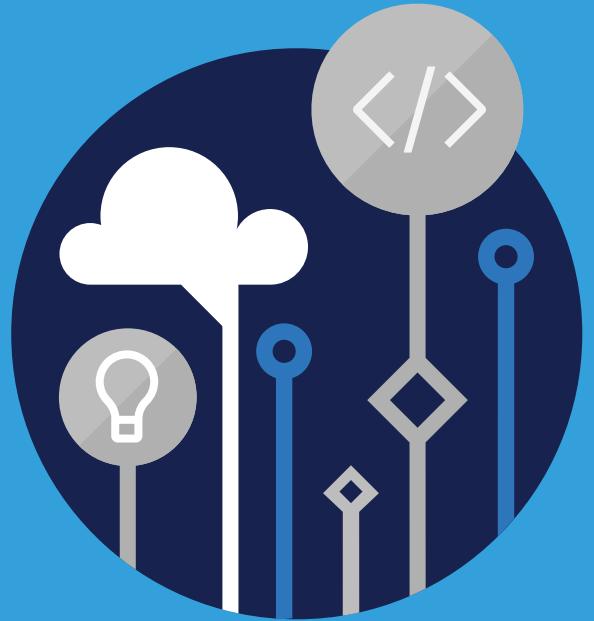


Microsoft  
Official  
Course



**AZ-220T00**

Microsoft Azure IoT  
Developer

# AZ-220T00

## Microsoft Azure IoT Developer

## II Disclaimer

---

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks><sup>1</sup> are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

---

<sup>1</sup> <http://www.microsoft.com/trademarks>

## MICROSOFT LICENSE TERMS

### MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS.  
IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

**If you comply with these license terms, you have the rights below for each license you acquire.**

#### 1. DEFINITIONS.

1. "Authorized Learning Center" means a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
2. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
3. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facility that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
4. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of an MPN Member, or (iii) a Microsoft full-time employee.
5. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
6. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
7. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals and developers on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics or Microsoft Business Group courseware.
8. "Microsoft IT Academy Program Member" means an active member of the Microsoft IT Academy Program.
9. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
10. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals and developers on Microsoft technologies.
11. "MPN Member" means an active Microsoft Partner Network program member in good standing.
12. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
13. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led

Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.

14. "Trainer" means (i) an academically accredited educator engaged by a Microsoft IT Academy Program Member to teach an Authorized Training Session, and/or (ii) a MCT.
  15. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.
2. **USE RIGHTS.** The Licensed Content is licensed not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.
- 2.1 Below are five separate sets of use rights. Only one set of rights apply to you.
    1. **If you are a Microsoft IT Academy Program Member:**
      1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
      2. For each license you acquire on behalf of an End User or Trainer, you may either:

distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**

provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**

provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content, **provided you comply with the following:**

        3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
        4. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
        5. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
        6. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**

provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**

provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content, **provided you comply with the following:**

3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
4. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
5. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
6. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,

7. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
8. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
9. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

**2. If you are a Microsoft Learning Competency Member:**

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or MCT, you may either:

distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**

provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**

you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content, **provided you comply with the following:**

3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
4. you will ensure that each End User attending a Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
5. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
6. you will ensure that each Trainer teaching a Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,
7. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
8. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
9. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and

10. you will only provide access to the Trainer Content to Trainers.

**3. If you are a MPN Member:**

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or Trainer, you may either:

distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**

provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**

you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content,

3. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
4. you will ensure that each End User attending a Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
5. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
6. you will ensure that each Trainer teaching a Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,
7. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
8. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
9. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
10. you will only provide access to the Trainer Content to Trainers.

**4. If you are an End User:**

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy

of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

##### 5. If you are a Trainer.

1. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.
2. You may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.
  - **2.2 Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.
  - **2.3 Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.
  - **2.4 Third Party Notices.** The Licensed Content may include third party code that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.
  - **2.5 Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.
3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("Pre-release"), then in addition to the other provisions in this agreement, these terms also apply:
  1. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
  2. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft

to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.

3. **Pre-release Term.** If you are a Microsoft IT Academy Program Member, Microsoft Learning Competency Member, MPN Member or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("**Pre-release term**"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.
4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
  - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
  - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
  - modify or create a derivative work of any Licensed Content,
  - publicly display, or make the Licensed Content available for others to access or use,
  - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
  - work around any technical limitations in the Licensed Content, or
  - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.
6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see [www.microsoft.com/exporting](http://www.microsoft.com/exporting).
7. **SUPPORT SERVICES.** Because the Licensed Content is "as is", we may not provide support services for it.
8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or

updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.

**10. ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.

**11. APPLICABLE LAW.**

1. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
2. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.

**12. LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.

**13. DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE. "YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**

**14. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

**Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.**

**Remarque: Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.**

**EXONÉRATION DE GARANTIE.** Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection des consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les

garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contre-façon sont exclues.

**LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES.**

Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

**EFFET JURIDIQUE.** Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised November 2014



# Contents

■	<b>Module 0 Welcome</b>	1
	Start here	1
■	<b>Module 1 Introduction to IoT and Azure IoT Services</b>	7
	Business Opportunities for IoT	7
	Introduction to IoT Solution Architecture	24
	IoT Hardware and Cloud Services	28
	Lab Scenario	42
	Course Labs	45
■	<b>Module 2 Devices and Device Communication</b>	47
	IoT Hub Concepts	47
	IoT Device Lifecycle Concepts	53
	IoT Developer Tools	59
	Device Configuration and Communication	73
	About the Module 2 Labs	81
■	<b>Module 3 Device Provisioning at Scale</b>	83
	Device Provisioning Service Terms and Concepts	83
	Configure and Manage the Device Provisioning Service	102
	Device Provisioning Tasks	111
	About the Module 3 Labs	119
■	<b>Module 4 Message Processing and Analytics</b>	121
	Messages and Message Processing	121
	Additional Considerations for IoT Hub Messaging	139
	Data Storage and the Lambda Architecture	145
	Azure Functions and Stream Analytics	154
	About the Module 4 Labs	211
■	<b>Module 5 Insights and Business Integration</b>	213
	Business Integration for IoT Solutions	213
	Data Visualization with Time Series Insights	235
	Data Visualization with Power BI	247
	About the Module 5 Labs	252
■	<b>Module 6 Azure IoT Edge Deployment Process</b>	253
	Introduction to Azure IoT Edge	253



Edge Deployment Process .....	275
Edge Gateway Devices .....	289
About the Module 6 Labs .....	302
<b>Module 7 Azure IoT Edge Modules and Containers</b> .....	303
Develop Custom Edge Modules .....	303
Offline and Local Storage .....	319
IoT Edge Storage .....	323
About the Module 7 Labs .....	332
<b>Module 8 Device Management</b> .....	333
Introduction to IoT Device Management .....	333
Manage IoT and IoT Edge Devices .....	347
Device Management at Scale .....	352
About the Module 8 Labs .....	365
<b>Module 9 Solution Testing, Diagnostics, and Logging</b> .....	367
Monitoring and Logging .....	367
Troubleshooting .....	404
About the Module 9 Labs .....	408
<b>Module 10 Azure Security Center and IoT Security Considerations</b> .....	409
Security Fundamentals for IoT Solutions .....	409
Introduction to Azure Security Center for IoT .....	425
Enhance Protection with Azure Security Center for IoT Agents .....	455
About the Module 10 Labs .....	463
<b>Module 11 Build an IoT Solution with Azure IoT Central</b> .....	465
Introduction to IoT Central .....	465
Create and Manage Device Templates .....	478
Manage Devices in Azure IoT Central .....	487
Business Integration .....	494
Data Visualizations and Analysis .....	498
About the Module 11 Labs .....	504

# Module 0 Welcome

## Start here

## About this course

### Course Description

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

**Level:** Intermediate

### Audience

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

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

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

### Prerequisites

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

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

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

### **Expected learning**

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

## **Course syllabus**

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

### M1: Introduction to IoT and Azure IoT Services

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

Lessons:

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

Labs:

- Getting Started with Azure
- Setting Started with Azure IoT Services

### M2: Devices and Device Communication

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

**Lessons:**

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

**Labs:**

- Setup the Development Environment
- Connect IoT Device to Azure

**M3: Device Provisioning at Scale**

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

**Lessons:**

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

**Labs:**

- Individual Enrollment of Devices in DPS
- Automatic Enrollment of Devices in DPS

**M4: Message Processing and Analytics**

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

**Lessons:**

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

**Labs:**

- Device Message Routing
- Filtering and Aggregating Message Data

**M5: Insights and Business Integration**

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

**Lessons:**

- Business Integration for IoT Solutions

- Data Visualization with Time Series Insights
- Data Visualization with Power BI

Labs:

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

#### M6: Azure IoT Edge Deployment Process

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

Lessons:

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

Labs:

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

#### M7: Azure IoT Edge Modules and Containers

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

Lessons:

- Develop Custom Edge Modules
- Offline and Local Storage

Labs:

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

#### M8: Device Management

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

Lessons:

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

Labs:

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

#### M9: Solution Testing, Diagnostics, and Logging

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

Lessons:

- Monitoring and Logging
- Troubleshooting

Labs:

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

#### M10: Azure Security Center and IoT Security Considerations

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

Lessons:

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

Labs:

- Implementing Azure Security Center for IoT

#### M11: Build an IoT Solution with IoT Central

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

Lessons:

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

Labs:

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

## AZ-220 certification exam

The AZ-220, **Microsoft Azure IoT Developer<sup>1</sup>**, certification exam is geared towards the person who is responsible for the implementation and the coding required to create and maintain the cloud and edge portion of an IoT solution.

---

<sup>1</sup> <https://www.microsoft.com/en-us/learning/exam-AZ-220.aspx>

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

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

- ✓ Learn more about the **certification changes<sup>2</sup>** that took effect on May 1, 2019.

## Resources

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

- **Azure IoT Fundamentals<sup>3</sup>**. The Azure IoT documentation page provides links to lots of resources.
- **Azure IoT Products<sup>4</sup>**. The Azure IoT Products page provides links to product pages.
- **Azure IoT Reference Architecture<sup>5</sup>**. This reference architecture shows a recommended architecture for IoT applications on Azure using PaaS (platform-as-a-service) components.
- **Building IoT solutions with Azure: a developer's guide<sup>6</sup>**. This guide provides an overview of Azure services that address key IoT solution requirements, as well as a step-by-step progression you can use to build proficiency and move toward a fully functioning solution quickly and easily.

## Obtaining and Redeeming Your Azure Pass

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

<sup>2</sup> <https://www.microsoft.com/en-us/learning/community-blog-post.aspx?BlogId=8&Id=375217>

<sup>3</sup> <https://docs.microsoft.com/en-us/azure/iot-fundamentals/>

<sup>4</sup> <https://azure.microsoft.com/en-us/product-categories/iot/>

<sup>5</sup> <https://docs.microsoft.com/en-us/azure/architecture/reference-architectures/iot/>

<sup>6</sup> <https://discover.microsoft.com/azure-iot-building-solutions-dev-guide/>

# Module 1 Introduction to IoT and Azure IoT Services

## Business Opportunities for IoT

### What is the Internet of Things?

Although IoT has matured as a technology in recent years, a range of definitions still proliferate. The most commonly accepted of these definitions is similar to the following:

*The Internet of Things is the network of physical devices that combine IP connectivity with software, sensors, actuators, and other electronics to directly integrate the physical world into our computer-based systems, resulting in efficiency improvements and economic benefits*

Some definitions take this a bit further by adding that IoT will make our lives easier, more efficient, and more cost effective.

Although these types of definitions do provide us with some specifics, they often try to encompass all possibilities within a single sentence, which can make them somewhat convoluted and confusing. Here is a simpler version of the definition above:

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

This definition gets more to the essence of what IoT is, but probably doesn't give us enough nuance to really understand the space. In fact, coming up with a definition is pretty challenging even though the core concepts are fairly straightforward. The international organization Institute of Electrical and Electronics Engineers (IEEE) has been developing a **white paper**<sup>1</sup> just on a definition of IoT. Their paper is 75 pages long (86 with glossary and notes) so far (it's not final) and the definition alone at the end of the paper is roughly four pages long (pages 70-73). If you want a comprehensive overview of the technology, this is the place to start.

Thankfully, we don't have to go as deep as the IEEE to get a good overview of the technology. Put simply, IoT involves two essential components:

1. A device-side (made up of individual devices) that acts as a data source

<sup>1</sup> [https://iot.ieee.org/images/files/pdf/IEEE\\_IoT\\_Towards\\_Definition\\_Internet\\_of\\_Things\\_Revision1\\_27MAY15.pdf](https://iot.ieee.org/images/files/pdf/IEEE_IoT_Towards_Definition_Internet_of_Things_Revision1_27MAY15.pdf)

2. A cloud-side that gathers data and provides resources for analyzing it

Of course, once we dig in we will find that both the device-side and the cloud-side involve complex implementations that provide hundreds of required features, and even the communication between the device and cloud requires secure communication protocols. But at least we have something simple that can get us started.

## Simple IoT Example

Perhaps the best way to understand an IoT solution is to consider an example. The specifics of this example are fictional but it should illustrate how the technology could be used.

### Scenario

Suppose a small town is trying to figure out how to price water during the summer months. They want the town to look nice and enable people to keep their lawns green but they also want to discourage people from wasting water which is in shorter supply during the summer. The town's officials need data to determine how often people actually *need* to water their lawns to keep them green and will use this data to help inform what price they should put on water usage during the summer.

### The IoT Solution

In order to collect the data they need, the town's officials select 100 homes at random across the town and ask them to install a small water sensor in their lawns. The sensor will detect the amount of moisture in the soil and send that data over the home's Wi-Fi connection to a central cloud service that will collect and store the data.

### The Devices

The devices have the following requirements:

- They must be small and unobtrusive.
- They must be able to connect to Wi-Fi and be monitored by the home owner.
- They must be battery operated and be able to run for 6 months without needing a new battery.
- They must be able to detect moisture in the soil in percent saturation.
- They must be able to store data for a 72-hour period in case the connection with the Wi-Fi router is lost.
- They must be able to provide a rudimentary failure signal.

There could be a lot of other device requirements but these are the basics the device must contain. City engineers visit each house that has agreed to install the sensor, place the device in the yard at an optimal location, connect the device to the home's Wi-Fi router, and test the connection with the cloud service.

### The Data

The devices are programmed to collect moisture data every hour for a 24 hour period and average the readings to form a single number that is sent to the cloud service for storage. The dataset includes the device ID, the GPS location, a time/date stamp, moisture level, and other relevant metadata.

## The Cloud Service

City engineers implement a cloud-based solution to listen for incoming data from each device and collect that data in a database. The cloud service also listens for failure signals and can alert city engineers of an actual or pending device failure. The cloud services include an IoT gateway that handles communication with the devices, a storage solution to store the data, stream analytics service to manage the data coming in from the devices and an analytic service to analyze the data and inform decision making.

## Analytics and Assessment

In our scenario, the town officials will use this data to better understand how often people actually are watering their lawns and how often they need to, and to make recommendations to both the homeowners and policy makers about water usage and costs. Since weather is variable, the city engineers may need to collect data for many months to get accurate and actionable data. But the first step is to collect the data in the first place and that's the power of IoT.

## Business Case

In some ways, the core value proposition of IoT has been around for centuries. Even before the advent of the Internet, people have been, out of necessity, using data to make decisions. A shop owner keeps an eye on inventory to determine what people tend to buy and ensures her shop is well stocked with those items. Engineers examine bridges or dams for early signs of failure and use that information to anticipate upcoming repairs and budgets for those repairs. A commuter watches the morning news for information about which route to work might be the quickest.

The promise of IoT is to:

- instrument the things around us (everything from cars to bridge parts to jewelry)
- collect, store, and analyze the data associated with these things
- generate real-time intelligence pertaining to items that we care about

Microsoft characterizes this "promise" in terms of things, insights, and actions:

IoT applications can be described as Things (or devices), sending data or events that are used to generate Insights, which are used to generate Actions to help improve a business or process. An example is an engine (a thing), sending pressure and temperature data that is used to evaluate whether the engine is performing as expected (an insight), which is used to proactively prioritize the maintenance schedule for the engine (an action).



In the sections below we examine the business and social impact that IoT can have and we look at the ways that IoT technology is being implemented.

## The Current and Future Impact of IoT

The IEEE has compiled data and makes the following claims [1] about its current and future impact:

- In 2015, the global wearables market had already increased 223% from the previous year (and data on IDC shows it grew by another 31.4% in Q4 of 2018 alone) [2]
- By 2020, 250 million vehicles will be connected to the Internet
- IoT will add 15 trillion dollars to the global economy over the next 20 years
- There will be 50 billion Internet-connected devices by the year 2020.

As the market becomes more mature, these figures will no doubt change but the current trend looks as if IoT is not only here to stay but positioned to make a substantial impact both socially, in business, and economically.

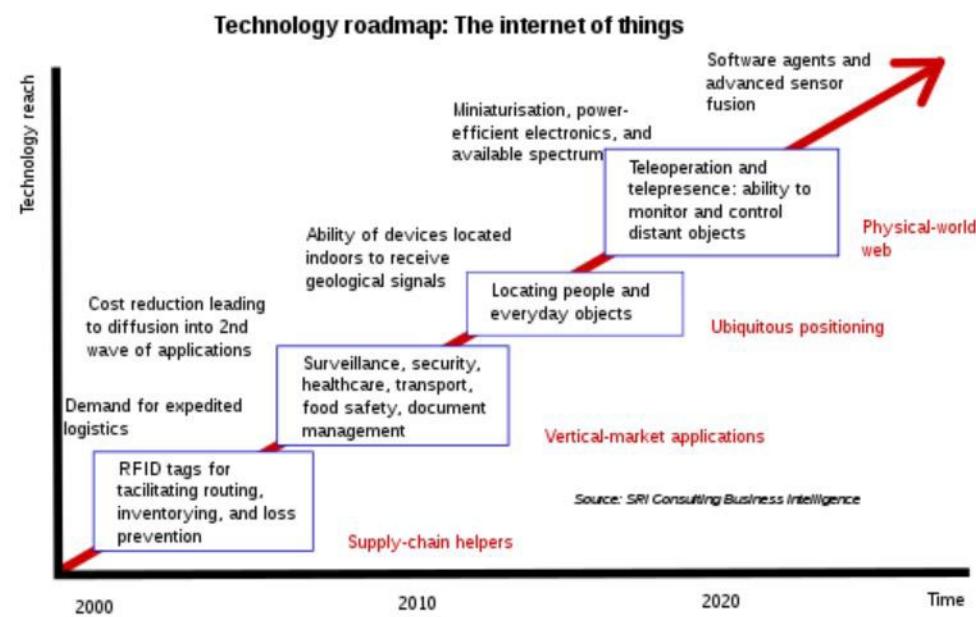
[1]: Source: <https://web.archive.org/web/20171106062938/http://www.comsoc.org/blog/infographic-internet-things-iot>

[2]: Source: <https://www.idc.com/getdoc.jsp?containerId=prUS44901819>

**Note:** The above links identify the original sources - given the temporary nature of much of the content published to the web, these resources may no longer be available or may have moved.

## Where the Technology is Headed

As with any new technology, predicting where it will go is difficult since advances in production, new innovations, and unexpected disrupter all have an impact. Given current trends it looks like Internet-connected devices will be as common place as the integrated circuit in years to come. In 2008, SRI Consulting Business Intelligence and the United States National Intelligence Council created the diagram below as a roadmap of where IoT would be headed. While the pacing of the advances may be inaccurate, the diagram does illustrate a fairly accurate picture of how IoT is advancing.



Of course, the success of IoT will not rely solely on advances in device technology or even cloud services (though advances in both are essential). Omar Valdez-de-Leon, in a 2017 article [3] for the IEEE focuses on 6 areas that will require investment as IoT technology advances:

1. **Platforms.** Platform development will enable developing solutions easier and enable both market verticals and a marketplace.
2. **APIs** (application programming interfaces). Valdez-de-Leon claims that, "APIs are the basic building blocks of an IoT ecosystem" and argues that building robust and market-friendly APIs will help create or ultimately destroy the market effects that will be essential for IoT adoption.
3. **Communities** Strong partner communities will enable IoT systems to inter-operate and that will create a true ecosystem around the technology. Proprietary and isolated systems will stifle growth.
4. **Branded Services** True to the open source model, some companies can demonstrate a commitment to the technology and find ways to make money by branding products (sometimes called "white labeling") using partner technology.
5. **Revenue Models** Generating revenue for IoT products and services will involve thinking differently about how to work with partners and pricing.
6. **Support** Often overlooked, ongoing maintenance of partner relationships, updating aging models, and ensuring partners support their products and services will be key to ensuring IoT moves beyond an interesting idea to becoming a part of the technological ecosystem.

[3]: Source: <https://iot.ieee.org/newsletter/may-2017/key-elements-and-enablers-for-developing-an-iot-ecosystem.html>

## Business Opportunities

When you hear IoT described, you might be inclined to think about the products and services you interact with on a daily basis—what business people call “consumer products.” The Nest thermostat might come to mind because it is a product that is regularly held up as a paradigm case of an Internet-connected device. The hardware uses data to make decisions about how you want your home heated or cooled, gives you reports on how you’re using energy, and interacts with other services to report on the weather outside your home.

But IoT is being adopted by businesses of all sizes to enable a host of productivity gains, make products safer and less prone to error, and build more efficient machines. Microsoft has highlighted specific use cases to illustrate how the technology is being adopted to enable companies to run their businesses. The German company **Bosch describes**<sup>2</sup> IoT in terms of outcome—what connected devices can produce.

Here is a sample of how IoT can benefit businesses and create new opportunities for companies adopting the technology.

## IoT for manufacturing

Microsoft **highlights**<sup>3</sup> the following benefits for implementing IoT solutions in a manufacturing scenario:

- **Improved visibility across your manufacturing operations**—make more informed decisions with a real-time picture of operational status
- **Improved utilization**—maximize asset performance and uptime with the visibility required for central monitoring and management

<sup>2</sup> <https://www.bosch.com/explore-and-experience/internet-of-things/>

<sup>3</sup> <https://www.microsoft.com/en-us/internet-of-things/manufacturing>

- **Reduced waste**—take faster action to reduce or prevent certain forms of waste, thanks to insight on key production metrics
- **Targeted cost savings**—benchmark resource usage and identify inefficiencies to support operational improvements
- **Improved quality**—detect and prevent quality problems by finding and addressing equipment issues sooner

Each of these items can enable a manufacturer to improve efficiency and reduce waste, increase safety (for both workers and customers), help drive profitability by using real-time data to make decisions and, with automation, make real-time adjustments to processes.

## IoT-enabled Smart City

Telecom company AT&T **describes<sup>4</sup>** the following benefits of a city fitted with smart devices that provide real-time data on things like parking, traffic, violent activities, and air quality:

- Enhanced citizen safety
- Optimized public services
- Economic growth
- Improved traffic flow
- Improved environment
- Empowered civic engagement

Businesses can use data from these systems to improve products, innovate on new products to meet needs that may have been hidden or unknown without the data, and help cities solve seemingly intractable problems.

Here's an example. The popular traffic app Waze enables users to report on road situations in real-time like the existence of a dangerous pothole or other hazardous conditions. That data could be used by city officials to develop maintenance plans to address issues like these in more efficient ways that improve public safety. While this does involve some user interaction, it's easy to see how cars or street lamps could be fitted with artificial intelligence systems to detect conditions like these and report them in an automated way.

## IoT-enabled Retail

Analytics company SAS reports that IoT is starting to find its way into the retail experience. Using devices like RFID tags, mobile devices, and even digital signs, retail is embracing the advantages IoT can provide. The **SAS report<sup>5</sup>** highlights five places where retail is using IoT to enhance their business and customer's experience:

1. **Predictive Equipment Maintenance.** As we've talked about, equipment, like refrigerators or point-of-sale devices, that are instrumented with IoT devices can help manufacturers predict failures and alert service staff when equipment needs to be maintained.
2. **Smart Transportation.** Internet-connected trucks and even products can help optimize the logistics around getting goods from one place to another.

<sup>4</sup> <https://www.business.att.com/content/productbrochures/iot-digital-infrastructure-brief.pdf>

<sup>5</sup> [https://www.sas.com/en\\_us/insights/articles/big-data/five-iot-applications-retailers-are-using-today.html](https://www.sas.com/en_us/insights/articles/big-data/five-iot-applications-retailers-are-using-today.html)

3. **Data-aware Warehouses.** Retailers can outfit their warehouses with sensors and inventory tags so product placement in the warehouse can be optimized for easier access and quicker delivery. SAS anticipates that using automation, warehouses will “self-organize” to be the most optimal at all times.
4. **Connected Consumer.** Gone will be the days of mass marketing. Being aware of who is in your store can help you optimize sales or promotions to meet the needs of individual shoppers. Shoppers benefit because they can get a sale price on products that matter to them.
5. **Smart Store.** Imagine a mall that uses IoT to determine how customers shop and then enable mall planners to place stores that optimize around those patterns. SAS expects that this is on its way.

## IoT Adoption

In the previous section, we talked about the opportunities business can take advantage of by adopting IoT solutions. While much of that is forward-looking, you may have come to see that IoT already is being adopted by many businesses and consumers.

- In a **2018 blog post<sup>6</sup>**, Microsoft gave an overview of companies that are starting to use IoT solutions (specifically built on Azure IoT) in what they call “Smart Buildings.” These companies are using IoT technology to help people navigate buildings, trigger maintenance requests, help engineers design better buildings and help property managers oversee their properties in a more efficient and effective way.
- Technology company Intel is working with many companies to enable IoT solutions. Intel **describes solutions<sup>7</sup>** that range from building smarter trains to smarter roads. They've helped a plastics company determine inefficiencies in their control system as well as improve their product cycle and reduce inventory risks. Intel's “wearables” solutions “improve worker efficiency and safety, including hands-free operations of industrial equipment.”

## Opportunities Abound

Whether its connecting a thermostat in your home or improving the efficiency of a jet engine, IoT solutions are the next big wave in technological advancement. Businesses already have begun to adopt IoT technology and the need for IoT architects and engineers will be growing over the next decade. Come join the revolution!

## IoT for Consumer and Businesses

Consider how you might answer the following questions:

- What makes one or more connected devices an IoT implementation?
- What is the difference between consumer devices and the devices used in an IoT business implementation?

Let's take a look at the first question - what is the difference between a connected device and an IoT device? For example, a personal computer generally is connected to the Internet. Does that make it an IoT device? Is a smartwatch or door lock that both have Internet connectivity IoT devices? Definitions can be hard to come by, but it's possible to put some boundaries around devices that should be included under the umbrella of IoT and those that should not.

<sup>6</sup> <https://blogs.microsoft.com/iot/2018/06/05/smart-buildings-are-built-on-azure-iot/>

<sup>7</sup> <https://www.intel.com/content/www/us/en/internet-of-things/solution-blueprints-and-case-studies.html>

## Characteristics of an IoT Device

Here are some characteristics that an IoT device must have.

1. **Connected to the Internet.** The most obvious criterion is that the device has a connection to the Internet with a unique identifier, and two-way communication. Both of these properties are important for a device to be considered a part of an IoT system. The device has to be unique to ensure secure communications with both the server and with other devices and the device has to be able to consume and deliver data. A primary value proposition for IoT is data *collection* as well as consumption.
2. **Secure.** Being able to uniquely identify a device on the Internet or within an IoT ecosystem is one aspect of security but making a device secure also means it's resistant to being hacked (both the hardware and software), uses encrypted communication protocols, and is immune to mimicry either by an alternate hardware device or a virtual device.
3. **Smart Features.** This simply means that the device must have sensors or hardware that enable it to collect specific data based on events (like smoke in the air or a light being switched on or a key being turned). There is an implication that the device should be able to do this without user interaction so it has an "embedded intelligence."
4. **Communication Capabilities.** The device should have the capability to communicate not only with cloud-based services but with other devices.
5. **Configurable.** The device should be remotely configurable or have the ability to self-adjust its configuration based on changes in the ecosystem. This includes the ability to automatically install updates, modify sensor receptivity, repair problems, and modify energy consumption among others.
6. **Programmable.** Like connectivity to the Internet, this should be a basic function of any connected device but certainly is true of IoT devices. The main idea here is that the function of the device should be able to be modified without having to make changes to its hardware. This may mean that a device has a number of sensors that could be activated or deactivated by software or, if a single-purpose device, the features should be able to be modified by software to accomplish a different task (for example, a thermostat that can deliver outside temperature readings from a service vs. taking the internal temperature readings from a sensor).

These probably wouldn't be considered "core" features but here are other properties to consider when defining a device as an "IoT device."

- **Replaceable.** In many scenarios, when an IoT device fails (for example, a sensor on an airplane engine or wind turbine), the device should be able to be replaced and it's entire firmware and software settings loaded onto the device quickly and easily. This could mean that every programmable feature of the device should be able to be stored in the cloud and downloaded to a replacement device.
- **Environmentally Flexible.** Depending on the scenario, the device should be able to maintain power, collect and store data, and smartly upload stored data in the event of a power outage, and/or loss of Internet connectivity.

## Consumer and Business Implementations

Now let's consider the second question - what is the difference between consumer devices and the devices used in an IoT business implementation? The difference between a consumer scenario and a business scenario often comes down to how the devices are being used and why. Or viewed a bit differently, the goals of the implementation and the data being generated.

When we consider the features of IoT devices above, some may be more important in business scenarios and others take more prominence in consumer scenarios. For example, designing a device so it can easily

be replaced may be more important in mission-critical business scenarios than it would be for a consumer device that checks the weather or turns on your lights.

Let's take a look at a couple of scenarios and how they might differ. Microsoft created a **case study**<sup>8</sup> for an IoT implementation for BaxEnergy—a company that supplies analytic and optimization solutions for energy companies. While this white paper largely is about data ingestion and processing, it illustrates features of an IoT solution that is relevant for this business but may not be relevant for a consumer device. Microsoft outlines the following benefits of IoT in three distinct areas.

## Data Ingestion

This area defines how sensors collect and queue data for delivery to the database. Specifically, the Microsoft solution:

- No complex setup for data acquisition via VPN
- Workload reduced to read/write operations by establishing queues
- Data flow divided into hot and cold paths
- Asynchronous model allows for temporary storage of the data without putting more pressure on the already busy databases

## Data in Motion

This category defines how the solution improves real-time monitoring of the energy plant. The article notes the following improvements:

- Visualizing real-time monitoring without accessing the database
- Presenting the data in nearly real time
- Portal still allowing for execution of queries on historical data
- Immediate notification as soon as the device is not sending any data

## Messaging and Analytics

This category describes how incoming data is analyzed workflows are triggered based on certain event parameters.

- Creation of automatic workflows and additional services
- Instantaneous notification to wind farm operators so they can take immediate action
- Cost-effective feature
- Real-time data analytics

The advantages provided to BaxEnergy from this particular IoT implementation may be true of many business scenarios. Real time monitoring and reporting, real-time analytics, problem reporting, and asynchronous data communication are essential in many business contexts. These probably wouldn't be true in most consumer contexts.

For example, taking the paradigm case of a connected thermostat, getting real-time information about energy usage may not be that important. A customer may only need (and actually prefer) weekly or monthly reports so the data the device is collecting doesn't need to be analyzed in real time or available

<sup>8</sup> <https://microsoft.github.io/techcasestudies/iot/2017/06/30/baxenergy.html>

immediately after it's collected. Similarly, the home thermostat may not need to be able to initiate additional workflows when specific events occur or queue data if the power goes out.

## Consumer versus Business Goals for IoT

### Consumer Products and IoT

Individual consumers implement cloud connected devices (such as doorbells, thermostats, and even refrigerators) in order to make their life easier, more comfortable, or more secure. Consumer devices in the home are not typically being used for the same purpose as an IoT device implemented in a business scenario.

While IoT in the consumer space is still fairly nascent, there regularly are new examples of the technology being used to improve customer experiences and expand product features. Here are a few scenarios (some with which you may already be familiar):

- **Connected Assistants.** Devices like the **Amazon Echo<sup>9</sup>** series are becoming more and more common in homes, allowing control of smart home devices such as lights and outlets, and external connection to services such as music streaming services.
- **Connected Refrigerators.** The **Samsung "Family Hub"<sup>10</sup>** line of refrigerators includes a large, touch screen that enables customers to more easily track their food inventory through the use of interior, web-enabled cameras, an easy-to-use shopping list, calendaring and TV mirroring features. While the element of data collection and analysis isn't central to this IoT solution, the refrigerator is an early look at the potential for connected appliances.
- **Connected Doorbells and Cameras.** Many companies have gotten into the consumer doorbell and camera business. The **Ring<sup>11</sup>** system and Google's **Nest Hello<sup>12</sup>** device are examples. These doorbells record and store video and enable two-way voice and one-way video calling for people at the door. The Nest device will do facial recognition and use AI to determine which type of object it detects (car, person, animal).
- **Connected Thermostats.** As we mentioned earlier, the connected thermostat is probably the most widely-used example of an IoT consumer device most likely because it was one of the first connected devices to check all the boxes in terms of using an IoT architecture. The most famous device is the **Nest Thermostat<sup>13</sup>** but there are others. These devices enable customers to view and control their indoor temperature anywhere using a mobile device and an Internet connection, set a heating and cooling schedule, view historical data on their home's temperature and energy consumption, and even get alerts when their heater's filter needs to be changed.

There are many other connected devices coming to market, that range from practical to weird. But the possibilities are nearly endless. Consumers are moving from a mindset of experimentation to anticipation that will soon evolve into expectation as connected devices enable customers to do more.

### Business Goals and IoT

In a way, business goals for IoT are simple compared with consumer goals. Businesses tend to implement IoT solutions in order to be more profitable, to increase safety for their work force, and to more easily comply with government regulations in order to create a better business environment. Profitability can be

<sup>9</sup> <https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?node=9818047011>

<sup>10</sup> <https://www.samsung.com/us/explore/family-hub-refrigerator/connected-hub/>

<sup>11</sup> <https://ring.com/>

<sup>12</sup> <https://nest.com/doorbell/nest-hello/overview/>

<sup>13</sup> <https://nest.com/thermostats/>

realized either directly through cost reductions or indirectly through competitive advantage. For example, businesses can use IoT to reduce their manufacturing or operating costs, which increase profits directly. Or, a business could use IoT to provide customers with improved service, resulting in increased market share (and overall profits). In most cases both the business and their customer benefit.

Business goals for IoT focus on improvement in one or more of the following areas:

- Product Quality and Extended Product Lifetime
- Service Reliability and Uptime
- Operating Efficiency
- Workforce safety
- Governmental compliance

We've been looking at specific business-focused scenarios for IoT solutions but you can read more at the **Microsoft IoT site<sup>14</sup>** to see examples of how industry and vertical lines of business are using the technology.

## Strategies for Implementing IoT

As with most technology, an enterprise may adopt IoT to solve specific business problems or to take advantage of new opportunities. In either case, the process starts with a project plan that defines how the technology will be used to meet the goals of the business.

Stephanie Jernigan and Sam Ransbotham in an article for MIT Sloan Management Review **offer the following guidance<sup>15</sup>** for getting an IoT solution off the ground.

1. **Keep the initial scope small.** Since an IoT solution involves devices as well as cloud services, doing small experiments with an initially low-cost investment gives business the ability to try things and adjust quickly without spending too much capital on the front side. They write, "The result of such an approach is that future phases aren't saddled with large compatibility requirements from the first phase. Low investments mean lower sunk costs for replacement (if necessary). And fewer relationships mean fewer affected systems in other organizations."
2. **Think about the short- and long-term value of IoT.** Companies may have an initial set of metrics they want to meet or goals they want to achieve with an IoT solution. But the authors advise that businesses should stay open to possibilities that they may not conceive of at the beginning of a project. In other words, it may be best to think of implementing an IoT *solution* and think of IoT as an on-going experiment.
3. **Consider Alternatives.** A good way to figure out what an IoT implementation ought to do is to think about other ways you might get the data you need. If implementing IoT is the *best* way to get that data, then its more likely that the project will be successful.

## IoT Governance

As you plan an IoT solution, engineering and deploying the solution are just the beginning of the solution as a whole. Given the complexities of an IoT solution, planning for how an IoT solution will be maintained and monitored is essential to make the project successful. This topic is often referred to as IoT Governance, and is a topic that Microsoft and others have worked on to provide guidance.

<sup>14</sup> <https://www.microsoft.com/en-us/internet-of-things>

<sup>15</sup> <https://sloanreview.mit.edu/article/getting-started-with-iot/>

Microsoft's cloud offering, Azure, has been built from the ground up to align with the needs of the Enterprise and has created a documentation hub, the **Azure Architecture Center**<sup>16</sup>, which provides access to resources that provide general guidelines and best practices for governance strategies.

Let's look at the major aspects of IoT governance in turn.

## Develop an IoT technical strategy

Similar to the MIT Sloan Review guidance, a successful IoT deployment will include a robust planning phase that will define all aspects of the solution. IBM recommends focusing on business objectives and the team members you'll need to not only build and deploy the solution but to maintain the hardware, software, and cloud services as well as analyze and act on the data.

Microsoft's guidance stresses the importance of identifying the key stakeholders and managing cross-team buy-in at this early stage. This helps to ensure that the essential business goals are identified and on-going sponsorship is maintained through the project lifecycle. Of course, prototyping and experimentation are important in this "Proof of Value" phase as are the operational aspects of the deployment including automation and fine-tuning the solution.

In order to accelerate the Proof of Value phase, Microsoft has provided **Azure IoT Central**<sup>17</sup> - a "Software as a Service" IoT Offering, as well as a number of **preconfigured solution accelerators**<sup>18</sup> that can be used as the basis of custom solutions:

- **Remote Monitoring**<sup>19</sup>
- **Connected Factory**<sup>20</sup>
- **Predictive Maintenance**<sup>21</sup>
- **Device Simulation**<sup>22</sup>

## Define an IoT reference architecture

A reference architecture is a guide against which all IoT implementations will be based. By using a reference architecture, you can ensure that when an IoT solution is being developed for your organization, each implementation is doing things in generally the same way. It also helps ensure that best practices are being followed and that reusable elements are being shared across implementations. The governance policy should enforce that the reference architecture includes all the elements and best practices that you want to be used across IoT implementations. To support the development of a governance strategy, Microsoft has a **dedicated documentation hub for governance**<sup>23</sup> that provides a collection of concepts and services that are designed to enable management of various Azure resources at scale.

Throughout this course, we'll be referring to Microsoft's own reference architecture document as a guide for how to implement an IoT solution based on Microsoft's software and services. As the **Azure IoT Reference Architecture document**<sup>24</sup> states:

<sup>16</sup> <https://docs.microsoft.com/en-us/azure/architecture/>

<sup>17</sup> <https://azure.microsoft.com/en-us/services/iot-central/>

<sup>18</sup> <https://www.azureioticsolutions.com/Accelerators>

<sup>19</sup> <https://www.azureioticsolutions.com/Accelerators#description/remote-monitoring>

<sup>20</sup> <https://www.azureioticsolutions.com/Accelerators#description/connected-factory>

<sup>21</sup> <https://www.azureioticsolutions.com/Accelerators#description/predictive-maintenance>

<sup>22</sup> <https://www.azureioticsolutions.com/Accelerators#description/device-simulation>

<sup>23</sup> <https://docs.microsoft.com/en-us/azure/governance/>

<sup>24</sup> <https://aka.ms/iotrefarchitecture>

Every organization has unique skills and experience and every IoT application has unique needs and considerations.

So while the reference document can be a good start for your own reference architecture document, the reference architecture and technology choices recommended should be modified as needed for each.

A key aspect of any reference architecture is security. As Microsoft states:

When designing a system, it is important to understand the potential threats to that system, and add appropriate defenses accordingly, as the system is designed and architected. It is important to design the product from the start with security in mind because understanding how an attacker might be able to compromise a system helps make sure appropriate mitigations are in place from the beginning.

The Azure IoT Reference Architecture has been designed to incorporate many of the foundational governance and security guiding principals:

- **Governance Design**<sup>25</sup>
- **Prescriptive subscription governance**<sup>26</sup>
- **Internet of Things (IoT) security architecture**<sup>27</sup>

## Acquire the right roles or skills on the development team

Because of the complexity of developing and maintaining IoT solutions, IBM recommends specific roles for building the team you'll need for the entire development and support life cycle of the solution. Specifically, they recommend the solution include:

- **IoT architect** role that defines the entire solution including the strategy, integration approach, and best practices.
- **IoT developer** who is focused on implementation and definition for the technical implementation of the solution.
- **Data analyst** role which focuses on all aspects of the data collection, modeling, and analysis and reporting strategy.
- **IoT tester** who manages quality control for the entire solution and ensures the system is secure.
- **Device SME** (subject matter expert) who defines the device specifications and works with the other roles to ensure the right devices are in place and how those devices should be managed over the life cycle of the solution.
- **Security Architect**. IBM calls this out as a distinct role for good reason. Security should be thought of as a solution, according to IBM, and not merely a feature of the system. The security architect defines all aspects of the security of the solution including data collection and analysis, network operation, and governance practices (among others).

In addition to the development team, IBM recommends forming an IoT "Center of Excellence" which essentially is a governance board (or person depending on the size of the project) that is responsible for defining and governing everything from the business side of the solution to the operational side. The CoE would work with the solution architect (who most likely would be a key member) and help create the reference architecture. IBM defines a number of other roles for the CoE including analyzing the solution for reuse opportunities, promoting the adoption of best practices, and working with vendors device and platform vendors that will develop key aspects of the solution.

<sup>25</sup> <https://docs.microsoft.com/en-us/azure/architecture/cloud-adoption/governance/governance-multiple-teams>

<sup>26</sup> <https://docs.microsoft.com/en-us/azure/architecture/cloud-adoption/appendix/azure-scaffold>

<sup>27</sup> <https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-security-architecture>

## Define your IoT governance processes and policies

All of the above would fall under IoT governance and requires written policies and processes that should be "followed, applied, and enforced" to make the IoT solution successful and secure.

## Business Case Investigations

As you might imagine, IoT can be implemented in both small and large businesses, and in nearly any type of business as well. Some of the industry verticals that have a high adoption rate are listed here:

- Manufacturing
- Healthcare
- Retail
- Smart Cities and Buildings
- Transportation and Logistics
- Energy
- Agriculture

The sections below investigate a number of these verticals in more detail.

## IoT For Manufacturing

Instrumenting a factory floor with IoT devices can enable factory managers and line workers to better anticipate problems, understand where critical systems are failing and improve the overall operation of a single factory or a network of factories around the world. The following list describes benefits of a connected factory solution.

- Monitor manufacturing equipment: Improve processes using industrial IoT. Use sensors and advanced analytics to predict needed maintenance, and reduce unplanned downtime cutting into production time. See <https://blogs.microsoft.com/iot/2016/04/29/intelligent-manufacturing-takes-plant-operations-to-the-next-level-of-iot/>
- Monitor customer equipment: Create new business models that offer predictive maintenance and performance monitoring for the equipment you produce, delivering a richer customer experience.
- Improve field service: Access sensor data to improve field service scheduling, ensuring the right technicians and tools are dispatched before potential issues become a major problem.

In addition to the general benefits listed above, IoT for manufacturing can be associated with the following implementations:

- **Capitalize on the factory of the future:** The factory of the future is enabling companies to seize new opportunities by expanding their value chain, implementing more agile production, and discovering new revenue streams. At the center of this change is IoT's ability to transform the entire connected manufacturing ecosystem, from R&D, through the supply chain, to customer service. See <https://enterprise.microsoft.com/en-us/trends/the-factory-of-the-future/>
- **Competing in the Digital Age of Manufacturing:** Operational technology and information technology are coming together for the first time, creating new opportunities for digital manufacturers. Connected products, people, and things produce terabytes of data every day that manufacturers can access and extract deep insights to optimize business and manufacturing processes better than ever before. More importantly, insight from this data helps manufacturers identify new revenue streams by developing high-value service offerings focused on how products and customers interact in the real

world. This transformation is changing the landscape for manufacturers, enabling them to differentiate themselves, achieve operational excellence and disrupt markets.

- **The Connected Factory:** An IoT-enabled factory will use devices that communicate with each other and the cloud to capture and collect real-time data to enable a more efficient and productive factory. At a very high level, a connected factory will use IoT devices to communicate machine information with other machines as well as cloud services, collect and store data from those machines, and enable the creation of reports and alerts.

## IoT for Healthcare

It may seem odd to think of healthcare as an industry since it's in the "business" of dealing with life and death, but the reality is that there is a lot of money involved in providing healthcare and finding ways to do more with less is a constant goal. But IoT has the potential for impacting much more than the economics of healthcare. IoT has the potential to be a disrupter in this vertical by providing better optimization of equipment and personnel, enable non-traditional options for self care, and help monitor and maintain essential equipment used in providing care.

- **Personalized Healthcare:** While there are many potential applications for IoT in the healthcare industry, applying solutions can be tricky as concerns about patient privacy, misdiagnoses, security breaches, and malfunctioning equipment can make patients and doctors hesitant about adopting the technology. When architecting an IoT solution for healthcare that involves direct patient interaction, special care has to be taken to account for these factors.
- **Short-term care planning:** Providing patients with a plan for self care once they leave a medical facility as well as reminding patients of their plan and monitoring how well they're following that plan is one scenario the reports calls out as an area ripe for IoT solutions. For example, an app on a mobile device or wearable that can help remind patients to take medication or follow a physical therapy routine and monitor how often they follow those instructions can help healthcare providers understand challenges, prepare for follow up visits, and create modifications to a plan that may be more effective for specific patients.
- **Chronic-disease management and home care:** IoT devices and services can help those in long-term care, chronic-disease management and in frail condition to give health-care providers real-time information on their condition (e.g. an elderly person that has fallen or a diabetic whose glucose levels have reached critical levels). And the collection of data across hundreds of thousands of patients can help healthcare providers build models that will service new patients in the future.
- **Population-based evidence creation:** Big data collected by IoT systems can help healthcare providers better understand health trends across populations and be used to prevent disease rather than merely manage it.

## IoT for Retail

In today's world, the topic of IoT is often introduced in the context of connected stores that use RFID tags to monitor inventory and enable a cashless payment system. Retail has the potential to offer the easiest entry point for an IoT developer since the vertical is so massive in both scope and opportunity. From vending machines to signage, the potential for IoT to disrupt the retail experience—for sellers and consumers—is massive.

While there are others, we can define four main benefits that IoT solutions offer to retailers and customers:

- **Operational Efficiency:** As we saw with IoT for Manufacturing, in the retail space, IoT can enable retailers to optimize their staff, delivery, security, and other parts of their supply chain. This can ensure

the right products are on the shelf at the right time, that customers are kept safe at all hours of the day or night, and that theft and loss are kept to a minimum. Equipping shipping containers that enable stores to track product availability, using cloud-enabled web cameras and using mobile devices to enable easier communication with staff and track their hours can reduce the overall load retailers are burdened with.

- **Inventory Management:** In a paper<sup>28</sup> outlining the benefits of IoT for the retail experience, Intel makes the claim that IoT can help retailers "achieve near-100-percent inventory accuracy." And this isn't merely about preventing loss. Using big data, retailers can ensure that products customers want and care about are always on the shelf and items purchased less often don't consume valuable shelf space.
- **Improved Customer Experience:** As we saw above with the Amazon Go store experience, customers can benefit from IoT solutions in retail by being able to purchase the products they want with less friction. By eliminating checkout lines and eliminating the need for customers to carry cash or credit cards, customers get more time back and can be safer shopping. Of course, privacy is always a concern in these scenarios so IoT architects need to take great care to find the right balance between technical advances and privacy concerns.
- **Cost Efficiency and Savings:** Each of the items listed above should result in more efficient operations and reduce costs in terms of both running the business and supporting customers. The idea is that retailers can not only outfit the retail experience with an IoT architecture but outfit the stores themselves to make retail operations "smart". More on this in the section discussing smart buildings.

## IoT for Smart Cities and Buildings

If you've installed and used any one of the many "smart home" devices on the market today, you may already be seeing the potential value of a smart home: improved safety through lock monitoring, automatic lighting, and security cameras and increased productivity and efficiency through home automation. These consumer scenarios, when extrapolated to the scale of a town or city, give us an idea of what is possible to make the places where we live, work, and play safer and more efficient.

While the architects of IoT solutions need to consider the right balance between innovation and personal privacy, the opportunities to improve civic life through IoT are enormous and, with proper controls and privacy considerations in place, can make communities safer, more efficient, and easier to live in.

- **Optimize natural resource use:** Making better use of natural resources is more than a political slogan. Doing more with less makes good economic sense (as any successful business person can attest to) and makes for better stewardship in using the resources we have. IoT implementations can help the citizens of cities minimize waste, use power more intelligently, and make the best use of resources for farming, mining, and energy.
- **Create safer cities:** City life is a buzzing array of complex variables that are under continual change. While cities have a "heartbeat" and rhythm, the regularity is a product of a variety of individual activities all occurring at the same time. From an IoT perspective, this can create a data management and analytics challenge. But it also creates a lot of problems for city managers:
  - How does a city ensure emergency services have enough resources to meet the demands of the city?
  - How do managers of those emergency resources deploy the resources in the most efficient ways to keep citizens safe?
  - How does a city manage high demand (say for a significant weather event)?

<sup>28</sup> <https://www.intel.com/content/www/us/en/internet-of-things/solution-briefs/smart-retail-solutions-top-10.html>

- Does demand vary based on the time of the day or the time of the year?
- **Create smart buildings:** As we've been seeing in the other verticals, IoT technology can bring value to a segment or vertical by collecting and aggregating data for insights that formerly would have been hidden without the data collection. These insights can help enable cost savings, efficiency, and even comfort and safety. All these are true when it comes to smart buildings.
- **Improve field service:** A "smart city" can provide quicker and more efficient services for public equipment and utilities. Providing the ability for municipalities and service providers to instrument equipment, get data, and respond to problems, can make cities safer and make life better in the city. From repairing street lamps and roadways to monitoring and anticipating problems in public utilities, IoT can help cities better maintain their infrastructure and respond to problems.

## IoT for Transportation and Logistics

Transportation advances are perhaps the most exciting area where IoT solutions are being developed. Getting from one place to another can be an arduous, frustrating, and sometimes dangerous thing to do. Anyone who has spent more than an hour sitting in traffic has said to themselves, "There has to be a better way." What makes this vertical so exciting is the many opportunities for improvement and innovation it presents.

Autonomous vehicles and self-driving cars is only a small part of the opportunity that this segment of IoT can address. As we saw in the "smart city" discussion, transportation must be viewed as an ecosystem where each of the parts are working together. The more complex the variables in that ecosystem, the more complex the solution will be.

And while economics is a factor in transportation solutions, governments may be more interested in efficiency and safety which creates a different dynamic for solution architects compared to the other verticals we've been looking at.

# Introduction to IoT Solution Architecture

## Data Flow and Processing

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

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

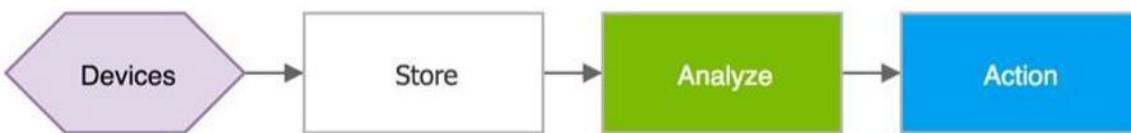
- Storage includes **in-memory caches<sup>29</sup>**, temporary queues and permanent archives (e.g. a database).
- Routing allows sending data records to one or more storage endpoints, analysis processes, and actions. Routing makes decisions on what data should go which target and when.
- Analysis is used to run data records through a set of conditions and can produce different output data records. For instance, input telemetry data encoded in one format may return output telemetry **encoded<sup>30</sup>** in another format.
- Original input data records and analysis output records are typically stored and available to display, and may trigger actions such as emails, instant messages, incident tickets, CRM tasks, device commands, etc.

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

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



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

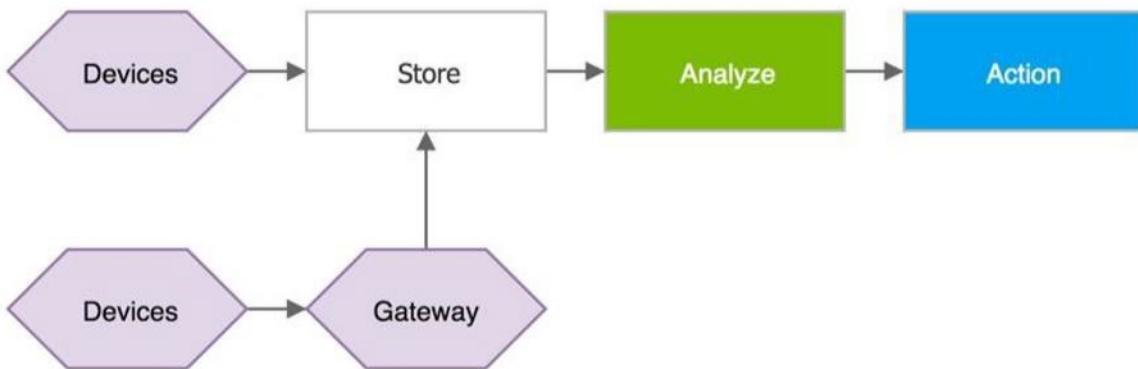


IoT architectures can also support multiple systems that can accept and do something with data. For instance, some telemetry storage and/or analysis may occur on premise, within devices and field/edge gateways. In other scenarios, **protocol translations<sup>31</sup>** may be required to connect constrained devices to the cloud. While the resulting graph is more complex, the logical building blocks are the same:

<sup>29</sup> [https://en.wikipedia.org/wiki/CPU\\_cache](https://en.wikipedia.org/wiki/CPU_cache)

<sup>30</sup> <https://en.wikipedia.org/wiki/Code>

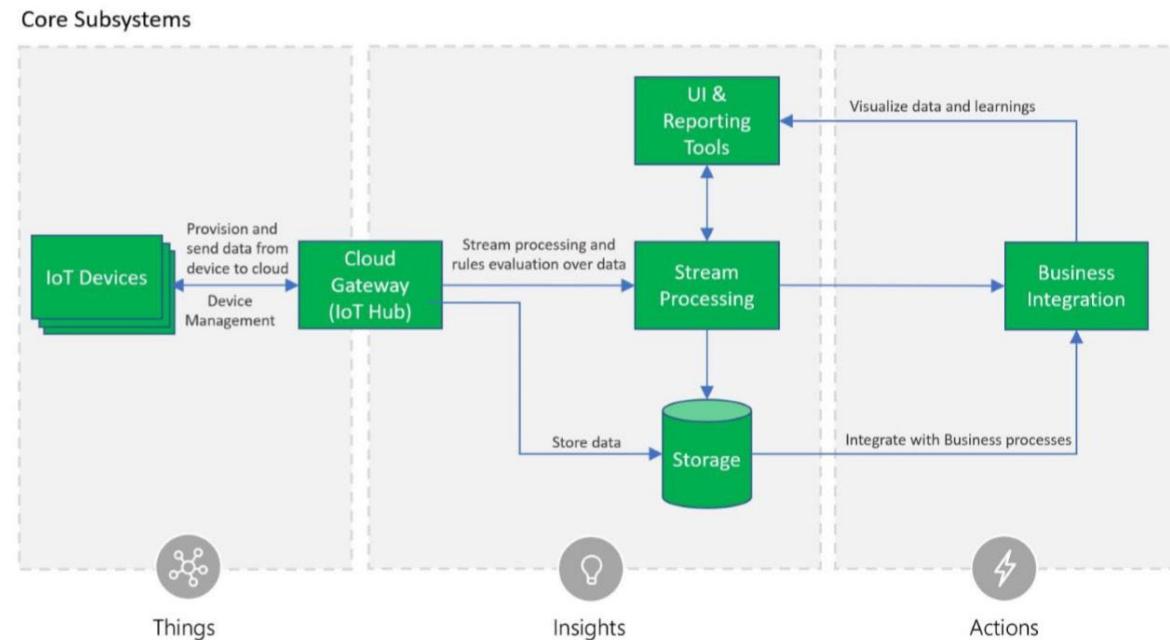
<sup>31</sup> [https://en.wikipedia.org/wiki/Protocol\\_converter](https://en.wikipedia.org/wiki/Protocol_converter)



## Subsystems of an IoT Architecture

At its core, an IoT solution consists of the following subsystems: 1) devices that have the ability to securely register with the cloud, and connectivity options for sending and receiving data with the cloud, 2) a cloud gateway service, or hub, to securely accept that data and provide device management capabilities, 3) stream processors that consume that data, integrate with business processes, and place the data into storage, and 4) a user interface to visualize telemetry data and facilitate device management.

These core subsystems can be aligned to the Things/Insights/Actions model that we discussed earlier.



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

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

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

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

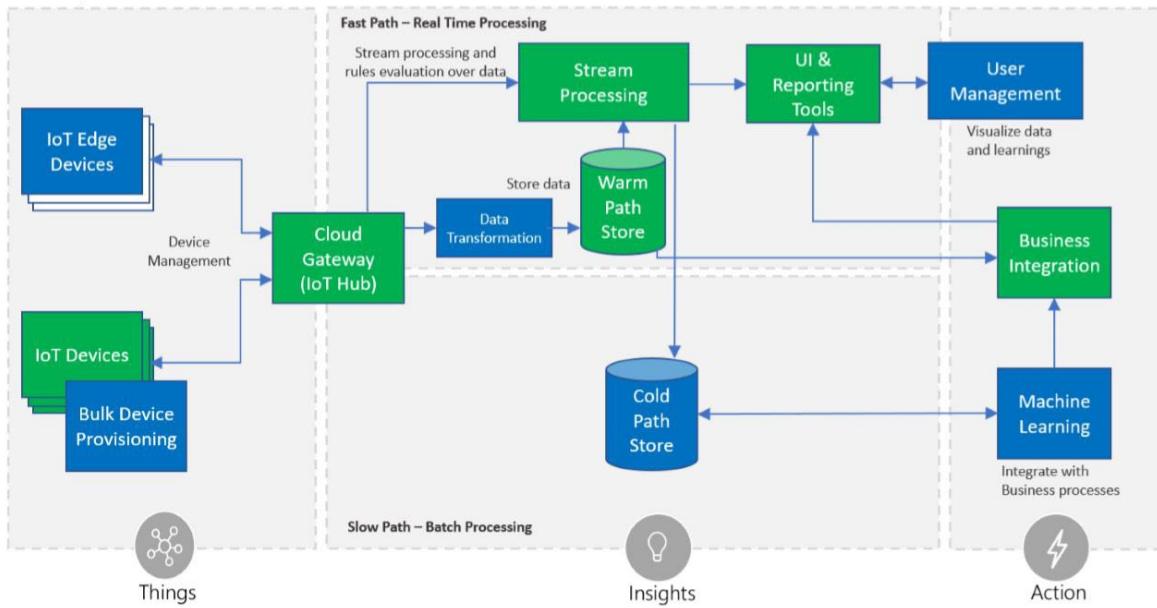
MCT USE ONLY. STUDENT USE PROHIBITED

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

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

## Optional Subsystems

In addition to the core subsystems many IoT applications will include subsystems for: 5) telemetry data transformation which allows restructuring, combination, or transformation of telemetry data sent from devices, 6) machine learning which allows predictive algorithms to be executed over historical telemetry data, enabling scenarios such as predictive maintenance, and 7) user management which allows splitting of functionality amongst different roles and users.



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

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

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

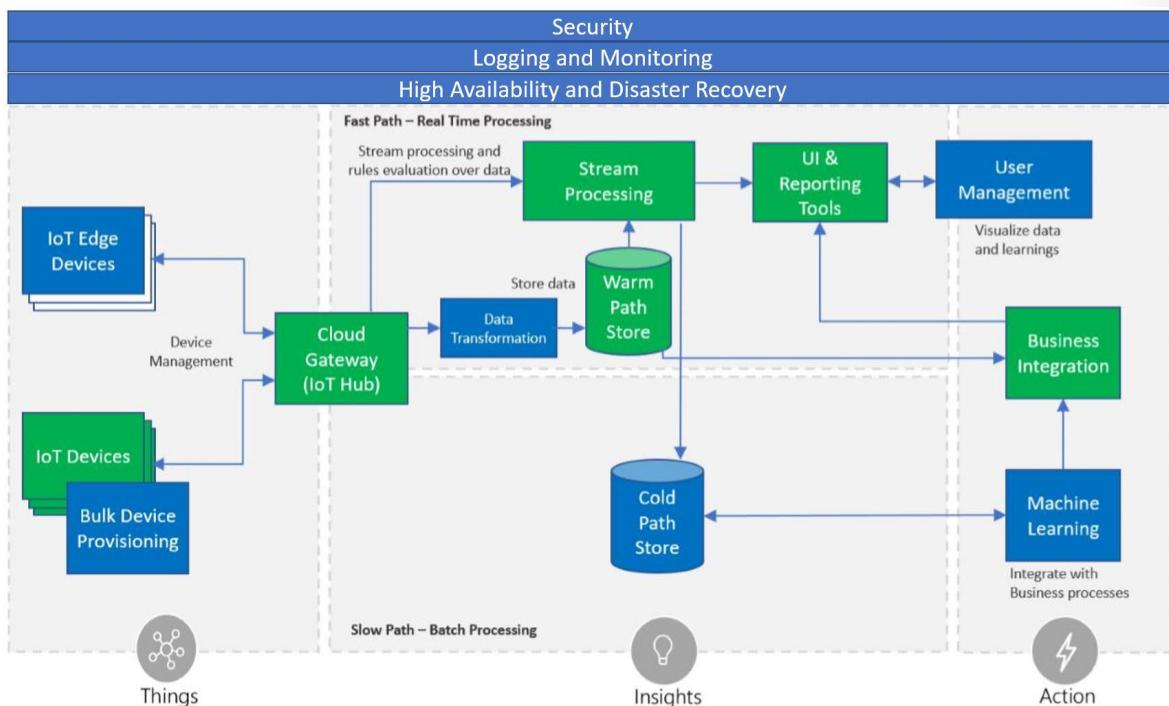
**Edge Devices:** These devices serve an active role in managing access and information flow. They may assist in device provisioning, data filtering, batching and aggregation, buffering of data, protocol translation, event rules processing, and more.

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

## Cross-Cutting Architectural Needs

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

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



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

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

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

# IoT Hardware and Cloud Services

## IoT Hardware Components

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

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

### IP-enabled IoT Devices

An IP-enabled device is, simply, a device that can establish a 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**"<sup>32</sup> and defines the way messages are delivered over a network. A message in networking terms is just a packet of information and single packet could deliver part of a text message or a video file. Most data that is transferred over the Internet uses this communication protocol.

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

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

### Non-IP Enabled Devices

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

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

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

<sup>32</sup> [https://en.wikipedia.org/wiki/Internet\\_Protocol](https://en.wikipedia.org/wiki/Internet_Protocol)

## Sensors

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

A **smart sensor** according to [the website<sup>33</sup>](#) *IoT Agenda* is "a device that takes input from the physical environment and uses built-in compute resources to perform predefined functions upon detection of specific input and then process data before passing it on." That is, the device itself processes the data to some degree before sending it to the next node in the IoT architecture.

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

## IoT Edge Devices and Field Gateways

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

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

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

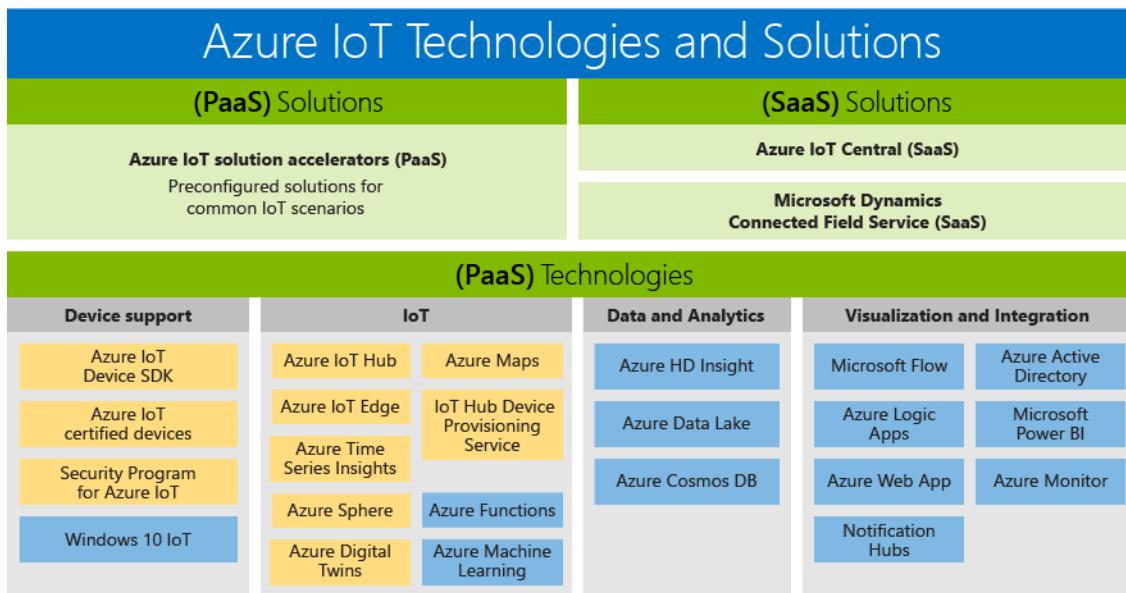
## Other Hardware

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

## Azure IoT Services and Technologies

Microsoft has built a portfolio that supports the needs of all customers, enabling everyone to access the benefits of digital transformation. The Azure IoT product portfolio is an overview of the available PaaS/SaaS technologies and solutions.

<sup>33</sup> <https://internetofthingsagenda.techtarget.com/definition/smart-sensor>



## Solutions

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

### Azure IoT solution accelerators (PaaS)

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

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

### Azure IoT Central (SaaS)

Azure IoT Central is a fully managed SaaS solution, which allows you to get started quickly with minimal IoT experience. If your business is pursuing speed over customization, SaaS models could be the perfect fit for your IoT implementation needs.

Organizations with fewer device models, more predictable scenarios, and limited IoT/IT capabilities can now reap the benefits of IoT through a SaaS approach. Businesses that previously lacked the time, money, and expertise to develop connected products, can now get started quickly with Azure IoT Central. Microsoft is leading the industry in providing a mature SaaS solution that addresses common IoT implementation requirements.

### Technologies (PaaS)

With the most comprehensive IoT portfolio of platform services, Platform-as-a-Service (PaaS) technologies that span the Azure platform enable you to easily create, customize, and control all aspects of your

IoT solution. Establish bi-directional communications with billions of IoT devices and manage your IoT devices at scale. Then integrate your IoT device data with other platform services, such as Azure Cosmos DB and Azure Time Series Insights, to enhance insights across your solution.

## Device support

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

## IoT Hub

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

## IoT Edge

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

## Spatial Intelligence

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

## Data and analytics

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

## Introduction to IoT Device Software

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

- Availability of the software you need
- Compatibility of the software with the devices you've chosen

- Compatibility with other software/cloud systems in your solution
- Reputation and longevity of the software provider
- The commitment of the software provider to update the operating system and tools to address security issues, bugs, and new features.
- Security and privacy requirements

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

## Device Operating Systems

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

OS	Type	Description
<b>Windows IoT Core</b> ( <a href="https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core">https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core</a> )	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.
<b>Ubuntu Core</b> ( <a href="https://www.ubuntu.com/core">https://www.ubuntu.com/core</a> )	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."
<b>Riot</b> ( <a href="https://www.riot-os.org/">https://www.riot-os.org/</a> )	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."

OS	Type	Description
<b>QNX</b> ( <a href="https://www.qnx.com">https://www.qnx.com</a> )	Managed	"Through sophisticated software solutions BlackBerry® QNX helps ensure the proper operation of embedded systems that require mission-critical and life-critical operations. Since 1980, BlackBerry QNX has established itself as a trusted partner for companies building vehicles, medical devices, heavy machinery, power and energy, robotics, and industrial automation systems that are required to be safety-certified, extremely reliable, and highly secure."
<b>Android Automotive</b> ( <a href="https://en.wikipedia.org/wiki/Android_Automotive">https://en.wikipedia.org/wiki/Android_Automotive</a> )	Managed	"Android Automotive is a variation of Google's Android operating system, tailored for its use in vehicle dashboards."

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

## Programming Languages

When it comes to programming devices, the operating system running on the device may determine what languages can be used to program it. Many modern hardware devices can support multiple languages and board engineers may develop specific flavors of hardware to support various languages. Microsoft's IoT core, for example, supports most languages that Windows develops 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.

- 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.
- Think about your development team.** When considering the programming languages you want to use in your solution, you will need to consider whether you want to use talent you already have at your disposal, bring on new resources, or use a blend of both. If your current software development team knows C# but doesn't know Python, choosing a platform that supports C# as a programming

language will most likely enable you to get to market quicker than having either to train existing talent in another language or bring on new talent that knows an alternate language.

3. **Think about your broader software environment.** Similar to item 2 above, when you think about what software platform you want to use, it can be helpful to think about the development environment across your business group or enterprise. By using a language that already is deployed in other areas of your business can make tasks like resource balancing, code sharing, source control, hiring, and similar factors more efficient.
4. **Choose a device or devices platform.** Once you've figured out what data you want to collect and have thought about your larger ecosystem, you'll be better informed when it comes to choosing a device platform. As we've said in other lessons, you may need more than one device platform so choosing platforms that are the most compatible with items 1-4 above will give you a more efficient overall environment in which to develop your solution.

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

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

## Software Development Kits

Before we leave this topic, let's briefly discuss IoT software development kits (SDKs) as a means by which you can more quickly create an environment to build your solution. If you've never used an **SDK<sup>34</sup>** before, these kits can accelerate the software development process by providing the developer with all the necessary tools, software packages, and integration software necessary to build a complete solution. While the degree to which any given SDK does this will vary with providers and companies, a good SDK will provide many if not most of the software-related tools needed to build a solution.

## Cloud Service Components of an IoT Solution

In this topic, you'll learn:

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

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

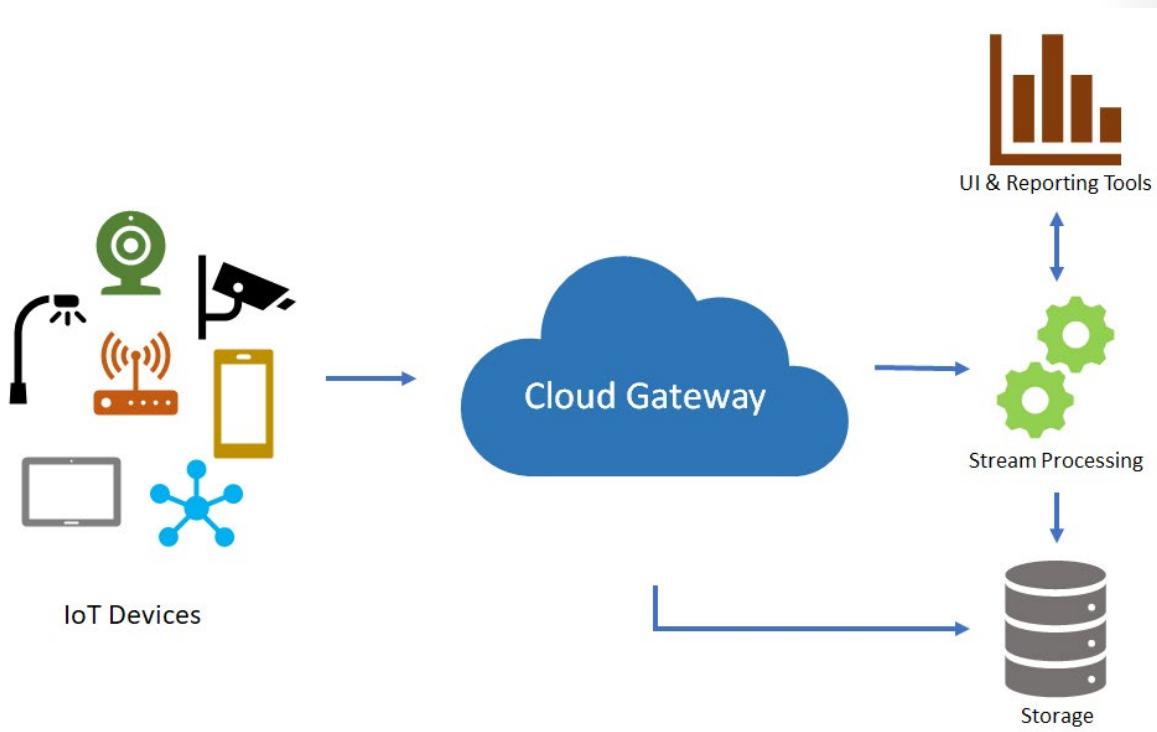
## Cloud Gateways

In an earlier topic in this lesson, we looked at, briefly, the concept of a field gateway—a piece of hardware that brokers communication between IoT devices and cloud services. Cloud gateways do more than

<sup>34</sup> [https://en.wikipedia.org/wiki/Software\\_development\\_kit](https://en.wikipedia.org/wiki/Software_development_kit)

broker communication. They provide a set of services that devices can run either locally or in the cloud. Cloud gateways can provide workloads such as (among others):

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



## Data Storage Options

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

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

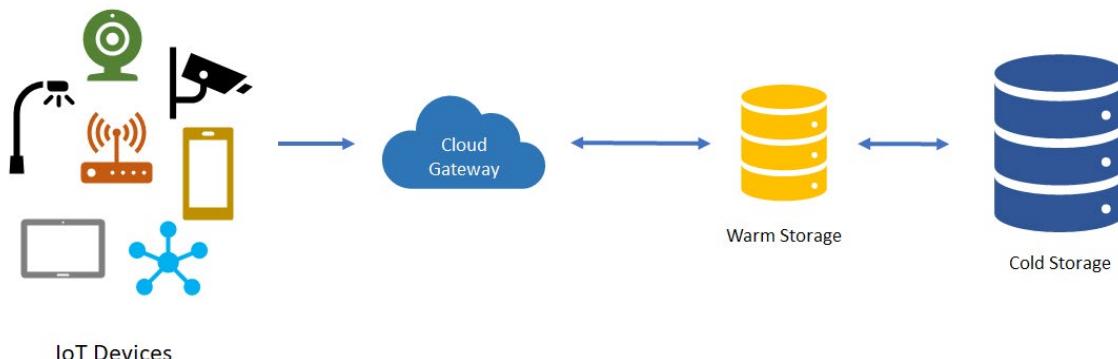
Here are some other concepts to be aware of when considering cloud storage.

Data is often time series data and is required to be stored where it can be used in visualization and reporting as well as later accessed for additional processing. It is common to have data split into “warm” and “cold” data stores. The **warm data store** holds recent data that needs to be accessed with low

latency. Data stored in **cold storage** is typically historical data. Most often the cold storage database solution chosen will be cheaper in cost but offer fewer query and reporting features than the warm database solution.

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

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



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

[^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 Architecture document**<sup>35</sup>.

## Analytics Services and Data Visualization

### Analytics

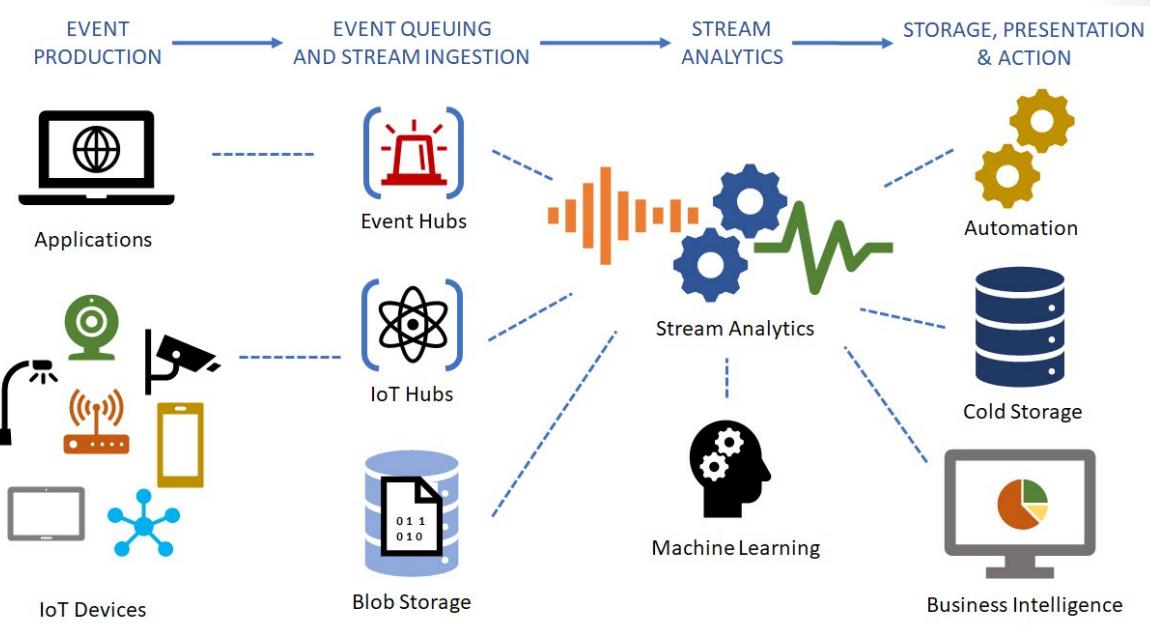
Once data is captured and stored, it only becomes useful when it provides insights into the physical world from which your IoT devices have captured the data. This is where analytic services come into play.

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**<sup>36</sup>. 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.

<sup>35</sup> <https://aka.ms/iotrefarchitecture>

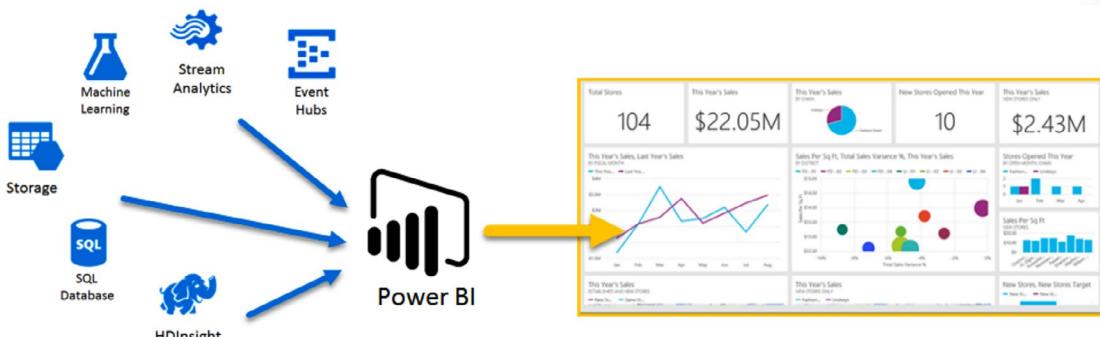
<sup>36</sup> [https://en.wikipedia.org/wiki/Semantic\\_data\\_model](https://en.wikipedia.org/wiki/Semantic_data_model)



## Data Visualization

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

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



## Machine Learning

Machine Learning (ML) is one of the more exiting developments in modern computer science. It's a complex field but one that is producing significant positive results with large datasets. As we've said throughout this course, IoT devices produces large large volumes of data. Analytic systems help engineers to model the existing data in meaningful ways. **Machine learning**<sup>37</sup> 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.

<sup>37</sup> [https://en.wikipedia.org/wiki/Machine\\_learning](https://en.wikipedia.org/wiki/Machine_learning)

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 quickly and get to the customer more quickly.

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

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

## Conclusion

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

## Features of Azure IoT Hub

IoT Hub is a managed service, hosted in the cloud, that acts as a central message hub for bi-directional communication between your IoT application and the devices it manages. You can use Azure IoT Hub to build IoT solutions with reliable and secure communications between millions of IoT devices and a cloud-hosted solution backend. You can connect virtually any device to IoT Hub.

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

IoT Hub's capabilities help you build scalable, full-featured IoT solutions such as managing industrial equipment used in manufacturing, tracking valuable assets in healthcare, and monitoring office building usage.

## Scale your solution

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

## Secure your communications

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

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

## Route device data

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

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

## Integrate with other services

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

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

## Configure and control your devices

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

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

## Make your solution highly available

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

## Connect your devices

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

## Quotas and limits

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

# Features of Azure IoT Hub Device Provisioning Service

## Registration and provisioning

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

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

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

## When to use Device Provisioning Service

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

- Zero-touch provisioning to a single IoT solution without hardcoding IoT Hub connection information at the factory (initial setup)
- Load balancing devices across multiple hubs
- Connecting devices to their owner's IoT solution based on sales transaction data (multitenancy)
- Connecting devices to a particular IoT solution depending on use-case (solution isolation)
- Connecting a device to the IoT hub with the lowest latency (geo-sharding)
- Reprovisioning based on a change in the device

- Rolling the keys used by the device to connect to IoT Hub (when not using X.509 certificates to connect)

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

## Features of the Device Provisioning Service

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

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

MCT USE ONLY. STUDENT USE PROHIBITED

## Lab Scenario

### Scenario Overview

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

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

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

### Lab Scenario

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

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

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

## Resources and Unique Names

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

### Unique ID

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

YourInitialsYYMMDD

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

GWB190123  
BHO190504  
CAH191216  
DM190911

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

gwb190123  
bho190504  
cah191216  
dm190911

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

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

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

## Resource Groups

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

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

## Publicly Visible Resources

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

If your Unique ID is: **CAH191216**

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

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

```
#!/bin/bash
```

```
YourID="{YOUR-ID}"  
RGName="AZ-220-RG"  
IoTHubName="AZ-220-HUB-$YourID"
```

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

```
private string _yourId = "{YOUR-ID}";  
private string _rgName = "AZ-220-RG";  
private string _iotHubName = $"AZ-220-HUB-{_yourId}";
```

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

## Course Labs

### Module Labs

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

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

**WARNING:** Be prepared for UI changes

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

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



## Module 2 Devices and Device Communication

### IoT Hub Concepts

#### IoT Hub Tiers

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

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

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

- What features do I plan to use?

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

- How much data do I plan to move daily?

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

#### Basic and standard tiers

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

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

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

## Message throughput

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

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

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

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

## Partitions

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

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

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

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

## Tier upgrade

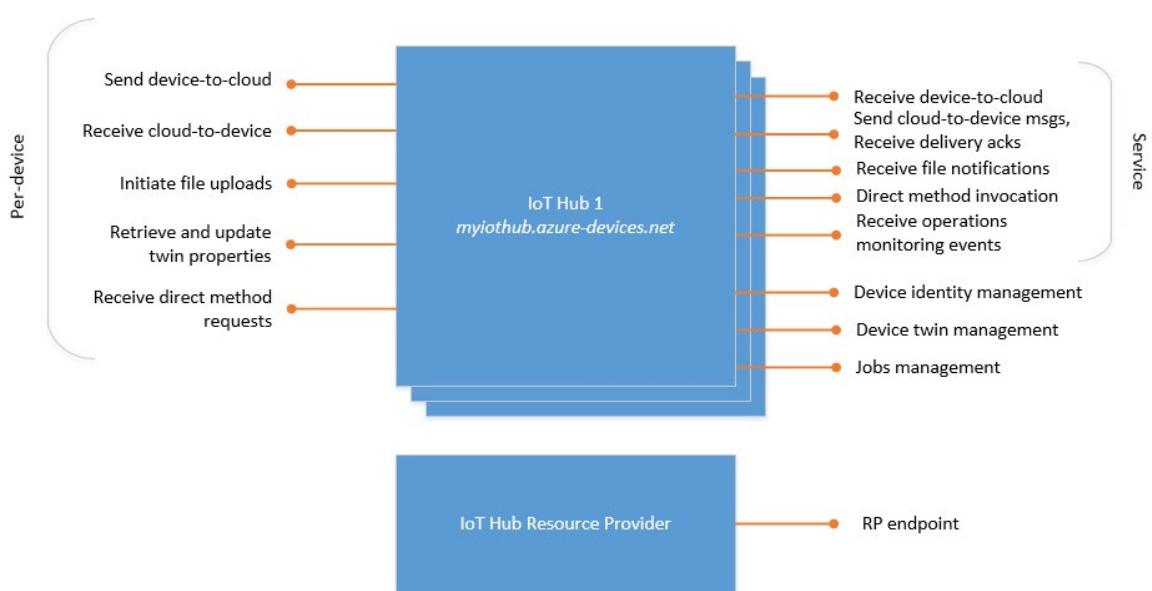
Once you create your IoT hub, you can upgrade from the basic tier to the standard tier without interrupting your existing operations. The partition configuration remains unchanged when you migrate from basic tier to standard tier.

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

## IoT Hub Endpoints

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

## Built-in Endpoints



The following list describes the endpoints:

- Resource provider. The IoT Hub resource provider exposes an Azure Resource Manager interface. This interface enables Azure subscription owners to create and delete IoT hubs, and to update IoT hub properties. IoT Hub properties govern hub-level security policies, as opposed to device-level access control, and functional options for cloud-to-device and device-to-cloud messaging. The IoT Hub resource provider also enables you to export device identities.
- Device identity management. Each IoT hub exposes a set of HTTPS REST endpoints to manage device identities (create, retrieve, update, and delete). Device identities are used for device authentication and access control.
- Device twin management. Each IoT hub exposes a set of service-facing HTTPS REST endpoints to query and update device twins (update tags and properties).
- Jobs management. Each IoT hub exposes a set of service-facing HTTPS REST endpoint to query and manage jobs.
- Device endpoints. For each device in the identity registry, IoT Hub exposes a set of endpoints:
  - Send device-to-cloud messages. A device uses this endpoint to send device-to-cloud messages.
  - Receive cloud-to-device messages. A device uses this endpoint to receive targeted cloud-to-device messages.
  - Initiate file uploads. A device uses this endpoint to receive an Azure Storage SAS URI from IoT Hub to upload a file.
  - Retrieve and update device twin properties. A device uses this endpoint to access its device twin's properties.
  - Receive direct method requests. A device uses this endpoint to listen for direct method's requests.These endpoints are exposed using MQTT v3.1.1, HTTPS 1.1, and AMQP 1.0 protocols. AMQP is also available over WebSockets on port 443.
- Service endpoints. Each IoT hub exposes a set of endpoints for your solution back end to communicate with your devices. With one exception, these endpoints are only exposed using the AMQP protocol. The method invocation endpoint is exposed over the HTTPS protocol.
  - Receive device-to-cloud messages. This endpoint is compatible with Azure Event Hubs. A back-end service can use it to read the device-to-cloud messages sent by your devices. You can create custom endpoints on your IoT hub in addition to this built-in endpoint.
  - Send cloud-to-device messages and receive delivery acknowledgments. These endpoints enable your solution back end to send reliable cloud-to-device messages, and to receive the corresponding delivery or expiration acknowledgments.
  - Receive file notifications. This messaging endpoint allows you to receive notifications of when your devices successfully upload a file.
  - Direct method invocation. This endpoint allows a back-end service to invoke a direct method on a device.
  - Receive operations monitoring events. This endpoint allows you to receive operations monitoring events if your IoT hub has been configured to emit them. For more information, see IoT Hub operations monitoring.

## Custom Endpoints

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

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

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

## Introduction to IoT Hub Security Features

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

### Access control and permissions

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

You can grant permissions in the following ways:

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

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

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

For example, in a typical IoT solution:

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

### Authentication

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

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

## Scope IoT hub-level credentials

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

## Security tokens

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

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

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

## Supported X.509 certificates

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

Supported certificates include:

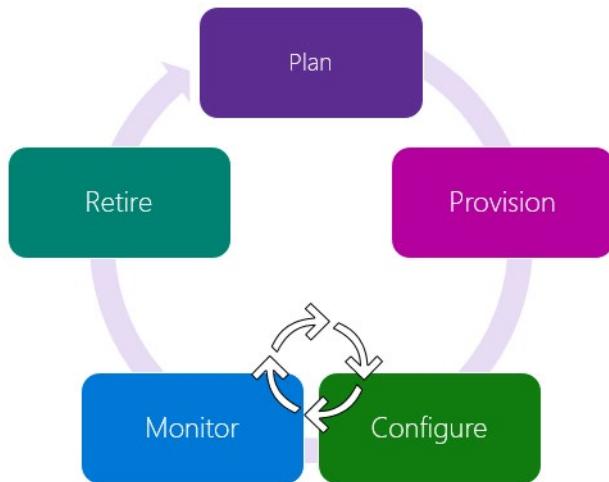
- An existing X.509 certificate. A device may already have an X.509 certificate associated with it. The device can use this certificate to authenticate with IoT Hub. Works with either thumbprint or CA authentication.
- CA-signed X.509 certificate. To identify a device and authenticate it with IoT Hub,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

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

# IoT Device Lifecycle Concepts

## Device Lifecycle Terms and Concepts

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



Each of the five stages includes goals and activities:

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

## Azure IoT Devices

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

**Note:** Microsoft maintains an online device catalog that provides a list of hardware devices certified to work with IoT Hub - **Azure Certified for IoT Device Catalog<sup>1</sup>**.

## Azure IoT Device Types

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

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

## Device Identity and Registration

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

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

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

Use the identity registry when you need to:

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

## Module Identity

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

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

<sup>1</sup> <https://catalog.azureiotsolutions.com/alldevices>

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

## Identity registry operations

The IoT Hub identity registry exposes the following operations:

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

## Create an IoT Device using the Azure Portal

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

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

## Introduction to Device Twins

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

Device twins store device-related information that:

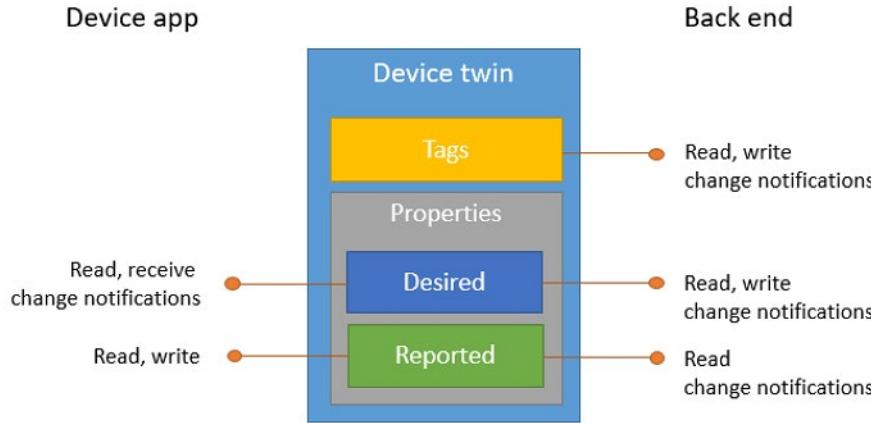
- Device and back ends can use to synchronize device conditions and configuration.
- The solution back end can use to query and target long-running operations.

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

A device twin is a JSON document that includes:

- Tags. A section of the JSON document that the solution back end can read from and write to. Tags are not visible to device apps.
- Desired properties. Used along with reported properties to synchronize device configuration or conditions. The solution back end can set desired properties, and the device app can read them. The device app can also receive notifications of changes in the desired properties.

- Reported properties. Used along with desired properties to synchronize device configuration or conditions. The device app can set reported properties, and the solution back end can read and query them.
- Device identity properties. The root of the device twin JSON document contains the read-only properties from the corresponding device identity stored in the identity registry.



## Example

The following example shows a device twin JSON document:

```
{
  "deviceId": "devA",
  "etag": "AAAAAAAAAAc=",
  "status": "enabled",
  "statusReason": "provisioned",
  "statusUpdateTime": "2001-01-01T00:00:00",
  "connectionState": "connected",
  "lastActivityTime": "2015-02-30T16:24:48.789Z",
  "cloudToDeviceMessageCount": 0,
  "authenticationType": "sas",
  "x509Thumbprint": {
    "primaryThumbprint": null,
    "secondaryThumbprint": null
  },
  "version": 2,
  "tags": {
    "$etag": "123",
    "deploymentLocation": {
      "building": "43",
      "floor": "1"
    }
  },
  "properties": {
    "desired": {
      "telemetryConfig": {
        "interval": "10s"
      }
    }
  }
}
```

```
        "sendFrequency": "5m"
    },
    "$metadata" : {...},
    "$version": 1
},
"reported": {
    "telemetryConfig": {
        "sendFrequency": "5m",
        "status": "success"
    },
    "batteryLevel": 55,
    "$metadata" : {...},
    "$version": 4
}
}
```

## Usage

Use device twins to:

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

## Purpose and Best Practices

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

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

## Introduction to Device Monitoring

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

*Further reading:*

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

## Introduction to Device Retirement

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

*Further reading:*

- How to use device twin properties
- Manage device identities

# IoT Developer Tools

## IoT Developer Tools Overview

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

The primary developer tool categories for Azure IoT include:

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

## Azure IoT Hub SDKs

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

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

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

- IoT Hub Device SDKs enable you to build apps that run on your IoT devices using device client or module client. These apps send telemetry to your IoT hub, and optionally receive messages, job, method, or twin updates from your IoT hub. You can also use module client to author modules for Azure IoT Edge runtime.

- IoT Hub Service SDKs enable you to build backend applications to manage your IoT hub, and optionally send messages, schedule jobs, invoke direct methods, or send desired property updates to your IoT devices or modules.

## Azure IoT Device SDKs Platform Support

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

Microsoft provides SDK support in the following ways:

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

## Azure IoT Hub Device SDKs

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

## Coding Language Support

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

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

## Platform Testing

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

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

## Supported Protocols

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

- MQTT
- MQTT over WebSockets
- AMQP
- AMQP over WebSockets
- HTTPS

## Azure IoT Hub Service SDKs

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

## Coding Language Support

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

## Backend App Scenarios

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

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

	Direct methods	Twin's desired properties	Cloud-to-device messages
Scenario	Commands that require immediate confirmation, such as turning on a fan.	Long-running commands intended to put the device into a certain desired state. For example, set the telemetry send interval to 30 minutes.	One-way notifications to the device app.

	Direct methods	Twin's desired properties	Cloud-to-device messages
Data flow	Two-way. The device app can respond to the method right away. The solution back end receives the outcome contextually to the request.	One-way. The device app receives a notification with the property change.	One-way. The device app receives the message
Durability	Disconnected devices are not contacted. The solution back end is notified that the device is not connected.	Property values are preserved in the device twin. Device will read it at next reconnection. Property values are retrievable with the IoT Hub query language.	Messages can be retained by IoT Hub for up to 48 hours.
Targets	Single device using deviceld, or multiple devices using jobs.	Single device using deviceld, or multiple devices using jobs.	Single device by deviceld.
Size	Maximum direct method payload size is 128 KB.	Maximum desired properties size is 8 KB.	Up to 64 KB messages.
Frequency	High. For more information, see IoT Hub limits.	Medium. For more information, see IoT Hub limits.	Low. For more information, see IoT Hub limits.
Protocol	Available using MQTT or AMQP.	Available using MQTT or AMQP.	Available on all protocols. Device must poll when using HTTPS.

## Visual Studio Code Extensions

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

The Azure IoT Tools collection includes the following extensions:

- Azure IoT Hub Toolkit
- Azure IoT Edge
- Azure IoT Device Workbench

## Installation

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

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

## Azure IoT Hub Toolkit

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

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

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

## Azure IoT Edge

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

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

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

## Azure IoT Device Workbench

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

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

## Azure CLI Tools

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

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

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

## Command-line Tools

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

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

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

- Azure Cloud Shell
- Windows Command Prompt app

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

## Install or run in Azure Cloud Shell

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

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

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

## Sign in

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

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

```
az login
```

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

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

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

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

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

## Extensions

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

### Find extensions

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

```
az extension list-available --output table
```

### Install extensions

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

```
az extension add --name <extension-name>
```

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

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

### Update extensions

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

```
az extension update --name <extension-name>
```

### Uninstall extensions

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

## Azure CLI Support for IoT Hub

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

### Azure CLI Commands for IoT Hub

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

### Hub Commands

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

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

## Command Usage

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

```
az iot hub <command name> --help
```

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

```
az iot hub create --help
```

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

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

Arguments
  --name -n                                [Required] : IoT Hub name.
  --resource-group -g                         [Required] : Name of resource group. You can configure
                                                        the default group
  using `az configure
  group=<name>`.

  --defaults
```

```

--c2d-max-delivery-count --cdd : The number of
times the IoT hub will attempt to deliver
a cloud-to-device message to a
device, between 1 and 100. Default: 10.

--c2d-ttl --ct : The amount of time for the device to
a message is available consume before it is expired by IoT
Hub, between 1 and 48 hours. Default: 1.

--fc --fileupload-storage-container-name : The name of the
root container where you upload files. The
container need not exist but should be
creatable using the connectionString
specified.

--fcs --fileupload-storage-connectionstring : The connection
string for the Azure Storage account to which
files are uploaded.

--fd --feedback-max-delivery-count : The number of
times the IoT hub attempts to deliver a message
on the feedback queue, between 1 and 100.
Default: 10.

--feedback-lock-duration --fld : The lock duration
for the feedback queue, between 5 and 300
seconds. Default: 5.

--feedback-ttl --ft : The period of time for which the IoT hub
will maintain the feedback for expiration or delivery of
cloud-to-device messages, between 1 and 48
hours. Default: 1.

--fileupload-notification-max-delivery-count --fnd : The number of times
the IoT hub will attempt to deliver
a file notification message, between 1
and 100. Default: 10.

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

```

```
file upload  
available for the service  
it is expired by IoT Hub,  
hours. Default: 1.  
--fileupload-notifications --fn  
whether to log  
uploaded files to the  
bound/filenotifications IoT  
Allowed values: false, true.  
--fileupload-sas-ttl --fst  
a SAS URI generated by  
before it expires, between  
Default: 1.  
--location -l  
IoT Hub. Default is the  
resource group.  
--partition-count  
partitions of the backing  
device-to-cloud messages.  
--rd --retention-day  
this IoT hub will  
to-cloud events, between 1  
fault: 1.  
--sku  
Azure IoT Hub. Default  
is free. Note that only  
instance is allowed in  
Exception will be thrown  
exceed one. Allowed  
B3, F1, S1, S2, S3.  
--unit  
Hub. Default: 1.
```

notification is  
to consume before  
between 1 and 48  
: A boolean indicating  
information about  
messages/service-  
Hub endpoint.

: The amount of time  
IoT Hub is valid  
1 and 24 hours.

: Location of your  
location of target

: The number of  
Event Hub for

Default: 2.  
: Specifies how long  
maintain device-  
and 7 days. De-

: Pricing tier for  
value is F1, which  
one free IoT hub  
each subscription.

if free instances  
values: B1, B2,  
Default: F1.  
: Units in your IoT

```

Global Arguments
  --debug                                : Increase logging
  verbosity to show all

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

  --output -o                            : Output format.
  Allowed values: json,
                                              jsonc, none,
  table, tsv, yaml. Default:
                                              json.

  --query                                : JMESPath query
  string. See
                                              http://jmespath.org/
  for more information

  --subscription                          : Name or ID of
  subscription. You can
                                              configure the
  default subscription using
                                              `az account set -s
NAME_OR_ID`.

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

```

#### Examples

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

```
az iot hub create --resource-group MyResourceGroup --name MyIoTHub
```

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

```
az iot hub create --resource-group MyResourceGroup --name MyIoTHub
--sku S1 --location
westus --partition-count 4
```

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

## IoT Hub Subgroups

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

IoT Hub Subgroups	Subgroup Description
certificate	Manage IoT Hub certificates.
configuration	Manage IoT device configurations at scale.

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

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

```
az iot hub <subgroup name> --help
```

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

```
az iot hub device-identity : Manage IoT devices.
```

Commands:

add-children	: Add specified comma-separated list of non edge device ids as children
create	: Create a device in an IoT Hub.
delete	: Delete an IoT Hub device.
export	: Export all device identities from an IoT Hub to an Azure Storage blob
get-parent	: Get the parent device of the specified device.
import	: Import device identities to an IoT Hub from a blob.
list	: List devices in an IoT Hub.
list-children	: Print comma-separated list of assigned child devices.
remove-children	: Remove non edge devices as children from specified edge device.
set-parent	: Set the parent device of the specified non-edge device.
show	: Get the details of an IoT Hub device.
show-connection-string	: Show a given IoT Hub device connection string.
update	: Update an IoT Hub device.

# Overview of Azure Cloud Shell

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

## Features

### Browser-based shell experience

Cloud Shell enables access to a browser-based command-line experience built with Azure management tasks in mind.

Leverage Cloud Shell to work untethered from a local machine in a way only the cloud can provide.

### Choice of preferred shell experience

Users can choose between Bash or PowerShell.

### Authenticated and configured Azure workstation

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

[View the full list of tools installed in Cloud Shell.](#)

### Integrated Cloud Shell editor

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

[Learn more about the Cloud Shell editor.](#)

### Integrated with docs.microsoft.com

You can use Cloud Shell directly from documentation hosted on [docs.microsoft.com](#)<sup>2</sup>. It is integrated in **Microsoft Learn**<sup>3</sup>, **Azure PowerShell**<sup>4</sup> and **Azure CLI documentation**<sup>5</sup> - click on the "Try It" button in a code snippet to open the immersive shell experience.

### Multiple access points

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

- [portal.azure.com](#)<sup>6</sup>

<sup>2</sup> <https://docs.microsoft.com>

<sup>3</sup> <https://docs.microsoft.com/learn/>

<sup>4</sup> <https://docs.microsoft.com/powershell/azure/overview>

<sup>5</sup> <https://docs.microsoft.com/cli/azure>

<sup>6</sup> <https://portal.azure.com>

- [shell.azure.com<sup>7</sup>](https://shell.azure.com)
- [Azure CLI documentation<sup>8</sup>](https://docs.microsoft.com/cli/azure)
- [Azure PowerShell documentation<sup>9</sup>](https://docs.microsoft.com/powershell/azure/overview)
- [Azure mobile app<sup>10</sup>](https://azure.microsoft.com/features/azure-portal/mobile-app/)
- [Visual Studio Code Azure Account extension<sup>11</sup>](https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account)

## Connect your Microsoft Azure Files storage

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

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

[!NOTE]

Azure storage firewall is not supported for cloud shell storage accounts.

## Concepts

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

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

## Pricing

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

<sup>7</sup> <https://shell.azure.com>

<sup>8</sup> <https://docs.microsoft.com/cli/azure>

<sup>9</sup> <https://docs.microsoft.com/powershell/azure/overview>

<sup>10</sup> <https://azure.microsoft.com/features/azure-portal/mobile-app/>

<sup>11</sup> <https://marketplace.visualstudio.com/items?itemName=ms-vscode.azure-account>

# Device Configuration and Communication

## Device Communication

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

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

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

## Device-to-Cloud Communications

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

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

## Cloud-to-Device Communications

IoT Hub provides three options for device apps to expose functionality to a back-end app:

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

## Communication Protocols

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

Protocol	When you should use this protocol
MQTT	Use on all devices that do not require to connect multiple devices (each with its own per-device credentials) over the same TLS connection.
MQTT over WebSocket	

Protocol	When you should use this protocol
AMQP	Use on field and cloud gateways to take advantage of connection multiplexing across devices.
AMQP over Websocket	Use for devices that cannot support other protocols.
HTTPS	Use for devices that cannot support other protocols.

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

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

## Port numbers

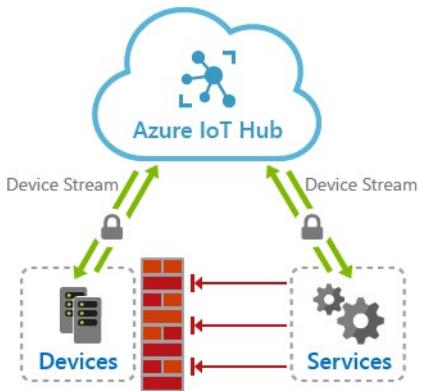
Devices can communicate with IoT Hub in Azure using various protocols. Typically, the choice of protocol is driven by the specific requirements of the solution. The following table lists the outbound ports that must be open for a device to be able to use a specific protocol:

Protocol	Port
MQTT	8883
MQTT over WebSockets	443
AMQP	5671
AMQP over WebSockets	443
HTTPS	443

## IoT Hub Device Streams (Preview)

Azure IoT Hub device streams facilitate the creation of secure bi-directional TCP tunnels for a variety of cloud-to-device communication scenarios. A device stream is mediated by an IoT Hub streaming endpoint which acts as a proxy between your device and service endpoints. This setup, depicted in the diagram below, is especially useful when devices are behind a network firewall or reside inside of a private network. As such, IoT Hub device streams help address customers' need to reach IoT devices in a

firewall-friendly manner and without the need to broadly opening up incoming or outgoing network firewall ports.



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

## Benefits

IoT Hub device streams provide the following benefits:

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

## Device stream workflows

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

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

## Connectivity Requirements

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

## Device-Side Code Implementation

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

The Sample Projects for C# include:

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

## MessageSample project

The MessageSample project includes the following code files:

- MessageSample.cs
- MessageSample.csproj
- Program.cs

## Examine Program.cs

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

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

```
// Copyright (c) Microsoft. All rights reserved.  
// Licensed under the MIT license. See LICENSE file in the project root for full license information.  
  
using System;  
  
namespace Microsoft.Azure.Devices.Client.Samples
```

```
{  
    public class Program  
    {  
        // String containing Hostname, Device Id & Device Key in one of the following formats:  
        // "HostName=<iothub_host_name>;DeviceId=<device_id>;SharedAccessKey=<device_key>"  
        // "HostName=<iothub_host_name>;CredentialType=SharedAccessSignature;DeviceId=<device_id>;SharedAccessSignature=SharedAccessSignature sr=<iot_host>/devices/<device_id>&sig=<token>&se=<expiry_time>";  
  
        // For this sample either  
        // - pass this value as a command-prompt argument  
        // - set the IOTHUB_DEVICE_CONN_STRING environment variable  
        // - create a launchSettings.json (see launchSettings.json.template) containing the variable  
  
        private static string s_deviceConnectionString = Environment.GetEnvironmentVariable("IOTHUB_DEVICE_CONN_STRING");  
  
        // Select one of the following transports used by DeviceClient to connect to IoT Hub.  
        private static TransportType s_transportType = TransportType.Amqp;  
        //private static TransportType s_transportType = TransportType.Mqtt;  
        //private static TransportType s_transportType = TransportType.Http1;  
        //private static TransportType s_transportType = TransportType.Amqp_WebSocket_Only;  
        //private static TransportType s_transportType = TransportType.Mqtt_WebSocket_Only;  
  
        public static int Main(string[] args)  
        {  
            if (string.IsNullOrEmpty(s_deviceConnectionString) && args.Length > 0)  
            {  
                s_deviceConnectionString = args[0];  
            }  
  
            DeviceClient deviceClient = DeviceClient.CreateFromConnectionString(s_deviceConnectionString,  
s_transportType);  
  
            if (deviceClient == null)  
            {  
                Console.WriteLine("Failed to create DeviceClient!");  
                return 1;  
            }  
  
            var sample = new MessageSample(deviceClient);  
            sample.RunSampleAsync().GetAwaiter().GetResult();  
  
            Console.WriteLine("Done.\n");  
  
            return 0;  
        }  
    }  
}
```

MCT USE ONLY. STUDENT USE PROHIBITED

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

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

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

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

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

## Examine the MessageSample.cs file

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

```
// Copyright (c) Microsoft. All rights reserved.  
// Licensed under the MIT license. See LICENSE file in the project root for full license information.
```

```
using System;  
using System.Text;  
using System.Threading.Tasks;  
  
namespace Microsoft.Azure.Devices.Client.Samples  
{  
    public class MessageSample  
    {  
        private const int MessageCount = 5;  
        private const int TemperatureThreshold = 30;  
        private static Random s_randomGenerator = new Random();  
        private float _temperature;  
        private float _humidity;  
        private DeviceClient _deviceClient;  
  
        public MessageSample(DeviceClient deviceClient)  
        {  
            _deviceClient = deviceClient ?? throw new ArgumentNullException(nameof(deviceClient));  
        }  
  
        public async Task RunSampleAsync()  
        {  
            await SendEvent().ConfigureAwait(false);  
            await ReceiveCommands().ConfigureAwait(false);  
        }  
  
        private async Task SendEvent()  
        {  
            string dataBuffer;
```

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

for (int count = 0; count < MessageCount; count++)
{
    _temperature = s_randomGenerator.Next(20, 35);
    _humidity = s_randomGenerator.Next(60, 80);
    dataBuffer = $"{{\"messageId\":{count},\"temperature\":\"{_temperature}\",\"humidity\":\"{_humidity}\",\"timeStamp\":{DateTime.Now.Ticks}}}";
}

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

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

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

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

}

private async Task ReceiveCommands()
{
    Console.WriteLine("\nDevice waiting for commands from IoTHub...\n");
    Console.WriteLine("Use the IoT Hub Azure Portal to send a message to this device.\n");

    Message receivedMessage;
    string messageData;

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

    if (receivedMessage != null)
    {
        messageData = Encoding.ASCII.GetString(receivedMessage.GetBytes());

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

        int propCount = 0;
        foreach (var prop in receivedMessage.Properties)
        {
            Console.WriteLine("\t\tProperty[{0}]> Key={1} : Value={2}", propCount++, prop.Key, prop.
Value);
        }

        await _deviceClient.CompleteAsync(receivedMessage).ConfigureAwait(false);
    }
    else
    {
        Console.WriteLine("\t{0}> Timed out", DateTime.NowToLocalTime());
    }
}
```

```
    }  
}  
}
```

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

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

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

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

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

# About the Module 2 Labs

## Module Labs

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

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

**WARNING:** Be prepared for UI changes

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

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



## Module 3 Device Provisioning at Scale

### Device Provisioning Service Terms and Concepts

#### Devices and Device Provisioning

The Device Provisioning Service supports the full lifecycle of a device, from the factory floor all the way through to retirement.

#### Provisioning Process

There are two distinct phases in the deployment process for a device. These two phases can be done independently:

- The manufacturing phase in which the device is created and prepared at the factory.
- The cloud setup phase in which the Device Provisioning Service is configured for automated provisioning.

Both these phases fit in seamlessly with existing manufacturing and deployment processes. The Device Provisioning Service even simplifies some deployment processes that involve a lot of manual work to get connection information onto the device.

#### Manufacturing Phase

This step is all about what happens on the manufacturing line, and is concerned with creating the hardware itself.

The Device Provisioning Service does not introduce a new step in the manufacturing process; rather, it ties into the existing step that installs the initial software and (ideally) the hardware security module (HSM) on the device. Instead of creating a device ID in this step, the device is programmed with the provisioning service information, enabling it to call the provisioning service to get its connection info/IoT solution assignment when it is switched on.

Also in this step, the manufacturer supplies the device deployer/operator with identifying key information. Supplying that information could be as simple as confirming that all devices have an X.509 certificate generated from a signing certificate provided by the device deployer/operator, or as complicated as extracting the public portion of a trusted platform module (TPM) endorsement key from each TPM device. These services are offered by many silicon manufacturers today.

## Cloud Setup Phase

This step is about configuring the cloud for proper automatic provisioning. Generally there are two types of users involved in the cloud setup step: someone who knows how devices need to be initially set up (a device operator), and someone else who knows how devices are to be split among the IoT hubs (a solution operator).

There is a one-time initial setup of the provisioning service that must occur, which is usually handled by the solution operator. Once the provisioning service is configured, it does not have to be modified unless the use case changes.

After the service has been configured for automatic provisioning, it must be prepared to enroll devices. This step is done by the device operator, who knows the desired configuration of the device(s) and is in charge of making sure the provisioning service can properly attest to the device's identity when it comes looking for its IoT hub. The device operator takes the identifying key information from the manufacturer and adds it to the enrollment list. There can be subsequent updates to the enrollment list as new entries are added or existing entries are updated with the latest information about the devices.

## Registration and Provisioning

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

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

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

## The Device Provisioning Service

Microsoft Azure provides a rich set of integrated public cloud services for all your IoT solution needs. The IoT Hub Device Provisioning Service is a helper service for IoT Hub that enables zero-touch, just-in-time provisioning to the right IoT hub without requiring human intervention, enabling customers to provision millions of devices in a secure and scalable manner.

## Features of the Device Provisioning Service

- Secure attestation support for both X.509 and TPM-based identities. We'll explain this more in a little bit.

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

## Cross-platform support

The Device Provisioning Service, like all Azure IoT services, works cross-platform with a variety of operating systems. Azure offers open-source SDKs in a variety of languages to facilitate connecting devices and managing the service. The Device Provisioning Service supports the following protocols for connecting devices:

- HTTPS
- AMQP
- AMQP over web sockets
- MQTT
- MQTT over web sockets

The Device Provisioning Service only supports HTTPS connections for service operations.

## When to use Device Provisioning Service

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

- Zero-touch provisioning to a single IoT solution without hardcoding IoT Hub connection information at the factory (initial setup)
- Load balancing devices across multiple hubs
- Connecting devices to their owner's IoT solution based on sales transaction data (multitenancy)
- Connecting devices to a particular IoT solution depending on use-case (solution isolation)
- Connecting a device to the IoT hub with the lowest latency (geo-sharding)
- Reprovisioning based on a change in the device
- Rolling the keys used by the device to connect to IoT Hub (when not using X.509 certificates to connect)

## Service Concepts

As we recognized earlier, device provisioning is a two part process. The first part is establishing the initial connection between the device and the IoT solution by registering the device. The second part is applying the proper configuration to the device based on the specific requirements of the solution. Once both

MCT USE ONLY. STUDENT USE PROHIBITED

steps have been completed, the device has been fully provisioned. Device Provisioning Service automates both steps to provide a seamless provisioning experience for the device.

## Service operations endpoint

The service operations endpoint is the endpoint for managing the service settings and maintaining the enrollment list. This endpoint is only used by the service administrator; it is not used by devices.

## Device provisioning endpoint

The device provisioning endpoint is the single endpoint all devices use for auto-provisioning. The URL is the same for all provisioning service instances, to eliminate the need to reflash devices with new connection information in supply chain scenarios. The ID scope ensures tenant isolation.

## Linked IoT hubs

The Device Provisioning Service can only provision devices to IoT hubs that have been linked to it. Linking an IoT hub to an instance of the Device Provisioning service gives the service read/write permissions to the IoT hub's device registry; with the link, a Device Provisioning service can register a device ID and set the initial configuration in the device twin. Linked IoT hubs may be in any Azure region. You may link hubs in other subscriptions to your provisioning service.

## Allocation policy

The service-level setting that determines how Device Provisioning Service assigns devices to an IoT hub. There are three supported allocation policies:

- Evenly weighted distribution: linked IoT hubs are equally likely to have devices provisioned to them. The default setting. If you are provisioning devices to only one IoT hub, you can keep this setting.
- Lowest latency: devices are provisioned to an IoT hub with the lowest latency to the device. If multiple linked IoT hubs would provide the same lowest latency, the provisioning service hashes devices across those hubs
- Static configuration via the enrollment list: specification of the desired IoT hub in the enrollment list takes priority over the service-level allocation policy.

## Enrollment

An enrollment is the record of devices or groups of devices that may register through auto-provisioning. The enrollment record contains information about the device or group of devices, including:

- the attestation mechanism used by the device
- the optional initial desired configuration
- desired IoT hub
- the desired device ID

There are two types of enrollments supported by Device Provisioning Service:

## Enrollment group

An enrollment group is a group of devices that share a specific attestation mechanism. Enrollment groups support both X.509 as well as symmetric keys. All devices in the X.509 enrollment group present X.509 certificates that have been signed by the same root or intermediate Certificate Authority (CA). Each device in the symmetric key enrollment group present SAS tokens derived from the group symmetric key. The enrollment group name and certificate name must be alphanumeric, lowercase, and may contain hyphens.

**Note:** We recommend using an enrollment group for a large number of devices that share a desired initial configuration, or for devices all going to the same tenant.

## Individual enrollment

An individual enrollment is an entry for a single device that may register. Individual enrollments may use either X.509 leaf certificates or SAS tokens (from a physical or virtual TPM) as attestation mechanisms. The registration ID in an individual enrollment is alphanumeric, lowercase, and may contain hyphens. Individual enrollments may have the desired IoT hub device ID specified.

**Note:** We recommend using individual enrollments for devices that require unique initial configurations, or for devices that can only authenticate using SAS tokens via TPM attestation.

## Registration

A registration is the record of a device successfully registering/provisioning to an IoT Hub via the Device Provisioning Service. Registration records are created automatically; they can be deleted, but they cannot be updated.

## Operations

Operations are the billing unit of the Device Provisioning Service. One operation is the successful completion of one instruction to the service. Operations include device registrations and re-registrations; operations also include service-side changes such as adding enrollment list entries, and updating enrollment list entries.

# Device Enrollment Concepts

This *Device Concepts* topic is most applicable to the manufacturing step of getting a device ready for deployment. In this step, the manufacturer is responsible for encoding the device identity info, and the Device Provisioning Service registration URL.

## Attestation mechanism

The attestation mechanism is the method used for confirming a device's identity. The attestation mechanism is also relevant to the enrollment list, which tells the provisioning service which method of attestation to use with a given device.

**Note:** IoT Hub uses "authentication scheme" for a similar concept in that service.

The Device Provisioning Service supports the following forms of attestation:

- X.509 certificates based on the standard X.509 certificate authentication flow.

- Trusted Platform Module (TPM) based on a nonce challenge, using the TPM standard for keys to present a signed Shared Access Signature (SAS) token. This does not require a physical TPM on the device, but the service expects to attest using the endorsement key per the TPM spec.
- Symmetric Key based on shared access signature (SAS) Security tokens, which include a hashed signature and an embedded expiration. For more information, see Symmetric key attestation.

Device secrets may also be stored in software (memory), but it is a less secure form of storage than an HSM.

## ID scope

The ID scope is assigned to a Device Provisioning Service when it is created by the user and is used to uniquely identify the specific provisioning service the device will register through. The ID scope is generated by the service and is immutable, which guarantees uniqueness.

**Note:** Uniqueness is important for long-running deployment operations and merger and acquisition scenarios.

## Registration ID

The registration ID is used to uniquely identify a device in the Device Provisioning Service. The registration ID must be unique in the provisioning service ID scope. Each device must have a registration ID. The registration ID is alphanumeric, case insensitive, and may contain special characters including colon, period, underscore and hyphen.

- In the case of TPM, the registration ID is provided by the TPM itself.
- In the case of X.509-based attestation, the registration ID is provided as the subject name of the certificate.

## Device ID

The device ID is the ID as it appears in IoT Hub. The desired device ID may be set in the enrollment entry, but it is not required to be set. Setting the desired device ID is only supported in individual enrollments. If no desired device ID is specified in the enrollment list, the registration ID is used as the device ID when registering the device.

## Device Enrollment

A device enrollment creates a record of a single device or a group of devices that may at some point register with the Azure IoT Hub Device Provisioning Service. The enrollment record contains the initial desired configuration for the device(s) as part of that enrollment, including the desired IoT hub.

## Enrollment Types

There are two ways you can enroll your devices with the provisioning service:

- Individual Enrollments: An Individual enrollment is an entry for a single device that may register. Individual enrollments may use either x509 certificates or SAS tokens (from a physical or virtual TPM) as attestation mechanisms. We recommend using individual enrollments for devices which require unique initial configurations, or for devices which can only use SAS tokens via TPM or virtual TPM as the attestation mechanism. Individual enrollments may have the desired IoT hub device ID specified.

- Group Enrollments: An Enrollment group is an entry for a group of devices that share a common attestation mechanism of X.509 certificates, signed by the same signing certificate, which can be the root certificate or the intermediate certificate, used to produce device certificate on physical device. Microsoft recommends using an enrollment group for a large number of devices which share a desired initial configuration, or for devices all going to the same tenant. Note that you can only enroll devices that use the X.509 attestation mechanism as enrollment groups.

## Security Concepts

Security concepts are relevant to all stages of device provisioning.

**Note:** Attestation Mechanism and Hardware Security Module topics have a direct relationship to security concepts. Refer to the Device Concepts section for related information.

## X.509 certificates

Using X.509 certificates as an attestation mechanism is an excellent way to scale production and simplify device provisioning. X.509 certificates are typically arranged in a certificate chain of trust in which each certificate in the chain is signed by the private key of the next higher certificate, and so on, terminating in a self-signed root certificate. This arrangement establishes a delegated chain of trust from the root certificate generated by a trusted root certificate authority (CA) down through each intermediate CA to the end-entity "leaf" certificate installed on a device.

Often the certificate chain represents some logical or physical hierarchy associated with devices. For example, a manufacturer may:

- issue a self-signed root CA certificate
- use the root certificate to generate a unique intermediate CA certificate for each factory
- use each factory's certificate to generate a unique intermediate CA certificate for each production line in the plant
- and finally use the production line certificate, to generate a unique device (end-entity) certificate for each device manufactured on the line.

## Root certificate

A root certificate is a self-signed X.509 certificate representing a certificate authority (CA). It is the terminus, or trust anchor, of the certificate chain. Root certificates can be self-issued by an organization or purchased from a root certificate authority. The root certificate can also be referred to as a root CA certificate.

## Intermediate certificate

An intermediate certificate is an X.509 certificate, which has been signed by the root certificate (or by another intermediate certificate with the root certificate in its chain). The last intermediate certificate in a chain is used to sign the leaf certificate. An intermediate certificate can also be referred to as an intermediate CA certificate.

## End-entity "leaf" certificate

The leaf certificate, or end-entity certificate, identifies the certificate holder. It has the root certificate in its certificate chain as well as zero or more intermediate certificates. The leaf certificate is not used to sign

MCT USE ONLY. STUDENT USE PROHIBITED

any other certificates. It uniquely identifies the device to the provisioning service and is sometimes referred to as the device certificate. During authentication, the device uses the private key associated with this certificate to respond to a proof of possession challenge from the service.

Leaf certificates used with an Individual enrollment entry have a requirement that the Subject Name must be set to the registration ID of the Individual Enrollment entry. Leaf certificates used with an Enrollment group entry should have the Subject Name set to the desired device ID which will be shown in the Registration Records for the authenticated device in the enrollment group.

## Controlling device access to the provisioning service with X.509 certificates

The provisioning service exposes two types of enrollment entry that you can use to control access for devices that use the X.509 attestation mechanism:

- Individual enrollment entries are configured with the device certificate associated with a specific device. These entries control enrollments for specific devices.
- Enrollment group entries are associated with a specific intermediate or root CA certificate. These entries control enrollments for all devices that have that intermediate or root certificate in their certificate chain.

When a device connects to the provisioning service, the service prioritizes more specific enrollment entries over less specific enrollment entries. That is, if an individual enrollment for the device exists, the provisioning service applies that entry. If there is no individual enrollment for the device and an enrollment group for the first intermediate certificate in the device's certificate chain exists, the service applies that entry, and so on, up the chain to the root. The service applies the first applicable entry that it finds, such that:

- If the first enrollment entry found is enabled, the service provisions the device.
- If the first enrollment entry found is disabled, the service does not provision the device.
- If no enrollment entry is found for any of the certificates in the device's certificate chain, the service does not provision the device.

This mechanism and the hierarchical structure of certificate chains provides powerful flexibility in how you can control access for individual devices as well as for groups of devices. For example, imagine five devices with the following certificate chains:

- Device 1: root certificate -> certificate A -> device 1 certificate
- Device 2: root certificate -> certificate A -> device 2 certificate
- Device 3: root certificate -> certificate A -> device 3 certificate
- Device 4: root certificate -> certificate B -> device 4 certificate
- Device 5: root certificate -> certificate B -> device 5 certificate

Initially, you can create a single enabled group enrollment entry for the root certificate to enable access for all five devices. If certificate B later becomes compromised, you can create a disabled enrollment group entry for certificate B to prevent Device 4 and Device 5 from enrolling. If still later Device 3 becomes compromised, you can create a disabled individual enrollment entry for its certificate. This revokes access for Device 3, but still allows Device 1 and Device 2 to enroll.

## Hardware security module

The hardware security module, or HSM, is used for secure, hardware-based storage of device secrets, and is the most secure form of secret storage. Both X.509 certificates and SAS tokens can be stored in the HSM. HSMs can be used with both attestation mechanisms the provisioning service supports.

**Note:** We strongly recommend using an HSM with devices to securely store secrets on your devices.

## Trusted Platform Module

TPM can refer to a standard for securely storing keys used to authenticate the platform, or it can refer to the I/O interface used to interact with the modules implementing the standard. TPMs can exist as discrete hardware, integrated hardware, firmware-based, or software-based. Device Provisioning Service only supports TPM 2.0.

TPM attestation is based on a nonce challenge, which uses the endorsement and storage root keys to present a signed Shared Access Signature (SAS) token.

## Endorsement key

The endorsement key is an asymmetric key contained inside the TPM, which was internally generated or injected at manufacturing time and is unique for every TPM. The endorsement key cannot be changed or removed. The private portion of the endorsement key is never released outside of the TPM, while the public portion of the endorsement key is used to recognize a genuine TPM.

## Storage root key

The storage root key is stored in the TPM and is used to protect TPM keys created by applications, so that these keys cannot be used without the TPM. The storage root key is generated when you take ownership of the TPM; when you clear the TPM so a new user can take ownership, a new storage root key is generated.

## TPM attestation

**Note:** Please be aware of the following:

- The information provided below assumes that you are using a discrete, firmware, or integrated TPM. Software emulated TPMs are well-suited for prototyping or testing, but they do not provide the same level of security as discrete, firmware, or integrated TPMs do. We do not recommend using software TPMs in production.
- The tasks described below are handled for you by the Device Provisioning Service device SDKs. If you are using the SDKs on your devices there is no need for you to implement anything additional.
- The information provided below is only relevant for devices using TPM 2.0 with HMAC key support and their endorsement keys. It is not for devices using X.509 certificates for authentication.

## Overview

TPMs use something called the endorsement key (EK) as the secure root of trust. The EK is unique to the TPM and changing it essentially changes the device into a new one.

There's another type of key that TPMs have, called the storage root key (SRK). An SRK may be generated by the TPM's owner after it takes ownership of the TPM. Taking ownership of the TPM is the TPM-specific

way of saying “someone sets a password on the HSM.” If a TPM device is sold to a new owner, the new owner can take ownership of the TPM to generate a new SRK. The new SRK generation ensures the previous owner can’t use the TPM. Because the SRK is unique to the owner of the TPM, the SRK can be used to seal data into the TPM itself for that owner. The SRK provides a sandbox for the owner to store their keys and provides access revocability if the device or TPM is sold. It’s like moving into a new house: taking ownership is changing the locks on the doors and destroying all furniture left by the previous owners (SRK), but you can’t change the address of the house (EK).

Once a device has been set up and ready to use, it will have both an EK and an SRK available for use.



One note on taking ownership of the TPM: Taking ownership of a TPM depends on many things, including TPM manufacturer, the set of TPM tools being used, and the device OS. Follow the instructions relevant to your system to take ownership.

The Device Provisioning Service uses the public part of the EK (EK\_pub) to identify and enroll devices. The device vendor can read the EK\_pub during manufacture or final testing and upload the EK\_pub to the provisioning service so that the device will be recognized when it connects to provision. The Device Provisioning Service does not check the SRK or owner, so “clearing” the TPM erases customer data, but the EK (and other vendor data) is preserved and the device will still be recognized by the Device Provisioning Service when it connects to provision.

## Detailed attestation process

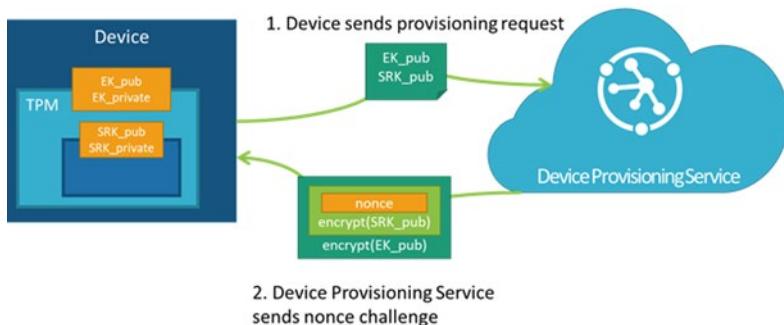
When a device with a TPM first connects to the Device Provisioning Service, the service first checks the provided EK\_pub against the EK\_pub stored in the enrollment list. If the EK\_pubs do not match, the device is not allowed to provision. If the EK\_pubs do match, the service then requires the device to prove ownership of the private portion of the EK via a nonce challenge, which is a secure challenge used to prove identity. The Device Provisioning Service generates a nonce and then encrypts it with the SRK and then the EK\_pub, both of which are provided by the device during the initial registration call. The TPM always keeps the private portion of the EK secure. This prevents counterfeiting and ensures SAS tokens are securely provisioned to authorized devices.

Let’s walk through the attestation process in detail.

## Device requests an IoT Hub assignment

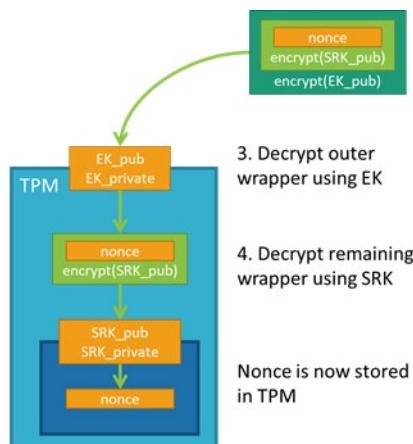
First the device connects to the Device Provisioning Service and requests to provision. In doing so, the device provides the service with its registration ID, an ID scope, and the EK\_pub and SRK\_pub from the

TPM. The service passes the encrypted nonce back to the device and asks the device to decrypt the nonce and use that to sign a SAS token to connect again and finish provisioning.



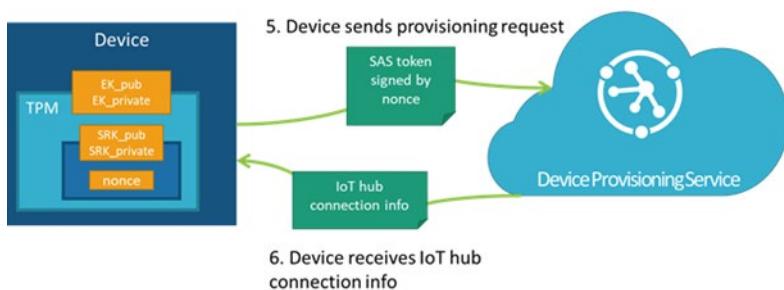
## Nonce challenge

The device takes the nonce and uses the private portions of the EK and SRK to decrypt the nonce into the TPM; the order of nonce encryption delegates trust from the EK, which is immutable, to the SRK, which can change if a new owner takes ownership of the TPM.



## Validate the nonce and receive credentials

The device can then sign a SAS token using the decrypted nonce and reestablish a connection to the Device Provisioning Service using the signed SAS token. With the Nonce challenge completed, the service allows the device to provision.



Now the device connects to IoT Hub, and you rest secure in the knowledge that your devices' keys are securely stored.

## Symmetric Key Attestation

Symmetric key attestation is a simple approach to authenticating a device with a Device Provisioning Service instance. This attestation method represents a "Hello world" experience for developers who are new to device provisioning, or do not have strict security requirements. Device attestation using a TPM or an X.509 certificate is more secure, and should be used for more stringent security requirements.

Symmetric key enrollments also provide a great way for legacy devices, with limited security functionality, to bootstrap to the cloud via Azure IoT.

### Symmetric key creation

By default, the Device Provisioning Service creates new symmetric keys with a default length of 32 bytes when new enrollments are saved with the Auto-generate keys option enabled.

You can also provide your own symmetric keys for enrollments by disabling this option. When specifying your own symmetric keys, your keys must have a key length between 16 bytes and 64 bytes. Also, symmetric keys must be provided in valid Base64 format.

### Detailed attestation process

Symmetric key attestation with the Device Provisioning Service is performed using the same Security tokens supported by IoT hubs to identify devices. These security tokens are Shared Access Signature (SAS) tokens.

SAS tokens have a hashed signature that is created using the symmetric key. The signature is recreated by the Device Provisioning Service to verify whether a security token presented during attestation is authentic or not.

SAS tokens have the following form:

```
SharedAccessSignature sig={signature}&se={expiry}&skn={policyName}&s-r={URL-encoded-resourceURI}
```

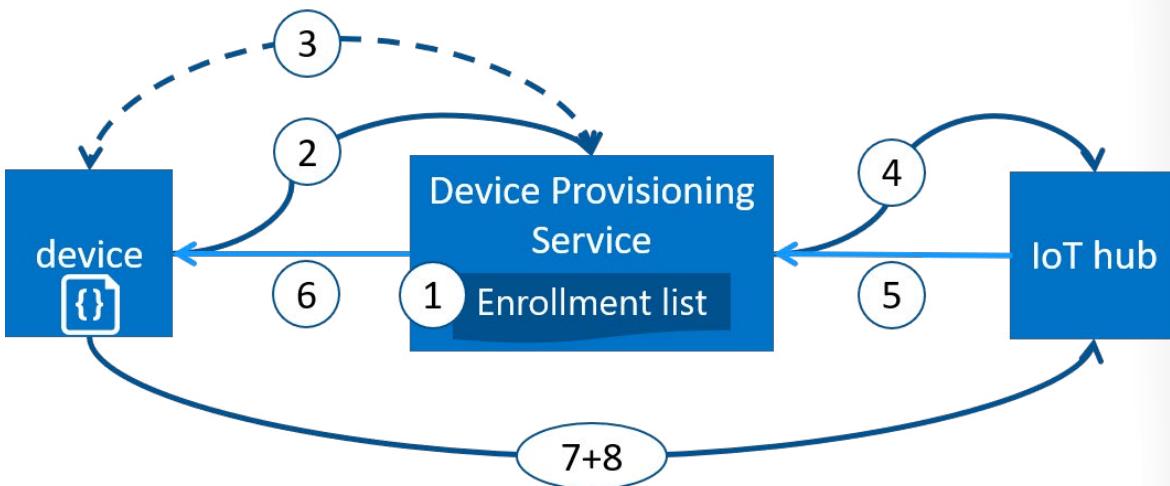
Here are the components of each token:

Value	Description
{signature}	An HMAC-SHA256 signature string. For individual enrollments, this signature is produced by using the symmetric key (primary or secondary) to perform the hash. For enrollment groups, a key derived from the enrollment group key is used to perform the hash. The hash is performed on a message of the form: URL-encoded-resourceURI + "\n" + expiry. <b>Important:</b> The key must be decoded from base64 before being used to perform the HMAC-SHA256 computation. Also, the signature result must be URL-encoded.

Value	Description
{resourceURI}	URI of the registration endpoint that can be accessed with this token, starting with scope ID for the Device Provisioning Service instance. For example, {Scope ID}/registrations/{Registration ID}
{expiry}	UTF8 strings for number of seconds since the epoch 00:00:00 UTC on 1 January 1970.
{URL-encoded-resourceURI}	Lower case URL-encoding of the lower case resource URI
{policyName}	The name of the shared access policy to which this token refers. The policy name used when provisioning with symmetric key attestation is <b>registration</b> .

## How DPS Works Behind the Scenes

All the scenarios listed above can be done using the provisioning service for zero-touch provisioning with the same flow. Many of the manual steps traditionally involved in provisioning are automated with the Device Provisioning Service to reduce the time to deploy IoT devices and lower the risk of manual error. The following section describes what goes on behind the scenes to get a device provisioned. The first step is manual, all of the following steps are automated.



1. Device manufacturer adds the device registration information to the enrollment list in the Azure portal.
2. Device contacts the provisioning service endpoint set at the factory. The device passes the identifying information to the provisioning service to prove its identity.
3. The provisioning service validates the identity of the device by validating the registration ID and key against the enrollment list entry using either a nonce challenge (Trusted Platform Module) or standard X.509 verification (X.509).
4. The provisioning service registers the device with an IoT hub and populates the device's desired twin state.
5. The IoT hub returns device ID information to the provisioning service.

6. The provisioning service returns the IoT hub connection information to the device. The device can now start sending data directly to the IoT hub.
7. The device connects to IoT hub.
8. The device gets the desired state from its device twin in IoT hub.

## Auto-Provisioning

The Device Provisioning Service enables just-in-time provisioning of devices to an IoT hub, without requiring human intervention. After successful provisioning, devices connect directly with their designated IoT Hub. This process is referred to as auto-provisioning, and provides an out-of-the-box registration and initial configuration experience for devices.

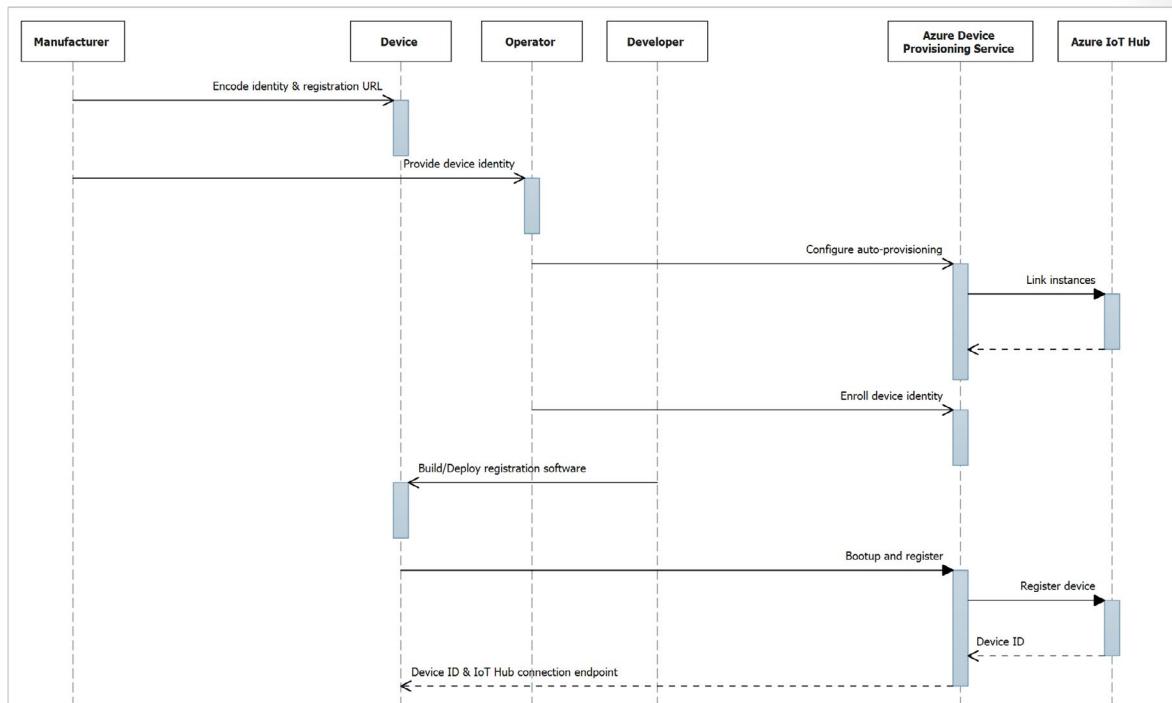
### Three Phases of Auto-Provisioning

Azure IoT auto-provisioning can be broken into three phases:

1. Service configuration - a one-time configuration of the Azure IoT Hub and IoT Hub Device Provisioning Service instances, establishing them and creating linkage between them.
2. Device enrollment - the process of making the Device Provisioning Service instance aware of the devices that will attempt to register in the future. Enrollment is accomplished by configuring device identity information in the provisioning service, as either an "individual enrollment" for a single device, or a "group enrollment" for multiple devices. Identity is based on the attestation mechanism the device is designed to use, which allows the provisioning service to attest to the device's authenticity during registration:
  - Trusted Platform Module (TPM): configured as an "individual enrollment", the device identity is based on the TPM registration ID and the public endorsement key. Given that TPM is a specification, the service only expects to attest per the specification, regardless of TPM implementation (hardware or software). TPM is a type of hardware security module (HSM).
  - X509: configured as either an "individual enrollment" or "group enrollment", the device identity is based on an X.509 digital certificate, which is uploaded to the enrollment as a .pem or .cer file.
3. Device registration and configuration - initiated upon boot up by registration software, which is built using a Device Provisioning Service client SDK appropriate for the device and attestation mechanism. The software establishes a connection to the provisioning service for authentication of the device, and subsequent registration in the IoT Hub. Upon successful registration, the device is provided with its IoT Hub unique device ID and connection information, allowing it to pull its initial configuration and begin the telemetry process. In production environments, this phase can occur weeks or months after the previous two phases.

### Auto-provisioning Operation

The following diagram summarizes the sequencing of operations during device auto-provisioning:



Operation	Description
Encode identity and registration URL	Based on the attestation mechanism used, the manufacturer is responsible for encoding the device identity info, and the Device Provisioning Service registration URL.
Provide device identity	As the originator of the device identity info, the manufacturer is responsible for communicating it to the operator (or a designated agent), or directly enrolling it to the Device Provisioning Service via APIs.
Configure auto-provisioning	This operation corresponds with the first phase of auto-provisioning.
Enroll device identity	This operation corresponds with the second phase of auto-provisioning.
Build/Deploy registration software	This operation corresponds with the third phase of auto-provisioning. The Developer is responsible for building and deploying the registration software to the device, using the appropriate SDK.
Bootup and register	This operation corresponds with the third phase of auto-provisioning, fulfilled by the device registration software built by the Developer.

MCT USE ONLY. STUDENT USE PROHIBITED

# Reprovisioning

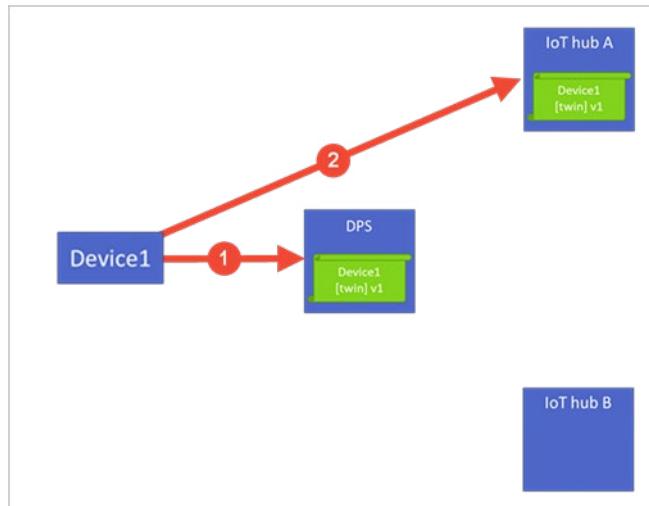
During the lifecycle of an IoT solution, it's common to move devices between IoT hubs. The reasons for this move may include the following scenarios:

- Geolocation / GeoLatency: As a device moves between locations, network latency is improved by having the device migrated to a closer IoT hub.
- Multi-tenancy: A device may be used within the same IoT solution and reassigned to a new customer, or customer site. This new customer may be serviced using a different IoT hub.
- Solution change: A device could be moved into a new or updated IoT solution. This reassignment may require the device to communicate with a new IoT hub that's connected to other back-end components.
- Quarantine: Similar to a solution change. A device that's malfunctioning, compromised, or out-of-date may be reassigned to an IoT hub that can only update and get back in compliance. Once the device is functioning properly, it's then migrated back to its main hub.

Reprovisioning support within the Device Provisioning Service addresses these needs. Devices can be automatically reassigned to new IoT hubs based on the reprovisioning policy that's configured on the device's enrollment entry.

## Device state data

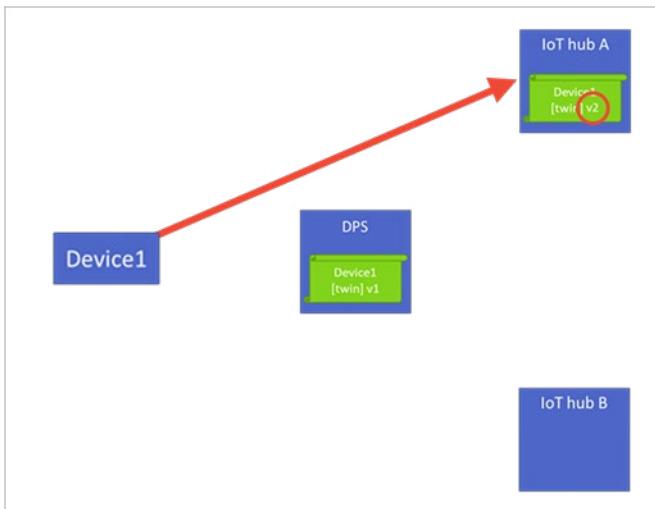
Device state data is composed of the device twin and device capabilities. This data is stored in the Device Provisioning Service instance and the IoT hub that a device is assigned to.



When a device is initially provisioned with a Device Provisioning Service instance, the following steps are done:

1. The device sends a provisioning request to a Device Provisioning Service instance. The service instance authenticates the device identity based on an enrollment entry, and creates the initial configuration of the device state data. The service instance assigns the device to an IoT hub based on the enrollment configuration and returns that IoT hub assignment to the device.
2. The provisioning service instance gives a copy of any initial device state data to the assigned IoT hub. The device connects to the assigned IoT hub and begins operations.

Over time, the device state data on the IoT hub may be updated by device operations and back-end operations. The initial device state information stored in the Device Provisioning Service instance stays untouched. This untouched device state data is the initial configuration.

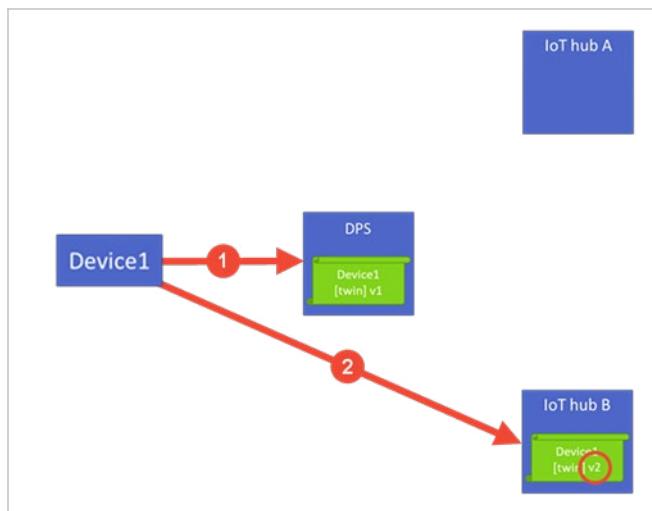


Depending on the scenario, as a device moves between IoT hubs, it may also be necessary to migrate device state updated on the previous IoT hub over to the new IoT hub. This migration is supported by reprovisioning policies in the Device Provisioning Service.

## Reprovisioning policies

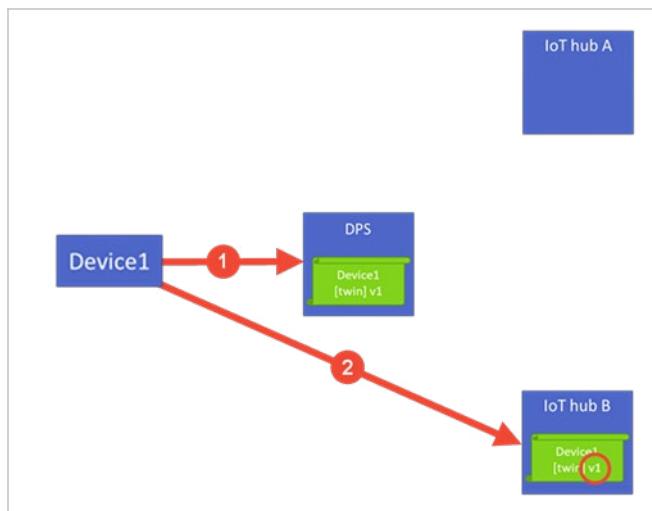
Depending on the scenario, a device usually sends a request to a provisioning service instance on reboot. It also supports a method to manually trigger provisioning on demand. The reprovisioning policy on an enrollment entry determines how the device provisioning service instance handles these provisioning requests. The policy also determines whether device state data should be migrated during reprovisioning. The same policies are available for individual enrollments and enrollment groups:

- Re-provision and migrate data: This policy is the default for new enrollment entries. This policy takes action when devices associated with the enrollment entry submit a new request (1). Depending on the enrollment entry configuration, the device may be reassigned to another IoT hub. If the device is changing IoT hubs, the device registration with the initial IoT hub will be removed. The updated device state information from that initial IoT hub will be migrated over to the new IoT hub (2). During migration, the device's status will be reported as Assigning.



- Re-provision and reset to initial config: This policy takes action when devices associated with the enrollment entry submit a new provisioning request (1). Depending on the enrollment entry configuration, the device may be reassigned to another IoT hub. If the device is changing IoT hubs, the device registration with the initial IoT hub will be removed. The initial configuration data that the provisioning service instance received when the device was provisioned is provided to the new IoT hub (2). During migration, the device's status will be reported as Assigning.

This policy is often used for a factory reset without changing IoT hubs.



- Never re-provision: The device is never reassigned to a different hub. This policy is provided for managing backwards compatibility.

## Managing backwards compatibility

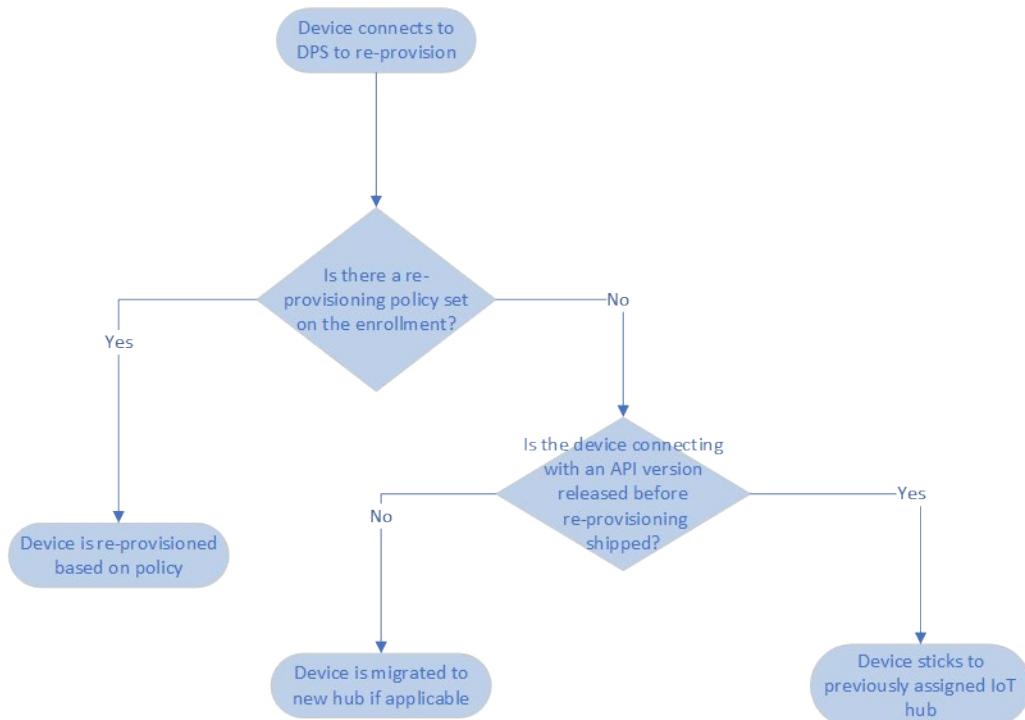
Before September 2018, device assignments to IoT hubs had a sticky behavior. When a device went back through the provisioning process, it would only be assigned back to the same IoT hub.

For solutions that have taken a dependency on this behavior, the provisioning service includes backwards compatibility. This behavior is presently maintained for devices according to the following criteria:

1. The devices connect with an API version before the availability of native reprovisioning support in the Device Provisioning Service. Refer to the API table below.
2. The enrollment entry for the devices doesn't have a reprovisioning policy set on them.

This compatibility makes sure that previously deployed devices experience the same behavior that's present during initial testing. To preserve the previous behavior, don't save a reprovisioning policy to these enrollments. If a reprovisioning policy is set, the reprovisioning policy takes precedence over the behavior. By allowing the reprovisioning policy to take precedence, customers can update device behavior without having to reimagine the device.

The following flow chart helps to show when the behavior is present:



The following table shows the API versions before the availability of native reprovisioning support in the Device Provisioning Service:

REST API	C SDK	Python SDK	Node SDK	Java SDK	.NET SDK
2018-04-01 and earlier	1.2.8 and earlier	1.4.2 and earlier	1.7.3 or earlier	1.13.0 or earlier	1.1.0 or earlier

**Note:** These values and links are likely to change. This is only a placeholder attempt to determine where the versions can be determined by a customer and what the expected versions will be.

# Configure and Manage the Device Provisioning Service

## Azure CLI Support for Device Provisioning

Azure CLI commands can be used to accomplish many of the tasks associated with the Azure IoT Hub Device Provisioning Service.

### Azure CLI Commands for DPS

Azure CLI Commands for DPS are available in the following categories:

- DPS Service
- Access Policy
- Certificates
- Linked Hub

The commands available in each of these categories are shown in the tables below.

Service Commands	Description
az iot dps create	Create an Azure IoT Hub device provisioning service.
az iot dps delete	Delete an Azure IoT Hub device provisioning service.
az iot dps list	List Azure IoT Hub device provisioning services.
az iot dps show	Get the details of an Azure IoT Hub device provisioning service.
az iot dps update	Update an Azure IoT Hub device provisioning service.

Access Policy Commands	Description
az iot dps access-policy	Manage Azure IoT Hub Device Provisioning Service access policies.
az iot dps access-policy create	Create a new shared access policy in an Azure IoT Hub device provisioning service.
az iot dps access-policy delete	Delete a shared access policies in an Azure IoT Hub device provisioning service.
az iot dps access-policy list	List all shared access policies in an Azure IoT Hub device provisioning service.
az iot dps access-policy show	Show details of a shared access policies in an Azure IoT Hub device provisioning service.
az iot dps access-policy update	Update a shared access policy in an Azure IoT Hub device provisioning service.

Certificate Commands	Description
az iot dps certificate	Manage Azure IoT Hub Device Provisioning Service certificates.

Certificate Commands	Description
az iot dps certificate create	Create/upload an Azure IoT Hub Device Provisioning Service certificate.
az iot dps certificate delete	Delete an Azure IoT Hub Device Provisioning Service certificate.
az iot dps certificate generate-verification-code	Generate a verification code for an Azure IoT Hub Device Provisioning Service certificate.
az iot dps certificate list	List all certificates contained within an Azure IoT Hub device provisioning service.
az iot dps certificate show	Show information about a particular Azure IoT Hub Device Provisioning Service certificate.
az iot dps certificate update	Update an Azure IoT Hub Device Provisioning Service certificate.
az iot dps certificate verify	Verify an Azure IoT Hub Device Provisioning Service certificate.

Linked Hub Commands	Description
az iot dps linked-hub	Manage Azure IoT Hub Device Provisioning Service linked IoT hubs.
az iot dps linked-hub create	Create a linked IoT hub in an Azure IoT Hub device provisioning service.
az iot dps linked-hub delete	Update a linked IoT hub in an Azure IoT Hub device provisioning service.
az iot dps linked-hub list	List all linked IoT hubs in an Azure IoT Hub device provisioning service.
az iot dps linked-hub show	Show details of a linked IoT hub in an Azure IoT Hub device provisioning service.
az iot dps linked-hub update	Update a linked IoT hub in an Azure IoT Hub device provisioning service.

## Using the DPS Service Commands

As noted previously, Azure CLI commands can be used to manage DPS at the service level.

**Note:** To view usage information for any Azure CLI command, enter the command followed by `--help`

### Create Command

The `az iot dps create` command can be used to create an Azure IoT Hub device provisioning service.

This command takes the following arguments:

command arguments	description
<code>-name</code>	IoT Provisioning Service name. Note: <code>-name</code> is a required argument.
<code>-resource-group</code>	Name of resource group. Note: <code>-resource-group</code> is a required argument.

command arguments	description
-location	Location of your IoT Provisioning Service. Default is the location of target resource group.
-sku	Pricing tier for the IoT provisioning service. Allowed values: S1. Default: S1.
-unit	Units in your IoT Provisioning Service. Default: 1.

For example, the following command can be used to create an Azure IoT Hub device provisioning service with the standard pricing tier S1, in the region of the resource group:

```
az iot dps create --name MyDps --resource-group MyResourceGroup
```

Or, to create an Azure IoT Hub device provisioning service with the standard pricing tier S1, in the 'eastus' region, use the following command:

```
az iot dps create --name MyDps --resource-group MyResourceGroup --location eastus
```

## Delete Command

The `az iot dps delete` command can be used to delete an Azure IoT Hub device provisioning service.

This command takes the following arguments:

command arguments	description
-ids	One or more resource IDs (space-delimited). If provided, no other 'Resource Id' arguments should be specified.
-name	IoT Provisioning Service name.
-resource-group	Name of resource group.

For example, the following command can be used to delete an Azure IoT Hub device provisioning service named 'MyDps':

```
az iot dps delete --name MyDps --resource-group MyResourceGroup
```

Or, to delete an Azure IoT Hub device provisioning service 'MyDps', use the following command:

```
az iot dps create --name MyDps --resource-group MyResourceGroup --location eastus
```

## Device Provisioning Service SDKs

A set of Azure IoT Provisioning Service SDKs are provided so that developers can use a convenience layer for writing clients that talk to the provisioning service. The SDKs also provide samples for common scenarios as well as a set of tools to simplify security attestation in development. The DPS SDKs provide support for managing the provisioning service as well as for managing device provisioning.

## Device Provisioning Service SDK Overview

The **Device Provisioning Service SDKs**<sup>1</sup> support the following dev languages:

- C#

---

<sup>1</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-sdks>

- C
- Java
- Node.js
- Python

The Azure Provisioning device and service SDKs for C# can be downloaded from NuGet as follows:

- **Provisioning Device Client SDK<sup>2</sup>**
- **Provisioning Service Client SDK<sup>3</sup>**

## Features of the Provisioning Device SDK

The Provisioning Device SDK supports the following protocols: MQTT, MQTT-WS, AMQP, AMQP-WS, and HTTPS.

Features	Description
TPM Individual Enrollment	This SDK supports connecting your device to the Device Provisioning Service via individual enrollment using Trusted Platform Module. TPM over MQTT (and MQTT-WS) is currently not supported by the Device Provisioning Service.
X.509 Individual Enrollment	This SDK supports connecting your device to the Device Provisioning Service via individual enrollment using X.509 root certificate.
X.509 Enrollment Group	This SDK supports connecting your device to the Device Provisioning Service via individual enrollment using X.509 leaf certificate.

**Note:** WebSocket support for MQTT/AMQP is limited to .NET Framework 4.x.

## Features of the Provisioning Service SDK

The Provisioning Service SDK can be used to programmatically enroll devices.

Feature	Description
CRUD Operation with TPM Individual Enrollment	Programmatically manage device enrollment using TPM with the service SDK.
Bulk CRUD Operation with TPM Individual Enrollment	Programmatically bulk manage device enrollment using TPM with the service SDK.
CRUD Operation with X.509 Individual Enrollment	Programmatically manage device enrollment using X.509 individual enrollment with the service SDK.
CRUD Operation with X.509 Group Enrollment	Programmatically manage device enrollment using X.509 group enrollment with the service SDK.
Query enrollments	Programmatically query registration states with the service SDK.

<sup>2</sup> <https://www.nuget.org/packages/Microsoft.Azure.Devices.Provisioning.Client/>

<sup>3</sup> <https://www.nuget.org/packages/Microsoft.Azure.Devices.Provisioning.Service/>

## Additional Tool Support

In addition to the sample code included with the SDKs that can be used for reference, Microsoft provides some additional tooling that may be useful during testing.

## Trusted Platform Module (TPM) simulator

TPM can refer to a standard for securely storing keys to authenticate the platform, or it can refer to the I/O interface used to interact with the modules implementing the standard. TPMs can exist as discrete hardware, integrated hardware, firmware-based, or software-based. In production, TPM is located on the device, either as discrete hardware, integrated hardware, or firmware-based. In the testing phase, a software-based TPM simulator is provided to developers.

**Note:** This simulator is only available for developing on Windows platform for now.

Steps for using the TPM simulator are:

1. Prepare the development environment and clone the GitHub repository:  

```
git clone https://github.com/Azure/azure-iot-sdk-java.git
```
2. Navigate to the TPM simulator folder under `azure-iot-sdk-java/provisioning/provisioning-tool/tpm-simulator/`.
3. Run `Simulator.exe` prior to running any client application for provisioning device.
4. Let the simulator run in the background throughout the provisioning process to obtain registration ID and Endorsement Key. Both values are only valid for one instance of the run.

## X.509 certificate generator

X.509 certificates can be used as an attestation mechanism to scale production and simplify device provisioning. There are several ways to obtain an X.509 certificate:

- For production environment, we recommend purchasing an X.509 CA certificate from a public root certificate authority.
- For testing environment, you can generate an X.509 root certificate or X.509 certificate chain using:
  - OpenSSL: You can use scripts for certificate generation:
    - Node.js
    - PowerShell or Bash
  - Device Identity Composition Engine (DICE) Emulator: DICE can be used for cryptographic device identity and attestation based on TLS protocol and X.509 client certificates. [Learn<sup>4</sup>](#) more about device identity with DICE.

## Using X.509 certificate generator with DICE emulator

The SDKs provide an X.509 certificate generator with DICE emulator, located in the Java SDK. This generator works cross-platform. The generated certificate can be used for development in other languages.

Currently, while the DICE Emulator outputs a root certificate, an intermediate certificate, a leaf certificate, and associated private key. However, the root certificate or intermediate certificate cannot be used to sign a separate leaf certificate. If you intend to test group enrollment scenario where one signing certifi-

<sup>4</sup> <https://www.microsoft.com/en-us/research/publication/device-identity-dice-riot-keys-certificates/>

cate is used to sign the leaf certificates of multiple devices, you can use OpenSSL to produce a chain of certificates.

To generate X.509 certificate using this generator:

1. Prepare the development environment and clone the GitHub repository:

```
git clone https://github.com/Azure/azure-iot-sdk-java.git
```

2. Change the root to azure-iot-sdk-java.

3. Run `mvn install -DskipTests=true` to download all required packages and compile the SDK

4. Navigate to the root for X.509 Certificate Generator in `azure-iot-sdk-java/provisioning/provisioning-tools/provisioning-x509-cert-generator`.

5. Build with `mvn clean install`

6. Run the tool using the following commands:

```
cd target
```

```
java -jar ./provisioning-x509-cert-generator-{version}-with-deps.jar
```

7. When prompted, you may optionally enter a Common Name for your certificates.

8. The tool locally generates a Client Cert, the Client Cert Private Key, Intermediate Cert, and the Root Cert.

Client Cert is the leaf certificate on the device. Client Cert and the associated Client Cert Private Key are needed in device client. Depending on what language you choose, the mechanism to put this in the client application may be different.

The root certificate or intermediate can be used to create an enrollment group or individual enrollment programmatically or using the portal.

## Control Access to DPS

The provisioning service uses permissions to grant access to each endpoint. Permissions limit the access to a service instance based on functionality. For example, a backend app must include a token containing security credentials along with every message it sends to the service.

### Access control and permissions

You can grant permissions in the following ways:

- Shared access authorization policies. Shared access policies can grant any combination of permissions. You can define policies in the Azure portal, or programmatically by using the Device Provisioning Service REST APIs. A newly created provisioning service has the following default policy:
  - `provisioningserviceowner`: Policy with all permissions.

### Authentication

Azure IoT Hub Device Provisioning Service grants access to endpoints by verifying a token against the shared access policies. Security credentials, such as symmetric keys, are never sent over the wire.

## Security tokens

The Device Provisioning Service uses security tokens to authenticate services to avoid sending keys on the wire. Additionally, security tokens are limited in time validity and scope. Azure IoT Device Provisioning Service SDKs automatically generate tokens without requiring any special configuration. Some scenarios do require you to generate and use security tokens directly. Such scenarios include the direct use of the HTTP surface.

## Security token structure

You use security tokens to grant time-bounded access for services to specific functionality in IoT Device Provisioning Service. To get authorization to connect to the provisioning service, services must send security tokens signed with either a shared access or symmetric key.

A token signed with a shared access key grants access to all the functionality associated with the shared access policy permissions.

The security token has the following format:

```
SharedAccessSignature sig={signature}&se={expiry}&skn={policyName}&s-
r={URL-encoded-resourceURI}
```

Here are the expected values:

Value	Description
{signature}	An HMAC-SHA256 signature string of the form: {URL-encoded-resourceURI} + "\n" + expiry. <b>Important:</b> The key is decoded from base64 and used as key to perform the HMAC-SHA256 computation.
{expiry}	UTF8 strings for number of seconds since the epoch 00:00:00 UTC on 1 January 1970.
{URL-encoded-resourceURI}	Lower case URL-encoding of the lower case resource URI. URI prefix (by segment) of the endpoints that can be accessed with this token, starting with host name of the IoT Device Provisioning Service (no protocol). For example, mydps.azure-devices-provisioning.net.
{policyName}	The name of the shared access policy to which this token refers.

**Note on prefix:** The URI prefix is computed by segment and not by character. For example /a/b is a prefix for /a/b/c but not for /a/bc.

The following Node.js snippet shows a function called **generateSasToken** that computes the token from the inputs `resourceUri`, `signingKey`, `policyName`, `expiresInMins`. The next sections detail how to initialize the different inputs for the different token use cases.

```
var generateSasToken = function(resourceUri, signingKey, policyName, expiresInMins) {
    resourceUri = encodeURIComponent(resourceUri);

    // Set expiration in seconds
    var expires = (Date.now() / 1000) + expiresInMins * 60;
    expires = Math.ceil(expires);
    var toSign = resourceUri + '\n' + expires;
```

```
// Use crypto
var hmac = crypto.createHmac('sha256', new Buffer(signingKey, 'base64'));
hmac.update(toSign);
var base64UriEncoded = encodeURIComponent(hmac.digest('base64'));

// Construct authorization string
var token = "SharedAccessSignature sr=" + resourceUri + "&sig="
+ base64UriEncoded + "&se=" + expires + "&skn=" + policyName;
return token;
};
```

As a comparison, the equivalent Python code to generate a security token is:

```
from base64 import b64encode, b64decode
from hashlib import sha256
from time import time
from urllib import quote_plus, urlencode
from hmac import HMAC

def generate_sas_token(uri, key, policy_name, expiry=3600):
    ttl = time() + expiry
    sign_key = "%s\n%d" % ((quote_plus(uri)), int(ttl))
    print sign_key
    signature = b64encode(HMAC(b64decode(key), sign_key, sha256).digest())

    rawtoken = {
        'sr': uri,
        'sig': signature,
        'se': str(int(ttl)),
        'skn': policy_name
    }

    return 'SharedAccessSignature ' + urlencode(rawtoken)
```

## Use security tokens from service components

Service components can only generate security tokens using shared access policies granting the appropriate permissions as explained previously.

Here are the service functions exposed on the endpoints:

Endpoint	Functionality
{your-service}.azure-devices-provisioning.net/enrollments	Provides device enrollment operations with the Device Provisioning Service.
{your-service}.azure-devices-provisioning.net/enrollmentGroups	Provides operations for managing device enrollment groups.
{your-service}.azure-devices-provisioning.net/registrations/{id}	Provides operations for retrieving and managing the status of device registrations.

As an example, a service generated using a pre-created shared access policy called **enrollmentread** would create a token with the following parameters:

- resource URL: {mydps}.azure-devices-provisioning.net,
- signing key: one of the keys of the enrollmentread policy,
- policy name: enrollmentread,
- an expiration time

```
var endpoint = "mydps.azure-devices-provisioning.net";
var policyName = 'enrollmentread';
var policyKey = '...';

var token = generateSasToken(endpoint, policyKey, policyName, 60);
```

The result, which would grant access to read all enrollment records, would be:

```
SharedAccessSignature sr=mydps.azure-devices-
provisioning.net&sig=JdyscqTpXdEJs49elIUCcohw2D1FDR3zfH5KqG-
Jo4r4%3D&se=1456973447&skn=enrollmentread
```

## Device Provisioning Service permissions

The following table lists the permissions you can use to control access to your IoT Device Provisioning Service.

Permission	Notes
ServiceConfig	Grants access to change the service configurations. This permission is used by backend cloud services.
EnrollmentRead	Grants read access to the device enrollments and enrollment groups. This permission is used by backend cloud services.
EnrollmentWrite	Grants write access to the device enrollments and enrollment groups. This permission is used by backend cloud services.
RegistrationStatusRead	Grants read access to the device registration status. This permission is used by backend cloud services.
RegistrationStatusWrite	Grants delete access to the device registration status. This permission is used by backend cloud services.

# Device Provisioning Tasks

## Device Enrollment Tools and Processes

The tool that you choose for a particular device enrollment scenario, and the corresponding implementation steps, will vary based on situational requirements, but the processes are discrete and defined.

### Device Enrollment Processes

There are three enrollment process that you will need to perform on a semi-regular basis:

- Create: Create an enrollment when you need to prepare a device that you want to have ready to register with the Azure IoT Hub Device Provisioning Service. The enrollment record will contain the initial desired configuration for the device.
- Update: Update a device enrollment if you want to change the IoT Hub that the device should be linked to, the device ID, or the initial device twin state for the device.
- Remove: Remove enrollments when an enrolled device will not be provisioned to an IoT Hub.

### Device Enrollment Tools

There are two primary tools that you should be familiar with for performing device enrollments:

- Device Provisioning Service in the portal
- Azure CLI with the Service SDKs

## Configure Verified CA Certificates

A verified X.509 Certificate Authority (CA) certificate is a CA certificate that has been uploaded and registered to your provisioning service and has gone through proof-of-possession with the service.

Proof-of-possession involves the following steps:

1. Get a unique verification code generated by the provisioning service for your X.509 CA certificate. You can do this from the Azure portal.
2. Create an X.509 verification certificate with the verification code as its subject and sign the certificate with the private key associated with your X.509 CA certificate.
3. Upload the signed verification certificate to the service. The service validates the verification certificate using the public portion of the CA certificate to be verified, thus proving that you are in possession of the CA certificate's private key.

Verified certificates play an important role when using enrollment groups. Verifying certificate ownership provides an additional security layer by ensuring that the uploader of the certificate is in possession of the certificate's private key. Verification prevents a malicious actor sniffing your traffic from extracting an intermediate certificate and using that certificate to create an enrollment group in their own provisioning service, effectively hijacking your devices. By proving ownership of the root or an intermediate certificate in a certificate chain, you're proving that you have permission to generate leaf certificates for the devices that will be registering as a part of that enrollment group. For this reason, the root or intermediate certificate configured in an enrollment group must either be a verified certificate or must roll up to a verified certificate in the certificate chain a device presents when it authenticates with the service.

## Register the public part of an X.509 certificate and get a verification code

To register a CA certificate with your provisioning service and get a verification code that you can use during proof-of-possession, follow these steps:

- In the Azure portal, navigate to your provisioning service and open Certificates from the left-hand menu.
- Click Add to add a new certificate.
- Enter a friendly display name for your certificate. Browse to the .cer or .pem file that represents the public part of your X.509 certificate. Click Upload.
- Once you get a notification that your certificate is successfully uploaded, click Save.
- Click on the certificate that you added in the previous step.
- In Certificate Details, click Generate Verification Code.
- The provisioning service creates a Verification Code that you can use to validate the certificate ownership. Copy the code to your clipboard.

## Digitally sign the verification code to create a verification certificate

Now, you need to sign the Verification Code with the private key associated with your X.509 CA certificate, which generates a signature. This is known as Proof of possession and results in a signed verification certificate.

Microsoft provides tools and samples that can help you create a signed verification certificate:

- The Azure IoT Hub C SDK provides PowerShell (Windows) and Bash (Linux) scripts to help you create CA and leaf certificates for development and to perform proof-of-possession using a verification code. You can download the files relevant to your system to a working folder and follow the instructions in the Managing CA certificates readme to perform proof-of-possession on a CA certificate.
- The Azure IoT Hub C# SDK contains the Group Certificate Verification Sample, which you can use to do proof-of-possession.

**Important:** In addition to performing proof-of-possession, the PowerShell and Bash scripts cited previously also allow you to create root certificates, intermediate certificates, and leaf certificates that can be used to authenticate and provision devices. These certificates should be used for development only. They should never be used in a production environment.

- The PowerShell and Bash scripts provided in the documentation and SDKs rely on OpenSSL. You may also use OpenSSL or other third-party tools to help you do proof-of-possession. For more information about tooling provided with the SDKs, see How to use tools provided in the SDKs.

## Upload the signed verification certificate

- Upload the resulting signature as a verification certificate to your provisioning service in the portal. In Certificate Details on the Azure portal, use the File Explorer icon next to the Verification Certificate .pem or .cer file field to upload the signed verification certificate from your system.
- Once the certificate is successfully uploaded, click Verify. The STATUS of your certificate changes to Verified in the Certificate Explorer list. Click Refresh if it does not update automatically.

# Roll Device Certificates

During the lifecycle of your IoT solution, you'll need to roll certificates. Two of the main reasons for rolling certificates would be a security breach, and certificate expirations. For now we will focus on rolling certificates due to expiration.

Rolling device certificates will involve updating the certificate stored on the device and the IoT hub. Afterwards, the device can reprovision itself with the IoT hub using normal auto-provisioning with the Device Provisioning Service.

## Obtain new certificates

There are many ways to obtain new certificates for your IoT devices. These include obtaining certificates from the device factory, generating your own certificates, and having a third party manage certificate creation for you.

Certificates are signed by each other to form a chain of trust from a root CA certificate to a leaf certificate. A signing certificate is the certificate used to sign the leaf certificate at the end of the chain of trust. A signing certificate can be a root CA certificate, or an intermediate certificate in chain of trust.

There are two different ways to obtain a signing certificate. The first way, which is recommended for production systems, is to purchase a signing certificate from a root certificate authority (CA). This way chains security down to a trusted source.

The second way is to create your own X.509 certificates using a tool like OpenSSL. This approach is great for testing X.509 certificates but provides few guarantees around security. We recommend you only use this approach for testing unless you are prepared to act as your own CA provider.

## Roll the certificate on the device

Certificates on a device should always be stored in a safe place like a hardware security module (HSM). The way you roll device certificates will depend on how they were created and installed in the devices in the first place.

If you got your certificates from a third party, you must look into how they roll their certificates. The process may be included in your arrangement with them, or it may be a separate service they offer.

If you're managing your own device certificates, you'll have to build your own pipeline for updating certificates. Make sure both old and new leaf certificates have the same common name (CN). By having the same CN, the device can reprovision itself without creating a duplicate registration record.

## Roll the certificate in the IoT hub

The device certificate can be manually added to an IoT hub. The certificate can also be automated using a Device Provisioning service instance. We will focus on this second case, where a Device Provisioning service instance is being used to support auto-provisioning.

When a device is initially provisioned through auto-provisioning, it boots-up, and contacts the provisioning service. The provisioning service responds by performing an identity check before creating a device identity in an IoT hub using the device's leaf certificate as the credential. The provisioning service then tells the device which IoT hub it's assigned to, and the device then uses its leaf certificate to authenticate and connect to the IoT hub.

Once a new leaf certificate has been rolled to the device, it can no longer connect to the IoT hub because it's using a new certificate to connect. The IoT hub only recognizes the device with the old certificate. The

result of the device's connection attempt will be an "unauthorized" connection error. To resolve this error, you must update the enrollment entry for the device to account for the device's new leaf certificate. Then the provisioning service can update the IoT Hub device registry information as needed when the device is reprovisioned.

One possible exception to this connection failure would be a scenario where you've created an Enrollment Group for your device in the provisioning service. In this case, if you aren't rolling the root or intermediate certificates in the device's certificate chain of trust, then the device will be recognized if the new certificate is part of the chain of trust defined in the enrollment group. If this scenario arises as a reaction to a security breach, you should at least blacklist the specific device certificates in the group that are considered to be breached.

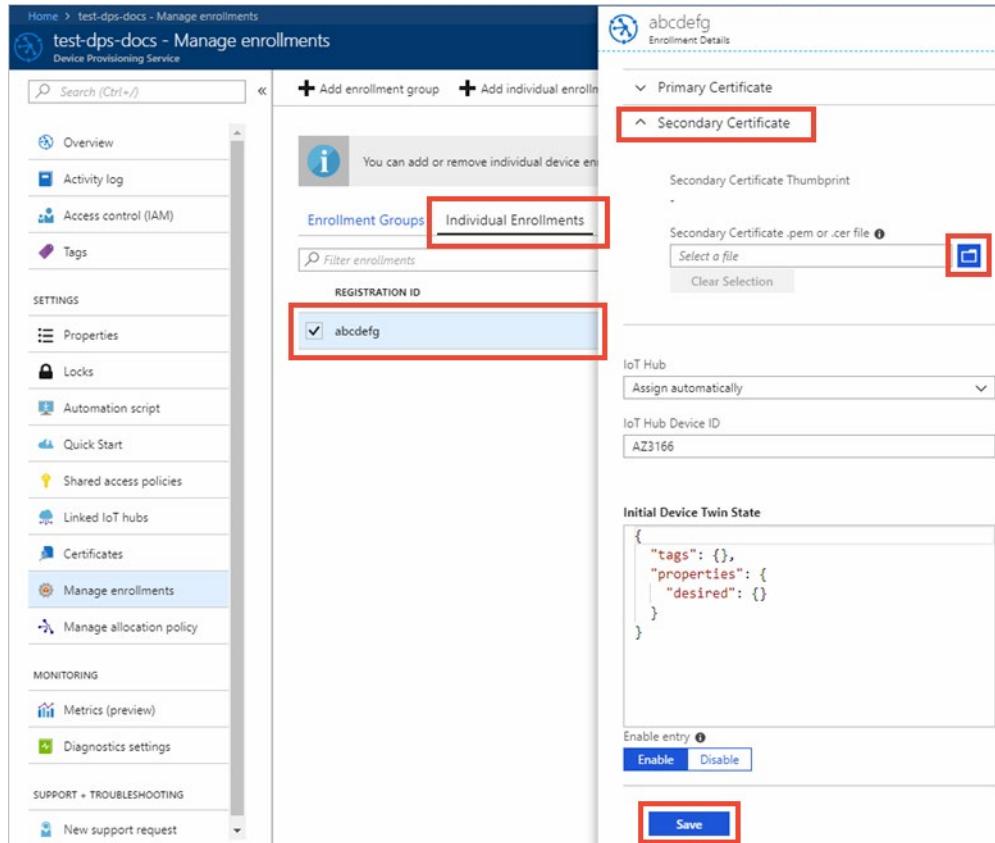
Updating enrollment entries for rolled certificates is accomplished on the Manage enrollments page of the Device Provisioning Service instance that has the enrollment entry for your device. How you handle updating the enrollment entry will depend on whether you're using individual enrollments, or group enrollments.

## Individual enrollments and certificate expiration

If you're rolling certificates to handle certificate expirations, you should use the secondary certificate configuration as follows to reduce downtime for devices attempting to provision.

Later when the secondary certificate also nears expiration, and needs to be rolled, you can rotate to using the primary configuration. Rotating between the primary and secondary certificates in this way reduces downtime for devices attempting to provision.

1. Open the Manage enrollments page of the Device Provisioning Service instance that has the enrollment entry for your device.
2. Click Individual Enrollments, and then click the registration ID entry in the list.
3. Click Secondary Certificate and then, click the folder icon to select the new certificate to be uploaded for the enrollment entry.
4. Click Save.



## Enrollment groups and certificate expiration

If you are rolling certificates to handle certificate expirations, you should use the secondary certificate configuration as follows to ensure no downtime for devices attempting to provision.

Later when the secondary certificate also nears expiration, and needs to be rolled, you can rotate to using the primary configuration. Rotating between the primary and secondary certificates in this way ensures no downtime for devices attempting to provision.

For more information about rolling certificates for enrollment groups with expiring certificates see <https://docs.microsoft.com/en-us/azure/iot-dps/how-to-roll-certificates#enrollment-groups-and-certificate-expiration>.

## Deprovisioning Process

You may find it necessary to deprovision devices that were previously auto-provisioned through the Device Provisioning Service. For example, a device may be sold or moved to a different IoT hub, or it may be lost, stolen, or otherwise compromised.

In general, deprovisioning a device involves two steps:

- Disenroll the device from your provisioning service, to prevent future auto-provisioning. Depending on whether you want to revoke access temporarily or permanently, you may want to either disable or

MCT USE ONLY. STUDENT USE PROHIBITED

delete an enrollment entry. For devices that use X.509 attestation, you may want to disable/delete an entry in the hierarchy of your existing enrollment groups.

- To learn how to disenroll a device, see **How to disenroll a device from Azure IoT Hub Device Provisioning Service<sup>5</sup>**.
- To learn how to disenroll a device programmatically using one of the provisioning service SDKs, see **Manage device enrollments with service SDKs<sup>6</sup>**.
- Deregister the device from your IoT Hub, to prevent future communications and data transfer. Again, you can temporarily disable or permanently delete the device's entry in the identity registry for the IoT Hub where it was provisioned. See **Disable devices<sup>7</sup>** to learn more about disablement. See "Device Management / IoT Devices" for your IoT Hub resource, in the Azure portal.

The exact steps you take to deprovision a device depend on its attestation mechanism and its applicable enrollment entry with your provisioning service.

## Manage Disenrollment

Proper management of device credentials is crucial for high-profile systems like IoT solutions. A best practice for such systems is to have a clear plan of how to revoke access for devices when their credentials, whether a shared access signatures (SAS) token or an X.509 certificate, might be compromised.

Enrollment in the Device Provisioning Service enables a device to be auto-provisioned. A provisioned device is one that has been registered with IoT Hub, allowing it to receive its initial device twin state and begin reporting telemetry data.

## Blacklist devices by using an individual enrollment entry

Individual enrollments apply to a single device and can use either X.509 certificates or SAS tokens (in a real or virtual TPM) as the attestation mechanism. (Devices that use SAS tokens as their attestation mechanism can be provisioned only through an individual enrollment.) To blacklist a device that has an individual enrollment, you can either disable or delete its enrollment entry.

## Temporarily Blacklist a Device

To temporarily blacklist the device by disabling its enrollment entry:

1. Sign in to the Azure portal and select All resources from the left menu.
2. In the list of resources, select the provisioning service that you want to blacklist your device from.
3. In your provisioning service, select Manage enrollments, and then select the Individual Enrollments tab.
4. Select the enrollment entry for the device that you want to blacklist.
5. On your enrollment page, scroll to the bottom, and select Disable for the Enable entry switch, and then select Save.

<sup>5</sup> <https://docs.microsoft.com/en-us/azure/iot-dps/how-to-revoke-device-access-portal>

<sup>6</sup> <https://docs.microsoft.com/en-us/azure/iot-dps/how-to-manage-enrollments-sdks>

<sup>7</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry#disable-devices>

## Permanently Blacklist a Device

To permanently blacklist the device by deleting its enrollment entry:

1. Sign in to the Azure portal and select All resources from the left menu.
2. In the list of resources, select the provisioning service that you want to blacklist your device from.
3. In your provisioning service, select Manage enrollments, and then select the Individual Enrollments tab.
4. Select the check box next to the enrollment entry for the device that you want to blacklist.
5. Select Delete at the top of the window, and then select Yes to confirm that you want to remove the enrollment.

After you finish the procedure, you should see your entry removed from the list of individual enrollments.

## Blacklist an X.509 intermediate or root CA certificate by using an enrollment group

X.509 certificates are typically arranged in a certificate chain of trust. If a certificate at any stage in a chain becomes compromised, trust is broken. The certificate must be blacklisted to prevent Device Provisioning Service from provisioning devices downstream in any chain that contains that certificate. To learn more about X.509 certificates and how they are used with the provisioning service, see X.509 certificates.

An enrollment group is an entry for devices that share a common attestation mechanism of X.509 certificates signed by the same intermediate or root CA. The enrollment group entry is configured with the X.509 certificate associated with the intermediate or root CA. The entry is also configured with any configuration values, such as twin state and IoT hub connection, that are shared by devices with that certificate in their certificate chain. To blacklist the certificate, you can either disable or delete its enrollment group.

## Blacklist specific devices in an enrollment group

Devices that implement the X.509 attestation mechanism use the device's certificate chain and private key to authenticate. When a device connects and authenticates with Device Provisioning Service, the service first looks for an individual enrollment that matches the device's credentials. The service then searches enrollment groups to determine whether the device can be provisioned. If the service finds a disabled individual enrollment for the device, it prevents the device from connecting. The service prevents the connection even if an enabled enrollment group for an intermediate or root CA in the device's certificate chain exists.

## Provision for Multitenancy

The allocation policies defined by the provisioning service support a variety of allocation scenarios. Two common scenarios are:

- **Geolocation / GeoLatency:** As a device moves between locations, network latency is improved by having the device provisioned to the IoT hub closest to each location. In this scenario, a group of IoT hubs, which span across regions, are selected for enrollments. The Lowest latency allocation policy is selected for these enrollments. This policy causes the Device Provisioning Service to evaluate device latency and determine the closest IoT hub out of the group of IoT hubs.
- **Multi-tenancy:** Devices used within an IoT solution may need to be assigned to a specific IoT hub or group of IoT hubs. The solution may require all devices for a particular tenant to communicate with a

specific group of IoT hubs. In some cases, a tenant may own IoT hubs and require devices to be assigned to their IoT hubs.

It is common to combine these two scenarios. For example, a multitenant IoT solution will commonly assign tenant devices using a group of IoT hubs that are scattered across regions. These tenant devices can be assigned to the IoT hub in that group that has the lowest latency based on geographic location.

Provisioning devices for the multitenant scenario across regions, as described in the example above, involves the following:

- two (or more) regional IoT hubs
- an enrollment group that uses a multitenant enrollment and specifies assigning devices based on lowest latency
- multiple devices provisioned in each region

With this configuration in place you will see that devices in each region are provisioned to the same tenant in the closest region.

For instructions that describe how to configure and test this multitenancy scenario, see [\*\*https://docs.microsoft.com/en-us/azure/iot-dps/how-to-provision-multitenant\*\*](https://docs.microsoft.com/en-us/azure/iot-dps/how-to-provision-multitenant).

## About the Module 3 Labs

### Module Labs

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

- Lab 5: Individual Enrollment of Device in DPS
- Lab 6: Automatic Enrollment of Devices in DPS

**WARNING:** Be prepared for UI changes

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

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



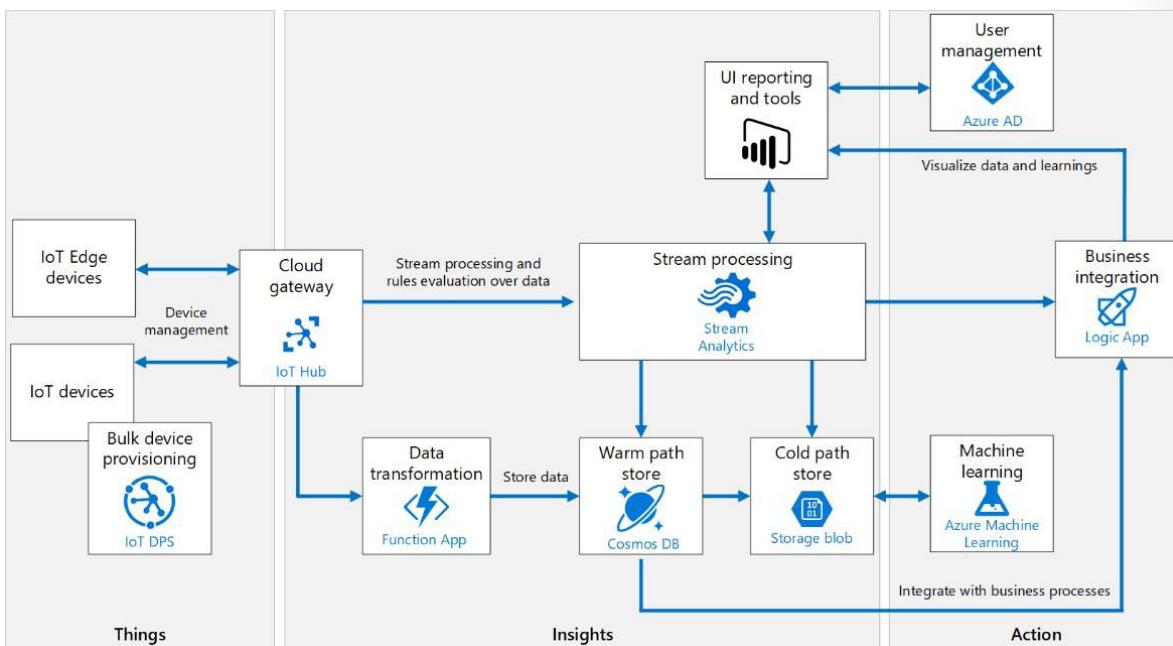
## Module 4 Message Processing and Analytics

### Messages and Message Processing

#### What is Message Processing

Message processing includes the series of actions that we use to communicate, modify, evaluate, react-to, and store message information.

The Azure IoT Reference Architecture diagram illustrates the primary message processing pathways.



As you can see, message processing involves a broad spectrum of protocols, services, and standards. Here are some of the topics that we will focus on:

- Message Format: To support seamless interoperability across protocols, IoT Hub defines a common message format for all device-facing protocols.

- Message Routing: This IoT Hub feature enables users to route device-to-cloud messages to service endpoints like Azure Storage containers, Event Hubs, Service Bus queues, and Service Bus topics. Routing also provides a querying capability to filter the data before routing it to the endpoints. In addition to device telemetry data, you can also send non-telemetry events that can be used to trigger actions.
- Event Grid: Azure Event Grid is a fully managed event routing service that uses a publish-subscribe model. IoT Hub and Event Grid work together to integrate IoT Hub events into Azure and non-Azure services, in near-real time. IoT Hub publishes device events, which are generally available, and now also publishes telemetry events, which is in public preview.
- Message Enrichment: Message enrichment is the ability of the IoT Hub to stamp messages with additional information before the messages are sent to the designated endpoint.
- Azure Stream Analytics: Azure Stream Analytics (ASA) is a real-time analytics and complex event-processing engine that is designed to analyze and process high volumes of fast streaming data from multiple sources simultaneously. ASA supports inputs and outputs using a wide range storage options.
- Azure Functions: Azure Functions is a solution for easily running small pieces of code, or "functions," in the cloud. Azure Functions lets you develop serverless applications on Microsoft Azure and is a great solution for processing data, integrating systems, and working with the Internet-of-Things. Azure Functions provides several templates that will help you to get you started with key IoT scenarios.
- Warm and Cold Storage: For architectures that produce significant amounts of data, a common pattern is to split the data into "warm" and "cold" data stores. Traditionally, data stored in cold storage is accessed infrequently, while data stored in warm storage accessed frequently.

## IoT Hub Common Message Format

To support seamless interoperability across protocols, IoT Hub defines a common message format for all device-facing protocols. This message format is used for both device-to-cloud routing and cloud-to-device 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<sup>1</sup>** events than **Service Bus<sup>2</sup>** messages in that there is a high volume of events passing through the service that can be read by multiple readers.

An IoT Hub message consists of:

- A predetermined set of *system properties* as listed below.
- A set of *application properties*. A dictionary of string properties that the application can define and access, without needing to deserialize the message body. IoT Hub never modifies these properties.
- An opaque binary body.

Property names and values can only contain ASCII alphanumeric characters (along with the additional characters listed below) when you send device-to-cloud messages using the HTTPS protocol or send cloud-to-device messages:

```
{'!', '#', '$', '%', '&', '^', '*', '+', '-', '.', '^', '_', '^', '|', '~'}
```

<sup>1</sup> <https://docs.microsoft.com/en-us/azure/event-hubs/>

<sup>2</sup> <https://docs.microsoft.com/en-us/azure/service-bus-messaging/>

Device-to-cloud messaging with IoT Hub has the following characteristics:

- Device-to-cloud messages are durable and retained in an IoT hub's default messages/events endpoint for up to seven days.
- 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.
- IoT Hub does not allow arbitrary partitioning. Device-to-cloud messages are partitioned based on their originating **deviceId**.
- As explained in **Control access to IoT Hub**<sup>3</sup>, IoT Hub enables per-device authentication and access control.
- You can stamp messages with information that goes into the application properties. For more information, please see message enrichments.

## System Properties of D2C IoT Hub Messages

Property	Description	User Settable?	Keyword for routing query
message-id	A user-settable identifier for the message used for request-reply patterns. Format: A case-sensitive string (up to 128 characters long) of ASCII 7-bit alphanumeric characters. The following additional characters can also be used {'-', ':', ';', '+', '%', '_', '#', '*', '?', '!', '(', ')', ',', '=', '@', '/', '\$', ""}	Yes	messageId
iothub-enqueuedtime	Date and time the Device-to-Cloud message was received by IoT Hub.	No	enqueuedTime
user-id	An ID used to specify the origin of messages. When messages are generated by IoT Hub, it is set to {iot hub name}.	Yes	userId
iothub-connection-device-id	An ID set by IoT Hub on device-to-cloud messages. It contains the <b>deviceId</b> of the device that sent the message.	No	connectionDeviceId

<sup>3</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-security>

Property	Description	User Settable?	Keyword for routing query
iothub-connection-module-id	An ID set by IoT Hub on device-to-cloud messages. It contains the <b>moduleId</b> of the device that sent the message.	No	connectionModuleId
iothub-connection-auth-generation-id	An ID set by IoT Hub on device-to-cloud messages. It contains the <b>connectionDeviceGenerationId</b> (as per <b>Device identity properties</b> ( <a href="https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry#device-identity-properties">https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry#device-identity-properties</a> )) of the device that sent the message.	No	connectionDeviceGenerationId
iothub-connection-auth-method	An authentication method set by IoT Hub on device-to-cloud messages. This property contains information about the authentication method used to authenticate the device sending the message.	No	connectionAuthMethod

## System Properties of C2D IoT Hub messages

Property	Description	User Settable?
message-id	A user-settable identifier for the message used for request-reply patterns. Format: A case-sensitive string (up to 128 characters long) of ASCII 7-bit alphanumeric characters. The following additional characters can also be used {‘-’, ‘.’, ‘!’, ‘+’, ‘%’, ‘_’, ‘#’, ‘*’, ‘?’ , ‘!’, ‘(’, ‘)’, ‘;’, ‘=’, ‘@’, ‘;’, ‘\$’, “”}.	Yes
sequence-number	A number (unique per device-queue) assigned by IoT Hub to each cloud-to-device message.	No
to	A destination specified in Cloud-to-Device messages.	No

Property	Description	User Settable?
absolute-expiry-time	Date and time of message expiration.	No
correlation-id	A string property in a response message that typically contains the MessageId of the request, in request-reply patterns.	Yes
user-id	An ID used to specify the origin of messages. When messages are generated by IoT Hub, it is set to {iot hub name}.	Yes
iothub-ack	A feedback message generator. This property is used in cloud-to-device messages to request IoT Hub to generate feedback messages as a result of the consumption of the message by the device. Possible values: <b>none</b> (default): no feedback message is generated, <b>positive</b> : receive a feedback message if the message was completed, <b>negative</b> : receive a feedback message if the message expired (or maximum delivery count was reached) without being completed by the device, or <b>full</b> : both positive and negative.	Yes

## Message size

IoT Hub measures message size in a protocol-agnostic way, considering only the actual payload. The size in bytes is calculated as the sum of the following values:

- The body size in bytes.
- The size in bytes of all the values of the message system properties.
- The size in bytes of all user property names and values.

Property names and values are limited to ASCII characters, so the length of the strings equals the size in bytes.

## Anti-spoofing properties

To avoid device spoofing in device-to-cloud messages, IoT Hub stamps all messages with the following properties:

- **iothub-connection-device-id**
- **iothub-connection-auth-generation-id**
- **iothub-connection-auth-method**

The first two contain the **deviceId** and **generationId** of the originating device, as per **Device identity properties**<sup>4</sup>.

The **iothub-connection-auth-method** property contains a JSON serialized object, with the following properties:

```
{
  "scope": "{ hub | device }",
  "type": "{ symkey | sas | x509 }",
  "issuer": "iothub"
}
```

## Introduction to Message Routing

IoT Hub Message Routing enables users to route device-to-cloud messages to service-facing endpoints. Routing also provides a querying capability to filter the data before routing it to the endpoints. Each routing query you configure has the following properties:

Property	Description
Name	The unique name that identifies the query.
Source	The origin of the data stream to be acted upon. For example, device telemetry.
Condition	The query expression for the routing query that is run against the message application properties, system properties, message body, device twin tags, and device twin properties to determine if it is a match for the endpoint. For more information about constructing a query, see the see message routing query syntax
Endpoint	The name of the endpoint where IoT Hub sends messages that match the query. We recommend that you choose an endpoint in the same region as your IoT hub.

A single message may match the condition on multiple routing queries, in which case IoT Hub delivers the message to the endpoint associated with each matched query. IoT Hub also automatically deduplicates message delivery, so if a message matches multiple queries that have the same destination, it is only written once to that destination.

## Endpoints and routing

An IoT hub has a default built-in messaging endpoint (messages/events).

You can create custom endpoints to route messages to by linking other services in your subscription to the hub. IoT Hub currently supports the following custom endpoints:

- Azure Storage containers
- Event Hubs
- Service Bus queues

<sup>4</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-identity-registry#device-identity-properties>

- Service Bus topics

When you use routing and custom endpoints, messages are only delivered to the built-in endpoint if they don't match any query. To deliver messages to the built-in endpoint as well as to a custom endpoint, add a route that sends messages to the built-in events endpoint.

**Note**

- IoT Hub only supports writing data to Azure Storage containers as blobs.
- Service Bus queues and Service Bus topics that have Sessions or Duplicate Detection enabled are not supported as custom endpoints.

## IoT Hub Built-in Endpoint

By default, messages are routed to the built-in service-facing endpoint (messages/events) that is compatible with Event Hubs. This endpoint is currently only exposed using the AMQP protocol on port 5671. An IoT hub exposes the following properties to enable you to control the built-in Event Hub-compatible messaging endpoint messages/events.

Property	Description
Partition count	Set this property at creation to define the number of partitions for device-to-cloud event ingestion.
Retention time	This property specifies how long in days messages are retained by IoT Hub. The default is one day, but it can be increased to seven days.

IoT Hub allows data retention in the built-in Event Hubs for a maximum of 7 days. You can set the retention time during creation of your IoT Hub. Data retention time in IoT Hub depends on your IoT hub tier and unit type. In terms of size, the built-in Event Hubs can retain messages of the maximum message size up to at least 24 hours of quota. For example, for 1 S1 unit IoT Hub provides enough storage to retain at least 400K messages of 4k size each. If your devices are sending smaller messages, they may be retained for longer (up to 7 days) depending on how much storage is consumed. We guarantee retaining the data for the specified retention time as a minimum.

IoT Hub also enables you to manage consumer groups on the built-in device-to-cloud receive endpoint. You can have up to 20 consumer groups for each IoT Hub.

If you're using message routing and the fallback route is enabled, all messages that don't match a query on any route go to the built-in endpoint. If you disable this fallback route, messages that don't match any query are dropped.

You can modify the retention time, either programmatically using the IoT Hub resource provider REST APIs, or with the Azure portal.

IoT Hub exposes the messages/events built-in endpoint for your back-end services to read the device-to-cloud messages received by your hub. This endpoint is Event Hub-compatible, which enables you to use any of the mechanisms the Event Hubs service supports for reading messages.

## Accessing the Built-in Endpoint

Some product integrations and Event Hubs SDKs are aware of IoT Hub and let you use your IoT hub service connection string to connect to the built-in endpoint.

When you use Event Hubs SDKs or product integrations that are unaware of IoT Hub, you need an Event Hub-compatible endpoint and Event Hub-compatible name. You can retrieve these values from IoT hub service in the Azure portal by opening the **Built-in endpoints** blade.

The Events section contains the following values: Partitions, Event Hub-compatible name, Event Hub-compatible endpoint, Retention time, and Consumer groups.

Each IoT hub comes with built-in system endpoints to handle system and device messages.

**Events**

Events is the default endpoint, and is used until custom routing rules are created.

Partitions: 4

Event Hub-compatible name: dochub

Event Hub-compatible endpoint: Endpoint=sb://iothub-ns-dochub-19403-b768544313-servicebus.windows.net/;SharedAccessKeyName=iothubowner;SharedAccessKey=axnzt04+SFAxF6jr2d2Dm

Retain for: 1 Days

Consumer Groups:

- CONSUMER GROUPS
- Default

Create new consumer group

**Cloud to device messaging**

Control message retention time and retry attempts.

Default TTL: 1 Hours

Feedback retention time: 1 Hours

In the portal, the Event Hub-compatible endpoint field contains a complete Event Hubs connection string that looks like: `Endpoint=sb://abcd1234namespace.servicebus.windows.net/;SharedAccessKeyName=iothubowner;SharedAccessKey=keykeykeykeykeykeykey=;EntityPath=iothub-ehub-abcd-1234-123456`. If the SDK you're using requires other values, then they would be:

Name	Value
Endpoint	<code>sb://abcd1234namespace.servicebus.windows.net/</code>
Hostname	<code>abcd1234namespace.servicebus.windows.net</code>
Namespace	<code>abcd1234namespace</code>

You can then use any shared access policy that has the ServiceConnect permissions to connect to the specified Event Hub.

The SDKs you can use to connect to the built-in Event Hub-compatible endpoint that IoT Hub exposes include:

Language	SDK	Example	Notes
.NET	<a href="https://github.com/Azure/azure-event-hubs-dotnet">https://github.com/Azure/azure-event-hubs-dotnet</a>	<a href="https://docs.microsoft.com/en-us/azure/iot-hub/quick-start-send-telemetry-dotnet">https://docs.microsoft.com/en-us/azure/iot-hub/quick-start-send-telemetry-dotnet</a>	Uses Event Hubs-compatible information
Java	<a href="https://github.com/Azure/azure-event-hubs-java">https://github.com/Azure/azure-event-hubs-java</a>	<a href="https://docs.microsoft.com/en-us/azure/iot-hub/quick-start-send-telemetry-java">https://docs.microsoft.com/en-us/azure/iot-hub/quick-start-send-telemetry-java</a>	Uses Event Hubs-compatible information

Language	SDK	Example	Notes
Node.js	<a href="https://github.com/Azure/azure-event-hubs-node">https://github.com/Azure/azure-event-hubs-node</a>	<a href="https://docs.microsoft.com/en-us/azure/iot-hub/quick-start-send-telemetry-node">https://docs.microsoft.com/en-us/azure/iot-hub/quick-start-send-telemetry-node</a>	Uses IoT Hub connection string
Python	<a href="https://github.com/Azure/azure-event-hubs-python">https://github.com/Azure/azure-event-hubs-python</a>	<a href="https://github.com/Azure/azure-event-hubs-python/blob/master/examples/iothub_recv.py">https://github.com/Azure/azure-event-hubs-python/blob/master/examples/iothub_recv.py</a>	Uses IoT Hub connection string

The product integrations you can use with the built-in Event Hub-compatible endpoint that IoT Hub exposes include:

- Azure Functions.
- Azure Stream Analytics.
- Time Series Insights.
- Apache Storm spout.
- Apache Spark integration.
- Azure Databricks.

## Message Routing to Multiple Endpoints

Message routing enables you to send messages from your devices to cloud services in an automated, scalable, and reliable manner. Message routing can be used for:

- Sending device telemetry messages as well as events namely, device lifecycle events, and device twin change events to the built-in-endpoint and custom endpoints.
- Filtering data before routing it to various endpoints by applying rich queries. Message routing allows you to query on the message properties and message body as well as device twin tags and device twin properties.

IoT Hub needs write access to these service endpoints for message routing to work. If you configure your endpoints through the Azure portal, the necessary permissions are added for you. Make sure you configure your services to support the expected throughput. For example, if you are using Event Hubs as a custom endpoint, you must configure the throughput units for that event hub so it can handle the ingress of events you plan to send via IoT Hub message routing. Similarly, when using a Service Bus Queue as an endpoint, you must configure the maximum size to ensure the queue can hold all the data ingressed, until it is egressed by consumers. When you first configure your IoT solution, you may need to monitor your additional endpoints and make any necessary adjustments for the actual load.

If a message matches multiple routes that point to the same endpoint, IoT Hub delivers the message to that endpoint only once. Therefore, you don't need to configure deduplication on your Service Bus queue or topic. In partitioned queues, partition affinity guarantees message ordering.

## Custom Endpoints

IoT hub supports Azure Storage containers, Event Hubs, Service Bus queues, and Service Bus topics as custom endpoints.

## Azure Storage Endpoint

There are two storage services IoT Hub can route messages to – Azure Blob Storage and Azure Data Lake Storage Gen2 (ADLS Gen2) accounts. Azure Data Lake Storage accounts are hierarchical namespace-enabled storage accounts built on top of blob storage. Both of these use blobs for their storage.

## Event Hubs

Event Hubs is a service that processes large amounts of event data (telemetry) from connected devices and applications. After you collect data into Event Hubs, you can store the data using a storage cluster or transform it using a real-time analytics provider. This large-scale event collection and processing capability is a key component of modern application architectures including the Internet of Things (IoT).

## Service Bus Queues and Service Bus Topics

Service Bus queues and topics used as IoT Hub endpoints must not have Sessions or Duplicate Detection enabled. If either of those options are enabled, the endpoint appears as Unreachable in the Azure portal.

## Fallback route

The fallback route sends all the messages that don't satisfy query conditions on any of the existing routes to the built-in-Event Hubs (messages/events), that is compatible with Event Hubs. If message routing is turned on, you can enable the fallback route capability. Once a route is created, data stops flowing to the built-in-endpoint, unless a route is created to that endpoint. If there are no routes to the built-in-endpoint and a fallback route is enabled, only messages that don't match any query conditions on routes will be sent to the built-in-endpoint. Also, if all existing routes are deleted, fallback route must be enabled to receive all data at the built-in-endpoint.

You can enable/disable the fallback route in the Azure portal->Message Routing blade. You can also use Azure Resource Manager for FallbackRouteProperties to use a custom endpoint for fallback route.

## Non-telemetry events

In addition to device telemetry, message routing also enables sending device twin change events, device lifecycle events, and digital twin change events (in public preview). For example, if a route is created with data source set to device twin change events, IoT Hub sends messages to the endpoint that contain the change in the device twin. Similarly, if a route is created with data source set to device lifecycle events, IoT Hub sends a message indicating whether the device was deleted or created. Finally, as part of the IoT Plug and Play public preview, a developer can create routes with data source set to digital twin change events and IoT Hub sends messages whenever a digital twin property is set or changed, a digital twin is replaced, or when a change event happens for the underlying device twin.

IoT Hub also integrates with Azure Event Grid to publish device events to support real-time integrations and automation of workflows based on these events. See **key differences between message routing and Event Grid<sup>5</sup>** to learn which works best for your scenario.

## Testing routes

When you create a new route or edit an existing route, you should test the route query with a sample message. You can test individual routes or test all routes at once and no messages are routed to the

<sup>5</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-event-grid-routing-comparison>

endpoints during the test. Azure portal, Azure Resource Manager, Azure PowerShell, and Azure CLI can be used for testing. Outcomes help identify whether the sample message matched the query, message did not match the query, or test couldn't run because the sample message or query syntax are incorrect. To learn more, see **Test Route<sup>6</sup>** and **Test all routes<sup>7</sup>**.

Ordering guarantees with at least once delivery

IoT Hub message routing guarantees ordered and at least once delivery of messages to the endpoints. This means that there can be duplicate messages and a series of messages can be retransmitted honoring the original message ordering. For example, if the original message order is [1, 2, 3, 4], you could receive a message sequence like [1, 2, 1, 2, 3, 1, 2, 3, 4]. The ordering guarantee is that if you ever receive message [1], it would always be followed by [2, 3, 4].

For handling message duplicates, we recommend stamping a unique identifier in the application properties of the message at the point of origin, which is usually a device or a module. The service consuming the messages can handle duplicate messages using this identifier.

## Latency

When you route device-to-cloud telemetry messages using built-in endpoints, there is a slight increase in the end-to-end latency after the creation of the first route.

In most cases, the average increase in latency is less than 500 ms. You can monitor the latency using `Routing: message latency for messages/events or d2c.endpoints.latency.builtIn.events` IoT Hub metric. Creating or deleting any route after the first one does not impact the end-to-end latency.

## Monitoring and troubleshooting

IoT Hub provides several metrics related to routing and endpoints to give you an overview of the health of your hub and messages sent. You can combine information from multiple metrics to identify root cause for issues. For example, use metric `Routing: telemetry messages dropped` or `d2c.telemetry.egress.dropped` to identify the number of messages that were dropped when they didn't match queries on any of the routes and fallback route was disabled. **IoT Hub metrics<sup>8</sup>** lists all metrics that are enabled by default for your IoT Hub.

You can use the REST API **Get Endpoint Health<sup>9</sup>** to get health status of the endpoints. We recommend using the IoT Hub metrics related to routing message latency to identify and debug errors when endpoint health is dead or unhealthy. For example, for endpoint type Event Hubs, you can monitor `d2c.endpoints.latency.eventHubs`. The status of an unhealthy endpoint will be updated to healthy when IoT Hub has established an eventually consistent state of health.

Using the `routes` diagnostic logs in Azure Monitor diagnostic settings, you can track errors that occur during evaluation of a routing query and endpoint health as perceived by IoT Hub, for example when an endpoint is dead. These diagnostic logs can be sent to Azure Monitor logs, Event Hubs, or Azure Storage for custom processing.

<sup>6</sup> <https://docs.microsoft.com/en-us/rest/api/iothub/iothubresource/testroute>

<sup>7</sup> <https://docs.microsoft.com/en-us/rest/api/iothub/iothubresource/testallroutes>

<sup>8</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-metrics>

<sup>9</sup> [https://docs.microsoft.com/rest/api/iothub/iothubresource/getendpointhealth#iothubresource\\_getendpointhealth](https://docs.microsoft.com/rest/api/iothub/iothubresource/getendpointhealth#iothubresource_getendpointhealth)

# Message Routing Query Syntax

Message routing enables users to route different data types namely, device telemetry messages, device lifecycle events, and device twin change events to various endpoints. You can also apply rich queries to this data before routing it to receive the data that matters to you.

Message routing allows you to query on the message properties and message body as well as device twin tags and device twin properties. If the message body is not JSON, message routing can still route the message, but queries cannot be applied to the message body. Queries are described as Boolean expressions where a Boolean true makes the query succeed which routes all the incoming data, and Boolean false fails the query and no data is routed. If the expression evaluates to null or undefined, it is treated as false and an error will be generated in diagnostic logs in case of a failure. The query syntax must be correct for the route to be saved and evaluated.

## Message routing query based on message properties

The IoT Hub defines a common format for all device-to-cloud messaging for interoperability across protocols. IoT Hub message assumes the following JSON representation of the message. System properties are added for all users and identify content of the message. Users can selectively add application properties to the message. We recommend using unique property names as IoT Hub device-to-cloud messaging is not case-sensitive. For example, if you have multiple properties with the same name, IoT Hub will only send one of the properties.

```
{
  "message": {
    "systemProperties": {
      "contentType": "application/json",
      "contentEncoding": "UTF-8",
      "iothub-message-source": "deviceMessages",
      "iothub-enqueuedtime": "2017-05-08T18:55:31.8514657Z"
    },
    "appProperties": {
      "processingPath": "{cold | warm | hot}",
      "verbose": "{true, false}",
      "severity": 1-5,
      "testDevice": "{true | false}"
    },
    "body": "{\"Weather\":{\"Temperature\":50}}"
  }
}
```

## System properties

System properties help identify contents and source of the messages.

As described in the **IoT Hub Messages**<sup>10</sup>, there are additional system properties in a message. In addition to `contentType`, `contentEncoding`, and `enqueuedTime`, the `connectionDeviceId` and `connectionModuleId` can also be queried.

---

<sup>10</sup> <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-messages-construct>

## Application properties

Application properties are user-defined strings that can be added to the message. These fields are optional.

## Query expressions

A query on message system properties needs to be prefixed with the \$ symbol. Queries on application properties are accessed with their name and should not be prefixed with the \$ symbol. If an application property name begins with \$, then IoT Hub will search for it in the system properties, and if it is not found, then it will look in the application properties. For example:

To query on system property contentEncoding

```
$contentEncoding = 'UTF-8'
```

To query on application property processingPath:

```
processingPath = 'hot'
```

To combine these queries, you can use Boolean expressions and functions:

```
$contentEncoding = 'UTF-8' AND processingPath = 'hot'
```

A full list of supported operators and functions is shown in Expression and conditions: <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-query-language#expressions-and-conditions>.

## Message routing query based on message body

To enable querying on message body, the message should be in a JSON encoded in either UTF-8, UTF-16 or UTF-32. The contentType must be set to application/JSON and contentEncoding to one of the supported UTF encodings in the system properties. If these properties are not specified, IoT Hub will not evaluate the query expression on the message body.

The following example shows how to create a message with a properly formed and encoded JSON body:

```
var messageBody = JSON.stringify(Object.assign({}, {
    "Weather": {
        "Temperature": 50,
        "Time": "2017-03-09T00:00:00.000Z",
        "PrevTemperatures": [
            20,
            30,
            40
        ],
        ".IsEnabled": true,
        "Location": {
            "Street": "One Microsoft Way",
            "City": "Redmond",
            "State": "WA"
        },
        "HistoricalData": [

```

```

    {
        "Month": "Feb",
        "Temperature": 40
    },
    {
        "Month": "Jan",
        "Temperature": 30
    }
]
}
));
}

// Encode message body using UTF-8
var messageBytes = Buffer.from(messageBody, "utf8");

var message = new Message(messageBytes);

// Set message body type and content encoding
message.contentEncoding = "utf-8";
message.contentType = "application/json";

// Add other custom application properties
message.properties.add("Status", "Active");

deviceClient.sendEvent(message, (err, res) => {
    if (err) console.log('error: ' + err.toString());
    if (res) console.log('status: ' + res.constructor.name);
});

```

## Query expressions

A query on message body needs to be prefixed with the \$body. You can use a body reference, body array reference, or multiple body references in the query expression. Your query expression can also combine a body reference with message system properties, and message application properties reference. For example, the following are all valid query expressions:

\$body.Weather.HistoricalData[0].Month = 'Feb'

\$body.Weather.Temperature = 50 AND \$body.Weather.IsEnabled

length(\$body.Weather.Location.State) = 2

\$body.Weather.Temperature = 50 AND processingPath = 'hot'

## Message routing query based on device twin

Message routing enables you to query on Device Twin tags and properties, which are JSON objects. Querying on module twin is not supported. A sample of Device Twin tags and properties is shown below.

```
{  
  "tags": {  
    "deploymentLocation": {  
      "building": "43",  
      "floor": "1"  
    }  
  },  
  "properties": {  
    "desired": {  
      "telemetryConfig": {  
        "sendFrequency": "5m"  
      },  
      "$metadata": {...},  
      "$version": 1  
    },  
    "reported": {  
      "telemetryConfig": {  
        "sendFrequency": "5m",  
        "status": "success"  
      },  
      "batteryLevel": 55,  
      "$metadata": {...},  
      "$version": 4  
    }  
  }  
}
```

## Query Expressions

A query on device twin properties needs to be prefixed with the `$twin`. Your query expression can also combine a twin tag or property reference with a body reference, message system properties, and message application properties reference. We recommend using unique names in tags and properties as the query is not case-sensitive. Also refrain from using `twin`, `$twin`, `body`, or `$body`, as a property names. For example, the following are all valid query expressions:

```
$twin.properties.desired.telemetryConfig.sendFrequency = '5m'
```

```
$body.Weather.Temperature = 50 AND $twin.properties.desired.telemetryConfig.sendFrequency = '5m'
```

```
$twin.tags.deploymentLocation.floor = 1
```

# Compare message routing and Event Grid for IoT Hub

Azure IoT Hub provides the capability to stream data from your connected devices and integrate that data into your business applications. IoT Hub offers two methods for integrating IoT events into other Azure services or business applications.

1. **IoT Hub message routing:** This IoT Hub feature enables users to route device-to-cloud messages to service endpoints like Azure Storage containers, Event Hubs, Service Bus queues, and Service Bus topics. Routing also provides a querying capability to filter the data before routing it to the endpoints. In addition to device telemetry data, you can also send non-telemetry events that can be used to trigger actions.
2. **IoT Hub integration with Event Grid:** Azure Event Grid is a fully managed event routing service that uses a publish-subscribe model. IoT Hub and Event Grid work together to integrate IoT Hub events into Azure and non-Azure services, in near-real time. IoT Hub publishes device events, which are generally available, and now also publishes telemetry events, which is in public preview.

## Differences

While both message routing and Event Grid enable alert configuration, there are some key differences between the two. Refer to the following table for details:

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Device messages and events	Yes, message routing can be used for telemetry data, report device twin changes, device lifecycle events, and digital twin change events (part of the IoT Plug and Play public preview).	Yes, Event Grid can be used for telemetry data but can also report when devices are created, deleted, connected, and disconnected from IoT Hub
Ordering	Yes, ordering of events is maintained.	No, order of events is not guaranteed.
Filtering	Rich filtering on message application properties, message system properties, message body, device twin tags, and device twin properties. Filtering isn't applied to digital twin change events.	Filtering based on event type, subject type and attributes in each event. When subscribing to telemetry events, you can apply additional filters on the data to filter on message properties, message body and device twin in your IoT Hub, before publishing to Event Grid.

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Endpoints	Event Hubs Azure Blob Storage Service Bus queue Service Bus topics  Paid IoT Hub SKUs (S1, S2, and S3) are limited to 10 custom endpoints. 100 routes can be created per IoT Hub.	Azure Functions Azure Automation Event Hubs Logic Apps Storage Blob Custom Topics Queue Storage Microsoft Flow Third-party services through WebHooks  500 endpoints per IoT Hub are supported.
Cost	There is no separate charge for message routing. Only ingress of telemetry into IoT Hub is charged. For example, if you have a message routed to three different endpoints, you are billed for only one message.	There is no charge from IoT Hub. Event Grid offers the first 100,000 operations per month for free, and then \$0.60 per million operations afterwards.

## Similarities

IoT Hub message routing and Event Grid have similarities too, some of which are detailed in the following table:

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Maximum message size	256 KB, device-to-cloud	256 KB, device-to-cloud
Reliability	High: Delivers each message to the endpoint at least once for each route. Expires all messages that are not delivered within one hour.	High: Delivers each message to the webhook at least once for each subscription. Expires all events that are not delivered within 24 hours.
Scalability	High: Optimized to support millions of simultaneously connected devices sending billions of messages.	High: Capable of routing 10,000,000 events per second per region.
Latency	Low: Near-real time.	Low: Near-real time.
Send to multiple endpoints	Yes, send a single message to multiple endpoints.	Yes, send a single message to multiple endpoints.
Security	IoT Hub provides per-device identity and revocable access control.	Event Grid provides validation at three points: event subscriptions, event publishing, and webhook event delivery.

## How to choose

IoT Hub message routing and the IoT Hub integration with Event Grid perform different actions to achieve similar results. They both take information from your IoT Hub solution and pass it on so that other services can react. So how do you decide which one to use? Consider the following questions to help guide your decision:

- What kind of data are you sending to the endpoints?

Use IoT Hub message routing when you have to send telemetry data to other services. Message routing also enables querying message application and system properties, message body, device twin tags, and device twin properties.

The IoT Hub integration with Event Grid works with events that occur in the IoT Hub service. These IoT Hub events include telemetry data, device created, deleted, connected, and disconnected. When subscribing to telemetry events, you can apply additional filters on the data to filter on message properties, message body and device twin in your IoT Hub, before publishing to Event Grid.

- What endpoints need to receive this information?

IoT Hub message routing supports limited number of unique endpoints and endpoint types, but you can build connectors to reroute the data and events to additional endpoints.

The IoT Hub integration with Event Grid supports 500 endpoints per IoT Hub and a larger variety of endpoint types. It natively integrates with Azure Functions, Logic Apps, Storage and Service Bus queues, and also works with webhooks to extend sending data outside of the Azure service ecosystem and into third-party business applications.

- Does it matter if your data arrives in order?

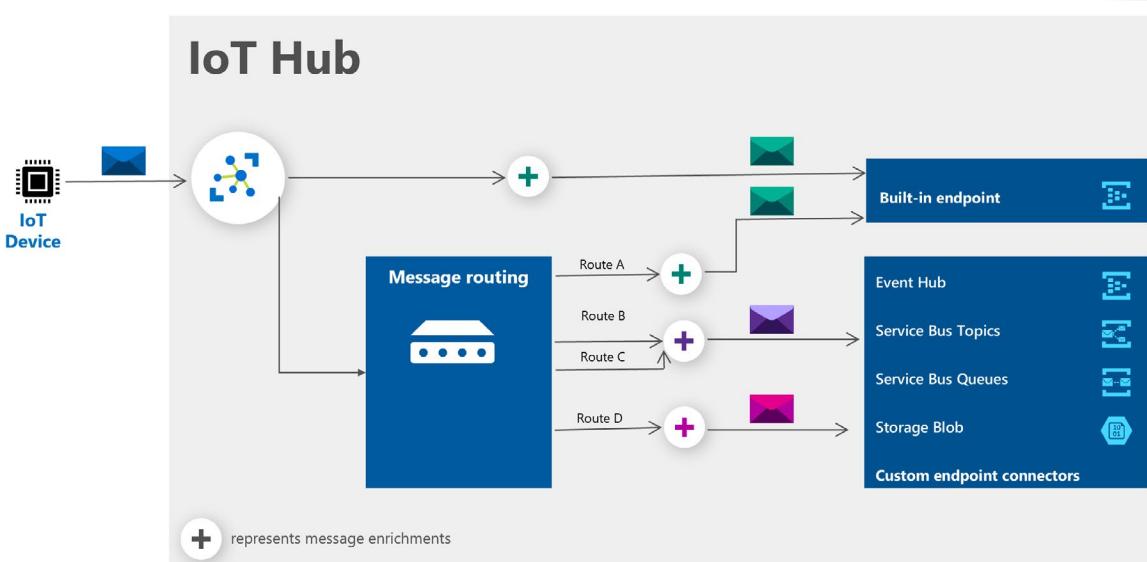
IoT Hub message routing maintains the order in which messages are sent, so that they arrive in the same way.

Event Grid does not guarantee that endpoints will receive events in the same order that they occurred. For those cases in which absolute order of messages is significant and/or in which a consumer needs a trustworthy unique identifier for messages, we recommend using message routing.

# Additional Considerations for IoT Hub Messaging

## Message Enrichments for D2C Messages

Message enrichment is the ability of the IoT Hub to stamp messages with additional information before the messages are sent to the designated endpoint. One reason to use message enrichments is to include data that can be used to simplify downstream processing. For example, enriching device telemetry messages with a device twin tag can reduce load on customers to make device twin API calls for this information.



A message enrichment has three key elements:

- Enrichment name or key
- A value
- One or more endpoints for which the enrichment should be applied.

The key is a string. A key can only contain alphanumeric characters or these special characters: hyphen (-), underscore (\_), and period (.)

The value can be any of the following examples:

- Any static string. Dynamic values such as conditions, logic, operations, and functions are not allowed. For example, if you develop a SaaS application that is used by several customers, you can assign an identifier to each customer and make that identifier available in the application. When the application runs, IoT Hub will stamp the device telemetry messages with the customer's identifier, making it possible to process the messages differently for each customer.
- The name of the IoT hub sending the message. This value is \$iothubname.
- Information from the device twin, such as its path. Examples would be \$twin.tags.field and \$twin.tags.latitude.

**Note:** At this time, only \$iothubname, \$twin.tags, \$twin.properties.desired, and \$twin.properties.reported are supported variables for message enrichment.

Message Enrichments are added as application properties to messages sent to chosen endpoint(s).

## Applying enrichments

The messages can come from any data source supported by IoT Hub message routing, including the following examples:

- device telemetry, such as temperature or pressure
- device twin change notifications – changes in the device twin
- device life-cycle events, such as when the device is created or deleted

You can add enrichments to messages that are going to the built-in endpoint of an IoT Hub, or messages that are being routed to custom endpoints such as Azure Blob storage, a Service Bus queue, or a Service Bus topic.

You can also add enrichments to messages that are being published to Event Grid by selecting the endpoint as Event Grid. For more information, see IoT Hub and Event Grid.

Enrichments are applied per endpoint. If you specify five enrichments to be stamped for a specific endpoint, all messages going to that endpoint are stamped with the same five enrichments.

## Limitations

- You can add up to 10 enrichments per IoT Hub for those hubs in the standard or basic tier. For IoT Hubs in the free tier, you can add up to 2 enrichments.
- In some cases, if you are applying an enrichment with a value set to a tag or property in the device twin, the value will be stamped as a string value. For example, if an enrichment value is set to `$twin.tags.field`, the messages will be stamped with the string "`$twin.tags.field`" rather than the value of that field from the twin. This happens in the following cases:
  - Your IoT Hub is in the basic tier. Basic tier IoT hubs do not support device twins.
  - Your IoT Hub is in the standard tier, but the device sending the message has no device twin.
  - Your IoT Hub is in the standard tier, but the device twin path used for the value of the enrichment does not exist. For example, if the enrichment value is set to `$twin.tags.location`, and the device twin does not have a `location` property under `tags`, the message is stamped with the string "`$twin.tags.location`".
- Updates to a device twin can take up to five minutes to be reflected in the corresponding enrichment value.
- The total message size, including the enrichments, can't exceed 256 KB. If a message size exceeds 256 KB, the IoT Hub will drop the message. You can use IoT Hub metrics to identify and debug errors when messages are dropped. For example, you can monitor `d2c.telemetry.egress.invalid`.
- Message enrichments don't apply to digital twin change events (part of the IoT Plug and Play public preview).

## Pricing

Message enrichments are available for no additional charge. Currently, you are charged when you send a message to an IoT Hub. You are only charged once for that message, even if the message goes to multiple endpoints.

# IoT Hub Quotas and Throttling

Each Azure subscription can have at most 50 IoT hubs, and at most 1 Free hub.

Each IoT hub is provisioned with a certain number of units in a specific tier. The tier and number of units determine the maximum daily quota of messages that you can send. The message size used to calculate the daily quota is 0.5 KB for a free tier hub and 4KB for all other tiers.

The tier also determines the throttling limits that IoT Hub enforces on all operations.

## Operation Throttles

Operation throttles are rate limitations that are applied in minute ranges and are intended to prevent abuse. They're also subject to traffic shaping.

The following table shows the enforced throttles. Values refer to an individual hub.

Throttle	Free, B1, and S1	B2 and S2	B3 and S3
Identity registry operations (create, retrieve, list, update, delete)	1.67/sec/unit (100/min/unit)	1.67/sec/unit (100/min/unit)	83.33/sec/unit (5,000/min/unit)
New device connections (this limit applies to the rate of new connections, not the total number of connections)	Higher of 100/sec or 12/sec/unit  For example, two S1 units are 2x12 = 24 new connections/sec, but you have at least 100 new connections/sec across your units. With nine S1 units, you have 108 new connections/sec (9x12) across your units.	120 new connections/sec/unit	6,000 new connections/sec/unit
Device-to-cloud sends	Higher of 100 send operations/sec or 12 send operations/sec/unit  For example, two S1 units are 2x12 = 24/sec, but you have at least 100 send operations/sec across your units. With nine S1 units, you have 108 send operations/sec (9x12) across your units.	120 send operations/sec/unit	6,000 send operations/sec/unit
Cloud-to-device sends*	1.67 send operations/sec/unit (100 messages/min/unit)	1.67 send operations/sec/unit (100 send operations/min/unit)	83.33 send operations/sec/unit (5,000 send operations/min/unit)
Cloud-to-device receives* (only when device uses HTTPS)	16.67 receive operations/sec/unit (1,000 receive operations/min/unit)	16.67 receive operations/sec/unit (1,000 receive operations/min/unit)	833.33 receive operations/sec/unit (50,000 receive operations/min/unit)

Throttle	Free, B1, and S1	B2 and S2	B3 and S3
File upload	1.67 file upload initiations/sec/unit (100/min/unit)	1.67 file upload initiations/sec/unit (100/min/unit)	83.33 file upload initiations/sec/unit (5,000/min/unit)
Direct methods*	160KB/sec/unit**	480KB/sec/unit**	24MB/sec/unit**
Queries	20/min/unit	20/min/unit	1,000/min/unit
Twin (device and module) reads*	100/sec	Higher of 100/sec or 10/sec/unit	500/sec/unit
Twin updates (device and module)*	50/sec	Higher of 50/sec or 5/sec/unit	250/sec/unit
Jobs operations* (create, update, list, delete)	1.67/sec/unit (100/min/unit)	1.67/sec/unit (100/min/unit)	83.33/sec/unit (5,000/min/unit)
Jobs device operations* (update twin, invoke direct method)	10/sec	Higher of 10/sec or 1/sec/unit	50/sec/unit
Configurations and edge deployments* (create, update, list, delete)	0.33/sec/unit (20/min/unit)	0.33/sec/unit (20/min/unit)	0.33/sec/unit (20/min/unit)
Device stream initiation rate*	5 new streams/sec	5 new streams/sec	5 new streams/sec
Maximum number of concurrently connected device streams*	50	50	50
Maximum device stream data transfer* (aggregate volume per day)	300 MB	300 MB	300 MB

\* This feature is not available in the basic tier of IoT Hub.

\*\* Throttling meter size is 4 KB.

## Throttling Details

- The meter size determines at what increments your throttling limit is consumed. If your direct call's payload is between 0 and 4 KB, it is counted as 4 KB. You can make up to 40 calls per second per unit before hitting the limit of 160 KB/sec/unit.
 

Similarly, if your payload is between 4 KB and 8 KB, each call accounts for 8 KB and you can make up to 20 calls per second per unit before hitting the max limit.

Finally, if your payload size is between 156KB and 160 KB, you'll be able to make only 1 call per second per unit in your hub before hitting the limit of 160 KB/sec/unit.
- For Jobs device operations (update twin, invoke direct method) for tier S2, 50/sec/unit only applies to when you invoke methods using jobs. If you invoke direct methods directly, the original throttling limit of 24 MB/sec/unit (for S2) applies.
- Quota is the aggregate number of messages you can send in your hub per day. You can find your hub's quota limit under the column Total number of messages /day on the IoT Hub pricing page.

- Your cloud-to-device and device-to-cloud throttles determine the maximum rate at which you can send messages – number of messages irrespective of 4 KB chunks. Each message can be up to 256 KB which is the maximum message size.
- It's a good practice to throttle your calls so that you don't hit/exceed the throttling limits. If you do hit the limit, IoT Hub responds with error code 429 and the client should back-off and retry. These limits are per hub (or in some cases per hub/unit). For more information, refer to Manage connectivity and reliable messaging/Retry patterns.

## Traffic shaping

To accommodate burst traffic, IoT Hub accepts requests above the throttle for a limited time. The first few of these requests are processed immediately. However, if the number of requests continues violate the throttle, IoT Hub starts placing the requests in a queue and processed at the limit rate. This effect is called traffic shaping. Furthermore, the size of this queue is limited. If the throttle violation continues, eventually the queue fills up, and IoT Hub starts rejecting requests with 429 ThrottlingException.

For example, you use a simulated device to send 200 device-to-cloud messages per second to your S1 IoT Hub (which has a limit of 100/sec D2C sends). For the first minute or two, the messages are processed immediately. However, since the device continues to send more messages than the throttle limit, IoT Hub begins to only process 100 messages per second and puts the rest in a queue. You start noticing increased latency. Eventually, you start getting 429 ThrottlingException as the queue fills up, and the "number of throttle errors" in IoT Hub's metrics starts increasing.

## Identity registry operations throttle

Device identity registry operations are intended for run-time use in device management and provisioning scenarios. Reading or updating a large number of device identities is supported through import and export jobs.

## Device connections throttle

The device connections throttle governs the rate at which new device connections can be established with an IoT hub. The device connections throttle does not govern the maximum number of simultaneously connected devices. The device connections rate throttle depends on the number of units that are provisioned for the IoT hub.

For example, if you buy a single S1 unit, you get a throttle of 100 connections per second. Therefore, to connect 100,000 devices, it takes at least 1,000 seconds (approximately 16 minutes). However, you can have as many simultaneously connected devices as you have devices registered in your identity registry.

## Other limits

IoT Hub enforces other operational limits:

Operation	Limit
Devices	The total number of devices plus modules that can be registered to a single IoT hub is capped at 1,000,000. The only way to increase this limit is to contact Microsoft Support.
File uploads	10 concurrent file uploads per device.

Operation	Limit
Jobs*	Maximum concurrent jobs is 1 (for Free and S1), 5 (for S2), and 10 (for S3). However, the max concurrent device import/export jobs is 1 for all tiers. Job history is retained up to 30 days.
Additional endpoints	Paid SKU hubs may have 10 additional endpoints. Free SKU hubs may have one additional endpoint.
Message routing queries	Paid SKU hubs may have 100 routing queries. Free SKU hubs may have five routing queries.
Message enrichments	Paid SKU hubs can have up to 10 message enrichments. Free SKU hubs can have up to 2 message enrichments.
Device-to-cloud messaging	Maximum message size 256 KB
Cloud-to-device messaging*	Maximum message size 64 KB. Maximum pending messages for delivery is 50 per device.
Direct method*	Maximum direct method payload size is 128 KB.
Automatic device configurations*	100 configurations per paid SKU hub. 20 configurations per free SKU hub.
IoT Edge automatic deployments*	20 modules per deployment. 100 deployments per paid SKU hub. 10 deployments per free SKU hub.
Twins*	Maximum size per twin section (tags, desired properties, reported properties) is 8 KB

\* This feature is not available in the basic tier of IoT Hub.

## Increasing the quota or throttle limit

At any given time, you can increase quotas or throttle limits by increasing the number of provisioned units in an IoT hub.

## Latency

IoT Hub strives to provide low latency for all operations. However, due to network conditions and other unpredictable factors it cannot guarantee a certain latency. When designing your solution, you should:

- Avoid making any assumptions about the maximum latency of any IoT Hub operation.
- Provision your IoT hub in the Azure region closest to your devices.
- Consider using Azure IoT Edge to perform latency-sensitive operations on the device or on a gateway close to the device.
- Multiple IoT Hub units affect throttling as described previously, but do not provide any additional latency benefits or guarantees.

# Data Storage and the Lambda Architecture

## Introduction to Lambda Architecture

There are several distinct purposes of recording telemetry readings generated by IoT devices:

- To be analyzed for anomalies, for preventive maintenance.
- For visualization by a remote human operator, to help in decision making.
- To be archived, perhaps for later analysis.

These distinct purposes have conflicting storage requirements. However, having conflicting goals doesn't need to be a bad thing. Conflicting goals for data storage lead us to hybrid systems, which can be very flexible and powerful.

This lesson describes the hybrid nature of the IoT lambda architecture and provides an introduction to data storage option for Azure IoT solutions.

### Data paths

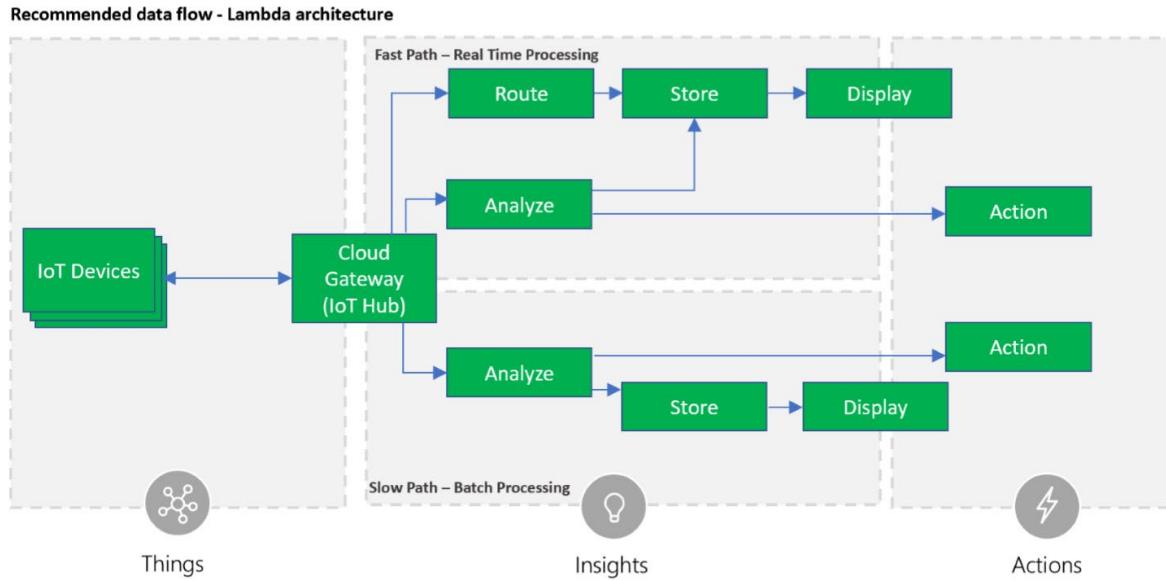
The conflict in Azure IoT is as follows. Telemetry data is coming in hot, there's lots of it, and it needs to be analyzed quickly. Preventive maintenance is the goal of this analysis. Also, all the data should be stored, both to archive it, and to run some deeper analysis over longer time periods. The deeper analysis is to try to detect longer term trends, or failure patterns, that might be difficult to detect with a shorter real-time sample.

One of the easiest ways of handling this duality at the device sensor end of things, is to send two messages. The first message contains only the telemetry data that needs analyzed in real time. The second message contains the telemetry, and all the other data that might be needed for deeper analysis or archiving.

The IoT Hub routes these two messages to different resources. It's common to use the familiar terms *hot*, *warm*, *cool*, and *cold* in data analysis. Hot clearly means a real-time approach is needed. Warm can have the same meaning, though perhaps the data is "near" real time, or at least, recent. Cool means the flow of data is slow. Cold means that the data is stored and not "flowing".

### Understand lambda architecture

The *Lambda architecture* of Azure IoT enables multiple paths. However, for the sake of explanation, let's work with two paths, hot and cold.



The hot path is the streaming telemetry routed into real-time analysis. This path is also the right path to trigger warnings and alerts.

The cold path is a batch processing path for telemetry data storage.

## The hot path

The IoT remote device pumps out *specific* telemetry. This telemetry is sent in its own message, routed by the IoT Hub for instant analysis and visualization. The analysis could be done by a human operator, say, using Azure Time Series Insights. This approach is described in this module.

Alternatively, the analysis could be handled by Azure ML models, via Azure Stream Analytics. This scenario is more complex, involves coding, and is described in other IoT Hub Learn modules.

## The cold path

The IoT remote device also sends out all telemetry, and logging, data. The IoT Hub directs these messages down a route to an Azure storage account. There are various storage resources available in Azure, and the next units describe these options.

## Issues with lambda architecture

Similar to most hybrid systems, there are issues. One of the main ones with IoT is the duplication of data and code. The more duplication there is, the greater the chance of an unwanted divergence between the duplicate copies. Developers of the IoT device sensor code need to ensure that the telemetry data being sent in the two messages is identical, where it should be. There may be code duplication in the analysis apps for the hot and cold paths. Duplication needs to be handled carefully, though is a near unavoidable consequence of a hybrid system.

# Common Storage Options for Azure IoT Solutions

There are several storage options available in Azure for IoT solutions.

## Azure Storage

Azure Storage is Microsoft's cloud storage solution for modern data storage scenarios. Azure Storage offers a massively scalable object store for data objects, a file system service for the cloud, a messaging store for reliable messaging, and a NoSQL store. Azure Storage is:

- Durable and highly available. Redundancy ensures that your data is safe in the event of transient hardware failures. You can also opt to replicate data across datacenters or geographical regions for additional protection from local catastrophe or natural disaster. Data replicated in this way remains highly available in the event of an unexpected outage.
- Secure. All data written to Azure Storage is encrypted by the service. Azure Storage provides you with fine-grained control over who has access to your data.
- Scalable. Azure Storage is designed to be massively scalable to meet the data storage and performance needs of today's applications.
- Managed. Microsoft Azure handles hardware maintenance, updates, and critical issues for you.
- Accessible. Data in Azure Storage is accessible from anywhere in the world over HTTP or HTTPS. Microsoft provides client libraries for Azure Storage in a variety of languages, including .NET, Java, Node.js, Python, PHP, Ruby, Go, and others, as well as a mature REST API. Azure Storage supports scripting in Azure PowerShell or Azure CLI. And the Azure portal and Azure Storage Explorer offer easy visual solutions for working with your data.

## Azure Storage Options

The storage options in Azure that are often used in IoT solutions include the following:

- Azure Storage services
- Azure Data Lake Gen2
- Azure Cosmos DB
- Azure SQL Database

# Azure Storage Options

## Azure Blob storage

Azure Blob storage is Microsoft's object storage solution for the cloud. Blob storage is optimized for storing massive amounts of unstructured data, such as text or binary data.

Blob storage is ideal for:

- Serving images or documents directly to a browser.
- Storing files for distributed access.
- Streaming video and audio.

- Storing data for backup and restore, disaster recovery, and archiving.
- Storing data for analysis by an on-premises or Azure-hosted service.

Objects in Blob storage can be accessed from anywhere in the world via HTTP or HTTPS. Users or client applications can access blobs via URLs, the Azure Storage REST API, Azure PowerShell, Azure CLI, or an Azure Storage client library. The storage client libraries are available for multiple languages, including .NET, Java, Node.js, Python, PHP, and Ruby.

## Additional Azure Storage Options

### Azure Files

Azure Files enables you to set up highly available network file shares that can be accessed by using the standard Server Message Block (SMB) protocol. That means that multiple VMs can share the same files with both read and write access. You can also read the files using the REST interface or the storage client libraries.

One thing that distinguishes Azure Files from files on a corporate file share is that you can access the files from anywhere in the world using a URL that points to the file and includes a shared access signature (SAS) token. You can generate SAS tokens; they allow specific access to a private asset for a specific amount of time.

File shares can be used for many common scenarios:

- Many on-premises applications use file shares. This feature makes it easier to migrate those applications that share data to Azure. If you mount the file share to the same drive letter that the on-premises application uses, the part of your application that accesses the file share should work with minimal, if any, changes.
- Configuration files can be stored on a file share and accessed from multiple VMs. Tools and utilities used by multiple developers in a group can be stored on a file share, ensuring that everybody can find them, and that they use the same version.
- Diagnostic logs, metrics, and crash dumps are just three examples of data that can be written to a file share and processed or analyzed later.

At this time, Active Directory-based authentication and access control lists (ACLs) are not supported, but they will be at some time in the future. The storage account credentials are used to provide authentication for access to the file share. This means anybody with the share mounted will have full read/write access to the share.

### Azure Queue storage

The Azure Queue service is used to store and retrieve messages. Queue messages can be up to 64 KB in size, and a queue can contain millions of messages. Queues are generally used to store lists of messages to be processed asynchronously.

For example, say you want your customers to be able to upload pictures, and you want to create thumbnails for each picture. You could have your customer wait for you to create the thumbnails while uploading the pictures. An alternative would be to use a queue. When the customer finishes their upload, write a message to the queue. Then have an Azure Function retrieve the message from the queue and create the thumbnails. Each of the parts of this processing can be scaled separately, giving you more control when tuning it for your usage.

## Azure Table storage

Azure Table storage is now part of Azure Cosmos DB. To see Azure Table storage documentation, see the Azure Table Storage Overview. In addition to the existing Azure Table storage service, there is a new Azure Cosmos DB Table API offering that provides throughput-optimized tables, global distribution, and automatic secondary indexes. To learn more and try out the new premium experience, please check out Azure Cosmos DB Table API.

## Types of storage accounts

Azure Storage offers several types of storage accounts. Each type supports different features and has its own pricing model. Consider these differences before you create a storage account to determine the type of account that is best for your applications. The types of storage accounts are:

- General-purpose v2 accounts: Basic storage account type for blobs, files, queues, and tables. Recommended for most scenarios using Azure Storage.
- General-purpose v1 accounts: Legacy account type for blobs, files, queues, and tables. Use general-purpose v2 accounts instead when possible.
- BlockBlobStorage accounts: Blob-only storage accounts with premium performance characteristics. Recommended for scenarios with high transaction rates, using smaller objects, or requiring consistently low storage latency.
- FileStorage accounts: Files-only storage accounts with premium performance characteristics. Recommended for enterprise or high performance scale applications.
- BlobStorage accounts: Legacy Blob-only storage accounts. Use general-purpose v2 accounts instead when possible.

## Securing access to storage accounts

Every request to Azure Storage must be authorized. Azure Storage supports the following authorization methods:

- Azure Active Directory (Azure AD) integration for blob and queue data. Azure Storage supports authentication and authorization with Azure AD for the Blob and Queue services via role-based access control (RBAC). Authorizing requests with Azure AD is recommended for superior security and ease of use. For more information, see [Authorize access to Azure blobs and queues using Azure Active Directory](#).
- Azure AD authorization over SMB for Azure Files (preview). Azure Files supports identity-based authorization over SMB (Server Message Block) through Azure Active Directory Domain Services. Your domain-joined Windows virtual machines (VMs) can access Azure file shares using Azure AD credentials. For more information, see [Overview of Azure Active Directory authorization over SMB for Azure Files \(preview\)](#).
- Authorization with Shared Key. The Azure Storage Blob, Queue, and Table services and Azure Files support authorization with Shared Key. A client using Shared Key authorization passes a header with every request that is signed using the storage account access key. For more information, see [Authorize with Shared Key](#).
- Authorization using shared access signatures (SAS). A shared access signature (SAS) is a string containing a security token that can be appended to the URI for a storage resource. The security token encapsulates constraints such as permissions and the interval of access. For more information, refer to [Using Shared Access Signatures \(SAS\)](#).

- Anonymous access to containers and blobs. A container and its blobs may be publicly available. When you specify that a container or blob is public, anyone can read it anonymously; no authentication is required. For more information, see [Manage anonymous read access to containers and blobs](#)

## Azure Data Lake Gen 2

Azure Data Lake Storage Gen2 is a set of capabilities dedicated to big data analytics, built on Azure Blob storage. Data Lake Storage Gen2 is the result of converging the capabilities of our two existing storage services, Azure Blob storage and Azure Data Lake Storage Gen1. Features from Azure Data Lake Storage Gen1, such as file system semantics, directory, and file level security and scale are combined with low-cost, tiered storage, high availability/disaster recovery capabilities from Azure Blob storage.

### Designed for enterprise big data analytics

Data Lake Storage Gen2 makes Azure Storage the foundation for building enterprise data lakes on Azure. Designed from the start to service multiple petabytes of information while sustaining hundreds of gigabits of throughput, Data Lake Storage Gen2 allows you to easily manage massive amounts of data.

A fundamental part of Data Lake Storage Gen2 is the addition of a hierarchical namespace to Blob storage. The hierarchical namespace organizes objects/files into a hierarchy of directories for efficient data access. A common object store naming convention uses slashes in the name to mimic a hierarchical directory structure. This structure becomes real with Data Lake Storage Gen2. Operations such as renaming or deleting a directory become single atomic metadata operations on the directory rather than enumerating and processing all objects that share the name prefix of the directory.

In the past, cloud-based analytics had to compromise in areas of performance, management, and security. Data Lake Storage Gen2 addresses each of these aspects in the following ways:

- Performance is optimized because you do not need to copy or transform data as a prerequisite for analysis. The hierarchical namespace greatly improves the performance of directory management operations, which improves overall job performance.
- Management is easier because you can organize and manipulate files through directories and subdirectories.
- Security is enforceable because you can define POSIX permissions on directories or individual files.
- Cost effectiveness is made possible as Data Lake Storage Gen2 is built on top of the low-cost Azure Blob storage. The additional features further lower the total cost of ownership for running big data analytics on Azure.

### Key features of Data Lake Storage Gen2

- Hadoop compatible access: Data Lake Storage Gen2 allows you to manage and access data just as you would with a Hadoop Distributed File System (HDFS). The new ABFS driver is available within all Apache Hadoop environments, including Azure HDInsight, Azure Databricks, and SQL Data Warehouse to access data stored in Data Lake Storage Gen2.
- A superset of POSIX permissions: The security model for Data Lake Gen2 supports ACL and POSIX permissions along with some extra granularity specific to Data Lake Storage Gen2. Settings may be configured through Storage Explorer or through frameworks like Hive and Spark.
- Cost effective: Data Lake Storage Gen2 offers low-cost storage capacity and transactions. As data transitions through its complete lifecycle, billing rates change keeping costs to a minimum via built-in features such as Azure Blob storage lifecycle.

- Optimized driver: The ABFS driver is optimized specifically for big data analytics. The corresponding REST APIs are surfaced through the endpoint `dfs.core.windows.net`.

## Azure Cosmos DB

Today's applications are required to be highly responsive and always online. To achieve low latency and high availability, instances of these applications need to be deployed in datacenters that are close to their users. Applications need to respond in real time to large changes in usage at peak hours, store ever increasing volumes of data, and make this data available to users in milliseconds.

### Key Benefits

- Azure Cosmos DB natively partitions your data for high availability and scalability. Azure Cosmos DB offers 99.99% guarantees for availability, throughput, low latency, and consistency on all single-region accounts and all multi-region accounts with relaxed consistency, and 99.999% read availability on all multi-region database accounts.
- Azure Cosmos DB has SSD backed storage with low-latency order-of-millisecond response times.
- Azure Cosmos DB's support for consistency levels like eventual, consistent prefix, session, and bounded-staleness allows for full flexibility and low cost-to-performance ratio. No database service offers as much flexibility as Azure Cosmos DB in levels consistency.
- Azure Cosmos DB has a flexible data-friendly pricing model that meters storage and throughput independently.
- Azure Cosmos DB's reserved throughput model allows you to think in terms of number of reads/writes instead of CPU/memory/IOPs of the underlying hardware.
- Azure Cosmos DB's design lets you scale to massive request volumes in the order of trillions of requests per day.

## Cosmos DB Scenarios for IoT

### Scenario 1

IoT use cases commonly share some patterns in how they ingest, process, and store data. First, these systems need to ingest bursts of data from device sensors of various locales. Next, these systems process and analyze streaming data to derive real-time insights. The data is then archived to cold storage for batch analytics. Microsoft Azure offers rich services that can be applied for IoT use cases including Azure Cosmos DB, Azure Event Hubs, Azure Stream Analytics, Azure Notification Hub, Azure Machine Learning, Azure HDInsight, and Power BI.

Bursts of data can be ingested by Azure IoT Hub as it offers high throughput data ingestion with low latency. Data ingested that needs to be processed for real-time insight can be funneled to Azure Stream Analytics for real-time analytics. Data can be loaded into Azure Cosmos DB for adhoc querying. Once the data is loaded into Azure Cosmos DB, the data is ready to be queried. In addition, new data and changes to existing data can be read on change feed. Change feed is a persistent, append only log that stores changes to Cosmos containers in sequential order. The all data or just changes to data in Azure Cosmos DB can be used as reference data as part of real-time analytics. In addition, data can further be refined and processed by connecting Azure Cosmos DB data to HDInsight for Pig, Hive, or Map/Reduce jobs. Refined data is then loaded back to Azure Cosmos DB for reporting.

## Scenario 2

The data generated from increasing network sensors brings an unprecedented visibility into previously opaque systems and processes. The key is to find actionable insights in this torrent of information regardless of where IoT devices are distributed around the globe. Azure Cosmos DB allows IOT companies to analyze high-velocity sensor, and time-series data in real-time anywhere around the world. It allows you to harness the true value of an interconnected world to deliver improved customer experiences, operational efficiencies, and new revenue opportunities.

## Azure SQL Database

Azure SQL Database is a fully managed Platform as a Service (PaaS) Database Engine that handles most of the database management functions such as upgrading, patching, backups, and monitoring without user involvement. Azure SQL Database is always running on the latest stable version of SQL Server Database Engine and patched OS with 99.99% availability. PaaS capabilities that are built-in into Azure SQL database enable you to focus on the domain specific database administration and optimization activities that are critical for your business.

### Key Features:

- **Business Continuity:** Business continuity in Azure SQL Database refers to the mechanisms, policies, and procedures that enable your business to continue operating in the face of disruption, particularly to its computing infrastructure. In the most of the cases, Azure SQL Database will handle the disruptive events that might happen in the cloud environment and keep your applications and business processes running.
- **High Availability:** The goal of the High Availability architecture in Azure SQL Database is to guarantee that your database is up and running 99.99% of time, without worrying about the impact of maintenance operations and outages. Azure automatically handles critical servicing tasks, such as patching, backups, Windows and SQL upgrades, as well as unplanned events such as underlying hardware, software, or network failures. When the underlying SQL instance is patched or fails over, the downtime is not noticeable if you employ retry logic in your app. Azure SQL Database can quickly recover even in the most critical circumstances ensuring that your data is always available.
- **Automated Backups:** SQL Database automatically creates the database backups that are kept between 7 and 35 days, and uses Azure read-access geo-redundant storage (RA-GRS) to ensure that they are preserved even if the data center is unavailable. These backups are created automatically. Database backups are an essential part of any business continuity and disaster recovery strategy because they protect your data from accidental corruption or deletion.
- **Long Term Backup Retention:** Many applications have regulatory, compliance, or other business purposes that require you to retain database backups beyond the 7-35 days provided by Azure SQL Database automatic backups. By using the long-term retention (LTR) feature, you can store specified SQL database full backups in RA-GRS blob storage for up to 10 years. You can then restore any backup as a new database.
- **Geo-Replication:** Auto-failover groups is a SQL Database feature that allows you to manage replication and failover of a group of databases on a SQL Database server or all databases in a managed instance to another region. It is a declarative abstraction on top of the existing active geo-replication feature, designed to simplify deployment and management of geo-replicated databases at scale. You can initiate failover manually or you can delegate it to the SQL Database service based on a user-defined policy. The latter option allows you to automatically recover multiple related databases in a secondary region after a catastrophic failure or other unplanned event that results in full or partial loss

- of the SQL Database service's availability in the primary region. A failover group can include one or multiple databases, typically used by the same application. Additionally, you can use the readable secondary databases to offload read-only query workloads. Because auto-failover groups involve multiple databases, these databases must be configured on the primary server. Auto-failover groups support replication of all databases in the group to only one secondary server in a different region.
- Scale Resources: Azure SQL Database enables you to dynamically add more resources to your database with minimal downtime.

# Azure Functions and Stream Analytics

## Azure Stream Analytics

### What is Azure Stream Analytics?

Azure Stream Analytics is a real-time analytics and complex event-processing engine that is designed to analyze and process high volumes of fast streaming data from multiple sources simultaneously. Patterns and relationships can be identified in information extracted from a number of input sources including devices, sensors, clickstreams, social media feeds, and applications. These patterns can be used to trigger actions and initiate workflows such creating alerts, feeding information to a reporting tool, or storing transformed data for later use. Also, Stream Analytics is available on Azure IoT Edge runtime, and supports the same exact language or syntax as cloud.

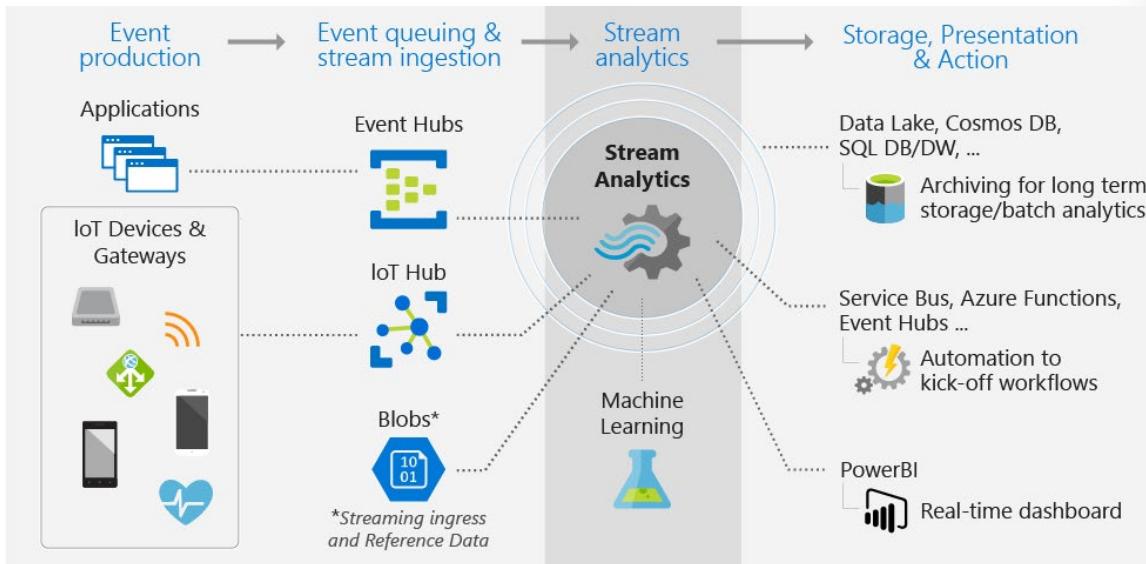
### How does Stream Analytics work?

An Azure Stream Analytics *job* consists of an input, query, and an output. Stream Analytics ingests data from Azure Event Hubs, Azure IoT Hub, or Azure Blob Storage. The query, which is based on SQL query language, can be used to easily filter, sort, aggregate, and join streaming data over a period of time. You can also extend this SQL language with JavaScript and C# user defined functions (UDFs). You can easily adjust the event ordering options and duration of time windows when performing aggregation operations through simple language constructs and/or configurations.

Each job has an output for the transformed data, and you can control what happens in response to the information you've analyzed. For example, you can:

- Send data to services such as Azure Functions, Service Bus Topics or Queues to trigger communications or custom workflows downstream.
- Send data to a Power BI dashboard for real-time dashboarding.
- Store data in other Azure storage services to train a machine learning model based on historical data or perform batch analytics.

The following image shows how data is sent to Stream Analytics, analyzed, and sent for other actions like storage or presentation:



## Key capabilities and benefits

Azure Stream Analytics is designed to be easy to use, flexible, reliable, and scalable to any job size.

## Ease of getting started

Azure Stream Analytics is easy to start. It only takes a few clicks to connect to multiple sources and sinks, creating an end-to-end pipeline. Stream Analytics can connect to Azure Event Hubs and Azure IoT Hub for streaming data ingestion, as well as Azure Blob storage to ingest historical data. Job input can also include static or slow-changing reference data from Azure Blob storage or SQL Database that you can join to streaming data to perform lookup operations.

Stream Analytics can route job output to many storage systems such as Azure Blob storage, Azure SQL Database, Azure Data Lake Store, and Azure CosmosDB. You can run batch analytics on stored output with Azure HDInsight, or you can send the output to another service, like Event Hubs for consumption or Power BI for real-time visualization.

## Programmer productivity

Azure Stream Analytics uses a simple SQL-based query language that has been augmented with powerful temporal constraints to analyze data in motion. To define job transformations, you use a simple, declarative Stream Analytics query language that lets you author complex temporal queries and analytics using simple SQL constructs. Because Stream Analytics query language is consistent to the SQL language, familiarity with SQL is sufficient to start creating jobs. You can also create jobs by using developer tools like Azure PowerShell, Stream Analytics Visual Studio tools, the Stream Analytics Visual Studio Code extension, or Azure Resource Manager templates. Using developer tools allow you to develop transformation queries offline and use the CI/CD pipeline to submit jobs to Azure.

The Stream Analytics query language offers a wide array of functions for analyzing and processing streaming data. This query language supports simple data manipulation, aggregation functions, and complex geospatial functions. You can edit queries in the portal and test them using sample data that is extracted from a live stream.

You can extend the capabilities of the query language by defining and invoking additional functions. You can define function calls in the Azure Machine Learning to take advantage of Azure Machine Learning solutions, and integrate JavaScript or C# user-defined functions (UDFs) or user-defined aggregates to perform complex calculations as part a Stream Analytics query.

## Fully managed

Azure Stream Analytics is a fully managed serverless (PaaS) offering on Azure. You don't have to provision any hardware or manage clusters to run your jobs. Azure Stream Analytics fully manages your job by setting up complex compute clusters in the cloud and taking care of the performance tuning necessary to run the job. Integration with Azure Event Hubs and Azure IoT Hub allows your job to ingest millions of events per second coming from a number of sources, to include connected devices, clickstreams, and log files. Using the partitioning feature of Event Hubs, you can partition computations into logical steps, each with the ability to be further partitioned to increase scalability.

Run in the cloud or on the intelligent edge

Azure Stream Analytics can run in the cloud, for large-scale analytics, or run on IoT Edge for ultra-low latency analytics. Azure Stream Analytics uses the same query language on both cloud and the edge, enabling developers to build truly hybrid architectures for stream processing.

## Low total cost of ownership

As a cloud service, Stream Analytics is optimized for cost. There are no upfront costs involved - you only pay for the streaming units you consume, and the amount of data processed. There is no commitment or cluster provisioning required, and you can scale the job up or down based on your business needs.

## Mission-critical ready

Azure Stream Analytics is available across multiple regions worldwide and is designed to run mission-critical workloads by supporting reliability, security and compliance requirements.

## Reliability

Azure Stream Analytics guarantees exactly-once event processing and at-least-once delivery of events, so events are never lost. Exactly-once processing is guaranteed with selected output as described in Event Delivery Guarantees.

Azure Stream Analytics has built-in recovery capabilities in case the delivery of an event fails. Stream Analytics also provides built-in checkpoints to maintain the state of your job and provides repeatable results.

As a managed service, Stream Analytics guarantees event processing with a 99.9% availability at a minute level of granularity. For more information, see the Stream Analytics SLA page.

## Security

In terms of security, Azure Stream Analytics encrypts all incoming and outgoing communications and supports TLS 1.2. Built-in checkpoints are also encrypted. Stream Analytics doesn't store the incoming data since all processing is done in-memory.

## Compliance

Azure Stream Analytics follows multiple compliance certifications as described in the overview of Azure compliance.

## Performance

Stream Analytics can process millions of events every second and it can deliver results with ultra low latencies. It allows you to scale-up and scale-out to handle large real-time and complex event processing applications. Stream Analytics supports higher performance by partitioning, allowing complex queries to be parallelized and executed on multiple streaming nodes. Azure Stream Analytics is built on Trill, a high-performance in-memory streaming analytics engine developed in collaboration with Microsoft Research.

## Azure Stream Analytics and Other Stream Processing Technologies

There are several services available for real-time analytics and streaming processing on Azure.

## When to use Azure Stream Analytics

Azure Stream Analytics is the recommended service for stream analytics on Azure. It's meant for a wide range of scenarios that include but aren't limited to:

- Dashboards for data visualization
- Real-time **alerts**<sup>11</sup> from temporal and spatial patterns or anomalies
- Extract, Transform, Load (ETL)
- **Event Sourcing pattern**<sup>12</sup>
- **IoT Edge**<sup>13</sup>

Adding an Azure Stream Analytics job to your application is the fastest way to get streaming analytics up and running in Azure, using the SQL language you already know. Azure Stream Analytics is a job service, so you don't have to spend time managing clusters, and you don't have to worry about downtime with a 99.9% SLA at the job level. Billing is also done at the job level making startup costs low (one Streaming Unit), but scalable (up to 192 Streaming Units). It's much more cost effective to run a few Stream Analytics jobs than it is to run and maintain a cluster.

Azure Stream Analytics has a rich out-of-the-box experience. You can immediately take advantage of the following features without any additional setup:

- Built-in temporal operators, such as **windowed aggregates**<sup>14</sup>, temporal joins, and temporal analytic functions.
- Native Azure **input**<sup>15</sup> and **output**<sup>16</sup> adapters

<sup>11</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-set-up-alerts>

<sup>12</sup> <https://docs.microsoft.com/en-us/azure/architecture/patterns/event-sourcing>

<sup>13</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-edge>

<sup>14</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-window-functions>

<sup>15</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-add-inputs>

<sup>16</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-define-outputs>

- Support for slow changing **reference data**<sup>17</sup> (also known as a lookup tables), including joining with geospatial reference data for geofencing.
- Integrated solutions, such as **Anomaly Detection**<sup>18</sup>
- Multiple time windows in the same query
- Ability to compose multiple temporal operators in arbitrary sequences.
- Under 100-ms end-to-end latency from input arriving at Event Hubs, to output landing in Event Hubs, including the network delay from and to Event Hubs, at sustained high throughput

## When to use other technologies

### You need to input from or output to Kafka

Azure Stream Analytics doesn't have an Apache Kafka input or output adapter. If you have events landing in or need to send to Kafka and you don't have a requirement to run your own Kafka cluster, you can continue to use Stream Analytics by sending events to Event Hubs using the Event Hubs Kafka API without changing the event sender. If you do need to run your own Kafka cluster, you can use Spark Structured Streaming, which is fully supported on Azure Databricks, or Storm on Azure HDInsight.

### You want to write UDFs, UDAs, and custom deserializers in a language other than JavaScript or C#

Azure Stream Analytics supports user-defined functions (UDF) or user-defined aggregates (UDA) in JavaScript for cloud jobs and C# for IoT Edge jobs. C# user-defined deserializers are also supported. If you want to implement a deserializer, a UDF, or a UDA in other languages, such as Java or Python, you can use Spark Structured Streaming. You can also run the Event Hubs `EventProcessorHost` on your own virtual machines to do arbitrary streaming processing.

### Your solution is in a multi-cloud or on-premises environment

Azure Stream Analytics is Microsoft's proprietary technology and is only available on Azure. If you need your solution to be portable across Clouds or on-premises, consider open-source technologies such as Spark Structured Streaming or Storm.

## Some Common ASA Patterns and Tools

### ASA Patterns

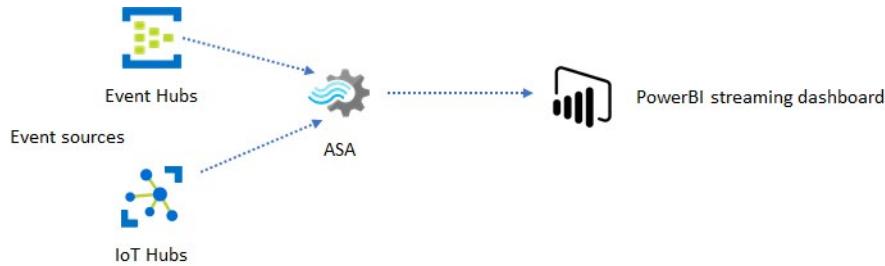
Azure Stream Analytics is key component of the larger IoT solution. A number of simple architectural patterns for Azure Stream Analytics can be used to develop more complex solutions in a wide variety of scenarios.

<sup>17</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-use-reference-data>

<sup>18</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-machine-learning-anomaly-detection>

## Create a Stream Analytics job to power real-time dashboarding experience

With Azure Stream Analytics, you can quickly stand up real-time dashboards and alerts. A simple solution ingests events from Event Hubs or IoT Hub, and feeds the Power BI dashboard with a streaming data set. For more information, see the detailed tutorial [Analyze phone call data with Stream Analytics and visualize results in Power BI dashboard](#).

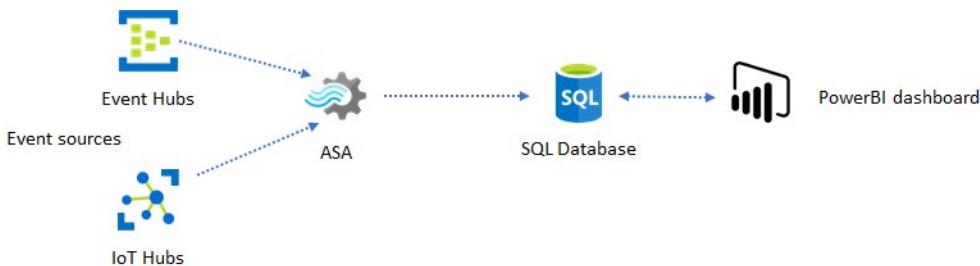


This solution can be built in just a few minutes from Azure portal. There is no extensive coding involved, and SQL language is used to express the business logic.

This solution pattern offers the lowest latency from the event source to the Power BI dashboard in a browser. Azure Stream Analytics is the only Azure service with this built-in capability.

## Use SQL for dashboard

The Power BI dashboard offers low latency, but it cannot be used to produce full fledged Power BI reports. A common reporting pattern is to output your data to a SQL database first. Then use Power BI's SQL connector to query SQL for the latest data.



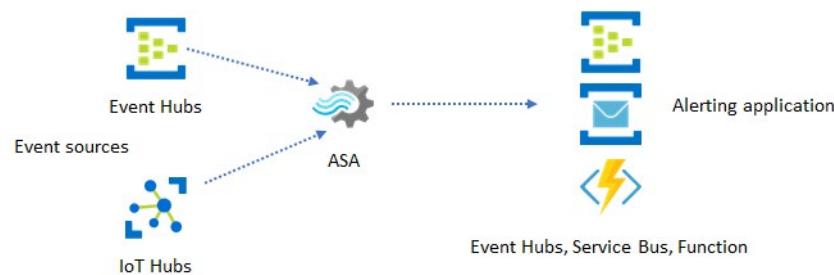
Using SQL database gives you more flexibility but at the expense of a slightly higher latency. This solution is optimal for jobs with latency requirements greater than one second. With this method, you can maximize Power BI capabilities to further slice and dice the data for reports, and much more visualization options. You also gain the flexibility of using other dashboard solutions, such as Tableau.

SQL is not a high throughput data store. The maximum throughput to a SQL database from Azure Stream Analytics is currently around 24 MB/s. If the event sources in your solution produce data at a higher rate, you need to use processing logic in Stream Analytics to reduce the output rate to SQL. Techniques such as filtering, windowed aggregates, pattern matching with temporal joins, and analytic functions can be used. The output rate to SQL can be further optimized using techniques described in Azure Stream Analytics output to Azure SQL Database.

## Incorporate real-time insights into your application with event messaging

The second most popular use of Stream Analytics is to generate real-time alerts. In this solution pattern, business logic in Stream Analytics can be used to detect temporal and spatial patterns or anomalies, then produce alerting signals. However, unlike the dashboard solution where Stream Analytics uses Power BI as a preferred endpoint, a number of intermediate data sinks can be used. These sinks include Event Hubs, Service Bus, and Azure Functions. You, as the application builder, need to decide which data sink works best for your scenario.

Downstream event consumer logic must be implemented to generate alerts in your existing business workflow. Because you can implement custom logic in Azure Functions, Azure Functions is the fastest way you can perform this integration. A tutorial for using Azure Function as the output for a Stream Analytics job can be found in Run Azure Functions from Azure Stream Analytics jobs. Azure Functions also supports various types of notifications including text and email. Logic App may also be used for such integration, with Event Hubs between Stream Analytics and Logic App.

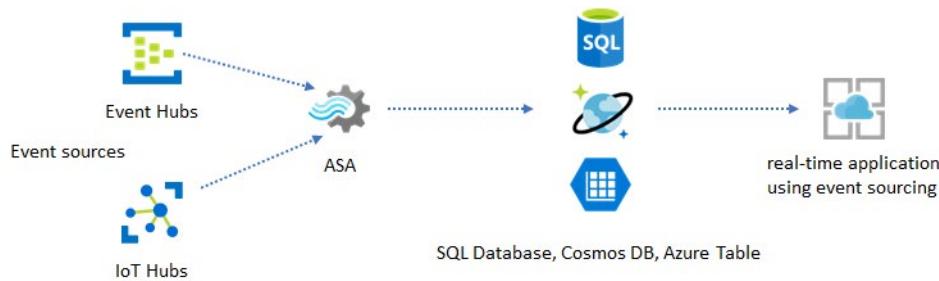


Event Hubs, on the other hand, offers the most flexible integration point. Many other services, like Azure Data Explorer and Time Series Insights can consume events from Event Hubs. Services can be connected directly to the Event Hubs sink from Azure Stream Analytics to complete the solution. Event Hubs is also the highest throughput messaging broker available on Azure for such integration scenarios.

## Incorporate real-time insights into your application through data stores

Most web services and web applications today use a request/response pattern to serve the presentation layer. The request/response pattern is simple to build and can be easily scaled with low response time using a stateless frontend and scalable stores, like Cosmos DB.

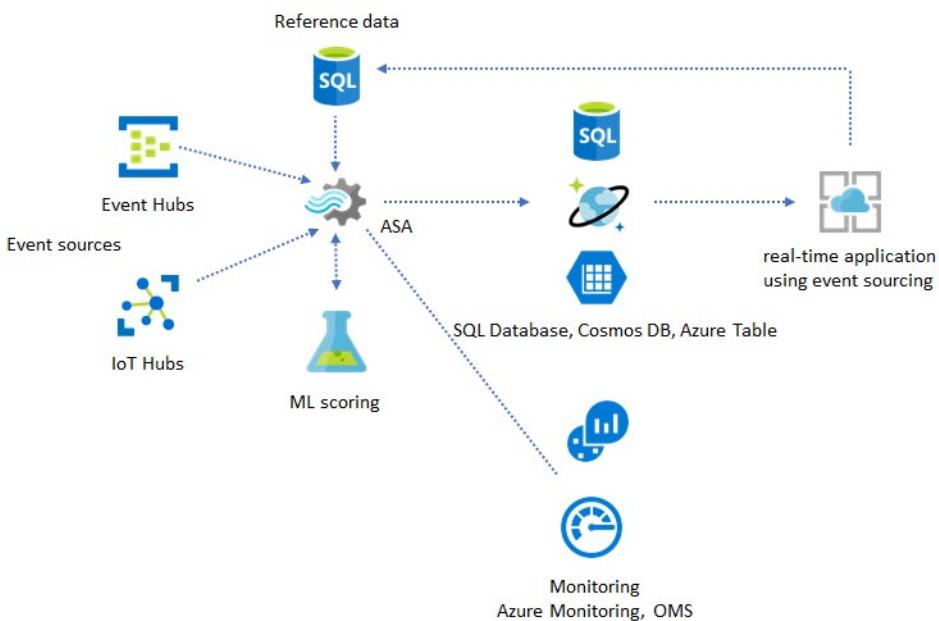
High data volume often creates performance bottlenecks in a CRUD-based system. The event sourcing solution pattern is used to address the performance bottlenecks. Temporal patterns and insights are also difficult and inefficient to extract from a traditional data store. Modern high-volume data driven applications often adopt a dataflow-based architecture. Azure Stream Analytics as the compute engine for data in motion is a linchpin in that architecture.



In this solution pattern, events are processed and aggregated into data stores by Azure Stream Analytics. The application layer interacts with data stores using the traditional request/response pattern. Because of Stream Analytics' ability to process a large number of events in real-time, the application is highly scalable without the need to bulk up the data store layer. The data store layer is essentially a materialized view in the system. Azure Stream Analytics output to Azure Cosmos DB describes how Cosmos DB is used as a Stream Analytics output.

## How to monitor ASA jobs

An Azure Stream Analytics job can be run 24/7 to process incoming events continuously in real time. Its uptime guarantee is crucial to the health of the overall application. While Stream Analytics is the only streaming analytics service in the industry that offers a 99.9% availability guarantee, you may still incur some level of down time. Over the years, Stream Analytics has introduced metrics, logs, and job states to reflect the health of the jobs. All of them are surfaced through Azure Monitor service and can be further exported to OMS.



There are two key things to monitor:

- Job failed state: First and foremost, you need to make sure the job is running. Without the job in the running state, no new metrics or logs are generated. Jobs can change to a failed state for various reasons, including having a high SU utilization level (i.e., running out of resources).

- Watermark delay metrics: This metric reflects how far behind your processing pipeline is in wall clock time (seconds). Some of the delay is attributed to the inherent processing logic. As a result, monitoring the increasing trend is much more important than monitoring the absolute value. The steady state delay should be addressed by your application design, not by monitoring or alerts.

Upon failure, activity logs and diagnostics logs are the best places to begin looking for errors.

## Build resilient and mission critical applications

Regardless of Azure Stream Analytics' SLA guarantee and how careful you run your end-to-end application, outages happen. If your application is mission critical, you need to be prepared for outages in order to recover gracefully.

For alerting applications, the most important thing is to detect the next alert. You may choose to restart the job from the current time when recovering, ignoring past alerts. The job start time semantics are by the first output time, not the first input time. The input is rewound backwards an appropriate amount of time to guarantee the first output at the specified time is complete and correct. You won't get partial aggregates and trigger alerts unexpectedly as a result.

You may also choose to start output from some amount of time in the past. Both Event Hubs and IoT Hub's retention policies hold a reasonable amount of data to allow processing from the past. The tradeoff is how fast you can catch up to the current time and start to generate timely new alerts. Data loses its value rapidly over time, so it's important to catch up to the current time quickly. There are two ways to catch up quickly:

- Provision more resources (SUs) when catching up.
- Restart from current time.

Restarting from current the time is simple to do, with the tradeoff of leaving a gap during processing. Restarting this way might be OK for alerting scenarios, but can be problematic for dashboard scenarios and is a non-starter for archiving and data warehousing scenarios.

Provisioning more resources can speed up the process, but the effect of having a processing rate surge is complex.

- Test that your job is scalable to a larger number of SUs. Not all queries are scalable. You need to make sure your query is parallelized.
- Make sure there are enough partitions in the upstream Event Hubs or IoT Hub that you can add more Throughput Units (TUs) to scale the input throughput. Remember, each Event Hubs TU maxes out at an output rate of 2 MB/s.
- Make sure you have provisioned enough resources in the output sinks (i.e., SQL Database, Cosmos DB), so they don't throttle the surge in output, which can sometimes cause the system to lock up.

The most important thing is to anticipate the processing rate change, test these scenarios before going into production, and be ready to scale the processing correctly during failure recovery time.

## ASA Input Types

Azure Stream Analytics jobs connect to one or more data inputs. Each input defines a connection to an existing data source. Stream Analytics accepts data incoming from several kinds of event sources including Event Hubs, IoT Hub, and Blob storage. The inputs are referenced by name in the streaming SQL query that you write for each job. In the query, you can join multiple inputs to blend data or compare streaming data with a lookup to reference data, and pass the results to outputs.

Stream Analytics has first-class integration with three kinds of resources as inputs:

- Azure Event Hubs
- Azure IoT Hub
- Azure Blob storage

These input resources can live in the same Azure subscription as your Stream Analytics job, or from a different subscription.

You can use the Azure portal, Azure PowerShell, .NET API, REST API, and Visual Studio to create, edit, and test Stream Analytics job inputs.

## Stream and reference inputs

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

### Data stream input

A data stream is an unbounded sequence of events over time. Stream Analytics jobs must include at least one data stream input. Event Hubs, IoT Hub, and Blob storage are supported as data stream input sources. Event Hubs are used to collect event streams from multiple devices and services. These streams might include 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, such as log files.

For more information about streaming data inputs, see Stream data as input into Stream Analytics here: <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-define-inputs>

### Reference data input

Stream Analytics also supports input known as reference data. Reference data is either completely static or changes slowly. It is typically used to perform correlation and lookups. For example, you might join data in the data stream input to data in the reference data, much as you would perform a SQL join to look up static values. Azure Blob storage and Azure SQL Database are currently supported as input sources for reference data. Reference data source blobs have a limit of up to 300 MB in size, depending on the query complexity and allocated Streaming Units (see the Size limitation section of the reference data documentation for more details).

For more information about reference data inputs, see Using reference data for lookups in Stream Analytics here: <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-use-reference-data>

## ASA Streaming Data Input

Stream Analytics has first-class integration with Azure data streams as inputs from three kinds of resources:

- Azure Event Hubs
- Azure IoT Hub
- Azure Blob storage

These input resources can live in the same Azure subscription as your Stream Analytics job or a different subscription.

## Compression

Stream Analytics supports compression across all data stream input sources. Supported compression types are: None, GZip, and Deflate compression. Support for compression is not available for reference data. If the input format is Avro data that is compressed, it's handled transparently. You don't need to specify compression type with Avro serialization.

## Create, edit, or test inputs

You can use the Azure portal, Visual Studio, and Visual Studio Code to add and view or edit existing inputs on your streaming job. You can also test input connections and test queries from sample data from the Azure portal, Visual Studio, and Visual Studio Code. When you write a query, you list the input in the FROM clause. You can get the list of available inputs from the Query page in the portal. If you wish to use multiple inputs, you can JOIN them or write multiple SELECT queries.

## Stream data from Event Hubs

Azure Event Hubs provides highly scalable publish-subscribe event ingestors. An event hub can collect millions of events per second so that you can process and analyze the massive amounts of data produced by your connected devices and applications. Together, Event Hubs and Stream Analytics provide an end-to-end solution for real-time analytics. Event Hubs lets you feed events into Azure in real-time, and Stream Analytics jobs can process those events in real-time. For example, you can send web clicks, sensor readings, or online log events to Event Hubs. You can then create Stream Analytics jobs to use Event Hubs as the input data streams for real-time filtering, aggregating, and correlation.

`EventEnqueuedUtcTime` is the timestamp of an event's arrival in an event hub and is the default timestamp of events coming from Event Hubs to Stream Analytics. To process the data as a stream using a timestamp in the event payload, you must use the **TIMESTAMP BY<sup>19</sup>** keyword.

## Event Hubs Consumer groups

You should configure each Stream Analytics event hub input to have its own consumer group. When a job contains a self-join or has multiple inputs, some inputs might be read by more than one reader downstream. This situation impacts the number of readers in a single consumer group. To avoid exceeding the Event Hubs limit of five readers per consumer group per partition, it's a best practice to designate a consumer group for each Stream Analytics job. There is also a limit of 20 consumer groups for a Standard tier event hub. For more information, see [Troubleshoot Azure Stream Analytics inputs<sup>20</sup>](#).

## Create an input from Event Hubs

The following table explains each property in the **New input** page in the Azure portal to stream data input from an event hub:

<sup>19</sup> <https://docs.microsoft.com/stream-analytics-query/timestamp-by-azure-stream-analytics>

<sup>20</sup> <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-troubleshoot-input>

Property	Description
Input alias	A friendly name that you use in the job's query to reference this input.
Subscription	Choose the subscription in which the Event hub resource exists.
Event Hub namespace	The Event Hub namespace is a container for a set of messaging entities. When you create a new event hub, you also create the namespace.
Event Hub name	The name of the event hub to use as input.
Event Hub policy name	The shared access policy that provides access to the Event Hub. Each shared access policy has a name, permissions that you set, and access keys. This option is automatically populated, unless you select the option to provide the Event Hub settings manually.
Event Hub consumer group (recommended)	It is highly recommended to use a distinct consumer group for each Stream Analytics job. This string identifies the consumer group to use to ingest data from the event hub. If no consumer group is specified, the Stream Analytics job uses the \$Default consumer group.
Event serialization format	The serialization format (JSON, CSV, Avro, or Other (Protobuf, XML, proprietary...)) of the incoming data stream. Ensure the JSON format aligns with the specification and doesn't include leading 0 for decimal numbers.
Encoding	UTF-8 is currently the only supported encoding format.
Event compression type	The compression type used to read the incoming data stream, such as None (default), GZip, or Deflate.

When your data comes from an Event Hub stream input, you have access to the following metadata fields in your Stream Analytics query:

Property	Description
EventProcessedUtcTime	The date and time that the event was processed by Stream Analytics.
EventEnqueuedUtcTime	The date and time that the event was received by Event Hubs.
PartitionId	The zero-based partition ID for the input adapter.

For example, using these fields, you can write a query like the following example:

```
SELECT
    EventProcessedUtcTime,
    EventEnqueuedUtcTime,
    PartitionId
FROM Input
```

**Note:** When using Event Hub as an endpoint for IoT Hub Routes, you can access to the IoT Hub metadata using the GetMetadataPropertyValue function - described here: <https://docs.microsoft.com/stream-analytics-query/getmetadatapropertyvalue>

## Stream data from IoT Hub

Azure IoT Hub is a highly scalable publish-subscribe event ingestor optimized for IoT scenarios.

The default timestamp of events coming from an IoT Hub in Stream Analytics is the timestamp that the event arrived in the IoT Hub, which is `EventEnqueuedUtcTime`. To process the data as a stream using a timestamp in the event payload, you must use the `TIMESTAMP BY` keyword.

## IoT Hub Consumer groups

You should configure each Stream Analytics IoT Hub input to have its own consumer group. When a job contains a self-join or when it has multiple inputs, some input might be read by more than one reader downstream. This situation impacts the number of readers in a single consumer group. To avoid exceeding the Azure IoT Hub limit of five readers per consumer group per partition, it's a best practice to designate a consumer group for each Stream Analytics job.

## Configure an IoT Hub as a data stream input

The following table explains each property in the New input page in the Azure portal when you configure an IoT Hub as a stream input.

Property	Description
Input alias	A friendly name that you use in the job's query to reference this input.
Subscription	Choose the subscription in which the IoT Hub resource exists.
IoT Hub	The name of the IoT Hub to use as input.
Endpoint	The endpoint for the IoT Hub.
Shared access policy name	The shared access policy that provides access to the IoT Hub. Each shared access policy has a name, permissions that you set, and access keys.
Shared access policy key	The shared access key used to authorize access to the IoT Hub. This option is automatically populated in unless you select the option to provide the IoT Hub settings manually.
Consumer group	It is highly recommended that you use a different consumer group for each Stream Analytics job. The consumer group is used to ingest data from the IoT Hub. Stream Analytics uses the <code>\$Default</code> consumer group unless you specify otherwise.
Event serialization format	The serialization format (JSON, CSV, Avro, or Other (Protobuf, XML, proprietary...)) of the incoming data stream. Ensure the JSON format aligns with the specification and doesn't include leading 0 for decimal numbers.

Property	Description
Encoding	UTF-8 is currently the only supported encoding format.
Event compression type	The compression type used to read the incoming data stream, such as None (default), GZip, or Deflate.

When you use stream data from an IoT Hub, you have access to the following metadata fields in your Stream Analytics query:

Property	Description
EventProcessedUtcTime	The date and time that the event was processed.
EventEnqueuedUtcTime	The date and time that the event was received by the IoT Hub.
PartitionId	The zero-based partition ID for the input adapter.
IoTHub.MessageId	An ID that's used to correlate two-way communication in IoT Hub.
IoTHub.CorrelationId	An ID that's used in message responses and feedback in IoT Hub.
IoTHub.ConnectionDeviceId	The authentication ID used to send this message. This value is stamped on servicebound messages by the IoT Hub.
IoTHub.ConnectionDeviceGenerationId	The generation ID of the authenticated device that was used to send this message. This value is stamped on servicebound messages by the IoT Hub.
IoTHub.EnqueueTime	The time when the message was received by the IoT Hub.

## Stream data from Blob storage

For scenarios with large quantities of unstructured data to store in the cloud, Azure Blob storage offers a cost-effective and scalable solution. Data in Blob storage is usually considered data at rest; however, blob data can be processed as a data stream by Stream Analytics.

Log processing is a commonly used scenario for using Blob storage inputs with Stream Analytics. In this scenario, telemetry data files have been captured from a system and need to be parsed and processed to extract meaningful data.

The default timestamp of Blob storage events in Stream Analytics is the timestamp that the blob was last modified, which is `BlobLastModifiedUtcTime`. If a blob is uploaded to a storage account at 13:00, and the Azure Stream Analytics job is started using the option Now at 13:01, the blob will not be picked up as its modified time falls outside the job run period.

If a blob is uploaded to a storage account container at 13:00, and the Azure Stream Analytics job is started using Custom Time at 13:00 or earlier, the blob will be picked up as its modified time falls inside the job run period.

If an Azure Stream Analytics job is started using Now at 13:00, and a blob is uploaded to the storage account container at 13:01, Azure Stream Analytics will pick up the blob.

To process the data as a stream using a timestamp in the event payload, you must use the `TIMESTAMP BY` keyword. A Stream Analytics job pulls data from Azure Blob storage input every second if the blob file is

available. If the blob file is unavailable, there is an exponential backoff with a maximum time delay of 90 seconds.

CSV-formatted inputs require a header row to define fields for the data set, and all header row fields must be unique.

**Note:** Stream Analytics does not support adding content to an existing blob file. Stream Analytics will view each file only once, and any changes that occur in the file after the job has read the data are not processed. Best practice is to upload all the data for a blob file at once and then add additional newer events to a different, new blob file.

Uploading a very large number of blobs at once might cause Stream Analytics to skip reading few blobs in rare cases. It is recommended to upload blobs at least 2 seconds apart to Blob storage. If this option is not feasible, you can use Event Hubs to stream large volumes of events.

## Configure Blob storage as a stream input

The following table explains each property in the New input page in the Azure portal when you configure Blob storage as a stream input.

Property	Description
Input alias	A friendly name that you use in the job's query to reference this input.
Subscription	Choose the subscription in which the IoT Hub resource exists.
Storage account	The name of the storage account where the blob files are located.
Storage account key	The secret key associated with the storage account. This option is automatically populated in unless you select the option to provide the Blob storage settings manually.
Container	The container for the blob input. Containers provide a logical grouping for blobs stored in the Microsoft Azure Blob service. When you upload a blob to the Azure Blob storage service, you must specify a container for that blob. You can choose either Use existing container or Create new to have a new container created.

Property	Description
Path pattern (optional)	<p>The file path used to locate the blobs within the specified container. If you want to read blobs from the root of the container, do not set a path pattern. Within the path, you can specify one or more instances of the following three variables: {date}, {time}, or {partition}</p> <p>Example 1: cluster1/logs/{date}/{time}/{partition}</p> <p>Example 2: cluster1/logs/{date}</p> <p>The * character is not an allowed value for the path prefix. Only valid Azure blob characters are allowed. No not include container names or file names.</p>
Date format (optional)	If you use the date variable in the path, the date format in which the files are organized. Example: YYYY/MM/DD
Time format (optional)	If you use the time variable in the path, the time format in which the files are organized. Currently the only supported value is HH for hours.
Event serialization format	The serialization format (JSON, CSV, Avro, or Other (Protobuf, XML, proprietary...)) of the incoming data stream. Ensure the JSON format aligns with the specification and doesn't include leading 0 for decimal numbers.
Encoding	For CSV and JSON, UTF-8 is currently the only supported encoding format.
Compression	The compression type used to read the incoming data stream, such as None (default), GZip, or Deflate.

When your data comes from a Blob storage source, you have access to the following metadata fields in your Stream Analytics query:

Property	Description
BlobName	The name of the input blob that the event came from.
EventProcessedUtcTime	The date and time that the event was processed by Stream Analytics.
BlobLastModifiedUtcTime	The date and time that the blob was last modified.
PartitionId	The zero-based partition ID for the input adapter.

For example, using these fields, you can write a query like the following example:

```
SELECT
    BlobName,
    EventProcessedUtcTime,
    BlobLastModifiedUtcTime
```

FROM Input

## ASA Reference Data Input

Reference data (also known as a lookup table) is a finite data set that is static or slowly changing in nature, used to perform a lookup or to augment your data streams. For example, in an IoT scenario, you could store metadata about sensors (which don't change often) in reference data and join it with real time IoT data streams. Azure Stream Analytics loads reference data in memory to achieve low latency stream processing. To make use of reference data in your Azure Stream Analytics job, you will generally use a Reference Data Join in your query.

Stream Analytics supports Azure Blob storage and Azure SQL Database as the storage layer for Reference Data. You can also transform and/or copy reference data to Blob storage from Azure Data Factory to use any number of cloud-based and on-premises data stores.

### Azure Blob storage

Reference data is modeled as a sequence of blobs (defined in the input configuration) in ascending order of the date/time specified in the blob name. It only supports adding to the end of the sequence by using a date/time greater than the one specified by the last blob in the sequence.

### Configure blob reference data

To configure your reference data, you first need to create an input that is of type Reference Data. The table below explains each property that you will need to provide while creating the reference data input with its description:

Property	Description
Input Alias	A friendly name that will be used in the job query to reference this input.
Storage Account	The name of the storage account where your blobs are located. If it's in the same subscription as your Stream Analytics Job, you can select it from the drop-down.
Storage Account Key	The secret key associated with the storage account. This gets automatically populated if the storage account is in the same subscription as your Stream Analytics job.
Storage Container	Containers provide a logical grouping for blobs stored in the Microsoft Azure Blob service. When you upload a blob to the Blob service, you must specify a container for that blob.

Property	Description
Path Pattern	The path used to locate your blobs within the specified container. Within the path, you may choose to specify one or more instances of the following 2 variables: {date}, {time} Example 1: products/{date}/{time}/product-list.csv Example 2: products/{date}/product-list.csv Example 3: product-list.csv  If the blob doesn't exist in the specified path, the Stream Analytics job will wait indefinitely for the blob to become available.
Date Format [optional]	If you have used {date} within the Path Pattern that you specified, then you can select the date format in which your blobs are organized from the drop-down of supported formats. Example: YYYY/MM/DD, MM/DD/YYYY, etc.
Time Format [optional]	If you have used {time} within the Path Pattern that you specified, then you can select the time format in which your blobs are organized from the drop-down of supported formats. Example: HH, HH/mm, or HH-mm.
Event Serialization Format	To make sure your queries work the way you expect, Stream Analytics needs to know which serialization format you're using for incoming data streams. For Reference Data, the supported formats are CSV and JSON.
Encoding	UTF-8 is the only supported encoding format at this time.

## Static reference data

If your reference data is not expected to change, then support for static reference data is enabled by specifying a static path in the input configuration. Azure Stream Analytics picks up the blob from the specified path. {date} and {time} substitution tokens aren't required. Because reference data is immutable in Stream Analytics, overwriting a static reference data blob is not recommended.

## Generate reference data on a schedule

If your reference data is a slowly changing data set, then support for refreshing reference data is enabled by specifying a path pattern in the input configuration using the {date} and {time} substitution tokens. Stream Analytics picks up the updated reference data definitions based on this path pattern. For example, a pattern of sample/{date}/{time}/products.csv with a date format of "YYYY-MM-DD" and a time format of "HH-mm" instructs Stream Analytics to pick up the updated blob sample/2015-04-16/17-30/products.csv at 5:30 PM on April 16th, 2015 UTC time zone.

Azure Stream Analytics automatically scans for refreshed reference data blobs at a one minute interval. If a blob with timestamp 10:30:00 is uploaded with a small delay (for example, 10:30:30), you will notice a small delay in Stream Analytics job referencing this blob. To avoid such scenarios, it is recommended to

upload the blob earlier than the target effective time (10:30:00 in this example) to allow the Stream Analytics job enough time to discover and load it in memory and perform operations.

**Note:**

Currently Stream Analytics jobs look for the blob refresh only when the machine time advances to the time encoded in the blob name. For example, the job will look for sample/2015-04-16/17-30/products.csv as soon as possible but no earlier than 5:30 PM on April 16th, 2015 UTC time zone. It will never look for a blob with an encoded time earlier than the last one that is discovered.

For example, once the job finds the blob sample/2015-04-16/17-30/products.csv it will ignore any files with an encoded date earlier than 5:30 PM April 16th, 2015 so if a late arriving sample/2015-04-16/17-25/products.csv blob gets created in the same container the job will not use it.

Likewise if sample/2015-04-16/17-30/products.csv is only produced at 10:03 PM April 16th, 2015 but no blob with an earlier date is present in the container, the job will use this file starting at 10:03 PM April 16th, 2015 and use the previous reference data until then.

An exception to this is when the job needs to re-process data back in time or when the job is first started. At start time the job is looking for the most recent blob produced before the job start time specified. This is done to ensure that there is a non-empty reference data set when the job starts. If one cannot be found, the job displays the following diagnostic: Initializing input without a valid reference data blob for UTC time <start time>.

Azure Data Factory can be used to orchestrate the task of creating the updated blobs required by Stream Analytics to update reference data definitions. Data Factory is a cloud-based data integration service that orchestrates and automates the movement and transformation of data. Data Factory supports connecting to a large number of cloud based and on-premises data stores and moving data easily on a regular schedule that you specify. For more information and step by step guidance on how to set up a Data Factory pipeline to generate reference data for Stream Analytics which refreshes on a pre-defined schedule, check out this GitHub sample.

## Tips on refreshing blob reference data

1. Do not overwrite reference data blobs as they are immutable.
2. The recommended way to refresh reference data is to:
  - Use {date}/{time} in the path pattern
  - Add a new blob using the same container and path pattern defined in the job input
  - Use a date/time greater than the one specified by the last blob in the sequence.
3. Reference data blobs are not ordered by the blob's "Last Modified" time but only by the time and date specified in the blob name using the {date} and {time} substitutions.
4. To avoid having to list large number of blobs, consider deleting very old blobs for which processing will no longer be done. Please note that ASA might have to reprocess a small amount in some scenarios like a restart.

## Azure SQL Database

Azure SQL Database reference data is retrieved by your Stream Analytics job and is stored as a snapshot in memory for processing. The snapshot of your reference data is also stored in a container in a storage account that you specify in the configuration settings. The container is auto-created when the job starts. If the job is stopped or enters a failed state, the auto-created containers are deleted when the job is restarted.

If your reference data is a slowly changing data set, you need to periodically refresh the snapshot that is used in your job. Stream Analytics allows you to set a refresh rate when you configure your Azure SQL Database input connection. The Stream Analytics runtime will query your Azure SQL Database at the interval specified by the refresh rate. The fastest refresh rate supported is once per minute. For each refresh, Stream Analytics stores a new snapshot in the storage account provided.

Stream Analytics provides two options for querying your Azure SQL Database. A snapshot query is mandatory and must be included in each job. Stream Analytics runs the snapshot query periodically based on your refresh interval and uses the result of the query (the snapshot) as the reference data set. The snapshot query should fit most scenarios, but if you run into performance issues with large data sets and fast refresh rates, you can use the delta query option. Queries that take more than 60 seconds to return reference data set will result in a timeout.

With the delta query option, Stream Analytics runs the snapshot query initially to get a baseline reference data set. After, Stream Analytics runs the delta query periodically based on your refresh interval to retrieve incremental changes. These incremental changes are continually applied to the reference data set to keep it updated. Using delta query may help reduce storage cost and network I/O operations.

## Configure SQL Database reference

To configure your SQL Database reference data, you first need to create Reference Data input. The table below explains each property that you will need to provide while creating the reference data input with its description. For more information, see [Use reference data from a SQL Database for an Azure Stream Analytics job](#).

You can use Azure SQL Database Managed Instance as a reference data input. You have to configure public endpoint in Azure SQL Database Managed Instance and then manually configure the following settings in Azure Stream Analytics. Azure virtual machine running SQL Server with a database attached is also supported by manually configuring the settings below.

Property	Description
Input alias	A friendly name that will be used in the job query to reference this input.
Subscription	Choose your subscription
Database	The Azure SQL Database that contains your reference data. For Azure SQL Database Managed Instance, it is required to specify the port 3342. For example, sampleserver.public.database.windows.net,3342
Username	The username associated with your Azure SQL Database.
Password	The password associated with your Azure SQL Database.
Refresh periodically	This option allows you to choose a refresh rate. Choosing “On” will allow you to specify the refresh rate in DD:HH:MM.
Snapshot query	This is the default query option that retrieves the reference data from your SQL Database.
Delta query	For advanced scenarios with large data sets and a short refresh rate, choose to add a delta query.

## Size limitation

Stream Analytics supports reference data with maximum size of 300 MB. The 300 MB limit of maximum size of reference data is achievable only with simple queries. As the complexity of query increases to include stateful processing, such as windowed aggregates, temporal joins and temporal analytic functions, it is expected that the maximum supported size of reference data decreases. If Azure Stream Analytics cannot load the reference data and perform complex operations, the job will run out of memory and fail. In such cases, SU % Utilization metric will reach 100%.

Number of Streaming Units	Approx. Max Size Supported (in MB)
1	50
3	150
6 and beyond	300

## ASA Queries

Queries in Azure Stream Analytics are expressed in a SQL-like query language.

Queries can be designed for simple pass-through logic that moves event data from one input stream into an output data store, or they can do rich pattern matching and temporal analysis to calculate aggregates over various time windows. Queries can also join data from multiple inputs to combine streaming events, and can do lookups against static reference data to enrich the event values. Queries can also write data to multiple outputs.

Queries are written using a combination of query syntax elements and functions to process input data types in order to produce a desired output.

## Query Syntax Elements

Azure Stream Analytics provides a variety of elements for building queries. They are summarized below.

Element	Summary
APPLY	The APPLY operator allows you to invoke a table-valued function for each row returned by an outer table expression of a query. There are two forms of APPLY:  CROSS APPLY returns only rows from the outer table that produce a result set from the table-valued function.  OUTER APPLY returns both rows that produce a result set, and rows that do not, with NULL values in the columns produced by the table-valued function.
CASE	CASE evaluates a list of conditions and returns one of multiple possible result expressions
COALESCE	COALESCE evaluates the arguments in order and returns the value of the first expression that initially does not evaluate to NULL.

Element	Summary
CREATE TABLE	CREATE TABLE is used to define the schema of the payload of the events coming into Azure Stream Analytics.
FROM	FROM specifies the input stream or a step name associated in a WITH clause. The FROM clause is always required for any SELECT statement.
GROUP BY	GROUP BY groups a selected set of rows into a set of summary rows grouped by the values of one or more columns or expressions.
HAVING	HAVING specifies a search condition for a group or an aggregate. HAVING can be used only with the SELECT expression.
INTO	INTO explicitly specifies an output stream, and is always associated with an SELECT expression. If not specified, the default output stream is "output".
JOIN and Reference Data JOIN	JOIN is used to combine records from two or more input sources. JOIN is temporal in nature, meaning that each JOIN must define how far the matching rows can be separated in time.  JOIN is also used to correlate persisted historical data or a slow changing dataset (aka. reference data) with the real-time event stream to make smarter decisions about the system. For example, join an event stream to a static dataset which maps IP Addresses to locations. This is the only JOIN supported in Stream Analytics where a temporal bound is not necessary.
MATCH_RECOGNIZE	MATCH_RECOGNIZE is used to search for a set of events over a data stream.
OVER	OVER defines the grouping of rows before an associated aggregate or analytic function is applied.
SELECT	SELECT is used to retrieve rows from input streams and enables the selection of one or many columns from one or many input streams in Azure Stream Analytics.
UNION	UNION combines two or more queries into a single result set that includes all the rows that belong to all queries in the union.
WHERE	WHERE specifies the search condition for the rows returned by the query.
WITH	WITH specifies a temporary named result set which can be referenced by a FROM clause in the query. This is defined within the execution scope of a single SELECT statement.

## Built-in Functions

### Types of Functions

Function Category	Description
Aggregate Functions	Operate on a collection of values but return a single, summarizing value.
Analytic Functions	Return a value based on defined constraints.
Array Functions	Returns information from an array.
GeoSpatial Functions	Perform specialized GeoSpatial functions.
Input Metadata Functions	Query the metadata of property in the data input.
Record Functions	Returns record properties or values.
Windowing Functions	Perform operations on events within a time window.
Scalar Functions	Operate on a single value and then return a single value. Scalar functions can be used wherever an expression is valid.

### Scalar Functions

A scalar function operates on a single value and then returns a single value. Scalar functions can be used wherever an expression is valid.

Function Category	Description
Conversion Functions	These functions allow you to cast data into different formats.
Date and Time Functions	These functions allow you to perform operations on DateTime formats.
Mathematical Functions	Represent the scalar functions that perform a calculation, usually based on input values that are provided as arguments, and return a numeric value.
String Functions	These functions allow you to convert strings to upper or lower case.

### Query Syntax Examples

The query examples listed below are sourced from the longer list of examples in the Stream Analytics query language reference. They were selected because they help to illustrate common query patterns and show basic syntax.

All query pattern examples included in the reference guide are based on the following toll booth scenario:

A vehicle tolling station is a common phenomenon – we encounter them in many expressways, bridges, and tunnels across the world. Each toll station has multiple toll booth lanes that a car can enter, which may be manual – meaning that you stop to pay the toll to an attendant, or automated – where a sensor placed on top of the booth scans an RFID card affixed to the windshield of your vehicle as you pass the

toll booth. It is easy to visualize the passage of vehicles through these toll stations as an event stream over which interesting operations can be performed.

## Pattern 1 - Convert Data Types

For this example, the input stream includes the make of the car, the time when the car passes through the booth, and the weight of the car. The goal is to calculate the sum of vehicle weight that passes through an individual booth during a defined time period. In this example, the car weight reported on the input stream is represented as a string data type. The weight values must be converted to INT before the SUM can be calculated.

### Input

Make	Time	Weight
Honda	2015-01-01T00:00:01.0000000Z	"1000"
Honda	2015-01-01T00:00:02.0000000Z	"2000"

### Query Solution

```
SELECT
    Make,
    SUM(CAST(Weight AS BIGINT)) AS Weight
FROM
    Input TIMESTAMP BY Time
GROUP BY
    Make,
    TumblingWindow(second, 10)
```

### Output

Make	Weight
Honda	3000

### Explanation

We use a CAST statement in the Weight field to specify its data type.

## Pattern - Send Data to Multiple Outputs

### Description

For this example, the input stream includes the make of a car and the time when that car passes through the booth. We have two goals. The first goal is to archive the input data to cold storage. The second goal is to support a downstream alerting system that sends alerts whenever the traffic rate for a particular make of vehicle exceeds a defined threshold value. To achieve both goals we need to process the data for multiple output targets. The first target output will receive all of the input data. For the second target

output will receive a record indicating the make of the vehicle, the time for the event, and the associated number of vehicles passing through the booth.

## Input

Make	Time
Honda	2015-01-01T00:00:01.0000000Z
Honda	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:03.0000000Z

## Query Solution

```

SELECT
    *
INTO
    ArchiveOutput
FROM
    Input TIMESTAMP BY Time

SELECT
    Make,
    System.TimeStamp() AS AsaTime,
    COUNT(*) AS [Count]
INTO
    AlertOutput
FROM
    Input TIMESTAMP BY Time
GROUP BY
    Make,
    TumblingWindow(second, 10)
HAVING
    [Count] >= 3
  
```

## Output 1

Make	Time
Honda	2015-01-01T00:00:01.0000000Z
Honda	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:01.0000000Z
Toyota	2015-01-01T00:00:02.0000000Z
Toyota	2015-01-01T00:00:03.0000000Z

## Output 2

Make	Time	Count
Toyota	2015-01-01T00:00:10.000000Z	3

## Explanation

The INTO clause tells Stream Analytics which of the outputs to write the data to from this statement. The first query is a pass-through of the data received to an output named ArchiveOutput. The second query does some simple aggregation and filtering, and it sends the results to a downstream alerting system, AlertOutput.

The full Stream Analytics query language reference guide can be found here: <https://docs.microsoft.com/en-us/stream-analytics-query/stream-analytics-query-language-reference>.

## ASA Queries - Parse Complex Data Types

Azure Stream Analytics support processing events in CSV, JSON, and Avro data formats. Both JSON and Avro data can be structured and contain some complex types such as nested objects (records) and arrays.

### Record data types

Record data types are used to represent JSON and Avro arrays when corresponding formats are used in the input data streams. These examples demonstrate a sample sensor, which is reading input events in JSON format. Here is example of a single event:

```
{
  "DeviceId": "12345",
  "Location" :
  {
    "Lat": 47,
    "Long": 122
  },
  "SensorReadings" :
  {
    "Temperature": 80,
    "Humidity": 70,
    "CustomSensor01": 5,
    "CustomSensor02": 99,
    "SensorMetadata" :
    {
      "Manufacturer": "ABC",
      "Version": "1.2.45"
    }
  }
}
```

## Access nested fields in known schema

Use dot notation (.) to easily access nested fields directly from your query. For example, this query selects the Latitude and Longitude coordinates under the Location property in the preceding JSON data. The dot notation can be used to navigate multiple levels as shown below.

```
SELECT
    DeviceID,
    Location.Lat,
    Location.Long,
    SensorReadings.SensorMetadata.Version
FROM input
```

## Select all properties

You can select all the properties of a nested record using '\*' wildcard. Consider the following example:

```
SELECT input.Location.*
FROM input
```

The result is:

```
{
    "Lat" : 47,
    "Long" : 122
}
```

## Access nested fields when property name is a variable

Use the GetRecordPropertyValue function if the property name is a variable.

For example, imagine a sample data stream needs to be joined with reference data containing thresholds for each device sensor. A snippet of such reference data is shown below.

```
{
    "DeviceId" : "12345",
    "SensorName" : "Temperature",
    "Value" : 75
}
```

```
SELECT
    input.DeviceID,
    thresholds.SensorName
FROM input    -- stream input
JOIN thresholds -- reference data input
ON
    input.DeviceId = thresholds.DeviceId
WHERE
    GetRecordPropertyValue(input.SensorReadings, thresholds.SensorName) > thresholds.Value
-- the where statement selects the property value coming from the reference data
```

## Convert record fields into separate events

To convert record fields into separate events, use the APPLY operator together with the GetRecordProperties function. For example, if the previous example had several records for SensorReading, the following query could be used to extract them into different events:

```
SELECT
    event.DeviceID,
    sensorReading.PropertyName,
    sensorReading.PropertyValue
FROM input as event
CROSS APPLY GetRecordProperties(event.SensorReadings) AS sensorReading
```

## Array data types

Array data types are an ordered collection of values. Some typical operations on array values are detailed below. These examples assume the input events have a property named “arrayField” that is an array datatype.

These examples use the functions GetArrayElement, GetArrayElements, GetArrayLength, and the APPLY operator.

## Working with a specific array element

Select array element at a specified index (selecting the first array element):

```
SELECT
    GetArrayElement(arrayField, 0) AS firstElement
FROM input
```

## Select array length

```
SELECT
    GetArrayLength(arrayField) AS arrayLength
FROM input
```

## Convert array elements into separate events

Select all array element as individual events. The APPLY operator together with the GetArrayElements built-in function extracts all array elements as individual events:

```
SELECT
    arrayElement.ArrayIndex,
    arrayElement.ArrayValue
FROM input as event
CROSS APPLY GetArrayElements(event.arrayField) AS arrayElement
```

# ASA Queries - Time Handling Considerations

Query design and development often includes solving practical time handling problems in the Azure Stream Analytics service. Time handling design decisions are closely related to event ordering factors.

## Background time concepts

To better frame the discussion, let's define some background concepts:

- **Event time:** The time when the original event happened. For example, when a moving car on the highway approaches a toll booth.
- **Processing time:** The time when the event reaches the processing system and is observed. For example, when a toll booth sensor sees the car and the computer system takes a few moments to process the data.
- **Watermark:** An event time marker that indicates up to what point events have been ingressed to the streaming processor. Watermarks let the system indicate clear progress on ingesting the events. By the nature of streams, the incoming event data never stops, so watermarks indicate the progress to a certain point in the stream.

The watermark concept is important. Watermarks allow Stream Analytics to determine when the system can produce complete, correct, and repeatable results that don't need to be retracted. The processing can be done in a guaranteed way that's predictable and repeatable. For example, if a recount needs to be done for some error handling condition, watermarks are safe starting and ending points.

## Choosing the best starting time

Stream Analytics gives users two choices for picking event time:

### 1. Arrival time

Arrival time is assigned at the input source when the event reaches the source. You can access arrival time by using the **EventEnqueuedUtcTime** property for Event Hubs inputs, **IoTHub.EnqueueTime** property for IoT Hub, and using the **BlobProperties.LastModified** property for blob input.

Using arrival time is the default behavior, and best used for data archiving scenarios, where there's no temporal logic necessary.

### 2. Application time (also named Event Time)

Application time is assigned when the event is generated, and it's part of the event payload. To process events by application time, use the **Timestamp by** clause in the select query. If the **Timestamp by** clause is absent, events are processed by arrival time.

It's important to use a timestamp in the payload when temporal logic is involved. That way, delays in the source system or in the network can be accounted for.

## How time progresses in Azure Stream Analytics

When using application time, the time progression is based on the incoming events. It's difficult for the stream processing system to know if there are no events, or if events are delayed. For this reason, Azure Stream Analytics generates heuristic watermarks in the following ways for each input partition:

1. Whenever there's any incoming event, the watermark is the largest event time we have seen so far minus the out-of-order tolerance window size.

- Whenever there is no incoming event, the watermark is the current estimated arrival time (the elapsed time on behind the scenes VM processing the events from last time an input event is seen plus that input event's arrival time) minus the late arrival tolerance window.

The arrival time can only be estimated, because the real arrival time is generated on the input event broker, such as Event Hubs, and not the Azure Stream Analytics VM processing the events.

The design serves two additional purposes, besides generating watermarks:

- The system generates results in a timely fashion with or without incoming events.

You have control over how timely they want to see the output results. In the Azure portal, on the **Event ordering** page of your Stream Analytics job, you can configure the **Out of order events** setting. When configuring that setting, consider the trade-off of timeliness with tolerance of out-of-order events in the event stream.

The late arrival tolerance window is important to keep generating watermarks, even in the absence of incoming events. At times, there may be a period where no incoming events come in, such as when an event input stream is sparse. That problem is exacerbated by the use of multiple partitions in the input event broker.

Streaming data processing systems without a late arrival tolerance window may suffer from delayed outputs when inputs are sparse and multiple partitions are used.

- The system behavior has to be repeatable. Repeatability is an important property of a streaming data processing system.

The watermark is derived from arrival time and application time. Both are persisted in the event broker, and thus repeatable. In the case the arrival time has to be estimated in the absence of events, Azure Stream Analytics journals the estimated arrival time for repeatability during replay for the purpose of failure recovery.

Notice that when you choose to use **arrival time** as the event time, there is no need to configure the out-of-order tolerance and late arrival tolerance. Since **arrival time** is guaranteed to be monotonically increasing in the input event broker, Azure Stream Analytics simply disregards the configurations.

## Late arriving events

By definition of late arrival tolerance window, for each incoming event, Azure Stream Analytics compares the **event time** with the **arrival time**; if the event time is outside of the tolerance window, you can configure the system to either drop the event or adjust the event's time to be within the tolerance.

Consider that after watermarks are generated, the service can potentially receive events with event time lower than the watermark. You can configure the service to either **drop** those events, or **adjust** the event's time to the watermark value.

As a part of the adjustment, the event's **System.Timestamp** is set to the new value, but the **event time** field itself is not changed. This adjustment is the only situation where an event's **System.Timestamp** can be different from the value in the event time field, and may cause unexpected results to be generated.

## Handling time variation with substreams

The heuristic watermark generation mechanism described here works well in most of the cases where time is mostly synchronized between the various event senders. However, in real life, especially in many IoT scenarios, the system has little control over the clock on the event senders. The event senders could be all sorts of devices in the field, perhaps on different versions of hardware and software.

Instead of using a watermark global to all events in an input partition, Stream Analytics has another mechanism called substreams to help you. You can utilize substreams in your job by writing a job query that uses the `TIMESTAMP BY` clause and the keyword `OVER`. To designate the substream, provide a key column name after the `OVER` keyword, such as a `deviceid`, so that system applies time policies by that column. Each substream gets its own independent watermark. This mechanism is useful to allow timely output generation, when dealing with large clock skews or network delays among event senders.

Substreams are a unique solution provided by Azure Stream Analytics, and are not offered by other streaming data processing systems. Stream Analytics applies the late arrival tolerance window to incoming events when substreams are used. The default setting (5 seconds) is likely too small for devices with divergent timestamps. We recommend that you start with 5 minutes, and make adjustments according to their device clock skew pattern.

## Early arriving events

You may have noticed another concept called early arrival window, that looks like the opposite of late arrival tolerance window. This window is fixed at 5 minutes, and serves a different purpose from late arrival one.

Because Azure Stream Analytics guarantees it always generates complete results, you can only specify **job start time** as the first output time of the job, not the input time. The job start time is required so that the complete window is processed, not just from the middle of the window.

Stream Analytics then derives the starting time from the query specification. However, because input event broker is only indexed by arrival time, the system has to translate the starting event time to arrival time. The system can start processing events from that point in the input event broker. With the early arriving window limit, the translation is straightforward. It's starting event time minus the 5-minute early arriving window. This calculation also means that the system drops all events that are seen having event time 5 minutes greater than arrival time.

This concept is used to ensure the processing is repeatable no matter where you start to output from. Without such a mechanism, it would not be possible to guarantee repeatability, as many other streaming systems claim they do.

## Side effects of event ordering time tolerances

Stream Analytics jobs have several **Event ordering** options. Two can be configured in the Azure portal: the **Out of order events** setting (out-of-order tolerance), and the **Events that arrive late** setting (late arrival tolerance). The **early arrival** tolerance is fixed and cannot be adjusted. These time policies are used by Stream Analytics to provide strong guarantees. However, these settings do have some sometimes unexpected implications:

1. Accidentally sending events that are too early.

Early events should not be outputted normally. It's possible that early events are sent to the output if sender's clock is running too fast though. All early arriving events are dropped, so you will not see any of them from the output.

2. Sending old events to Event Hubs to be processed by Azure Stream Analytics.

While old events may seem harmless at first, because of the application of the late arrival tolerance, the old events may be dropped. If the events are too old, the `System.Timestamp` value is altered during event ingestion. Due to this behavior, currently Azure Stream Analytics is more suited for near-real-time event processing scenarios, instead of historical event processing scenarios. You can set the **Events that arrive late** time to the largest possible value (20 days) to work around this behavior in some cases.

3. Outputs seem to be delayed.

The first watermark is generated at the calculated time: the **maximum event time** the system has observed so far, minus the out-of-order tolerance window size. By default, the out-of-order tolerance is configured to zero (00 minutes and 00 seconds). When you set it to a higher, non-zero time value, the streaming job's first output is delayed by that value of time (or greater) due to the first watermark time that is calculated.

4. Inputs are sparse.

When there is no input in a given partition, the watermark time is calculated as the **arrival time** minus the late arrival tolerance window. As a result, if input events are infrequent and sparse, the output can be delayed by that amount of time. The default **Events that arrive late** value is 5 seconds. You should expect to see some delay when sending input events one at a time, for example. The delays can get worse, when you set **Events that arrive late** window to a large value.

5. **System.Timestamp** value is different from the time in the **event time** field.

As described previously, the system adjusts event time by the out-of-order tolerance or late arrival tolerance windows. The **System.Timestamp** value of the event is adjusted, but not the **event time** field.

## Metrics to observe

You can observe a number of the Event ordering time tolerance effects through Stream Analytics job metrics. The following metrics are relevant:

Metric	Description
<b>Out-of-Order Events</b>	Indicates the number of events received out of order, that were either dropped or given an adjusted timestamp. This metric is directly impacted by the configuration of the <b>Out of order events</b> setting on the <b>Event ordering</b> page on the job in the Azure portal.
<b>Late Input Events</b>	Indicates the number of events arriving late from the source. This metric includes events that have been dropped or have had their timestamp was adjusted. This metric is directly impacted by the configuration of the <b>Events that arrive late</b> setting in the <b>Event ordering</b> page on the job in the Azure portal.
<b>Early Input Events</b>	Indicates the number of events arriving early from the source that have either been dropped, or their timestamp has been adjusted if they are beyond 5 minutes early.
<b>Watermark Delay</b>	Indicates the delay of the streaming data processing job. See more information in the following section.

## Watermark Delay details

The **Watermark delay** metric is computed as the wall clock time of the processing node minus the largest watermark it has seen so far.

There can be several reasons this metric value is larger than 0 under normal operation:

1. Inherent processing delay of the streaming pipeline. Normally this delay is nominal.
2. The out-of-order tolerance window introduced delay, because watermark is reduced by the size of the tolerance window.
3. The late arrival window introduced delay, because watermark is reduced by the size the tolerance window.
4. Clock skew of the processing node generating the metric.

There are a number of other resource constraints that can cause the streaming pipeline to slow down. The watermark delay metric can rise due to:

1. Not enough processing resources in Stream Analytics to handle the volume of input events. One option to scale up resources is to adjust Streaming Units, see: <https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-streaming-unit-consumption>.
2. Not enough throughput within the input event brokers, so they are throttled. For possible solutions, see Automatically scale up Azure Event Hubs throughput units here: <https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-auto-inflate>.
3. Output sinks are not provisioned with enough capacity, so they are throttled. The possible solutions vary widely based on the flavor of output service being used.

## Output event frequency

Azure Stream Analytics uses watermark progress as the only trigger to produce output events. Because the watermark is derived from input data, it is repeatable during failure recovery and also in user initiated reprocessing.

When using windowed aggregates, the service only produces outputs at the end of the windows. In some cases, users may want to see partial aggregates generated from the windows. Partial aggregates are not supported currently in Azure Stream Analytics.

In other streaming solutions, output events could be materialized at various trigger points, depending on external circumstances. It's possible in some solutions that the output events for a given time window could be generated multiple times. As the input values are refined, the aggregate results become more accurate. Events could be speculated at first, and revised over time. For example, when a certain device is offline from the network, an estimated value could be used by a system. Later on, the same device comes online to the network. Then the actual event data could be included in the input stream. The output results from processing that time window produces more accurate output.

## Illustrated example of watermarks

The following images illustrate how watermarks progress in different circumstances.

This table shows the example data that is charted below. Notice that the event time and the arrival time vary, sometimes matching and sometimes not.

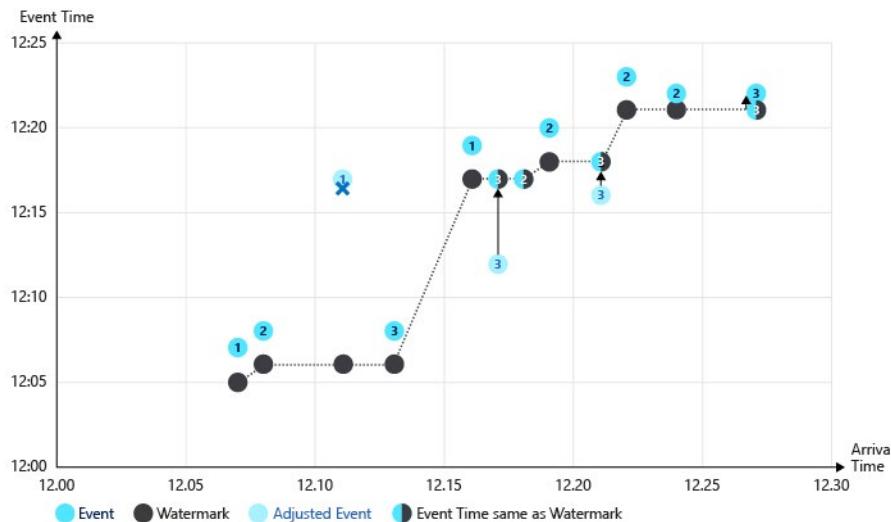
Event time	Arrival time	DeviceId
12:07	12:07	device1
12:08	12:08	device2
12:17	12:11	device1
12:08	12:13	device3

Event time	Arrival time	DeviceId
12:19	12:16	device1
12:12	12:17	device3
12:17	12:18	device2
12:20	12:19	device2
12:16	12:21	device3
12:23	12:22	device2
12:22	12:24	device2
12:21	12:27	device3

In this illustration, the following tolerances are used:

- Early arrival windows is 5 minutes
- Late arriving window is 5 minutes
- Reorder window is 2 minutes

#### 1. Illustration of watermark progressing through these events:



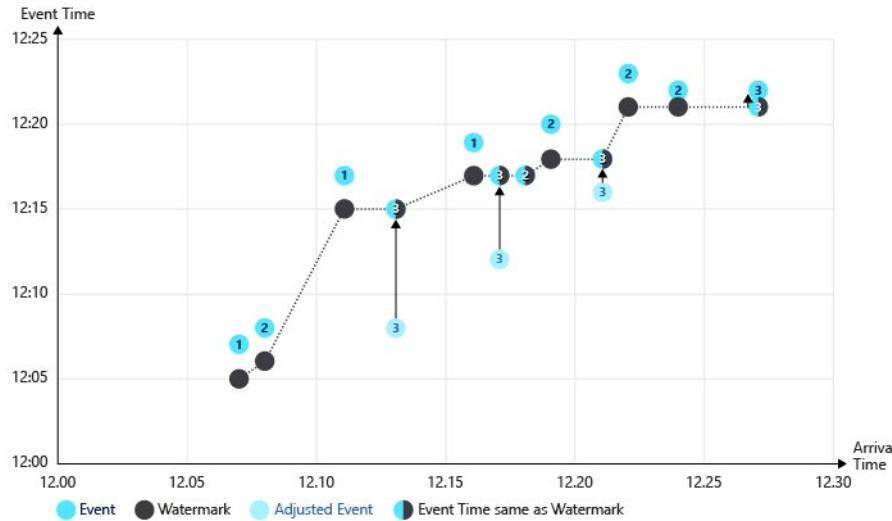
Notable processes illustrated in the preceding graphic:

- The first event (device1), and second event (device2) have aligned times and are processed without adjustment. The watermark progresses on each event.
- When the third event (device1) is processed, the arrival time (12:11) precedes the event time (12:17). The event arrived 6 minutes early, so the event is dropped due to the 5-minute early arrival tolerance.

The watermark doesn't progress in this case of an early event.

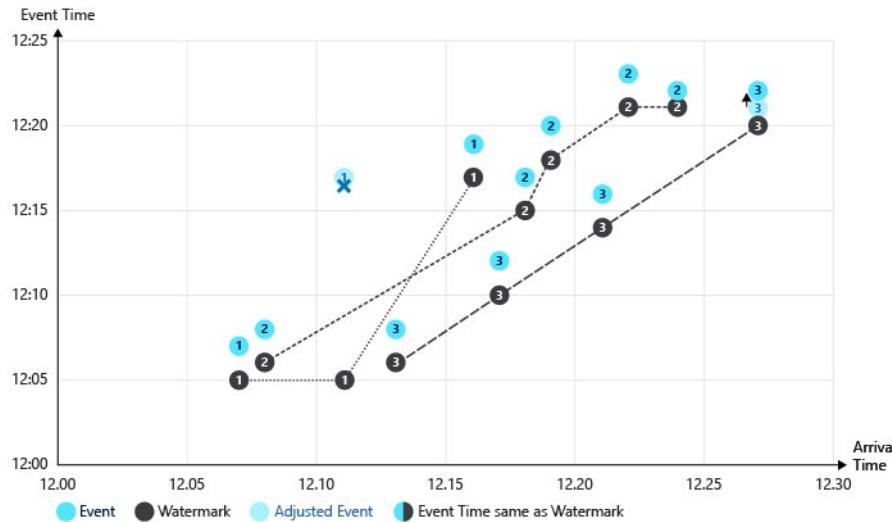
- The fourth event (device3), and fifth event (device1) have aligned times and are processed without adjustment. The watermark progresses on each event.
- When the sixth event (device3) is processed, the arrival time (12:17) and the event time (12:12) is below the watermark level. The event time is adjusted to the water mark level (12:17).
- When the twelfth event (device3) is processed, the arrival time (12:27) is 6 minutes ahead of the event time (12:21). The late arrival policy is applied. The event time is adjusted (12:22), which is above the watermark (12:21) so no further adjustment is applied.

2. Second illustration of watermark progressing without an early arrival policy:



In this example, no early arrival policy is applied. Outlier events that arrive early raise the watermark significantly. Notice the third event (deviceld1 at time 12:11) is not dropped in this scenario, and the watermark is raised to 12:15. The fourth event time is adjusted forward 7 minutes (12:08 to 12:15) as a result.

3. In the final illustration, substreams are used (OVER the Deviceld). Multiple watermarks are tracked, one per stream. There are fewer events with their times adjusted as a result.

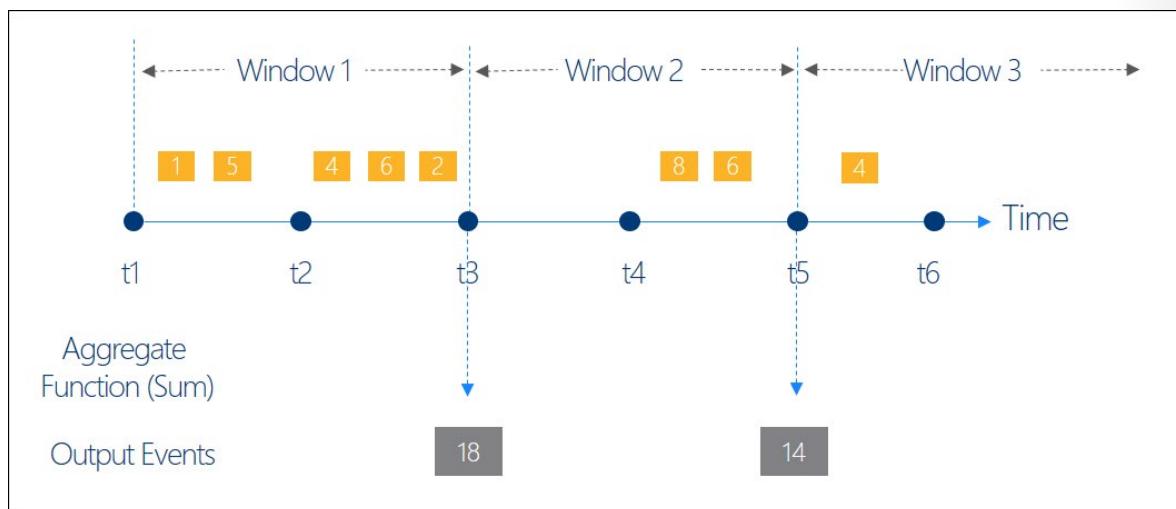


## Introduction to ASA Windowing Functions

In time-streaming scenarios, performing operations on the data contained in temporal windows is a common pattern. Stream Analytics has native support for windowing functions, enabling developers to author complex stream processing jobs with minimal effort.

There are four kinds of temporal windows to choose from: Tumbling, Hopping, Sliding, and Session windows. You use the window functions in the GROUP BY clause of the query syntax in your Stream Analytics jobs. You can also aggregate events over multiple windows using the Windows() function.

All the windowing operations output results at the end of the window. The output of the window will be single event based on the aggregate function used. The output event will have the time stamp of the end of the window and all window functions are defined with a fixed length.



## Tumbling window

Tumbling window functions are used to segment a data stream into distinct time segments and perform a function against them, such as the example below. The key differentiators of a Tumbling window are that they repeat, do not overlap, and an event cannot belong to more than one tumbling window.

**Tell me the count of tweets per time zone every 10 seconds**

A 10-second Tumbling Window

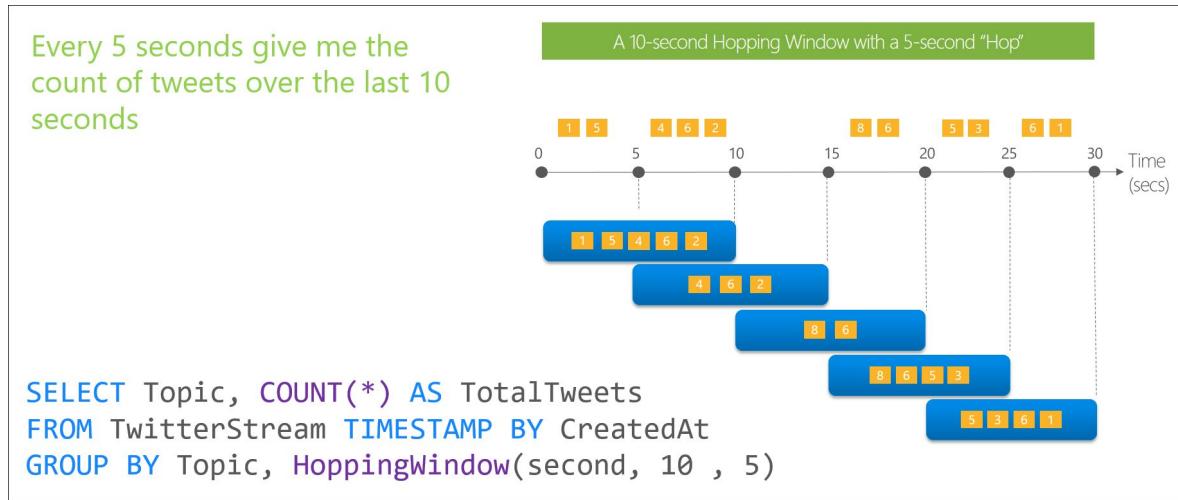
The diagram shows a timeline from 0 to 30 seconds. The timeline is divided into four non-overlapping windows of 10-second duration each. The windows are labeled 0-10, 10-20, 20-30, and 30-35. The events are represented by orange boxes. In the first window (0-10), events are 1, 5, 4, 6, and 2. In the second window (10-20), events are 8 and 6. In the third window (20-30), events are 5, 3, 6, and 1. The fourth window (30-35) is shown but has no visible events. Blue boxes highlight the events within each window, and arrows point from the window labels to the corresponding blue boxes.

```

SELECT TimeZone, COUNT(*) AS Count
FROM TwitterStream TIMESTAMP BY CreatedAt
GROUP BY TimeZone, TumblingWindow(second,10)
    
```

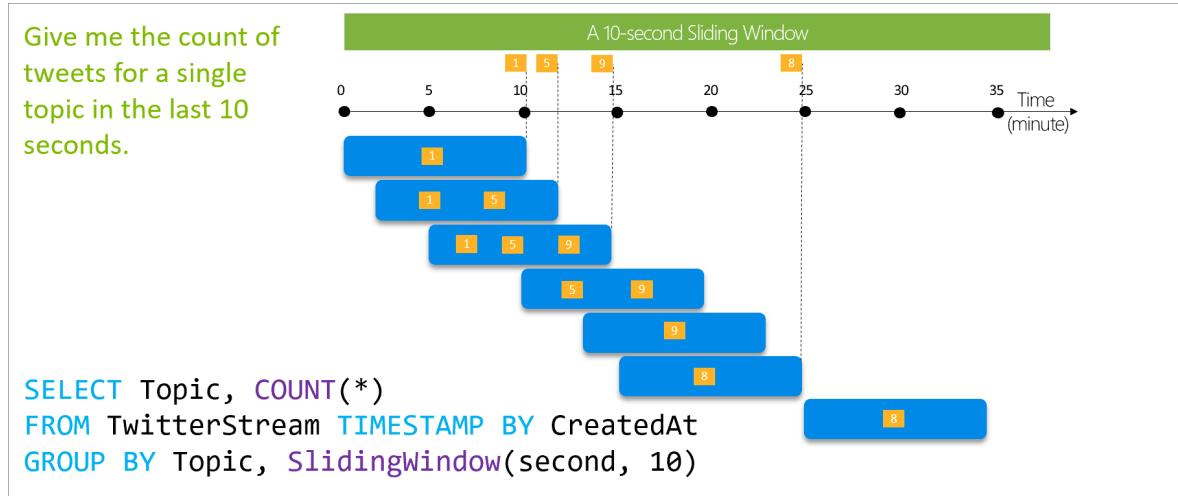
## Hopping window

Hopping window functions hop forward in time by a fixed period. It may be easy to think of them as Tumbling windows that can overlap, so events can belong to more than one Hopping window result set. To make a Hopping window the same as a Tumbling window, specify the hop size to be the same as the window size.



## Sliding window

Sliding window functions, unlike Tumbling or Hopping windows, produce an output only when an event occurs. Every window will have at least one event and the window continuously moves forward by an  $\epsilon$  (epsilon). Like hopping windows, events can belong to more than one sliding window.



## Session window

Session window functions group events that arrive at similar times, filtering out periods of time where there is no data. It has three main parameters: timeout, maximum duration, and partitioning key (optional).

Tell me the count of tweets that occur within 5 minutes to each other.

```

    graph LR
      0((0)) --> 5((5))
      5 --> 9((9))
      9 --> 15((15))
      15 --> 20((20))
      20 --> 25((25))
      25 --> 30((30))
      30 --> 35((35))
      35 --> 40((40))

      subgraph SessionWindows [Session Windows with 5 minutes timeout]
        S1[1, 5]
        S2[9, 15, 20, 25]
        S3[35]
      end
  
```

**SELECT** Topic, **COUNT(\*)**  
**FROM** TwitterStream **TIMESTAMP BY** CreatedAt  
**GROUP BY** Topic, SessionWindow(minute, 5, 10)

A session window begins when the first event occurs. If another event occurs within the specified timeout from the last ingested event, then the window extends to include the new event. Otherwise if no events occur within the timeout, then the window is closed at the timeout.

If events keep occurring within the specified timeout, the session window will keep extending until maximum duration is reached. The maximum duration checking intervals are set to be the same size as the specified max duration. For example, if the max duration is 10, then the checks on if the window exceed maximum duration will happen at t = 0, 10, 20, 30, etc.

When a partition key is provided, the events are grouped together by the key and session window is applied to each group independently. This partitioning is useful for cases where you need different session windows for different users or devices.

## ASA Outputs

Outputs let you store and save the results of the Stream Analytics job. By using the output data, you can do further business analytics and data warehousing of your data.

When you design your Stream Analytics query, refer to the name of the output by using the INTO clause. You can use a single output per job, or multiple outputs per streaming job (if you need them) by providing multiple INTO clauses in the query.

To create, edit, and test Stream Analytics job outputs, you can use the Azure portal, Azure PowerShell, .NET API, REST API, and Visual Studio.

Some outputs types support partitioning. Output batch sizes vary to optimize throughput.

## Azure Data Lake Storage Gen 1

Stream Analytics supports Azure Data Lake Storage Gen 1. Azure Data Lake Storage is an enterprise-wide, hyperscale repository for big data analytic workloads. You can use Data Lake Storage to store data of any size, type, and ingestion speed for operational and exploratory analytics. Stream Analytics needs to be authorized to access Data Lake Storage.

Azure Data Lake Storage output from Stream Analytics is currently not available in the Azure China 21Vianet and Azure Germany (T-Systems International) regions.

The following table lists property names and their descriptions to configure your Data Lake Storage Gen 1 output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to Data Lake Store.
Subscription	The subscription that contains your Azure Data Lake Storage account.
Account name	The name of the Data Lake Store account where you're sending your output. You're presented with a drop-down list of Data Lake Store accounts that are available in your subscription.
Path prefix pattern	<p>The file path that's used to write your files within the specified Data Lake Store account. You can specify one or more instances of the {date} and {time} variables:</p> <p>Example 1: folder1/logs/{date}/{time}</p> <p>Example 2: folder1/logs/{date}</p> <p>The time stamp of the created folder structure follows UTC and not local time.</p> <p>If the file path pattern doesn't contain a trailing slash (/), the last pattern in the file path is treated as a file name prefix.</p> <p>New files are created in these circumstances:</p> <ul style="list-style-type: none"> <li>Change in output schema</li> <li>External or internal restart of a job</li> </ul>
Date format	Optional. If the date token is used in the prefix path, you can select the date format in which your files are organized. Example: YYYY/MM/DD
Time format	Optional. If the time token is used in the prefix path, specify the time format in which your files are organized. Currently the only supported value is HH.
Event serialization format	The serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	If you're using CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time.
Delimiter	Applicable only for CSV serialization. Stream Analytics supports a number of common delimiters for serializing CSV data. Supported values are comma, semicolon, space, tab, and vertical bar.

Property name	Description
Format	Applicable only for JSON serialization. Line separated specifies that the output is formatted by having each JSON object separated by a new line. Array specifies that the output is formatted as an array of JSON objects. This array is closed only when the job stops or Stream Analytics has moved on to the next time window. In general, it's preferable to use line-separated JSON, because it doesn't require any special handling while the output file is still being written to.
Authentication mode	You can authorize access to your Data Lake Storage account using Managed Identity or User token. Once you grant access, you can revoke access by changing the user account password, deleting the Data Lake Storage output for this job, or deleting the Stream Analytics job.

## SQL Database

You can use Azure SQL Database as an output for data that's relational in nature or for applications that depend on content being hosted in a relational database. Stream Analytics jobs write to an existing table in SQL Database. The table schema must exactly match the fields and their types in your job's output. You can also specify Azure SQL Data Warehouse as an output via the SQL Database output option. To learn about ways to improve write throughput, see the Stream Analytics with Azure SQL Database as output article.

You can also use Azure SQL Database Managed Instance as an output. You have to configure public endpoint in Azure SQL Database Managed Instance and then manually configure the following settings in Azure Stream Analytics. Azure virtual machine running SQL Server with a database attached is also supported by manually configuring the settings below.

The following table lists the property names and their description for creating a SQL Database output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to this database.
Database	The name of the database where you're sending your output.
Server name	The SQL Database server name. For Azure SQL Database Managed Instance, it is required to specify the port 3342. For example, sampleserver.public.database.windows.net,3342
Username	The username that has write access to the database. Stream Analytics supports only SQL authentication.
Password	The password to connect to the database.

Property name	Description
Table	The table name where the output is written. The table name is case-sensitive. The schema of this table should exactly match the number of fields and their types that your job output generates.
Inherit partition scheme	An option for inheriting the partitioning scheme of your previous query step, to enable fully parallel topology with multiple writers to the table. For more information, see Azure Stream Analytics output to Azure SQL Database.
Max batch count	The recommended upper limit on the number of records sent with every bulk insert transaction.

## Blob storage and Azure Data Lake Gen2

Data Lake Storage Gen2 makes Azure Storage the foundation for building enterprise data lakes on Azure. Designed from the start to service multiple petabytes of information while sustaining hundreds of gigabits of throughput, Data Lake Storage Gen2 allows you to easily manage massive amounts of data. A fundamental part of Data Lake Storage Gen2 is the addition of a hierarchical namespace to Blob storage.

Azure Blob storage offers a cost-effective and scalable solution for storing large amounts of unstructured data in the cloud. For an introduction on Blob storage and its usage, see [Upload, download, and list blobs with the Azure portal](#).

The following table lists the property names and their descriptions for creating a blob or ADLS Gen2 output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to this blob storage.
Storage account	The name of the storage account where you're sending your output.
Storage account key	The secret key associated with the storage account.
Storage container	A logical grouping for blobs stored in the Azure Blob service. When you upload a blob to the Blob service, you must specify a container for that blob.

MCT USE ONLY. STUDENT USE PROHIBITED

Property name	Description
Path pattern	<p>Optional. The file path pattern that's used to write your blobs within the specified container.</p> <p>In the path pattern, you can choose to use one or more instances of the date and time variables to specify the frequency that blobs are written: {date}, {time}</p> <p>You can use custom blob partitioning to specify one custom {field} name from your event data to partition blobs. The field name is alphanumeric and can include spaces, hyphens, and underscores. Restrictions on custom fields include the following:</p> <ul style="list-style-type: none"> <li>Field names aren't case-sensitive. For example, the service can't differentiate between column "ID" and column "id."</li> <li>Nested fields are not permitted. Instead, use an alias in the job query to "flatten" the field.</li> <li>Expressions can't be used as a field name.</li> </ul> <p>This feature enables the use of custom date/time format specifier configurations in the path. Custom date and time formats must be specified one at a time, enclosed by the {datetime:&lt;specifier&gt;} keyword. Allowable inputs for &lt;specifier&gt; are yyyy, MM, M, dd, d, HH, H, mm, m, ss, or s. The {datetime:&lt;specifier&gt;} keyword can be used multiple times in the path to form custom date/time configurations.</p> <p>Examples:</p> <ul style="list-style-type: none"> <li>Example 1: cluster1/logs/{date}/{time}</li> <li>Example 2: cluster1/logs/{date}</li> <li>Example 3: cluster1/{client_id}/{date}/{time}</li> <li>Example 4: cluster1/{datetime:ss}/{myField} where the query is: SELECT data.myField AS myField FROM Input;</li> <li>Example 5: cluster1/year={datetime:yyyy}/month={datetime:MM}/day={datetime:dd}</li> </ul> <p>The time stamp of the created folder structure follows UTC and not local time.</p> <p>File naming uses the following convention:</p>
Date format	Optional. If the date token is used in the prefix path, you can select the date format in which your files are organized. Example: YYYY/MM/DD

Property name	Description
Time format	Optional. If the time token is used in the prefix path, specify the time format in which your files are organized. Currently the only supported value is HH.
Event serialization format	Serialization format for output data. JSON, CSV, Avro, and Parquet are supported.
Minimum rows (Parquet only)	The number of minimum rows per batch. For Parquet, every batch will create a new file. The current default value is 2,000 rows and the allowed maximum is 10,000 rows.
Maximum time (Parquet only)	The maximum wait time per batch. After this time, the batch will be written to the output even if the minimum rows requirement is not met. The current default value is 1 minute and the allowed maximum is 2 hours. If your blob output has path pattern frequency, the wait time cannot be higher than the partition time range.
Encoding	If you're using CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time.
Delimiter	Applicable only for CSV serialization. Stream Analytics supports a number of common delimiters for serializing CSV data. Supported values are comma, semicolon, space, tab, and vertical bar.
Format	Applicable only for JSON serialization. Line separated specifies that the output is formatted by having each JSON object separated by a new line. Array specifies that the output is formatted as an array of JSON objects. This array is closed only when the job stops or Stream Analytics has moved on to the next time window. In general, it's preferable to use line-separated JSON, because it doesn't require any special handling while the output file is still being written to.

When you're using Blob storage as output, a new file is created in the blob in the following cases:

- If the file exceeds the maximum number of allowed blocks (currently 50,000). You might reach the maximum allowed number of blocks without reaching the maximum allowed blob size. For example, if the output rate is high, you can see more bytes per block, and the file size is larger. If the output rate is low, each block has less data, and the file size is smaller.
- If there's a schema change in the output, and the output format requires fixed schema (CSV and Avro).
- If a job is restarted, either externally by a user stopping it and starting it, or internally for system maintenance or error recovery.
- If the query is fully partitioned, and a new file is created for each output partition.
- If the user deletes a file or a container of the storage account.
- If the output is time partitioned by using the path prefix pattern, and a new blob is used when the query moves to the next hour.

- If the output is partitioned by a custom field, and a new blob is created per partition key if it does not exist.
- If the output is partitioned by a custom field where the partition key cardinality exceeds 8,000, and a new blob is created per partition key.

## Event Hubs

The Azure Event Hubs service is a highly scalable publish-subscribe event ingestor. It can collect millions of events per second. One use of an event hub as output is when the output of a Stream Analytics job becomes the input of another streaming job. For information about the maximum message size and batch size optimization, see the output batch size section.

You need a few parameters to configure data streams from event hubs as an output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to this event hub.
Event hub namespace	A container for a set of messaging entities. When you created a new event hub, you also created an event hub namespace.
Event hub name	The name of your event hub output.
Event hub policy name	The shared access policy, which you can create on the event hub's Configure tab. Each shared access policy has a name, permissions that you set, and access keys.
Event hub policy key	The shared access key that's used to authenticate access to the event hub namespace.
Partition key column	Optional. A column that contains the partition key for event hub output.
Event serialization format	The serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	For CSV and JSON, UTF-8 is the only supported encoding format at this time.
Delimiter	Applicable only for CSV serialization. Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab, and vertical bar.
Format	Applicable only for JSON serialization. Line separated specifies that the output is formatted by having each JSON object separated by a new line. Array specifies that the output is formatted as an array of JSON objects.
Property columns	Optional. Comma-separated columns that need to be attached as user properties of the outgoing message instead of the payload. More information about this feature is in the section Custom metadata properties for output.

MCT USE ONLY. STUDENT USE PROHIBITED

## Power BI

You can use Power BI as an output for a Stream Analytics job to provide for a rich visualization experience of analysis results. You can use this capability for operational dashboards, report generation, and metric-driven reporting.

Power BI output from Stream Analytics is currently not available in the Azure China 21Vianet and Azure Germany (T-Systems International) regions.

The following table lists property names and their descriptions to configure your Power BI output.

Property name	Description
Output alias	Provide a friendly name that's used in queries to direct the query output to this Power BI output.
Group workspace	To enable sharing data with other Power BI users, you can select groups inside your Power BI account or choose My Workspace if you don't want to write to a group. Updating an existing group requires renewing the Power BI authentication.
Dataset name	Provide a dataset name that you want the Power BI output to use.
Table name	Provide a table name under the dataset of the Power BI output. Currently, Power BI output from Stream Analytics jobs can have only one table in a dataset.
Authorize connection	You need to authorize with Power BI to configure your output settings. Once you grant this output access to your Power BI dashboard, you can revoke access by changing the user account password, deleting the job output, or deleting the Stream Analytics job.

For a walkthrough of configuring a Power BI output and dashboard, see the Azure Stream Analytics and Power BI tutorial.

### Note

Don't explicitly create the dataset and table in the Power BI dashboard. The dataset and table are automatically populated when the job is started and the job starts pumping output into Power BI. If the job query doesn't generate any results, the dataset and table aren't created. If Power BI already had a dataset and table with the same name as the one provided in this Stream Analytics job, the existing data is overwritten.

## Create a schema

Azure Stream Analytics creates a Power BI dataset and table schema for the user if they don't already exist. In all other cases, the table is updated with new values. Currently, only one table can exist within a dataset.

Power BI uses the first-in, first-out (FIFO) retention policy. Data will collect in a table until it hits 200,000 rows.

## Convert a data type from Stream Analytics to Power BI

Azure Stream Analytics updates the data model dynamically at runtime if the output schema changes. Column name changes, column type changes, and the addition or removal of columns are all tracked.

This table covers the data type conversions from Stream Analytics data types to Power BI Entity Data Model (EDM) types, if a Power BI dataset and table don't exist.

From Stream Analytics	To Power BI
bigint	Int64
nvarchar(max)	String
datetime	Datetime
float	Double
Record array	String type, constant value "IRecord" or "IArray"

## Update the schema

Stream Analytics infers the data model schema based on the first set of events in the output. Later, if necessary, the data model schema is updated to accommodate incoming events that might not fit into the original schema.

Avoid the SELECT \* query to prevent dynamic schema update across rows. In addition to potential performance implications, it might result in uncertainty of the time taken for the results. Select the exact fields that need to be shown on the Power BI dashboard. Additionally, the data values should be compliant with the chosen data type.

Previous/current	Int64	String		Datetime	Double
Int64	Int64		String	String	Double
Double	Double	String	String	Double	
String	String	String	String	String	
Datetime	String	String	Datetime	String	

## Table storage

Azure Table storage offers highly available, massively scalable storage, so that an application can automatically scale to meet user demand. Table storage is Microsoft's NoSQL key/attribute store, which you can use for structured data with fewer constraints on the schema. Azure Table storage can be used to store data for persistence and efficient retrieval.

The following table lists the property names and their descriptions for creating a table output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to this table storage.
Storage account	The name of the storage account where you're sending your output.
Storage account key	The access key associated with the storage account.
Table name	The name of the table. The table gets created if it doesn't exist.

Property name	Description
Partition key	The name of the output column that contains the partition key. The partition key is a unique identifier for the partition within a table that forms the first part of an entity's primary key. It's a string value that can be up to 1 KB in size.
Row key	The name of the output column that contains the row key. The row key is a unique identifier for an entity within a partition. It forms the second part of an entity's primary key. The row key is a string value that can be up to 1 KB in size.
Batch size	The number of records for a batch operation. The default (100) is sufficient for most jobs. See the Table Batch Operation spec for more details on modifying this setting.

## Service Bus queues

Service Bus queues offer a FIFO message delivery to one or more competing consumers. Typically, messages are received and processed by the receivers in the temporal order in which they were added to the queue. Each message is received and processed by only one message consumer.

The following table lists the property names and their descriptions for creating a queue output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to this Service Bus queue.
Service Bus namespace	A container for a set of messaging entities.
Queue name	The name of the Service Bus queue.
Queue policy name	When you create a queue, you can also create shared access policies on the queue's Configure tab. Each shared access policy has a name, permissions that you set, and access keys.
Queue policy key	The shared access key that's used to authenticate access to the Service Bus namespace.
Event serialization format	The serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	For CSV and JSON, UTF-8 is the only supported encoding format at this time.
Delimiter	Applicable only for CSV serialization. Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab, and vertical bar.
Format	Applicable only for JSON type. Line separated specifies that the output is formatted by having each JSON object separated by a new line. Array specifies that the output is formatted as an array of JSON objects.

Property name	Description
Property columns	Optional. Comma-separated columns that need to be attached as user properties of the outgoing message instead of the payload. More information about this feature is in the section Custom metadata properties for output.
System Property columns	Optional. Key value pairs of System Properties and corresponding column names that need to be attached to the outgoing message instead of the payload. More information about this feature is in the section System properties for Service Bus Queue and Topic outputs

The number of partitions is based on the Service Bus SKU and size. Partition key is a unique integer value for each partition.

## Service Bus Topics

Service Bus queues provide a one-to-one communication method from sender to receiver. Service Bus topics provide a one-to-many form of communication.

The following table lists the property names and their descriptions for creating a Service Bus topic output.

Property name	Description
Output alias	A friendly name used in queries to direct the query output to this Service Bus topic.
Service Bus namespace	A container for a set of messaging entities. When you created a new event hub, you also created a Service Bus namespace.
Topic name	Topics are messaging entities, similar to event hubs and queues. They're designed to collect event streams from devices and services. When a topic is created, it's also given a specific name. The messages sent to a topic aren't available unless a subscription is created, so ensure there's one or more subscriptions under the topic.
Topic policy name	When you create a Service Bus topic, you can also create shared access policies on the topic's Configure tab. Each shared access policy has a name, permissions that you set, and access keys.
Topic policy key	The shared access key that's used to authenticate access to the Service Bus namespace.
Event serialization format	The serialization format for output data. JSON, CSV, and Avro are supported.
Encoding	If you're using CSV or JSON format, an encoding must be specified. UTF-8 is the only supported encoding format at this time.

Property name	Description
Delimiter	Applicable only for CSV serialization. Stream Analytics supports a number of common delimiters for serializing data in CSV format. Supported values are comma, semicolon, space, tab, and vertical bar.
Property columns	Optional. Comma-separated columns that need to be attached as user properties of the outgoing message instead of the payload. More information about this feature is in the section Custom metadata properties for output.
System Property columns	Optional. Key value pairs of System Properties and corresponding column names that need to be attached to the outgoing message instead of the payload. More information about this feature is in the section System properties for Service Bus Queue and Topic outputs

The number of partitions is based on the Service Bus SKU and size. The partition key is a unique integer value for each partition.

## Azure Cosmos DB

Azure Cosmos DB is a globally distributed database service that offers limitless elastic scale around the globe, rich query, and automatic indexing over schema-agnostic data models. To learn about Azure Cosmos DB container options for Stream Analytics, see the Stream Analytics with Azure Cosmos DB as output article.

Azure Cosmos DB output from Stream Analytics is currently not available in the Azure China 21Vianet and Azure Germany (T-Systems International) regions.

### Note

At this time, Azure Stream Analytics only supports connection to Azure Cosmos DB by using the SQL API. Other Azure Cosmos DB APIs are not yet supported. If you point Azure Stream Analytics to the Azure Cosmos DB accounts created with other APIs, the data might not be properly stored.

The following table describes the properties for creating an Azure Cosmos DB output.

Property name	Description
Output alias	An alias to refer this output in your Stream Analytics query.
Sink	Azure Cosmos DB.
Import option	Choose either Select Cosmos DB from your subscription or Provide Cosmos DB settings manually.
Account ID	The name or endpoint URI of the Azure Cosmos DB account.
Account key	The shared access key for the Azure Cosmos DB account.
Database	The Azure Cosmos DB database name.

Property name	Description
Container name	The container name to be used, which must exist in Cosmos DB. Example: MyContainer: A container named "MyContainer" must exist.
Document ID	Optional. The name of the field in output events that's used to specify the primary key on which insert or update operations are based.

## Azure Functions

Azure Functions is a serverless compute service that you can use to run code on-demand without having to explicitly provision or manage infrastructure. It lets you implement code that's triggered by events occurring in Azure or partner services. This ability of Azure Functions to respond to triggers makes it a natural output for Azure Stream Analytics. This output adapter enables users to connect Stream Analytics to Azure Functions, and run a script or piece of code in response to a variety of events.

Azure Functions output from Stream Analytics is currently not available in the Azure China 21Vianet and Azure Germany (T-Systems International) regions.

Azure Stream Analytics invokes Azure Functions via HTTP triggers. The Azure Functions output adapter is available with the following configurable properties:

Property name	Description
Function app	The name of your Azure Functions app.
Function	The name of the function in your Azure Functions app.
Key	If you want to use an Azure Function from another subscription, you can do so by providing the key to access your function.
Max batch size	A property that lets you set the maximum size for each output batch that's sent to your Azure function. The input unit is in bytes. By default, this value is 262,144 bytes (256 KB).
Max batch count	A property that lets you specify the maximum number of events in each batch that's sent to Azure Functions. The default value is 100.

When Azure Stream Analytics receives a 413 ("http Request Entity Too Large") exception from an Azure function, it reduces the size of the batches that it sends to Azure Functions. In your Azure function code, use this exception to make sure that Azure Stream Analytics doesn't send oversized batches. Also, make sure that the maximum batch count and size values used in the function are consistent with the values entered in the Stream Analytics portal.

### Note

During test connection, Stream Analytics sends an empty batch to Azure Functions to test if the connection between the two works. Make sure that your Functions app handles empty batch requests to make sure test connection passes.

Also, in a situation where there's no event landing in a time window, no output is generated. As a result, the computeResult function isn't called. This behavior is consistent with the built-in windowed aggregate functions.

## Custom metadata properties for output

You can attach query columns as user properties to your outgoing messages. These columns don't go into the payload. The properties are present in the form of a dictionary on the output message. Key is the column name and value is the column value in the properties dictionary. All Stream Analytics data types are supported except Record and Array.

Supported outputs:

- Service Bus queue
- Service Bus topic
- Event hub

In the following example, we add the two fields DeviceId and DeviceStatus to the metadata.

- Query: `select *, DeviceId, DeviceStatus from iotHubInput`
- Output configuration: `DeviceId,DeviceStatus`

## System properties for Service Bus Queue and Topic outputs

You can attach query columns as system properties to your outgoing service bus Queue or Topic messages. These columns don't go into the payload instead the corresponding BrokeredMessage system property is populated with the query column values. These system properties are supported - `MessageId`, `ContentType`, `Label`, `PartitionKey`, `ReplyTo`, `SessionId`, `CorrelationId`, `To`, `ForcePersistence`, `TimeToLive`, `ScheduledEnqueueTimeUtc`. String values of these columns are parsed as corresponding system property value type and any parsing failures are treated as data errors. This field is provided as a JSON object format. Details about this format are as follows -

- Surrounded by curly braces {}.
- Written in key/value pairs.
- Keys and values must be strings.
- Key is the system property name and value is the query column name.
- Keys and values are separated by a colon.
- Each key/value pair is separated by a comma.

This shows how to use this property –

- Query: `select *, column1, column2 INTO queueOutput FROM iotHubInput`
- System Property Columns: `{ "MessageId": "column1", "PartitionKey": "column2" }`

This sets the `MessageId` on service bus queue messages with `column1`'s values and `PartitionKey` is set with `column2`'s values.

## Partitioning

The following table summarizes the partition support and the number of output writers for each output type:

Output type	Partitioning support	Partition key	Number of output writers	
Azure Data Lake Store	Yes	Use {date} and {time} tokens in the path prefix pattern. Choose the date format, such as YYYY/MM/DD, DD/MM/YYYY, or MM-DD-YYYY. HH is used for the time format.	Follows the input partitioning for fully parallelizable queries.	
Azure SQL Database	Yes, needs to enabled.	Based on the PARTITION BY clause in the query.	When Inherit Partitioning option is enabled, follows the input partitioning for fully parallelizable queries. To learn more about achieving better write throughput performance when you're loading data into Azure SQL Database, see Azure Stream Analytics output to Azure SQL Database.	
Azure Blob storage	Yes	Use {date} and {time} tokens from your event fields in the path pattern. Choose the date format, such as YYYY/MM/DD, DD/MM/YYYY, or MM-DD-YYYY. HH is used for the time format. Blob output can be partitioned by a single custom event attribute {fieldname} or {datetime:<specifier>}.	Follows the input partitioning for fully parallelizable queries.	

MCT USE ONLY. STUDENT USE PROHIBITED

Output type	Partitioning support	Partition key	Number of output writers	
Azure Event Hubs	Yes	Yes	Varies depending on partition alignment. When the partition key for event hub output is equally aligned with the upstream (previous) query step, the number of writers is the same as the number of partitions in event hub output. Each writer uses the EventHubSender class to send events to the specific partition. When the partition key for event hub output is not aligned with the upstream (previous) query step, the number of writers is the same as the number of partitions in that prior step. Each writer	uses the SendBatchAsync class in EventHubClient to send events to all the output partitions.
Power BI	No	None	Not applicable.	
Azure Table storage	Yes	Any output column.	Follows the input partitioning for fully parallelized queries.	
Azure Service Bus topic	Yes	Automatically chosen. The number of partitions is based on the Service Bus SKU and size. The partition key is a unique integer value for each partition.	Same as the number of partitions in the output topic.	

Output type	Partitioning support	Partition key	Number of output writers	
Azure Service Bus queue	Yes	Automatically chosen. The number of partitions is based on the Service Bus SKU and size. The partition key is a unique integer value for each partition.	Same as the number of partitions in the output queue.	
Azure Cosmos DB	Yes	Based on the PARTITION BY clause in the query.	Follows the input partitioning for fully parallelized queries.	
Azure Functions	Yes	Based on the PARTITION BY clause in the query.	Follows the input partitioning for fully parallelized queries.	

The number of output writers can also be controlled using `INTO <partition count>` clause in your query, which can be helpful in achieving a desired job topology. If your output adapter is not partitioned, lack of data in one input partition will cause a delay up to the late arrival amount of time. In such cases, the output is merged to a single writer, which might cause bottlenecks in your pipeline.

## Output batch size

Azure Stream Analytics uses variable-size batches to process events and write to outputs. Typically the Stream Analytics engine doesn't write one message at a time, and uses batches for efficiency. When the rate of both the incoming and outgoing events is high, Stream Analytics uses larger batches. When the egress rate is low, it uses smaller batches to keep latency low.

The following table explains some of the considerations for output batching:

Output type	Max message size	Batch size optimization
Azure Data Lake Store	See Data Lake Storage limits: <a href="https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#data-lake-store-limits">https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#data-lake-store-limits</a>	Use up to 4 MB per write operation.
Azure SQL Database	Configurable using Max batch count. 10,000 maximum and 100 minimum rows per single bulk insert by default.	Every batch is initially bulk inserted with maximum batch count. Batch is split in half (until minimum batch count) based on retryable errors from SQL.
Azure Blob storage	See Azure Storage limits: <a href="https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#storage-limits">https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#storage-limits</a>	The maximum blob block size is 4 MB. The maximum blob block count is 50,000.

Output type	Max message size	Batch size optimization
Azure Event Hubs	256 KB or 1 MB per message.	When input/output partitioning isn't aligned, each event is packed individually in EventData and sent in a batch of up to the maximum message size. This also happens if custom metadata properties are used.  When input/output partitioning is aligned, multiple events are packed into a single EventData instance, up to the maximum message size, and sent.
Power BI	See Power BI Rest API limits: <a href="https://msdn.microsoft.com/library/dn950053.aspx">https://msdn.microsoft.com/library/dn950053.aspx</a>	
Azure Table storage	See Azure Storage limits: <a href="https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#storage-limits">https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#storage-limits</a>	The default is 100 entities per single transaction. You can configure it to a smaller value as needed.
Azure Service Bus queue	256 KB per message for Standard tier, 1MB for Premium tier.	Use a single event per message.
Azure Service Bus topic	256 KB per message for Standard tier, 1MB for Premium tier.	Use a single event per message.
Azure Cosmos DB	See Azure Cosmos DB limits: <a href="https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#azure-cosmos-db-limits">https://docs.microsoft.com/en-us/azure/azure-subscription-service-limits#azure-cosmos-db-limits</a>	Batch size and write frequency are adjusted dynamically based on Azure Cosmos DB responses. There are no predetermined limitations from Stream Analytics.
Azure Functions		The default batch size is 262,144 bytes (256 KB). The default event count per batch is 100. The batch size is configurable and can be increased or decreased in the Stream Analytics output options.

## Azure Functions

Azure Functions is a solution for easily running small pieces of code, or “functions,” in the cloud. You can write just the code you need for the problem at hand, without worrying about a whole application or the infrastructure to run it. Functions can make development even more productive, and you can use your development language of choice, such as C#, Java, JavaScript, PowerShell, and Python. Pay only for the

time your code runs and trust Azure to scale as needed. Azure Functions lets you develop **serverless**<sup>21</sup> applications on Microsoft Azure.

This topic provides a high-level overview of Azure Functions. If you want to jump right in and get started with Functions, start with **Create your first Azure Function**<sup>22</sup>. If you are looking for more technical information about Functions, see the **developer reference**<sup>23</sup>.

## Features

Here are some key features of Functions:

- Choice of language - Write functions using your choice of C#, Java, Javascript, Python, and other languages. See Supported languages for the complete list.
- Pay-per-use pricing model - Pay only for the time spent running your code. See the Consumption hosting plan option in the pricing section.
- Bring your own dependencies - Functions supports NuGet and NPM, so you can use your favorite libraries.
- Integrated security - Protect HTTP-triggered functions with OAuth providers such as Azure Active Directory, Facebook, Google, Twitter, and Microsoft Account.
- Simplified integration - Easily leverage Azure services and software-as-a-service (SaaS) offerings. See the integrations section for some examples.
- Flexible development - Code your functions right in the portal or set up continuous integration and deploy your code through GitHub, Azure DevOps Services, and other supported development tools.
- Open-source - The Functions runtime is open-source and available on GitHub.

## What can I do with Functions?

Functions is a great solution for processing data, integrating systems, working with the internet-of-things (IoT), and building simple APIs and microservices. Consider Functions for tasks like image or order processing, file maintenance, or for any tasks that you want to run on a schedule.

Functions provides templates to get you started with key scenarios, including the following:

- **HTTPTrigger** - Trigger the execution of your code by using an HTTP request. For an example, see Create your first function.
- **TimerTrigger** - Execute cleanup or other batch tasks on a predefined schedule. For an example, see Create a function triggered by a timer.
- **CosmosDBTrigger** - Process Azure Cosmos DB documents when they are added or updated in collections in a NoSQL database. For more information, see Azure Cosmos DB bindings.
- **BlobTrigger** - Process Azure Storage blobs when they are added to containers. You might use this function for image resizing. For more information, see Blob storage bindings.
- **QueueTrigger** - Respond to messages as they arrive in an Azure Storage queue. For more information, see Azure Queue storage bindings.

<sup>21</sup> <https://azure.microsoft.com/solutions/serverless/>

<sup>22</sup> <https://docs.microsoft.com/en-us/azure/azure-functions/functions-create-first-azure-function>

<sup>23</sup> <https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference>

- EventGridTrigger - Respond to events delivered to a subscription in Azure Event Grid. Supports a subscription-based model for receiving events, which includes filtering. A good solution for building event-based architectures. For an example, see Automate resizing uploaded images using Event Grid.
- EventHubTrigger - Respond to events delivered to an Azure Event Hub. Particularly useful in application instrumentation, user experience or workflow processing, and internet-of-things (IoT) scenarios. For more information, see Event Hubs bindings.
- ServiceBusQueueTrigger - Connect your code to other Azure services or on-premises services by listening to message queues. For more information, see Service Bus bindings.
- ServiceBusTopicTrigger - Connect your code to other Azure services or on-premises services by subscribing to topics. For more information, see Service Bus bindings.

Azure Functions supports triggers, which are ways to start execution of your code, and bindings, which are ways to simplify coding for input and output data. For a detailed description of the triggers and bindings that Azure Functions provides, see Azure Functions triggers and bindings developer reference.

## Integrations

Azure Functions integrates with various Azure and 3rd-party services. These services can trigger your function and start execution, or they can serve as input and output for your code. The following service integrations are supported by Azure Functions:

- Azure Cosmos DB
- Azure Event Hubs
- Azure Event Grid
- Azure Notification Hubs
- Azure Service Bus (queues and topics)
- Azure Storage (blob, queues, and tables)
- On-premises (using Service Bus)
- Twilio (SMS messages)

## How much does Functions cost?

Azure Functions has two kinds of pricing plans. Choose the one that best fits your needs:

- Consumption plan - When your function runs, Azure provides all of the necessary computational resources. You don't have to worry about resource management, and you only pay for the time that your code runs.
- App Service plan - Run your functions just like your web apps. When you are already using App Service for your other applications, you can run your functions on the same plan at no additional cost.

# About the Module 4 Labs

## Module Labs

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

- Lab 7: Device Message Routing

**WARNING:** Be prepared for UI changes

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

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



## Module 5 Insights and Business Integration

### Business Integration for IoT Solutions

#### IoT Developer Role in Business Integration

IoT applications can integrate with technology systems across an organization. Devices, groups of devices, business rules and actions, and access and associations between devices and users are controlled. Important parts of the application backend are the “custom” control logic of the solution, device discovery and visualization, device state management and command execution, as well as device management which controls device life cycle, enables distribution of configuration and software updates, and allows remote control of devices. Business integration is often driven from backend processing systems; i.e. the integration of the IoT environment into downstream business systems such as CRM, ERP, and line-of-business (LOB) applications.

#### Application Backend Processing

Unlike traditional business systems, the business logic of an IoT solution might be spread across different components of the system. Solution device management will commonly use compute nodes, whereas the analytics portion of the solution will be largely implemented directly inside the respective analytics systems.

In some cases, simple solutions may not have independently deployed and managed “business logic” application backend, but the core logic may exist as rule expressions hosted inside stream processing, some of the analytics capabilities, and/or as part of the business workflows and connector components.

#### Roles Contributing to IoT Solution Implementation

In addition to the solution architects who envision and plan the IoT solution, and the data analysts and data engineers who process the data generated by the IoT solution (to establish insights, develop ML models, and propose actions), there are three roles that have primary responsibilities in the implementation of the solution.

- **IoT hardware manufacturer/integrator:** Manufacturers of IoT hardware, integrators assembling hardware from various manufacturers, or suppliers providing hardware for an IoT deployment manu-

factured or integrated by other suppliers. Involved in development and integration of firmware, embedded operating systems, and embedded software.

- **IoT solution developer:** The development of an IoT solution is typically done by a solution developer. This developer may be part of an in-house team or a system integrator specializing in this activity. The IoT solution developer can develop various components of the IoT solution from scratch or integrate various standard or open-source components.
- **IoT solution operator:** After the IoT solution is deployed, it requires long-term operations, monitoring, upgrades, and maintenance. These tasks can be done by an in-house team that consists of information technology specialists, hardware operations and maintenance teams, and domain specialists who monitor the correct behavior of the overall IoT infrastructure.

## Azure IoT Developer Responsibilities

The IoT solution developer is the person who has the greatest working knowledge of the devices and the generated telemetry:

- The Azure IoT Developer is responsible for the implementation and the coding required to create and maintain the cloud and edge portion of an IoT solution. In addition to configuring and maintaining the devices by using cloud services, the IoT Developer also sets up the physical devices. The IoT Developer is responsible for maintaining the devices throughout the life cycle.
- The IoT Developer implements designs for IoT solutions, including device topology, connectivity, debugging and security. The IoT Developer deploys compute/containers and configures device networking. The IoT Developer implements designs for solutions to manage data pipelines, including monitoring and data transformation as it relates to IoT. The IoT Developer works with data engineers and other stakeholders to ensure successful business integration.
- IoT Developers should have a good understanding of Azure services, including data storage options, data analysis, data processing, and platform-as-a-service options. IoT Developers must be able to program in at least one Azure-supported language: C, .NET (C#), Node.js, Python, or Java.

## Azure Messaging Services

Azure offers four services that assist with delivering event and messages throughout a solution. These services are:

- Event Grid
- Event Hubs
- Service Bus
- Azure Storage Queues

Although they have some similarities, each service is designed for particular scenarios. This article describes the differences between these services, and helps you understand which one to choose for your application. In many cases, the messaging services are complementary and can be used together.

## Event vs. message services

There's an important distinction to note between services that deliver an event and services that deliver a message.

## Event

An event is a lightweight notification of a condition or a state change. The publisher of the event has no expectation about how the event is handled. The consumer of the event decides what to do with the notification. Events can be discrete units or part of a series.

Discrete events report state change and are actionable. To take the next step, the consumer only needs to know that something happened. The event data has information about what happened but doesn't have the data that triggered the event. For example, an event notifies consumers that a file was created. It may have general information about the file, but it doesn't have the file itself. Discrete events are ideal for serverless solutions that need to scale.

Series events report a condition and are analyzable. The events are time-ordered and interrelated. The consumer needs the sequenced series of events to analyze what happened.

## Message

A message is raw data produced by a service to be consumed or stored elsewhere. The message contains the data that triggered the message pipeline. The publisher of the message has an expectation about how the consumer handles the message. A contract exists between the two sides. For example, the publisher sends a message with the raw data, and expects the consumer to create a file from that data and send a response when the work is done.

## Comparison of services

Service	Purpose	Type	When to use
Event Grid	Reactive programming	Event distribution (discrete)	React to status changes
Event Hubs	Big data pipeline	Event streaming (series)	Telemetry and distributed data streaming
Service Bus	High-value enterprise messaging	Message	Order processing and financial transactions
Azure Storage Queues	Simple, reliable, persistent messaging within and between services	Message	Very large message stores (over 80 GB), unreliable consumers

## Event Grid

Event Grid is an eventing backplane that enables event-driven, reactive programming. It uses a publish-subscribe model. Publishers emit events, but have no expectation about which events are handled. Subscribers decide which events they want to handle.

Event Grid is deeply integrated with Azure services and can be integrated with third-party services. It simplifies event consumption and lowers costs by eliminating the need for constant polling. Event Grid efficiently and reliably routes events from Azure and non-Azure resources. It distributes the events to registered subscriber endpoints. The event message has the information you need to react to changes in services and applications. Event Grid isn't a data pipeline, and doesn't deliver the actual object that was updated.

Event Grid supports dead-lettering for events that aren't delivered to an endpoint.

It has the following characteristics:

- dynamically scalable

- low cost
- serverless
- at least once delivery

## Event Hubs

Azure Event Hubs is a big data pipeline. It facilitates the capture, retention, and replay of telemetry and event stream data. The data can come from many concurrent sources. Event Hubs allows telemetry and event data to be made available to a variety of stream-processing infrastructures and analytics services. It is available either as data streams or bundled event batches. This service provides a single solution that enables rapid data retrieval for real-time processing as well as repeated replay of stored raw data. It can capture the streaming data into a file for processing and analysis.

It has the following characteristics:

- low latency
- capable of receiving and processing millions of events per second
- at least once delivery

## Service Bus

Service Bus is intended for traditional enterprise applications. These enterprise applications require transactions, ordering, duplicate detection, and instantaneous consistency. Service Bus enables cloud-native applications to provide reliable state transition management for business processes. When handling high-value messages that cannot be lost or duplicated, use Azure Service Bus. Service Bus also facilitates highly secure communication across hybrid cloud solutions and can connect existing on-premises systems to cloud solutions.

Service Bus is a brokered messaging system. It stores messages in a "broker" (for example, a queue) until the consuming party is ready to receive the messages.

It has the following characteristics:

- reliable asynchronous message delivery (enterprise messaging as a service) that requires polling
- advanced messaging features like FIFO, batching/sessions, transactions, dead-lettering, temporal control, routing and filtering, and duplicate detection
- at least once delivery
- optional in-order delivery

## Azure Storage Queues

Azure Queue Storage is a service for storing large numbers of messages. You access messages from anywhere in the world via authenticated calls using HTTP or HTTPS. A queue message can be up to 64 KB in size. A queue may contain millions of messages, up to the total capacity limit of a storage account. Queues are commonly used to create a backlog of work to process asynchronously.

For more information on selecting between the two message solutions, refer to **Storage queues and Service Bus queues - compared and contrasted<sup>1</sup>**.

---

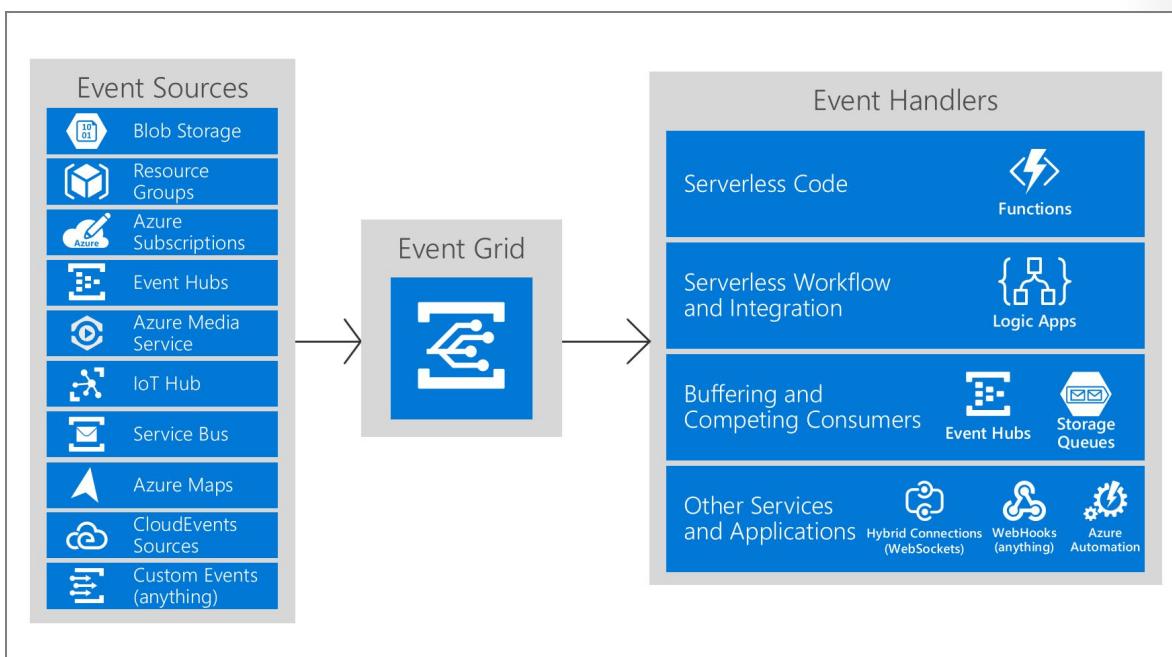
<sup>1</sup> <https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted>

## Business Integration with Event Grid

Azure Event Grid allows you to easily build applications with event-based architectures. First, select the Azure resource you would like to subscribe to, and then give the event handler or WebHook endpoint to send the event to. Event Grid has built-in support for events coming from Azure services, like storage blobs and resource groups. Event Grid also has support for your own events, using custom topics.

You can use filters to route specific events to different endpoints, multicast to multiple endpoints, and make sure your events are reliably delivered.

Azure Event Grid is deployed to maximize availability by natively spreading across multiple fault domains in every region, and across availability zones (in regions that support them). For a list of regions that are supported by Event Grid, see Products available by region <https://azure.microsoft.com/global-infrastructure/services/?products=event-grid&regions=all>.



This image shows how Event Grid connects sources and handlers, and isn't a comprehensive list of supported integrations.

### Event sources

Currently, the following Azure services support sending events to Event Grid:

- Azure Subscriptions (management operations)
- Container Registry
- Custom Topics
- Event Hubs
- IoT Hub
- Media Services
- Resource Groups (management operations)
- Service Bus

- Storage Blob
- Azure Maps

## Event handlers

Currently, the following Azure services support handling events from Event Grid:

- Azure Automation
- Azure Functions
- Event Hubs
- Hybrid Connections
- Logic Apps
- Power Automate (Formerly known as Microsoft Flow)
- Queue Storage
- Service Bus
- WebHooks

## Concepts

There are five concepts in Azure Event Grid that let you get going:

- Events - What happened.
- Event sources - Where the event took place.
- Topics - The endpoint where publishers send events.
- Event subscriptions - The endpoint or built-in mechanism to route events, sometimes to more than one handler. Subscriptions are also used by handlers to intelligently filter incoming events.
- Event handlers - The app or service reacting to the event.

## Events Definition

An event is the smallest amount of information that fully describes something that happened in the system. Every event has common information like: source of the event, time the event took place, and unique identifier. Every event also has specific information that is only relevant to the specific type of event. For example, an event about a new file being created in Azure Storage has details about the file, such as the lastTimeModified value. Or, an Event Hubs event has the URL of the Capture file.

An event of size up to 64 KB is covered by General Availability (GA) Service Level Agreement (SLA). The support for an event of size up to 1 MB is currently in preview. Events over 64 KB are charged in 64-KB increments.

## Event Sources Definition

An event source is where the event happens. Each event source is related to one or more event types. For example, Azure Storage is the event source for blob created events. IoT Hub is the event source for device created events. Your application is the event source for custom events that you define. Event sources are responsible for sending events to Event Grid.

## Topics Definition

The event grid topic provides an endpoint where the source sends events. The publisher creates the event grid topic, and decides whether an event source needs one topic or more than one topic. A topic is used for a collection of related events. To respond to certain types of events, subscribers decide which topics to subscribe to.

System topics are built-in topics provided by Azure services. You don't see system topics in your Azure subscription because the publisher owns the topics, but you can subscribe to them. To subscribe, you provide information about the resource you want to receive events from. As long as you have access to the resource, you can subscribe to its events.

Custom topics are application and third-party topics. When you create or are assigned access to a custom topic, you see that custom topic in your subscription.

When designing your application, you have flexibility when deciding how many topics to create. For large solutions, create a custom topic for each category of related events. For example, consider an application that sends events related to modifying user accounts and processing orders. It's unlikely any event handler wants both categories of events. Create two custom topics and let event handlers subscribe to the one that interests them. For small solutions, you might prefer to send all events to a single topic. Event subscribers can filter for the event types they want.

## Event Subscriptions Definition

A subscription tells Event Grid which events on a topic you're interested in receiving. When creating the subscription, you provide an endpoint for handling the event. You can filter the events that are sent to the endpoint. You can filter by event type, or subject pattern.

The event subscription is automatically expired after that date. Set an expiration for event subscriptions that are only needed for a limited time and you don't want to worry about cleaning up those subscriptions. For example, when creating an event subscription to test a scenario, you might want to set an expiration.

## Event Handlers Definition

From an Event Grid perspective, an event handler is the place where the event is sent. The handler takes some further action to process the event. Event Grid supports several handler types. You can use a supported Azure service or your own webhook as the handler. Depending on the type of handler, Event Grid follows different mechanisms to guarantee the delivery of the event. For HTTP webhook event handlers, the event is retried until the handler returns a status code of 200 – OK. For Azure Storage Queue, the events are retried until the Queue service successfully processes the message push into the queue.

## Additional Concept Considerations

### Security

Event Grid provides security for subscribing to topics, and publishing topics. When subscribing, you must have adequate permissions on the resource or event grid topic. When publishing, you must have a SAS token or key authentication for the topic.

## Event delivery

If Event Grid can't confirm that an event has been received by the subscriber's endpoint, it redelivers the event.

## Batching

When using a custom topic, events must always be published in an array. This can be a batch of one for low-throughput scenarios, however, for high volume use cases, it's recommended that you batch several events together per publish to achieve higher efficiency. Batches can be up to 1 MB. Each event should still not be greater than 64 KB (General Availability) or 1 MB (preview).

## Capabilities

Here are some of the key features of Azure Event Grid:

- Simplicity - Point and click to aim events from your Azure resource to any event handler or endpoint.
- Advanced filtering - Filter on event type or event publish path to make sure event handlers only receive relevant events.
- Fan-out - Subscribe several endpoints to the same event to send copies of the event to as many places as needed.
- Reliability - 24-hour retry with exponential backoff to make sure events are delivered.
- Pay-per-event - Pay only for the amount you use Event Grid.
- High throughput - Build high-volume workloads on Event Grid with support for millions of events per second.
- Built-in Events - Get up and running quickly with resource-defined built-in events.
- Custom Events - Use Event Grid route, filter, and reliably deliver custom events in your app.

## How much does Event Grid cost?

Azure Event Grid uses a pay-per-event pricing model, so you only pay for what you use. The first 100,000 operations per month are free. Operations are defined as event ingress, subscription delivery attempts, management calls, and filtering by subject suffix. For details, see the pricing page <https://azure.microsoft.com/pricing/details/event-grid/>.

## React to IoT Hub events by using Event Grid to trigger actions

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

## Regional availability

The Event Grid integration is available for IoT hubs located in the regions where Event Grid is supported. For the latest list of regions, see An introduction to Azure Event Grid.

## Event types

IoT Hub publishes the following event types:

Event type	Description
Microsoft.Devices.DeviceCreated	Published when a device is registered to an IoT hub.
Microsoft.Devices.DeviceDeleted	Published when a device is deleted from an IoT hub.
Microsoft.Devices.DeviceConnected	Published when a device is connected to an IoT hub.
Microsoft.Devices.DeviceDisconnected	Published when a device is disconnected from an IoT hub.
Microsoft.Devices.DeviceTelemetry	Published when a device telemetry message is sent to an IoT hub

Use either the Azure portal or Azure CLI to configure which events to publish from each IoT hub. For an example, try the tutorial [Send email notifications about Azure IoT Hub events using Logic Apps](#).

## Event schema

IoT Hub events contain all the information you need to respond to changes in your device lifecycle. You can identify an IoT Hub event by checking that the `eventType` property starts with **Microsoft.Devices**. For more information about how to use Event Grid event properties, see the Event Grid event schema.

## Device connected schema

The following example shows the schema of a device connected event:

## Device Telemetry schema

Device telemetry message must be in a valid JSON format with the `contentType` set to **application/json** and `contentEncoding` set to **UTF-8** in the message system properties. Both of these properties are case insensitive. If the content encoding is not set, then IoT Hub will write the messages in base 64 encoded format.

You can enrich device telemetry events before they are published to Event Grid by selecting the endpoint as Event Grid. For more information, see Message Enrichments Overview.

The following example shows the schema of a device telemetry event:

```
[{
  "id": "9af86784-8d40-fe2g-8b2a-bab65e106785",
  "topic": "/SUBSCRIPTIONS/<subscription ID>/RESOURCEGROUPS/<resource group name>/PROVIDERS/MICROSOFT.DEVICES/IOTHUBS/<hub name>",
  "subject": "devices/LogicAppTestDevice",
  "eventType": "Microsoft.Devices.DeviceTelemetry",
  "eventTime": "2019-01-07T20:58:30.48Z",
  "data": {
    "body": {
      "Weather": {
        "Temperature": 900
      },
      "Location": "USA"
    },
    "properties": {
      "Status": "Active"
    },
    "systemProperties": {
      "iothub-content-type": "application/json",
      "iothub-content-encoding": "utf-8",
      "iothub-connection-device-id": "d1",
      "iothub-connection-auth-method": "{\"scope\":\"device\",\"type\":\"sas\",\"issuer\":\"iothub\",\"acceptingIpFilterRule\":null}",
      "iothub-connection-auth-generation-id": "123455432199234570",
      "iothub-enqueuedtime": "2019-01-07T20:58:30.48Z",
      "iothub-message-source": "Telemetry"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Device created schema

The following example shows the schema of a device created event:

```
[{
  "id": "56afc886-767b-d359-d59e-0da7877166b2",
  "topic": "/SUBSCRIPTIONS/<subscription ID>/RESOURCEGROUPS/<resource group name>/PROVIDERS/MICROSOFT.DEVICES/IOTHUBS/<hub name>",
```

```

"subject": "devices/LogicAppTestDevice",
"eventType": "Microsoft.Devices.DeviceCreated",
"eventTime": "2018-01-02T19:17:44.4383997Z",
"data": {
  "twin": {
    "deviceId": "LogicAppTestDevice",
    "etag": "AAAAAAAAAAE=",
    "deviceEtag": "null",
    "status": "enabled",
    "statusUpdateTime": "0001-01-01T00:00:00",
    "connectionState": "Disconnected",
    "lastActivityTime": "0001-01-01T00:00:00",
    "cloudToDeviceMessageCount": 0,
    "authenticationType": "sas",
    "x509Thumbprint": {
      "primaryThumbprint": null,
      "secondaryThumbprint": null
    },
    "version": 2,
    "properties": {
      "desired": {
        "$metadata": {
          "$lastUpdated": "2018-01-02T19:17:44.4383997Z"
        },
        "$version": 1
      },
      "reported": {
        "$metadata": {
          "$lastUpdated": "2018-01-02T19:17:44.4383997Z"
        },
        "$version": 1
      }
    }
  },
  "hubName": "egtesthub1",
  "deviceId": "LogicAppTestDevice"
},
"dataVersion": "1",
"metadataVersion": "1"
}]

```

For a detailed description of each property, see Azure Event Grid event schema for IoT Hub.

## Filter events

IoT Hub event subscriptions can filter events based on event type, data content and subject, which is the device name.

Event Grid enables filtering on event types, subjects and data content. While creating the Event Grid subscription, you can choose to subscribe to selected IoT events. Subject filters in Event Grid work based

on **Begins With** (prefix) and **Ends With** (suffix) matches. The filter uses an AND operator, so events with a subject that match both the prefix and suffix are delivered to the subscriber.

The subject of IoT Events uses the format:

devices/{deviceId}

Event Grid also allows for filtering on attributes of each event, including the data content. This allows you to choose what events are delivered based contents of the telemetry message. Please see advanced filtering to view examples. For filtering on the telemetry message body, you must set the `contentType` to **application/json** and `contentEncoding` to **UTF-8** in the message **system properties**<sup>2</sup>. Both of these properties are case insensitive.

For non-telemetry events like DeviceConnected, DeviceDisconnected, DeviceCreated and DeviceDeleted, the Event Grid filtering can be used when creating the subscription. For telemetry events, in addition to the filtering in Event Grid, users can also filter on device twins, message properties and body through the message routing query.

When you subscribe to telemetry events via Event Grid, IoT Hub creates a default message route to send data source type device messages to Event Grid. For more information about message routing, see IoT Hub message routing. This route will be visible in the portal under IoT Hub > Message Routing. Only one route to Event Grid is created regardless of the number of EG subscriptions created for telemetry events. So, if you need several subscriptions with different filters, you can use the OR operator in these queries on the same route. The creation and deletion of the route is controlled through subscription of telemetry events via Event Grid. You cannot create or delete a route to Event Grid using IoT Hub Message Routing.

To filter messages before telemetry data is sent, you can update your routing query. Note that routing query can be applied to the message body only if the body is JSON. You must also set the `contentType` to **application/json** and `contentEncoding` to **UTF-8** in the message **system properties**<sup>3</sup>.

## Limitations for device connected and device disconnected events

To receive device connection state events, a device must do either a 'D2C Send Telemetry' OR a 'C2D Receive Message' operation with IoT Hub. However, note that if a device is using AMQP protocol to connect with IoT Hub, it is recommended that they do a 'C2D Receive Message' operation otherwise their connection state notifications may be delayed by few minutes. If your device is using MQTT protocol, IoT Hub will keep the C2D link open. For AMQP, you can open the C2D link by calling the **Receive Async API**<sup>4</sup>, for IoT Hub C# SDK, or device client for AMQP.

The D2C link is open if you are sending telemetry.

If the device connection is flickering, which means the device connects and disconnects frequently, we will not send every single connection state, but will publish the current connection state taken at a periodic snapshot, till the flickering continues. Receiving either the same connection state event with different sequence numbers or different connection state events both mean that there was a change in the device connection state.

<sup>2</sup> <https://docs.microsoft.com/azure/iot-hub/iot-hub-devguide-routing-query-syntax#system-properties>

<sup>3</sup> <https://docs.microsoft.com/azure/iot-hub/iot-hub-devguide-routing-query-syntax#system-properties>

<sup>4</sup> <https://docs.microsoft.com/dotnet/api/microsoft.azure.devices.client.deviceclient.receiveasync?view=azure-dotnet>

## Tips for consuming events

Applications that handle IoT Hub events should follow these suggested practices:

- Multiple subscriptions can be configured to route events to the same event handler, so don't assume that events are from a particular source. Always check the message topic to ensure that it comes from the IoT hub that you expect.
- Don't assume that all events you receive are the types that you expect. Always check the eventType before processing the message.
- Messages can arrive out of order or after a delay. Use the etag field to understand if your information about objects is up-to-date for device created or device deleted events.

## Message Routing or Event Grid Integration?

Azure IoT Hub provides the capability to stream data from your connected devices and integrate that data into your business applications. IoT Hub offers two methods for integrating IoT events into other Azure services or business applications.

1. **IoT Hub message routing:** This IoT Hub feature enables users to route device-to-cloud messages to service endpoints like Azure Storage containers, Event Hubs, Service Bus queues, and Service Bus topics. Routing also provides a querying capability to filter the data before routing it to the endpoints. In addition to device telemetry data, you can also send non-telemetry events that can be used to trigger actions.
2. **IoT Hub integration with Event Grid:** Azure Event Grid is a fully managed event routing service that uses a publish-subscribe model. IoT Hub and Event Grid work together to integrate IoT Hub events into Azure and non-Azure services, in near-real time. IoT Hub publishes device events, which are generally available, and now also publishes telemetry events, which is in public preview.

## Differences

While both message routing and Event Grid enable alert configuration, there are some key differences between the two. Refer to the following table for details:

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Device messages and events	Yes, message routing can be used for telemetry data, report device twin changes, device lifecycle events, and digital twin change events (part of the IoT Plug and Play public preview).	Yes, Event Grid can be used for telemetry data but can also report when devices are created, deleted, connected, and disconnected from IoT Hub
Ordering	Yes, ordering of events is maintained.	No, order of events is not guaranteed.

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Filtering	Rich filtering on message application properties, message system properties, message body, device twin tags, and device twin properties. Filtering isn't applied to digital twin change events.	Filtering based on event type, subject type and attributes in each event. When subscribing to telemetry events, you can apply additional filters on the data to filter on message properties, message body and device twin in your IoT Hub, before publishing to Event Grid.
Endpoints	Event Hubs Azure Blob Storage Service Bus queue Service Bus topics  Paid IoT Hub SKUs (S1, S2, and S3) are limited to 10 custom endpoints. 100 routes can be created per IoT Hub.	Azure Functions Azure Automation Event Hubs Logic Apps Storage Blob Custom Topics Queue Storage Microsoft Flow Third-party services through WebHooks  500 endpoints per IoT Hub are supported.
Cost	There is no separate charge for message routing. Only ingress of telemetry into IoT Hub is charged. For example, if you have a message routed to three different endpoints, you are billed for only one message.	There is no charge from IoT Hub. Event Grid offers the first 100,000 operations per month for free, and then \$0.60 per million operations afterwards.

## Similarities

IoT Hub message routing and Event Grid have similarities too, some of which are detailed in the following table:

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Maximum message size	256 KB, device-to-cloud	256 KB, device-to-cloud
Reliability	High: Delivers each message to the endpoint at least once for each route. Expires all messages that are not delivered within one hour.	High: Delivers each message to the webhook at least once for each subscription. Expires all events that are not delivered within 24 hours.
Scalability	High: Optimized to support millions of simultaneously connected devices sending billions of messages.	High: Capable of routing 10,000,000 events per second per region.

Feature	IoT Hub message routing	IoT Hub integration with Event Grid
Latency	Low: Near-real time.	Low: Near-real time.
Send to multiple endpoints	Yes, send a single message to multiple endpoints.	Yes, send a single message to multiple endpoints.
Security	IoT Hub provides per-device identity and revocable access control.	Event Grid provides validation at three points: event subscriptions, event publishing, and webhook event delivery.

## How to choose

IoT Hub message routing and the IoT Hub integration with Event Grid perform different actions to achieve similar results. They both take information from your IoT Hub solution and pass it on so that other services can react. So how do you decide which one to use? Consider the following questions to help guide your decision:

- What kind of data are you sending to the endpoints?

Use IoT Hub message routing when you have to send telemetry data to other services. Message routing also enables querying message application and system properties, message body, device twin tags, and device twin properties.

The IoT Hub integration with Event Grid works with events that occur in the IoT Hub service. These IoT Hub events include telemetry data, device created, deleted, connected, and disconnected. When subscribing to telemetry events, you can apply additional filters on the data to filter on message properties, message body and device twin in your IoT Hub, before publishing to Event Grid.

- What endpoints need to receive this information?

IoT Hub message routing supports limited number of unique endpoints and endpoint types, but you can build connectors to reroute the data and events to additional endpoints.

The IoT Hub integration with Event Grid supports 500 endpoints per IoT Hub and a larger variety of endpoint types. It natively integrates with Azure Functions, Logic Apps, Storage and Service Bus queues, and also works with webhooks to extend sending data outside of the Azure service ecosystem and into third-party business applications.

- Does it matter if your data arrives in order?

IoT Hub message routing maintains the order in which messages are sent, so that they arrive in the same way.

Event Grid does not guarantee that endpoints will receive events in the same order that they occurred. For those cases in which absolute order of messages is significant and/or in which a consumer needs a trustworthy unique identifier for messages, we recommend using message routing.

## Introduction to Azure Logic Apps

Azure Logic Apps is a cloud service that helps you schedule, automate, and orchestrate tasks, business processes, and workflows when you need to integrate apps, data, systems, and services across enterprises or organizations. Logic Apps simplifies how you design and build scalable solutions for app integration, data integration, system integration, enterprise application integration (EAI), and business-to-business (B2B) communication, whether in the cloud, on premises, or both.

For example, here are just a few workloads you can automate with logic apps:

- Process and route orders across on-premises systems and cloud services.
- Send email notifications with Office 365 when events happen in various systems, apps, and services.
- Move uploaded files from an SFTP or FTP server to Azure Storage.
- Monitor tweets for a specific subject, analyze the sentiment, and create alerts or tasks for items that need review.

To build enterprise integration solutions with Azure Logic Apps, you can choose from a growing gallery with hundreds of ready-to-use connectors, which include services such as Azure Service Bus, Azure Functions, Azure Storage, SQL Server, Office 365, Dynamics, Salesforce, BizTalk, SAP, Oracle DB, file shares, and more. Connectors provide triggers, actions, or both for creating logic apps that securely access and process data in real time.

## How Logic Apps work

Every logic app workflow starts with a trigger, which fires when a specific event happens, or when new available data meets specific criteria. Many triggers provided by the connectors in Logic Apps include basic scheduling capabilities so that you can set up how regularly your workloads run. For more complex scheduling or advanced recurrences, you can use a Recurrence trigger as the first step in any workflow. Learn more about schedule-based workflows.

Each time that the trigger fires, the Logic Apps engine creates a logic app instance that runs the actions in the workflow. These actions can also include data conversions and flow controls, such as conditional statements, switch statements, loops, and branching. For example, this logic app starts with a Dynamics 365 trigger with the built-in criteria "When a record is updated". If the trigger detects an event that matches this criteria, the trigger fires and runs the workflow's actions. Here, these actions include XML transformation, data updates, decision branching, and email notifications.

You can build your logic apps visually with the Logic Apps Designer, which is available in the Azure portal through your browser and in Visual Studio. For more custom logic apps, you can create or edit logic app definitions in JavaScript Object Notation (JSON) by working in the "code view" editor. You can also use Azure PowerShell commands and Azure Resource Manager templates for select tasks. Logic apps deploy and run in the cloud on Azure.

## Why to use Logic Apps

With businesses moving toward digitization, logic apps help you connect legacy, modern, and cutting-edge systems more easily and quickly by providing prebuilt APIs as Microsoft-managed connectors. That way, you can focus on your apps' business logic and functionality. You don't have to worry about building, hosting, scaling, managing, maintaining, and monitoring your apps. Logic Apps handles these concerns for you. Plus, you pay only for what you use based on a consumption pricing model.

In many cases, you won't have to write code. But if you must write some code, you can create code snippets with Azure Functions and run that code on-demand from logic apps. Also, if your logic apps need to interact with events from Azure services, custom apps, or other solutions, you can use Azure Event Grid with your logic apps for event monitoring, routing, and publishing.

Logic Apps, Functions, and Event Grid are fully managed by Microsoft Azure, which frees you from worries about building, hosting, scaling, managing, monitoring, and maintaining your solutions. With the capability to create "serverless" apps and solutions, you can just focus on the business logic. These services automatically scale to meet your needs, make integrations faster, and help you build robust

cloud apps with minimal code. Plus, you pay only for what you use, based on a consumption pricing model.

## Connectors for Azure Logic Apps

Connectors provide quick access from Azure Logic Apps to events, data, and actions across other apps, services, systems, protocols, and platforms. By using connectors in your logic apps, you expand the capabilities for your cloud and on-premises apps to perform tasks with the data that you create and already have.

**Note:** To integrate with a service or API that doesn't have connector, you can either directly call the service over a protocol such as HTTP or create a custom connector.

Connectors are available either as built-in triggers and actions or as managed connectors:

- **Built-ins:** These built-in triggers and actions are “native” to Azure Logic Apps and help you create logic apps that run on custom schedules, communicate with other endpoints, receive and respond to requests, and call Azure functions, Azure API Apps (Web Apps), your own APIs managed and published with Azure API Management, and nested logic apps that can receive requests. You can also use built-in actions that help you organize and control your logic app's workflow, and also work with data.
- **Managed connectors:** Deployed and managed by Microsoft, these connectors provide triggers and actions for accessing cloud services, on-premises systems, or both, including Office 365, Azure Blob Storage, SQL Server, Dynamics, Salesforce, SharePoint, and more. Some connectors specifically support business-to-business (B2B) communication scenarios and require an integration account that's linked to your logic app. Before using certain connectors, you might have to first create connections, which are managed by Azure Logic Apps.

## Components of a Connector

Each connector offers a set of operations classified as ‘Actions’ and ‘Triggers’. Once you connect to the underlying service, these operations can be easily leveraged within your apps and workflows.

### Actions

Actions are changes directed by a user. For example, you would use an action to look up, write, update, or delete data in a SQL database. All actions directly map to operations defined in the Swagger.

### Triggers

Several connectors provide triggers that can notify your app when specific events occur. For example, the FTP connector has the OnUpdatedFile trigger. You can build either a Logic App or Flow that listens to this trigger and performs an action whenever the trigger fires.

There are two types of trigger.

- **Polling Triggers:** These triggers call your service at a specified frequency to check for new data. When new data is available, it causes a new run of your workflow instance with the data as input.
- **Push Triggers:** These triggers listen for data on an endpoint, that is, they wait for an event to occur. The occurrence of this event causes a new run of your workflow instance.

## Built-ins

Logic Apps provides built-in triggers and actions so you can create schedule-based workflows, help your logic apps communicate with other apps and services, control the workflow through your logic apps, and manage or manipulate data.

Built-in triggers include the following:

Trigger	Description
Recurrence	<ul style="list-style-type: none"> <li>- Run your logic app on a specified schedule, ranging from basic to complex recurrences, with the Recurrence trigger.</li> <li>- Pause your logic app for a specified duration with the Delay action.</li> <li>- Pause your logic app until the specified date and time with the Delay until action.</li> </ul>
HTTP	Communicate with any endpoint over HTTP with both triggers and actions for HTTP, HTTP + Swagger, and HTTP + Webhook.
Request	<ul style="list-style-type: none"> <li>- Make your logic app callable from other apps or services, trigger on Event Grid resource events, or trigger on responses to Azure Security Center alerts with the Request trigger.</li> <li>- Send responses to an app or service with the Response action.</li> </ul>
Batch messages	<ul style="list-style-type: none"> <li>- Process messages in batches with the Batch messages trigger.</li> <li>- Call logic apps that have existing batch triggers with the Send messages to batch action.</li> </ul>
Azure Functions	Call Azure functions that run custom code snippets (C# or Node.js) from your logic apps.
Azure API Management	Call triggers and actions defined by your own APIs that you manage and publish with Azure API Management.
Azure App Services	Call Azure API Apps, or Web Apps, hosted on Azure App Service. The triggers and actions defined by these apps appear like any other first-class triggers and actions when Swagger is included.
Azure Logic Apps	Call other logic apps that start with a Request trigger.

## Control workflow

Logic Apps provides built-in actions for structuring and controlling the actions in your logic app's workflow:

Action	Description
Condition	Evaluate a condition and run different actions based on whether the condition is true or false.

Action	Description
For each	Perform the same actions on every item in an array.
Scope	Group actions into scopes, which get their own status after the actions in the scope finish running.
Switch	Group actions into cases, which are assigned unique values except for the default case. Run only that case whose assigned value matches the result from an expression, object, or token. If no matches exist, run the default case.
Terminate	Stop an actively running logic app workflow.
Until	Repeat actions until the specified condition is true or some state has changed.

## Manage or manipulate data

Logic Apps provides built-in actions for working with data outputs and their formats:

Action	Description
Data Operations	<p>Perform operations with data:</p> <ul style="list-style-type: none"> <li>- Compose: Create a single output from multiple inputs with various types.</li> <li>- Create CSV table: Create a comma-separated-value (CSV) table from an array with JSON objects.</li> <li>- Create HTML table: Create an HTML table from an array with JSON objects.</li> <li>- Filter array: Create an array from items in another array that meet your criteria.</li> <li>- Join: Create a string from all items in an array and separate those items with the specified delimiter.</li> <li>- Parse JSON: Create user-friendly tokens from properties and their values in JSON content so you can use those properties in your workflow.</li> <li>- Select: Create an array with JSON objects by transforming items or values in another array and mapping those items to specified properties.</li> </ul>

Action	Description
Date Time	<p>Perform operations with timestamps:</p> <ul style="list-style-type: none"> <li>- Add to time: Add the specified number of units to a timestamp.</li> <li>- Convert time zone: Convert a timestamp from the source time zone to the target time zone.</li> <li>- Current time: Return the current timestamp as a string.</li> <li>- Get future time: Return the current timestamp plus the specified time units.</li> <li>- Get past time: Return the current timestamp minus the specified time units.</li> <li>- Subtract from time: Subtract a number of time units from a timestamp.</li> </ul>
Variables	<p>Perform operations with variables:</p> <ul style="list-style-type: none"> <li>- Append to array variable: Insert a value as the last item in an array stored by a variable.</li> <li>- Append to string variable: Insert a value as the last character in a string stored by a variable.</li> <li>- Decrement variable: Decrease a variable by a constant value.</li> <li>- Increment variable: Increase a variable by a constant value.</li> <li>- Initialize variable: Create a variable and declare its data type and initial value.</li> <li>- Set variable: Assign a different value to an existing variable.</li> </ul>

## Managed API connectors

Logic Apps provides these popular Standard connectors for automating tasks, processes, and workflows with these services or systems.

Connector	Description
Azure Service Bus	Manage asynchronous messages, sessions, and topic subscriptions with the most commonly used connector in Logic Apps.
SQL Server	Connect to your SQL Server on premises or an Azure SQL Database in the cloud so you can manage records, run stored procedures, or perform queries.
Office 365 Outlook	Connect to your Office 365 email account so you can create and manage emails, tasks, calendar events and meetings, contacts, requests, and more.
Azure Blob Storage	Connect to your storage account so you can create and manage blob content.

Connector	Description
SFTP	Connect to SFTP servers you can access from the internet so you can work with your files and folders.
SharePoint Online	Connect to SharePoint Online so you can manage files, attachments, folders, and more.
Dynamics 365 CRM Online	Connect to your Dynamics 365 account so you can create and manage records, items, and more.
FTP	Connect to FTP servers you can access from the internet so you can work with your files and folders.
Salesforce	Connect to your Salesforce account so you can create and manage items such as records, jobs, objects, and more.
Twitter	Connect to your Twitter account so you can manage tweets, followers, your timeline, and more. Save your tweets to SQL, Excel, or SharePoint.
Azure Event Hubs	Consume and publish events through an Event Hub. For example, get output from your logic app with Event Hubs, and then send that output to a real-time analytics provider.
Azure Event Grid	Monitor events published by an Event Grid, for example, when Azure resources or third-party resources change.

## On-premises connectors

Logic Apps provides Standard connectors for accessing data and resources in on-premises systems. Before you can create a connection to an on-premises system, you must first download, install, and set up an on-premises data gateway. This gateway provides a secure communication channel without having to set up the necessary network infrastructure.

## Integration account connectors

Logic Apps provides Standard connectors for building business-to-business (B2B) solutions with your logic apps when you create and pay for an integration account, which is available through the Enterprise Integration Pack (EIP) in Azure. With this account, you can create and store B2B artifacts such as trading partners, agreements, maps, schemas, certificates, and so on. To use these artifacts, associate your logic apps with your integration account. If you currently use BizTalk Server, these connectors might seem familiar already.

## Triggers and Actions

Connectors can provide triggers, actions, or both. A trigger is the first step in any logic app, usually specifying the event that fires the trigger and starts running your logic app. For example, the FTP connector has a trigger that starts your logic app "when a file is added or modified". Some triggers regularly check for the specified event or data and then fire when they detect the specified event or data. Other triggers wait but fire instantly when a specific event happens or when new data is available. Triggers also pass along any required data to your logic app. Your logic app can read and use that data throughout the

workflow. For example, the Twitter connector has a trigger, "When a new tweet is posted", that passes the tweet's content into your logic app's workflow.

After a trigger fires, Azure Logic Apps creates an instance of your logic app and starts running the actions in your logic app's workflow. Actions are the steps that follow the trigger and perform tasks in your logic app's workflow. For example, you can create a logic app that gets customer data from a SQL database and process that data in later actions.

Here are the general kinds of triggers that Azure Logic Apps provides:

- Recurrence trigger: This trigger runs on a specified schedule and isn't tightly associated with a particular service or system.
- Polling trigger: This trigger regularly polls a specific service or system based on the specified schedule, checking for new data or whether a specific event happened. If new data is available or the specific event happened, the trigger creates and runs a new instance of your logic app, which can now use the data that's passed as input.
- Push trigger: This trigger waits and listens for new data or for an event to happen. When new data is available or when the event happens, the trigger creates and runs new instance of your logic app, which can now use the data that's passed as input.

## Connector configuration

Each connector's triggers and actions provide their own properties for you to configure. Many connectors also require that you first create a connection to the target service or system and provide authentication credentials or other configuration details before you can use a trigger or action in your logic app. For example, you must authorize a connection to a Twitter account for accessing data or to post on your behalf.

For connectors that use Azure Active Directory (Azure AD) OAuth, creating a connection means signing into the service, such as Office 365, Salesforce, or GitHub, where your access token is encrypted and securely stored in an Azure secret store. Other connectors, such as FTP and SQL, require a connection that has configuration details, such as the server address, username, and password. These connection configuration details are also encrypted and securely stored. Learn more about encryption in Azure.

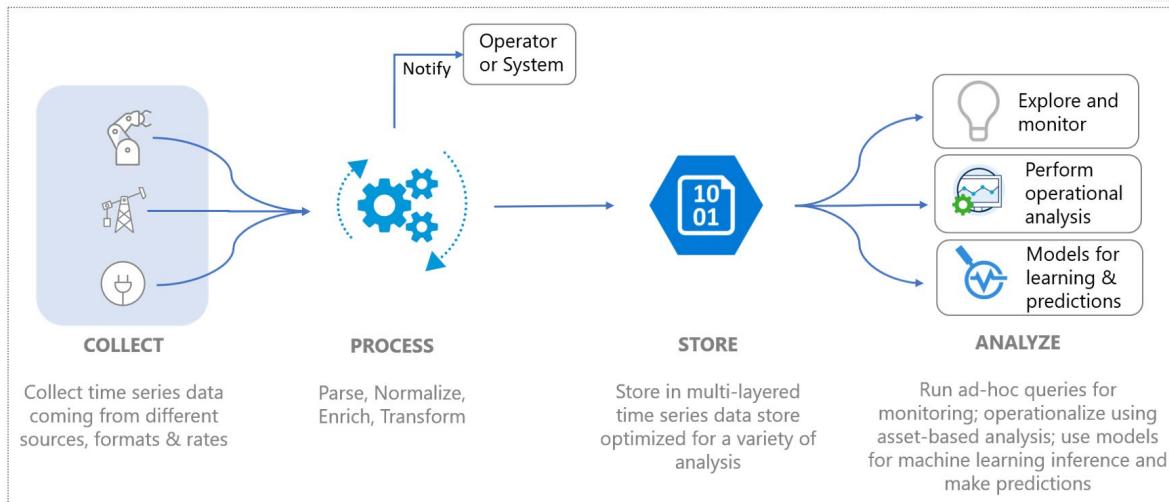
Connections can access the target service or system for as long as that service or system allows. For services that use Azure AD OAuth connections, such as Office 365 and Dynamics, Azure Logic Apps refreshes access tokens indefinitely. Other services might have limits on how long Azure Logic Apps can use a token without refreshing. Generally, some actions invalidate all access tokens, such as changing your password.

# Data Visualization with Time Series Insights

## What is Azure Time Series Insights?

Azure Time Series Insights is built to store, visualize, and query large amounts of time series data, such as that generated by IoT devices. If you want to store, manage, query, or visualize time series data in the cloud, Time Series Insights is likely right for you.

The following image shows a typical IoT data flow.



Time Series Insights has four key jobs:

- It's fully integrated with cloud gateways like Azure IoT Hub and Azure Event Hubs. It easily connects to these event sources and parses JSON from messages and structures that have data in clean rows and columns. It joins metadata with telemetry, and then indexes your data in a columnar store.
- Time Series Insights manages the storage of your data. To make sure that data is always easily accessible, it stores your data in memory and SSDs for up to 400 days. You can interactively query billions of events in seconds—on demand.
- Time Series Insights provides out-of-the-box visualization through the Time Series Insights explorer.
- Time Series Insights provides a query service, both in the Time Series Insights explorer and by using APIs that are easy to integrate to embed your time series data into custom applications.

If you build an application for internal consumption or for external customers to use, you can use Time Series Insights as a back end. You can use it to index, store, and aggregate time series data. To build a custom visualization and user experience on top, use the Client SDK. Time Series Insights is also equipped with several Query APIs to enable these customized scenarios.

Time series data represents how an asset or process changes over time. Time series data is indexed by timestamps, and time is the most meaningful axis along which such data is organized. Time series data typically arrives in sequential order, so it's treated as an insert rather than an update to your database.

It can be a challenge to store, index, query, analyze, and visualize time series data in large volumes. Azure Time Series Insights captures and stores every new event as a row, and change is efficiently measured over time. As a result, you can look backwards to draw insights from the past to help predict future change.

## Primary scenarios

- Store time series data in a scalable way.

At its core, Time Series Insights has a database designed with time series data in mind. Because it's scalable and fully managed, Time Series Insights handles the work of storing and managing events.

- Explore data in near real time.

Time Series Insights provides an explorer that visualizes all data that streams into an environment. Shortly after you connect to an event source, you can view, explore, and query event data within Time Series Insights. The data helps you to validate whether a device emits data as expected and to monitor an IoT asset for health, productivity, and overall effectiveness.

- Perform root-cause analysis and detect anomalies.

Time Series Insights has tools like patterns and perspective views to conduct and save multistep root-cause analysis. Time Series Insights also works with alerting services like Azure Stream Analytics so that you can view alerts and detected anomalies in near real time in the Time Series Insights explorer.

- Gain a global view of time series data that streams from disparate locations for multi-asset or site comparison.

You can connect multiple event sources to a Time Series Insights environment. This way you can view data that streams in from multiple, disparate locations together in near real time. Users can take advantage of this visibility to share data with business leaders. They can collaborate better with domain experts who can apply their expertise to help solve problems, apply best practices, and share learnings.

- Build a customer application on top of Time Series Insights.

Time Series Insights exposes REST Query APIs that you can use to build applications that use time series data.

## Capabilities

- **Get started quickly:** Azure Time Series Insights doesn't require upfront data preparation, so you can quickly connect to millions of events in your IoT hub or event hub. After you connect, you can visualize and interact with sensor data to quickly validate your IoT solutions. You can interact with your data without writing code, and you don't need to learn a new language. Time Series Insights provides a granular, free-text query surface for advanced users, and point and click exploration.
- **Near real-time insights:** Time Series Insights can ingest millions of sensor events per day, with one-minute latency. Time Series Insights helps you gain insights into your sensor data. Use it to spot trends and anomalies, conduct root-cause analyses, and avoid costly downtime. Cross-correlation between real-time and historical data helps you find hidden trends in the data.
- **Build custom solutions:** Embed Azure Time Series Insights data into your existing applications. You also can create new custom solutions with the Time Series Insights REST APIs. Create personalized views you can share for others to explore your insights.
- **Scalability:** Time Series Insights is designed to support IoT at scale. It can ingress from 1 million to 100 million events per day, with a default retention span of 31 days. You can visualize and analyze live data streams in near real time, alongside historical data.

# Why use Azure Time Series Insights?

Azure Time Series Insights Preview is an end-to-end platform-as-a-service (PaaS) offering. You can use it to collect, process, store, analyze, and query data at an Internet of Things (IoT) scale.

## Definition of IoT data

Industrial IoT data in asset-intensive organizations often lacks structural consistency due to the varied nature of devices and sensors in an industrial setting. Data from these streams are characterized by significant gaps, and sometimes corrupted messages, and false readings. IoT data is often meaningful in the context of additional data inputs that come from first-party or third-party sources, such as CRM or ERP that add context for end-to-end workflows. Inputs from third-party data sources such as weather data can help augment telemetry streams in a given installation.

In some industrial IoT scenarios, only a fraction of the data gets used for operational and business purposes, and analysis often requires contextualization of the data in order to be effective. Industrial data is often historicized for in-depth analysis over longer time spans to understand and correlate trends.

Turning collected IoT data into actionable insights requires:

- Data processing to clean, filter, interpolate, transform, and prepare data for analysis.
- A structure to navigate through and understand the data, that is, to normalize and contextualize the data.
- Cost-effective storage for long or infinite retention of processed (or derived) data and raw data.

Such data provides consistent, comprehensive, current, and correct information for business analysis and reporting.

## Azure Time Series Insights for industrial IoT

The IoT landscape is diverse with customers spanning a variety of industry segments including manufacturing, automotive, energy, utilities, smart buildings, and consulting. Across this broad range of industrial IoT market, cloud-native solutions that provide comprehensive analytics targeted at large-scale IoT data are still evolving.

Azure Time Series Insights addresses this market need by providing a turnkey, end-to-end IoT analytics solution with rich semantic modeling for contextualization of time series data, asset-based insights, and best-in-class user experience for discovery, trending, anomaly detection and operational intelligence.

A rich operational analytics platform combined with our interactive data exploration capabilities, you can use Time Series Insights to derive more value out of data collected from IoT assets. The preview offering supports:

- Multi-layered storage solution with warm and cold analytics support providing customers the option to route data between warm and cold for interactive analytics over warm data as well as operational intelligence over decades of historical data.
  - A highly interactive warm analytics solution to perform frequent, and large number of queries over shorter time span data
  - A scalable, performant, and cost optimized time series data lake based on Azure Storage allowing customers to trend years' worth of time series data in seconds.
- Semantic model support that describes the domain and metadata associated with the derived and raw signals from assets and devices.

- Flexible analytics platform to store historical time series data in customer-owned Azure Storage account, thereby allowing customers to have ownership of their IoT data. Data is stored in open source Apache Parquet format that enables connectivity and interop across a variety of data scenarios including predictive analytics, machine learning, and other custom computations done using familiar technologies including Spark, Databricks, and Jupyter.
- Rich analytics with enhanced query APIs and user experience that combines asset-based data insights with rich, ad hoc data analytics with support for interpolation, scalar and aggregate functions, categorical variables, scatter plots, and time shifting time series signals for in-depth analysis.
- Enterprise grade platform to support the scale, performance, security, and reliability needs of our enterprise IoT customers.
- Extensibility and integration support for end-to-end analytics. Time Series Insights provides an extensible analytics platform for a variety of data scenarios. Time Series Insights Power BI connector enables customers to take the queries they do in Time Series Insights directly into Power BI to get unified view of their BI and time series analytics in a single pane of glass.

## Features and Benefits

Azure Time Series Insights provides a scalable pay-as-you-go pricing model for data processing, storage (data and metadata), and query, enabling customers to tune their usage to suit their business demands.

With the introduction of these key industrial IoT capabilities, Time Series Insights also provides the following key benefits.

Benefit	Description
Multilayered storage for IoT-scale time series data	With a shared data processing pipeline for ingesting data, you can ingest data into both warm and cold stores. Use warm store for interactive queries and cold store for storing large volumes of data. To learn more about how to take advantage of high-performing asset-based queries, see queries.
Time Series Model to contextualize raw telemetry and derive asset-based insights	You can use the time series model to create instances, hierarchies, types, and variables for your time series data. To learn more about Time Series Model, see Time Series Model.
Smooth and continuous integration with other data solutions	Data in Time Series Insights cold store is stored in open-source Apache Parquet files. This enables data integration with other data solutions, 1st or 3rd party, for scenarios that include business intelligence, advanced machine learning, and predictive analytics.
Near real-time data exploration	The Azure Time Series Insights Preview explorer user experience provides visualization for all data streaming through the ingestion pipeline. After you connect an event source, you can view, explore, and query event data. In this way, you can validate whether a device emits data as expected. You also can monitor an IoT asset for health, productivity, and overall effectiveness.

Benefit	Description
Extensibility and integration	The Azure Time Series Insights Power BI Connector integration is available directly in the Time Series Explorer user experience through the Export option, allowing customers to export the time series queries they create in our user experience directly into the Power BI desktop and view their time series charts alongside other BI analytics. This opens the door to a new class of scenarios for industrial IoT enterprises who have invested in Power BI by providing a single pane of glass over analytics from various data sources including IoT time series.
Custom applications built on the Time Series Insights platform	Time Series Insights supports the JavaScript SDK. The SDK provides rich controls and simplified access to queries. Use the SDK to build custom IoT applications on top of Time Series Insights to suit your business needs. You also can use the Time Series Insights Query APIs directly to drive data into custom IoT applications.

## Configure the TSI Environment

Azure Time Series Insights is a fully managed analytics, storage, and visualization service that makes it incredibly simple to explore and analyze billions of IoT events simultaneously. Time Series Insights gives you a global view of your data, letting you quickly validate your IoT solution and avoid costly downtime to mission-critical devices by helping you discover hidden trends, spot anomalies, and conduct root-cause analyses in near real-time.

- Find actionable insights in seconds
- Start in seconds, scale in minutes
- Create a global view of your IoT-scale data
- Leverage Time Series Insights in your Apps and Solutions

## Create Your TSI Resource

You can create a new Azure Time Series Insights resource in the Azure portal using the following steps:

1. Search the Azure Marketplace for "Time Series Insights".

Azure Marketplace [See all](#)

Get started  
Recently created  
AI + Machine Learning  
Analytics  
Blockchain  
Compute  
Containers  
Databases  
Developer Tools  
DevOps  
Identity  
Integration  
**Internet of Things**  
Media  
Mixed Reality  
IT & Management Tools  
Networking  
Software as a Service (SaaS)  
Security  
Storage  
Web

Featured [See all](#)

IoT Central Application	<a href="#">Learn more</a>
IoT Hub	<a href="#">Quickstart tutorial</a>
IoT Hub Device Provisioning Service	<a href="#">Quickstart tutorial</a>
PREVIEW Digital Twins (preview)	<a href="#">Learn more</a>
Time Series Insights	<a href="#">Quickstart tutorial</a>
Stream Analytics job	<a href="#">Quickstart tutorial</a>
Machine Learning Studio (classic) Workspace	<a href="#">Learn more</a>
PREVIEW Data Box Edge / Data Box Gateway (preview)	<a href="#">Learn more</a>
Event Grid Topic	<a href="#">Learn more</a>
Function App	<a href="#">Quickstart tutorial</a>

- On the Create Time Series Insights environment blade, fill in the parameters on the Basic tab

Parameter	Action
Environment name	Enter a unique name for the Azure Time Series Insights Preview environment.
Subscription	Enter the subscription where you want to create the Azure Time Series Insights Preview environment. A best practice is to use the same subscription as the rest of the IoT resources that are created by the device simulator.
Resource group	Select an existing resource group or create a new resource group for the Azure Time Series Insights Preview environment resource. A resource group is a container for Azure resources. A best practice is to use the same resource group as the other IoT resources that are created by the device simulator.

Parameter	Action
Location	Select a data center region for your Azure Time Series Insights Preview environment. To avoid additional latency, it's best to create your Azure Time Series Insights Preview environment in the same region as your IoT hub created by the device simulator.
Tier	Select PAYG (pay-as-you-go). This is the SKU for the Azure Time Series Insights Preview product.
Property ID	Enter a value that uniquely identifies your time series instance. The value you enter in the Property ID box cannot be changed later. When the data source is an IoT Hub, iothub-connection-device-id is often used. To learn more about Time Series ID, see Best practices for choosing a Time Series ID.
Storage account name	Enter a globally unique name for a new storage account.
Enable warm store	Select Yes to enable warm store.
Data retention (in days)	Choose the default option of 7 days.

**Create Time Series Insights environment**  
Microsoft

[Basics](#) [Event Source](#) [Review + Create](#)

Create a Time Series Insights environment that you'll use to explore and query time series data. [Learn more](#)

**ENVIRONMENT DETAILS**

Choose the subscription that will house your new environment. Use resource groups to organize and manage resources in that subscription. Note that these details can't be edited after they're saved.

Environment name * ⓘ	MyTutorial
Subscription * ⓘ	My Subscription
Resource group * ⓘ	My Resource Group <a href="#">Create new</a>
Location * ⓘ	Location

**PRICING**

Choose a pricing tier. If you aren't sure which tier to choose, [visit our pricing page](#) to learn more.

Tier * ⓘ	S1    S2 <b>PAYG (Preview)</b>
Capacity ⓘ	Ingress rate: Scalable based on usage needs. <a href="#">View current limits</a> . Storage capacity: Subject to <a href="#">Azure Storage limits</a> . Estimated cost: Varies depending on usage. <a href="#">View metered pricing information</a> .

MCT USE ONLY. STUDENT USE PROHIBITED

**TIME SERIES ID**

**Time Series ID** acts as a partition key for your data and as a primary key for your time series model. It is important that you specify the appropriate Time Series ID during environment creation, since you can't change it later.

\* Property name  ✓ Delete

For example, deviceId, objectId or a tag name

**STORAGE CONFIGURATION**

Configure a cold store for long term durable storage with pay-as-you-go pricing. Optionally, enable a warm store if you need faster and unlimited queries over the most recent data. [Learn more](#)

**Cold store**

Creates a new Azure Storage resource in the subscription and region you've chosen for the TSI environment. You will incur data storage and transaction charges for the data that Time Series Insights reads and writes to this storage resource. [Learn more](#)

Storage account name \*  ✓

Storage account replication \*

**Warm store**

Creates a warm store for the TSI environment optimized for higher query performance and unlimited queries. The warm store can be removed from the environment at a later time. By selecting "Yes", you will incur data storage costs. [Learn more](#)

Enable warm store \*  Yes  No

Data retention time (in days)

**Review + create** Next: Event Source » Download a template for automation

Once you have this information entered, navigate to the Event Source tab.

### 3. Fill in the parameters on the Event Source tab

Parameter	Action
Create an event source?	Select Yes.
Name	Enter a unique value for the event source name.
Source type	Select IoT Hub.
Select a hub	Choose Select existing.
Subscription	Select the subscription that you are using for this course.
IoT Hub name	Select the IoT hub name that you are using for this course.
IoT Hub access policy	Select iothubowner.
IoT Hub consumer group	Select New, enter a unique name, and then select Add. The consumer group must be a unique value in Azure Time Series Insights Preview.

MCT USE ONLY. STUDENT USE PROHIBITED

Parameter	Action
Timestamp property	This value is used to identify the Timestamp property in your incoming telemetry data. Time Series Insights defaults to the incoming timestamp from IoT Hub.

Home > New > Create Time Series Insights environment

## Create Time Series Insights environment

Basics Event Source Review + Create

An event source is the IoT Hub or Event Hub that feeds data into your Time Series Insights environment. [Learn more](#).

**EVENT SOURCE DETAILS**

Create an event source?  Yes  No

Name \*

Source type \*

Select a hub \*

Subscription \*

IoT Hub name \*

IoT Hub access policy name \*

**CONSUMER GROUP**

**Note:** This consumer group should be used exclusively for this event source as there can be only one active reader from a given consumer group at a time.

IoT Hub consumer group \*

**TIMESTAMP**

Create an event source timestamp property name. If you don't enter a value, we'll use the message enqueued time from the event source. [Learn more](#).

Property name

**Review + create** [« Previous: Basics](#) [Download a template for automation](#)

Once you have this information entered, navigate to the Review + Create tab.

4. Use the Review + Create tab to ensure that you entered values correctly, and then click Create to deploy your TSI service.

You have access to your Azure Time Series Insights Preview environment by default if you are an owner of the Azure subscription. You can verify that you have access by opening your TSI service and checking to see that your credentials are listed on the Data Access Policies blade.

## TSI Connection to IoT Hub

Once you have IoT Hub and TSI instances up and running, you are ready to create a dedicated consumer group in the IoT hub for the Time Series Insights environment to consume from. Each Time Series Insights event source must have its own dedicated consumer group that isn't shared with any other consumer. If multiple readers consume events from the same consumer group, all readers are likely to see failures.

## Create an IoT Hub Consumer Group for TSI

Applications use consumer groups to pull data from Azure IoT Hub. To reliably read data from your IoT hub, provide a dedicated consumer group that's used only by this Time Series Insights environment.

The screenshot shows the 'Events' configuration page for an IoT Hub named 'ContosolotHub'. The left sidebar lists various settings like Overview, Activity log, and Built-in endpoints, with 'Built-in endpoints' highlighted by a red box. The main area shows configuration for the 'Events' endpoint. It includes fields for Partitions (set to 4), Event Hub-compatible name (auto-generated), Event Hub-compatible endpoint (auto-generated), Retain for (1 day), and Consumer Groups. The 'Consumer Groups' section is highlighted with a red box, showing a dropdown set to '\$Default' and a 'Create new consumer group' button. Below this, there are sections for Cloud to device messaging with fields for Default TTL (1 hour), Feedback retention time (1 hour), and Maximum delivery count (10 attempts).

Under **Consumer groups**, enter a unique name for the consumer group. You will use this same name in your Time Series Insights environment when you create your event source.

## Create a TSI Event Source for IoT Hub

Switching over to the TSI side, you need to create the Event Source that you will be using to access IoT Hub data.

To begin the process of creating your new event source, you will first provide an Event source name (a name that's unique to this Time Series Insights environment) and specify that your Source will be an IoT Hub. Once you have these properties set, you will have a choice between using an IoT Hub from an available subscription and providing IoT Hub settings manually. The property settings requirements will be different based on your choice.

**Note:** If you want to choose advanced options, you should choose the **Provide IoT Hub settings manually** option.

## Use IoT Hub from Available Subscriptions

The following table describes the properties that are required for the **Use IoT Hub from available subscriptions** option:

Property	Description
Subscription	The subscription the desired IoT hub belongs to.
IoT hub name	The name of the selected IoT hub.
IoT hub policy name	Select the shared access policy. You can find the shared access policy on the IoT hub settings tab. Each shared access policy has a name, permissions that you set, and access keys. The shared access policy for your event source must have service connect permissions.
IoT hub policy key	The key is prepopulated.
IoT hub consumer group	The consumer group that reads events from the IoT hub. We highly recommend that you use a dedicated consumer group for your event source.
Event serialization format	Currently, JSON is the only available serialization format. The event messages must be in this format or no data can be read.

Property	Description
Timestamp property name	To determine this value, you need to understand the message format of the message data that's sent to the IoT hub. This value is the name of the specific event property in the message data that you want to use as the event timestamp. The value is case-sensitive. If left blank, the event enqueue time in the event source is used as the event timestamp.

## Provide IoT Hub settings manually

The following table describes the properties that are required for the **Use IoT Hub from available subscriptions** option:

Property	Description
Subscription ID	The subscription the desired IoT hub belongs to.
Resource group	The resource group name in which the IoT hub was created.
IoT hub name	The name of your IoT hub. When you created your IoT hub, you entered a name for the IoT hub.
IoT hub policy name	The shared access policy. You can create the shared access policy on the IoT hub settings tab. Each shared access policy has a name, permissions that you set, and access keys. The shared access policy for your event source must have service connect permissions.
IoT hub policy key	The shared access key that's used to authenticate access to the Azure Service Bus namespace. Enter the primary or secondary key here.
IoT hub consumer group	The consumer group that reads events from the IoT hub. We highly recommend that you use a dedicated consumer group for your event source.
Event serialization format	Currently, JSON is the only available serialization format. The event messages must be in this format or no data can be read.
Timestamp property name	To determine this value, you need to understand the message format of the message data that's sent to the IoT hub. This value is the name of the specific event property in the message data that you want to use as the event timestamp. The value is case-sensitive. If left blank, the event enqueue time in the event source is used as the event timestamp.

# Data Visualization with Power BI

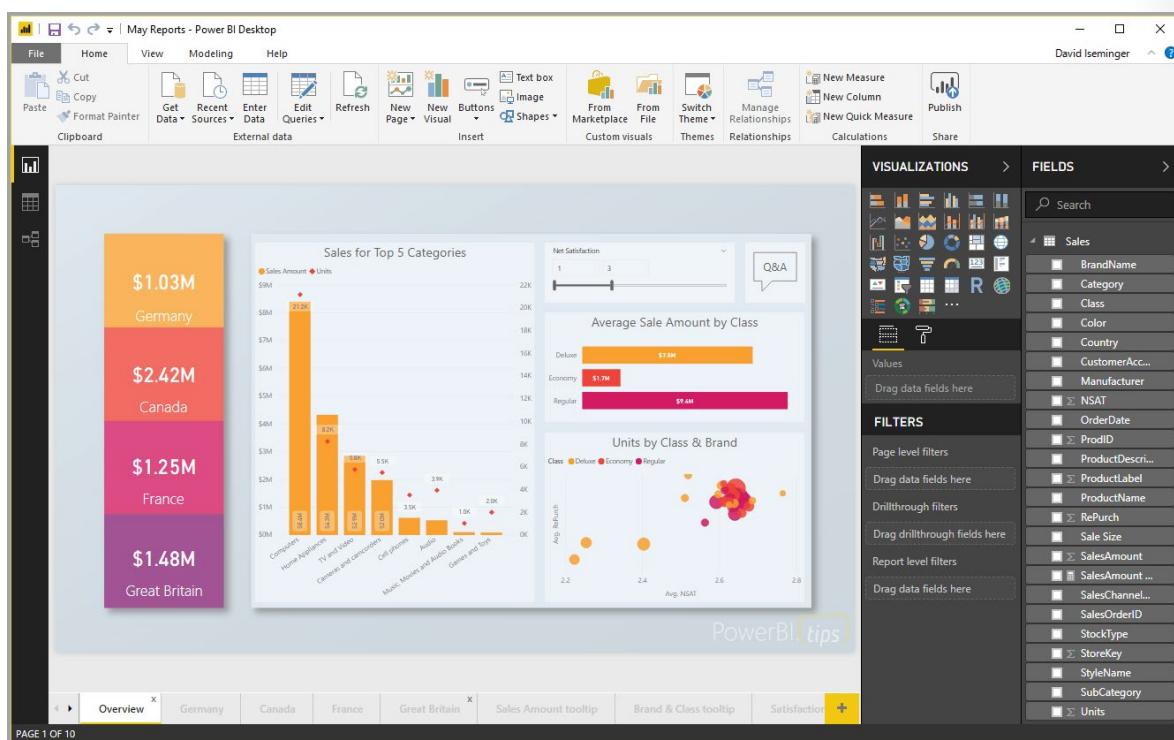
## What is Power BI Desktop?

Power BI is a business analytics tool with rich visualization capabilities that allows you to share insights and results across your organization.

Power BI Desktop is a free application you can install on your local computer that lets you connect to, transform, and visualize your data. With Power BI Desktop, you can connect to multiple different sources of data, and combine them (often called modeling) into a data model that lets you build visuals, and collections of visuals you can share as reports, with other people inside your organization. Most users who work on Business Intelligence projects use Power BI Desktop to create reports, and then use the Power BI service to share their reports with others.

The most common uses for Power BI Desktop are the following:

- Connect to data
- Transform and clean that data, to create a data model
- Create visuals, such as charts or graphs, that provide visual representations of the data
- Create reports that are collections of visuals, on one or more report pages
- Share reports with others using the Power BI service



## Views in Power BI Desktop

There are three views in Power BI Desktop, shown along the left side of the canvas. The views, shown in the order they appear, are the following:

- Report View - this is where you create reports and visuals, and where most of your creation time is spent.
- Data View - here you can see the tables, measures, and other data used in the data model associated with your report, and transform the data for best use in the report's model.
- Model View - in this view you see and manage the relationships among tables in your data model.

## Connect to data

To get started with Power BI Desktop, the first step is to connect to data. There are many different data sources you can connect to from Power BI Desktop. To connect to data, simply select the Home ribbon, then select Get Data > More.

## Transform and clean data, create a model

In Power BI Desktop, you can clean and transform data using the built-in Query Editor. With Query Editor you can make changes to your data, such as changing a data type, removing columns, or combining data from multiple sources. It's a little bit like sculpting - you can start with a large block of clay (or data), then shave pieces off or add others as needed, until the shape of the data is how you want it.

## Create visuals

Once you have a data model, you can drag fields onto the report canvas to create visuals. A visual is a graphic representation of the data in your model.

## Create reports

More often, you'll want to create a collection of visuals that show various aspects of the data you have used to create your model in Power BI Desktop. A collection of visuals, in one Power BI Desktop file, is called a report. A report can have one or more pages, just like an Excel file can have one or more worksheets. In the following image you see the first page of a Power BI Desktop report, named Overview (you can see the tab near the bottom of the image). In this report, there are ten pages.

## Share reports

Once a report is ready to share with others, you can Publish the report to the Power BI service, and make it available to anyone in your organization who has a Power BI license. To publish a Power BI Desktop report, you select the Publish button from the Home ribbon in Power BI Desktop.

## Connect to Azure IoT Data Sources

Data types are organized in the following categories:

- All
- File
- Database

- Power BI
- Azure
- Online Services
- Other

## Azure Data Connections

The **Azure** category provides the following data connections:

- Azure SQL Database
- Azure SQL Data Warehouse
- Azure Analysis Services database
- Azure Blob Storage
- Azure Table Storage
- Azure Cosmos DB
- Azure Data Lake Storage Gen2 (Beta)
- Azure Data Lake Storage Gen1
- Azure HDInsight (HDFS)
- Azure HDInsight Spark
- HDInsight Interactive Query
- Azure Data Explorer (Kusto)

## Connect to data

To connect to your data source, select **Get Data** on the **Home** ribbon. The Get Data window appears, where you can choose from the many different data sources to which Power BI Desktop can connect.

Once you have selected your data source, Power BI Desktop opens the Navigator window, where you can choose which data you would like to load into Power BI Desktop.

After you have selected the data that will be used for visualizations, a Fields pane is used to access the data.

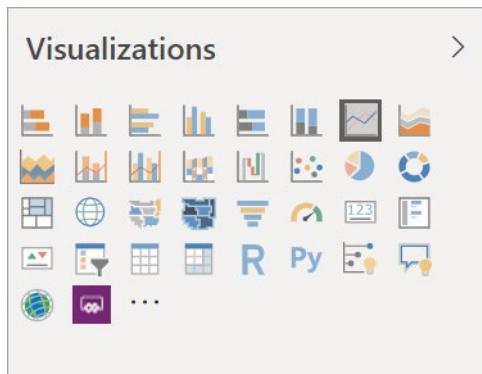
## Data Visualization in Power BI

Visualizations (known as visuals for short) display insights that have been discovered in the data. A Power BI report might have a single page with one visual or it might have pages full of visuals. In Power BI, visuals can be pinned from reports to dashboards.

It's important to make the distinction between report designers and report consumers. If you are the person building or modifying the report, then you are a designer. Designers have edit permissions to the report and its underlying dataset. In Power BI Desktop, this means you can open the dataset in Data view and create visuals in Report view. In Power BI service, this means you can open the data set or report in the report editor in Editing view. If a report or dashboard has been shared with you, you are a report consumer. You'll be able to view and interact with the report and its visuals but you won't be able to make as many changes as a designer can.

## Power BI Visualizations

There are many different visual types available directly from the Power BI Visualizations pane.



Visualization	Description
Area charts: Basic (Layered) and Stacked	The Basic Area chart is based on the line chart with the area between the axis and line filled in.
Bar and column charts	Bar charts are the standard for looking at a specific value across different categories.
Cards: Multi row Cards: Single number	A Card is used when it is important to track a single number or value.
Combo charts	A Combo chart combines a column chart and a line chart. Choose from Line and Stacked Column and Line and Clustered Column.
Doughnut charts	Doughnut charts are similar to Pie charts. They show the relationship of parts to a whole.
Funnel charts	Funnels help visualize a process that has stages and items flow sequentially from one stage to the next. Use a funnel when there is a sequential flow between stages, such as a sales process that starts with leads and ends with purchase fulfillment.
Gauge charts	Displays current status in the context of a goal.
Key influencers chart	A key influencer chart displays the major contributors to a selected result or value.
KPIs	Displays progress toward a measurable goal.
Line charts	Emphasize the overall shape of an entire series of values, usually over time.
Maps: Basic maps	Used to associate both categorical and quantitative information with spatial locations.
Maps: ArcGIS maps	Use when you want to layer data over a regional map. For example, layer symbols that represent demographic measures over a regional area to show value densities
Maps: Filled maps (Choropleth)	The more intense the color, the larger the value.
Maps: Shape maps	Compares regions by color.

MCT USE ONLY. STUDENT USE PROHIBITED

Visualization	Description
Matrix	A table supports two dimensions, but a matrix makes it easier to display data meaningfully across multiple dimensions – it supports a stepped layout. The matrix automatically aggregates the data and enables drill down.
Pie charts	Pie charts show the relationship of parts to a whole.
Q&A visual	Similar to the Q&A experience on dashboards, the Q&A visual lets you ask questions about your data using natural language.
R script visuals	Visuals created with R scripts, commonly called R visuals, can present advanced data shaping and analytics such as forecasting, using the rich analytics and visualization power of R. R visuals can be created in Power BI Desktop and published to the Power BI service.
Ribbon chart	Ribbon charts show which data category has the highest rank (largest value). Ribbon charts are effective at showing rank change, with the highest range (value) always displayed on top for each time period.
Scatter and Bubble charts	Display relationships between 2 (scatter) or 3 (bubble) quantitative measures – whether or not, in which order, etc.
Scatter-high density	Too many data points on a visual can bog it down, so a sophisticated sampling algorithm is used to ensure the accuracy of the visualization.
Slicers	A slicer is used to narrow the portion of the dataset shown in other visualizations in a report.
Standalone images	Used to display a static image, such as a company logo
Tables	Work well with quantitative comparisons among items where there are many categories.
Treemaps	Charts of colored rectangles, with size representing value. They can be hierarchical, with rectangles nested within the main rectangles.
Waterfall charts	Waterfall charts show a running total as values are added or subtracted.

## About the Module 5 Labs

### Module Labs

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

- Lab 8: Visualize Data in Power BI
- Lab 9: Integrate IoT Hub with Event Grid
- Lab 10: Explore and Analyze Data with Time Series Insights

**WARNING:** Be prepared for UI changes

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

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

## Module 6 Azure IoT Edge Deployment Process

### Introduction to Azure IoT Edge

#### What is Azure IoT Edge?

Azure IoT Edge moves cloud analytics and custom business logic to devices so that your organization can focus on business insights instead of data management. Scale out your IoT solution by packaging your business logic into standard containers, then you can deploy those containers to any of your devices and monitor it all from the cloud.

Analytics drives business value in IoT solutions, but not all analytics needs to be in the cloud. If you want to respond to emergencies as quickly as possible, you can run anomaly detection workloads at the edge. If you want to reduce bandwidth costs and avoid transferring terabytes of raw data, you can clean and aggregate the data locally then only send the insights to the cloud for analysis.

Azure IoT Edge is made up of three components:

- **IoT Edge modules** are containers that run Azure services, third-party services, or your own code. Modules are deployed to IoT Edge devices and execute locally on those devices.
- The **IoT Edge runtime** runs on each IoT Edge device and manages the modules deployed to each device.
- A **cloud-based interface** enables you to remotely monitor and manage IoT Edge devices.

#### IoT Edge modules

IoT Edge modules are units of execution, implemented as Docker compatible containers, that run your business logic at the edge. Multiple modules can be configured to communicate with each other, creating a pipeline of data processing. You can develop custom modules or package certain Azure services into modules that provide insights offline and at the edge.

#### Artificial intelligence at the edge

Azure IoT Edge allows you to deploy complex event processing, machine learning, image recognition, and other high value AI without writing it in-house. Azure services like Azure Functions, Azure Stream

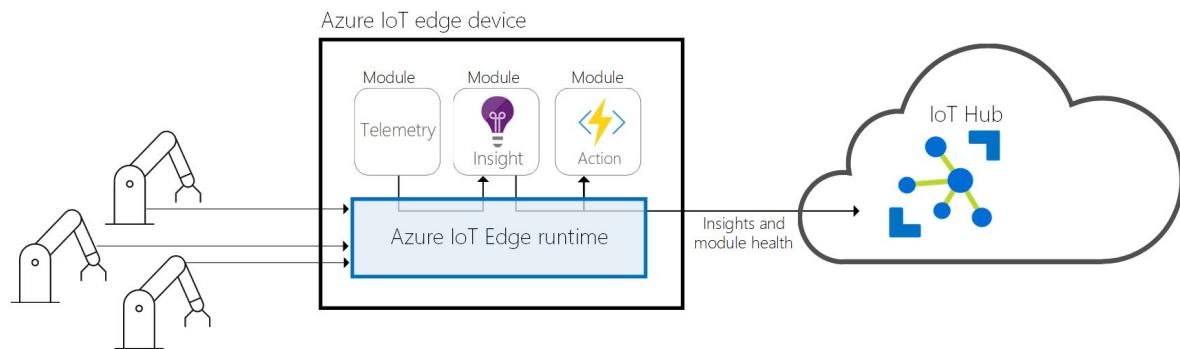
Analytics, and Azure Machine Learning can all be run on-premises via Azure IoT Edge. You're not limited to Azure services, though. Anyone is able to create AI modules and make them available to the community for use through the Azure Marketplace.

## Bring your own code

When you want to deploy your own code to your devices, Azure IoT Edge supports that, too. Azure IoT Edge holds to the same programming model as the other Azure IoT services. You can run the same code on a device or in the cloud. Azure IoT Edge supports both Linux and Windows so you can code to the platform of your choice. It supports Java, .NET Core 2.0, Node.js, C, and Python so your developers can code in a language they already know and use existing business logic.

### IoT Edge runtime

The Azure IoT Edge runtime enables custom and cloud logic on IoT Edge devices. The runtime sits on the IoT Edge device, and performs management and communication operations.



How you use an Azure IoT Edge device is up to you. The runtime is often used to deploy AI to gateway devices which aggregate and process data from other on-premises devices, but this deployment model is just one option.

The Azure IoT Edge runtime runs on a large set of IoT devices that enables using it in a wide variety of ways. It supports both Linux and Windows operating systems and abstracts hardware details. Use a device smaller than a Raspberry Pi 3 if you're not processing much data, or use an industrial server to run resource-intensive workloads.

## IoT Edge cloud interface

It's difficult to manage the software life cycle for millions of IoT devices that are often different makes and models or geographically scattered. Workloads are created and configured for a particular type of device, deployed to all of your devices, and monitored to catch any misbehaving devices. These activities can't be done on a per device basis and must be done at scale.

Azure IoT Edge integrates seamlessly with Azure IoT solution accelerators to provide one control plane for your solution's needs. Cloud services allow you to:

- Create and configure a workload to be run on a specific type of device.
- Send a workload to a set of devices.
- Monitor workloads running on devices in the field.

# IoT Edge Runtime

The IoT Edge runtime is responsible for the following functions on IoT Edge devices:

- Install and update workloads on the device.
- Maintain Azure IoT Edge security standards on the device.
- Ensure that IoT Edge modules are always running.
- Report module health to the cloud for remote monitoring.
- Manage communication between downstream devices and IoT Edge devices.
- Manage communication between modules on the IoT Edge device.
- Manage communication between the IoT Edge device and the cloud.

These responsibilities can be grouped into two categories, communication and module management, which are performed by two corresponding components of the IoT Edge runtime. The **IoT Edge hub** is responsible for communication, while the **IoT Edge agent** deploys and monitors the modules.

Both the IoT Edge hub and the IoT Edge agent are modules, just like any other module running on an IoT Edge device. They're sometimes referred to as the runtime modules.

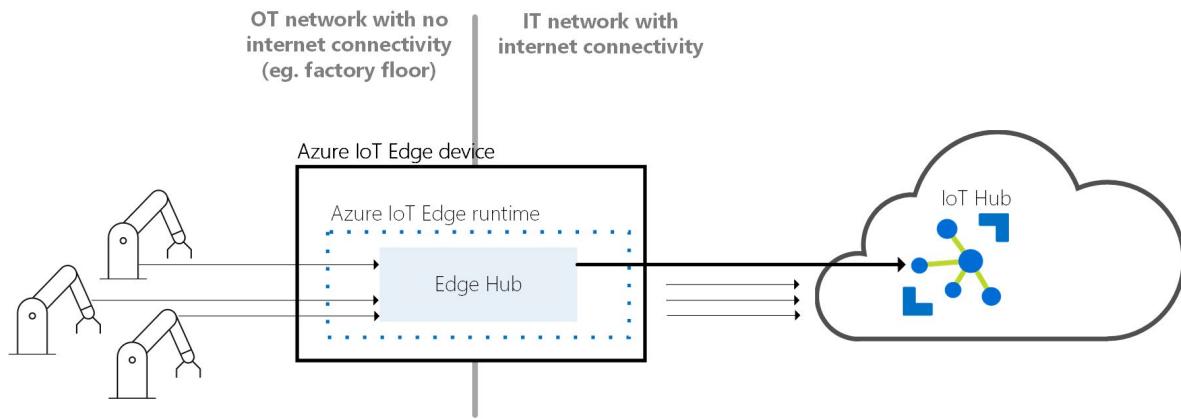
## IoT Edge hub

The IoT Edge hub acts as a local proxy for IoT Hub by exposing the same protocol endpoints as IoT Hub. This consistency means that clients (whether devices or modules) can connect to the IoT Edge runtime just as they would to IoT Hub.

**Note:** IoT Edge hub supports clients that connect using MQTT or AMQP. It does not support clients that use HTTP.

The IoT Edge hub is not a full version of IoT Hub running locally. There are some things that the IoT Edge hub silently delegates to IoT Hub. For example, IoT Edge hub forwards authentication requests to IoT Hub when a device first tries to connect. After the first connection is established, security information is cached locally by IoT Edge hub. Subsequent connections from that device are allowed without having to authenticate to the cloud.

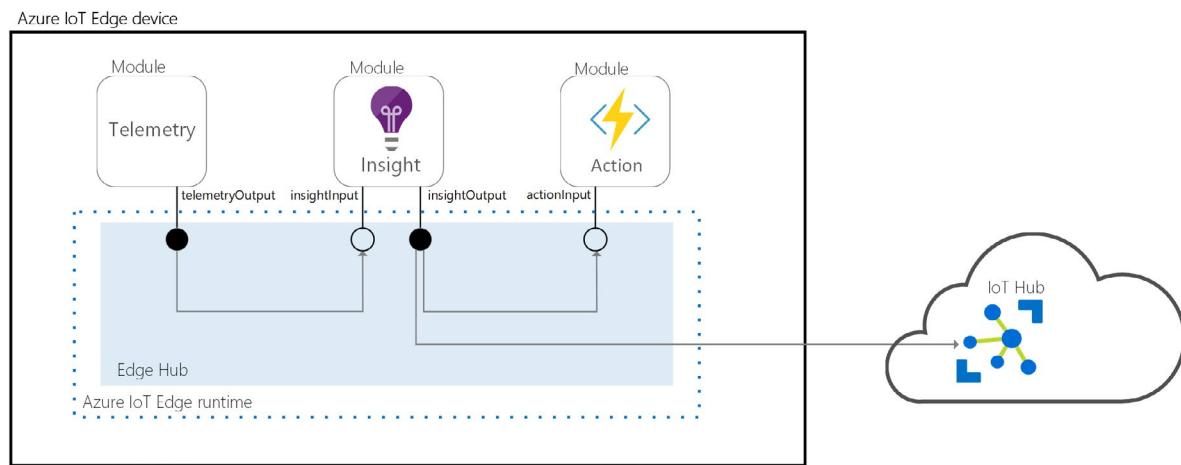
To reduce the bandwidth your IoT Edge solution uses, the IoT Edge hub optimizes how many actual connections are made to the cloud. IoT Edge hub takes logical connections from clients like modules or downstream devices and combines them for a single physical connection to the cloud. The details of this process are transparent to the rest of the solution. Clients think they have their own connection to the cloud even though they are all being sent over the same connection.



IoT Edge hub can determine whether it's connected to IoT Hub. If the connection is lost, IoT Edge hub saves messages or twin updates locally. Once a connection is reestablished, it syncs all the data. The location used for this temporary cache is determined by a property of the IoT Edge hub's module twin. The size of the cache is not capped and will grow as long as the device has storage capacity.

## Module communication

IoT Edge hub facilitates module to module communication. Using IoT Edge hub as a message broker keeps modules independent from each other. Modules only need to specify the inputs on which they accept messages and the outputs to which they write messages. A solution developer can stitch these inputs and outputs together so that the modules process data in the order specific to that solution.



To send data to the IoT Edge hub, a module calls the `SendEventAsync` method. The first argument specifies on which output to send the message. The following pseudocode sends a message on output1:

```
ModuleClient client = await ModuleClient.CreateFromEnvironmentAsync(transportSettings);  
await client.OpenAsync();  
await client.SendEventAsync("output1", message);
```

To receive a message, register a callback that processes messages coming in on a specific input. The following pseudocode registers the function `messageProcessor` to be used for processing all messages received on input1:

```
await client.SetInputMessageHandlerAsync("input1", messageProcessor, userContext);
```

For more information about the `ModuleClient` class and its communication methods, see the API reference for your preferred SDK language.

The solution developer is responsible for specifying the rules that determine how IoT Edge hub passes messages between modules. Routing rules are defined in the cloud and pushed down to IoT Edge hub in its module twin. The same syntax for IoT Hub routes is used to define routes between modules in Azure IoT Edge.

## IoT Edge agent

The IoT Edge agent is the other module that makes up the Azure IoT Edge runtime. It is responsible for instantiating modules, ensuring that they continue to run, and reporting the status of the modules back to IoT Hub. This configuration data is written as a property of the IoT Edge agent module twin.

The IoT Edge security daemon starts the IoT Edge agent on device startup. The agent retrieves its module twin from IoT Hub and inspects the deployment manifest. The deployment manifest is a JSON file that declares the modules that need to be started.

Each item in the deployment manifest contains specific information about a module and is used by the IoT Edge agent for controlling the module's lifecycle. Some of the more interesting properties are:

- **settings.image** – The container image that the IoT Edge agent uses to start the module. The IoT Edge agent must be configured with credentials for the container registry if the image is protected by a password. Credentials for the container registry can be configured remotely using the deployment manifest, or on the IoT Edge device itself by updating the `config.yaml` file in the IoT Edge program folder.
- **settings.createOptions** – A string that is passed directly to the Moby container daemon when starting a module's container. Adding options in this property allows for advanced configurations like port forwarding or mounting volumes into a module's container.
- **status** – The state in which the IoT Edge agent places the module. Usually, this value is set to running as most people want the IoT Edge agent to immediately start all modules on the device. However, you could specify the initial state of a module to be stopped and wait for a future time to tell the IoT Edge agent to start a module. The IoT Edge agent reports the status of each module back to the cloud in the reported properties. A difference between the desired property and the reported property is an indicator of a misbehaving device. The supported statuses are:
  - Downloading
  - Running
  - Unhealthy
  - Failed
  - Stopped
- **restartPolicy** – How the IoT Edge agent restarts a module. Possible values include:
  - never – The IoT Edge agent never restarts the module.
  - on-failure - If the module crashes, the IoT Edge agent restarts it. If the module shuts down cleanly, the IoT Edge agent does not restart it.
  - on-unhealthy - If the module crashes or is considered unhealthy, the IoT Edge agent restarts it.

- always - If the module crashes, is considered unhealthy, or shuts down in any way, the IoT Edge agent restarts it.
- **imagePullPolicy** - Whether the IoT Edge agent attempts to pull the latest image for a module automatically or not. If you don't specify a value, the default is onCreate. Possible values include:
  - on-create - When starting a module or updating a module based on a new deployment manifest, the IoT Edge agent will attempt to pull the module image from the container registry.
  - never - The IoT Edge agent will never attempt to pull the module image from the container registry. The expectation is that the module image is cached on the device, and any module image updates are made manually or managed by a third-party solution.

The IoT Edge agent sends runtime response to IoT Hub. Here is a list of possible responses:

- 200 - OK
- 400 - The deployment configuration is malformed or invalid.
- 417 - The device doesn't have a deployment configuration set.
- 412 - The schema version in the deployment configuration is invalid.
- 406 - The IoT Edge device is offline or not sending status reports.
- 500 - An error occurred in the IoT Edge runtime.

## Introduction to Azure IoT Edge Modules

Azure IoT Edge lets you deploy and manage business logic on the edge in the form of modules. Azure IoT Edge modules are the smallest unit of computation managed by IoT Edge, and can contain Azure services (such as Azure Stream Analytics) or your own solution-specific code. To understand how modules are developed, deployed, and maintained, it helps to think of the four conceptual elements of a module:

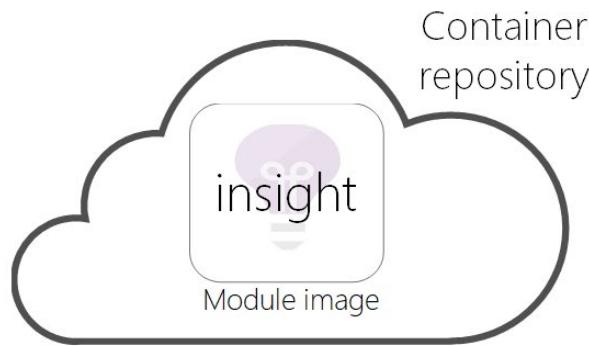
- A module image is a package containing the software that defines a module.
- A module instance is the specific unit of computation running the module image on an IoT Edge device. The module instance is started by the IoT Edge runtime.
- A module identity is a piece of information (including security credentials) stored in IoT Hub, that is associated to each module instance.
- A module twin is a JSON document stored in IoT Hub, that contains state information for a module instance, including metadata, configurations, and conditions.

## Module images and instances

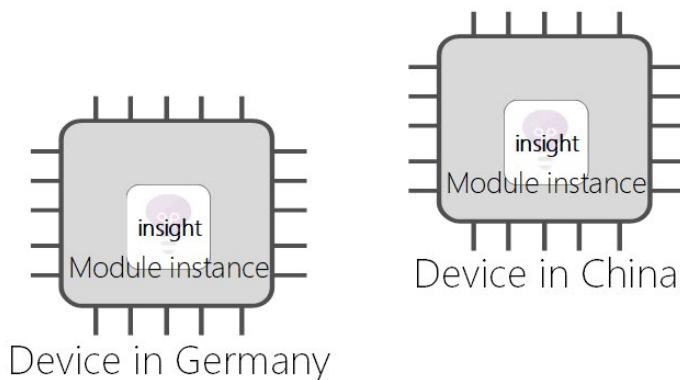
IoT Edge module images contain applications that take advantage of the management, security, and communication features of the IoT Edge runtime. You can develop your own module images, or export one from a supported Azure service, such as Azure Stream Analytics. The images exist in the cloud and they can be updated, changed, and deployed in different solutions. For instance, a module that uses machine learning to predict production line output exists as a separate image than a module that uses computer vision to control a drone.

Each time a module image is deployed to a device and started by the IoT Edge runtime, a new instance of that module is created. Two devices in different parts of the world could use the same module image. However, each device would have its own module instance when the module is started on the device.

Module image  
lives in the cloud



Module instances  
run on-premises



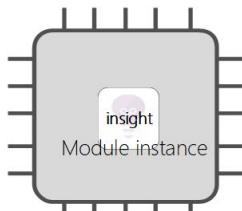
In implementation, modules images exist as container images in a repository, and module instances are containers on devices.

## Module identities

When a new module instance is created by the IoT Edge runtime, the instance is associated with a corresponding module identity. The module identity is stored in IoT Hub, and is employed as the addressing and security scope for all local and cloud communications for that specific module instance.

The identity associated with a module instance depends on the identity of the device on which the instance is running and the name you provide to that module in your solution. For instance, if you call `insight` a module that uses an Azure Stream Analytics, and you deploy it on a device called `Hannover01`, the IoT Edge runtime creates a corresponding module identity called `/devices/Hannover01/modules/insight`.

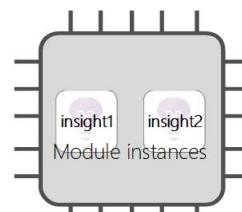
Clearly, in scenarios when you need to deploy one module image multiple times on the same device, you can deploy the same image multiple times with different names.



Device in Germany

Device /devices/Hannover01

Modules /devices/Hannover01/modules/insight



Device in China

Device /devices/Shenzhen01

Modules /devices/Shenzhen01/modules/insight1  
/devices/Shenzhen01/modules/insight2

## Module twins

Each module instance also has a corresponding module twin that you can use to configure the module instance. The instance and the twin are associated with each other through the module identity.

A module twin is a JSON document that stores module information and configuration properties. This concept parallels the device twin concept from IoT Hub. The structure of a module twin is the same as a device twin. The APIs used to interact with both types of twins are also the same. The only difference between the two is the identity used to instantiate the client SDK.

```
// Create a ModuleClient object. This ModuleClient will act on behalf of a
// module since it is created with a module's connection string instead
// of a device connection string.
ModuleClient client = new ModuleClient.CreateFromEnvironmentAsync(settings);
await client.OpenAsync();

// Get the module twin
Twin twin = await client.GetTwinAsync();
```

## Offline capabilities

Azure IoT Edge modules can operate offline indefinitely after syncing with IoT Hub at least once. IoT Edge devices can also extend this offline capability to other IoT devices.

## Module Twin Properties of Edge Runtime Modules

The module twins for the **IoT Edge hub** and **IoT Edge agent** modules (the two IoT Edge Runtime modules) provide properties that can be used manage module communication and lifecycle.

Each module twin includes:

- **Desired properties.** The solution backend can set desired properties, and the module can read them. The module can also receive notifications of changes in the desired properties. Desired properties are used along with reported properties to synchronize module configuration or conditions.
- **Reported properties.** The module can set reported properties, and the solution backend can read and query them. Reported properties are used along with desired properties to synchronize module configuration or conditions.

## EdgeAgent desired properties

The module twin for the IoT Edge agent is called `$edgeAgent` and coordinates the communications between the IoT Edge agent running on a device and IoT Hub. The desired properties are set when applying a deployment manifest on a specific device as part of a single-device or at-scale deployment.

Property	Description	Required	
<code>schemaVersion</code>	Has to be "1.0"	Yes	
<code>runtime.type</code>	Has to be "docker"	Yes	
<code>runtime.settings.minDockerVersion</code>	Set to the minimum Docker version required by this deployment manifest	Yes	
<code>runtime.settings.loggingOptions</code>	A stringified JSON containing the logging options for the IoT Edge agent container. Docker logging options	No	
<code>runtime.settings.registryCredentials.{registryId}.username</code>	The username of the container registry. For Azure Container Registry, the username is usually the registry name.  Registry credentials are necessary for any module images that are not public.	No	
<code>runtime.settings.registryCredentials.{registryId}.password</code>	The password for the container registry.	No	
<code>runtime.settings.registryCredentials.{registryId}.address</code>	The address of the container registry. For Azure Container Registry, the address is usually {registry name}.azurecr.io.	No	
<code>systemModules.edgeAgent.type</code>	Has to be "docker"	Yes	

Property	Description	Required	
systemModules.edgeAgent.settings.image	The URI of the image of the IoT Edge agent. Currently, the IoT Edge agent is not able to update itself.	Yes	
systemModules.edgeAgent.settings.createOptions	A stringified JSON containing the options for the creation of the IoT Edge agent container. Docker create options	No	
systemModules.edgeAgent.configuration.id	The ID of the deployment that deployed this module.	IoT Hub sets this property when the manifest is applied using a deployment. Not part of a deployment manifest.	
systemModules.edgeHub.type	Has to be "docker"	Yes	
systemModules.edgeHub.status	Has to be "running"	Yes	
systemModules.edgeHub.restartPolicy	Has to be "always"	Yes	
systemModules.edgeHub.settings.image	The URI of the image of the IoT Edge hub.	Yes	
systemModules.edgeHub.settings.createOptions	A stringified JSON containing the options for the creation of the IoT Edge hub container. Docker create options	No	
systemModules.edgeHub.configuration.id	The ID of the deployment that deployed this module.	IoT Hub sets this property when the manifest is applied using a deployment. Not part of a deployment manifest.	
modules.{moduleId}.version	A user-defined string representing the version of this module.	Yes	
modules.{moduleId}.type	Has to be "docker"	Yes	
modules.{moduleId}.status	{"running"   "stopped"}	Yes	
modules.{moduleId}.restartPolicy	{"never"   "on-failure"   "on-unhealthy"   "always"}	Yes	

Property	Description	Required	
modules.{moduleId}.imagePullPolicy	{"on-create"}	"never"	No
modules.{moduleId}.settings.image	The URI to the module image.	Yes	
modules.{moduleId}.settings.createOptions	A stringified JSON containing the options for the creation of the module container. Docker create options	No	
modules.{moduleId}.configuration.id	The ID of the deployment that deployed this module.	IoT Hub sets this property when the manifest is applied using a deployment. Not part of a deployment manifest.	

## EdgeAgent reported properties

The IoT Edge agent reported properties include three main pieces of information:

1. The status of the application of the last-seen desired properties;
2. The status of the modules currently running on the device, as reported by the IoT Edge agent; and
3. A copy of the desired properties currently running on the device.

This last piece of information, a copy of the current desired properties, is useful to tell whether the device has applied the latest desired properties or is still running a previous deployment manifest.

**Note:** The reported properties of the IoT Edge agent are useful as they can be queried with the IoT Hub query language to investigate the status of deployments at scale.

The following table does not include the information that is copied from the desired properties.

Property	Description
lastDesiredVersion	This integer refers to the last version of the desired properties processed by the IoT Edge agent.
lastDesiredStatus.code	This status code refers to the last desired properties seen by the IoT Edge agent. Allowed values: 200 - Success, 400 - Invalid configuration, 412 - Invalid schema version, 417 - the desired properties are empty, 500 - Failed
lastDesiredStatus.description	Text description of the status
deviceHealth	healthy if the runtime status of all modules is either running or stopped, unhealthy otherwise
configurationHealth.{deploymentId}.health	healthy if the runtime status of all modules set by the deployment {deploymentId} is either running or stopped, unhealthy otherwise
runtime.platform.OS	Reporting the OS running on the device
runtime.platform.architecture	Reporting the architecture of the CPU on the device

Property	Description
systemModules.edgeAgent.runtimeStatus	The reported status of IoT Edge agent: {"running"   "unhealthy"}
systemModules.edgeAgent.statusDescription	Text description of the reported status of the IoT Edge agent.
systemModules.edgeHub.runtimeStatus	Status of IoT Edge hub: { "running"   "stopped"   "failed"   "backoff"   "unhealthy" }
systemModules.edgeHub.statusDescription	Text description of the status of IoT Edge hub if unhealthy.
systemModules.edgeHub.exitCode	The exit code reported by the IoT Edge hub container if the container exits
systemModules.edgeHub.startTimeUtc	Time when IoT Edge hub was last started
systemModules.edgeHub.lastExitTimeUtc	Time when IoT Edge hub last exited
systemModules.edgeHub.lastRestartTimeUtc	Time when IoT Edge hub was last restarted
systemModules.edgeHub.restartCount	Number of times this module was restarted as part of the restart policy.
modules.{moduleId}.runtimeStatus	Status of the module: { "running"   "stopped"   "failed"   "backoff"   "unhealthy" }
modules.{moduleId}.statusDescription	Text description of the status of the module if unhealthy.
modules.{moduleId}.exitCode	The exit code reported by the module container if the container exits
modules.{moduleId}.startTimeUtc	Time when the module was last started
modules.{moduleId}.lastExitTimeUtc	Time when the module last exited
modules.{moduleId}.lastRestartTimeUtc	Time when the module was last restarted
modules.{moduleId}.restartCount	Number of times this module was restarted as part of the restart policy.

## EdgeHub desired properties

The module twin for the IoT Edge hub is called `$edgeHub` and coordinates the communications between the IoT Edge hub running on a device and IoT Hub. The desired properties are set when applying a deployment manifest on a specific device as part of a single-device or at-scale deployment.

Property	Description	Required in the deployment manifest
schemaVersion	Has to be "1.0"	Yes
routes.{routeName}	A string representing an IoT Edge hub route.	The routes element can be present but empty.
storeAndForwardConfiguration.timeToLiveSecs	The time in seconds that IoT Edge hub keeps messages if disconnected from routing endpoints, whether IoT Hub or a local module. The value can be any positive integer.	Yes

## EdgeHub reported properties

Property	Description
lastDesiredVersion	This integer refers to the last version of the desired properties processed by the IoT Edge hub.
lastDesiredStatus.code	The status code referring to last desired properties seen by the IoT Edge hub. Allowed values: 200 Success, 400 Invalid configuration, 500 Failed
lastDesiredStatus.description	Text description of the status.
clients.{device or moduleId}.status	The connectivity status of this device or module. Possible values {"connected"   "disconnected"}. Only module identities can be in disconnected state. Downstream devices connecting to IoT Edge hub appear only when connected.
clients.{device or moduleId}.lastConnectTime	Last time the device or module connected.
clients.{device or moduleId}.lastDisconnectTime	Last time the device or module disconnected.

## How Azure IoT Edge uses Certificates

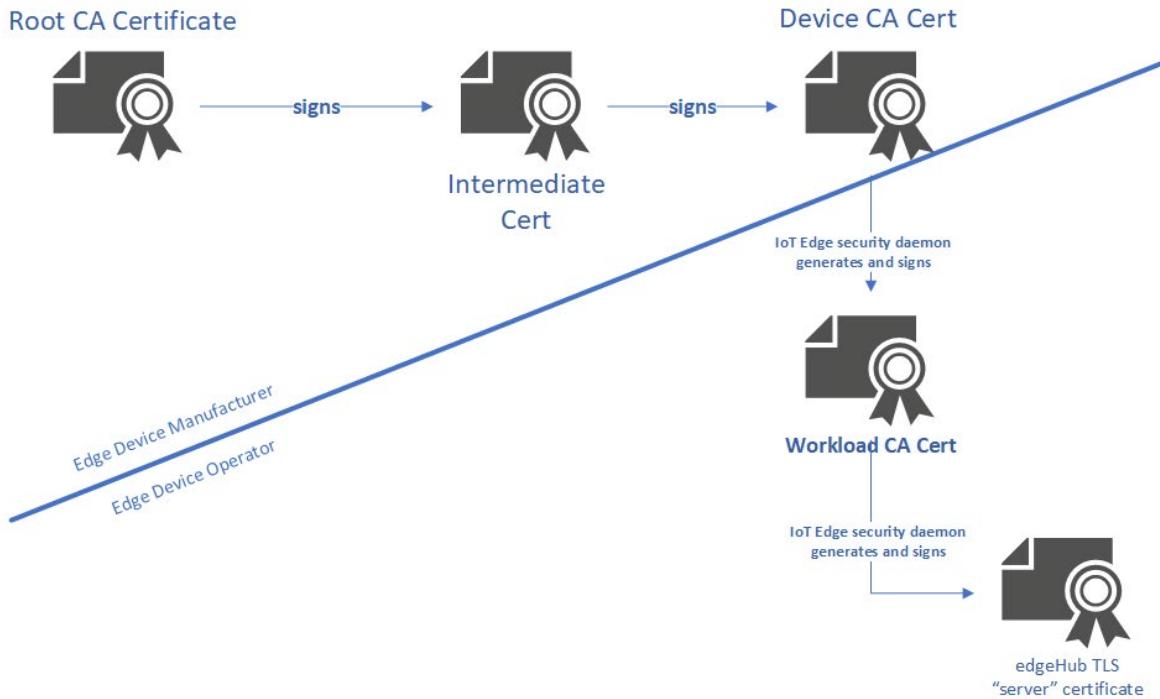
IoT Edge certificates are used for the modules and downstream IoT devices to verify the identity and legitimacy of the IoT Edge hub runtime module that they connect to. These verifications enable a TLS (transport layer security) secure connection between the runtime, the modules, and the IoT devices. Like IoT Hub itself, IoT Edge requires a secure and encrypted connection from IoT downstream (or leaf) devices and IoT Edge modules. To establish a secure TLS connection, the IoT Edge hub module presents a server certificate chain to connecting clients in order for them to verify its identity.

### IoT Edge certificates

Usually, manufacturers are not the end users of an IoT Edge device. Sometimes the only relationship between the two is when the end user, or operator, purchases a generic device made by the manufacturer. Other times, the manufacturer works under contract to build a custom device on behalf of the operator. The IoT Edge certificate design attempts to take both scenarios into account.

The following figure illustrates IoT Edge's usage of certificates. There may be zero, one, or many intermediate signing certificates between the root CA certificate and the device CA certificate, depending on the number of entities involved. Here we show one case.

### Azure IoT Edge Certificates (typical)



## Certificate authority

The certificate authority, or 'CA' for short, is an entity that issues digital certificates. A certificate authority acts as a trusted third party between the owner, and the receiver of the certificate. A digital certificate certifies the ownership of a public key by the receiver of the certificate. The certificate chain of trust works by initially issuing a root certificate, which is the basis for trust in all certificates issued by the authority. Afterwards, the owner can use the root certificate to issue additional intermediate certificates ('leaf' certificates).

## Root CA certificate

A root CA certificate is the root of trust of the entire process. In production scenarios, this CA certificate is usually purchased from a trusted commercial certificate authority like Baltimore, Verisign, or DigiCert. Should you have complete control over the devices connecting to your IoT Edge devices, it's possible to use a corporate level certificate authority. In either event, the entire certificate chain from the IoT Edge hub up rolls up to it, so the leaf IoT devices must trust the root certificate. You can store the root CA certificate either in the trusted root certificate authority store, or provide the certificate details in your application code.

## Intermediate certificates

In a typical manufacturing process for creating secure devices, root CA certificates are rarely used directly, primarily because of the risk of leakage or exposure. The root CA certificate creates and digitally signs

one or more intermediate CA certificates. There may only be one, or there may be a chain of these intermediate certificates. Scenarios that would require a chain of intermediate certificates include:

- A hierarchy of departments within a manufacturer.
- Multiple companies involved serially in the production of a device.
- A customer buying a root CA and deriving a signing certificate for the manufacturer to sign the devices they make on that customer's behalf.

In any case, the manufacturer uses an intermediate CA certificate at the end of this chain to sign the device CA certificate placed on the end device. Generally, these intermediate certificates are closely guarded at the manufacturing plant. They undergo strict processes, both physical and electronic for their usage.

## Device CA certificate

The device CA certificate is generated from and signed by the final intermediate CA certificate in the process. This certificate is installed on the IoT Edge device itself, preferably in secure storage such as a hardware security module (HSM). In addition, a device CA certificate uniquely identifies an IoT Edge device. For IoT Edge, the device CA certificate can issue other certificates. For example, the device CA certificate issues leaf device certificates that are used to authenticate devices to the Azure IoT Device Provisioning Service.

## IoT Edge Workload CA

The IoT Edge Security Manager generates the workload CA certificate, the first on the "operator" side of the process, when IoT Edge first starts. This certificate is generated from and signed by the "device CA certificate". This certificate, which is just another intermediate signing certificate, is used to generate and sign any other certificates used by the IoT Edge runtime. Today, that is primarily the IoT Edge hub server certificate discussed in the following section, but in the future may include other certificates for authenticating IoT Edge components.

## IoT Edge hub server certificate

The IoT Edge hub server certificate is the actual certificate presented to leaf devices and modules for identity verification during establishment of the TLS connection required by IoT Edge. This certificate presents the full chain of signing certificates used to generate it up to the root CA certificate, which the leaf IoT device must trust. When generated by the IoT Edge Security Manager, the common name (CN), of this IoT Edge hub certificate is set to the 'hostname' property in the config.yaml file after conversion to lower case. This is a common source of confusion with IoT Edge.

## Production implications

A reasonable question might be "why does IoT Edge need the 'workload CA' extra certificate? Couldn't it use the device CA certificate to directly generate the IoT Edge hub server certificate?". Technically, it could. However, the purpose of this "workload" intermediate certificate is to separate concerns between the device manufacturer and the device operator. Imagine a scenario where an IoT Edge device is sold or transferred from one customer to another. You would likely want the device CA certificate provided by the manufacturer to be immutable. However, the "workload" certificates specific to operation of the device should be wiped and recreated for the new deployment.

Because manufacturing and operation processes are separated, consider the following implications when preparing production devices:

- With any certificate-based process, the root CA certificate and all intermediate CA certificates should be secured and monitored during the entire process of rolling out an IoT Edge device. The IoT Edge device manufacturer should have strong processes in place for proper storage and usage of their intermediate certificates. In addition, the device CA certificate should be kept in as secure storage as possible on the device itself, preferably a hardware security module.
- The IoT Edge hub server certificate is presented by IoT Edge hub to the connecting client devices and modules. The common name (CN) of the device CA certificate must not be the same as the "hostname" that will be used in config.yaml on the IoT Edge device. The name used by clients to connect to IoT Edge (for example, via the GatewayHostName parameter of the connection string or the CONNECT command in MQTT) can't be the same as common name used in the device CA certificate. This restriction is because the IoT Edge hub presents its entire certificate chain for verification by clients. If the IoT Edge hub server certificate and the device CA certificate both have the same CN, you get in a verification loop and the certificate invalidates.
- Because the device CA certificate is used by the IoT Edge security daemon to generate the final IoT Edge certificates, it must itself be a signing certificate, meaning it has certificate signing capabilities. Applying "V3 Basic constraints CA:True" to the device CA certificate automatically sets up the required key usage properties.

## Dev/Test implications

To ease development and test scenarios, Microsoft provides a set of convenience scripts for generating non-production certificates suitable for IoT Edge in the transparent gateway scenario.

These scripts generate certificates that follow the certificate chain structure explained in this article. The following commands generate the "root CA certificate" and a single "intermediate CA certificate".

Bash

```
./certGen.sh create_root_and_intermediate
```

PowerShell

```
New-CACertsCertChain rsa
```

Likewise, these commands generate the "Device CA Certificate".

Bash

```
./certGen.sh create_edge_device_ca_certificate "<gateway device name>"
```

PowerShell

```
New-CACertsEdgeDeviceCA "<gateway device name>"
```

**Note:** The <gateway device name> passed into those scripts should not be the same as the "hostname" parameter in config.yaml. The scripts help you avoid any issues by appending a ".ca" string to the <gateway device name> to prevent the name collision in case a user sets up IoT Edge using the same name in both places. However, it's good practice to avoid using the same name.

**Tip:** To connect your device IoT “leaf” devices and applications that use our IoT device SDK through IoT Edge, you must add the optional `GatewayHostName` parameter on to the end of the device’s connection string. When the Edge Hub Server Certificate is generated, it is based on a lower-cased version of the hostname from `config.yaml`, therefore, for the names to match and the TLS certificate verification to succeed, you should enter the `GatewayHostName` parameter in lower case.

## Example of IoT Edge certificate hierarchy

To illustrate an example of this certificate path, the following screenshot is from a working IoT Edge device set up as a transparent gateway. OpenSSL is used to connect to the IoT Edge hub, validate, and dump out the certificates.

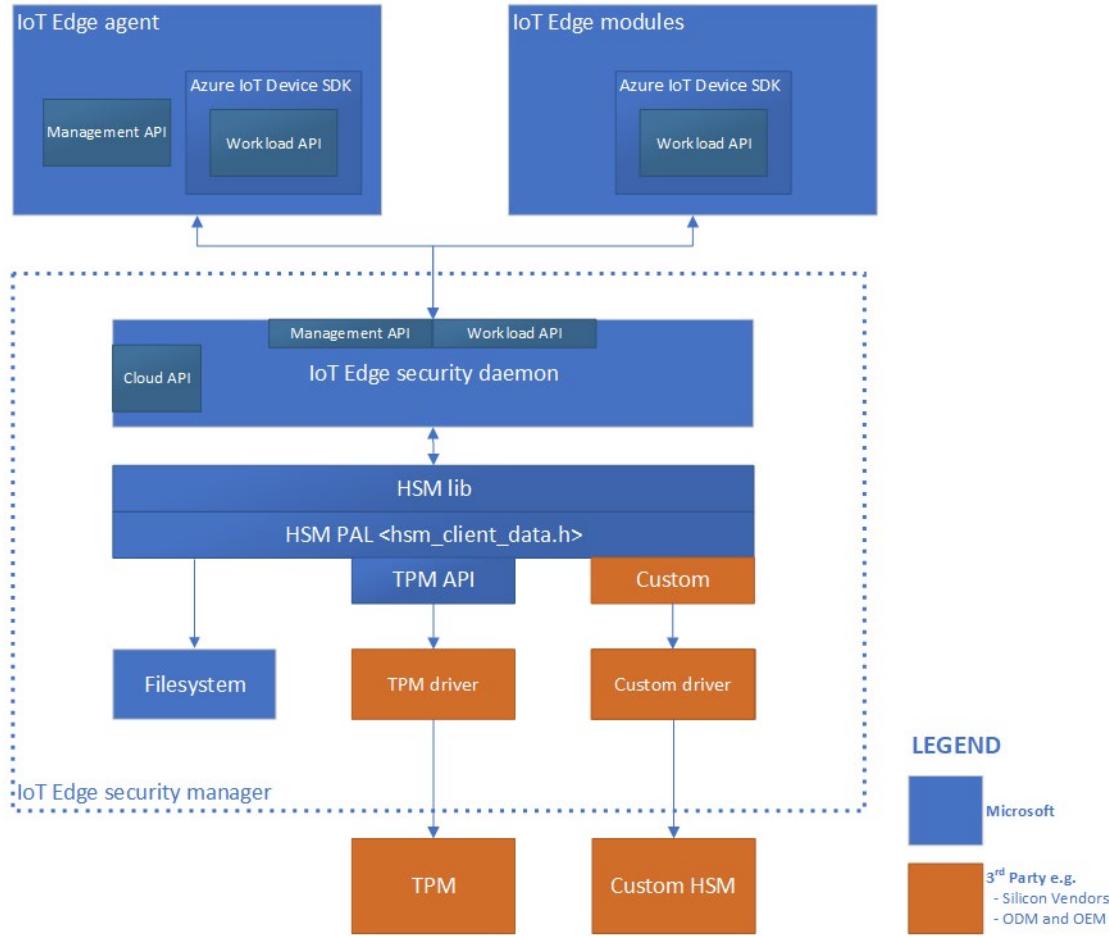
```
stevebus@edgeUbuntu:~$ openssl s_client -connect iotedgegw.local:8883
CONNECTED(00000003)
depth=4 CN = Azure IoT Hub CA Cert Test Only
verify return:1
depth=3 CN = Azure IoT Hub Intermediate Cert Test Only
verify return:1
depth=2 CN = iotgateway.ca
verify return:1
depth=1 CN = iotedge workload ca
verify return:1
depth=0 CN = iotedgegw.local
verify return:1
---
```

You can see the hierarchy of certificate depth represented in the screenshot:

Root CA Certificate	Azure IoT Hub CA Cert Test Only
Intermediate CA Certificate	Azure IoT Hub Intermediate Cert Test Only
Device CA Certificate	iotgateway.ca (“iotgateway” was passed in as the < gateway host name > to the convenience scripts)
Workload CA Certificate	iotedge workload ca
IoT Edge Hub Server Certificate	iotedgegw.local (matches the ‘hostname’ from config.yaml)

## Azure IoT Edge Security Manager

The Azure IoT Edge security manager is a well-bounded security core for protecting the IoT Edge device and all its components by abstracting the secure silicon hardware. It is the focal point for security hardening and provides technology integration point to original equipment manufacturers (OEM).



IoT Edge security manager aims to defend the integrity of the IoT Edge device and all inherent software operations. The security manager transitions trust from underlying hardware root of trust hardware (if available) to bootstrap the IoT Edge runtime and monitor ongoing operations. The IoT Edge security manager is software working along with secure silicon hardware (where available) to help deliver the highest security assurances possible.

The responsibilities of the IoT Edge security manager include, but aren't limited to:

- Secured and measured bootstrapping of the Azure IoT Edge device.
- Device identity provisioning and transition of trust where applicable.
- Host and protect device components of cloud services like Device Provisioning Service.
- Securely provision IoT Edge modules with unique identities.
- Gatekeeper to device hardware root of trust through notary services.
- Monitor the integrity of IoT Edge operations at runtime.

IoT Edge security manager includes three components:

- IoT Edge security daemon.
- Hardware security module platform abstraction Layer (HSM PAL).
- Optional but highly recommended hardware silicon root of trust or HSM.

## The IoT Edge security daemon

The IoT Edge security daemon is responsible for the logical operations of IoT Edge security manager. It represents a significant portion of the trusted computing base of the IoT Edge device.

## Design principles

The IoT Edge security daemon follows two core principles: maximize operational integrity, and minimize bloat and churn.

### Maximize operational integrity

The IoT Edge security daemon operates with the highest integrity possible within the defense capability of any given root of trust hardware. With proper integration, the root of trust hardware measures and monitors the security daemon statically and at runtime to resist tampering.

Physical access is always a threat to IoT devices. Hardware root of trust plays an important role in defending the integrity of the IoT Edge security daemon. Hardware root of trust come in two varieties:

- secure elements for the protection of sensitive information like secrets and cryptographic keys.
- secure enclaves for the protection of secrets like keys, and sensitive workloads like metering and billing.

Two kinds of execution environments exist to use hardware root of trust:

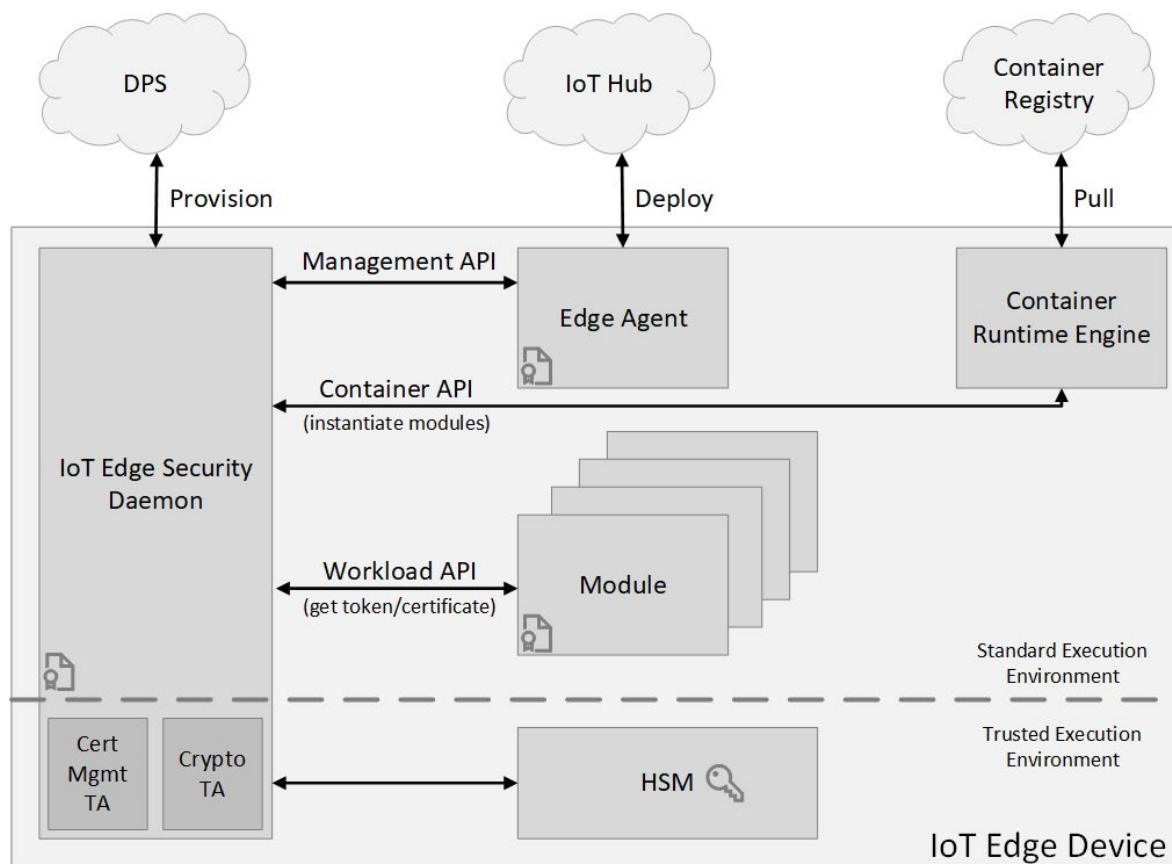
- The standard or rich execution environment (REE) that relies on the use of secure elements to protect sensitive information.
- The trusted execution environment (TEE) that relies on the use of secure enclave technology to protect sensitive information and offer protection to software execution.

For devices using secure enclaves as hardware root of trust, sensitive logic within IoT Edge security daemon should be inside the enclave. Non-sensitive portions of the security daemon can be outside of the TEE. In any case, original design manufacturers (ODM) and original equipment manufacturers (OEM) should extend trust from their HSM to measure and defend the integrity of the IoT Edge security daemon at boot and runtime.

### Minimize bloat and churn

Another core principle for the IoT Edge security daemon is to minimize churn. For the highest level of trust, the IoT Edge security daemon can tightly couple with the device hardware root of trust and operate as native code. It's common for these types of realizations to update the daemon software through the hardware root of trust's secure update paths (as opposed to OS provided update mechanisms), which can be challenging in some scenarios. While security renewal is recommended for IoT devices, excessive update requirements or large update payloads can expand the threat surface in many ways. Examples include skipping of updates to maximize operational availability or root of trust hardware too constrained to process large update payloads. As such, the design of IoT Edge security daemon is concise to keep the footprint and trusted computing base small and to minimize update requirements.

## Architecture of IoT Edge security daemon



The IoT Edge security daemon takes advantage of any available hardware root of trust technology for security hardening. It also allows for split-world operation between a standard/rich execution environment (REE) and a trusted execution environment (TEE) when hardware technologies offer trusted execution environments. Role-specific interfaces enable the major components of IoT Edge to assure the integrity of the IoT Edge device and its operations.

## Cloud interface

The cloud interface allows the IoT Edge security daemon to access cloud services such as cloud complements to device security like security renewal. For example, the IoT Edge security daemon currently uses this interface to access the Azure IoT Hub Device Provisioning Service for device identity lifecycle management.

## Management API

IoT Edge security daemon offers a management API, which is called by the IoT Edge agent when creating/starting/stopping/removing an IoT Edge module. The security daemon stores “registrations” for all active modules. These registrations map a module’s identity to some properties of the module. A few examples for these properties are the process identifier (pid) of the process running in the container or the hash of the docker container’s contents.

These properties are used by the workload API (described below) to verify that the caller is authorized to perform an action.

The management API is a privileged API, callable only from the IoT Edge agent. Since the IoT Edge security daemon bootstraps and starts the IoT Edge agent, it can create an implicit registration for the IoT Edge agent, after it has attested that the IoT Edge agent has not been tampered with. The same attestation process that the workload API uses also restricts access to the management API to only the IoT Edge agent.

## Container API

The container API interacts with the container system in use for module management, like Moby or Docker.

## Workload API

The workload API is accessible to all modules. It provides proof of identity, either as an HSM rooted signed token or an X509 certificate, and the corresponding trust bundle to a module. The trust bundle contains CA certificates for all the other servers that the modules should trust.

The IoT Edge security daemon uses an attestation process to guard this API. When a module calls this API, the security daemon attempts to find a registration for the identity. If successful, it uses the properties of the registration to measure the module. If the result of the measurement process matches the registration, a new proof of identity is generated. The corresponding CA certificates (trust bundle) are returned to the module. The module uses this certificate to connect to IoT Hub, other modules, or start a server. When the signed token or certificate nears expiration, it's the responsibility of the module to request a new certificate.

## Integration and maintenance

Microsoft maintains the main code base for the IoT Edge security daemon on GitHub.

## Installation and updates

Installation and updates of the IoT Edge security daemon are managed through the operating system's package management system. IoT Edge devices with hardware root of trust should provide additional hardening to the integrity of the daemon by managing its lifecycle through the secure boot and updates management systems. Device makers should explore these avenues based on their respective device capabilities.

## Versioning

The IoT Edge runtime tracks and reports the version of the IoT Edge security daemon. The version is reported as the `runtime.platform.version` attribute of the IoT Edge agent module reported property.

## Hardware security module platform abstraction layer (HSM PAL)

The HSM PAL abstracts all root of trust hardware to isolate the developer or user of IoT Edge from their complexities. It includes a combination of application programming interface (API) and trans-domain communication procedures, for example communication between a standard execution environment and a secure enclave. The actual implementation of the HSM PAL depends on the specific secure hardware in use. Its existence enables the use of virtually any secure silicon hardware.

## Secure silicon root of trust hardware

Secure silicon is necessary to anchor trust inside the IoT Edge device hardware. Secure silicon come in variety to include Trusted Platform Module (TPM), embedded Secure Element (eSE), ARM TrustZone, Intel SGX, and custom secure silicon technologies. The use of secure silicon root of trust in devices is recommended given the threats associated with physical accessibility of IoT devices.

## IoT Edge security manager integration and maintenance

The IoT Edge security manager aims to identify and isolate the components that defend the security and integrity of the Azure IoT Edge platform for custom hardening. Third parties, like device makers, should make use of custom security features available with their device hardware. See next steps section for links that demonstrate how to harden the Azure IoT security manager with the Trusted Platform Module (TPM) on Linux and Windows platforms. These examples use software or virtual TPMs but directly apply to using discrete TPM devices.

# Edge Deployment Process

## Introduction to IoT Edge Deployment

Azure IoT Edge devices follow a device lifecycle that is similar to other types of IoT devices:

1. Provision new IoT Edge devices by imaging a device with an OS and installing the IoT Edge runtime.
2. Configure the devices to run IoT Edge modules, and then monitor their health.
3. Finally, retire devices when they are replaced or become obsolete.

The actions accomplished during the Edge deployment process are aligned primarily with the middle stage, where the modules that a device will be running are defined and where monitoring is implemented.

Azure IoT Edge provides two ways to configure the modules to run on IoT Edge devices:

- one for development and fast iterations on a single device
- one for managing large fleets of IoT Edge devices

Both approaches can be implemented either programmatically or by using the Azure portal.

We use the term **IoT Edge automatic deployments** to refer to configuration and monitoring stages for fleets of devices.

When targeting groups or a large number of devices, you can use tags in the device twins to specify which devices you'd like your modules deployed to.

At a high level, IoT Edge automatic deployments include the following:

1. An operator defines a deployment that describes a set of modules as well as the target devices. Each deployment has a deployment manifest that reflects this information.
2. The IoT Hub service communicates with all targeted devices to configure them with the desired modules.
3. The IoT Hub service retrieves status from the IoT Edge devices and makes them available to the operator. For example, an operator can see when an Edge device is not configured successfully or if a module fails during runtime.
4. At any time, new IoT Edge devices that meet the targeting conditions are configured for the deployment. For example, a deployment that targets all IoT Edge devices in Washington State automatically configures a new IoT Edge device once it is provisioned and added to the Washington State device group.

## Deployment

An IoT Edge automatic deployment assigns IoT Edge module images to run as instances on a targeted set of IoT Edge devices. It works by configuring an IoT Edge deployment manifest to include a list of modules with the corresponding initialization parameters. A deployment can be assigned to a single device (based on Device ID) or to a group of devices (based on tags). Once an IoT Edge device receives a deployment manifest, it downloads and installs the container images from the respective container repositories, and configures them accordingly. Once a deployment is created, an operator can monitor the deployment status to see whether targeted devices are correctly configured.

Only IoT Edge devices can be configured with a deployment. The following prerequisites must be on the device before it can receive the deployment:

- The base operating system
- A container management system, like Moby or Docker
- Provisioning of the IoT Edge runtime

## Deployment manifest

A deployment manifest is a JSON document that describes the modules to be configured on the targeted IoT Edge devices. It contains the configuration metadata for all the modules, including the required system modules (specifically the IoT Edge agent and IoT Edge hub).

The configuration metadata for each module includes:

- Version
- Type
- Status (for example, running or stopped)
- Restart policy
- Image and container registry
- Routes for data input and output

If the module image is stored in a private container registry, the IoT Edge agent holds the registry credentials.

## Target condition

The target condition is continuously evaluated throughout the lifetime of the deployment. Any new devices that meet the requirements are included, and any existing devices that no longer do are removed. The deployment is reactivated if the service detects any target condition change.

For instance, you have a deployment A with a target condition tags.environment = 'prod'. When you kick off the deployment, there are 10 production devices. The modules are successfully installed in these 10 devices. The IoT Edge Agent Status is shown as 10 total devices, 10 successful responses, 0 failure responses, and 0 pending responses. Now you add five more devices with tags.environment = 'prod'. The service detects the change and the IoT Edge Agent Status becomes 15 total devices, 10 successful responses, 0 failure responses, and 5 pending responses when it tries to deploy to the five new devices.

Use any Boolean condition on device twins tags or deviceld to select the target devices. If you want to use condition with tags, you need to add "tags":{} section in the device twin under the same level as properties.

Target condition examples:

- deviceld ='linuxprod1'
- tags.environment ='prod'
- tags.environment = 'prod' AND tags.location = 'westus'
- tags.environment = 'prod' OR tags.location = 'westus'
- tags.operator = 'John' AND tags.environment = 'prod' NOT deviceld = 'linuxprod1'

Here are some constraints when you construct a target condition:

- In device twin, you can only build a target condition using tags or deviceld.
- Double quotes aren't allowed in any portion of the target condition. Use single quotes.
- Single quotes represent the values of the target condition. Therefore, you must escape the single quote with another single quote if it's part of the device name. For example, to target a device called operator'sDevice, write deviceld='operator''sDevice'.
- Numbers, letters, and the following characters are allowed in target condition values:  
-:.\_%#\*?!(),=@;\$.

## Priority

A priority defines whether a deployment should be applied to a targeted device relative to other deployments. A deployment priority is a positive integer, with larger numbers denoting higher priority. If an IoT Edge device is targeted by more than one deployment, the deployment with the highest priority applies. Deployments with lower priorities are not applied, nor are they merged. If a device is targeted with two or more deployments with equal priority, the most recently created deployment (determined by the creation timestamp) applies.

## Labels

Labels are string key/value pairs that you can use to filter and group of deployments. A deployment may have multiple labels. Labels are optional and do no impact the actual configuration of IoT Edge devices.

## Deployment status

A deployment can be monitored to determine whether it applied successfully for any targeted IoT Edge device. A targeted Edge device will appear in one or more of the following status categories:

- **Target** shows the IoT Edge devices that match the Deployment targeting condition.
- **Actual** shows the targeted IoT Edge devices that are not targeted by another deployment of higher priority.
- **Healthy** shows the IoT Edge devices that have reported back to the service that the modules have been deployed successfully.
- **Unhealthy** shows the IoT Edge devices have reported back to the service that one or modules have not been deployed successfully. To further investigate the error, connect remotely to those devices and view the log files.
- **Unknown** shows the IoT Edge devices that did not report any status pertaining this deployment. To further investigate, view service info and log files.

## Phased rollout

A phased rollout is an overall process whereby an operator deploys changes to a broadening set of IoT Edge devices. The goal is to make changes gradually to reduce the risk of making wide scale breaking changes.

A phased rollout is executed in the following phases and steps:

1. Establish a test environment of IoT Edge devices by provisioning them and setting a device twin tag like `tag.environment='test'`. The test environment should mirror the production environment that the deployment will eventually target.
2. Create a deployment including the desired modules and configurations. The targeting condition should target the test IoT Edge device environment.
3. Validate the new module configuration in the test environment.
4. Update the deployment to include a subset of production IoT Edge devices by adding a new tag to the targeting condition. Also, ensure that the priority for the deployment is higher than other deployments currently targeted to those devices
5. Verify that the deployment succeeded on the targeted IoT Devices by viewing the deployment status.
6. Update the deployment to target all remaining production IoT Edge devices.

## Rollback

Deployments can be rolled back if you receive errors or misconfigurations. Because a deployment defines the absolute module configuration for an IoT Edge device, an additional deployment must also be targeted to the same device at a lower priority even if the goal is to remove all modules.

Perform rollbacks in the following sequence:

1. Confirm that a second deployment is also targeted at the same device set. If the goal of the rollback is to remove all modules, the second deployment should not include any modules.
2. Modify or remove the target condition expression of the deployment you wish to roll back so that the devices no longer meet the targeting condition.
3. Verify that the rollback succeeded by viewing the deployment status.
  - The rolled-back deployment should no longer show status for the devices that were rolled back.
  - The second deployment should now include deployment status for the devices that were rolled back.

## Deployment Manifest

The deployment manifest tells your device which modules to install and how to configure them to work together.

The deployment manifest is a JSON document that describes:

- The IoT Edge agent module twin, which includes three components.
  - The container image for each module that runs on the device.
  - The credentials to access private container registries that contain module images.
  - Instructions for how each module should be created and managed.
- The IoT Edge hub module twin, which includes how messages flow between modules and eventually to IoT Hub.
- Optionally, the desired properties of any additional module twins.

All IoT Edge devices must be configured with a deployment manifest. A newly installed IoT Edge runtime reports an error code until configured with a valid manifest. You can create a deployment manifest using the Azure IoT Edge portal, REST APIs, or the IoT Hub Service SDK.

## Create a deployment manifest

At a high level, a deployment manifest is a list of module twins that are configured with their desired properties. A deployment manifest tells an IoT Edge device (or a group of devices) which modules to install and how to configure them. Deployment manifests include the desired properties for each module twin. IoT Edge devices report back the reported properties for each module.

Two modules are required in every deployment manifest: \$edgeAgent, and \$edgeHub. These modules are part of the IoT Edge runtime that manages the IoT Edge device and the modules running on it.

In addition to the two runtime modules, you can add up to 20 modules of your own to run on an IoT Edge device.

A deployment manifest that contains only the IoT Edge runtime (edgeAgent and edgeHub) is valid.

Deployment manifests follow this structure:

```
{  
    "modulesContent": {  
        "$edgeAgent": { // required  
            "properties.desired": {  
                // desired properties of the Edge agent  
                // includes the image URLs of all modules  
                // includes container registry credentials  
            }  
        },  
        "$edgeHub": { //required  
            "properties.desired": {  
                // desired properties of the Edge hub  
                // includes the routing information between modules, and to IoT Hub  
            }  
        },  
        "module1": { // optional  
            "properties.desired": {  
                // desired properties of module1  
            }  
        },  
        "module2": { // optional  
            "properties.desired": {  
                // desired properties of module2  
            }  
        },  
        ...  
    }  
}
```

## Configure modules

Define how the IoT Edge runtime installs the modules in your deployment. The IoT Edge agent is the runtime component that manages installation, updates, and status reporting for an IoT Edge device. Therefore, the `$edgeAgent` module twin requires the configuration and management information for all modules. This information includes the configuration parameters for the IoT Edge agent itself.

For a complete list of properties that can or must be included, see Properties of the IoT Edge agent and IoT Edge hub - <https://docs.microsoft.com/en-us/azure/iot-edge/module-edgeagent-edgehub>.

The \$edgeAgent properties follow this structure:

```
    "$edgeAgent": {
        "properties.desired": {
            "schemaVersion": "1.0",
            "runtime": {
                "settings": {
                    "registryCredentials": {
                        // give the edge agent access to
                        // container images that aren't public
                    }
                }
            }
        },
        "systemModules": {
            "edgeAgent": {
                // configuration and management details
            },
            "edgeHub": {
                // configuration and management details
            }
        },
        "modules": {
            "module1": { // optional
                // configuration and management details
            },
            "module2": { // optional
                // configuration and management details
            }
        }
    }
}
```

## Declare routes

The IoT Edge hub manages communication between modules, IoT Hub, and any leaf devices. Therefore, the `$edgeHub` module twin contains a desired property called routes that declares how messages are passed within a deployment. You can have multiple routes within the same deployment.

Routes are declared in the `$edgeHub` desired properties with the following syntax:

```
"$edgeHub": {  
    "properties.desired": {
```

```

    "routes": {
        "route1": "FROM <source> WHERE <condition> INTO <sink>",
        "route2": "FROM <source> WHERE <condition> INTO <sink>"
    },
}
}

```

Every route needs a source and a sink, but the condition is an optional piece that you can use to filter messages.

## Source

The source specifies where the messages come from. IoT Edge can route messages from modules or leaf devices.

Using the IoT SDKs, modules can declare specific output queues for their messages using the `ModuleClient` class. Output queues aren't necessary, but are helpful for managing multiple routes. Leaf devices can use the `DeviceClient` class of the IoT SDKs to send messages to IoT Edge gateway devices in the same way that they would send messages to IoT Hub.

The source property can be any of the following values:

Source	Description
<code>/*</code>	All device-to-cloud messages or twin change notifications from any module or leaf device
<code>/twinChangeNotifications</code>	Any twin change (reported properties) coming from any module or leaf device
<code>/messages/*</code>	Any device-to-cloud message sent by a module through some or no output, or by a leaf device
<code>/messages/modules/*</code>	Any device-to-cloud message sent by a module through some or no output
<code>/messages/modules/&lt;moduleId&gt;/*</code>	Any device-to-cloud message sent by a specific module through some or no output
<code>/messages/modules/&lt;moduleId&gt;/outputs/*</code>	Any device-to-cloud message sent by a specific module through some output
<code>/messages/modules/&lt;moduleId&gt;/outputs/&lt;output&gt;</code>	Any device-to-cloud message sent by a specific module through a specific output

## Condition

The condition is optional in a route declaration. If you want to pass all messages from the source to the sink, just leave out the `WHERE` clause entirely. Or you can use the IoT Hub query language to filter for certain messages or message types that satisfy the condition. IoT Edge routes don't support filtering messages based on twin tags or properties.

The messages that pass between modules in IoT Edge are formatted the same as the messages that pass between your devices and Azure IoT Hub. All messages are formatted as JSON and have `systemProperties`, `appProperties`, and `body` parameters.

You can build queries around any of the three parameters with the following syntax:

- System properties: `$(<propertyName>)` or `{ $(<propertyName>) }`

- Application properties: <propertyName>
- Body properties: \$body.<propertyName>

An example that is specific to IoT Edge is when you want to filter for messages that arrived at a gateway device from a leaf device. Messages that come from modules include a system property called **connectionModuleId**. So if you want to route messages from leaf devices directly to IoT Hub, use the following route to exclude module messages:

```
FROM /messages/* WHERE NOT IS_DEFINED($connectionModuleId) INTO $upstream
```

## Sink

The sink defines where the messages are sent. Only modules and IoT Hub can receive messages. Messages can't be routed to other devices. There are no wildcard options in the sink property.

The sink property can be any of the following values:

Sink	Description
\$upstream	Send the message to IoT Hub
BrokeredEndpoint("/modules/<moduleId>/inputs/<input>")	Send the message to a specific input of a specific module

IoT Edge provides at-least-once guarantees. The IoT Edge hub stores messages locally in case a route can't deliver the message to its sink. For example, if the IoT Edge hub can't connect to IoT Hub, or the target module isn't connected.

IoT Edge hub stores the messages up to the time specified in the `storeAndForwardConfiguration.timeToLiveSecs` property of the IoT Edge hub desired properties.

## Define or update desired properties

The deployment manifest specifies desired properties for each module deployed to the IoT Edge device. Desired properties in the deployment manifest overwrite any desired properties currently in the module twin.

If you do not specify a module twin's desired properties in the deployment manifest, IoT Hub won't modify the module twin in any way. Instead, you can set the desired properties programmatically.

The same mechanisms that allow you to modify device twins are used to modify module twins.

## Deployment manifest example

The following example shows what a valid deployment manifest document may look like.

```
{
  "modulesContent": {
    "$edgeAgent": {
      "properties.desired": {
        "schemaVersion": "1.0",
        "runtime": {
          "type": "docker",
          "settings": {
            "minDockerVersion": "v1.25",
            "loggingOptions": ""
          }
        }
      }
    }
  }
}
```

```
"registryCredentials": {  
    "ContosoRegistry": {  
        "username": "myacr",  
        "password": "<password>",  
        "address": "myacr.azurecr.io"  
    }  
},  
"systemModules": {  
    "edgeAgent": {  
        "type": "docker",  
        "settings": {  
            "image": "mcr.microsoft.com/azureiotedge-agent:1.0",  
            "createOptions": ""  
        }  
    },  
    "edgeHub": {  
        "type": "docker",  
        "status": "running",  
        "restartPolicy": "always",  
        "settings": {  
            "image": "mcr.microsoft.com/azureiotedge-hub:1.0",  
            "createOptions": ""  
        }  
    },  
    "modules": {  
        "SimulatedTemperatureSensor": {  
            "version": "1.0",  
            "type": "docker",  
            "status": "running",  
            "restartPolicy": "always",  
            "settings": {  
                "image": "mcr.microsoft.com/azureiotedge-simulated-temperature-sensor:1.0",  
                "createOptions": "{}"  
            }  
        },  
        "filtermodule": {  
            "version": "1.0",  
            "type": "docker",  
            "status": "running",  
            "restartPolicy": "always",  
            "settings": {  
                "image": "myacr.azurecr.io/filtermodule:latest",  
                "createOptions": "{}"  
            }  
        }  
    }  
},
```

```
$edgeHub": {  
    "properties.desired": {  
        "schemaVersion": "1.0",  
        "routes": {  
            "sensorToFilter": "FROM /messages/modules/SimulatedTemperatureSensor/outputs/temperature-  
Output INTO BrokeredEndpoint(\"/modules/filtermodule/inputs/input1\")",  
            "filterToTolothub": "FROM /messages/modules/filtermodule/outputs/output1 INTO $upstream"  
        },  
        "storeAndForwardConfiguration": {  
            "timeToLiveSecs": 10  
        }  
    }  
}
```

## Deployment Checklist

When you're ready to take your IoT Edge solution from development into production, having a checklist will help you to ensure that your solution is configured for ongoing performance.

**Note:** Each section below begins by dividing associated work into two sections: **important** to complete before going to production, or **helpful** for you to know. This distinction is helpful when prioritizing efforts.

## Device configuration

IoT Edge devices can be anything from a Raspberry Pi to a laptop to a virtual machine running on a server. You may have access to the device either physically or through a virtual connection, or it may be isolated for extended periods of time. Either way, you want to make sure that it's configured to work appropriately.

- Important
  - Install production certificates
  - Have a device management plan
  - Use Moby as the container engine
- Helpful
  - Choose upstream protocol

## Install production certificates

Every IoT Edge device in production needs a device certificate authority (CA) certificate installed on it. That CA certificate is then declared to the IoT Edge runtime in the config.yaml file. To make development and testing easier, the IoT Edge runtime creates temporary certificates if no certificates are declared in the config.yaml file. However, these temporary certificates expire after three months and aren't secure for production scenarios.

The steps for configuring the certificates are the same whether the device is going to be used as a gateway or not.

You can generate sample certificates for testing, but never use sample certificates in production.

## Have a device management plan

Before you put any device in production you should know how you're going to manage future updates. For an IoT Edge device, the list of components to update may include:

- Device firmware
- Operating system libraries
- Container engine, like Moby
- IoT Edge daemon
- CA certificates

The current methods for updating the IoT Edge daemon require physical or SSH access to the IoT Edge device. If you have many devices to update, consider adding the update steps to a script or use an automation tool like Ansible.

## Use Moby as the container engine

A container engine is a prerequisite for any IoT Edge device. Only moby-engine is supported in production. Other container engines, like Docker, do work with IoT Edge and it's ok to use these engines for development. The moby-engine can be redistributed when used with Azure IoT Edge, and Microsoft provides servicing for this engine.

## Choose upstream protocol

The protocol (and therefore the port used) for upstream communication to IoT Hub can be configured for both the IoT Edge agent and the IoT Edge hub. The default protocol is AMQP, but you may want to change that depending on your network setup.

The two runtime modules both have an **UpstreamProtocol** environment variable. The valid values for the variable are:

- MQTT
- AMQP
- MQTTWS
- AMQPWS

Configure the UpstreamProtocol variable for the IoT Edge agent in the config.yaml file on the device itself. For example, if your IoT Edge device is behind a proxy server that blocks AMQP ports, you may need to configure the IoT Edge agent to use AMQP over WebSocket (AMQPWS) to establish the initial connection to IoT Hub.

Once your IoT Edge device connects, be sure to continue configuring the UpstreamProtocol variable for both runtime modules in future deployments. An example of this process is provided in [Configure an IoT Edge device to communicate through a proxy server](https://docs.microsoft.com/en-us/azure/iot-edge/how-to-configure-proxy-support) <https://docs.microsoft.com/en-us/azure/iot-edge/how-to-configure-proxy-support>.

## Deployment

For your Deployment checklist, the following items fall into the **helpful** category.

## Be consistent with upstream protocol

If you configured the IoT Edge agent on your IoT Edge device to use a different protocol than the default AMQP, then you should declare the same protocol in all future deployments. For example, if your IoT Edge device is behind a proxy server that blocks AMQP ports, you probably configured the device to connect over AMQP over WebSocket (AMQPWS). When you deploy modules to the device, configure the same AMQPWS protocol for the IoT Edge agent and IoT Edge hub, or else the default AMQP will override the settings and prevent you from connecting again.

You only have to configure the `UpstreamProtocol` environment variable for the IoT Edge agent and IoT Edge hub modules. Any additional modules adopt whatever protocol is set in the runtime modules.

## Set up host storage for system modules

The IoT Edge hub and agent modules use local storage to maintain state and enable messaging between modules, devices, and the cloud. For better reliability and performance, configure the system modules to use storage on the host filesystem.

## Reduce memory space used by IoT Edge hub

If you're deploying constrained devices with limited memory available, you can configure IoT Edge hub to run in a more streamlined capacity and use less disk space. These configurations do limit the performance of the IoT Edge hub, however, so find the right balance that works for your solution.

Three guidelines that you can follow are:

- Don't optimize for performance on constrained devices
- Disable unused protocols
- Do not use debug versions of module images

## Container management

For your Container management checklist, the following items fall into the **important** category.

## Manage access to your container registry

Before you deploy modules to production IoT Edge devices, ensure that you control access to your container registry so that outsiders can't access or make changes to your container images. Use a private, not public, container registry to manage container images.

## Use tags to manage versions

A tag is a docker concept that you can use to distinguish between versions of docker containers. Tags are suffixes like 1.0 that go on the end of a container repository. For example, `mcr.microsoft.com/azureiot-edge-agent:1.0`. Tags are mutable and can be changed to point to another container at any time, so your team should agree on a convention to follow as you update your module images moving forward.

Tags also help you to enforce updates on your IoT Edge devices. When you push an updated version of a module to your container registry, increment the tag. Then, push a new deployment to your devices with the tag incremented. The container engine will recognize the incremented tag as a new version and will pull the latest module version down to your device.

## Networking

For your Networking checklist, the following items fall into the **helpful** category.

### Review outbound/inbound configuration

Communication channels between Azure IoT Hub and IoT Edge are always configured to be outbound. For most IoT Edge scenarios, only three connections are necessary. The container engine needs to connect with the container registry (or registries) that holds the module images. The IoT Edge runtime needs to connect with IoT Hub to retrieve device configuration information, and to send messages and telemetry. And if you use automatic provisioning, the IoT Edge daemon needs to connect to the Device Provisioning Service.

### Allow connections from IoT Edge devices

If your networking setup requires that you explicitly permit connections made from IoT Edge devices, review the following list of IoT Edge components:

- IoT Edge agent opens a persistent AMQP/MQTT connection to IoT Hub, possibly over WebSockets.
- IoT Edge hub opens a single persistent AMQP connection or multiple MQTT connections to IoT Hub, possibly over WebSockets.
- IoT Edge daemon makes intermittent HTTPS calls to IoT Hub.

In all three cases, the DNS name would match the pattern \*.azure-devices.net.

Additionally, the Container engine makes calls to container registries over HTTPS. To retrieve the IoT Edge runtime container images, the DNS name is mcr.microsoft.com. The container engine connects to other registries as configured in the deployment.

### Configure communication through a proxy

If your devices are going to be deployed on a network that uses a proxy server, they need to be able to communicate through the proxy to reach IoT Hub and container registries.

## Solution management

For your Solution management checklist, the following items fall into the **helpful** category.

### Set up logs and diagnostics

On Linux, the IoT Edge daemon uses journals as the default logging driver. You can use the command-line tool `journalctl` to query the daemon logs. On Windows, the IoT Edge daemon uses PowerShell diagnostics. Use `Get-IoTEdgeLog` to query logs from the daemon. IoT Edge modules use the JSON driver for logging, which is the default.

When you're testing an IoT Edge deployment, you can usually access your devices to retrieve logs and troubleshoot. In a deployment scenario, you may not have that option. Consider how you're going to gather information about your devices in production. One option is to use a logging module that collects information from the other modules and sends it to the cloud.

## Place limits on log size

By default the Moby container engine does not set container log size limits. Over time this can lead to the device filling up with logs and running out of disk space.

## Consider tests and CI/CD pipelines

For the most efficient IoT Edge deployment scenario, consider integrating your production deployment into your testing and CI/CD pipelines. Azure IoT Edge supports multiple CI/CD platforms, including Azure DevOps.

# Edge Gateway Devices

## Using an Edge Device as a Gateway

Gateways in IoT Edge solutions provide device connectivity and edge analytics to IoT devices that otherwise wouldn't have those capabilities. Azure IoT Edge can be used to satisfy all needs for an IoT gateway regardless of whether they are related to connectivity, identity, or edge analytics.

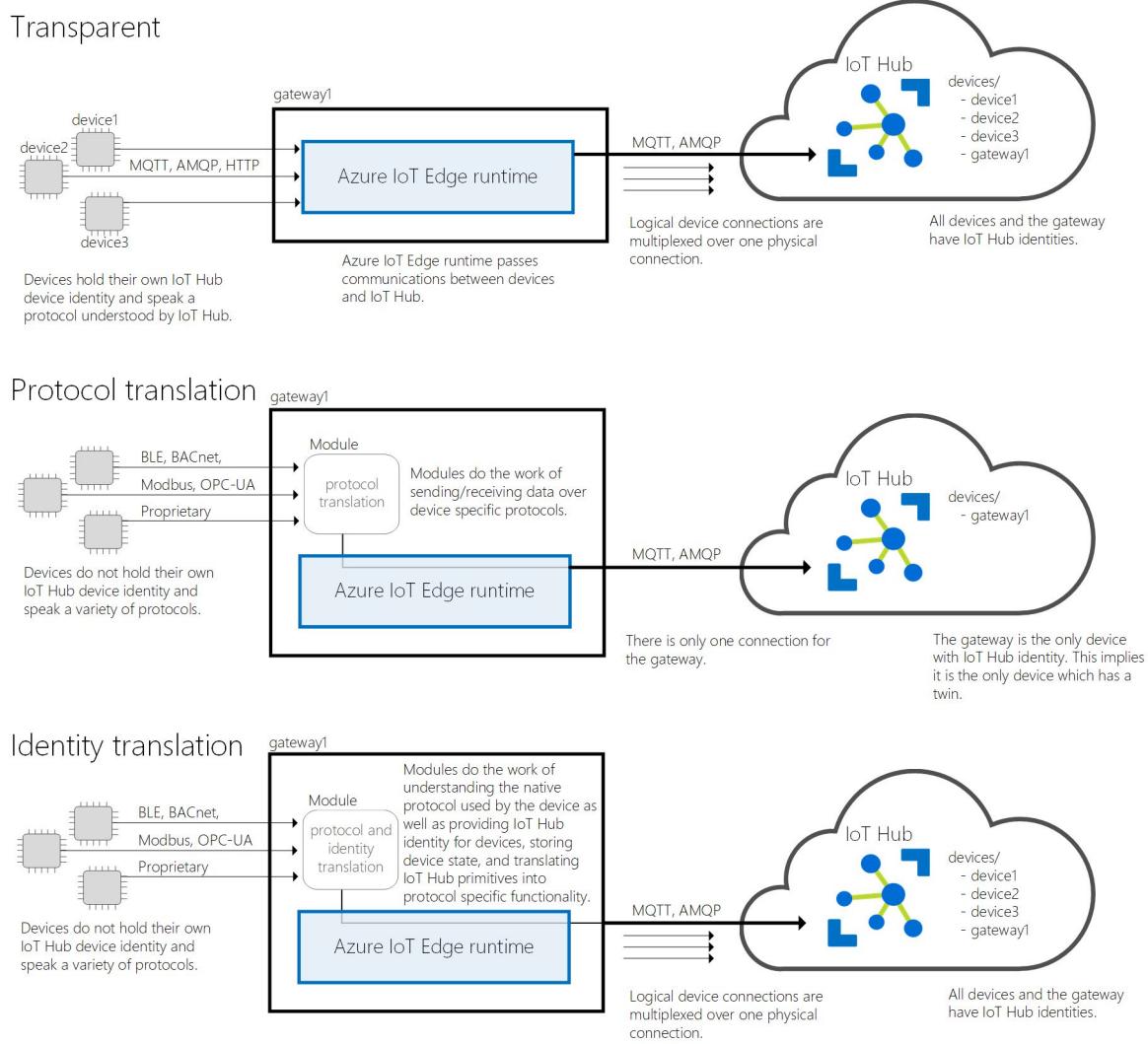
For the purpose of this training, gateway patterns refer to characteristics of downstream device connectivity and device identity, not how device data is processed on the gateway.

### Patterns

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

- Transparent – Devices that theoretically could connect to IoT Hub can connect to a gateway device instead. The downstream devices have their own IoT Hub identities and are using any of the MQTT, AMQP, or HTTP protocols. The gateway simply passes communications between the devices and IoT Hub. The devices are unaware that they are communicating with the cloud via a gateway, and a user interacting with the devices in IoT Hub is unaware of the intermediate gateway device. Thus, the gateway is transparent. Refer to [Create a transparent gateway<sup>1</sup>](#) for specifics on using an IoT Edge device as a transparent gateway.
- Protocol translation – Also known as an opaque gateway pattern, devices that do not support MQTT, AMQP, or HTTP can use a gateway device to send data to IoT Hub on their behalf. The gateway understands the protocol used by the downstream devices, and is the only device that has an identity in IoT Hub. All information looks like it is coming from one device, the gateway. Downstream devices must embed additional identifying information in their messages if cloud applications want to analyze the data on a per-device basis. Additionally, IoT Hub primitives like twins and methods are only available for the gateway device, not downstream devices.
- Identity translation - Devices that cannot connect to IoT Hub can connect to a gateway device, instead. The gateway provides IoT Hub identity and protocol translation on behalf of the downstream devices. The gateway is smart enough to understand the protocol used by the downstream devices, provide them identity, and translate IoT Hub primitives. Downstream devices appear in IoT Hub as first-class devices with twins and methods. A user can interact with the devices in IoT Hub and is unaware of the intermediate gateway device.

<sup>1</sup> <https://docs.microsoft.com/en-us/azure/iot-edge/how-to-create-transparent-gateway>



## Use cases

All gateway patterns provide the following benefits:

- Analytics at the edge – Use AI services locally to process data coming from downstream devices without sending full-fidelity telemetry to the cloud. Find and react to insights locally and only send a subset of data to IoT Hub.
- Downstream device isolation – The gateway device can shield all downstream devices from exposure to the Internet. It can sit in between an OT network that does not have connectivity and an IT network that provides access to the web.
- Connection multiplexing - All devices connecting to IoT Hub through an IoT Edge gateway use the same underlying connection.
- Traffic smoothing - The IoT Edge device will automatically implement exponential backoff if IoT Hub throttles traffic, while persisting the messages locally. This benefit makes your solution resilient to spikes in traffic.

- Offline support - The gateway device stores messages and twin updates that cannot be delivered to IoT Hub.

A gateway that does protocol translation can also perform edge analytics, device isolation, traffic smoothing, and offline support to existing devices and new devices that are resource constrained. Many existing devices are producing data that can power business insights; however they were not designed with cloud connectivity in mind. Opaque gateways allow this data to be unlocked and used in an IoT solution.

A gateway that does identity translation provides the benefits of protocol translation and additionally allows for full manageability of downstream devices from the cloud. All devices in your IoT solution show up in IoT Hub regardless of the protocol they use.

## Cheat sheet

Here is a quick cheat sheet that compares IoT Hub primitives when using transparent, opaque (protocol), and proxy gateways.

	Transparent gateway	Protocol translation	Identity translation
Identities stored in the IoT Hub identity registry	Identities of all connected devices	Only the identity of the gateway device	Identities of all connected devices
Device twin	Each connected device has its own device twin	Only the gateway has a device and module twins	Each connected device has its own device twin
Direct methods and cloud-to-device messages	The cloud can address each connected device individually	The cloud can only address the gateway device	The cloud can address each connected device individually
IoT Hub throttles and quotas	Apply to each device	Apply to the gateway device	Apply to each device

When using an opaque gateway (protocol translation) pattern, all devices connecting through that gateway share the same cloud-to-device queue, which can contain at most 50 messages. It follows that the opaque gateway pattern should be used only when few devices are connecting through each field gateway, and their cloud-to-device traffic is low.

## Authenticate a Downstream Device to a Transparent Gateway

In a transparent gateway scenario, downstream devices (sometimes called leaf devices or child devices) need identities in IoT Hub like any other device.

There are three general steps to set up a successful transparent gateway connection.

- The gateway device needs to be able to securely connect to downstream devices, receive communications from downstream devices, and route messages to the proper destination. For more information, see Configure an IoT Edge device to act as a transparent gateway.
- The downstream device needs to have a device identity to be able to authenticate with IoT Hub, and know to communicate through its gateway device.
- The downstream device needs to be able to securely connect to its gateway device. For more information, see Connect a downstream device to an Azure IoT Edge gateway.

Downstream devices can authenticate with IoT Hub using one of three methods: symmetric keys (sometimes referred to as shared access keys), X.509 self-signed certificates, or X.509 certificate authority (CA) signed certificates. The authentication steps are similar to the steps used to set up any non-IoT-Edge device with IoT Hub, with small differences to declare the gateway relationship.

## Prerequisites

An IoT Edge device configured to act as a transparent gateway.

If you're using X.509 authentication for your downstream device, you need to use the same certificate generating script that you used to configure the transparent gateway.

**Note:** In the instructional materials below, we use a gateway hostname to create the certificates, and then refer to the gateway hostname in the connection string of the downstream devices. The gateway hostname is declared in the **hostname** parameter of the config.yaml file on the IoT Edge gateway device. The gateway hostname needs to be resolvable to an IP Address, either using DNS or a host file entry.

## Symmetric key authentication

Symmetric key authentication, or shared access key authentication, is the simplest way to authenticate with IoT Hub. With symmetric key authentication, a base64 key is associated with your IoT device ID in IoT Hub. You include that key in your IoT applications so that your device can present it when it connects to IoT Hub.

## Create the device identity

Add a new IoT device in your IoT hub, using either the Azure portal, Azure CLI, or the IoT extension for Visual Studio Code. Remember that downstream devices need to be identified in IoT Hub as regular IoT device, not IoT Edge devices.

When you create the new device identity, provide the following information:

- Create an ID for your device.
- Select **Symmetric key** as the authentication type.
- Optionally, choose to **Set a parent device** and select the IoT Edge gateway device that this downstream device will connect through. This step is optional for symmetric key authentication, but it's recommended because setting a parent device enables offline capabilities for your downstream device. You can always update the device details to add or change the parent later.

The screenshot shows the 'Create a device' dialog box. Key fields highlighted with red boxes are: 'Device ID' (containing 'The ID of the new device'), 'Authentication type' (with 'Symmetric key' selected), 'Parent device' (with 'No parent device' and 'Set a parent device' options), and 'Set a parent device' (which is also highlighted). A blue 'Save' button is at the bottom.

You can use the IoT extension for Azure CLI to complete the same operation. The following example creates a new IoT device with symmetric key authentication and assigns a parent device:

```
az iot hub device-identity create -n {iothub name} -d {device ID} --pd {gateway device ID}
```

## Connect to IoT Hub through a gateway

The same process is used to authenticate regular IoT devices to IoT Hub with symmetric keys also applies to downstream devices. The only difference is that you need to add a pointer to the gateway device to route the connection or, in offline scenarios, to handle the authentication on behalf of IoT Hub.

For symmetric key authentication, there's no additional steps that you need to take on your device for it to authenticate with IoT Hub. You still need the certificates in place so that your downstream device can connect to its gateway device, as described in Connect a downstream device to an Azure IoT Edge gateway.

After creating an IoT device identity in the portal, you can retrieve its primary or secondary keys. One of these keys needs to be included in the connection string that you include in any application that communicates with IoT Hub. For symmetric key authentication, IoT Hub provides the fully formed connection string in the device details for your convenience. You need to add extra information about the gateway device to the connection string.

Symmetric key connection strings for downstream devices need the following components:

- The IoT hub that the device connects to: Hostname={iothub name}.azure-devices.net
- The device ID registered with the hub: DeviceID={device ID}
- Either the primary or secondary key: SharedAccessKey={key}

- The gateway device that the device connects through. Provide the hostname value from the IoT Edge gateway device's config.yaml file: GatewayHostName={gateway hostname}

All together, a complete connection string looks like:

```
HostName=myiothub.azure-devices.net;DeviceId=myDownstreamDevice;SharedAccessKey=xxxxyyzzz;GatewayHostName=myGatewayDevice
```

If you established a parent/child relationship for this downstream device, then you can simplify the connection string by calling the gateway directly as the connection host. For example:

```
HostName=myGatewayDevice;DeviceId=myDownstreamDevice;SharedAccessKey=xxxxyyzzz
```

## X.509 authentication

There are two ways to authenticate an IoT device using X.509 certificates. Whichever way you choose to authenticate, the steps to connect your device to IoT Hub are the same. Choose either self-signed or CA-signed certs for authentication, then continue to learn how to connect to IoT Hub.

### Create the device identity with X.509 self-signed certificates

For X.509 self-signed authentication, sometimes referred to as thumbprint authentication, you need to create new certificates to place on your IoT device. These certificates have a thumbprint in them that you share with IoT Hub for authentication.

The easiest way to test this scenario is to use the same machine that you used to create certificates for the transparent gateway.

That machine should already be set up with the right tool, root CA certificate, and intermediate CA certificate to create the IoT device certificates. You can copy the final certificates and their private keys over to your downstream device afterwards. Complete the following steps

- set up openssl on your machine
- clone the IoT Edge repo to access certificate creation scripts
- make a working directory called <WRKDIR> to hold the certificates

The default certificates are meant for developing and testing, so only last 30 days.

You should have created a root CA certificate and an intermediate certificate.

1. Navigate to your working directory in either a bash or PowerShell window.
2. Create two certificates (primary and secondary) for the downstream device. Provide your device name and then the primary or secondary label. This information is used to name the files so that you can keep track of certificates for multiple devices.

PowerShell

```
New-CACertsDevice "<device name>-primary"  
New-CACertsDevice "<device name>-secondary"
```

Bash

```
./certGen.sh create_device_certificate "<device name>-primary"  
./certGen.sh create_device_certificate "<device name>-secondary"
```

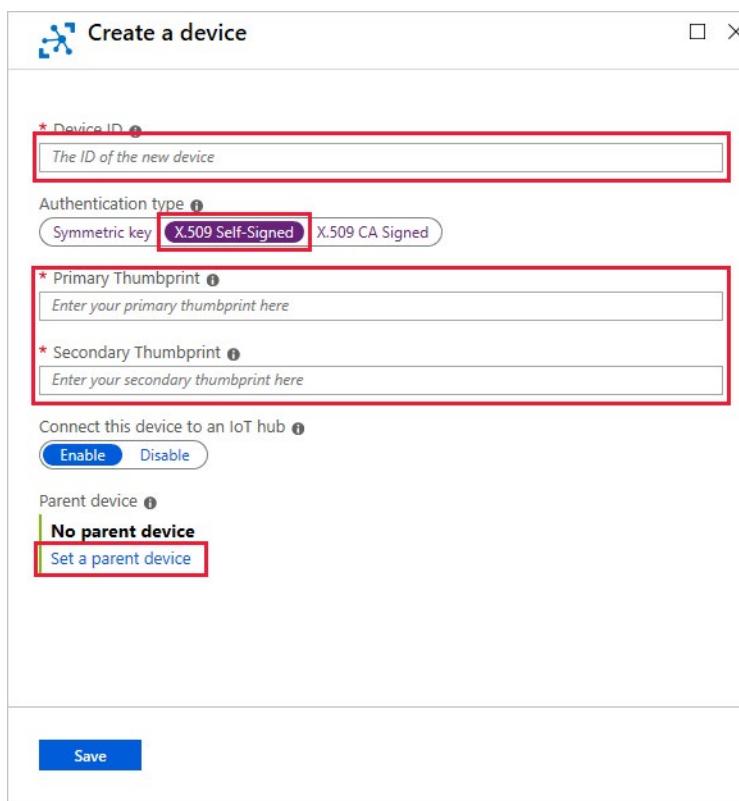
3. Retrieve the SHA1 fingerprint (called a thumbprint in the IoT Hub interface) from each certificate, which is a 40 hexadecimal character string. Use the following openssl command to view the certificate and find the fingerprint:

PowerShell / bash

```
openssl x509 -in <WORKDIR>/certs/iot-device-<device name>-primary.cert.pem -text -fingerprint | sed 's/[:]//g'
```

4. Navigate to your IoT hub in the Azure portal and create a new IoT device identity with the following values:

- Select X.509 Self-Signed as the authentication type.
- Paste the hexadecimal strings that you copied from your device's primary and secondary certificates.
- Select Set a parent device and choose the IoT Edge gateway device that this downstream device will connect through. A parent device is required for X.509 authentication of a downstream device.



5. Copy the following files to any directory on your downstream device:

```
<WRKDIR>\certs\azure-iot-test-only.root.ca.cert.pem
<WRKDIR>\certs\iot-device-<device name>*.cert.pem
<WRKDIR>\certs\iot-device-<device id>*.cert.pfx
<WRKDIR>\certs\iot-device-<device name>*-full-chain.cert.pem
<WRKDIR>\private\iot-device-<device name>*.key.pem
```

You'll reference these files in the leaf device applications that connect to IoT Hub. You can use a service like Azure Key Vault or a function like Secure copy protocol to move the certificate files.

You can use the IoT extension for Azure CLI to complete the same device creation operation. The following example creates a new IoT device with X.509 self-signed authentication and assigns a parent device:

```
az iot hub device-identity create -n {iothub name} -d {device ID} --pd {gateway device ID} --am x509_thumbprint --ptp {primary thumbprint} --stp {secondary thumbprint}
```

## Create the device identity with X.509 CA signed certificates

For X.509 certificate authority (CA) signed authentication, you need a root CA certificate registered in IoT Hub that you use to sign certificates for your IoT device. Any device using a certificate that was issued by the root CA certificate or any of its intermediate certificates will be permitted to authenticate.

1. Follow the instructions in the Register X.509 CA certificates to your IoT hub section of Set up X.509 security in your Azure IoT hub <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-security-x509-get-started#register-x509-ca-certificates-to-your-iot-hub>. In that section, you perform the following steps:
  - Upload a root CA certificate. If you're using the certificates that you created in the transparent gateway article, upload <WRKDIR>/certs/azure-iot-test-only.root.ca.cert.pem as the root certificate file.
  - Verify that you own that root CA certificate. You can verify possession with the cert tools in <WRKDIR>.

PowerShell

```
New-CACertsVerificationCert "<verification code from Azure portal>"
```

Bash

```
./certGen.sh create_verification_certificate <verification code from Azure portal>"
```

2. Follow the instructions in the Create an X.509 device for your IoT hub section of Set up X.509 security in your Azure IoT hub. In that section, you perform the following steps:
  - Add a new device. Provide a lowercase name for device ID, and choose the authentication type X.509 CA Signed.
  - Set a parent device. For downstream devices, select Set a parent device and choose the IoT Edge gateway device that will provide the connection to IoT Hub.
3. Create a certificate chain for your downstream device. Use the same root CA certificate that you uploaded to IoT Hub to make this chain. Use the same lowercase device ID that you gave to your device identity in the portal.

PowerShell

```
New-CACertsDevice "<device id>"
```

Bash

```
./certGen.sh create_device_certificate "<device id>"
```

4. Copy the following files to any directory on your downstream device:

```
<WRKDIR>\certs\azure-iot-test-only.root.ca.cert.pem
<WRKDIR>\certs\iot-device-<device id>*.cert.pem
<WRKDIR>\certs\iot-device-<device id>*.cert.pfx
<WRKDIR>\certs\iot-device-<device id>*-full-chain.cert.pem
<WRKDIR>\private\iot-device-<device id>*.key.pem
```

You'll reference these files in the leaf device applications that connect to IoT Hub. You can use a service like Azure Key Vault or a function like Secure copy protocol to move the certificate files.

You can use the IoT extension for Azure CLI to complete the same device creation operation. The following example creates a new IoT device with X.509 CA signed authentication and assigns a parent device:

```
az iot hub device-identity create -n {iothub name} -d {device ID} --pd {gateway device ID} --am x509_ca
```

## Connect to IoT Hub through a gateway

Each Azure IoT SDK handles X.509 authentication a little differently. However, the same process is used to authenticate regular IoT devices to IoT Hub with X.509 certificates also applies to downstream devices. The only difference is that you need to add a pointer to the gateway device to route the connection or, in offline scenarios, to handle the authentication on behalf of IoT Hub. In general, you can follow the same X.509 authentication steps for all IoT Hub devices, then simply replace the value of Hostname in the connection string to be the hostname of your gateway device.

**Important:** The following code sample demonstrates how the IoT Hub SDKs use certificates to authenticate devices. In a production deployment, you should store all secrets like private or SAS keys in a hardware secure module (HSM).

For an example of a C# program authenticating to IoT Hub with X.509 certificates, see Set up X.509 security in your Azure IoT hub <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-security-x509-get-started#authenticate-your-x509-device-with-the-x509-certificates>. Some of the key lines of that sample are included here to demonstrate the authentication process.

When declaring the hostname for your DeviceClient instance, use the IoT Edge gateway device's hostname. The hostname can be found in the gateway device's config.yaml file.

If you're using the test certificates provided by the IoT Edge git repository, the key to the certificates is **1234**.

```
try
{
    var cert = new X509Certificate2(@"<absolute-path-to-your-device-pfx-file>", "1234");
    var auth = new DeviceAuthenticationWithX509Certificate("<device-id>", cert);
    var deviceClient = DeviceClient.Create("<gateway hostname>", auth, TransportType.Amqp_Tcp_Only);

    if (deviceClient == null)
    {
        Console.WriteLine("Failed to create DeviceClient!");
    }
    else
    {
        Console.WriteLine("Successfully created DeviceClient!");
        SendEvent(deviceClient).Wait();
    }
}
```

```

        Console.WriteLine("Exiting...\n");
    }
    catch (Exception ex)
    {
        Console.WriteLine("Error in sample: {0}", ex.Message);
    }
}

```

Code samples for other coding languages are available as follows:

Language	Description / Link
C	iotedge_downstream_device_sample <a href="https://github.com/Azure/azure-iot-sdk-c/tree/x509_edge_bugbash/iothub_client/samples/iotedge_downstream_device_sample">https://github.com/Azure/azure-iot-sdk-c/tree/x509_edge_bugbash/iothub_client/samples/iotedge_downstream_device_sample</a>
Node.js	simple_sample_device_x509.js <a href="https://github.com/Azure/azure-iot-sdk-node/blob/master/device/samples/simple_sample_device_x509.js">https://github.com/Azure/azure-iot-sdk-node/blob/master/device/samples/simple_sample_device_x509.js</a>
Python	The Python SDK currently only supports using X509 certificates and keys from files, not ones which are defined inline
Java	SendEventX509.java <a href="https://github.com/Azure/azure-iot-sdk-python/blob/master/device/samples/iothub_client_sample_x509.py">https://github.com/Azure/azure-iot-sdk-python/blob/master/device/samples/iothub_client_sample_x509.py</a>

## Configure the Transparent Gateway Connection for a Downstream Device

In a transparent gateway scenario, one or more devices can pass their messages through a single gateway device that maintains the connection to IoT Hub. A downstream device can be any application or platform that has an identity created with the Azure IoT Hub cloud service. In many cases, these applications use the Azure IoT device SDK. A downstream device could even be an application running on the IoT Edge gateway device itself.

The sections below describe common problems with downstream device connections and guide you in setting up your downstream devices by:

- Explaining transport layer security (TLS) and certificate fundamentals.
- Explaining how TLS libraries work across different operating systems and how each operating system deals with certificates.
- Walking through Azure IoT samples in several languages to help get you started.

## Prerequisites

You will need to have the **azure-iot-test-only.root.ca.cert.pem** certificate file that was generated for your transparent gateway available on your downstream device. Your downstream device uses this certificate to validate the identity of the gateway device.

## Prepare a downstream device

A downstream device can be any application or platform that has an identity created with the Azure IoT Hub cloud service. In many cases, these applications use the Azure IoT device SDK. A downstream device could even be an application running on the IoT Edge gateway device itself. However, another IoT Edge device cannot be downstream of an IoT Edge gateway.

**Note:** IoT devices that have identities registered in IoT Hub can use module twins to isolate different process, hardware, or functions on a single device. IoT Edge gateways support downstream module connections using symmetric key authentication but not X.509 certificate authentication.

To connect a downstream device to an IoT Edge gateway, you need two things:

- A device or application that's configured with an IoT Hub device connection string appended with information to connect it to the gateway.  
This step is explained in [Authenticate a downstream device to Azure IoT Hub](#).
- The device or application has to trust the gateway's root CA certificate to validate the TLS connections to the gateway device.

This step is explained in detail in the rest of this article. This step can be performed one of two ways: by installing the CA certificate in the operating system's certificate store, or (for certain languages) by referencing the certificate within applications using the Azure IoT SDKs.

## Transport layer security and certificate fundamentals

The challenge of securely connecting downstream devices to IoT Edge is just like any other secure client/server communication that occurs over the Internet. A client and a server securely communicate over the Internet using Transport layer security (TLS). TLS is built using standard Public key infrastructure (PKI) constructs called certificates. TLS is a fairly involved specification and addresses a wide range of topics related to securing two endpoints. This section summarizes the concepts relevant for you to securely connect devices to an IoT Edge gateway.

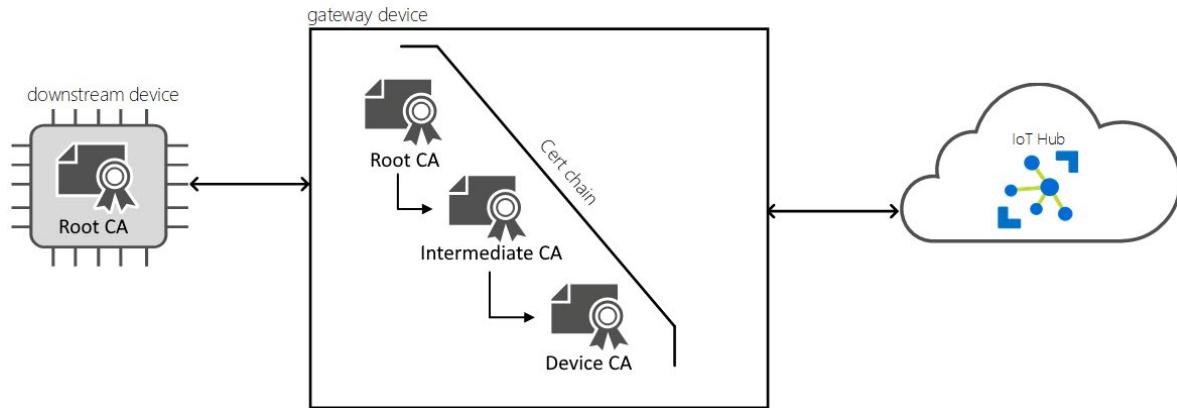
When a client connects to a server, the server presents a chain of certificates, called the server certificate chain. A certificate chain typically comprises a root certificate authority (CA) certificate, one or more intermediate CA certificates, and finally the server's certificate itself. A client establishes trust with a server by cryptographically verifying the entire server certificate chain. This client validation of the server certificate chain is called server chain validation. The client cryptographically challenges the service to prove possession of the private key associated with the server certificate in a process called proof of possession. The combination of server chain validation and proof of possession is called server authentication. To validate a server certificate chain, a client needs a copy of the root CA certificate that was used to create (or issue) the server's certificate. Normally when connecting to websites, a browser comes pre-configured with commonly used CA certificates so the client has a seamless process.

When a device connects to Azure IoT Hub, the device is the client and the IoT Hub cloud service is the server. The IoT Hub cloud service is backed by a root CA certificate called Baltimore CyberTrust Root, which is publicly available and widely used. Since the IoT Hub CA certificate is already installed on most devices, many TLS implementations (OpenSSL, Schannel, LibreSSL) automatically use it during server certificate validation. A device that may successfully connect to IoT Hub may have issues trying to connect to an IoT Edge gateway.

When a device connects to an IoT Edge gateway, the downstream device is the client and the gateway device is the server. Azure IoT Edge allows operators (or users) to build gateway certificate chains however they see fit. The operator may choose to use a public CA certificate, like Baltimore, or use a self-signed (or in-house) root CA certificate. Public CA certificates often have a cost associated with them, so are

typically used in production scenarios. Self-signed CA certificates are preferred for development and testing. The transparent gateway setup articles listed in the introduction use self-signed root CA certificates.

When you use a self-signed root CA certificate for an IoT Edge gateway, it needs to be installed on or provided to all the downstream devices attempting to connect to the gateway.



## Provide the root CA certificate

To verify the gateway device's certificates, the downstream device needs its own copy of the root CA certificate. If you used the scripts provided in the IoT Edge git repository to create test certificates, then the root CA certificate is called `azure-iot-test-only.root.ca.cert.pem`. If you haven't already as part of the other downstream device preparation steps, move this certificate file to any directory on your downstream device. You can use a service like Azure Key Vault or a function like Secure copy protocol to move the certificate file.

## Install certificates in the OS

Installing the root CA certificate in the operating system's certificate store generally allows most applications to use the root CA certificate. There are some exceptions, like NodeJS applications that don't use the OS certificate store but rather use the Node runtime's internal certificate store. If you can't install the certificate at the operating system level, you will need to use the Azure IoT SDKs to install certificates.

### Ubuntu

The following commands are an example of how to install a CA certificate on an Ubuntu host. This example assumes that you're using the `azure-iot-test-only.root.ca.cert.pem` certificate from the prerequisites articles, and that you've copied the certificate into a location on the downstream device.

```
sudo cp <path>/azure-iot-test-only.root.ca.cert.pem /usr/local/share/ca-certificates/azure-iot-test-only.root.ca.cert.pem.crt  
sudo update-ca-certificates
```

You should see a message that says, "Updating certificates in /etc/ssl/certs... 1 added, 0 removed; done."

## Windows

The following steps are an example of how to install a CA certificate on a Windows host. This example assumes that you're using the **azure-iot-test-only.root.ca.cert.pem** certificate from the prerequisites articles, and that you've copied the certificate into a location on the downstream device.

You can install certificates using PowerShell's Import-Certificate as an administrator:

```
import-certificate <file path>\azure-iot-test-only.root.ca.cert.pem -certstorelocation cert:\LocalMachine\root
```

You can also install certificates using the certlm utility:

1. In the Start menu, search for and select **Manage computer certificates**. A utility called **certlm** opens.
2. Navigate to **Certificates - Local Computer > Trusted Root Certification Authorities**.
3. Right-click **Certificates** and select **All Tasks > Import**. The certificate import wizard should launch.
4. Follow the steps as directed and import certificate file <path>/azure-iot-test-only.root.ca.cert.pem. When completed, you should see a "Successfully imported" message.

Typically applications use the Windows provided TLS stack called Schannel to securely connect over TLS. Schannel requires that any certificates be installed in the Windows certificate store before attempting to establish a TLS connection.

## About the Module 6 Labs

### Module Labs

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

- Lab 11: Introduction to IoT Edge Deployments
- Lab 12: Implement an IoT Edge gateway

**WARNING:** Be prepared for UI changes

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

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

## Module 7 Azure IoT Edge Modules and Containers

### Develop Custom Edge Modules

#### Introduction to Module Development

Azure IoT Edge modules can connect with other Azure services and contribute to your larger cloud data pipeline.

#### IoT Edge runtime environment

The IoT Edge runtime provides the infrastructure to integrate the functionality of multiple IoT Edge modules and to deploy them onto IoT Edge devices. At a high level, any program can be packaged as an IoT Edge module. However, to take full advantage of IoT Edge communication and management functionalities, a program running in a module can connect to the local IoT Edge hub, integrated in the IoT Edge runtime.

#### Using the IoT Edge hub

The IoT Edge hub provides two main functionalities: proxy to IoT Hub, and local communications.

#### IoT Hub primitives

IoT Hub sees a module instance analogously to a device, in the sense that:

- it has a module twin that is distinct and isolated from the device twin and the other module twins of that device;
- it can send device-to-cloud messages;
- it can receive direct methods targeted specifically at its identity.

Currently, a module cannot receive cloud-to-device messages nor use the file upload feature.

When writing a module, you can use the Azure IoT Device SDK to connect to the IoT Edge hub and use the above functionality as you would when using IoT Hub with a device application, the only difference being that, from your application back-end, you have to refer to the module identity instead of the device identity.

## Device-to-cloud messages

To enable complex processing of device-to-cloud messages, IoT Edge hub provides declarative routing of messages between modules, and between modules and IoT Hub. Declarative routing allows modules to intercept and process messages sent by other modules and propagate them into complex pipelines. For more information, see [deploy modules and establish routes in IoT Edge](#).

An IoT Edge module, as opposed to a normal IoT Hub device application, can receive device-to-cloud messages that are being proxied by its local IoT Edge hub in order to process them.

IoT Edge hub propagates the messages to your module based on declarative routes described in the deployment manifest. When developing an IoT Edge module, you can receive these messages by setting message handlers.

To simplify the creation of routes, IoT Edge adds the concept of module input and output endpoints. A module can receive all device-to-cloud messages routed to it without specifying any input, and can send device-to-cloud messages without specifying any output. Using explicit inputs and outputs, though, makes routing rules simpler to understand.

Finally, device-to-cloud messages handled by the Edge hub are stamped with the following system properties:

Property	Description
\$connectionDeviceId	The device ID of the client that sent the message
\$connectionModuleId	The module ID of the module that sent the message
\$inputName	The input that received this message. Can be empty.
\$outputName	The output used to send the message. Can be empty.

## Connecting to IoT Edge hub from a module

Connecting to the local IoT Edge hub from a module involves two steps:

1. Create a `ModuleClient` instance in your application.
2. Make sure your application accepts the certificate presented by the IoT Edge hub on that device.

Create a `ModuleClient` instance to connect your module to the IoT Edge hub running on the device, similar to how `DeviceClient` instances connect IoT devices to IoT Hub. For more information about the `ModuleClient` class and its communication methods, see the API reference for your preferred SDK language: C#, C, Python, Java, or Node.js.

## Azure IoT Edge Supported Systems

It is important to understand which systems and components are supported by IoT Edge, whether officially or in preview.

## Language and Device Architecture Support

IoT Edge supports multiple operating systems, device architectures, and development languages so that you can build the scenario that matches your needs. Use this section to understand your options for developing custom IoT Edge modules.

### Linux

For all languages in the following table, IoT Edge supports development for AMD64 and ARM32 Linux devices.

Development language	Development tools
C	Visual Studio Code Visual Studio 2017/2019
C#	Visual Studio Code Visual Studio 2017/2019
Java	Visual Studio Code
Node.js	Visual Studio Code
Python	Visual Studio Code

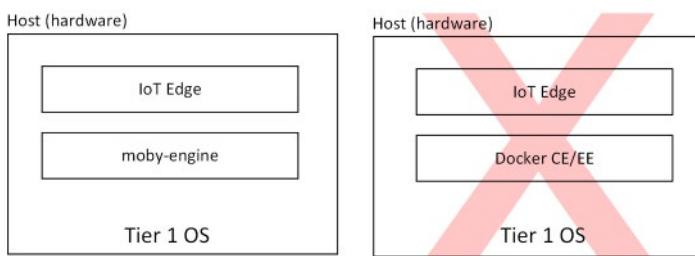
### Windows

For all languages in the following table, IoT Edge supports development for AMD64 Windows devices.

Development language	Development tools
C	Visual Studio 2017/2019
C#	Visual Studio Code (no debugging capabilities) Visual Studio 2017/2019

## Container Engines

Azure IoT Edge modules are implemented as containers, so IoT Edge needs a container engine to launch them. Microsoft provides a container engine, moby-engine, to fulfill this requirement. This container engine is based on the Moby open-source project. Docker CE and Docker EE are other popular container engines. They're also based on the Moby open-source project and are compatible with Azure IoT Edge. Microsoft provides best effort support for systems using those container engines; however, Microsoft can't ship fixes for issues in them. For this reason, Microsoft recommends using moby-engine on production systems.



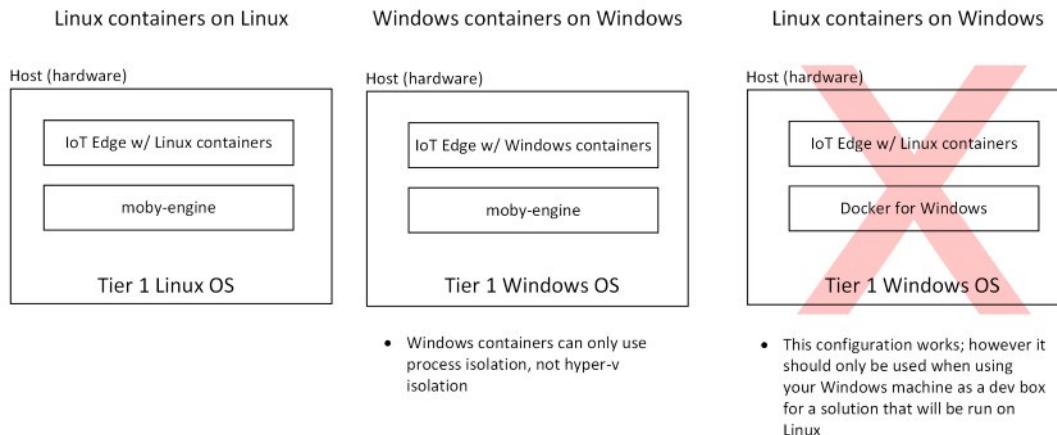
- This configuration works; however it should only be used when using your Windows machine as a dev box for a solution that will be run on Linux

## Operating Systems

Azure IoT Edge runs on most operating systems that can run containers; however, all of these systems are not equally supported. Operating systems are grouped into tiers that represent the level of support users can expect.

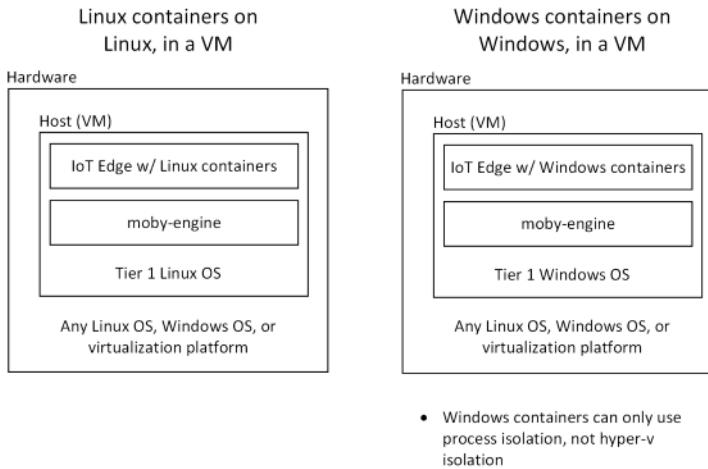
- Tier 1 systems are supported. For tier 1 systems, Microsoft:
  - has this operating system in automated tests
  - provides installation packages for them
- Tier 2 systems are compatible with Azure IoT Edge and can be used relatively easily. For tier 2 systems:
  - Microsoft has done ad hoc testing on the platforms or knows of a partner successfully running Azure IoT Edge on the platform
  - Installation packages for other platforms may work on these platforms

The family of the host OS must always match the family of the guest OS used inside a module's container. In other words, you can only use Linux containers on Linux and Windows containers on Windows. When using Windows, only process isolated containers are supported, not Hyper-V isolated containers.



## Virtual Machines

Azure IoT Edge can be run in virtual machines. Using a virtual machine as an IoT Edge device is common when customers want to augment existing infrastructure with edge intelligence. The family of the host VM OS must match the family of the guest OS used inside a module's container. This requirement is the same as when Azure IoT Edge is run directly on a device. Azure IoT Edge is agnostic of the underlying virtualization technology and works in VMs powered by platforms like Hyper-V and vSphere.



## Minimum System Requirements

Azure IoT Edge runs great on devices as small as a Raspberry Pi3 to server grade hardware. Choosing the right hardware for your scenario depends on the workloads that you want to run. Making the final device decision can be complicated; however, you can easily start prototyping a solution on traditional laptops or desktops.

Experience while prototyping will help guide your final device selection. Questions you should consider include:

- How many modules are in your workload?
- How many layers do your modules' containers share?
- In what language are your modules written?
- How much data will your modules be processing?
- Do your modules need any specialized hardware for accelerating their workloads?
- What are the desired performance characteristics of your solution?
- What is your hardware budget?

## Module Development and Test Tooling

Azure IoT Edge moves your existing business logic to devices operating at the edge. To prepare your applications and workloads to run as IoT Edge modules, you need to build them as containers. The first step is to configure your development environment so that you can successfully create an IoT Edge solution. Once you have your development environment set up, then you can learn how to Develop your own IoT Edge modules.

In any IoT Edge solution, there are at least two machines to consider. One is the IoT Edge device (or devices) itself, which runs the IoT Edge module. The other is the development machine that you use to build, test, and deploy modules. For testing purposes, the two machines can be the same. You can run IoT Edge on your development machine and deploy modules to it.

**Note:** This topic is focused on the tools that you use (on your development machine) to build, test, and deploy modules.

## Operating system

Azure IoT Edge runs on a specific set of supported operating systems. For developing for IoT Edge, you can use most operating systems that can run a container engine. The container engine is a requirement on the development machine to build your modules as containers and push them to a container registry.

If your development machine can't run Azure IoT Edge, see the section below to learn about testing tools that help you test and debug locally.

The operating system of your development machine doesn't have to match the operating system of your IoT Edge device. However, the container operating system must be consistent between development machine and IoT Edge device. For example, you can develop modules on a Windows machine and deploy them to a Linux device. The Windows machine needs to run Linux containers to build the modules for the Linux device.

## Container engine

The central concept of IoT Edge is that you can remotely deploy your business and cloud logic to devices by packaging it into containers. To build containers, you need a container engine on your development machine.

The only supported container engine for IoT Edge devices in production is Moby. However, any container engine compatible with the Open Container Initiative, like Docker, is capable of building IoT Edge module images.

## Development tools

Both Visual Studio and Visual Studio Code have add-on extensions to help develop IoT Edge solutions. These extensions provide language-specific templates to help create and deploy new IoT Edge scenarios. The Azure IoT Edge extensions for Visual Studio and Visual Studio Code help you code, build, deploy, and debug your IoT Edge solutions. You can create an entire IoT Edge solution that contains multiple modules, and the extensions automatically update a deployment manifest template with each new module addition. With the extensions, you can also manage your IoT devices from within Visual Studio or Visual Studio Code. Deploy modules to a device, monitor the status, and view messages as they arrive at IoT Hub. Both extensions use the IoT EdgeHub dev tool to enable local running and debugging of modules on your development machine as well.

If you prefer to develop with other editors or from the CLI, the Azure IoT Edge dev tool provides commands so that you can develop and test from the command line. You can create new IoT Edge scenarios, build module images, run modules in a simulator, and monitor messages sent to IoT Hub.

## Visual Studio Code extension

The Azure IoT Edge extension for Visual Studio Code provides IoT Edge module templates built on programming languages including C, C#, Java, Node.js, and Python as well as Azure functions in C#.

For more information and to download, see [Azure IoT Tools for Visual Studio Code](#).

In addition to the IoT Edge extensions, you may find it helpful to install additional extensions for developing. For example, you can use Docker Support for Visual Studio Code to manage your images, containers, and registries. Additionally, all the major supported languages have extensions for Visual Studio Code that can help when you're developing modules.

## Prerequisites

The module templates for some languages and services have prerequisites that are necessary to build the project folders on your development machine with Visual Studio Code.

Module template	Prerequisite
Azure Functions	.NET Core 2.1 SDK
C	Git
C#	.NET Core 2.1 SDK
Java	Java SE Development Kit 10 Set the JAVA_HOME environment variable Maven
Node.js	Node.js Yeoman Azure IoT Edge Node.js module generator
Python	Python Pip Git

## Visual Studio 2017/2019 extension

The Azure IoT Edge tools for Visual Studio provide an IoT Edge module template built on C# and C.

For more information and to download, see the following:

- Azure IoT Edge Tools for Visual Studio 2017 <https://marketplace.visualstudio.com/items?itemName=vsc-iot.vsiotedge-tools>
- Azure IoT Edge Tools for Visual Studio 2019 <https://marketplace.visualstudio.com/items?itemName=vsc-iot.vs16iotedge-tools>

## IoT Edge dev tool

The Azure IoT Edge dev tool simplifies IoT Edge development with command-line abilities. This tool provides CLI commands to develop, debug, and test modules. The IoT Edge dev tool works with your development system, whether you've manually installed the dependencies on your machine or are using the IoT Edge dev container.

For more information and to get started, see IoT Edge dev tool wiki <https://github.com/Azure/iot-edgedev/wiki>.

## Testing tools

Several testing tools exist to help you simulate IoT Edge devices or debug modules more efficiently. The following table shows a high-level comparison between the tools, and then individual sections describe each tool more specifically.

Only the IoT Edge runtime is supported for production deployments, but the following tools allow you to simulate or easily create IoT Edge devices for development and testing purposes. These tools aren't mutually exclusive, but can work together for a complete development experience.

Tool	Also known as	Supported platforms	Best for
IoT EdgeHub dev tool	iotedgehubdev	Windows, Linux, MacOS	Simulating a device to debug modules.
IoT Edge dev container	microsoft/iotedge-dev	Windows, Linux, MacOS	Developing without installing dependencies.
IoT Edge runtime in a container	iotedgec	Windows, Linux, MacOS, ARM	Testing on a device that may not support the runtime.
IoT Edge device container	toolboc/azure-iot-edge-device-container	Windows, Linux, MacOS, ARM	Testing a scenario with many IoT Edge devices at scale.

## IoT EdgeHub dev tool

The Azure IoT EdgeHub dev tool provides a local development and debug experience. The tool helps start IoT Edge modules without the IoT Edge runtime so that you can create, develop, test, run, and debug IoT Edge modules and solutions locally. You don't have to push images to a container registry and deploy them to a device for testing.

The IoT EdgeHub dev tool was designed to work in tandem with the Visual Studio and Visual Studio Code extensions, as well as with the IoT Edge dev tool. It supports inner loop development as well as outer loop testing, so integrates with the DevOps tools too.

For more information and to install, see Azure IoT EdgeHub dev tool <https://pypi.org/project/iotedgehubdev/>.

## IoT Edge dev container

The Azure IoT Edge dev container is a Docker container that has all the dependencies that you need for IoT Edge development. This container makes it easy to get started with whichever language you want to develop in, including C#, Python, Node.js, and Java. All you need to install is a container engine, like Docker or Moby, to pull the container to your development machine.

For more information, see Azure IoT Edge dev container <https://hub.docker.com/r/microsoft/iot-edge-dev/>.

## IoT Edge runtime in a container

The IoT Edge runtime in a container provides a complete runtime that takes your device connection string as an environment variable. This container enables you to test IoT Edge modules and scenarios on a system that may not support the runtime natively, like MacOS. Any modules that you deploy will be started outside of the runtime container. If you want the runtime and any deployed modules to exist within the same container, consider the IoT Edge device container instead.

For more information, see Running Azure IoT Edge in a container <https://github.com/Azure/iot-edge-dev/tree/master/docker/runtime>.

## IoT Edge device container

The IoT Edge device container is a complete IoT Edge device, ready to be launched on any machine with a container engine. The device container includes the IoT Edge runtime and a container engine itself. Each instance of the container is a fully functional self-provisioning IoT Edge device. The device container

supports remote debugging of modules, as long as there is a network route to the module. The device container is good for quickly creating large numbers of IoT Edge devices to test at-scale scenarios or Azure Pipelines. It also supports deployment to kubernetes via helm.

For more information, see Azure IoT Edge device container <https://github.com/toolboc/azure-iot-edge-device-container>.

## DevOps tools

When you're ready to develop at-scale solutions for extensive production scenarios, take advantage of modern DevOps principles including automation, monitoring, and streamlined software engineering processes. IoT Edge has extensions to support DevOps tools including Azure DevOps, Azure DevOps Projects, and Jenkins. If you want to customize an existing pipeline or use a different DevOps tool like CircleCI or TravisCI, you can do so with the CLI features included in the IoT Edge dev tool.

## Configure the VS Code Dev Environment

Visual Studio Code can be used as the main tool to develop and debug modules, but there are several additional tools required.

There are two ways to debug modules written in C#, Node.js, or Java in Visual Studio Code: You can either attach a process in a module container or launch the module code in debug mode. To debug modules written in Python or C, you can only attach to a process in Linux amd64 containers.

If you aren't familiar with the debugging capabilities of Visual Studio Code, read about Debugging.

## Computer and OS Configuration Requirements

You can use a computer or a virtual machine running Windows, macOS, or Linux as your development machine. On Windows computers you can develop either Windows or Linux modules.

- To develop Windows modules, use a Windows computer running version 1809/build 17763 or newer.
- To develop Linux modules, use a Windows computer that meets the requirements for Docker Desktop.
  - Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later).
  - Hyper-V and Containers Windows features must be enabled. The following hardware prerequisites are required to successfully run Client Hyper-V on Windows 10:
    - 64 bit processor with Second Level Address Translation (SLAT)
    - 4GB system RAM
    - BIOS-level hardware virtualization support must be enabled in the BIOS settings.

## Visual Studio Code Configuration Requirements

Install Visual Studio Code first and then add the following extensions:

- Azure IoT Tools
- Docker extension
- Visual Studio extension(s) specific to the language you're developing in:
  - C#, including Azure Functions: C# extension
  - Python: Python extension

- Java: Java Extension Pack for Visual Studio Code
- C: C/C++ extension

## Language-Specific Tools

You'll also need to install some additional, language-specific tools to develop your module:

- C#, including Azure Functions: .NET Core 2.1 SDK
- Python: Python and Pip for installing Python packages (typically included with your Python installation).
- Node.js: Node.js. You'll also want to install Yeoman and the Azure IoT Edge Node.js Module Generator.
- Java: Java SE Development Kit 10 and Maven. You'll need to set the JAVA\_HOME environment variable to point to your JDK installation.

## Container Tools

To build and deploy your module image, you need Docker to build the module image and a container registry to hold the module image:

- Docker Community Edition on your development machine.
- Azure Container Registry or Docker Hub

Tip: You can use a local Docker registry for prototype and testing purposes instead of a cloud registry.

## Debug Tools

Unless you're developing your module in C, you also need the Python-based Azure IoT EdgeHub Dev Tool in order to set up your local development environment to debug, run, and test your IoT Edge solution. If you haven't already done so, install Python (2.7/3.6+) and Pip and then install iotedgehubdev by running this command in your terminal.

```
pip install --upgrade iotedgehubdev
```

**Note:** If you have multiple Python including pre-installed python 2.7 (for example, on Ubuntu or macOS), make sure you are using the correct `pip` or `pip3` to install `iotedgehubdev`

To test your module on a device, you'll need an active IoT hub with at least one IoT Edge device. To use your computer as an IoT Edge device, follow the steps in the quickstart:

- for Linux - Quickstart: Deploy your first IoT Edge module to a virtual Linux device <https://docs.microsoft.com/en-us/azure/iot-edge/quickstart-linux>
- for Windows - Quickstart: Deploy your first IoT Edge module to a virtual Windows device <https://docs.microsoft.com/en-us/azure/iot-edge/quickstart>.

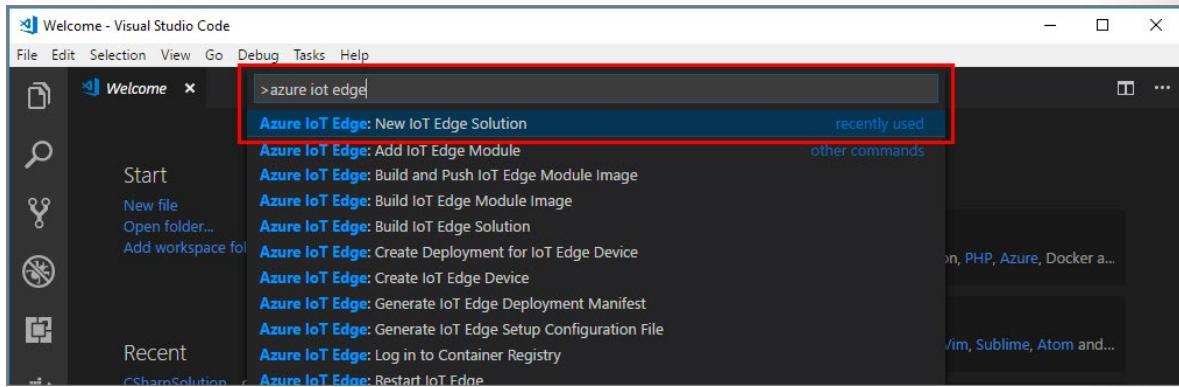
If you are running IoT Edge daemon on your development machine, you might need to stop EdgeHub and EdgeAgent before you move to next step.

## Develop Custom Modules with VS Code

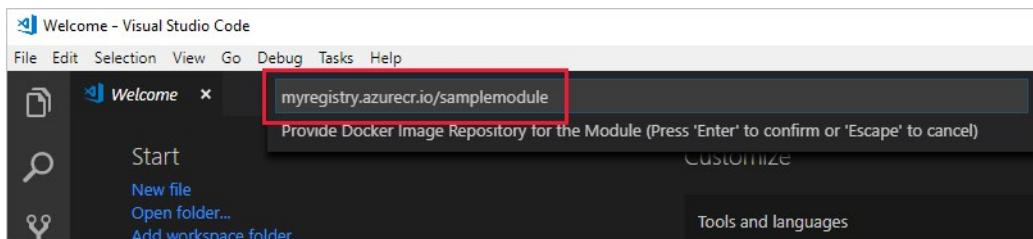
Visual Studio Code and the Azure IoT Tools can be used to create an IoT Edge module in your preferred development language (including Azure Functions, written in C#). You start by creating a solution, and then generating the first module in that solution. Each solution can contain multiple modules.

## Create a new solution template

1. Open VS Code
2. Select View > Command Palette.
3. In the command palette, enter and run the command Azure IoT Edge: New IoT Edge Solution.



4. Browse to the folder where you want to create the new solution and then select Select folder.
5. Enter a name for your solution.
6. Select a module template for your preferred development language to be the first module in the solution.
7. Enter a name for your module. Choose a name that's unique within your container registry.
8. Provide the name of the module's image repository. Visual Studio Code autopopulates the module name with localhost:5000/<your module name>. Replace it with your own registry information. If you use a local Docker registry for testing, then localhost is fine. If you use Azure Container Registry, then use the login server from your registry's settings. The login server looks like <registry name>.azurecr.io. Only replace the localhost:5000 part of the string so that the final result looks like <registry name>.azurecr.io/<your module name>.



Visual Studio Code takes the information you provided, creates an IoT Edge solution, and then loads it in a new window.

There are four items within the solution:

- A .vscode folder contains debug configurations.
- A modules folder has subfolders for each module. Within the folder for each module there is a file, module.json, that controls how modules are built and deployed. This file would need to be modified to change the module deployment container registry from localhost to a remote registry. At this point, you only have one module. But you can add more in the command palette with the command Azure IoT Edge: Add IoT Edge Module.

- An .env file lists your environment variables. If Azure Container Registry is your registry, you'll have an Azure Container Registry username and password in it.
  - **Note:** The environment file is only created if you provide an image repository for the module. If you accepted the localhost defaults to test and debug locally, then you don't need to declare environment variables.
- A deployment.template.json file lists your new module along with a sample SimulatedTemperatureSensor module that simulates data you can use for testing.

## Add additional modules

To add additional modules to your solution, run the command Azure IoT Edge: Add IoT Edge Module from the command palette. You can also right-click the modules folder or the deployment.template.json file in the Visual Studio Code Explorer view and then select Add IoT Edge Module.

## Develop your module

The default module code that comes with the solution is located at the following location:

- Azure Function (C#): modules > <your module name> > <your module name>.cs
- C#: modules > <your module name> > Program.cs
- Python: modules > <your module name> > main.py
- Node.js: modules > <your module name> > app.js
- Java: modules > <your module name> > src > main > java > com > edgemodulemodules > App.java
- C: modules > <your module name> > main.c

The module and the deployment.template.json file are set up so that you can build the solution, push it to your container registry, and deploy it to a device to start testing without touching any code. The module is built to simply take input from a source (in this case, the SimulatedTemperatureSensor module that simulates data) and pipe it to IoT Hub.

When you're ready to customize the template with your own code, use the Azure IoT Hub SDKs to build modules that address the key needs for IoT solutions such as security, device management, and reliability.

## Debug Edge Modules

### Debug a module without a container (C#)

If you're developing in C#, Node.js, or Java, your module requires use of a `ModuleClient` object in the default module code so that it can start, run, and route messages. You'll also use the default input channel `input1` to take action when the module receives messages.

### Set up IoT Edge simulator for IoT Edge solution

On your development machine, you can start an IoT Edge simulator instead of installing the IoT Edge security daemon so that you can run your IoT Edge solution.

1. In device explorer on the left side, right-click on your IoT Edge device ID, and then select Setup IoT Edge Simulator to start the simulator with the device connection string.

2. You can see the IoT Edge Simulator has been successfully set up by reading the progress detail in the integrated terminal.

## Set up IoT Edge simulator for single module app

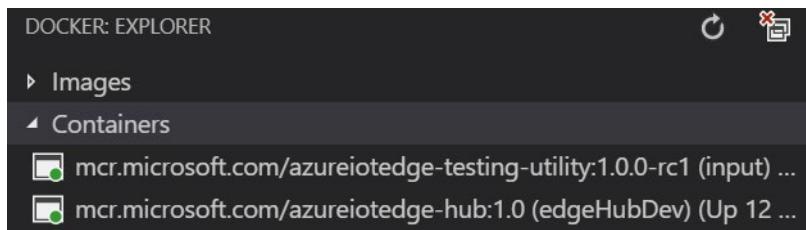
To set up and start the simulator, run the command Azure IoT Edge: Start IoT Edge Hub Simulator for Single Module from the Visual Studio Code command palette. When prompted, use the value `input1` from the default module code (or the equivalent value from your code) as the input name for your application. The command triggers the `iotedgehubdev` CLI and then starts the IoT Edge simulator and a testing utility module container. You can see the outputs below in the integrated terminal if the simulator has been started in single module mode successfully. You can also see a curl command to help send message through. You will use it later.

```
D:\Workspaces\Edge-GA\Docs\EdgeSolution>iotedgehubdev start -i "input1"
Network azure-iot-edge-dev is external, skipping
IoT Edge Simulator has been started in single module mode.
Please run `iotedgehubdev modulecred` to get credential to connect your module.
And send message through:

curl --header "Content-Type: application/json" --request POST --data '{"inputName": "input1","data":"hello world"}' http://
/localhost:53000/api/v1/messages

Please refer to https://github.com/Azure/iot-edge-testing-utility/blob/master/swagger.json for detail schema
```

You can use the Docker Explorer view in Visual Studio Code to see the module's running status.



The `edgeHubDev` container is the core of the local IoT Edge simulator. It can run on your development machine without the IoT Edge security daemon and provides environment settings for your native module app or module containers. The input container exposes REST APIs to help bridge messages to the target input channel on your module.

## Debug module in launch mode

1. Prepare your environment for debugging according to the requirements of your development language (C# in this case), set a breakpoint in your module, and select the debug configuration to use:
    - In the Visual Studio Code integrated terminal, change the directory to the `<your module name>` folder, and then run the following command to build .NET Core application.  
`dotnet build`
    - Open the file `Program.cs` and add a breakpoint.
    - Navigate to the Visual Studio Code Debug view by selecting **View > Debug**. Select the debug configuration `<your module name> Local Debug (.NET Core)` from the dropdown.
- Note:** If your .NET Core `TargetFramework` is not consistent with your program path in `launch.json`, you'll need to manually update the program path in `launch.json` to match the `TargetFramework` in your `.csproj` file so that Visual Studio Code can successfully launch this program.

2. Click **Start Debugging** or press **F5** to start the debug session.
3. In the Visual Studio Code integrated terminal, run the following command to send a Hello World message to your module. This is the command shown in previous steps when you set up IoT Edge simulator.

```
curl --header "Content-Type: application/json" --request POST --data '{"inputName": "input1", "data": "Hello world"}' http://localhost:53000/api/v1/messages
```

**Note:** If you are using Windows, making sure the shell of your Visual Studio Code integrated terminal is **Git Bash** or **WSL Bash**. You cannot run the `curl` command from a PowerShell or command prompt.

**Tip:** You can also use **PostMan** or other API tools to send messages through instead of `curl`.

1. In the Visual Studio Code Debug view, you'll see the variables in the left panel.
2. To stop your debugging session, select the **Stop** button or press **Shift + F5**, and then run **Azure IoT Edge: Stop IoT Edge Simulator** in the command palette to stop the simulator and clean up.

## Debug in attach mode with IoT Edge Simulator (C#)

Your default solution contains two modules, one is a simulated temperature sensor module and the other is the pipe module. The simulated temperature sensor sends messages to the pipe module and then the messages are piped to the IoT Hub. In the module folder you created, there are several Docker files for different container types. Use any of the files that end with the extension `.debug` to build your module for testing.

Currently, debugging in attach mode is supported only as follows:

- C# modules, including those for Azure Functions, support debugging in Linux amd64 containers
- Node.js modules support debugging in Linux amd64 and arm32v7 containers, and Windows amd64 containers
- Java modules support debugging in Linux amd64 and arm32v7 containers

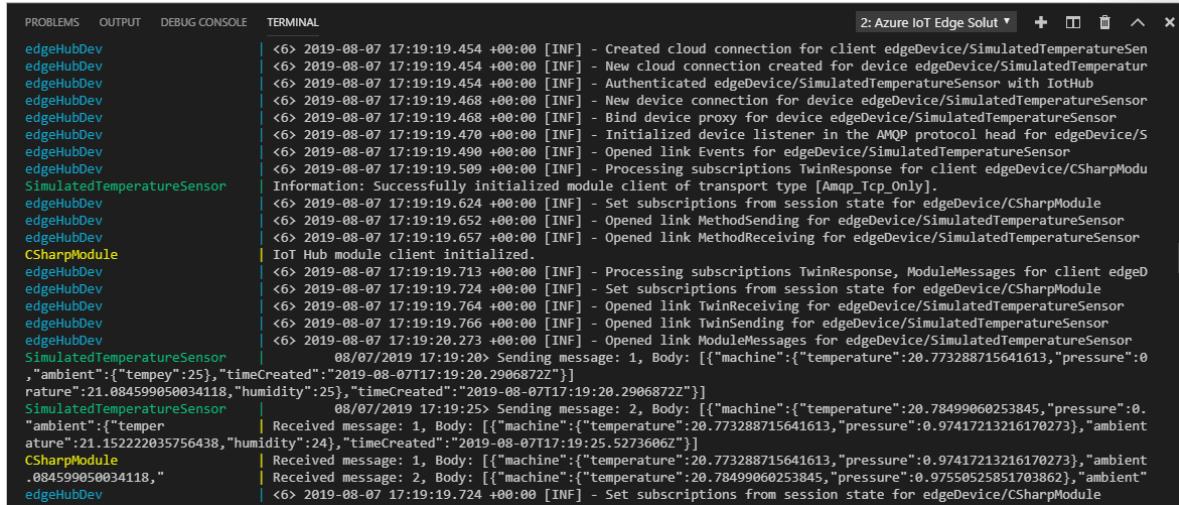
## Set up IoT Edge simulator for IoT Edge solution

In your development machine, you can start an IoT Edge simulator instead of installing the IoT Edge security daemon so that you can run your IoT Edge solution.

1. In device explorer on the left side, right-click on your IoT Edge device ID, and then select **Setup IoT Edge Simulator** to start the simulator with the device connection string.
2. You can see the IoT Edge Simulator has been successfully set up by reading the progress detail in the integrated terminal.

## Build and run container for debugging and debug in attach mode

1. Open your module file (`Program.cs`, `app.js`, `App.java`, or `<your module name>.cs`) and add a break-point.
2. In the Visual Studio Code Explorer view, right-click the `deployment.debug.template.json` file for your solution and then select **Build and Run IoT Edge solution in Simulator**. You can watch all the module container logs in the same window. You can also navigate to the Docker view to watch container status.



3. Navigate to the Visual Studio Code Debug view and select the debug configuration file for your module. The debug option name should be similar to **<your module name> Remote Debug**
  4. Select **Start Debugging** or press **F5**. Select the process to attach to.
  5. In Visual Studio Code Debug view, you'll see the variables in the left panel.
  6. To stop the debugging session, first select the Stop button or press **Shift + F5**, and then select **Azure IoT Edge: Stop IoT Edge Simulator** from the command palette.

# Debug a module with the IoT Edge runtime

In each module folder, there are several Docker files for different container types. Use any of the files that end with the extension **.debug** to build your module for testing.

When debugging modules using this method, your modules are running on top of the IoT Edge runtime. The IoT Edge device and your Visual Studio Code can be on the same machine, or more typically, Visual Studio Code is on the development machine and the IoT Edge runtime and modules are running on another physical machine. In order to debug from Visual Studio Code, you must:

- Set up your IoT Edge device, build your IoT Edge module(s) with the **.debug** Dockerfile, and then deploy to the IoT Edge device.
  - Expose the IP and port of the module so that the debugger can be attached.
  - Update the `launch.json` so that Visual Studio Code can attach to the process in the container on the remote machine. This file is located in the `.vscode` folder in your workspace and updates each time you add a new module that supports debugging.

## Build and deploy your module to the IoT Edge device

1. In Visual Studio Code, open the `deployment.debug.template.json` file, which contains the debug version of your module images with the proper `createOptions` values set.
  2. In the Visual Studio Code command palette:
    - Run the command **Azure IoT Edge: Build and Push IoT Edge solution**.
    - Select the `deployment.debug.template.json` file for your solution.

- In the **Azure IoT Hub Devices** section of the Visual Studio Code Explorer view:

- Right-click an IoT Edge device ID and then select **Create Deployment for Single Device**.

**Tip:** To confirm that the device you've chosen is an IoT Edge device, select it to expand the list of modules and verify the presence of \$edgeHub and \$edgeAgent. Every IoT Edge device includes these two modules.

- Navigate to your solution's **config** folder, select the `deployment.debug.amd64.json` file, and then select **Select Edge Deployment Manifest**.

You'll see the deployment successfully created with a deployment ID in the integrated terminal.

You can check your container status by running the `docker ps` command in the terminal. If your Visual Studio Code and IoT Edge runtime are running on the same machine, you can also check the status in the Visual Studio Code Docker view.

## Expose the IP and port of the module for the debugger

You can skip this section if your modules are running on the same machine as Visual Studio Code, as you are using `localhost` to attach to the container and already have the correct port settings in the `.debug` Dockerfile, module's container `createOptions` settings, and `launch.json` file.

**Note:** If your modules and Visual Studio Code are running on separate machines, follow the instructions (for C#, including Azure Functions) below:

- Complete the steps provided here: Configure the SSH channel on your development machine and IoT Edge device <https://github.com/OmniSharp/omnisharp-vscode/wiki/Attaching-to-remote-processes>
- And then edit the `launch.json` file to attach.

## Debug your module

- In the Visual Studio Code Debug view, select the debug configuration file for your module. The debug option name should be similar to **<your module name> Remote Debug**
- Open the module file for your development language and add a breakpoint:
  - Azure Function (C#): Add your breakpoint to the file `<your module name>.cs`.
  - C#: Add your breakpoint to the file `Program.cs`.
- Select **Start Debugging** or press **F5**. Select the process to attach to.
- In the Visual Studio Code Debug view, you'll see the variables in the left panel.

**Note:** The preceding example shows how to debug IoT Edge modules on containers. It added exposed ports to your module's container `createOptions` settings. After you finish debugging your modules, we recommend you remove these exposed ports for production-ready IoT Edge modules.

## Build and debug a module remotely

With recent changes in both the Docker and Moby engines to support SSH connections, and a new setting in Azure IoT Tools that enables injection of environment settings into the Visual Studio Code command palette and Azure IoT Edge terminals, you can now build and debug modules on remote devices.

# Offline and Local Storage

## Extended Offline Capabilities

Azure IoT Edge supports extended offline operations on your IoT Edge devices, and enables offline operations on non-IoT Edge child devices too. As long as an IoT Edge device has had one opportunity to connect to IoT Hub, it and any child devices can continue to function with intermittent or no Internet connection.

### How it works

When an IoT Edge device goes into offline mode, the IoT Edge hub takes on three roles. First, it stores any messages that would go upstream and saves them until the device reconnects. Second, it acts on behalf of IoT Hub to authenticate modules and child devices so that they can continue to operate. Third, it enables communication between child devices that normally would go through IoT Hub.

The following example shows how an IoT Edge scenario operates in offline mode:

1. Configure devices

IoT Edge devices automatically have offline capabilities enabled. To extend that capability to other IoT devices, you need to declare a parent-child relationship between the devices in IoT Hub. Then, you configure the child devices to trust their assigned parent device and route the device-to-cloud communications through the parent as a gateway.

1. Sync with IoT Hub

At least once after installation of the IoT Edge runtime, the IoT Edge device needs to be online to sync with IoT Hub. In this sync, the IoT Edge device gets details about any child devices assigned to it. The IoT Edge device also securely updates its local cache to enable offline operations and retrieves settings for local storage of telemetry messages.

1. Go offline

While disconnected from IoT Hub, the IoT Edge device, its deployed modules, and any children IoT devices can operate indefinitely. Modules and child devices can start and restart by authenticating with the IoT Edge hub while offline. Telemetry bound upstream to IoT Hub is stored locally. Communication between modules or between child IoT devices is maintained through direct methods or messages.

1. Reconnect and resync with IoT Hub

Once the connection with IoT Hub is restored, the IoT Edge device syncs again. Locally stored messages are delivered in the same order in which they were stored. Any differences between the desired and reported properties of the modules and devices are reconciled. The IoT Edge device updates any changes to its set of assigned child IoT devices.

### Restrictions and limits

The extended offline capabilities described here are available in IoT Edge version 1.0.7 or higher. Earlier versions have a subset of offline features. Existing IoT Edge devices that don't have extended offline capabilities can't be upgraded by changing the runtime version, but must be reconfigured with a new IoT Edge device identity to gain these features.

Extended offline support is available in all regions where IoT Hub is available, except East US.

Only non-IoT Edge devices can be added as child devices.

IoT Edge devices and their assigned child devices can function indefinitely offline after the initial, one-time sync. However, storage of messages depends on the time to live (TTL) setting and the available disk space for storing the messages.

## Set up parent and child devices

For an IoT Edge device to extend its extended offline capabilities to child IoT devices, you need to complete two steps. First, declare the parent-child relationships in the Azure portal. Second, create a trust relationship between the parent device and any child devices, then configure device-to-cloud communications to go through the parent as a gateway.

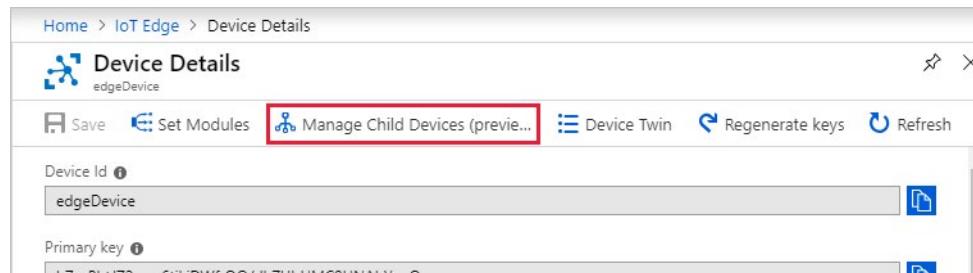
## Assign child devices

Child devices can be any non-IoT Edge device registered to the same IoT Hub. Parent devices can have multiple child devices, but a child device only has one parent. There are three options to set child devices to an edge device: through the Azure portal, using the Azure CLI, or using the IoT Hub service SDK.

The following sections provide examples of how you can declare the parent/child relationship in IoT Hub for existing IoT devices. If you're creating new device identities for your child devices, see Authenticate a downstream device to Azure IoT Hub for more information.

## Option 1: IoT Hub Portal

You can declare the parent-child relationship when creating a new device. Or for existing devices, you can declare the relationship from the device details page of either the parent IoT Edge device or the child IoT device.

A screenshot of the Azure IoT Hub Device Details page. The top navigation bar shows 'Home > IoT Edge > Device Details'. The main title is 'Device Details' with a sub-label 'edgeDevice'. Below the title are several buttons: 'Save', 'Set Modules', 'Manage Child Devices (preview...)', 'Device Twin', 'Regenerate keys', and 'Refresh'. The 'Manage Child Devices (preview...)' button is highlighted with a red box. Underneath these buttons, there are two input fields: 'Device Id' containing 'edgeDevice' and 'Primary key' containing a long hexidecimal string. To the right of each input field is a small blue download icon.

## Option 2: Use the az command-line tool

Using the Azure command-line interface with IoT extension (v0.7.0 or newer), you can manage parent child relationships with the device-identity subcommands. The example below uses a query to assign all non-IoT Edge devices in the hub to be child devices of an IoT Edge device.

```
# Set IoT Edge parent device
edge_device="edge-device1"

# Get All IoT Devices
device_list=$(az iot hub query \
    --hub-name replace-with-hub-name \
    --subscription replace-with-sub-name \
    --resource-group replace-with-rg-name \
    -q "SELECT * FROM devices WHERE capabilities.iotEdge = false" \
    --query 'join(` `, [].deviceId)' -o tsv)
```

```
# Add all IoT devices to IoT Edge (as child)
az iot hub device-identity add-children \
--device-id $edge_device \
--child-list $device_list \
--hub-name replace-with-hub-name \
--resource-group replace-with-rg-name \
--subscription replace-with-sub-name
```

You can modify the query to select a different subset of devices. The command may take several seconds if you specify a large set of devices.

## Option 3: Use IoT Hub Service SDK

Finally, you can manage parent child relationships programmatically using either C#, Java or Node.js IoT Hub Service SDK.

## Set up the parent device as a gateway

You can think of a parent/child relationship as a transparent gateway, where the child device has its own identity in IoT Hub but communicates through the cloud via its parent. For secure communication, the child device needs to be able to verify that the parent device comes from a trusted source. Otherwise, third-parties could set up malicious devices to impersonate parents and intercept communications.

## Specify DNS servers

To improve robustness, it is highly recommended you specify the DNS server addresses used in your environment. To set your DNS server for IoT Edge, either set DNS server address in container engine settings or per module in the IoT Edge deployment. For additional details, see the resolution for **Edge Agent module continually reports 'empty config file' and no modules start on device** in the troubleshooting article <https://docs.microsoft.com/en-us/azure/iot-edge/troubleshoot#edge-agent-module-continually-reports-empty-config-file-and-no-modules-start-on-the-device>.

## Optional offline settings

If your devices go offline, the IoT Edge parent device stores all device-to-cloud messages until the connection is reestablished. The IoT Edge hub module manages the storage and forwarding of offline messages. For devices that may go offline for extended periods of time, optimize performance by configuring two IoT Edge hub settings.

First, increase the time to live setting so that the IoT Edge hub will keep messages long enough for your device to reconnect. Then, add additional disk space for message storage.

## Time to live

The time to live setting is the amount of time (in seconds) that a message can wait to be delivered before it expires. The default is 7200 seconds (two hours). The maximum value is only limited by the maximum value of an integer variable, which is around 2 billion.

This setting is a desired property of the IoT Edge hub, which is stored in the module twin. You can configure it in the Azure portal or directly in the deployment manifest.

```
 "$edgeHub": {  
   "properties.desired": {  
     "schemaVersion": "1.0",  
     "routes": {},  
     "storeAndForwardConfiguration": {  
       "timeToLiveSecs": 7200  
     }  
   }  
 }
```

## Host storage for system modules

Messages and module state information are stored in the IoT Edge hub's local container filesystem by default. For improved reliability, especially when operating offline, you can also dedicate storage on the host IoT Edge device.

# IoT Edge Storage

## Azure Blob Storage on IoT Edge

Azure Blob Storage on IoT Edge provides a block blob storage solution at the edge. A blob storage module on your IoT Edge device behaves like an Azure block blob service, except the block blobs are stored locally on your IoT Edge device. You can access your blobs using the same Azure storage SDK methods or block blob API calls that you're already used to.

A blob storage module is useful in the following scenarios:

- where data needs to be stored locally until it can be processed or transferred to the cloud. This data can be videos, images, finance data, hospital data, or any other unstructured data.
- when devices are located in a place with limited connectivity.
- when you want to efficiently process the data locally to get low latency access to the data, such that you can respond to emergencies as quickly as possible.
- when you want to reduce bandwidth costs and avoid transferring terabytes of data to the cloud. You can process the data locally and send only the processed data to the cloud.

Blob storage modules include **deviceToCloudUpload** and **deviceAutoDelete** features.

**deviceToCloudUpload** is a configurable functionality. This function automatically uploads the data from your local blob storage to Azure with intermittent internet connectivity support. It allows you to:

- Turn ON/OFF the deviceToCloudUpload feature.
- Choose the order in which the data is copied to Azure like NewestFirst or OldestFirst.
- Specify the Azure Storage account to which you want your data uploaded.
- Specify the containers you want to upload to Azure. This module allows you to specify both source and target container names.

[!NOTE] Remember that in this context, "container" refers to the top-level blob storage organization system, and not compute containers.

- Choose the ability to delete the blobs immediately, after upload to cloud storage is finished
- Do full blob upload (using Put Blob operation) and block level upload (using Put Block and Put Block List operations).

This module uses block level upload, when your blob consists of blocks. Here are some of the common scenarios:

- Your application updates some blocks of a previously uploaded blob, this module uploads only the updated blocks and not the whole blob.
- The module is uploading blob and internet connection goes away, when the connectivity is back again it uploads only the remaining blocks and not the whole blob.

If an unexpected process termination (like power failure) happens during a blob upload, all blocks that were due for the upload will be uploaded again once the module comes back online.

**deviceAutoDelete** is a configurable functionality. This function automatically deletes your blobs from the local storage when the specified duration (measured in minutes) expires. It allows you to:

- Turn ON/OFF the deviceAutoDelete feature.
- Specify the time in minutes (deleteAfterMinutes) after which the blobs will be automatically deleted.

- Choose the ability to retain the blob while it's uploading if the deleteAfterMinutes value expires.

## deviceToCloudUpload and deviceAutoDelete properties

Use the module's desired properties to set deviceToCloudUploadProperties and deviceAutoDeleteProperties. Desired properties can be set during deployment or changed later by editing the module twin without the need to redeploy. We recommend checking the "Module Twin" for reported configuration and configurationValidation to make sure values are correctly propagated.

### deviceToCloudUploadProperties

The name of this setting is deviceToCloudUploadProperties. If you are using the IoT Edge simulator, set the values to the related environment variables for these properties, which you can find in the explanation section.

Property	Possible Values	Explanation
uploadOn	true, false	<p>Set to false by default. If you want to turn the feature on, set this field to true.</p> <p>Environment variable: deviceToCloudUploadProperties_uploadOn={false,true}</p>
uploadOrder	NewestFirst, OldestFirst	<p>Allows you to choose the order in which the data is copied to Azure. Set to OldestFirst by default. The order is determined by last modified time of Blob.</p> <p>Environment variable: deviceToCloudUploadProperties_uploadOrder={NewestFirst,OldestFirst}</p>

Property	Possible Values	Explanation
cloudStorageConnectionString		"DefaultEndpointsProtocol=https;AccountName=<your Azure Storage Account Name>;AccountKey=<your Azure Storage Account Key>;EndpointSuffix=<your endpoint suffix>" is a connection string that allows you to specify the storage account to which you want your data uploaded. Specify Azure Storage Account Name, Azure Storage Account Key, End point suffix. Add appropriate EndpointSuffix of Azure where data will be uploaded, it varies for Global Azure, Government Azure, and Microsoft Azure Stack.  You can choose to specify Azure Storage SAS connection string here. But you have to update this property when it expires.  Environment variable: deviceToCloudUploadProperties__cloudStorageConnectionString=<connection string>

Property	Possible Values	Explanation
storageContainersForUpload	<p>"&lt;source container name1&gt;": {"target": "&lt;target container name&gt;"},</p> <p>"&lt;source container name1&gt;": {"target": "%h-%d-%m-%c"},</p> <p>"&lt;source container name1&gt;": {"target": "%d-%c"}</p>	<p>Allows you to specify the container names you want to upload to Azure. This module allows you to specify both source and target container names. If you don't specify the target container name, it will automatically assign the container name as &lt;IoTHubName&gt;-&lt;iotEdgeDeviceID&gt;-&lt;ModuleName&gt;-&lt;SourceContainerName&gt;. You can create template strings for target container name, check out the possible values column.</p> <ul style="list-style-type: none"> <li>* %h -&gt; IoT Hub Name (3-50 characters).</li> <li>* %d -&gt; IoT Edge Device ID (1 to 129 characters).</li> <li>* %m -&gt; Module Name (1 to 64 characters).</li> <li>* %c -&gt; Source Container Name (3 to 63 characters).</li> </ul> <p>Maximum size of the container name is 63 characters, while automatically assigning the target container name if the size of container exceeds 63 characters it will trim each section (IoTHubName, iotEdgeDeviceID, ModuleName, SourceContainerName) to 15 characters.</p> <p>Environment variable: deviceToCloudUploadProperties_storageContainersForUpload_&lt;sourceName&gt;_target=&lt;targetName&gt;</p>
deleteAfterUpload	true, false	<p>Set to false by default. When it is set to true, it will automatically delete the data when upload to cloud storage is finished.</p> <p>Environment variable: deviceToCloudUploadProperties_deleteAfterUpload={false,true}</p>

## deviceAutoDeleteProperties

The name of this setting is deviceAutoDeleteProperties. If you are using the IoT Edge simulator, set the values to the related environment variables for these properties, which you can find in the explanation section.

Property	Possible Values	Explanation
deleteOn	true, false	<p>Set to false by default. If you want to turn the feature on, set this field to true.</p> <p>Environment variable: deviceAutoDeleteProperties__deleteOn={false,true}</p>
deleteAfterMinutes	<minutes>	<p>Specify the time in minutes. The module will automatically delete your blobs from local storage when this value expires.</p> <p>Environment variable: deviceAutoDeleteProperties__ deleteAfterMinutes=&lt;minutes&gt;</p>
retainWhileUploading	true, false	<p>By default it is set to true, and it will retain the blob while it is uploading to cloud storage if deleteAfterMinutes expires. You can set it to false and it will delete the data as soon as deleteAfterMinutes expires.</p> <p>Note: For this property to work uploadOn should be set to true.</p> <p>Environment variable: deviceAutoDeleteProperties__retainWhileUploading={false,true}</p>

## Connect to your blob storage module

You can use the account name and account key that you configured for your module to access the blob storage on your IoT Edge device.

Specify your IoT Edge device as the blob endpoint for any storage requests that you make to it. You can Create a connection string for an explicit storage endpoint using the IoT Edge device information and the account name that you configured.

- For modules that are deployed on the same device as where the Azure Blob Storage on IoT Edge module is running, the blob endpoint is: `http://<module name>:11002/<account name>`.
- For modules or applications running on a different device, you have to choose the right endpoint for your network. Depending on your network setup, choose an endpoint format such that the data traffic

from your external module or application can reach the device running the Azure Blob Storage on IoT Edge module. The blob endpoint for this scenario is one of:

- `http://<device IP >:11002/<account name>`
- `http://<IoT Edge device hostname>:11002/<account name>`
- `http://<fully qualified domain name>:11002/<account name>`

## Connect to your local storage with Azure Storage Explorer

You can use Azure Storage Explorer to connect to your local storage account.

1. Download and install Azure Storage Explorer
2. Connect to Azure Storage using a connection string
3. Provide connection string:

```
DefaultEndpointsProtocol=http;BlobEndpoint=http://<host device  
name>:11002/<your local account name>;AccountName=<your local account  
name>;AccountKey=<your local account key>;
```

1. Go through the steps to connect.
2. Create container inside your local storage account
3. Start uploading files as Block blobs.

**Note:** This module does not support Page blobs.

1. You can choose to connect your Azure storage accounts in Storage Explorer, too. This configuration gives you a single view for both your local storage account and Azure storage account

## Supported storage operations

Blob storage modules on IoT Edge use the Azure Storage SDKs, and are consistent with the 2017-04-17 version of the Azure Storage API for block blob endpoints.

Because not all Azure Blob Storage operations are supported by Azure Blob Storage on IoT Edge, this section lists the status of each.

### Account

Supported:

- List containers

Unsupported:

- Get and set blob service properties
- Preflight blob request
- Get blob service stats
- Get account information

## Containers

Supported:

- Create and delete container
- Get container properties and metadata
- List blobs
- Get and set container ACL
- Set container metadata

Unsupported:

- Lease container

## Blobs

Supported:

- Put, get, and delete blob
- Get and set blob properties
- Get and set blob metadata

Unsupported:

- Lease blob
- Snapshot blob
- Copy and abort copy blob
- Undelete blob
- Set blob tier

## Block blobs

Supported:

- Put block
- Put and get block list

Unsupported:

- Put block from URL

## Module Access to Local Storage

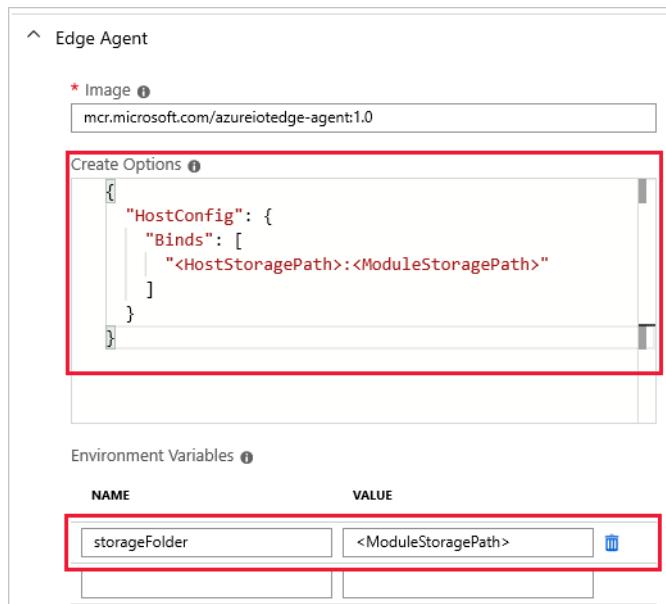
In addition to storing data using Azure storage services or in your device's container storage, you can also dedicate storage on the host IoT Edge device itself for improved reliability, especially when operating offline.

## Link module storage to device storage

To enable a link from module storage to the storage on the host system, create an environment variable for your module that points to a storage folder in the container. Then, use the create options to bind that storage folder to a folder on the host machine.

For example, if you wanted to enable the IoT Edge hub to store messages in your device's local storage and retrieve them later, you can configure the environment variables and the create options in the Azure portal in the **Configure advanced Edge Runtime settings** section.

1. For both IoT Edge hub and IoT Edge agent, add an environment variable called **storageFolder** that points to a directory in the module.
2. For both IoT Edge hub and IoT Edge agent, add binds to connect a local directory on the host machine to a directory in the module. For example:



Or, you can configure the local storage directly in the deployment manifest. For example:

```
"systemModules": {
  "edgeAgent": {
    "settings": {
      "image": "mcr.microsoft.com/azureiotedge-agent:1.0",
      "createOptions": {
        "HostConfig": {
          "Binds": ["<HostStoragePath>:<ModuleStoragePath>"]
        }
      },
      "type": "docker",
      "env": {
        "storageFolder": {
          "value": "<ModuleStoragePath>"
        }
      }
    }
  }
}
```

```
"edgeHub": {  
    "settings": {  
        "image": "mcr.microsoft.com/azureiotedge-hub:1.0",  
        "createOptions": {  
            "HostConfig": {  
                "Binds": ["<HostStoragePath>:<ModuleStoragePath>"],  
                "PortBindings": {"5671/tcp": [{"HostPort": "5671"}], "8883/tcp": [{"HostPort": "8883"}]}, "443/tcp": [{"HostPort": "443"}]}  
            },  
            "type": "docker",  
            "env": {  
                "storageFolder": {  
                    "value": "<ModuleStoragePath>"  
                }  
            },  
            "status": "running",  
            "restartPolicy": "always"  
        }  
    }  
}
```

Replace `<HostStoragePath>` and `<ModuleStoragePath>` with your host and module storage path; both values must be an absolute path.

For example, on a Linux system, `"Binds": ["/etc/iotedge/storage/:/iotedge/storage/"]` means the directory **/etc/iotedge/storage** on your host system is mapped to the directory **/iotedge/storage/** in the container. On a Windows system, as another example, `"Binds": ["C:\\temp:C:\\\\contemp"]` means the directory **C:\temp** on your host system is mapped to the directory **C:\contemp** in the container.

Additionally, on Linux devices, make sure that the user profile for your module has the required read, write, and execute permissions to the host system directory. Returning to the earlier example of enabling IoT Edge hub to store messages in your device's local storage, you need to grant permissions to its user profile, UID 1000. (The IoT Edge agent operates as root, so it doesn't need additional permissions.) There are several ways to manage directory permissions on Linux systems, including using `chown` to change the directory owner and then `chmod` to change the permissions, such as:

```
sudo chown 1000 <HostStoragePath>  
sudo chmod 700 <HostStoragePath>
```

You can find more details about create options from docker docs <https://docs.docker.com/engine/api/v1.32/#operation/ContainerCreate>.

## About the Module 7 Labs

### Module Labs

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

- Lab 13: Create and Deploy a Custom Edge Module
- Lab 14: Implement Restricted Network and Offline Scenarios for IoT Edge

**WARNING:** Be prepared for UI changes

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

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

## Module 8 Device Management

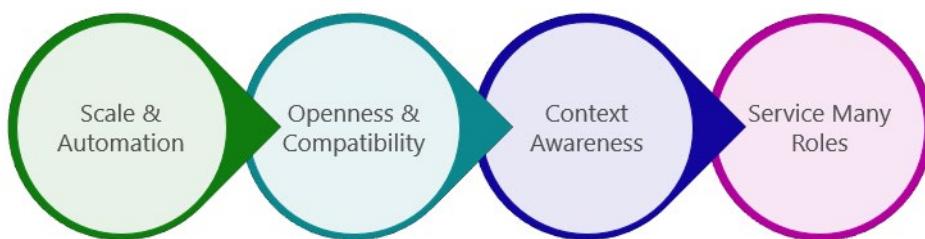
### Introduction to IoT Device Management

#### What is Device Management?

Azure IoT Hub provides the features and an extensibility model that enable device and back-end developers to build robust device management solutions. Devices range from constrained sensors and single purpose microcontrollers, to powerful gateways that route communications for groups of devices. In addition, the use cases and requirements for IoT operators vary significantly across industries. Despite this variation, 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:



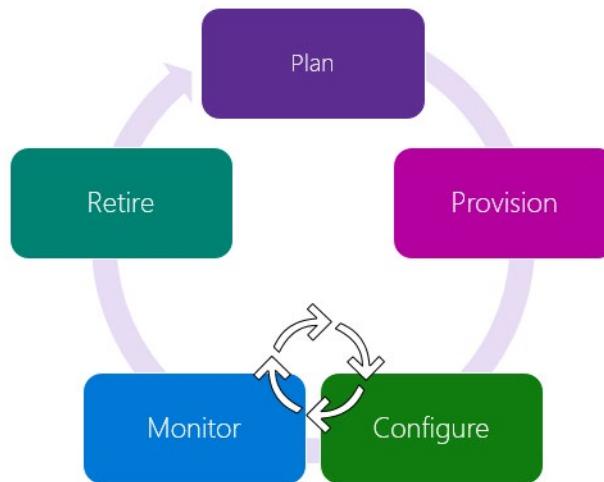
- **Scale and automation:** IoT solutions require simple tools that can automate routine tasks and enable a relatively small operations staff to manage millions of devices. Day-to-day, operators expect to handle device operations remotely, in bulk, and to only be alerted when issues arise that require their direct attention.
- **Openness and compatibility:** The device ecosystem is extraordinarily diverse. Management tools must be tailored to accommodate a multitude of device classes, platforms, and protocols. Operators

must be able to support many types of devices, from the most constrained embedded single-process chips, to powerful and fully functional computers.

- **Context awareness:** IoT environments are dynamic and ever-changing. Service reliability is paramount. Device management operations must take into account the following factors to ensure that maintenance downtime doesn't affect critical business operations or create dangerous conditions:
  - SLA maintenance windows
  - Network and power states
  - In-use conditions
  - Device geolocation
- **Service many roles:** Support for the unique workflows and processes of IoT operations roles is crucial. The operations staff must work harmoniously with the given constraints of internal IT departments. They must also find sustainable ways to surface realtime device operations information to supervisors and other business managerial roles.

## Device Lifecycle

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



Within each of these five stages, there are several device operator requirements that should be fulfilled to provide a complete solution:

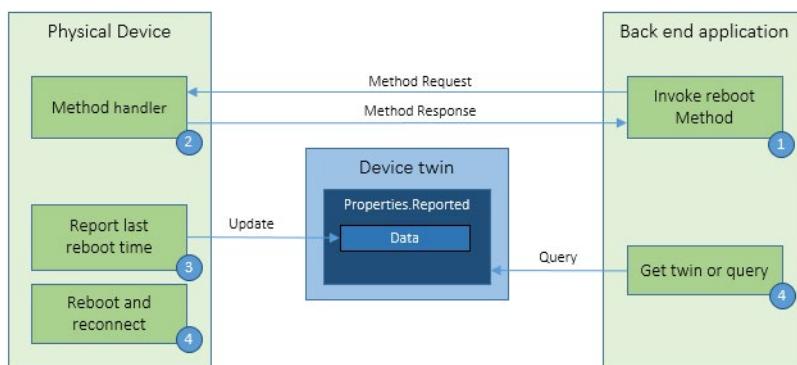
- **Plan:** Enable operators to create a device metadata scheme that enables them to easily and accurately query for, and target a group of devices for bulk management operations. You can use the device twin to store this device metadata in the form of tags and properties.
- **Provision:** Securely provision new devices to IoT Hub and enable operators to immediately discover device capabilities. Use the IoT Hub identity registry to create flexible device identities and credentials, and perform this operation in bulk by using a job. Build devices to report their capabilities and conditions through device properties in the device twin.
- **Configure:** Facilitate bulk configuration changes and firmware updates to devices while maintaining both health and security. Perform these device management operations in bulk by using desired properties or with direct methods and broadcast jobs.

- **Monitor:** Monitor overall device collection health, the status of ongoing operations, and alert operators to issues that might require their attention. Apply the device twin to allow devices to report realtime operating conditions and status of update operations. Build powerful dashboard reports that surface the most immediate issues by using device twin queries.
- **Retire:** Replace or decommission devices after a failure, upgrade cycle, or at the end of the service lifetime. Use the device twin to maintain device info if the physical device is being replaced, or archived if being retired. Use the IoT Hub identity registry for securely revoking device identities and credentials.

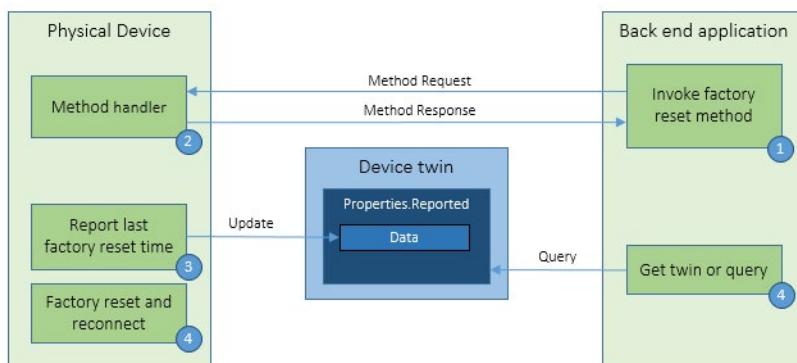
## Device Management Patterns

IoT Hub enables the following set of device management patterns.

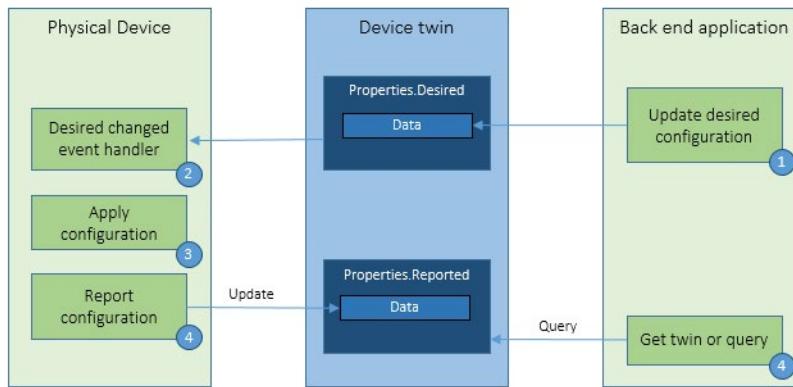
**Reboot:** The back-end app informs the device through a direct method that it has initiated a reboot. The device uses the reported properties to update the reboot status of the device.



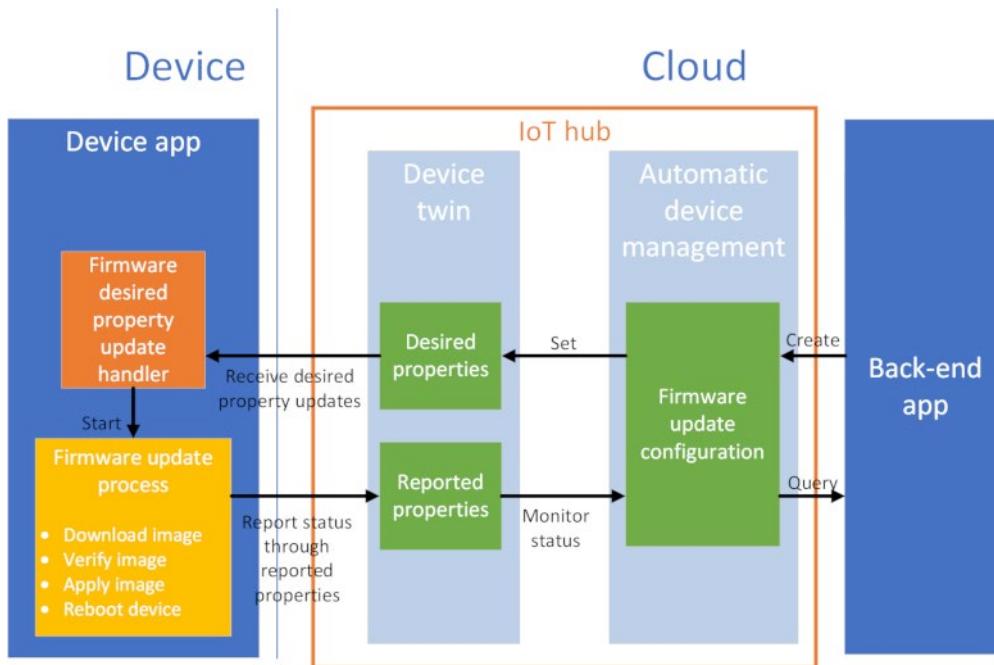
**Factory Reset:** The back-end app informs the device through a direct method that it has initiated a factory reset. The device uses the reported properties to update the factory reset status of the device.



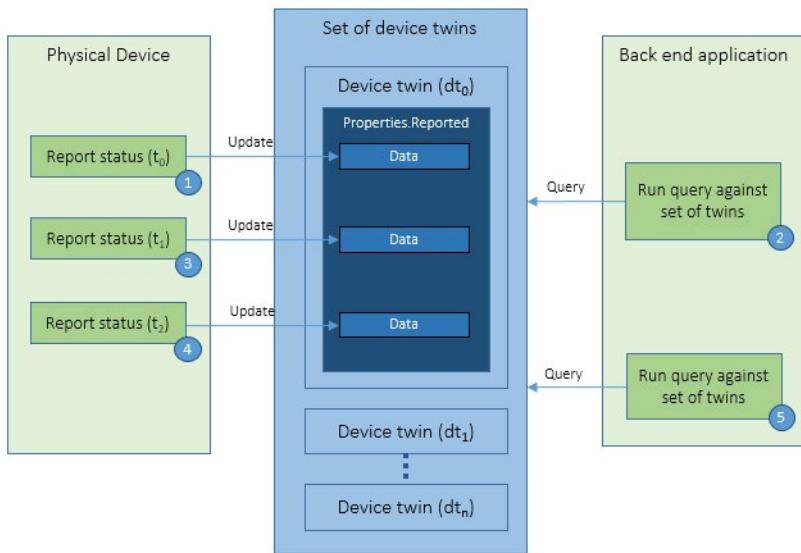
**Configuration:** The back-end app uses the desired properties to configure software running on the device. The device uses the reported properties to update configuration status of the device.



**Firmware Update:** The back-end app uses an automatic device management configuration to select the devices to receive the update, to tell the devices where to find the update, and to monitor the update process. The device initiates a multistep process to download, verify, and apply the firmware image, and then reboot the device before reconnecting to the IoT Hub service. Throughout the multistep process, the device uses the reported properties to update the progress and status of the device.



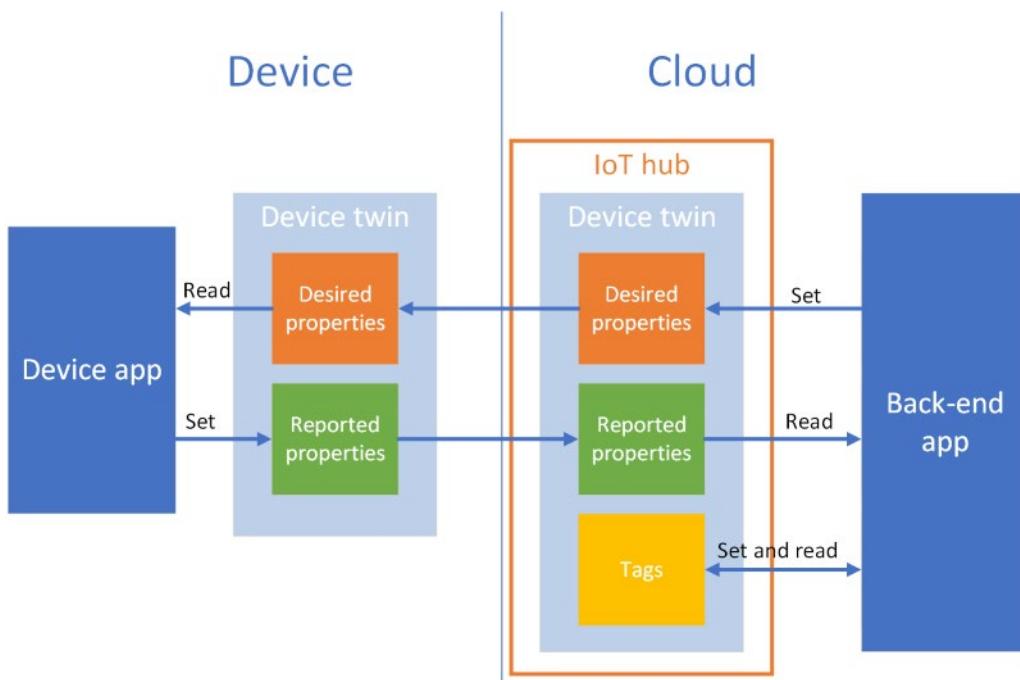
**Reporting Progress and Status:** The solution back end runs device twin queries, across a set of devices, to report on the status and progress of actions running on the devices.



## Device Configuration with Device Twins

Configuring devices from your back-end service is a core component of most device management solutions.

To synchronize state information between a device and an IoT hub, you use device twins. A desired property is set by a back-end application and read by a device. A reported property is set by a device and read by a back-end application. Device Twins also include a **tag** that is set by a back-end application, but is never sent to the device. You use tags to organize your devices.



## Device Twin Properties

Consider the following “properties” section of a device twin document:

```
"properties": {
  "desired": {
    "telemetryConfig": {
      "sendFrequency": "5m"
    },
    "$metadata": {...},
    "$version": 1
  },
  "reported": {
    "telemetryConfig": {
      "sendFrequency": "5m",
      "status": "success"
    },
    "$metadata": {...},
    "$version": 4
  }
}
```

In this example, the `telemetryConfig` device twin desired and reported properties are used by the solution back end and the device app to synchronize the telemetry configuration for this device. An update to the device configuration could be implemented as follows:

1. The solution back end sets the desired property with the desired configuration value.
2. The device app is notified of the change immediately if connected, or at the first reconnect. The device app then reports the updated configuration (or an error condition using the `status` property).
3. The solution back end can track the results of the configuration operation across many devices by querying device twins.

## Back-end operations

The solution back end operates on the device twin using the following atomic operations, exposed through HTTPS:

- Retrieve device twin by ID. This operation returns the device twin document, including tags and desired and reported system properties.
- Partially update device twin. This operation enables the solution back end to partially update the tags or desired properties in a device twin. The partial update is expressed as a JSON document that adds or updates any property. Properties set to `null` are removed. The following example creates a new `desired` property with value `{"newProperty": "newValue"}`, overwrites the existing value of `existingProperty` with `"other(newValue"`, and removes `otherOldProperty`. No other changes are made to existing desired properties or tags:

```
{
  "properties": {
    "desired": {
      "newProperty": {
        "nestedProperty": "newValue"
      },
      ...
    }
  }
}
```

```

        "existingProperty": "othernewValue",
        "otherOldProperty": null
    }
}
}
}

```

- Replace desired properties. This operation enables the solution back end to completely overwrite all existing desired properties and substitute a new JSON document for `properties/desired`.
- Replace tags. This operation enables the solution back end to completely overwrite all existing tags and substitute a new JSON document for `tags`.
- Receive twin notifications. This operation allows the solution back end to be notified when the twin is modified. To do so, your IoT solution needs to create a route and to set the Data Source equal to `twinChangeEvents`. By default, no such routes pre-exist, so no twin notifications are sent. If the rate of change is too high, or for other reasons such as internal failures, the IoT Hub might send only one notification that contains all changes. Therefore, if your application needs reliable auditing and logging of all intermediate states, you should use device-to-cloud messages. The twin notification message includes properties and body.
  - Properties

Name	Value
<code>\$content-type</code>	<code>application/json</code>
<code>\$iothub-enqueuedtime</code>	Time when the notification was sent
<code>\$iothub-message-source</code>	<code>twinChangeEvents</code>
<code>\$content-encoding</code>	<code>utf-8</code>
<code>deviceId</code>	ID of the device
<code>hubName</code>	Name of IoT Hub
<code>operationTimestamp</code>	ISO8601 timestamp of operation
<code>iothub-message-schema</code>	<code>deviceLifecycleNotification</code>
<code>opType</code>	" <code>replaceTwin</code> " or " <code>updateTwin</code> "

Message system properties are prefixed with the `$` symbol.

- Body

This section includes all the twin changes in a JSON format. It uses the same format as a patch, with the difference that it can contain all twin sections: `tags`, `properties.reported`, `properties.desired`, and that it contains the `"$metadata"` elements.

In addition to these operations, the solution back end can:

- Query the device twins using the SQL-like IoT Hub query language.
- Perform operations on large sets of device twins using jobs.

## Device operations

The device app operates on the device twin using the following atomic operations:

- Retrieve device twin. This operation returns the device twin document (including desired and reported system properties) for the currently connected device.

**Note:** Tags are not visible to device apps.

- Partially update reported properties. This operation enables the partial update of the reported properties of the currently connected device. This operation uses the same JSON update format that the solution back end uses for a partial update of desired properties.
- Observe desired properties. The currently connected device can choose to be notified of updates to the desired properties when they happen. The device receives the same form of update (partial or full replacement) executed by the solution back end.

All the preceding operations require the DeviceConnect permission.

The Azure IoT device SDKs make it easy to use the preceding operations from many languages and platforms.

## Device twin metadata

IoT Hub maintains the timestamp of the last update for each JSON object in device twin desired and reported properties. The timestamps are in UTC and encoded in the ISO8601 format YYYY-MM-DDTHH:MM:SS.fffZ.

For example:

```
"properties": {  
    "desired": {  
        "telemetryConfig": {  
            "sendFrequency": "5m"  
        },  
        "$metadata": {  
            "telemetryConfig": {  
                "sendFrequency": {  
                    "$lastUpdated": "2019-08-30T16:24:48.789Z"  
                },  
                "$lastUpdated": "2019-08-30T16:24:48.789Z"  
            },  
            "$lastUpdated": "2019-08-30T16:24:48.789Z"  
        },  
        "$version": 23  
    },  
    "reported": {  
        "telemetryConfig": {  
            "sendFrequency": "5m",  
            "status": "success"  
        },  
        "$metadata": {  
            "telemetryConfig": {  
                "sendFrequency": "5m",  
                "status": {  
                    "$lastUpdated": "2019-08-31T16:35:48.789Z"  
                },  
                "$lastUpdated": "2019-08-31T16:35:48.789Z"  
            },  
            "$lastUpdated": "2019-08-31T16:35:48.789Z"  
        },  
        "$version": 123  
    }  
}
```

}

This information is kept at every level (not just the leaves of the JSON structure) to preserve updates that remove object keys.

## Optimistic concurrency

Tags, desired, and reported properties all support optimistic concurrency. Tags have an ETag, as per RFC7232, that represents the tag's JSON representation. You can use ETags in conditional update operations from the solution back end to ensure consistency.

Device twin desired and reported properties do not have ETags, but have a \$version value that is guaranteed to be incremental. Similarly to an ETag, the version can be used by the updating party to enforce consistency of updates. For example, a device app for a reported property or the solution back end for a desired property.

Versions are also useful when an observing agent (such as the device app observing the desired properties) must reconcile races between the result of a retrieve operation and an update notification.

## Device reconnection flow

IoT Hub does not preserve desired properties update notifications for disconnected devices. It follows that a device that is connecting must retrieve the full desired properties document, in addition to subscribing for update notifications. Given the possibility of races between update notifications and full retrieval, the following flow must be ensured:

- Device app connects to an IoT hub.
- Device app subscribes for desired properties update notifications.
- Device app retrieves the full document for desired properties.

The device app can ignore all notifications with \$version less or equal than the version of the full retrieved document. This approach is possible because IoT Hub guarantees that versions always increment.

**Note:** This logic is already implemented in the Azure IoT device SDKs. This description is useful only if the device app cannot use any of Azure IoT device SDKs and must program the MQTT interface directly.

## Direct Methods

IoT Hub gives you the ability to invoke direct methods on devices from the cloud. Direct methods represent a request-reply interaction with a device similar to an HTTP call in that they succeed or fail immediately (after a user-specified timeout). This approach is useful for scenarios where the course of immediate action is different depending on whether the device was able to respond.

**Note:** To implement Direct Methods, you need to be running the standard tier of the IoT Hub service.

Direct methods have the following features:

- Each direct method targets a single device. You can use IoT Hub jobs to invoke direct methods on multiple devices, and schedule method invocation for disconnected devices.
- Anyone with service connect permissions on IoT Hub may invoke a method on a device.

- Direct methods follow a request-response pattern and are meant for communications that require immediate confirmation of their result. For example, interactive control of the device, such as turning on a fan.

## Method lifecycle

Direct methods are implemented on the device and may require zero or more inputs in the method payload to correctly instantiate. You invoke a direct method through a service-facing URI (`{iot hub}/twins/{device id}/methods/`). A device receives direct methods through a device-specific MQTT topic (`$iothub/methods/POST/{method name}/`) or through AMQP links (the `IoThub-method-name` and `IoThub-status` application properties).

**Note:** When you invoke a direct method on a device, property names and values can only contain US-ASCII printable alphanumeric, except any in the following set: `{$', '(', ')', '<', '>', '@', ',', ';', ':', '\', "'", '/', '[', ']', '?', '=', '{', '}', SP, HT}`

Direct methods are synchronous and either succeed or fail after the timeout period (default: 30 seconds, settable up to 300 seconds). Direct methods are useful in interactive scenarios where you want a device to act if and only if the device is online and receiving commands. For example, turning on a light from a phone. In these scenarios, you want to see an immediate success or failure so the cloud service can act on the result as soon as possible. The device may return some message body as a result of the method, but it isn't required for the method to do so. There is no guarantee on ordering or any concurrency semantics on method calls.

Direct methods are HTTPS-only from the cloud side, and MQTT or AMQP from the device side.

The payload for method requests and responses is a JSON document up to 128 KB.

## Invoke a direct method from a back-end app

### Method invocation

Direct method invocations on a device are HTTPS calls that are made up of the following items:

- The request URI specific to the device along with the API version:

`https://fully-qualified-iothubname.azure-devices.net/twins/{deviceId}/methods?api-version=2018-06-30`

- The POST method
- Headers that contain the authorization, request ID, content type, and content encoding.
- A transparent JSON body in the following format:

```
{  
  "methodName": "reboot",  
  "responseTimeoutInSeconds": 200,  
  "payload": {  
    "input1": "someInput",  
    "input2": "anotherInput"  
  }  
}
```

Timeout is in seconds. If timeout is not set, it defaults to 30 seconds.

## Example

The following is a barebones example using curl.

```
curl -X POST \
  https://iothubname.azure-devices.net/twins/myfirstdevice/methods?api-version=2018-06-30 \
  -H 'Authorization: SharedAccessSignature sr=iothubname.azure-devices.net&sig=x&se=x&skn=iothu-
bowner' \
  -H 'Content-Type: application/json' \
  -d '{
    "methodName": "reboot",
    "responseTimeoutInSeconds": 200,
    "payload": {
      "input1": "someInput",
      "input2": "anotherInput"
    }
}'
```

## Response

The back-end app receives a response that is made up of the following items:

- HTTP status code, which is used for errors coming from the IoT Hub, including a 404 error for devices not currently connected.
- Headers that contain the ETag, request ID, content type, and content encoding.
- A JSON body in the following format:

```
{
  "status" : 201,
  "payload" : {...}
}
```

Both `status` and `body` are provided by the device and used to respond with the device's own status code and/or description.

## Method invocation for IoT Edge modules

Invoking direct methods using a module ID is supported in the IoT Service Client C# SDK.

For this purpose, use the `ServiceClient.InvokeDeviceMethodAsync()` method and pass in the `deviceId` and `moduleId` as parameters.

## Handle a Direct Method on a Device

A Direct Method on a device can receive requests using either the MQTT or AMQP protocol.

# MQTT

## Method invocation

Devices receive direct method requests on the MQTT topic: \$iothub/methods/POST/{method name}/?\$rid={request id}. The number of subscriptions per device is limited to 5. It is therefore recommended not to subscribe to each direct method individually. Instead consider subscribing to \$iothub/methods/POST/# and then filter the delivered messages based on your desired method names.

The body that the device receives is in the following format:

```
{  
    "input1": "someInput",  
    "input2": "anotherInput"  
}
```

Method requests are QoS 0.

## Response

The device sends responses to \$iothub/methods/res/{status}/?\$rid={request id}, where:

- The status property is the device-supplied status of method execution.
- The \$rid property is the request ID from the method invocation received from IoT Hub.

The body is set by the device and can be any status.

# AMQP

The following section is for the AMQP protocol.

## Method invocation

The device receives direct method requests by creating a receive link on address amqps:// {hostname} : 5671 / devices / {deviceId} / methods / deviceBound.

The AMQP message arrives on the receive link that represents the method request. It contains the following sections:

- The correlation ID property, which contains a request ID that should be passed back with the corresponding method response.
- An application property named IoThub-methodname, which contains the name of the method being invoked.
- The AMQP message body containing the method payload as JSON.

## Response

The device creates a sending link to return the method response on address amqps:// {hostname} : 5671 / devices / {deviceId} / methods / deviceBound.

The method's response is returned on the sending link and is structured as follows:

- The correlation ID property, which contains the request ID passed in the method's request message.
- An application property named `IoTHub-status`, which contains the user supplied method status.
- The AMQP message body containing the method response as JSON.

## Comparing Device Management Approaches

IoT Hub provides three options for device apps to expose functionality to a back-end app:

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

Of these options, direct methods and device twin properties are good choices for device management.

## Choosing Between Device Twin and Direct Method Approaches

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

	Direct methods	Twin's desired properties	Cloud-to-device messages
Scenario	Commands that require immediate confirmation, such as turning on a fan.	Long-running commands intended to put the device into a certain desired state. For example, set the telemetry send interval to 30 minutes.	One-way notifications to the device app.
Data flow	Two-way. The device app can respond to the method right away. The solution back end receives the outcome contextually to the request.	One-way. The device app receives a notification with the property change.	One-way. The device app receives the message
Durability	Disconnected devices are not contacted. The solution back end is notified that the device is not connected.	Property values are preserved in the device twin. Device will read it at next reconnection. Property values are retrievable with the IoT Hub query language.	Messages can be retained by IoT Hub for up to 48 hours.
Targets	Single device using <code>deviceld</code> , or multiple devices using jobs.	Single device using <code>deviceld</code> , or multiple devices using jobs.	Single device by <code>deviceld</code> .

	Direct methods	Twin's desired properties	Cloud-to-device messages
Size	Maximum direct method payload size is 128 KB.	Maximum desired properties size is 8 KB.	Up to 64 KB messages.
Frequency	High. For more information, see IoT Hub limits.	Medium. For more information, see IoT Hub limits.	Low. For more information, see IoT Hub limits.
Protocol	Available using MQTT or AMQP.	Available using MQTT or AMQP.	Available on all protocols. Device must poll when using HTTPS.

# Manage IoT and IoT Edge Devices

## Device Management Tools

There are three primary tools that we will be using to manage devices in this course, and the good news is that you are already familiar with them.

### IoT Hub (Azure portal)

The Azure portal can be used to implement device management in the following ways:

- Update Device Twin properties and tags for a device
- Invoke a Direct Method on a device
- Implement Automatic Device Management

### Azure CLI

Azure CLI and the Azure CLI extension for IoT commands can be used to implement device management in the following ways:

- Update Device Twin properties and tags for a device
- Invoke a Direct Method on a device
- Perform Device Twin queries across devices

### VS Code

Visual Studio Code and the Azure IoT Tools extension can be used to implement device management in the following ways:

- Create device code (such as direct methods or code used to help manage devices using device twin properties)
- Create backend apps (to support specific management tasks as well as jobs that support management at scale)

## Device Management using the IoT Extension for Azure CLI

The IoT extension for Azure CLI gives IoT developers command-line access to all IoT Hub, IoT Edge, and IoT Hub Device Provisioning Service capabilities. This includes the device management capabilities provided by the IoT Hub service.

Management option	Task
Direct methods	Make a device act such as starting or stopping sending messages or rebooting the device.
Twin desired properties	Put a device into certain states, such as setting an LED to green or setting the telemetry send interval to 30 minutes.

Management option	Task
Twin reported properties	Get the reported state of a device. For example, the device reports the LED is blinking now.
Twin tags	Store device-specific metadata in the cloud. For example, the deployment location of a vending machine.
Device twin queries	Query all device twins to retrieve those twins with arbitrary conditions, such as identifying the devices that are available for use.

**Note** If you don't have the IoT extension installed, the simplest way to install it is to run `az extension add --name azure-cli-iot-ext`

Before you can enter any device management commands, you need to sign in to your Azure account:  
`az login`

## Direct Methods

To invoke a direct method on a device, use the following command

```
az iot hub invoke-device-method --device-id <your device id> \
--hub-name <your hub name> \
--method-name <the method name> \
--method-payload <the method payload>
```

## Device twin desired properties

Set a desired property interval = 3000 by running the following command:

```
az iot hub device-twin update -n <your hub name> \
-d <your device id> --set properties.desired.interval = 3000
```

This property can be read from your device.

## Device twin reported properties

Get the reported properties of the device by running the following command:

```
az iot hub device-twin show -n <your hub name> -d <your device id>
```

One of the twin reported properties is `$metadata.$lastUpdated`, which shows the last time the device app updated its reported property set.

## Device twin tags

Display the tags and properties of the device by running the following command:

```
az iot hub device-twin show --hub-name <your hub name> --device-id <your device id>
```

Add a field role = temperature&humidity to the device by running the following command:

```
az iot hub device-twin update \
--hub-name <your hub name> \
--device-id <your device id> \
--set tags = '{"role":"temperature&humidity"}'
```

## Device twin queries

Query devices with a tag of role = 'temperature&humidity' by running the following command:

```
az iot hub query --hub-name <your hub name> \
--query-command "SELECT * FROM devices WHERE tags.role = 'temperature&humidity'"
```

Query all devices except those with a tag of role = 'temperature&humidity' by running the following command:

```
az iot hub query --hub-name <your hub name> \
--query-command "SELECT * FROM devices WHERE tags.role != 'temperature&humidity'"
```

## Device Management using the Azure IoT Tools for VS Code

Azure IoT Tools extension for Visual Studio Code includes the ability to perform device management tasks.

Management option	Task
Invoke Direct Method	Make a device perform an action, such as starting or stopping the process of sending messages, or rebooting the device.
Edit Device Twin	Examine reported properties to get the reported state of a device. For example, the device reports the LED is blinking now. Use desired properties to put a device into certain states, such as setting an LED to green or setting the telemetry send interval to 30 minutes.
Cloud-to-device messages	Send notifications to a device. For example, "It is very likely to rain today. Don't forget to bring an umbrella."

The Azure IoT Tools extension provides additional capabilities for working with devices in addition to these device management capabilities.

## Access Your IoT Hub and Devices

When using the Azure IoT Tools extension for VS Code, the first step is to access to your IoT Hub through your Azure subscription.

- In the bottom left corner of the VS Code window, click **Azure IoT Hub**.

When the Azure IoT Tools extension for VS Code is installed, the Azure IoT Hub section is added to the VS Code Explorer pane.

1. To the right of Azure IoT Hub, click the ellipsis (...).

The ellipsis (...) provides access to the More Options context menu.

1. On the content menu, click **Select IoT Hub**

A pop-up will be displayed on screen to let you sign in to Azure.

After you sign in, your Azure Subscription list will be shown.

1. Select the Azure Subscription that you will be using.
2. Select the IoT Hub that you will be using.

After a few seconds, the VS Code Explorer pane will be updated to show a Devices section under Azure IoT Hub. The Devices section will display a list of the devices connected to the IoT Hub that you selected.

## Access Device Management Commands

You can access the device management commands by right-clicking a device in the Explorer pane, and then selecting a command from the context menu.

### Invoke a Direct Method

To invoke a direct method on a device:

1. In the VS Code Explorer pane, right-click the device that you are interested in.
2. On the context menu for your device, click **Invoke Direct Method**.
3. Enter the method name in the input box, and then the associated payload value.

The results will be shown in OUTPUT > Azure IoT Hub Toolkit view. If the direct method that you specify does exist on the device, you will see message similar to the following:

Failed to invoke Direct Method: Not found

### Review a Device Twin

To review the contents of a device twin document (json file):

1. In the VS Code Explorer pane, right-click the device that you are interested in.
2. On the context menu for your device, click **Edit Device Twin**.

An azure-iot-device-twin.json file will be opened in VS Code showing the contents of the device twin document.

### Update a Device Twin

After opening a device twin document, you can update desired properties as follows:

1. Make some edits to the properties.desired field.

You can also make changes to tags. Tags can be used to support of device management tasks that act on a group of devices that have the same tag value setting.

2. Right-click anywhere within the azure-iot-device-twin.json document.

This will open the VS Code context menu for the open document.

3. On the context menu, to save the changes to the device twin, click **Update Device Twin**.

## Send Cloud-to-Device Messages

To send a message from your IoT hub to your device, follow these steps:

1. In the VS Code Explorer pane, right-click the device that you are interested in.
2. On the context menu for your device, click **Send C2D Message to Device**.
3. Enter the message in input box.

Results will be shown in OUTPUT > Azure IoT Hub Toolkit view.

# Device Management at Scale

## Schedule jobs on multiple devices

Typically, back-end apps enable device administrators and operators to update and interact with IoT devices in bulk and at a scheduled time. Jobs execute device twin updates and direct methods against a set of devices at a scheduled time. For example, an operator would use a back-end app that initiates and tracks a job to reboot a set of devices in building 43 and floor 3 at a time that would not be disruptive to the operations of the building.

Consider using jobs when you need to schedule and track progress any of the following activities on a set of devices:

- Update desired properties
- Update tags
- Invoke direct methods

## Job lifecycle

Jobs are initiated by the solution back end and maintained by IoT Hub. You can initiate a job through a service-facing URI (PUT `https://<iot hub>/jobs/v2/<jobID>?api-version=2018-06-30`) and query for progress on an executing job through a service-facing URI (GET `https://<iot hub>/jobs/v2/<jobID>?api-version=2018-06-30`). To refresh the status of running jobs once a job is initiated, run a job query.

**Note:** When you initiate a job, property names and values can only contain US-ASCII printable alphanumeric, except any in the following set: \$ ( ) < > @ , ; : \ " / [ ] ? = { } SP HT

## Jobs to execute direct methods

The following snippet shows the HTTPS 1.1 request details for executing a direct method on a set of devices using a job:

```
PUT /jobs/v2/<jobId>?api-version=2018-06-30

Authorization: <config.sharedAccessSignature>
Content-Type: application/json; charset=utf-8

{
    "jobId": "<jobId>",
    "type": "scheduleDeviceMethod",
    "cloudToDeviceMethod": {
        "methodName": "<methodName>",
        "payload": <payload>,
        "responseTimeoutInSeconds": methodTimeoutInSeconds
    },
    "queryCondition": "<queryOrDevices>", // query condition
    "startTime": <jobStartTime>,           // as an ISO-8601 date string
    "maxExecutionTimeInSeconds": <maxExecutionTimeInSeconds>
}
```

The query condition can also be on a single device ID or on a list of device IDs as shown in the following examples:

```
"queryCondition" = "deviceId = 'MyDevice1'"
"queryCondition" = "deviceId IN ['MyDevice1','MyDevice2']"
"queryCondition" = "deviceId IN ['MyDevice1']"
```

IoT Hub Query Language <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-devguide-query-language> covers IoT Hub query language in additional detail.

The following snippet shows the request and response for a job scheduled to call a direct method named testMethod on all devices on contoso-hub-1:

```
PUT https://contoso-hub-1.azure-devices.net/jobs/v2/job01?api-version=2018-06-30 HTTP/1.1
Authorization: SharedAccessSignature sr=contoso-hub-1.azure-devices.net&sig=68iv-----v8Hxal-g%3D&se=1556849884&skn=iothubowner
Content-Type: application/json; charset=utf-8
Host: contoso-hub-1.azure-devices.net
Content-Length: 317

{
    "jobId": "job01",
    "type": "scheduleDeviceMethod",
    "cloudToDeviceMethod": {
        "methodName": "testMethod",
        "payload": {},
        "responseTimeoutInSeconds": 30
    },
    "queryCondition": "*",
    "startTime": "2019-05-04T15:53:00.077Z",
    "maxExecutionTimeInSeconds": 20
}

HTTP/1.1 200 OK
Content-Length: 65
Content-Type: application/json; charset=utf-8
Vary: Origin
Server: Microsoft-HTTPAPI/2.0
Date: Fri, 03 May 2019 01:46:18 GMT

{"jobId":"job01","type":"scheduleDeviceMethod","status":"queued"}
```

## Jobs to update device twin properties

The following snippet shows the HTTPS 1.1 request details for updating device twin properties using a job:

```
PUT /jobs/v2/<jobId>?api-version=2018-06-30
```

```
Authorization: <config.sharedAccessSignature>
Content-Type: application/json; charset=utf-8

{
    "jobId": "<jobId>",
    "type": "scheduleUpdateTwin",
    "updateTwin": <patch> // Valid JSON object
    "queryCondition": "<queryOrDevices>", // query condition
    "startTime": <jobStartTime>, // as an ISO-8601 date string
    "maxExecutionTimeInSeconds": <maxExecutionTimeInSeconds>
}
```

**Note:** The updateTwin property requires a valid etag match; for example, etag="\*".

The following snippet shows the request and response for a job scheduled to update device twin properties for test-device on contoso-hub-1:

```
PUT https://contoso-hub-1.azure-devices.net/jobs/v2/job02?api-version=2018-06-30 HTTP/1.1
Authorization: SharedAccessSignature sr=contoso-hub-1.azure-devices.net&sig=BN0U-----RuA%3D&se=1556925787&sk-niothubowner
Content-Type: application/json; charset=utf-8
Host: contoso-hub-1.azure-devices.net
Content-Length: 339

{
    "jobId": "job02",
    "type": "scheduleUpdateTwin",
    "updateTwin": {
        "properties": {
            "desired": {
                "test1": "value1"
            }
        },
        "etag": "*"
    },
    "queryCondition": "deviceId = 'test-device'",
    "startTime": "2019-05-08T12:19:56.868Z",
    "maxExecutionTimeInSeconds": 20
}

HTTP/1.1 200 OK
Content-Length: 63
Content-Type: application/json; charset=utf-8
Vary: Origin
Server: Microsoft-HTTPAPI/2.0
Date: Fri, 03 May 2019 22:45:13 GMT

{"jobId":"job02","type":"scheduleUpdateTwin","status":"queued"}
```

## Querying for progress on jobs

The following snippet shows the HTTPS 1.1 request details for querying for jobs:

```
GET /jobs/v2/query?api-version=2018-06-30[&jobType=<jobType>][&jobStatus=<jobStatus>][&pageSize=<pageSize>][&continuationToken=<continuationToken>]
```

```
Authorization: <config.sharedAccessSignature>
Content-Type: application/json; charset=utf-8
```

The continuationToken is provided from the response.

You can query for the job execution status on each device using the IoT Hub query language for device twins, jobs, and message routing.

## Jobs Properties

The following list shows the properties and corresponding descriptions, which can be used when querying for jobs or job results.

Property	Description
jobId	Application provided ID for the job.
startTime	Application provided start time (ISO-8601) for the job.
endTime	IoT Hub provided date (ISO-8601) for when the job completed. Valid only after the job reaches the 'completed' state.
type	Types of jobs: scheduleUpdateTwin: A job used to update a set of desired properties or tags. scheduleDeviceMethod: A job used to invoke a device method on a set of device twins.
status	Current state of the job. Possible values for status: pending: Scheduled and waiting to be picked up by the job service. scheduled: Scheduled for a time in the future. running: Currently active job. canceled: Job has been canceled. failed: Job failed. completed: Job has completed.

Property	Description
deviceJobStatistics	Statistics about the job's execution. deviceJobStatistics properties: deviceJobStatistics.deviceCount: Number of devices in the job. deviceJobStatistics.failedCount: Number of devices where the job failed. deviceJobStatistics.succeededCount: Number of devices where the job succeeded. deviceJobStatistics.runningCount: Number of devices that are currently running the job. deviceJobStatistics.pendingCount: Number of devices that are pending to run the job.

## Automatic Device Management using the Azure Portal

Automatic device management in Azure IoT Hub automates many of the repetitive and complex tasks of managing large device fleets. With automatic device management, you can target a set of devices based on their properties, define a desired configuration, and then let IoT Hub update the devices when they come into scope. This update is done using an automatic device configuration, which lets you summarize completion and compliance, handle merging and conflicts, and roll out configurations in a phased approach.

**Note:** Automatic device management requires the Standard tier of the IoT Hub service.

Automatic device management works by updating a set of device twins with desired properties and reporting a summary that's based on device twin reported properties. It introduces a new class and JSON document called a Configuration that has three parts:

- The target condition defines the scope of device twins to be updated. The target condition is specified as a query on device twin tags and/or reported properties.
- The target content defines the desired properties to be added or updated in the targeted device twins. The content includes a path to the section of desired properties to be changed.
- The metrics define the summary counts of various configuration states such as Success, In Progress, and Error. Custom metrics are specified as queries on device twin reported properties. System metrics are the default metrics that measure twin update status, such as the number of device twins that are targeted and the number of twins that have been successfully updated.

Automatic device configurations run for the first time shortly after the configuration is created and then at five minute intervals. Metrics queries run each time the automatic device configuration runs.

## Identify Devices using Tags

Before you create a configuration, you must specify which devices you want to affect. Azure IoT Hub identifies devices using tags in the device twin. Each device can have multiple tags, and you can define them any way that makes sense for your solution. For example, if you manage devices in different locations, add the following tags to a device twin:

```
"tags": {
    "location": {
        "state": "Washington",
    }
}
```

```

    "city": "Tacoma"
}
},

```

## Create a configuration

You can use the Azure portal to begin the process of creating a Configuration as follows:

1. Open your IoT Hub.
2. Select **IoT device configuration**.
3. Select **Add Configuration**.

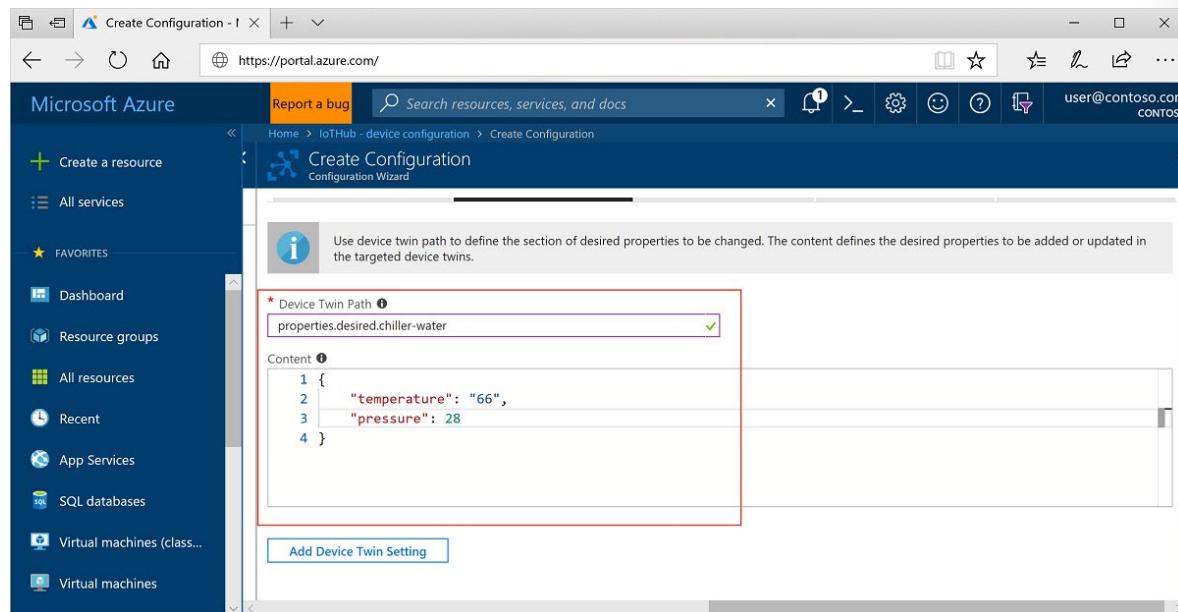
Use the following five steps to complete the process.

### Name and Label

1. Give your configuration a unique name that is up to 128 lowercase letters. Avoid spaces and the following invalid characters: & ^ [ ] { } \ | " < > /
2. Add labels to help track your configurations. Labels are **Name, Value** pairs that describe your configuration. For example, HostPlatform, Linux or Version, 3.0.1.
3. Select **Next** to move to the next step.

### Specify Settings

This section specifies the target content to be set in targeted device twins. There are two inputs for each set of settings. The first is the device twin path, which is the path to the JSON section within the twin desired properties that will be set. The second is the JSON content to be inserted in that section. For example, set the Device Twin Path and Content to the following:



MCT USE ONLY. STUDENT USE PROHIBITED

You can also set individual settings by specifying the entire path in the Device Twin Path and the value in the Content with no brackets. For example, set the Device Twin Path to `properties.desired.chiller-water.temperature` and set the Content to 66.

If two or more configurations target the same Device Twin Path, the Content from the highest priority configuration will apply (priority is defined in Step 4).

If you wish to remove a property, specify the property value to null.

You can add additional settings by selecting **Add Device Twin Setting**.

## Specify Metrics (optional)

Metrics provide summary counts of the various states that a device may report back after applying configuration content. For example, you may create a metric for pending settings changes, a metric for errors, and a metric for successful settings changes.

1. Enter a name for Metric Name.
2. Enter a query for Metric Criteria. The query is based on device twin reported properties. The metric represents the number of rows returned by the query.

For example:

```
SELECT deviceId FROM devices  
WHERE properties.reported.chillerWaterSettings.status='pending'
```

You can include a clause that the configuration was applied, for example:

```
/* Include the double brackets. */  
SELECT deviceId FROM devices  
WHERE configurations.[[yourconfigname]].status='Applied'
```

## Target Devices

Use the tags property from your device twins to target the specific devices that should receive this configuration. You can also target devices by device twin reported properties.

Since multiple configurations may target the same device, you should give each configuration a priority number. If there's ever a conflict, the configuration with the highest priority wins.

1. Enter a positive integer for the configuration **Priority**. The highest numerical value is considered the highest priority. If two configurations have the same priority number, the one that was created most recently wins.
2. Enter a **Target condition** to determine which devices will be targeted with this configuration. The condition is based on device twin tags or device twin reported properties and should match the expression format. For example, `tags.environment='test'` or `properties.reported.chillerProperties.model='4000x'`. You can specify \* to target all devices.
3. Select **Next** to move on to the final step.

## Review Configuration

Review your configuration information, then select Submit.

## Monitor a configuration

To view the details of a configuration and monitor the devices running it, use the following steps:

1. In the Azure portal, go to your IoT hub.
2. Select **IoT device configuration**.
3. Inspect the configuration list. For each configuration, you can view the following details:
  - **ID** - the name of the configuration.
  - **Target condition** - the query used to define targeted devices.
  - **Priority** - the priority number assigned to the configuration.
  - **Creation time** - the timestamp from when the configuration was created. This timestamp is used to break ties when two configurations have the same priority.
  - **System metrics** - metrics that are calculated by IoT Hub and cannot be customized by developers. Targeted specifies the number of device twins that match the target condition. Applies specified the number of device twins that have been modified by the configuration, which can include partial modifications in the event that a separate, higher priority configuration also made changes.
  - **Custom metrics** - metrics that have been specified by the developer as queries against device twin reported properties. Up to five custom metrics can be defined per configuration.
4. Select the configuration that you want to monitor.
5. Inspect the configuration details. You can use tabs to view specific details about the devices that received the configuration.
  - **Target Condition** - the devices that match the target condition.
  - **Metrics** - a list of system metrics and custom metrics. You can view a list of devices that are counted for each metric by selecting the metric in the drop-down and then selecting **View Devices**.
  - **Device Twin Settings** - the device twin settings that are set by the configuration.
  - **Configuration Labels** - key-value pairs used to describe a configuration. Labels have no impact on functionality.

## Modify a configuration

When you modify a configuration, the changes immediately replicate to all targeted devices.

If you update the target condition, the following updates occur:

- If a device twin didn't meet the old target condition, but meets the new target condition and this configuration is the highest priority for that device twin, then this configuration is applied to the device twin.
- If a device twin no longer meets the target condition, the settings from the configuration will be removed and the device twin will be modified by the next highest priority configuration.
- If a device twin currently running this configuration no longer meets the target condition and doesn't meet the target condition of any other configurations, then the settings from the configuration will be removed and no other changes will be made on the twin.

To modify a configuration, use the following steps:

1. In the Azure portal, go to your IoT hub.

2. Select **IoT device configuration**.
3. Select the configuration that you want to modify.
4. Make updates to the following fields:
  - Target condition
  - Labels
  - Priority
  - Metrics
5. Select **Save**.
6. Follow the steps in Monitor a configuration to watch the changes roll out.

## Delete a configuration

When you delete a configuration, any device twins take on their next highest priority configuration. If device twins don't meet the target condition of any other configuration, then no other settings are applied.

1. In the Azure portal, go to your IoT hub.
2. Select **IoT device configuration**.
3. Use the checkbox to select the configuration that you want to delete.
4. Select **Delete**.
5. A prompt will ask you to confirm.

## Automatic Device Management using Azure CLI

Automatic device management using Azure CLI is functionality equivalent to the process for using the Azure portal.

### CLI prerequisites

- An IoT hub in your Azure subscription.
- Azure CLI in your environment. At a minimum, your Azure CLI version must be 2.0.24 or above. Use az --version to validate. This version supports az extension commands and introduces the Knack command framework.
- The IoT extension for Azure CLI.

## Create a configuration

You configure target devices by creating a configuration that consists of the target content and metrics.

Use the following command to create a configuration:

```
az iot hub configuration create --config-id [configuration id] \
--labels [labels] --content [file path] --hub-name [hub name] \
--target-condition [target query] --priority [int] \
--metrics [metric queries]
```

- **--config-id** - The name of the configuration that will be created in the IoT hub. Give your configuration a unique name that is up to 128 lowercase letters. Avoid spaces and the following invalid characters: & ^ [ ] { } \ | " < > /
- **--labels** - Add labels to help track your configuration. Labels are Name, Value pairs that describe your deployment. For example, HostPlatform, Linux or Version, 3.0.1
- **--content** - Inline JSON or file path to the target content to be set as twin desired properties.
- **--hub-name** - Name of the IoT hub in which the configuration will be created. The hub must be in the current subscription. Switch to the desired subscription with the command `az account set -s [subscription name]`
- **--target-condition** - Enter a target condition to determine which devices will be targeted with this configuration. The condition is based on device twin tags or device twin desired properties and should match the expression format. For example, `tags.environment='test'` or `properties.desired.devicemodel='4000x'`.
- **--priority** - A positive integer. In the event that two or more configurations are targeted at the same device, the configuration with the highest numerical value for Priority will apply.
- **--metrics** - Filepath to the metric queries. Metrics provide summary counts of the various states that a device may report back after applying configuration content. For example, you may create a metric for pending settings changes, a metric for errors, and a metric for successful settings changes.

## Monitor a configuration

Use the following command to display the contents of a configuration:

```
az iot hub configuration show --config-id [configuration id] \
--hub-name [hub name]
```

- **--config-id** - The name of the configuration that exists in the IoT hub.
- **--hub-name** - Name of the IoT hub in which the configuration exists. The hub must be in the current subscription. Switch to the desired subscription with the command `az account set -s [subscription name]`

Inspect the configuration in the command window. The metrics property lists a count for each metric that is evaluated by each hub:

- **targetedCount** - A system metric that specifies the number of device twins in IoT Hub that match the targeting condition.
- **appliedCount** - A system metric specifies the number of devices that have had the target content applied.
- **Your custom metric** - Any metrics you've defined are user metrics.

You can show a list of device IDs or objects for each of the metrics by using the following command:

```
az iot hub configuration show-metric --config-id [configuration id] \
--metric-id [metric id] --hub-name [hub name] --metric-type [type]
```

- **--config-id** - The name of the deployment that exists in the IoT hub.
- **--metric-id** - The name of the metric for which you want to see the list of device IDs, for example `appliedCount`.

- `--hub-name` - Name of the IoT hub in which the deployment exists. The hub must be in the current subscription. Switch to the desired subscription with the command `az account set -s [subscription name]`
- `--metric-type` - Metric type can be `system` or `user`. System metrics are `targetedCount` and `appliedCount`. All other metrics are user metrics.

## Modify a configuration

When you modify a configuration, the changes immediately replicate to all targeted devices.

If you update the target condition, the following updates occur:

- If a device twin didn't meet the old target condition, but meets the new target condition and this configuration is the highest priority for that device twin, then this configuration is applied to the device twin.
- If a device twin no longer meets the target condition, the settings from the configuration will be removed and the device twin will be modified by the next highest priority configuration.
- If a device twin currently running this configuration no longer meets the target condition and doesn't meet the target condition of any other configurations, then the settings from the configuration will be removed and no other changes will be made on the twin.

Use the following command to update a configuration:

```
az iot hub configuration update --config-id [configuration id] \
--hub-name [hub name] --set [property1.property2='value']
```

- `--config-id` - The name of the configuration that exists in the IoT hub.
- `--hub-name` - Name of the IoT hub in which the configuration exists. The hub must be in the current subscription. Switch to the desired subscription with the command `az account set -s [subscription name]`
- `--set` - Update a property in the configuration. You can update the following properties:
  - `targetCondition` - for example `targetCondition=tags.location.state='Oregon'`
  - `labels`
  - `priority`

## Delete a configuration

When you delete a configuration, any device twins take on their next highest priority configuration. If device twins don't meet the target condition of any other configuration, then no other settings are applied.

Use the following command to delete a configuration:

```
az iot hub configuration delete --config-id [configuration id] \
--hub-name [hub name]
```

`--config-id` - The name of the configuration that exists in the IoT hub.

`--hub-name` - Name of the IoT hub in which the configuration exists. The hub must be in the current subscription. Switch to the desired subscription with the command `az account set -s [subscription name]`

# Device Configuration Best Practices

Device configuration best practices can be instrumental to successful device management. Best practices apply to the following roles:

- **IoT hardware manufacturer/integrator:** Manufacturers of IoT hardware, integrators assembling hardware from various manufacturers, or suppliers providing hardware for an IoT deployment manufactured or integrated by other suppliers. Involved in development and integration of firmware, embedded operating systems, and embedded software.
- **IoT solution developer:** The development of an IoT solution is typically done by a solution developer. This developer may be part of an in-house team or a system integrator specializing in this activity. The IoT solution developer can develop various components of the IoT solution from scratch, or integrate various standard or open-source components.
- **IoT solution operator:** After the IoT solution is deployed, it requires long-term operations, monitoring, upgrades, and maintenance. These tasks can be done by an in-house team that consists of information technology specialists, hardware operations and maintenance teams, and domain specialists who monitor the correct behavior of the overall IoT infrastructure.

## IoT hardware manufacturer/integrator

The following are best practices for hardware manufacturers and integrators dealing with embedded software development:

- Implement device twins: Device twins enable synchronizing desired configuration from the cloud and for reporting current configuration and device properties. The best way to implement device twins within embedded applications is through the Azure IoT SDKs. Device twins are best suited for configuration because they:
  - Support bi-directional communication.
  - Allow for both connected and disconnected device states.
  - Follow the principle of eventual consistency.
  - Are fully queriable in the cloud.
- Structure the device twin for device management: The device twin should be structured such that device management properties are logically grouped together into sections. Doing so will enable configuration changes to be isolated without impacting other sections of the twin. For example, create a section within desired properties for firmware, another section for software, and a third section for network settings.
- Report device attributes that are useful for device management: Attributes like physical device make and model, firmware, operating system, serial number, and other identifiers are useful for reporting and as parameters for targeting configuration changes.
- Define the main states for reporting status and progress: Top-level states should be enumerated so that they can be reported to the operator. For example, a firmware update would report status as Current, Downloading, Applying, In Progress, and Error. Define additional fields for more information on each state.

## IoT solution developer

The following are best practices for IoT solution developers who are building systems based in Azure:

- Implement device twins: All of the benefits for using listed for the manufacturer also apply to the solution developer.
- Organize devices using device twin tags: The solution should allow the operator to define quality rings or other sets of devices based on various deployment strategies such as canary. Device organization can be implemented within your solution using device twin tags and queries. Device organization is necessary to allow for configuration roll outs safely and accurately.
- Implement automatic device configurations: Automatic device configurations deploy and monitor configuration changes to large sets of IoT devices via device twins.

Automatic device configurations target sets of device twins via the target condition, which is a query on device twin tags or reported properties. The target content is the set of desired properties that will be set within the targeted device twins. The target content should align with the device twin structure defined by the IoT hardware manufacturer/integrator. The metrics are queries on device twin reported properties and should also align with the device twin structure defined by the IoT hardware manufacturer/integrator.

Automatic device configurations run for the first time shortly after the configuration is created and then at five minute intervals. They also benefit from the IoT Hub performing device twin operations at a rate that will never exceed the throttling limits for device twin reads and updates.

- Use the Device Provisioning Service: Solution developers should use the Device Provisioning Service to assign device twin tags to new devices, such that they will be automatically configured by automatic device configurations that are targeted at twins with that tag.

## IoT solution operator

The following are best practices for IoT solution operators who using an IoT solution built on Azure:

- Organize devices for management: The IoT solution should define or allow for the creation of quality rings or other sets of devices based on various deployment strategies such as canary. The sets of devices will be used to roll out configuration changes and to perform other at-scale device management operations.
- Perform configuration changes using a phased roll out: A phased roll out is an overall process whereby an operator deploys changes to a broadening set of IoT devices. The goal is to make changes gradually to reduce the risk of making wide scale breaking changes. The operator should use the solution's interface to create an automatic device configuration and the targeting condition should target an initial set of devices (such as a canary group). The operator should then validate the configuration change in the initial set of devices.

Once validation is complete, the operator will update the automatic device configuration to include a larger set of devices. The operator should also set the priority for the configuration to be higher than other configurations currently targeted to those devices. The roll out can be monitored using the metrics reported by the automatic device configuration.

- Perform rollbacks in case of errors or misconfigurations: An automatic device configuration that causes errors or misconfigurations can be rolled back by changing the targeting condition so that the devices no longer meet the targeting condition. Ensure that another automatic device configuration of lower priority is still targeted for those devices. Verify that the rollback succeeded by viewing the metrics: The rolled-back configuration should no longer show status for untargeted devices, and the second configuration's metrics should now include counts for the devices that are still targeted.

# About the Module 8 Labs

## Module Labs

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

- Lab 15: Manage Devices using Device Twins and Direct Methods
- Lab 16: Implement Automatic Device Management

**WARNING:** Be prepared for UI changes

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

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



## Module 9 Solution Testing, Diagnostics, and Logging

### Monitoring and Logging

#### Azure Monitor - Alerts and Log Analytics

The Azure IoT Hub service uses Azure Monitor to provide support for Alerts and Log Analytics.

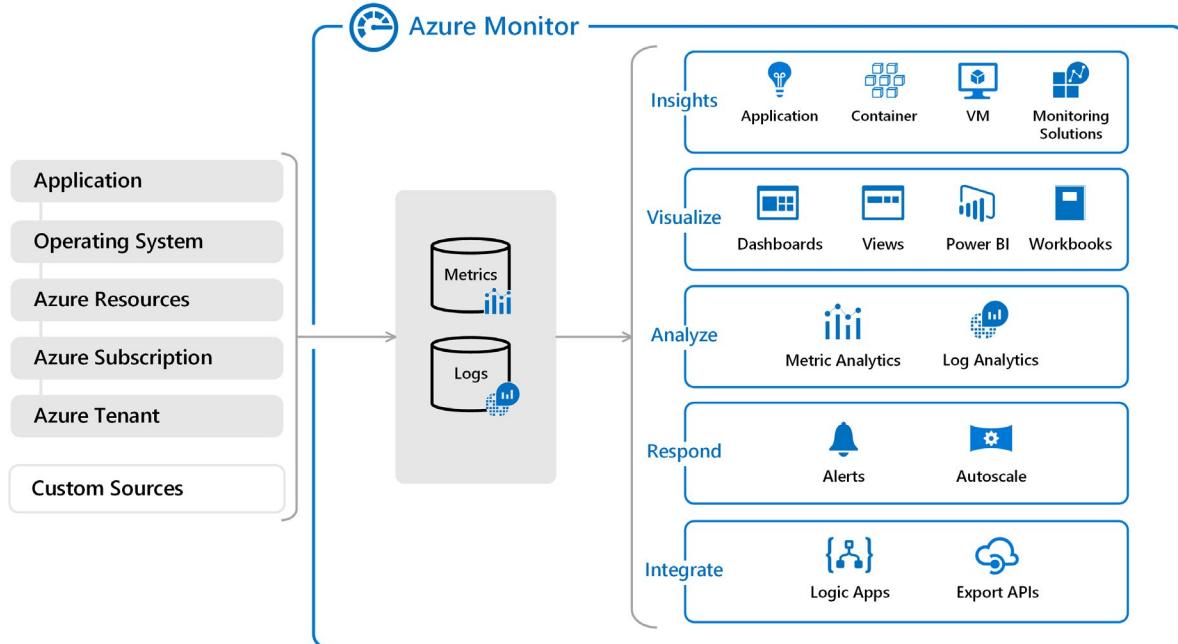
Azure Monitor maximizes the availability and performance of your applications and services by delivering a comprehensive solution for collecting, analyzing, and acting on telemetry from your cloud and on-premises environments. It helps you understand how your applications are performing and proactively identifies issues affecting them and the resources they depend on.

Just a few examples of what you can do with Azure Monitor include:

- Detect and diagnose issues across applications and dependencies with Application Insights.
- Correlate infrastructure issues with Azure Monitor for VMs and Azure Monitor for Containers.
- Drill into your monitoring data with Log Analytics for troubleshooting and deep diagnostics.
- Support operations at scale with smart alerts and automated actions.
- Create visualizations with Azure dashboards and workbooks.

### Overview

The following diagram gives a high-level view of Azure Monitor. At the center of the diagram are the data stores for metrics and logs, which are the two fundamental types of data used by Azure Monitor. On the left are the sources of monitoring data that populate these data stores. On the right are the different functions that Azure Monitor performs with this collected data such as analysis, alerting, and streaming to external systems.



## Monitoring data platform

All data collected by Azure Monitor fits into one of two fundamental types, metrics and logs. Metrics are numerical values that describe some aspect of a system at a particular point in time. They are lightweight and capable of supporting near real-time scenarios. Logs contain different kinds of data organized into records with different sets of properties for each type. Telemetry such as events and traces are stored as logs in addition to performance data so that it can all be combined for analysis.

For many Azure resources, you'll see data collected by Azure Monitor right in their Overview page in the Azure portal. The Azure IoT Hub Overview page includes charts for **Device twin operation** and **Device to cloud messages**. Click on any of the graphs to open the data in metrics explorer in the Azure portal, which allows you to chart the values of multiple metrics over time. You can view the charts interactively or pin them to a dashboard to view them with other visualizations.

Log data collected by Azure Monitor can be analyzed with queries to quickly retrieve, consolidate, and analyze collected data. You can create and test queries using Log Analytics in the Azure portal and then either directly analyze the data using these tools or save queries for use with visualizations or alert rules.

Azure Monitor uses a version of the Kusto query language used by Azure Data Explorer that is suitable for simple log queries but also includes advanced functionality such as aggregations, joins, and smart analytics. You can quickly learn the query language using multiple lessons. Particular guidance is provided to users who are already familiar with SQL and Splunk.

## What data does Azure Monitor collect?

Azure Monitor can collect data from a variety of sources. You can think of monitoring data for your applications in tiers ranging from your application, any operating system and services it relies on, down to the platform itself. Azure Monitor collects data from each of the following tiers:

- Application monitoring data: Data about the performance and functionality of the code you have written, regardless of its platform.

- Guest OS monitoring data: Data about the operating system on which your application is running. This could be running in Azure, another cloud, or on-premises.
- Azure resource monitoring data: Data about the operation of an Azure resource.
- Azure subscription monitoring data: Data about the operation and management of an Azure subscription, as well as data about the health and operation of Azure itself.
- Azure tenant monitoring data: Data about the operation of tenant-level Azure services, such as Azure Active Directory.

As soon as you create an Azure subscription and start adding resources such as virtual machines and web apps, Azure Monitor starts collecting data. Activity logs record when resources are created or modified. Metrics tell you how the resource is performing and the resources that it's consuming.

Extend the data you're collecting into the actual operation of the resources by enabling diagnostics and adding an agent to compute resources. This will collect telemetry for the internal operation of the resource and allow you to configure different data sources to collect logs and metrics from Windows and Linux guest operating system.

## Responding to critical situations

In addition to allowing you to interactively analyze monitoring data, an effective monitoring solution must be able to proactively respond to critical conditions identified in the data that it collects. This could be sending a text or mail to an administrator responsible for investigating an issue. Or you could launch an automated process that attempts to correct an error condition.

## Alerts

Alerts in Azure Monitor proactively notify you of critical conditions and potentially attempt to take corrective action. Alert rules based on metrics provide near real time alerting based on numeric values, while rules based on logs allow for complex logic across data from multiple sources.

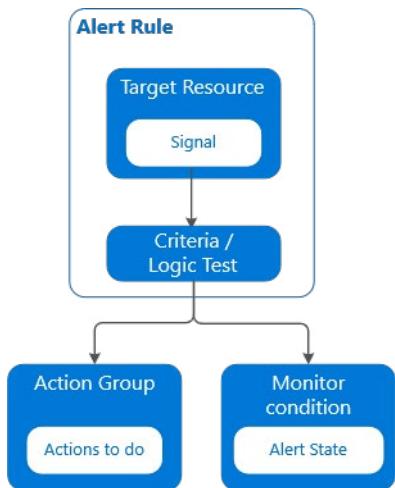
Alert rules in Azure Monitor use action groups, which contain unique sets of recipients and actions that can be shared across multiple rules. Based on your requirements, action groups can perform such actions as using webhooks to have alerts start external actions or to integrate with your ITSM tools.

The unified alert experience in Azure Monitor includes alerts that were previously managed by Log Analytics and Application Insights. In the past, Azure Monitor, Application Insights, Log Analytics, and Service Health had separate alerting capabilities. Over time, Azure improved and combined both the user interface and different methods of alerting. The consolidation is still in process.

**Note:** You can view classic alerts only in the classic alerts user screen in the Azure Portal. You get this screen from the View classic alerts button on the Alerts blade of IoT Hub in the Azure portal.

## Overview of Alerts in Azure

The diagram below represents the flow of alerts.



Alert rules are separated from alerts and the actions taken when an alert fires. The alert rule captures the target and criteria for alerting. The alert rule can be in an enabled or a disabled state. Alerts only fire when enabled.

The following are key attributes of an alert rule:

- Target Resource: Defines the scope and signals available for alerting. A target can be any Azure resource. Example targets: a virtual machine, a storage account, a virtual machine scale set, a Log Analytics workspace, or an Application Insights resource. For certain resources (like virtual machines), you can specify multiple resources as the target of the alert rule.
- Signal: Emitted by the target resource. Signals can be of the following types: metric, activity log, Application Insights, and log.
- Criteria: A combination of signal and logic applied on a target resource. Examples:
  - Percentage CPU > 70%
  - Server Response Time > 4 ms
  - Result count of a log query > 100
- Alert Name: A specific name for the alert rule configured by the user.
- Alert Description: A description for the alert rule configured by the user.
- Severity: The severity of the alert after the criteria specified in the alert rule is met. Severity can range from 0 to 4.
  - Sev 0 = Critical
  - Sev 1 = Error
  - Sev 2 = Warning
  - Sev 3 = Informational
  - Sev 4 = Verbose
- Action: A specific action taken when the alert is fired.

## What You Can Alert On

You can alert on metrics and logs. These include but are not limited to:

- Metric values
- Log search queries
- Activity log events
- Health of the underlying Azure platform
- Tests for website availability

With the consolidation of alerting services still in process, there are some alerting capabilities that are not yet in the new alerts system.

Monitor source	Signal type	Description
Service health	Activity log	Not supported. See Create activity log alerts on service notifications.
Application Insights	Web availability tests	Not supported. See Web test alerts. Available to any website that's instrumented to send data to Application Insights. Receive a notification when availability or responsiveness of a website is below expectations.

## IoT Hub Metrics

IoT Hub metrics provide you with access to data that describes the state of the Azure IoT resources in your Azure subscription. IoT Hub metrics enable you to assess the overall health of the IoT Hub service and the devices connected to it. User-facing statistics are important because they help you see what is going on with your IoT hub and help root-cause issues without needing to contact Azure support.

Metrics are enabled by default.

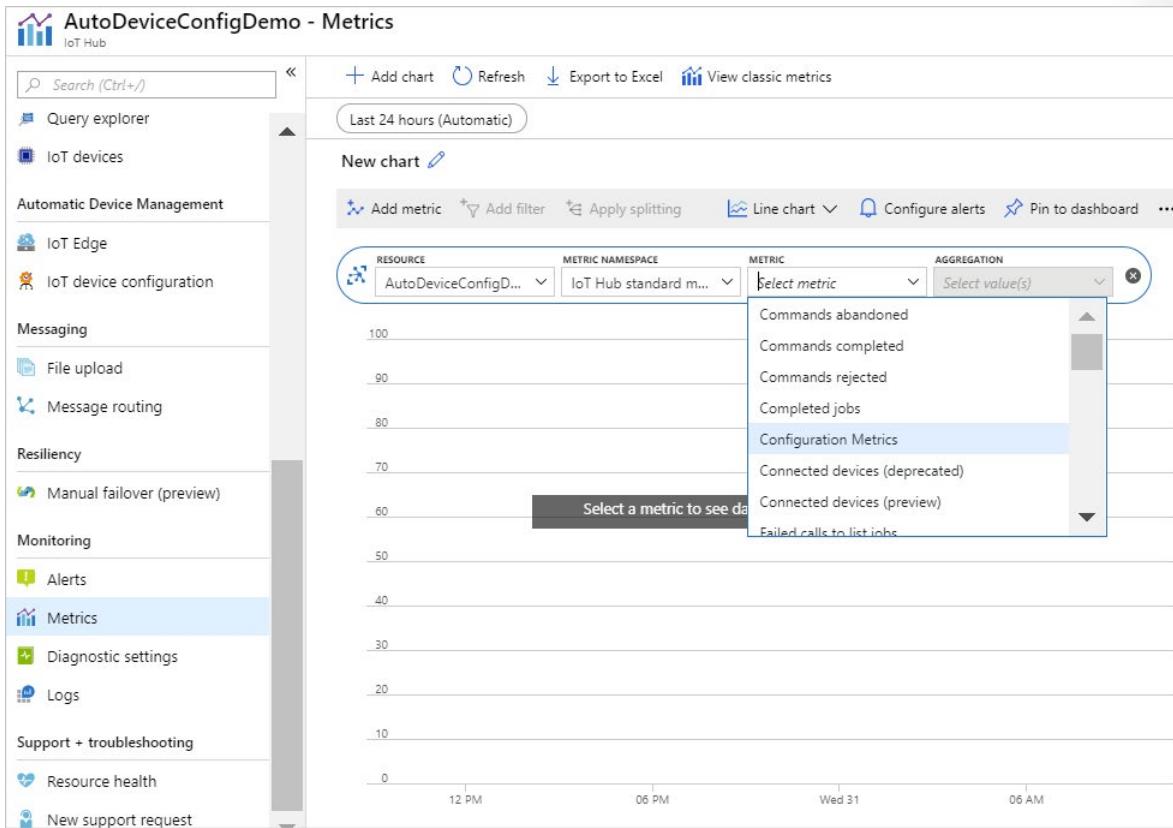
**Note:** You can use IoT Hub metrics to view information about IoT Plug and Play devices connected to your IoT Hub. IoT Plug and Play devices are part of the IoT Plug and Play public preview <https://docs.microsoft.com/en-us/azure/iot-pnp/overview-iot-plug-and-play>.

## How to View IoT Hub Metrics

On the Overview blade of your IoT hub, click **Metrics**.

The screenshot shows the Azure IoT Hub Metrics blade for the 'AutoDeviceConfigDemo' hub. The left sidebar lists navigation options: Automatic Device Management (IoT Edge, IoT device configuration), Messaging (File upload, Message routing), Resiliency (Manual failover (preview)), Monitoring (Alerts, Metrics, Diagnostic settings, Logs), Support + troubleshooting (Resource health, New support request). The 'Metrics' option is highlighted with a red circle. The main content area displays hub details: Resource group (AutoDeviceConfigDemo), Hostname (AutoDeviceConfigDemo.azure-devices.net), Status (Active), Location (West Central US), Pricing and scale tier (S1 - Standard), Number of IoT Hub units (1). Below these details are sections for Subscription (change <subscription name>) and Tags (change Click here to add tags). Two promotional cards are shown: one about IoT Hub Device Provisioning Service and another about learning more about IoT Hub.

On the Metrics blade, you can view the metrics for your IoT hub and create custom views of your metrics.



To send your metrics data to an Event Hubs endpoint or an Azure Storage account, click **Diagnostics settings**, then click **Add diagnostic setting**

## IoT Hub Metrics and How to Use Them

IoT Hub provides metrics that give you an overview of the health of your hub and the total number of connected devices, as well as metrics that give you specific details related to key operations. You can combine information from multiple metrics to paint a bigger picture of the state of the IoT hub. The following tables describes the metrics each IoT hub tracks, and how each metric relates to the overall status of the IoT hub.

## D2C Telemetry Ingress and Egress

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.telemetry.ingress.allProtocol	Telemetry message send attempts	Count	Total	Number of device-to-cloud telemetry messages attempted to be sent to your IoT hub	No Dimensions

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.telemetry.ingress.success	Telemetry messages sent	Count	Total	Number of device-to-cloud telemetry messages sent successfully to your IoT hub	No Dimensions
d2c.telemetry.ingress.sendThrottle	Number of throttling errors	Count	Total	Number of throttling errors due to device throughput throttles	No Dimensions
d2c.telemetry.egress.success	Routing: telemetry messages delivered	Count	Total	The number of times messages were successfully delivered to all endpoints using IoT Hub routing. If a message is routed to multiple endpoints, this value increases by one for each successful delivery. If a message is delivered to the same endpoint multiple times, this value increases by one for each successful delivery.	No Dimensions

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.telemetry.egress.dropped	Routing: telemetry messages dropped	Count	Total	The number of times messages were dropped by IoT Hub routing due to dead endpoints. This value does not count messages delivered to fallback route as dropped messages are not delivered there.	No Dimensions
d2c.telemetry.egress.orphaned	Routing: telemetry messages orphaned	Count	Total	The number of times messages were orphaned by IoT Hub routing because they didn't match any routing rules (including the fallback rule).	No Dimensions
d2c.telemetry.egress.invalid	Routing: telemetry messages incompatible	Count	Total	The number of times IoT Hub routing failed to deliver messages due to an incompatibility with the endpoint. This value does not include retries.	No Dimensions
d2c.telemetry.egress.fallback	Routing: messages delivered to fallback	Count	Total	The number of times IoT Hub routing delivered messages to the endpoint associated with the fallback route.	No Dimensions

**MCT USE ONLY. STUDENT USE PROHIBITED**

## C2D Commands and Methods

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
c2d.commands.egress.complete.success	Commands completed	Count	Total	Number of cloud-to-device commands completed successfully by the device	No Dimensions
c2d.commands.egress.abandon.success	Commands abandoned	Count	Total	Number of cloud-to-device commands abandoned by the device	No Dimensions
c2d.commands.egress.reject.success	Commands rejected	Count	Total	Number of cloud-to-device commands rejected by the device	No Dimensions
c2d.methods.success	Successful direct method invocations	Count	Total	The count of all successful direct method calls.	No Dimensions
c2d.methods.failure	Failed direct method invocations	Count	Total	The count of all failed direct method calls.	No Dimensions
c2d.methods.requestSize	Request size of direct method invocations	Bytes	Average	The average, min, and max of all successful direct method requests.	No Dimensions
c2d.methods.responseSize	Response size of direct method invocations	Bytes	Average	The average, min, and max of all successful direct method responses.	No Dimensions

## Hub and Devices

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
dailyMessageQuotaUsed	Total number of messages used	Count	Average	Number of total messages used today. This is a cumulative value that is reset to zero at 00:00 UTC every day.	No Dimensions
deviceData-Usage	Total device data usage	Bytes	Total	Bytes transferred to and from any devices connected to IoT Hub	No Dimensions
totalDevice-Count	Total devices (preview)	Count	Average	Number of devices registered to your IoT hub	No Dimensions
connectedDeviceCount	Connected devices (preview)	Count	Average	Number of devices connected to your IoT hub	No Dimensions
configurations	Configuration Metrics	Count	Total	Metrics for Configuration Operations	No Dimensions
devices.totalDevices	Total devices (deprecated)	Count	Total	Number of devices registered to your IoT hub	No Dimensions
devices.connectedDevices.allProtocol	Connected devices (deprecated)	Count	Total	Number of devices connected to your IoT hub	No Dimensions

## D2C Endpoints

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.endpoints.egress.eventHubs	Routing: messages delivered to Event Hub	Count	Total	The number of times IoT Hub routing successfully delivered messages to Event Hub endpoints.	No Dimensions
d2c.endpoints.latency.eventHubs	Routing: message latency for Event Hub	Milliseconds	Average	The average latency (milliseconds) between message ingress to IoT Hub and message ingress into an Event Hub endpoint.	No Dimensions
d2c.endpoints.egress.serviceBusQueues	Routing: messages delivered to Service Bus Queue	Count	Total	The number of times IoT Hub routing successfully delivered messages to Service Bus queue endpoints.	No Dimensions
d2c.endpoints.latency.serviceBusQueues	Routing: message latency for Service Bus Queue	Milliseconds	Average	The average latency (milliseconds) between message ingress to IoT Hub and telemetry message ingress into a Service Bus queue endpoint.	No Dimensions

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.endpoints.egress.serviceBusTopics	Routing: messages delivered to Service Bus Topic	Count	Total	The number of times IoT Hub routing successfully delivered messages to Service Bus topic endpoints.	No Dimensions
d2c.endpoints.latency.serviceBusTopics	Routing: message latency for Service Bus Topic	Milliseconds	Average	The average latency (milliseconds) between message ingress to IoT Hub and telemetry message ingress into a Service Bus topic endpoint.	No Dimensions
d2c.endpoints.egress.builtIn.events	Routing: messages delivered to messages/events	Count	Total	The number of times IoT Hub routing successfully delivered messages to the built-in endpoint (messages/events). This metric only starts working when routing is enabled for the IoT hub.	No Dimensions

MCT USE ONLY. STUDENT USE PROHIBITED

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.endpoints.latency.builtin.events	Routing: message latency for messages/events	Milliseconds	Average	The average latency (milliseconds) between message ingress to IoT Hub and telemetry message ingress into the built-in endpoint (messages/events). This metric only starts working when routing is enabled for the IoT hub.	No Dimensions
d2c.endpoints.egress.storage	Routing: messages delivered to storage	Count	Total	The number of times IoT Hub routing successfully delivered messages to storage endpoints.	No Dimensions
d2c.endpoints.latency.storage	Routing: message latency for storage	Milliseconds	Average	The average latency (milliseconds) between message ingress to IoT Hub and telemetry message ingress into a storage endpoint.	No Dimensions
d2c.endpoints.egress.storage.bytes	Routing: data delivered to storage	Bytes	Total	The amount of data (bytes) IoT Hub routing delivered to storage endpoints.	No Dimensions

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.endpoints.egress.storage.blobs	Routing: blobs delivered to storage	Count	Total	The number of times IoT Hub routing delivered blobs to storage endpoints.	No Dimensions

## Event Grid

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
EventGridDeliveries	Event Grid deliveries (preview)	Count	Total	The number of IoT Hub events published to Event Grid. Use the Result dimension for the number of successful and failed requests. EventType dimension shows the type of event. To see the where the requests come from, use the EventType dimension.	Result, Event-Type

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
EventGridLatency	Event Grid latency (preview)	Milliseconds	Average	The average latency (milliseconds) from when the IoT Hub event was generated to when the event was published to Event Grid. This number is an average between all event types. Use the EventType dimension to see latency of a specific type of event.	EventType

## Device Twins

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.twin.read.success	Successful twin reads from devices	Count	Total	The count of all successful device-initiated twin reads.	No Dimensions
d2c.twin.read.failure	Failed twin reads from devices	Count	Total	The count of all failed device-initiated twin reads.	No Dimensions
d2c.twin.read.size	Response size of twin reads from devices	Bytes	Average	The average, min, and max of all successful device-initiated twin reads.	No Dimensions
d2c.twin.update.success	Successful twin updates from devices	Count	Total	The count of all successful device-initiated twin updates.	No Dimensions
d2c.twin.update.failure	Failed twin updates from devices	Count	Total	The count of all failed device-initiated twin updates.	No Dimensions

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
d2c.twin.update.size	Size of twin updates from devices	Bytes	Average	The average, min, and max size of all successful device-initiated twin updates.	No Dimensions
c2d.twin.read.success	Successful twin reads from back end	Count	Total	The count of all successful back-end-initiated twin reads. This count doesn't include twin reads initiated from twin queries.	No Dimensions
c2d.twin.read.failure	Failed twin reads from back end	Count	Total	The count of all failed back-end-initiated twin reads.	No Dimensions
c2d.twin.read.size	Response size of twin reads from back end	Bytes	Average	The average, min, and max of all successful back-end-initiated twin reads.	No Dimensions
c2d.twin.update.success	Successful twin updates from back end	Count	Total	The count of all successful back-end-initiated twin updates.	No Dimensions
c2d.twin.update.failure	Failed twin updates from back end	Count	Total	The count of all failed back-end-initiated twin updates.	No Dimensions
c2d.twin.update.size	Size of twin updates from back end	Bytes	Average	The average, min, and max size of all successful back-end-initiated twin updates.	No Dimensions
twinQueries.success	Successful twin queries	Count	Total	The count of all successful twin queries.	No Dimensions

MCT USE ONLY. STUDENT USE PROHIBITED

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
twinQueries.failure	Failed twin queries	Count	Total	The count of all failed twin queries.	No Dimensions
twinQueries.resultSize	Twin queries result size	Bytes	Average	The average, min, and max of the result size of all successful twin queries.	No Dimensions

## IoT Hub Jobs

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
jobs.createTwinUpdateJob.success	Successful creations of twin update jobs	Count	Total	The count of all successful creation of twin update jobs.	No Dimensions
jobs.createTwinUpdateJob.failure	Failed creations of twin update jobs	Count	Total	The count of all failed creation of twin update jobs.	No Dimensions
jobs.createDirectMethodJob.success	Successful creations of method invocation jobs	Count	Total	The count of all successful creation of direct method invocation jobs.	No Dimensions
jobs.createDirectMethodJob.failure	Failed creations of method invocation jobs	Count	Total	The count of all failed creation of direct method invocation jobs.	No Dimensions
jobs.listJobs.success	Successful calls to list jobs	Count	Total	The count of all successful calls to list jobs.	No Dimensions
jobs.listJobs.failure	Failed calls to list jobs	Count	Total	The count of all failed calls to list jobs.	No Dimensions
jobs.cancelJob.success	Successful job cancellations	Count	Total	The count of all successful calls to cancel a job.	No Dimensions
jobs.cancelJob.failure	Failed job cancellations	Count	Total	The count of all failed calls to cancel a job.	No Dimensions

Metric	Metric Display Name	Unit	Aggregation Type	Description	Dimensions
jobs.queryJobs.success	Successful job queries	Count	Total	The count of all successful calls to query jobs.	No Dimensions
jobs.queryJobs.failure	Failed job queries	Count	Total	The count of all failed calls to query jobs.	No Dimensions
jobs.completed	Completed jobs	Count	Total	The count of all completed jobs.	No Dimensions
jobs.failed	Failed jobs	Count	Total	The count of all failed jobs.	No Dimensions

## Set Up Diagnostics Logs

Businesses that implement Azure IoT Hub expect reliable performance from their resources. To help you maintain a close watch on your operations, IoT Hub is fully integrated with Azure Monitor and Azure Resource Health. These two services work to provide you with the data you need to keep your IoT solutions up and running in a healthy state.

Azure Monitor is a single source of monitoring and logging for all your Azure services. You can send the diagnostic logs that Azure Monitor generates to Azure Monitor logs, Event Hubs, or Azure Storage for custom processing. Azure Monitor's metrics and diagnostics settings give you visibility into the performance of your resources. Continue reading this article to learn how to Use Azure Monitor with your IoT hub.

**Important:** The events emitted by the IoT Hub service using Azure Monitor diagnostic logs are not guaranteed to be reliable or ordered. Some events might be lost or delivered out of order. Diagnostic logs also aren't meant to be real-time, and it may take several minutes for events to be logged to your choice of destination.

Azure Resource Health helps you diagnose and get support when an Azure issue impacts your resources. A dashboard provides current and past health status for each of your IoT hubs.

## Use Azure Monitor

Azure Monitor provides diagnostics information for Azure resources, which means that you can monitor operations that take place within your IoT hub.

Azure Monitor's diagnostics settings replaces the IoT Hub operations monitor. If you currently use operations monitoring, you should migrate your workflows. For more information, see Migrate from operations monitoring to diagnostics settings <https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-migrate-to-diagnostics-settings>.

To learn more about the specific metrics and events that Azure Monitor watches, see Supported metrics with Azure Monitor <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/metrics-supported> and Supported services, schemas, and categories for Azure Diagnostic Logs <https://docs.microsoft.com/en-us/azure/azure-monitor/platform/metrics-supported>.

Choose where you want to send the logs. You can select any combination of the three options:

- Archive to a storage account
- Stream to an event hub

- Send to Log Analytics **see pricing:** <https://azure.microsoft.com/en-us/pricing/details/monitor/><sup>1</sup>

Choose which operations you want to monitor, and enable logs for those operations. The operations that diagnostic settings can report on are:

- Connections
- DeviceTelemetry
- C2DCommands
- DeviceIdentityOperations
- FileUploadOperations
- Routes
- D2CTwinOperations
- C2DTwinOperations
- TwinQueries
- JobsOperations
- DirectMethods
- DistributedTracing
- Configurations
- DeviceStreams
- AllMetrics

If you want to turn on diagnostics settings with PowerShell, use the following code:

```
Connect-AzAccount  
Select-AzSubscription -SubscriptionName <subscription that includes your IoT Hub>  
Set-AzDiagnosticSetting -ResourceId <your resource Id> -ServiceBusRuleId <your service bus rule Id>  
-Enabled $true
```

New settings take effect in about 10 minutes. After that, logs appear in the configured archival target on the Diagnostics settings blade.

## Understand the logs

Azure Monitor tracks different operations that occur in IoT Hub. Each category has a schema that defines how events in that category are reported.

## Connections

The connections category tracks device connect and disconnect events from an IoT hub as well as errors. This category is useful for identifying unauthorized connection attempts and or alerting when you lose connection to devices.

**Note:** For reliable connection status of devices check Device heartbeat.

```
{  
  "records":
```

<sup>1</sup> <https://azure.microsoft.com/en-us/pricing/details/monitor/>

```
[
  {
    "time": " UTC timestamp",
    "resourceId": "Resource Id",
    "operationName": "deviceConnect",
    "category": "Connections",
    "level": "Information",
    "properties": "{\"deviceId\":\"<deviceId>\",\"protocol\":<protocol>,\"authType\":\"{\\"-scope\\\":\\\"device\\\",\\\"type\\\":\\\"sas\\\",\\\"issuer\\\":\\\"iothub\\\",\\\"acceptingIpFilter-Rule\\\":null\\\"},\"maskedIpAddress\":\"<maskedIpAddress>\"}",
    "location": "Resource location"
  }
]
}
```

## Cloud-to-device commands

The cloud-to-device commands category tracks errors that occur at the IoT hub and are related to the cloud-to-device message pipeline. This category includes errors that occur from:

- Sending cloud-to-device messages (like unauthorized sender errors),
- Receiving cloud-to-device messages (like delivery count exceeded errors), and
- Receiving cloud-to-device message feedback (like feedback expired errors).

This category does not catch errors when the cloud-to-device message is delivered successfully but then improperly handled by the device.

```
{
  "records":
  [
    {
      "time": " UTC timestamp",
      "resourceId": "Resource Id",
      "operationName": "messageExpired",
      "category": "C2DCommands",
      "level": "Error",
      "resultType": "Event status",
      "resultDescription": "MessageDescription",
      "properties": "{\"deviceId\":\"<deviceId>\",\"messageId\":\"<messageId>\",\"messageSizeInBytes\":\"<messageSize>\",\"protocol\":\"Amqp\",\"deliveryAcknowledgement\":\"<None, NegativeOnly, PositiveOnly, Full>\",\"deliveryCount\":\"0\", \"expiryTime\":\"<timestamp>\",\"timeInSystem\":\"<timeIn-System>\",\"ttl\":<ttl>, \"EventProcessedUtcTime\":\"<UTC timestamp>\",\"EventEnqueuedUtcTime\":\"<UTC timestamp>\",\"maskedIpAddress\":\"<maskedIpAddress>\",\"statusCode\":\"4XX\"}",
      "location": "Resource location"
    }
  ]
}
```

## Device identity operations

The device identity operations category tracks errors that occur when you attempt to create, update, or delete an entry in your IoT hub's identity registry. Tracking this category is useful for provisioning scenarios.

```
{
  "records": [
    {
      "time": "UTC timestamp",
      "resourceId": "Resource Id",
      "operationName": "get",
      "category": "DeviceIdentityOperations",
      "level": "Error",
      "resultType": "Event status",
      "resultDescription": "MessageDescription",
      "properties": "{\"maskedIpAddress\":\"<maskedIpAddress>\",\"deviceId\":\"<deviceId>\",\"statusCode\":\"4XX\"}",
      "location": "Resource location"
    }
  ]
}
```

## Routes

The message routing category tracks errors that occur during message route evaluation and endpoint health as perceived by IoT Hub. This category includes events such as:

- A rule evaluates to “undefined”,
- IoT Hub marks an endpoint as dead, or
- Any errors received from an endpoint.

This category does not include specific errors about the messages themselves (like device throttling errors), which are reported under the “device telemetry” category.

```
{
  "records": [
    {
      "time": "UTC timestamp",
      "resourceId": "Resource Id",
      "operationName": "endpointUnhealthy",
      "category": "Routes",
      "level": "Error",
      "properties": "{\"deviceId\": \"<deviceId>\",\"endpointName\":\"<endpointName>\",\"messageId\":<messageId>,\"details\":\"<errorDetails>\",\"routeName\":\"<routeName>\"}",
      "location": "Resource location"
    }
  ]
}
```

## Device telemetry

The device telemetry category tracks errors that occur at the IoT hub and are related to the telemetry pipeline. This category includes errors that occur when sending telemetry events (such as throttling) and receiving telemetry events (such as unauthorized reader). This category cannot catch errors caused by code running on the device itself.

```
{  
  "records":  
  [  
    {  
      "time": "UTC timestamp",  
      "resourceId": "Resource Id",  
      "operationName": "ingress",  
      "category": "DeviceTelemetry",  
      "level": "Error",  
      "resultType": "Event status",  
      "resultDescription": "MessageDescription",  
      "properties": "{\"deviceid\":\"<deviceId>\",\"batching\":\"0\", \"messageSizeInBytes\":<messageSizeInBytes>, \"EventProcessedUtcTime\":<UTC timestamp>, \"EventEnqueuedUtcTime\":<UTC timestamp>, \"partitionId\":\"1\"}",  
      "location": "Resource location"  
    }  
  ]  
}
```

# File upload operations

The file upload category tracks errors that occur at the IoT hub and are related to file upload functionality. This category includes:

- Errors that occur with the SAS URI, such as when it expires before a device notifies the hub of a completed upload.
  - Failed uploads reported by the device.
  - Errors that occur when a file is not found in storage during IoT Hub notification message creation.

This category cannot catch errors that directly occur while the device is uploading a file to storage.

```
{  
  "records":  
  [  
    {  
      "time": "UTC timestamp",  
      "resourceId": "Resource Id",  
      "operationName": "ingress",  
      "category": "FileUploadOperations",  
      "level": "Error",  
      "resultType": "Event status",  
      "resultDescription": "MessageDescription",  
      "durationMs": "1",  
      "properties": "{\"deviceid\":\"<deviceId>\",\"protocol\":\"<protocol>\",\"authType\":\"<\\\"\\\"-scope\\\"\\\" device\\\"\\\" type\\\"\\\":\"\\\"sas\\\"\\\",\"issuer\\\"\\\":\"\\\"iothub\\\"\\\",\"acceptingIpFilter-"}  
    }  
  ]  
}
```

```
Rule\\\" :null\\\" , "blobUri\\\" : "http://bloburi.com\\\"",  
    "location": "Resource location"  
  }  
]  
}
```

## Cloud-to-device twin operations

The cloud-to-device twin operations category tracks service-initiated events on device twins. These operations can include get twin, update or replace tags, and update or replace desired properties.

```
{  
  "records":  
  [  
    {  
      "time": "UTC timestamp",  
      "resourceId": "Resource Id",  
      "operationName": "read",  
      "category": "C2DTwinOperations",  
      "level": "Information",  
      "durationMs": "1",  
      "properties": "{\"deviceId\":\"<deviceId>\",\"sdkVersion\":\"<sdkVersion>\",\"message-  
Size\":\"<messageSize>\",\",  
      "location": "Resource location"  
    }  
  ]  
}
```

## Device-to-cloud twin operations

The device-to-cloud twin operations category tracks device-initiated events on device twins. These operations can include get twin, update reported properties, and subscribe to desired properties.

```
{  
  "records":  
  [  
    {  
      "time": "UTC timestamp",  
      "resourceId": "Resource Id",  
      "operationName": "update",  
      "category": "D2CTwinOperations",  
      "level": "Information",  
      "durationMs": "1",  
      "properties": "{\"deviceId\":\"<deviceId>\",\"protocol\":\"<protocol>\",\"authentication-  
Type\":\"\\\\\"scope\\\\\\\\\"\\\\\\\\\"device\\\\\\\\\"\\\\\\\\\"type\\\\\\\\\"\\\\\\\\\"sas\\\\\\\\\"\\\\\\\\\"issuer\\\\\\\\\"\\\\\\\\\"iothub\\\\\\\\\"\\\\\\\\\"acceptingIpFil-  
terRule\\\\\\\\\" :null\\\\\\\\\"\",  
      "location": "Resource location"  
    }  
  ]  
}
```

```
}
```

## Twin queries

The twin queries category reports on query requests for device twins that are initiated in the cloud.

```
{
  "records":
  [
    {
      "time": "UTC timestamp",
      "resourceId": "Resource Id",
      "operationName": "query",
      "category": "TwinQueries",
      "level": "Information",
      "durationMs": "1",
      "properties": "{\"query\":\"<twin query>\",\"sdkVersion\":\"<sdkVersion>\",\"messageSize\":\"<messageSize>\",\"pageSize\":\"<pageSize>\",\"continuation\":\"<true, false>\",\"resultSize\":\"<resultSize>\",\"location\": \"Resource location\"}",
      "location": "Resource location"
    }
  ]
}
```

## Jobs operations

The jobs operations category reports on job requests to update device twins or invoke direct methods on multiple devices. These requests are initiated in the cloud.

```
{
  "records":
  [
    {
      "time": "UTC timestamp",
      "resourceId": "Resource Id",
      "operationName": "jobCompleted",
      "category": "JobsOperations",
      "level": "Information",
      "durationMs": "1",
      "properties": "{\"jobId\":\"<jobId>\",\"sdkVersion\":\"<sdkVersion>\",\"messageSize\": <messageSize>, \"filter\":\"DeviceId IN ['1414ded9-b445-414d-89b9-e48e8c6285d5']\", \"start-TimeUtc\":\"Wednesday, September 13, 2017\", \"duration\":\"0\"}",
      "location": "Resource location"
    }
  ]
}
```

## Direct Methods

The direct methods category tracks request-response interactions sent to individual devices. These requests are initiated in the cloud.

```
{  
  "records":  
  [  
    {  
      "time": "UTC timestamp",  
      "resourceId": "Resource Id",  
      "operationName": "send",  
      "category": "DirectMethods",  
      "level": "Information",  
      "durationMs": "1",  
      "properties": "{\"deviceId\":<messageSize>, \"RequestSize\": 1, \"ResponseSize\": 1, \"sdkVersion\": \"2017-07-11\"}",  
      "location": "Resource location"  
    }  
  ]  
}
```

## Read logs from Azure Event Hubs

After you set up event logging through diagnostics settings, you can create applications that read out the logs so that you can take action based on the information in them. This sample code retrieves logs from an event hub:

```
class Program  
{  
  static string connectionString = "{your AMS eventhub endpoint connection string}";  
  static string monitoringEndpointName = "{your AMS event hub endpoint name}";  
  static EventHubClient eventHubClient;  
  //This is the Diagnostic Settings schema  
  class AzureMonitorDiagnosticLog  
  {  
    string time { get; set; }  
    string resourceId { get; set; }  
    string operationName { get; set; }  
    string category { get; set; }  
    string level { get; set; }  
    string resultType { get; set; }  
    string resultDescription { get; set; }  
    string durationMs { get; set; }  
    string callerIpAddress { get; set; }  
    string correlationId { get; set; }  
    string identity { get; set; }  
    string location { get; set; }  
    Dictionary<string, string> properties { get; set; }  
  };  
}
```

```
static void Main(string[] args)
{
    Console.WriteLine("Monitoring. Press Enter key to exit.\n");
    eventHubClient = EventHubClient.CreateFromConnectionString(connectionString, monitoringEndpointName);
    var d2cPartitions = eventHubClient.GetRuntimeInformationAsync().PartitionIds;
    CancellationTokenSource cts = new CancellationTokenSource();
    var tasks = new List<Task>();
    foreach (string partition in d2cPartitions)
    {
        tasks.Add(ReceiveMessagesFromDeviceAsync(partition, cts.Token));
    }
    Console.ReadLine();
    Console.WriteLine("Exiting... ");
    cts.Cancel();
    Task.WaitAll(tasks.ToArray());
}

private static async Task ReceiveMessagesFromDeviceAsync(string partition, CancellationToken ct)
{
    var eventHubReceiver = eventHubClient.GetDefaultConsumerGroup().CreateReceiver(partition,
    DateTime.UtcNow);
    while (true)
    {
        if (ct.IsCancellationRequested)
        {
            await eventHubReceiver.CloseAsync();
            break;
        }
        EventData eventData = await eventHubReceiver.ReceiveAsync(new TimeSpan(0,0,10));
        if (eventData != null)
        {
            string data = Encoding.UTF8.GetString(eventData.GetBytes());
            Console.WriteLine("Message received. Partition: {0} Data: '{1}'", partition, data);
            var deserializer = new JavaScriptSerializer();
            //deserialize json data to azure monitor object
            AzureMonitorDiagnosticLog message = new JavaScriptSerializer().Deserialize<AzureMonitorDiagnosticLog>(result);
        }
    }
}
```

## Use Azure Resource Health

Use Azure Resource Health to monitor whether your IoT hub is up and running. You can also learn whether a regional outage is impacting the health of your IoT hub. To understand specific details about the health state of your Azure IoT Hub, we recommend that you use Azure Monitor as described in the previous section.

Azure IoT Hub indicates health at a regional level. If a regional outage impacts your IoT hub, the health status shows as Unknown. To learn more, see Resource types and health checks in Azure resource health <https://docs.microsoft.com/en-us/azure/service-health/resource-health-checks-resource-types>.

To check the health of your IoT hubs, follow these steps:

- Sign in to the Azure portal.
- Navigate to Service Health > Resource health.
- From the drop-down boxes, select your subscription then select IoT Hub as the resource type.

To learn more about how to interpret health data, see Azure resource health overview <https://docs.microsoft.com/en-us/azure/service-health/resource-health-overview>.

## Device Connection State and Lifecycle Notifications

### Device heartbeat

The IoT Hub identity registry contains a field called **connectionState**. Only use the **connectionState** field during development and debugging. IoT solutions should not query the field at run time. For example, do not query the **connectionState** field to check if a device is connected before you send a cloud-to-device message or an SMS. We recommend subscribing to the device disconnected event on Event Grid to get alerts and monitor the device connection state.

If your IoT solution needs to know if a device is connected, you can implement the *heartbeat pattern*. In the heartbeat pattern, the device sends device-to-cloud messages at least once every fixed amount of time (for example, at least once every hour). Therefore, even if a device does not have any data to send, it still sends an empty device-to-cloud message (usually with a property that identifies it as a heartbeat). On the service side, the solution maintains a map with the last heartbeat received for each device. If the solution does not receive a heartbeat message within the expected time from the device, it assumes that there is a problem with the device.

A more complex implementation could include the information from Azure Monitor and Azure Resource Health to identify devices that are trying to connect or communicate but failing. When you implement the heartbeat pattern, make sure to check IoT Hub Quotas and Throttles.

**Note:** If an IoT solution uses the connection state solely to determine whether to send cloud-to-device messages, and messages are not broadcast to large sets of devices, consider using the simpler short expiry time pattern. This pattern achieves the same result as maintaining a device connection state registry using the heartbeat pattern, while being more efficient. If you request message acknowledgements, IoT Hub can notify you about which devices are able to receive messages and which are not.

### Device and Module Lifecycle Notifications

IoT Hub can notify your IoT solution when an identity is created or deleted by sending lifecycle notifications. To do so, your IoT solution needs to create a route and to set the Data Source equal to **DeviceLifecycleEvents** or **ModuleLifecycleEvents**. By default, no lifecycle notifications are sent, that is, no such routes pre-exist. The notification message includes properties, and body.

Properties: Message system properties are prefixed with the /\$ symbol.

Notification message for device:

Name	Value
\$content-type	application/json
\$iothub-enqueuedtime	Time when the notification was sent
\$iothub-message-source	deviceLifecycleEvents
\$content-encoding	utf-8
opType	createDeviceIdentity or deleteDeviceIdentity
hubName	Name of IoT Hub
deviceId	ID of the device
operationTimestamp	ISO8601 timestamp of operation
iothub-message-schema	deviceLifecycleNotification

Body: This section is in JSON format and represents the twin of the created device identity. For example,

```
{
  "deviceId": "11576-ailn-test-0-67333793211",
  "etag": "AAAAAAAAAAE=",
  "properties": {
    "desired": {
      "$metadata": {
        "$lastUpdated": "2016-02-30T16:24:48.789Z"
      },
      "$version": 1
    },
    "reported": {
      "$metadata": {
        "$lastUpdated": "2016-02-30T16:24:48.789Z"
      },
      "$version": 1
    }
  }
}
```

Notification message for module:

Name	Value
\$content-type	application/json
\$iothub-enqueuedtime	Time when the notification was sent
\$iothub-message-source	moduleLifecycleEvents
\$content-encoding	utf-8
opType	createModuleIdentity or deleteModuleIdentity
hubName	Name of IoT Hub
moduleId	ID of the module
operationTimestamp	ISO8601 timestamp of operation
iothub-message-schema	moduleLifecycleNotification

Body: This section is in JSON format and represents the twin of the created module identity. For example,

```
{
  "deviceId": "11576-ailn-test-0-67333793211",
  "moduleId": "tempSensor",
```

```

"etag": "AAAAAAAEE=",
"properties": {
    "desired": {
        "$metadata": {
            "$lastUpdated": "2016-02-30T16:24:48.789Z"
        },
        "$version": 1
    },
    "reported": {
        "$metadata": {
            "$lastUpdated": "2016-02-30T16:24:48.789Z"
        },
        "$version": 1
    }
}
}

```

## Device identity properties

Device identities are represented as JSON documents with the following properties:

Property	Options	Description
deviceId	required, read-only on updates	A case-sensitive string (up to 128 characters long) of ASCII 7-bit alphanumeric characters plus certain special characters: - . + % _ # * ? ! ( ) , = @ \$ '
generationId	required, read-only	An IoT hub-generated, case-sensitive string up to 128 characters long. This value is used to distinguish devices with the same <b>deviceId</b> , when they have been deleted and re-created.
etag	required, read-only	A string representing a weak ETag for the device identity, as per RFC7232.
auth	optional	A composite object containing authentication information and security materials.
auth.symkey	optional	A composite object containing a primary and a secondary key, stored in base64 format.
status	required	An access indicator. Can be <b>Enabled</b> or <b>Disabled</b> . If <b>Enabled</b> , the device is allowed to connect. If <b>Disabled</b> , this device cannot access any device-facing endpoint.

Property	Options	Description
statusReason	optional	A 128 character-long string that stores the reason for the device identity status. All UTF-8 characters are allowed.
statusUpdateTime	read-only	A temporal indicator, showing the date and time of the last status update.
connectionState	read-only	A field indicating connection status: either <b>Connected</b> or <b>Disconnected</b> . This field represents the IoT Hub view of the device connection status.  <b>Important:</b> This field should be used only for development/debugging purposes. The connection state is updated only for devices using MQTT or AMQP. Also, it is based on protocol-level pings (MQTT pings, or AMQP pings), and it can have a maximum delay of only 5 minutes. For these reasons, there can be false positives, such as devices reported as connected but that are disconnected.
connectionStateUpdatedTime	read-only	A temporal indicator, showing the date and last time the connection state was updated.
lastActivityTime	read-only	A temporal indicator, showing the date and last time the device connected, received, or sent a message.

**Note:** Connection state can only represent the IoT Hub view of the status of the connection. Updates to this state may be delayed, depending on network conditions and configurations.

**Note:** Currently the device SDKs do not support using the + and # characters in the deviceld.

## Module identity properties

Module identities are represented as JSON documents with the following properties:

Property	Options	Description
deviceld	required, read-only on updates	A case-sensitive string (up to 128 characters long) of ASCII 7-bit alphanumeric characters plus certain special characters: - . + % _ # * ? ! ( ) , = @ \$ '

Property	Options	Description
moduleId	required, read-only on updates	A case-sensitive string (up to 128 characters long) of ASCII 7-bit alphanumeric characters plus certain special characters: - . + % _ # * ? ! ( ) , = @ \$ '
generationId	required, read-only	An IoT hub-generated, case-sensitive string up to 128 characters long. This value is used to distinguish devices with the same <b>deviceld</b> , when they have been deleted and re-created.
etag	required, read-only	A string representing a weak ETag for the device identity, as per RFC7232.
auth	optional	A composite object containing authentication information and security materials.
auth.symkey	optional	A composite object containing a primary and a secondary key, stored in base64 format.
status	required	An access indicator. Can be <b>Enabled</b> or <b>Disabled</b> . If <b>Enabled</b> , the device is allowed to connect. If <b>Disabled</b> , this device cannot access any device-facing endpoint.
statusReason	optional	A 128 character-long string that stores the reason for the device identity status. All UTF-8 characters are allowed.
statusUpdateTime	read-only	A temporal indicator, showing the date and time of the last status update.

Property	Options	Description
connectionState	read-only	A field indicating connection status: either <b>Connected</b> or <b>Disconnected</b> . This field represents the IoT Hub view of the device connection status.  <b>Important:</b> This field should be used only for development/debugging purposes. The connection state is updated only for devices using MQTT or AMQP. Also, it is based on protocol-level pings (MQTT pings, or AMQP pings), and it can have a maximum delay of only 5 minutes. For these reasons, there can be false positives, such as devices reported as connected but that are disconnected.
connectionStateUpdatedTime	read-only	A temporal indicator, showing the date and last time the connection state was updated.
lastActivityTime	read-only	A temporal indicator, showing the date and last time the device connected, received, or sent a message.

**Note:** Currently the device SDKs do not support using the + and # characters in the **deviceId** and **moduleId**.

## Quotas and Throttling

Each IoT hub is provisioned with a certain number of units in a specific tier. The tier and number of units determine the maximum daily quota of messages that you can send. The message size used to calculate the daily quota is 0.5 KB for a free tier hub and 4KB for all other tiers.

The tier also determines the throttling limits that IoT Hub enforces on all operations.

**Note:** During public preview, IoT Plug and Play devices will send separate messages per interface, which may increase the number of messages counted towards your message quota.

## Operation throttles

Operation throttles are rate limitations that are applied in minute ranges and are intended to prevent abuse. They're also subject to traffic shaping.

The following table shows the enforced throttles. Values refer to an individual hub.

Throttle	Free, B1, and S1	B2 and S2	B3 and S3
Identity registry operations (create, retrieve, list, update, delete)	1.67/sec/unit (100/min/unit)	1.67/sec/unit (100/min/unit)	83.33/sec/unit (5,000/min/unit)
New device connections (this limit applies to the rate of new connections, not the total number of connections)	Higher of 100/sec or 12/sec/unit  For example, two S1 units are $2 \times 12 = 24$ new connections/sec, but you have at least 100 new connections/sec across your units. With nine S1 units, you have 108 new connections/sec ( $9 \times 12$ ) across your units.	120 new connections/sec/unit	6,000 new connections/sec/unit
Device-to-cloud sends	Higher of 100 send operations/sec or 12 send operations/sec/unit  For example, two S1 units are $2 \times 12 = 24$ /sec, but you have at least 100 send operations/sec across your units. With nine S1 units, you have 108 send operations/sec ( $9 \times 12$ ) across your units.	120 send operations/sec/unit	6,000 send operations/sec/unit
Cloud-to-device sends *	1.67 send operations/sec/unit (100 messages/min/unit)	1.67 send operations/sec/unit (100 send operations/min/unit)	83.33 send operations/sec/unit (5,000 send operations/min/unit)
Cloud-to-device receives * (only when device uses HTTPS)	16.67 receive operations/sec/unit (1,000 receive operations/min/unit)	16.67 receive operations/sec/unit (1,000 receive operations/min/unit)	833.33 receive operations/sec/unit (50,000 receive operations/min/unit)
File upload	1.67 file upload initiations/sec/unit (100/min/unit)	1.67 file upload initiations/sec/unit (100/min/unit)	83.33 file upload initiations/sec/unit (5,000/min/unit)
Direct methods *	160KB/sec/unit **	480KB/sec/unit **	24MB/sec/unit **
Queries	20/min/unit	20/min/unit	1,000/min/unit
Twin (device and module) reads *	100/sec	Higher of 100/sec or 10/sec/unit	500/sec/unit
Twin updates (device and module) *	50/sec	Higher of 50/sec or 5/sec/unit	250/sec/unit
Jobs operations * (create, update, list, delete)	1.67/sec/unit (100/min/unit)	1.67/sec/unit (100/min/unit)	83.33/sec/unit (5,000/min/unit)

Throttle	Free, B1, and S1	B2 and S2	B3 and S3
Jobs device operations * (update twin, invoke direct method)	10/sec	Higher of 10/sec or 1/sec/unit	50/sec/unit
Configurations and edge deployments * (create, update, list, delete)	0.33/sec/unit (20/min/unit)	0.33/sec/unit (20/min/unit)	0.33/sec/unit (20/min/unit)
Device stream initiation rate *	5 new streams/sec	5 new streams/sec	5 new streams/sec
Maximum number of concurrently connected device streams *	50	50	50
Maximum device stream data transfer1 (aggregate volume per day)	300 MB	300 MB	300 MB

\* This feature is not available in the basic tier of IoT Hub. For more information, see [How to choose the right IoT Hub](#).

\*\* Throttling meter size is 4 KB.

## Throttling Details

- The meter size determines at what increments your throttling limit is consumed. If your direct call's payload is between 0 and 4 KB, it is counted as 4 KB. You can make up to 40 calls per second per unit before hitting the limit of 160 KB/sec/unit.  
Similarly, if your payload is between 4 KB and 8 KB, each call accounts for 8 KB and you can make up to 20 calls per second per unit before hitting the max limit.  
Finally, if your payload size is between 156KB and 160 KB, you'll be able to make only 1 call per second per unit in your hub before hitting the limit of 160 KB/sec/unit.
- For Jobs device operations (update twin, invoke direct method) for tier S2, 50/sec/unit only applies to when you invoke methods using jobs. If you invoke direct methods directly, the original throttling limit of 24 MB/sec/unit (for S2) applies.
- Quota is the aggregate number of messages you can send in your hub per day. You can find your hub's quota limit under the column Total number of messages /day on the IoT Hub pricing page.
- Your cloud-to-device and device-to-cloud throttles determine the maximum rate at which you can send messages – number of messages irrespective of 4 KB chunks. Each message can be up to 256 KB which is the maximum message size.

It's a good practice to throttle your calls so that you don't hit/exceed the throttling limits. If you do hit the limit, IoT Hub responds with error code 429 and the client should back-off and retry. These limits are per hub (or in some cases per hub/unit). For more information, refer to [Manage connectivity and reliable messaging/Retry patterns](#).

## Traffic shaping

To accommodate burst traffic, IoT Hub accepts requests above the throttle for a limited time. The first few of these requests are processed immediately. However, if the number of requests continues violate the

MCT USE ONLY. STUDENT USE PROHIBITED

throttle, IoT Hub starts placing the requests in a queue and processed at the limit rate. This effect is called traffic shaping. Furthermore, the size of this queue is limited. If the throttle violation continues, eventually the queue fills up, and IoT Hub starts rejecting requests with `429 ThrottlingException`.

For example, you use a simulated device to send 200 device-to-cloud messages per second to your S1 IoT Hub (which has a limit of 100/sec D2C sends). For the first minute or two, the messages are processed immediately. However, since the device continues to send more messages than the throttle limit, IoT Hub begins to only process 100 messages per second and puts the rest in a queue. You start noticing increased latency. Eventually, you start getting `429 ThrottlingException` as the queue fills up, and the “number of throttle errors” in IoT Hub’s metrics starts increasing.

## Identity registry operations throttle

Device identity registry operations are intended for run-time use in device management and provisioning scenarios. Reading or updating a large number of device identities is supported through import and export jobs.

## Device connections throttle

The device connections throttle governs the rate at which new device connections can be established with an IoT hub. The device connections throttle does not govern the maximum number of simultaneously connected devices. The device connections rate throttle depends on the number of units that are provisioned for the IoT hub.

For example, if you buy a single S1 unit, you get a throttle of 100 connections per second. Therefore, to connect 100,000 devices, it takes at least 1,000 seconds (approximately 16 minutes). However, you can have as many simultaneously connected devices as you have devices registered in your identity registry.

## Other limits

IoT Hub enforces other operational limits:

Operation	Limit
Devices	The total number of devices plus modules that can be registered to a single IoT hub is capped at 1,000,000. The only way to increase this limit is to contact Microsoft Support.
File uploads	10 concurrent file uploads per device.
Jobs *	Maximum concurrent jobs is 1 (for Free and S1), 5 (for S2), and 10 (for S3). However, the max concurrent device import/export jobs is 1 for all tiers. Job history is retained up to 30 days.
Additional endpoints	Paid SKU hubs may have 10 additional endpoints. Free SKU hubs may have one additional endpoint.
Message routing queries	Paid SKU hubs may have 100 routing queries. Free SKU hubs may have five routing queries.
Message enrichments	Paid SKU hubs can have up to 10 message enrichments. Free SKU hubs can have up to 2 message enrichments.
Device-to-cloud messaging	Maximum message size 256 KB

Operation	Limit
Cloud-to-device messaging *	Maximum message size 64 KB. Maximum pending messages for delivery is 50 per device.
Direct method *	Maximum direct method payload size is 128 KB.
Automatic device configurations *	100 configurations per paid SKU hub. 20 configurations per free SKU hub.
IoT Edge automatic deployments *	20 modules per deployment. 100 deployments per paid SKU hub. 10 deployments per free SKU hub.
Twins *	Maximum size of desired properties and reported properties sections are 32 KB each. Maximum size of tags section is 8 KB.

\* This feature is not available in the basic tier of IoT Hub. For more information, see [How to choose the right IoT Hub](#).

## Increasing the quota or throttle limit

At any given time, you can increase quotas or throttle limits by increasing the number of provisioned units in an IoT hub.

## Latency

IoT Hub strives to provide low latency for all operations. However, due to network conditions and other unpredictable factors it cannot guarantee a certain latency. When designing your solution, you should:

- Avoid making any assumptions about the maximum latency of any IoT Hub operation.
- Provision your IoT hub in the Azure region closest to your devices.
- Consider using Azure IoT Edge to perform latency-sensitive operations on the device or on a gateway close to the device.

Multiple IoT Hub units affect throttling as described previously, but do not provide any additional latency benefits or guarantees.

# Troubleshooting

## Device-Side Connectivity Management

The connectivity and reliable messaging features in Azure IoT device SDKs can help you to design device applications that are more resilient.

Applying proper guidance to device-side code can help you to address the following scenarios:

- Fixing a dropped network connection
- Switching between different network connections
- Reconnecting because of service transient connection errors

### Designing for resiliency

IoT devices often rely on non-continuous or unstable network connections (for example, GSM or satellite). Errors can occur when devices interact with cloud-based services because of intermittent service availability and infrastructure-level or transient faults. An application that runs on a device has to manage the mechanisms for connection, re-connection, and the retry logic for sending and receiving messages. Also, the retry strategy requirements depend heavily on the device's IoT scenario, context, capabilities.

The Azure IoT Hub device SDKs aim to simplify connecting and communicating from cloud-to-device and device-to-cloud. These SDKs provide a robust way to connect to Azure IoT Hub and a comprehensive set of options for sending and receiving messages. Developers can also modify existing implementation to customize a better retry strategy for a given scenario.

### Connection and retry

This section gives an overview of the re-connection and retry patterns available when managing connections. It details implementation guidance for using a different retry policy in your device application and lists relevant APIs from the device SDKs.

### Error patterns

Connection failures can happen at many levels:

- Network errors: disconnected socket and name resolution errors
- Protocol-level errors for HTTP, AMQP, and MQTT transport: detached links or expired sessions
- Application-level errors that result from either local mistakes: invalid credentials or service behavior (for example, exceeding the quota or throttling)

The device SDKs detect errors at all three levels. OS-related errors and hardware errors are not detected and handled by the device SDKs.

### Retry patterns

The following steps describe the retry process when connection errors are detected:

1. The SDK detects the error and the associated error in the network, protocol, or application.
2. The SDK uses the error filter to determine the error type and decide if a retry is needed.

3. If the SDK identifies an unrecoverable error, operations like connection, send, and receive are stopped. The SDK notifies the user. Examples of unrecoverable errors include an authentication error and a bad endpoint error.
4. If the SDK identifies a recoverable error, it retries according to the specified retry policy until the defined timeout elapses. Note that the SDK uses Exponential back-off with jitter retry policy by default.
5. When the defined timeout expires, the SDK stops trying to connect or send. It notifies the user.
6. The SDK allows the user to attach a callback to receive connection status changes.

The SDKs provide three retry policies:

- Exponential back-off with jitter: This default retry policy tends to be aggressive at the start and slow down over time until it reaches a maximum delay. The design is based on Retry guidance from Azure Architecture Center.
- Custom retry: For some SDK languages, you can design a custom retry policy that is better suited for your scenario and then inject it into the `RetryPolicy`. Custom retry isn't available on the C SDK.
- No retry: You can set retry policy to "no retry," which disables the retry logic. The SDK tries to connect once and send a message once, assuming the connection is established. This policy is typically used in scenarios with bandwidth or cost concerns. If you choose this option, messages that fail to send are lost and can't be recovered.

## Retry policy APIs

SDK	SetRetryPolicy method	Policy implementations	Implementation guidance
C/iOS	IOTHUB_CLIENT_RESULT IoTHubClient_SetRetryPolicy	Default: IOTHUB_CLIENT_RETRY_EXPONENTIAL_BACKOFF Custom: use available <code>retryPolicy</code> No retry: IOTHUB_CLIENT_RETRY_NONE	C/iOS implementation
Java	SetRetryPolicy	Default: Exponential-BackoffWithJitter class Custom: implement <code>RetryPolicy</code> interface No retry: <code>NoRetry</code> class	Java implementation
.NET	DeviceClient.SetRetryPolicy	Default: Exponential-Backoff class Custom: implement <code>IRetryPolicy</code> interface No retry: <code>NoRetry</code> class	C# implementation
Node	setRetryPolicy	Default: Exponential-BackoffWithJitter class Custom: implement <code>RetryPolicy</code> interface No retry: <code>NoRetry</code> class	Node implementation
Python	Coming soon	Coming soon	Coming soon

The following C# code samples illustrate this flow:

## .NET implementation guidance

The following code sample shows how to define and set the default retry policy:

```
// define/set default retry policy  
IRetryPolicy retryPolicy = new ExponentialBackoff(int.MaxValue, TimeSpan.FromMilliseconds(100),  
TimeSpan.FromSeconds(10), TimeSpan.FromMilliseconds(100));  
SetRetryPolicy(retryPolicy);
```

To avoid high CPU usage, the retries are throttled if the code fails immediately. For example, when there's no network or route to the destination. The minimum time to execute the next retry is 1 second.

If the service responds with a throttling error, the retry policy is different and can't be changed via public API:

```
// throttled retry policy  
IRetryPolicy retryPolicy = new ExponentialBackoff(RetryCount, TimeSpan.FromSeconds(10),  
TimeSpan.FromSeconds(60), TimeSpan.FromSeconds(5)); SetRetryPolicy(retryPolicy);
```

The retry mechanism stops after DefaultOperationTimeoutInMilliseconds, which is currently set at 4 minutes.

## Troubleshooting Guide for D2C Communication Issues

The following issue troubleshooting checklists give you some things to try before you file a support ticket.

### Cannot connect to your Azure IoT hub

If your device doesn't seem to be able to connect to your Azure IoT hub, here are a few things to verify:

1. Are your credentials correct?
  - if you're using x509 certificates, double-check that the thumbprint in the registry matches the one of the certificate you're trying to use
  - if you're using a connection string with a shared access key, make sure it matches the device or a policy with the DeviceConnect capability.
  - if you're using a shared access signature, make sure the expiry is correct and that you're using the right shared access key to sign it.
2. Verify in your device registry (using the azure portal) that your device is enabled
3. Can you get through the firewall?
  - The easiest thing to try first is to run a tool that uses your device credentials and checks for a connection using all supported protocols. The iothub-diagnostics tool <https://github.com/azure/iothub-diagnostics> is designed to do just that and provides the results in the form of a report.
  - if you cannot run iothub-diagnostics you can try to run through the same steps manually:
    - ping a known website to verify name resolution and outbound traffic works

- Change the transport used to instantiate the client (Amqp, AmqpWs, Mqtt, MqttWs, and Http).
4. Try running the default samples <https://github.com/Azure/azure-iot-sdk-node/tree/master/device/samples>.
    - If the samples can connect, try finding differences between how you instantiate the client and how the samples do. It might be a simple typo.
    - If the samples cannot connect and neither can iothub-diagnostics it's likely an issue with the credentials or your network.

## Not detecting disconnections

The hard thing about disconnections is that they often seem random and if the SDK is not firing an error, there's no way to know what's going on. Or is there?

1. Could the retry logic be just delaying things?
  - By default the retry logic will go on for 4 minutes. Have you waited that long?
  - If you don't want to wait, try disabling the retry logic by calling `client.setRetryPolicy(new NoRetry());`
2. Need detailed logs? The SDK uses the debug library for logging
  - Set the `DEBUG` environment variable and re-run your application. A few good values for the `DEBUG` environment variable to get you started:
    - `azure*` will log SDK activity but not the underlying transport library
    - `amqp10*` will log the low-level AMQP library activity
    - `*` will log everything
  - debug logs to `stderr` by default, and can be quite verbose especially if set to `*`.
  - If you're saving those logs in order to post them in an issue, be careful to scrape for confidential information!

## Failing to send some messages

That's another tricky one. It looks like some messages are being sent, but not all of them. What gives? The first question to ask is How do you know some messages aren't being sent?

1. If it's because the callback is called with an error, the error object might give you more clues than just a message. Pay attention to the type of the error itself:
  - If it's a custom SDK type it should be pretty explicit, but if it's not enough, look at the properties of the error and try to see if there's a protocol-specific error in there.
  - If it's a generic Error it means the SDK failed to translate that error. Please file an issue and give us as many details as possible including the values of the error properties and the error stack.
2. If it's because you're not seeing the messages in your cloud application, try checking:
  - On the device side, the arguments passed to the callback of the send operation.
  - Try using iothub-explorer <https://github.com/Azure/iothub-explorer> with the `monitor-events` subcommand to check if the messages show up on the event-hubs compatible endpoint of your IoT Hub. If they do, at least you know that the device is acting properly. If they don't, you know it's unlikely to be a service issue and can track down device-side issues

## About the Module 9 Labs

### Module Labs

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

- Lab 17: Configure IoT Hub Monitoring

**WARNING:** Be prepared for UI changes

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

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

---

## Module 10 Azure Security Center and IoT Security Considerations

### Security Fundamentals for IoT Solutions

#### Security Recommendations

Microsoft suggests a number of security recommendations for individuals and companies working on IoT solutions. Implementing these recommendations will help you fulfill your security obligations as described in Microsoft's shared responsibility model.

Some of the recommendations described below can be automatically monitored by Azure Security Center. Azure Security Center is the first line of defense in protecting your resources in Azure. It periodically analyzes the security state of your Azure resources to identify potential security vulnerabilities. It then provides you with recommendations on how to address them.

#### General

Recommendation	Comments	Supported by ASC
Stay up-to-date	Use the latest versions of supported platforms, programming languages, protocols, and frameworks.	-
Keep authentication keys safe	Keep the device IDs and their authentication keys physically safe after deployment. This will avoid a malicious device masquerading as a registered device.	-

Recommendation	Comments	Supported by ASC
Use device SDKs when possible	Device SDKs implement a variety of security features, such as, encryption, authentication, and so on, to assist you in developing a robust and secure device application. Microsoft's continued investment in the SDKs means that you will benefit as support for new security advancements are added.	-

## Identity and Access Management

Recommendation	Comments	Supported by ASC
Define access control for the hub	Understand and define the type of access each component will have in your IoT Hub solution, based on the functionality. The allowed permissions are Registry Read, RegistryReadWrite, ServiceConnect, and DeviceConnect. Default shared access policies in your IoT hub can also help define the permissions for each component based on its role.	-
Define access control for back-end services	Data ingested by your IoT Hub solution can be consumed by other Azure services such as Cosmos DB, Stream Analytics, App Service, Logic Apps, and Blob storage. Make sure to understand and allow appropriate access permissions as documented for these services.	-

## Data protection

Recommendation	Comments	Supported by ASC
Secure device authentication	Ensure secure communication between your devices and your IoT hub, by using either a unique identity key or security token, or an on-device X.509 certificate for each device. Use the appropriate method to use security tokens based on the chosen protocol (MQTT, AMQP, or HTTPS).	-

Recommendation	Comments	Supported by ASC
Secure device communication	IoT Hub secures the connection to the devices using Transport Layer Security (TLS) standard, supporting versions 1.2 and 1.0. Use TLS 1.2 to ensure maximum security.	-
Secure service communication	IoT Hub provides endpoints to connect to backend services such as Azure Storage or Event Hubs using only the TLS protocol, and no endpoint is exposed on an unencrypted channel. Once this data reaches these backend services for storage or analysis, make sure to employ appropriate security and encryption methods for that service, and protect sensitive information at the backend.	-

## Networking

Recommendation	Comments	Supported by ASC
Protect access to your devices	Keep hardware ports in your devices to a bare minimum to avoid unwanted access. Additionally, build mechanisms to prevent or detect physical tampering of the device. Read IoT security best practices for details.	-
Build secure hardware	Incorporate security features such as encrypted storage, or Trusted Platform Module (TPM), to keep devices and infrastructure more secure. Keep the device operating system and drivers upgraded to latest versions, and if space permits, install antivirus and antimalware capabilities. Read IoT security architecture to understand how this can help mitigate several security threats.	-

## Monitoring

Recommendation	Comments	Supported by ASC
Monitor unauthorized access to your devices	Use your device operating system's logging feature to monitor any security breaches or physical tampering of the device or its ports.	-
Monitor your IoT solution from the cloud	Monitor the overall health of your IoT Hub solution using the metrics in Azure Monitor.	-
Set up diagnostics	Closely watch your operations by logging events in your solution, and then sending the diagnostic logs to Azure Monitor to get visibility into the performance. Read Monitor and diagnose problems in your IoT hub for more information.	-

## Security from the Ground Up

The Internet of Things (IoT) poses unique security, privacy, and compliance challenges to businesses worldwide. Unlike traditional cyber technology where these issues revolve around software and how it is implemented, IoT concerns what happens when the cyber and the physical worlds converge. Protecting IoT solutions requires ensuring secure provisioning of devices, secure connectivity between these devices and the cloud, and secure data protection in the cloud during processing and storage. Working against such functionality, however, are resource-constrained devices, geographic distribution of deployments, and a large number of devices within a solution.

## Microsoft Azure - secure IoT infrastructure for your business

Microsoft Azure offers a complete cloud solution, one that combines a constantly growing collection of integrated cloud services—analytics, machine learning, storage, security, networking, and web—with an industry-leading commitment to the protection and privacy of your data.

Microsoft's systems provide continuous intrusion detection and prevention, service attack prevention, regular penetration testing, and forensic tools that help identify and mitigate threats. Multi-factor authentication provides an extra layer of security for end users to access the network. And for the application and the host provider, Microsoft offers access control, monitoring, anti-malware, vulnerability scanning, patches, and configuration management.

The Azure IoT Hub offers a fully-managed service that enables reliable and secure bi-directional communication between IoT devices and Azure services such as Azure Machine Learning and Azure Stream Analytics by using per-device security credentials and access control.

## Secure device provisioning and authentication

Secure device provisioning means providing a unique identity key for each device, which can be used by the IoT infrastructure to communicate with the device while it is in operation. The generated key with a

user-selected device ID forms the basis of a token used in all communication between the device and the Azure IoT Hub.

Device IDs can be associated with a device during manufacturing (that is, flashed in a hardware trust module) or can use an existing fixed identity as a proxy (for example CPU serial numbers). Since changing this identifying information in the device is not simple, it is important to introduce logical device IDs in case the underlying device hardware changes but the logical device remains the same. In some cases, the association of a device identity can happen at device deployment time (for example, an authenticated field engineer physically configures a new device while communicating with the solution backend). The Azure IoT Hub identity registry provides secure storage of device identities and security keys for a solution. Individual or groups of device identities can be added to an allow list, or a block list, enabling complete control over device access.

Azure IoT Hub access control policies in the cloud enable activation and disabling any device identity, providing a way to disassociate a device from an IoT deployment when required. This association and disassociation of devices is based on each device identity.

Additional device security features include:

- Devices do not accept unsolicited network connections. They establish all connections and routes in an outbound-only fashion. For a device to receive a command from the backend, the device must initiate a connection to check for any pending commands to process. Once a connection between the device and IoT Hub is securely established, messaging from the cloud to the device and device to the cloud can be sent transparently.
- Devices only connect to or establish routes to well-known services with which they are peered, such as an Azure IoT Hub.
- System-level authorization and authentication use per-device identities, making access credentials and permissions near-instantly revocable.

## Secure connectivity

Azure IoT Hub supports secure connectivity using industry proven protocols - HTTPS, AMQP, and MQTT.

Azure IoT Hub offers durability of messaging between cloud and devices through a system of acknowledgments in response to messages. Additional durability for messaging is achieved by caching messages in the IoT Hub for up to seven days for telemetry and two days for commands. This enables devices that connect sporadically, due to power or connectivity concerns, to receive these commands. Azure IoT Hub maintains a per-device queue for each device.

Scalability requires the ability to securely interoperate with a wide range of devices. Azure IoT hub enables secure connection to both IP-enabled and non-IP-enabled devices (using an IoT Edge device as a gateway).

Additional connection security features include:

- The communication path between devices and Azure IoT Hub, or between gateways and Azure IoT Hub, is secured using industry-standard Transport Layer Security (TLS) with Azure IoT Hub authenticated using X.509 protocol.
- In order to protect devices from unsolicited inbound connections, Azure IoT Hub does not open any connection to the device. The device initiates all connections.

## Secure processing and storage in the cloud

Using Azure Active Directory (AAD) for user authentication and authorization, Azure IoT Hub can provide a policy-based authorization model for data in the cloud, enabling easy access management that can be audited and reviewed.

Once data is in the cloud, it can be processed and stored in any user-defined workflow. Access to each part of the data is controlled with Azure Active Directory, depending on the storage service used.

All keys used by the IoT infrastructure are stored in the cloud in secure storage, with the ability to roll over in case keys need to be reprovisioned. Data can be stored in Azure Cosmos DB or in SQL databases, enabling definition of the level of security desired. Additionally, Azure provides a way to monitor and audit all access to your data to alert you of any intrusion or unauthorized access.

## IoT Security Best Practices

Securing an Internet of Things (IoT) infrastructure requires a rigorous security-in-depth strategy. This strategy requires you to secure data in the cloud, protect data integrity while in transit over the public Internet, and securely provision devices. Each layer builds greater security assurance in the overall infrastructure.

### Secure an IoT Infrastructure

This security-in-depth strategy can be developed and executed with active participation of various players involved with the manufacturing, development, and deployment of IoT devices and infrastructure. Following is a high-level description of these players.

- IoT hardware manufacturer/integrator: Typically, these players are the manufacturers of IoT hardware being deployed, integrators assembling hardware from various manufacturers, or suppliers providing hardware for an IoT deployment manufactured or integrated by other suppliers.
- IoT solution developer: The development of an IoT solution is typically done by a solution developer. This developer may part of an in-house team or a system integrator (SI) specializing in this activity. The IoT solution developer can develop various components of the IoT solution from scratch, integrate various off-the-shelf or open-source components, or adopt solution accelerators with minor adaptation.
- IoT solution deployer: After an IoT solution is developed, it needs to be deployed in the field. This process involves deployment of hardware, interconnection of devices, and deployment of solutions in hardware devices or the cloud.
- IoT solution operator: After the IoT solution is deployed, it requires long-term operations, monitoring, upgrades, and maintenance. These tasks can be done by an in-house team that comprises information technology specialists, hardware operations and maintenance teams, and domain specialists who monitor the correct behavior of overall IoT infrastructure.

The sections that follow provide best practices for each of these players to help develop, deploy, and operate a secure IoT infrastructure.

### IoT Hardware Manufacturer/Integrator

The following are the best practices for IoT hardware manufacturers and hardware integrators.

- Scope hardware to minimum requirements: The hardware design should include the minimum features required for operation of the hardware, and nothing more. An example is to include USB

ports only if necessary for the operation of the device. These additional features open the device for unwanted attack vectors that should be avoided.

- Make hardware tamper proof: Build in mechanisms to detect physical tampering, such as opening of the device cover or removing a part of the device. These tamper signals may be part of the data stream uploaded to the cloud, which could alert operators of these events.
- Build around secure hardware: If COGS permits, build security features such as secure and encrypted storage, or boot functionality based on Trusted Platform Module (TPM). These features make devices more secure and help protect the overall IoT infrastructure.
- Make upgrades secure: Firmware upgrades during the lifetime of the device are inevitable. Building devices with secure paths for upgrades and cryptographic assurance of firmware versions will allow the device to be secure during and after upgrades.

## IoT Solution Developer

The following are the best practices for IoT solution developers:

- Follow secure software development methodology: Development of secure software requires ground-up thinking about security, from the inception of the project all the way to its implementation, testing, and deployment. The choices of platforms, languages, and tools are all influenced with this methodology. The Microsoft Security Development Lifecycle provides a step-by-step approach to building secure software.
- Choose open-source software with care: Open-source software provides an opportunity to quickly develop solutions. When you're choosing open-source software, consider the activity level of the community for each open-source component. An active community ensures that software is supported and that issues are discovered and addressed. Alternatively, an obscure and inactive open-source software project might not be supported and issues are not likely be discovered.
- Integrate with care: Many software security flaws exist at the boundary of libraries and APIs. Functionality that may not be required for the current deployment might still be available via an API layer. To ensure overall security, make sure to check all interfaces of components being integrated for security flaws.

## IoT Solution Deployer

The following are best practices for IoT solution deployers:

- Deploy hardware securely: IoT deployments may require hardware to be deployed in unsecure locations, such as in public spaces or unsupervised locales. In such situations, ensure that hardware deployment is tamper-proof to the maximum extent. If USB or other ports are available on the hardware, ensure that they are covered securely. Many attack vectors can use these as entry points.
- Keep authentication keys safe: During deployment, each device requires device IDs and associated authentication keys generated by the cloud service. Keep these keys physically safe even after the deployment. Any compromised key can be used by a malicious device to masquerade as an existing device.

## IoT Solution Operator

The following are the best practices for IoT solution operators:

- Keep the system up-to-date: Ensure that device operating systems and all device drivers are upgraded to the latest versions. If you turn on automatic updates in Windows 10 (IoT or other SKUs), Microsoft

- keeps it up-to-date, providing a secure operating system for IoT devices. Keeping other operating systems (such as Linux) up-to-date helps ensure that they are also protected against malicious attacks.
- Protect against malicious activity: If the operating system permits, install the latest antivirus and antimalware capabilities on each device operating system. This practice can help mitigate most external threats. You can protect most modern operating systems against threats by taking appropriate steps.
  - Audit frequently: Auditing IoT infrastructure for security-related issues is key when responding to security incidents. Most operating systems provide built-in event logging that should be reviewed frequently to make sure no security breach has occurred. Audit information can be sent as a separate telemetry stream to the cloud service where it can be analyzed.
  - Physically protect the IoT infrastructure: The worst security attacks against IoT infrastructure are launched using physical access to devices. One important safety practice is to protect against malicious use of USB ports and other physical access. One key to uncovering breaches that might have occurred is logging of physical access, such as USB port use. Again, Windows 10 (IoT and other SKUs) enables detailed logging of these events.
  - Protect cloud credentials: Cloud authentication credentials used for configuring and operating an IoT deployment are possibly the easiest way to gain access and compromise an IoT system. Protect the credentials by changing the password frequently, and refrain from using these credentials on public machines.

Capabilities of different IoT devices vary. Some devices might be computers running common desktop operating systems, and some devices might be running very light-weight operating systems. The security best practices described previously might be applicable to these devices in varying degrees. If provided, additional security and deployment best practices from the manufacturers of these devices should be followed.

Some legacy and constrained devices might not have been designed specifically for IoT deployment. These devices might lack the capability to encrypt data, connect with the Internet, or provide advanced auditing. In these cases, a modern and secure field gateway can aggregate data from legacy devices and provide the security required for connecting these devices over the Internet. Field gateways can provide secure authentication, negotiation of encrypted sessions, receipt of commands from the cloud, and many other security features.

## IoT Security Architecture and Threat Modeling

Azure IoT Developers are not responsible for the architectural design of an IoT solution, but having an understanding of the threats to a solution is important.

## Solution Architecture and Security

When designing a system, it is important to understand the potential threats to that system, and add appropriate defenses accordingly, as the system is designed and architected. It is important to design the product from the start with security in mind because understanding how an attacker might be able to compromise a system helps make sure appropriate mitigations are in place from the beginning.

Design teams use threat modeling techniques to consider mitigations as the system is designed rather than after a system is deployed. This fact is critically important, because retrofitting security defenses to a myriad of devices in the field is infeasible, error prone and leaves customers at risk.

## Threat Modeling

The objective of threat modeling is to understand how an attacker might be able to compromise a system and then make sure appropriate mitigations are in place.

### What to consider for threat modeling

You should look at the solution as a whole and also focus on the following areas:

- The security and privacy features
- The features whose failures are security relevant
- The features that touch a trust boundary

### Who performs threat modeling

Threat modeling is a process like any other. It is a good idea to treat the threat model document like any other component of the solution and validate it as a team. Many development teams do an excellent job capturing the functional requirements for the system that benefit customers. However, identifying non-obvious ways that someone might misuse the system is more challenging. Threat modeling can help development teams understand what an attacker might do and why.

### How to perform threat modeling

The threat modeling process is composed of four steps; the steps are:

- Model the application
- Enumerate Threats
- Mitigate threats
- Validate the mitigations

## Security in IoT

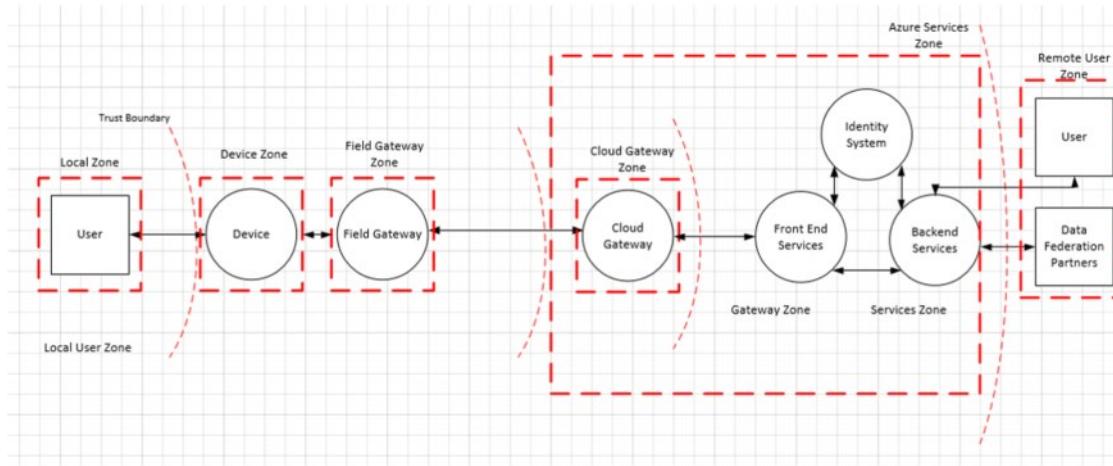
Connected special-purpose devices have a significant number of potential interaction surface areas and interaction patterns, all of which must be considered to provide a framework for securing digital access to those devices. The term “digital access” is used here to distinguish from any operations that are carried out through direct device interaction where access security is provided through physical access control. For example, putting the device into a room with a lock on the door. While physical access cannot be denied using software and hardware, measures can be taken to prevent physical access from leading to system interference.

In order to optimize security best practices, it is recommended that a typical IoT architecture is divided into several component/zones as part of the threat modeling exercise. These zone are:

- Device
- Edge Gateway (an Edge device used as a Field Gateway)
- Cloud gateways (IoT Hub)
- Services

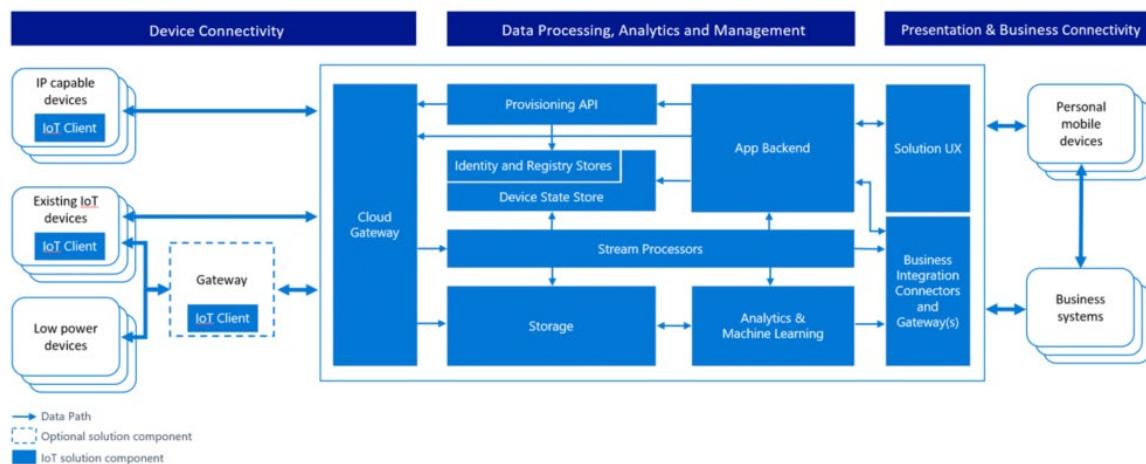
Each zone is separated by a Trust Boundary, which is noted as the dotted red line in the following diagram. It represents a transition of data/information from one source to another. During this transition,

the data/information could be subject to Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service and Elevation of Privilege (STRIDE).

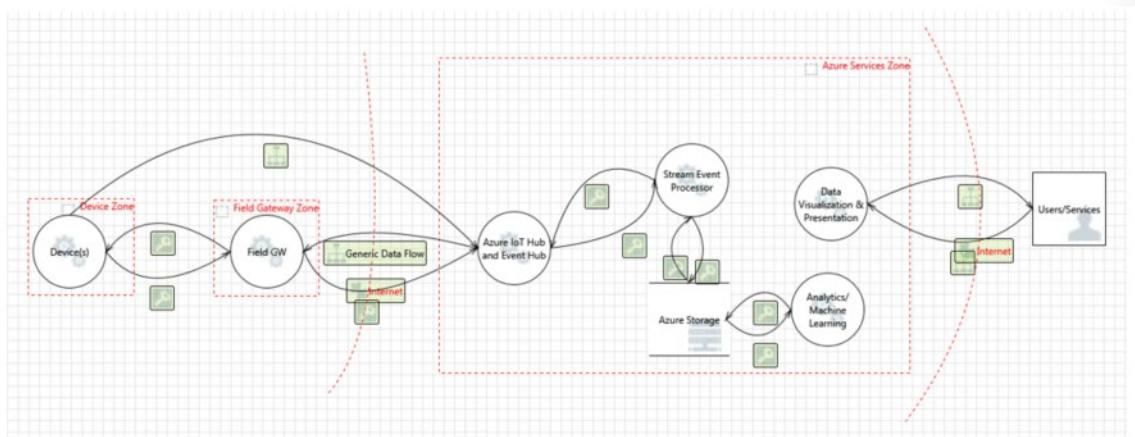


We can use the Azure IoT Reference Architecture to demonstrate how to think about threat modeling for IoT and how to address the threats identified. This approach identifies four main areas of focus:

- Devices and Data Sources
- Data Transport
- Device and Event Processing
- Presentation



The following diagram provides a simplified view of Microsoft's IoT Architecture using a Data Flow Diagram model that is used by the Microsoft Threat Modeling Tool:



You should notice that this diagram combines the *Devices and Data Sources* and *Data Transport* focus areas, and places a trust boundary on each side of an Azure Services Zone for the solution.

## Common Security Threats and Mitigations

For an Azure IoT solution, threats most commonly target the physical devices or one of the trust boundaries identified in the simplified threat model diagram above.

Consider the following STRIDE definitions:

- Spoofing (S): A spoofing attack occurs when an attacker pretends to be someone they're not. Spoofing attacks can happen locally. For example, an attacker may extract cryptographic key material from a device, either at the software or hardware level, and subsequently access the system with a different physical using the identity of the device the key material was taken from.
- Tampering (T): Tampering attacks occur when the attacker modifies the data in transit (and for IoT may include compromising the physical device). For example, an attacker may compromise a physical device to obtain cryptographic key materials, then intercept and suppress data from the device on the communication path, and finally leverage extracted key material to replace the data with false data that is authenticated with the stolen key material.
- Repudiation (R): Repudiation occurs when someone performs an action and then claims that they didn't actually do it. Repudiation is typically associated with the ability to properly track and log user actions and for Azure IoT, this mitigated by the Azure IoT Hub service. Repudiation does not apply to attacks against physical devices.
- Information Disclosure (I): Information Disclosure threats are usually quite straightforward - can the attacker view data that they're not supposed to view? For example, if you're transferring data from one computer to another and the attacker can sniff the data on the wire, then your component is subject to an information disclosure threat.
- Denial of Service (D): Denial of service threats occur when an attacker can degrade or deny service to users. For IoT, this includes rendering a device incapable of functioning or communicating. For example, a surveillance camera that had its power or network connection intentionally knocked out cannot report data.
- Elevation of Privilege (E): An elevation of privilege threat occurs when an attacker has the ability to gain privileges that they'd not normally have. For IoT this may be forcing a device to do something other than the intended function. For example, a valve that is programmed to open half way can be tricked to open all the way.

## Common Threats Against Physical Devices

Component	Threat	Mitigation	Risk	Implementation
Device	S	Assigning identity to the device and authenticating the device	Replacing device or part of the device with some other device. How do you know you are talking to the right device?	Authenticating the device, using Transport Layer Security (TLS) or IPSec. Infrastructure should support using pre-shared key (PSK) on those devices that cannot handle full asymmetric cryptography. Leverage Azure AD, OAuth
Device	TID	Apply tamperproof mechanisms to the device, for example, by making it hard to impossible to extract keys and other cryptographic material from the device.	The risk is if someone is tampering the device (physical interference). How are you sure, that device has not been tampered with.	The most effective mitigation is a trusted platform module (TPM) capability that allows storing keys in special on-chip circuitry from which the keys cannot be read, but can only be used for cryptographic operations that use the key but never disclose the key. Memory encryption of the device. Key management for the device. Signing the code.

Component	Threat	Mitigation	Risk	Implementation
Device	E	Having access control of the device. Authorization scheme.	If the device allows for individual actions to be performed based on commands from an outside source, or even compromised sensors, it allows the attack to perform operations not otherwise accessible.	Having authorization scheme for the device
Edge Gateway (Field Gateway)	S	Authenticating the Field gateway to Cloud Gateway (such as cert based, PSK, or Claim based.)	If someone can spoof Field Gateway, then it can present itself as any device.	TLS RSA/PSK, IPSec, RFC 4279. All the same key storage and attestation concerns of devices in general – best case is use TPM. 6LowPAN extension for IPSec to support Wireless Sensor Networks (WSN).
Edge Gateway (Field Gateway)	TID	Protect the Field Gateway against tampering (TPM?)	Spoofing attacks that trick the cloud gateway thinking it is talking to field gateway could result in information disclosure and data tampering	Memory encryption, TPM's, authentication.
Edge Gateway (Field Gateway)	E	Access control mechanism for Field Gateway		

Here are some examples of threats to physical devices:

- Spoofing: An attacker may extract cryptographic key material from a device, either at the software or hardware level, and subsequently access the system with a different physical or virtual device under the identity of the device the key material has been taken from.
- Denial of Service: A device can be rendered incapable of functioning or communicating by interfering with radio frequencies or cutting wires. For example, a surveillance camera that had its power or network connection intentionally knocked out cannot report data, at all.
- Tampering: An attacker may partially or wholly replace the software running on the device, potentially allowing the replaced software to leverage the genuine identity of the device if the key material or the cryptographic facilities holding key materials were available to the illicit program.

- Tampering: A surveillance camera that's showing a visible-spectrum picture of an empty hallway could be aimed at a photograph of such a hallway. A smoke or fire sensor could be reporting someone holding a lighter under it. In either case, the device may be technically fully trustworthy towards the system, but it reports manipulated information.
- Tampering: An attacker may leverage extracted key material to intercept and suppress data from the device on the communication path and replace it with false data that is authenticated with the stolen key material.
- Tampering: An attacker may partially or completely replace the software running on the device, potentially allowing the replaced software to leverage the genuine identity of the device if the key material or the cryptographic facilities holding key materials were available to the illicit program.
- Information Disclosure: If the device is running manipulated software, such manipulated software could potentially leak data to unauthorized parties.
- Information Disclosure: An attacker may leverage extracted key material to inject itself into the communication path between the device and a controller or field gateway or cloud gateway to siphon off information.
- Denial of Service: The device can be turned off or turned into a mode where communication is not possible (which is intentional in many industrial machines).
- Tampering: The device can be reconfigured to operate in a state unknown to the control system (outside of known calibration parameters) and thus provide data that can be misinterpreted
- Elevation of Privilege: A device that does specific function can be forced to do something else. For example, a valve that is programmed to open half way can be tricked to open all the way.
- Denial of Service: The device can be turned into a state where communication is not possible.
- Tampering: The device can be reconfigured to operate in a state unknown to the control system (outside of known calibration parameters) and thus provide data that can be misinterpreted.

## Common Threats Against Communication

Component	Threat	Mitigation	Risk	Implementation
Device to IoT Hub	TID	(D)TLS (PSK/RSA) to encrypt the traffic	Eavesdropping or interfering with the communication between the device and the gateway	Security on the protocol level. With custom protocols, you need to figure out how to protect them. In most cases, the communication takes place from the device to the IoT Hub (device initiates the connection).

Component	Threat	Mitigation	Risk	Implementation
Device to Device	TID	(D)TLS (PSK/RSA) to encrypt the traffic.	Reading data in transit between devices. Tampering with the data. Overloading the device with new connections	Security on the protocol level (MQTT/AMQP/HTTP/CoAP). With custom protocols, you need to figure out how to protect them. The mitigation for the DoS threat is to peer devices through a cloud or field gateway and have them only act as clients towards the network. The peering may result in a direct connection between the peers after having been brokered by the gateway
External Entity to Device	TID	Strong pairing of the external entity to the device	Eavesdropping the connection to the device. Interfering the communication with the device	Securely pairing the external entity to the device NFC/Bluetooth LE. Controlling the operational panel of the device (Physical)
Edge Gateway to Cloud Gateway	TID	TLS (PSK/RSA) to encrypt the traffic.	Eavesdropping or interfering the communication between the device and the gateway	Security on the protocol level (MQTT/AMQP/HTTP/CoAP). With custom protocols, you need to figure out how to protect them.
Device to Cloud Gateway	TID	TLS (PSK/RSA) to encrypt the traffic.	Eavesdropping or interfering the communication between the device and the gateway	Security on the protocol level (MQTT/AMQP/HTTP/CoAP). With custom protocols, you need to figure out how to protect them.

MCT USE ONLY. STUDENT USE PROHIBITED

Here are some examples of threats to communication:

- Denial of Service: Constrained devices are generally under DoS threat when they actively listen for inbound connections or unsolicited datagrams on a network, because an attacker can open many connections in parallel and not service them or service them slowly, or the device can be flooded with unsolicited traffic. In both cases, the device can effectively be rendered inoperable on the network.
- Spoofing, Information Disclosure: Constrained devices and special-purpose devices often have one-for-all security facilities like password or PIN protection, or they wholly rely on trusting the network, meaning they grant access to information when a device is on the same network, and that network is often only protected by a shared key. That means that when the shared secret to device or network is disclosed, it is possible to control the device or observe data emitted from the device.
- Spoofing: an attacker may intercept or partially override the broadcast and spoof the originator (man in the middle)
- Tampering: an attacker may intercept or partially override the broadcast and send false information
- Information Disclosure: an attacker may eavesdrop on a broadcast and obtain information without authorization Denial of Service: an attacker may jam the broadcast signal and deny information distribution

# Introduction to Azure Security Center for IoT

## What is Azure Security Center?

Azure Security Center provides you the tools needed to harden your network, secure your services and make sure you're on top of your security posture.

Azure Security Center addresses the three most urgent security challenges:

- Rapidly changing workloads – It's both a strength and a challenge of the cloud. On the one hand, end users are empowered to do more. On the other, how do you make sure that the ever-changing services people are using and creating are up to your security standards and follow security best practices?
- Increasingly sophisticated attacks - Wherever you run your workloads, the attacks keep getting more sophisticated. You have to secure your public cloud workloads, which are, in effect, an Internet facing workload that can leave you even more vulnerable if you don't follow security best practices.
- Security skills are in short supply - The number of security alerts and alerting systems far outnumbers the number of administrators with the necessary background and experience to make sure your environments are protected. Staying up-to-date with the latest attacks is a constant challenge, making it impossible to stay in place while the world of security is an ever-changing front.

To help you protect yourself against these challenges, Security Center provides you with the tools to:

- Strengthen security posture: Security Center assesses your environment and enables you to understand the status of your resources, and whether they are secure.
- Protect against threats: Security Center assesses your workloads and raises threat prevention recommendations and threat detection alerts.
- Get secure faster: In Security Center, everything is done in cloud speed. Because it is natively integrated, deployment of Security Center is easy, providing you with autoprovisioning and protection with Azure services.

## Introduction to Azure Security Center for IoT

Azure Security Center for IoT enables you to unify security management and enable end-to-end threat detection and analysis across hybrid cloud workloads and your Azure IoT solution.

Azure Security Center for IoT is composed of the following components:

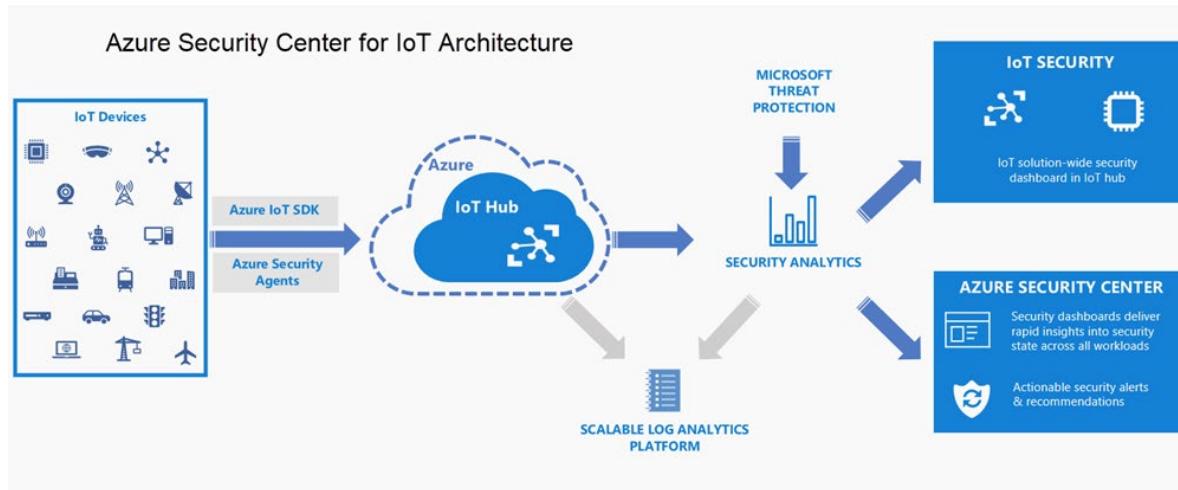
- IoT Hub integration
- Device agents (optional)
- Send security message SDK
- Analytics pipeline

## Secure your entire IoT solution from IoT devices to Azure cloud.

Choose from our seamless agentless solution or take advantage of agent-based comprehensive security. Azure Security Center for IoT provides threat prevention and analysis for every device, IoT Edge and IoT Hub, across your IoT assets.

As billions of new devices are connected to the Internet, and integrated into our daily lives and our businesses, your security operations teams must ensure their security strategies evolve quickly enough to cover each new attack surface. Like any other system, to comprehensively secure your IoT solution, it requires protection at every stage of implementation.

Azure Security Center for IoT simplifies hybrid workload protection by delivering unified visibility and control, adaptive threat prevention, and intelligent threat detection and response across workloads running on edge, on-premises, in Azure, and in other clouds.



## Unified visibility and control

Get a unified view of security across all of your on-premises and cloud workloads, including your Azure IoT solution. Onboard new devices, and apply security policies across your workloads (Leaf devices, Microsoft Edge devices, IoT Hub) to ensure compliance with security standards and improved security posture.

## Adaptive threat prevention

Use Azure Security Center for IoT to continuously monitor the security of machines, networks, and Azure services. Choose from hundreds of built-in security assessments or create your own in the central Azure Security Center for IoT Hub dashboard. Optimize your security settings and improve your security score with actionable recommendations across virtual machines, networks, apps, and data. With newly added IoT capabilities, you can now reduce the attack surface for your Azure IoT solution and remediate issues before they can be exploited.

## Intelligent threat detection and response

Use advanced analytics and the Microsoft Intelligent Security Graph to get an edge over evolving cyber-attacks. Built-in behavioral analytics and machine learning identify attacks and zero-day exploits. Monitor your IoT solution for incoming attacks and post-breach activity. Streamline device investigation and remediation with interactive tools and contextual threat intelligence.

## Azure Security Center for IoT Prerequisites

### Minimum requirements

- IoT Hub Standard tier
  - RBAC role Owner level privileges
- Log Analytics Workspace
- Azure Security Center (recommended)
  - Use of Azure Security Center is a recommendation, and not a requirement. Without Azure Security Center, you'll be unable to view your other Azure resources within IoT Hub.

### Working with Azure Security Center for IoT service

Azure Security Center for IoT insights and reporting are available using Azure IoT Hub and Azure Security Center. To enable Azure Security Center for IoT on your Azure IoT Hub, an account with Owner level privileges is required. After enabling ASC for IoT in your IoT Hub, Azure Security Center for IoT insights are displayed as the Security feature in Azure IoT Hub and as IoT in Azure Security Center.

### Supported service regions

Azure Security Center for IoT is currently supported for IoT Hubs in the following Azure regions:

- Central US
- East US
- East US 2
- West Central US
- West US
- West US2
- Central US South
- North Central US
- Canada Central
- Canada East
- North Europe
- Brazil South
- France Central
- UK West
- UK South
- West Europe
- Northern Europe
- Japan West
- Japan East

- Australia Southeast
- Australia East
- East Asia
- Southeast Asia
- Korea Central
- Korea South
- Central India
- South India

Azure Security Center for IoT routes all traffic from all European regions to the West Europe regional data center and all remaining regions to the Central US regional data center.

## Where's my IoT Hub?

Check your IoT Hub location to verify service availability before you begin.

- Open your IoT Hub.
- Click Overview.
- Verify the location listed matches one of the supported service regions.

## Supported platforms for agents

Azure Security Center for IoT agents supports a growing list of devices and platforms.

# Security Center for IoT Deployment Options

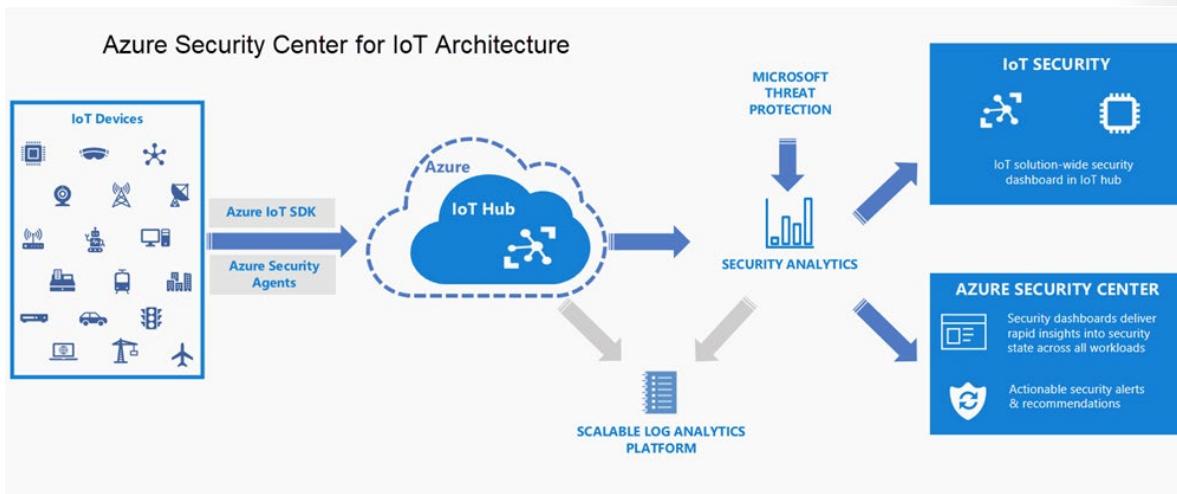
Azure Security Center for IoT works in one of two feature workflows: Built-in and Enhanced

## Built-in

In Built-in mode, Azure Security Center for IoT is enabled when you elect to turn on the Security option in your IoT Hub. Offering real-time monitoring, recommendations and alerts, Built-in mode offers single-step device visibility and unmatched security. Built-in mode does not require agent installation on any devices and uses advanced analytics on logged activities to analyze and protect your field device.

## Enhanced

In Enhanced mode, after turning on the Security option in your IoT Hub and installing Azure Security Center for IoT device agents on your devices, the agents collect, aggregate and analyze raw security events from your devices. Raw security events can include IP connections, process creation, user logins, and other security-relevant information. Azure Security Center for IoT device agents also handle event aggregation to help avoid high network throughput. The agents are highly customizable, allowing you to use them for specific tasks, such as sending only important information at the fastest SLA, or for aggregating extensive security information and context into larger segments, avoiding higher service costs.



Device agents, and other applications use the Azure send security message SDK to send security information into Azure IoT Hub. IoT Hub picks up this information and forwards it to the Azure Security Center for IoT service.

Once the Azure Security Center for IoT service is enabled, in addition to the forwarded data, IoT Hub also sends out all of its internal data for analysis by Azure Security Center for IoT. This data includes device-cloud operation logs, device identities, and Hub configuration. All of this information helps to create the Azure Security Center for IoT analytics pipeline.

Azure Security Center for IoT analytics pipeline also receives additional threat intelligence streams from various sources within Microsoft and Microsoft partners. The Azure Security Center for IoT entire analytics pipeline works with every customer configuration made on the service (such as custom alerts and use of the send security message SDK).

Using the analytics pipeline, Azure Security Center for IoT combines all of the streams of information to generate actionable recommendations and alerts. The pipeline contains both custom rules created by security researchers and experts as well as machine learning models searching for deviation from standard device behavior and risk analysis.

Azure Security Center for IoT recommendations and alerts (analytics pipeline output) is written to the Log Analytics workspace of each customer. Including the raw events in the workspace as well as the alerts and recommendations enables deep dive investigations and queries using the exact details of the suspicious activities detected.

## Configure Built-in IoT Hub Integration

The built-in option for Azure Security Center for IoT enables you to use the service without using Azure Security Center for IoT security agents.

### Enable Azure Security Center for IoT on your IoT Hub

To enable security on your IoT Hub:

- Open your IoT Hub in the Azure portal.
- On the left side menu, under the Security section, click **Overview**.
- On the Overview blade, click **Secure your IoT solution**.

That's all that need to be completed to enable Azure Security Center for IoT on your IoT Hub. If Azure Security Center for IoT was previously enabled and then disabled, you can re-enable from the Overview blade by clicking the Settings button and then selecting Enable.

## Geolocation and IP address handling

To secure your IoT solution, IP addresses of incoming and outgoing connections to and from your IoT devices, IoT Edge, and IoT Hub(s) are collected and stored by default. This information is essential to detect abnormal connectivity from suspicious IP sources. For example, when attempts are made to establish connections from an IP source of a known botnet or from an IP source outside your geolocation. Azure Security Center for IoT service offers the flexibility to enable and disable collection of IP address data at any time.

To enable or disable collection of IP address data:

- Open your IoT Hub and then select Overview from the Security menu.
- Choose the Settings screen and modify the geolocation and/or IP handling settings as you wish.

## Log Analytics creation

When Azure Security Center for IoT is turned on, a default Azure Log Analytics workspace is created to store raw security events, alerts, and recommendations for your IoT devices, IoT Edge, and IoT Hub. Each month, the first five (5) GB of data ingested per customer to the Azure Log Analytics service is free. Every GB of data ingested into your Azure Log Analytics workspace is retained at no charge for the first 31 days.

To change the workspace configuration of Log Analytics:

- Open your IoT Hub and then select Overview from the Security menu.
- Choose the Settings screen and modify the workspace configuration of Log Analytics settings as you wish.

## Customize your IoT security solution

By default, turning on the Azure Security Center for IoT solution automatically secures all IoT Hubs under your Azure subscription.

In addition to automatic relationship detection, you can also pick and choose which other Azure resource groups to tag as part of your IoT solution.

Your selections allow you to add entire subscriptions, resource groups, or single resources.

After defining all of the resource relationships, Azure Security Center for IoT leverages Azure Security Center to provide you security recommendations and alerts for these resources.

To turn Azure Security Center for IoT service on a specific IoT Hub on or off:

- Open your IoT Hub and then select Overview from the Security menu.
- Choose the Settings screen and modify the security settings of any IoT hub in your Azure subscription as you wish.

To add new resource to your IoT solution, do the following:

- Open your IoT Hub in Azure portal.
- On the left side menu, under the Security section, click **Resources**.
- Click Edit, and then choose the resources groups that belong to your IoT solution.

- Click **Add**.

## Azure Security Center for IoT security alerts

Azure Security Center for IoT continuously analyzes your IoT solution using advanced analytics and threat intelligence to alert you to malicious activity.

In addition, you can create custom alerts based on your knowledge of expected device behavior.

An alert acts as an indicator of potential compromise, and should be investigated and remediated.

In this article, you will find a list of built-in alerts which can be triggered on your IoT Hub and/or IoT devices.

In addition to built-in alerts, Azure Security Center for IoT allows you to define custom alerts based on expected IoT Hub and/or device behavior.

For more details, see [Create custom alerts](#).

### Built-in alerts for IoT devices

Name	Severity	Data Source	Description	Suggested remediation steps	
<b>High severity</b>					
Binary Command Line	High	Agent	LA Linux binary being called/ executed from the command line was detected. This process may be legitimate activity, or an indication that your device is compromised.	Review the command with the user that ran it and check if this is something legitimately expected to run on the device. If not, escalate the alert to your information security team.	
Disable firewall	High	Agent	Possible manipulation of on-host firewall detected. Malicious actors often disable the on-host firewall in an attempt to exfiltrate data.	Review with the user that ran the command to confirm if this was legitimate expected activity on the device. If not, escalate the alert to your information security team.	

MCT USE ONLY. STUDENT USE PROHIBITED

Name	Severity	Data Source	Description	Suggested remediation steps	
Port forwarding detection	High	Agent	Initiation of port forwarding to an external IP address detected.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Possible attempt to disable Auditd logging detected	High	Agent	Linux Auditd system provides a way to track security-relevant information on the system. The system records as much information about the events that are happening on your system as possible. This information is crucial for mission-critical environments to determine who violated the security policy and the actions they performed. Disabling Auditd logging may prevent your ability to discover violations of security policies used on the system.	Check with the device owner if this was legitimate activity with business reasons. If not, this event may be hiding activity by malicious actors. Immediately escalated the incident to your information security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Reverse shells	High	Agent	Analysis of host data on %{Compro- mised Host} detected a potential reverse shell. Reverse shells are often used to get a compromised machine to call back into a machine controlled by a malicious actor.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Successful Bruteforce attempt	High	Agent	Multiple unsuccessful login attempts were identified, followed by a successful login. Attempted Bruteforce attack may have succeeded on the device.	Review SSH Bruteforce alert and the activity on the devices. If the activity was malicious: Roll out password reset for compro- mised ac- counts. Investigate and remediate (if found) devices for malware.	
Successful local login	High	Agent	Successful local sign in to the device detect- ed	Make sure the signed in user is an author- ized party.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Web shell	High	Agent	Possible web shell detected. Malicious actors commonly upload a web shell to a compromised machine to gain persistence or for further exploitation.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
<b>Medium severity</b>					
Behavior similar to common Linux bots detected	Medium	Agent	Execution of a process normally associated with common Linux botnets detected.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Behavior similar to Fairware ransomware detected	Medium	Agent	Execution of rm -rf commands applied to suspicious locations detected using analysis of host data. Because rm -rf recursively deletes files, it is normally only used on discrete folders. In this case, it is being used in a location that could remove a large amount of data. Fairware ransomware is known to execute rm -rf commands in this folder.	Review with the user that ran the command this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Behavior similar to ransomware detected	Medium	Agent	Execution of files similar to known ransomware that may prevent users from accessing their system, or personal files, and may demand ransom payment to regain access.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

MCT USE ONLY. STUDENT USE PROHIBITED

Name	Severity	Data Source	Description	Suggested remediation steps	
Crypto coin miner container image detected	Medium	Agent	Container detecting running known digital currency mining images.	<p>1. If this behavior is not intended, delete the relevant container image.</p> <p>2. Make sure that the Docker daemon is not accessible via an unsafe TCP socket.</p> <p>3. Escalate the alert to the information security team.</p>	
Crypto coin miner image	Medium	Agent	Execution of a process normally associated with digital currency mining detected.	Verify with the user that ran the command if this was legitimate activity on the device. If not, escalate the alert to the information security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Detected suspicious use of the nohup command	Medium	Agent	Suspicious use of the nohup command on host detected. Malicious actors commonly run the nohup command from a temporary directory, effectively allowing their executables to run in the background. Seeing this command run on files located in a temporary directory is not expected or usual behavior.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Detected suspicious use of the useradd command	Medium	Agent	Suspicious use of the useradd command detected on the device.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

MCT USE ONLY. STUDENT USE PROHIBITED

Name	Severity	Data Source	Description	Suggested remediation steps	
Exposed Docker daemon by TCP socket	Medium	Agent	Machine logs indicate that your Docker daemon (dockerd) exposes a TCP socket. By default, Docker configuration, does not use encryption or authentication when a TCP socket is enabled. Default Docker configuration enables full access to the Docker daemon, by anyone with access to the relevant port.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Failed local login	Medium	Agent	A failed local login attempt to the device was detected.	Make sure no unauthorized party has physical access to the device.	
File downloads from a known malicious source detected	Medium	Agent	Download of a file from a known malware source detected.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
htaccess file access detected	Medium	Agent	Analysis of host data detected possible manipulation of an htaccess file. Htaccess is a powerful configuration file that allows you to make multiple changes to a web server running Apache Web software, including basic redirect functionality, and more advanced functions, such as basic password protection. Malicious actors often modify htaccess files on compromised machines to gain persistence.	Confirm this is legitimate expected activity on the host. If not, escalate the alert to your information security team.	
Known attack tool	Medium	Agent	A tool often associated with malicious users attacking other machines in some way was detected.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

**MCT USE ONLY. STUDENT USE PROHIBITED**

Name	Severity	Data Source	Description	Suggested remediation steps	
IoT agent attempted and failed to parse the module twin configuration	Medium	Agent	The Azure Security Center for IoT security agent failed to parse the module twin configuration due to type mismatches in the configuration object	Validate your module twin configuration against the IoT agent configuration schema, fix all mismatches.	
Local host reconnaissance detected	Medium	Agent	Execution of a command normally associated with common Linux bot reconnaissance detected.	Review the suspicious command line to confirm that it was executed by a legitimate user. If not, escalate the alert to your information security team.	
Mismatch between script interpreter and file extension	Medium	Agent	Mismatch between the script interpreter and the extension of the script file provided as input detected. This type of mismatch is commonly associated with attacker script executions.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Possible backdoor detected	Medium	Agent	A suspicious file was downloaded and then run on a host in your subscription. This type of activity is commonly associated with the installation of a backdoor.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Potential loss of data detected	Medium	Agent	Possible data egress condition detected using analysis of host data. Malicious actors often egress data from compromised machines.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Potential overriding of common files	Medium	Agent	Common executable overwritten on the device. Malicious actors are known to overwrite common files as a way to hide their actions or as a way to gain persistence.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

**MCT USE ONLY. STUDENT USE PROHIBITED**

Name	Severity	Data Source	Description	Suggested remediation steps	
Privileged container detected	Medium	Agent	Machine logs indicate that a privileged Docker container is running. A privileged container has full access to host resources. If compromised, a malicious actor can use the privileged container to gain access to the host machine.	If the container doesn't need to run in privileged mode, remove the privileges from the container.	
Removal of system logs files detected	Medium	Agent	Suspicious removal of log files on the host detected.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Space after file-name	Medium	Agent	Execution of a process with a suspicious extension detected using analysis of host data. Suspicious extensions may trick users into thinking files are safe to be opened and can indicate the presence of malware on the system.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Suspected malicious credentials access tools detected	Medium	Agent	Detection usage of a tool commonly associated with malicious attempts to access credentials.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Suspicious compilation detected	Medium	Agent	Suspicious compilation detected. Malicious actors often compile exploits on a compromised machine to escalate privileges.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	

MCT USE ONLY. STUDENT USE PROHIBITED

Name	Severity	Data Source	Description	Suggested remediation steps	
Suspicious file download followed by file run activity	Medium	Agent	Analysis of host data detected a file that was downloaded and run in the same command. This technique is commonly used by malicious actors to get infected files onto victim machines.	Review with the user that ran the command if this was legitimate activity that you expect to see on the device. If not, escalate the alert to the information security team.	
Suspicious IP address communication	Medium	Agent	Communication with a suspicious IP address detected.	Verify if the connection is legitimate. Consider blocking communication with the suspicious IP.	
x.509 device certificate thumbprint mismatch	Medium	IoT Hub	x.509 device certificate thumbprint did not match configuration.	Review alerts on the devices. No further action required.	
x.509 certificate expired	Medium	IoT Hub	X.509 device certificate has expired.	This could be a legitimate device with an expired certificate or an attempt to impersonate a legitimate device. If the legitimate device is currently communicating correctly this is likely an impersonation attempt.	
<b>LOW severity</b>					

Name	Severity	Data Source	Description	Suggested remediation steps	
Bash history cleared	Low	Agent	Bash history log cleared. Malicious actors commonly erase bash history to hide their own commands from appearing in the logs.	Review with the user that ran the command that the activity in this alert to see if you recognize this as legitimate administrative activity. If not, escalate the alert to the information security team.	
Device silent	Low	Agent	Device has not sent any telemetry data in the last 72 hours.	Make sure device is online and sending data. Check that the Azure Security Agent is running on the device.	
Expired SAS Token	Low	IoT Hub	Expired SAS token used by a device	May be a legitimate device with an expired token, or an attempt to impersonate a legitimate device. If the legitimate device is currently communicating correctly, this is likely an impersonation attempt.	
Failed Brute-force attempt	Low	Agent	Multiple unsuccessful login attempts identified. Potential Brute-force attack attempt failed on the device.	Review SSH Brute-force alerts and the activity on the device. No further action required.	

**MCT USE ONLY. STUDENT USE PROHIBITED**

Name	Severity	Data Source	Description	Suggested remediation steps	
Invalid SAS token signature	Low	IoT Hub	A SAS token used by a device has an invalid signature. The signature does not match either the primary or secondary key.	Review the alerts on the devices. No further action required.	
Local user added to one or more groups	Low	Agent	New local user added to a group on this device. Changes to user groups are uncommon, and can indicate a malicious actor may be collecting additional permissions.	Verify if the change is consistent with the permissions required by the affected user. If the change is inconsistent, escalate to your Information Security team.	
Local user deleted from one or more groups	Low	Agent	A local user was deleted from one or more groups. Malicious actors are known to use this method in an attempt to deny access to legitimate users or to delete the history of their actions.	Verify if the change is consistent with the permissions required by the affected user. If the change is inconsistent, escalate to your Information Security team.	

Name	Severity	Data Source	Description	Suggested remediation steps	
Local user deletion detected	Low	Agent	Deletion of a local user detected. Local user deletion is uncommon, a malicious actor may be trying to deny access to legitimate users or to delete the history of their actions.	Verify if the change is consistent with the permissions required by the affected user. If the change is inconsistent, escalate to your Information Security team.	

## Built-in alerts for IoT Hub

Severity	Name	Description	Suggested remediation
<b>Medium severity</b>			
New certificate added to an IoT Hub	Medium	A certificate named '{DescCertificateName}' was added to IoT Hub '{Descl0THubName}'. If this action was made by an unauthorized party, it may indicate malicious activity.	1. Make sure the certificate was added by an authorized party. 2. If it was not added by an authorized party, remove the certificate and escalate the alert to the organizational security team.
Certificate deleted from an IoT Hub	Medium	A certificate named '{DescCertificateName}' was deleted from IoT Hub '{Descl0THubName}'. If this action was made by an unauthorized party, it may indicate a malicious activity.	1. Make sure the certificate was removed by an authorized party. 2. If the certificate was not removed by an authorized party, add the certificate back, and escalate the alert to the organizational security team.
Unsuccessful attempt detected to add a certificate to an IoT Hub	Medium	There was an unsuccessful attempt to add certificate '{DescCertificateName}' to IoT Hub '{Descl0THubName}'. If this action was made by an unauthorized party, it may indicate malicious activity.	Make sure permissions to change certificates are only granted to authorized parties.

Severity	Name	Description	Suggested remediation
Unsuccessful attempt detected to delete a certificate from an IoT Hub	Medium	There was an unsuccessful attempt to delete certificate '{DescCertificateName}' from IoT Hub '{DescIoTHubName}'. If this action was made by an unauthorized party, it may indicate malicious activity.	Make sure permissions to change certificates are only granted to an authorized party.
<b>Low severity</b>			
Attempt to add or edit a diagnostic setting of an IoT Hub detected	Low	Attempt to add or edit the diagnostic settings of an IoT Hub has been detected. Diagnostic settings enable you to recreate activity trails for investigation purposes when a security incident occurs or your network is compromised. If this action was not made by an authorized party, it may indicate malicious activity.	1. Make sure the certificate was removed by an authorized party. 2. If the certificate was not removed by an authorized party, add the certificate back and escalate the alert to your information security team.
Attempt to delete a diagnostic setting from an IoT Hub detected	Low	There was '{DescAttemptStatusMessage}' attempt to add or edit diagnostic setting '{DescDiagnosticSettingName}' of IoT Hub '{DescIoTHubName}'. Diagnostic setting enables you to recreate activity trails for investigation purposes when a security incident occurs or your network is compromised. If this action was not made by an authorized party, it may indicate a malicious activity.	Make sure permissions to change diagnostics settings are granted only to an authorized party.

## Next steps

- Azure Security Center for IoT service Overview

- Learn how to Access your security data
- Learn more about Investigating a device

## Quickstart: Create custom alerts

Using custom security groups and alerts, takes full advantage of the end-to-end security information and categorical device knowledge to ensure better security across your IoT solution.

### Why use custom alerts?

You know your IoT devices best.

For customers who fully understand their expected device behavior, Azure Security Center for IoT allows you to translate this understanding into a device behavior policy and alert on any deviation from expected, normal behavior.

### Security groups

Security groups enable you to define logical groups of devices, and manage their security state in a centralized way.

These groups can represent devices with specific hardware, devices deployed in a certain location, or any other group suitable to your specific needs.

Security groups are defined by a device twin tag property named **SecurityGroup**. By default, each IoT solution on IoT Hub has one security group named **default**. Change the value of the **SecurityGroup** property to change the security group of a device.

For example:

```
{  
    "deviceId": "VM-Contoso12",  
    "etag": "AAAAAAAAAM=",  
    "deviceEtag": "ODA1BzA5QjM2",  
    "status": "enabled",  
    "statusUpdateTime": "0001-01-01T00:00:00",  
    "connectionState": "Disconnected",  
    "lastActivityTime": "0001-01-01T00:00:00",  
    "cloudToDeviceMessageCount": 0,  
    "authenticationType": "sas",  
    "x509Thumbprint": {  
        "primaryThumbprint": null,  
        "secondaryThumbprint": null  
    },  
    "version": 4,  
    "tags": {  
        "SecurityGroup": "default"  
    },
```

Use security groups to group your devices into logical categories. After creating the groups, assign them to the custom alerts of your choice, for the most effective end-to-end IoT security solution.

## Customize an alert

1. Open your IoT Hub.
2. Click **Custom alerts** in the **Security** section.
3. Choose a security group you wish to apply the customization to.
4. Click **Add a custom alert**.
5. Select a custom alert from the dropdown list.
6. Edit the required properties, click **OK**.
7. Make sure to click **SAVE**. Without saving the new alert, the alert is deleted the next time you close IoT Hub.

## Alerts available for customization

The following table provides a summary of alerts available for customization.

Severity	Name	Data Source	Description	Suggested remediation
Low	Custom alert - number of cloud to device messages in AMQP protocol is outside the allowed range	IoT Hub	Number of cloud to device messages (AMQP protocol) within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of rejected cloud to device messages in AMQP protocol is outside the allowed range	IoT Hub	Number of cloud to device messages (AMQP protocol) rejected by the device, within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of device to cloud messages in AMQP protocol is outside the allowed range	IoT Hub	The amount of device to cloud messages (AMQP protocol) within a specific time window is outside the currently configured and allowable range.	

Severity	Name	Data Source	Description	Suggested remediation
Low	Custom alert - number of direct method invokes is outside the allowed range	IoT Hub	The amount of direct method invokes within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of file uploads is outside the allowed range	IoT Hub	The amount of file uploads within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of cloud to device messages in HTTP protocol is outside the allowed range	IoT Hub	The amount of cloud to device messages (HTTP protocol) in a time window is not in the configured allowed range	
Low	Custom alert - number of rejected cloud to device messages in HTTP protocol is not in the allowed range	IoT Hub	The amount of cloud to device messages (HTTP protocol) within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of device to cloud messages in HTTP protocol is outside the allowed range	IoT Hub	The amount of device to cloud messages (HTTP protocol)within a specific time window is outside the currently configured and allowable range.	

MCT USE ONLY. STUDENT USE PROHIBITED

Severity	Name	Data Source	Description	Suggested remediation
Low	Custom alert - number of cloud to device messages in MQTT protocol is outside the allowed range	IoT Hub	The amount of cloud to device messages (MQTT protocol) within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of rejected cloud to device messages in MQTT protocol is outside the allowed range	IoT Hub	The amount of cloud to device messages (MQTT protocol) rejected by the device within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of device to cloud messages in MQTT protocol is outside the allowed range	IoT Hub	The amount of device to cloud messages (MQTT protocol) within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of command queue purges is outside the allowed range	IoT Hub	The amount of command queue purges within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of module twin updates is outside the allowed range	IoT Hub	The amount of module twin updates within a specific time window is outside the currently configured and allowable range.	

Severity	Name	Data Source	Description	Suggested remediation
Low	Custom alert - number of unauthorized operations is outside the allowed range	IoT Hub	The amount of unauthorized operations within a specific time window is outside the currently configured and allowable range.	
Low	Custom alert - number of active connections is outside the allowed range	Agent	Number of active connections within a specific time window is outside the currently configured and allowable range.	Investigate the device logs. Learn where the connection originated and determine if it is benign or malicious. If malicious, remove possible malware and understand source. If benign, add the source to the allowed connection list.
Low	Custom alert - outbound connection created to an IP that isn't allowed	Agent	An outbound connection was created to an IP that is outside your allowed IP list.	Investigate the device logs. Learn where the connection originated and determine if it is benign or malicious. If malicious, remove possible malware and understand source. If benign, add the source to the allowed IP list.
Low	Custom alert - number of failed local logins is outside the allowed range	Agent	The amount of failed local logins within a specific time window is outside the currently configured and allowable range.	

MCT USE ONLY. STUDENT USE PROHIBITED

Severity	Name	Data Source	Description	Suggested remediation
Low	Custom alert - login of a user that is not on the allowed user list	Agent	A local user outside your allowed user list, logged in to the device.	If you are saving raw data, navigate to your log analytics account and use the data to investigate the device, identify the source and then fix the allow/block list for those settings. If you are not currently saving raw data, go to the device and fix the allow/block list for those settings.
Low	Custom alert - a process was executed that is not allowed	Agent	A process that is not allowed was executed on the device.	If you are saving raw data, navigate to your log analytics account and use the data to investigate the device, identify the source and then fix the allow/block list for those settings. If you are not currently saving raw data, go to the device and fix the allow/block list for those settings.

## Next steps

Advance to the next article to learn how to deploy a security agent...

[!div class="nextstepaction"]

Deploy a security agent

# Enhance Protection with Azure Security Center for IoT Agents

## Introduction to Security Agents

Azure Security Center for IoT security agents offer enhanced security capabilities, such as monitoring remote connections, active applications, login events, and operating system configuration best practices. Security agents handle raw event collection from the device operating system, event aggregation to reduce cost, and configuration through a device module twin. Security messages are sent through your IoT Hub, into Azure Security Center for IoT analytics services.

A reference architecture for Linux and Windows security agents is provided with support for both C# and C.

You can use the following workflow to deploy and test your Azure Security Center for IoT security agents:

1. Enable Azure Security Center for IoT service to your IoT Hub
2. If your IoT Hub has no registered devices, Register a new device.
3. Create an azureiotsecurity security module for your devices.

To install the agent on an Azure simulated device instead of installing on an actual device, spin up a new Azure Virtual Machine (VM) in an available zone.

4. Deploy an Azure Security Center for IoT security agent on your IoT device, or new VM.
5. Follow the instructions for **trigger\_events** to run a harmless simulation of an attack.

The simulated attack provides an opportunity to verify that Azure Security Center for IoT alert are triggered as expected.

## Understand Security Agent Options

Security agents offer the same set of features for C and C#, and support for similar configuration options is provided for both language options.

The C-based security agent has a lower memory footprint, and is the ideal choice for devices with fewer available resources.

	C-based security agent	C#-based security agent	
Open-source	Available under MIT license in GitHub	Available under MIT license in GitHub	
Development language	C	C#	
Supported Windows platforms?	No	Yes	
Windows prerequisites		WMI	
Supported Linux platforms?	Yes, x64 and x86	Yes, x64 only	
Linux prerequisites	libunwind8, libcurl3, uuid-runtime, auditd, audispd-plugins	libunwind8, libcurl3, uuid-runtime, auditd, audispd-plugins, sudo, netstat, iptables	
Disk footprint	10.5 MB	90 MB	33 MB

	C-based security agent	C#-based security agent	
Authentication to IoT Hub	Yes	Yes	
Security data collection	Yes	Yes	
Event aggregation	Yes	Yes	
Remote configuration through security module twin	Yes	Yes	

## Security Agent Installation Guidelines

For Windows: The Install SecurityAgent.ps1 script must be executed from an Administrator PowerShell window.

For Linux: The InstallSecurityAgent.sh must be run as superuser. We recommend prefixing the installation command with "sudo".

## Choose an Agent “Flavor”

Answer the following questions about your IoT devices to select the correct agent:

- Are you using Windows Server or Windows IoT Core?

Deploy a C#-based security agent for Windows.

- Are you using a Linux distribution with x86 architecture?

Deploy a C-based security agent for Linux.

- Are you using a Linux distribution with x64 architecture?

Both agent flavors can be used. Deploy a C-based security agent for Linux and/or Deploy a C#-based security agent for Linux.

Both agent flavors offer the same set of features and support similar configuration options. See Security agent comparison to learn more.

## Supported platforms

The following list includes all currently supported platforms.

Azure Security Center for IoT agent	Operating System	Architecture
C	Ubuntu 16.04	x64
C	Ubuntu 18.04	x64
C	Debian 9	x64, x86
C#	Ubuntu 16.04	x64
C#	Ubuntu 18.04	x64
C#	Debian 9	x64
C#	Windows Server 2016	X64
C#	Windows 10 IoT Core, build 17763	x64

# Security Agent Authentication

Azure Security Center for IoT provides reference architecture for security agents that log, process, aggregate, and send security data through IoT Hub.

Security agents are designed to work in a constrained IoT environment, and are highly customizable in terms of values they provide when compared to the resources they consume.

Security agents support the following features:

- Collect raw security events from the underlying Operating System (Linux, Windows).
- Aggregate raw security events into messages sent through IoT Hub.
- Authenticate with existing device identity, or a dedicated module identity.
- Configure remotely through use of the **azureiotsecurity** module twin.

A security module is required for each device onboarded to Azure Security Center for IoT in the IoT Hub. To authenticate the device, Azure Security Center for IoT can use one of two methods:

- SecurityModule option
- Device option

## Authentication methods

You can use the following information to help you choose between the two methods for authentication:

- SecurityModule authentication mode

The agent is authenticated using the security module identity independently of the device identity. Use this authentication type if you would like the security agent to use a dedicated authentication method through security module (symmetric key only).

- Device authentication mode

In this method, the security agent first authenticates with the device identity. After the initial authentication, the Azure Security Center for IoT agent performs a REST call to the IoT Hub using the REST API with the authentication data of the device. The Azure Security Center for IoT agent then requests the security module authentication method and data from the IoT Hub. In the final step, the Azure Security Center for IoT agent performs an authentication against the Azure Security Center for IoT module.

Use this authentication type if you would like the security agent to reuse an existing device authentication method (self-signed certificate or symmetric key).

## Authentication methods known limitations

SecurityModule authentication mode only supports symmetric key authentication.

CA-Signed certificate is not supported by Device authentication mode.

## Security agent installation parameters

When deploying a security agent, authentication details must be provided as arguments. These arguments are documented in the following table.

Linux Parameter Name	Windows Parameter Name	Shorthand Parameter	Description	Options
authentication-identity	Authentication-Identity	aui	Authentication identity	<b>SecurityModule or Device</b>
authentication-method	Authentication-Method	aum	Authentication method	<b>SymmetricKey or SelfSignedCertificate</b>
file-path	FilePath	f	Absolute full path for the file containing the certificate or the symmetric key	
host-name	HostName	hn	FQDN of the IoT Hub	Example: ContosoIoTHub.azure-devices.net
device-id	DeviceId	di	Device ID	Example: MyDevice1
certificate-location-kind	CertificateLocationKind	cl	Certificate storage location	<b>LocalFile or Store</b>

When using the install security agent script, the following configuration is performed automatically. To edit the security agent authentication manually, edit the config file.

## Change authentication method after deployment

When deploying a security agent with an installation script, a configuration file is automatically created. To change authentication methods after deployment, manual editing of the configuration file is required.

### C#-based security agent

Edit *Authentication.config* with the following parameters:

```
<Authentication>
  <add key="deviceId" value="" />
  <add key="gatewayHostname" value="" />
  <add key="filePath" value="" />
  <add key="type" value="" />
  <add key="identity" value="" />
  <add key="certificateLocationKind" value="" />
</Authentication>
```

### C-based security agent

Edit *LocalConfiguration.json* with the following parameters:

```
"Authentication": {
    "Identity" : "",
    "AuthenticationMethod" : "",
    "FilePath" : "",
    "DeviceId" : "",
```

```
"HostName" : ""  
}
```

## Security Recommendations

Azure Security Center for IoT scans your Azure resources and IoT devices and provides security recommendations to reduce your attack surface. Security recommendations are actionable and aim to aid customers in complying to security best practices.

### Recommendations for IoT devices

Device recommendations provide insights and suggestions to improve device security posture.

Severity	Name	Data Source	Description
Medium	Open Ports on device	Agent	A listening endpoint was found on the device.
Medium	Permissive firewall policy found in one of the chains.	Agent	Allowed firewall policy found (INPUT/OUTPUT). Firewall policy should deny all traffic by default, and define rules to allow necessary communication to/from the device.
Medium	Permissive firewall rule in the input chain was found	Agent	A rule in the firewall has been found that contains a permissive pattern for a wide range of IP addresses or ports.
Medium	Permissive firewall rule in the output chain was found	Agent	A rule in the firewall has been found that contains a permissive pattern for a wide range of IP addresses or ports.
Medium	Operation system baseline validation has failed	Agent	Device doesn't comply with CIS Linux benchmarks.

### Operational recommendations for IoT devices

Operational recommendations provide insights and suggestions to improve security agent configuration.

Severity	Name	Data Source	Description
Low	Agent sends unutilized messages	Agent	10% or more of security messages were smaller than 4 KB during the last 24 hours.

Severity	Name	Data Source	Description
Low	Security twin configuration not optimal	Agent	Security twin configuration is not optimal.
Low	Security twin configuration conflict	Agent	Conflicts were identified in the security twin configuration.

## Recommendations for IoT Hub

Recommendation alerts provide insight and suggestions for actions to improve the security posture of your environment.

Severity	Name	Data Source	Description
High	Identical authentication credentials used by multiple devices	IoT Hub	IoT Hub authentication credentials are used by multiple devices. This may indicate an illegitimate device impersonating a legitimate device. Duplicate credential use increases the risk of device impersonation by a malicious actor.
Medium	Default IP filter policy should be deny	IoT Hub	IP filter configuration should have rules defined for allowed traffic, and should by default, deny all other traffic by default.
Medium	IP filter rule includes large IP range	IoT Hub	An allow IP filter rule source IP range is too large. Overly permissive rules can expose your IoT hub to malicious actors.
Low	Enable diagnostics logs in IoT Hub	IoT Hub	Enable logs and retain them for up to a year. Retaining logs enables you to recreate activity trails for investigation purposes when a security incident occurs or your network is compromised.

## Baseline and Custom Checks

A baseline establishes standard behavior for each device and makes it easier to establish unusual behavior or deviation from expected norms. Baseline custom checks establish a custom list of checks for each device baseline using the Module identity twin of the device.

### Setting baseline properties

1. In your IoT Hub, locate and select the device you wish to change.
2. Click on the device, and then click the azureiotsecurity module.
3. Click Module Identity Twin.
4. Upload the baseline custom checks file to the device.
5. Add baseline properties to the security module and click Save.

### Baseline custom check file example

To configure baseline custom checks:

```
"desired": {
  "ms_iotn:urn_azureiot_Security_SecurityAgentConfiguration": {
    "baselineCustomChecksEnabled": {
      "value" : true
    },
    "baselineCustomChecksFilePath": {
      "value" : "/home/user/full_path.xml"
    },
    "baselineCustomChecksFileHash": {
      "value" : "#hashexample!"
    }
  }
},
```

### Baseline custom check properties

Name	Status	Valid values	Default values	Description
baselineCustomChecksEnabled	Required: true	Valid values: Boolean	Default value: false	Max time interval before high priority messages is sent.
baselineCustomChecksFilePath	Required: true	Valid values: String, null	Default value: null	Full path of the baseline xml configuration

Name	Status	Valid values	Default values	Description
baselineCustom-ChecksFileHash	Required: true	Valid values: String, null	Default value: null	sha256sum of the xml configuration file. Use the sha256sum reference for additional information.

# About the Module 10 Labs

## Module Labs

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

- Lab 19: Implement Azure Security Center for IoT

**WARNING:** Be prepared for UI changes

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

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



# Module 11 Build an IoT Solution with Azure IoT Central

## Introduction to IoT Central

### What is Azure IoT Central?

**Note:** This training includes the use of features that are only available in the preview release of Azure IoT Central.

**Warning** The IoT Plug and Play capabilities in Azure IoT Central are currently in public preview. Don't use an IoT Plug and Play enabled IoT Central application template for production workloads. For production environments use an IoT central application created from a current, generally available, application template.

Azure IoT Central is an IoT app platform that reduces the burden and cost of developing, managing, and maintaining enterprise-grade IoT solutions. Choosing to build with Azure IoT Central gives you the opportunity to focus your time, money, and energy on transforming your business with IoT data, rather than just maintaining and updating a complex and continually evolving IoT infrastructure.

The easy-to-use interface makes it simple to monitor device conditions, create rules, and manage millions of devices and their data throughout their life cycle. Furthermore, it enables you to act on device insights by extending IoT intelligence into line-of-business applications.

**Note:** As an Azure IoT Developer it is important to know when and how to implement Azure IoT Central for your business or a customer.

### User Personas

There are four primary users, or personas, who interact with the Azure IoT Central application. You can think of these personas as four different people, or as one person performing four different roles:

- A builder is responsible for defining the types of devices that connect to the application and customizing the application for the operator.
- An operator manages the devices connected to the application.

- An administrator is responsible for tasks such as managing user roles and permissions within the application.
- A device developer creates the code that runs on a device connected to your application.

## Create your Azure IoT Central application

As a solution builder, you use Azure IoT Central to create a custom, cloud-hosted IoT solution for your organization. A custom IoT solution typically consists of:

- A cloud-based application that receives telemetry from your devices and enables you to manage those devices.
- Multiple devices running custom code connected to your cloud-based application.

You can quickly deploy a new IoT Central application and then customize it to your specific requirements in your browser. As a solution builder, you use the web-based tools to create a device template for the devices that connect to your application. A device template is the blueprint that defines the characteristics and behavior of a type of device such as the:

- Telemetry it sends.
- Business properties that an operator can modify.
- Device properties that are set by a device and are read-only in the application.
- Properties, that an operator sets, that determine the behavior of the device.

This device template includes:

- A device capability model that describes the capabilities a device should implement such as the telemetry it sends and the properties it reports.
- Cloud properties that aren't stored on the device.
- Customizations, dashboards, and forms that are part of your IoT Central application.

## Create device templates

IoT Plug and Play enables IoT Central to integrate devices without you writing any embedded device code. At the core of IoT Plug and Play, is a device capability model schema that describes device capabilities. In an IoT Central preview application, device templates use these IoT Plug and Play device capability models.

As a solution builder, you have several options for creating device templates:

- Design the device template in IoT Central and then implement its device capability model in your device code.
- Import a device capability model from the Azure Certified for IoT device catalog and then add any cloud properties, customizations, and dashboards your IoT Central application needs.
- Create a device capability model using Visual Studio code. Implement your device code from the model, and connect your device to your IoT Central application. IoT Central finds the device capability model from a repository and creates a simple device template for you.
- Create a device capability model using Visual Studio code. Implement your device code from the model. Manually import the device capability model into your IoT Central application and then add any cloud properties, customizations, and dashboards your IoT Central application needs.

As a solution builder, you can use IoT Central to generate code for test devices to validate your device templates.

## Customize the UI

As a solution builder, you can also customize the IoT Central application UI for the operators who are responsible for the day-to-day use of the application. Customizations that a solution builder can make include:

- Defining the layout of properties and settings on a device template.
- Configuring custom dashboards to help operators discover insights and resolve issues faster.
- Configuring custom analytics to explore time series data from your connected devices.

## Connect your devices

After the builder defines the types of devices that can connect to the application, a device developer creates the code to run on the devices. As a device developer, you use Microsoft's open-source Azure IoT SDKs to create your device code. These SDKs have broad language, platform, and protocol support to meet your needs to connect your devices to your Azure IoT Central application. The SDKs help you implement the following device capabilities:

- Create a secure connection.
- Send telemetry.
- Report status.
- Receive configuration updates.

## Azure IoT Edge devices

As well as devices created using the Azure IoT SDKs, you can also connect Azure IoT Edge devices to an IoT Central application. Azure IoT Edge lets you run cloud intelligence and custom logic directly on IoT devices managed by IoT Central. The IoT Edge runtime enables you to:

- Install and update workloads on the device.
- Maintain Azure IoT Edge security standards on the device.
- Ensure that IoT Edge modules are always running.
- Report module health to the cloud for remote monitoring.
- Manage communication between downstream leaf devices and an IoT Edge device, between modules on an IoT Edge device, and between an IoT Edge device and the cloud.

## Manage your application

Azure IoT Central applications are fully hosted by Microsoft, which reduces the administration overhead of managing your applications.

As an operator, you use the Azure IoT Central application to manage the devices in your Azure IoT Central solution. Operators do tasks such as:

- Monitoring the devices connected to the application.
- Troubleshooting and remediating issues with devices.

- Provisioning new devices.

As a solution builder, you can define custom rules and actions that operate over data streaming from connected devices. An operator can enable or disable these rules at the device level to control and automate tasks within the application.

Administrators manage access to your application with user roles and permissions.

## Quotas

Each Azure subscription has default quotas that could impact the scope of your IoT solution. Currently, IoT Central limits the number of applications you can deploy in a subscription to 10. If you need to increase this limit, contact Microsoft support.

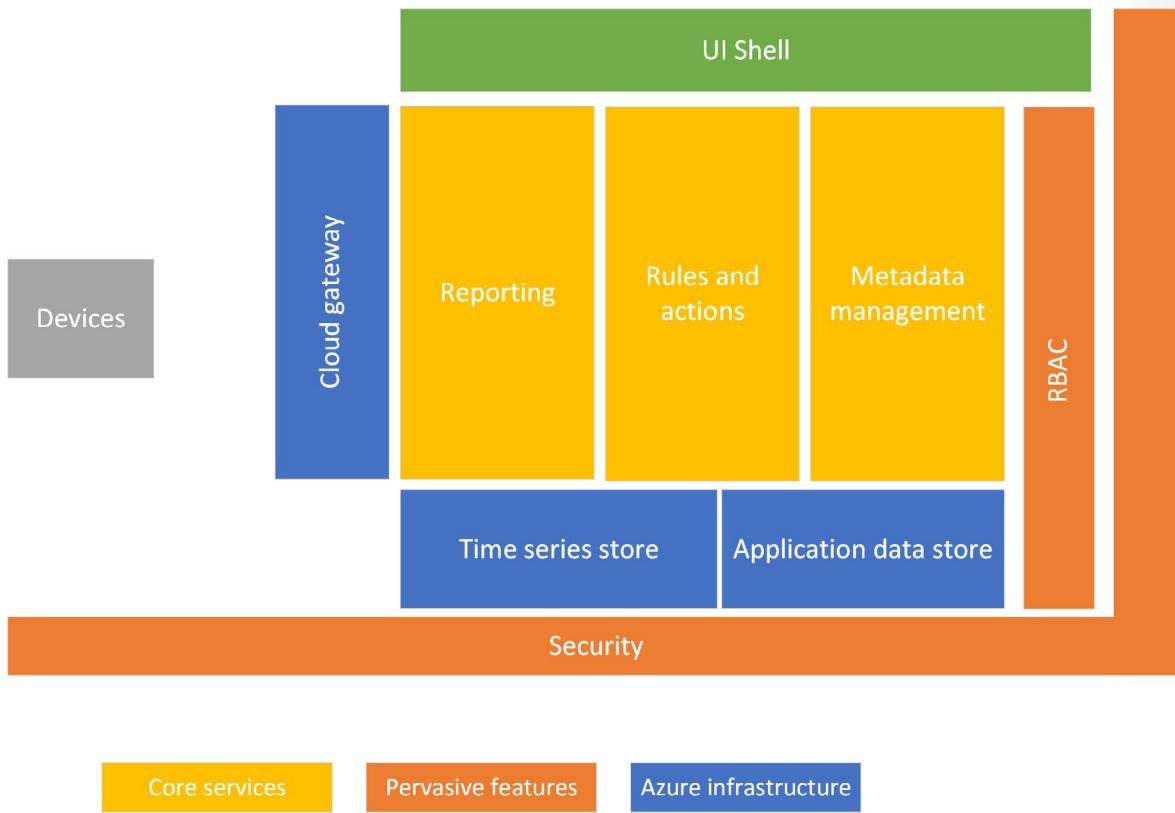
## Known issues

**Note:** These known issues only apply to the IoT Central preview applications.

- Rules don't support all actions (only email).
- For complex types - rules, analytics, and device groups aren't supported.
- Continuous data export doesn't support the Avro format (incompatibility).
- Simulated devices don't support all complex types.
- GeoJSON isn't currently supported.
- Map tile isn't currently supported.
- Jobs don't support complex types.
- Array schema types aren't supported.
- Application template export and application copy aren't supported.
- Only the C device SDK and the Node.js device and service SDKs are supported.
- It's only available in the United States and Europe locations.
- Device capability models must have all the interfaces defined inline in the same file.

## Azure IoT Central Architecture

Azure IoT Central app platform is built on top of the Azure IoT PaaS services and the architecture shown here:



## Devices

Devices exchange data with your Azure IoT Central application. A device can:

- Send measurements such as telemetry.
- Synchronize settings with your application.

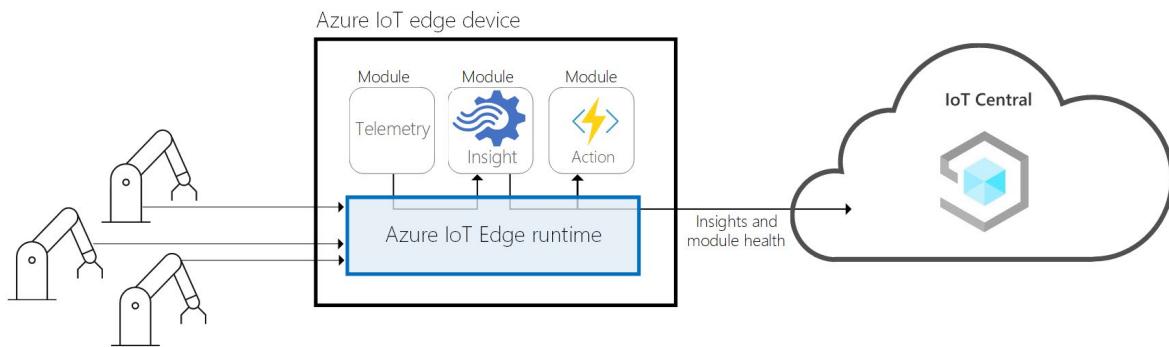
In Azure IoT Central, the data that a device can exchange with your application is specified in a device template.

## Azure IoT Edge devices

As well as devices created using the Azure IoT SDKs, you can also connect Azure IoT Edge devices to an IoT Central application. IoT Edge lets you run cloud intelligence and custom logic directly on IoT devices managed by IoT Central. The IoT Edge runtime enables you to:

- Install and update workloads on the device.
- Maintain IoT Edge security standards on the device.
- Ensure that IoT Edge modules are always running.
- Report module health to the cloud for remote monitoring.
- Manage communication between downstream leaf devices and an IoT Edge device, between modules on an IoT Edge device, and between an IoT Edge device and the cloud.

MCT USE ONLY. STUDENT USE PROHIBITED



IoT Central enables the following capabilities to for IoT Edge devices:

- Device templates to describe the capabilities of an IoT Edge device, such as:
  - Deployment manifest upload capability, which helps you manage a manifest for a fleet of devices.
  - Modules that run on the IoT Edge device.
  - The telemetry each module sends.
  - The properties each module reports.
  - The commands each module responds to.
  - The relationships between an IoT Edge gateway device capability model and downstream device capability model.
  - Cloud properties that aren't stored on the IoT Edge device.
  - Customizations, dashboards, and forms that are part of your IoT Central application.
- The ability to provision IoT Edge devices at scale using Azure IoT device provisioning service.
- Rules and actions.
- Custom dashboards and analytics.
- Continuous data export of telemetry from IoT Edge devices.

## IoT Edge device types

IoT Central classifies IoT Edge device types as follows:

- Standalone devices. An IoT Edge standalone device can have downstream leaf devices, but these devices aren't provisioned in IoT Central.
- Gateway devices with downstream devices. Both gateway device and downstream devices are provisioned in IoT Central.

## Cloud gateway

Azure IoT Central uses Azure IoT Hub as a cloud gateway that enables device connectivity.

## Data stores

Azure IoT Central stores application data in the cloud. Application data stored includes:

- Device templates.

- Device identities.
- Device metadata.
- User and role data.

Azure IoT Central uses a time series store for the measurement data sent from your devices.

## Analytics

The analytics service is responsible for generating the custom reporting data that the application displays. An operator can customize the analytics displayed in the application. The analytics service is built on top of Azure Time Series Insights and processes the measurement data sent from your devices.

## Rules and actions

Rules and actions work closely together to automate tasks within the application. A builder can define rules based on device telemetry such as the temperature exceeding a defined threshold. Azure IoT Central uses a stream processor to determine when the rule conditions are met. When a rule condition is met, it triggers an action defined by the builder. For example, an action can send an email to notify an engineer that the temperature in a device is too high.

IoT Central device templates provide a device definition that includes:

- **Device capability model:** Device capability models specify the capabilities of a device such as the telemetry it sends, the properties that define the device state, and the commands the device responds to. Device capabilities are organized into one or more interfaces.
- **Cloud properties:** Cloud properties specify the properties IoT Central stores for a device. These properties are only stored in IoT Central and are never sent to a device.
- **Views:** Views specify the dashboards and forms the builder creates to let the operator monitor and manage the devices.
- **Customizations:** Customizations let the builder override some of the definitions in the device capability model to make them more relevant to the IoT Central application.

An IoT Central application can have one or more simulated and real devices based on each device template.

## Data export

In an Azure IoT Central application, you can continuously export your data to your own Azure Event Hubs and Azure Service Bus instances. You can also periodically export your data to your Azure Blob storage account. IoT Central can export measurements, devices, and device templates.

## Batch device updates

In an Azure IoT Central application, you can create and run jobs to manage connected devices. These jobs let you do bulk updates to device properties or settings, or run commands. For example, you can create a job to increase the fan speed for multiple refrigerated vending machines.

## Role-based access control (RBAC)

An administrator can define access rules for an Azure IoT Central application using the predefined roles. An administrator can assign users to roles that determine what areas of the application the user has access to.

## Security

Security features within Azure IoT Central include:

- Data is encrypted in transit and at rest.
- Authentication is provided either by Azure Active Directory or Microsoft Account. Two-factor authentication is supported.
- Full tenant isolation.
- Device level security.

## UI shell

The UI shell is a modern, responsive, HTML5 browser-based application. An administrator can customize the UI of the application by applying custom themes and modifying the help links to point to your own custom help resources.

An operator can create personalized application dashboards. You can have several dashboards that display different data and switch between them.

## Introduction to Application Templates

Application templates in Azure IoT Central are a tool to help solution builders kickstart their IoT solution development. You can use app templates for everything from getting a feel for what is possible, to fully customizing and your application for resale to your customers.

Application templates consist of:

- Sample operator dashboards
- Sample device templates
- Simulated devices producing real-time data
- Pre-configured rules and jobs
- Rich documentation including tutorials and how-tos

## Available Templates

Azure IoT Central is an industry agnostic application platform, however, industry focused templates are available and more are on the way:

- Retail
  - Connected logistics
  - Digital distribution center
  - In-store analytics - condition monitoring
  - In-store analytics - checkout

- Smart Inventory Management
- Energy
  - Smart meter monitoring
  - Solar panel monitoring
- Government
  - Connected waste management
  - Water consumption monitoring
  - Water quality monitoring
- Healthcare
  - Continuous patient monitoring

## Connecting Devices

Azure IoT Central uses the Azure IoT Hub Device Provisioning service (DPS) to manage all device registration and connection.

Using DPS enables:

- IoT Central to support onboarding and connecting devices at scale.
- You to generate device credentials and configure the devices offline without registering the devices through IoT Central UI.
- Devices to connect using shared access signatures (SAS).
- Devices to connect using industry-standard X.509 certificates.
- You to use your own device IDs to register devices in IoT Central. Using your own device IDs simplifies integration with existing back-office systems.
- A single, consistent way to connect devices to IoT Central.

### Connect a single device

This approach is useful when you're experimenting with IoT Central or testing devices. You can use the device connection information from your IoT Central application to connect a device to your IoT Central application using the Device Provisioning Service (DPS).

### Connect devices at scale using SAS

To connect devices to IoT Central at scale using SAS, you need to register and then set up the devices.

### Register devices in bulk

To register a large number of devices with your IoT Central application, use a CSV file to import device IDs and device names. Once your devices are registered, you can export a CSV file from your IoT Central application that includes the connection information for the imported devices (including device keys and X509 certificate thumbprints).

## Set up your devices

Use the connection information from the export file in your device code to enable your devices to connect and send data to IoT to your IoT Central application.

## Connect devices using X.509 certificates

In a production environment, using X.509 certificates is the recommended device authentication mechanism for IoT Central.

The following steps describe how to connect devices to IoT Central using X.509 certificates:

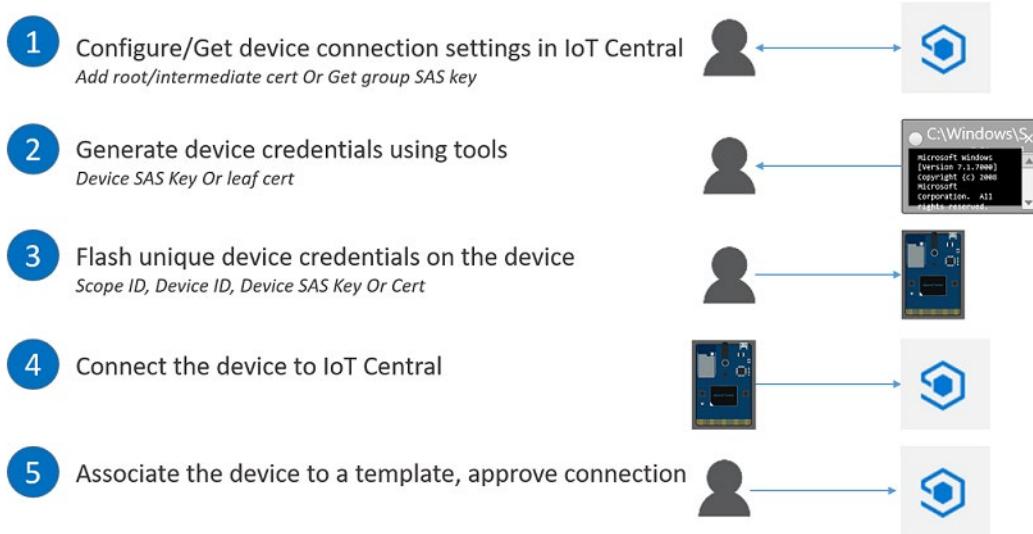
1. In your IoT Central application, add and verify the intermediate or root X.509 certificate you're using to generate device certificates.
  - In the UI menu, navigate to Administration > Device Connection > Certificates (X.509) and add X.509 root or intermediate certificate you're using to generate the leaf device certificates.
  - It is important to verify that the uploader of the certificate has the certificate's private key.
    - To generate a verification code, select the button next to Verification Code.
    - Use the verification code that you generated to create an X.509 verification certificate.
    - Save the certificate as a .cer file.
    - Upload the signed verification certificate and select Verify.
2. Use a CSV file to import and register devices in your IoT Central application.
3. Set up your devices.

Generate the leaf certificates using the uploaded root certificate. Use the Device ID as the CNAME value in the leaf certificates. The device ID should be all lower case. Then program your devices with provisioning service information. When a device is switched on for the first time, it retrieves its connection information for your IoT Central application from DPS.

## Connect without registering devices

A key scenario IoT Central enables is for OEMs to mass manufacture devices that can connect to an IoT Central application without first being registered. A manufacturer must generate suitable credentials, and configure the devices in the factory. When a device turns on for the first time, it connects automatically to an IoT Central application. An IoT Central operator must approve the device before it can start sending data.

The following diagram outlines this flow:



The following steps describe this process in more detail. The steps differ slightly depending on whether you're using SAS or X.509 certificates for device authentication:

1. Configure your connection settings:
  - **X.509 Certificates:** Add and verify the root/intermediate certificate and use it to generate the device certificates in the following step.
  - **SAS:** Copy the primary key. This key is the group SAS key for the IoT Central application. Use the key to generate the device SAS keys in the following step.
2. Generate your device credentials
  - **Certificates X.509:** Generate the leaf-certificates for your devices using the root or intermediate certificate you added to your IoT Central application. Make sure you use the lower-case **Device ID** as the CNAME in the leaf certificates. For testing purposes only, use this command-line tool to generate device certificates <https://github.com/Azure/azure-iot-sdk-c/blob/master/tools/CACertificates/CACertificateOverview.md>.
  - **SAS:** Use this command line tool to generate device SAS keys <https://www.npmjs.com/package/dps-keygen>. Use the group **Primary Key** from the previous step. The Device ID must be lower-case.
3. To set up your devices, flash each device with the **Scope ID**, **Device ID**, and **X.509 device certificate** or **SAS key**.
4. Then turn on the device for it to connect to your IoT Central application. When you switch on a device, it first connects to DPS to retrieve its IoT Central registration information.
5. The connected device initially shows up as **Unassociated** on the **Devices** page. The device provisioning status is **Registered**. **Migrate** the device to the appropriate device template and approve the device to connect to your IoT Central application. The device can then retrieve a connection string from IoT Hub and start sending data. Device provisioning is now complete and the provisioning status is now **Provisioned**.

## Individual enrollment based device connectivity

For customers connecting devices that have authentication credentials that are per device individual enrollment is the option. An individual enrollment is an entry for a single device that may connect. Individual enrollments may use either X.509 leaf certificates or SAS tokens (from a physical or virtual TPM) as attestation mechanisms. The device ID (aka registration ID) in an individual enrollment is alphanumeric, lowercase, and may contain hyphens.

**Note:** When you create an Individual enrollment for a device it takes precedence over the default Group enrollment based attestations (SAS, X509) in your app.

## Creating individual enrollments

IoT Central supports the following attestation mechanisms

1. **Symmetric key attestation:** Symmetric key attestation is a simple approach to authenticating a device with a Device Provisioning Service instance. To create an individual enrollment with Symmetric keys; open the Connect dialog, select Individual Enrollment and Mechanism "SAS" and input the Primary and Secondary keys. SAS keys must be base64 encoded. Here is the link to code samples to help write your device code to provision devices using Symmetric keys and individual enrollments.
2. **X.509 certificates:** X.509 certificates as the title suggests is a cert based attestation mechanism, an excellent way to scale production. To create an individual enrollment with Symmetric keys select Individual Enrollment and Mechanism "X.509" and upload the primary and secondary certificates and save to create the enrollment. Device certificates used with an Individual enrollment entry have a requirement that the Subject Name must be set to the Device ID (aka registration ID) of the Individual Enrollment entry.
3. **TPM attestation:** TPM stands for Trusted Platform Module and is a type of hardware security module (HSM) and is one of the most secure ways to connect your devices. This article assumes you are using a discrete, firmware, or integrated TPM. Software emulated TPMs are well-suited for prototyping or testing, but they do not provide the same level of security as discrete, firmware, or integrated TPMs do. We do not recommend using software TPMs in production. To create an individual enrollment with Symmetric keys select Individual Enrollment and Mechanism "TPM" and input the endorsement keys to create the enrollment. For more information about types of TPMs, you can learn more about TPM attestation here. Here is the link to code samples to help write your device code to provision devices using TPM. To create a TPM based attestation, type in the endorsement key and save.

## Device status

When a real device connects to your IoT Central application, its device status changes as follows:

1. The device status is first **Registered**. This status means the device is created in IoT Central, and has a device ID. A device is registered when:
  - A new real device is added on the **Devices** page.
  - A set of devices is added using **Import** on the **Devices** page.
2. The device status changes to **Provisioned** when the device that connected to your IoT Central application with valid credentials completes the provisioning step. In this step, the device retrieves a connection string from IoT Hub. The device can now connect to IoT Hub and start sending data.
3. An operator can block a device. When a device is blocked, it can't send data to your IoT Central application. Blocked devices have a status of **Blocked**. An operator must reset the device before it can

resume sending data. When an operator unblocks a device the status returns to its previous value, **Registered** or **Provisioned**.

4. The device status is **Waiting for Approval** which means the **Auto Approve** option is disabled and requires all the devices connecting to IoT Central be explicitly approved by an operator. Devices not registered manually on the **Devices** page, but connected with valid credentials will have the device status **Waiting for Approval**. Operators can approve these devices from the **Devices** page using the **Approve** button.
5. The device status is **Unassociated** which means that the devices connecting to IoT Central do not have a Device Template associated to them. This typically happens in the following scenarios:
  - A set of devices is added using **Import** on the **Devices** page without specifying the Device Template
  - Devices not registered manually on the **Devices** page connected with valid credentials but without specifying the Template ID during registration.
  - The Operator can associate a device to a Template from the **Devices** page using the **Migrate** button.

# Create and Manage Device Templates

## Introduction to Device Templates

A device template is a blueprint that defines the characteristics and behaviors of a type of device that connects to an Azure IoT Central application.

A device template contains:

- A device capability model that specifies the telemetry, properties, and commands that the device implements. These capabilities are organized into one or more interfaces.
- Cloud properties that define information that your IoT Central application stores about your devices. For example, a cloud property might record the date a device was last serviced. This information is never shared with the device.
- Customizations let the builder override some of the definitions in the device capability model. For example, the builder can override the name of a device property. Property names appear in IoT Central dashboards and forms.
- Dashboards and forms let the builder create a UI that lets operators monitor and manage the devices connected to your application.

## Device Template Example

As an example, a builder could create a device template for a connected fan that has the following characteristics:

- Sends temperature telemetry
- Sends location property
- Sends fan motor error events
- Sends fan operating state
- Provides a writeable fan speed property
- Provides a command to restart the device
- Gives you an overall view of the device via a dashboard

From this device template, an operator can create and connect real fan devices. All these fans have measurements, properties, and commands that operators use to monitor and manage them. Operators use the device dashboards and forms to interact with the fan devices.

**Note:** Only builders and administrators can create, edit, and delete device templates. Any user can create devices on the Devices page from existing device templates.

## Support for IoT Plug and Play

IoT Plug and Play enables IoT Central to integrate devices, without you writing any embedded device code. At the core of IoT Plug and Play is a device capability model schema that describes device capabilities. In an IoT Central Preview application, device templates use these IoT Plug and Play device capability models.

## Creating a device template

As a builder, you have several options for creating device templates:

- Design the device template in IoT Central, and then implement its device capability model in your device code.
- Import a device capability model from the Azure Certified for IoT device catalog. Then add any cloud properties, customizations, and dashboards your IoT Central application needs.
- Create a device capability model by using Visual Studio Code. Implement your device code from the model. Manually import the device capability model into your IoT Central application, and then add any cloud properties, customizations, and dashboards your IoT Central application needs.
- Create a device capability model by using Visual Studio Code. Implement your device code from the model, and connect your real device to your IoT Central application by using a device-first connection. IoT Central finds and imports the device capability model from the public repository for you. You can then add any cloud properties, customizations, and dashboards your IoT Central application needs to the device template.

### Create a device template from the device catalog

As a builder, you can quickly start building out your solution by using an IoT Plug and Play certified device. See the list in the Azure IoT Device Catalog. IoT Central integrates with the device catalog so you can import a device capability model from any of these IoT Plug and Play certified devices. To create a device template from one of these devices in IoT Central:

1. Go to the Device Templates page in your IoT Central application.
2. Select + New, and then select any of the IoT Plug and Play certified devices from the catalog. IoT Central creates a device template based on this device capability model.
3. Add any cloud properties, customizations, or views to your device template.
4. Select Publish to make the template available for operators to view and connect devices.

### Create a device template from scratch

To create a device template in IoT Central:

1. Go to the Device Templates page in your IoT Central application.
2. Select + New > Custom.
3. Enter a name for your template, such as Environmental Sensor.
4. Press Enter. IoT Central creates an empty device template.

### Manage a device template

You can rename or delete a template from the template's home page.

After you've added a device capability model to your template, you can publish it. Until you've published the template, you can't connect a device based on this template for your operators to see in the Devices page.

# Create a Device Capability Model

A device capability model describes an IoT Plug and Play device and defines the set of interfaces implemented by the device. A device capability model typically corresponds to a physical device, product, or SKU. You use the Digital Twin Definition Language to define a device capability model.

## Approaches for Creating a Capability Model

To create a device capability model, you can:

- Use IoT Central to create a custom model from scratch.
- Import a model from a JSON file. A device builder might have used Visual Studio Code to author a device capability model for your application.
- Select one of the devices from the Device Catalog. This option imports the device capability model that the manufacturer has published for this device. A device capability model imported like this is automatically published.

## Manage a Capability Model

After you create a device capability model, you can:

- Add interfaces to the model. A model must have at least one interface.
- Edit model metadata, such as its ID, namespace, and name.
- Delete the model.

## Create an Interface

A device capability must have at least one interface. An interface is a reusable collection of capabilities.

To create an interface:

1. Go to your device capability model, and choose + Add Interface.
2. On the Select an Interface page, you can:
  - Create a custom interface from scratch.
  - Import an existing interface from a file. A device builder might have used Visual Studio Code to author an interface for your device.
  - Choose one of the standard interfaces, such as the Device Information interface. Standard interfaces specify the capabilities common to many devices. These standard interfaces are published by Azure IoT, and can't be versioned or edited.
3. After you create an interface, choose Edit Identity to change the display name of the interface.
4. If you choose to create a custom interface from scratch, you can add your device's capabilities. Device capabilities are telemetry, properties, and commands.

## Telemetry

Telemetry is a stream of values sent from the device, typically from a sensor. For example, a sensor might report the ambient temperature.

The following table shows the configuration settings for a telemetry capability:

MCT USE ONLY. STUDENT USE PROHIBITED

Field	Description
Display Name	The display name for the telemetry value used on dashboards and forms.
Name	The name of the field in the telemetry message. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary.
Capability Type	Telemetry.
Semantic Type	The semantic type of the telemetry, such as temperature, state, or event. The choice of semantic type determines which of the following fields are available.
Schema	The telemetry data type, such as double, string, or vector. The available choices are determined by the semantic type. Schema isn't available for the event and state semantic types.
Severity	Only available for the event semantic type. The severities are Error, Information, or Warning.
State Values	Only available for the state semantic type. Define the possible state values, each of which has display name, name, enumeration type, and value.
Unit	A unit for the telemetry value, such as mph, %, or °C.
Display Unit	A display unit for use on dashboards and forms.
Comment	Any comments about the telemetry capability.
Description	A description of the telemetry capability.

## Properties

Properties represent point-in-time values. For example, a device can use a property to report the target temperature it's trying to reach. You can set writeable properties from IoT Central.

The following table shows the configuration settings for a property capability:

Field	Description
Display Name	The display name for the property value used on dashboards and forms.
Name	The name of the property. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary.
Capability Type	Property.
Semantic Type	The semantic type of the property, such as temperature, state, or event. The choice of semantic type determines which of the following fields are available.
Schema	The property data type, such as double, string, or vector. The available choices are determined by the semantic type. Schema isn't available for the event and state semantic types.

Field	Description
Writeable	If the property isn't writeable, the device can report property values to IoT Central. If the property is writeable, the device can report property values to IoT Central and IoT Central can send property updates to the device.
Severity	Only available for the event semantic type. The severities are Error, Information, or Warning.
State Values	Only available for the state semantic type. Define the possible state values, each of which has display name, name, enumeration type, and value.
Unit	A unit for the property value, such as mph, %, or °C.
Display Unit	A display unit for use on dashboards and forms.
Comment	Any comments about the property capability.
Description	A description of the property capability.

## Commands

You can call device commands from IoT Central. Commands optionally pass parameters to the device and receive a response from the device. For example, you can call a command to reboot a device in 10 seconds.

The following table shows the configuration settings for a command capability:

Field	Description
Display Name	The display name for the command used on dashboards and forms.
Name	The name of the command. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary.
Capability Type	Command.
Command	SynchronousExecutionType
Comment	Any comments about the command capability.
Description	A description of the command capability.
Request	If enabled, a definition of the request parameter, including: name, display name, schema, unit, and display unit.
Response	If enabled, a definition of the command response, including: name, display name, schema, unit, and display unit.

## Manage an Interface

If you haven't published the interface, you can edit the capabilities defined by the interface. After you publish the interface, if you want to make any changes, you'll need to create a new version of the device template and version the interface. You can make changes that don't require versioning, such as display names or units, in the Customize section of the UI.

You can also export the interface as a JSON file if you want to reuse it in another capability model.

## Add Cloud Properties to a Device Template

Use cloud properties to store information about devices in IoT Central. Cloud properties are never sent to a device. For example, you can use cloud properties to store the name of the customer who has installed the device, or the device's last service date.

The following table shows the configuration settings for a cloud property:

Field	Description
Display Name	The display name for the cloud property value used on dashboards and forms.
Name	The name of the cloud property. IoT Central generates a value for this field from the display name, but you can choose your own value if necessary.
Semantic Type	The semantic type of the property, such as temperature, state, or event. The choice of semantic type determines which of the following fields are available.
Schema	The cloud property data type, such as double, string, or vector. The available choices are determined by the semantic type.

## Add Customizations to a Device Template

Use customizations when you need to modify an imported interface or add IoT Central-specific features to a capability. You can only customize fields that don't break interface compatibility. For example, you can:

- Customize the display name and units of a capability.
- Add a default color to use when the value appears on a chart.
- Specify initial, minimum, and maximum values for a property.

You can't customize the capability name or capability type. If there are changes you can't make in the Customize section, you'll need to version your device template and interface to modify the capability.

## Generate default views

Generating default views is a quick way to visualize your important device information. You have up to three default views generated for your device template:

- Commands provides a view with device commands, and allows your operator to dispatch them to your device.
- Overview provides a view with device telemetry, displaying charts and metrics.
- About provides a view with device information, displaying device properties.

After you've selected Generate default views, you see that they have been automatically added under the Views section of your device template.

## Add Dashboards to a Device Template

Add dashboards to a device template to enable operators to visualize a device by using charts and metrics. You can have multiple dashboards for a device template.

To add a dashboard to a device template:

1. Go to your device template, and select **Views**.
2. Choose **Visualizing the Device**.
3. Enter a name for your dashboard in **Dashboard Name**.
4. Add tiles to your dashboard from the list of static, property, cloud property, telemetry, and command tiles.

Drag and drop the tiles you want to add to your dashboard.

To plot multiple telemetry values on a single chart tile, select the telemetry values, and then select **Combine**.

You can configure each tile to customize how it displays data.

5. Arrange and resize the tiles on your dashboard.
6. Save the changes.

## Configure preview device to view dashboard

To view and test your dashboard, select Configure preview device. This enables you to see the dashboard as your operator sees it after it's published. Use this option to validate that your views show the correct data. You can choose from the following:

- No preview device.
- The real test device you've configured for your device template.
- An existing device in your application, by using the device ID.

## Add Forms to a Device Template

Add forms to a device template to enable operators to manage a device by viewing and setting properties. Operators can only edit cloud properties and writeable device properties. You can have multiple forms for a device template.

To add a form to a device template:

1. Go to your device template, and select Views.
2. Choose Editing Device and Cloud data.
3. Enter a name for your form in Form Name.
4. Select the number of columns to use to lay out your form.
5. Add properties to an existing section on your form, or select properties and choose Add Section. Use sections to group properties on your form. You can add a title to a section.
6. Configure each property on the form to customize its behavior.
7. Arrange the properties on your form.
8. Save the changes.

## Version a Device Template

Once you have published your device template, the device capability model shows as Published with lock icons next to the model. In order to make changes to the device capability model, you will need to create a new version of the device template. Meanwhile the cloud properties, customizations, and views can all be edited at any time without needing to version the device template. Once you have saved any of these changes, you can publish the device template to make the latest changes available for the operator to view in Device Explorer.

### Add customizations to the device template without versioning

Certain elements of your device capabilities can be edited without needing to version your device template and interfaces. For example, some of these fields include display name, semantic type, minimum value, maximum value, decimal places, color, unit, display unit, comment, and description. To add one of these customizations:

1. Go to the Device Templates page.
2. Select the device template you wish to customize.
3. Choose the Customize tab.
4. All of the capabilities defined in your device capability model will be listed here. All of the fields you can edit here can be saved and used across your application, without needing to version your device template. If there are fields you wish to edit that are read-only, you will need to version your device template to change these. Select a field you wish to edit and enter in any new values.
5. Click Save. Now these values will override anything that was initially saved in your device template and will be used across the application.

### Versioning a Device Template

Creating a new version of your device template will create a draft version of the template where the device capability model can be edited. Any published interfaces will remain published until they are individually versioned. In order to modify a published interface, you must first create a new device template version.

The device template should only be versioned when you are trying to edit a part of the device capability model that you can not edit in the customizations section of the device template.

In order to version a device template:

1. Go to the Device Templates page.
2. Select the device template you are trying to version.
3. Click the Version button at the top of the page and give the template a new name. We have suggested a new name for you which can be edited.
4. Click Create.

Now your device template is in draft mode. You will see your interfaces are still locked and must be individually versioned to be edited.

## Versioning an Interface

Versioning an interface allows you to add, update, and remove the capabilities inside the interface you had already created.

In order to version an interface:

1. Go to the Device Templates page.
2. Select the device template you have in a draft mode.
3. Select the interface that is in published mode that you wish to version and edit.
4. Click the Version button at the top of the interface page.
5. Click Create.

Now your interface is in draft mode. You will be able to add or edit capabilities to your interface without breaking existing customizations and views.

**Note:** Standard interfaces published by Azure IoT can not be versioned or edited. These standard interfaces are used for device certification.

**Note:** Once the interface has been published, you can not delete any of its capabilities even in a draft mode. Capabilities can only be edited or added to the interface in draft mode.

## Migrate a device across device template versions

You can create multiple versions of the device template. Over time, you will have multiple connected devices using these device templates. You can migrate devices from one version of your device template to another. The following steps describe how to migrate a device:

1. Go to the Device Explorer page.
2. Select the device you need to migrate to another version.
3. Choose Migrate.
4. Select the device template with the version number you want to migrate the device to and choose Migrate.

# Manage Devices in Azure IoT Central

## Manage Your Devices

As an Azure IoT Central operator, you can:

- Use the Devices page to view, add, and delete devices connected to your Azure IoT Central application.
- Maintain an up-to-date inventory of your devices.
- Keep your device metadata up-to-date by changing the values stored in the device properties from your views.
- Control the behavior of your devices by updating a setting on a specific device from your views.

## View Your Devices

To view an individual device:

1. Choose **Devices** on the left pane.

The Devices page provides access to your devices and device templates.

2. Choose a device template.

The right-hand pane of the Devices page lists the devices created from the chosen device template.

3. Choose an individual device to see the device details page for that device.

## Add a Device

To add a device to your Azure IoT Central application:

1. Choose **Devices** on the left pane.
2. Choose the device template from which you want to create a device.
3. Choose **+ New**.
4. Turn the **Simulated** toggle to **On** or **Off**.

A real device is for a physical device that you connect to your Azure IoT Central application. A simulated device has sample data generated for you by Azure IoT Central.

5. Click **Create**.

Your new device now appears in your device list for this template. Select the device to see the device details page that contains all views for the device.

## Import devices

To connect large number of devices to your application, you can bulk import devices from a CSV file. The CSV file should have the following columns and headers:

- **IOTC\_DeviceID** - the device ID should be all lowercase.
- **IOTC\_DeviceName** - this column is optional.

To bulk-register devices in your application:

1. Choose **Devices** on the left pane.
2. On the left panel, choose the device template for which you want to bulk create the devices.

**Note:** If you don't have a device template yet then you can import devices under All devices and register them without a template. After devices have been imported, you can then migrate them to a template.

3. Select **Import**.
4. Select the CSV file that has the list of Device IDs to be imported.

Device import starts once the file has been uploaded. You can track the import status in the Device Operations panel. This panel appears automatically after the import starts or you can access it through the bell icon in the top right-hand corner.

Once the import completes, a success message is shown in the Device Operations panel.

If the device import operation fails, you see an error message on the Device Operations panel. A log file capturing all the errors is generated that you can download.

## Migrating devices to a template

If you register devices by starting the import under All devices, then the devices are created without any device template association. Devices must be associated with a template to explore the data and other details about the device.

Follow these steps to associate devices with a template:

1. Choose **Devices** on the left pane.
2. On the left panel, choose **All devices**.
3. Select the devices you want to associate with a template.

The display grid showing your devices includes a Device Template column. A value of "Unassociated" is assigned for any devices that are not currently associated with a device template.

4. Select **Migrate**.  
A list of available device templates will be displayed.
5. Choose the desired template and select **Migrate**.

The selected devices will now be associated with the device template that you chose.

## Export devices

To connect a real device to IoT Central, you need its connection string. You can export device details in bulk to get the information you need to create device connection strings. The export process creates a CSV file with the device identity, device name, and keys for all the selected devices.

To bulk export devices from your application:

1. Choose **Devices** on the left pane.
2. On the left panel, choose the device template from which you want to export the devices.
3. Select the devices that you want to export, and then select the **Export** action.

The export process starts. You can track the status using the Device Operations panel.

When the export completes, a success message is shown along with a link to download the generated file.

4. To download the file to a local folder on the disk, select the **Download File** link.

The exported CSV file contains the following columns: device ID, device name, device keys, and X509 certificate thumbprints

## Delete a device

To delete either a real or simulated device from your Azure IoT Central application:

1. Choose **Devices** on the left pane.
2. Choose the device template of the device you want to delete.  
Use the filter tools to filter and search for your devices.
3. Check the box next to the devices that you want to delete.
4. Choose **Delete**.

You can track the status of this deletion in your **Device Operations** panel.

## Change a property

Cloud properties are the device metadata associated with the device, such as city and serial number.

Writeable properties control the behavior of a device. In other words, they enable you to provide inputs to your device. Device properties are set by the device and are read-only within IoT Central. You can view and update properties on the **Device Details** views for your device.

1. Choose **Devices** on the left pane.
2. Choose the device template of the device whose properties you want to change, and then select the target device.
3. Choose the view that contains properties for your device.

This view enables you to see the properties of your device and their current values. Cloud properties and writeable properties have editable fields, while device properties are read-only. For writeable properties, you can see their sync status at the bottom of the field. A Save button is located at the top of the page.

4. Modify the properties to the values you need.

You can modify multiple properties one at a time and update them all at the same time.

5. Choose **Save**.

If you saved writeable properties, the values are sent to your device. When the device confirms the change for the writeable property, the status returns back to synced. If you saved a cloud property, the value is updated.

## Introduction to Device Groups

A device group is a list of devices that are grouped together because they match some specified criteria. Device groups help you manage, visualize, and analyze devices at scale by grouping devices into smaller, logical groups. For example, you can create a device group to list all the air conditioner devices in Seattle to enable a technician to find the devices for which they're responsible.

## To Create a Device Group

1. Choose Device Groups on the left pane.
2. Select **+ New**.
3. Give your device group a name.

You can also add a description for the device group.

**Note:** A device group can only contain devices from a single device template.

4. Create a query to identify the devices that will belong to the device group.

You can add multiple queries and devices that meet all the criteria are placed in the device group. The device group you create is accessible to anyone who has access to the application, so anyone can view, modify, or delete the device group.

**Note:** The device group is a dynamic query. Every time you view the list of devices, there may be different devices in the list. The list depends on which devices currently meet the criteria of the query.

5. Click **Save**.

## Analytics on a Device Group

You can use Analytics with a device group to analyze the telemetry from the devices in the group.

To analyze the telemetry for a device group:

1. Choose Analytics on the left pane.
2. Select a device group, and then select desired telemetry types.

You can specify an aggregation type for telemetry values. The default aggregation type is **Average**.

You can use **Split by** to change how the aggregate data is shown. For example, if you split by device ID you see a plot for each device when you select Analyze.

3. Select **Analyze** to view a plot of the specified telemetry values.

## Manage Devices at Scale using Jobs

You can use Microsoft Azure IoT Central to manage your connected devices at scale using jobs. Jobs let you do bulk updates to device properties, settings, and commands. This article walks you through how to get started using jobs in your own application.

**Note:** Jobs for Azure IoT Edge devices is currently not supported.

## Create and run a job

1. Navigate to **Jobs** from the navigation pane.
2. Select **+ New** to create a new job.
3. Enter a name and description to identify the job you're creating.
4. Select the device set you want your job to apply to.

After selecting the device set, you see the right-hand side populate with the devices in the device set.

If you select a broken device set, no devices display and you see a message that your device set is broken.

5. Choose the **Job type** that you want to define

The options are Settings, Properties, or Commands.

6. Select + next to the type of job selected, and then add the job operations.
7. On the right-hand side, choose the devices you'd like to run the job on.

By selecting the top check box, all devices are selected in the entire device set. By selecting the check box near Name, all devices on the current page are selected.

The following screenshot illustrates the process in the UI:

- We select a Settings job type, and then define an operation to set the fan speed for a refrigerated vending machine.
- Then, on the right-hand side, we select all of the available refrigerator devices

The screenshot shows the 'Jobs' section of the Azure IoT Central interface. A new job is being created with the following details:

- Name:** Set Fan Speed
- Description:** Example: Change the fan speed on the device
- Device set:** Refrigerated Vending Machine (1.0.C)
- Job type:** Settings
- Operations:** Set Fan Speed to 1000 (under Fan Speed setting)

On the right, a list of devices is shown with checkboxes:
 

- Name:** Refrigerator 1 (selected)
- Name:** Refrigerator 3 (selected)
- Name:** Refrigerator 2 (selected)

8. After selecting your devices, choose **Run** or **Save**.

The job now appears on your main Jobs page. On this view, you can see your currently running job and the history of any previously run jobs. Your running job always shows up at the top of the list. Your saved job can be opened again at any time to continue editing or to run.

## Review Job Information

To see an overview of your jobs, select a job from the list on your Jobs page. This overview contains the job details, devices, and device status values. From this overview, you can also select Download Job Details to download a .csv file of your job details, including the devices and their status values.

This information can be useful for troubleshooting.

## View the Job Status

After a job is created, the Status column updates with the latest status message of the job. The following table lists the possible status values:

Status message	Status meaning
Completed	This job has been executed on all devices.
Failed	This job has failed and not fully executed on devices.
Pending	This job hasn't yet begun executing on devices.
Running	This job is currently executing on devices.
Stopped	This job has been manually stopped by a user.

The status message is followed by an overview of the devices in the job. The following table lists the possible device status values:

Status message	Status meaning
Succeeded	The number of devices that the job successfully executed on.
Failed	The number of devices that the job has failed to execute on.

## View the Device Status

To view the status of the job and all the affected devices, select the job. To download a .csv file that includes the job details, including the list of devices and their status values, select Download job details. Next to each device name, you see one of the following status messages:

Status message	Status meaning
Completed	The job has been executed on this device.
Failed	The job has failed to execute on this device. The error message shows more information.
Pending	The job hasn't yet executed on this device.

**Note:** If a device has been deleted, you can't select the device and it displays as deleted with the device ID.

## Running and Stopping Jobs

To stop a running job, select it and choose Stop on the panel. The job status changes to reflect the job is stopped.

To run a job that's currently stopped, select the stopped job. Choose Run on the panel. The job status changes to reflect the job is now running again.

## Copy a Job

To copy an existing job you've created, select it from the main jobs page and select Copy. A new copy of the job configuration opens for you to edit. You can save or run the new job. If any changes have been made to your selected device set, they're reflected in this copied job for you to edit.

MCT USE ONLY. STUDENT USE PROHIBITED

# Business Integration

## Introduction to Rules

Rules in IoT Central serve as a customizable response tool that trigger on actively monitored events from connected devices.

### Select Target Devices

Rules are applied to specific devices. Use the target devices section to select the kind of devices that a rule will be applied to. Filters allow you to further refine what devices should be included. The filters use properties on the device template to filter down the set of devices. Filters themselves don't trigger an action. In the following screenshot, the devices that are being targeted are of device template type Refrigerator. The filter states that the rule should only include Refrigerators where the Manufactured State property equals Washington.

The screenshot shows the 'Target devices' configuration. It includes a dropdown for 'Device template \*' set to 'Refrigerator', and a filter row for 'Manufactured State' set to 'Equals' 'Washington'. A '+ Filter' button is visible below the row.

### Use multiple conditions

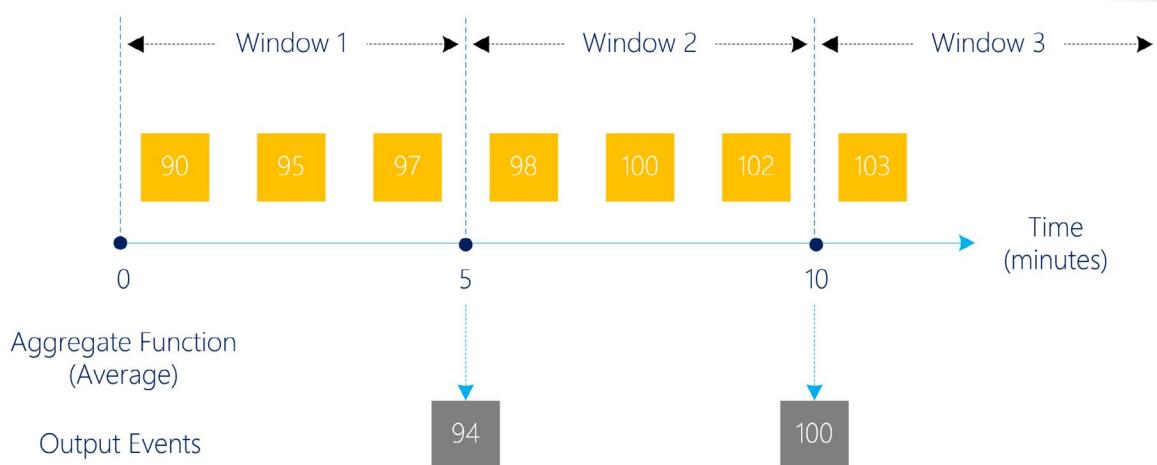
Conditions are what rules trigger on. Currently, when you add multiple conditions to a rule, they're logically AND'd together. In other words, all conditions must be met for the rule to evaluate as true.

In the following screenshot, the conditions check when the temperature is greater than 90 and the humidity is less than 10. When both of these statements are true, the rule evaluates to true and triggers an action.

The screenshot shows the 'Conditions' configuration. It includes two rows for 'Telemetry \*': one for 'Temperature' with 'Operator \*' 'Is greater than' '90' and another for 'Humidity' with 'Operator \*' 'Is less than' '10'. A '+ Condition' button is visible below the rows. At the bottom, there's a 'Time aggregation' section with a radio button for 'Off' and a button to 'Select a time window'.

## Use aggregate windowing

Rules evaluate aggregate time windows as tumbling windows. In the screenshot below, the time window is five minutes. Every five minutes, the rule evaluates on the last five minutes of data. The data is only evaluated once in the window to which it corresponds.



## Use rules with IoT Edge modules

A restriction applies to rules that are applied to IoT Edge modules. Rules on telemetry from different modules aren't evaluated as valid rules. Take the following as an example. The first condition of the rule is on a temperature telemetry from Module A. The second condition of the rule is on a humidity telemetry on Module B. Since the two conditions are from different modules, this is an invalid set of conditions. The rule isn't valid and will throw an error on trying to save the rule.

## Create a Rule and Set Up Notifications

You can use Azure IoT Central to remotely monitor your connected devices. Azure IoT Central rules enable you to monitor your devices in near real time and automatically invoke actions, such as send an email. In just a few clicks, you can define a condition to monitor telemetry from your devices and configure a corresponding action. This article explains how to create rules to monitor telemetry sent by the device.

Devices use telemetry to send numerical data from the device. A rule triggers when the selected device telemetry crosses a specified threshold.

### Create a rule

To create a telemetry rule, the device template must have at least one telemetry measurement defined.

1. In the left pane, select Rules.
2. To add a new rule, select +.
3. Enter a name that identifies the rule, and press Enter.
4. Select a device template.

By default, the rule automatically applies to all the devices associated with the device template. To filter for a subset of the devices, select + **Filter** and use device properties to identify the devices.

To disable the rule, toggle the Enabled/Disabled button in the rule header

## Configure the Rule Conditions

Conditions define the criteria that the rule monitors. For example, if you are using sensors to manage the environmental conditions in a work space, you could configure a rule to fire when the reported temperature exceeds 80° F.

1. Select a telemetry property from the Telemetry dropdown.  
For example, for an environmental sensor, you might choose telemetry for temperature values.
2. Specify the conditional operator and value that you want associated with the telemetry data.  
For example, you might choose **Is greater than** as the **Operator** and enter **80** as the **Value**.
3. Optionally, you can set a Time aggregation. When you select a time aggregation, you must also select an aggregation type, such as average or sum from the aggregation drop-down.

Without aggregation, the rule triggers for each telemetry data point that meets the condition. For example, if the rule is configured to trigger when temperature is above 80 then the rule triggers almost instantly when the device reports temperature > 80.

With aggregation, the rule triggers if the aggregate value of the telemetry data points in the time window meets the condition. For example, if the rule is configured to trigger when temperature is above 80, time aggregation is set to 10 minutes, and the aggregation type is average, then the rule triggers when the device reports an average temperature > 80, calculated over a 10-minute interval.

You can add multiple conditions to a rule by selecting **+** **Condition**. When multiple conditions are specified, all the conditions must be met for the rule to trigger. Each condition is joined by an implicit **AND** clause. If you're using time aggregation with multiple conditions, all the telemetry values must be aggregated.

## Configure actions

After you define the condition, you set up the actions to take when the rule fires. Actions are invoked when all the conditions specified in the rule evaluate to true. Currently, email is the only available action.

1. Select **+** **Email** in the **Actions** section.
2. Enter a display name for the action, specify a recipient email address, and provide a message for the body of the email.

For example, following along with the temperature scenario from above:

- Specify "Temperature warning" as the **Display name** for the action.
- Specify an email address that you can monitor during testing in the **To** field.
- Specify "You should check the device!" to appear in the body of the email in the **Note** field.

3. To save the action, choose Done.

You can add multiple actions to a rule.

4. To save the rule, choose Save.

The rule goes live within a few minutes and starts monitoring telemetry being sent to your application. When the condition specified in the rule is met, the rule triggers the configured email action.

## Delete a Rule

If you no longer need a rule, delete it by opening the rule and choosing **Delete**.

## Enable or Disable a Rule

Choose the rule you want to enable or disable. Toggle the Enabled / Disabled button in the rule to enable or disable the rule for all devices that are scoped in the rule.

## Enable or Disable a Rule for a Device

Choose the rule you want to enable or disable. Add a filter in the **Scopes** section to include or exclude a certain device in the device template.

# Data Visualizations and Analysis

## Configure a Dashboard

The Dashboard is the page that loads when users who have access to the application navigate to the application's URL. If you created your application from one of the Application Templates, your application will have a pre-defined dashboard to start. If you created your application from the Custom Application application template, your dashboard will be blank to start.

**Note:** Users can create multiple dashboards in addition to the default application dashboard. These dashboards can be personal to the user only, or shared across all users of the application.

### Add Tiles

Open your Dashboard page from the left pane of the IoT Central application. To begin customizing the default dashboard for your application, select **Edit** at the top-left of the Dashboard page. Selecting Edit opens the dashboard library panel. The library contains the tiles and dashboard primitives that you can use to customize the dashboard.

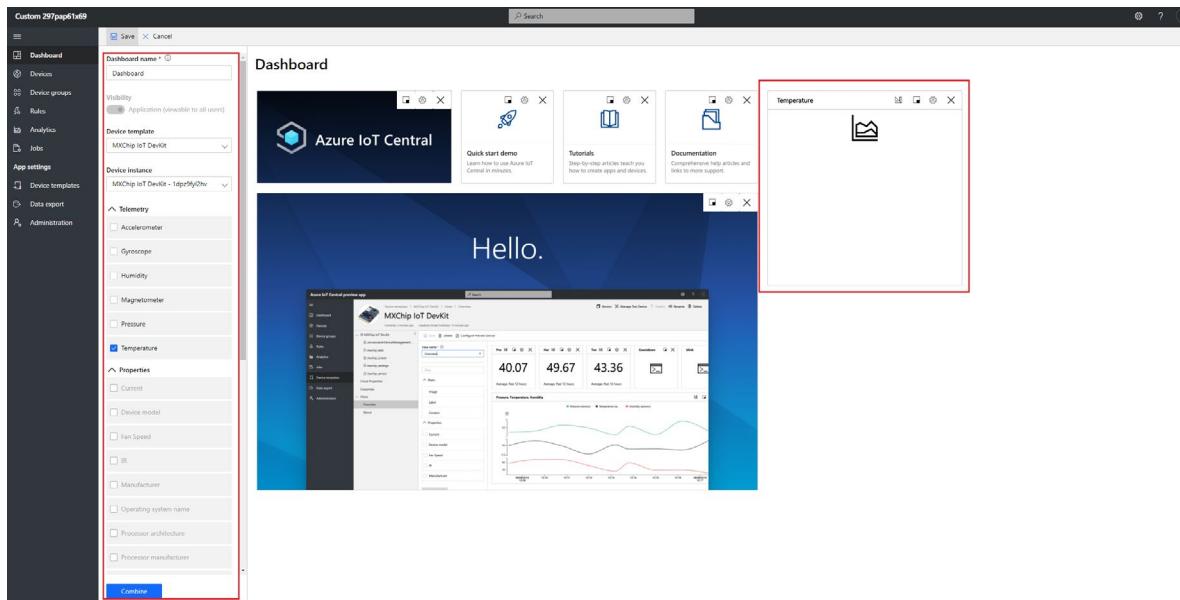
For example, if you want to configure a dashboard that will be used to monitor the environment in an office space, you can add a Telemetry tile for the current temperature of a device. To do so:

1. Select a Device Template.
  2. Select a Device Instance for the device you want to see on a dashboard tile.
- You will see a list of the device's properties that can be used on the tile.
3. To create the tile on the dashboard, click on telemetry property and drag it to the dashboard area.

For the office environment scenario, you would drag Temperature to the dashboard area. You could also click the checkbox next to Temperature, and then click Combine.

The screenshot below helps to illustrate the process on the UI.

4. Select **Save** in the top left to save the tile to the dashboard.

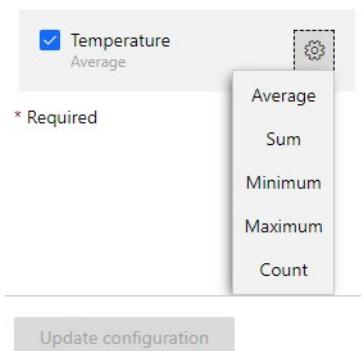


Now when an operator views the default application dashboard, they see the new tile with the Temperature for the device. Each tile has a pre-selected graph, chart, etc. that will be displayed when the tile is created. However, users can choose to edit and change this visualization.

## Edit Tiles

To edit a tile on the dashboard, first click **Edit** at the top left of the page, which will open edit mode for the dashboard and all its tiles.

Then click the **Gear** icon in the top-right corner of the tile you wish to edit. Here you can edit aspects of the tile including its title, its visualization, aggregation, etc.



You can also change the chart visualization by clicking the Ruler icon on the tile.

A screenshot of the dashboard edit interface. On the left, there are settings for "Dashboard name" (set to "Dashboard"), "Visibility" (set to "Application (viewable to all users)", "Device template" (dropdown), "Device instance" (dropdown), and "Custom tiles" (list: Image, Label, Content). On the right, a tile titled "Temperatures" is selected. A context menu is open over the tile, listing visualization options: KPI (selected), Bar chart, Heatmap, Line chart, Pie chart, and Last Known Value. The tile itself shows a small bar chart icon.

## Tile Types

The following table summarizes the usage of tiles in Azure IoT Central:

Tile	Dashboard	Description
Content	Application and device set dashboards	Markdown supported tiles are clickable tiles that display heading and description text. You can also use this tile as a link tile to enable a user to navigate to a URL related to your application.
Image	Application and device set dashboards	Image tiles display a custom image and can be clickable. Use an image tile to add graphics to a dashboard and optionally enable a user to navigate to a URL relevant to your application.
Label	Application dashboards	Label tiles display custom text on a dashboard. You can choose the size of the text. Use a label tile to add relevant information to the dashboard such descriptions, contact details, or help.
Map	Application and device set dashboards	Map tiles display the location and state of a device on a map. For example, you can display where a device is and whether its fan is switched on.
Line Chart	Application and device dashboards	Line chart tiles display a chart of aggregate measurement for a device for a time period. For example, you can display a line chart that shows the average temperature and pressure of a device for the last hour.
Bar Chart	Application and device dashboards	Bar chart tiles display a chart of aggregate measurements for a device for a time period. For example, you can display a bar chart that shows the average temperature and pressure of a device for the last hour.
Pie Chart	Application and device set dashboards	Pie chart tiles display a chart of aggregate measurements for a device for a time period.
Heat Map	Application and device set dashboards	Heat Map tiles display information about the device set, represented as colors.

Tile	Dashboard	Description
Event History	Application and device dashboards	Event History tiles display the events for a device over a time period. For example, you can use it to show all the temperature changes for a device during the last hour.
State History	Application and device dashboards	State history tiles display the measurement values for a time period. For example, you can use it to show the temperature values for a device during the last hour.
KPI	Application and device dashboards	KPI tiles display an aggregate telemetry or event measurement for a time period. For example, you can use it to show the maximum temperature reached for a device during the last hour.
Last Known Value	Application and device dashboards	Last known value tiles display the latest value for a telemetry or state measurement. For example, you can use this tile to display the most recent measurements of temperature, pressure and humidity for a device.

## Analyze Your Device Data

Azure IoT Central provides rich analytics capabilities to analyze historical trends and correlate various telemetries from your devices.

### Understanding the Analytics UI

Analytics user interface is made of three main components:

- Data configuration panel: On the configuration panel, start by selecting the device group for which you want to analyze the data. Next, select the telemetry that you want to analyze and select the aggregation method for each telemetry. Split By control helps to group the data by using the device properties as dimensions.
- Time control: Time control is used to select the duration for which you want to analyze the data. You can drag either end of the time slider to select the time span. Time control also has an Interval size slider that controls the bucket or the interval size used to aggregate the data.
- Chart control: Chart control visualizes the data as a line chart. You can toggle the visibility of specific lines by interacting with the chart legend.

## Querying your data

You'll need to start by choosing a device group, and the telemetry that you want to analyze. Once you're done, select Analyze to start visualizing your data.

- Device group: A device group is a user-defined group of your devices. For example, all Refrigerators in Oakland, or All version 2.0 wind turbines.
- Telemetry: Select the telemetry that you want to analyze and explore. You can select multiple telemetries to analyze together. Default aggregation method is set to Average for numerical and Count for string data-type respectively. Supported aggregation methods for Numeric data types are Average, Maximum, Minimum, Count and, Sum. Supported aggregation methods for string data type are count.
- Split by: 'Split by' control helps to group the data by using the device properties as dimensions. Values of the device and cloud properties are joined along with the telemetry as and when it is sent by the device. If the cloud or device property has been updated, then you will see the telemetry grouped by different values on the chart.

**Tip:** To view data for each device separately, select Device Id in the 'Split by' control.

## Interacting with your data

Once you've queried your data, you can start visualizing it on the line chart. You can show/hide telemetry, change the time duration, view telemetry in a data grid.

- Time editor panel: By default we'll retrieve data from the past one day. You can drag either end of the time slider to change the time duration. You can also use the calendar control to select one of the predefined time buckets or select a custom time range. Time control also has an Interval size slider that controls the bucket or the interval size used to aggregate the data.



- Inner date range slider tool: Use the two endpoint controls by dragging them over the time span you want. This inner date range is constrained by the outer date range slider control.
- Outer date range slider control: Use the endpoint controls to select the outer date range, which will be available for your inner date range control.
- Increase and decrease date range buttons: Increase or decrease your time span by selecting either button for the interval you want.
- Interval-size slider: Use it to zoom in and out of intervals over the same time span. This action provides more precise control of movement between large slices of time. You can use it to see granular, high-resolution views of your data, even down to milliseconds. The slider's default starting point is set as the most optimal view of the data from your selection, which balances resolution, query speed, and granularity.
- Date range picker: With this web control, you can easily select the date and time ranges you want. You can also use the control to switch between different time zones. After you make the changes to apply to your current workspace, select Save.

**Tip:** Interval size is determined dynamically based on the selected time span. Smaller time spans will enable aggregating the data into very granular intervals of up to a few seconds.

- Chart Legend: Chart legend shows the selected telemetry on the chart. You can hover over each item on the legend to bring it into focus on the chart. When using 'Split By', the telemetry is grouped by the respective values of the selected dimension. You can toggle the visibility of each specific telemetry or the whole group by clicking on the group name.
- Y-axis format control: y-axis mode cycles through the available y-axis view options. This control is available only when different telemetries are being visualized. You can set the y-axis by choosing from one of three modes:
  - Stacked: A graph for every telemetry is stacked and each of the graphs have their own y-axis. This mode is set as default.
  - Shared: A graph for every telemetry is plotted against the same y-axis.
  - Overlap: Use it to stack multiple lines on the same y-axis, with the y-axis data changing based on the selected line.
- Zoom control: Zoom lets you drill further into your data. If you find a time period you'd like to focus on within your result set, use your mouse pointer to grab the area and then drag it to the endpoint of your choice. Then right click on the selected area and click Zoom.

Under the ellipsis, there are more chart controls to interact with the data.

- Display Grid: Your results are available in a table format, enabling you to view the specific value for each data point.
- Drop a Marker: 'Drop Marker' control provides a way to anchor certain data points on the chart. It is useful when you are trying to compare data for multiple lines across different time periods.

## About the Module 11 Labs

### Module Labs

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

- Lab 20: Build an IoT Solution with IoT Central

**WARNING:** Be prepared for UI changes

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

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