**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**

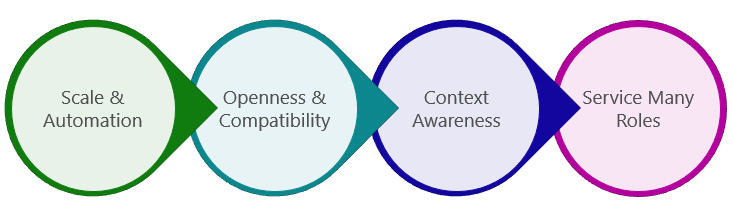
In this topic, you'll learn:

* 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.

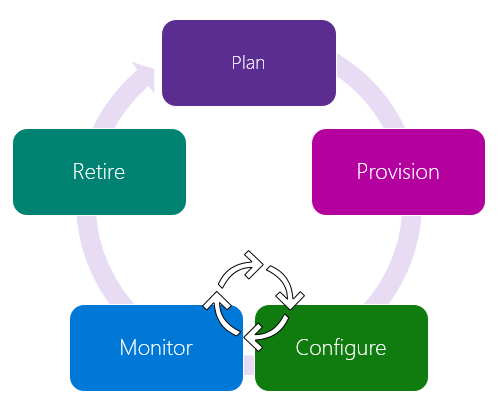
**Openness and compatibility**: Management tools must be tailored to accommodate a multitude of device classes, platforms, and protocols.

**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**

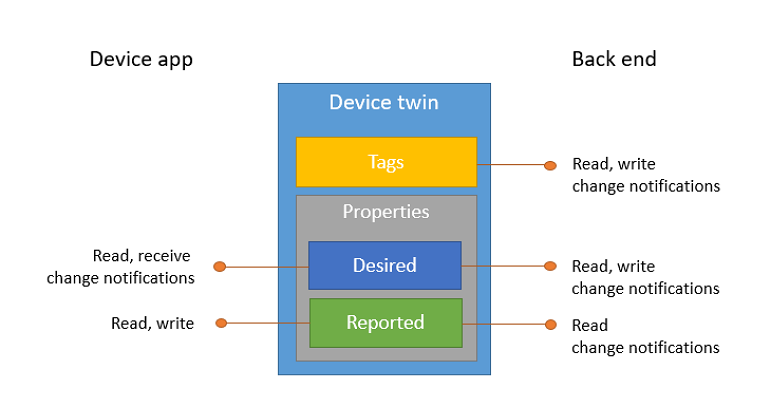
Azure IoT Hub implements the service-assisted communication pattern to mediate the interactions between your devices and your solution back end. The goal of service-assisted communication is to establish trustworthy, [bidirectional](https://en.wikipedia.org/wiki/Duplex_(telecommunications)) communication 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.