Contents

[Logic Apps 1](#_Toc50680398)

[Key terms 2](#_Toc50680399)

[Azure Functions 3](#_Toc50680400)

[Event Grid ( IFTT for Azure service & plugins) 4](#_Toc50680401)

[Terminology 4](#_Toc50680402)

[Event Hub ( Kafka ) 5](#_Toc50680403)

[Key architecture components 6](#_Toc50680404)

[Features and terminology in Azure Event Hubs 6](#_Toc50680405)

[Namespace 6](#_Toc50680406)

[Event Hubs for Apache Kafka 7](#_Toc50680407)

[Event publishers 7](#_Toc50680408)

[Publishing an event 7](#_Toc50680409)

[Capture 7](#_Toc50680410)

[SAS tokens 7](#_Toc50680411)

[Consumer groups 7](#_Toc50680412)

[Event Hub FAQs 8](#_Toc50680413)

[Azure Relay 8](#_Toc50680414)

[Service Bus 8](#_Toc50680415)

[Event vs. message services 9](#_Toc50680416)

[Event 9](#_Toc50680417)

[Message 9](#_Toc50680418)

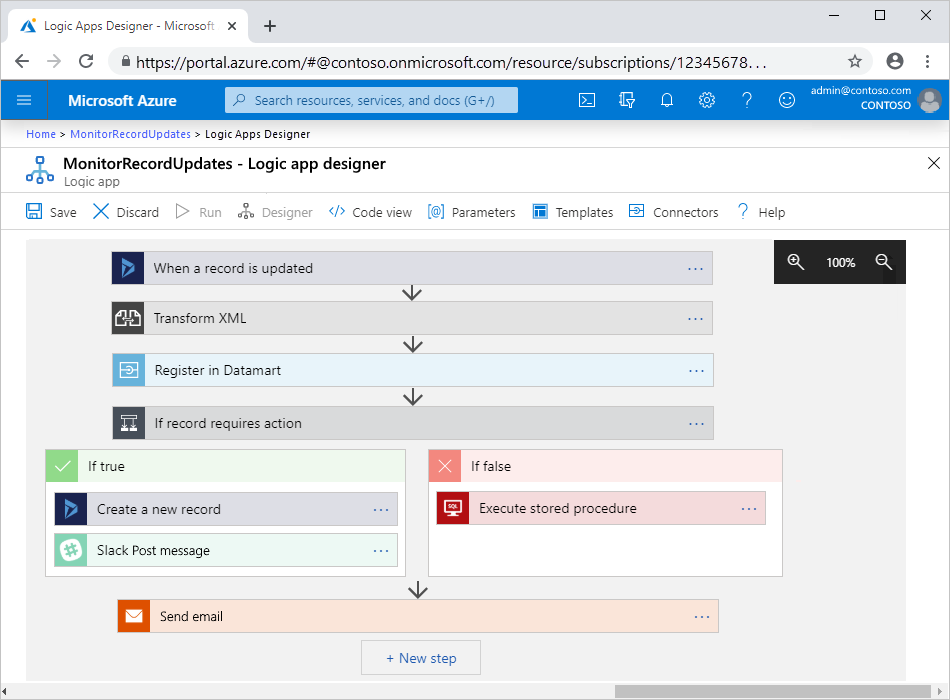
[Event Grid vs Event Hub vs Service Bus 10](#_Toc50680419)

# Logic Apps

<https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview>

[Azure Logic Apps](https://azure.microsoft.com/services/logic-apps) is a cloud service that helps you schedule, automate, and orchestrate tasks, business processes, and [workflows](https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-overview#logic-app-concepts) when you need to integrate apps, data, systems, and services across enterprises or organizations.

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.



## Key terms

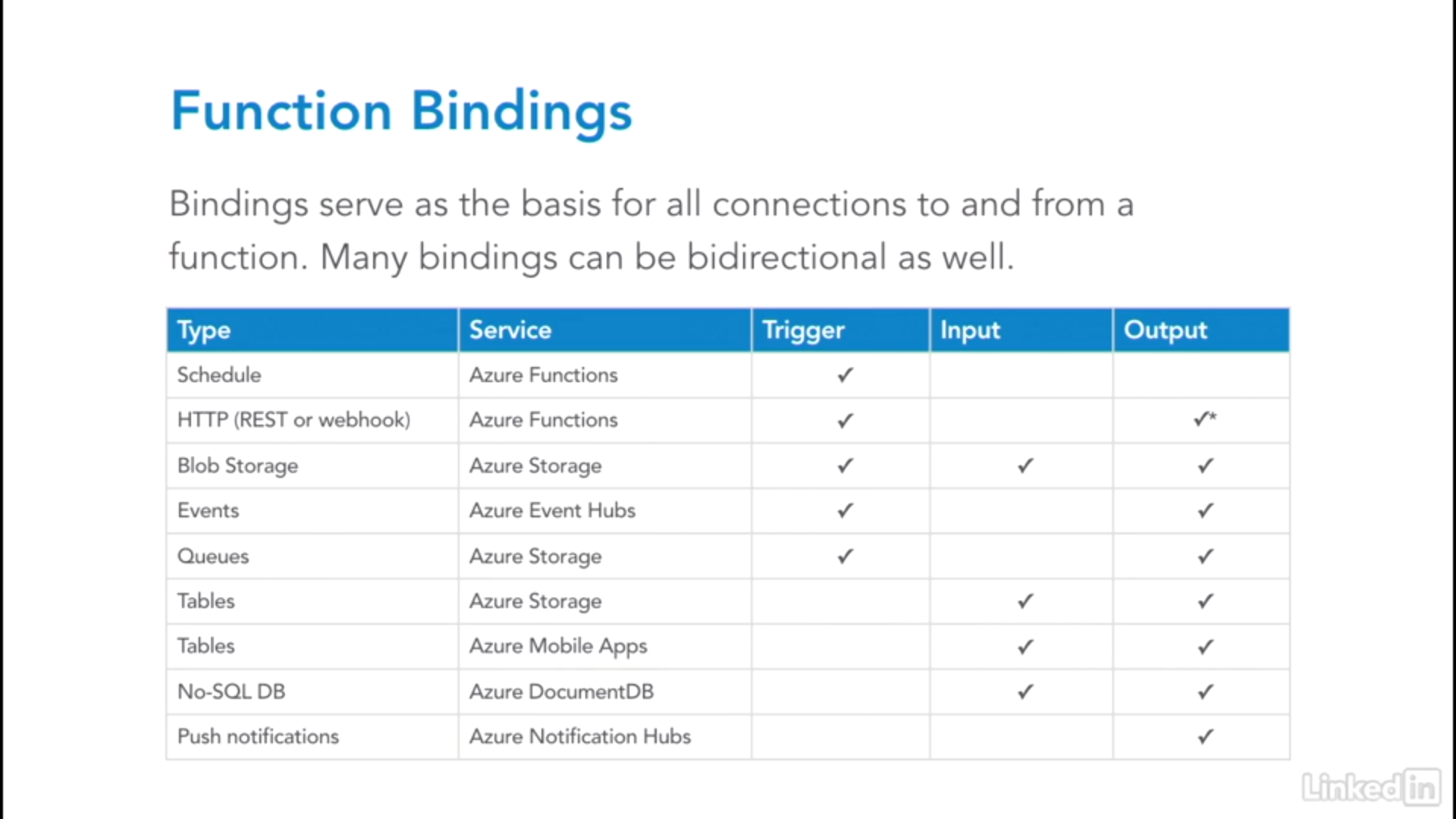
* **Workflow**: Visualize, design, build, automate, and deploy business processes as series of steps.
* **Managed connectors**: Your logic apps need access to data, services, and systems. You can use prebuilt Microsoft-managed connectors that are designed to connect, access, and work with your data. See [Connectors for Azure Logic Apps](https://docs.microsoft.com/en-us/azure/connectors/apis-list).
* **Triggers**: Many Microsoft-managed connectors provide triggers that fire when events or new data meet specified conditions. For example, an event might be getting an email or detecting changes in your Azure Storage account. Each time the trigger fires, the Logic Apps engine creates a new logic app instance that runs the workflow.
* **Actions**: Actions are all the steps that happen after the trigger. Each action usually maps to an operation that's defined by a managed connector, custom API, or custom connector.
* **Enterprise Integration Pack**: For more advanced integration scenarios, Logic Apps includes capabilities from BizTalk Server. The Enterprise Integration Pack provides connectors that help logic apps easily perform validation, transformation, and more.

<https://docs.microsoft.com/en-us/azure/logic-apps/quickstart-create-first-logic-app-workflow>

# Azure Functions

In Azure Functions, a function app provides the execution context for your individual functions. Function app behaviors apply to all functions hosted by a given function app. All functions in a function app must be of the same [language](https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages).

|  | **Durable Functions** | **Logic Apps** |
| --- | --- | --- |
| **Development** | Code-first (imperative) | Designer-first (declarative) |
| **Connectivity** | [About a dozen built-in binding types](https://docs.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings#supported-bindings), write code for custom bindings | [Large collection of connectors](https://docs.microsoft.com/en-us/azure/connectors/apis-list), [Enterprise Integration Pack for B2B scenarios](https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-enterprise-integration-overview), [build custom connectors](https://docs.microsoft.com/en-us/azure/logic-apps/custom-connector-overview) |
| **Actions** | Each activity is an Azure function; write code for activity functions | [Large collection of ready-made actions](https://docs.microsoft.com/en-us/azure/logic-apps/logic-apps-workflow-actions-triggers) |
| **Monitoring** | [Azure Application Insights](https://docs.microsoft.com/en-us/azure/azure-monitor/app/app-insights-overview) | [Azure portal](https://docs.microsoft.com/en-us/azure/logic-apps/quickstart-create-first-logic-app-workflow), [Azure Monitor logs](https://docs.microsoft.com/en-us/azure/logic-apps/monitor-logic-apps) |
| **Management** | [REST API](https://docs.microsoft.com/en-us/azure/azure-functions/durable/durable-functions-http-api), [Visual Studio](https://docs.microsoft.com/en-us/visualstudio/azure/vs-azure-tools-resources-managing-with-cloud-explorer?view=vs-2019) | [Azure portal](https://docs.microsoft.com/en-us/azure/logic-apps/quickstart-create-first-logic-app-workflow), [REST API](https://docs.microsoft.com/en-us/rest/api/logic/), [PowerShell](https://docs.microsoft.com/en-us/powershell/module/az.logicapp), [Visual Studio](https://docs.microsoft.com/en-us/azure/logic-apps/manage-logic-apps-with-visual-studio) |
| **Execution context** | Can run [locally](https://docs.microsoft.com/en-us/azure/azure-functions/functions-runtime-overview) or in the cloud | Runs only in the cloud |



It has Kudu Console to debug and see meta information

# Messaging System

Design consideration and patterns

<https://docs.microsoft.com/en-us/azure/architecture/guide/technology-choices/messaging>

1. Event Grid
2. Event Hub
3. Service Bus

<https://azure.microsoft.com/en-us/blog/events-data-points-and-messages-choosing-the-right-azure-messaging-service-for-your-data/>

# Event Grid ( IFTT for Azure service & plugins)

<https://medium.com/microsoftazure/azure-event-grid-the-whole-story-4b7b4ec4ad23>

[Azure Event Grid](https://jlik.me/e1r) is a cloud service that provides infrastructure for event-driven computing. Event Grid focuses on **events** ( messages ) that declare, “something happened.”

Most Azure services automatically send messages through Event Grid and many can directly consume messages “out of the box.” Azure Event Grid supports a “push model.” In many cases it can also lead to cost savings because it removes the overhead of polling on a regular basis and instead triggers code only when it is needed to consume an event.

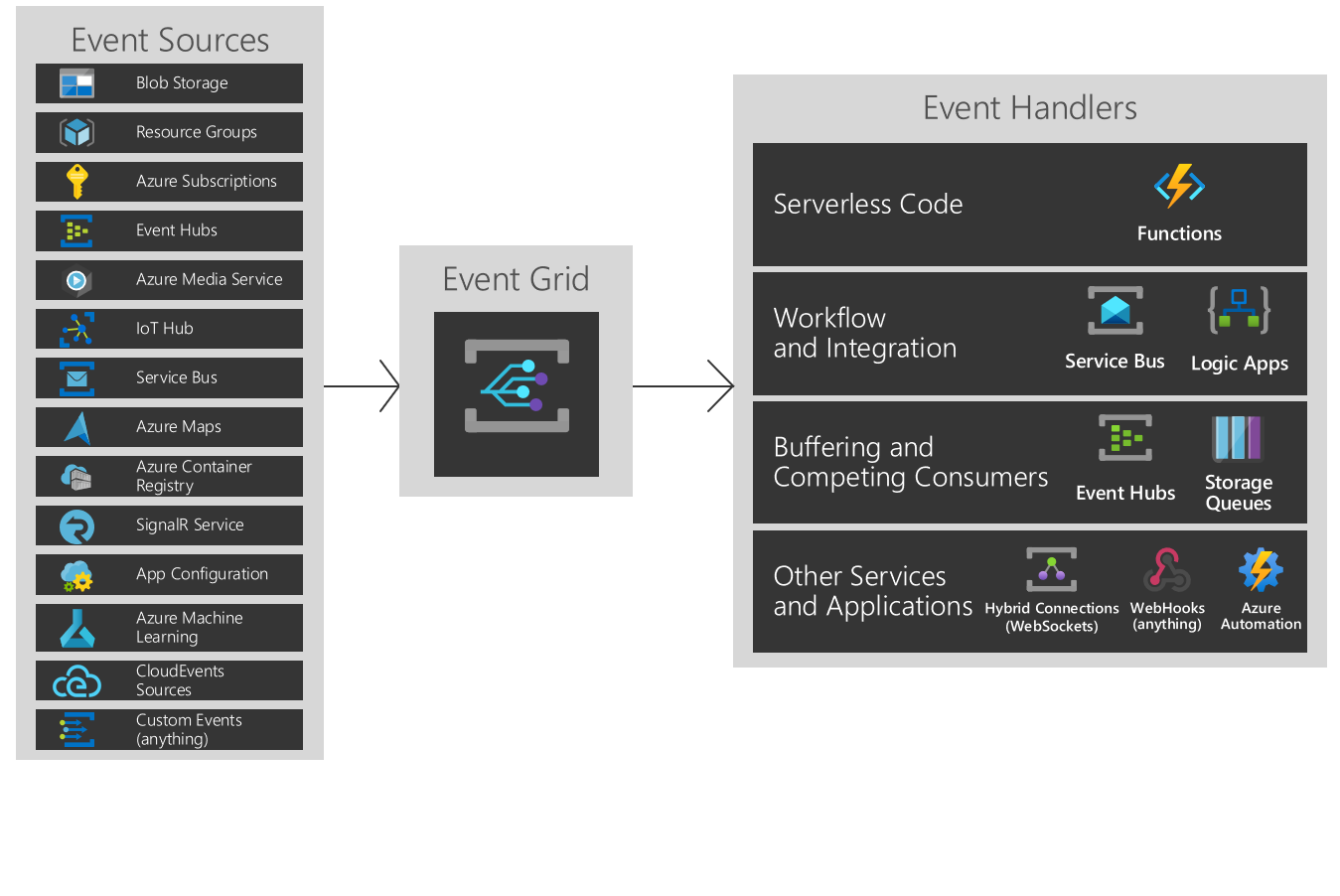
Here’s a list of the major key features of Azure Event Grid:

* Fully managed event routing service
* Built to support event-driven and serverless applications

### Terminology

There are several concepts that are useful to understand when working with Event Grid.

* **Events** — what happened (i.e. “file was uploaded” or “SKU was added”)
* **Event Publishers** — where the event happened (i.e. “web app” or “blob storage” or “CLI tool”)
* **Topics** — a channel for related events (i.e. “storage events” or “inventory events”)
* **Event Subscriptions** — how to receive events. A subscription informs Event Grid that an event should be routed to a handler. A single event can have multiple subscriptions, and subscriptions are named so they can be unsubscribed later if need be.
* **Event Handlers** — the app or service that receives and responds to the event (i.e. “Azure Function” or “Azure Logic App” or “my custom Ruby on Rails app”)



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

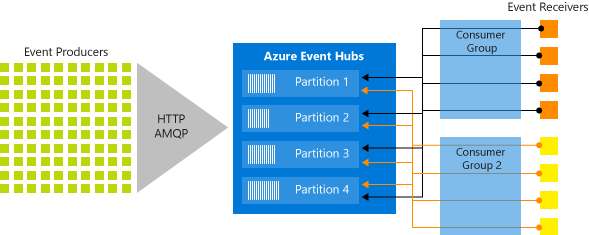
# Event Hub ( Kafka )

<https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-about>

Azure Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second.

The following scenarios are some of the scenarios where you can use Event Hubs:

* Anomaly detection (fraud/outliers)
* Application logging
* Analytics pipelines, such as clickstreams
* Live dashboarding
* Archiving data
* Transaction processing
* User telemetry processing
* Device telemetry streaming



## Key architecture components

Event Hubs contains the following [key components](https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-features):

* **Event producers**: Any entity that sends data to an event hub. Event publishers can publish events using HTTPS or AMQP 1.0 or Apache Kafka (1.0 and above)
* **Partitions**: Each consumer only reads a specific subset, or partition, of the message stream.
* **Consumer groups**: A view (state, position, or offset) of an entire event hub. Consumer groups enable consuming applications to each have a separate view of the event stream. They read the stream independently at their own pace and with their own offsets.
* **Throughput units**: Pre-purchased units of capacity that control the throughput capacity of Event Hubs.
* **Event receivers**: Any entity that reads event data from an event hub. All Event Hubs consumers connect via the AMQP 1.0 session. The Event Hubs service delivers events through a session as they become available. All Kafka consumers connect via the Kafka protocol 1.0 and later.

## Features and terminology in Azure Event Hubs

<https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-features>

### Namespace

An Event Hubs namespace provides a unique scoping container, referenced by its [fully qualified domain name](https://en.wikipedia.org/wiki/Fully_qualified_domain_name), in which you create one or **more event hubs or Kafka topics**.

### Event Hubs for Apache Kafka

[This feature](https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-for-kafka-ecosystem-overview) provides an endpoint that enables customers to talk to Event Hubs using the Kafka protocol. This integration provides customers a Kafka endpoint. **Event Hubs for Apache Kafka supports Kafka protocol 1.0 and later.**

### Event publishers

Any entity that sends data to an event hub is an event producer, or event publisher. Event publishers can publish events using HTTPS or AMQP 1.0 or Kafka 1.0 and later. Event publishers use a Shared Access Signature (SAS) token to identify themselves to an event hub, and can have a unique identity, or use a common SAS token.

### Publishing an event

You can publish an event via AMQP 1.0, Kafka 1.0 (and later), or HTTPS. A single publication (event data instance) **has a limit of 1 MB**, regardless of whether it is a single event or a batch. Publishing events larger than this threshold results in an error. It is a best practice for publishers to be unaware of partitions within the event hub and to only specify a partition key (introduced in the next section), or their identity via their SAS token.

### Capture

[Event Hubs Capture](https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-capture-overview) enables you to automatically capture the streaming data in Event Hubs and save it to your choice of either a Blob storage account, or an Azure Data Lake Service account. You can enable Capture from the Azure portal, and specify a minimum size and time window to perform the capture.

### SAS tokens

Event Hubs uses Shared Access Signatures, which are available at the namespace and event hub level. A SAS token is generated from a SAS key and is an SHA hash of a URL, encoded in a specific format. Using the name of the key (policy) and the token, Event Hubs can regenerate the hash and thus authenticate the sender.

### Consumer groups

There can be at most 5 concurrent readers on a partition per consumer group; however **it is recommended that there is only one active receiver on a partition per consumer group**. Within a single partition, each reader receives all of the messages. If you have multiple readers on the same partition, then you process duplicate messages. You need to handle this in your code, which may not be trivial. However, it's a valid approach in some scenarios.

It has the following characteristics:

* low latency
* capable of receiving and processing millions of events per second
* at least once delivery

## Event Hub FAQs

<https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-faq>

1. When do I create a new namespace vs. use an existing namespace?

Capacity allocations ([throughput units (TUs)](https://docs.microsoft.com/en-us/azure/event-hubs/event-hubs-faq#throughput-units)) are billed at the namespace level. A namespace is also associated with a region.

You may want to create a new namespace instead of using an existing one in one of the following scenarios:

* You need an Event Hub associated with a new region.
* You need an Event Hub associated with a different subscription.
* You need an Event Hub with a distinct capacity allocation (that is, the capacity need for the namespace with the added event hub would exceed the 40 TU threshold and you don't want to go for the dedicated cluster)

1. What is the difference between Event Hubs Basic and Standard tiers?

The Standard tier of Azure Event Hubs provides features beyond what is available in the Basic tier. The following features are included with Standard:

Longer event retention

Additional brokered connections, with an overage charge for more than the number included

More than a single consumer group

Capture

Kafka integration

1. What is the maximum retention period for events?

Event Hubs Standard tier currently supports a maximum **retention period of seven days**.

1. How many partitions do I need?

The number of partitions is specified at creation and must be between 2 and 32. The partition count isn't changeable, so you should consider long-term scale when setting partition count.

# Azure Relay

# Service Bus

Microsoft Azure Service Bus is a fully managed enterprise integration message broker. Service Bus can decouple applications and services. Service Bus is intended for traditional enterprise applications. These enterprise applications require transactions, ordering, duplicate detection, and instantaneous consistency.

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 solution

Data is transferred between different applications and services using **messages**. A message is in binary format and can contain JSON, XML, or just text.

**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

**Some common messaging scenarios are:**

Messaging. Transfer business data, such as sales or purchase orders, journals, or inventory movements.

* Decouple applications. Improve reliability and scalability of applications and services. Client and service don't have to be online at the same time.
* Topics and subscriptions. Enable 1:n relationships between publishers and subscribers.
* Message sessions. Implement workflows that require message ordering or message deferral.

## Event vs. message services

<https://docs.microsoft.com/en-us/azure/event-grid/compare-messaging-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.

The event data has information about what happened but doesn't have the data that triggered the event. 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.

## Event Grid vs Event Hub vs Service Bus

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

## Service Bus vs Storage Queue or Azure Queue

<https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted>

Storage queues, which are part of the Azure storage infrastructure, feature a simple REST-based GET/PUT/PEEK interface, providing reliable, persistent messaging within and between services.

As a solution architect/developer, **you should consider using Storage queues when**:

* Your application must **store over 80 GB of messages** in a queue.
* Your application wants to track progress for processing a message inside of the queue. This is useful if the worker processing a message crashes. **A subsequent worker can then use that information to continue from where the prior worker left off**.
* **You require server side logs of all of the transactions executed against your queues**.

As a solution architect/developer, **you should consider using Service Bus queues when**:

* Your solution must be able to receive messages without having to poll the queue. With Service Bus, this can be achieved through the use of the long-polling receive operation using the TCP-based protocols that Service Bus supports.
* Your solution requires the queue to provide a **guaranteed first-in-first-out (FIFO) ordered delivery**.
* Your solution must be able to support **automatic duplicate detection**.
* You want your application to process messages as parallel long-running streams (messages are associated with a stream using the [SessionId](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.sessionid) property on the message). In this model, each node in the consuming application competes for streams, as opposed to messages. When a stream is given to a consuming node, the node can examine the state of the application stream state using transactions.
* Your solution requires transactional behavior and atomicity when sending or receiving multiple messages from a queue.
* Your application handles messages that can exceed 64 KB but will not likely approach the 256 KB limit.
* You deal with a requirement to provide a role-based access model to the queues, and different rights/permissions for senders and receivers. For more information, see the following articles:
  + [Authenticate with managed identities](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-managed-service-identity)
  + [Authenticate from an application](https://docs.microsoft.com/en-us/azure/service-bus-messaging/authenticate-application)
* Your queue size will not grow larger than 80 GB.
* You want to use the AMQP 1.0 standards-based messaging protocol. For more information about AMQP, see [Service Bus AMQP Overview](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-amqp-overview).
* You can envision an eventual migration from queue-based point-to-point communication to a message exchange pattern that enables seamless integration of additional receivers (subscribers), each of which receives independent copies of either some or all messages sent to the queue. The latter refers to the publish/subscribe capability natively provided by Service Bus.
* Your messaging solution must be able to support the "At-Most-Once" delivery guarantee without the need for you to build the additional infrastructure components.
* You would like to be able to publish and consume batches of messages.

| **Comparison Criteria** | **Storage queues** | **Service Bus queues** |
| --- | --- | --- |
| Ordering guarantee | **No**  For more information, see the first note in the “Additional Information” section. | **Yes - First-In-First-Out (FIFO)**  (through the use of messaging sessions) |
| Delivery guarantee | **At-Least-Once** | **At-Least-Once** (using PeekLock receive mode - this is the default)  **At-Most-Once** (using ReceiveAndDelete receive mode)  Learn more about various [Receive modes](https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-queues-topics-subscriptions#receive-modes) |
| Atomic operation support | **No** | **Yes** |
| Receive behavior | **Non-blocking**  (completes immediately if no new message is found) | **Blocking with/without timeout**  (offers long polling, or the ["Comet technique"](https://go.microsoft.com/fwlink/?LinkId=613759))  **Non-blocking**  (through the use of .NET managed API only) |
| Push-style API | **No** | **Yes**  [QueueClient.OnMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.queueclient.onmessage#Microsoft_ServiceBus_Messaging_QueueClient_OnMessage_System_Action_Microsoft_ServiceBus_Messaging_BrokeredMessage__) and [MessageSessionHandler.OnMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.messagesessionhandler.onmessage#Microsoft_ServiceBus_Messaging_MessageSessionHandler_OnMessage_Microsoft_ServiceBus_Messaging_MessageSession_Microsoft_ServiceBus_Messaging_BrokeredMessage__) sessions .NET API. |
| Receive mode | **Peek & Lease** | **Peek & Lock**  **Receive & Delete** |
| Exclusive access mode | **Lease-based** | **Lock-based** |
| Lease/Lock duration | **30 seconds (default)**  **7 days (maximum)** (You can renew or release a message lease using the [UpdateMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.azure.storage.queue.cloudqueue.updatemessage) API.) | **60 seconds (default)**  You can renew a message lock using the [RenewLock](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.renewlock" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_RenewLock) API. |
| Lease/Lock precision | **Message level**  (each message can have a different timeout value, which you can then update as needed while processing the message, by using the [UpdateMessage](https://docs.microsoft.com/en-us/dotnet/api/microsoft.azure.storage.queue.cloudqueue.updatemessage) API) | **Queue level**  (each queue has a lock precision applied to all of its messages, but you can renew the lock using the [RenewLock](https://docs.microsoft.com/en-us/dotnet/api/microsoft.servicebus.messaging.brokeredmessage.renewlock" \l "Microsoft_ServiceBus_Messaging_BrokeredMessage_RenewLock) API.) |
| Batched receive | **Yes**  (explicitly specifying message count when retrieving messages, up to a maximum of 32 messages) | **Yes**  (implicitly enabling a pre-fetch property or explicitly through the use of transactions) |
| Batched send | **No** | **Yes**  (through the use of transactions or client-side batching) |