently the only supported input source for reference data. Reference data source blobs are limited to 100MB in size.

For more information on configuring inputs, see https://go.microsoft.com/fwlink/?linkid=848177

**Configuring Outputs**

When authoring a Stream Analytics job, consider how the resulting data will be consumed. How will you view the results of the Stream Analytics job and where will you store it?

In order to enable a variety of application patterns, Azure Stream Analytics has different options for storing output and viewing analysis results. This makes it easy to view job output and gives you flexibility in the consumption and storage of the job output for data warehousing and other purposes. Any output configured in the job must exist before the job is started and events start flowing. For example, if you use Blob storage as an output, the job will not create a storage account automatically. It needs to be created by the user before the ASA job is started.

We looked at various storage options above so refer to that content for your options.

**Configuring Queries**

Queries in Azure Stream Analytics are expressed in a SQL-like query language, which is documented in the [Stream Analytics Query Language Reference guide](https://msdn.microsoft.com/library/azure/dn834998.aspx). Using the Stream Analytics query language in the in-browser query editor, you get intellisense auto-complete to help you can quickly and easily implement time series queries, including temporal-based joins, windowed aggregates, temporal filters, and other common operations such as joins, aggregates, projections, and filters. In addition, in-browser query testing against a sample data file enables quick, iterative development.

For an explanation of how to implement Query patterns that support the real-world scenarios listed below, review [Query examples for common Stream Analytics](https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-stream-analytics-query-patterns) usage patterns

**Conclusion**

In this topic, we provided an overview of the many data storage options available for your IoT solution. Each has a specific purpose and you may use one or many of these options depending on the needs of your architecture. We also covered ways in which you can process data coming from your devices. Well talk more about how to present the data you've collected and stored in the next topic.

You can learn more about how to work with data in a course devoted to this topic in this series. So while this gives you a taste of what is available, you'll need to explore the topic more deeply in the other course in order to learn how to use these tools and services in your implementation.

**IoT Business Intelligence using PowerBI**

In this topic, you'll learn:

* What PowerBI is and how it can be used in business intelligence scenarios
* The basics of using PowerBI to create reports

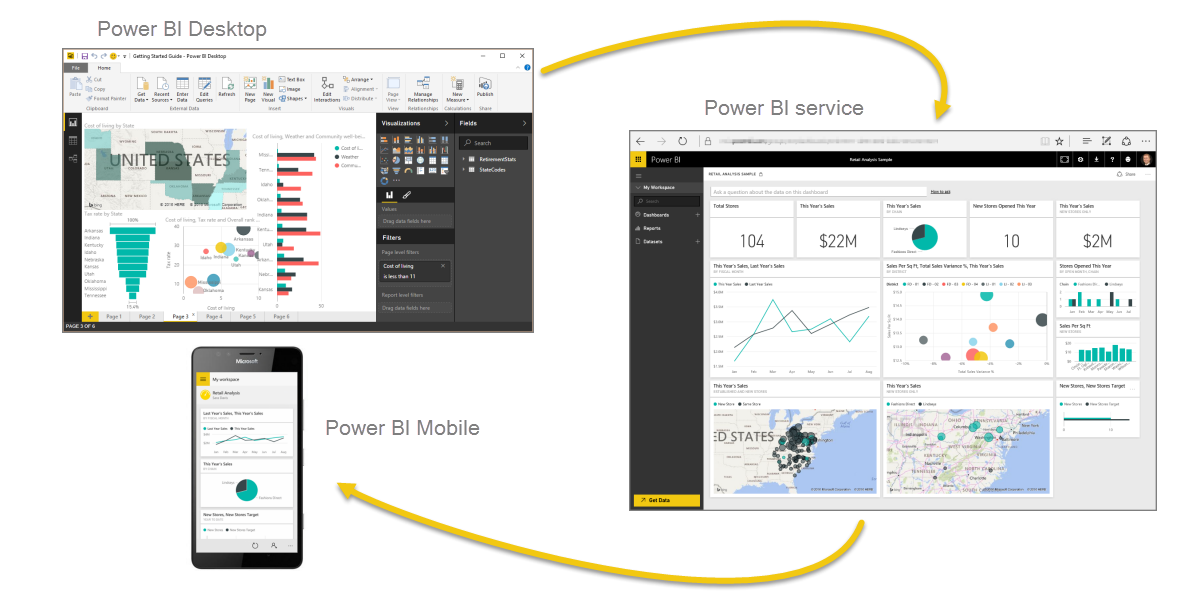
Power BI[^1] is a collection of software services, apps, and connectors that work together to turn your unrelated sources of data into coherent, visually immersive, and interactive insights. Whether your data is a simple Excel spreadsheet, or a collection of cloud-based and on-premises hybrid data warehouses, Power BI lets you connect to your data sources, visualize (or discover) what’s important, and share that with anyone or everyone you want.

[^1]: While this topic covers PowerBI as a tool for presenting your data in a meaningful way, be aware that Azure data services can be used with other data analytics tools like [Tableau](https://www.tableau.com/solutions/azure).

Power BI can be simple and fast – capable of creating quick insights from an Excel spreadsheet or a local database. But Power BI is also robust and enterprise-grade, ready for extensive modeling and real-time analytics, as well as custom development. So it can be your personal report and visualization tool, and can also serve as the analytics and decision engine behind group projects, divisions, or entire corporations.

**The Parts of Power BI**

Power BI consists of a Windows desktop application called Power BI Desktop, an online SaaS (Software as a Service) service called the Power BI service, and mobile Power BI apps available on Windows phones and tablets, as well as for iOS and Android devices.



PowerBI - Apps and Services

These three elements – the Desktop, the service, and Mobile – are designed to let people create, share, and consume business insights in the way that serves them, or their role, most effectively.

**Power BI Workflow**

The common flow of activity in Power BI is the following:

* Bring data into Power BI Desktop, and create a report.
* Publish to the Power BI service, where you create new visualizations or build dashboards.
* Share your dashboards with others, especially people who are on the go.

**Creating Reports with Power BI Desktop**

With Power BI Desktop, you connect to data (usually multiple data sources), shape that data (with queries that build insightful, compelling data models), and use that model to create reports (which others can leverage, build upon, and share).

When the steps are completed to your satisfaction – connect, shape, and report – you can save that work in Power BI Desktop file format, which is the .pbix extension. Power BI Desktop files can be shared like any other file, but the most compelling way to share Power BI Desktop files is to upload them (share them) on the Power BI service.

Power BI Desktop centralizes, simplifies, and streamlines what can otherwise be a scattered, disconnected, and arduous process of designing and creating business intelligence repositories and reports.