Module 11: Integrating Azure Solution Components using Messaging Services

Contents:

Module overview

This module describes and compares the integration and messaging services available for solutions hosted on the Azure platform. Messaging services described include Azure Storage Queues, Service Bus Queues, Service Bus Relay, IoT Hubs, Event Hubs, and Notification Hubs. Integration services include Azure Functions and Logic Apps.

Objectives

After completing this module, students will be able to:

- Compare Storage Queues to Service Bus Queues.
- Identify when to use Azure Functions or Logic Apps for integration components in a solution.
- Describe the differences between IoT Hubs, Event Hubs and Time Series Insights.

Lesson 1: Event Messaging

This lesson introduces Service Bus and Event Grid as key components of an event messaging architecture hosted in Azure.

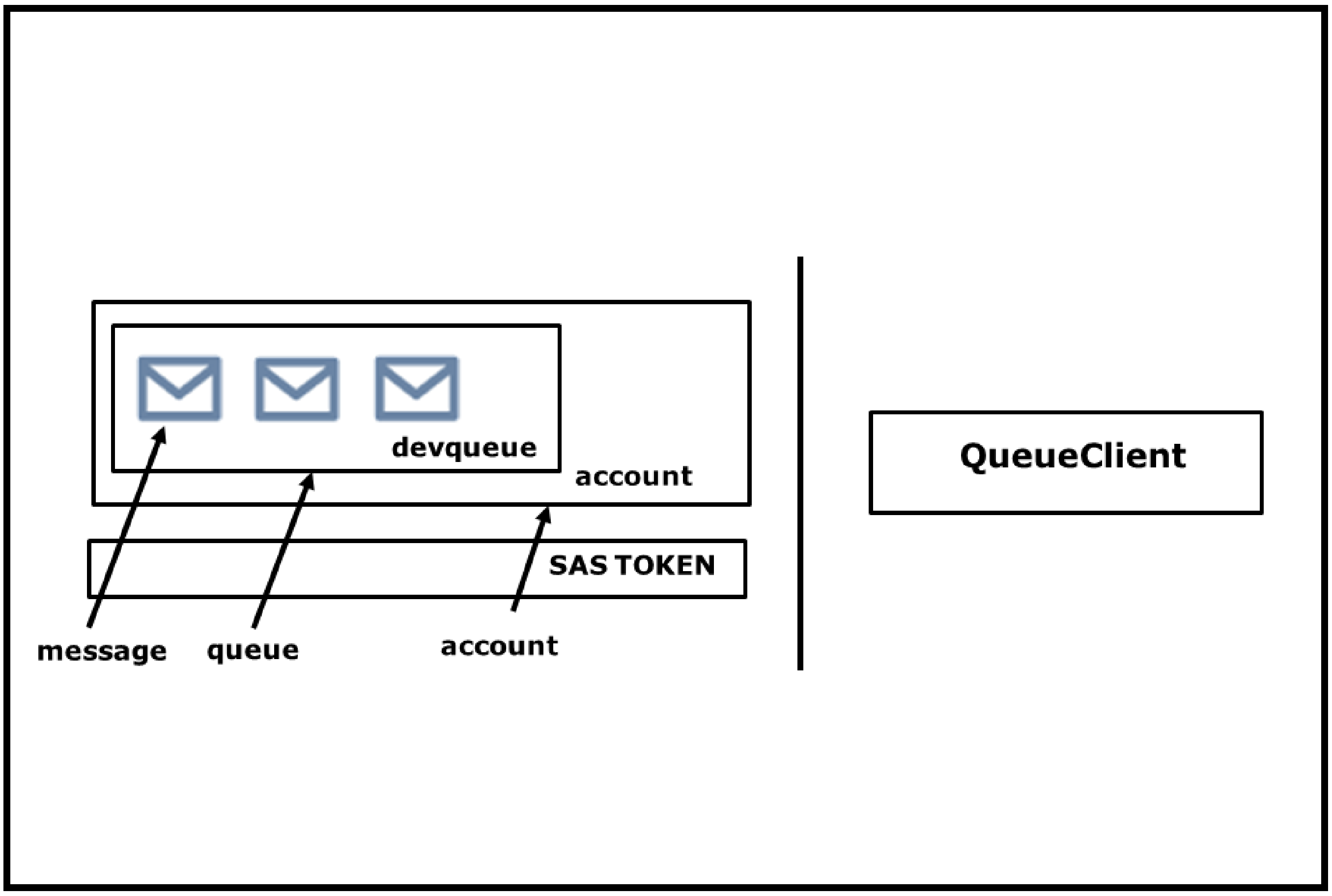
Lesson objectives

After completing this lesson, you will be able to:

- Describe various Service Bus offerings and compare them to equivalent offerings in other services.

- Identify when to use Event Grid as a messaging component of a solution in Azure.

Storage Queues



Azure Storage Queues are a feature of the Azure Storage service for storing large numbers of messages that can be accessed from anywhere in the world via authenticated calls using HTTP or HTTPS. A single queue message can be up to 64 KB in size, and a queue can contain millions of messages, up to the total capacity limit of a storage account. A storage account can contain up to 500 TB of blob, queue, and table data.

Common uses of queue storage include:

- Creating a backlog of work to process asynchronously
- Passing messages from an Azure Web role to an Azure Worker role

Queue Service Components

The Queue service contains the following components:

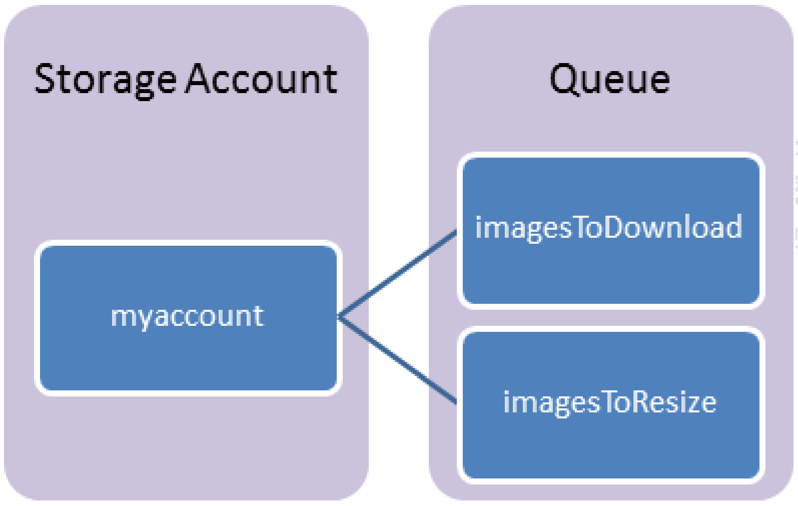


FIGURE 11.1: QUEUE SERVICE COMPONENTS

- **URL format**: Queues are addressable using the following URL format:

  http://[account].queue.core.windows.net/<queue>

  The following URL addresses one of the queues in the diagram:

  http://[account].queue.core.windows.net/imagesToDownload
- **Queue**: A queue contains a set of messages. All messages must be in a queue.
- **Message**: A message, in any format, of up to 64KB.

Storage Queue Message Handling

Queue messages can be managed in a variety of different ways. The following is a list of actions that you can perform on a queue or it's messages:

- **Create/Delete Queue**

  Client SDKs, PowerShell or the REST API can be used to create a new queue instance or remove an existing one.
- **Measure Queue Length**

  You can get an estimate of the number of messages in the queue. Do to race conditions and technical implementation, this count is not guaranteed and should be treated in your application as an approximate queue length.
- **Insert Message into Queue**

  New messages can be added to the queue. These messages have a body that are either a string (in UTF-8 format) or a **byte** array.
- **Retrieve the Next Message**

A copy of the next available message is retrieved and the message is made **invisible** for a specific duration. During this time, you can process the message. Other queue consumers will not see this message in the queue when they retrieve messages while it is invisible. After a specific amount of time, the invisibility duration will elapse and the message is available to other queue consumers.

- **Extend Message Lease**

  If you need more time to process a retrieved queue message, you can return to the queue and update the invisibility duration for the queue message. This will ensure that the message is not prematurely available to other queue consumers.

- **Peek at the Next Message**

  You can retrieve a copy of the next available message in the queue without making the message invisible to other queue consumers.

- **Update a Message**

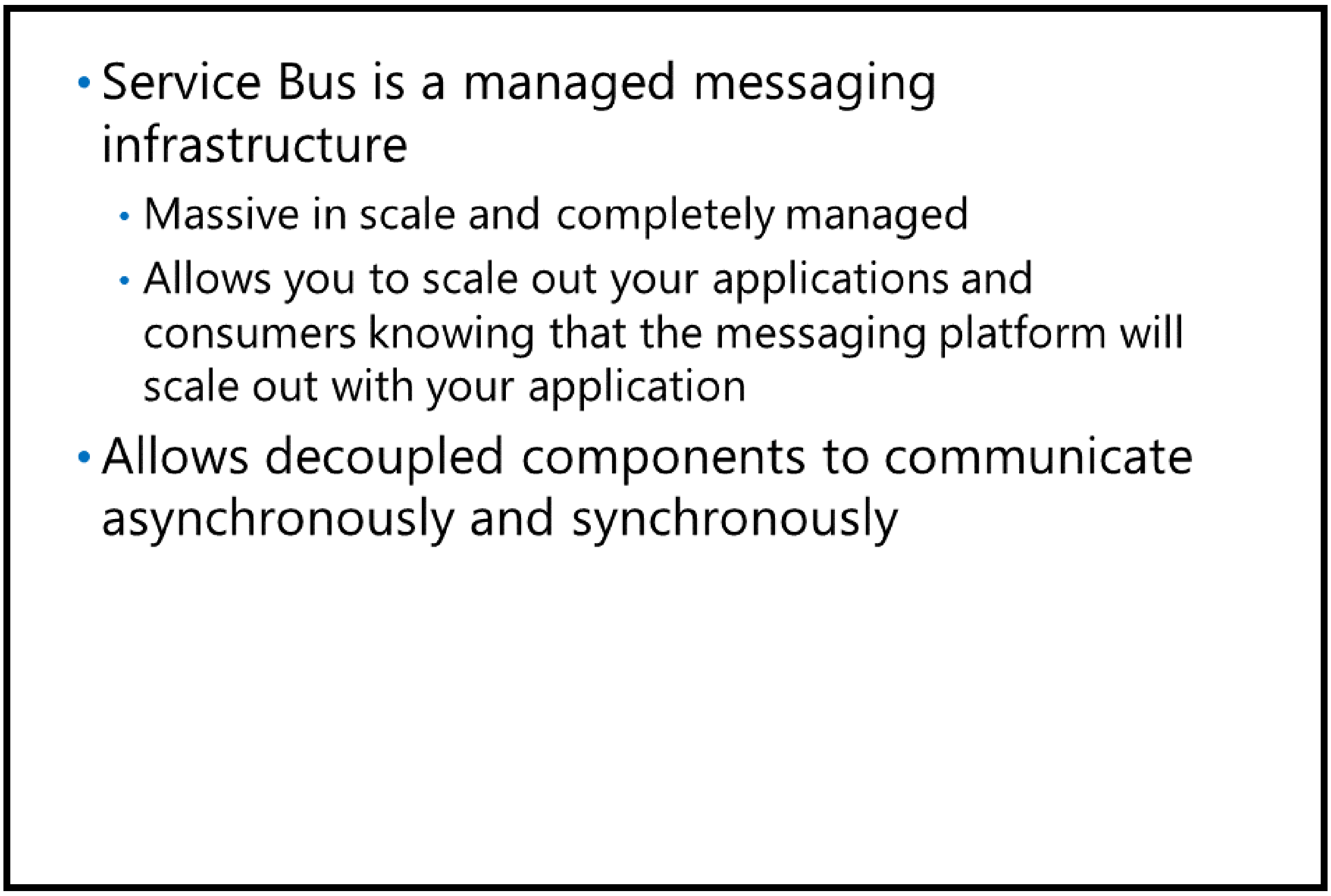
  The contents of a retrieved message can be updated in the queue if you need to have a concept of checkpoints or state for queue messages.

- **Delete a Message**

Once you have completed processing a message, you must delete the message from the queue if you do not wish for it to be processed by any more queue consumers. Otherwise the message invisibility will timeout and the message will be processed again by other queue consumers.

Because of the workflow, queue messages and their processors/consumers must be designed to be idempotent and should not have side effects from processing the same message multiple times. For example, if a queue message contains data that needs to be stored in a SQL database, your processor should check to see if an existing record exists and then update that record or create a new record. Otherwise, you may end up with false duplicate data from your queue processors processing the same message multiple times.

Service Bus



- Service Bus is a managed messaging infrastructure
  - Massive in scale and completely managed
  - Allows you to scale out your applications and consumers knowing that the messaging platform will scale out with your application
- Allows decoupled components to communicate asynchronously and synchronously

Azure Service Bus provides a hosted, secure, and widely available infrastructure for widespread communication, large-scale event distribution, naming, and service publishing. Service Bus provides connectivity options for Windows Communication Foundation (WCF) and other service endpoints – including REST endpoints -- that would otherwise be difficult or impossible to reach. Endpoints can be located behind network address translation (NAT) boundaries, or bound to frequently-changing, dynamically-assigned IP addresses, or both.

Service Bus provides both “relayed” and “brokered” messaging capabilities. In the relayed messaging pattern, the relay service supports direct one-way messaging, request/response messaging, and peer-to-peer messaging. Brokered messaging provides durable, asynchronous messaging components such as Queues, Topics, and Subscriptions, with features that support publish-subscribe and temporal decoupling: senders and receivers do not have to be online at the same time; the messaging infrastructure reliably stores messages until the receiving party is ready to receive them.

Service Bus services are typically partitioned into namespaces as each namespace provide both a service and security boundary.

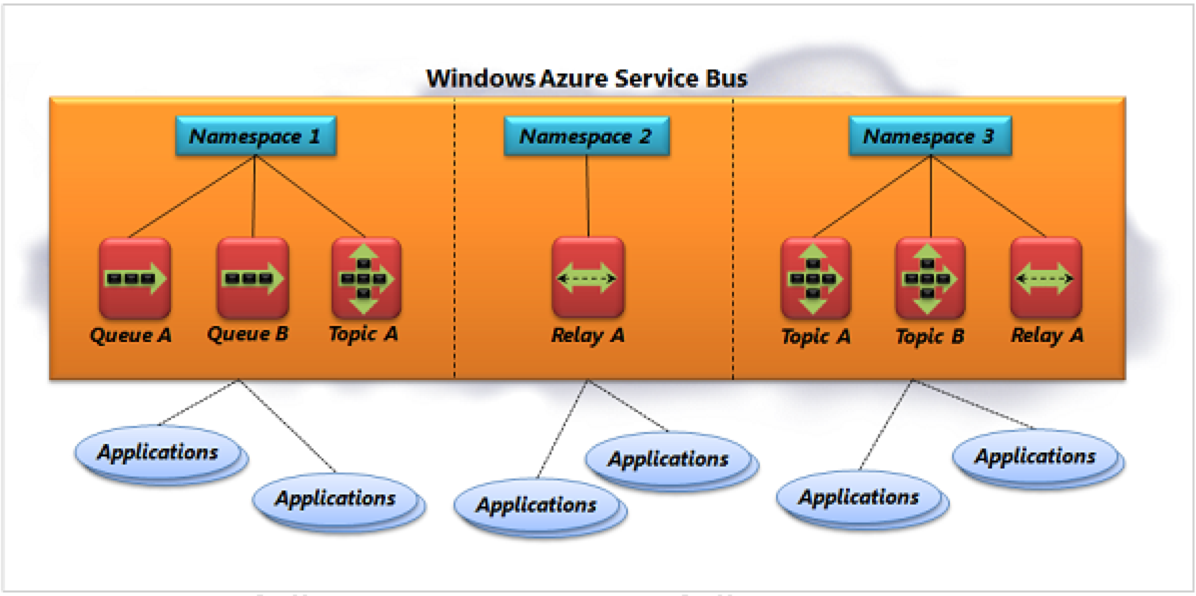


FIGURE 11.2: SERVICE BUS NAMESPACES

When you create a queue, topic, relay, or Event Hub, you give it a name. Combined with whatever you called your namespace, this name creates a unique identifier for the object. Applications can provide this name to Service Bus, then use that queue, topic, relay, or Event Hub to communicate with one another.

To use any of these objects, Windows applications can use Windows Communication Foundation (WCF). For queues, topics, and Event Hubs Windows applications can also use Service Bus-defined messaging APIs. To make these objects easier to use from non-Windows applications, Microsoft provides SDKs for Java, Node.js, and other languages. You can also access queues, topics, and Event Hubs using REST APIs over HTTP.

Service Bus Queues

Service Bus Queue is a brokered messaging system, which is similar to the Queue service Azure Storage. By using these queues, application modules that are distributed do not need to communicate directly with each other. These application modules can instead communicate by using the queues. This ensures that there is a separation between message generators and message processors. This separation provides the flexibility of having one or more application component instances that are generating messages and one or more application component instances that are processing the same messages. If an instance encounters an irrecoverable exceptional condition, other instances can continue processing the messages. If the workload for the entire application is increased, new instances can be created to handle the load. These scenarios are common and critical when developing and designing cloud applications.

Service Bus queues do implement a familiar first in, first out (FIFO) message delivery strategy. Service Bus queues can also guarantee that a message is received and processed both at least and at most once by the message consumers.

Service Bus Relay

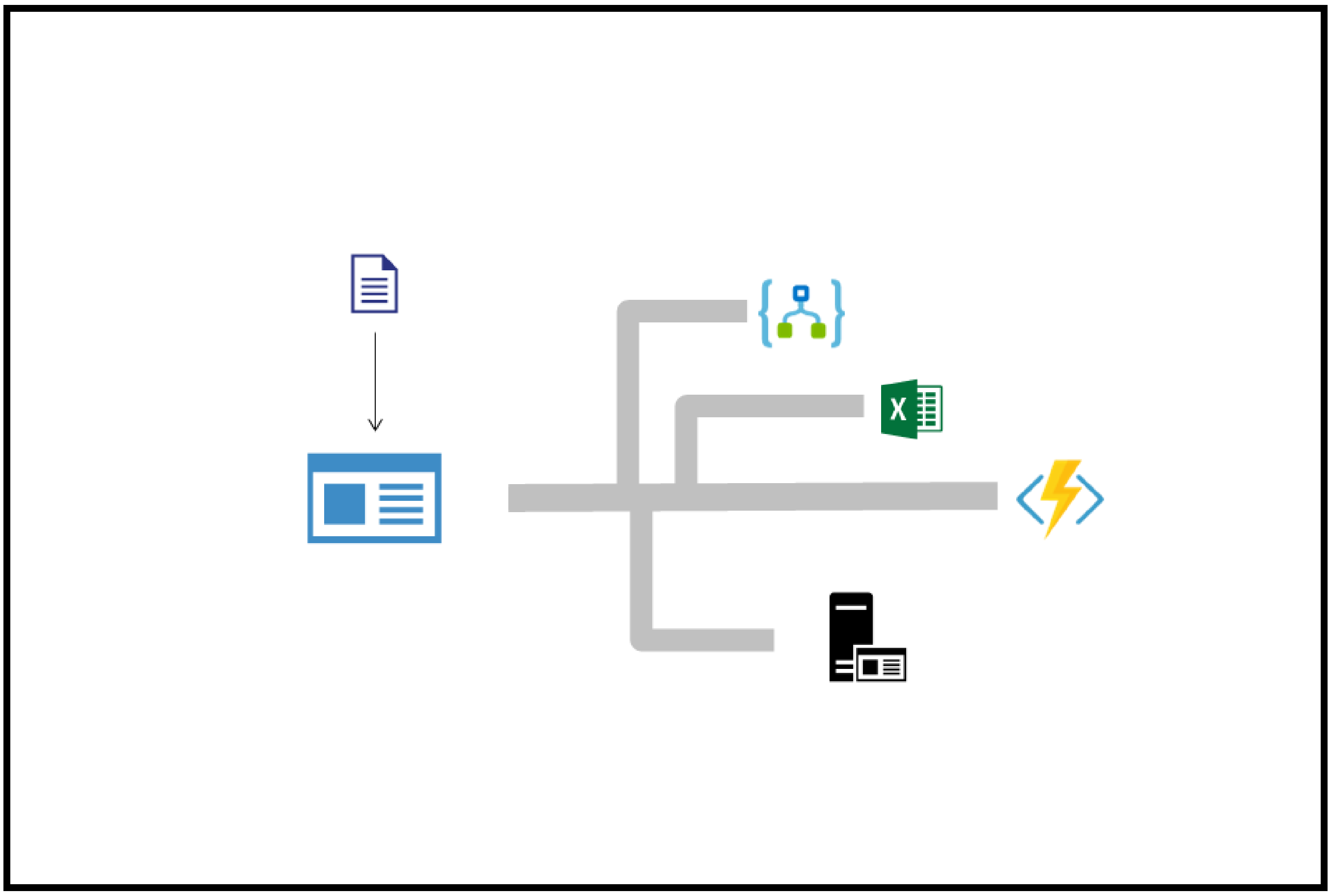
Service Bus includes a relay component that can connect existing services to new client applications without exposing the true location or address of the service. Some of the advantages of using a relay include:

- Services that need to communicate directly with external client applications or devices (For example, mobile devices) typically need to be placed in a special subnet or a virtual network with a unique NAT or firewall configuration. The address for the service endpoint will need to be publicly addressable in order for client devices to connect. In some enterprises, this can be considered dangerous or unacceptable. With Service Bus Relay, the service makes an outbound connection to the relay and bypasses many of the complex network configurations that are necessary for inbound connections

- Although mobile applications are deployed and updated regularly, end users might not update their applications as regularly as you want them to. If your service needs to be migrated to a new network or moved to a new IP address, this can cause a lapse of connectivity for your mobile applications. Using Service Bus Relay, your mobile applications address a publicly accessible and permanent uniform resource identifier (URI). You are then free to make changes and migrate your service within your organization's infrastructure. The new service instance or location simply needs to connect to the relay for client devices to access it. This enables more mobility for services that are connected to the applications that are already deployed.

Service Bus Relay also supports direct peer-to-peer communication. This is negotiated if the relay determines that the connecting application can easily and directly address the service.

Event Grid



Azure Event Grid is a single service designed to managed and route systemic events from any source service in your Azure subscription. Event Grid is designed to eliminate the need to poll existing services and enable applications to run in an event-driven manner where applications are triggered only when needed, and consume compute only when necessary.

Event Grid can be used to publish custom events from your workloads or pre-determined events designed with each Azure service. Event Grid can be used in a variety of ways including:

- Integrate various components of your workloads together using a publish-subscribe mode.
- Enable your solution to listen to events from third-party B2B services of publish events for the third-party services to consume.
- Create serverless compute that is triggered by a specific Azure service event such as the creation of a database or VM.
- Automate the deployment of resources by subscribing to ARM events.

Azure Event Grid allows users to build applications with event-based architectures with ease. You merely select the Azure resource you wish to use, then select the WebHook endpoint or event handler to send the event. Event Grid has integrated support for events coming from Azure services, as well as support for third-party events. You can even use filters to route specific events to different or even multiple endpoints.

An Event Grid instance can use point-and-click to configure events, moving them from your Azure resource to an event handler or endpoint. It also has reliability, allowing the use of a 24-hour retry with exponential backoff so users can ensure events are delivered on time. Event Grid can also build high-volume workloads with support for millions of these events per second, and even custom events can be routed, filtered, and delivered. With all the power of Event Gird, you can rest assured that it is pay per event, meaning that you only pay for what you use with Event Grid.

Event Grid can provide users with several options to improve serverless, integration, and ops automation work. Event Grid can connect data sources and event handlers to provide a serverless application architecture. For example, you can use Event Grid to trigger a function to run image analysis everytime a new item becomes added to a blob storage container. Event Grid can also connect an application with other services. As an example, a user can create a storage blob to send your app's event data to Event grid, while taking advantage of its reliable delivery, direct integration, and advanced routing with Azure. Lastly, it also allows users to speed an automation process. A user can request Event Grid to notify Azure Automation when a virtual machine start or a SQL Database is spun up as one of many examples. Said events can be used to tag virtual machines, file work items, put metadata into operations tools, or even automatically check to ensure services configurations are correct.

Lesson 2: Integration

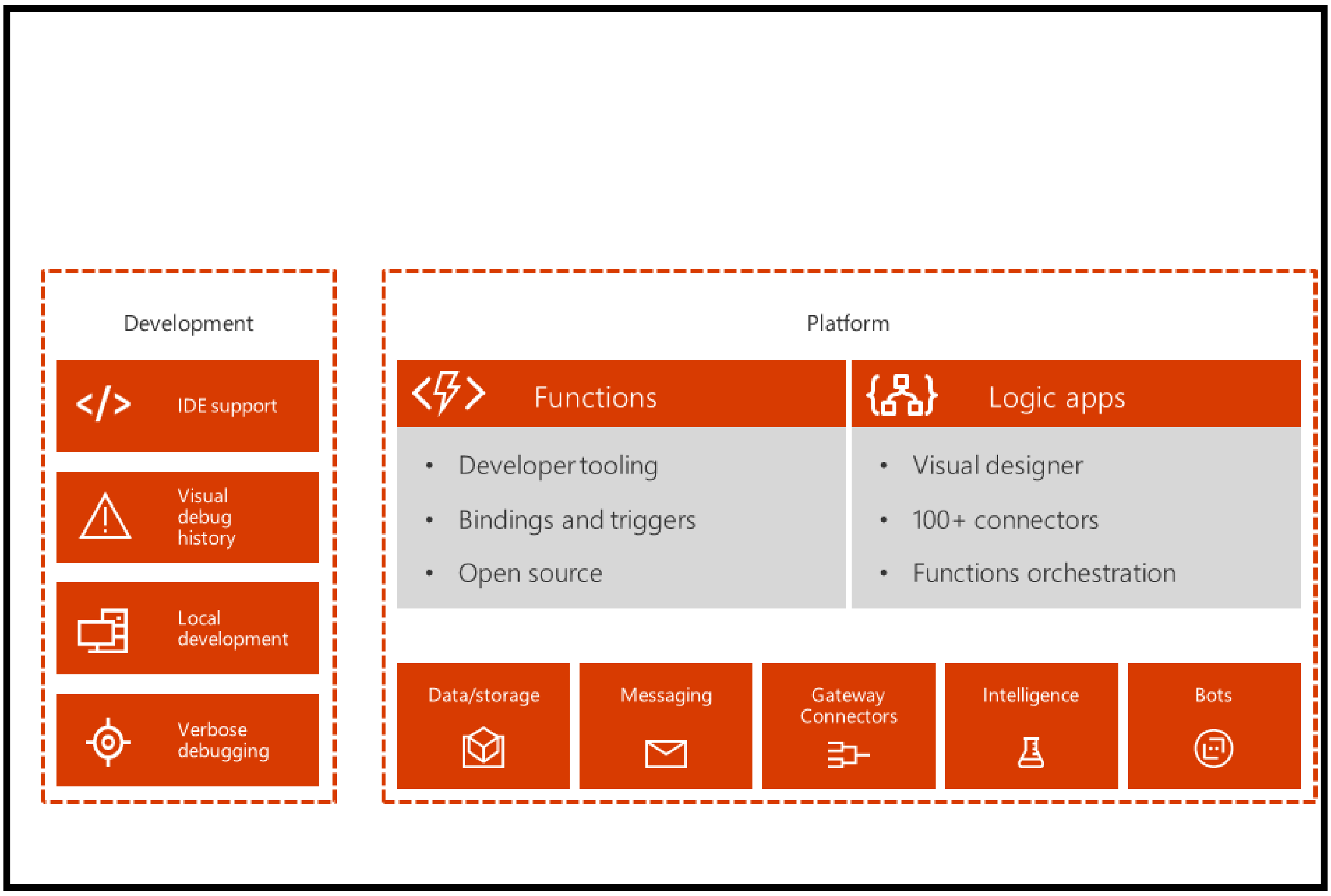
This lesson reviews the Logic Apps and Function Apps offerings in Azure and how they can be used as part of a full-stack serverless workload.

Lesson objectives

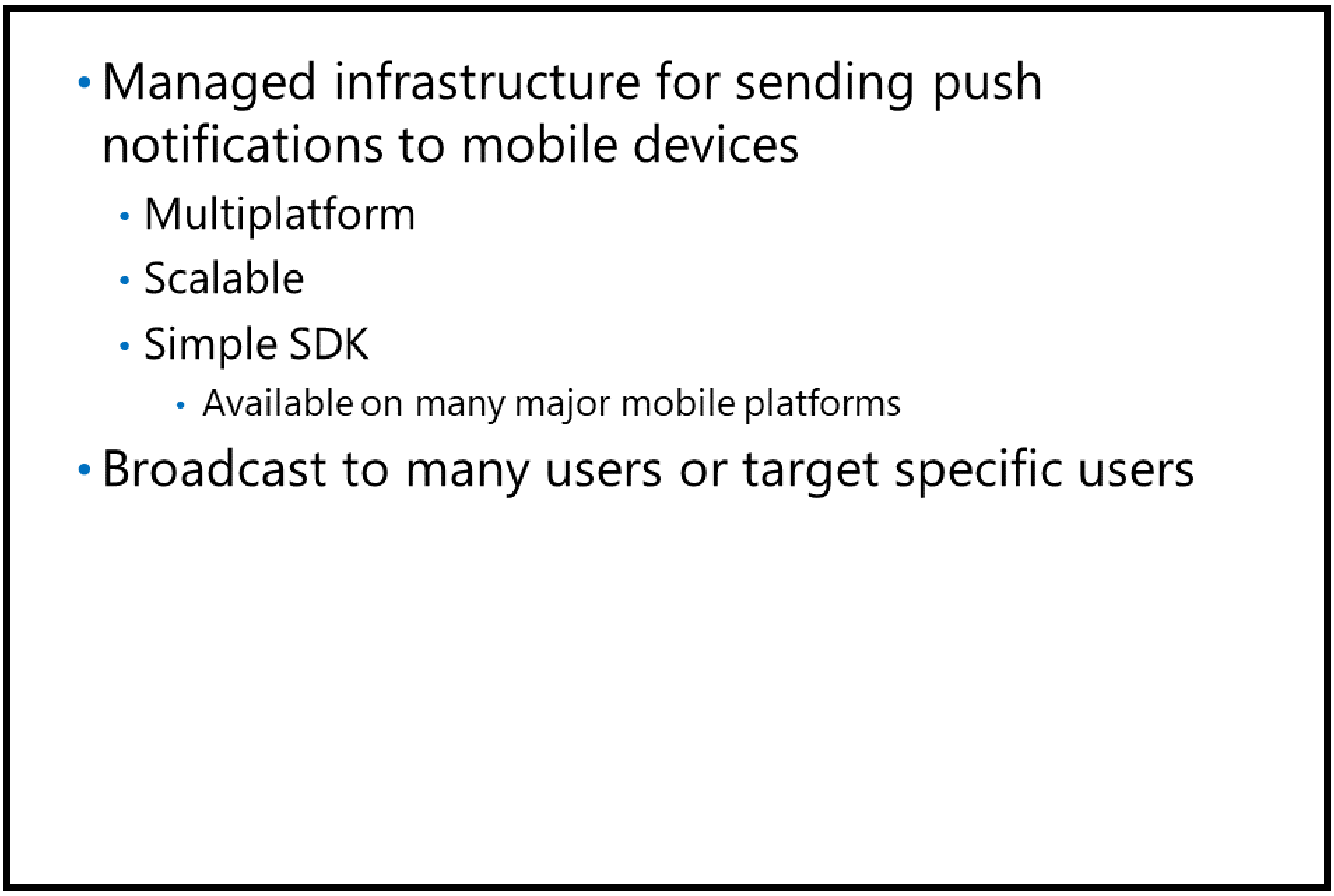
After completing this lesson, you will be able to:

- Integrate application components in Azure using Logic Apps and Functions

Serverless Integration



Notification Hubs



- Managed infrastructure for sending push notifications to mobile devices
  - Multiplatform
  - Scalable
  - Simple SDK
    - Available on many major mobile platforms
- Broadcast to many users or target specific users

Notification Hubs is a combination of a service infrastructure and a set of client libraries that allows you to publish push notifications to your mobile applications from any application service component. With Notification Hubs, you can send notifications that are personalized to a specific user, notifications that are distributed to many users across various platforms, and notifications that are filtered to a specific set of users. The Notification Hubs infrastructure abstracts the implementation of the various Platform Notification Systems (PNS) for each mobile platform. By using a single method call, you can send notifications to various device platforms without having to implement a different message structure or communication mechanism for each platform.

You can use notification hubs in a variety of scenarios including:

- Send wide-reaching news notifications to all devices with your mobile application installed
- Send a notification to a subset of your users that is determined based on a tag, label, or location
- Send specific notifications to a user for the activities that are related to their specific account

Benefits

Notification hubs eliminate the challenges that are involved in managing push notifications. Notification Hubs use a full multiplatform, scaled-out push notification infrastructure, and considerably reduce the push-specific code that runs in the app. Notification hubs implement all the functionality of a push infrastructure. Devices are only responsible for registering PNS handles, and the backend is responsible for sending platform-independent messages to users or interest groups.

Notification hubs provide a push infrastructure with the following advantages:

- **Multiple platforms:**
- Support for all major mobile platforms—Windows, Windows Phone, iOS, and Android.
- No platform-specific protocols. The application only communicates with Notification Hubs.
- Device handle management. Notification Hubs maintains the handle registry and feedback from PNSs.

- **Works with any backend**. Works with Cloud or on-premises applications that are written in .NET, PHP, Java, or Node.
- **Scale**. Notification hubs scale to millions of devices without the need of rearchitecting or sharding.
- **Rich set of delivery patterns**. Associate devices with tags, representing logical users or interest groups.
  - Broadcast. Allows for near-simultaneous broadcast to millions of devices with a single Application Programming Interface (API) call.
  - Unicast/Multicast. Push to tags representing individual users, including all their

devices; or a wider group. For example, a user could use the app on separate devices (tablet, phone, etc.) and would require push notifications to either be pushed to all devices or a specific device.

  - Segmentation. Push to a complex segment that is defined by tag expressions (For example, devices in New York following the Yankees).

- **Personalization**. Each device can have one or more templates to achieve per-device localization and personalization without affecting the backend code.

Platform Notifications

At a high level, all platform notification systems follow the same pattern:

1. The client application contacts the PNS to retrieve its handle. The handle type depends on the system. For Windows Notification Service (WNS), it is a URI or notification channel. For Apple Push Notification Service (APNS), it is a token.
2. The client application stores this handle in the app backend for later usage. For WNS, the backend is typically a cloud service. For APNS, the system is called a provider.
3. To send a push notification, the app backend contacts the PNS by using the handle to target an instance of a specific client application.
4. The PNS forwards the notification to the device specified by the handle.

Notification Hubs can be used in flexible ways to register devices and eventually send a message to the devices. Devices can register themselves and receive notifications using the following method:

1. The client device reaches out to the PNS by using the Notification Hubs SDK. It registers a unique PNS handle that is used by the service to send notifications to this device whether the application is running or not.
2. The client device can alternatively send its PNS handle to the application backend to have the application register the device.
3. When the application backend sends a message to the Notification Hubs service, the service handles sending the message to the appropriate target clients by using their registered PNS handles. The application backend simply requests the message is sent and the Notification Hubs service and the PNS handle the actual distribution of messages to client devices.

Lesson 3: Internet of Things (IoT)

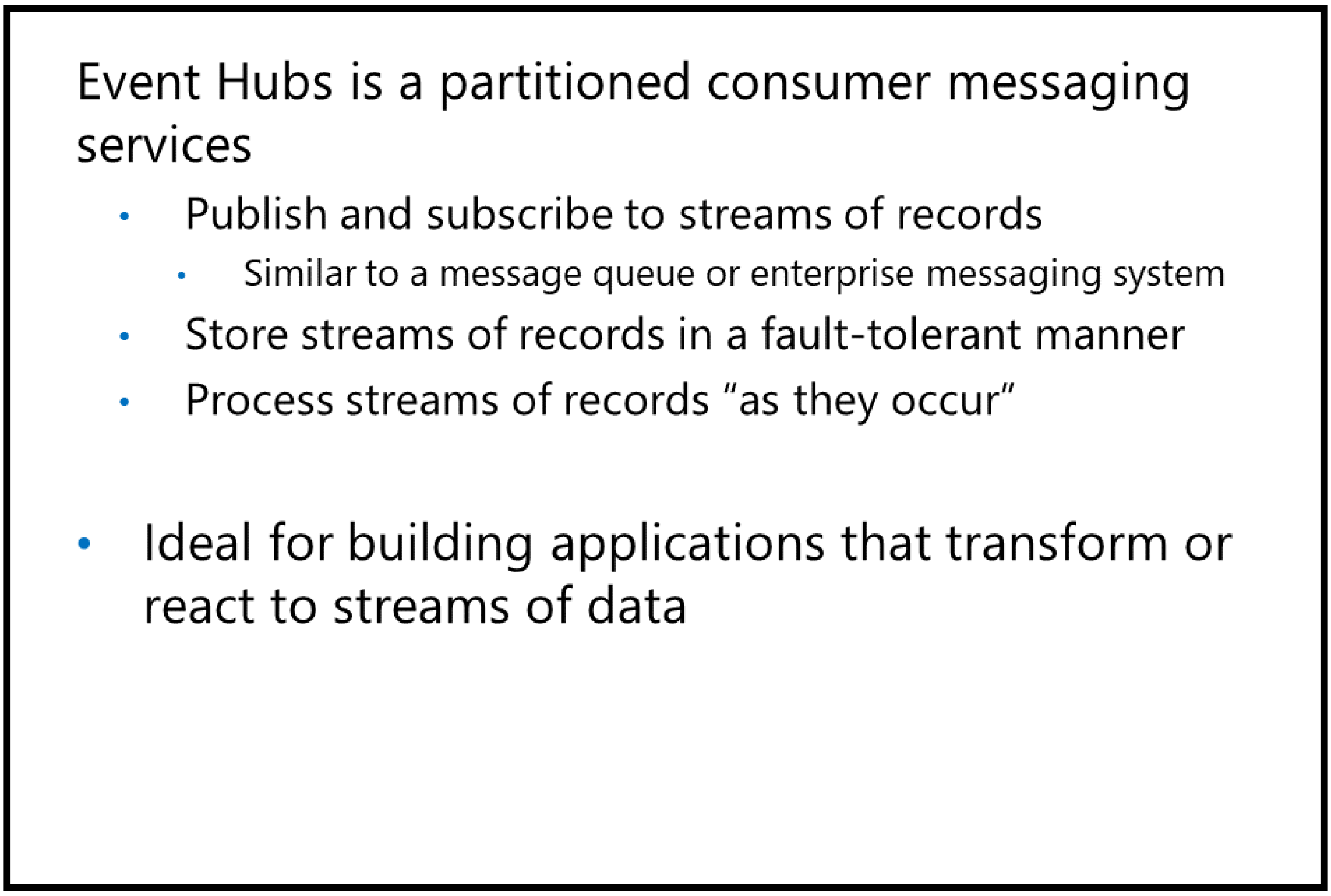
This lesson covers the two most common real-time message streaming options available in Azure, Event Hubs and IoT Hubs.

Lesson objectives

After completing this lesson, you will be able to:

- Identify the various streaming queue services in Azure.
- Compare and contrast Event Hubs and IoT Hubs.
- Architect a real-time streaming solution using Event or IoT Hubs and Azure data processing components.

Event Hubs



Event Hubs is a partitioned consumer messaging services

- Publish and subscribe to streams of records
  - Similar to a message queue or enterprise messaging system
- Store streams of records in a fault-tolerant manner
- Process streams of records "as they occur"

- Ideal for building applications that transform or react to streams of data

Event Hubs

As a scalable data streaming platform with an event ingestion service, Azure Event Hubs can be used to receive and process millions of events per second. It can process telemetry, events, or even information produced by distributed software and devices as well as store said information. Information sent to an event hub can become converted or stored using any real-time analytics provider or batching/storage adapter.

Azure Event Hubs features and role

Azure Event Hubs provides various features that can assist with storing and processing events. It can provide an entity to send data to event hubs, capable of publishing via AMQP 1.0 or HTTPS. Also, it can capture Event Hubs streaming data to store in an Azure Blob storage account. It can allow you to partition the data, allowing each consumer to only read a specific subset of the event stream, along with the capabilities for each consumer to act independently.

Also, they can identify and authenticate the event publisher though Shared Access Signature Token, or SAS Token. Azure Event Hubs use SAS tokens generated from a SAS key to regenerate the hash and authenticate the sender.

Also, Azure Event Hubs also contains Throughput units or Pre-purchased units of capacity. Throughput units can include the capacity of either 1 MB per second or 100 events per second, whichever comes first if its Ingress while maintaining up to 2 MB per second if its egress. These units are billed hourly for a minimum of one hour, and up to a maximum of 20 throughput units per Event Hubs namespace.

Azure Event Hubs commonly can be used as a “front line” for an event pipeline of a solution architecture, sometimes known as an event investor. An event investor is a service or component that sits between the event consumers and publishers to separate the production of an event stream that obtained said events. Azure Event Hubs can provide message stream handling, but the capabilities of said service are different from traditional enterprise messaging.

The following figure depicts this architecture:

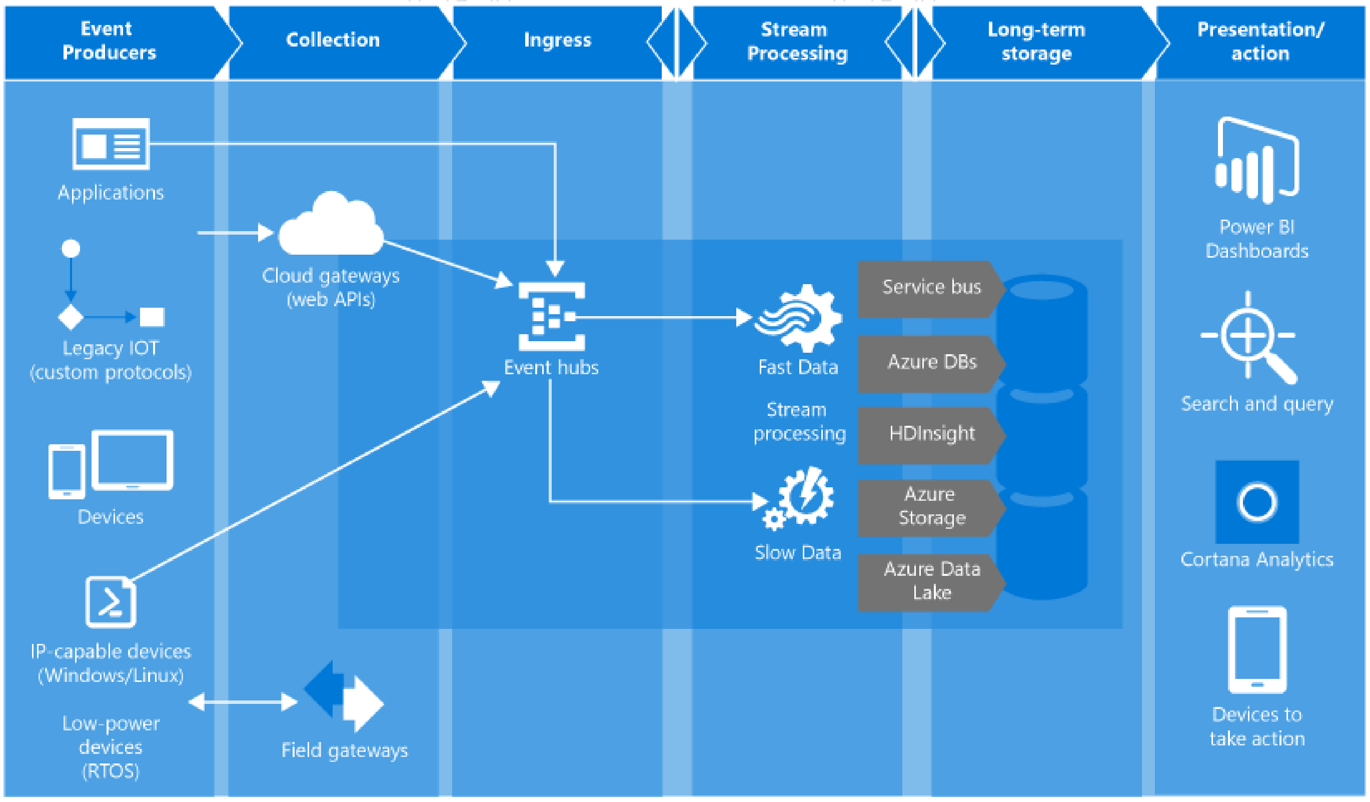


FIGURE 11.3: EVENT HUB ARCHITECTURE

IoT Hubs

Azure IoT hub service is a managed service that can provide reliable and secure bi-directional communication between a multitude of IoT devices with a solution back end. The service can provide a multitude of functions such as providing multiple device-to-cloud and cloud-to-device communication options, a declarative message routing option built in that sends the message to other Azure services, and more.

In addition to its features, Azure IoT Hub provides a way to clarify message routes based on routing rules to control where the hub can send device-to-cloud messages. These rules do not require you to write any code to implement. You can also receive detailed logs for identifying connectivity events.

Azure IoT Hub Service and Azure Event Hubs are two separate products despite the similar description. IoT can provide additional features such as Device twins, which can be used to store and research device state information. While IoT provides the device-to-cloud and cloud-to-device communication options mentioned in the previous paragraph, Event hubs can only provide event ingress communication.

When using Azure IoT Hub, it implements a service-assisted communication pattern to act as a mediator between your devices and the solution in the back end. The communication provided to establish bi-directional, reliable communication path between a control system.

To maintain such a communication path, Azure IoT follows a set of principles. It focuses first and foremost on Security, not allowing it to accept any unsolicited network information. It even uses a device dedicated to this, which establish all connections and routes in an outbound direction only. The path between both device and service or even between a device and gateway must be secured at the application protocol layer.

In an IoT solution, a gateway can typically be either a protocol gateway deployed in a cloud or a field gateway deployed locally with a device. With a field gateway, you can make time-sensitive decisions, run analytics on edge, provide device management services or even enforce security and privacy constraints. With a protocol gateway protocol, translations are carried out, such as MQTT to AMQP.

Comparing Event Hubs and IoT Hubs

Event Hubs and IoT Hubs are similar services but they each have unique differences detailed in this table:

Area	IoT Hub	Event Hubs
Communication patterns	Enables **device-to-cloud communications** (messaging, file uploads, and reported properties) and **cloud-to-device communications** (direct methods, desired properties, messaging).	Only enables event ingress (usually considered for device-to-cloud scenarios).
Device state information	Device twins can store and query device state information.	No device state information can be stored.
Device protocol support	Supports MQTT, MQTT over WebSockets, AMQP, AMQP over WebSockets, and HTTPS. Additionally, IoT Hub works with the Azure IoT protocol gateway, a customizable protocol gateway implementation to support custom protocols.	Supports AMQP, AMQP over WebSockets, and HTTPS.
Security	Provides per-device identity and revocable access control. See the Security section of the IoT Hub developer guide.	Provides Event Hubs-wide shared access policies, with limited revocation support through publisher's policies. IoT solutions are often required to implement a custom solution to support per-device credentials and anti-spoofing measures.
Operations monitoring	Enables IoT solutions to subscribe to a rich set of device identity management and connectivity events such as individual device authentication errors, throttling, and bad format exceptions. These events enable you to quickly identify connectivity problems at the individual device level.	Exposes only aggregate metrics.
Scale	Is optimized to support millions of simultaneously connected devices.	Meters the connections as per Azure Event Hubs quotas. On the other hand, Event Hubs enables you to specify the partition for each message sent.

Area	IoT Hub	Event Hubs
Device SDKs	Provides device SDKs for a large variety of platforms and languages, in addition to direct MQTT, AMQP, and HTTPS APIs.	Is supported on .NET, Java, and C, in addition to AMQP and HTTPS send interfaces.
File upload	Enables IoT solutions to upload files from devices to the cloud. Includes a file notification endpoint for workflow integration and an operations monitoring category for debugging support.	Not supported.
Route messages to multiple endpoints	Up to 10 custom endpoints are supported. Rules determine how messages are routed to custom endpoints. For more information, see Send and receive messages with IoT Hub.	Requires additional code to be written and hosted for message dispatching.

IoT Remote Monitoring

You can use an IoT Hub to receive messages from a device and then push them to a Logic App for processing. The Logic App can send notifications for messages that cross specific thresholds and store aggregate data about all messages to a database.

Time Series Insights

The Azure Time Series Insights service is a product built for visualizing, storing, and querying vast amounts of time series information, such as information generated by IoT devices. If you have plans to store, manage query, or even visualize time series data in the cloud, Azure Time Series Insights may be the product for you.

For those unsure, whether their information could be considered time series a few factors can help determine the difference for you. By definition, time series data shows how an asset or process changes over time. It has a timestamp that is unique to its kind which is most meaningful as an axis. The time series data arrives in time order and can usually be an insert rather than an update to your database. Because time series data obtains and stores every new event as a row, the changes can be measured over a time frame, enabling one to not only look back into the past but also predict the future with the data.

You may want to consider using Azure Time Series Insights in various situations, including:

- Storing and maintaining time series data in a scalable format
  - Azure Time Series Insights provides a database with time series data in mind. It handles the work of storing and maintaining events due to its nature as a scalable and fully managed database.
- Near real-time data visualization
  - Azure Time Series Insights can provide an explorer to visualize data streaming into an environment. When you connect an event source, the event data can be viewed, queried, and explored with Azure Time Series Insights.
- Producing customer applications
  - You can build applications that use time series data using the exposed REST Query APIs provided by Azure Time Series Insights.
- Needing a global view of time series data streaming from different locations for multiple sites

or assets.

- Azure Time Series Insights allows you to connect multiple event sources to the environment. It means that data streaming in different or multiple locations can be viewed together in near real-time. Users of Azure Time Series Insights can use this visibility to share data with business leaders as well as provide better collaboration with domain experts who can apply their expertise to help solve problems.

Azure Time Series also has a multitude of capabilities. It requires no upfront data preparation, making setup a quick process. It can obtain and store millions of sensor events per day with a one-minute latency, giving it the ability to provide near real-time insights. One can even embed Azure Time Series Insights data into existing applications, allowing one to create custom solutions with sharing capabilities for others to explore your insights.

Lab: Deploying Messaging Components to Facilitate Communication Between Azure Resources

Scenario

An established healthcare client has used Service Bus extensively throughout the years to manage communication between their custom software components. After learning about Logic Apps, the client would like to create Logic Apps that can consume Service Bus queue messages so that they can minimize the amount of code that their team needs to write and maintain annually.

Objectives

- Deploy Service Bus namespace using ARM Template.
- Deploy Logic App using ARM Template.
- Consume Queue Messages using Logic App.

Lab setup

Estimated Time: 60 minutes

Virtual machine: **20535A-SEA-ARCH**

User name: **Admin**

Password: **Pa55w.rd**

The lab steps for this course change frequently due to updates to Microsoft Azure. Microsoft Learning updates the lab steps frequently, so they are not available in this manual. Your instructor will provide you with the lab documentation.

Exercise 1: Deploy Service Bus Namespace

Exercise 2: Deploy Logic App

Exercise 3: Cleanup Subscription

Review Question(s)

Module review and takeaways

Review Question(s)