

# Contents

[Event Hubs Documentation](#)

[Overview](#)

[What is Event Hubs?](#)

[Terminology](#)

[Quickstarts](#)

[Create an event hub](#)

[Azure portal](#)

[Azure CLI](#)

[Azure PowerShell](#)

[ARM template](#)

[Send and receive events](#)

[.NET \(Azure.Messaging.EventHubs\)](#)

[Java \(azure-messaging-eventhubs\)](#)

[Java \(azure-eventhubs - legacy\)](#)

[Python \(azure-eventhub\)](#)

[JavaScript \(azure/event-hubs\)](#)

[Go](#)

[C \(send only\)](#)

[Apache Storm \(receive only\)](#)

[Capture events](#)

[Use the Azure portal to enable Event Hubs Capture](#)

[Use a Resource Manager template to enable Event Hubs Capture](#)

[Read captured data using Python](#)

[Stream into Event Hubs for Apache Kafka](#)

[Use Schema Registry](#)

[Create a schema registry](#)

[Validate events using a schema \(.NET and AMQP\)](#)

[Create a dedicated cluster](#)

[Tutorials](#)

Visualize data anomalies on Event Hubs data streams

Analyze fraudulent call data with Stream Analytics

Store captured data in a Azure Synapse Analytics

Process Apache Kafka for Event Hubs events using Stream analytics

Stream data into Azure Databricks using Event Hubs

Samples

Code samples

Concepts

Tiers

Premium tier

Dedicated tier

Compare tiers

Event Hubs for Apache Kafka

Event Hubs Capture

Azure Schema Registry in Event Hubs

Event processing using SDKs

Event processor host (legacy SDK)

Event processor (latest SDK)

Federation

Multi-site and multi-region federation

Event replication tasks and applications

Event replication task patterns

Configured replication tasks

Replicate events between Event Hubs

Send events to Azure Service Bus

Acquire events from Azure Service Bus

Geo-disaster recovery

Geo-disaster recovery and Geo-replication

Security

Authorization

Authorize access to Azure Event Hubs

Authorize access with Azure Active Directory



[Create a Spring Cloud Stream Binder app](#)

[Use the Spring Boot Starter for Apache Kafka](#)

[Integrate with Apache Kafka](#)

[Kafka developer guide for Event Hubs](#)

[Kafka migration guide for Event Hubs](#)

[Kafka troubleshooting guide for Event Hubs](#)

[Mirror a Kafka broker in an event hub](#)

[Connect Apache Spark to an event hub](#)

[Connect Apache Flink to an event hub](#)

[Integrate Apache Kafka Connect](#)

[Integrate Apache Kafka Connect with Debezium](#)

[Connect Akka Streams to an event hub](#)

[FAQ - Event Hubs for Kafka](#)

[Manage and monitor](#)

[Monitor Azure Event Hubs](#)

[Stream Azure Diagnostics data using Event Hubs](#)

[Automatically scale throughput units](#)

[Event Hubs management libraries](#)

[Secure](#)

[Allow access from specific IP addresses](#)

[Allow access from specific virtual networks](#)

[Allow access via private endpoints](#)

[Encrypt data using customer-managed keys](#)

[Troubleshoot](#)

[Troubleshoot connectivity issues](#)

[Troubleshoot authentication and authorization issues](#)

[Kafka troubleshooting guide for Event Hubs](#)

[EventHubsException](#)

[.NET exceptions](#)

[Resource Manager exceptions](#)

[Move](#)

[Move a namespace to another region](#)

[Move a dedicated cluster to another region](#)

## Reference

[Monitor data Reference](#)

[SDKs](#)

[.NET](#)

[Client library](#)

[Migrate from Microsoft.Azure.Event Hubs to Azure.Messaging.EventHubs](#)

[Azure.Messaging.EventHubs \(latest\)](#)

[Microsoft.Azure.EventHubs \(legacy\)](#)

[Microsoft.ServiceBus.Messaging \(legacy\)](#)

[Management library](#)

[Java](#)

[Client library](#)

[Migrate from azure-eventhubs to azure-messaging-eventhubs](#)

[azure-messaging-eventhubs \(latest\)](#)

[azure-eventhubs \(legacy\)](#)

[Management library](#)

[Python](#)

[Client library](#)

[Management library](#)

[JavaScript](#)

[Client library](#)

[Management library](#)

[Go](#)

[Client library](#)

[Management library](#)

[Azure CLI](#)

[REST](#)

[Resource Manager template](#)

[Quotas](#)

[Apache Kafka configurations](#)

[Azure Policy built-ins](#)

## Resources

[Build your skills with Microsoft Learn](#)

[FAQ](#)

[Azure Roadmap](#)

[Pricing](#)

[Pricing calculator](#)

[Service updates](#)

[Stack Overflow](#)

[Videos](#)

# Azure Event Hubs — A big data streaming platform and event ingestion service

11/26/2021 • 5 minutes to read • [Edit Online](#)

Azure Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters.

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

## Why use Event Hubs?

Data is valuable only when there's an easy way to process and get timely insights from data sources. Event Hubs provides a distributed stream processing platform with low latency and seamless integration, with data and analytics services inside and outside Azure to build your complete big data pipeline.

Event Hubs represents the "front door" for an event pipeline, often called an *event ingestor* in solution architectures. An event ingestor is a component or service that sits between event publishers and event consumers to decouple the production of an event stream from the consumption of those events. Event Hubs provides a unified streaming platform with time retention buffer, decoupling event producers from event consumers.

The following sections describe key features of the Azure Event Hubs service:

## Fully managed PaaS

Event Hubs is a fully managed Platform-as-a-Service (PaaS) with little configuration or management overhead, so you focus on your business solutions. [Event Hubs for Apache Kafka ecosystems](#) gives you the PaaS Kafka experience without having to manage, configure, or run your clusters.

## Support for real-time and batch processing

Ingest, buffer, store, and process your stream in real time to get actionable insights. Event Hubs uses a [partitioned consumer model](#), enabling multiple applications to process the stream concurrently and letting you control the speed of processing. Azure Event Hubs also integrates with [Azure Functions](#) for a serverless architecture.

## Capture event data

[Capture](#) your data in near-real time in an [Azure Blob storage](#) or [Azure Data Lake Storage](#) for long-term retention

or micro-batch processing. You can achieve this behavior on the same stream you use for deriving real-time analytics. Setting up capture of event data is fast. There are no administrative costs to run it, and it scales automatically with Event Hubs [throughput units](#) or [processing units](#). Event Hubs enables you to focus on data processing rather than on data capture.

## Scalable

With Event Hubs, you can start with data streams in megabytes, and grow to gigabytes or terabytes. The [Auto-inflate](#) feature is one of the many options available to scale the number of throughput units or processing units to meet your usage needs.

## Rich ecosystem

With a broad ecosystem based on the industry-standard AMQP 1.0 protocol and available in various languages [.NET](#), [Java](#), [Python](#), [JavaScript](#), you can easily start processing your streams from Event Hubs. All supported client languages provide low-level integration. The ecosystem also provides you with seamless integration with Azure services like Azure Stream Analytics and Azure Functions and thus enables you to build serverless architectures.

## Event Hubs for Apache Kafka

[Event Hubs for Apache Kafka ecosystems](#) furthermore enables [Apache Kafka \(1.0 and later\)](#) clients and applications to talk to Event Hubs. You don't need to set up, configure, and manage your own Kafka and Zookeeper clusters or use some Kafka-as-a-Service offering not native to Azure.

## Event Hubs premium and dedicated

Event Hubs **premium** caters to high-end streaming needs that require superior performance, better isolation with predictable latency and minimal interference in a managed multitenant PaaS environment. On top of all the features of the standard offering, the premium tier offers several extra features such as dynamic partition scale up, extended retention, and customer-managed-keys. For more information, see [Event Hubs Premium](#).

Event Hubs **dedicated** tier offers single-tenant deployments for customers with the most demanding streaming needs. This single-tenant offering has a guaranteed 99.99% SLA and is available only on our dedicated pricing tier. An Event Hubs cluster can ingest millions of events per second with guaranteed capacity and subsecond latency. Namespaces and event hubs created within the dedicated cluster include all features of the premium offering and more. For more information, see [Event Hubs Dedicated](#).

See [comparison between Event Hubs tiers](#) for more details.

## Event Hubs on Azure Stack Hub

Event Hubs on Azure Stack Hub allows you to realize hybrid cloud scenarios. Streaming and event-based solutions are supported, for both on-premises and Azure cloud processing. Whether your scenario is hybrid (connected), or disconnected, your solution can support processing of eventsstreams at large scale. Your scenario is only bound by the Event Hubs cluster size, which you can provision according to your needs.

The Event Hubs editions (on Azure Stack Hub and on Azure) offer a high degree of feature parity. This parity means SDKs, samples, PowerShell, CLI, and portals offer a similar experience, with few differences.

Event Hubs on Stack is free during public preview. For more information, see [Event Hubs on Azure Stack Hub overview](#).

## Key architecture components

Event Hubs contains the following [key components](#):

- **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 or processing 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.

The following figure shows the Event Hubs stream processing architecture:

## Next steps

To get started using Event Hubs, see the [Send and receive events](#) tutorials:

- [.NET Core](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [Go](#)
- [C \(send only\)](#)
- [Apache Storm \(receive only\)](#)

To learn more about Event Hubs, see the following articles:

- [Event Hubs features overview](#)
- [Frequently asked questions](#).

# Features and terminology in Azure Event Hubs

11/26/2021 • 15 minutes to read • [Edit Online](#)

Azure Event Hubs is a scalable event processing service that ingests and processes large volumes of events and data, with low latency and high reliability. See [What is Event Hubs?](#) for a high-level overview.

This article builds on the information in the [overview article](#), and provides technical and implementation details about Event Hubs components and features.

## TIP

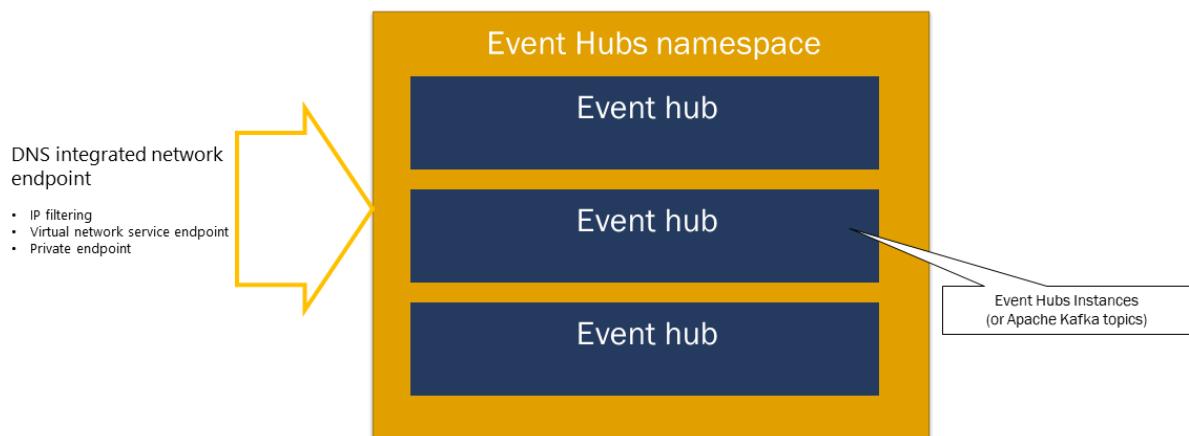
The protocol support for [Apache Kafka clients](#) (versions >= 1.0) provides network endpoints that enable applications built to use Apache Kafka with any client to use Event Hubs. Most existing Kafka applications can simply be reconfigured to point to an Event Hub namespace instead of a Kafka cluster bootstrap server.

From the perspective of cost, operational effort, and reliability, Azure Event Hubs is a great alternative to deploying and operating your own Kafka and Zookeeper clusters and to Kafka-as-a-Service offerings not native to Azure.

In addition to getting the same core functionality as of the Apache Kafka broker, you also get access to Azure Event Hub features like automatic batching and archiving via [Event Hubs Capture](#), automatic scaling and balancing, disaster recovery, cost-neutral availability zone support, flexible and secure network integration, and multi-protocol support including the firewall-friendly AMQP-over-WebSockets protocol.

## Namespace

An Event Hubs namespace is a management container for event hubs (or topics, in Kafka parlance). It provides DNS-integrated network endpoints and a range of access control and network integration management features such as [IP filtering](#), [virtual network service endpoint](#), and [Private Link](#).



## Event publishers

Any entity that sends data to an event hub is an *event publisher* (synonymously used with *event producer*). Event publishers can publish events using HTTPS or AMQP 1.0 or the Kafka protocol. Event publishers use Azure Active Directory based authorization with OAuth2-issued JWT tokens or an Event Hub-specific Shared Access Signature (SAS) token gain publishing access.

### Publishing an event

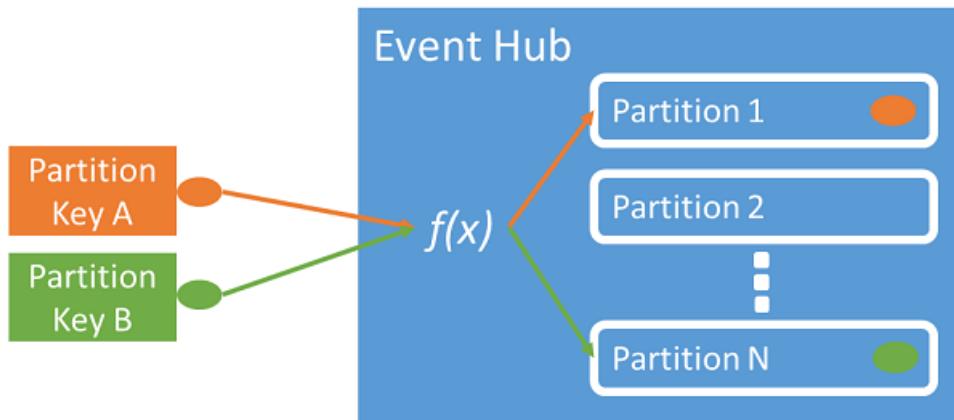
You can publish an event via AMQP 1.0, the Kafka protocol, or HTTPS. The Event Hubs service provides [REST API](#) and [.NET, Java, Python, JavaScript](#), and [Go](#) client libraries for publishing events to an event hub. For other

runtimes and platforms, you can use any AMQP 1.0 client, such as [Apache Qpid](#).

The choice to use AMQP or HTTPS is specific to the usage scenario. AMQP requires the establishment of a persistent bidirectional socket in addition to transport level security (TLS) or SSL/TLS. AMQP has higher network costs when initializing the session, however HTTPS requires additional TLS overhead for every request. AMQP has significantly higher performance for frequent publishers and can achieve much lower latencies when used with asynchronous publishing code.

You can publish events individually or batched. A single publication has a limit of 1 MB, regardless of whether it's a single event or a batch. Publishing events larger than this threshold will be rejected.

Event Hubs throughput is scaled by using partitions and throughput-unit allocations (see below). It's a best practice for publishers to remain unaware of the specific partitioning model chosen for an event hub and to only specify a *partition key* that is used to consistently assign related events to the same partition.



Event Hubs ensures that all events sharing a partition key value are stored together and delivered in order of arrival. If partition keys are used with publisher policies, then the identity of the publisher and the value of the partition key must match. Otherwise, an error occurs.

### Event retention

Published events are removed from an event hub based on a configurable, timed-based retention policy. Here are a few important points:

- The **default value and shortest** possible retention period is **1 day (24 hours)**.
- For Event Hubs **Standard**, the maximum retention period is **7 days**.
- For Event Hubs **Premium** and **Dedicated**, the maximum retention period is **90 days**.
- If you change the retention period, it applies to all messages including messages that are already in the event hub.

Event Hubs retains events for a configured retention time that applies across all partitions. Events are automatically removed when the retention period has been reached. If you specify a retention period of one day, the event will become unavailable exactly 24 hours after it has been accepted. You cannot explicitly delete events.

If you need to archive events beyond the allowed retention period, you can have them automatically stored in Azure Storage or Azure Data Lake by turning on the [Event Hubs Capture feature](#). If you need to search or analyze such deep archives, you can easily import them into [Azure Synapse](#) or other similar stores and analytics platforms.

The reason for Event Hubs' limit on data retention based on time is to prevent large volumes of historic customer data getting trapped in a deep store that is only indexed by a timestamp and only allows for sequential access. The architectural philosophy here's that historic data needs richer indexing and more direct access than the real-time eventing interface that Event Hubs or Kafka provide. Event stream engines aren't well suited to play the role of data lakes or long-term archives for event sourcing.

#### NOTE

Event Hubs is a real-time event stream engine and is not designed to be used instead of a database and/or as a permanent store for infinitely held event streams.

The deeper the history of an event stream gets, the more you will need auxiliary indexes to find a particular historical slice of a given stream. Inspection of event payloads and indexing aren't within the feature scope of Event Hubs (or Apache Kafka). Databases and specialized analytics stores and engines such as [Azure Data Lake Store](#), [Azure Data Lake Analytics](#) and [Azure Synapse](#) are therefore far better suited for storing historic events.

[Event Hubs Capture](#) integrates directly with Azure Blob Storage and Azure Data Lake Storage and, through that integration, also enables [flowing events directly into Azure Synapse](#).

If you want to use the [Event Sourcing](#) pattern for your application, you should align your snapshot strategy with the retention limits of Event Hubs. Do not aim to rebuild materialized views from raw events starting at the beginning of time. You would surely come to regret such a strategy once your application is in production for a while and is well used, and your projection builder has to churn through years of change events while trying to catch up to the latest and ongoing changes.

#### Publisher policy

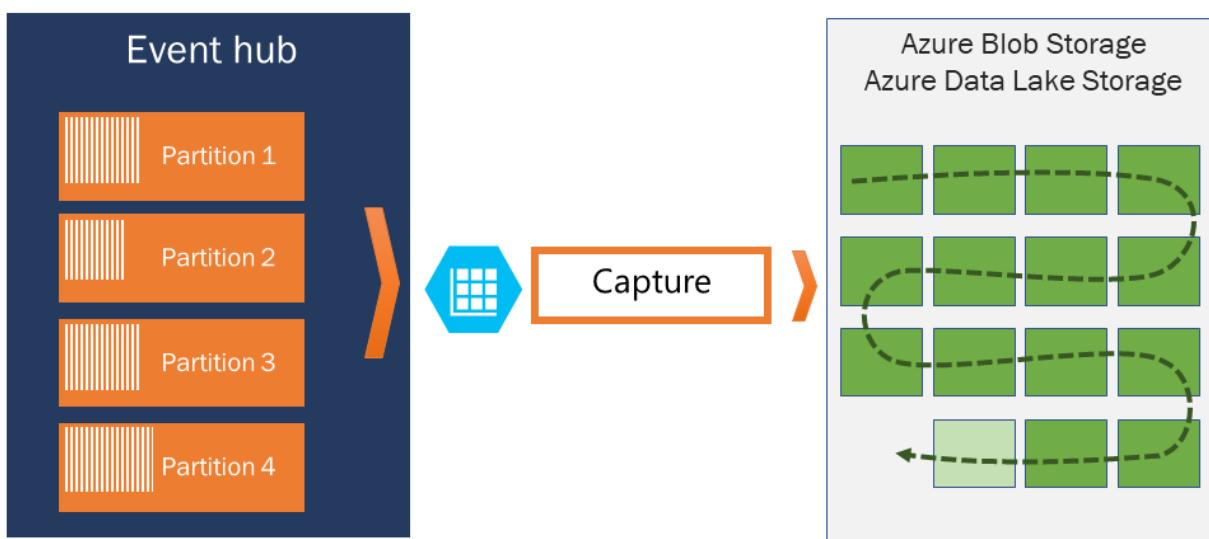
Event Hubs enables granular control over event publishers through *publisher policies*. Publisher policies are run-time features designed to facilitate large numbers of independent event publishers. With publisher policies, each publisher uses its own unique identifier when publishing events to an event hub, using the following mechanism:

```
//<my namespace>.servicebus.windows.net/<event hub name>/publishers/<my publisher name>
```

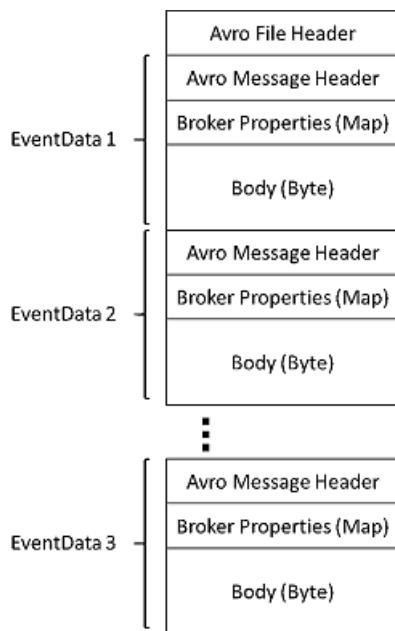
You don't have to create publisher names ahead of time, but they must match the SAS token used when publishing an event, in order to ensure independent publisher identities. When using publisher policies, the **PartitionKey** value is set to the publisher name. To work properly, these values must match.

## Capture

[Event Hubs Capture](#) 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 Storage account. You can enable capture from the Azure portal, and specify a minimum size and time window to perform the capture. Using Event Hubs Capture, you specify your own Azure Blob Storage account and container, or Azure Data Lake Storage account, one of which is used to store the captured data. Captured data is written in the Apache Avro format.

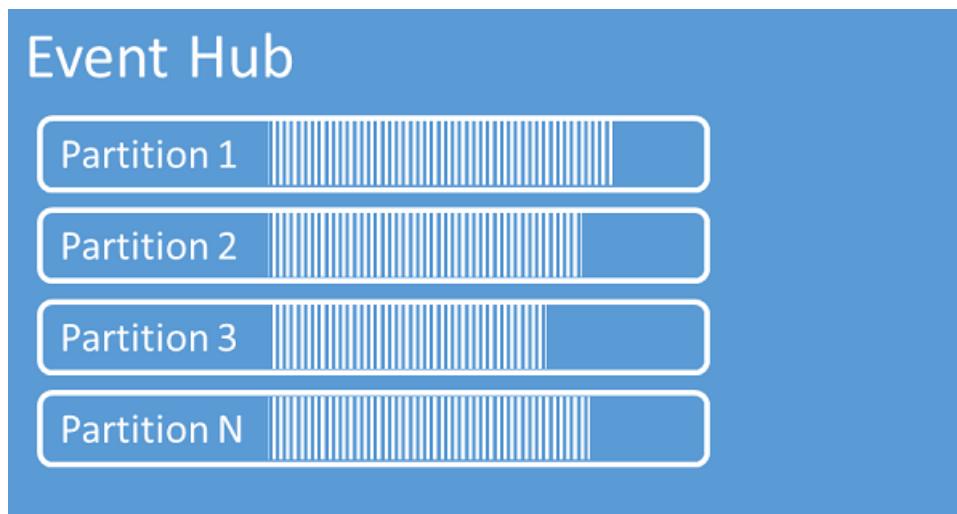


The files produced by Event Hubs Capture have the following Avro schema:

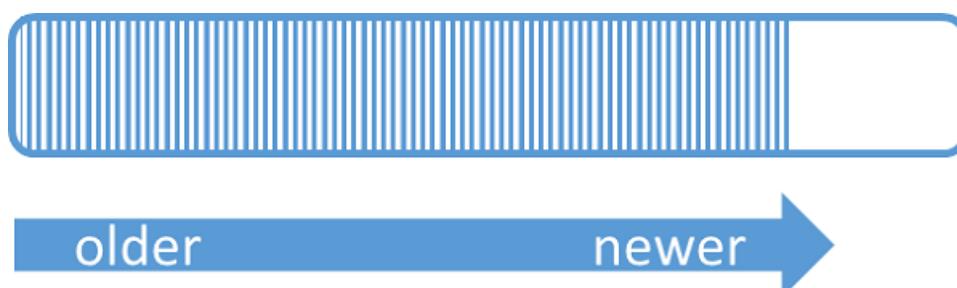


## Partitions

Event Hubs organizes sequences of events sent to an event hub into one or more partitions. As newer events arrive, they're added to the end of this sequence.



A partition can be thought of as a "commit log". Partitions hold event data that contains body of the event, a user-defined property bag describing the event, metadata such as its offset in the partition, its number in the stream sequence, and service-side timestamp at which it was accepted.



### Advantages of using partitions

Event Hubs is designed to help with processing of large volumes of events, and partitioning helps with that in two ways:

- Even though Event Hubs is a PaaS service, there's a physical reality underneath, and maintaining a log that preserves the order of events requires that these events are being kept together in the underlying storage and its replicas and that results in a throughput ceiling for such a log. Partitioning allows for multiple parallel logs to be used for the same event hub and therefore multiplying the available raw IO throughput capacity.
- Your own applications must be able to keep up with processing the volume of events that are being sent into an event hub. It may be complex and requires substantial, scaled-out, parallel processing capacity. The capacity of a single process to handle events is limited, so you need several processes. Partitions are how your solution feeds those processes and yet ensures that each event has a clear processing owner.

## Number of partitions

The number of partitions is specified at the time of creating an event hub. It must be between 1 and the maximum partition count allowed for each pricing tier. For the partition count limit for each tier, see [this article](#).

We recommend that you choose at least as many partitions as you expect that are required during the peak load of your application for that particular event hub. You can't change the partition count for an event hub after its creation except for the event hub in a dedicated cluster and premium tier. The partition count for an event hub in a [dedicated Event Hubs cluster](#) can be [increased](#) after the event hub has been created, but the distribution of streams across partitions will change when it's done as the mapping of partition keys to partitions changes, so you should try hard to avoid such changes if the relative order of events matters in your application.

Setting the number of partitions to the maximum permitted value is tempting, but always keep in mind that your event streams need to be structured such that you can indeed take advantage of multiple partitions. If you need absolute order preservation across all events or only a handful of substreams, you may not be able to take advantage of many partitions. Also, many partitions make the processing side more complex.

It doesn't matter how many partitions are in an event hub when it comes to pricing. It depends on the number of pricing units ([throughput units \(TUs\)](#) for the standard tier, [processing units \(PUs\)](#) for the premium tier, and [capacity units \(CUs\)](#) for the dedicated tier) for the namespace or the dedicated cluster. For example, an event hub of the standard tier with 32 partitions or with 1 partition incur the exact same cost when the namespace is set to 1 TU capacity. Also, you can scale TUs or PUs on your namespace or CUs of your dedicated cluster independent of the partition count.

## Mapping of events to partitions

You can use a partition key to map incoming event data into specific partitions for the purpose of data organization. The partition key is a sender-supplied value passed into an event hub. It is processed through a static hashing function, which creates the partition assignment. If you don't specify a partition key when publishing an event, a round-robin assignment is used.

The event publisher is only aware of its partition key, not the partition to which the events are published. This decoupling of key and partition insulates the sender from needing to know too much about the downstream processing. A per-device or user unique identity makes a good partition key, but other attributes such as geography can also be used to group related events into a single partition.

Specifying a partition key enables keeping related events together in the same partition and in the exact order in which they arrived. The partition key is some string that is derived from your application context and identifies the interrelationship of the events. A sequence of events identified by a partition key is a *stream*. A partition is a multiplexed log store for many such streams.

### NOTE

While you can send events directly to partitions, we don't recommend it, especially when high availability is important to you. It downgrades the availability of an event hub to partition-level. For more information, see [Availability and Consistency](#).

## 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. Normally, SAS tokens for event publishers are created with only **send** privileges on a specific event hub. This SAS token URL mechanism is the basis for publisher identification introduced in the publisher policy. For more information about working with SAS, see [Shared Access Signature Authentication with Service Bus](#).

## Event consumers

Any entity that reads event data from an event hub is an *event consumer*. All Event Hubs consumers connect via the AMQP 1.0 session and events are delivered through the session as they become available. The client does not need to poll for data availability.

### Consumer groups

The publish/subscribe mechanism of Event Hubs is enabled through *consumer groups*. A consumer group is a view (state, position, or offset) of an entire event hub. Consumer groups enable multiple consuming applications to each have a separate view of the event stream, and to read the stream independently at their own pace and with their own offsets.

In a stream processing architecture, each downstream application equates to a consumer group. If you want to write event data to long-term storage, then that storage writer application is a consumer group. Complex event processing can then be performed by another, separate consumer group. You can only access partitions through a consumer group. There's always a default consumer group in an event hub, and you can create up to the [maximum number of consumer groups](#) for the corresponding pricing tier.

There can be at most 5 concurrent readers on a partition per consumer group; however **it's recommended that there's 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.

Some clients offered by the Azure SDKs are intelligent consumer agents that automatically manage the details of ensuring that each partition has a single reader and that all partitions for an event hub are being read from. This allows your code to focus on processing the events being read from the event hub so it can ignore many of the details of the partitions. For more information, see [Connect to a partition](#).

The following examples show the consumer group URI convention:

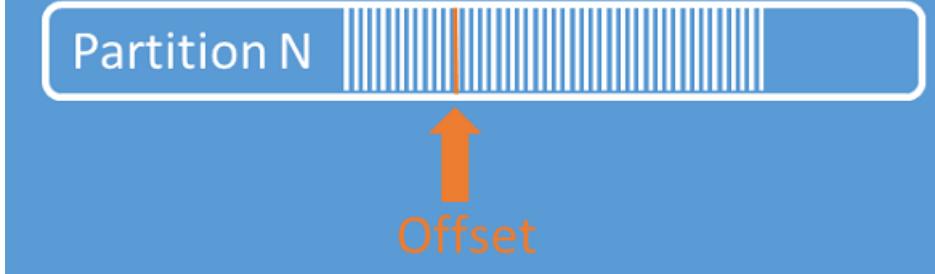
```
//<my namespace>.servicebus.windows.net/<event hub name>/<Consumer Group #1>
//<my namespace>.servicebus.windows.net/<event hub name>/<Consumer Group #2>
```

The following figure shows the Event Hubs stream processing architecture:

### Stream offsets

An *offset* is the position of an event within a partition. You can think of an offset as a client-side cursor. The offset is a byte numbering of the event. This offset enables an event consumer (reader) to specify a point in the event stream from which they want to begin reading events. You can specify the offset as a timestamp or as an offset value. Consumers are responsible for storing their own offset values outside of the Event Hubs service. Within a partition, each event includes an offset.

# Event Hub



## Checkpointing

*Checkpointing* is a process by which readers mark or commit their position within a partition event sequence. Checkpointing is the responsibility of the consumer and occurs on a per-partition basis within a consumer group. This responsibility means that for each consumer group, each partition reader must keep track of its current position in the event stream, and can inform the service when it considers the data stream complete.

If a reader disconnects from a partition, when it reconnects it begins reading at the checkpoint that was previously submitted by the last reader of that partition in that consumer group. When the reader connects, it passes the offset to the event hub to specify the location at which to start reading. In this way, you can use checkpointing to both mark events as "complete" by downstream applications, and to provide resiliency if a failover between readers running on different machines occurs. It's possible to return to older data by specifying a lower offset from this checkpointing process. Through this mechanism, checkpointing enables both failover resiliency and event stream replay.

### IMPORTANT

Offsets are provided by the Event Hubs service. It's the responsibility of the consumer to checkpoint as events are processed.

### NOTE

If you are using Azure Blob Storage as the checkpoint store in an environment that supports a different version of Storage Blob SDK than those typically available on Azure, you'll need to use code to change the Storage service API version to the specific version supported by that environment. For example, if you are running [Event Hubs on an Azure Stack Hub version 2002](#), the highest available version for the Storage service is version 2017-11-09. In this case, you need to use code to target the Storage service API version to 2017-11-09. For an example on how to target a specific Storage API version, see these samples on GitHub:

- [.NET](#)
- [Java](#)
- [JavaScript or TypeScript](#)
- [Python](#)

## Common consumer tasks

All Event Hubs consumers connect via an AMQP 1.0 session, a state-aware bidirectional communication channel. Each partition has an AMQP 1.0 session that facilitates the transport of events segregated by partition.

### Connect to a partition

When connecting to partitions, it's common practice to use a leasing mechanism to coordinate reader connections to specific partitions. This way, it's possible for every partition in a consumer group to have only one active reader. Checkpointing, leasing, and managing readers are simplified by using the clients within the

Event Hubs SDKs, which act as intelligent consumer agents. These are:

- The [EventProcessorClient](#) for .NET
- The [EventProcessorClient](#) for Java
- The [EventHubConsumerClient](#) for Python
- The [EventHubConsumerClient](#) for JavaScript/TypeScript

#### Read events

After an AMQP 1.0 session and link is opened for a specific partition, events are delivered to the AMQP 1.0 client by the Event Hubs service. This delivery mechanism enables higher throughput and lower latency than pull-based mechanisms such as HTTP GET. As events are sent to the client, each event data instance contains important metadata such as the offset and sequence number that are used to facilitate checkpointing on the event sequence.

Event data:

- Offset
- Sequence number
- Body
- User properties
- System properties

It's your responsibility to manage the offset.

## Next steps

For more information about Event Hubs, visit the following links:

- Get started with Event Hubs
  - [.NET](#)
  - [Java](#)
  - [Python](#)
  - [JavaScript](#)
- [Event Hubs programming guide](#)
- [Availability and consistency in Event Hubs](#)
- [Event Hubs FAQ](#)
- [Event Hubs samples](#)

# Quickstart: Create an event hub using Azure portal

11/26/2021 • 3 minutes to read • [Edit Online](#)

Azure Event Hubs is a Big Data streaming platform and event ingestion service that can receive and process millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see [Event Hubs overview](#) and [Event Hubs features](#).

In this quickstart, you create an event hub using the [Azure portal](#).

## Prerequisites

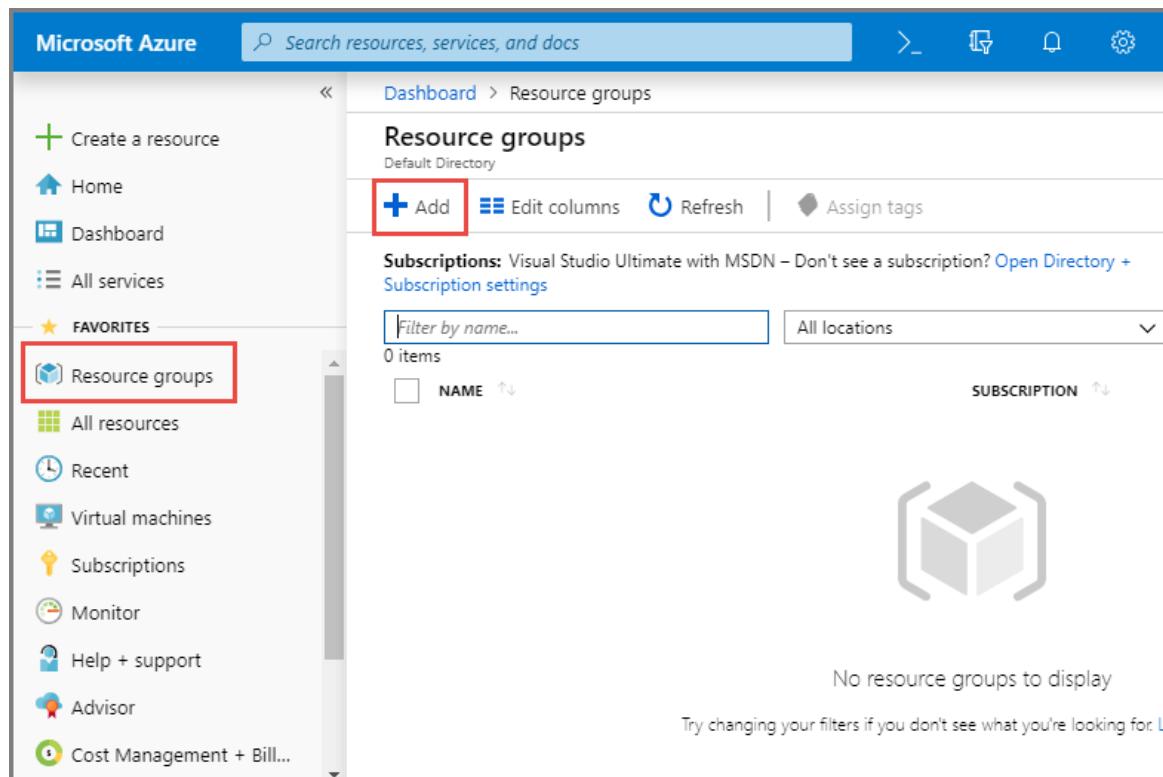
To complete this quickstart, make sure that you have:

- Azure subscription. If you don't have one, [create a free account](#) before you begin.

## Create a resource group

A resource group is a logical collection of Azure resources. All resources are deployed and managed in a resource group. To create a resource group:

1. Sign in to the [Azure portal](#).
2. In the left navigation, select **Resource groups**. Then select **Add**.

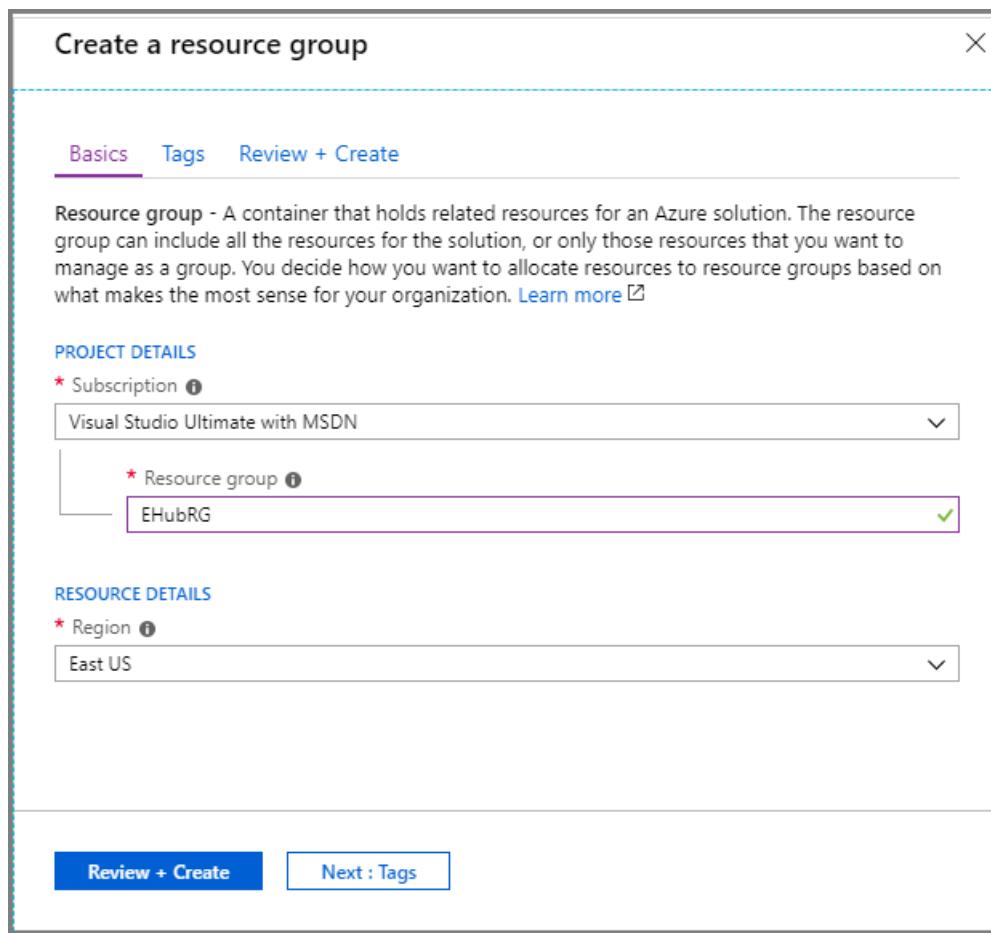


The screenshot shows the Microsoft Azure portal interface. The left sidebar has a 'FAVORITES' section with 'Resource groups' highlighted by a red box. The main content area is titled 'Resource groups' and shows a table with one row: '0 items'. The table has columns for 'NAME' and 'SUBSCRIPTION'. At the top of the table, there is a red box around the '+ Add' button. The top navigation bar includes a search bar, a dashboard icon, and other account-related icons.

3. For **Subscription**, select the name of the Azure subscription in which you want to create the resource group.
4. Type a unique **name for the resource group**. The system immediately checks to see if the name is available in the currently selected Azure subscription.

5. Select a **region** for the resource group.

6. Select **Review + Create**.



7. On the **Review + Create** page, select **Create**.

## Create an Event Hubs namespace

An Event Hubs namespace provides a unique scoping container, in which you create one or more event hubs. To create a namespace in your resource group using the portal, do the following actions:

1. In the Azure portal, and select **Create a resource** at the top left of the screen.
2. Select **All services** in the left menu, and select star (\*) next to **Event Hubs** in the **Analytics** category.  
Confirm that **Event Hubs** is added to **FAVORITES** in the left navigational menu.

**Microsoft Azure**

All services Filter By category ▾

**App Services** ★

DATABASES (16)

- Azure Cosmos DB ★
- Azure Database for PostgreSQL servers ★
- SQL data warehouses ★
- SQL Server stretch databases ★
- Virtual clusters ★
- SQL managed instances ★

ANALYTICS (14)

- SQL data warehouses ★
- Data factories ★
- Stream Analytics jobs ★
- Event Hubs ★
- Data Lake Storage Gen1 ★

SQL databases

Azure Database for MariaDB servers

Azure Database Migration Services

Data factories

Managed databases

Azure Databricks

Power BI Embedded

Data Lake Analytics

Event Hubs Clusters

Azure Data Explorer Clusters

**All services**

**FAVORITES**

- Resource groups
- All resources
- Recent
- Virtual machines
- Subscriptions
- Monitor
- Help + support
- Advisor
- Cost Management + Billing
- Azure Active Directory
- DevTest Labs
- Lab Accounts

3. Select **Event Hubs** under **FAVORITES** in the left navigational menu, and select **Add** on the toolbar.

Dashboard > Event Hubs

**Event Hubs** Default Directory

**+ Add** Edit columns Refresh Assign tags

Subscriptions: Visual Studio Ultimate with MSDN – Don't see a subscription? Open Directory + Subscription settings

Filter by name... All resource groups All locations All tags No grouping

0 items

NAME	TYPE	RESOURCE GROUP	LOCATION	SUBSCRIPTION
No event hubs namespaces to display				

Try changing your filters if you don't see what you're looking for

Create event hubs namespace

**Event Hubs**

**Resource groups**

**All resources**

**Recent**

**Virtual machines**

**Subscriptions**

**Monitor**

**Help + support**

**Advisor**

**Cost Management + Billing**

**Azure Active Directory**

**DevTest Labs**

**Lab Accounts**

4. On the **Create namespace** page, take the following steps:

- Select the **subscription** in which you want to create the namespace.
- Select the **resource group** you created in the previous step.
- Enter a **name** for the namespace. The system immediately checks to see if the name is available.
- Select a **location** for the namespace.
- Choose **Basic** for the **pricing tier**. To learn about differences between tiers, see [Quotas and limits](#), [Event Hubs Premium](#), and [Event Hubs Dedicated](#) articles.
- Leave the **throughput units** (for standard tier) or **processing units** (for premium tier) settings as it is. To learn about throughput units or processing units: [Event Hubs scalability](#).

g. Select **Review + Create** at the bottom of the page.

The screenshot shows the 'Create Namespace' wizard in the Microsoft Azure portal. The top navigation bar includes 'Microsoft Azure', a search bar, and a user profile icon. The breadcrumb path is 'All services > myehubrg > New > Event Hubs > Create Namespace'. On the left, there's a vertical sidebar with various service icons. The main content area has a title 'Create Namespace' with a 'Event Hubs' icon. Below it, tabs for 'Basics', 'Features', 'Tags', and 'Review + create' are visible, with 'Basics' being the active tab. The 'PROJECT DETAILS' section asks to select a subscription and resource group. The 'Subscription' dropdown is set to 'Visual Studio Ultimate with MSDN'. The 'Resource group' dropdown is set to 'myehubrg' with a 'Create new' link. The 'INSTANCE DETAILS' section allows setting a namespace name, location, pricing tier, and throughput units. The 'Namespace name' field contains 'contosoehub' with '.servicebus.windows.net' suffix. The 'Location' dropdown is set to '(US) East US'. The 'Pricing tier' dropdown is set to 'Standard (20 Consumer groups, 1000 Brokered connections)'. The 'Throughput Units' slider is set to 1. At the bottom, there are buttons for 'Review + create' (highlighted in blue), '< Previous', and 'Next: Features >'.

h. On the **Review + Create** page, review the settings, and select **Create**. Wait for the deployment to complete.

The screenshot shows the 'Create Namespace' step in the Microsoft Azure portal. At the top, there's a green success message: 'Validation succeeded.' Below it, the 'Review + create' tab is selected. The page displays the 'Event Hubs Namespace' configuration with the following details:

Setting	Value
Namespace name	contosoehub
Subscription	Visual Studio Ultimate with MSDN
Resource group	myehubrg
Location	East US
Pricing tier	Standard
Throughput Units	1

Below the main configuration, there are sections for 'Features' and 'Availability Zones', both set to 'Disabled'. At the bottom, there are 'Create', '< Previous', and 'Next >' buttons.

- i. On the Deployment page, select **Go to resource** to navigate to the page for your namespace.

The screenshot shows the 'NoMarketplace | Overview' page in the Microsoft Azure portal. It displays the deployment status:

**Your deployment is complete**

Deployment details:

Detail	Value
Deployment name:	NoMarketplace
Subscription:	Visual Studio Ultimate with MSDN
Resource group:	myehubrg
Start time:	5/4/2020, 10:10:41 AM
Correlation ID:	d0bba908-9d7d-4b77-a026-

Navigation links include 'Deployment details (Download)', 'Next steps', and a prominent red 'Go to resource' button.

- j. Confirm that you see the Event Hubs Namespace page similar to the following example:

contosoehub  
Event Hubs Namespace

Search (Ctrl+ /) <

+ Event Hub Delete Refresh

Resource group (change) : myehubrg Created : Monday, May 4, 2020, 10:10:44 EDT

Status : Active Updated : Monday, May 4, 2020, 10:11:29 EDT

Location : East US Pricing tier : Standard

Subscription (change) : Visual Studio Ultimate with MSDN Throughput Units : 1 unit

Subscription ID : <Your Azure subscription ID> Auto-inflate throughput... : Disabled

Host name : contosoehub.servicebus.windows.net

Tags (change) : Click here to add tags

NAMESPACE CONTENTS 0 EVENT HUBS KAFKA SURFACE ENABLED

Show metrics: Requests Messages Throughput

For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 day

100  
90  
80  
70  
60  
50  
40  
30  
20  
10  
0

9:30 AM 9:45 AM 10 AM UTC-0

Incoming Requests (Sum) contosoehub 0 Successful Requests ... contosoehub 0 Server Errors. (Sum) contosoehub 0 User Errors. (Sum) contosoehub 0 Throttled Requests. ... contosoehub 0

Search to filter items...

Name	Status	Message Retention	Partition Count
No Event Hubs yet.			

### NOTE

Azure Event Hubs provides you with a Kafka endpoint. This endpoint enables your Event Hubs namespace to natively understand [Apache Kafka](#) message protocol and APIs. With this capability, you can communicate with your event hubs as you would with Kafka topics without changing your protocol clients or running your own clusters. Event Hubs supports [Apache Kafka versions 1.0](#) and later. For more information, see [Use Event Hubs from Apache Kafka applications](#).

## Create an event hub

To create an event hub within the namespace, do the following actions:

1. On the Event Hubs Namespace page, select **Event Hubs** in the left menu.
2. At the top of the window, select **+ Event Hub**.

The screenshot shows the 'myehubns - Event Hubs' blade in the Azure portal. On the left, there's a navigation menu with options like Overview, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings (Shared access policies, Geo-Recovery, Firewalls and virtual networks, Properties, Locks, Automation script), and Entities (Event Hubs). The 'Event Hubs' item is highlighted with a red box. In the main content area, there's a search bar and a table with columns: NAME, STATUS, MESSAGE RETENTION, and PARTITION COUNT. A message says 'no Event Hubs yet.' There's also a '+ Event Hub' button and a Refresh button.

3. Type a name for your event hub, then select **Create**.

The screenshot shows the 'Create Event Hub' dialog box. It has fields for Name (set to 'myeventhub'), Partition Count (set to 2), Message Retention (set to 1), and Capture (set to On). At the bottom is a 'Create' button.

Setting	Value
Name	myeventhub
Partition Count	2
Message Retention	1
Capture	On

The **partition count** setting allows you to parallelize consumption across many consumers. For more information, see [Partitions](#).

The **message retention** setting specifies how long the Event Hubs service keeps data. For more information, see [Event retention](#).

4. You can check the status of the event hub creation in alerts. After the event hub is created, you see it in the list of event hubs.

NAME	STATUS	MESSAGE RETENTION	PARTITION COUNT
myeventhub	Active	1	2

## Next steps

In this article, you created a resource group, an Event Hubs namespace, and an event hub. For step-by-step instructions to send events to (or) receive events from an event hub, see these tutorials:

- [.NET Core](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [Go](#)
- [C \(send only\)](#)
- [Apache Storm \(receive only\)](#)

# Quickstart: Create an event hub using Azure CLI

11/26/2021 • 2 minutes to read • [Edit Online](#)

Azure Event Hubs is a Big Data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see [Event Hubs overview](#) and [Event Hubs features](#).

In this quickstart, you create an event hub using Azure CLI.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).  
[Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires version 2.0.4 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Set the subscription context

The following steps are not required if you're running commands in Cloud Shell. If you're running the CLI locally, perform the following steps to sign in to Azure and set your current subscription:

Set the current subscription context. Replace `MyAzureSub` with the name of the Azure subscription you want to use:

```
az account set --subscription MyAzureSub
```

## Create a resource group

A resource group is a logical collection of Azure resources. All resources are deployed and managed in a resource group. Run the following command to create a resource group:

```
# Create a resource group. Specify a name for the resource group.  
az group create --name <resource group name> --location eastus
```

## Create an Event Hubs namespace

An Event Hubs namespace provides a unique scoping container, referenced by its fully qualified domain name, in which you create one or more event hubs. To create a namespace in your resource group, run the following command:

```
# Create an Event Hubs namespace. Specify a name for the Event Hubs namespace.  
az eventhubs namespace create --name <Event Hubs namespace> --resource-group <resource group name> -l  
<region, for example: East US>
```

## Create an event hub

Run the following command to create an event hub:

```
# Create an event hub. Specify a name for the event hub.  
az eventhubs eventhub create --name <event hub name> --resource-group <resource group name> --namespace-name  
<Event Hubs namespace>
```

Congratulations! You have used Azure CLI to create an Event Hubs namespace, and an event hub within that namespace.

## Next steps

In this article, you created a resource group, an Event Hubs namespace, and an event hub. For step-by-step instructions to send events to (or) receive events from an event hub, see the [Send and receive events](#) tutorials:

- [.NET Core](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [Go](#)
- [C \(send only\)](#)
- [Apache Storm \(receive only\)](#)

# Quickstart: Create an event hub using Azure PowerShell

11/26/2021 • 3 minutes to read • [Edit Online](#)

Azure Event Hubs is a Big Data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see [Event Hubs overview](#) and [Event Hubs features](#).

In this quickstart, you create an event hub using Azure PowerShell.

## Prerequisites

### NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

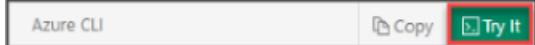
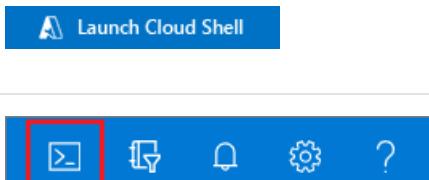
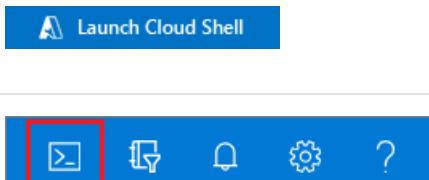
To complete this tutorial, make sure you have:

- Azure subscription. If you don't have one, [create a free account](#) before you begin.
- [Visual Studio 2019](#).
- [.NET Core SDK](#), version 2.0 or later.

## Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the Azure portal.	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.

2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you're using PowerShell locally, you must run the latest version of PowerShell to complete this quickstart. If you need to install or upgrade, see [Install and Configure Azure PowerShell](#).

## Create a resource group

A resource group is a logical collection of Azure resources, and you need a resource group to create an event hub.

The following example creates a resource group in the East US region. Replace `myResourceGroup` with the name of the resource group you want to use:

```
New-AzResourceGroup -Name myResourceGroup -Location eastus
```

## Create an Event Hubs namespace

Once your resource group is made, create an Event Hubs namespace within that resource group. An Event Hubs namespace provides a unique fully-qualified domain name in which you can create your event hub. Replace `namespace_name` with a unique name for your namespace:

```
New-AzEventHubNamespace -ResourceGroupName myResourceGroup -NamespaceName namespace_name -Location eastus
```

## Create an event hub

Now that you have an Event Hubs namespace, create an event hub within that namespace:

Allowed period for `MessageRetentionInDays` is between 1 and 7 days.

```
New-AzEventHub -ResourceGroupName myResourceGroup -NamespaceName namespace_name -EventHubName eventhub_name  
-MessageRetentionInDays 3
```

Congratulations! You have used Azure PowerShell to create an Event Hubs namespace, and an event hub within that namespace.

## Next steps

In this article, you created the Event Hubs namespace, and used sample applications to send and receive events from your event hub. For step-by-step instructions to send events to (or) receive events from an event hub, see the [Send and receive events](#) tutorials:

- [.NET Core](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [Go](#)
- [C \(send only\)](#)
- [Apache Storm \(receive only\)](#)

# Quickstart: Create an event hub by using an ARM template

11/26/2021 • 3 minutes to read • [Edit Online](#)

Azure Event Hubs is a Big Data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see [Event Hubs overview](#) and [Event Hubs features](#). In this quickstart, you create an event hub by using an [Azure Resource Manager template \(ARM template\)](#). You deploy an ARM template to create a namespace of type [Event Hubs](#), with one event hub.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

 [Deploy to Azure](#)

## Prerequisites

If you don't have an Azure subscription, [create a free account](#) before you begin.

## Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        " projectName": {
            "type": "string",
            "metadata": {
                "description": "Specifies a project name that is used to generate the Event Hub name and the Namespace name."
            }
        },
        " location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Specifies the Azure location for all resources."
            }
        },
        " eventHubSku": {
            "type": "string",
            "defaultValue": "Standard",
            "allowedValues": [ "Basic", "Standard" ],
            "metadata": {
                "description": "Specifies the messaging tier for Event Hub Namespace."
            }
        }
    },
    "variables": {
        " eventHubNamespaceName": "[concat(parameters('projectName'), 'ns')]",
        " eventHubName": "[parameters('projectName')]"
    },
    "resources": [
        {
            "type": "Microsoft.EventHub/namespaces",
            "apiVersion": "2018-01-01-preview",
            "name": "[variables('eventHubNamespaceName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "[parameters('eventHubSku')]",
                "tier": "[parameters('eventHubSku')]",
                "capacity": 1
            },
            "properties": {
                "isAutoInflateEnabled": false,
                "maximumThroughputUnits": 0
            }
        },
        {
            "type": "Microsoft.EventHub/namespaces/eventhubs",
            "apiVersion": "2017-04-01",
            "name": "[concat(variables('eventHubNamespaceName'), '/', variables('eventHubName'))]",
            "location": "[parameters('location')]",
            "dependsOn": [
                "[resourceId('Microsoft.EventHub/namespaces', variables('eventHubNamespaceName'))]"
            ],
            "properties": {
                "messageRetentionInDays": 7,
                "partitionCount": 1
            }
        }
    ]
}
```

The resources defined in the template include:

- [Microsoft.EventHub/namespaces](#)

- [Microsoft.EventHub/namespaces/eventhubs](#)

To find more template samples, see [Azure Quickstart Templates](#).

## Deploy the template

To deploy the template:

1. Select **Try it** from the following code block, and then follow the instructions to sign in to the Azure Cloud Shell.

```
$projectName = Read-Host -Prompt "Enter a project name that is used for generating resource names"
.setLocation = Read-Host -Prompt "Enter the location (i.e. centralus)"
$resourceGroupName = "${projectName}rg"
$templateUri = "https://raw.githubusercontent.com/Azure/azure-quickstart-templates/master/quickstarts/microsoft.eventhub/eventhubs-create-namespace-and-eventhub/azuredeploy.json"

New-AzResourceGroup -Name $resourceGroupName -Location $location
New-AzResourceGroupDeployment -ResourceGroupName $resourceGroupName -TemplateUri $templateUri -projectName $projectName

Write-Host "Press [ENTER] to continue ..."
```

It takes a few moments to create an event hub.

2. Select **Copy** to copy the PowerShell script.
3. Right-click the shell console, and then select **Paste**.

## Validate the deployment

To verify the deployment, you can either open the resource group from the [Azure portal](#), or use the following Azure PowerShell script. If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"
$resourceGroupName = "${projectName}rg"
$namespaceName = "${projectName}ns"

Get-AzEventHub -ResourceGroupName $resourceGroupName -Namespace $namespaceName

Write-Host "Press [ENTER] to continue ..."
```

## Clean up resources

When the Azure resources are no longer needed, clean up the resources you deployed by deleting the resource group. If the Cloud Shell is still open, you don't need to copy/run the first line (Read-Host).

```
$projectName = Read-Host -Prompt "Enter the same project name that you used in the last procedure"
$resourceGroupName = "${projectName}rg"

Remove-AzResourceGroup -ResourceGroupName $resourceGroupName

Write-Host "Press [ENTER] to continue ..."
```

## Next steps

In this article, you created an Event Hubs namespace, and an event hub in the namespace. For step-by-step instructions to send events to (or) receive events from an event hub, see the [Send and receive events](#) tutorials:

- [.NET Core](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)
- [Go](#)
- [C \(send only\)](#)
- [Apache Storm \(receive only\)](#)

# Send events to and receive events from Azure Event Hubs - .NET (Azure.Messaging.EventHubs)

11/26/2021 • 8 minutes to read • [Edit Online](#)

This quickstart shows how to send events to and receive events from an event hub using the `Azure.Messaging.EventHubs` .NET library.

## Prerequisites

If you're new to Azure Event Hubs, see [Event Hubs overview](#) before you do this quickstart.

To complete this quickstart, you need the following prerequisites:

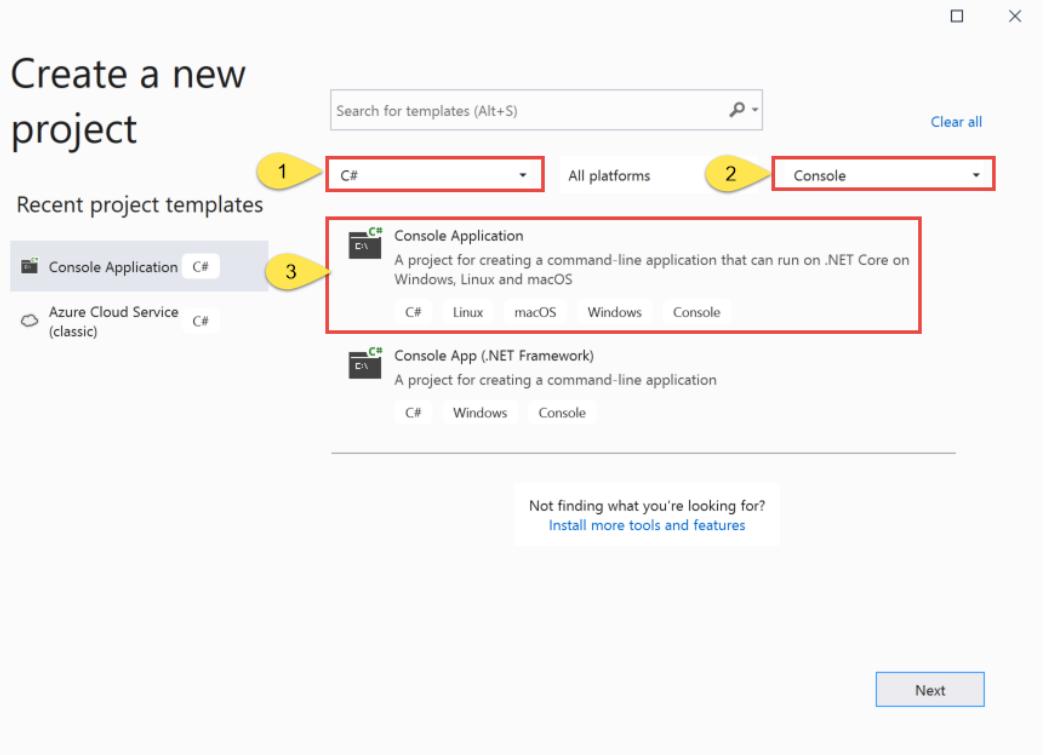
- **Microsoft Azure subscription.** To use Azure services, including Azure Event Hubs, you need a subscription. If you don't have an existing Azure account, you can sign up for a [free trial](#) or use your MSDN subscriber benefits when you [create an account](#).
- **Microsoft Visual Studio 2019.** The Azure Event Hubs client library makes use of new features that were introduced in C# 8.0. You can still use the library with previous C# language versions, but the new syntax won't be available. To make use of the full syntax, it is recommended that you compile with the [.NET Core SDK](#) 3.0 or higher and [language version](#) set to `latest`. If you're using Visual Studio, versions before Visual Studio 2019 aren't compatible with the tools needed to build C# 8.0 projects. Visual Studio 2019, including the free Community edition, can be downloaded [here](#).
- **Create an Event Hubs namespace and an event hub.** The first step is to use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#). Then, get the **connection string for the Event Hubs namespace** by following instructions from the article: [Get connection string](#). You use the connection string later in this quickstart.

## Send events

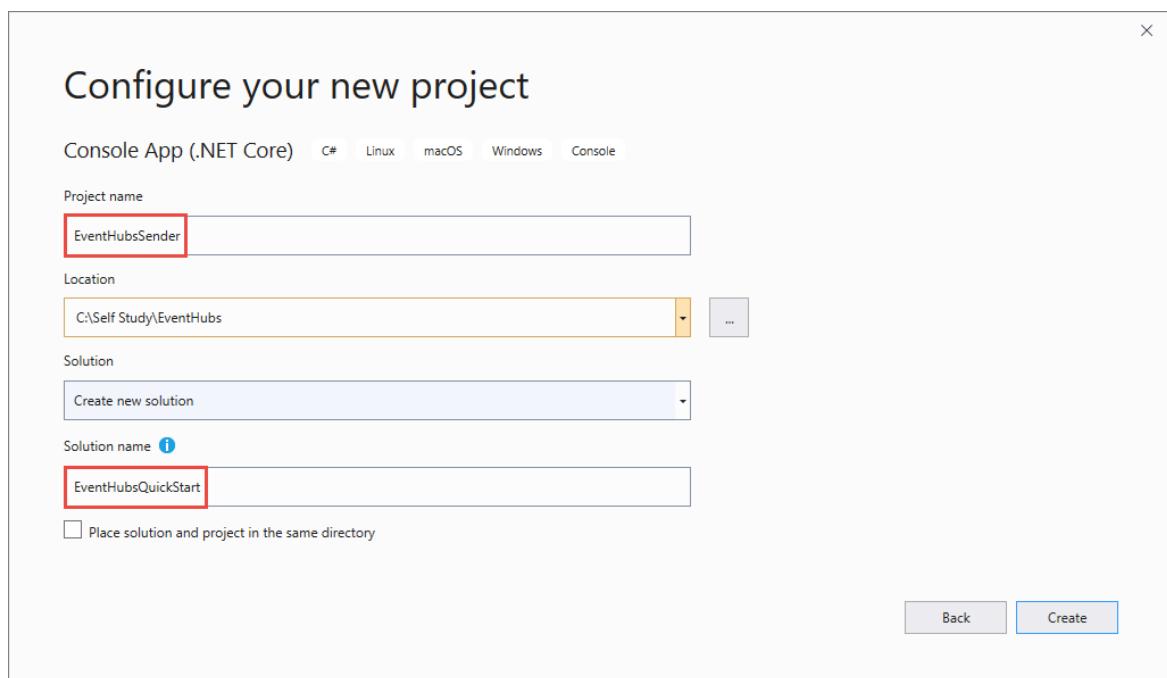
This section shows you how to create a .NET Core console application to send events to an event hub.

### Create a console application

1. Start Visual Studio 2019.
2. Select **Create a new project**.
3. On the **Create a new project** dialog box, do the following steps: If you don't see this dialog box, select **File** on the menu, select **New**, and then select **Project**.
  - a. Select **C#** for the programming language.
  - b. Select **Console** for the type of the application.
  - c. Select **Console Application** from the results list.
  - d. Then, select **Next**.



4. Enter **EventHubsSender** for the project name, **EventHubsQuickStart** for the solution name, and then select **OK** to create the project.



## Add the Event Hubs NuGet package

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.EventHubs NuGet package:

```
Install-Package Azure.Messaging.EventHubs
```

## Write code to send events to the event hub

1. Add the following `using` statements to the top of the `Program.cs` file:

```
using System;
using System.Text;
using System.Threading.Tasks;
using Azure.Messaging.EventHubs;
using Azure.Messaging.EventHubs.Producer;
```

2. Add constants to the `Program` class for the Event Hubs connection string and the event hub name.

```
// connection string to the Event Hubs namespace
private const string connectionString = "<EVENT HUBS NAMESPACE - CONNECTION STRING>";

// name of the event hub
private const string eventHubName = "<EVENT HUB NAME>";

// number of events to be sent to the event hub
private const int numOfEvents = 3;
```

**NOTE**

Replace placeholder values with the connection string to your namespace and the name of the event hub. Make sure that the connection string is the namespace-level connection string.

3. Add the following static property to the `Program` class. See the code comments.

```
// The Event Hubs client types are safe to cache and use as a singleton for the lifetime
// of the application, which is best practice when events are being published or read regularly.
static EventHubProducerClient producerClient;
```

4. Replace the `Main` method with the following `async Main` method. See the code comments for details.

```
static async Task Main()
{
    // Create a producer client that you can use to send events to an event hub
    producerClient = new EventHubProducerClient(connectionString, eventHubName);

    // Create a batch of events
    using EventDataBatch eventBatch = await producerClient.CreateBatchAsync();

    for (int i = 1; i <= numOfEvents; i++)
    {
        if (!eventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes($"Event {i}"))))
        {
            // if it is too large for the batch
            throw new Exception($"Event {i} is too large for the batch and cannot be sent.");
        }
    }

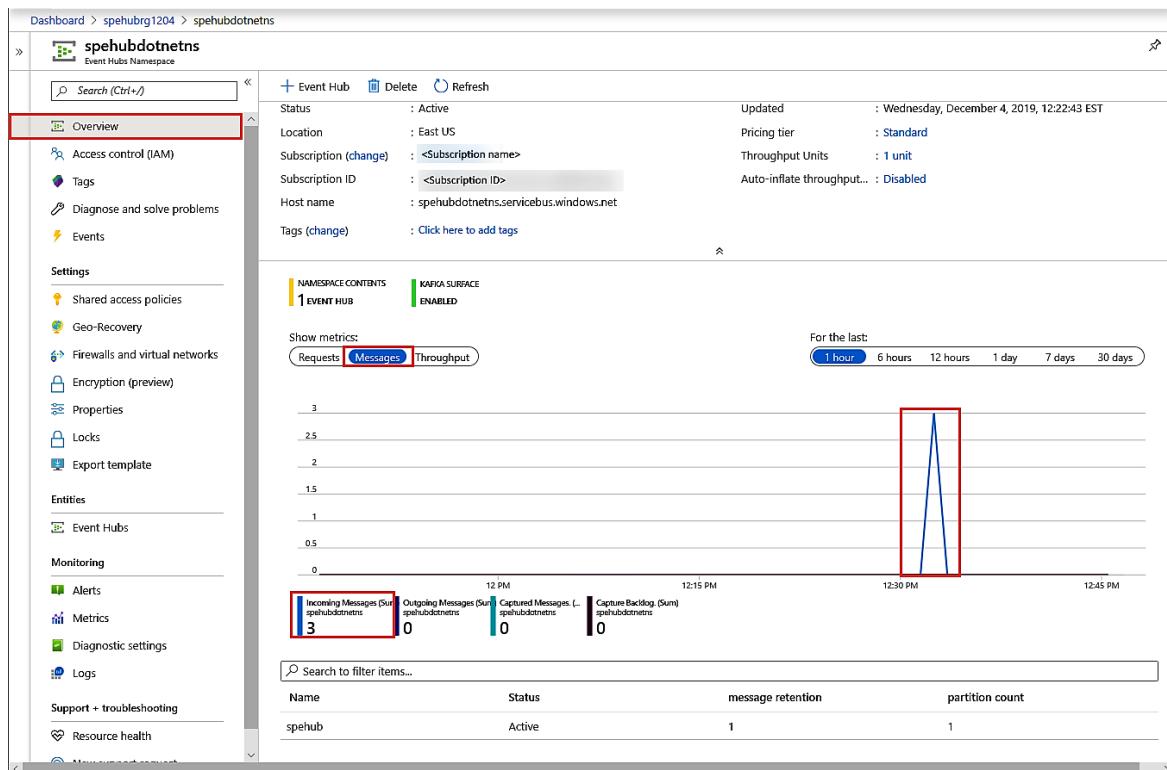
    try
    {
        // Use the producer client to send the batch of events to the event hub
        await producerClient.SendAsync(eventBatch);
        Console.WriteLine($"A batch of {numOfEvents} events has been published.");
    }
    finally
    {
        await producerClient.DisposeAsync();
    }
}
```

5. Build the project, and ensure that there are no errors.

6. Run the program and wait for the confirmation message.

A batch of 3 events has been published.

7. In the Azure portal, you can verify that the event hub has received the events. Switch to **Messages** view in the **Metrics** section. Refresh the page to update the chart. It may take a few seconds for it to show that the messages have been received.



#### NOTE

For the complete source code with more informational comments, see [this file on the GitHub](#)

## Receive events

This section shows how to write a .NET Core console application that receives events from an event hub using an event processor. The event processor simplifies receiving events from event hubs by managing persistent checkpoints and parallel receptions from those event hubs. An event processor is associated with a specific event Hub and a consumer group. It receives events from multiple partitions in the event hub, passing them to a handler delegate for processing using code that you provide.

## WARNING

If you run this code on **Azure Stack Hub**, you will experience runtime errors unless you target a specific Storage API version. That's because the Event Hubs SDK uses the latest available Azure Storage API available in Azure that may not be available on your Azure Stack Hub platform. Azure Stack Hub may support a different version of Storage Blob SDK than those typically available on Azure. If you are using Azure Blob Storage as a checkpoint store, check the [supported Azure Storage API version for your Azure Stack Hub build](#) and target that version in your code.

For example, If you are running on Azure Stack Hub version 2005, the highest available version for the Storage service is version 2019-02-02. By default, the Event Hubs SDK client library uses the highest available version on Azure (2019-07-07 at the time of the release of the SDK). In this case, besides following steps in this section, you will also need to add code to target the Storage service API version 2019-02-02. For an example on how to target a specific Storage API version, see [this sample on GitHub](#).

## Create an Azure Storage and a blob container

In this quickstart, you use Azure Storage as the checkpoint store. Follow these steps to create an Azure Storage account.

1. [Create an Azure Storage account](#)
2. [Create a blob container](#)
3. [Get the connection string to the storage account](#)

Note down the connection string and the container name. You'll use them in the receive code.

## Create a project for the receiver

1. In the Solution Explorer window, right-click the **EventHubQuickStart** solution, point to **Add**, and select **New Project**.
2. Select **Console application**, and select **Next**.
3. Enter **EventHubsReceiver** for the **Project name**, and select **Create**.
4. In the **Solution Explorer** window, right-click **EventHubsReceiver**, and select **Set as a Startup Project**.

## Add the Event Hubs NuGet package

1. Select **Tools > NuGet Package Manager > Package Manager Console** from the menu.
2. In the **Package Manager Console** window, confirm that **EventHubsReceiver** is selected for the **Default project**. If not, use the drop-down list to select **EventHubsReceiver**.
3. Run the following command to install the **Azure.Messaging.EventHubs** NuGet package:

```
Install-Package Azure.Messaging.EventHubs
```

4. Run the following command to install the **Azure.Messaging.EventHubs.Processor** NuGet package:

```
Install-Package Azure.Messaging.EventHubs.Processor
```

## Update the Main method

1. Add the following `using` statements at the top of the **Program.cs** file.

```
using System;
using System.Text;
using System.Threading.Tasks;
using Azure.Storage.Blobs;
using Azure.Messaging.EventHubs;
using Azure.Messaging.EventHubs.Consumer;
using Azure.Messaging.EventHubs.Processor;
```

2. Add constants to the `Program` class for the Event Hubs connection string and the event hub name. Replace placeholders in brackets with the proper values that you got when creating the event hub. Replace placeholders in brackets with the proper values that you got when creating the event hub and the storage account (access keys - primary connection string). Make sure that the `{Event Hubs namespace connection string}` is the namespace-level connection string, and not the event hub string.

```
private const string ehubNamespaceConnectionString = "<EVENT HUBS NAMESPACE - CONNECTION STRING>";
private const string eventHubName = "<EVENT HUB NAME>";
private const string blobStorageConnectionString = "<AZURE STORAGE CONNECTION STRING>";
private const string blobContainerName = "<BLOB CONTAINER NAME>";
```

3. Add the following static properties to the `Program` class.

```
static BlobContainerClient storageClient;

// The Event Hubs client types are safe to cache and use as a singleton for the lifetime
// of the application, which is best practice when events are being published or read regularly.
static EventProcessorClient processor;
```

4. Replace the `Main` method with the following `async Main` method. See the code comments for details.

```
static async Task Main()
{
    // Read from the default consumer group: $Default
    string consumerGroup = EventHubConsumerClient.DefaultConsumerGroupName;

    // Create a blob container client that the event processor will use
    storageClient = new BlobContainerClient(blobStorageConnectionString, blobContainerName);

    // Create an event processor client to process events in the event hub
    processor = new EventProcessorClient(storageClient, consumerGroup,
    ehubNamespaceConnectionString, eventHubName);

    // Register handlers for processing events and handling errors
    processor.ProcessEventAsync += ProcessEventHandler;
    processor.ProcessErrorAsync += ProcessErrorHandler;

    // Start the processing
    await processor.StartProcessingAsync();

    // Wait for 30 seconds for the events to be processed
    await Task.Delay(TimeSpan.FromSeconds(30));

    // Stop the processing
    await processor.StopProcessingAsync();
}
```

5. Now, add the following event and error handler methods to the class.

```
static async Task ProcessEventHandler(ProcessEventArgs eventArgs)
{
    // Write the body of the event to the console window
    Console.WriteLine("\tReceived event: {0}",
        Encoding.UTF8.GetString(eventArgs.Data.Body.ToArray()));

    // Update checkpoint in the blob storage so that the app receives only new events the next
    time it's run
    await eventArgs.UpdateCheckpointAsync(eventArgs.CancellationToken);
}

static Task ProcessErrorHandler(ProcessErrorEventArgs eventArgs)
{
    // Write details about the error to the console window
    Console.WriteLine($"\\tPartition '{eventArgs.PartitionId}': an unhandled exception was
encountered. This was not expected to happen.");
    Console.WriteLine(eventArgs.Exception.Message);
    return Task.CompletedTask;
}
```

6. Build the project, and ensure that there are no errors.

**NOTE**

For the complete source code with more informational comments, see [this file on the GitHub](#).

7. Run the receiver application.
8. You should see a message that the events have been received.

```
Received event: Event 1
Received event: Event 2
Received event: Event 3
```

These events are the three events you sent to the event hub earlier by running the sender program.

## Next steps

Check out the samples on GitHub.

- [Event Hubs samples on GitHub](#)
- [Event processor samples on GitHub](#)
- [Azure role-based access control \(Azure RBAC\) sample](#)

# Use Java to send events to or receive events from Azure Event Hubs (azure-messaging-eventhubs)

11/26/2021 • 7 minutes to read • [Edit Online](#)

This quickstart shows how to send events to and receive events from an event hub using the `azure-messaging-eventhubs` Java package.

## IMPORTANT

This quickstart uses the new `azure-messaging-eventhubs` package. For a quickstart that uses the old `azure-eventhubs` and `azure-eventhubs-eph` packages, see [Send and receive events using azure-eventhubs and azure-eventhubs-eph](#).

## Prerequisites

If you're new to Azure Event Hubs, see [Event Hubs overview](#) before you do this quickstart.

To complete this quickstart, you need the following prerequisites:

- Microsoft Azure subscription. To use Azure services, including Azure Event Hubs, you need a subscription. If you don't have an existing Azure account, you can [sign up for a free trial](#) or use your MSDN subscriber benefits when you [create an account](#).
- A Java development environment. This quickstart uses [Eclipse](#). Java Development Kit (JDK) with version 8 or above is required.
- Create an Event Hubs namespace and an event hub. The first step is to use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#). Then, get the [connection string for the Event Hubs namespace](#) by following instructions from the article: [Get connection string](#). You use the connection string later in this quickstart.

## Send events

This section shows you how to create a Java application to send events an event hub.

### Add reference to Azure Event Hubs library

The Java client library for Event Hubs is available in the [Maven Central Repository](#). You can reference this library using the following dependency declaration inside your Maven project file:

```
<dependency>
    <groupId>com.azure</groupId>
    <artifactId>azure-messaging-eventhubs</artifactId>
    <version>5.7.0</version>
</dependency>
```

## NOTE

Update the `version` to the latest version published to the [Maven repository](#).

## Write code to send messages to the event hub

For the following sample, first create a new Maven project for a console/shell application in your favorite Java development environment. Add a class named `Sender`, and add the following code to the class:

### IMPORTANT

Update `<Event Hubs namespace connection string>` with the connection string to your Event Hubs namespace.  
Update `<Event hub name>` with the name of your event hub in the namespace.

```
import com.azure.messaging.eventhubs.*;
import java.util.Arrays;
import java.util.List;

public class Sender {
    private static final String connectionString = "<Event Hubs namespace connection string>";
    private static final String eventHubName = "<Event hub name>";

    public static void main(String[] args) {
        publishEvents();
    }
}
```

### Add code to publish events to the event hub

Add a method named `publishEvents` to the `Sender` class:

```
/**
 * Code sample for publishing events.
 * @throws IllegalArgumentException if the EventData is bigger than the max batch size.
 */
public static void publishEvents() {
    // create a producer client
    EventHubProducerClient producer = new EventHubClientBuilder()
        .connectionString(connectionString, eventHubName)
        .buildProducerClient();

    // sample events in an array
    List<EventData> allEvents = Arrays.asList(new EventData("Foo"), new EventData("Bar"));

    // create a batch
    EventDataBatch eventDataBatch = producer.createBatch();

    for (EventData eventData : allEvents) {
        // try to add the event from the array to the batch
        if (!eventDataBatch.tryAdd(eventData)) {
            // if the batch is full, send it and then create a new batch
            producer.send(eventDataBatch);
            eventDataBatch = producer.createBatch();

            // Try to add that event that couldn't fit before.
            if (!eventDataBatch.tryAdd(eventData)) {
                throw new IllegalArgumentException("Event is too large for an empty batch. Max size: "
                    + eventDataBatch.getMaxSizeInBytes());
            }
        }
    }
    // send the last batch of remaining events
    if (eventDataBatch.getCount() > 0) {
        producer.send(eventDataBatch);
    }
    producer.close();
}
```

Build the program, and ensure that there are no errors. You'll run this program after you run the receiver program.

## Receive events

The code in this tutorial is based on the [EventProcessorClient sample on GitHub](#), which you can examine to see the full working application.

### WARNING

If you run this code on Azure Stack Hub, you will experience runtime errors unless you target a specific Storage API version. That's because the Event Hubs SDK uses the latest available Azure Storage API available in Azure that may not be available on your Azure Stack Hub platform. Azure Stack Hub may support a different version of Azure Blob Storage SDK than those typically available on Azure. If you are using Azure Blob Storage as a checkpoint store, check the supported Azure Storage API version for your Azure Stack Hub build and target that version in your code.

For example, If you are running on Azure Stack Hub version 2005, the highest available version for the Storage service is version 2019-02-02. By default, the Event Hubs SDK client library uses the highest available version on Azure (2019-07-07 at the time of the release of the SDK). In this case, besides following steps in this section, you will also need to add code to target the Storage service API version 2019-02-02. For an example on how to target a specific Storage API version, see [this sample on GitHub](#).

## Create an Azure Storage and a blob container

In this quickstart, you use Azure Storage (specifically, Blob Storage) as the checkpoint store. Checkpointing is a process by which an event processor marks or commits the position of the last successfully processed event within a partition. Marking a checkpoint is typically done within the function that processes the events. To learn more about checkpointing, see [Event processor](#).

Follow these steps to create an Azure Storage account.

1. [Create an Azure Storage account](#)
2. [Create a blob container](#)
3. [Get the connection string to the storage account](#)

Note down the **connection string** and the **container name**. You'll use them in the receive code.

## Add Event Hubs libraries to your Java project

Add the following dependencies in the pom.xml file.

- [azure-messaging-eventhubs](#)
- [azure-messaging-eventhubs-checkpointstore-blob](#)

```
<dependencies>
    <dependency>
        <groupId>com.azure</groupId>
        <artifactId>azure-messaging-eventhubs</artifactId>
        <version>5.7.0</version>
    </dependency>
    <dependency>
        <groupId>com.azure</groupId>
        <artifactId>azure-messaging-eventhubs-checkpointstore-blob</artifactId>
        <version>1.6.0</version>
    </dependency>
</dependencies>
```

1. Add the following **import** statements at the top of the Java file.

```
import com.azure.messaging.eventhubs.*;
import com.azure.messaging.eventhubs.checkpointstore.blob.BlobCheckpointStore;
import com.azure.messaging.eventhubs.models.*;
import com.azure.storage.blob.*;
import java.util.function.Consumer;
```

2. Create a class named `Receiver`, and add the following string variables to the class. Replace the placeholders with the correct values.

#### IMPORTANT

Replace the placeholders with the correct values.

- `<Event Hubs namespace connection string>` with the connection string to your Event Hubs namespace.  
Update
- `<Event hub name>` with the name of your event hub in the namespace.
- `<Storage connection string>` with the connection string to your Azure storage account.
- `<Storage container name>` with the name of your container in your Azure Blob storage.

```
private static final String connectionString = "<Event Hubs namespace connection string>";
private static final String eventHubName = "<Event hub name>";
private static final String storageConnectionString = "<Storage connection string>";
private static final String storageContainerName = "<Storage container name>";
```

3. Add the following `main` method to the class.

```
public static void main(String[] args) throws Exception {
    // Create a blob container client that you use later to build an event processor client to
    // receive and process events
    BlobContainerAsyncClient blobContainerAsyncClient = new BlobContainerClientBuilder()
        .connectionString(storageConnectionString)
        .containerName(storageContainerName)
        .buildAsyncClient();

    // Create a builder object that you will use later to build an event processor client to receive
    // and process events and errors.
    EventProcessorClientBuilder eventProcessorClientBuilder = new EventProcessorClientBuilder()
        .connectionString(connectionString, eventHubName)
        .consumerGroup(EventHubClientBuilder.DEFAULT_CONSUMER_GROUP_NAME)
        .processEvent(PARTITION_PROCESSOR)
        .processError(ERROR_HANDLER)
        .checkpointStore(new BlobCheckpointStore(blobContainerAsyncClient));

    // Use the builder object to create an event processor client
    EventProcessorClient eventProcessorClient =
        eventProcessorClientBuilder.buildEventProcessorClient();

    System.out.println("Starting event processor");
    eventProcessorClient.start();

    System.out.println("Press enter to stop.");
    System.in.read();

    System.out.println("Stopping event processor");
    eventProcessorClient.stop();
    System.out.println("Event processor stopped.");

    System.out.println("Exiting process");
}
```

4. Add the two helper methods (`PARTITION_PROCESSOR` and `ERROR_HANDLER`) that process events and errors to the `Receiver` class.

```
public static final Consumer<EventContext> PARTITION_PROCESSOR = eventContext -> {
    PartitionContext partitionContext = eventContext.getPartitionContext();
    EventData eventData = eventContext.getEventData();

    System.out.printf("Processing event from partition %s with sequence number %d with body: %s%n",
        partitionContext.getPartitionId(), eventData.getSequenceNumber(),
        eventData.getBodyAsString());

    // Every 10 events received, it will update the checkpoint stored in Azure Blob Storage.
    if (eventData.getSequenceNumber() % 10 == 0) {
        eventContext.updateCheckpoint();
    }
};

public static final Consumer<ErrorContext> ERROR_HANDLER = errorContext -> {
    System.out.printf("Error occurred in partition processor for partition %s, %s.%n",
        errorContext.getPartitionContext().getPartitionId(),
        errorContext.getThrowable());
};
```

5. Build the program, and ensure that there are no errors.

## Run the applications

1. Run the `Receiver` application first.
2. Then, run the `Sender` application.
3. In the `Receiver` application window, confirm that you see the events that were published by the `Sender` application.

```
Starting event processor
Press enter to stop.
Processing event from partition 0 with sequence number 331 with body: Foo
Processing event from partition 0 with sequence number 332 with body: Bar
```

4. Press **ENTER** in the receiver application window to stop the application.

```
Starting event processor
Press enter to stop.
Processing event from partition 0 with sequence number 331 with body: Foo
Processing event from partition 0 with sequence number 332 with body: Bar

Stopping event processor
Event processor stopped.
Exiting process
```

## Next steps

See the following samples on GitHub:

- [azure-messaging-eventhubs samples](#)
- [azure-messaging-eventhubs-checkpointstore-blob samples](#).

# Use Java to send events to or receive events from Azure Event Hubs (azure-eventhubs)

11/26/2021 • 11 minutes to read • [Edit Online](#)

This quickstart shows how to send events to and receive events from an event hub using the `azure-eventhubs` Java package.

## WARNING

This quickstart uses the old `azure-eventhubs` and `azure-eventhubs-eph` packages. For a quickstart that uses the latest `azure-messaging-eventhubs` package, see [Send and receive events using azure-messaging-eventhubs](#). To move your application from using the old package to new one, see the [Guide to migrate from azure-eventhubs to azure-messaging-eventhubs](#).

## Prerequisites

If you are new to Azure Event Hubs, see [Event Hubs overview](#) before you do this quickstart.

To complete this quickstart, you need the following prerequisites:

- **Microsoft Azure subscription.** To use Azure services, including Azure Event Hubs, you need a subscription. If you don't have an existing Azure account, you can sign up for a [free trial](#) or use your MSDN subscriber benefits when you [create an account](#).
- A Java development environment. This quickstart uses [Eclipse](#).
- **Create an Event Hubs namespace and an event hub.** The first step is to use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#). Then, get the value of access key for the event hub by following instructions from the article: [Get connection string](#). You use the access key in the code you write later in this quickstart. The default key name is: `RootManageSharedAccessKey`.

## Send events

This section shows you how to create a Java application to send events an event hub.

## NOTE

You can download this quickstart as a sample from the [GitHub](#), replace `EventHubConnectionString` and `EventHubName` strings with your event hub values, and run it. Alternatively, you can follow the steps in this quickstart to create your own.

## Add reference to Azure Event Hubs library

The Java client library for Event Hubs is available for use in Maven projects from the [Maven Central Repository](#). You can reference this library using the following dependency declaration inside your Maven project file:

```
<dependency>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>azure-eventhubs</artifactId>
    <version>2.2.0</version>
</dependency>
```

For different types of build environments, you can explicitly obtain the latest released JAR files from the [Maven Central Repository](#).

For a simple event publisher, import the *com.microsoft.azure.eventhubs* package for the Event Hubs client classes and the *com.microsoft.azure.servicebus* package for utility classes such as common exceptions that are shared with the Azure Service Bus messaging client.

### Write code to send messages to the event hub

For the following sample, first create a new Maven project for a console/shell application in your favorite Java development environment. Add a class named `SimpleSend`, and add the following code to the class:

```
import com.google.gson.Gson;
import com.google.gson.GsonBuilder;
import com.microsoft.azure.eventhubs.ConnectionStringBuilder;
import com.microsoft.azure.eventhubs.EventData;
import com.microsoft.azure.eventhubs.EventHubClient;
import com.microsoft.azure.eventhubs.EventHubException;

import java.io.IOException;
import java.nio.charset.Charset;
import java.time.Instant;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;

public class SimpleSend {

    public static void main(String[] args)
        throws EventHubException, ExecutionException, InterruptedException, IOException {

    }
}
```

### Construct connection string

Use the `ConnectionStringBuilder` class to construct a connection string value to pass to the Event Hubs client instance. Replace the placeholders with the values you obtained when you created the namespace and event hub:

```
final ConnectionStringBuilder connStr = new ConnectionStringBuilder()
    .setNamespaceName("<EVENTHUB NAMESPACE")
    .setEventHubName("EVENT HUB")
    .setSasKeyName("RootManageSharedAccessKey")
    .setSasKey("SHARED ACCESS KEY");
```

### Write code to send events

Create a singular event by transforming a string into its UTF-8 byte encoding. Then, create a new Event Hubs client instance from the connection string and send the message:

```

final Gson gson = new GsonBuilder().create();

// The Executor handles all asynchronous tasks and this is passed to the EventHubClient instance.
// This enables the user to segregate their thread pool based on the work load.
// This pool can then be shared across multiple EventHubClient instances.
// The following sample uses a single thread executor, as there is only one EventHubClient instance,
// handling different flavors of ingestion to Event Hubs here.
final ScheduledExecutorService executorService = Executors.newScheduledThreadPool(4);

// Each EventHubClient instance spins up a new TCP/TLS connection, which is expensive.
// It is always a best practice to reuse these instances. The following sample shows this.
final EventHubClient ehClient = EventHubClient.createSync(connStr.toString(), executorService);

try {
    for (int i = 0; i < 10; i++) {

        String payload = "Message " + Integer.toString(i);
        byte[] payloadBytes = gson.toJson(payload).getBytes(Charset.defaultCharset());
        EventData sendEvent = EventData.create(payloadBytes);

        // Send - not tied to any partition
        // Event Hubs service will round-robin the events across all Event Hubs partitions.
        // This is the recommended & most reliable way to send to Event Hubs.
        ehClient.sendSync(sendEvent);
    }

    System.out.println(Instant.now() + ": Send Complete...");
    System.out.println("Press Enter to stop.");
    System.in.read();
} finally {
    ehClient.closeSync();
    executorService.shutdown();
}

```

Build and run the program, and ensure that there are no errors.

Congratulations! You have now sent messages to an event hub.

### **Appendix: How messages are routed to EventHub partitions**

Before messages are retrieved by consumers, they have to be published to the partitions first by the publishers. When messages are published to event hub synchronously using the `sendSync()` method on the `com.microsoft.azure.eventhubs.EventHubClient` object, the message could be sent to a specific partition or distributed to all available partitions in a round-robin manner depending on whether the partition key is specified or not.

When a string representing the partition key is specified, the key will be hashed to determine which partition to send the event to.

When the partition key is not set, then messages will round-robin to all available partitions

```
// Serialize the event into bytes
byte[] payloadBytes = gson.toJson(messagePayload).getBytes(Charset.defaultCharset());

// Use the bytes to construct an {@link EventData} object
EventData sendEvent = EventData.create(payloadBytes);

// Transmits the event to event hub without a partition key
// If a partition key is not set, then we will round-robin to all topic partitions
eventHubClient.sendSync(sendEvent);

// the partitionKey will be hash'ed to determine the partitionId to send the eventData to.
eventHubClient.sendSync(sendEvent, partitionKey);

// close the client at the end of your program
eventHubClient.closeSync();
```

## Receive events

The code in this tutorial is based on the [EventProcessorSample code on GitHub](#), which you can examine to see the full working application.

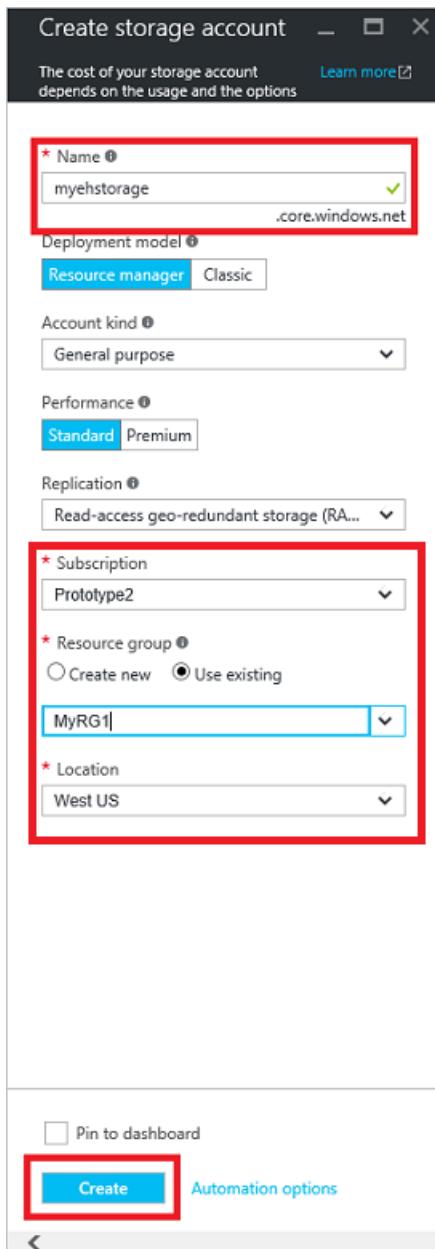
### Receive messages with EventProcessorHost in Java

**EventProcessorHost** is a Java class that simplifies receiving events from Event Hubs by managing persistent checkpoints and parallel receives from those Event Hubs. Using EventProcessorHost, you can split events across multiple receivers, even when hosted in different nodes. This example shows how to use EventProcessorHost for a single receiver.

### Create a storage account

To use EventProcessorHost, you must have an [Azure Storage account][Azure Storage account]:

1. Sign in to the [Azure portal](#), and select **Create a resource** on the left-hand side of the screen.
2. Select **Storage**, then select **Storage account**. In the **Create storage account** window, type a name for the storage account. Complete the rest of the fields, select your desired region, and then select **Create**.



3. Select the newly created storage account, and then select Access Keys:

The screenshot shows the 'Access keys' section of the Azure Storage account settings. The 'key1' row is selected, and its value 'key1 displayed here' is highlighted with a red box. A copy icon (a clipboard with a pencil) next to the value is also highlighted with a red box.

NAME	KEY
key1	key1 displayed here
key2	key2 displayed here

Copy the key1 value to a temporary location. You use it later in this tutorial.

### Create a Java project using the EventProcessor Host

The Java client library for Event Hubs is available for use in Maven projects from the [Maven Central Repository](#), and can be referenced using the following dependency declaration inside your Maven project file:

```
<dependency>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>azure-eventhubs</artifactId>
    <version>2.2.0</version>
</dependency>
<dependency>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>azure-eventhubs-eph</artifactId>
    <version>2.4.0</version>
</dependency>
```

For different types of build environments, you can explicitly obtain the latest released JAR files from the [Maven Central Repository](#).

1. For the following sample, first create a new Maven project for a console/shell application in your favorite Java development environment. The class is called `ErrorNotificationHandler`.

```

import java.util.function.Consumer;
import com.microsoft.azure.eventprocessorhost.ExceptionReceivedEventArgs;

public class ErrorNotificationHandler implements Consumer<ExceptionReceivedEventArgs>
{
    @Override
    public void accept(ExceptionReceivedEventArgs t)
    {
        System.out.println("SAMPLE: Host " + t.getHostname() + " received general error notification
during " + t.getAction() + ":" + t.getException().toString());
    }
}

```

2. Use the following code to create a new class called `EventProcessorSample`. Replace the placeholders with the values used when you created the event hub and storage account:

```

package com.microsoft.azure.eventhubs.samples.eventprocessorsample;

import com.microsoft.azure.eventhubs.ConnectionStringBuilder;
import com.microsoft.azure.eventhubs.EventData;
import com.microsoft.azure.eventprocessorhost.CloseReason;
import com.microsoft.azure.eventprocessorhost.EventProcessorHost;
import com.microsoft.azure.eventprocessorhost.EventProcessorOptions;
import com.microsoft.azure.eventprocessorhost.ExceptionReceivedEventArgs;
import com.microsoft.azure.eventprocessorhost.IEventProcessor;
import com.microsoft.azure.eventprocessorhost.PartitionContext;

import java.util.concurrent.ExecutionException;
import java.util.function.Consumer;

public class EventProcessorSample
{
    public static void main(String args[]) throws InterruptedException, ExecutionException
    {
        String consumerGroupName = "$Default";
        String namespaceName = "----NamespaceName----";
        String eventHubName = "----EventHubName----";
        String sasKeyName = "----SharedAccessSignatureKeyName----";
        String sasKey = "----SharedAccessSignatureKey----";
        String storageConnectionString = "----AzureStorageConnectionString---";
        String storageContainerName = "----StorageContainerName----";
        String hostNamePrefix = "----HostNamePrefix----";

        ConnectionStringBuilder eventHubConnectionString = new ConnectionStringBuilder()
            .setNamespaceName(namespaceName)
            .setEventHubName(eventHubName)
            .setSasKeyName(sasKeyName)
            .setSasKey(sasKey);

        EventProcessorHost host = new EventProcessorHost(
            EventProcessorHost.createHostName(hostNamePrefix),
            eventHubName,
            consumerGroupName,
            eventHubConnectionString.toString(),
            storageConnectionString,
            storageContainerName);

        System.out.println("Registering host named " + host.getHostName());
        EventProcessorOptions options = new EventProcessorOptions();
        options.setExceptionNotification(new ErrorNotificationHandler());

        host.registerEventProcessor(EventProcessor.class, options)
            .whenComplete((unused, e) ->
        {
            if (e != null)
            {

```

```

        System.out.println("Failure while registering: " + e.toString());
        if (e.getCause() != null)
        {
            System.out.println("Inner exception: " + e.getCause().toString());
        }
    }
})
.thenAccept((unused) ->
{
    System.out.println("Press enter to stop.");
    try
    {
        System.in.read();
    }
    catch (Exception e)
    {
        System.out.println("Keyboard read failed: " + e.toString());
    }
})
.thenCompose((unused) ->
{
    return host.unregisterEventProcessor();
})
.exceptionally((e) ->
{
    System.out.println("Failure while unregistering: " + e.toString());
    if (e.getCause() != null)
    {
        System.out.println("Inner exception: " + e.getCause().toString());
    }
    return null;
})
.get(); // Wait for everything to finish before exiting main!

System.out.println("End of sample");
}
}
}

```

3. Create one more class called `EventProcessor`, using the following code:

```

public static class EventProcessor implements IEventProcessor
{
    private int checkpointBatchingCount = 0;

    // OnOpen is called when a new event processor instance is created by the host.
    @Override
    public void onOpen(PartitionContext context) throws Exception
    {
        System.out.println("SAMPLE: Partition " + context.getPartitionId() + " is opening");
    }

    // OnClose is called when an event processor instance is being shut down.
    @Override
    public void onClose(PartitionContext context, CloseReason reason) throws Exception
    {
        System.out.println("SAMPLE: Partition " + context.getPartitionId() + " is closing for reason
" + reason.toString());
    }

    // onError is called when an error occurs in EventProcessorHost code that is tied to this
partition, such as a receiver failure.
    @Override
    public void onError(PartitionContext context, Throwable error)
    {
        System.out.println("SAMPLE: Partition " + context.getPartitionId() + " onError: " +
error.toString());
    }
}

```

```

        }

        // onEvents is called when events are received on this partition of the Event Hub.
        @Override
        public void onEvents(PartitionContext context, Iterable<EventData> events) throws Exception
        {
            System.out.println("SAMPLE: Partition " + context.getPartitionId() + " got event batch");
            int eventCount = 0;
            for (EventData data : events)
            {
                try
                {
                    System.out.println("SAMPLE (" + context.getPartitionId() + "," +
data.getSystemProperties().getOffset() + "," +
data.getSystemProperties().getSequenceNumber() + "): " + new
String(data.getBytes(), "UTF8"));
                    eventCount++;

                    // Checkpointing persists the current position in the event stream for this partition
                    // and means that the next
                    // time any host opens an event processor on this event hub+consumer group+partition
                    // combination, it will start
                    // receiving at the event after this one.
                    this.checkpointBatchingCount++;
                    if ((checkpointBatchingCount % 5) == 0)
                    {
                        System.out.println("SAMPLE: Partition " + context.getPartitionId() + "
checkpointing at " +
data.getSystemProperties().getOffset() + "," +
data.getSystemProperties().getSequenceNumber());
                        // Checkpoints are created asynchronously. It is important to wait for the result
                        // of checkpointing
                        // before exiting onEvents or before creating the next checkpoint, to detect
                        // errors and to ensure proper ordering.
                        context.checkpoint(data).get();
                    }
                }
                catch (Exception e)
                {
                    System.out.println("Processing failed for an event: " + e.toString());
                }
            }
            System.out.println("SAMPLE: Partition " + context.getPartitionId() + " batch size was " +
eventCount + " for host " + context.getOwner());
        }
    }
}

```

This tutorial uses a single instance of `EventProcessorHost`. To increase throughput, we recommend that you run multiple instances of `EventProcessorHost`, preferably on separate machines. It provides redundancy as well. In those cases, the various instances automatically coordinate with each other in order to load balance the received events. If you want multiple receivers to each process *all* the events, you must use the `ConsumerGroup` concept. When receiving events from different machines, it might be useful to specify names for `EventProcessorHost` instances based on the machines (or roles) in which they are deployed.

## Publishing Messages to EventHub

Before messages are retrieved by consumers, they have to be published to the partitions first by the publishers. It is worth noting that when messages are published to event hub synchronously using the `sendSync()` method on the `com.microsoft.azure.eventhubs.EventHubClient` object, the message could be sent to a specific partition or distributed to all available partitions in a round-robin manner depending on whether the partition key is specified or not.

When a string representing the partition key is specified, the key is hashed to determine which partition to send the event to.

When the partition key is not set, then messages are round-robin'd to all available partitions

```
// Serialize the event into bytes
byte[] payloadBytes = gson.toJson(messagePayload).getBytes(Charset.defaultCharset());

// Use the bytes to construct an {@link EventData} object
EventData sendEvent = EventData.create(payloadBytes);

// Transmits the event to event hub without a partition key
// If a partition key is not set, then we will round-robin to all topic partitions
eventHubClient.sendSync(sendEvent);

// the partitionKey will be hash'ed to determine the partitionId to send the eventData to.
eventHubClient.sendSync(sendEvent, partitionKey);
```

## Implementing a Custom CheckpointManager for EventProcessorHost (EPH)

The API provides a mechanism to implement your custom checkpoint manager for scenarios where the default implementation is not compatible with your use case.

The default checkpoint manager uses blob storage but if you override the checkpoint manager used by EPH with your own implementation, you can use any store you want to back your checkpoint manager implementation.

Create a class that implements the interface com.microsoft.azure.eventprocessorhost.ICheckpointManager

Use your custom implementation of the checkpoint manager  
(com.microsoft.azure.eventprocessorhost.ICheckpointManager)

Within your implementation, you can override the default checkpointing mechanism and implement our own checkpoints based on your own data store (like SQL Server, CosmosDB, and Azure Cache for Redis). We recommend that the store used to back your checkpoint manager implementation is accessible to all EPH instances that are processing events for the consumer group.

You can use any datastore that is available in your environment.

The com.microsoft.azure.eventprocessorhost.EventProcessorHost class provides you with two constructors that allow you to override the checkpoint manager for your EventProcessorHost.

## Next steps

Read the following articles:

- [EventProcessorHost](#)
- [Features and terminology in Azure Event Hubs](#)
- [Event Hubs FAQ](#)

# Send events to or receive events from event hubs by using Python (azure-eventhub)

11/26/2021 • 4 minutes to read • [Edit Online](#)

This quickstart shows how to send events to and receive events from an event hub using the `azure-eventhub` Python package.

## Prerequisites

If you're new to Azure Event Hubs, see [Event Hubs overview](#) before you do this quickstart.

To complete this quickstart, you need the following prerequisites:

- **Microsoft Azure subscription.** To use Azure services, including Azure Event Hubs, you need a subscription. If you don't have an existing Azure account, you can sign up for a [free trial](#) or use your MSDN subscriber benefits when you [create an account](#).
- Python 2.7 or 3.6 or later, with PIP installed and updated.
- The Python package for Event Hubs.

To install the package, run this command in a command prompt that has Python in its path:

```
pip install azure-eventhub
```

Install the following package for receiving the events by using Azure Blob storage as the checkpoint store:

```
pip install azure-eventhub-checkpointstoreblob-aio
```

- **Create an Event Hubs namespace and an event hub.** The first step is to use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#). Then, get the **connection string for the Event Hubs namespace** by following instructions from the article: [Get connection string](#). You use the connection string later in this quickstart.

## Send events

In this section, you create a Python script to send events to the event hub that you created earlier.

1. Open your favorite Python editor, such as [Visual Studio Code](#).
2. Create a script called `send.py`. This script sends a batch of events to the event hub that you created earlier.
3. Paste the following code into `send.py`.

```

import asyncio
from azure.eventhub.aio import EventHubProducerClient
from azure.eventhub import EventData

async def run():
    # Create a producer client to send messages to the event hub.
    # Specify a connection string to your event hubs namespace and
    # the event hub name.
    producer = EventHubProducerClient.from_connection_string(conn_str="EVENT HUBS NAMESPACE -
CONNECTION STRING", eventhub_name="EVENT HUB NAME")
    async with producer:
        # Create a batch.
        event_data_batch = await producer.create_batch()

        # Add events to the batch.
        event_data_batch.add(EventData('First event '))
        event_data_batch.add(EventData('Second event'))
        event_data_batch.add(EventData('Third event'))

        # Send the batch of events to the event hub.
        await producer.send_batch(event_data_batch)

loop = asyncio.get_event_loop()
loop.run_until_complete(run())

```

#### **NOTE**

For the complete source code, including informational comments, go to the [GitHub send\\_async.py page](#).

## Receive events

This quickstart uses Azure Blob storage as a checkpoint store. The checkpoint store is used to persist checkpoints (that is, the last read positions).

#### **WARNING**

If you run this code on Azure Stack Hub, you will experience runtime errors unless you target a specific Storage API version. That's because the Event Hubs SDK uses the latest available Azure Storage API available in Azure that may not be available on your Azure Stack Hub platform. Azure Stack Hub may support a different version of Storage Blob SDK than those typically available on Azure. If you are using Azure Blob Storage as a checkpoint store, check the [supported Azure Storage API version for your Azure Stack Hub build](#) and target that version in your code.

For example, If you are running on Azure Stack Hub version 2005, the highest available version for the Storage service is version 2019-02-02. By default, the Event Hubs SDK client library uses the highest available version on Azure (2019-07-07 at the time of the release of the SDK). In this case, besides following steps in this section, you will also need to add code to target the Storage service API version 2019-02-02. For an example on how to target a specific Storage API version, see the [synchronous](#) and [asynchronous](#) samples on GitHub.

### Create an Azure storage account and a blob container

Create an Azure storage account and a blob container in it by doing the following steps:

1. [Create an Azure Storage account](#)
2. [Create a blob container](#)
3. [Get the connection string to the storage account](#)

Be sure to record the connection string and container name for later use in the receive code.

## Create a Python script to receive events

In this section, you create a Python script to receive events from your event hub:

1. Open your favorite Python editor, such as [Visual Studio Code](#).
2. Create a script called `recv.py`.
3. Paste the following code into `recv.py`.

```
import asyncio
from azure.eventhub.aio import EventHubConsumerClient
from azure.eventhub.extensions.checkpointstoreblobaio import BlobCheckpointStore

async def on_event(partition_context, event):
    # Print the event data.
    print("Received the event: \"{}\" from the partition with ID: \"{}\"".format(event.body_as_str(encoding='UTF-8'), partition_context.partition_id))

    # Update the checkpoint so that the program doesn't read the events
    # that it has already read when you run it next time.
    await partition_context.update_checkpoint(event)

async def main():
    # Create an Azure blob checkpoint store to store the checkpoints.
    checkpoint_store = BlobCheckpointStore.from_connection_string("AZURE STORAGE CONNECTION STRING",
"BLOB CONTAINER NAME")

    # Create a consumer client for the event hub.
    client = EventHubConsumerClient.from_connection_string("EVENT HUBS NAMESPACE CONNECTION STRING",
consumer_group="$Default", eventhub_name="EVENT HUB NAME", checkpoint_store=checkpoint_store)
    async with client:
        # Call the receive method. Read from the beginning of the partition (starting_position: "-1")
        await client.receive(on_event=on_event, starting_position="-1")

if __name__ == '__main__':
    loop = asyncio.get_event_loop()
    # Run the main method.
    loop.run_until_complete(main())
```

### NOTE

For the complete source code, including additional informational comments, go to the [GitHub recv\\_with\\_checkpoint\\_store\\_async.py page](#).

## Run the receiver app

To run the script, open a command prompt that has Python in its path, and then run this command:

```
python recv.py
```

## Run the sender app

To run the script, open a command prompt that has Python in its path, and then run this command:

```
python send.py
```

The receiver window should display the messages that were sent to the event hub.

## Next steps

In this quickstart, you've sent and received events asynchronously. To learn how to send and receive events synchronously, go to the [GitHub sync\\_samples page](#).

For all the samples (both synchronous and asynchronous) on GitHub, go to [Azure Event Hubs client library for Python samples](#).

# Send events to or receive events from event hubs by using JavaScript (azure/event-hubs)

11/26/2021 • 6 minutes to read • [Edit Online](#)

This quickstart shows how to send events to and receive events from an event hub using the [azure/event-hubs](#) JavaScript package.

## Prerequisites

If you are new to Azure Event Hubs, see [Event Hubs overview](#) before you do this quickstart.

To complete this quickstart, you need the following prerequisites:

- **Microsoft Azure subscription.** To use Azure services, including Azure Event Hubs, you need a subscription. If you don't have an existing Azure account, you can sign up for a [free trial](#) or use your MSDN subscriber benefits when you [create an account](#).
- Node.js version 8.x or later. Download the latest [long-term support \(LTS\) version](#).
- Visual Studio Code (recommended) or any other integrated development environment (IDE).
- An active Event Hubs namespace and event hub. To create them, do the following steps:
  1. In the [Azure portal](#), create a namespace of type *Event Hubs*, and then obtain the management credentials that your application needs to communicate with the event hub.
  2. To create the namespace and event hub, follow the instructions at [Quickstart: Create an event hub by using the Azure portal](#).
  3. Continue by following the instructions in this quickstart.
  4. To get the connection string for your Event Hub namespace, follow the instructions in [Get connection string](#). Record the connection string to use later in this quickstart.
- **Create an Event Hubs namespace and an event hub.** The first step is to use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#). Then, get the **connection string for the Event Hubs namespace** by following instructions from the article: [Get connection string](#). You use the connection string later in this quickstart.

## Install the npm package

To install the [Node Package Manager \(npm\) package for Event Hubs](#), open a command prompt that has *npm* in its path, change the directory to the folder where you want to keep your samples, and then run this command:

```
npm install @azure/event-hubs
```

For the receiving side, you need to install two more packages. In this quickstart, you use Azure Blob storage to persist checkpoints so that the program doesn't read the events that it has already read. It performs metadata checkpoints on received messages at regular intervals in a blob. This approach makes it easy to continue receiving messages later from where you left off.

Run the following commands:

```
npm install @azure/storage-blob
```

```
npm install @azure/eventhubs-checkpointstore-blob
```

## Send events

In this section, you create a JavaScript application that sends events to an event hub.

1. Open your favorite editor, such as [Visual Studio Code](#).
2. Create a file called *send.js*, and paste the following code into it:

```
const { EventHubProducerClient } = require("@azure/event-hubs");

const connectionString = "EVENT HUBS NAMESPACE CONNECTION STRING";
const eventHubName = "EVENT HUB NAME";

async function main() {

    // Create a producer client to send messages to the event hub.
    const producer = new EventHubProducerClient(connectionString, eventHubName);

    // Prepare a batch of three events.
    const batch = await producer.createBatch();
    batch.tryAdd({ body: "First event" });
    batch.tryAdd({ body: "Second event" });
    batch.tryAdd({ body: "Third event" });

    // Send the batch to the event hub.
    await producer.sendBatch(batch);

    // Close the producer client.
    await producer.close();

    console.log("A batch of three events have been sent to the event hub");
}

main().catch((err) => {
    console.log("Error occurred: ", err);
});
```

3. In the code, use real values to replace the following:
  - EVENT HUBS NAMESPACE CONNECTION STRING
  - EVENT HUB NAME
4. Run `node send.js` to execute this file. This command sends a batch of three events to your event hub.
5. In the Azure portal, verify that the event hub has received the messages. In the **Metrics** section, switch to **Messages** view. Refresh the page to update the chart. It might take a few seconds for it to show that the messages have been received.

**Event Hub Overview**

Status : Active Updated : Wednesday, December 4, 2019, 12:22:43 EST

Location : East US Pricing tier : Standard

Subscription (change) : <Subscription name> Throughput Units : 1 unit

Subscription ID : <Subscription ID> Auto-inflate throughput... : Disabled

Host name : spehubdotnetns.servicebus.windows.net

Tags (change) : Click here to add tags

**NAMESPACE CONTENTS** 1 EVENT HUB KAFKA SURFACE ENABLED

Show metrics: Requests Messages Throughput For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

3  
2.5  
2  
1.5  
1  
0.5  
0

12 PM 12:15 PM 12:30 PM 12:45 PM

Incoming Messages (Sum) Outgoing Messages (Sum) Captured Messages (Sum) Capture Backlog (Sum)

spehubdotnetns 0 0 0 0

Search to filter items...

Name	Status	message retention	partition count
spehub	Active	1	1

#### NOTE

For the complete source code, including additional informational comments, go to the [GitHub sendEvents.js](#) page.

Congratulations! You have now sent events to an event hub.

## Receive events

In this section, you receive events from an event hub by using an Azure Blob storage checkpoint store in a JavaScript application. It performs metadata checkpoints on received messages at regular intervals in an Azure Storage blob. This approach makes it easy to continue receiving messages later from where you left off.

#### WARNING

If you run this code on Azure Stack Hub, you will experience runtime errors unless you target a specific Storage API version. That's because the Event Hubs SDK uses the latest available Azure Storage API available in Azure that may not be available on your Azure Stack Hub platform. Azure Stack Hub may support a different version of Storage Blob SDK than those typically available on Azure. If you are using Azure Blob Storage as a checkpoint store, check the [supported Azure Storage API version](#) for your Azure Stack Hub build and target that version in your code.

For example, If you are running on Azure Stack Hub version 2005, the highest available version for the Storage service is version 2019-02-02. By default, the Event Hubs SDK client library uses the highest available version on Azure (2019-07-07 at the time of the release of the SDK). In this case, besides following steps in this section, you will also need to add code to target the Storage service API version 2019-02-02. For an example on how to target a specific Storage API version, see [JavaScript](#) and [TypeScript](#) samples on GitHub.

### Create an Azure storage account and a blob container

To create an Azure storage account and a blob container in it, do the following actions:

1. [Create an Azure storage account](#)
2. [Create a blob container in the storage account](#)
3. [Get the connection string to the storage account](#)

Be sure to record the connection string and container name for later use in the receive code.

## Write code to receive events

1. Open your favorite editor, such as [Visual Studio Code](#).
2. Create a file called *receive.js*, and paste the following code into it:

```
const { EventHubConsumerClient } = require("@azure/event-hubs");
const { ContainerClient } = require("@azure/storage-blob");
const { BlobCheckpointStore } = require("@azure/eventhubs-checkpointstore-blob");

const connectionString = "EVENT HUBS NAMESPACE CONNECTION STRING";
const eventHubName = "EVENT HUB NAME";
const consumerGroup = "$Default"; // name of the default consumer group
const storageConnectionString = "AZURE STORAGE CONNECTION STRING";
const containerName = "BLOB CONTAINER NAME";

async function main() {
    // Create a blob container client and a blob checkpoint store using the client.
    const containerClient = new ContainerClient(storageConnectionString, containerName);
    const checkpointStore = new BlobCheckpointStore(containerClient);

    // Create a consumer client for the event hub by specifying the checkpoint store.
    const consumerClient = new EventHubConsumerClient(consumerGroup, connectionString, eventHubName,
checkpointStore);

    // Subscribe to the events, and specify handlers for processing the events and errors.
    const subscription = consumerClient.subscribe({
        processEvents: async (events, context) => {
            if (events.length === 0) {
                console.log(`No events received within wait time. Waiting for next interval`);
                return;
            }

            for (const event of events) {
                console.log(`Received event: '${event.body}' from partition: '${context.partitionId}' and
consumer group: '${context.consumerGroup}'`);
            }
            // Update the checkpoint.
            await context.updateCheckpoint(events[events.length - 1]);
        },
        processError: async (err, context) => {
            console.log(`Error : ${err}`);
        }
    });
}

// After 30 seconds, stop processing.
await new Promise((resolve) => {
    setTimeout(async () => {
        await subscription.close();
        await consumerClient.close();
        resolve();
    }, 30000);
});
}

main().catch((err) => {
    console.log("Error occurred: ", err);
});
```

3. In the code, use real values to replace the following values:

- EVENT HUBS NAMESPACE CONNECTION STRING

- EVENT HUB NAME
- AZURE STORAGE CONNECTION STRING
- BLOB CONTAINER NAME

4. Run `node receive.js` in a command prompt to execute this file. The window should display messages about received events.

**NOTE**

For the complete source code, including additional informational comments, go to the [GitHub receiveEventsUsingCheckpointStore.js page](#).

Congratulations! You have now received events from your event hub. The receiver program will receive events from all the partitions of the default consumer group in the event hub.

## Next steps

Check out these samples on GitHub:

- [JavaScript samples](#)
- [TypeScript samples](#)

# Quickstart: Send events to or receive events from Event Hubs using Go

11/26/2021 • 6 minutes to read • [Edit Online](#)

Azure Event Hubs is a Big Data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see [Event Hubs overview](#) and [Event Hubs features](#).

This tutorial describes how to write Go applications to send events to or receive events from an event hub.

## NOTE

You can download this quickstart as a sample from the [GitHub](#), replace `EventHubConnectionString` and `EventHubName` strings with your event hub values, and run it. Alternatively, you can follow the steps in this tutorial to create your own.

## Prerequisites

To complete this tutorial, you need the following prerequisites:

- Go installed locally. Follow [these instructions](#) if necessary.
- An active Azure account. If you don't have an Azure subscription, create a [free account](#) before you begin.
- **Create an Event Hubs namespace and an event hub.** Use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#).

## Send events

This section shows you how to create a Go application to send events to an event hub.

### Install Go package

Get the Go package for Event Hubs with `go get` or `dep`. For example:

```
go get -u github.com/Azure/azure-event-hubs-go
go get -u github.com/Azure/azure-amqp-common-go/...
# or
dep ensure -add github.com/Azure/azure-event-hubs-go
dep ensure -add github.com/Azure/azure-amqp-common-go
```

### Import packages in your code file

To import the Go packages, use the following code example:

```
import (
    aad "github.com/Azure/azure-amqp-common-go/aad"
    eventhubs "github.com/Azure/azure-event-hubs-go"
)
```

## Create service principal

Create a new service principal by following the instructions in [Create an Azure service principal with Azure CLI 2.0](#). Save the provided credentials in your environment with the following names. Both the Azure SDK for Go and the Event Hubs packages are preconfigured to look for these variable names:

```
export AZURE_CLIENT_ID=
export AZURE_CLIENT_SECRET=
export AZURE_TENANT_ID=
export AZURE_SUBSCRIPTION_ID=
```

Now, create an authorization provider for your Event Hubs client that uses these credentials:

```
tokenProvider, err := aad.NewJWTProvider(aad.JWTProviderWithEnvironmentVars())
if err != nil {
    log.Fatalf("failed to configure AAD JWT provider: %s\n", err)
}
```

## Create Event Hubs client

The following code creates an Event Hubs client:

```
hub, err := eventhubs.NewHub("namespaceName", "hubName", tokenProvider)
ctx, cancel := context.WithTimeout(context.Background(), 10 * time.Second)
defer hub.Close(ctx)
if err != nil {
    log.Fatalf("failed to get hub %s\n", err)
}
```

## Write code to send messages

In the following snippet, use (1) to send messages interactively from a terminal, or (2) to send messages within your program:

```
// 1. send messages at the terminal
ctx = context.Background()
reader := bufio.NewReader(os.Stdin)
for {
    fmt.Printf("Input a message to send: ")
    text, _ := reader.ReadString('\n')
    hub.Send(ctx, eventhubs.NewEventFromString(text))
}

// 2. send messages within program
ctx = context.Background()
hub.Send(ctx, eventhubs.NewEventFromString("hello Azure!"))
```

## Extras

Get the IDs of the partitions in your event hub:

```
info, err := hub.GetRuntimeInformation(ctx)
if err != nil {
    log.Fatalf("failed to get runtime info: %s\n", err)
}
log.Printf("got partition IDs: %s\n", info.PartitionIDs)
```

Run the application to send events to the event hub.

Congratulations! You have now sent messages to an event hub.

# Receive events

## Create a Storage account and container

State such as leases on partitions and checkpoints in the event stream are shared between receivers using an Azure Storage container. You can create a storage account and container with the Go SDK, but you can also create one by following the instructions in [About Azure storage accounts](#).

Samples for creating Storage artifacts with the Go SDK are available in the [Go samples repo](#) and in the sample corresponding to this tutorial.

## Go packages

To receive the messages, get the Go packages for Event Hubs with `go get` or `dep`:

```
go get -u github.com/Azure/azure-event-hubs-go/...
go get -u github.com/Azure/azure-amqp-common-go/...
go get -u github.com/Azure/go-autorest/...

# or

dep ensure -add github.com/Azure/azure-event-hubs-go
dep ensure -add github.com/Azure/azure-amqp-common-go
dep ensure -add github.com/Azure/go-autorest
```

## Import packages in your code file

To import the Go packages, use the following code example:

```
import (
    aad "github.com/Azure/azure-amqp-common-go/aad"
    eventhubs "github.com/Azure/azure-event-hubs-go"
    eph "github.com/Azure/azure-event-hubs-go/eph"
    storageLeaser "github.com/Azure/azure-event-hubs-go/storage"
    azure "github.com/Azure/go-autorest/autorest/azure"
)
```

## Create service principal

Create a new service principal by following the instructions in [Create an Azure service principal with Azure CLI 2.0](#). Save the provided credentials in your environment with the following names: Both Azure SDK for Go and Event Hubs package are preconfigured to look for these variable names.

```
export AZURE_CLIENT_ID=
export AZURE_CLIENT_SECRET=
export AZURE_TENANT_ID=
export AZURE_SUBSCRIPTION_ID=
```

Next, create an authorization provider for your Event Hubs client that uses these credentials:

```
tokenProvider, err := aad.NewJWTProvider(aad.JWTProviderWithEnvironmentVars())
if err != nil {
    log.Fatalf("failed to configure AAD JWT provider: %s\n", err)
}
```

## Get metadata struct

Get a struct with metadata about your Azure environment using the Azure Go SDK. Later operations use this struct to find correct endpoints.

```

azureEnv, err := azure.EnvironmentFromName("AzurePublicCloud")
if err != nil {
    log.Fatalf("could not get azure.Environment struct: %s\n", err)
}

```

## Create credential helper

Create a credential helper that uses the previous Azure Active Directory (AAD) credentials to create a Shared Access Signature (SAS) credential for Storage. The last parameter tells this constructor to use the same environment variables as used previously:

```

cred, err := storageLeaser.NewAADSASCredential(
    subscriptionID,
    resourceGroupName,
    storageAccountName,
    storageContainerName,
    storageLeaser.AADSASCredentialWithEnvironmentVars())
if err != nil {
    log.Fatalf("could not prepare a storage credential: %s\n", err)
}

```

## Create a check pointer and a leaser

Create a **leaser**, responsible for leasing a partition to a particular receiver, and a **check pointer**, responsible for writing checkpoints for the message stream so that other receivers can begin reading from the correct offset.

Currently, a single **StorageLeaserCheckpointer** is available that uses the same Storage container to manage both leases and checkpoints. In addition to the storage account and container names, the **StorageLeaserCheckpointer** needs the credential created in the previous step and the Azure environment struct to correctly access the container.

```

leaserCheckpointer, err := storageLeaser.NewStorageLeaserCheckpointer(
    cred,
    storageAccountName,
    storageContainerName,
    azureEnv)
if err != nil {
    log.Fatalf("could not prepare a storage leaserCheckpointer: %s\n", err)
}

```

## Construct Event Processor Host

You now have the pieces needed to construct an EventProcessorHost, as follows. The same **StorageLeaserCheckpointer** is used as both a leaser and check pointer, as described previously:

```

ctx := context.Background()
p, err := eph.New(
    ctx,
    nsName,
    hubName,
    tokenProvider,
    leaserCheckpointer,
    leaserCheckpointer)
if err != nil {
    log.Fatalf("failed to create EPH: %s\n", err)
}
defer p.Close(context.Background())

```

## Create handler

Now create a handler and register it with the Event Processor Host. When the host is started, it applies this and any other specified handlers to incoming messages:

```
handler := func(ctx context.Context, event *eventhubs.Event) error {
    fmt.Printf("received: %s\n", string(event.Data))
    return nil
}

// register the handler with the EPH
_, err := p.RegisterHandler(ctx, handler)
if err != nil {
    log.Fatalf("failed to register handler: %s\n", err)
}
```

### Write code to receive messages

With everything set up, you can start the Event Processor Host with `Start(context)` to keep it permanently running, or with `StartNonBlocking(context)` to run only as long as messages are available.

This tutorial starts and runs as follows; see the GitHub sample for an example using `StartNonBlocking`:

```
ctx := context.Background()
err = p.Start()
if err != nil {
    log.Fatalf("failed to start EPH: %s\n", err)
}
```

## Next steps

Read the following articles:

- [EventProcessorHost](#)
- [Features and terminology in Azure Event Hubs](#)
- [Event Hubs FAQ](#)

# Quickstart: Send events to Azure Event Hubs using C

11/26/2021 • 3 minutes to read • [Edit Online](#)

## Introduction

Azure Event Hubs is a Big Data streaming platform and event ingestion service, capable of receiving and processing millions of events per second. Event Hubs can process and store events, data, or telemetry produced by distributed software and devices. Data sent to an event hub can be transformed and stored using any real-time analytics provider or batching/storage adapters. For detailed overview of Event Hubs, see [Event Hubs overview](#) and [Event Hubs features](#).

This tutorial describes how to send events to an event hub using a console application in C.

## Prerequisites

To complete this tutorial, you need the following:

- A C development environment. This tutorial assumes the gcc stack on an Azure Linux VM with Ubuntu 14.04.
- [Microsoft Visual Studio](#).
- **Create an Event Hubs namespace and an event hub.** Use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#). Get the value of access key for the event hub by following instructions from the article: [Get connection string](#). You use the access key in the code you write later in this tutorial. The default key name is: `RootManageSharedAccessKey`.

## Write code to send messages to Event Hubs

In this section shows how to write a C app to send events to your event hub. The code uses the Proton AMQP library from the [Apache Qpid project](#). This is analogous to using Service Bus queues and topics with AMQP from C as shown [in this sample](#). For more information, see the [Qpid Proton documentation](#).

1. From the [Qpid AMQP Messenger page](#), follow the instructions to install Qpid Proton, depending on your environment.
2. To compile the Proton library, install the following packages:

```
sudo apt-get install build-essential cmake uuid-dev openssl libssl-dev
```

3. Download the [Qpid Proton library](#), and extract it, e.g.:

```
wget https://archive.apache.org/dist/qpid/proton/0.7/qpid-proton-0.7.tar.gz
tar xvfz qpid-proton-0.7.tar.gz
```

4. Create a build directory, compile and install:

```

cd qpid-proton-0.7
mkdir build
cd build
cmake -DCMAKE_INSTALL_PREFIX=/usr ..
sudo make install

```

5. In your work directory, create a new file called **sender.c** with the following code. Remember to replace the values for your SAS key/name, event hub name, and namespace. You must also substitute a URL-encoded version of the key for the **SendRule** created earlier. You can URL-encode it [here](#).

```

#include "proton/message.h"
#include "proton/messenger.h"

#include <getopt.h>
#include <proton/util.h>
#include <sys/time.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>

#define check(messenger) \
{ \
    if(pn_messenger_errno(messenger)) \
    { \
        printf("check\n"); \
        die(__FILE__, __LINE__, pn_error_text(pn_messenger_error(messenger))); \
    } \
}

pn_timestamp_t time_now(void)
{
    struct timeval now;
    if (gettimeofday(&now, NULL)) pn_fatal("gettimeofday failed\n");
    return ((pn_timestamp_t)now.tv_sec) * 1000 + (now.tv_usec / 1000);
}

void die(const char *file, int line, const char *message)
{
    printf("Dead\n");
    fprintf(stderr, "%s:%i: %s\n", file, line, message);
    exit(1);
}

int sendMessage(pn_messenger_t * messenger) {
    char * address = (char *) "amqps://{{SAS Key Name}}:{SAS key}@{{namespace
name}}.servicebus.windows.net/{{event hub name}}";
    char * msgtext = (char *) "Hello from C!";

    pn_message_t * message;
    pn_data_t * body;
    message = pn_message();

    pn_message_set_address(message, address);
    pn_message_set_content_type(message, (char*) "application/octect-stream");
    pn_message_set_inferred(message, true);

    body = pn_message_body(message);
    pn_data_put_binary(body, pn_bytes(strlen(msgtext), msgtext));

    pn_messenger_put(messenger, message);
    check(messenger);
    pn_messenger_send(messenger, 1);
    check(messenger);
}

```

```
    pn_message_free(message);
}

int main(int argc, char** argv) {
    printf("Press Ctrl-C to stop the sender process\n");

    pn_messenger_t *messenger = pn_messenger(NULL);
    pn_messenger_set_outgoing_window(messenger, 1);
    pn_messenger_start(messenger);

    while(true) {
        sendMessage(messenger);
        printf("Sent message\n");
        sleep(1);
    }

    // release messenger resources
    pn_messenger_stop(messenger);
    pn_messenger_free(messenger);

    return 0;
}
```

## 6. Compile the file, assuming `gcc`:

```
gcc sender.c -o sender -lqpid-proton
```

### NOTE

This code uses an outgoing window of 1 to force the messages out as soon as possible. It is recommended that your application try to batch messages to increase throughput. See the [Qpid AMQP Messenger page](#) for information about how to use the Qpid Proton library in this and other environments, and from platforms for which bindings are provided (currently Perl, PHP, Python, and Ruby).

Run the application to send messages to the event hub.

Congratulations! You have now sent messages to an event hub.

## Next steps

Read the following articles:

- [EventProcessorHost](#)
- [Features and terminology in Azure Event Hubs.](#)

# Quickstart: Receive events from Event Hubs using Apache Storm

11/26/2021 • 4 minutes to read • [Edit Online](#)

Apache Storm is a distributed real-time computation system that simplifies reliable processing of unbounded streams of data. This section shows how to use an Azure Event Hubs Storm spout to receive events from Event Hubs. Using Apache Storm, you can split events across multiple processes hosted in different nodes. The Event Hubs integration with Storm simplifies event consumption by transparently checkpointing its progress using Storm's Zookeeper installation, managing persistent checkpoints and parallel receives from Event Hubs.

For more information about Event Hubs receive patterns, see the [Event Hubs overview](#).

## Prerequisites

Before you start with the quickstart, **create an Event Hubs namespace and an event hub**. Use the [Azure portal](#) to create a namespace of type Event Hubs, and obtain the management credentials your application needs to communicate with the event hub. To create a namespace and an event hub, follow the procedure in [this article](#).

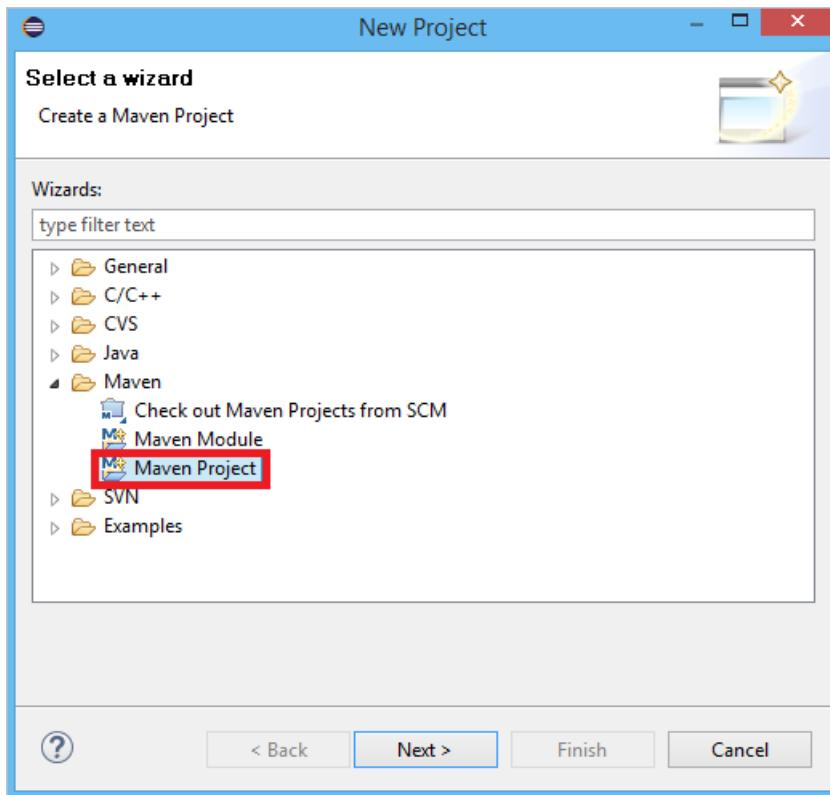
## Create project and add code

This tutorial uses an [HDInsight Storm](#) installation, which comes with the Event Hubs spout already available.

1. Follow the [HDInsight Storm - Get Started](#) procedure to create a new HDInsight cluster, and connect to it via Remote Desktop.
2. Copy the `%STORM_HOME%\examples\eventhubspout\eventhubs-storm-spout-0.9-jar-with-dependencies.jar` file to your local development environment. This contains the events-storm-spout.
3. Use the following command to install the package into the local Maven store. This enables you to add it as a reference in the Storm project in a later step.

```
mvn install:install-file -Dfile=target\eventhubs-storm-spout-0.9-jar-with-dependencies.jar -  
DgroupId=com.microsoft.eventhubs -DartifactId=eventhubs-storm-spout -Dversion=0.9 -Dpackaging=jar
```

4. In Eclipse, create a new Maven project (click **File**, then **New**, then **Project**).



5. Select **Use default Workspace location**, then click **Next**
6. Select the **maven-archetype-quickstart** archetype, then click **Next**
7. Insert a **GroupId** and **ArtifactId**, then click **Finish**
8. In **pom.xml**, add the following dependencies in the `<dependency>` node.

```
<dependency>
    <groupId>org.apache.storm</groupId>
    <artifactId>storm-core</artifactId>
    <version>0.9.2-incubating</version>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>com.microsoft.eventhubs</groupId>
    <artifactId>eventhubs-storm-spout</artifactId>
    <version>0.9</version>
</dependency>
<dependency>
    <groupId>com.netflix.curator</groupId>
    <artifactId>curator-framework</artifactId>
    <version>1.3.3</version>
    <exclusions>
        <exclusion>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
        </exclusion>
        <exclusion>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-log4j12</artifactId>
        </exclusion>
    </exclusions>
    <scope>provided</scope>
</dependency>
```

9. In the **src** folder, create a file called **Config.properties** and copy the following content, substituting the `receive rule key` and `event hub name` values:

```

eventhubspout.username = ReceiveRule
eventhubspout.password = {receive rule key}
eventhubspout.namespace = ioteventhub-ns
eventhubspout.entitypath = {event hub name}
eventhubspout.partitions.count = 16

# if not provided, will use storm's zookeeper settings
# zookeeper.connectionstring=localhost:2181

eventhubspout.checkpoint.interval = 10
eventhub.receiver.credits = 10

```

The value for `eventhub.receiver.credits` determines how many events are batched before releasing them to the Storm pipeline. For the sake of simplicity, this example sets this value to 10. In production, it should usually be set to higher values; for example, 1024.

10. Create a new class called `LoggerBolt` with the following code:

```

import java.util.Map;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import backtype.storm.task.OutputCollector;
import backtype.storm.task.TopologyContext;
import backtype.storm.topology.OutputFieldsDeclarer;
import backtype.storm.topology.base.BaseRichBolt;
import backtype.storm.tuple.Tuple;

public class LoggerBolt extends BaseRichBolt {
    private OutputCollector collector;
    private static final Logger logger = LoggerFactory
        .getLogger(LoggerBolt.class);

    @Override
    public void execute(Tuple tuple) {
        String value = tuple.getString(0);
        logger.info("Tuple value: " + value);

        collector.ack(tuple);
    }

    @Override
    public void prepare(Map map, TopologyContext context, OutputCollector collector) {
        this.collector = collector;
        this.count = 0;
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        // no output fields
    }
}

```

This Storm bolt logs the content of the received events. This can easily be extended to store tuples in a storage service. The [HDInsight Storm with Event Hub example](#) uses this same approach to store data into Azure Storage and Power BI.

11. Create a class called `LogTopology` with the following code:

```

import java.io.FileReader;
import java.util.Properties;
import backtype.storm.Config;
import backtype.storm.LocalCluster;

```

```

import backtype.storm.StormSubmitter;
import backtype.storm.generated.StormTopology;
import backtype.storm.topology.TopologyBuilder;
import com.microsoft.eventhubs.samples.EventCount;
import com.microsoft.eventhubs.spout.EventHubSpout;
import com.microsoft.eventhubs.spout.EventHubSpoutConfig;

public class LogTopology {
    protected EventHubSpoutConfig spoutConfig;
    protected int numWorkers;

    protected void readEHConfig(String[] args) throws Exception {
        Properties properties = new Properties();
        if (args.length > 1) {
            properties.load(new FileReader(args[1]));
        } else {
            properties.load(EventCount.class.getClassLoader()
                .getResourceAsStream("Config.properties"));
        }

        String username = properties.getProperty("eventhubspout.username");
        String password = properties.getProperty("eventhubspout.password");
        String namespaceName = properties
            .getProperty("eventhubspout.namespace");
        String entityPath = properties.getProperty("eventhubspout.entitypath");
        String zkEndpointAddress = properties
            .getProperty("zookeeper.connectionstring"); // opt
        int partitionCount = Integer.parseInt(properties
            .getProperty("eventhubspout.partitions.count"));
        int checkpointIntervalInSeconds = Integer.parseInt(properties
            .getProperty("eventhubspout.checkpoint.interval"));
        int receiverCredits = Integer.parseInt(properties
            .getProperty("eventhub.receiver.credits")); // prefetch count
                                                // (opt)
        System.out.println("Eventhub spout config: ");
        System.out.println(" partition count: " + partitionCount);
        System.out.println(" checkpoint interval: "
            + checkpointIntervalInSeconds);
        System.out.println(" receiver credits: " + receiverCredits);

        spoutConfig = new EventHubSpoutConfig(username, password,
            namespaceName, entityPath, partitionCount, zkEndpointAddress,
            checkpointIntervalInSeconds, receiverCredits);

        // set the number of workers to be the same as partition number.
        // the idea is to have a spout and a logger bolt co-exist in one
        // worker to avoid shuffling messages across workers in storm cluster.
        numWorkers = spoutConfig.getPartitionCount();

        if (args.length > 0) {
            // set topology name so that sample Trident topology can use it as
            // stream name.
            spoutConfig.setTopologyName(args[0]);
        }
    }

    protected StormTopology buildTopology() {
        TopologyBuilder topologyBuilder = new TopologyBuilder();

        EventHubSpout eventHubSpout = new EventHubSpout(spoutConfig);
        topologyBuilder.setSpout("EventHubsSpout", eventHubSpout,
            spoutConfig.getPartitionCount()).setNumTasks(
            spoutConfig.getPartitionCount());
        topologyBuilder
            .setBolt("LoggerBolt", new LoggerBolt(),
                spoutConfig.getPartitionCount())
            .localOrShuffleGrouping("EventHubsSpout")
            .setNumTasks(spoutConfig.getPartitionCount());
        return topologyBuilder.createTopology();
    }
}

```

```

    }

protected void runScenario(String[] args) throws Exception {
    boolean runLocal = true;
    readEHConfig(args);
    StormTopology topology = buildTopology();
    Config config = new Config();
    config.setDebug(false);

    if (runLocal) {
        config.setMaxTaskParallelism(2);
        LocalCluster localCluster = new LocalCluster();
        localCluster.submitTopology("test", config, topology);
        Thread.sleep(5000000);
        localCluster.shutdown();
    } else {
        config.setNumWorkers(numWorkers);
        StormSubmitter.submitTopology(args[0], config, topology);
    }
}

public static void main(String[] args) throws Exception {
    LogTopology topology = new LogTopology();
    topology.runScenario(args);
}
}

```

This class creates a new Event Hubs spout, using the properties in the configuration file to instantiate it. It is important to note that this example creates as many spouts tasks as the number of partitions in the event hub, in order to use the maximum parallelism allowed by that event hub.

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)
- [Create an event hub](#)
- [Event Hubs FAQ](#)

# Enable capturing of events streaming through Azure Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

Azure Event Hubs Capture enables you to automatically deliver the streaming data in Event Hubs to an [Azure Blob storage](#) or [Azure Data Lake Storage Gen1 or Gen 2](#) account of your choice.

You can configure Capture at the event hub creation time using the [Azure portal](#). You can either capture the data to an Azure [Blob storage](#) container, or to an [Azure Data Lake Storage Gen 1 or Gen 2](#) account.

For more information, see the [Event Hubs Capture overview](#).

## IMPORTANT

The destination storage (Azure Storage or Azure Data Lake Storage) account must be in the same subscription as the event hub.

## Capture data to Azure Storage

When you create an event hub, you can enable Capture by clicking the **On** button in the [Create Event Hub](#) portal screen. You then specify a Storage Account and container by clicking **Azure Storage** in the **Capture Provider** box. Because Event Hubs Capture uses service-to-service authentication with storage, you do not need to specify a storage connection string. The resource picker selects the resource URI for your storage account automatically. If you use Azure Resource Manager, you must supply this URI explicitly as a string.

The default time window is 5 minutes. The minimum value is 1, the maximum 15. The **Size** window has a range of 10-500 MB.

Dashboard > spehubns0206 - Overview > spehubns0206 > Create Event Hub

## Create Event Hub

Event Hubs

\* Name ⓘ  
spehub

Partition Count ⓘ  
2

Message Retention ⓘ  
1

Capture ⓘ  
 On  
 Off
 

Note: Enabling Capture will result in additional charges to this account. Learn more about our pricing [here](#).

Time window (minutes)  
5

Size window (MB)  
300

Do not emit empty files when no events occur during the Capture time window

Capture Provider  
Azure Storage

\* Azure Storage Container

Storage Account

Sample Capture file name formats

Capture file name format ⓘ

### NOTE

You can enable or disable emitting empty files when no events occur during the Capture window.

## Capture data to Azure Data Lake Storage Gen 2

- Follow [Create a storage account](#) article to create an Azure Storage account. Set **Hierarchical namespace** to **Enabled** on the **Advanced** tab to make it an Azure Data Lake Storage Gen 2 account. The Azure Storage account must be in the same subscription as the event hub.
- When creating an event hub, do the following steps:
  - Select **On** for **Capture**.
  - Select **Azure Storage** as the capture provider. The **Azure Data Lake Store** option you see for the **Capture provider** is for the Gen 1 of Azure Data Lake Storage. To use a Gen 2 of Azure Data

Lake Storage, you select Azure Storage.

- c. Select the **Select Container** button.

**Create Event Hub**

Name \*  ✓

Partition Count

Message Retention

**Capture** 1  
 On  Off

Note: Enabling Capture will result in additional charges to this account. Learn more about our pricing [here](#).

Time window (minutes)

Size window (MB)

Do not emit empty files when no events occur during the Capture time window

**Capture Provider** 2 Azure Storage ▼

Azure Storage Container \*  3 Select Container

Storage Account

Sample Capture file name formats  ▼

Capture file name format  {Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

e.g. spcapturedl2ehubsns/myehub/0/2020/1/12/17/1/48

**Create**

3. Select the **Azure Data Lake Storage Gen 2** account from the list.

All services > spehubrg > spcapturedl2ehubsns - Event Hubs > Create Event Hub > Storage accounts			
Storage accounts			
<a href="#">+</a> Storage account <span style="color: blue;">↻</span> Refresh			X
<input type="text" value="Search storage accounts"/>			<input checked="" type="checkbox"/> Show classic storage accounts
Name	Type	Resource Group	Location
mydlgen2storage	Standard-RAGRS	spehubrg	East US

4. Select the **container** (file system in Data Lake Storage Gen 2).

All services > spehubrg > spcapturedl2ehubs - Event Hubs > Create Event Hub > Storage accounts > Containers

**Containers**

mydldgen2storage

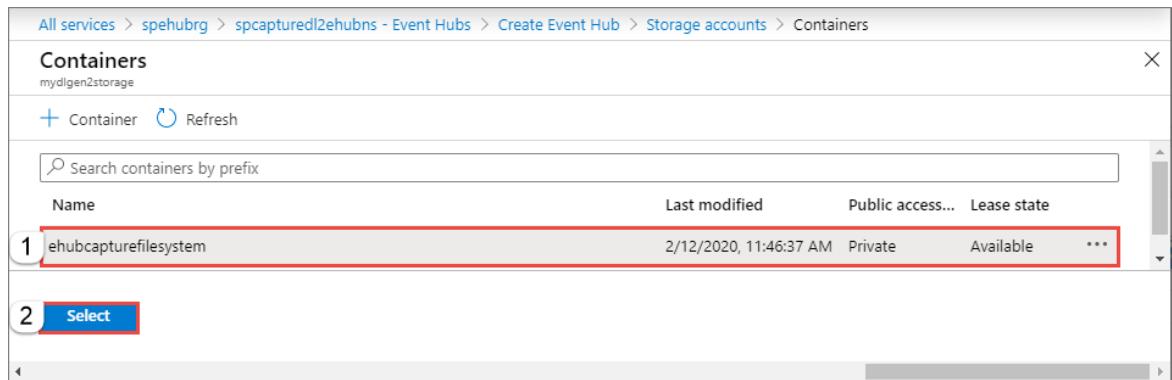
+ Container Refresh

Search containers by prefix

Name	Last modified	Public access...	Lease state
1 ehubcapturefilesystem	2/12/2020, 11:46:37 AM	Private	Available

2 Select

...



5. On the **Create Event Hub** page, select **Create**.

**Create Event Hub**

Event Hubs

Name \* ⓘ myehub ✓

Partition Count ⓘ 1

Message Retention ⓘ 1

Capture ⓘ **On** Off

Note: Enabling Capture will result in additional charges to this account. Learn more about our pricing [here](#).

Time window (minutes) 5

Size window (MB) 300

Do not emit empty files when no events occur during the Capture time window

Capture Provider Azure Storage

Azure Storage Container \* ehubcapturefilesystem ✓ Select Container

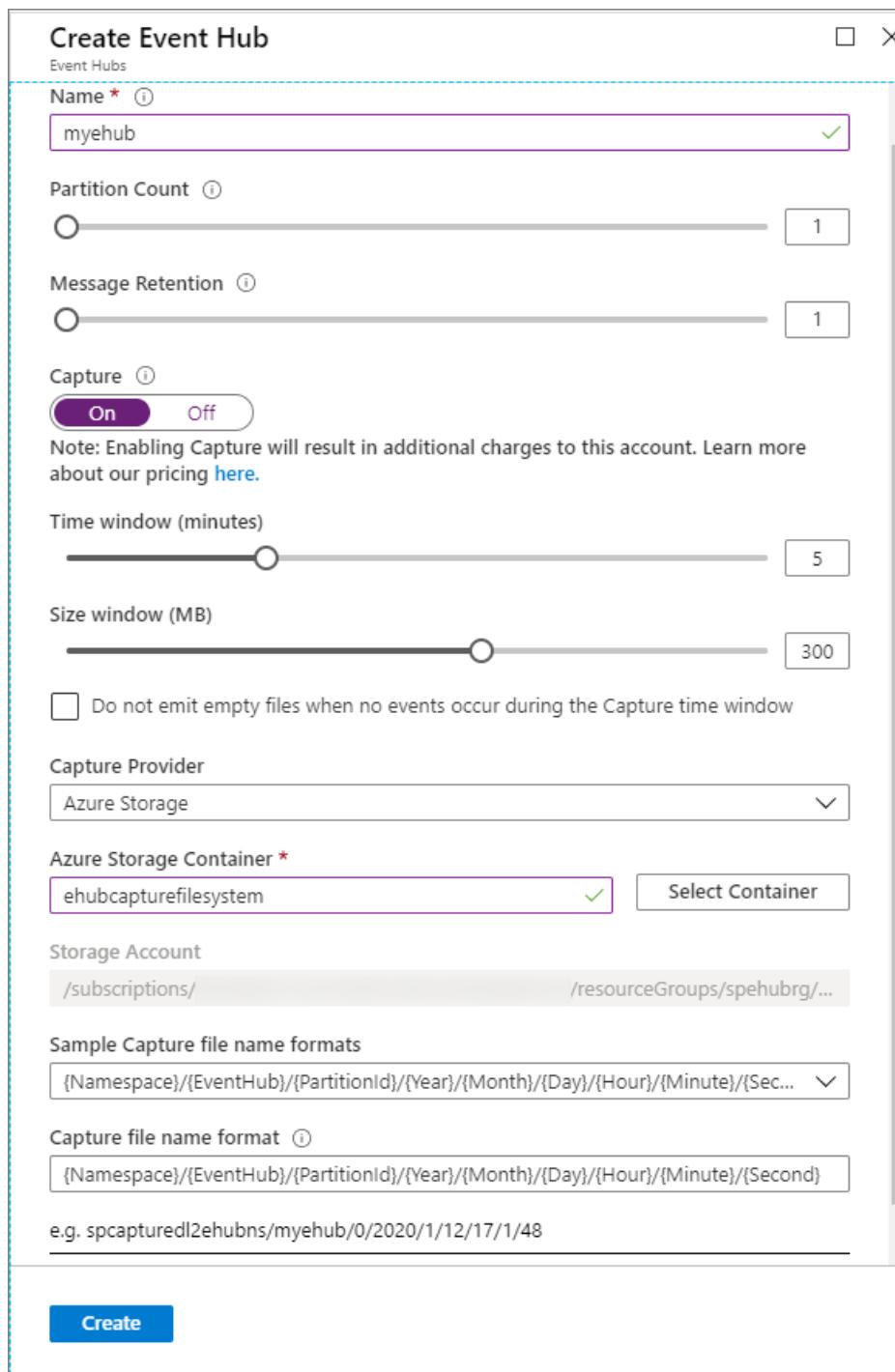
Storage Account /subscriptions/ /resourceGroups/spehubrg/...

Sample Capture file name formats {Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Sec...} ▾

Capture file name format ⓘ {Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

e.g. spcapturedl2ehubs/myehub/0/2020/1/12/17/1/48

**Create**



**NOTE**

The container you create in a Azure Data Lake Storage Gen 2 using this user interface (UI) is shown under **File systems** in **Storage Explorer**. Similarly, the file system you create in a Data Lake Storage Gen 2 account shows up as a container in this UI.

## Capture data to Azure Data Lake Storage Gen 1

To capture data to Azure Data Lake Storage Gen 1, you create a Data Lake Storage Gen 1 account, and an event hub:

### Create an Azure Data Lake Storage Gen 1 account and folders

1. Create a Data Lake Storage account, following the instructions in [Get started with Azure Data Lake Storage Gen 1 using the Azure portal](#).
2. Follow the instructions in the [Assign permissions to Event Hubs](#) section to create a folder within the Data Lake Storage Gen 1 account in which you want to capture the data from Event Hubs, and assign permissions to Event Hubs so that it can write data into your Data Lake Storage Gen 1 account.

### Create an event hub

1. The event hub must be in the same Azure subscription as the Azure Data Lake Storage Gen 1 account you created. Create the event hub, clicking the **On** button under **Capture** in the [Create Event Hub](#) portal page.
2. In the [Create Event Hub](#) portal page, select **Azure Data Lake Store** from the **Capture Provider** box.
3. In **Select Store** next to the **Data Lake Store** drop-down list, specify the Data Lake Storage Gen 1 account you created previously, and in the **Data Lake Path** field, enter the path to the data folder you created.

Dashboard > spehubsns0206 - Overview > spehubsns0206 > Create Event Hub

## Create Event Hub

Event Hubs

\* Name ⓘ  
spehub

Partition Count ⓘ  
2

Message Retention ⓘ  
1

**Capture ⓘ**

On  Off

Note: Enabling Capture will result in additional charges to this account. Learn more about our pricing [here](#).

Time window (minutes)  
5

Size window (MB)  
300

Do not emit empty files when no events occur during the Capture time window

Capture Provider  
Azure Data Lake Store

Learn about setting Data Lake Store access permissions for Event Hubs [here](#).

\* Data Lake Store  
 Select Store  
The value should not be empty.

\* Data Lake Path ⓘ

Sample Capture file name formats  
{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

Capture file name format ⓘ

**Create**

## Add or configure Capture on an existing event hub

You can configure Capture on existing event hubs that are in Event Hubs namespaces. To enable Capture on an existing event hub, or to change your Capture settings, click the namespace to load the overview screen, then click the event hub for which you want to enable or change the Capture setting. Finally, click the **Capture** option on the left side of the open page and then edit the settings, as shown in the following figures:

### Azure Blob Storage

**myehub (spehub0624ns/myehub) | Capture**

Event Hubs Instance

Search (Ctrl+ /) Save changes Discard

Overview Access control (IAM) Diagnose and solve problems

Settings Shared access policies Properties Locks

Entities Consumer groups

Features

**Capture** (highlighted)

Process data

Automation Tasks (preview) Export template

Support + troubleshooting New Support Request

Azure Event Hubs Capture enables you to automatically deliver the streaming data in Event Hubs to an Azure Blob storage or Azure Data Lake Store account of your choice, with the added flexibility of specifying a time or size interval. Setting up Capture is fast, there are no administrative costs to run it, and it scales automatically with Event Hubs throughput units. Event Hubs Capture is the easiest way to load streaming data into Azure, and enables you to focus on data processing rather than on data capture. Learn more about capture [here](#).

**Capture** On Off

Note: Enabling Capture will result in additional charges to this account. Learn more about our pricing [here](#).

Time window (minutes) 5

Size window (MB) 300

Do not emit empty files when no events occur during the Capture time window

Capture Provider Azure Storage Account

Azure Storage Container \* capturedevents Select Container

Storage Account /subscriptions/00000000-0000-0000-000000000000/resourceGroups/spehub0624rg/providers/Microsoft.Storage/storageAccou...

Sample Capture file name formats `{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}`

Capture file name format `{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}`

e.g. spehub0624ns/myehub/0/2021/09/16/15/57/17

## Azure Data Lake Storage Gen 2

It's same as above (for Azure Blob Storage) except that you will be selecting a container from an Azure Data Lake Storage Gen 2 account.

## Azure Data Lake Storage Gen 1

Dashboard > spehubns0206 - Overview > spehubns0206 - Event Hubs > spehub - Capture

**spehub - Capture** Event Hubs Instance

Search (Ctrl+ /) Save changes Discard

Overview Access control (IAM) Diagnose and solve problems

Settings Shared access policies Properties Locks Automation script

Entities Consumer groups

Features

**Capture** (highlighted)

Process data

Support + troubleshooting New support request

Azure Event Hubs Capture enables you to automatically deliver the streaming data in Event Hubs to an Azure Blob storage or Azure Data Lake Store account of your choice, with the added flexibility of specifying a time or size interval. Setting up Capture is fast, there are no administrative costs to run it, and it scales automatically with Event Hubs throughput units. Event Hubs Capture is the easiest way to load streaming data into Azure, and enables you to focus on data processing rather than on data capture. Learn more about capture [here](#).

**Capture** On Off

Note: Enabling Capture will result in additional charges to this account. Learn more about our pricing [here](#).

Time window (minutes) 5

Size window (MB) 300

Do not emit empty files when no events occur during the Capture time window

Capture Provider Azure Data Lake Store

Learn about setting Data Lake Store access permissions for Event Hubs [here](#).

\* Data Lake Store mfdatalake Select Store

\* Data Lake Path /captureddata

Sample Capture file name formats `{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}`

Capture file name format `{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}`

e.g. spehubns0206/spehub/0/2019/1/6/15/47/14

## Next steps

- Learn more about Event Hubs capture by reading the [Event Hubs Capture overview](#).
- You can also configure Event Hubs Capture using Azure Resource Manager templates. For more information, see [Enable Capture using an Azure Resource Manager template](#).
- [Learn how to create an Azure Event Grid subscription with an Event Hubs namespace as its source](#)
- [Get started with Azure Data Lake Store using the Azure portal](#)

# Create a namespace with event hub and enable Capture using a template

11/26/2021 • 6 minutes to read • [Edit Online](#)

This article shows how to use an Azure Resource Manager template that creates an [Event Hubs](#) namespace, with one event hub instance, and also enables the [Capture feature](#) on the event hub. The article describes how to define which resources are deployed, and how to define parameters that are specified when the deployment is executed. You can use this template for your own deployments, or customize it to meet your requirements.

This article also shows how to specify that events are captured into either Azure Storage Blobs or an Azure Data Lake Store, based on the destination you choose.

For more information about creating templates, see [Authoring Azure Resource Manager templates](#). For the JSON syntax and properties to use in a template, see [Microsoft.EventHub resource types](#).

For more information about patterns and practices for Azure Resources naming conventions, see [Azure Resources naming conventions](#).

For the complete templates, click the following GitHub links:

- [Event hub and enable Capture to Storage template](#)
- [Event hub and enable Capture to Azure Data Lake Store template](#)

## NOTE

To check for the latest templates, visit the [Azure Quickstart Templates](#) gallery and search for Event Hubs.

## What will you deploy?

With this template, you deploy an Event Hubs namespace with an event hub, and also enable [Event Hubs Capture](#). Event Hubs Capture enables you to automatically deliver the streaming data in Event Hubs to Azure Blob storage or Azure Data Lake Store, within a specified time or size interval of your choosing. Click the following button to enable Event Hubs Capture into Azure Storage:



Click the following button to enable Event Hubs Capture into Azure Data Lake Store:



## Parameters

With Azure Resource Manager, you define parameters for values you want to specify when the template is deployed. The template includes a section called `Parameters` that contains all the parameter values. You should define a parameter for those values that vary based on the project you are deploying or based on the environment you are deploying to. Do not define parameters for values that always stay the same. Each parameter value is used in the template to define the resources that are deployed.

The template defines the following parameters.

## **eventHubNamespaceName**

The name of the Event Hubs namespace to create.

```
"eventHubNamespaceName":{  
    "type":"string",  
    "metadata":{  
        "description":"Name of the EventHub namespace"  
    }  
}
```

## **eventHubName**

The name of the event hub created in the Event Hubs namespace.

```
"eventHubName":{  
    "type":"string",  
    "metadata":{  
        "description":"Name of the event hub"  
    }  
}
```

## **messageRetentionInDays**

The number of days to retain the messages in the event hub.

```
"messageRetentionInDays":{  
    "type":"int",  
    "defaultValue": 1,  
    "minValue":"1",  
    "maxValue":"7",  
    "metadata":{  
        "description":"How long to retain the data in event hub"  
    }  
}
```

## **partitionCount**

The number of partitions to create in the event hub.

```
"partitionCount":{  
    "type":"int",  
    "defaultValue":2,  
    "minValue":2,  
    "maxValue":32,  
    "metadata":{  
        "description":"Number of partitions chosen"  
    }  
}
```

## **captureEnabled**

Enable Capture on the event hub.

```
"captureEnabled":{  
    "type":"string",  
    "defaultValue":"true",  
    "allowedValues": [  
        "false",  
        "true"],  
    "metadata":{  
        "description":"Enable or disable the Capture for your event hub"  
    }  
}
```

## **captureEncodingFormat**

The encoding format you specify to serialize the event data.

```
"captureEncodingFormat":{  
    "type":"string",  
    "defaultValue":"Avro",  
    "allowedValues": [  
        "Avro"],  
    "metadata":{  
        "description":"The encoding format in which Capture serializes the EventData"  
    }  
}
```

## **captureTime**

The time interval in which Event Hubs Capture starts capturing the data.

```
"captureTime":{  
    "type":"int",  
    "defaultValue":300,  
    "minValue":60,  
    "maxValue":900,  
    "metadata":{  
        "description":"The time window in seconds for the capture"  
    }  
}
```

## **captureSize**

The size interval at which Capture starts capturing the data.

```
"captureSize":{  
    "type":"int",  
    "defaultValue":314572800,  
    "minValue":10485760,  
    "maxValue":524288000,  
    "metadata":{  
        "description":"The size window in bytes for capture"  
    }  
}
```

## **captureNameFormat**

The name format used by Event Hubs Capture to write the Avro files. Note that a Capture name format must contain `{Namespace}` , `{EventHub}` , `{PartitionId}` , `{Year}` , `{Month}` , `{Day}` , `{Hour}` , `{Minute}` , and `{Second}` fields. These can be arranged in any order, with or without delimiters.

```
"captureNameFormat": {  
    "type": "string",  
    "defaultValue": "{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}",  
    "metadata": {  
        "description": "A Capture Name Format must contain {Namespace}, {EventHub}, {PartitionId}, {Year}, {Month}, {Day}, {Hour}, {Minute} and {Second} fields. These can be arranged in any order with or without delimiters. E.g. Prod_{EventHub}/{Namespace}\\"{PartitionId}_{Year}_{Month}_{Day}_{Hour}_{Minute}_{Second}"  
    }  
}
```

## apiVersion

The API version of the template.

```
"apiVersion":{  
    "type":"string",  
    "defaultValue":"2017-04-01",  
    "metadata":{  
        "description":"ApiVersion used by the template"  
    }  
}
```

Use the following parameters if you choose Azure Storage as your destination.

## destinationStorageAccountId

Capture requires an Azure Storage account resource ID to enable capturing to your desired Storage account.

```
"destinationStorageAccountId":{  
    "type": "string",  
    "metadata": {  
        "description": "Your existing Storage account resource ID where you want the blobs be captured"  
    }  
}
```

## blobContainerName

The blob container in which to capture your event data.

```
"blobContainerName":{  
    "type": "string",  
    "metadata": {  
        "description": "Your existing storage container in which you want the blobs captured"  
    }  
}
```

Use the following parameters if you choose Azure Data Lake Store Gen 1 as your destination. You must set permissions on your Data Lake Store path, in which you want to Capture the event. To set permissions, see [Capture data to Azure Data Lake Storage Gen 1](#).

## subscriptionId

Subscription ID for the Event Hubs namespace and Azure Data Lake Store. Both these resources must be under the same subscription ID.

```
"subscriptionId": {  
    "type": "string",  
    "metadata": {  
        "description": "Subscription ID of both Azure Data Lake Store and Event Hubs namespace"  
    }  
}
```

### **dataLakeAccountName**

The Azure Data Lake Store name for the captured events.

```
"dataLakeAccountName": {  
    "type": "string",  
    "metadata": {  
        "description": "Azure Data Lake Store name"  
    }  
}
```

### **dataLakeFolderPath**

The destination folder path for the captured events. This is the folder in your Data Lake Store to which the events will be pushed during the capture operation. To set permissions on this folder, see [Use Azure Data Lake Store to capture data from Event Hubs](#).

```
"dataLakeFolderPath": {  
    "type": "string",  
    "metadata": {  
        "description": "Destination capture folder path"  
    }  
}
```

## Resources to deploy for Azure Storage as destination to captured events

Creates a namespace of type **EventHub**, with one event hub, and also enables Capture to Azure Blob Storage.

```

"resources": [
    {
        "apiVersion": "[variables('ehVersion')]",
        "name": "[parameters('eventHubNamespaceName')]",
        "type": "Microsoft.EventHub/Namespace",
        "location": "[variables('location')]",
        "sku": {
            "name": "Standard",
            "tier": "Standard"
        },
        "resources": [
            {
                "apiVersion": "2017-04-01",
                "name": "[parameters('eventHubNamespaceName')]",
                "type": "Microsoft.EventHub/Namespace",
                "location": "[resourceGroup().location]",
                "sku": {
                    "name": "Standard"
                },
                "properties": {
                    "isAutoInflateEnabled": "true",
                    "maximumThroughputUnits": "7"
                },
                "resources": [
                    {
                        "apiVersion": "2017-04-01",
                        "name": "[parameters('eventHubName')]",
                        "type": "EventHubs",
                        "dependsOn": [
                            "[concat('Microsoft.EventHub/namespaces/', parameters('eventHubNamespaceName'))]"
                        ],
                        "properties": {
                            "messageRetentionInDays": "[parameters('messageRetentionInDays')]",
                            "partitionCount": "[parameters('partitionCount')]",
                            "captureDescription": {
                                "enabled": "true",
                                "skipEmptyArchives": false,
                                "encoding": "[parameters('captureEncodingFormat')]",
                                "intervalInSeconds": "[parameters('captureTime')]",
                                "sizeLimitInBytes": "[parameters('captureSize')]",
                                "destination": {
                                    "name": "EventHubArchive.AzureBlockBlob",
                                    "properties": {
                                        "storageAccountResourceId": "[parameters('destinationStorageAccountId')]",
                                        "blobContainer": "[parameters('blobContainerName')]",
                                        "archiveNameFormat": "[parameters('captureNameFormat')]"
                                    }
                                }
                            }
                        }
                    }
                ]
            }
        ]
    }
]

```

## Resources to deploy for Azure Data Lake Store as destination

Creates a namespace of type **EventHub**, with one event hub, and also enables Capture to Azure Data Lake Store.

```

"resources": [
    {
        "apiVersion": "2017-04-01",
        "name": "[parameters('namespaceName')]",
        "type": "Microsoft.EventHub/Namespaces",
        "location": "[variables('location')]",
        "sku": {
            "name": "Standard",
            "tier": "Standard"
        },
        "resources": [
            {
                "apiVersion": "2017-04-01",
                "name": "[parameters('eventHubName')]",
                "type": "EventHubs",
                "dependsOn": [
                    "[concat('Microsoft.EventHub/namespaces/', parameters('namespaceName'))]"
                ],
                "properties": {
                    "path": "[parameters('eventHubName')]",
                    "captureDescription": {
                        "enabled": "true",
                        "skipEmptyArchives": false,
                        "encoding": "[parameters('archiveEncodingFormat')]",
                        "intervalInSeconds": "[parameters('captureTime')]",
                        "sizeLimitInBytes": "[parameters('captureSize')]",
                        "destination": {
                            "name": "EventHubArchive.AzureDataLake",
                            "properties": {
                                "DataLakeSubscriptionId": "[parameters('subscriptionId')]",
                                "DataLakeAccountName": "[parameters('dataLakeAccountName')]",
                                "DataLakeFolderPath": "[parameters('dataLakeFolderPath')]",
                                "ArchiveNameFormat": "[parameters('captureNameFormat')]"
                            }
                        }
                    }
                }
            }
        ]
    }
]

```

#### **NOTE**

You can enable or disable emitting empty files when no events occur during the Capture window by using the `skipEmptyArchives` property.

## Commands to run deployment

To deploy the resources to Azure, you must be signed in to your Azure account and you must use the Azure Resource Manager module. To learn about using Azure Resource Manager with either Azure PowerShell or Azure CLI, see:

- [Manage Azure resources by using Azure PowerShell](#)
- [Manage Azure resources by using Azure CLI](#).

The following examples assume you already have a resource group in your account with the specified name.

## PowerShell

## NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

Deploy your template to enable Event Hubs Capture into Azure Storage:

```
New-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -TemplateFile  
https://raw.githubusercontent.com/Azure/azure-quickstart-  
templates/master/quickstarts/microsoft.eventhub/eventhubs-create-namespace-and-enable-  
capture/azuredeploy.json
```

Deploy your template to enable Event Hubs Capture into Azure Data Lake Store:

```
New-AzResourceGroupDeployment -ResourceGroupName <resource-group-name> -TemplateFile  
https://raw.githubusercontent.com/Azure/azure-quickstart-  
templates/master/quickstarts/microsoft.eventhub/eventhubs-create-namespace-and-enable-capture-for-  
adls/azuredeploy.json
```

## Azure CLI

Azure Blob Storage as destination:

```
az deployment group create <my-resource-group> <my-deployment-name> --template-uri  
[https://raw.githubusercontent.com/Azure/azure-quickstart-  
templates/master/quickstarts/microsoft.eventhub/eventhubs-create-namespace-and-enable-  
capture/azuredeploy.json[]]
```

Azure Data Lake Store as destination:

```
az deployment group create <my-resource-group> <my-deployment-name> --template-uri  
[https://raw.githubusercontent.com/Azure/azure-quickstart-  
templates/master/quickstarts/microsoft.eventhub/eventhubs-create-namespace-and-enable-capture-for-  
adls/azuredeploy.json[]]
```

## Next steps

You can also configure Event Hubs Capture via the [Azure portal](#). For more information, see [Enable Event Hubs Capture using the Azure portal](#).

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)
- [Create an event hub](#)
- [Event Hubs FAQ](#)

# Capture Event Hubs data in Azure Storage and read it by using Python (azure-eventhub)

11/26/2021 • 6 minutes to read • [Edit Online](#)

You can configure an event hub so that the data that's sent to an event hub is captured in an Azure storage account or Azure Data Lake Storage Gen 1 or Gen 2. This article shows you how to write Python code to send events to an event hub and read the captured data from **Azure Blob storage**. For more information about this feature, see [Event Hubs Capture feature overview](#).

This quickstart uses the [Azure Python SDK](#) to demonstrate the Capture feature. The `sender.py` app sends simulated environmental telemetry to event hubs in JSON format. The event hub is configured to use the Capture feature to write this data to Blob storage in batches. The `capturereader.py` app reads these blobs and creates an append file for each device. The app then writes the data into CSV files.

In this quickstart, you:

- Create an Azure Blob storage account and container in the Azure portal.
- Create an Event Hubs namespace by using the Azure portal.
- Create an event hub with the Capture feature enabled and connect it to your storage account.
- Send data to your event hub by using a Python script.
- Read and process files from Event Hubs Capture by using another Python script.

## Prerequisites

- Python with PIP and the following packages installed. The code in this article has been tested against these versions.
  - Python 3.7
  - azure-eventhub 5.2.0
  - azure-storage-blob 12.6.0
  - avro-python3 1.10.1
- An Azure subscription. If you don't have one, [create a free account](#) before you begin.
- An active Event Hubs namespace and event hub. [Create an Event Hubs namespace and an event hub in the namespace](#). Record the name of the Event Hubs namespace, the name of the event hub, and the primary access key for the namespace. To get the access key, see [Get an Event Hubs connection string](#). The default key name is `RootManageSharedAccessKey`. For this quickstart, you need only the primary key. You don't need the connection string.
- An Azure storage account, a blob container in the storage account, and a connection string to the storage account. If you don't have these items, do the following:
  1. [Create an Azure storage account](#)
  2. [Create a blob container in the storage account](#)
  3. [Get the connection string to the storage account](#)Be sure to record the connection string and container name for later use in this quickstart.
- Enable the Capture feature for the event hub. To do so, follow the instructions in [Enable Event Hubs Capture using the Azure portal](#). Select the storage account and the blob container you created in the preceding step. You can also enable the feature when you create an event hub.

# Create a Python script to send events to your event hub

In this section, you create a Python script that sends 200 events (10 devices \* 20 events) to an event hub. These events are a sample environmental reading that's sent in JSON format.

1. Open your favorite Python editor, such as [Visual Studio Code](#).
2. Create a script called *sender.py*.
3. Paste the following code into *sender.py*.

```
import time
import os
import uuid
import datetime
import random
import json

from azure.eventhub import EventHubProducerClient, EventData

# This script simulates the production of events for 10 devices.
devices = []
for x in range(0, 10):
    devices.append(str(uuid.uuid4()))

# Create a producer client to produce and publish events to the event hub.
producer = EventHubProducerClient.from_connection_string(conn_str="EVENT HUBS NAMESPCE CONNECTION STRING", eventhub_name="EVENT HUB NAME")

for y in range(0,20):    # For each device, produce 20 events.
    event_data_batch = producer.create_batch() # Create a batch. You will add events to the batch later.
    for dev in devices:
        # Create a dummy reading.
        reading = {'id': dev, 'timestamp': str(datetime.datetime.utcnow()), 'uv': random.random(), 'temperature': random.randint(70, 100), 'humidity': random.randint(70, 100)}
        s = json.dumps(reading) # Convert the reading into a JSON string.
        event_data_batch.add(EventData(s)) # Add event data to the batch.
    producer.send_batch(event_data_batch) # Send the batch of events to the event hub.

# Close the producer.
producer.close()
```

4. Replace the following values in the scripts:

- Replace `EVENT HUBS NAMESPCE CONNECTION STRING` with the connection string for your Event Hubs namespace.
- Replace `EVENT HUB NAME` with the name of your event hub.

5. Run the script to send events to the event hub.
6. In the Azure portal, you can verify that the event hub has received the messages. Switch to **Messages** view in the **Metrics** section. Refresh the page to update the chart. It might take a few seconds for the page to display that the messages have been received.

Resource group (change)  
spehubrg1204

Status  
Active

Location  
West US 2

Subscription (change)  
<Your Azure subscription name>

Subscription ID  
<Your Azure subscription ID>

Host name  
spehubcapturesns.servicebus.windows.net

Tags (change)  
Click here to add tags

NAMESPACE CONTENTS  
1 EVENT HUB    KAFKA SURFACE  
ENABLED

Show metrics:  
Requests    **Messages**    Throughput

For the last:  
1 hour    6 hours    12 hours    1 day    7 days    30 days

Metrics chart:

Time	Value
1:30 PM	Incoming Messages (Sun spehubcaptures) 200
1:45 PM	Outgoing Messages (Sun spehubcaptures) 0
2 PM	Captured Messages (Sun spehubcaptures) 200
2:15 PM	Capture Backlog. (Sum spehubcaptures) 0

## Create a Python script to read your Capture files

In this example, the captured data is stored in Azure Blob storage. The script in this section reads the captured data files from your Azure storage account and generates CSV files for you to easily open and view. You will see 10 files in the current working directory of the application. These files will contain the environmental readings for the 10 devices.

1. In your Python editor, create a script called *capturereader.py*. This script reads the captured files and creates a file for each device to write the data only for that device.
2. Paste the following code into *capturereader.py*.

```

import os
import string
import json
import uuid
import avro.schema

from azure.storage.blob import ContainerClient, BlobClient
from avro.datafile import DataFileReader, DataFileWriter
from avro.io import DatumReader, DatumWriter


def processBlob2(filename):
    reader = DataFileReader(open(filename, 'rb'), DatumReader())
    dict = {}
    for reading in reader:
        parsed_json = json.loads(reading["Body"])
        if not 'id' in parsed_json:
            return
        if not parsed_json['id'] in dict:
            list = []
            dict[parsed_json['id']] = list
        else:
            list = dict[parsed_json['id']]
            list.append(parsed_json)
    reader.close()
    for device in dict.keys():
        filename = os.getcwd() + '\\\\' + str(device) + '.csv'
        deviceFile = open(filename, "a")
        for r in dict[device]:
            deviceFile.write(", ".join([str(r[x]) for x in r.keys()])+'\n')

def startProcessing():
    print('Processor started using path: ' + os.getcwd())
    # Create a blob container client.
    container = ContainerClient.from_connection_string("AZURE STORAGE CONNECTION STRING",
    container_name="BLOB CONTAINER NAME")
    blob_list = container.list_blobs() # List all the blobs in the container.
    for blob in blob_list:
        # Content_length == 508 is an empty file, so process only content_length > 508 (skip empty
        files).
        if blob.size > 508:
            print('Downloaded a non empty blob: ' + blob.name)
            # Create a blob client for the blob.
            blob_client = ContainerClient.get_blob_client(container, blob=blob.name)
            # Construct a file name based on the blob name.
            cleanName = str.replace(blob.name, '/', '_')
            cleanName = os.getcwd() + '\\\\' + cleanName
            with open(cleanName, "wb+") as my_file: # Open the file to write. Create it if it doesn't
            exist.
                my_file.write(blob_client.download_blob().readall()) # Write blob contents into the
            file.
            processBlob2(cleanName) # Convert the file into a CSV file.
            os.remove(cleanName) # Remove the original downloaded file.
            # Delete the blob from the container after it's read.
            container.delete_blob(blob.name)

    startProcessing()

```

- Replace `AZURE STORAGE CONNECTION STRING` with the connection string for your Azure storage account. The name of the container you created in this quickstart is *capture*. If you used a different name for the container, replace *capture* with the name of the container in the storage account.

## Run the scripts

- Open a command prompt that has Python in its path, and then run these commands to install Python

prerequisite packages:

```
pip install azure-storage-blob  
pip install azure-eventhub  
pip install avro-python3
```

**NOTE**

The code in this article has been tested against these versions.

- Python 3.7
- azure-eventhub 5.2.0
- azure-storage-blob 12.6.0
- avro-python3 1.10.1

2. Change your directory to the directory where you saved *sender.py* and *capturereader.py*, and run this command:

```
python sender.py
```

This command starts a new Python process to run the sender.

3. Wait a few minutes for the capture to run, and then enter the following command in your original command window:

```
python capturereader.py
```

This capture processor uses the local directory to download all the blobs from the storage account and container. It processes any that are not empty, and it writes the results as CSV files into the local directory.

## Next steps

Check out [Python samples on GitHub](#).

# Quickstart: Data streaming with Event Hubs using the Kafka protocol

11/26/2021 • 2 minutes to read • [Edit Online](#)

This quickstart shows how to stream into Event Hubs without changing your protocol clients or running your own clusters. You learn how to use your producers and consumers to talk to Event Hubs with just a configuration change in your applications.

## NOTE

This sample is available on [GitHub](#)

## Prerequisites

To complete this quickstart, make sure you have the following prerequisites:

- Read through the [Event Hubs for Apache Kafka](#) article.
- An Azure subscription. If you do not have one, create a [free account](#) before you begin.
- [Java Development Kit \(JDK\) 1.7+](#).
- [Download and install](#) a Maven binary archive.
- [Git](#)

## Create an Event Hubs namespace

When you create an Event Hubs namespace, the Kafka endpoint for the namespace is automatically enabled. You can stream events from your applications that use the Kafka protocol into event hubs. Follow step-by-step instructions in the [Create an event hub using Azure portal](#) to create an Event Hubs namespace. If you are using a dedicated cluster, see [Create a namespace and event hub in a dedicated cluster](#).

## NOTE

Event Hubs for Kafka isn't supported in the **basic** tier.

## Send and receive messages with Kafka in Event Hubs

1. Clone the [Azure Event Hubs for Kafka repository](#).
2. Navigate to `azure-event-hubs-for-kafka/quickstart/java/producer`.
3. Update the configuration details for the producer in `src/main/resources/producer.config` as follows:

### TLS/SSL:

```
bootstrap.servers=NAMESPACE.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

## IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXX"
```

### OAuth:

```
bootstrap.servers=NAMESPACENAME.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;
sasl.login.callback.handler.class=CustomAuthenticateCallbackHandler;
```

You can find the source code for the sample handler class `CustomAuthenticateCallbackHandler` on GitHub [here](#).

4. Run the producer code and stream events into Event Hubs:

```
mvn clean package
mvn exec:java -Dexec.mainClass="TestProducer"
```

5. Navigate to [azure-event-hubs-for-kafka/quickstart/java/consumer](#).

6. Update the configuration details for the consumer in `src/main/resources/consumer.config` as follows:

#### TLS/SSL:

```
bootstrap.servers=NAMESPACENAME.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXX"
```

#### OAuth:

```
bootstrap.servers=NAMESPACENAME.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;
sasl.login.callback.handler.class=CustomAuthenticateCallbackHandler;
```

You can find the source code for the sample handler class `CustomAuthenticateCallbackHandler` on GitHub [here](#).

You can find all the OAuth samples for Event Hubs for Kafka [here](#).

7. Run the consumer code and process events from event hub using your Kafka clients:

```
mvn clean package
mvn exec:java -Dexec.mainClass="TestConsumer"
```

If your Event Hubs Kafka cluster has events, you now start receiving them from the consumer.

## Next steps

In this article, you learned how to stream into Event Hubs without changing your protocol clients or running your own clusters. To learn more, see [Apache Kafka developer guide for Azure Event Hubs](#).

# Create an Azure Event Hubs schema registry

11/26/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to create a schema group with schemas in a schema registry hosted by Azure Event Hubs. For an overview of the Schema Registry feature of Azure Event Hubs, see [Azure Schema Registry in Event Hubs](#).

## NOTE

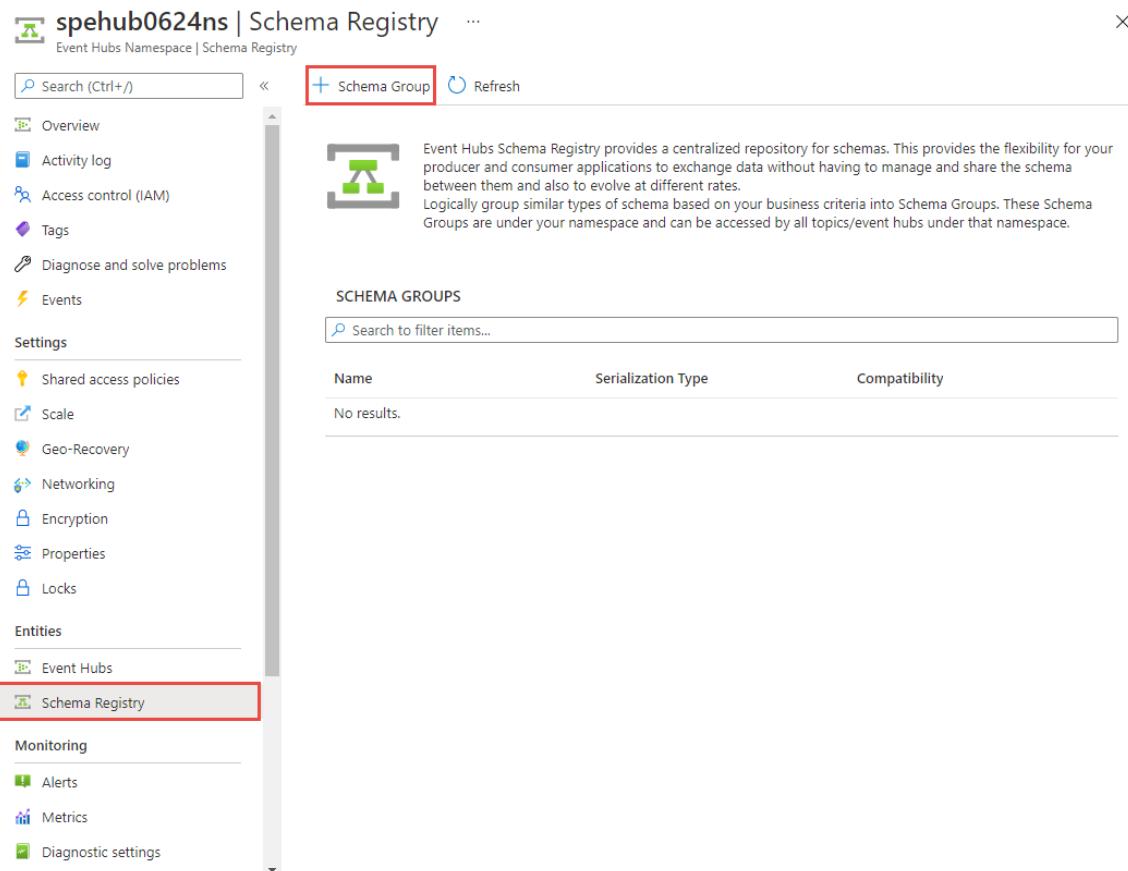
- The feature isn't available in the **basic** tier.
- If the event hub is in a **virtual network**, you won't be able to create schemas in the Azure portal unless you access the portal from a VM in the same virtual network.

## Prerequisites

[Create an Event Hubs namespace](#). You can also use an existing namespace.

## Create a schema group

1. Navigate to the [Event Hubs Namespace](#) page.
2. Select **Schema Registry** on the left menu. To create a schema group, select **+ Schema Group** on the toolbar.



3. On the **Create Schema Group** page, do these steps:

- a. Enter a **name** for the schema group.

- b. For **Serialization type**, pick a serialization format that applies to all schemas in the schema group. Currently, **Avro** is the only type supported, so select **Avro**.
- c. Select a **compatibility mode** for all schemas in the group. For **Avro**, forward and backward compatibility modes are supported.
- d. Then, select **Create** to create the schema group.

**Create Schema Group** ×

Schema

Name *	<input type="text" value="contososchemagroup"/> <span style="color: green;">✓</span>
Serialization Type *	<input type="text" value="Avro"/>
Compatibility *	<input type="text" value="None"/>

**Create**

4. Select the name of the **schema group** in the list of schema groups.

**spehub0624ns | Schema Registry** ×

Event Hubs Namespace | Schema Registry

Search (Ctrl+/) + Schema Group Refresh

Overview Activity log Access control (IAM) Tags Diagnose and solve problems Events

Shared access policies Scale Geo-Recovery Networking Encryption Properties Locks

Event Hubs Schema Registry provides a centralized repository for schemas. This provides the flexibility for your producer and consumer applications to exchange data without having to manage and share the schema between them and also to evolve at different rates. Logically group similar types of schema based on your business criteria into Schema Groups. These Schema Groups are under your namespace and can be accessed by all topics/event hubs under that namespace.

**SCHEMA GROUPS**

Name	Serialization Type	Compatibility
contososchemagroup	Avro	None

5. You see the **Schema Group** page for the group.

The screenshot shows the Azure portal interface for managing a schema group. The left sidebar includes links for Overview, Diagnose and solve problems, Settings, Locks, Automation, Tasks (preview), Export template, Support + troubleshooting, and New support request. The main content area has tabs for Schema, Delete, Refresh, and Essentials. The Essentials tab shows details such as Resource group (spehub0624rg), Location (East US), Subscription (Contoso subscription), Subscription ID (00000000-0000-0000-000000000000), and Namespace (spehub0624ns). A note about Schemas defines the contract between producers and consumers. Below this is a 'SCHEMAS' section with a table header (Name, ID, Latest Version) and a message stating 'No results.'

## Add a schema to the schema group

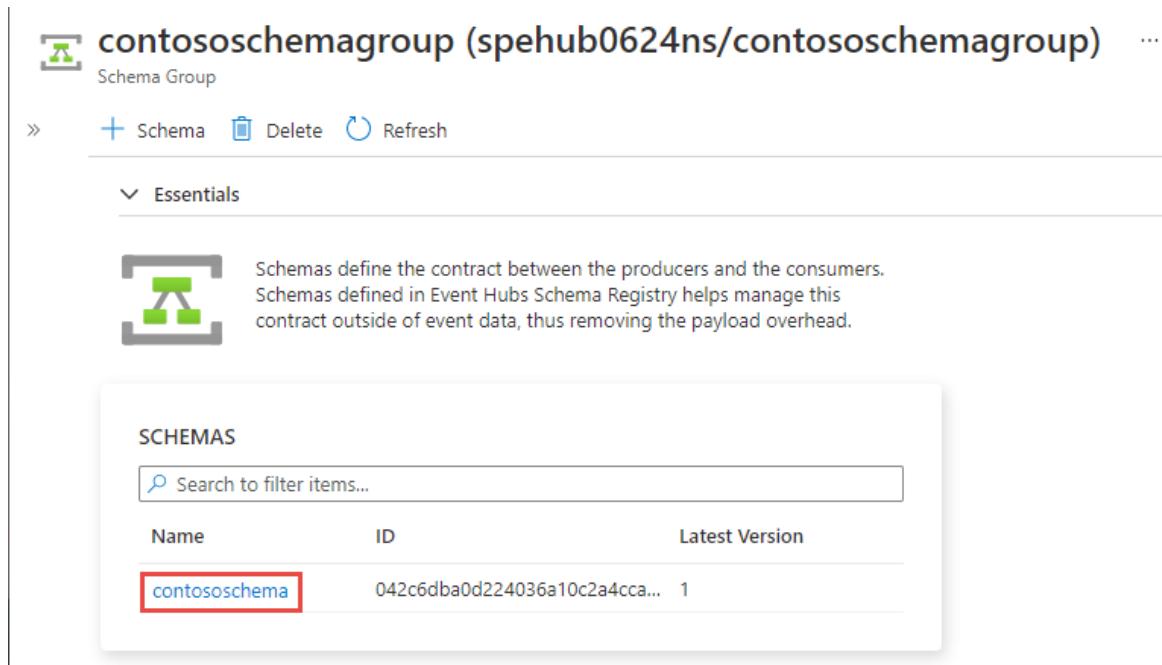
In this section, you add a schema to the schema group using the Azure portal.

1. On the **Schema Group** page, select **+ Schema** on the toolbar.
2. On the **Create Schema** page, do these steps:
  - a. For **Name**, enter **orderschema**.
  - b. Enter the following **schema** into the text box. You can also select file with the schema.

```
{  
  "namespace": "com.azure.schemaregistry.samples",  
  "type": "record",  
  "name": "Order",  
  "fields": [  
    {  
      "name": "id",  
      "type": "string"  
    },  
    {  
      "name": "amount",  
      "type": "double"  
    }  
  ]  
}
```

- c. Select **Create**.

3. Select the **schema** from the list of schemas.



contososchemagroup (spehub0624ns/contososchemagroup) ...

Schema Group

» + Schema Delete Refresh

▼ Essentials

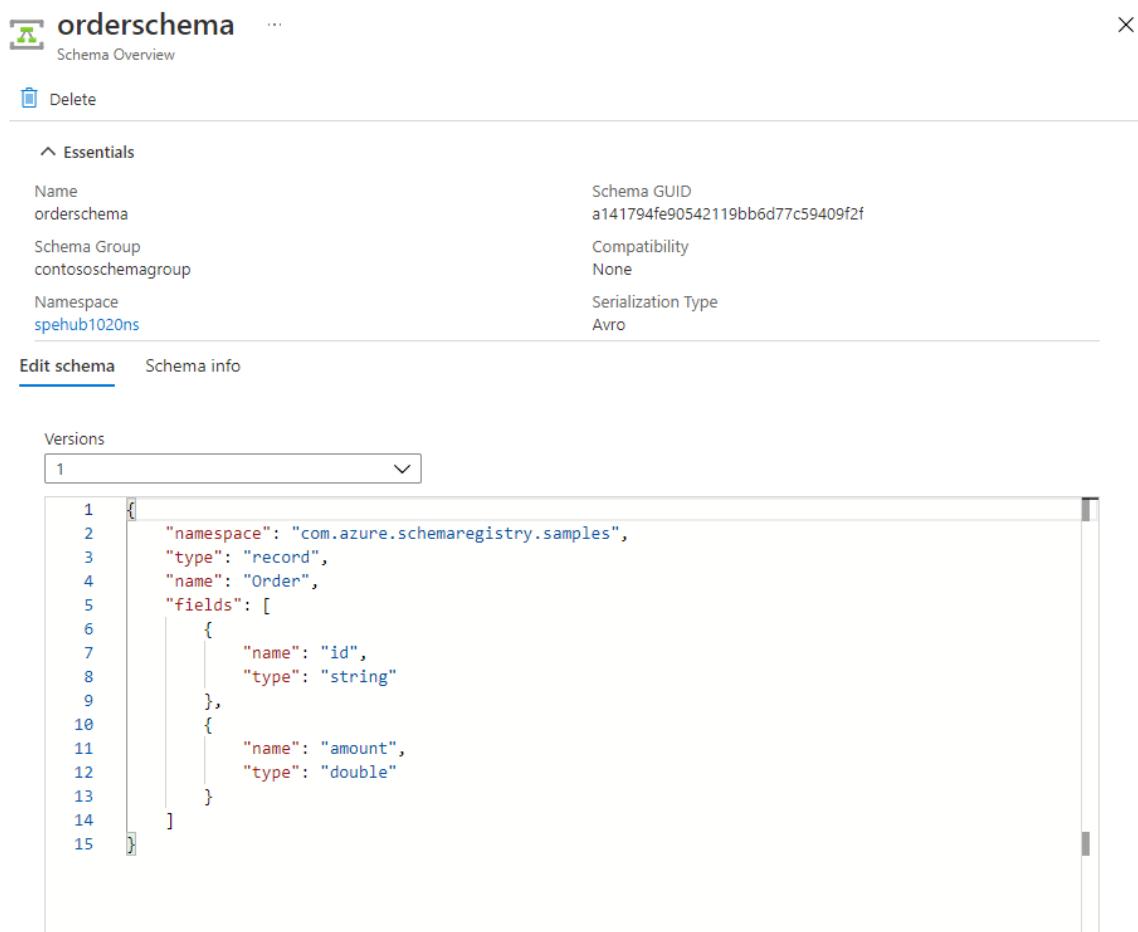
Schemas define the contract between the producers and the consumers. Schemas defined in Event Hubs Schema Registry helps manage this contract outside of event data, thus removing the payload overhead.

SCHEMAS

Search to filter items...

Name	ID	Latest Version
contososchema	042c6dba0d224036a10c2a4cca...	1

4. You see the Schema Overview page for the schema.



orderschema ...

Schema Overview

Delete

Essentials

Name	orderschema	Schema GUID	a141794fe90542119bb6d77c59409f2f
Schema Group	contososchemagroup	Compatibility	None
Namespace	spehub1020ns	Serialization Type	Avro

Edit schema Schema info

Versions

1

```
1  [
2   "namespace": "com.azure.schemaregistry.samples",
3   "type": "record",
4   "name": "Order",
5   "fields": [
6     {
7       "name": "id",
8       "type": "string"
9     },
10    {
11      "name": "amount",
12      "type": "double"
13    }
14  ]
15 }
```

5. If there are multiple versions of a schema, you see them in the Versions drop-down list. Select a version to switch to that version schema.

## Create a new version of schema

1. Update the schema in the text box, and select Validate. In the following example, a new field `description` has been added to the schema.

orderschema ...

Schema Overview

Delete

Essentials

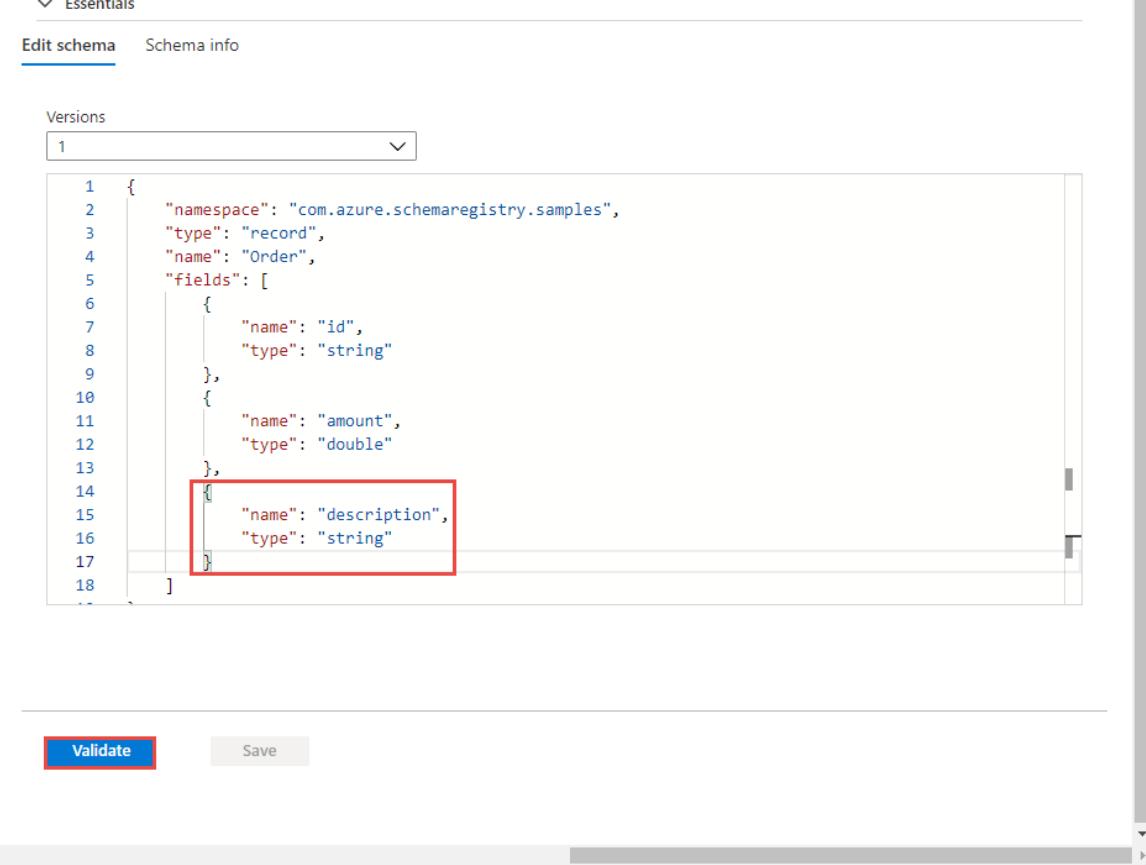
Edit schema Schema info

Versions

1

```
1 {  
2   "namespace": "com.azure.schemaregistry.samples",  
3   "type": "record",  
4   "name": "Order",  
5   "fields": [  
6     {  
7       "name": "id",  
8       "type": "string"  
9     },  
10    {  
11      "name": "amount",  
12      "type": "double"  
13    },  
14    {  
15      "name": "description",  
16      "type": "string"  
17    }  
18  ]
```

Validate Save



2. Review validation status and changes, and select **Save**.

orderschema ...

Schema Overview

Delete

Essentials

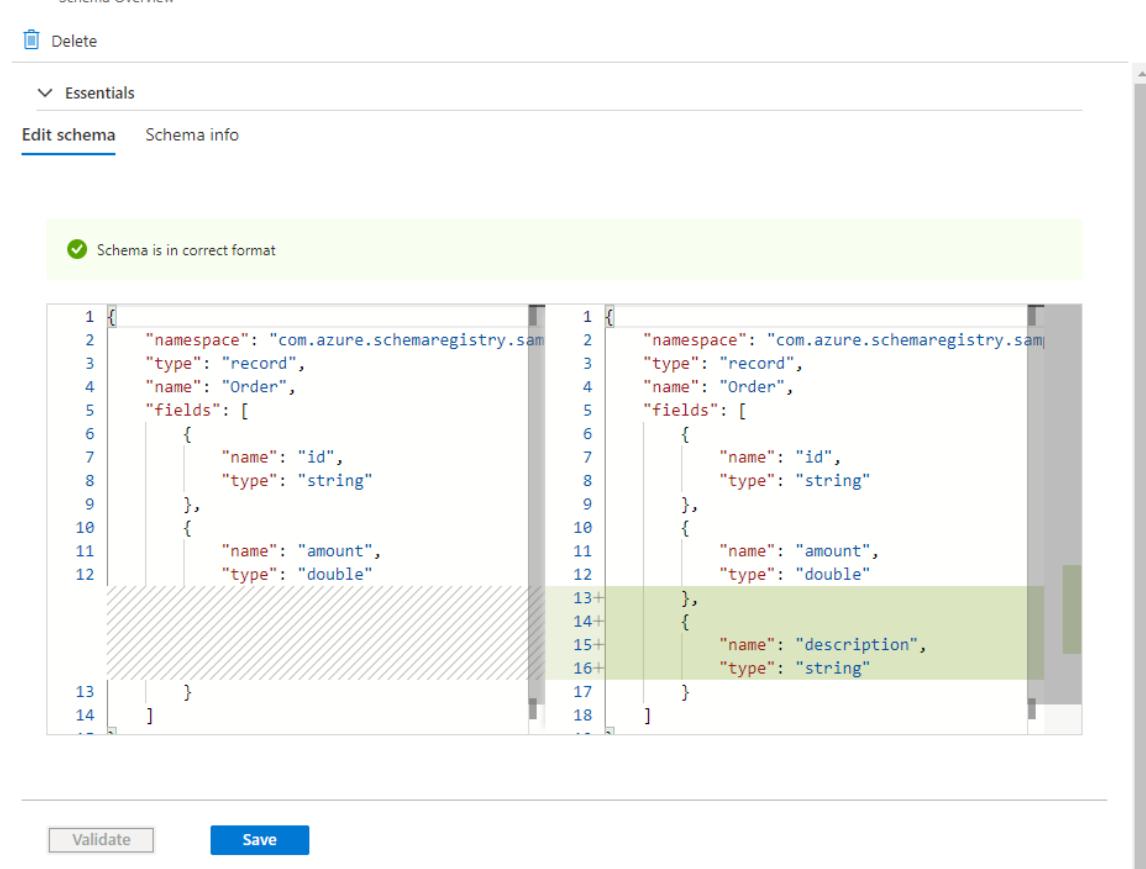
Edit schema Schema info

✓ Schema is in correct format

```
1 {  
2   "namespace": "com.azure.schemaregistry.sam  
3   "type": "record",  
4   "name": "Order",  
5   "fields": [  
6     {  
7       "name": "id",  
8       "type": "string"  
9     },  
10    {  
11      "name": "amount",  
12      "type": "double"  
13    }  
14  ]
```

```
1 {  
2   "namespace": "com.azure.schemaregistry.sam  
3   "type": "record",  
4   "name": "Order",  
5   "fields": [  
6     {  
7       "name": "id",  
8       "type": "string"  
9     },  
10    {  
11      "name": "amount",  
12      "type": "double"  
13    },  
14    {  
15      "name": "description",  
16      "type": "string"  
17    }  
18  ]
```

Validate Save



3. You see that **2** is selected for the version on the Schema Overview page.

The screenshot shows the 'Schema Overview' page for the 'orderschema' in the 'contososchemagroup'. The 'Edit schema' tab is active. A dropdown menu labeled 'Versions' has '2' selected, indicated by a red box. The main area displays the schema code:

```
1 {  
2   "namespace": "com.azure.schemaregistry.samples",  
3   "type": "record",  
4   "name": "Order",  
5   "fields": [  
6     {  
7       "name": "id",  
8       "type": "string"  
9     },  
10    {  
11      "name": "amount",  
12      "type": "double"  
13    },  
14    {  
15      "name": "description",  
16      "type": "string"  
17    }  
18  ]  
}
```

At the bottom, there are 'Validate' and 'Save' buttons. A success message box is visible in the top right corner: 'Saving new version of schema' and 'Successfully saved schema: orderschema'.

4. Select **1** to see the version 1 of the schema.

## Next steps

For more information about schema registry, see [Azure Schema Registry in Event Hubs](#).

# Validate schema when sending and receiving events - AMQP and .NET

11/26/2021 • 9 minutes to read • [Edit Online](#)

This quickstart shows how to send events to and receive events from an event hub with schema validation using the `Azure.Messaging.EventHubs` .NET library.

## Prerequisites

If you're new to Azure Event Hubs, see [Event Hubs overview](#) before you do this quickstart.

To complete this quickstart, you need the following prerequisites:

- Follow instructions from the quickstart: [Create an Event Hubs namespace and an event hub](#).
- Follow instructions from [Get the connection string](#) to get a connection string to your Event Hubs namespace. Note down the following settings that you'll use in the current quickstart:
  - Connection string for the Event Hubs namespace
  - Name of the event hub
- Complete the [.NET quickstart](#) to become familiar with sending events to and receiving events from event hubs using .NET. If you have already done the .NET quickstart before, you can skip this step.
- Follow instructions from [Create schemas using Schema Registry](#) to create a schema group and a schema. When creating a schema, follow instructions from the [Create a schema](#) in the current quickstart article.
- **Microsoft Visual Studio 2019.** The Azure Event Hubs client library makes use of new features that were introduced in C# 8.0. You can still use the library with previous C# language versions, but the new syntax won't be available. To make use of the full syntax, we recommended that you compile with the [.NET Core SDK 3.0 or higher](#) and [language version](#) set to `latest`. If you're using Visual Studio, versions before Visual Studio 2019 aren't compatible with the tools needed to build C# 8.0 projects. Visual Studio 2019, including the free Community edition, can be downloaded [here](#).

## Create a schema

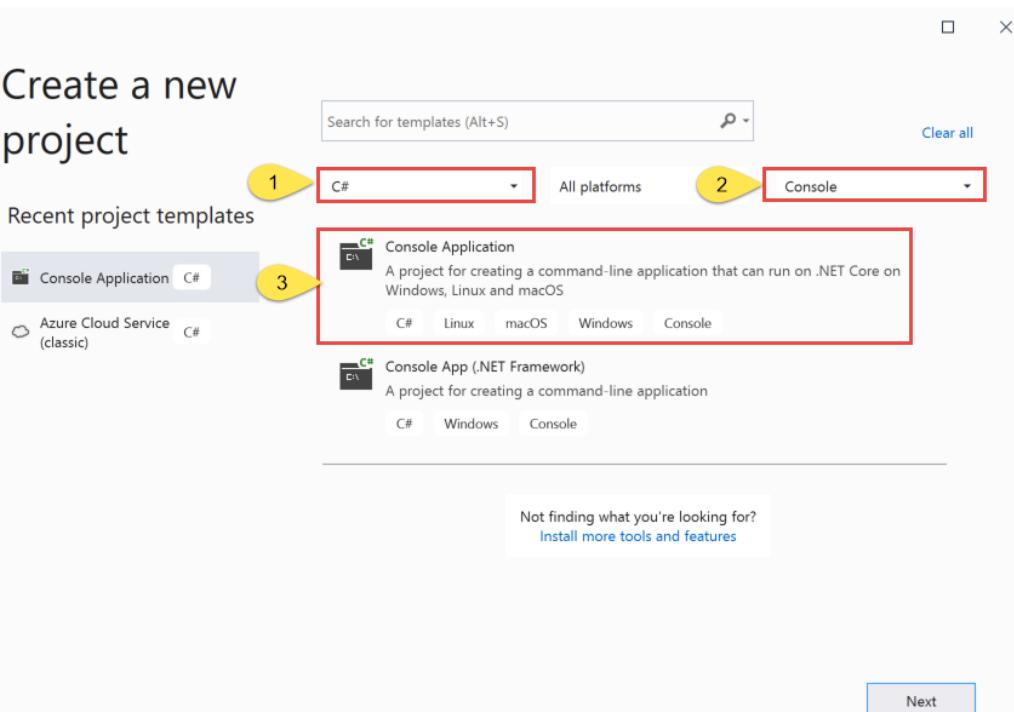
1. Create a schema group named `contoso-sg` using the Schema Registry portal. Use Avro as the serialization type and `None` for the compatibility mode.
2. In that schema group, create a new Avro schema with schema name:  
`Microsoft.Azure.Data.SchemaRegistry.example.Order` using the following schema content.

```
{
  "namespace": "Microsoft.Azure.Data.SchemaRegistry.example",
  "type": "record",
  "name": "Order",
  "fields": [
    {
      "name": "id",
      "type": "string"
    },
    {
      "name": "amount",
      "type": "double"
    },
    {
      "name": "description",
      "type": "string"
    }
  ]
}
```

## Produce events to event hubs with schema validation

### Create console application for event producer

1. Start Visual Studio 2019.
2. Select **Create a new project**.
3. On the **Create a new project** dialog box, do the following steps: If you don't see this dialog box, select **File** on the menu, select **New**, and then select **Project**.
  - a. Select **C#** for the programming language.
  - b. Select **Console** for the type of the application.
  - c. Select **Console Application** from the results list.
  - d. Then, select **Next**.



4. Enter **OrderProducer** for the project name, **SRQuickStart** for the solution name, and then select **OK** to

create the project.

## Add the Event Hubs NuGet package

1. Select Tools > NuGet Package Manager > Package Manager Console from the menu.
2. Run the following command to install the Azure.Messaging.EventHubs NuGet package:

```
Install-Package Azure.Messaging.EventHubs
Install-Package Azure.Identity
Install-Package Microsoft.Azure.Data.SchemaRegistry.ApacheAvro -Version 1.0.0-beta.2
Install-Package Azure.ResourceManager.Compute -Version 1.0.0-beta.1
```

3. Authenticate producer applications to connect to Azure via Visual Studio as shown [here](#).

## Code generation using the Avro schema

1. You can use the same schema content and create the Avro schema file `Order.avsc` file inside the OrderProducer project.
2. Then you can use this schema file to generate code for .NET. You can use any external code generation tool such as [avrogen](#) for code generation. (For example you can run `avrogen -s .\Order.avsc` to generate code).
3. Once you generate code, you should have the corresponding C# types available inside your project. For the above Avro schema, it generates the C# types in `Microsoft.Azure.Data.SchemaRegistry.example` namespace.

## Write code to serialize and send events to the event hub

1. Add the following `using` statements to the top of the `Program.cs` file:

```
using System;
using System.IO;
using System.Threading;
using Azure.Data.SchemaRegistry;
using Azure.Identity;
using Microsoft.Azure.Data.SchemaRegistry.ApacheAvro;
using Azure.Messaging.EventHubs;
using Azure.Messaging.EventHubs.Producer;
using System.Threading.Tasks;
```

2. Also you can import the generated types related to `order` schema as shown below.

```
using Microsoft.Azure.Data.SchemaRegistry.example;
```

3. Add constants to the `Program` class for the Event Hubs connection string and the event hub name.

```
// connection string to the Event Hubs namespace
private const string connectionString = "<EVENT HUBS NAMESPACE - CONNECTION STRING>";

// name of the event hub
private const string eventHubName = "<EVENT HUB NAME>";

// Schema Registry endpoint
private const string schemaRegistryEndpoint = "<EVENT HUBS NAMESPACE>.servicebus.windows.net";

// name of the consumer group
private const string schemaGroup = "<SCHEMA GROUP>";
```

**NOTE**

Replace placeholder values with the connection string to your namespace, the name of the event hub, and schema group.

4. Add the following static property to the `Program` class. See the code comments.

```
// The Event Hubs client types are safe to cache and use as a singleton for the lifetime  
// of the application, which is best practice when events are being published or read regularly.  
static EventHubProducerClient producerClient;
```

5. Replace the `Main` method with the following `async Main` method. See the code comments for details.

```
static async Task Main()  
{  
    // Create a producer client that you can use to send events to an event hub  
    producerClient = new EventHubProducerClient(connectionString, eventHubName);  
  
    // Create a schema registry client that you can use to serialize and validate data.  
    var schemaRegistryClient = new SchemaRegistryClient(endpoint: schemaRegistryEndpoint,  
    credential: new DefaultAzureCredential());  
  
    // Create a batch of events  
    using EventDataBatch eventBatch = await producerClient.CreateBatchAsync();  
  
    // Create a new order object using the generated type/class 'Order'.  
    var sampleOrder = new Order { id = "12345", amount = 55.99, description = "This is a sample  
order." };  
  
    using var memoryStream = new MemoryStream();  
    // Create an Avro object serializer using the Schema Registry client object.  
    var producerSerializer = new SchemaRegistryAvroObjectSerializer(schemaRegistryClient,  
    schemaGroup, new SchemaRegistryAvroObjectSerializerOptions { AutoRegisterSchemas = true });  
  
    // Serialize events data to the memory stream object.  
    producerSerializer.Serialize(memoryStream, sampleOrder, typeof(Order),  
    CancellationToken.None);  
  
    byte[] _memoryStreamBytes;  
    _memoryStreamBytes = memoryStream.ToArray();  
  
    // Create event data with serialized data and add it to an event batch.  
    eventBatch.TryAdd(new EventData(_memoryStreamBytes));  
  
    // Send serialized event data to event hub.  
    await producerClient.SendAsync(eventBatch);  
    Console.WriteLine("A batch of 1 order has been published.");  
}
```

6. Build the project, and ensure that there are no errors.

7. Run the program and wait for the confirmation message.

```
A batch of 1 order has been published.
```

8. In the Azure portal, you can verify that the event hub has received the events. Switch to **Messages** view in the **Metrics** section. Refresh the page to update the chart. It may take a few seconds for it to show that the messages have been received.

Name	Status	message retention	partition count
spehub	Active	1	1

## Consume events from event hubs with schema validation

This section shows how to write a .NET Core console application that receives events from an event hub and use schema registry to de-serialize event data.

### Create consumer application

1. In the Solution Explorer window, right-click the SRQuickStart solution, point to **Add**, and select **New Project**.
2. Select **Console application**, and select **Next**.
3. Enter **OrderConsumer** for the **Project name**, and select **Create**.
4. In the **Solution Explorer** window, right-click **OrderConsumer**, and select **Set as a Startup Project**.

### Add the Event Hubs NuGet package

1. Select **Tools > NuGet Package Manager > Package Manager Console** from the menu.
2. In the **Package Manager Console** window, confirm that **OrderConsumer** is selected for the **Default project**. If not, use the drop-down list to select **OrderConsumer**.
3. Run the following command to install the required NuGet package:

```
Install-Package Azure.Messaging.EventHubs
Install-Package Azure.Messaging.EventHubs.Processor
Install-Package Azure.Identity
Install-Package Microsoft.Azure.Data.SchemaRegistry.ApacheAvro -Version 1.0.0-beta.2
Install-Package Azure.ResourceManager.Compute -Version 1.0.0-beta.1
```

4. Authenticate producer applications to connect to Azure via Visual Studio as shown [here](#).

### Code generation using the Avro schema

1. You can use the same schema content and create the Avro schema file `order.avsc` file inside the `OrderProducer` project.
2. Then you can use this schema file to generate code for .NET. For this, you can use any external code

generation tool such as [avrogen](#). (For example you can run `avrogen -s .\Order.avsc` to generate code).

- Once you generate code, you should have the corresponding C# types available inside your project. For the above Avro schema, it generates the C# types in `Microsoft.Azure.Data.SchemaRegistry.example` namespace.

## Write code to receive events and deserialize them using Schema Registry

- Add the following `using` statements to the top of the `Program.cs` file:

```
using System;
using System.IO;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using Azure.Data.SchemaRegistry;
using Azure.Identity;
using Microsoft.Azure.Data.SchemaRegistry.ApacheAvro;
using Azure.Storage.Blobs;
using Azure.Messaging.EventHubs;
using Azure.Messaging.EventHubs.Consumer;
using Azure.Messaging.EventHubs.Processor;
```

- Also you can import the generated types related to `order` schema as shown below.

```
using Microsoft.Azure.Data.SchemaRegistry.example;
```

- Add constants to the `Program` class for the Event Hubs connection string and the event hub name.

```
// connection string to the Event Hubs namespace
private const string connectionString = "<EVENT HUBS NAMESPACE - CONNECTION STRING>";

// name of the event hub
private const string eventHubName = "<EVENT HUB NAME>";

private const string blobStorageConnectionString = "<AZURE STORAGE CONNECTION STRING>";

private const string blobContainerName = "<BLOB CONTAINER NAME>";

// Schema Registry endpoint
private const string schemaRegistryEndpoint = "<EVENT HUBS NAMESPACE>.servicebus.windows.net";

// name of the consumer group
private const string schemaGroup = "<SCHEMA GROUP>";
```

- Add the following static properties to the `Program` class.

```
static BlobContainerClient storageClient;

// The Event Hubs client types are safe to cache and use as a singleton for the lifetime
// of the application, which is best practice when events are being published or read regularly.
static EventProcessorClient processor;
```

- Replace the `Main` method with the following `async Main` method. See the code comments for details.

```

static async Task Main()
{
    // Read from the default consumer group: $Default
    string consumerGroup = EventHubConsumerClient.DefaultConsumerGroupName;

    // Create a blob container client that the event processor will use
    storageClient = new BlobContainerClient(blobStorageConnectionString, blobContainerName);

    // Create an event processor client to process events in the event hub
    processor = new EventProcessorClient(storageClient, consumerGroup,
    ehubNamespaceConnectionString, eventHubName);

    // Register handlers for processing events and handling errors
    processor.ProcessEventAsync += ProcessEventHandler;
    processor.ProcessErrorAsync += ProcessErrorHandler;

    // Start the processing
    await processor.StartProcessingAsync();

    // Wait for 30 seconds for the events to be processed
    await Task.Delay(TimeSpan.FromSeconds(30));

    // Stop the processing
    await processor.StopProcessingAsync();
}

```

6. Now, add the following event handler method that includes event de-serialization logic with the schema registry

```

static async Task ProcessEventHandler(ProcessEventArgs eventArgs)
{
    // Create a schema registry client that you can use to deserialize and validate data.
    var schemaRegistryClient = new SchemaRegistryClient(endpoint: schemaRegistryEndpoint,
    credential: new DefaultAzureCredential());

    // Retrieve event data and convert it to a byte array.
    byte[] _memoryStreamBytes = eventArgs.Data.Body.ToArray();
    using var consumerMemoryStream = new MemoryStream(_memoryStreamBytes);

    var consumerSerializer = new SchemaRegistryAvroObjectSerializer(schemaRegistryClient,
    schemaGroup, new SchemaRegistryAvroObjectSerializerOptions { AutoRegisterSchemas = false });
    consumerMemoryStream.Position = 0;

    // Deserialize event data and create order object using schema registry.
    Order sampleOrder = (Order)consumerSerializer.Deserialize(consumerMemoryStream,
    typeof(Order), CancellationToken.None);
    Console.WriteLine("Received - Order ID: " + sampleOrder.id);

    // Update checkpoint in the blob storage so that the app receives only new events the next
    time it's run
    await eventArgs.UpdateCheckpointAsync(eventArgs.CancellationToken);
}

```

7. Now, add the following error handler methods to the class.

```
static Task ProcessErrorHandler(ProcessErrorEventArgs eventArgs)
{
    // Write details about the error to the console window
    Console.WriteLine($"\\tPartition '{ eventArgs.PartitionId}': an unhandled exception was
encountered. This was not expected to happen.");
    Console.WriteLine(eventArgs.Exception.Message);
    return Task.CompletedTask;
}
```

8. Build the project, and ensure that there are no errors.
9. Run the receiver application.
10. You should see a message that the events have been received.

```
Received - Order ID: 12345
```

These events are the three events you sent to the event hub earlier by running the sender program.

## Next steps

Check out [Azure Schema Registry client library for .NET](#) for additional information.

# Quickstart: Create a dedicated Event Hubs cluster using Azure portal

11/26/2021 • 4 minutes to read • [Edit Online](#)

Event Hubs clusters offer single-tenant deployments for customers with the most demanding streaming needs. This offering has a guaranteed 99.99% SLA and is available only on our Dedicated pricing tier. An [Event Hubs cluster](#) can ingress millions of events per second with guaranteed capacity and subsecond latency. Namespaces and event hubs created within a cluster include all features of the premium offering and more, but without any ingress limits. The Dedicated offering also includes the popular [Event Hubs Capture](#) feature at no additional cost, allowing you to automatically batch and log data streams to [Azure Blob Storage](#) or [Azure Data Lake Storage Gen 1](#).

Dedicated clusters are provisioned and billed by **Capacity Units (CUs)**, a pre-allocated amount of CPU and memory resources. You can purchase 1, 2, 4, 8, 12, 16 or 20 CUs for each cluster. In this quickstart, we will walk you through creating a 1 CU Event Hubs cluster through the Azure portal.

## NOTE

This self-serve experience is currently available in preview on [Azure Portal](#). If you have any questions about the Dedicated offering, please reach out to the [Event Hubs team](#).

## Prerequisites

To complete this quickstart, make sure that you have:

- An Azure account. If you don't have one, [purchase an account](#) before you begin. This feature isn't supported with a free Azure account.
- [Visual Studio 2017 Update 3](#) (version 15.3, 26730.01) or later.
- [.NET Standard SDK](#), version 2.0 or later.
- [Created a resource group](#).

## Create an Event Hubs Dedicated Cluster

An Event Hubs cluster provides a unique scoping container in which you can create one or more namespaces. In this Preview phase of the portal self-serve experience, you can create 1 CU clusters in select regions. If you need a cluster larger than 1 CU, you can submit an Azure support request to scale up your cluster after its creation.

## IMPORTANT

You won't be able to delete the cluster for at least 4 hours after you create it. Therefore, you will be charged for a minimum 4 hours of usage of the cluster. For more information on pricing, see [Event Hubs - Pricing](#).

To create a cluster in your resource group using the Azure portal, complete the following steps:

1. Follow [this link](#) to create a cluster on Azure portal. Conversely, select **All services** from the left navigation pane, then type in **Event Hubs Clusters** in the search bar and select **Event Hubs Clusters** from the list of results.
2. On the **Event Hubs Clusters** page, select **+ Create** on the toolbar.

3. On the **Create Cluster** page, configure the following settings:

- a. Enter a **name** for the cluster. The system immediately checks to see if the name is available.
- b. Select the **subscription** in which you want to create the cluster.
- c. Select the **resource group** in which you want to create the cluster.
- d. Select a **location** for the cluster. If your preferred region is grayed out, it is temporarily out of capacity and you can submit a [support request](#) to the Event Hubs team.
- e. Select the **Next: Tags** button at the bottom of the page. You may have to wait a few minutes for the system to fully provision the resources.

The screenshot shows the 'Create Event Hubs Cluster' wizard on the 'Basics' step. The title bar says 'Create Event Hubs Cluster' with a '...>' button. Below it is a 'Event Hubs Cluster' icon. The main area has tabs: 'Basics' (selected), 'Tags', and 'Review + create'. Under 'Cluster Details', there's a note about creating a 1 CU Event Hubs cluster. The 'Cluster name \*' field contains 'contosoehubcluster' with a green checkmark. The 'Subscription \*' dropdown is set to 'Visual Studio Enterprise Subscription'. Under 'Resource group \*', it shows '(New) spehubdedicatedrg' with a dropdown arrow and a 'Create new' link. In the 'Location and Pricing Tier' section, there's a note about Dedicated tier availability. The 'Location \*' dropdown is set to 'West US'. Under 'Pricing tier', '1 CU' is selected, indicated by a green vertical bar, while 'Dedicated' is also listed. At the bottom are buttons: 'Review + create' (blue), '< Previous' (disabled), 'Next: Tags >', and a large progress bar.

4. On the **Tags** page, configure the following:

- a. Enter a **name** and a **value** for the tag you want to add. This step is **optional**.
- b. Select the **Review + Create** button.

**Create Event Hubs Cluster**

Event Hubs Cluster

Basics Tags Review + create

Add a tag

Tags are name/value pairs that enable you to categorize and view consolidated billing by applying the same tag to multiple resources and resource groups. [Learn more](#)

Name ⓘ	Value ⓘ	Resource
<input type="text"/>	:	<input type="text"/> Event Hubs Cluster

**Review + create** < Previous Next: Review + create >

- On the **Review + Create** page, review the details, and select **Create**.

**Create Event Hubs Cluster**

Event Hubs Cluster

Validation succeeded.

Basics Tags Review + create

Event Hubs Cluster by Microsoft

**Estimated cost per hour**  
USD [View pricing details](#)

**Basics**

Cluster name	contosoehubcluster
Subscription	Visual Studio Enterprise Subscription
Resource group	spehubdedicatedrdr
Location	West US
Capacity	1 CU

**Create** < Previous Next >

- After the creation is successful, select **Go to resource** to navigate to the home page for your Event Hubs cluster.

## Create a namespace and event hub within a cluster

- To create a namespace within a cluster, on the **Event Hubs Cluster** page for your cluster, select **+ Namespace** from the top menu.

The screenshot shows the Azure Event Hubs Cluster Overview page for the 'contosohubcluster'. The 'Namespace' button in the top navigation bar is highlighted with a red box. The page displays cluster statistics: 0 NAMESPACES, ACTIVE status, and 1 cu Capacity. A metrics chart tracks 'Incoming Requests (Sum)' and 'Successful Requests (Sum)' for the 'contosohubcluster' from 8:45 PM to 9:15 PM UTC-04:00. The successful requests chart shows a sharp increase from 0 to approximately 25 requests. Below the chart, a table titled 'CLUSTER NAMESPACES' shows a search bar and a table with columns: Name, Status, Created, and Updated. The table displays the message 'No Event Hubs Namespaces found.'

2. On the **Create a namespace** page, do the following steps:

- Enter a **name for the namespace**. The system checks to see if the name is available.
- The namespace inherits the following properties:
  - Subscription ID
  - Resource Group
  - Location
  - Cluster Name
- Select **Create** to create the namespace. Now you can manage your cluster.

The screenshot shows the 'Create Namespace in Cluster' dialog box. It prompts the user to specify properties for the namespace, starting with the 'Name' field, which is currently empty. Below the form, it states that the namespace will inherit properties from its parent cluster, specifically listing the Subscription ID, Resource Group, Location, and Event Hubs Cluster. At the bottom of the dialog is a blue 'Create' button.

PROPERTY	VALUE
Subscription ID	<Your Azure subscription ID>
Resource Group	EventHubClusterRG
Location	eastus2
Event Hubs Cluster	FirstEverEHCluster

3. Once your namespace is created, you can [create an event hub](#) as you would normally create one within a namespace.

# Submit a support request

If you wish to change the size of your cluster after creation or if your preferred region isn't available, submit a support request by following these steps:

1. In [Azure portal](#), select **Help + support** from the left menu.
2. Select **+ New support request** from the Support menu.
3. On the support page, follow these steps:
  - a. For **Issue Type**, select **Technical** from the drop-down list.
  - b. For **Subscription**, select your subscription.
  - c. For **Service**, select **My services**, and then select **Event Hubs**.
  - d. For **Resource**, select your cluster if it exists already, otherwise select **General Question/Resource Not Available**.
  - e. For **Problem type**, select **Quota**.
  - f. For **Problem subtype**, select one of the following values from the drop-down list:
    - a. Select **Request for Dedicated SKU** to request for the feature to be supported in your region.
    - b. Select **Request to Scale Up or Scale Down Dedicated Cluster** if you want to scale up or scale down your dedicated cluster.
  - g. For **Subject**, describe the issue.

The screenshot shows the Azure portal's 'Help + support' interface. On the left sidebar, under 'Support', the 'Help + support' option is highlighted. The main area displays the 'Basics' tab of a support request form. The form fields are as follows:

- \* Issue type: Technical
- \* Subscription: Your Azure subscription
- \* Service: My services (Event Hubs selected)
- \* Resource: danskafkahub
- \* Problem type: Quota
- \* Problem subtype: Request for Dedicated SKU
- \* Subject: In a few words, describe your issue

# Delete a dedicated cluster

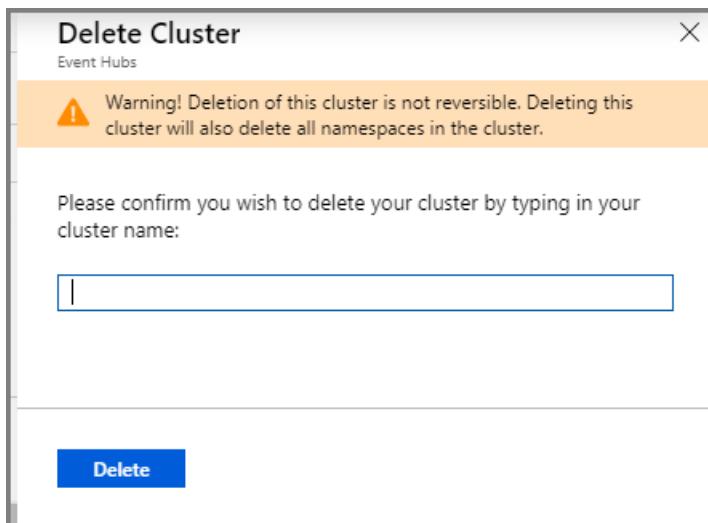
1. To delete the cluster, select **Delete** from the top menu.

## IMPORTANT

You won't be able to delete the cluster for at least 4 hours after you create it. Therefore, you will be charged for a minimum 4 hours of usage of the cluster. For more information on pricing, see [Event Hubs - Pricing](#).

2. A message will appear confirming your wish to delete the cluster.

3. Type the **name of the cluster** and select **Delete** to delete the cluster.



## Next steps

In this article, you created an Event Hubs cluster. For step-by-step instructions to send and receive events from an event hub, and capture events to an Azure storage or Azure Data Lake Store, see the following tutorials:

- Send and receive events
  - [.NET Core](#)
  - [Java](#)
  - [Python](#)
  - [JavaScript](#)
- [Use Azure portal to enable Event Hubs Capture](#)
- [Use Azure Event Hubs for Apache Kafka](#)

# Tutorial: Visualize data anomalies in real-time events sent to Azure Event Hubs

11/26/2021 • 9 minutes to read • [Edit Online](#)

With Azure Event Hubs, you can use Azure Stream Analytics to check the incoming data and pull out the anomalies, which you can then visualize in Power BI. Let's say you have thousands of devices constantly sending real-time data to an event hub, adding up to millions of events per second. How do you check that much data for anomalies, or errors, in the data? For example, what if the devices are sending credit card transactions, and you need to capture anywhere you have multiple transactions in multiple countries/regions within a 5-second time interval? This could happen if someone steals credit cards and then uses them to purchase items around the globe at the same time.

In this tutorial, you simulate this example. You run an application that creates and sends credit card transactions to an event hub. Then you read the stream of data in real time with Azure Stream Analytics, which separates the valid transactions from the invalid transactions, and then use Power BI to visually identify the transactions that are tagged as invalid.

In this tutorial, you learn how to:

- Create an Event Hubs namespace
- Create an event hub
- Run the app that sends credit card transactions
- Configure a Stream Analytics job to process those transactions
- Configure a Power BI visualization to show the results

## Prerequisites

Before you start, make sure you have completed the following steps:

- If you don't have an Azure subscription, create a [free account](#) before you begin.
- [Create an Event Hubs namespace and an event hub in the namespace](#).
- Follow instructions from [Get an Event Hubs connection string](#). Note down the connection string to the Event Hubs namespace and the name of the event hub.
- Install [Visual Studio](#). You use a Visual Studio solution to run an app to produce and send test event data to the event hub.
- You need a Power BI account to analyze output from a Stream Analytics job. You can [try Power BI for free](#).

## Run app to produce test event data

The Event Hubs [samples on GitHub](#) include an Anomaly Detector app that produces test data for you. It simulates the use of credit cards by writing credit card transactions to the event hub, including occasionally writing several transactions for the same credit card in multiple locations so that they are tagged as anomalies. To run this app, follow these steps:

1. Download the [Azure Event Hubs samples](#) from GitHub and unzip it locally.
2. Navigate to the folder `\azure-event-hubs-master\samples\DotNet\` folder.
3. Switch to the `Azure.Messaging.EventHubs\AnomalyDetector\` folder and double-click on `AnomalyDetector.sln` to open the solution in Visual Studio.

To use the old version of the sample that uses the old Microsoft.Azure.EventHubs package, open the solution from the **Microsoft.Azure.EventHubs\AnomalyDetector** folder.

4. Open Program.cs and replace **Event Hubs connection string** with the connection string you saved when running the script.
5. Replace **Event Hub name** with your event hub name. Click F5 to run the application. It starts sending events to your event hub, and continues until it has sent 1000 events. There are a few instances where the app needs to be running for you to retrieve data. These cases are pointed out in the following instructions, where needed.

## Set up Azure Stream Analytics

Now you can stream data into your event hub. To use that data in a Power BI visualization, start by setting up a Stream Analytics job to retrieve the data that is then fed into the Power BI visualization.

### Create the Stream Analytics job

1. In the Azure portal, click **Create a resource**. Type **stream analytics** into the search box and press **Enter**. Select **Stream Analytics Job**. Click **Create** on the Stream Analytics job pane.
2. Enter the following information for the job:

**Job name:** Use **contosoEHjob**. This field is the name of the job and it must be globally unique.

**Subscription:** Select your subscription.

**Resource group:** Use the same resource group used by your event hub (**ContosoResourcesEH**).

**Location:** Use the same Azure region that you used before.

## New Stream Analytics job

This will create a new Stream Analytics job. You will be charged according to Azure Stream Analytics billing model. Learn more. →

Job name \*

 ✓

Subscription \*

 ▾

Resource group \*

 ▾  
[Create new](#)

Location \*

 ▾

Hosting environment ⓘ

Cloud    Edge

Streaming units (1 to 192) ⓘ

3

Secure all private data assets needed by this job in my Storage account. ⓘ

---

**Create**

Accept the defaults for the rest of the fields. Click **Create**.

### Add an input to the Stream Analytics job

If you're not in the portal at the **Stream Analytics Job** pane, you can get back to your Stream Analytics job by clicking **Resource Groups** in the portal, then selecting your resource group (**ContosoResourcesEH**). This action shows all of the resources in the group, and you can then select your stream analytics job.

The inputs for the Steam Analytics job are the credit card transactions from the event hub.

1. In the left menu, select **Inputs** in the **Job Topology** section.
2. In the **Inputs** pane, click **Add stream input** and select **Event Hubs**. On the screen that appears, fill in the following fields:

**Input alias:** Use **contosoinputs**. This field is the name of the input stream, used when defining the query for the data.

**Subscription:** Select your Azure subscription.

**Event Hubs namespace:** Select your Event Hub namespace.

**Event Hub name:** Click **Use existing** and select your event hub.

**Event Hubs consumer group:** Leave this field blank to use the default consumer group.

Accept defaults for the rest of the fields.

## Event Hub

New input

**Input alias \***

 ✓

Provide Event Hub settings manually  
 Select Event Hub from your subscriptions

**Subscription**

**Event Hub namespace \*** ⓘ

**Event Hub name \*** ⓘ

Create new  Use existing

**Event Hub consumer group \*** ⓘ

Create new  Use existing

**Authentication mode**

Managed Identity authentication will be enabled for this Stream Analytics job if it is not enabled. The Azure Event Hubs Data Owner role will be granted to the Managed Identity for this Stream Analytics job when you click Save. If grant fails follow the manual grant steps [here](#).

**Partition key** ⓘ

**Event serialization format \*** ⓘ

**Encoding** ⓘ

**Save**

3. Click **Save**.

### Add an output to the Stream Analytics job

1. In the left menu, select **Outputs** in the **Job Topology** section. This field is the name of the output stream, used when defining the query for the data.
2. In the **Outputs** pane, click **Add**, and then select **Power BI**. On the screen that appears, complete the following fields:

**Output alias:** Use **contosooutputs**. This field is the unique alias for the output.

**Dataset name:** Use **contosoehdataset**. This field is the name of the dataset to be used in Power BI.

**Table name:** Use **contosoehtable**. This field is the name of the table to be used in Power BI.

Accept the defaults for the rest of the fields.

Power BI

New output

\* Output alias  
contosooutputs ✓

Group workspace  
Authorize connection to load workspaces ▾

\* Dataset name  
contosoehdataset ✓

\* Table name  
contosoehtablename ✓

**Authorize connection**  
You'll need to authorize with Power BI to configure your output settings.

**Authorize**

Don't have a Microsoft Power BI account yet?  
[Sign up](#)

**i** Note: You are granting this output permanent access to your Power BI dashboard. Should you need to revoke this access in the future you can do one of the following:  
1. Change the user account password.  
2. Delete this output.  
3. Delete this job.

**Save**

3. Click **Authorize**, and sign in to your Power BI account.

4. Accept the defaults for the rest of the fields.

5. Click **Save**.

### Configure the query of the Stream Analytics job

This query is used to retrieve the data that is ultimately sent to the Power BI visualization. It uses **contosoinputs** and **contosooutputs**, which you previously defined when setting up the job. This query retrieves the credit card transactions that it deems fraudulent, which are transactions in which the same credit card number has multiple transactions in different locations in the same five-second interval.

1. Under **Job Topology**, click **Query**.

2. Replace the query with the following one:

```

/* criteria for fraud:
   credit card purchases with the same card
   in different locations within 5 seconds
*/
SELECT System.Timestamp AS WindowEnd,
       COUNT(*) as FraudulentUses
INTO contosooutputs
FROM contosoinputs CS1 TIMESTAMP BY [Timestamp]
    JOIN contosoinputs CS2 TIMESTAMP BY [Timestamp]
    /* where the credit card # is the same */
    ON CS1.CreditCardId = CS2.CreditCardId
    /* and time between the two is between 0 and 5 seconds */
    AND DATEDIFF(second, CS1, CS2) BETWEEN 0 AND 5
    /* where the location is different */
WHERE CS1.Location != CS2.Location
GROUP BY TumblingWindow(Duration(second, 1))

```

3. Click **Save**.

### Test the query for the Stream Analytics job

1. Run the Anomaly Detector app to send data to the event hub while you're setting up and running the test.
2. In the Query pane, click the dots next to the **contosoinputs** input, and then select **Sample data from input**.
3. Specify that you want three minutes of data, then click **OK**. Wait until you're notified that the data has been sampled.
4. Click **Test** and make sure you're getting results. Results are displayed in the **Results** section of the bottom pane on the right under the query.
5. Close the Query pane.

### Run the Stream Analytics job

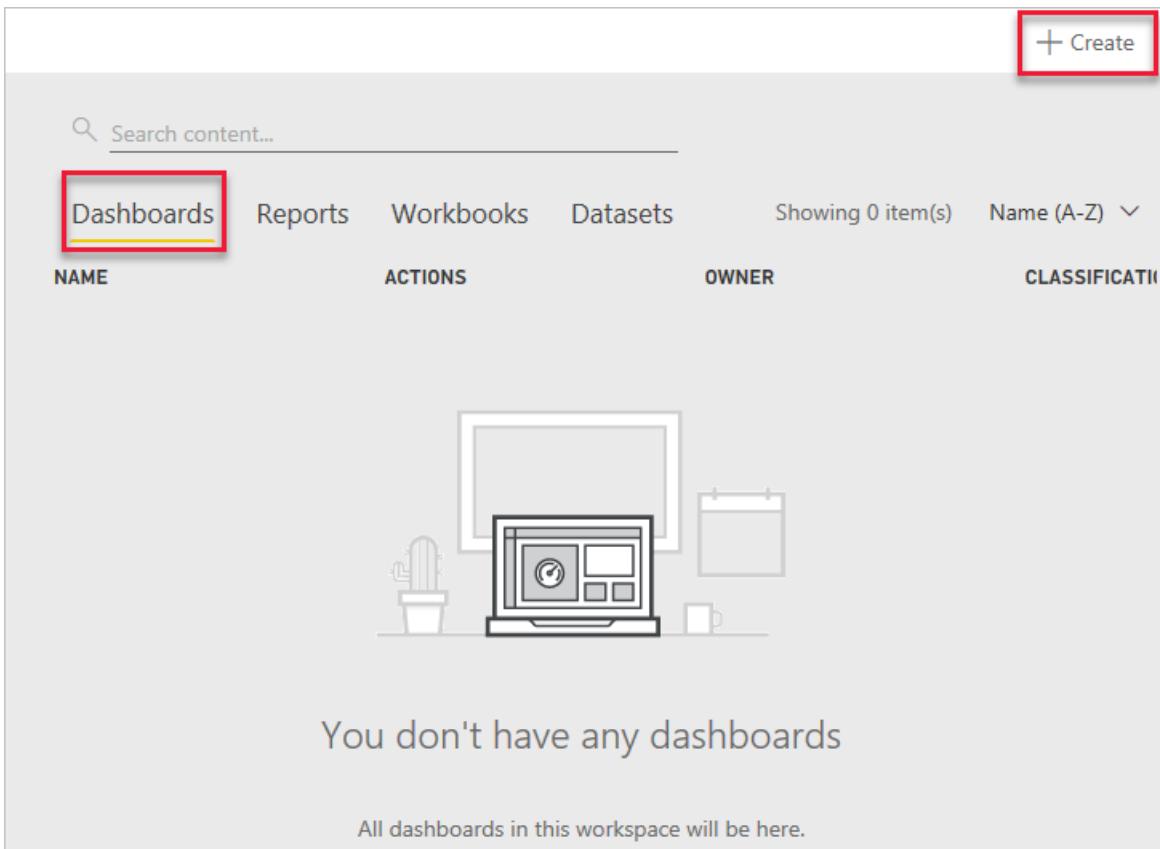
In the Stream Analytics job, click **Start**, then **Now**, then **Start**. Once the job successfully starts, the job status changes from **Stopped** to **Running**.

## Set up the Power BI visualizations

1. Run the Anomaly Detector app to send data to the event hub while you're setting up the Power BI visualization. You may need to run it multiple times, as it only generates 1000 transactions each time it runs.
2. Sign in to your [Power BI](#) account.
3. Go to **My Workspace**.
4. Click **Datasets**.

You should see the dataset that you specified when you created the output for the Stream Analytics job (**contosoehdataset**). It may take 5-10 minutes for the dataset to appear for the first time.

5. Click **Dashboards**, then click **Create** and select **Dashboard**.

A screenshot of the Google Data Studio interface. At the top right, there is a red-bordered 'Create' button with a plus sign. Below it is a search bar with the placeholder 'Search content...'. The main navigation bar has tabs for 'Dashboards' (which is highlighted with a red border), 'Reports', 'Workbooks', and 'Datasets'. To the right of these tabs, it says 'Showing 0 item(s)' and 'Name (A-Z) ▾'. Below the navigation is a table header with columns: 'NAME', 'ACTIONS', 'OWNER', and 'CLASSIFICATION'. The body of the table is empty, showing a placeholder image of a dashboard with a clock and charts. A message below the table says 'You don't have any dashboards' and 'All dashboards in this workspace will be here.'

6. Specify the name of the dashboard, then click **Create**. Use **Credit Card Anomalies**.

A screenshot of a 'Create dashboard' dialog box. At the top left is the title 'Create dashboard' and at the top right is a close button 'X'. Below the title is a label 'Dashboard name' followed by a text input field containing 'Credit Card Anomalies'. At the bottom right are two buttons: a yellow 'Create' button and a grey 'Cancel' button.

7. On the Dashboard page, click **Add tile**, select **Custom Streaming Data** in the **REAL-TIME DATA** section, then click **Next**.

## Add tile

Select source

### MEDIA



Web content



Image



Text box



Video

### REAL-TIME DATA



Custom Streaming Data

Next

Cancel

8. Select your dataset (**contosoehdataset**) and click **Next**.

## Add a custom streaming data tile

Choose a streaming dataset

+ Add streaming dataset

### YOUR DATASETS

contosoehdataset

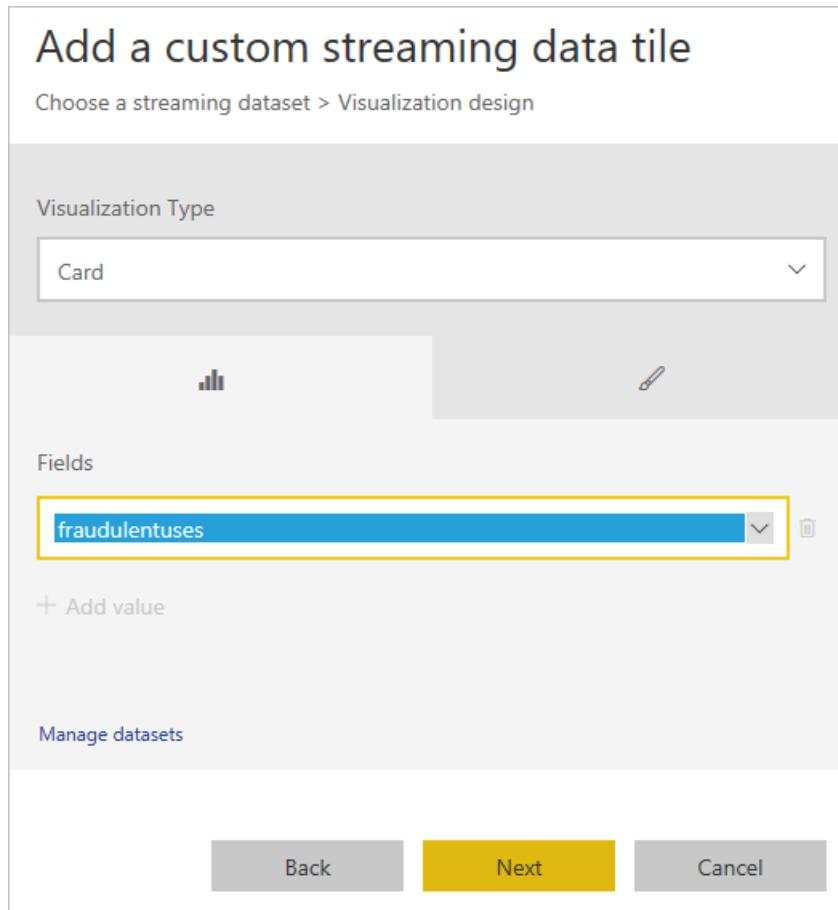
[Manage datasets](#)

Back

Next

Cancel

9. Select **Card** for visualization type. Under **Fields**, click **Add value**, then select `fraudulentuses`.



Click **Next**.

10. Set the title to **Fraudulent uses** and the subtitle to **Sum in last few minutes**. Click **Apply**. It saves the tile to your dashboard.

## Tile details

\* Required

### Details

Display title and subtitle

#### Title

Fraudulent uses

#### Subtitle

Sum in last few minutes.

### Functionality

Set custom link

#### Link type

External link

Link to a dashboard or report in the current workspace

#### URL \*

#### Open custom link in the same tab?

Yes

No

[Restore default](#)

[Technical Details](#)

Back

Apply

Cancel

### IMPORTANT

When you run the sample application and stream data to the event hub, the number on this tile changes rapidly (every second). It's because the Stream Analytics query actually updates the value **every second**. Update the query to a 3 minute tumbling window to see the sum in the last few minutes.

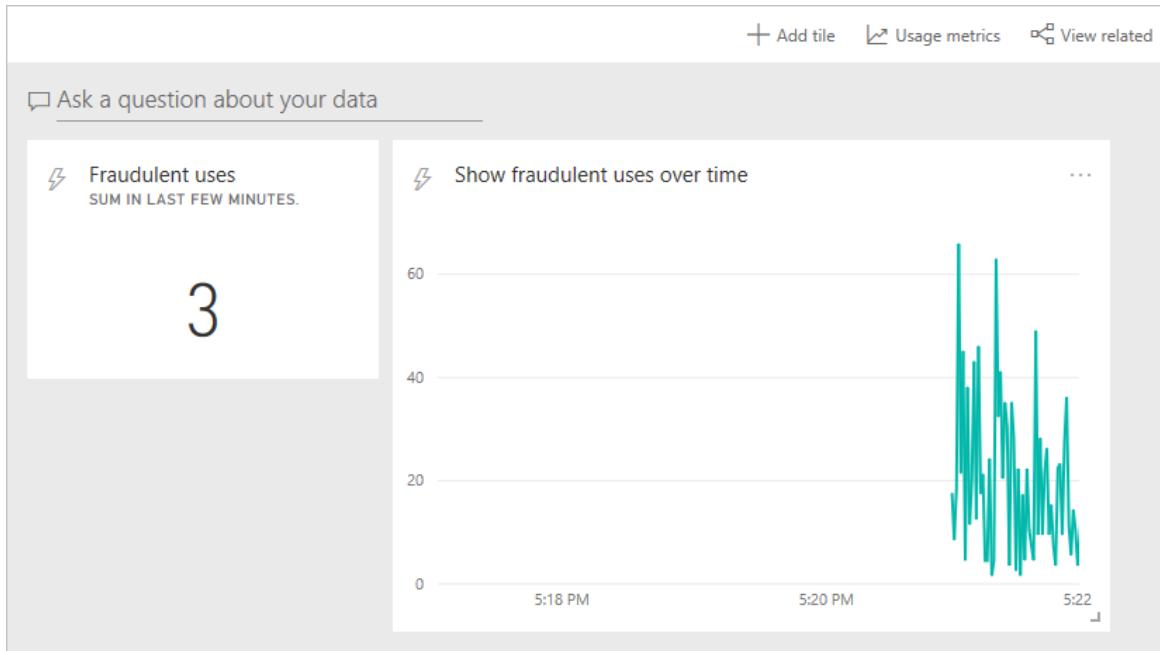
11. Add another visualization. Repeat the first few steps again:

- Click **Add Tile**.
- Select **Custom Streaming Data**.
- Click **Next**.
- Select your dataset and then click **Next**.

12. Under **Visualization Type**, select **Line chart**.

13. Under **Axis**, click **Add Value**, and select `windowend`.

14. Under **Values**, click **Add value** and select `fraudulentuses`.
15. Under **Time window to display**, select the last five minutes. Click **Next**.
16. Specify **Show fraudulent uses over time** for the title and leave the subtitle for the tile blank, then click **Apply**. You are returned to your dashboard.
17. Run the Anomaly Detector app again to send some data to the event hub. You see the **Fraudulent uses** tile change as it analyzes the data, and the line chart shows data.



## Clean up resources

If you want to remove all of the resources you've created, remove the Power BI visualization data, then delete the resource group. Deleting the resource group deletes all resources contained within the group. In this case, it removes the event hub, Event Hub namespace, stream analytics job, and the resource group itself.

### Clean up resources in the Power BI visualization

Log into your Power BI account. Go to **My Workspace**. On the line with your dashboard name, click the trash can icon. Next, go to **DataSets** and click the trash can icon to delete the dataset (`contosoehdataset`).

### Clean up resources

Delete the resource group that contains all the resources you created as part of this tutorial.

## Next steps

In this tutorial, you learned how to:

- Create an Event Hubs namespace
- Create an event hub
- Run the app that simulates events and sends them to the event hub
- Configure a Stream Analytics job to process events sent to the hub
- Configure a Power BI visualization to show the results

Advance to the next article to learn more about Azure Event Hubs.

[Get started sending messages to Azure Event Hubs in .NET Standard](#)

# Tutorial: Analyze fraudulent call data with Stream Analytics and visualize results in Power BI dashboard

11/26/2021 • 14 minutes to read • [Edit Online](#)

This tutorial shows you how to analyze phone call data using Azure Stream Analytics. The phone call data, generated by a client application, contains fraudulent calls, which are filtered by the Stream Analytics job. You can use the techniques from this tutorial for other types of fraud detection, such as credit card fraud or identity theft.

In this tutorial, you learn how to:

- Generate sample phone call data and send it to Azure Event Hubs.
- Create a Stream Analytics job.
- Configure job input and output.
- Define queries to filter fraudulent calls.
- Test and start the job.
- Visualize results in Power BI.

## Prerequisites

Before you start, make sure you have completed the following steps:

- If you don't have an Azure subscription, create a [free account](#).
- Download the phone call event generator app [TelcoGenerator.zip](#) from the Microsoft Download Center or get the source code from [GitHub](#).
- You will need Power BI account.

## Sign in to Azure

Sign in to the [Azure portal](#).

## Create an Azure Event Hub

Before Stream Analytics can analyze the fraudulent calls data stream, the data needs to be sent to Azure. In this tutorial, you will send data to Azure by using [Azure Event Hubs](#).

Use the following steps to create an Event Hub and send call data to that Event Hub:

1. Sign in to the [Azure portal](#).
2. Select **Create a resource > Internet of Things > Event Hubs**.

3. Fill out the **Create Namespace** pane with the following values:

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	asaTutorialEventHub	A unique name to identify the event hub namespace.
Subscription	<Your subscription>	Select an Azure subscription where you want to create the event hub.
Resource group	MyASADemoRG	Select <b>Create New</b> and enter a new resource-group name for your account.
Location	West US2	Location where the event hub namespace can be deployed.

4. Use default options on the remaining settings and select **Review + create**. Then select **Create** to start the deployment.

## Create Namespace

Event Hubs

[Basics](#)   [Features](#)   [Tags](#)   [Review + create](#)

### PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

&lt;your subscription&gt;

Resource group \*

MyASADemoRG

[Create new](#)

### INSTANCE DETAILS

Enter required settings for this namespace, including a price tier and configuring the number of throughput units.

Namespace name \*

asaTutorialEventHub

.servicebus.windows.net

Location \*

West US 2

Pricing tier ([View full pricing details](#)) \*

Standard (20 Consumer groups, 1000 Brokered connections)

Throughput Units \*

 1[Review + create](#)[< Previous](#)[Next: Features >](#)

5. When the namespace has finished deploying, go to **All resources** and find *asaTutorialEventHub* in the list of Azure resources. Select *asaTutorialEventHub* to open it.
6. Next select **+ Event Hub** and enter a **Name** for the Event Hub. Set the **Partition Count** to 2. Use the default options in the remaining settings and select **Create**. Then wait for the deployment to succeed.

Microsoft Azure Search resources, services, and docs (G+/-) Cloud Shell Help Feedback ? Smile

Home > Microsoft.EventHub > asaTutorialEventHub >

## Create Event Hub

Event Hubs

Name \* ⓘ MyEventHub ✓

Partition Count ⓘ 2

Message Retention ⓘ 1

Capture ⓘ Off

Create



### Grant access to the event hub and get a connection string

Before an application can send data to Azure Event Hubs, the event hub must have a policy that allows access. The access policy produces a connection string that includes authorization information.

1. Navigate to the event hub you created in the previous step, *MyEventHub*. Select **Shared access policies** under **Settings**, and then select **+ Add**.
2. Name the policy **MyPolicy** and ensure **Manage** is checked. Then select **Create**.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various options like Overview, Access control (IAM), and Settings. Under Settings, the 'Shared access policies' section is highlighted with a red box. In the main content area, there's a search bar and a '+ Add' button. Below it, a table has 'Policy' and 'Claims' columns. A note says 'no policies have been set up yet.' To the right, a modal window titled 'Add SAS Policy' is open, also with a red border. It has a 'Policy name \*' field containing 'MyPolicy' with a checkmark. Underneath are three checkboxes: 'Manage' (checked), 'Send' (unchecked), and 'Listen' (checked). At the bottom of the modal is a 'Create' button.

3. Once the policy is created, select the policy name to open the policy. Find the **Connection string-primary key**. Select the **copy** button next to the connection string.

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various options like Overview, Access control (IAM), and Settings. Under Settings, the 'Shared access policies' section is highlighted with a red box. In the main content area, there's a search bar and a '+ Add' button. Below it, a table has 'Policy' and 'Claims' columns. A row for 'MyPolicy' is selected, with a red box around it. To the right, a modal window titled 'SAS Policy: MyPolicy' is open, also with a red border. It has a 'Save' button at the top. Under 'Manage' checkboxes, 'Manage' is checked, while 'Send' and 'Listen' are unchecked. Below that is a 'Primary key' field containing a long string of characters, with a copy icon to its right. A tooltip 'Copy to clipboard' appears over this icon. Below the primary key is a 'Secondary key' field containing another long string, with a copy icon to its right. At the bottom are 'Connection string-primary' and 'Connection string-secondary key' fields, each with a copy icon to its right.

4. Paste the connection string into a text editor. You need this connection string in the next section.

The connection string looks as follows:

```
Endpoint=sb://<Your event hub namespace>.servicebus.windows.net/;SharedAccessKeyName=<Your shared access policy name>;SharedAccessKey=<generated key>;EntityPath=<Your event hub name>
```

Notice that the connection string contains multiple key-value pairs separated with semicolons: **Endpoint**, **SharedAccessKeyName**, **SharedAccessKey**, and **EntityPath**.

## Start the event generator application

Before you start the TelcoGenerator app, you should configure it to send data to the Azure Event Hubs you created earlier.

1. Extract the contents of [TelcoGenerator.zip](#) file.
2. Open the `TelcoGenerator\TelcoGenerator\telcodatagen.exe.config` file in a text editor of your choice. There is more than one `.config` file, so be sure that you open the correct one.
3. Update the `<appSettings>` element in the config file with the following details:
  - Set the value of the *EventHubName* key to the value of the EntityPath in the connection string.
  - Set the value of the *Microsoft.ServiceBus.ConnectionString* key to the connection string without the EntityPath value. Don't forget to remove the semicolon that precedes the EntityPath value.
4. Save the file.
5. Next open a command window and change to the folder where you unzipped the TelcoGenerator application. Then enter the following command:

```
.\telcodatagen.exe 1000 0.2 2
```

This command takes the following parameters:

- Number of call data records per hour.
- Percentage of fraud probability, which is how often the app should simulate a fraudulent call. The value 0.2 means that about 20% of the call records will look fraudulent.
- Duration in hours, which is the number of hours that the app should run. You can also stop the app at any time by ending the process (**Ctrl+C**) at the command line.

After a few seconds, the app starts displaying phone call records on the screen as it sends them to the event hub. The phone call data contains the following fields:

RECORD	DEFINITION
CallrecTime	The timestamp for the call start time.
SwitchNum	The telephone switch used to connect the call. For this example, the switches are strings that represent the country/region of origin (US, China, UK, Germany, or Australia).
CallingNum	The phone number of the caller.
CallingIMSI	The International Mobile Subscriber Identity (IMSI). It's a unique identifier of the caller.

RECORD	DEFINITION
CalledNum	The phone number of the call recipient.
CalledIMSI	International Mobile Subscriber Identity (IMSI). It's a unique identifier of the call recipient.

## Create a Stream Analytics job

Now that you have a stream of call events, you can create a Stream Analytics job that reads data from the event hub.

1. To create a Stream Analytics job, navigate to the [Azure portal](#).
2. Select **Create a resource** and search for **Stream Analytics job**. Select the **Stream Analytics job** tile and select **Create**.
3. Fill out the **New Stream Analytics job** form with the following values:

SETTING	SUGGESTED VALUE	DESCRIPTION
Job name	ASATutorial	A unique name to identify the event hub namespace.
Subscription	<Your subscription>	Select an Azure subscription where you want to create the job.
Resource group	MyASADemoRG	Select <b>Use existing</b> and enter a new resource-group name for your account.
Location	West US2	Location where the job can be deployed. It's recommended to place the job and the event hub in the same region for best performance and so that you don't pay to transfer data between regions.
Hosting environment	Cloud	Stream Analytics jobs can be deployed to cloud or edge. Cloud allows you to deploy to Azure Cloud, and Edge allows you to deploy to an IoT Edge device.
Streaming units	1	Streaming units represent the computing resources that are required to execute a job. By default, this value is set to 1. To learn about scaling streaming units, see <a href="#">understanding and adjusting streaming units</a> article.

4. Use default options on the remaining settings, select **Create**, and wait for the deployment to succeed.

## New Stream Analytics job

X

**i** This will create a new Stream Analytics job. You will be charged according to Azure Stream Analytics billing model. Learn more. →

Job name \*

ASATutorial



Subscription \*

<your subscription>



Resource group \*

<your resource group>



[Create new](#)

Location \*

West US 2



Hosting environment ⓘ

Cloud

Edge

Streaming units (1 to 192) ⓘ



1

Secure all private data assets needed by this job in my Storage account. ⓘ

**Create**

## Configure job input

The next step is to define an input source for the job to read data using the event hub you created in the previous section.

1. From the Azure portal, open the **All resources** page, and find the *ASATutorial* Stream Analytics job.
2. In the **Job Topology** section of the Stream Analytics job, select **Inputs**.
3. Select **+ Add stream input** and **Event hub**. Fill out the input form with the following values:

SETTING	SUGGESTED VALUE	DESCRIPTION
Input alias	CallStream	Provide a friendly name to identify your input. Input alias can contain alphanumeric characters, hyphens, and underscores only and must be 3-63 characters long.
Subscription	<Your subscription>	Select the Azure subscription where you created the event hub. The event hub can be in same or a different subscription as the Stream Analytics job.
Event hub namespace	asaTutorialEventHub	Select the event hub namespace you created in the previous section. All the event hub namespaces available in your current subscription are listed in the dropdown.
Event Hub name	MyEventHub	Select the event hub you created in the previous section. All the event hubs available in your current subscription are listed in the dropdown.
Event Hub policy name	MyPolicy	Select the event hub shared access policy you created in the previous section. All the event hubs policies available in your current subscription are listed in the dropdown.

4. Use default options on the remaining settings and select **Save**.

The screenshot shows the Azure Stream Analytics job configuration interface. On the left, the sidebar has a tree view with 'ASATutorial | Inputs' selected. Under 'Job topology', 'Inputs' is highlighted with a red box. In the main area, there's a 'New input' button with a plus sign. To its right is a detailed configuration form for an Event Hub input. The form includes fields for 'Input alias \*' (set to 'CallStream'), 'Subscription' (set to 'Azure Big Data and Beyond Docs Team'), 'Event Hub namespace \*' (set to 'asaTutorialEventHub'), 'Event Hub name \*' (set to 'myeventhub', with 'Use existing' selected), 'Event Hub policy name \*' (set to 'MyPolicy', with 'Use existing' selected), and 'Event Hub consumer group' (set to 'ASATutorial\_CallStream\_consumer\_group'). A blue 'Save' button is at the bottom right of the form.

## Configure job output

The last step is to define an output sink where the job can write the transformed data. In this tutorial, you output and visualize data with Power BI.

1. From the Azure portal, open **All resources**, and select the *ASATutorial* Stream Analytics job.
2. In the **Job Topology** section of the Stream Analytics job, select the **Outputs** option.
3. Select **+ Add > Power BI**. Then, select **Authorize** and follow the prompts to authenticate Power BI.

The screenshot shows the Azure Stream Analytics job configuration interface. On the left, the sidebar has a tree view with 'ASATutorial | Outputs' selected. Under 'Job topology', 'Outputs' is highlighted with a red box. In the main area, there's a '+ Add' button with a plus sign. To its right is a list of output types: Event Hub, SQL Database, Blob storage/ADLS Gen2, Table storage, Service Bus topic, Service Bus queue, Cosmos DB, Power BI, Data Lake Storage Gen1, Azure Function, and Azure Synapse Analytics (formerly SQL DW). The 'Power BI' option is highlighted with a red box. To the right is a 'Power BI' configuration form. It includes an 'Authorize' button, which is highlighted with a red box. Below it, there's a note: 'Don't have a Microsoft Power BI account yet? Sign up'. At the bottom, there's a note: 'You are granting temporary access in order to get Power BI dashboard metadata.'

4. Fill the output form with the following details and select **Save**:

SETTING	SUGGESTED VALUE
Output alias	MyPBloutput
Group workspace	My workspace
Dataset name	ASAdataset
Table name	ASATable
Authentication mode	User token

# Power BI

X

New output

Currently authorized as Mary McCready (mamccrea@microsoft.com)

Output alias \*

MyPBloutput



Group workspace

My workspace



Dataset name \* ⓘ

ASAdataset



Table name \*

ASATable



Authentication mode

User token



Note: You are granting this output permanent access to your Power BI dashboard. Should you need to revoke this access in the future you can do one of the following:

1. Change the user account password.
2. Delete this output.
3. Delete this job.

**Save**

This tutorial uses the *User token* authentication mode. To use Managed Identity, see [Use Managed Identity to authenticate your Azure Stream Analytics job to Power BI](#).

## Create queries to transform real-time data

At this point, you have a Stream Analytics job set up to read an incoming data stream. The next step is to create a query that analyzes the data in real time. The queries use a SQL-like language that has some extensions specific

to Stream Analytics.

In this section of the tutorial, you create and test several queries to learn a few ways in which you can transform an input stream for analysis.

The queries you create here will just display the transformed data to the screen. In a later section, you'll write the transformed data to Power BI.

To learn more about the language, see the [Azure Stream Analytics Query Language Reference](#).

### Test using a pass-through query

If you want to archive every event, you can use a pass-through query to read all the fields in the payload of the event.

1. Navigate to your Stream Analytics job in the Azure portal and select **Query** under *Job topology*.

2. In the query window, enter this query:

```
SELECT  
*  
FROM  
CallStream
```

#### NOTE

As with SQL, keywords are not case-sensitive, and whitespace is not significant.

In this query, `CallStream` is the alias that you specified when you created the input. If you used a different alias, use that name instead.

3. Select **Test query**.

The Stream Analytics job runs the query against the sample data from the input and displays the output at the bottom of the window. The results indicate that the Event Hub and the Streaming Analytics job are configured correctly.

The screenshot shows the Azure Stream Analytics Query editor interface. On the left, there's a navigation sidebar with links like Home, ASATutorial, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Properties, Locks, Inputs, Functions, Query (which is selected and highlighted with a red border), Outputs, Configure, Environment, Storage account settings, and Scale. The main area has tabs for Inputs (1) and Outputs (2). Under Inputs (1), it shows CallStream. Under Outputs (2), it shows MyPBloutput. A central pane contains a code editor with a query:

```
>>> Test query Save query Discard changes  
1 SELECT  
2 *  
3 FROM  
4 CallStream
```

Below the code editor is a preview pane with tabs for Input preview and Test results. The Test results tab is selected and highlighted with a red border. It shows the following table with 50 rows of data:

RecordType	SystemIdentity	FileNum	SwitchNum	CallingNum
"MO"	"d0"	"571"	"Australia"	"678973773"
"MO"	"d0"	"569"	"US"	"123437762"
"MO"	"d0"	"567"	"China"	"678922820"
"MO"	"d0"	"565"	"China"	"234590598"
"MO"	"d0"	"563"	"China"	"567891282"
"MO"	"d0"	"560"	"US"	"123458024"
"MO"	"d0"	"558"	"China"	"678908335"

At the bottom of the preview pane, it says "Success" and "Ln 4, Col 18".

The exact number of records you see will depend on how many records were captured in the sample.

## Reduce the number of fields using a column projection

In many cases, your analysis doesn't need all the columns from the input stream. You can use a query to project a smaller set of returned fields than in the pass-through query.

Run the following query and notice the output.

```
SELECT CallRecTime, SwitchNum, CallingIMSI, CallingNumCalledNum
FROM
    CallStream
```

## Count incoming calls by region: Tumbling window with aggregation

Suppose you want to count the number of incoming calls per region. In streaming data, when you want to perform aggregate functions like counting, you need to segment the stream into temporal units, since the data stream itself is effectively endless. You do this using a Streaming Analytics [window function](#). You can then work with the data inside that window as a unit.

For this transformation, you want a sequence of temporal windows that don't overlap—each window will have a discrete set of data that you can group and aggregate. This type of window is referred to as a *Tumbling window*. Within the Tumbling window, you can get a count of the incoming calls grouped by `SwitchNum`, which represents the country/region where the call originated.

1. Paste the following query in the query editor:

```
SELECT
    System.Timestamp as WindowEnd, SwitchNum, COUNT(*) as CallCount
FROM
    CallStream TIMESTAMP BY CallRecTime
GROUP BY TUMBLINGWINDOW(s, 5), SwitchNum
```

This query uses the `Timestamp By` keyword in the `FROM` clause to specify which timestamp field in the input stream to use to define the Tumbling window. In this case, the window divides the data into segments by the `CallRecTime` field in each record. (If no field is specified, the windowing operation uses the time that each event arrives at the event hub. See "Arrival Time Vs Application Time" in [Stream Analytics Query Language Reference](#).

The projection includes `System.Timestamp`, which returns a timestamp for the end of each window.

To specify that you want to use a Tumbling window, you use the `TUMBLINGWINDOW` function in the `GROUP BY` clause. In the function, you specify a time unit (anywhere from a microsecond to a day) and a window size (how many units). In this example, the Tumbling window consists of 5-second intervals, so you will get a count by country/region for every 5 seconds' worth of calls.

2. Select **Test query**. In the results, notice that the timestamps under `WindowEnd` are in 5-second increments.

## Detect SIM fraud using a self-join

For this example, consider fraudulent usage to be calls that originate from the same user but in different locations within 5 seconds of one another. For example, the same user can't legitimately make a call from the US and Australia at the same time.

To check for these cases, you can use a self-join of the streaming data to join the stream to itself based on the `CallRecTime` value. You can then look for call records where the `CallingIMSI` value (the originating number) is the same, but the `SwitchNum` value (country/region of origin) is not the same.

When you use a join with streaming data, the join must provide some limits on how far the matching rows can be separated in time. As noted earlier, the streaming data is effectively endless. The time bounds for the

relationship are specified inside the `ON` clause of the join, using the `DATEDIFF` function. In this case, the join is based on a 5-second interval of call data.

- Paste the following query in the query editor:

```
SELECT System.Timestamp AS WindowEnd, COUNT(*) AS FraudulentCalls
INTO "MyPBIoutput"
FROM "CallStream" CS1 TIMESTAMP BY CallRecTime
JOIN "CallStream" CS2 TIMESTAMP BY CallRecTime
ON CS1.CallingIMSI = CS2.CallingIMSI
AND DATEDIFF(ss, CS1, CS2) BETWEEN 1 AND 5
WHERE CS1.SwitchNum != CS2.SwitchNum
GROUP BY TumblingWindow(Duration(second, 1))
```

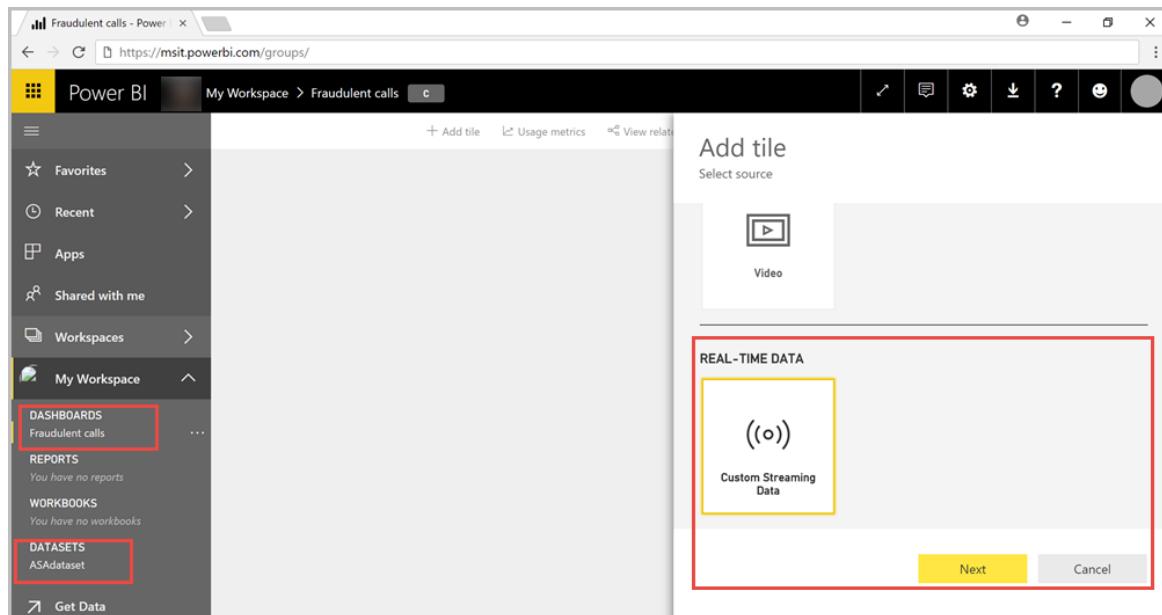
This query is like any SQL join except for the `DATEDIFF` function in the join. This version of `DATEDIFF` is specific to Streaming Analytics, and it must appear in the `ON...BETWEEN` clause. The parameters are a time unit (seconds in this example) and the aliases of the two sources for the join. This is different from the standard SQL `DATEDIFF` function.

The `WHERE` clause includes the condition that flags the fraudulent call: the originating switches are not the same.

- Select **Test query**. Review the output, and then select **Save query**.

## Start the job and visualize output

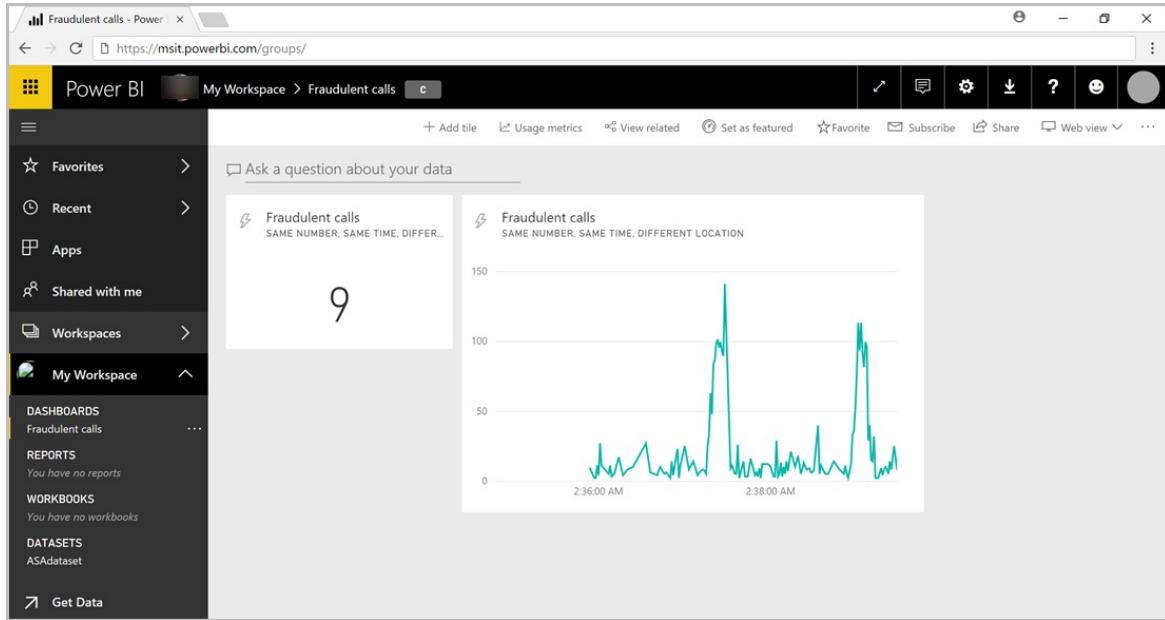
- To start the job, navigate to the job **Overview** and select **Start**.
- Select **Now** for job output start time and select **Start**. You can view the job status in the notification bar.
- Once the job succeeds, navigate to **Power BI** and sign in with your work or school account. If the Stream Analytics job query is outputting results, the `ASAdataset` dataset you created exists under the **Datasets** tab.
- From your Power BI workspace, select **+ Create** to create a new dashboard named *Fraudulent Calls*.
- At the top of the window, select **Edit** and **Add tile**. Then select **Custom Streaming Data** and **Next**. Choose the **ASAdataset** under **Your Datasets**. Select **Card** from the **Visualization type** dropdown, and add **fraudulent calls** to **Fields**. Select **Next** to enter a name for the tile, and then select **Apply** to create the tile.



6. Follow the step 5 again with the following options:

- When you get to Visualization Type, select Line chart.
- Add an axis and select **windowend**.
- Add a value and select **fraudulentcalls**.
- For **Time window to display**, select the last 10 minutes.

7. Your dashboard should look like the example below once both tiles are added. Notice that, if your event hub sender application and Streaming Analytics application are running, your Power BI dashboard periodically updates as new data arrives.



## Embedding your Power BI Dashboard in a Web Application

For this part of the tutorial, you'll use a sample [ASP.NET](#) web application created by the Power BI team to embed your dashboard. For more information about embedding dashboards, see [embedding with Power BI](#) article.

To set up the application, go to the [PowerBI-Developer-Samples](#) GitHub repository and follow the instructions under the **User Owns Data** section (use the redirect and homepage URLs under the **integrate-web-app** subsection). Since we are using the Dashboard example, use the **integrate-web-app** sample code located in the [GitHub repository](#). Once you've got the application running in your browser, follow these steps to embed the dashboard you created earlier into the web page:

1. Select **Sign in to Power BI**, which grants the application access to the dashboards in your Power BI account.
2. Select the **Get Dashboards** button, which displays your account's Dashboards in a table. Find the name of the dashboard you created earlier, **powerbi-embedded-dashboard**, and copy the corresponding **EmbedUrl**.
3. Finally, paste the **EmbedUrl** into the corresponding text field and select **Embed Dashboard**. You can now view the same dashboard embedded within a web application.

## Next steps

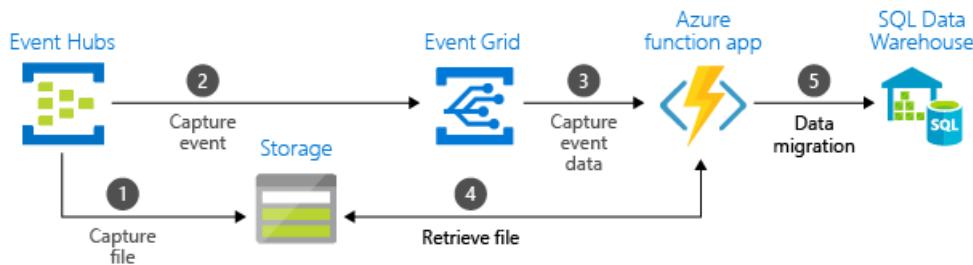
In this tutorial, you created a simple Stream Analytics job, analyzed the incoming data, and presented results in a Power BI dashboard. To learn more about Stream Analytics jobs, continue to the next tutorial:

[Run Azure Functions within Stream Analytics jobs](#)

# Tutorial: Migrate captured Event Hubs data to Azure Synapse Analytics using Event Grid and Azure Functions

11/26/2021 • 10 minutes to read • [Edit Online](#)

Azure Event Hubs [Capture](#) enables you to automatically capture the streaming data in Event Hubs in an Azure Blob storage or Azure Data Lake Storage. This tutorial shows you how to migrate captured Event Hubs data from Storage to Azure Synapse Analytics by using an Azure function that's triggered by [Event Grid](#).



This diagram depicts the workflow of the solution you build in this tutorial:

1. Data sent to an Azure event hub is captured in an Azure blob storage.
2. When the data capture is complete, an event is generated and sent to an Azure event grid.
3. The event grid forwards this event data to an Azure function app.
4. The function app uses the blob URL in the event data to retrieve the blob from the storage.
5. The function app migrates the blob data to an Azure Synapse Analytics.

In this article, you take the following steps:

- Deploy the required infrastructure for the tutorial
- Publish code to a Functions App
- Create an Event Grid subscription
- Stream sample data into Event Hubs
- Verify captured data in Azure Synapse Analytics

## Prerequisites

To complete this tutorial, you must have:

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- [Visual studio 2019](#) with workloads for: .NET desktop development, Azure development, ASP.NET and web development, Node.js development, and Python development.
- Download the [EventHubsCaptureEventGridDemo sample project](#) to your computer.
  - WindTurbineDataGenerator – A simple publisher that sends sample wind turbine data to a capture-enabled event hub
  - FunctionDWDumper – An Azure Function that receives an Event Grid notification when an Avro file is captured to the Azure Storage blob. It receives the blob's URI path, reads its contents, and pushes this data to Azure Synapse Analytics (dedicated SQL pool).

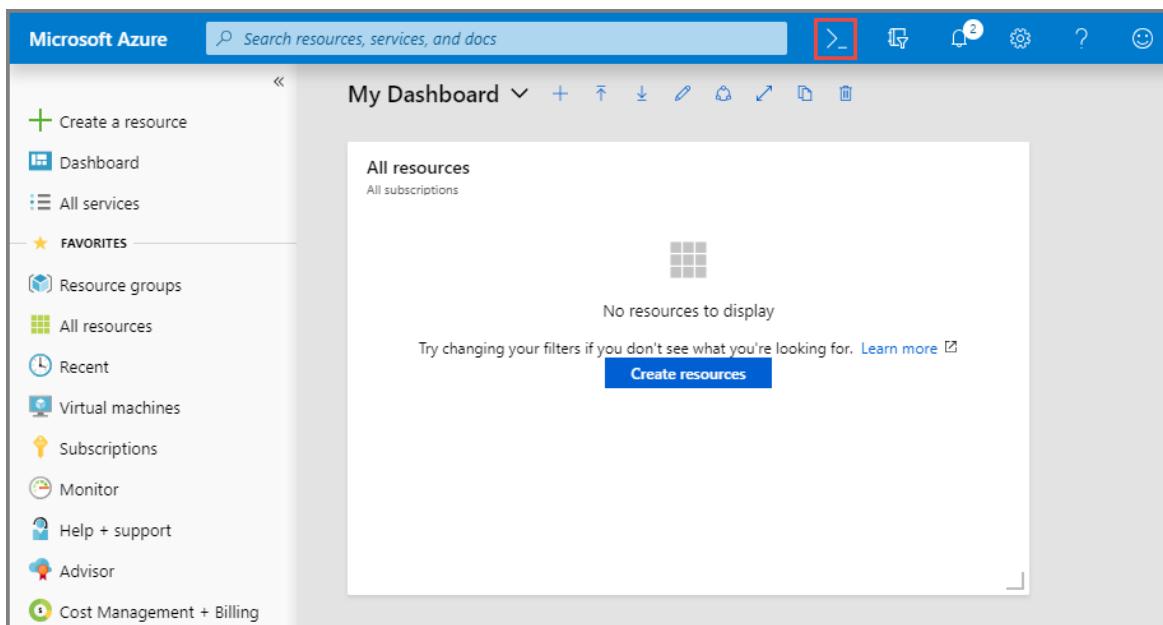
## Deploy the infrastructure

In this step, you deploy the required infrastructure with a [Resource Manager template](#). When you deploy the template, the following resources are created:

- Event hub with the Capture feature enabled.
- Storage account for the captured files.
- App service plan for hosting the function app
- Function app for processing the event
- SQL Server for hosting the data warehouse
- Azure Synapse Analytics (dedicated SQL pool) for storing the migrated data

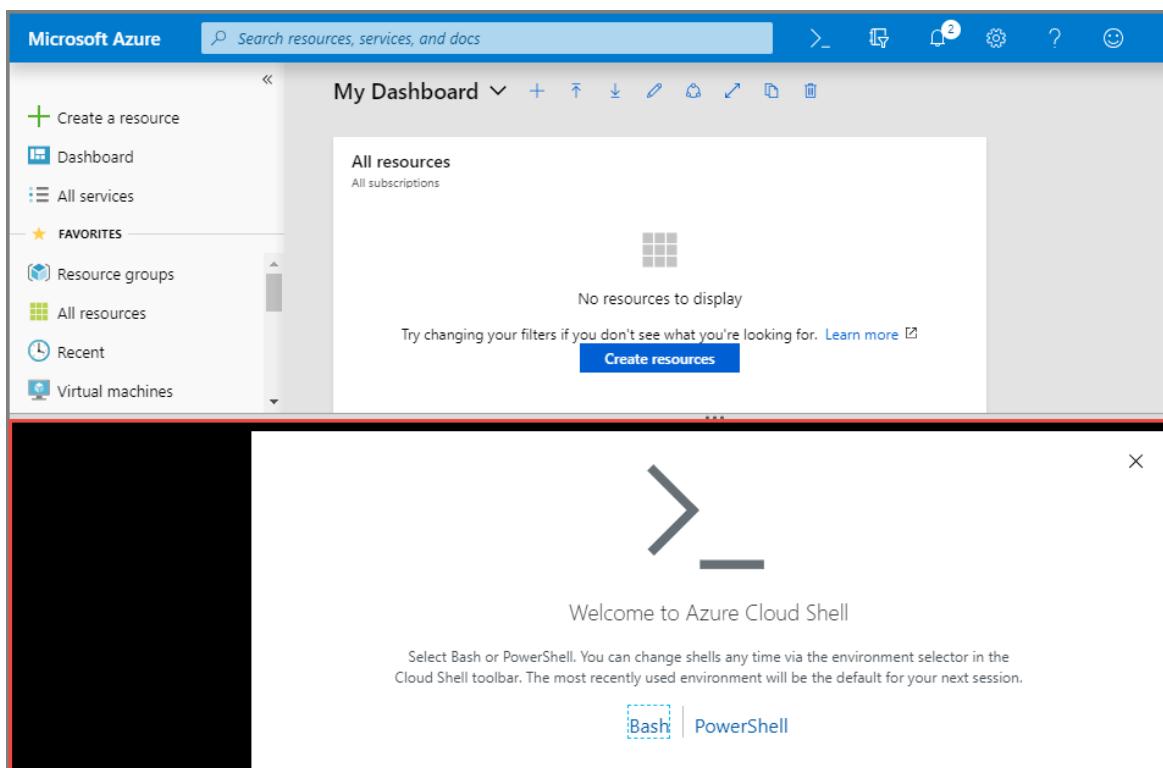
## Use Azure CLI to deploy the infrastructure

1. Sign in to the [Azure portal](#).
2. Select **Cloud Shell** button at the top.



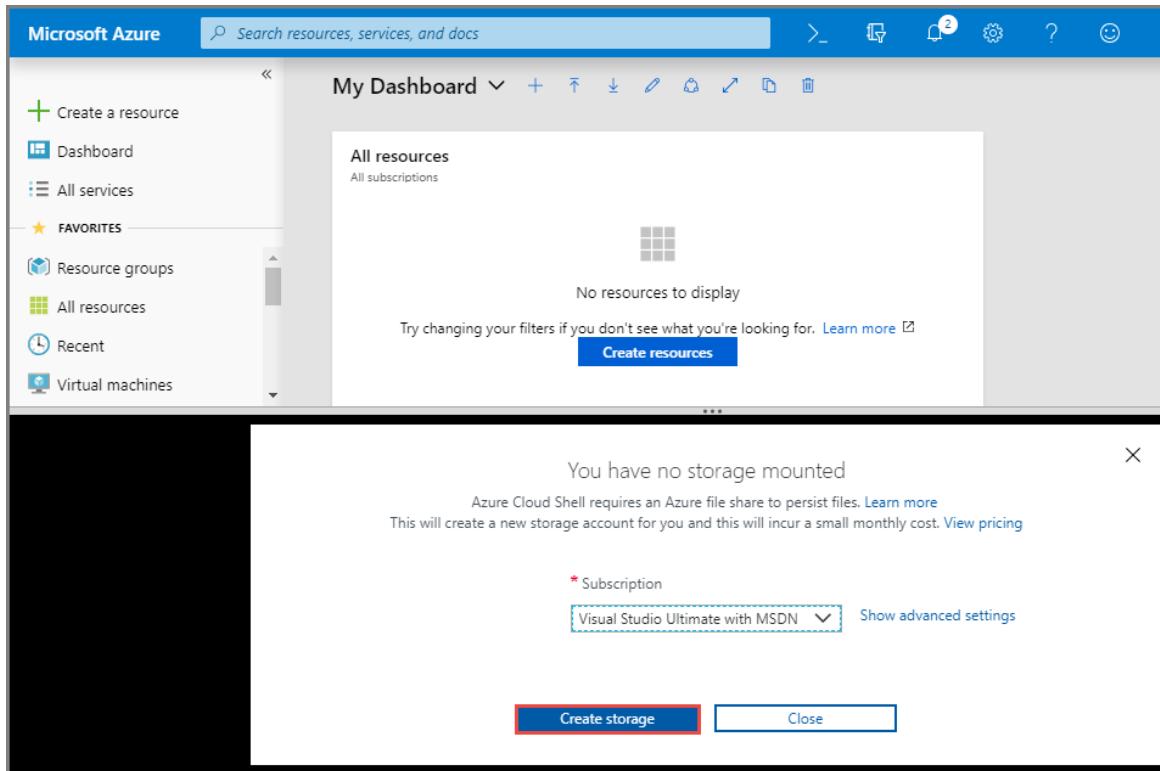
The screenshot shows the Microsoft Azure portal dashboard. On the left, there's a sidebar with various navigation links like 'Create a resource', 'Dashboard', 'All services', and 'FAVORITES' (which includes 'Resource groups', 'All resources', 'Recent', 'Virtual machines', 'Subscriptions', 'Monitor', 'Help + support', 'Advisor', and 'Cost Management + Billing'). The main area is titled 'My Dashboard' and shows a message 'No resources to display'. At the top right of the dashboard area, there's a blue button labeled 'Create resources'. Above the dashboard, there's a toolbar with icons for search, refresh, and other settings. A red box highlights the 'Cloud Shell' button, which is located at the very top right of the dashboard area.

3. You see the Cloud Shell opened at the bottom of the browser.

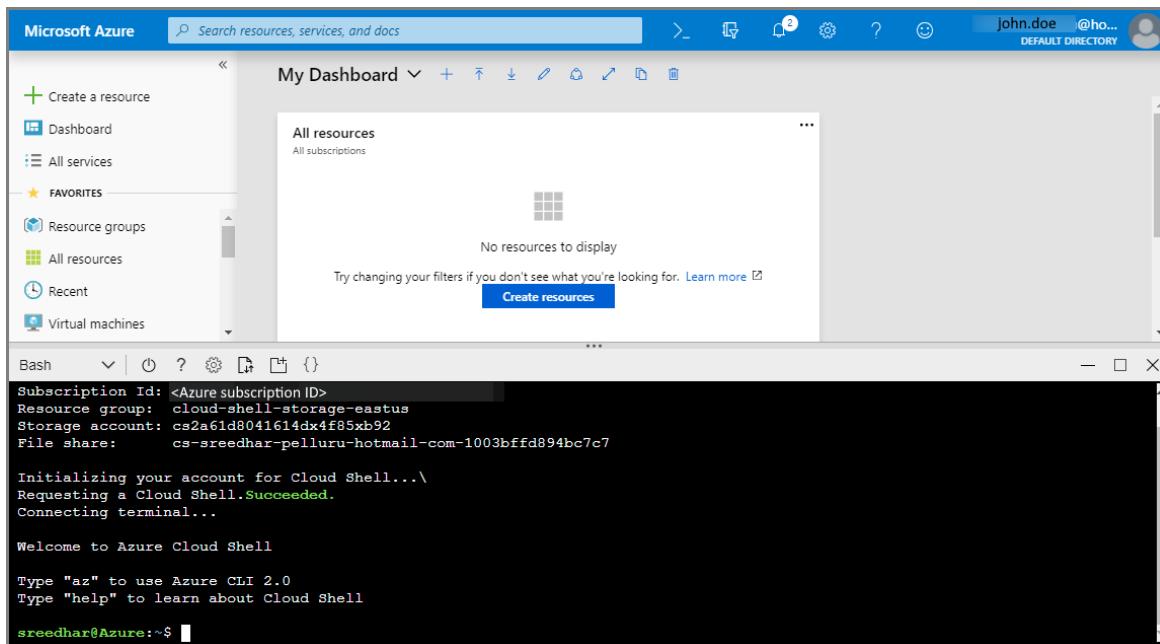


The screenshot shows the Microsoft Azure portal dashboard with the Cloud Shell interface open at the bottom. The dashboard itself is mostly empty, showing a message 'No resources to display'. The Cloud Shell interface is a large black window at the bottom with a white header. The header has the 'Cloud Shell' title and a close button. Below the header, it says 'Welcome to Azure Cloud Shell'. It then instructs the user to 'Select Bash or PowerShell. You can change shells any time via the environment selector in the Cloud Shell toolbar. The most recently used environment will be the default for your next session.' At the bottom of the Cloud Shell window, there are two buttons: 'Bash' (highlighted with a red border) and 'PowerShell'.

- In the Cloud Shell, if you see an option to select between **Bash** and **PowerShell**, select **Bash**.
- If you are using the Cloud Shell for the first time, create a storage account by selecting **Create storage**. Azure Cloud Shell requires an Azure storage account to store some files.



- Wait until the Cloud Shell is initialized.



- Create an Azure resource group by running the following CLI command:

- Copy and paste the following command into the Cloud Shell window. Change the resource group name and location if you want.

```
az group create -l eastus -n rgDataMigration
```

- Press **ENTER**.

Here is an example:

```
user@Azure:~$ az group create -l eastus -n rgDataMigration
{
  "id": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/rgDataMigration",
  "location": "eastus",
  "managedBy": null,
  "name": "rgDataMigration",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

8. Deploy all the resources mentioned in the previous section (event hub, storage account, functions app, Azure Synapse Analytics) by running the following CLI command:

- Copy and paste the command into the Cloud Shell window. Alternatively, you may want to copy/paste into an editor of your choice, set values, and then copy the command to the Cloud Shell.

#### IMPORTANT

Specify values for the following entities before running the command:

- Name of the resource group you created earlier.
- Name for the event hub namespace.
- Name for the event hub. You can leave the value as it is (hubdatamigration).
- Name for the SQL server.
- Name of the SQL user and password.
- Name for the database.
- Name of the storage account.
- Name for the function app.

```
az deployment group create \
  --resource-group rgDataMigration \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-
samples/master/event-grid/EventHubsDataMigration.json \
  --parameters eventHubNamespaceName=<event-hub-namespace> eventHubName=hubdatamigration
sqlServerName=<sql-server-name> sqlServerUserName=<user-name> sqlServerPassword=<password>
sqlServerDatabaseName=<database-name> storageName=<unique-storage-name> functionAppName=<app-
name>
```

- Press **ENTER** in the Cloud Shell window to run the command. This process may take a while since you are creating a bunch of resources. In the result of the command, ensure that there have been no failures.

9. Close the Cloud Shell by selecting the **Cloud Shell** button in the portal (or) X button in the top-right corner of the Cloud Shell window.

#### Verify that the resources are created

- In the Azure portal, select **Resource groups** on the left menu.
- Filter the list of resource groups by entering the name of your resource group in the search box.
- Select your resource group in the list.

All services >

## Resource groups

Default Directory

+ Add Manage view Refresh Export to CSV Open query Assign tags Feedback

Filter by name... Subscription == all Location == all Add filter

Showing 1 to 3 of 3 records.

Name	Subscription	Location
cloud-shell-storage-eastus	Visual Studio Ultimate with MSDN	East US
NetworkWatcherRG	Visual Studio Ultimate with MSDN	East US
rgDataMigration	Visual Studio Ultimate with MSDN	East US

No grouping List view

< Page 1 > of 1

4. Confirm that you see the following resources in the resource group:

rgDataMigration

Resource group

Search (Ctrl+J)

+ Add Edit columns Delete resource group Refresh Export to CSV Open query ...

Overview

Activity log Access control (IAM) Tags Events

Settings Deployments Policies Properties Locks

Cost Management Cost analysis Cost alerts (preview)

Essentials

Filter by name... Type == all Location == all Add filter

Showing 1 to 6 of 6 records. Show hidden types No grouping List view

Name	Type	Location
spehubfuncapp1207	Function App	East US
spehubfuncapp1207Plan	App Service plan	East US
spehubns1207	Event Hubs Namespace	East US
spehubsqldb1207 (spehubsqlsvr1207/spehubsql1207)	Dedicated SQL pool (formerly SQL DW)	East US
spehubsqlsvr1207	SQL server	East US
spehubstorage1207	Storage account	East US

### Create a table in Azure Synapse Analytics

Create a table in your data warehouse by running the [CreateDataWarehouseTable.sql](#) script. To run the script, you can use Visual Studio or the Query Editor in the portal. The following steps show you how to use the Query Editor:

1. In the list of resources in the resource group, select your **Dedicated SQL pool**.
2. On the **Dedicated SQL pool** page, in the **Common Tasks** section on the left menu, select **Query editor (preview)**.

The screenshot shows the Azure portal interface for a Dedicated SQL pool. The top navigation bar includes a search bar, pause, scale, restore, new restore point, delete, and a link to 'New Synapse workspace'. On the left, there's a sidebar with 'Dynamic Data Masking', 'Firewalls and virtual networks', 'Security Center', and 'Transparent data encryption'. Under 'Common Tasks', options include 'View streaming jobs', 'Load Data', and 'Query editor (preview)', which is highlighted with a red box. Other tasks like 'Build dashboards + reports', 'Model and cache data', and 'Open in Visual Studio' are also listed.

3. Enter the name of **user** and **password** for the SQL server, and select **OK**. If you see a message about allowing your client to access the SQL server, follow these steps:
  - a. Select the link: **Set server firewall**.
  - b. On the **Firewall settings** page, select **Add client IP** on the toolbar, and then select **Save** on the toolbar.
  - c. Select **OK** on the success message.
  - d. Navigate back to the **Dedicated SQL pool** page, and select **Query editor (preview)** on the left menu.
  - e. Enter **user** and **password**, and then select **OK**.
4. In the query window, copy and run the following SQL script:

```
CREATE TABLE [dbo].[Fact_WindTurbineMetrics] (
    [DeviceId] nvarchar(50) COLLATE SQL_Latin1_General_CI_AS NULL,
    [MeasureTime] datetime NULL,
    [GeneratedPower] float NULL,
    [WindSpeed] float NULL,
    [TurbineSpeed] float NULL
)
WITH (CLUSTERED COLUMNSTORE INDEX, DISTRIBUTION = ROUND_ROBIN);
```

The screenshot shows the Azure Data Studio interface. On the left, there's a sidebar with a connection named "spegridehubsqldb (sqluser)" and a note about limited object explorer. Below it are links for Tables, Views, and Stored Procedures. The main area is titled "Query 1" and contains a SQL script to create a table:

```

1  CREATE TABLE [dbo].[Fact_WindTurbineMetrics] (
2      [DeviceId] nvarchar(50) COLLATE SQL_Latin1_General_CI_AS NULL,
3      [MeasureTime] datetime NULL,
4      [GeneratedPower] float NULL,
5      [WindSpeed] float NULL,
6      [TurbineSpeed] float NULL
7  )
8  WITH (CLUSTERED COLUMNSTORE INDEX, DISTRIBUTION = ROUND_ROBIN);

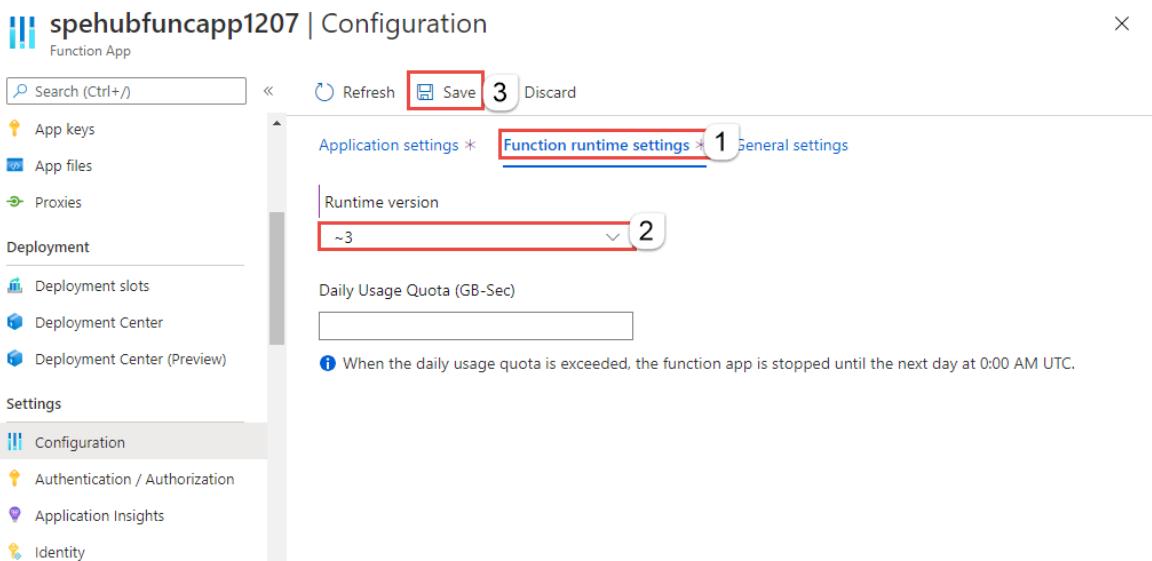
```

Below the query editor, there are tabs for "Results" and "Messages". The "Messages" tab shows the message "Query succeeded: Affected rows: 0". At the bottom, a yellow bar indicates "Query succeeded | 1s".

5. Keep this tab or window open so that you can verify that the data is created at the end of the tutorial.

### Update the function runtime version

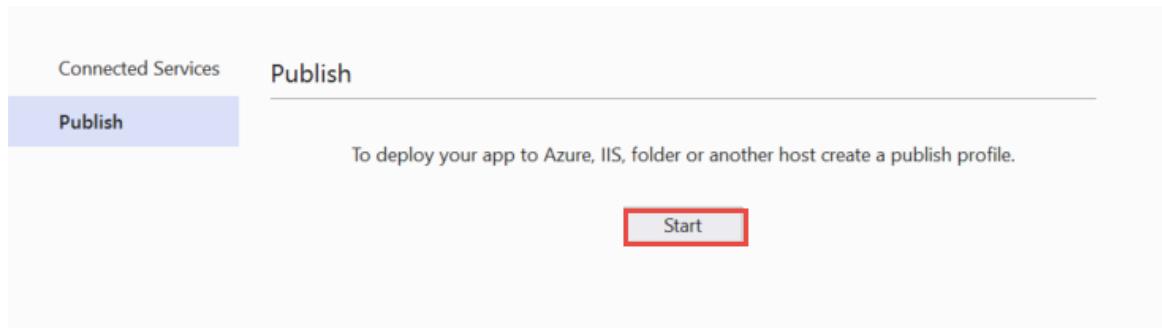
1. Open another tab in the web browser, and navigate to [Azure portal](#).
2. In the Azure portal, select **Resource groups** on the left menu.
3. Select the resource group in which the function app exists.
4. Select the **function app** in the list of resources in the resource group.
5. Select **Configuration** under **Settings** on the left menu.
6. Switch to the **Function runtime settings** tab in the right pane.
7. Update the **runtime version** to **~3**.



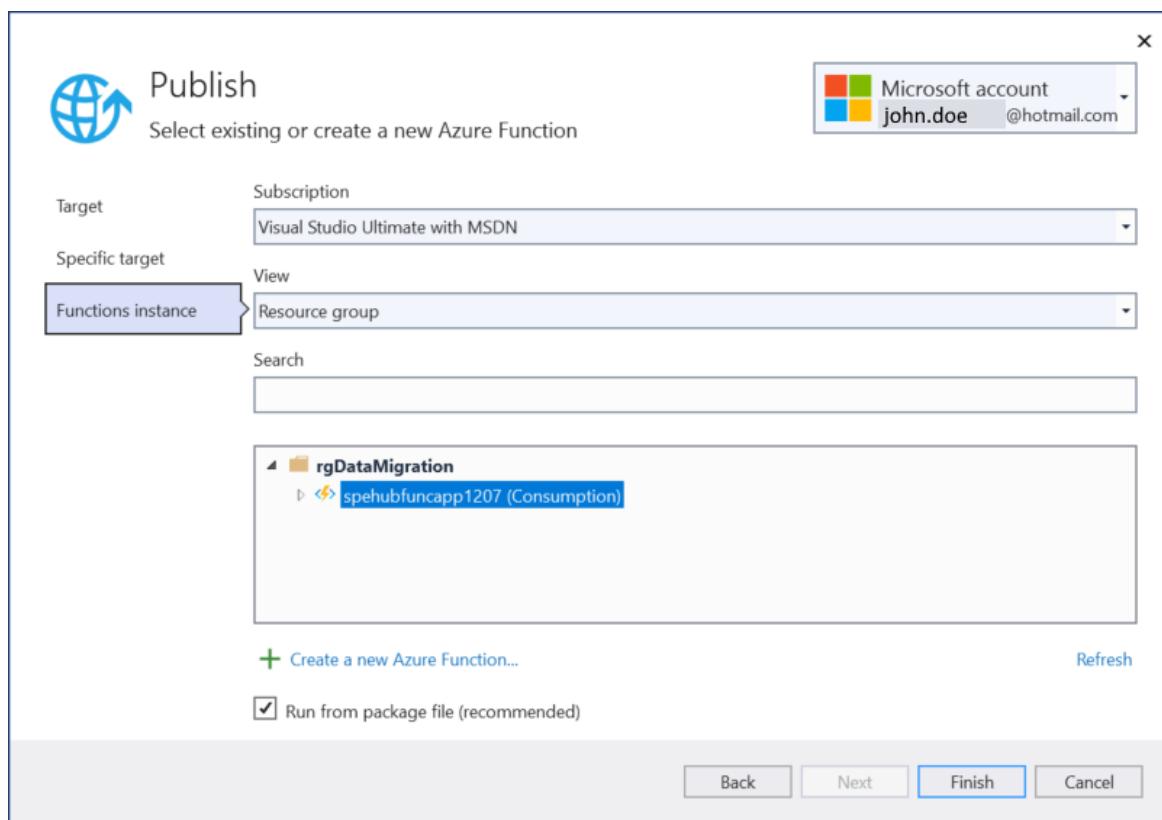
8. Select **Save** on the toolbar.
9. On the **Save changes** confirmation popup, select **Continue**.

# Publish the Azure Functions app

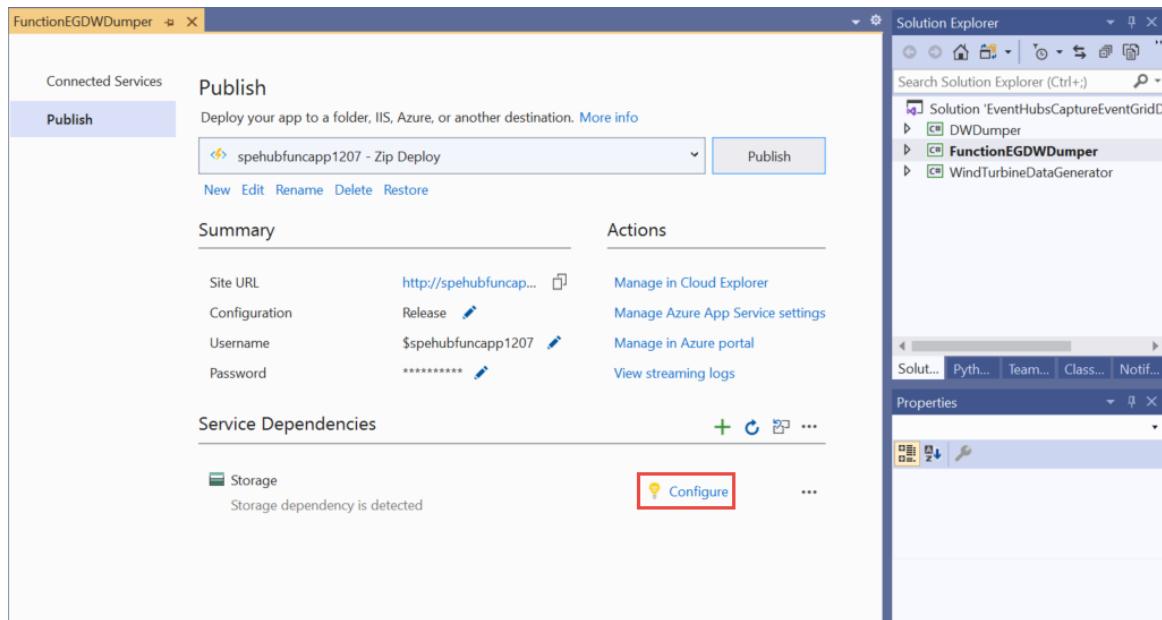
1. Launch Visual Studio.
2. Open the `EventHubsCaptureEventGridDemo.sln` solution that you downloaded from the [GitHub](#) as part of the prerequisites. You can find it in the `/samples/e2e/EventHubsCaptureEventGridDemo` folder.
3. In Solution Explorer, right-click `FunctionEGDWDumper` project, and select **Publish**.
4. If you see the following screen, select **Start**.



5. In the **Publish** dialog box, select **Azure** for **Target**, and select **Next**.
6. Select **Azure Function App (Windows)**, and select **Next**.
7. On the **Functions instance** tab, select your Azure subscription, expand the resource group, and select your function app, and then select **Finish**. You need to sign into your Azure account if you haven't already done so.

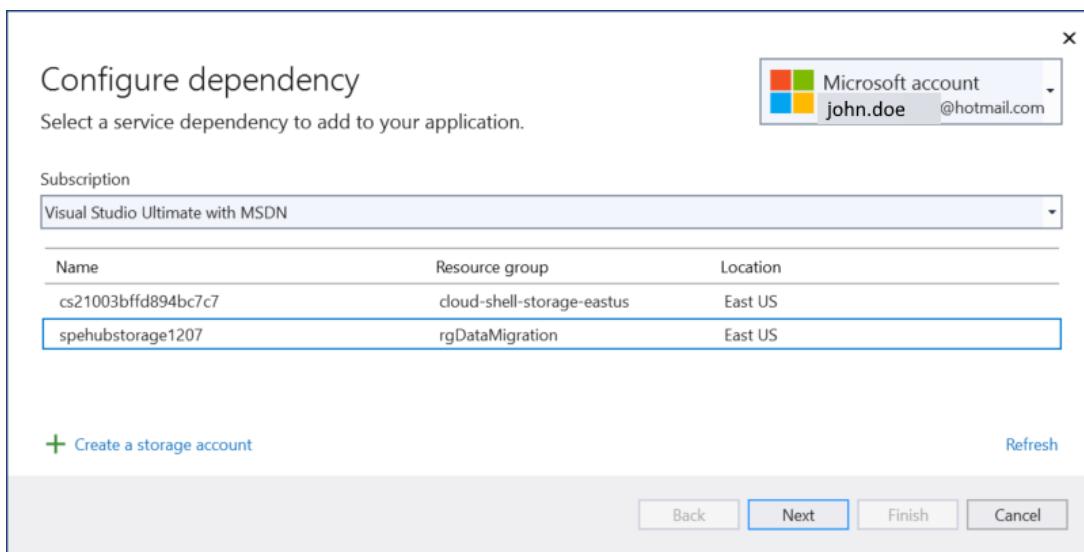


8. On the **Publish** page, in the **Service Dependencies** section, select **Configure for Storage**.

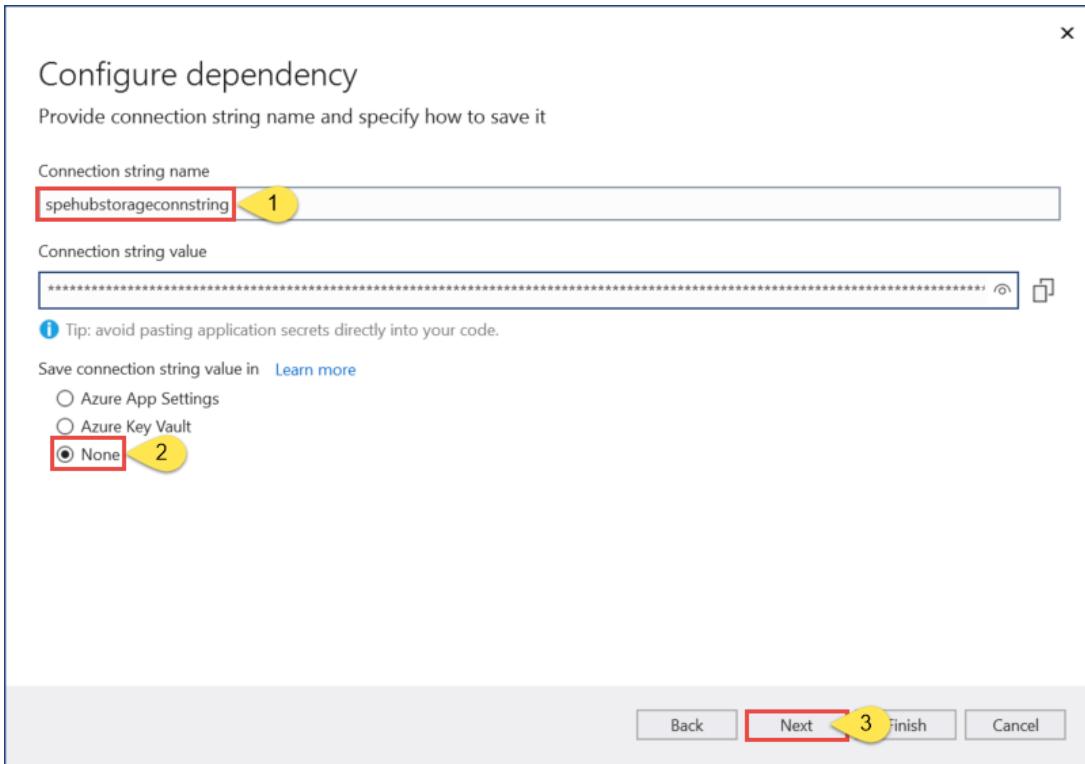


9. On the **Configure dependency** page, follow these steps:

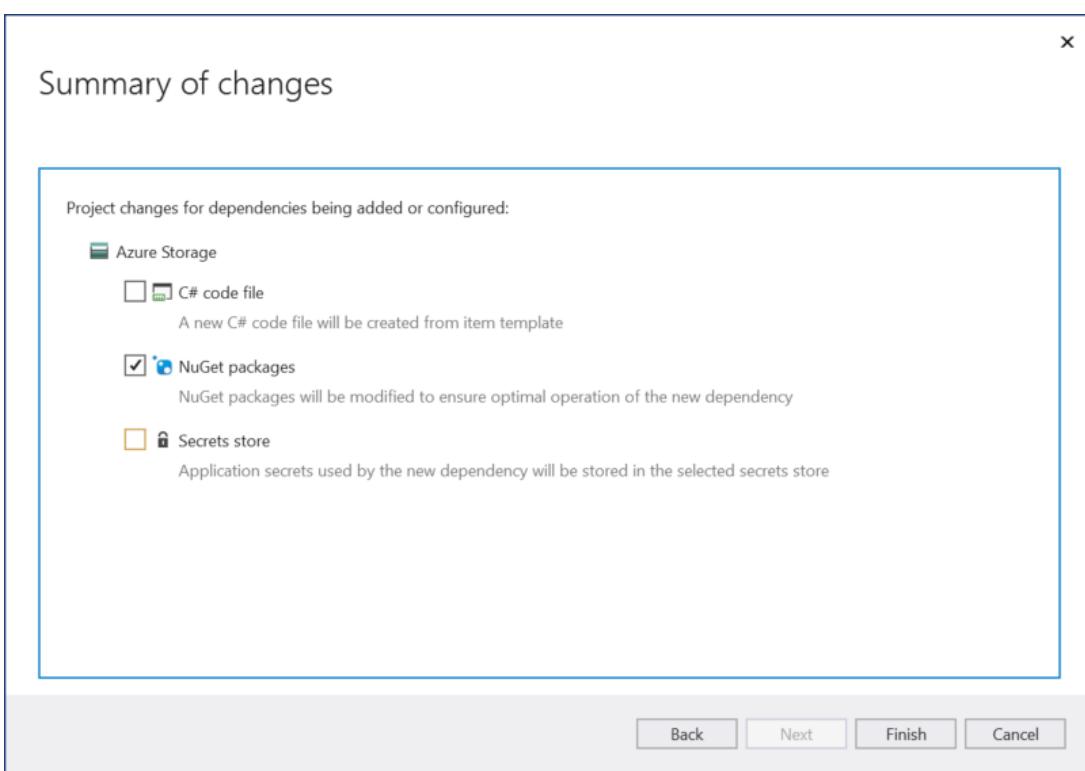
- a. select the **storage account** you created earlier, and then select **Next**.



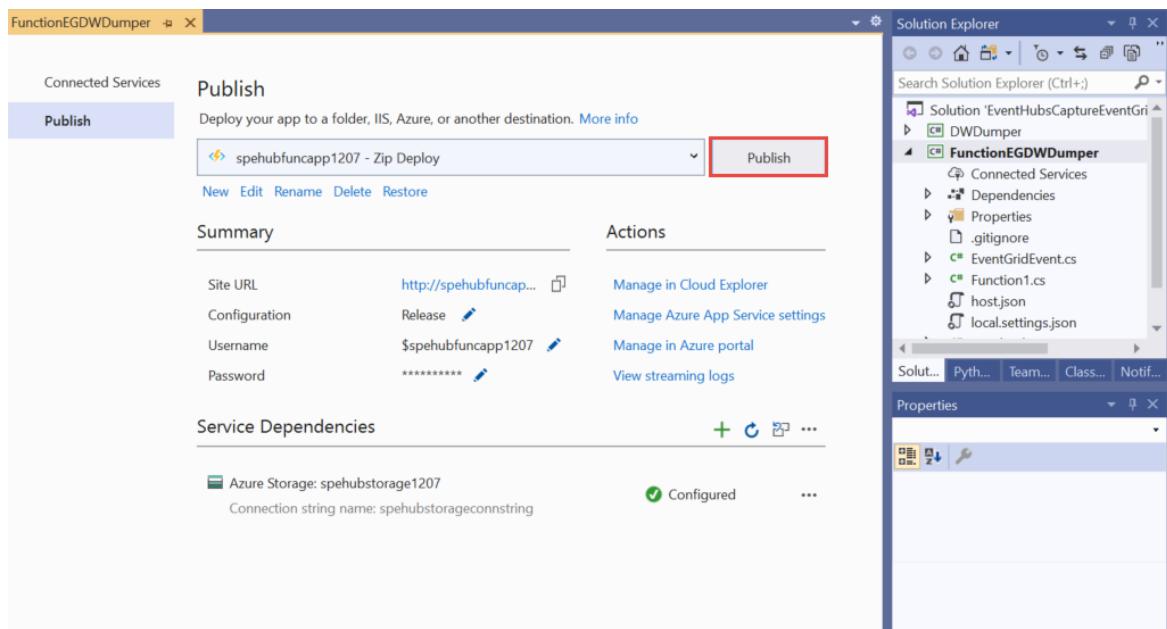
- b. Specify a name for the connection string, and select **None** for the **Save connection string** option, and then select **Next**.



- c. Clear the **C# code file** and **Secrets store** option, and then select **Finish**.



10. When Visual Studio has configured the profile, select **Publish**.



- In the tab that has the **Azure Function** page open, select **Functions** on the left menu. Confirm that the **EventGridTriggerMigrateData** function shows up in the list. If you don't see it, try publishing from Visual Studio again, and then refresh the page in the portal.

The screenshot shows the 'Functions' blade for the 'spehubfuncapp1207' function app. A red box highlights the 'Functions' link in the left sidebar. The main area displays a table of functions, with the first row ('EventGridTriggerMigrateData') highlighted by a red box. The table columns are Name, Trigger, and Status. The status for this function is 'Enabled'. A warning message above the table states: 'Your app is currently in read only mode because you are running from a package file. To make any changes update the content in your zip file and WEBSITE\_RUN\_FROM\_PACKAGE app setting.'

After publishing the function, you're ready to subscribe to the event.

## Subscribe to the event

- In a new tab or new window of a web browser, navigate to the [Azure portal](#).
- In the Azure portal, select **Resource groups** on the left menu.
- Filter the list of resource groups by entering the name of your resource group in the search box.
- Select your resource group in the list.
- Select the **Event Hubs namespace** from the list of resources.
- On the **Event Hubs Namespace** page, select **Events** on the left menu, and then select **+ Event Subscription** on the toolbar.

The screenshot shows the Azure portal interface for an Event Hubs Namespace named 'spehubns1207'. The left sidebar has a red box around the 'Events' item under the 'Settings' section. The main content area has a red box around the 'Event Subscriptions' tab. The page title is 'spehubns1207 | Events' and the sub-page title is 'Event Subscriptions'. A large heading 'Events, automated.' is displayed. Below it, a description states: 'Build reactive, event-driven apps with a fully managed event routing service that is built into Azure. Event Grid helps you build automation into your cloud infrastructure, create serverless apps, and integrate across services and clouds.' A 'Learn more' link is present. To the right, there's a section titled 'Example Scenarios' with three items: 'Execute code to resize an image whenever one is uploaded', 'Copy files with a certain extension to a different file store', and 'Process file content through a cognitive service and generate a metadata file'. At the bottom, a note says 'Azure Event Grid natively supports these resources as event handlers.' followed by a 'Learn more' link.

7. On the **Create Event Subscription** page, follow these steps:

- a. Enter a name for the **event subscription**.
- b. Enter a name for the **system topic**. A system topic provides an endpoint for the sender to send events. For more information, see [System topics](#)
- c. For **Endpoint Type**, select **Azure Function**.
- d. For **Endpoint**, select the link.
- e. On the **Select Azure Function** page, follow these steps if they aren't automatically filled.
  - a. Select the Azure subscription that has the Azure function.
  - b. Select the resource group for the function.
  - c. Select the function app.
  - d. Select the deployment slot.
  - e. Select the function **EventGridTriggerMigrateData**.
- f. On the **Select Azure Function** page, select **Confirm Selection**.
- g. Then, back on the **Create Event Subscription** page, select **Create**.

**Select Azure Function**

Subscription: Visual Studio Ultimate with MSDN

Resource group: rgDataMigration

Function app: spehubfuncapp1207

Slot: Production

Function: EventGridTriggerMigrateData

**Create Event Subscription**

**Event Grid**

**Basic Filters Additional Features**

**EVENT SUBSCRIPTION DETAILS**

Name: CaptureEventFuncSubscription

Event Schema: Event Grid Schema

**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. Learn more

Topic Type: Event Hubs Namespace

Source Resource: spehubs1207

System Topic Name: spehubstoragesystopic

**EVENT TYPES**

Pick which event types get pushed to your destination. Learn more

Filter to Event Types: Capture File Created

**ENDPOINT DETAILS**

Pick an event handler to receive your events. Learn more

Endpoint Type: Azure Function (change)

Endpoint: Select an endpoint

**Create**

**Confirm Selection**

8. Verify that the event subscription is created. Switch to the **Event Subscriptions** tab on the **Events** page for the Event Hubs namespace.

**spehubs1207 | Events**

Event Hubs Namespace

**Event Subscription**

**Get Started Event Subscriptions**

**Essentials**

Show metrics: General Errors Latency Dead-Letter For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

Metrics chart:

- Published Events (Sum) spehubstoragesystopic: 5
- Matched Events (Sum) spehubstoragesystopic: 4
- Delivered Events (Sum) spehubstoragesystopic: 2
- Dead Lettered Events (Sum) spehubstoragesystopic: 1
- Delivery Failed Events (Sum) spehubstoragesystopic: --
- Dropped Events (Sum) spehubstoragesystopic: --

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
CaptureEventFuncSubscription	AzureFunction			Microsoft.EventHub.CaptureFileCreated

9. Select the App Service plan (not the App Service) in the list of resources in the resource group.

## Run the app to generate data

You've finished setting up your event hub, dedicate SQL pool (formerly SQL Data Warehouse), Azure function app, and event subscription. Before running an application that generates data for event hub, you need to configure a few values.

- In the Azure portal, navigate to your resource group as you did earlier.
- Select the Event Hubs namespace.
- In the **Event Hubs Namespace** page, select **Shared access policies** on the left menu.

4. Select **RootManageSharedAccessKey** in the list of policies.

**spehubns1207 | Shared access policies**

Event Hubs Namespace

Search (Ctrl+ /) << Add

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events

Settings

Shared access policies

Scale

Geo-Recovery

Search to filter items...

Policy	Claims
RootManageSharedAccessKey	Manage, Send, Listen

5. Select the copy button next to the **Connection string-primary key** text box.

6. Go back to your Visual Studio solution.

7. Right-click **WindTurbineDataGenerator** project, and select Set as Startup project.

8. In the WindTurbineDataGenerator project, open **program.cs**.

9. Replace <EVENT HUBS NAMESPACE CONNECTION STRING> with the connection string you copied from the portal.

10. Replace <EVENT HUB NAME> with the name of the event hub.

```
private const string EventHubConnectionString =
"Endpoint=sb://demomigrationnamespace.servicebus.windows.net/...";
private const string EventHubName = "hubdatamigration";
```

11. Build the solution. Run the `WindTurbineGenerator.exe` application.

12. After a couple of minutes, in the other browser tab where you have the query window open, query the table in your data warehouse for the migrated data.

```
select * from [dbo].[Fact_WindTurbineMetrics]
```

The screenshot shows the Azure SQL Database Query editor interface. The left sidebar contains navigation links for connection strings, properties, locks, security, auditing, data discovery & classification, dynamic data masking, firewalls, virtual networks, security center, and transparent data encryption. Below these are common tasks like viewing streaming jobs, loading data, using the query editor (which is currently selected), building dashboards, modeling data, and opening in Visual Studio. The monitoring section includes query activity and alerts.

The main area displays a query titled "Query 1" with the following content:

```
1 select * from [Fact_WindTurbineMetrics]
```

A message in the object explorer states: "Showing limited object explorer here. For full capability please open SSDT."

The results pane shows a table with the following data:

DeviceId	MeasureTime	GeneratedPower	WindSpeed	TurbineSpeed
Turbine_0	2020-12-07T21:28:56.393...	47.5	285	5.7000028610229
Turbine_57	2020-12-07T21:29:11.137...	47.5	285	5.7000028610229
Turbine_0	2020-12-07T21:29:13.1363...	42.5	255	5.1000038146973
Turbine_0	2020-12-07T21:29:13.277...	27.5	165	3.3000019073486
Turbine_57	2020-12-07T21:29:24.303...	47.5	285	5.7000028610229
Turbine_14	2020-12-07T21:29:37.263...	15	90	1.8000007152557

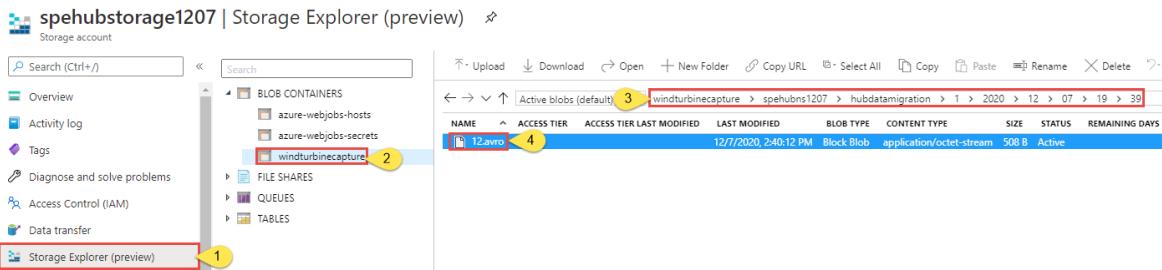
The status bar at the bottom indicates "Query succeeded | 0s".

# Monitor the solution

This section helps you with monitoring or troubleshooting the solution.

## View captured data in the storage account

1. Navigate to the resource group and select the storage account used for capturing event data.
2. On the **Storage account** page, select **Storage Explorer (preview)** on the left menu.
3. Expand **BLOB CONTAINERS**, and select **windturbinecapture**.
4. Open the folder named same as your **Event Hubs namespace** in the right pane.
5. Open the folder named same as your event hub (**hubdatamigration**).
6. Drill through the folders and you see the AVRO files. Here's an example:



## Verify that the Event Grid trigger invoked the function

1. Navigate to the resource group and select the function app.
2. Select **Functions** on the left menu.
3. Select the **EventGridTriggerMigrateData** function from the list.
4. On the **Function** page, select **Monitor** on the left menu.
5. Select **Configure** to configure application insights to capture invocation logs.
6. Create a new **Application Insights** resource or use an existing resource.
7. Navigate back to the **Monitor** page for the function.
8. Confirm that the client application (**WindTurbineDataGenerator**) that's sending the events is still running. If not, run the app.
9. Wait for a few minutes (5 minutes or more) and select the **Refresh** button to see function invocations.

The screenshot shows the Azure Function Monitor for the 'EventGridTriggerMigrateData' function. The 'Monitor' tab is selected. At the top, there are summary metrics: Success Count (0) and Error Count (0) for the last 30 days. Below this, the 'Invocation Traces' section displays a table of 20 recent invocations. Each row includes the date (UTC), success status (green checkmark for success), result code (e.g., 202), and duration (ms). The table is as follows:

Date (UTC)	Success	Result Code	Duration (ms)
2020-12-07 21:50:12.881	✓ Success	202	11
2020-12-07 21:50:12.326	✓ Success	202	7
2020-12-07 21:49:12.576	✓ Success	202	7
2020-12-07 21:49:12.306	✓ Success	202	11
2020-12-07 21:48:12.549	✓ Success	202	5
2020-12-07 21:48:12.499	✓ Success	202	16
2020-12-07 21:47:22.007	✓ Success	202	902
2020-12-07 21:47:22.007	✓ Success	202	909

#### 10. Select an invocation to see details.

Event Grid distributes event data to the subscribers. The following example shows event data generated when data streaming through an event hub is captured in a blob. In particular, notice the `fileUrl` property in the `data` object points to the blob in the storage. The function app uses this URL to retrieve the blob file with captured data.

```
{
  "topic": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourcegroups/rgDataMigration/providers/Microsoft.EventHub/namespaces/spehubns1207",
  "subject": "hubdatamigration",
  "eventType": "Microsoft.EventHub.CaptureFileCreated",
  "id": "4538f1a5-02d8-4b40-9f20-36301ac976ba",
  "data": {
    "fileUrl":
      "https://spehubstorage1207.blob.core.windows.net/windturbinecapture/spehubns1207/hubdatamigration/0/2020/12/07/21/49/12.avro",
    "fileType": "AzureBlockBlob",
    "partitionId": "0",
    "sizeInBytes": 473444,
    "eventCount": 2800,
    "firstSequenceNumber": 55500,
    "lastSequenceNumber": 58299,
    "firstEnqueueTime": "2020-12-07T21:49:12.556Z",
    "lastEnqueueTime": "2020-12-07T21:50:11.534Z"
  },
  "dataVersion": "1",
  "metadataVersion": "1",
  "eventTime": "2020-12-07T21:50:12.7065524Z"
}
```

#### Verify that the data is stored in the dedicated SQL pool

In the browser tab where you have the query window open, query the table in your dedicated SQL pool for the migrated data.

The screenshot shows the Azure Synapse Analytics Query editor (preview) interface. On the left, there's a navigation sidebar with sections like Connection strings, Properties, Locks, Security, Auditing, Data Discovery & Classification, Dynamic Data Masking, Firewalls and virtual networks, Security Center, and Transparent data encryption. Below that is a Common Tasks section with View streaming jobs, Load Data, Query editor (preview), Build dashboards + reports, Model and cache data, Open in Visual Studio, and Monitoring. The main area has tabs for Login, New Query, Open query, and Feedback. A search bar at the top says "Search (Ctrl+ /)". The main content area has a title "Query 1" and a message "Showing limited object explorer here. For full capability, please open SSDT.". It contains a code editor with the query "select \* from [Fact\_WindTurbineMetrics]". Below the code editor is a results grid with columns: DeviceId, MeasureTime, GeneratedPower, WindSpeed, and TurbineSpeed. The results show data for turbines 0, 57, and 14 across different dates and times. At the bottom, a yellow bar indicates "Query succeeded | 0s".

DeviceId	MeasureTime	GeneratedPower	WindSpeed	TurbineSpeed
Turbine_0	2020-12-07T21:28:56.393...	47.5	285	5.7000028610229
Turbine_57	2020-12-07T21:29:11.137...	47.5	285	5.7000028610229
Turbine_0	2020-12-07T21:29:01.363...	42.5	255	5.1000038146973
Turbine_0	2020-12-07T21:29:13.277...	27.5	165	3.3000019073486
Turbine_57	2020-12-07T21:29:24.303...	47.5	285	5.7000028610229
Turbine_14	2020-12-07T21:29:37.263...	15	90	1.8000007152557

## Next steps

You can use powerful data visualization tools with your data warehouse to achieve actionable insights.

This article shows how to use [Power BI with Azure Synapse Analytics](#)

# Tutorial: Process Apache Kafka for Event Hubs events using Stream analytics

11/26/2021 • 5 minutes to read • [Edit Online](#)

This article shows how to stream data into Event Hubs and process it with Azure Stream Analytics. It walks you through the following steps:

1. Create an Event Hubs namespace.
2. Create a Kafka client that sends messages to the event hub.
3. Create a Stream Analytics job that copies data from the event hub into an Azure blob storage.

You do not need to change your protocol clients or run your own clusters when you use the Kafka endpoint exposed by an event hub. Azure Event Hubs supports [Apache Kafka version 1.0](#) and above.

## Prerequisites

To complete this quickstart, make sure you have the following prerequisites:

- An Azure subscription. If you do not have one, create a [free account](#) before you begin.
- [Java Development Kit \(JDK\) 1.7+](#).
- [Download](#) and [install](#) a Maven binary archive.
- [Git](#)
- An [Azure Storage account](#). If you don't have one, [create one](#) before proceeding further. The Stream Analytics job in this walkthrough stores the output data in an Azure blob storage.

## Create an Event Hubs namespace

When you create an Event Hubs namespace, the Kafka endpoint for the namespace is automatically enabled. You can stream events from your applications that use the Kafka protocol into event hubs. Follow step-by-step instructions in the [Create an event hub using Azure portal](#) to create an Event Hubs namespace. If you are using a dedicated cluster, see [Create a namespace and event hub in a dedicated cluster](#).

### NOTE

Event Hubs for Kafka isn't supported in the **basic** tier.

## Send messages with Kafka in Event Hubs

1. Clone the [Azure Event Hubs for Kafka repository](#) to your machine.
2. Navigate to the folder: `azure-event-hubs-for-kafka/quickstart/java/producer`.
3. Update the configuration details for the producer in `src/main/resources/producer.config`. Specify the **name** and **connection string** for the **event hub namespace**.

```
bootstrap.servers={EVENT HUB NAMESPACE}.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{CONNECTION STRING for EVENT HUB NAMESPACE}";
```

4. Navigate to `azure-event-hubs-for-kafka/quickstart/java/producer/src/main/java/`, and open **TestDataReporter.java** file in an editor of your choice.

5. Comment out the following line of code:

```
//final ProducerRecord<Long, String> record = new ProducerRecord<Long, String>(TOPIC,
time, "Test Data " + i);
```

6. Add the following line of code in place of the commented code:

```
final ProducerRecord<Long, String> record = new ProducerRecord<Long, String>(TOPIC, time,
"{ \"eventData\": \"Test Data " + i + "\" }");
```

This code sends the event data in **JSON** format. When you configure input for a Stream Analytics job, you specify **JSON** as the format for the input data.

7. **Run the producer** and stream into Event Hubs. On a Windows machine, when using a **Node.js command prompt**, switch to the `azure-event-hubs-for-kafka/quickstart/java/producer` folder before running these commands.

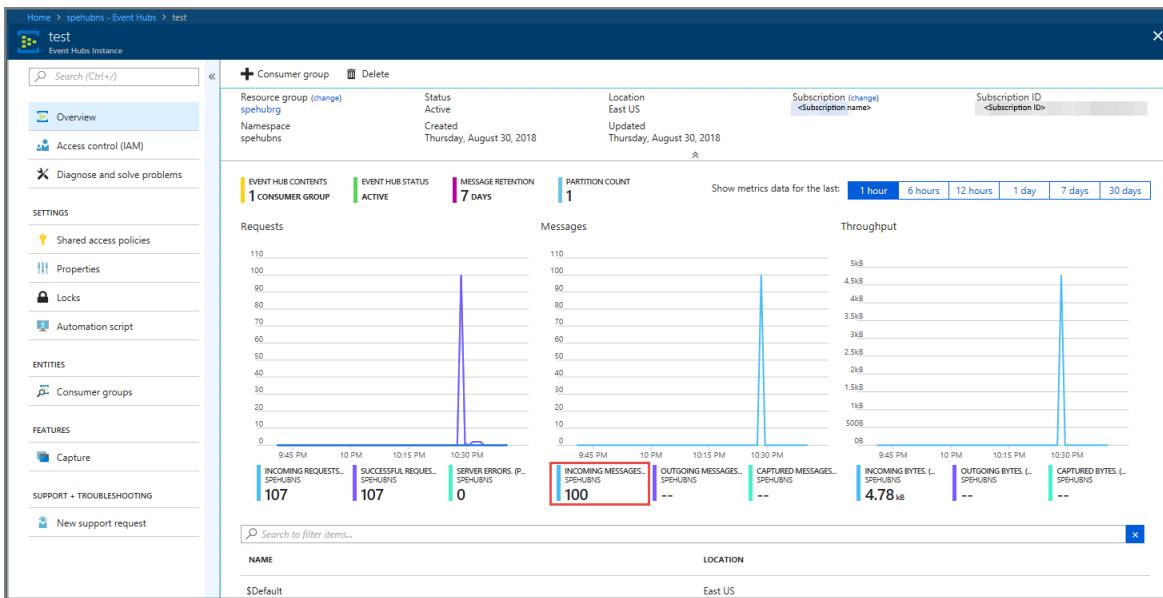
```
mvn clean package
mvn exec:java -Dexec.mainClass="TestProducer"
```

## Verify that event hub receives the data

1. Select **Event Hubs** under **ENTITIES**. Confirm that you see an event hub named **test**.

NAME	STATUS	MESSAGE RETENTION	PARTITION COUNT
test	Active	7	1

2. Confirm that you see messages coming in to the event hub.



## Process event data using a Stream Analytics job

In this section, you create an Azure Stream Analytics job. The Kafka client sends events to the event hub. You create a Stream Analytics job that takes event data as input and outputs it to an Azure blob storage. If you don't have an [Azure Storage account](#), [create one](#).

The query in the Stream Analytics job passes through the data without performing any analytics. You can create a query that transforms the input data to produce output data in a different format or with gained insights.

### Create a Stream Analytics job

1. Select [+ Create a resource](#) in the [Azure portal](#).
2. Select **Analytics** in the [Azure Marketplace](#) menu, and select **Stream Analytics job**.
3. On the **New Stream Analytics** page, do the following actions:
  - a. Enter a **name** for the job.
  - b. Select your **subscription**.
  - c. Select **Create new** for the **resource group** and enter the name. You can also use [an existing](#) resource group.
  - d. Select a **location** for the job.
  - e. Select **Create** to create the job.

Job name: spstreamanalyticsjob

Subscription: <Your Azure subscription>

Resource group: Create new: spsajobrg

Location: East US

Hosting environment: Cloud

Streaming units (1 to 120): 3

Create   Automation options

## Configure job input

- In the notification message, select **Go to resource** to see the Stream Analytics job page.
- Select **Inputs** in the **JOB TOPOLOGY** section on the left menu.
- Select **Add stream input**, and then select **Event Hub**.

SOURCE TYPE	SOURCE

- On the **Event Hub input** configuration page, do the following actions:
  - Specify an **alias** for the input.
  - Select your **Azure subscription**.

c. Select the **event hub namespace** you created earlier.

d. Select **test** for the **event hub**.

e. Select **Save**.

The screenshot shows the 'Event Hub' configuration dialog box. It has a title bar 'Event Hub' and a close button 'X'. Below the title is a section for 'New input'. The first field is 'Input alias' with the value 'myeventhub' highlighted in purple. Below it is a radio button group for 'Provide Event Hub settings manually' (unchecked) and 'Select Event Hub from your subscriptions' (checked). The next section is 'Subscription' with a dropdown menu showing '<Your Azure subscription>'. The following sections are: 'Event Hub namespace' (selected value 'spehubsns'), 'Event Hub name' (radio buttons for 'Create new' (unchecked) and 'Use existing' (checked), with a dropdown menu showing 'test'), 'Event Hub policy name' (selected value 'RootManageSharedAccessKey'), 'Event Hub policy key' (a redacted password), 'Event Hub consumer group' (a redacted consumer group name), 'Event serialization format' (selected value 'JSON'), 'Encoding' (selected value 'UTF-8'), and 'Event compression type' (selected value 'None'). At the bottom is a blue 'Save' button.

### Configure job output

1. Select **Outputs** in the **JOB TOPOLOGY** section on the menu.

2. Select **+ Add** on the toolbar, and select **Blob storage**

3. On the Blob storage output settings page, do the following actions:

a. Specify an **alias** for the output.

b. Select your **Azure subscription**.

c. Select your **Azure Storage account**.

d. Enter a **name for the container** that stores the output data from the Stream Analytics query.

e. Select **Save**.

## Define a query

After you have a Stream Analytics job setup to read an incoming data stream, the next step is to create a transformation that analyzes data in real time. You define the transformation query by using [Stream Analytics Query Language](#). In this walkthrough, you define a query that passes through the data without performing any transformation.

1. Select **Query**.
2. In the query window, replace `[YourOutputAlias]` with the output alias you created earlier.
3. Replace `[YourInputAlias]` with the input alias you created earlier.
4. Select **Save** on the toolbar.

```

1. SELECT
2. *
3. INTO
4. [myblobstorage]
5. FROM
6. [myeventhub]

```

## Run the Stream Analytics job

1. Select **Overview** on the left menu.
2. Select **Start**.

Home > spstreamanalyticsjob

**spstreamanalyticsjob**  
Stream Analytics job

Search (Ctrl+ /)

Start | Stop | Delete

**Overview** (highlighted with a red box)

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

SETTINGS

Locks

JOB TOPOLOGY

Inputs

Functions

Query

Outputs

CONFIGURE

Scale

Locale

Event ordering

Error policy

Compatibility level

GENERAL

Tools

Resource group (change) spsajobrg  
Status Stopped  
Location East US  
Subscription (change)  
Subscription ID [REDACTED]

Send feedback UserVoice  
Created Wednesday, August 29, 2018 10:58:57 PM  
Started Wednesday, August 29, 2018 11:22:56 PM  
Last output Wednesday, August 29, 2018 11:27:24 PM  
Hosting environment Cloud

Inputs  
1 myeventhub

Outputs  
1 myblobstorage

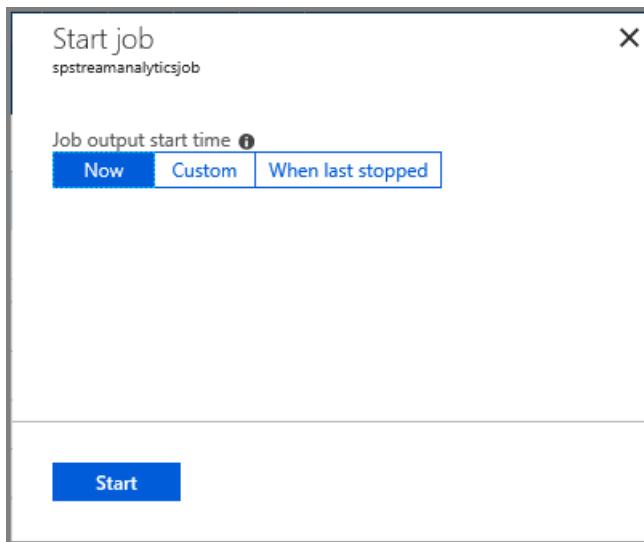
Query

```
1 SELECT *
2 INTO myblobstorage
3 FROM myeventhub
```

Monitoring

Resource utilization

- On the Start job page, select Start.



- Wait until the status of the job changes from Starting to running.

Home > spstreamanalyticsjob

**spstreamanalyticsjob**  
Stream Analytics job

Search (Ctrl+ /)

Start | Stop | Delete

**Running** (highlighted with a red box)

Resource group (change) spsajobrg  
Status Running  
Location East US  
Created Wednesday, August 29, 2018 10:58:57 PM  
Started Thursday, August 30, 2018 12:43:26 PM

Send feedback UserVoice

## Test the scenario

- Run the Kafka producer again to send events to the event hub.

```
mvn exec:java -Dexec.mainClass="TestProducer"
```

- Confirm that you see output data is generated in the Azure blob storage. You see a JSON file in the

container with 100 rows that look like the following sample rows:

```
{"eventData":"Test Data 0","EventProcessedUtcTime":"2018-08-30T03:27:23.1592910Z","PartitionId":0,"EventEnqueuedUtcTime":"2018-08-30T03:27:22.9220000Z"}  
{"eventData":"Test Data 1","EventProcessedUtcTime":"2018-08-30T03:27:23.3936511Z","PartitionId":0,"EventEnqueuedUtcTime":"2018-08-30T03:27:22.9220000Z"}  
{"eventData":"Test Data 2","EventProcessedUtcTime":"2018-08-30T03:27:23.3936511Z","PartitionId":0,"EventEnqueuedUtcTime":"2018-08-30T03:27:22.9220000Z"}
```

The Azure Stream Analytics job received input data from the event hub and stored it in the Azure blob storage in this scenario.

## Next steps

In this article, you learned how to stream into Event Hubs without changing your protocol clients or running your own clusters. To learn more about Event Hubs for Apache Kafka, see [Apache Kafka developer guide for Azure Event Hubs](#).

# Git repositories with samples for Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

You can find Event Hubs samples on [GitHub](#). These samples demonstrate key features in [Azure Event Hubs](#). This article categorizes and describes the samples available, with links to each.

## .NET samples

VERSION	SAMPLES LOCATION
Azure.Messaging.EventHubs version 5 (latest)	<a href="#">Event Hubs samples on GitHub</a> <a href="#">Event Hubs Processor samples on GitHub</a>
Microsoft.Azure.EventHubs version 4 (legacy)	<a href="#">GitHub location</a>

## Java samples

VERSION	SAMPLES LOCATION
azure-messaging-eventhubs version 5 (latest)	<a href="#">GitHub location</a>
azure-eventhubs version 3 (legacy)	<a href="#">GitHub location</a>

## Python samples

VERSION	SAMPLES LOCATION
azure-eventhub version 5 (latest)	<a href="#">GitHub location</a>
azure-eventhub version 1 (legacy)	<a href="#">GitHub location</a>

## JavaScript samples

VERSION	SAMPLES LOCATION
azure/event-hubs version 5 (latest)	<a href="#">GitHub location</a>
azure/event-hubs version 2 (legacy)	<a href="#">GitHub location</a>

## Go samples

You can find Go samples for Azure Event Hubs in the [azure-event-hubs-go](#) GitHub repository.

## Azure CLI samples

You can find Azure CLI samples for Azure Event Hubs in the [azure-event-hubs](#) GitHub repository.

## Azure PowerShell samples

You can find Azure PowerShell samples for Azure Event Hubs in the [azure-event-hubs](#) GitHub repository.

## Apache Kafka samples

You can find samples for the Event Hubs for Apache Kafka feature in the [azure-event-hubs-for-kafka](#) GitHub repository.

## Next steps

You can learn more about Event Hubs in the following articles:

- [Event Hubs overview](#)
- [Event Hubs features](#)
- [Event Hubs FAQ](#)

# Overview of Event Hubs Premium

11/26/2021 • 4 minutes to read • [Edit Online](#)

The Event Hubs Premium tier is designed for high-end streaming scenarios that require elastic, superior performance with predictable latency. The performance is achieved by providing reserved compute, memory, and storage resources, which minimize cross-tenant interference in a managed multi-tenant PaaS environment.

Event Hubs Premium introduces a new, two-tier, native-code log engine that provides far more predictable and much lower send and passthrough latencies than the prior generation, without any durability compromises.

Event Hubs Premium replicates every event to three replicas, distributed across Azure availability zones where available, and all replicas are synchronously flushed to the underlying fast storage before the send operation is reported as completed. Events that are not read immediately or that need to be re-read later can be retained up to 90 days, transparently held in an availability-zone redundant storage tier. Events in both the fast storage and retention storage tiers are encrypted; in Event Hubs Premium, the encryption keys can be supplied by you.

In addition to these storage-related features and all capabilities and protocol support of the Event Hubs Standard offering, the isolation model of Event Hubs Premium enables new features like dynamic partition scale-up and yet-to-be-added future capabilities. You also get far more generous quota allocations. Event Hubs Capture is included at no extra cost.

The Premium offering is billed by [Processing Units \(PUs\)](#) which correspond to a share of isolated resources (CPU, Memory, and Storage) in the underlying infrastructure.

In comparison to Dedicated offering, since Event Hubs Premium provides isolation inside a very large multi-tenant environment that can shift resources quickly, it can scale far more elastically and quicker and PUs can be dynamically adjusted. Therefore, Event Hubs Premium will often be a more cost effective option for mid-range (<120MB/sec) throughput requirements, especially with changing loads throughout the day or week, when compared to Event Hubs Dedicated.

## NOTE

Please note that Event Hubs Premium will only support TLS 1.2 or greater .

For the extra robustness gained by availability-zone support, the minimal deployment scale for Event Hubs Dedicated is 8 Capacity Units (CU), but you will have availability zone support in Event Hubs Premium from the first PU in all AZ regions.

You can purchase 1, 2, 4, 8 and 16 Processing Units for each namespace. Since Event Hubs Premium is a capacity-based offering, the achievable throughput is not set by a throttle as it is in Event Hubs Standard, but depends on the work you ask Event Hubs to do, similar to Event Hubs Dedicated. The effective ingest and stream throughput per PU will depend on various factors, including:

- Number of producers and consumers
- Payload size
- Partition count
- Egress request rate
- Usage of Event Hubs Capture, Schema Registry, and other advanced features

Refer the [comparison between Event Hubs SKUs](#) for more details.

#### **NOTE**

All Event Hubs namespaces are enabled for the Apache Kafka RPC protocol by default can be used by your existing Kafka based applications. Having Kafka enabled on your cluster does not affect your non-Kafka use cases; there is no option or need to disable Kafka on a cluster.

## Why Premium?

Premium Event Hubs offers three compelling benefits for customers who require better isolation in a multitenant environment with low latency and high throughput data ingestion needs.

### **Superior performance with the new two-tier storage engine**

Event Hubs premium uses a new two-tier log storage engine that drastically improves the data ingress performance with substantially reduced overall latency and latency jitter without compromising the durability guarantees.

### **Better isolation and predictability**

Event Hubs premium offers an isolated compute and memory capacity to achieve more predictable latency and far reduced *noisy neighbor* impact risk in a multi-tenant deployment.

Event Hubs premium implements a *Cluster in Cluster* model in its multitenant clusters to provide predictability and performance while retaining all the benefits of a managed multitenant PaaS environment.

### **Cost savings and scalability**

As Event Hubs Premium is a multitenant offering, it can dynamically scale more flexibly and very quickly. Capacity is allocated in Processing Units that allocate isolated pods of CPU/Memory inside the cluster. The number of those pods can be scaled up/down per namespace. Therefore, Event Hubs Premium is a low-cost option for messaging scenarios with the overall throughput range that is less than 120 MB/s but higher than what you can achieve with the standard SKU.

## Quotas and limits

The premium tier offers all the features of the standard plan, but with better performance, isolation and more generous quotas. For more quotas and limits, see [Event Hubs quotas and limits](#)

## FAQs

### **What can I achieve with a Processing Unit?**

How much you can ingest and stream with a Processing Unit depends on various factors such as your producers, consumers, the rate at which you're ingesting and processing, and much more. Please refer [Scaling with Event Hubs](#) more details on Processing Units.

### **Can I migrate my Standard namespaces to Premium namespace?**

We don't currently don't support migrating from standard namespaces to premium namespace.

## Next steps

You can start using Event Hubs Premium via [Azure portal](#). Refer [Event Hubs Premium pricing](#) for more details on pricing and [Event Hubs FAQ](#) to find answers to some frequently asked questions about Event Hubs.

# Overview of Event Hubs Dedicated

11/26/2021 • 4 minutes to read • [Edit Online](#)

*Event Hubs clusters* offer single-tenant deployments for customers with the most demanding streaming needs. This single-tenant offering has a guaranteed 99.99% SLA and is available only on our Dedicated pricing tier. An Event Hubs cluster can ingress millions of events per second with guaranteed capacity and subsecond latency. Namespaces and event hubs created within the Dedicated cluster include all features of the premium offering and more, but without any ingress limits. It also includes the popular [Event Hubs Capture](#) feature at no additional cost. This feature allows you to automatically batch and log data streams to Azure Storage or Azure Data Lake.

Clusters are provisioned and billed by **Capacity Units (CUs)**, a pre-allocated amount of CPU and memory resources. You can purchase 1, 2, 4, 8, 12, 16 or 20 CUs for each cluster. How much you can ingest and stream per CU depends on a variety of factors, such as the following ones:

- Number of producers and consumers
- Payload shape
- Egress rate

#### NOTE

All Event Hubs clusters are Kafka-enabled by default and support Kafka endpoints that can be used by your existing Kafka based applications. Having Kafka enabled on your cluster does not affect your non-Kafka use cases; there is no option or need to disable Kafka on a cluster.

## Why Dedicated?

Dedicated Event Hubs offers three compelling benefits for customers who need enterprise-level capacity:

#### **Single-tenancy guarantees capacity for better performance**

A dedicated cluster guarantees capacity at full scale. It can ingest up to gigabytes of streaming data with fully durable storage and subsecond latency to accommodate any burst in traffic.

#### **Inclusive and exclusive access to features**

The Dedicated offering includes features like Capture at no additional cost and exclusive access to features like Bring Your Own Key (BYOK). The service also manages load balancing, OS updates, security patches, and partitioning. So, you can spend less time on infrastructure maintenance and more time on building client-side features.

## Event Hubs Dedicated quotas and limits

The Event Hubs Dedicated offering is billed at a fixed monthly price, with a minimum of 4 hours of usage. The dedicated tier offers all the features of the premium plan, but with enterprise-scale capacity and limits for customers with demanding workloads.

For more quotas and limits, see [Event Hubs quotas and limits](#)

## How to onboard

The self-serve experience to [create an Event Hubs cluster](#) through the [Azure portal](#) is now in Preview. If you have any questions or need help with onboarding to Event Hubs Dedicated, contact the [Event Hubs team](#).

# FAQs

## What can I achieve with a cluster?

For an Event Hubs cluster, how much you can ingest and stream depends on various factors such as your producers, consumers, the rate at which you're ingesting and processing, and much more.

The following table shows the benchmark results that we achieved during our testing:

Payload Shape	Receivers	Ingress Bandwidth	Ingress Messages	Egress Bandwidth	Egress Messages	Total TUs	TUs per CU
Batches of 100x1KB	2	400 MB/sec	400k messages/sec	800 MB/sec	800k messages/sec	400 TUs	100 TUs
Batches of 10x10KB	2	666 MB/sec	66.6k messages/sec	1.33 GB/sec	133k messages/sec	666 TUs	166 TUs
Batches of 6x32KB	1	1.05 GB/sec	34k messages / sec	1.05 GB/sec	34k messages/sec	1000 TUs	250 TUs

In the testing, the following criteria was used:

- A dedicated-tier Event Hubs cluster with four capacity units (CUs) was used.
- The event hub used for ingestion had 200 partitions.
- The data that was ingested was received by two receiver applications receiving from all partitions.

## Can I scale up/down my cluster?

After creation, clusters are billed for a minimum of 4 hours of usage. In the Preview release of the self-serve experience, you can submit a [support request](#) to the Event Hubs team under **Technical > Quota > Request to Scale Up or Scale Down Dedicated Cluster** to scale your cluster up or down. It may take up to 7 days to complete the request to scale down your cluster.

## How does Geo-DR work with my cluster?

You can geo-pair a namespace under a Dedicated-tier cluster with another namespace under a Dedicated-tier cluster. We don't encourage pairing a dedicated-tier namespace with a namespace in our standard offering because the throughput limit will be incompatible and result in errors.

## Can I migrate my Standard namespaces to belong to a Dedicated-tier cluster?

We don't currently support an automated migration process for migrating your event hubs data from a Standard namespace to a Dedicated one.

## Next steps

Contact your Microsoft sales representative or Microsoft Support to get additional details about Event Hubs Dedicated. You can also create a cluster or learn more about Event Hubs pricing tiers by visiting the following links:

- [Create an Event Hubs cluster through the Azure portal](#)
- [Event Hubs Dedicated pricing](#). You can also contact your Microsoft sales representative or Microsoft Support to get additional details about Event Hubs Dedicated capacity.
- The [Event Hubs FAQ](#) contains pricing information and answers some frequently asked questions about Event Hubs.

# Compare Azure Event Hubs tiers

11/26/2021 • 2 minutes to read • [Edit Online](#)

This article compares tiers of Azure Event Hubs.

## NOTE

This article compares only features and quotas of tiers. For pricing information, see [Azure Event Hubs pricing](#)

## Features

The following table shows the list of features that are available (or not available) in a specific tier of Azure Event Hubs.

FEATURE	BASIC	STANDARD	PREMIUM	DEDICATED
Tenancy	Multi-tenant	Multi-tenant	Multi-tenant with resource isolation	Exclusive single tenant
Private link	N/A	Yes	Yes	Yes
Customer-managed key (Bring your own key)	N/A	N/A	Yes	Yes
Capture	N/A	Priced separately	Included	Included
Dynamic Partition scale out	N/A	N/A	Yes	Yes
Ingress events	Pay per million events	Pay per million events	Included	Included

## NOTE

**Included** in the table means the feature is available and there's no separate charge for using it.

## Quotas

The following table shows limits that may be different for basic, standard, and dedicated tiers. In the table CU is **capacity unit**, PU is **processing unit**, TU is **throughput unit**.

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Maximum size of Event Hubs publication	256 KB	1 MB	1 MB	1 MB

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Number of consumer groups per event hub	1	20	100	1000 No limit per CU
Number of brokered connections per namespace	100	5,000	10000 per processing unit per PU	100,000 per CU
Maximum retention period of event data	1 day	7 days	90 days 1 TB per PU	90 days 10 TB per CU
Maximum TUs or PUs or CUs	40 TUs	40 TUs	16 PUs	20 CUs
Number of partitions per event hub	32	32	100 per event hub, 200 per PU	1024 per event hub 2000 per CU
Number of namespaces per subscription	1000	1000	1000	1000 (50 per CU)
Number of event hubs per namespace	10	10	100 per PU	1000
Capture	N/A	Pay per hour	Included	Included
Size of the schema registry (namespace) in mega bytes	N/A	25	100	1024
Number of schema groups in a schema registry or namespace	N/A	1 - excluding the default group	100 1 MB per schema	1000 1 MB per schema
Number of schema versions across all schema groups	N/A	25	1000	10000
Throughput per unit	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	No limits per PU *	No limits per CU *

\* Depends on various factors such as resource allocation, number of partitions, storage, and so on.

#### NOTE

You can publish events individually or batched. The publication limit (according to SKU) applies regardless of whether it is a single event or a batch. Publishing events larger than the maximum threshold will be rejected.

## Next steps

For pricing information for each tier, see [Azure Event Hubs pricing](#)

# Use Azure Event Hubs from Apache Kafka applications

11/26/2021 • 9 minutes to read • [Edit Online](#)

Event Hubs provides an endpoint compatible with the Apache Kafka® producer and consumer APIs that can be used by most existing Apache Kafka client applications as an alternative to running your own Apache Kafka cluster. Event Hubs supports Apache Kafka's producer and consumer APIs clients at version 1.0 and above.

## What does Event Hubs for Kafka provide?

The Event Hubs for Apache Kafka feature provides a protocol head on top of Azure Event Hubs that is protocol compatible with Apache Kafka clients built for Apache Kafka server versions 1.0 and later and supports for both reading from and writing to Event Hubs, which are equivalent to Apache Kafka topics.

You can often use the Event Hubs Kafka endpoint from your applications without code changes compared to your existing Kafka setup and only modify the configuration: Update the connection string in configurations to point to the Kafka endpoint exposed by your event hub instead of pointing to your Kafka cluster. Then, you can start streaming events from your applications that use the Kafka protocol into Event Hubs.

Conceptually, Kafka and Event Hubs are very similar: they're both partitioned logs built for streaming data, whereby the client controls which part of the retained log it wants to read. The following table maps concepts between Kafka and Event Hubs.

### Kafka and Event Hub conceptual mapping

KAFKA CONCEPT	EVENT HUBS CONCEPT
Cluster	Namespace
Topic	Event Hub
Partition	Partition
Consumer Group	Consumer Group
Offset	Offset

### Key differences between Apache Kafka and Event Hubs

While [Apache Kafka](#) is software you typically need to install and operate, Event Hubs is a fully managed, cloud-native service. There are no servers, disks, or networks to manage and monitor and no brokers to consider or configure, ever. You create a namespace, which is an endpoint with a fully qualified domain name, and then you create Event Hubs (topics) within that namespace.

For more information about Event Hubs and namespaces, see [Event Hubs features](#). As a cloud service, Event Hubs uses a single stable virtual IP address as the endpoint, so clients don't need to know about the brokers or machines within a cluster. Even though Event Hubs implements the same protocol, this difference means that all Kafka traffic for all partitions is predictably routed through this one endpoint rather than requiring firewall access for all brokers of a cluster.

Scale in Event Hubs is controlled by how many [throughput units \(TUs\)](#) or [processing units](#) you purchase. If you enable the [Auto-Inflate](#) feature for a standard tier namespace, Event Hubs automatically scales up TUs when you reach the throughput limit. This feature works also works with the Apache Kafka protocol support. For a premier tier namespace, you can increase the number of processing units assigned to the namespace.

### Is Apache Kafka the right solution for your workload?

Coming from building applications using Apache Kafka, it's also useful to understand that Azure Event Hubs is part of a fleet of services, which also includes [Azure Service Bus](#), and [Azure Event Grid](#).

While some providers of commercial distributions of Apache Kafka might suggest that Apache Kafka is a one-stop-shop for all your messaging platform needs, the reality is that Apache Kafka doesn't implement, for

instance, the [competing-consumer](#) queue pattern, doesn't have support for [publish-subscribe](#) at a level that allows subscribers access to the incoming messages based on server-evaluated rules other than plain offsets, and it has no facilities to track the lifecycle of a job initiated by a message or sidelining faulty messages into a dead-letter queue, all of which are foundational for many enterprise messaging scenarios.

To understand the differences between patterns and which pattern is best covered by which service, see the [Asynchronous messaging options in Azure](#) guidance. As an Apache Kafka user, you may find that communication paths you have so far realized with Kafka, can be realized with far less basic complexity and yet more powerful capabilities using either Event Grid or Service Bus.

If you need specific features of Apache Kafka that aren't available through the Event Hubs for Apache Kafka interface or if your implementation pattern exceeds the [Event Hubs quotas](#), you can also run a [native Apache Kafka cluster in Azure HDInsight](#).

## Security and authentication

Every time you publish or consume events from an Event Hubs for Kafka, your client is trying to access the Event Hubs resources. You want to ensure that the resources are accessed using an authorized entity. When using Apache Kafka protocol with your clients, you can set your configuration for authentication and encryption using the SASL mechanisms. When using Event Hubs for Kafka requires the TLS-encryption (as all data in transit with Event Hubs is TLS encrypted), it can be done specifying the **SASL\_SSL** option in your configuration file.

Azure Event Hubs provides multiple options to authorize access to your secure resources.

- OAuth 2.0
- Shared access signature (SAS)

### OAuth 2.0

Event Hubs integrates with Azure Active Directory (Azure AD), which provides an **OAuth 2.0** compliant centralized authorization server. With Azure AD, you can use Azure role-based access control (Azure RBAC) to grant fine grained permissions to your client identities. You can use this feature with your Kafka clients by specifying **SASL\_SSL** for the protocol and **OAUTHBEARER** for the mechanism. For details about Azure roles and levels for scoping access, see [Authorize access with Azure AD](#).

```
bootstrap.servers=NAMESPACENAME.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=OAUTHBEARER
sasl.jaas.config=org.apache.kafka.common.security.oauthbearer.OAuthBearerLoginModule required;
sasl.login.callback.handler.class=CustomAuthenticateCallbackHandler
```

### Shared Access Signature (SAS)

Event Hubs also provides the **Shared Access Signatures (SAS)** for delegated access to Event Hubs for Kafka resources. Authorizing access using OAuth 2.0 token-based mechanism provides superior security and ease of use over SAS. The built-in roles can also eliminate the need for ACL-based authorization, which has to be maintained and managed by the user. You can use this feature with your Kafka clients by specifying **SASL\_SSL** for the protocol and **PLAIN** for the mechanism.

```
bootstrap.servers=NAMESPACENAME.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXXXX"
```

**NOTE**

When using SAS authentication with Kafka clients, established connections aren't disconnected when the SAS key is regenerated.

**Samples**

For a [tutorial](#) with step-by-step instructions to create an event hub and access it using SAS or OAuth, see [Quickstart: Data streaming with Event Hubs using the Kafka protocol](#).

For more [samples](#) that show how to use OAuth with Event Hubs for Kafka, see [samples on GitHub](#).

## Other Event Hubs features

The Event Hubs for Apache Kafka feature is one of three protocols concurrently available on Azure Event Hubs, complementing HTTP and AMQP. You can write with any of these protocols and read with any another, so that your current Apache Kafka producers can continue publishing via Apache Kafka, but your reader can benefit from the native integration with Event Hubs' AMQP interface, such as Azure Stream Analytics or Azure Functions. Conversely, you can readily integrate Azure Event Hubs into AMQP routing networks as a target endpoint, and yet read data through Apache Kafka integrations.

Additionally, Event Hubs features such as [Capture](#), which enables extremely cost efficient long-term archival via Azure Blob Storage and Azure Data Lake Storage, and [Geo Disaster Recovery](#) also work with the Event Hubs for Kafka feature.

## Apache Kafka feature differences

The goal of Event Hubs for Apache Kafka is to provide access to Azure Event Hub's capabilities to applications that are locked into the Apache Kafka API and would otherwise have to be backed by an Apache Kafka cluster.

As explained [above](#), the Azure Messaging fleet provides rich and robust coverage for a multitude of messaging scenarios, and although the following features aren't currently supported through Event Hubs' support for the Apache Kafka API, we point out where and how the desired capability is available.

**Transactions**

[Azure Service Bus](#) has robust transaction support that allows receiving and settling messages and sessions while sending outbound messages resulting from message processing to multiple target entities under the consistency protection of a transaction. The feature set not only allows for exactly once processing of each message in a sequence, but also avoids the risk of another consumer inadvertently reprocessing the same messages as it would be the case with Apache Kafka. Service Bus is the recommended service for transactional message workloads.

**Compression**

The client-side [compression](#) feature of Apache Kafka compresses a batch of multiple messages into a single message on the producer side and decompresses the batch on the consumer side. The Apache Kafka broker treats the batch as a special message.

This feature is fundamentally at odds with Azure Event Hubs' multi-protocol model, which allows for messages, even those sent in batches, to be individually retrievable from the broker and through any protocol.

The payload of any Event Hub event is a byte stream and the content can be compressed with an algorithm of your choosing. The Apache Avro encoding format supports compression natively.

**Log Compaction**

Apache Kafka log compaction is a feature that allows evicting all but the last record of each key from a partition, which effectively turns an Apache Kafka topic into a key-value store where the last value added overrides the previous one. This feature is presently not implemented by Azure Event Hubs. The key-value store pattern, even with frequent updates, is far better supported by database services like [Azure Cosmos DB](#). For more information, see [Log Projection](#).

**Kafka Streams**

Kafka Streams is a client library for stream analytics that is part of the Apache Kafka open-source project, but is separate from the Apache Kafka event stream broker.

The most common reason Azure Event Hubs customers ask for Kafka Streams support is because they're interested in Confluent's "ksqlDB" product. "ksqlDB" is a proprietary shared source project that is [licensed such](#)

that no vendor "offering software-as-a-service, platform-as-a-service, infrastructure-as-a-service, or other similar online services that compete with Confluent products or services" is permitted to use or offer "ksqlDB" support. Practically, if you use ksqlDB, you must either operate Kafka yourself or you must use Confluent's cloud offerings. The licensing terms might also affect Azure customers who offer services for a purpose excluded by the license.

Standalone and without ksqlDB, Kafka Streams has fewer capabilities than many alternative frameworks and services, most of which have built-in streaming SQL interfaces, and all of which integrate with Azure Event Hubs today:

- [Azure Stream Analytics](#)
- [Azure Synapse Analytics \(via Event Hubs Capture\)](#)
- [Azure Databricks](#)
- [Apache Samza](#)
- [Apache Storm](#)
- [Apache Spark](#)
- [Apache Flink](#)
- [Akka Streams](#)

The listed services and frameworks can generally acquire event streams and reference data directly from a diverse set of sources through adapters. Kafka Streams can only acquire data from Apache Kafka and your analytics projects are therefore locked into Apache Kafka. To use data from other sources, you're required to first import data into Apache Kafka with the Kafka Connect framework.

If you must use the Kafka Streams framework on Azure, [Apache Kafka on HDInsight](#) will provide you with that option. Apache Kafka on HDInsight provides full control over all configuration aspects of Apache Kafka, while being fully integrated with various aspects of the Azure platform, from fault/update domain placement to network isolation to monitoring integration.

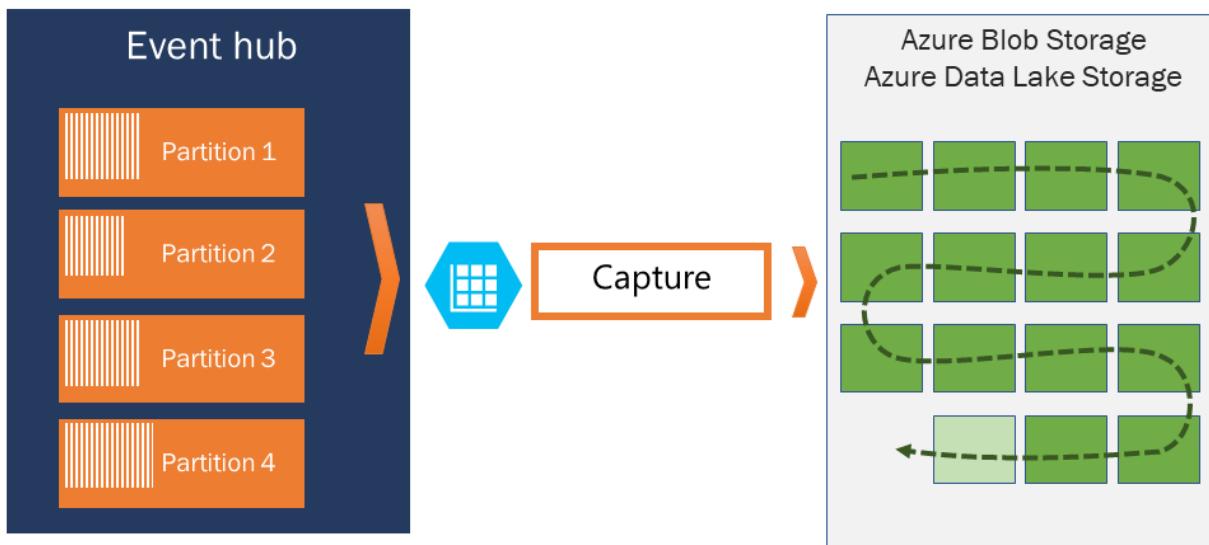
## Next steps

This article provided an introduction to Event Hubs for Kafka. To learn more, see [Apache Kafka developer guide for Azure Event Hubs](#).

# Capture events through Azure Event Hubs in Azure Blob Storage or Azure Data Lake Storage

11/26/2021 • 6 minutes to read • [Edit Online](#)

Azure Event Hubs enables you to automatically capture the streaming data in Event Hubs in an [Azure Blob storage](#) or [Azure Data Lake Storage Gen 1 or Gen 2](#) account of your choice, with the added flexibility of specifying a time or size interval. Setting up Capture is fast, there are no administrative costs to run it, and it scales automatically with Event Hubs [throughput units](#) in the standard tier or [processing units](#) in the premium tier. Event Hubs Capture is the easiest way to load streaming data into Azure, and enables you to focus on data processing rather than on data capture.



## NOTE

Configuring Event Hubs Capture to use Azure Data Lake Storage Gen 2 is same as configuring it to use an Azure Blob Storage. For details, see [Configure Event Hubs Capture](#).

Event Hubs Capture enables you to process real-time and batch-based pipelines on the same stream. This means you can build solutions that grow with your needs over time. Whether you're building batch-based systems today with an eye towards future real-time processing, or you want to add an efficient cold path to an existing real-time solution, Event Hubs Capture makes working with streaming data easier.

## IMPORTANT

The destination storage (Azure Storage or Azure Data Lake Storage) account must be in the same subscription as the event hub.

## How Event Hubs Capture works

Event Hubs is a time-retention durable buffer for telemetry ingress, similar to a distributed log. The key to scaling in Event Hubs is the [partitioned consumer model](#). Each partition is an independent segment of data and is consumed independently. Over time this data ages off, based on the configurable retention period. As a result, a given event hub never gets "too full."

Event Hubs Capture enables you to specify your own Azure Blob storage account and container, or Azure Data Lake Storage account, which are used to store the captured data. These accounts can be in the same region as your event hub or in another region, adding to the flexibility of the Event Hubs Capture feature.

Captured data is written in [Apache Avro](#) format: a compact, fast, binary format that provides rich data structures with inline schema. This format is widely used in the Hadoop ecosystem, Stream Analytics, and Azure Data Factory. More information about working with Avro is available later in this article.

### Capture windowing

Event Hubs Capture enables you to set up a window to control capturing. This window is a minimum size and time configuration with a "first wins policy," meaning that the first trigger encountered causes a capture operation. If you have a fifteen-minute, 100 MB capture window and send 1 MB per second, the size window triggers before the time window. Each partition captures independently and writes a completed block blob at the time of capture, named for the time at which the capture interval was encountered. The storage naming convention is as follows:

```
{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}
```

The date values are padded with zeroes; an example filename might be:

```
https://mystorageaccount.blob.core.windows.net/mycontainer/mynamespace/myeventhub/0/2017/12/08/03/03/17.avro
```

In the event that your Azure storage blob is temporarily unavailable, Event Hubs Capture will retain your data for the data retention period configured on your event hub and back fill the data once your storage account is available again.

### Scaling throughput units or processing units

In the standard tier of Event Hubs, the traffic is controlled by [throughput units](#) and in the premium tier Event Hubs, it's controlled by [processing units](#). Event Hubs Capture copies data directly from the internal Event Hubs storage, bypassing throughput unit or processing unit egress quotas and saving your egress for other processing readers, such as Stream Analytics or Spark.

Once configured, Event Hubs Capture runs automatically when you send your first event, and continues running. To make it easier for your downstream processing to know that the process is working, Event Hubs writes empty files when there is no data. This process provides a predictable cadence and marker that can feed your batch processors.

## Setting up Event Hubs Capture

You can configure Capture at the event hub creation time using the [Azure portal](#), or using Azure Resource Manager templates. For more information, see the following articles:

- [Enable Event Hubs Capture using the Azure portal](#)
- [Create an Event Hubs namespace with an event hub and enable Capture using an Azure Resource Manager template](#)

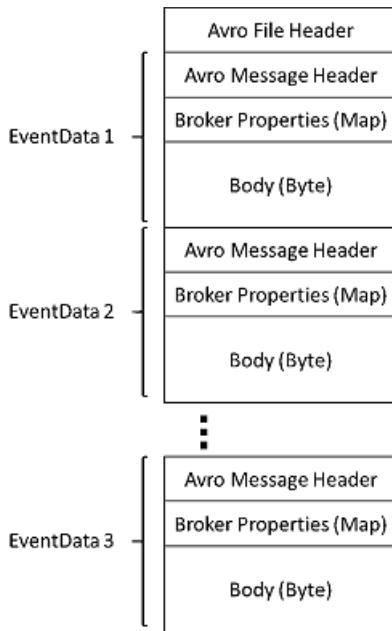
#### NOTE

If you enable the Capture feature for an existing event hub, the feature captures events that arrive at the event hub **after** the feature is turned on. It doesn't capture events that existed in the event hub before the feature was turned on.

## Exploring the captured files and working with Avro

Event Hubs Capture creates files in Avro format, as specified on the configured time window. You can view these files in any tool such as [Azure Storage Explorer](#). You can download the files locally to work on them.

The files produced by Event Hubs Capture have the following Avro schema:



An easy way to explore Avro files is by using the [Avro Tools](#) jar from Apache. You can also use [Apache Drill](#) for a lightweight SQL-driven experience or [Apache Spark](#) to perform complex distributed processing on the ingested data.

### Use Apache Drill

[Apache Drill](#) is an "open-source SQL query engine for Big Data exploration" that can query structured and semi-structured data wherever it is. The engine can run as a standalone node or as a huge cluster for great performance.

A native support to Azure Blob storage is available, which makes it easy to query data in an Avro file, as described in the documentation:

#### [Apache Drill: Azure Blob Storage Plugin](#)

To easily query captured files, you can create and execute a VM with Apache Drill enabled via a container to access Azure Blob storage. See the following sample: [Streaming at Scale with Event Hubs Capture](#).

### Use Apache Spark

[Apache Spark](#) is a "unified analytics engine for large-scale data processing." It supports different languages, including SQL, and can easily access Azure Blob storage. There are a few options to run Apache Spark in Azure, and each provides easy access to Azure Blob storage:

- [HDInsight: Address files in Azure storage](#)
- [Azure Databricks: Azure Blob storage](#)
- [Azure Kubernetes Service](#)

### Use Avro Tools

[Avro Tools](#) are available as a jar package. After you download the jar file, you can see the schema of a specific Avro file by running the following command:

```
java -jar avro-tools-1.9.1.jar getschema <name of capture file>
```

This command returns

```
{
    "type": "record",
    "name": "EventData",
    "namespace": "Microsoft.ServiceBus.Messaging",
    "fields": [
        {"name": "SequenceNumber", "type": "long"},
        {"name": "Offset", "type": "string"},
        {"name": "EnqueuedTimeUtc", "type": "string"},
        {"name": "SystemProperties", "type": {"type": "map", "values": [
            ["long", "double", "string", "bytes"]]}},
        {"name": "Properties", "type": {"type": "map", "values": ["long", "double", "string", "bytes"]}},
        {"name": "Body", "type": ["null", "bytes"]}
    ]
}
```

You can also use Avro Tools to convert the file to JSON format and perform other processing.

To perform more advanced processing, download and install Avro for your choice of platform. At the time of this writing, there are implementations available for C, C++, C#, Java, NodeJS, Perl, PHP, Python, and Ruby.

Apache Avro has complete Getting Started guides for [Java](#) and [Python](#). You can also read the [Getting started with Event Hubs Capture](#) article.

## How Event Hubs Capture is charged

Event Hubs Capture is metered similarly to [throughput units](#) (standard tier) or [processing units](#) (in premium tier): as an hourly charge. The charge is directly proportional to the number of throughput units or processing units purchased for the namespace. As throughput units or processing units are increased and decreased, Event Hubs Capture meters increase and decrease to provide matching performance. The meters occur in tandem. For pricing details, see [Event Hubs pricing](#).

Capture does not consume egress quota as it is billed separately.

## Integration with Event Grid

You can create an Azure Event Grid subscription with an Event Hubs namespace as its source. The following tutorial shows you how to create an Event Grid subscription with an event hub as a source and an Azure Functions app as a sink: [Process and migrate captured Event Hubs data to a Azure Synapse Analytics using Event Grid and Azure Functions](#).

## Next steps

Event Hubs Capture is the easiest way to get data into Azure. Using Azure Data Lake, Azure Data Factory, and Azure HDInsight, you can perform batch processing and other analytics using familiar tools and platforms of your choosing, at any scale you need.

Learn how to enable this feature using the Azure portal and Azure Resource Manager template:

- [Use the Azure portal to enable Event Hubs Capture](#)
- [Use an Azure Resource Manager template to enable Event Hubs Capture](#)

# Azure Schema Registry in Azure Event Hubs

11/26/2021 • 5 minutes to read • [Edit Online](#)

In many event streaming and messaging scenarios, the event or message payload contains structured data. Schema-driven formats such as [Apache Avro](#) are often used to serialize or deserialize such structured data.

An event producer uses a schema to serialize event payload and publish it to an event broker such as Event Hubs. Event consumers read event payload from the broker and de-serialize it using the same schema. So, both producers and consumers can validate the integrity of the data with the same schema.

## What is Azure Schema Registry?

**Azure Schema Registry** is a feature of Event Hubs, which provides a central repository for schemas for event-driven and messaging-centric applications. It provides the flexibility for your producer and consumer applications to **exchange data without having to manage and share the schema**. It also provides a simple governance framework for reusable schemas and defines relationship between schemas through a grouping construct (schema groups).

With schema-driven serialization frameworks like Apache Avro, moving serialization metadata into shared schemas can also help with **reducing the per-message overhead**. That's because each message won't need to have the metadata (type information and field names) as it's the case with tagged formats such as JSON.

Having schemas stored alongside the events and inside the eventing infrastructure ensures that the metadata that's required for serialization or de-serialization is always in reach and schemas can't be misplaced.

### NOTE

The feature isn't available in the **basic tier**.

## Schema Registry information flow

The information flow when you use schema registry is the same for all protocols that you use to publish or consume events from Azure Event Hubs.

The following diagram shows how the information flows when event producers and consumers use Schema Registry with the **Kafka** protocol.

### Producer

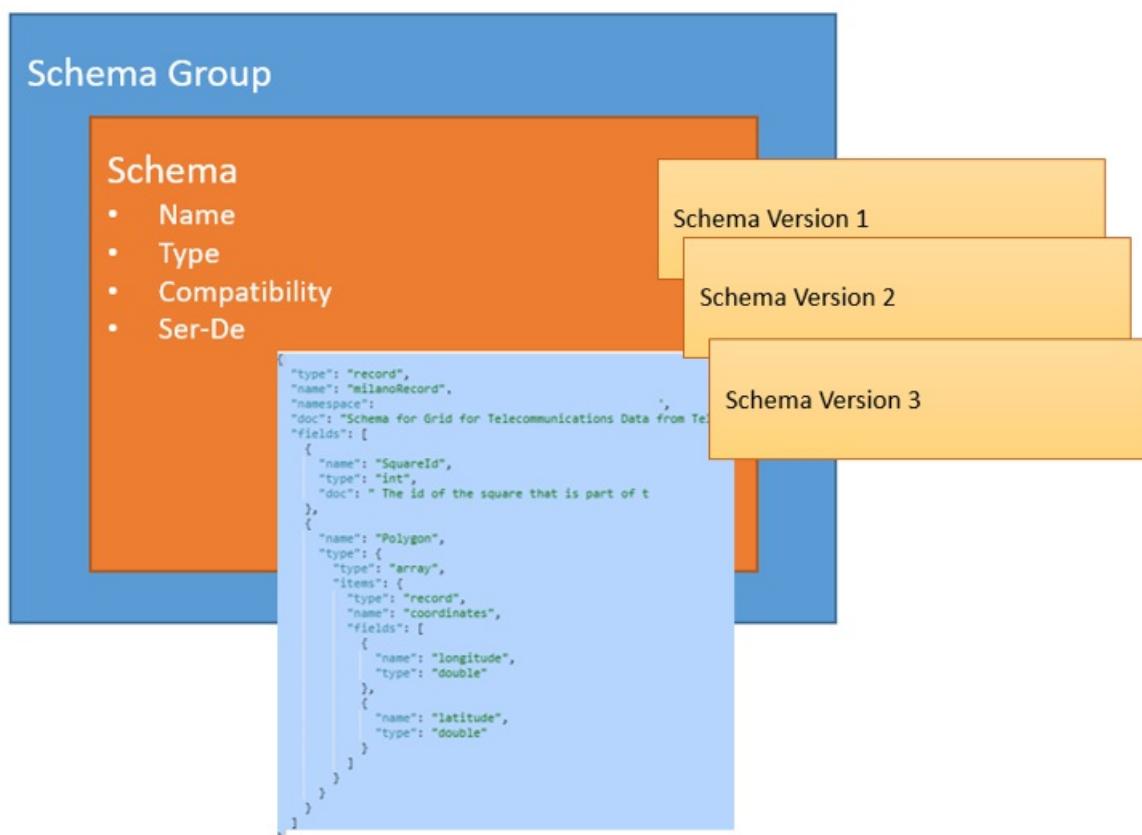
1. Kafka producer application uses `KafkaAvroSerializer` to serialize event data using the specified schema. Producer application provides details of the schema registry endpoint and other optional parameters that are required for schema validation.
2. The serializer looks for the schema in the schema registry to serialize event data. If it finds the schema, then the corresponding schema ID is returned. You can configure the producer application to auto register the schema with the schema registry if it doesn't exist.
3. Then the serializer prepends the schema ID to the serialized data that is published to the Event Hubs.

## Consumer

1. Kafka consumer application uses `KafkaAvroDeserializer` to deserialize data that it receives from the event hub.
2. The deserializer uses the schema ID (prepended by the producer) to retrieve schema from the schema registry.
3. The de-serializer uses the schema to deserialize event data that it receives from the event hub.
4. The schema registry client uses caching to prevent redundant schema registry lookups in the future.

## Schema Registry elements

An Event Hubs namespace now can host schema groups alongside event hubs (or Kafka topics). It hosts a schema registry and can have multiple schema groups. In spite of being hosted in Azure Event Hubs, the schema registry can be used universally with all Azure messaging services and any other message or events broker. Each of these schema groups is a separately securable repository for a set of schemas. Groups can be aligned with a particular application or an organizational unit.



### Schema groups

Schema group is a logical group of similar schemas based on your business criteria. A schema group can hold multiple versions of a schema. The compatibility enforcement setting on a schema group can help ensure that newer schema versions are backwards compatible.

The security boundary imposed by the grouping mechanism helps ensures that trade secrets don't inadvertently leak through metadata in situations where the namespace is shared among multiple partners. It also allows for application owners to manage schemas independent of other applications that share the same namespace.

### Schemas

Schemas define the contract between producers and consumers. A schema defined in an Event Hubs schema registry helps manage the contract outside of event data, thus removing the payload overhead. A schema has a name, type (example: record, array, and so on.), compatibility mode (none, forward, backward, full), and serialization type (only Avro for now). You can create multiple versions of a schema and retrieve and use a

specific version of a schema.

## Schema evolution

Schemas need to evolve with the business requirement of producers and consumers. Azure Schema Registry supports schema evolution by introducing compatibility modes at the schema group level. When you create a schema group, you can specify the compatibility mode of the schemas that you include in that schema group. When you update a schema, the change should comply with the assigned compatibility mode and then only it creates a new version of the schema.

Azure Schema Registry for Event Hubs support following compatibility modes.

### Backward compatibility

Backward compatibility mode allows the consumer code to use a new version of schema but it can process messages with old version of the schema. When you use backward compatibility mode in a schema group, it allows following changes to be made on a schema.

- Delete fields.
- Add optional fields.

### Forward compatibility

Forward compatibility allows the consumer code to use an old version of the schema but it can read messages with the new schema. Forward compatibility mode allows following changes to be made on a schema.

- Add fields
- Delete optional fields

### No compatibility

When the `None` compatibility mode is used, the schema registry doesn't do any compatibility checks when you update schemas.

## Client SDKs

You can use one of the following libraries to include an Avro serializer, which you can use to serialize and deserialize payloads containing Schema Registry schema identifiers and Avro-encoded data.

- .NET - [Microsoft.Azure.Data.SchemaRegistry.ApacheAvro](#)
- Java - [azure-data-schemaregistry-avro](#)
- Python - [azure-schemaregistry-avroserializer](#)
- JavaScript - [@azure/schema-registry-avro](#)
- Apache Kafka - Run Kafka-integrated Apache Avro serializers and deserializers backed by Azure Schema Registry. The Java client's Apache Kafka client serializer for the Azure Schema Registry can be used in any Apache Kafka scenario and with any Apache Kafka® based deployment or cloud service.

## Limits

For limits (for example: number of schema groups in a namespace) of Event Hubs, see [Event Hubs quotas and limits](#)

## Azure role-based access control

When accessing the schema registry programmatically, you need to register an application in Azure Active Directory (Azure AD) and add the security principal of the application to one of the Azure role-based access control (Azure RBAC) roles:

ROLE	DESCRIPTION
Owner	Read, write, and delete Schema Registry groups and schemas.
Contributor	Read, write, and delete Schema Registry groups and schemas.
Schema Registry Reader	Read and list Schema Registry groups and schemas.
Schema Registry Contributor	Read, write, and delete Schema Registry groups and schemas.

For instructions on creating registering an application using the Azure portal, see [Register an app with Azure AD](#). Note down the client ID (application ID), tenant ID, and the secret to use in the code.

## Next steps

- To learn how to create a schema registry using the Azure portal, see [Create an Event Hubs schema registry using the Azure portal](#).
- See the following **Schema Registry Avro client library** samples.
  - [.NET](#)
  - [Java](#)
  - [JavaScript](#)
  - [Python](#)
  - [Kafka Avro Integration for Azure Schema Registry](#)

# Event processor host

11/26/2021 • 13 minutes to read • [Edit Online](#)

## NOTE

This article applies to the old version of Azure Event Hubs SDK. For current version of the SDK, see [Balance partition load across multiple instances of your application](#). To learn how to migrate your code to the newer version of the SDK, see these migration guides.

- [.NET](#)
- [Java](#)
- [Python](#)
- [Java Script](#)

Azure Event Hubs is a powerful telemetry ingestion service that can be used to stream millions of events at low cost. This article describes how to consume ingested events using the *Event Processor Host* (EPH); an intelligent consumer agent that simplifies the management of checkpointing, leasing, and parallel event readers.

The key to scale for Event Hubs is the idea of partitioned consumers. In contrast to the [competing consumers](#) pattern, the partitioned consumer pattern enables high scale by removing the contention bottleneck and facilitating end to end parallelism.

## Home security scenario

As an example scenario, consider a home security company that monitors 100,000 homes. Every minute, it gets data from various sensors such as a motion detector, door/window open sensor, glass break detector, etc., installed in each home. The company provides a web site for residents to monitor the activity of their home in near real time.

Each sensor pushes data to an event hub. The event hub is configured with 16 partitions. On the consuming end, you need a mechanism that can read these events, consolidate them (filter, aggregate, etc.) and dump the aggregate to a storage blob, which is then projected to a user-friendly web page.

## Write the consumer application

When designing the consumer in a distributed environment, the scenario must handle the following requirements:

1. **Scale:** Create multiple consumers, with each consumer taking ownership of reading from a few Event Hubs partitions.
2. **Load balance:** Increase or reduce the consumers dynamically. For example, when a new sensor type (for example, a carbon monoxide detector) is added to each home, the number of events increases. In that case, the operator (a human) increases the number of consumer instances. Then, the pool of consumers can rebalance the number of partitions they own, to share the load with the newly added consumers.
3. **Seamless resume on failures:** If a consumer (**consumer A**) fails (for example, the virtual machine hosting the consumer suddenly crashes), then other consumers must be able to pick up the partitions owned by **consumer A** and continue. Also, the continuation point, called a *checkpoint* or *offset*, should be at the exact point at which **consumer A** failed, or slightly before that.
4. **Consume events:** While the previous three points deal with the management of the consumer, there must be code to consume the events and do something useful with it; for example, aggregate it and upload it to

blob storage.

Instead of building your own solution for this, Event Hubs provides this functionality through the [IEventProcessor](#) interface and the [EventProcessorHost](#) class.

## IEventProcessor interface

First, consuming applications implement the [IEventProcessor](#) interface, which has four methods: [OpenAsync](#), [CloseAsync](#), [ProcessErrorAsync](#), and [ProcessEventsAsync](#). This interface contains the actual code to consume the events that Event Hubs sends. The following code shows a simple implementation:

```
public class SimpleEventProcessor : IEventProcessor
{
    public Task CloseAsync(PartitionContext context, CloseReason reason)
    {
        Console.WriteLine($"Processor Shutting Down. Partition '{context.PartitionId}', Reason: '{reason}' .");
        return Task.CompletedTask;
    }

    public Task OpenAsync(PartitionContext context)
    {
        Console.WriteLine($"SimpleEventProcessor initialized. Partition: '{context.PartitionId}'");
        return Task.CompletedTask;
    }

    public Task ProcessErrorAsync(PartitionContext context, Exception error)
    {
        Console.WriteLine($"Error on Partition: {context.PartitionId}, Error: {error.Message}");
        return Task.CompletedTask;
    }

    public Task ProcessEventsAsync(PartitionContext context, IEnumerable<EventData> messages)
    {
        foreach (var eventData in messages)
        {
            var data = Encoding.UTF8.GetString(eventData.Body.Array, eventData.Body.Offset,
eventData.Body.Count);
            Console.WriteLine($"Message received. Partition: '{context.PartitionId}', Data: '{data}'");
        }
        return context.CheckpointAsync();
    }
}
```

Next, instantiate an [EventProcessorHost](#) instance. Depending on the overload, when creating the [EventProcessorHost](#) instance in the constructor, the following parameters are used:

- **hostName**: the name of each consumer instance. Each instance of [EventProcessorHost](#) must have a unique value for this variable within a consumer group, so don't hard code this value.
- **eventHubPath**: The name of the event hub.
- **consumerGroupName**: Event Hubs uses **\$Default** as the name of the default consumer group, but it is a good practice to create a consumer group for your specific aspect of processing.
- **eventHubConnectionString**: The connection string to the event hub, which can be retrieved from the Azure portal. This connection string should have **Listen** permissions on the event hub.
- **storageConnectionString**: The storage account used for internal resource management.

## IMPORTANT

- Don't enable the soft delete feature on the storage account that's used as a checkpoint store.
- Don't use a hierarchical storage (Azure Data Lake Storage Gen 2) as a checkpoint store.

Finally, consumers register the [EventProcessorHost](#) instance with the Event Hubs service. Registering an event processor class with an instance of [EventProcessorHost](#) starts event processing. Registering instructs the Event Hubs service to expect that the consumer app consumes events from some of its partitions, and to invoke the [IEventProcessor](#) implementation code whenever it pushes events to consume.

## NOTE

The consumerGroupName is case-sensitive. Changes to the consumerGroupName can result in reading all partitions from the start of the stream.

## Example

As an example, imagine that there are 5 virtual machines (VMs) dedicated to consuming events, and a simple console application in each VM, which does the actual consumption work. Each console application then creates one [EventProcessorHost](#) instance and registers it with the Event Hubs service.

In this example scenario, let's say that 16 partitions are allocated to the 5 [EventProcessorHost](#) instances. Some [EventProcessorHost](#) instances might own a few more partitions than others. For each partition that an [EventProcessorHost](#) instance owns, it creates an instance of the [SimpleEventProcessor](#) class. Therefore, there are 16 instances of [SimpleEventProcessor](#) overall, with one assigned to each partition.

The following list summarizes this example:

- 16 Event Hubs partitions.
- 5 VMs, 1 consumer app (for example, Consumer.exe) in each VM.
- 5 EPH instances registered, 1 in each VM by Consumer.exe.
- 16 [SimpleEventProcessor](#) objects created by the 5 EPH instances.
- 1 Consumer.exe app might contain 4 [SimpleEventProcessor](#) objects, since the 1 EPH instance may own 4 partitions.

## Partition ownership tracking

Ownership of a partition to an EPH instance (or a consumer) is tracked through the Azure Storage account that is provided for tracking. You can visualize the tracking as a simple table, as follows. You can see the actual implementation by examining the blobs under the Storage account provided:

CONSUMER GROUP NAME	PARTITION ID	HOST NAME (OWNER)	LEASE (OR OWNERSHIP) ACQUIRED TIME	OFFSET IN PARTITION (CHECKPOINT)
\$Default	0	Consumer_VM3	2018-04-15T01:23:45	156
\$Default	1	Consumer_VM4	2018-04-15T01:22:13	734
\$Default	2	Consumer_VM0	2018-04-15T01:22:56	122

CONSUMER GROUP NAME	PARTITION ID	HOST NAME (OWNER)	LEASE (OR OWNERSHIP) ACQUIRED TIME	OFFSET IN PARTITION (CHECKPOINT)
:				
:				
\$Default	15	Consumer_VM3	2018-04-15T01:22:56	976

Here, each host acquires ownership of a partition for a certain duration (the lease duration). If a host fails (VM shuts down), then the lease expires. Other hosts try to get ownership of the partition, and one of the hosts succeeds. This process resets the lease on the partition with a new owner. This way, only a single reader at a time can read from any given partition within a consumer group.

## Receive messages

Each call to [ProcessEventsAsync](#) delivers a collection of events. It is your responsibility to handle these events. If you want to make sure the processor host processes every message at least once, you need to write your own keep retrying code. But be cautious about poisoned messages.

It is recommended that you do things relatively fast; that is, do as little processing as possible. Instead, use consumer groups. If you need to write to storage and do some routing, it is better to use two consumer groups and have two [IEventProcessor](#) implementations that run separately.

At some point during your processing, you might want to keep track of what you have read and completed.

Keeping track is critical if you must restart reading, so you don't return to the beginning of the stream.

[EventProcessorHost](#) simplifies this tracking by using *checkpoints*. A checkpoint is a location, or offset, for a given partition, within a given consumer group, at which point you are satisfied that you have processed the messages. Marking a checkpoint in [EventProcessorHost](#) is accomplished by calling the [CheckpointAsync](#) method on the [PartitionContext](#) object. This operation is done within the [ProcessEventsAsync](#) method but can also be done in [CloseAsync](#).

## Checkpointing

The [CheckpointAsync](#) method has two overloads: the first, with no parameters, checkpoints to the highest event offset within the collection returned by [ProcessEventsAsync](#). This offset is a "high water" mark; it assumes you have processed all recent events when you call it. If you use this method in this way, be aware that you are expected to call it after your other event processing code has returned. The second overload lets you specify an [EventData](#) instance to checkpoint. This method enables you to use a different type of watermark to checkpoint. With this watermark, you can implement a "low water" mark: the lowest sequenced event you are certain has been processed. This overload is provided to enable flexibility in offset management.

When the checkpoint is performed, a JSON file with partition-specific information (specifically, the offset), is written to the storage account supplied in the constructor to [EventProcessorHost](#). This file is continually updated. It is critical to consider checkpointing in context - it would be unwise to checkpoint every message. The storage account used for checkpointing probably would not handle this load, but more importantly checkpointing every single event is indicative of a queued messaging pattern for which a Service Bus queue might be a better option than an event hub. The idea behind Event Hubs is that you get "at least once" delivery at great scale. By making your downstream systems idempotent, it is easy to recover from failures or restarts that result in the same events being received multiple times.

## Thread safety and processor instances

By default, [EventProcessorHost](#) is thread safe and behaves in a synchronous manner with respect to the instance of [IEventProcessor](#). When events arrive for a partition, [ProcessEventsAsync](#) is called on the [IEventProcessor](#) instance for that partition and will block further calls to [ProcessEventsAsync](#) for the partition. Subsequent messages and calls to [ProcessEventsAsync](#) queue up behind the scenes as the message pump continues to run in the background on other threads. This thread safety removes the need for thread-safe collections and dramatically increases performance.

## Shut down gracefully

Finally, [EventProcessorHost.UnregisterEventProcessorAsync](#) enables a clean shutdown of all partition readers and should always be called when shutting down an instance of [EventProcessorHost](#). Failure to do so can cause delays when starting other instances of [EventProcessorHost](#) due to lease expiration and Epoch conflicts. Epoch management is covered in detail in the [Epoch](#) section of the article.

## Lease management

Registering an event processor class with an instance of [EventProcessorHost](#) starts event processing. The host instance obtains leases on some partitions of the Event Hub, possibly grabbing some from other host instances, in a way that converges on an even distribution of partitions across all host instances. For each leased partition, the host instance creates an instance of the provided event processor class, then receives events from that partition, and passes them to the event processor instance. As more instances get added and more leases are grabbed, [EventProcessorHost](#) eventually balances the load among all consumers.

As explained previously, the tracking table greatly simplifies the autoscale nature of [EventProcessorHost.UnregisterEventProcessorAsync](#). As an instance of [EventProcessorHost](#) starts, it acquires as many leases as possible, and begins reading events. As the leases near expiration, [EventProcessorHost](#) attempts to renew them by placing a reservation. If the lease is available for renewal, the processor continues reading, but if it is not, the reader is closed and [CloseAsync](#) is called. [CloseAsync](#) is a good time to perform any final cleanup for that partition.

[EventProcessorHost](#) includes a [PartitionManagerOptions](#) property. This property enables control over lease management. Set these options before registering your [IEventProcessor](#) implementation.

## Control Event Processor Host options

Additionally, one overload of [RegisterEventProcessorAsync](#) takes an [EventProcessorOptions](#) object as a parameter. Use this parameter to control the behavior of [EventProcessorHost.UnregisterEventProcessorAsync](#) itself. [EventProcessorOptions](#) defines four properties and one event:

- **MaxBatchSize:** The maximum size of the collection you want to receive in an invocation of [ProcessEventsAsync](#). This size is not the minimum, only the maximum size. If there are fewer messages to be received, [ProcessEventsAsync](#) executes with as many as were available.
- **PrefetchCount:** A value used by the underlying AMQP channel to determine the upper limit of how many messages the client should receive. This value should be greater than or equal to [MaxBatchSize](#).
- **InvokeProcessorAfterReceiveTimeout:** If this parameter is `true`, [ProcessEventsAsync](#) is called when the underlying call to receive events on a partition times out. This method is useful for taking time-based actions during periods of inactivity on the partition.
- **InitialOffsetProvider:** Enables a function pointer or lambda expression to be set, which is called to provide the initial offset when a reader begins reading a partition. Without specifying this offset, the reader starts at the oldest event, unless a JSON file with an offset has already been saved in the storage account supplied to the [EventProcessorHost](#) constructor. This method is useful when you want to change the behavior of the reader startup. When this method is invoked, the object parameter contains the partition ID for which the reader is being started.

- [ExceptionReceivedEventArgs](#): Enables you to receive notification of any underlying exceptions that occur in [EventProcessorHost](#). If things are not working as you expect, this event is a good place to start looking.

## Epoch

Here is how the receive epoch works:

### With Epoch

Epoch is a unique identifier (epoch value) that the service uses, to enforce partition/lease ownership. You create an Epoch-based receiver using the [CreateEpochReceiver](#) method. This method creates an Epoch-based receiver. The receiver is created for a specific event hub partition from the specified consumer group.

The epoch feature provides users the ability to ensure that there is only one receiver on a consumer group at any point in time, with the following rules:

- If there is no existing receiver on a consumer group, the user can create a receiver with any epoch value.
- If there is a receiver with an epoch value  $e_1$  and a new receiver is created with an epoch value  $e_2$  where  $e_1 \leq e_2$ , the receiver with  $e_1$  will be disconnected automatically, receiver with  $e_2$  is created successfully.
- If there is a receiver with an epoch value  $e_1$  and a new receiver is created with an epoch value  $e_2$  where  $e_1 > e_2$ , then creation of  $e_2$  will fail with the error: A receiver with epoch  $e_1$  already exists.

### No Epoch

You create a non-Epoch-based receiver using the [CreateReceiver](#) method.

There are some scenarios in stream processing where users would like to create multiple receivers on a single consumer group. To support such scenarios, we do have ability to create a receiver without epoch and in this case we allow upto 5 concurrent receivers on the consumer group.

### Mixed Mode

We don't recommend application usage where you create a receiver with epoch and then switch to no-epoch or vice-versa on the same consumer group. However, when this behavior occurs, the service handles it using the following rules:

- If there is a receiver already created with epoch  $e_1$  and is actively receiving events and a new receiver is created with no epoch, the creation of new receiver will fail. Epoch receivers always take precedence in the system.
- If there was a receiver already created with epoch  $e_1$  and got disconnected, and a new receiver is created with no epoch on a new MessagingFactory, the creation of new receiver will succeed. There is a caveat here that our system will detect the "receiver disconnection" after ~10 minutes.
- If there are one or more receivers created with no epoch, and a new receiver is created with epoch  $e_1$ , all the old receivers get disconnected.

#### NOTE

We recommend using different consumer groups for applications that use epochs and for those that do not use epochs to avoid errors.

## Next steps

Now that you're familiar with the Event Processor Host, see the following articles to learn more about Event Hubs:

- Get started with Event Hubs
  - [.NET Core](#)

- Java
  - Python
  - JavaScript
- Event Hubs programming guide
  - Availability and consistency in Event Hubs
  - Event Hubs FAQ
  - Event Hubs samples on GitHub

# Balance partition load across multiple instances of your application

11/26/2021 • 7 minutes to read • [Edit Online](#)

To scale your event processing application, you can run multiple instances of the application and have it balance the load among themselves. In the older versions, [EventProcessorHost](#) allowed you to balance the load between multiple instances of your program and checkpoint events when receiving. In the newer versions (5.0 onwards), [EventProcessorClient](#) (.NET and Java), or [EventHubConsumerClient](#) (Python and JavaScript) allows you to do the same. The development model is made simpler by using events. You subscribe to the events that you're interested in by registering an event handler. If you're using the old version of the client library, see the following migration guides: [.NET](#), [Java](#), [Python](#), and [JavaScript](#).

This article describes a sample scenario for using multiple instances to read events from an event hub and then give you details about features of event processor client, which allows you to receive events from multiple partitions at once and load balance with other consumers that use the same event hub and consumer group.

## NOTE

The key to scale for Event Hubs is the idea of partitioned consumers. In contrast to the [competing consumers](#) pattern, the partitioned consumer pattern enables high scale by removing the contention bottleneck and facilitating end to end parallelism.

## Example scenario

As an example scenario, consider a home security company that monitors 100,000 homes. Every minute, it gets data from various sensors such as a motion detector, door/window open sensor, glass break detector, and so on, installed in each home. The company provides a web site for residents to monitor the activity of their home in near real time.

Each sensor pushes data to an event hub. The event hub is configured with 16 partitions. On the consuming end, you need a mechanism that can read these events, consolidate them (filter, aggregate, and so on) and dump the aggregate to a storage blob, which is then projected to a user-friendly web page.

## Write the consumer application

When designing the consumer in a distributed environment, the scenario must handle the following requirements:

1. **Scale:** Create multiple consumers, with each consumer taking ownership of reading from a few Event Hubs partitions.
2. **Load balance:** Increase or reduce the consumers dynamically. For example, when a new sensor type (for example, a carbon monoxide detector) is added to each home, the number of events increases. In that case, the operator (a human) increases the number of consumer instances. Then, the pool of consumers can rebalance the number of partitions they own, to share the load with the newly added consumers.
3. **Seamless resume on failures:** If a consumer (**consumer A**) fails (for example, the virtual machine hosting the consumer suddenly crashes), then other consumers can pick up the partitions owned by **consumer A** and continue. Also, the continuation point, called a *checkpoint* or *offset*, should be at the exact point at which **consumer A** failed, or slightly before that.
4. **Consume events:** While the previous three points deal with the management of the consumer, there must

be code to consume the events and do something useful with it. For example, aggregate it and upload it to blob storage.

## Event processor or consumer client

You don't need to build your own solution to meet these requirements. The Azure Event Hubs SDKs provide this functionality. In .NET or Java SDKs, you use an event processor client (`EventProcessorClient`), and in Python and JavaScript SDKs, you use `EventHubConsumerClient`. In the old version of SDK, it was the event processor host (`EventProcessorHost`) that supported these features.

For the majority of production scenarios, we recommend that you use the event processor client for reading and processing events. The processor client is intended to provide a robust experience for processing events across all partitions of an event hub in a performant and fault tolerant manner while providing a means to checkpoint its progress. Event processor clients can work cooperatively within the context of a consumer group for a given event hub. Clients will automatically manage distribution and balancing of work as instances become available or unavailable for the group.

## Partition ownership tracking

An event processor instance typically owns and processes events from one or more partitions. Ownership of partitions is evenly distributed among all the active event processor instances associated with an event hub and consumer group combination.

Each event processor is given a unique identifier and claims ownership of partitions by adding or updating an entry in a checkpoint store. All event processor instances communicate with this store periodically to update its own processing state as well as to learn about other active instances. This data is then used to balance the load among the active processors. New instances can join the processing pool to scale up. When instances go down, either because of failures or to scale down, partition ownership is gracefully transferred to other active processors.

Partition ownership records in the checkpoint store keep track of Event Hubs namespace, event hub name, consumer group, event processor identifier (also known as owner), partition ID, and the last modified time.

EVENT HUBS NAMESPACE	EVENT HUB NAME	CONSUMER GROUP	OWNER	PARTITION ID	LAST MODIFIED TIME
mynamespace.servicbus.windows.net	myeventhub	myconsumergroup	3be3f9d3-9d9e-4c50-9491-85ece8334ff6	0	2020-01-15T01:22:15
mynamespace.servicbus.windows.net	myeventhub	myconsumergroup	f5cc5176-ce96-4bb4-bbaa-a0e3a9054ecf	1	2020-01-15T01:22:17
mynamespace.servicbus.windows.net	myeventhub	myconsumergroup	72b980e9-2efc-4ca7-ab1b-ffd7bece8472	2	2020-01-15T01:22:10
		:			
		:			
mynamespace.servicbus.windows.net	myeventhub	myconsumergroup	844bd8fb-1f3a-4580-984d-6324f9e208af	15	2020-01-15T01:22:00

Each event processor instance acquires ownership of a partition and starts processing the partition from last known [checkpoint](# Checkpointing). If a processor fails (VM shuts down), then other instances detect it by looking at the last modified time. Other instances try to get ownership of the partitions previously owned by the inactive instance, and the checkpoint store guarantees that only one of the instances succeeds in claiming ownership of a partition. So, at any given point of time, there is at most one processor that receives events from a partition.

## Receive messages

When you create an event processor, you specify the functions that will process events and errors. Each call to the function that processes events delivers a single event from a specific partition. It's your responsibility to handle this event. If you want to make sure the consumer processes every message at least once, you need to write your own code with retry logic. But be cautious about poisoned messages.

We recommend that you do things relatively fast. That is, do as little processing as possible. If you need to write to storage and do some routing, it's better to use two consumer groups and have two event processors.

## Checkpointing

*Checkpointing* is a process by which an event processor marks or commits the position of the last successfully processed event within a partition. Marking a checkpoint is typically done within the function that processes the events and occurs on a per-partition basis within a consumer group.

If an event processor disconnects from a partition, another instance can resume processing the partition at the checkpoint that was previously committed by the last processor of that partition in that consumer group. When the processor connects, it passes the offset to the event hub to specify the location at which to start reading. In this way, you can use checkpointing to both mark events as "complete" by downstream applications and to provide resiliency when an event processor goes down. It's possible to return to older data by specifying a lower offset from this checkpointing process.

When the checkpoint is performed to mark an event as processed, an entry in checkpoint store is added or updated with the event's offset and sequence number. Users should decide the frequency of updating the checkpoint. Updating after each successfully processed event can have performance and cost implications as it triggers a write operation to the underlying checkpoint store. Also, checkpointing every single event is indicative of a queued messaging pattern for which a Service Bus queue might be a better option than an event hub. The idea behind Event Hubs is that you get "at least once" delivery at great scale. By making your downstream systems idempotent, it's easy to recover from failures or restarts that result in the same events being received multiple times.

### NOTE

If you are using Azure Blob Storage as the checkpoint store in an environment that supports a different version of Storage Blob SDK than those typically available on Azure, you'll need to use code to change the Storage service API version to the specific version supported by that environment. For example, if you are running [Event Hubs on an Azure Stack Hub version 2002](#), the highest available version for the Storage service is version 2017-11-09. In this case, you need to use code to target the Storage service API version to 2017-11-09. For an example on how to target a specific Storage API version, see these samples on GitHub:

- [.NET](#)
- [Java](#)
- [JavaScript or TypeScript](#)
- [Python](#)

## Thread safety and processor instances

By default, the function that processes the events is called sequentially for a given partition. Subsequent events and calls to this function from the same partition queue up behind the scenes as the event pump continues to run in the background on other threads. Events from different partitions can be processed concurrently and any shared state that is accessed across partitions have to be synchronized.

## Next steps

See the following quick starts:

- [.NET Core](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)

# Multi-site and multi-region federation

11/26/2021 • 13 minutes to read • [Edit Online](#)

Many sophisticated solutions require the same event streams to be made available for consumption in multiple locations, or they require event streams to be collected in multiple locations and then consolidated into a specific location for consumption. There's also often the need to enrich or reduce event streams or do event format conversions, also for within a single region and solution.

Practically, that means your solution will maintain multiple Event Hubs, often in different regions and Event Hubs namespaces, and then replicate events between them. You might also exchange events with sources and targets like [Azure Service Bus](#), [Azure IoT Hub](#), or [Apache Kafka](#).

Maintaining multiple active Event Hubs in different regions also allows clients to choose and switch between them if their contents are being merged, which makes the overall system more resilient against regional availability issues.

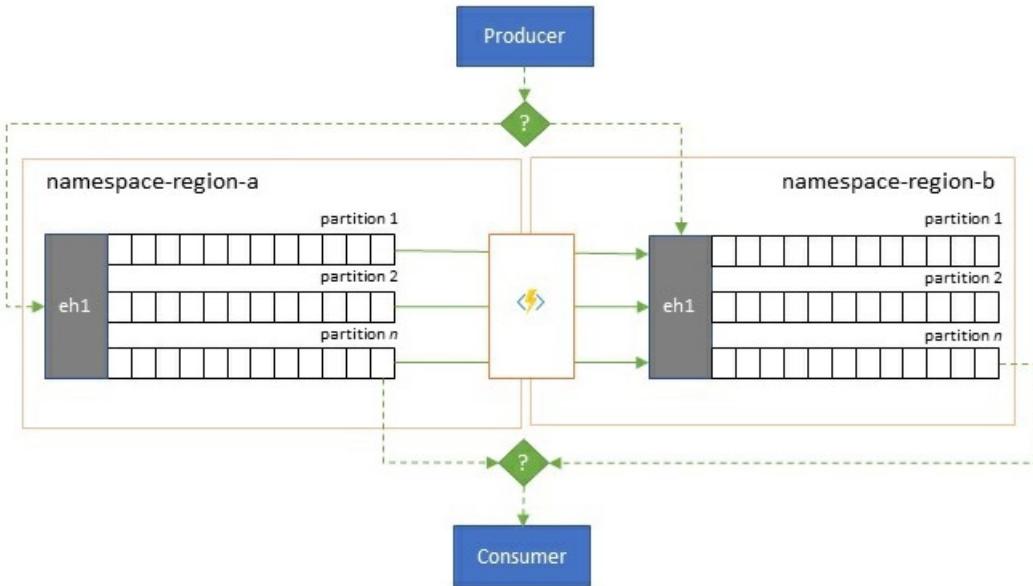
This "Federation" chapter explains federation patterns and how to realize these patterns using the serverless [Azure Stream Analytics](#) or the [Azure Functions](#) runtimes, with the option of having your own transformation or enrichment code right in the event flow path.

## Federation Patterns

There are many potential motivations for why you may want to move events between different Event Hubs or other sources and targets, and we enumerate the most important patterns in this section and also link to more detailed guidance for the respective pattern.

- [Resiliency against regional availability events](#)
- [Latency optimization](#)
- [Validation, reduction, and enrichment](#)
- [Integration with analytics services](#)
- [Consolidation and normalization of event streams](#)
- [Splitting and routing of event streams](#)
- [Log projections](#)

### **Resiliency against regional availability events**



While maximum availability and reliability are the top operational priorities for Event Hubs, there are nevertheless many ways in which a producer or consumer might be prevented from talking to its assigned "primary" Event Hub because of networking or name resolution issues, or where an Event Hub might indeed be temporarily unresponsive or returning errors.

Such conditions aren't "disastrous" such that you'll want to abandon the regional deployment altogether as you might do in a [disaster recovery](#) situation, but the business scenario of some applications might already be impacted by availability events that last not more than a few minutes or even seconds.

There are two foundational patterns to address such scenarios:

- The [replication](#) pattern is about replicating the contents of a primary Event Hub to a secondary Event Hub, whereby the primary Event Hub is generally used by the application for both producing and consuming events and the secondary serves as a fallback option in case the primary Event Hub is becoming unavailable. Since replication is unidirectional, from the primary to the secondary, a switchover of both producers and consumers from an unavailable primary to the secondary will cause the old primary to no longer receive new events and it will therefore be no longer current. Pure replication is therefore only suitable for one-way failover scenarios. Once the failover has been performed, the old primary is abandoned and a new secondary Event Hub needs to be created in a different target region.
- The [merge](#) pattern extends the replication pattern by performing a continuous merge of the contents of two or more Event Hubs. Each event originally produced into one of the Event Hubs included in the scheme is replicated to the other Event Hubs. As events are replicated, they are annotated such that they are subsequently ignored by the replication process of the replication target. The results of using the merge pattern are two or more Event Hubs that will contain the same set of events in an eventually consistent fashion.

In either case, the contents of the Event Hubs will not be identical. Events from any one producer and grouped by the same partition key will appear in the same relative order as originally submitted, but the absolute order of events may differ. This is especially true for scenarios where the partition count of source and target Event Hubs differ, which is desirable for several of the extended patterns described here. A [splitter or router](#) may obtain a slice of a much larger Event Hub with hundreds of partitions and funnel into a smaller Event Hub with just a handful of partitions, more suitable for handling the subset with limited processing resources. Conversely, a [consolidation](#) may funnel data from several smaller Event Hubs into a single, larger Event Hub with more partitions to cope with the consolidated throughput and processing needs.

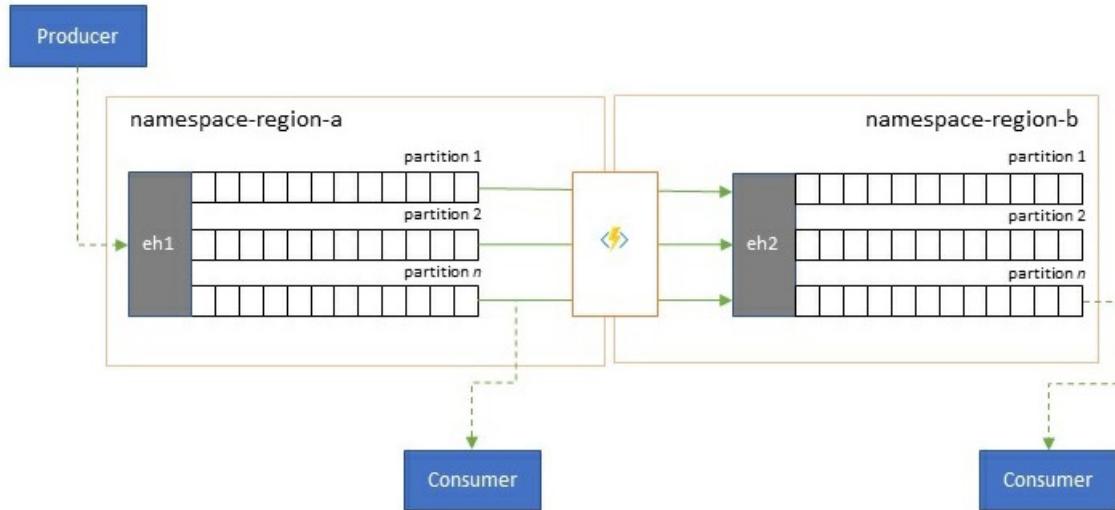
The criterion for keeping events together is the partition key and not the original partition ID. Further

considerations about relative order and how to perform a failover from one Event Hub to the next without relying on the same scope of stream offsets is discussed in [replication](#) pattern description.

Guidance:

- [Replication pattern](#)
- [Merge pattern](#)

## Latency optimization



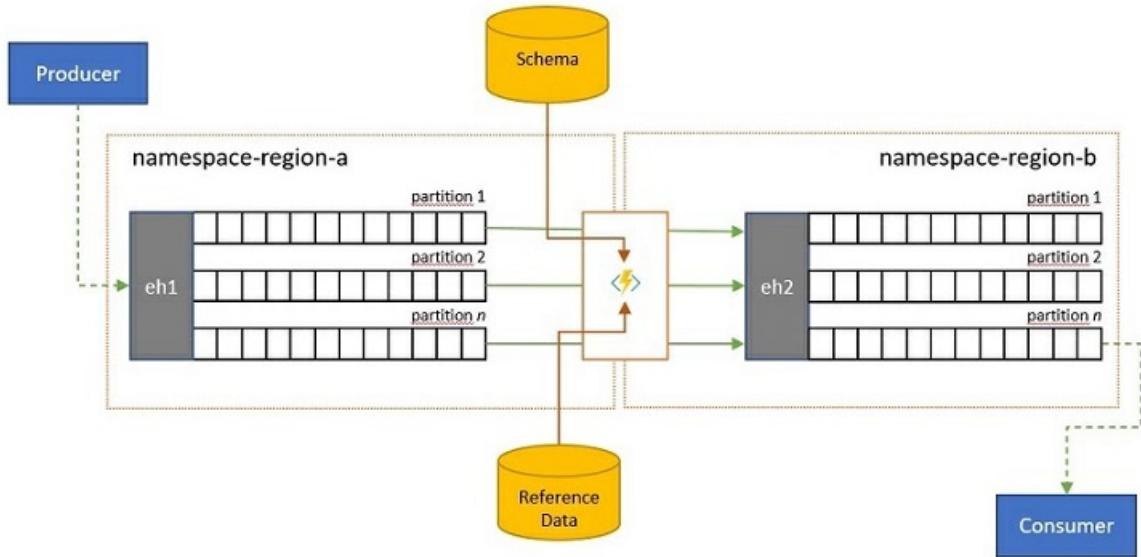
Event streams are written once by producers, but may be read any number of times by event consumers. For scenarios where an event stream in a region is shared by multiple consumers, and needs to be accessed repeatedly during analytics processing residing in a different region, or with throughout demands that would starve out concurrent consumers, it may be beneficial to place a copy of the event stream near the analytics processor to reduce the roundtrip latency.

Good examples for when replication should be preferred over consuming events remotely from across regions are especially those where the regions are extremely far apart, for instance Europe and Australia being nearly antipodes, geographically and network latencies can easily exceed 250 ms for any round trip. You can't accelerate the speed of light, but you can reduce the number of high-latency round trips to interact with data.

Guidance:

- [Replication pattern](#)

## Validation, reduction, and enrichment



Event streams may be submitted into an Event Hub by clients external to your own solution. Such event streams may require for externally submitted events to be checked for compliance with a given schema, and for non-compliant events to be dropped.

In scenarios where clients are extremely bandwidth constrained as it is the case in many "Internet of Things" scenarios with metered bandwidth, or where events are originally sent over non-IP networks with constrained packet sizes, the events may have to be enriched with reference data to add further context for being usable by downstream event processors.

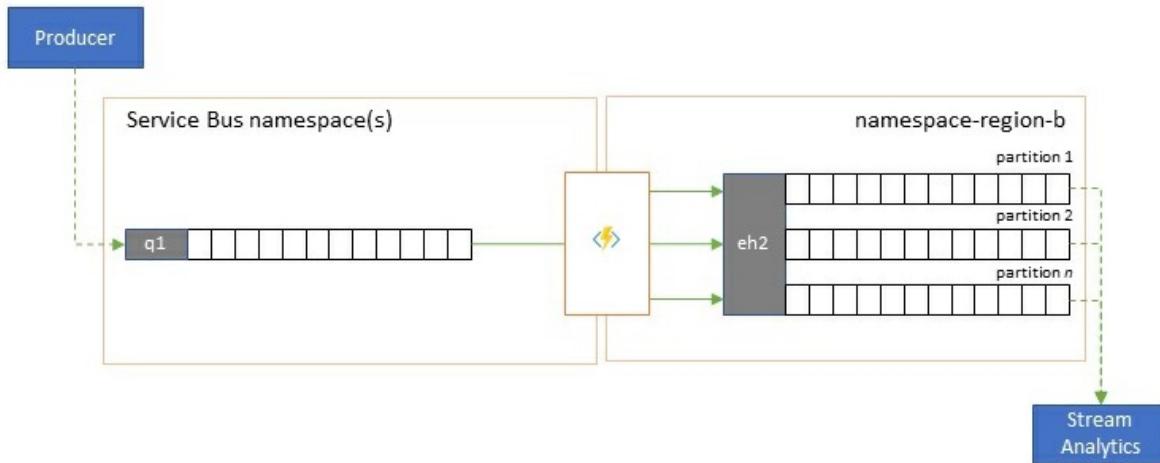
In other cases, especially when streams are being consolidated, the event data may have to be reduced in complexity or sheer size by omitting some detail.

Any of these operations may occur as part of replication, consolidation, or merge flows.

Guidance:

- [Editor pattern](#)

### Integration with analytics services



Several of Azure's cloud-native analytics services like Azure Stream Analytics or Azure Synapse work best with streamed or pre-batched data served up from Azure Event Hubs, and Azure Event Hubs also enables integration with several open-source analytics packages such as Apache Samza, Apache Flink, Apache Spark, and Apache Storm.

If your solution primarily uses Service Bus or Event Grid, you can make these events easily accessible to such

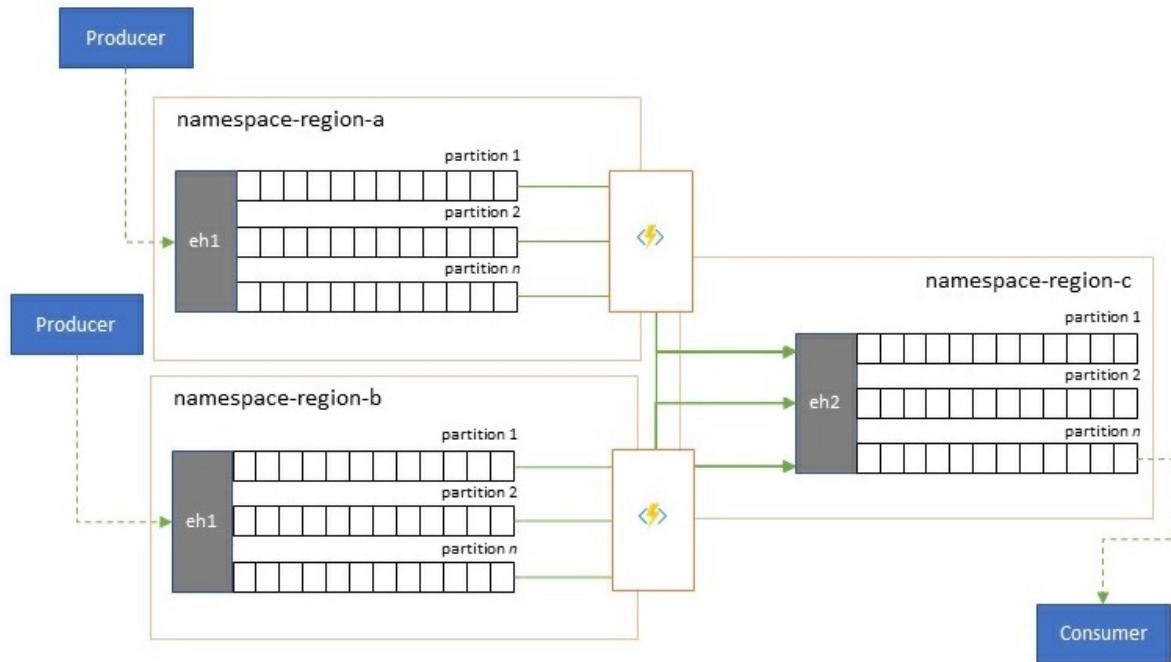
analytics systems and also for archival with Event Hubs Capture if you funnel them into Event Hub. Event Grid can do so natively with its [Event Hub integration](#), for Service Bus you follow the [Service Bus replication guidance](#).

Azure Stream Analytics [integrates with Event Hubs directly](#).

Guidance:

- [Replication pattern](#)

### Consolidation and normalization of event streams



Global solutions are often composed of regional footprints that are largely independent including having their own analytics capabilities, but supra-regional and global analytics perspectives will require an integrated perspective and that's why a central consolidation of the same event streams that are evaluated in the respective regional footprints for the local perspective.

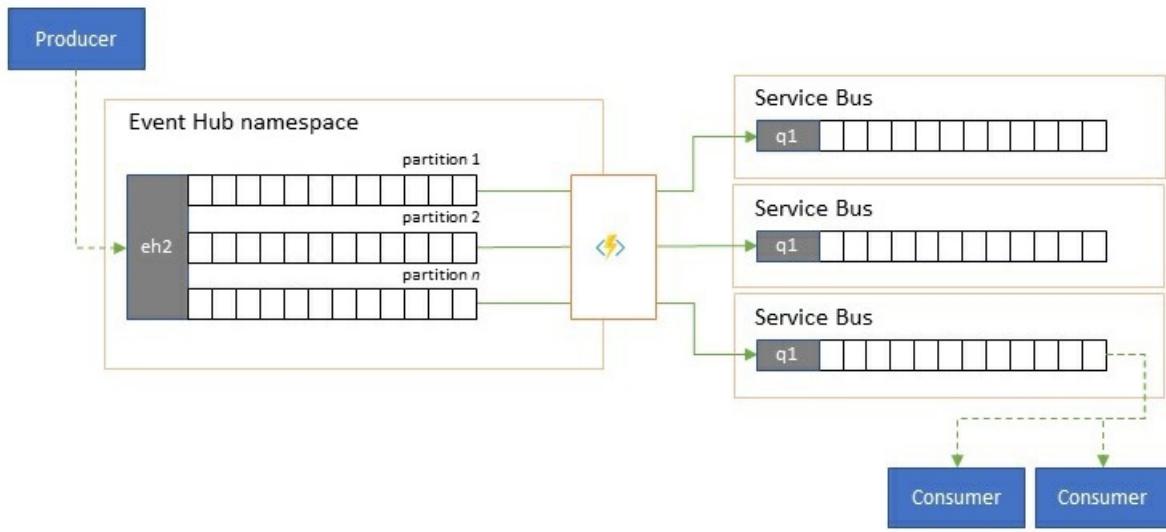
Normalization is a flavor of the consolidation scenario, whereby two or more incoming event streams carry the same kind of events, but with different structures or different encodings, and the events must be transcoded or transformed before they can be consumed.

Normalization may also include cryptographic work such as decrypting end-to-end encrypted payloads and re-encrypting it with different keys and algorithms for the downstream consumer audience.

Guidance:

- [Merge pattern](#)
- [Editor pattern](#)

### Splitting and routing of event streams



Azure Event Hubs is occasionally used in "publish-subscribe" style scenarios where an incoming torrent of ingested events far exceeds the capacity of Azure Service Bus or Azure Event Grid, both of which have native publish-subscribe filtering and distribution capabilities and are preferred for this pattern.

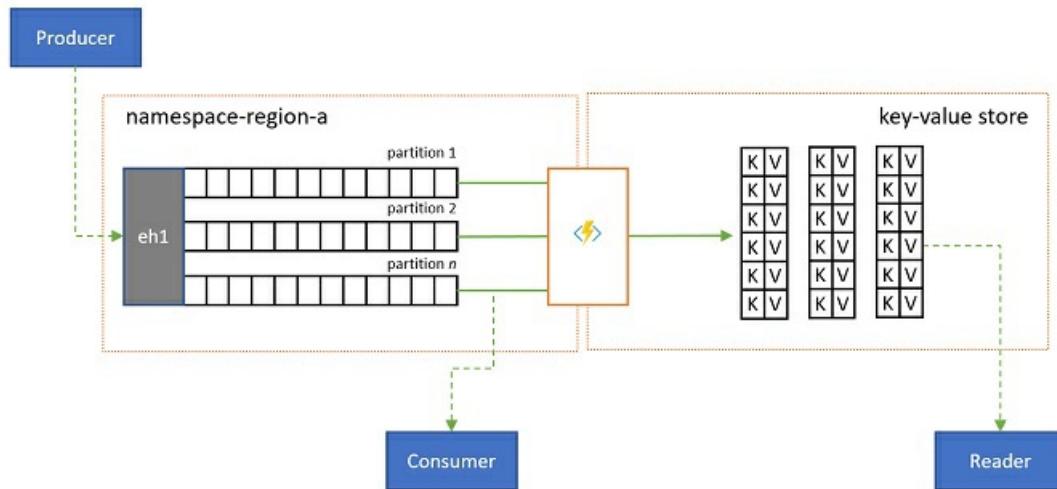
While a true "publish-subscribe" capability leaves it to subscribers to pick the events they want, the splitting pattern has the producer map events to partitions by a predetermined distribution model and designated consumers then exclusively pull from "their" partition. With the Event Hub buffering the overall traffic, the content of a particular partition, representing a fraction of the original throughput volume, may then be replicated into a queue for reliable, transactional, competing consumer consumption.

Many scenarios where Event Hubs is primarily used for moving events within an application within a region have some cases where select events, maybe just from a single partition, also have to be made available elsewhere. This scenario is similar to the splitting scenario, but might use a scalable router that considers all the messages arriving in an Event Hub and cherry-picks just a few for onward routing and might differentiate routing targets by event metadata or content.

Guidance:

- [Routing pattern](#)

### Log projections



In some scenarios, you will want to have access to the latest value sent for any substream of an event, and commonly distinguished by the partition key. In Apache Kafka, this is often achieved by enabling "log

"compaction" on a topic, which discards all but the latest event labeled with any unique key. The log compaction approach has three compounding disadvantages:

- The compaction requires a continuous reorganization of the log, which is an excessively expensive operation for a broker that is optimized for append-only workloads.
- Compaction is destructive and does not allow for a compacted and non-compacted perspective of the same stream.
- A compacted stream still has a sequential access model, meaning that finding the desired value in the log requires reading the entire log in the worst case, which typically leads to optimizations that implement the exact pattern presented here: projecting the log contents into a database or cache.

Ultimately, a compacted log is a key-value store and as such, it is the worst possible implementation option for such a store. It is far more efficient for lookups and for queries to create and use a permanent projection of the log onto a proper key-value store or some other database.

Since events are immutable and the order is always preserved in a log, any projection of a log into a key-value store will always be identical for the same range of events, meaning that a projection you keep updated always provides an authoritative view and there is never any good reason to rebuild it from the log contents once built.

Guidance:

- [Log projection](#)

## Replication application technologies

Implementing the patterns above requires a scalable and reliable execution environment for the replication tasks that you want to configure and run. On Azure, the runtime environments that are best suited for such tasks are stateless tasks are [Azure Stream Analytics](#) for stateful stream replication tasks and [Azure Functions](#) for stateless replication tasks.

### Stateful replication applications in Azure Stream Analytics

For stateful replication applications that need to consider relationships between events, create composite events, enrich events or reduce events, create data aggregations, and transform event payloads, [Azure Stream Analytics](#) is the best implementation option.

In Azure Stream Analytics, you [create jobs](#) that integrate [inputs](#) and [outputs](#) and integrate the data from the inputs through [queries](#) that yield a result that is then made available on the outputs.

Queries are based on the [SQL query language](#) and 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 one or several outputs 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 (for example, Azure Data Lake, Azure Synapse Analytics, etc.) to perform batch analytics or train machine learning models based on very large, indexed pools of historical data.
- Store projections (also called "materialized views") in databases ([SQL Database](#), [Cosmos DB](#) ).

### Stateless replication applications in Azure Functions

For stateless replication tasks where you want to forward events without considering their payloads or

processes them singly without having to consider the relationships of events (except their relative order), you can use Azure Functions, which provides enormous flexibility.

Azure Functions has prebuilt, scalable triggers and output bindings for [Azure Event Hubs](#), [Azure IoT Hub](#), [Azure Service Bus](#), [Azure Event Grid](#), and [Azure Queue Storage](#), as well as custom extensions for [RabbitMQ](#), and [Apache Kafka](#). Most triggers will dynamically adapt to the throughput needs by scaling the number of concurrently executing instances up and down based on documented metrics.

For building log projections, Azure Functions supports output bindings for [Cosmos DB](#) and [Azure Table Storage](#).

Azure Functions can run under a [Azure managed identity](#) and with that, it can hold the configuration values for credentials in tightly access-controlled storage inside of [Azure Key Vault](#).

Azure Functions furthermore allows the replication tasks to directly integrate with Azure virtual networks and [service endpoints](#) for all Azure messaging services, and it is readily integrated with [Azure Monitor](#).

With the Azure Functions consumption plan, the prebuilt triggers can even scale down to zero while no messages are available for replication, which means you incur no costs for keeping the configuration ready to scale back up; the key downside of using the consumption plan is that the latency for replication tasks "waking up" from this state is significantly higher than with the hosting plans where the infrastructure is kept running.

In contrast to all of this, most common replication engines for messaging and eventing, such as Apache Kafka's [MirrorMaker](#) require you to provide a hosting environment and scale the replication engine yourself. That includes configuring and integrating the security and networking features and facilitating the flow of monitoring data, and then you still don't have an opportunity to inject custom replication tasks into the flow.

### Choosing between Azure Functions and Azure Stream Analytics

Azure Stream Analytics (ASA) is the best option whenever you need to process the payload of your events while replicating them. ASA can copy events one by one or it can create aggregates that condense the information of event streams before forwarding it. It can [readily lean on complementing reference data](#) held in Azure Blob Storage or Azure SQL Database without having to import such data into a stream.

With ASA, you can easily create persistent, materialized views of streams in hyper-scale databases. It's a far superior approach to the clunky "log compaction" model of Apache Kafka and the volatile, client-side table projections of Kafka Streams.

ASA can readily process events having payloads encoded in the [CSV, JSON, and Apache Avro formats](#) and you can plug in [custom deserializers](#) for any other format.

For all replication tasks where you want to copy event streams "as-is" and without touching the payloads, or if you need to implement a router, perform cryptographic work, change the encoding of payloads, or if otherwise need full control over the data stream contents, Azure Functions is the best option.

## Next Steps

In this article, we explored a range of federation patterns and explained the role of Azure Functions as the event and messaging replication runtime in Azure.

Next, you might want to read up how to set up a replicator application with Azure Stream Analytics or Azure Functions and then how to replicate event flows between Event Hubs and various other eventing and messaging systems:

- [Event replication task patterns](#)
- [Process data with Azure Stream Analytics](#)
- [Event replicator applications in Azure Functions](#)
- [Replicating events between Event Hubs](#)
- [Replicating events to Azure Service Bus](#)

- Use Apache Kafka MirrorMaker with Event Hubs

# Event replication tasks and applications with Azure Functions

11/26/2021 • 5 minutes to read • [Edit Online](#)

## TIP

For all stateful replication tasks where you need to consider the payloads of your events and transform, aggregate, enrich, or reduce them, use [Azure Stream Analytics](#) instead of Azure Functions.

As explained in the [event replication and cross-region federation](#) article, stateless replication of event streams between pairs of Event Hubs and between Event Hubs and other event stream sources and targets leans on Azure Functions.

[Azure Functions](#) is a scalable and reliable execution environment for configuring and running serverless applications, including event replication and federation tasks.

In this overview, you will learn about Azure Functions' built-in capabilities for such applications, about code blocks that you can adapt and modify for transformation tasks, and about how to configure an Azure Functions application such that it integrates ideally with Event Hubs and other Azure Messaging services. For many details, this article will point to the Azure Functions documentation.

## What is a replication task?

A replication task receives events from a source and forwards them to a target. Most replication tasks will forward events unchanged and at most perform mapping between metadata structures if the source and target protocols differ.

Replication tasks are generally stateless, meaning that they do not share state or other side-effects across sequential or parallel executions of a task. That is also true for batching and chaining, which can both be implemented on top of the existing state of a stream.

This makes replication tasks different from aggregation tasks, which are generally stateful, and are the domain of analytics frameworks and services like [Azure Stream Analytics](#).

## Replication applications and tasks in Azure Functions

In Azure Functions, a replication task is implemented using a [trigger](#) that acquires input messages from a configured source and an [output binding](#) that forwards messages copied from the source to a configured target.

TRIGGER	OUTPUT
<a href="#">Azure Event Hubs trigger</a>	<a href="#">Azure Event Hubs output binding</a>
<a href="#">Azure Service Bus trigger</a>	<a href="#">Azure Service Bus output binding</a>
<a href="#">Azure IoT Hub trigger</a>	<a href="#">Azure IoT Hub output binding</a>
<a href="#">Azure Event Grid trigger</a>	<a href="#">Azure Event Grid output binding</a>
<a href="#">Azure Queue Storage trigger</a>	<a href="#">Azure Queue Storage output binding</a>
<a href="#">Apache Kafka trigger</a>	<a href="#">Apache Kafka output binding</a>
<a href="#">RabbitMQ trigger</a>	<a href="#">RabbitMQ output binding</a>
	<a href="#">Azure Notification Hubs output binding</a>
	<a href="#">Azure SignalR service output binding</a>
	<a href="#">Twilio SendGrid output binding</a>

Replication tasks are deployed into the replication application through the same deployment methods as any other Azure Functions application. You can configure multiple tasks into the same application.

With Azure Functions Premium, multiple replication applications can share the same underlying resource pool, called an App Service Plan. That means you can easily collocate replication tasks written in .NET with replication tasks that are written in Java, for instance. That will matter if you want to take advantage of specific libraries such as Apache Camel that are only available for Java and if those are the best option for a particular integration path, even though you would commonly prefer a different language and runtime for your other replication tasks.

Whenever available, you should prefer the batch-oriented triggers over triggers that deliver individual events or messages and you should always obtain the complete event or message structure rather than rely on Azure Function's [parameter binding expressions](#).

The name of the function should reflect the pair of source and target you are connecting, and you should prefix references to connection strings or other configuration elements in the application configuration files with that name.

#### Data and metadata mapping

Once you've decided on a pair of input trigger and output binding, you will have to perform some mapping between the different event or message types, unless the type of your trigger and the output is the same.

For simple replication tasks that copy messages between Event Hubs and Service Bus, you do not have to write your own code, but can lean on a [utility library that is provided](#) with the replication samples.

#### Retry policy

To avoid data loss during availability event on either side of a replication function, you need to configure the retry policy to be robust. Refer to the [Azure Functions documentation on retries](#) to configure the retry policy.

The policy settings chosen for the example projects in the [sample repository](#) configure an exponential backoff strategy with retry intervals from 5 seconds to 15 minutes with infinite retries to avoid data loss.

For Service Bus, review the "[using retry support on top of trigger resilience](#)" section to understand the interaction of triggers and the maximum delivery count defined for the queue.

#### Setting up a replication application host

A replication application is an execution host for one or more replication tasks.

It's an Azure Functions application that is configured to run either on the consumption plan or (recommended) on an Azure Functions Premium plan. All replication applications must run under a [system-assigned or user-assigned managed identity](#).

The linked Azure Resource Manager (ARM) templates create and configure a replication application with:

- An Azure Storage account for tracking the replication progress and for logs.
- A system-assigned managed identity.
- Azure Monitor and Application Insights integration for monitoring.

Replication applications that must access Event Hubs bound to an Azure virtual network (VNet) must use the Azure Functions Premium plan and be configured to attach to the same VNet, which is also one of the available options.

	DEPLOY	VISUALIZE
Azure Functions Consumption Plan	 Deploy to Azure	 Visualize
Azure Functions Premium Plan	 Deploy to Azure	 Visualize
Azure Functions Premium Plan with VNet	 Deploy to Azure	 Visualize

#### Examples

The [samples repository](#) contains several examples of replication tasks that copy events between Event Hubs and/or between Service Bus entities.

For copying event between Event Hubs, you use an Event Hub Trigger with an Event Hub output binding:

```
[FunctionName("telemetry")]
[ExponentialBackoffRetry(-1, "00:00:05", "00:05:00")]
public static Task Telemetry(
    [EventHubTrigger("telemetry", ConsumerGroup = "$USER_FUNCTIONS_APP_NAME.telemetry", Connection =
    "telemetry-source-connection")] EventData[] input,
    [EventHub("telemetry-copy", Connection = "telemetry-target-connection")] EventHubClient outputClient,
    ILogger log)
{
    return EventHubReplicationTasks.ForwardToEventHub(input, outputClient, log);
}
```

For copying messages between Service Bus entities, you use the Service Bus trigger and output binding:

```
[FunctionName("jobs-transfer")]
[ExponentialBackoffRetry(-1, "00:00:05", "00:05:00")]
public static Task JobsTransfer(
    [ServiceBusTrigger("jobs-transfer", Connection = "jobs-transfer-source-connection")] Message[] input,
    [ServiceBus("jobs", Connection = "jobs-target-connection")] IAsyncCollector<Message> output,
    ILogger log)
{
    return ServiceBusReplicationTasks.ForwardToServiceBus(input, output, log);
}
```

The helper methods can make it easy to replicate between Event Hubs and Service Bus:

SOURCE	TARGET	ENTRY POINT
Event Hub	Event Hub	Azure.Messaging.Replication.EventHubReplicationTasks.ForwardToEventHub
Event Hub	Service Bus	Azure.Messaging.Replication.EventHubReplicationTasks.ForwardToServiceBus
Service Bus	Event Hub	Azure.Messaging.Replication.ServiceBusReplicationTasks.ForwardFromEventHub
Service Bus	Service Bus	Azure.Messaging.Replication.ServiceBusReplicationTasks.ForwardFromServiceBus

## Monitoring

To learn how you can monitor your replication app, please refer to the [monitoring section](#) of the Azure Functions documentation.

A particularly useful visual tool for monitoring replication tasks is the Application Insights [Application Map](#), which is automatically generated from the captured monitoring information and allows exploring the reliability and performance of the replication task source and target transfers.

For immediate diagnostic insights, you can work with the [Live Metrics](#) portal tool, which provides low latency visualization of log details.

## Next steps

- [Azure Functions Deployments](#)
- [Azure Functions Diagnostics](#)
- [Azure Functions Networking Options](#)
- [Azure Application Insights](#)

# Event replication tasks patterns

11/26/2021 • 12 minutes to read • [Edit Online](#)

The [federation overview](#) and the [replicator functions overview](#) explain the rationale for and the basic elements of replication tasks, and it's recommended that you familiarize yourself with them before continuing with this article.

In this article, we detail implementation guidance for several of the patterns highlighted in the overview section.

## Replication

The Replication pattern copies events from one Event Hub to the next, or from an Event Hub to some other destination like a Service Bus queue. The events are forwarded without making any modifications to the event payload.

The implementation of this pattern is covered by the [Event replication between Event Hubs](#) and [Event replication between Event Hubs and Service Bus](#) samples and the [Use Apache Kafka MirrorMaker with Event Hubs](#) tutorial for the specific case of replicating data from an Apache Kafka broker into Event Hubs.

### Streams and order preservation

Replication, either through Azure Functions or Azure Stream Analytics, does not aim to assure the creation of exact 1:1 clones of a source Event Hub into a target Event Hub, but focuses on preserving the relative order of events where the application requires it. The application communicates this by grouping related events with the same partition key and [Event Hubs arranges messages with the same partition key sequentially in the same partition](#).

#### IMPORTANT

The "offset" information is unique for each Event Hub and offsets for the same events will differ across Event Hub instances. To locate a position in a copied event stream, use time-based offsets and refer to the [propagated service-assigned metadata](#).

Time-based offsets start your receiver at a specific point in time:

- `EventPosition.FromStart()` - Read all retained data again.
- `EventPosition.FromEnd()` - Read all new data from the time of connection.
- `EventPosition.FromEnqueuedTime(dateTime)` - All data starting from a given date and time.

In the `EventProcessor`, you set the position through the `InitialOffsetProvider` on the `EventProcessorOptions`. With the other receiver APIs, the position is passed through the constructor.

The pre-built replication function helpers [provided as samples](#) that are used in the Azure Functions based guidance ensure that event streams with the same partition key retrieved from a source partition are submitted into the target Event Hub as a batch in the original stream and with the same partition key.

If the partition count of the source and target Event Hub is identical, all streams in the target will map to the same partitions as they did in the source. If the partition count is different, which matters in some of the further patterns described in the following, the mapping will differ, but streams are always kept together and in order.

The relative order of events belonging to different streams or of independent events without a partition key in a target partition may always differ from the source partition.

### Service-assigned metadata

The service-assigned metadata of an event obtained from the source Event Hub, the original enqueue time, sequence number, and offset, are replaced by new service-assigned values in the target Event Hub, but with the [helper functions](#), replication tasks that are provided in our samples, the original values are preserved in user properties: `repl-enqueue-time` (ISO8601 string), `repl-sequence`, `repl-offset`.

Those properties are of type string and contain the stringified value of the respective original properties. If the event is forwarded multiple times, the service-assigned metadata of the immediate source is appended to the already existing properties, with values separated by semicolons.

## Failover

If you are using replication for disaster recovery purposes, to protect against regional availability events in the Event Hubs service, or against network interruptions, any such failure scenario will require performing a failover from one Event Hub to the next, telling producers and/or consumers to use the secondary endpoint.

For all failover scenarios, it is assumed that the required elements of the namespaces are structurally identical, meaning that Event Hubs and Consumer Groups are identically named and that shared access signature rules and/or role-based access control rules are set up in the same way. You can create (and update) a secondary namespace by following the [guidance for moving namespaces](#) and omitting the cleanup step.

To force producers and consumers to switch, you need to make the information about which namespace to use available for lookup in a location that is easy to reach and update. If producers or consumers encounter frequent or persistent errors, they should consult that location and adjust their configuration. There are numerous ways to share that configuration, but we point out two in the following: DNS and file shares.

### DNS-based failover configuration

One candidate approach is to hold the information in DNS SRV records in a DNS you control and point to the respective Event Hub endpoints.

#### IMPORTANT

Mind that Event Hubs does not allow for its endpoints to be directly aliased with CNAME records, which means you will use DNS as a resilient lookup mechanism for endpoint addresses and not to directly resolve IP address information.

Assume you own the domain `example.com` and, for your application, a zone `test.example.com`. For two alternate Event Hubs, you will now create two further nested zones, and an SRV record in each.

The SRV records are, following common convention, prefixed with `_azure_eventhubs._amqp` and hold two endpoint records: One for AMQP-over-TLS on port 5671 and one for AMQP-over-WebSockets on port 443, both pointing to the Event Hubs endpoint of the namespace corresponding to the zone.

ZONE	SRV RECORD
<code>eh1.test.example.com</code>	<code>_azure_servicebus._amqp.eh1.test.example.com</code> 1 1 5671 eh1-test-example-com.servicebus.windows.net 2 2 443 eh1-test-example-com.servicebus.windows.net
<code>eh2.test.example.com</code>	<code>_azure_servicebus._amqp.eh2.test.example.com</code> 1 1 5671 eh2-test-example-com.servicebus.windows.net 2 2 443 eh2-test-example-com.servicebus.windows.net

In your application's zone, you will then create a CNAME entry that points to the subordinate zone corresponding to your primary Event Hub:

CNAME RECORD	ALIAS
eventhub.test.example.com	eh1.test.example.com

Using a DNS client that allows for querying CNAME and SRV records explicitly (the built-in clients of Java and .NET only allow for simple resolution of names to IP addresses), you can then resolve the desired endpoint. With [DnsClient.NET](#), for instance, the lookup function is:

```
static string GetEventHubName(string aliasName)
{
    const string SrvRecordPrefix = "_azure_eventhub._amqp.";
    LookupClient lookup = new LookupClient();

    return (from CNameRecord alias in (lookup.Query(aliasName, QueryType.CNAME).Answers)
            from SrvRecord srv in lookup.Query(SrvRecordPrefix + alias.CanonicalName, QueryType.SRV).Answers
            where srv.Port == 5671
            select srv.Target).FirstOrDefault()?.Value.TrimEnd('.');
}
```

The function returns the target host name registered for port 5671 of the zone currently aliased with the CNAME as shown above.

Performing a failover requires editing the CNAME record and pointing it to the alternate zone.

The advantage of using DNS, and specifically [Azure DNS](#), is that Azure DNS information is globally replicated and therefore resilient against single-region outages.

This procedure is similar to how the [Event Hubs Geo-DR](#) works, but fully under your own control and also works with active/active scenarios.

#### File-share based failover configuration

The simplest alternative to using DNS for sharing endpoint information is to put the name of the primary endpoint into a plain-text file and serve the file from an infrastructure that is robust against outages and still allows updates.

If you already run a highly available web site infrastructure with global availability and content replication, adding such a file there and republish the file if a switch is needed.

#### Caution

You should only publish the endpoint name in this way, not a full connection string including secrets.

#### Extra considerations for failing over consumers

For Event Hub consumers, further considerations for the failover strategy depend on the needs of the event processor.

If there is a disaster that requires rebuilding a system, including databases, from backup data, and the databases are fed directly or via intermediate processing from the events held in the Event Hub, you will restore the backup and then want to start replaying events into the system from the moment at which the database backup was created and not from the moment the original system was destroyed.

If a failure only affects a slice of a system or indeed only a single Event Hub, which has become unreachable, you will likely want to continue processing events from about the same position where processing was interrupted.

To realize either scenario and using the event processor of your respective Azure SDK, [you will create a new checkpoint store](#) and provide an initial partition position, based on the *timestamp* that you want to resume processing from.

If you still have access to the checkpoint store of the Event Hub you're switching away from, the [propagated metadata](#) discussed above will help you to skip events that were already handled and resume precisely from

where you last left off.

## Merge

The merge pattern has one or more replication tasks pointing to one target, possibly concurrently with regular producers also sending events to the same target.

Variations of these patterns are:

- Two or more replication functions concurrently acquiring events from separate sources and sending them to the same target.
- One more replication function acquiring events from a source while the target is also used directly by producers.
- The prior pattern, but mirrored between two or more Event Hubs, resulting in those Event Hubs containing the same streams, no matter where events are produced.

The first two pattern variations are trivial and do not differ from plain replication tasks.

The last scenario requires excluding already replicated events from being replicated again. The technique is demonstrated and explained in the [EventHubToEventHubMerge](#) sample.

## Editor

The editor pattern builds on the [replication](#) pattern, but messages are modified before they are forwarded.

Examples for such modifications are:

- **Transcoding** - If the event content (also referred to as "body" or "payload") arrives from the source encoded using the *Apache Avro* format or some proprietary serialization format, but the expectation of the system owning the target is for the content to be *JSON* encoded, a transcoding replication task will first deserialize the payload from *Apache Avro* into an in-memory object graph and then serialize that graph into the *JSON* format for the event that is being forwarded. Transcoding also includes **content compression** and decompression tasks.
- **Transformation** - Events that contain structured data may require reshaping of that data for easier consumption by downstream consumers. This may involve work like flattening nested structures, pruning extraneous data elements, or reshaping the payload to exactly fit a given schema.
- **Batching** - Events may be received in batches (multiple events in a single transfer) from a source, but have to be forwarded singly to a target, or vice versa. A task may therefore forward multiple events based on a single input event transfer or aggregate a set of events that are then transferred together.
- **Validation** - Event data from external sources often need to be checked for whether they are in compliance with a set of rules before they may be forwarded. The rules may be expressed using schemas or code. Events that are found not to be in compliance may be dropped, with the issue noted in logs, or may be forwarded to a special target destination to handle them further.
- **Enrichment** - Event data coming from some sources may require enrichment with further context for it to be usable in target systems. This may involve looking up reference data and embedding that data with the event, or adding information about the source that is known to the replication task, but not contained in the events.
- **Filtering** - Some events arriving from a source might have to be withheld from the target based on some rule. A filter tests the event against a rule and drops the event if the event does not match the rule. Filtering out duplicate events by observing certain criteria and dropping subsequent events with the same values is a form of filtering.
- **Cryptography** - A replication task may have to decrypt content arriving from the source and/or encrypt content forwarded onwards to a target, and/or it may have to verify the integrity of content and metadata relative to a signature carried in the event, or attach such a signature.

- **Attestation** - A replication task may attach metadata, potentially protected by a digital signature, to an event that attests that the event has been received through a specific channel or at a specific time.
- **Chaining** - A replication task may apply signatures to streams of events such that the integrity of the stream is protected and missing events can be detected.

The transformation, batching, and enrichment patterns are generally best implemented with [Azure Stream Analytics](#) jobs.

All those patterns can be implemented using Azure Functions, using the [Event Hubs Trigger](#) for acquiring events and the [Event Hub output binding](#) for delivering them.

## Routing

The routing pattern builds on the [replication](#) pattern, but instead of having one source and one target, the replication task has multiple targets, illustrated here in C#:

```
[FunctionName("EH2EH")]
public static async Task Run(
    [EventHubTrigger("source", Connection = "EventHubConnectionAppSetting")] EventData[] events,
    [EventHub("dest1", Connection = "EventHubConnectionAppSetting")] EventHubClient output1,
    [EventHub("dest2", Connection = "EventHubConnectionAppSetting")] EventHubClient output2,
    ILogger log)
{
    foreach (EventData eventData in events)
    {
        // send to output1 and/or output2 based on criteria
        EventHubReplicationTasks.ConditionalForwardToEventHub(input, output1, log, (eventData) => {
            return ( inputEvent.SystemProperties.SequenceNumber%2==0 ) ? inputEvent : null;
        });
        EventHubReplicationTasks.ConditionalForwardToEventHub(input, output2, log, (eventData) => {
            return ( inputEvent.SystemProperties.SequenceNumber%2!=0 ) ? inputEvent : null;
        });
    }
}
```

The routing function will consider the message metadata and/or the message payload and then pick one of the available destinations to send to.

In Azure Stream Analytics, you can achieve the same with defining multiple outputs and then executing a query per output.

```
select * into dest1output from inputSource where Info = 1
select * into dest2output from inputSource where Info = 2
```

## Log projection

The log projection pattern flattens the event stream onto an indexed database, with events becoming records in the database. Typically, events are added to the same collection or table, and the Event Hub partition key becomes part of the primary key looking for making the record unique.

Log projection can produce a time-series historian of your event data or a compacted view, whereby only the latest event is retained for each partition key. The shape of the target database is ultimately up to you and your application's needs. This pattern is also referred to as "event sourcing".

**TIP**

You can easily create log projections into Azure SQL Database and Azure Cosmos DB in Azure Stream Analytics and you should prefer that option.

The following Azure Function projects the contents of an Event Hub compacted into an Azure CosmosDB collection.

```
[FunctionName("Eh1ToCosmosDb1Json")]
[ExponentialBackoffRetry(-1, "00:00:05", "00:05:00")]
public static async Task Eh1ToCosmosDb1Json(
    [EventHubTrigger("eh1", ConsumerGroup = "Eh1ToCosmosDb1", Connection = "Eh1ToCosmosDb1-source-
connection")] EventData[] input,
    [CosmosDB(databaseName: "SampleDb", collectionName: "foo", ConnectionStringSetting =
"CosmosDBConnection")] IAsyncCollector<object> output,
    ILogger log)
{
    foreach (var ev in input)
    {
        if (!string.IsNullOrEmpty(ev.SystemProperties.PartitionKey))
        {
            var record = new
            {
                id = ev.SystemProperties.PartitionKey,
                data = JsonDocument.Parse(ev.Body),
                properties = ev.Properties
            };
            await output.AddAsync(record);
        }
    }
}
```

## Next steps

- [Event replicator applications in Azure Functions](#)
- [Replicating events between Event Hubs](#)
- [Replicating events to Azure Service Bus](#)

# Configured replication tasks - Azure Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

Azure Functions allows for creating configuration-only replication tasks that lean on a pre-built entry point. The [configuration-based replication samples for Azure Functions](#) illustrate how to leverage [pre-built helpers](#) in your own code or avoid handling code altogether and just use configuration.

## Create a replication task

To create such a replication task, first create a new folder underneath the project folder. The name of the new folder is the name of the function, for instance `europeToAsia` or `telemetry`. The name has no direct correlation with the messaging entities being used and serves only for you to identify them. You can create dozens of functions in the same project.

Next, create a `function.json` file in the folder. The file configures the function. Start with the following content:

```
{  
    "configurationSource": "config",  
    "bindings" : [  
        {  
            "direction": "in",  
            "name": "input"  
        },  
        {  
            "direction": "out",  
            "name": "output"  
        }  
    ],  
    "retry": {  
        "strategy": "exponentialBackoff",  
        "maxRetryCount": -1,  
        "minimumInterval": "00:00:05",  
        "maximumInterval": "00:05:00"  
    },  
    "disabled": false,  
    "scriptFile": "../bin/Azure.Messaging.Replication.dll",  
    "entryPoint": "Azure.Messaging.Replication.*"  
}
```

In that file, you need to complete three configuration steps that depend on which entities you want to connect:

1. [Configure the input direction](#)
2. [Configure the output direction](#)
3. [Configure the entry point](#)

### Configure the input direction

#### Event Hub input

If you want to receive events from an Event Hub, add configuration information to the top section within "bindings" that sets

- **type** - the "eventHubTrigger" type.
- **connection** - the name of the app settings value for the Event Hub connection string.
- **eventHubName** - the name of the Event Hub within the namespace identified by the connection string.
- **consumerGroup** - the name of the consumer group. Mind that the name is enclosed in percent signs and therefore also refers to an app settings value.

```
...  
    "bindings" : [  
        {  
            "direction": "in",  
            "type": "eventHubTrigger",  
            "connection": "functionname-source-connection",  
            "eventHubName": "eventHubA",  
            "consumerGroup" : "%functionname-source-consumergroup%",  
            "name": "input"  
        }  
    ...
```

#### Service Bus Queue input

If you want to receive events from a Service Bus queue, add configuration information to the top section within

"bindings" that sets

- **type** - the "serviceBusTrigger" type.
- **connection** - the name of the app settings value for the Service Bus connection string.
- **queueName** - the name of the Service Bus Queue within the namespace identified by the connection string.

```
...
"bindings" : [
{
  "direction": "in",
  "type": "serviceBusTrigger",
  "connection": "functionname-source-connection",
  "queueName": "queue-a",
  "name": "input"
}
...
]
```

#### Service Bus Topic input

If you want to receive events from a Service Bus topic, add configuration information to the top section within "bindings" that sets

- **type** - the "serviceBusTrigger" type.
- **connection** - the name of the app settings value for the Service Bus connection string. This value must be `{FunctionName}-source-connection` if you want to use the provided scripts.
- **topicName** - the name of the Service Bus Topic within the namespace identified by the connection string.
- **subscriptionName** - the name of the Service Bus Subscription on the given topic within the namespace identified by the connection string.

```
...
"bindings" : [
{
  "direction": "in",
  "type": "serviceBusTrigger",
  "connection": "functionname-source-connection",
  "topicName": "topic-x",
  "subscriptionName" : "sub-y",
  "name": "input"
}
...
]
```

#### Configure the output direction

##### Event Hub output

If you want to forward events to an Event Hub, add configuration information to the bottom section within "bindings" that sets

- **type** - the "eventHub" type.
- **connection** - the name of the app settings value for the Event Hub connection string.
- **eventHubName** - the name of the Event Hub within the namespace identified by the connection string.

```
...
"bindings" : [
{
  ...
},
{
  "direction": "out",
  "type": "eventHub",
  "connection": "functionname-target-connection",
  "eventHubName": "eventHubB",
  "name": "output"
}
...
]
```

##### Service Bus Queue output

If you want to forward events to a Service Bus Queue, add configuration information to the bottom section within "bindings" that sets

- **type** - the "serviceBus" type.
- **connection** - the name of the app settings value for the Service Bus connection string.
- **queueName** - the name of the Service Bus queue within the namespace identified by the connection string.

```

...
"bindings" : [
  {
    ...
  },
  {
    "direction": "out",
    "type": "serviceBus",
    "connection": "functionname-target-connection",
    "queueName": "queue-b",
    "name": "output"
  }
...

```

#### Service Bus Topic output

If you want to forward events to a Service Bus Topic, add configuration information to the bottom section within "bindings" that sets

- **type** - the "serviceBus" type.
- **connection** - the name of the app settings value for the Service Bus connection string.
- **topicName** - the name of the Service Bus topic within the namespace identified by the connection string.

```

...
"bindings" : [
  {
    ...
  },
  {
    "direction": "out",
    "type": "serviceBus",
    "connection": "functionname-target-connection",
    "topicName": "topic-b",
    "name": "output"
  }
...

```

#### Configure the entry point

The entry point configuration picks one of the standard replication tasks. If you are modifying the `Azure.Messaging.Replication` project, you can also add tasks and refer to them here. For instance:

```

...
"scriptFile": "../dotnet/bin/Azure.Messaging.Replication.dll",
"entryPoint": "Azure.Messaging.Replication.EventHubReplicationTasks.ForwardToEventHub"
...

```

The following table gives you the correct values for combinations of sources and targets:

SOURCE	TARGET	ENTRY POINT
Event Hub	Event Hub	<code>Azure.Messaging.Replication.EventHubReplicationTasks.ForwardToEventHub</code>
Event Hub	Service Bus	<code>Azure.Messaging.Replication.EventHubReplicationTasks.ForwardToServiceBus</code>
Service Bus	Event Hub	<code>Azure.Messaging.Replication.ServiceBusReplicationTasks.ForwardToEventHub</code>
Service Bus	Service Bus	<code>Azure.Messaging.Replication.ServiceBusReplicationTasks.ForwardToServiceBus</code>

#### Retry policy

Refer to the [Azure Functions documentation on retries](#) to configure the retry policy. The policy settings chosen throughout the projects in this repository configure an exponential backoff strategy with retry intervals from 5 seconds to 5 minutes with infinite retries to avoid data loss.

For Service Bus, review the "[using retry support on top of trigger resilience](#)" section to understand the interaction of triggers and the maximum delivery count defined for the queue.

#### Build, deploy, and configure

While you are focusing on configuration, the tasks still require building a deployable application and to configure the Azure Functions hosts such that it has all the required information to connect to the given endpoints.

This is illustrated, together with reusable scripts, in the [configuration-based replication samples for Azure](#)

Functions.

## Next Steps

- [Replicating event data between Event Hubs](#)
- [Replicating event data to Service Bus](#)
- [Replicating event data from Service Bus](#)

# Azure Event Hubs - Geo-disaster recovery

11/26/2021 • 11 minutes to read • [Edit Online](#)

Resilience against disastrous outages of data processing resources is a requirement for many enterprises and in some cases even required by industry regulations.

Azure Event Hubs already spreads the risk of catastrophic failures of individual machines or even complete racks across clusters that span multiple failure domains within a datacenter and it implements transparent failure detection and failover mechanisms such that the service will continue to operate within the assured service-levels and typically without noticeable interruptions in the event of such failures. If an Event Hubs namespace has been created with the enabled option for [availability zones](#), the outage risk is further spread across three physically separated facilities, and the service has enough capacity reserves to instantly cope with the complete, catastrophic loss of the entire facility.

The all-active Azure Event Hubs cluster model with availability zone support provides resiliency against grave hardware failures and even catastrophic loss of entire datacenter facilities. Still, there might be grave situations with widespread physical destruction that even those measures cannot sufficiently defend against.

The Event Hubs Geo-disaster recovery feature is designed to make it easier to recover from a disaster of this magnitude and abandon a failed Azure region for good and without having to change your application configurations. Abandoning an Azure region will typically involve several services and this feature primarily aims at helping to preserve the integrity of the composite application configuration.

The Geo-Disaster recovery feature ensures that the entire configuration of a namespace (Event Hubs, Consumer Groups and settings) is continuously replicated from a primary namespace to a secondary namespace when paired, and it allows you to initiate a once-only failover move from the primary to the secondary at any time. The failover move will re-point the chosen alias name for the namespace to the secondary namespace and then break the pairing. The failover is nearly instantaneous once initiated.

## IMPORTANT

- The feature enables instantaneous continuity of operations with the same configuration, but **does not replicate the event data**. Unless the disaster caused the loss of all zones, the event data that is preserved in the primary Event Hub after failover will be recoverable and the historic events can be obtained from there once access is restored. For replicating event data and operating corresponding namespaces in active/active configurations to cope with outages and disasters, don't lean on this Geo-disaster recovery feature set, but follow the [replication guidance](#).
- Azure Active Directory (Azure AD) role-based access control (RBAC) assignments to entities in the primary namespace aren't replicated to the secondary namespace. Create role assignments manually in the secondary namespace to secure access to them.

## Outages and disasters

It's important to note the distinction between "outages" and "disasters." An **outage** is the temporary unavailability of Azure Event Hubs, and can affect some components of the service, such as a messaging store, or even the entire datacenter. However, after the problem is fixed, Event Hubs becomes available again. Typically, an outage doesn't cause the loss of messages or other data. An example of such an outage might be a power failure in the datacenter. Some outages are only short connection losses because of transient or network issues.

A **disaster** is defined as the permanent, or longer-term loss of an Event Hubs cluster, Azure region, or datacenter. The region or datacenter may or may not become available again, or may be down for hours or days. Examples of such disasters are fire, flooding, or earthquake. A disaster that becomes permanent might cause the loss of

some messages, events, or other data. However, in most cases there should be no data loss and messages can be recovered once the data center is back up.

The Geo-disaster recovery feature of Azure Event Hubs is a disaster recovery solution. The concepts and workflow described in this article apply to disaster scenarios, and not to transient, or temporary outages. For a detailed discussion of disaster recovery in Microsoft Azure, see [this article](#).

## Basic concepts and terms

The disaster recovery feature implements metadata disaster recovery, and relies on primary and secondary disaster recovery namespaces.

The Geo-disaster recovery feature is available for the [standard, premium, and dedicated SKUs](#) only. You don't need to make any connection string changes, as the connection is made via an alias.

The following terms are used in this article:

- *Alias*: The name for a disaster recovery configuration that you set up. The alias provides a single stable Fully Qualified Domain Name (FQDN) connection string. Applications use this alias connection string to connect to a namespace.
- *Primary/secondary namespace*: The namespaces that correspond to the alias. The primary namespace is "active" and receives messages (can be an existing or new namespace). The secondary namespace is "passive" and doesn't receive messages. The metadata between both is in sync, so both can seamlessly accept messages without any application code or connection string changes. To ensure that only the active namespace receives messages, you must use the alias.
- *Metadata*: Entities such as event hubs and consumer groups; and their properties of the service that are associated with the namespace. Only entities and their settings are replicated automatically. Messages and events aren't replicated.
- *Failover*: The process of activating the secondary namespace.

## Supported namespace pairs

The following combinations of primary and secondary namespaces are supported:

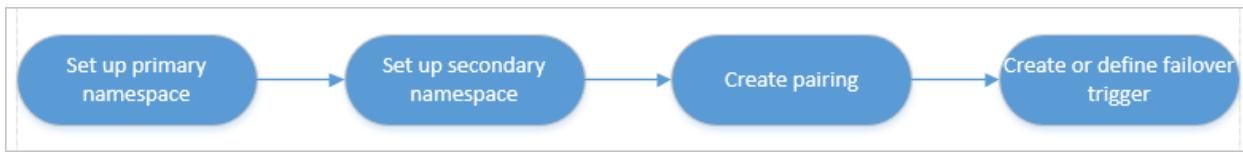
PRIMARY NAMESPACE TIER	ALLOWED SECONDARY NAMESPACE TIER
Standard	Standard, Dedicated
Premium	Premium
Dedicated	Dedicated

### NOTE

You can't pair namespaces that are in the same dedicated cluster. You can pair namespaces that are in separate clusters.

## Setup and failover flow

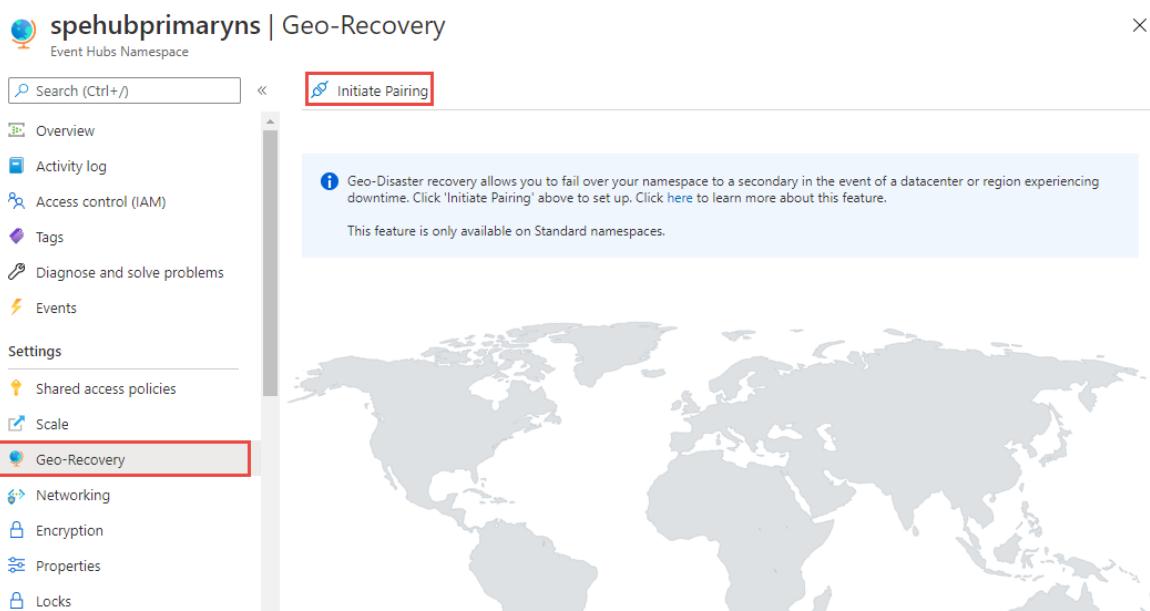
The following section is an overview of the failover process, and explains how to set up the initial failover.



## Setup

You first create or use an existing primary namespace, and a new secondary namespace, then pair the two. This pairing gives you an alias that you can use to connect. Because you use an alias, you don't have to change connection strings. Only new namespaces can be added to your failover pairing.

1. Create the primary namespace.
2. Create the secondary namespace in a different region. This step is optional. You can create the secondary namespace while creating the pairing in the next step.
3. In the Azure portal, navigate to your primary namespace.
4. Select **Geo-recovery** on the left menu, and select **Initiate pairing** on the toolbar.



5. On the **Initiate pairing** page, follow these steps:

- a. Select an existing secondary namespace or create one in a different region. In this example, an existing namespace is selected.
- b. For **Alias**, enter an alias for the geo-dr pairing.
- c. Then, select **Create**.

# Initiate Pairing

Geo-Disaster Recovery

X

## Secondary Namespace

**i** Create a new standard namespace, or specify an existing one, to act as the secondary for Geo-replication

### Subscription

Contoso subscription

### Resource group \*

spehubsecondaryrg

### Location

West US

### Secondary Namespace name

Create new  Use existing

spehubsecn

.servicebus.windows.net

### Alias

### Alias \*

myehubgeodrpair

Create

6. You should see the Geo-DR Alias page. You can also navigate to this page from the primary namespace by selecting **Geo-recovery** on the left menu.

## myehubgeodrpair (spehubprimaryns/myehubgeodrpair)

Event Hubs Geo-DR Alias

Search (Ctrl+J)

Initiate Pairing

Break Pairing

Failover

Delete Alias

Refresh

Overview

Access control (IAM)

Diagnose and solve problems

Settings

Shared access policies

Locks

Automation

Tasks (preview)

Export template

Support + troubleshooting

New support request

Essentials

Alias Name

myehubgeodrpair

Secondary Namespace

spehubgeodrsecn

Pairing Status

Paired



Namespace	Role	Region
spehubprimaryns	Primary	East US
spehubgeodrsecn	Secondary	West US

7. On the Geo-DR Alias page, select **Shared access policies** on the left menu to access the primary connection string for the alias. Use this connection string instead of using the connection string to the primary/secondary namespace directly.
8. On this **Overview** page, you can do the following actions:
  - a. Break the pairing between primary and secondary namespaces. Select **Break pairing** on the toolbar.
  - b. Manually failover to the secondary namespace. Select **Failover** on the toolbar.

**WARNING**

Failing over will activate the secondary namespace and remove the primary namespace from the Geo-Disaster Recovery pairing. Create another namespace to have a new geo-disaster recovery pair.

Finally, you should add some monitoring to detect if a failover is necessary. In most cases, the service is one part of a large ecosystem, thus automatic failovers are rarely possible, as often failovers must be performed in sync with the remaining subsystem or infrastructure.

### Example

In one example of this scenario, consider a Point of Sale (POS) solution that emits either messages or events. Event Hubs passes those events to some mapping or reformatting solution, which then forwards mapped data to another system for further processing. At that point, all of these systems might be hosted in the same Azure region. The decision of when and what part to fail over depends on the flow of data in your infrastructure.

You can automate failover either with monitoring systems, or with custom-built monitoring solutions. However, such automation takes extra planning and work, which is out of the scope of this article.

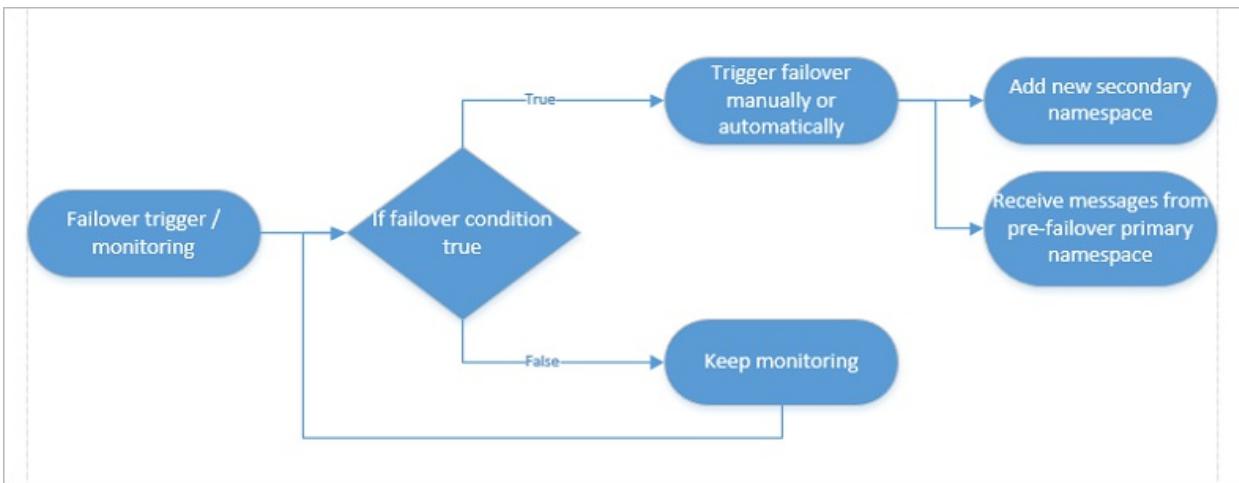
### Failover flow

If you initiate the failover, two steps are required:

1. If another outage occurs, you want to be able to fail over again. Therefore, set up another passive namespace and update the pairing.
2. Pull messages from the former primary namespace once it's available again. After that, use that namespace for regular messaging outside of your geo-recovery setup, or delete the old primary namespace.

**NOTE**

Only fail forward semantics are supported. In this scenario, you fail over and then re-pair with a new namespace. Failing back is not supported; for example, in a SQL cluster.



## Management

If you made a mistake; for example, you paired the wrong regions during the initial setup, you can break the pairing of the two namespaces at any time. If you want to use the paired namespaces as regular namespaces, delete the alias.

## Considerations

Note the following considerations to keep in mind:

1. By design, Event Hubs geo-disaster recovery does not replicate data, and therefore you cannot reuse the old offset value of your primary event hub on your secondary event hub. We recommend restarting your event receiver with one of the following methods:
  - `EventPosition.FromStart()` - If you wish read all data on your secondary event hub.
  - `EventPosition.FromEnd()` - If you wish to read all new data from the time of connection to your secondary event hub.
  - `EventPosition.FromEnqueuedTime(dateTime)` - If you wish to read all data received in your secondary event hub starting from a given date and time.
2. In your failover planning, you should also consider the time factor. For example, if you lose connectivity for longer than 15 to 20 minutes, you might decide to initiate the failover.
3. The fact that no data is replicated means that current active sessions aren't replicated. Additionally, duplicate detection and scheduled messages may not work. New sessions, scheduled messages, and new duplicates will work.
4. Failing over a complex distributed infrastructure should be [rehearsed](#) at least once.
5. Synchronizing entities can take some time, approximately 50-100 entities per minute.

## Availability Zones

Event Hubs supports [Availability Zones](#), providing fault-isolated locations within an Azure region. The Availability Zones support is only available in [Azure regions with availability zones](#). Both metadata and data (events) are replicated across data centers in the availability zone.

When creating a namespace, you see the following highlighted message when you select a region that has availability zones.

## Create Namespace

Event Hubs

Basics Tags Review + create

### PROJECT DETAILS

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

MDP\_492270

Resource group \*

Create new

### INSTANCE DETAILS

Enter required settings for this namespace, including a price tier and configuring the number of units (capacity).

Namespace name \*

.servicebus.windows.net

Location \*

East US



i The region selected supports Availability zones. Your namespace will have Availability Zones enabled. [Learn more](#).

Pricing tier (View full pricing details) \*

Basic (1 Consumer group, 100 Brokered connections)

Throughput Units \*

1

[Review + create](#)

< Previous

Next: Tags >

## Private endpoints

This section provides more considerations when using Geo-disaster recovery with namespaces that use private endpoints. To learn about using private endpoints with Event Hubs in general, see [Configure private endpoints](#).

### New pairings

If you try to create a pairing between a primary namespace with a private endpoint and a secondary namespace without a private endpoint, the pairing will fail. The pairing will succeed only if both primary and secondary namespaces have private endpoints. We recommend that you use same configurations on the primary and secondary namespaces and on virtual networks in which private endpoints are created.

#### NOTE

When you try to pair the primary namespace with private endpoint and a secondary namespace, the validation process only checks whether a private endpoint exists on the secondary namespace. It doesn't check whether the endpoint works or will work after failover. It's your responsibility to ensure that the secondary namespace with private endpoint will work as expected after failover.

To test that the private endpoint configurations are same on primary and secondary namespaces, send a read request (for example: [Get Event Hub](#)) to the secondary namespace from outside the virtual network, and verify that you receive an error message from the service.

### Existing pairings

If pairing between primary and secondary namespace already exists, private endpoint creation on the primary

namespace will fail. To resolve, create a private endpoint on the secondary namespace first and then create one for the primary namespace.

#### NOTE

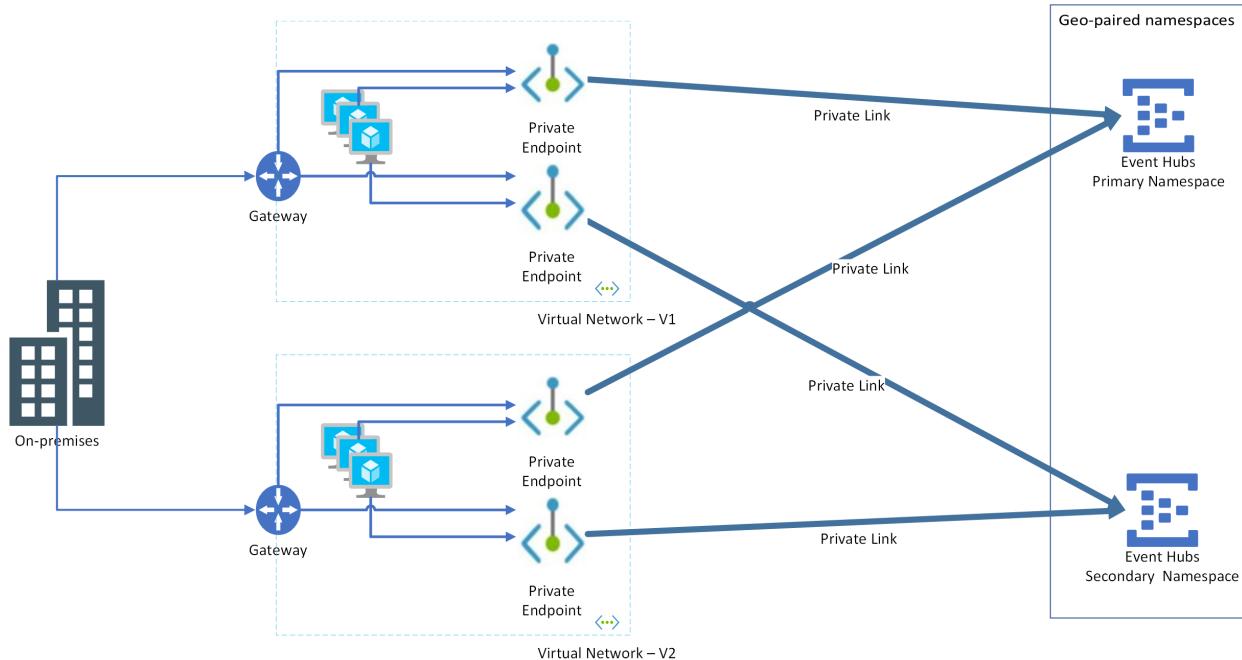
While we allow read-only access to the secondary namespace, updates to the private endpoint configurations are permitted.

### Recommended configuration

When creating a disaster recovery configuration for your application and Event Hubs namespaces, you must create private endpoints for both primary and secondary Event Hubs namespaces against virtual networks hosting both primary and secondary instances of your application.

Let's say you have two virtual networks: VNET-1, VNET-2 and these primary and secondary namespaces: EventHubs-Namespace1-Primary, EventHubs-Namespace2-Secondary. You need to do the following steps:

- On EventHubs-Namespace1-Primary, create two private endpoints that use subnets from VNET-1 and VNET-2
- On EventHubs-Namespace2-Secondary, create two private endpoints that use the same subnets from VNET-1 and VNET-2



Advantage of this approach is that failover can happen at the application layer independent of Event Hubs namespace. Consider the following scenarios:

**Application-only failover:** Here, the application won't exist in VNET-1 but will move to VNET-2. As both private endpoints are configured on both VNET-1 and VNET-2 for both primary and secondary namespaces, the application will just work.

**Event Hubs namespace-only failover:** Here again, since both private endpoints are configured on both virtual networks for both primary and secondary namespaces, the application will just work.

#### NOTE

For guidance on geo-disaster recovery of a virtual network, see [Virtual Network - Business Continuity](#).

## Role-based access control

Azure Active Directory (Azure AD) role-based access control (RBAC) assignments to entities in the primary namespace aren't replicated to the secondary namespace. Create role assignments manually in the secondary namespace to secure access to them.

## Next steps

Review the following samples or reference documentation.

- [.NET GeoDR sample](#)
- [Java GeoDR sample](#)
- [.NET - Azure.Messaging.EventHubs samples](#)
- [.NET - Microsoft.Azure.EventHubs samples](#)
- [Java - azure-messaging-eventhubs samples](#)
- [Java - azure-eventhubs samples](#)
- [Python samples](#)
- [JavaScript samples](#)
- [TypeScript samples](#)
- [REST API reference](#)

# Authorize access to Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

Every time you publish or consume events/data from an event hub, your client is trying to access Event Hubs resources. Every request to a secure resource must be authorized so that the service can ensure that the client has the required permissions to publish/consume the data.

Azure Event Hubs offers the following options for authorizing access to secure resources:

- Azure Active Directory
- Shared access signature

## NOTE

This article applies to both Event Hubs and [Apache Kafka](#) scenarios.

## Azure Active Directory

Azure Active Directory (Azure AD) integration for Event Hubs resources provides Azure role-based access control (Azure RBAC) for fine-grained control over a client's access to resources. You can use Azure RBAC to grant permissions to security principal, which may be a user, a group, or an application service principal. The security principal is authenticated by Azure AD to return an OAuth 2.0 token. The token can be used to authorize a request to access an Event Hubs resource.

For more information about authenticating with Azure AD, see the following articles:

- [Authenticate requests to Azure Event Hubs using Azure Active Directory](#)
- [Authorize access to Event Hubs resources using Azure Active Directory](#).

## Shared access signatures

Shared access signatures (SAS) for Event Hubs resources provide limited delegated access to Event Hubs resources. Adding constraints on time interval for which the signature is valid or on permissions it grants provides flexibility in managing resources. For more information, see [Authenticate using shared access signatures \(SAS\)](#).

Authorizing users or applications using an OAuth 2.0 token returned by Azure AD provides superior security and ease of use over shared access signatures (SAS). With Azure AD, there's no need to store the access tokens with your code and risk potential security vulnerabilities. While you can continue to use shared access signatures (SAS) to grant fine-grained access to Event Hubs resources, Azure AD offers similar capabilities without the need to manage SAS tokens or worry about revoking a compromised SAS.

By default, all Event Hubs resources are secured, and are available only to the account owner. Although you can use any of the authorization strategies outlined above to grant clients access to Event Hub resources. Microsoft recommends using Azure AD when possible for maximum security and ease of use.

For more information about authorization using SAS, see [Authorizing access to Event Hubs resources using Shared Access Signatures](#).

## Next steps

- Review [Azure RBAC samples](#) published in our GitHub repository.
- See the following articles:
  - [Authenticate requests to Azure Event Hubs from an application using Azure Active Directory](#)
  - [Authenticate a managed identity with Azure Active Directory to access Event Hubs Resources](#)
  - [Authenticate requests to Azure Event Hubs using Shared Access Signatures](#)
  - [Authorize access to Event Hubs resources using Azure Active Directory](#)
  - [Authorize access to Event Hubs resources using Shared Access Signatures](#)

# Authorize access to Event Hubs resources using Azure Active Directory

11/26/2021 • 4 minutes to read • [Edit Online](#)

Azure Event Hubs supports using Azure Active Directory (Azure AD) to authorize requests to Event Hubs resources. With Azure AD, you can use Azure role-based access control (Azure RBAC) to grant permissions to a security principal, which may be a user, or an application service principal. To learn more about roles and role assignments, see [Understanding the different roles](#).

## Overview

When a security principal (a user, or an application) attempts to access an Event Hubs resource, the request must be authorized. With Azure AD, access to a resource is a two-step process.

1. First, the security principal's identity is authenticated, and an OAuth 2.0 token is returned. The resource name to request a token is `https://eventhubs.azure.net/`, and it's the same for all clouds/tenants. For Kafka clients, the resource to request a token is `https://<namespace>.servicebus.windows.net`.
2. Next, the token is passed as part of a request to the Event Hubs service to authorize access to the specified resource.

The authentication step requires that an application request contains an OAuth 2.0 access token at runtime. If an application is running within an Azure entity such as an Azure VM, a virtual machine scale set, or an Azure Function app, it can use a managed identity to access the resources. To learn how to authenticate requests made by a managed identity to Event Hubs service, see [Authenticate access to Azure Event Hubs resources with Azure Active Directory and managed identities for Azure Resources](#).

The authorization step requires that one or more Azure roles be assigned to the security principal. Azure Event Hubs provides Azure roles that encompass sets of permissions for Event Hubs resources. The roles that are assigned to a security principal determine the permissions that the principal will have. For more information about Azure roles, see [Azure built-in roles for Azure Event Hubs](#).

Native applications and web applications that make requests to Event Hubs can also authorize with Azure AD. To learn how to request an access token and use it to authorize requests for Event Hubs resources, see [Authenticate access to Azure Event Hubs with Azure AD from an application](#).

## Assign Azure roles for access rights

Azure Active Directory (Azure AD) authorizes access rights to secured resources through [Azure role-based access control \(Azure RBAC\)](#). Azure Event Hubs defines a set of Azure built-in roles that encompass common sets of permissions used to access event hub data and you can also define custom roles for accessing the data.

When an Azure role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of subscription, the resource group, the Event Hubs namespace, or any resource under it. An Azure AD security principal may be a user, or an application service principal, or a [managed identity for Azure resources](#).

## Azure built-in roles for Azure Event Hubs

Azure provides the following Azure built-in roles for authorizing access to Event Hubs data using Azure AD and OAuth:

ROLE	DESCRIPTION
Azure Event Hubs Data owner	Use this role to give complete access to Event Hubs resources.
Azure Event Hubs Data sender	Use this role to give the send access to Event Hubs resources.
Azure Event Hubs Data receiver	Use this role to give the consuming/receiving access to Event Hubs resources.

For Schema Registry built-in roles, see [Schema Registry roles](#).

## Resource scope

Before you assign an Azure role to a security principal, determine the scope of access that the security principal should have. Best practices dictate that it's always best to grant only the narrowest possible scope.

The following list describes the levels at which you can scope access to Event Hubs resources, starting with the narrowest scope:

- **Consumer group:** At this scope, role assignment applies only to this entity. Currently, the Azure portal doesn't support assigning an Azure role to a security principal at this level.
- **Event hub:** Role assignment applies to the Event Hub entity and the consumer group under it.
- **Namespace:** Role assignment spans the entire topology of Event Hubs under the namespace and to the consumer group associated with it.
- **Resource group:** Role assignment applies to all the Event Hubs resources under the resource group.
- **Subscription:** Role assignment applies to all the Event Hubs resources in all of the resource groups in the subscription.

### NOTE

- Keep in mind that Azure role assignments may take up to five minutes to propagate.
- This content applies to both Event Hubs and Event Hubs for Apache Kafka. For more information on Event Hubs for Kafka support, see [Event Hubs for Kafka - security and authentication](#).

For more information about how built-in roles are defined, see [Understand role definitions](#). For information about creating Azure custom roles, see [Azure custom roles](#).

## Samples

- [Microsoft.Azure.EventHubs samples](#).

These samples use the old **Microsoft.Azure.EventHubs** library, but you can easily update it to using the latest **Azure.Messaging.EventHubs** library. To move the sample from using the old library to new one, see the [Guide to migrate from Microsoft.Azure.EventHubs to Azure.Messaging.EventHubs](#).

- [Azure.Messaging.EventHubs samples](#)

This sample has been updated to use the latest **Azure.Messaging.EventHubs** library.

- [Event Hubs for Kafka - OAuth samples](#).

## Next steps

- Learn how to assign an Azure built-in role to a security principal, see [Authenticate access to Event Hubs resources using Azure Active Directory](#).
- Learn [how to create custom roles with Azure RBAC](#).
- Learn [how to use Azure Active Directory with EH](#)

See the following related articles:

- [Authenticate requests to Azure Event Hubs from an application using Azure Active Directory](#)
- [Authenticate a managed identity with Azure Active Directory to access Event Hubs Resources](#)
- [Authenticate requests to Azure Event Hubs using Shared Access Signatures](#)
- [Authorize access to Event Hubs resources using Shared Access Signatures](#)

# Authorizing access to Event Hubs resources using Shared Access Signatures

11/26/2021 • 6 minutes to read • [Edit Online](#)

A shared access signature (SAS) provides you with a way to grant limited access to resources in your Event Hubs namespace. SAS guards access to Event Hubs resources based on authorization rules. These rules are configured either on a namespace, or an entity (event hub or topic). This article provides an overview of the SAS model, and reviews SAS best practices.

## What are shared access signatures?

A shared access signature (SAS) provides delegated access to Event Hubs resources based on authorization rules. An authorization rule has a name, is associated with specific rights, and carries a pair of cryptographic keys. You use the rule's name and key via the Event Hubs clients or in your own code to generate SAS tokens. A client can then pass the token to Event Hubs to prove authorization for the requested operation.

SAS is a claim-based authorization mechanism using simple tokens. Using SAS, keys are never passed on the wire. Keys are used to cryptographically sign information that can later be verified by the service. SAS can be used similar to a username and password scheme where the client is in immediate possession of an authorization rule name and a matching key. SAS can be used similar to a federated security model, where the client receives a time-limited and signed access token from a security token service without ever coming into possession of the signing key.

### NOTE

Azure Event Hubs supports authorizing to Event Hubs resources using Azure Active Directory (Azure AD). Authorizing users or applications using OAuth 2.0 token returned by Azure AD provides superior security and ease of use over shared access signatures (SAS). With Azure AD, there is no need to store the tokens in your code and risk potential security vulnerabilities.

Microsoft recommends using Azure AD with your Azure Event Hubs applications when possible. For more information, see [Authorize access to Azure Event Hubs resource using Azure Active Directory](#).

### IMPORTANT

SAS (Shared Access Signatures) tokens are critical to protect your resources. While providing granularity, SAS grants clients access to your Event Hubs resources. They should not be shared publicly. When sharing, if required for troubleshooting reasons, consider using a reduced version of any log files or deleting the SAS tokens (if present) from the log files, and make sure the screenshots don't contain the SAS information either.

## Shared access authorization policies

Each Event Hubs namespace and each Event Hubs entity (an event hub instance or a Kafka topic) has a shared access authorization policy made up of rules. The policy at the namespace level applies to all entities inside the namespace, irrespective of their individual policy configuration. For each authorization policy rule, you decide on three pieces of information: name, scope, and rights. The name is a unique name in that scope. The scope is the URI of the resource in question. For an Event Hubs namespace, the scope is the fully qualified domain name (FQDN), such as `https://<yournamespace>.servicebus.windows.net/`.

The rights provided by the policy rule can be a combination of:

- **Send** – Gives the right to send messages to the entity
- **Listen** – Gives the right to listen or receive to the entity
- **Manage** – Gives the right to manage the topology of the namespace, including creation and deletion of entities

**NOTE**

The **Manage** right includes the **Send** and **Listen** rights.

A namespace or entity policy can hold up to 12 shared access authorization rules, providing room for the three sets of rules, each covering the basic rights, and the combination of Send and Listen. This limit underlines that the SAS policy store isn't intended to be a user or service account store. If your application needs to grant access to Event Hubs resources based on user or service identities, it should implement a security token service that issues SAS tokens after an authentication and access check.

An authorization rule is assigned a **primary key** and a **secondary key**. These keys are cryptographically strong keys. Don't lose them or leak them. They'll always be available in the Azure portal. You can use either of the generated keys, and you can regenerate them at any time. If you regenerate or change a key in the policy, all previously issued tokens based on that key become instantly invalid. However, ongoing connections created based on such tokens will continue to work until the token expires.

When you create an Event Hubs namespace, a policy rule named **RootManageSharedAccessKey** is automatically created for the namespace. This policy has **manage** permissions for the entire namespace. It's recommended that you treat this rule like an administrative root account and don't use it in your application. You can create additional policy rules in the **Configure** tab for the namespace in the portal, via PowerShell or Azure CLI.

## Best practices when using SAS

When you use shared access signatures in your applications, you need to be aware of two potential risks:

- If a SAS is leaked, it can be used by anyone who obtains it, which can potentially compromise your Event Hubs resources.
- If a SAS provided to a client application expires and the application is unable to retrieve a new SAS from your service, then application's functionality may be hindered.

The following recommendations for using shared access signatures can help mitigate these risks:

- **Have clients automatically renew the SAS if necessary:** Clients should renew the SAS well before expiration, to allow time for retries if the service providing the SAS is unavailable. If your SAS is meant to be used for a small number of immediate, short-lived operations that are expected to be completed within the expiration period, then it may be unnecessary as the SAS is not expected to be renewed. However, if you have a client that is routinely making requests via SAS, then the possibility of expiration comes into play. The key consideration is to balance the need for the SAS to be short-lived (as previously stated) with the need to ensure that client is requesting renewal early enough (to avoid disruption due to the SAS expiring prior to a successful renewal).
- **Be careful with the SAS start time:** If you set the start time for SAS to **now**, then due to clock skew (differences in current time according to different machines), failures may be observed intermittently for the first few minutes. In general, set the start time to be at least 15 minutes in the past. Or, don't set it at all, which will make it valid immediately in all cases. The same generally applies to the expiry time as well. Remember that you may observe up to 15 minutes of clock skew in either direction on any request.
- **Be specific with the resource to be accessed:** A security best practice is to provide user with the

minimum required privileges. If a user only needs read access to a single entity, then grant them read access to that single entity, and not read/write/delete access to all entities. It also helps lessen the damage if a SAS is compromised because the SAS has less power in the hands of an attacker.

- **Don't always use SAS:** Sometimes the risks associated with a particular operation against your Event Hubs outweigh the benefits of SAS. For such operations, create a middle-tier service that writes to your Event Hubs after business rule validation, authentication, and auditing.
- **Always use HTTPS:** Always use HTTPS to create or distribute a SAS. If a SAS is passed over HTTP and intercepted, an attacker performing a man-in-the-middle attack is able to read the SAS and then use it just as the intended user could have, potentially compromising sensitive data or allowing for data corruption by the malicious user.

## Conclusion

Share access signatures are useful for providing limited permissions to Event Hubs resources to your clients. They are vital part of the security model for any application using Azure Event Hubs. If you follow the best practices listed in this article, you can use SAS to provide greater flexibility of access to your resources, without compromising the security of your application.

## Next steps

See the following related articles:

- [Authenticate requests to Azure Event Hubs from an application using Azure Active Directory](#)
- [Authenticate a managed identity with Azure Active Directory to access Event Hubs Resources](#)
- [Authenticate requests to Azure Event Hubs using Shared Access Signatures](#)
- [Authorize access to Event Hubs resources using Azure Active Directory](#)

# Authenticate a managed identity with Azure Active Directory to access Event Hubs Resources

11/26/2021 • 5 minutes to read • [Edit Online](#)

Azure Event Hubs supports Azure Active Directory (Azure AD) authentication with [managed identities for Azure resources](#). Managed identities for Azure resources can authorize access to Event Hubs resources using Azure AD credentials from applications running in Azure Virtual Machines (VMs), Function apps, Virtual Machine Scale Sets, and other services. By using managed identities for Azure resources together with Azure AD authentication, you can avoid storing credentials with your applications that run in the cloud.

This article shows how to authorize access to an event hub by using a managed identity from an Azure VM.

## Enable managed identities on a VM

Before you can use managed identities for Azure Resources to authorize Event Hubs resources from your VM, you must first enable managed identities for Azure Resources on the VM. To learn how to enable managed identities for Azure Resources, see one of these articles:

- [Azure portal](#)
- [Azure PowerShell](#)
- [Azure CLI](#)
- [Azure Resource Manager template](#)
- [Azure Resource Manager client libraries](#)

## Grant permissions to a managed identity in Azure AD

To authorize a request to Event Hubs service from a managed identity in your application, first configure Azure role-based access control (Azure RBAC) settings for that managed identity. Azure Event Hubs defines Azure roles that encompass permissions for sending and reading from Event Hubs. When the Azure role is assigned to a managed identity, the managed identity is granted access to Event Hubs data at the appropriate scope.

For more information about assigning Azure roles, see [Authenticate with Azure Active Directory for access to Event Hubs resources](#).

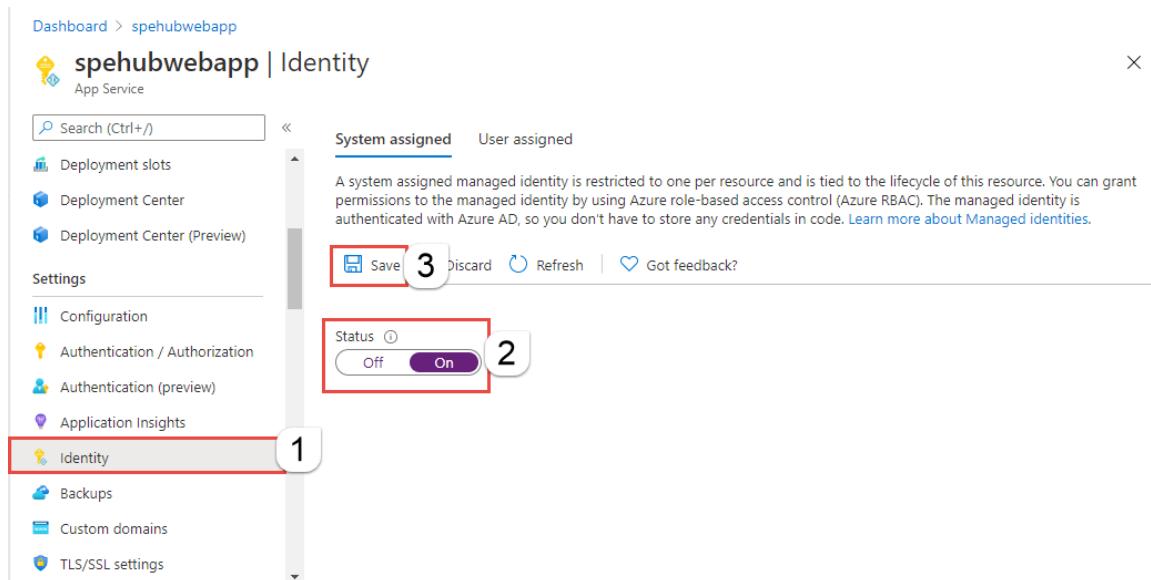
## Use Event Hubs with managed identities

To use Event Hubs with managed identities, you need to assign the role and the appropriate scope to the identity. The procedure in this section uses a simple application that runs under a managed identity and accesses Event Hubs resources.

Here we're using a sample web application hosted in [Azure App Service](#). For step-by-step instructions for creating a web application, see [Create an ASP.NET Core web app in Azure](#)

Once the application is created, follow these steps:

1. Go to **Settings** and select **Identity**.
2. Select the **Status** to be **On**.
3. Select **Save** to save the setting.



#### 4. Select Yes on the information message.

Once you've enabled this setting, a new service identity is created in your Azure Active Directory (Azure AD) and configured into the App Service host.

Now, assign this service identity to a role in the required scope in your Event Hubs resources.

#### To Assign Azure roles using the Azure portal

Assign one of the [Event Hubs roles](#) to the managed identity at the desired scope (Event Hubs namespace, resource group, subscription). For detailed steps, see [Assign Azure roles using the Azure portal](#).

##### NOTE

For a list of services that support managed identities, see [Services that support managed identities for Azure resources](#).

#### Test the web application

1. Create an Event Hubs namespace and an event hub.
2. Deploy the web app to Azure. See the following tabbed section for links to the web application on GitHub.
3. Ensure that the SendReceive.aspx is set as the default document for the web app.
4. Enable **identity** for the web app.
5. Assign this identity to the **Event Hubs Data Owner** role at the namespace level or event hub level.
6. Run the web application, enter the namespace name and event hub name, a message, and select **Send**. To receive the event, select **Receive**.
  - [Azure.Messaging.EventHubs \(latest\)](#)
  - [Microsoft.Azure.EventHubs \(legacy\)](#)

You can now launch your web application and point your browser to the sample aspx page. You can find the sample web application that sends and receives data from Event Hubs resources in the [GitHub repo](#).

Install the latest package from [NuGet](#), and start sending events to Event Hubs using **EventHubProducerClient** and receiving events using **EventHubConsumerClient**.

##### NOTE

For a Java sample that uses a managed identity to publish events to an event hub, see [Publish events with Azure identity sample on GitHub](#).

```

protected async void btnSend_Click(object sender, EventArgs e)
{
    await using (EventHubProducerClient producerClient = new EventHubProducerClient(txtNamespace.Text,
txtEventHub.Text, new DefaultAzureCredential()))
    {
        // create a batch
        using (EventDataBatch eventBatch = await producerClient.CreateBatchAsync())
        {

            // add events to the batch. only one in this case.
            eventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes(txtData.Text)));

            // send the batch to the event hub
            await producerClient.SendAsync(eventBatch);
        }

        txtOutput.Text = $"{DateTime.Now} - SENT{Environment.NewLine}{txtOutput.Text}";
    }
}

protected async void btnReceive_Click(object sender, EventArgs e)
{
    await using (var consumerClient = new
EventHubConsumerClient(EventHubConsumerClient.DefaultConsumerGroupName, $""
{txtNamespace.Text}.servicebus.windows.net", txtEventHub.Text, new DefaultAzureCredential()))
    {
        int eventsRead = 0;
        try
        {
            using CancellationTokenSource cancellationSource = new CancellationTokenSource();
            cancellationSource.CancelAfter(TimeSpan.FromSeconds(5));

            await foreach (PartitionEvent partitionEvent in
consumerClient.ReadEventsAsync(cancellationSource.Token))
            {
                txtOutput.Text = $"Event Read: { Encoding.UTF8.GetString(partitionEvent.Data.Body.ToArray())}
{ Environment.NewLine}" + txtOutput.Text;
                eventsRead++;
            }
        }
        catch (TaskCanceledException ex)
        {
            txtOutput.Text = $"Number of events read: {eventsRead}{ Environment.NewLine}" + txtOutput.Text;
        }
    }
}

```

## Event Hubs for Kafka

You can use Apache Kafka applications to send messages to and receive messages from Azure Event Hubs using managed identity OAuth. See the following sample on GitHub: [Event Hubs for Kafka - send and receive messages using managed identity OAuth](#).

## Samples

- [Azure.Messaging.EventHubs samples](#)
  - [.NET](#)
  - [Java](#)
- [Microsoft.Azure.EventHubs samples](#).

These samples use the old **Microsoft.Azure.EventHubs** library, but you can easily update it to using the latest **Azure.Messaging.EventHubs** library. To move the sample from using the old library to new one,

see the [Guide to migrate from Microsoft.Azure.EventHubs to Azure.Messaging.EventHubs](#). This sample has been updated to use the latest **Azure.Messaging.EventHubs** library.

- [Event Hubs for Kafka - send and receive messages using managed identity OAuth](#)

## Next steps

- See the following article to learn about managed identities for Azure resources: [What is managed identities for Azure resources?](#)
- See the following related articles:
  - [Authenticate requests to Azure Event Hubs from an application using Azure Active Directory](#)
  - [Authenticate requests to Azure Event Hubs using Shared Access Signatures](#)
  - [Authorize access to Event Hubs resources using Azure Active Directory](#)
  - [Authorize access to Event Hubs resources using Shared Access Signatures](#)

# Authenticate an application with Azure Active Directory to access Event Hubs resources

11/26/2021 • 5 minutes to read • [Edit Online](#)

Microsoft Azure provides integrated access control management for resources and applications based on Azure Active Directory (Azure AD). A key advantage of using Azure AD with Azure Event Hubs is that you don't need to store your credentials in the code anymore. Instead, you can request an OAuth 2.0 access token from the Microsoft Identity platform. The resource name to request a token is `https://eventhubs.azure.net/`, and it's the same for all clouds/tenants (For Kafka clients, the resource to request a token is

`https://<namespace>.servicebus.windows.net/`). Azure AD authenticates the security principal (a user, group, or service principal) running the application. If the authentication succeeds, Azure AD returns an access token to the application, and the application can then use the access token to authorize request to Azure Event Hubs resources.

When a role is assigned to an Azure AD security principal, Azure grants access to those resources for that security principal. Access can be scoped to the level of subscription, the resource group, the Event Hubs namespace, or any resource under it. An Azure AD security can assign roles to a user, a group, an application service principal, or a [managed identity for Azure resources](#).

## NOTE

A role definition is a collection of permissions. Azure role-based access control (Azure RBAC) controls how these permissions are enforced through role assignment. A role assignment consists of three elements: security principal, role definition, and scope. For more information, see [Understanding the different roles](#).

## Built-in roles for Azure Event Hubs

Azure provides the following Azure built-in roles for authorizing access to Event Hubs data using Azure AD and OAuth:

- [Azure Event Hubs Data Owner](#): Use this role to give complete access to Event Hubs resources.
- [Azure Event Hubs Data Sender](#): Use this role to give send access to Event Hubs resources.
- [Azure Event Hubs Data Receiver](#): Use this role to give receiving access to Event Hubs resources.

For Schema Registry built-in roles, see [Schema Registry roles](#).

## IMPORTANT

Our preview release supported adding Event Hubs data access privileges to Owner or Contributor role. However, data access privileges for Owner and Contributor role are no longer honored. If you are using the Owner or Contributor role, switch to using the Azure Event Hubs Data Owner role.

## Authenticate from an application

A key advantage of using Azure AD with Event Hubs is that your credentials no longer need to be stored in your code. Instead, you can request an OAuth 2.0 access token from Microsoft identity platform. Azure AD authenticates the security principal (a user, a group, or service principal) running the application. If authentication succeeds, Azure AD returns the access token to the application, and the application can then use

the access token to authorize requests to Azure Event Hubs.

Following sections shows you how to configure your native application or web application for authentication with Microsoft identity platform 2.0. For more information about Microsoft identity platform 2.0, see [Microsoft identity platform \(v2.0\) overview](#).

For an overview of the OAuth 2.0 code grant flow, see [Authorize access to Azure Active Directory web applications using the OAuth 2.0 code grant flow](#).

### Register your application with an Azure AD tenant

The first step in using Azure AD to authorize Event Hubs resources is registering your client application with an Azure AD tenant from the [Azure portal](#). When you register your client application, you supply information about the application to AD. Azure AD then provides a client ID (also called an application ID) that you can use to associate your application with Azure AD runtime. To learn more about the client ID, see [Application and service principal objects in Azure Active Directory](#).

The following images show steps for registering a web application:

The screenshot shows the 'Register an application' wizard in the Azure portal. It includes fields for 'Name', 'Supported account types', 'Redirect URI (optional)', and a note about agreeing to Microsoft Platform Policies. A 'Register' button is at the bottom.

Dashboard > App registrations > Register an application

## Register an application

**\* Name**  
The user-facing display name for this application (this can be changed later).

|

**Supported account types**  
Who can use this application or access this API?

Accounts in this organizational directory only (Default Directory only - Single tenant)

Accounts in any organizational directory (Any Azure AD directory - Multitenant)

Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)

[Help me choose...](#)

**Redirect URI (optional)**  
We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web e.g. <https://myapp.com/auth>

By proceeding, you agree to the Microsoft Platform Policies [\[link\]](#)

**Register**

#### NOTE

If you register your application as a native application, you can specify any valid URI for the Redirect URI. For native applications, this value does not have to be a real URL. For web applications, the redirect URI must be a valid URI, because it specifies the URL to which tokens are provided.

After you've registered your application, you'll see the **Application (client) ID** under **Settings**:

The screenshot shows the Azure portal's App registrations section. On the left, a sidebar lists options like Overview, Quickstart, Manage, API permissions, and Support + Troubleshooting. The main area displays details for the application 'mywebappregistration'. Key fields shown include Display name, Application (client) ID (highlighted with a red box), Directory (tenant) ID, Object ID, Supported account types (My organization only), Redirect URIs (1 web, 0 public client), and Managed application in local directory. Below this, sections for Call APIs (with icons for various Microsoft services) and Documentation (links to Microsoft identity platform, authentication scenarios, etc.) are visible. A 'View API Permissions' button is at the bottom of the main content area.

For more information about registering an application with Azure AD, see [Integrating applications with Azure Active Directory](#).

### Create a client secret

The application needs a client secret to prove its identity when requesting a token. To add the client secret, follow these steps.

1. Navigate to your app registration in the Azure portal.
2. Select the **Certificates & secrets** setting.
3. Under **Client secrets**, select **New client secret** to create a new secret.
4. Provide a description for the secret, and choose the wanted expiration interval.
5. Immediately copy the value of the new secret to a secure location. The fill value is displayed to you only once.

The screenshot shows the 'Client secrets' section within the 'Certificates & secrets' blade. It includes a description of what a client secret is and a link to learn more. A table lists existing secrets, with one entry for 'EventHubapplicationsecret1' showing its expiration date (7/25/2020) and a redacted value. A 'New client secret' button is at the top left of the table area.

DESCRIPTION	EXPIRES	VALUE
EventHubapplicationsecret1	7/25/2020	*@*****

## Assign Azure roles using the Azure portal

Assign one of the [Event Hubs roles](#) to the application's service principal at the desired scope (Event Hubs namespace, resource group, subscription). For detailed steps, see [Assign Azure roles using the Azure portal](#).

Once you define the role and its scope, you can test this behavior with samples [in this GitHub location](#). To learn more on managing access to Azure resources using Azure RBAC and the Azure portal, see [this article](#).

### Client libraries for token acquisition

Once you've registered your application and granted it permissions to send/receive data in Azure Event Hubs, you can add code to your application to authenticate a security principal and acquire OAuth 2.0 token. To authenticate and acquire the token, you can use either one of the [Microsoft identity platform authentication libraries](#) or another open-source library that supports OpenID or Connect 1.0. Your application can then use the

access token to authorize a request against Azure Event Hubs.

For a list of scenarios for which acquiring tokens is supported, see the [Scenarios](#) section of the [Microsoft Authentication Library \(MSAL\) for .NET](#) GitHub repository.

## Samples

- [Microsoft.Azure.EventHubs samples](#).

These samples use the old **Microsoft.Azure.EventHubs** library, but you can easily update it to using the latest **Azure.Messaging.EventHubs** library. To move the sample from using the old library to new one, see the [Guide to migrate from Microsoft.Azure.EventHubs to Azure.Messaging.EventHubs](#).

- [Azure.Messaging.EventHubs samples](#)

This sample has been updated to use the latest **Azure.Messaging.EventHubs** library.

## Next steps

- To learn more about Azure RBAC, see [What is Azure role-based access control \(Azure RBAC\)?](#)
- To learn how to assign and manage Azure role assignments with Azure PowerShell, Azure CLI, or the REST API, see these articles:
  - [Add or remove Azure role assignments using Azure PowerShell](#)
  - [Add or remove Azure role assignments using Azure CLI](#)
  - [Add or remove Azure role assignments using the REST API](#)
  - [Add Azure role assignments using Azure Resource Manager templates](#)

See the following related articles:

- [Authenticate a managed identity with Azure Active Directory to access Event Hubs Resources](#)
- [Authenticate requests to Azure Event Hubs using Shared Access Signatures](#)
- [Authorize access to Event Hubs resources using Azure Active Directory](#)
- [Authorize access to Event Hubs resources using Shared Access Signatures](#)

# Authenticate access to Event Hubs resources using shared access signatures (SAS)

11/26/2021 • 9 minutes to read • [Edit Online](#)

Shared access signature (SAS) gives you granular control over the type of access you grant to the clients who has the shared access signature. Here are some of the controls you can set in a SAS:

- The interval over which the SAS is valid, including the start time and expiry time.
- The permissions granted by the SAS. For example, a SAS for an Event Hubs namespace might grant the listen permission, but not the send permission.
- Only clients that present valid credentials can send data to an event hub.
- A client can't impersonate another client.
- A rogue client can be blocked from sending data to an event hub.

This article covers authenticating the access to Event Hubs resources using SAS. To learn about **authorizing** access to Event Hubs resources using SAS, see [this article](#).

## NOTE

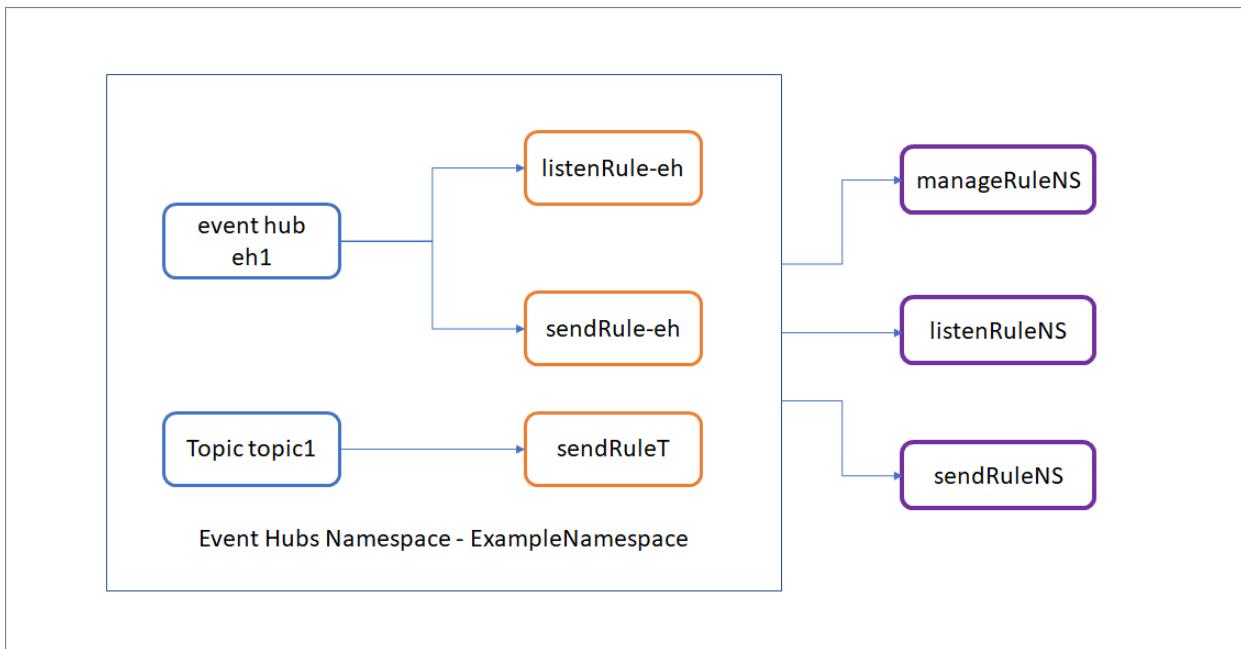
Microsoft recommends that you use Azure AD credentials when possible as a security best practice, rather than using the shared access signatures, which can be more easily compromised. While you can continue to use shared access signatures (SAS) to grant fine-grained access to your Event Hubs resources, Azure AD offers similar capabilities without the need to manage SAS tokens or worry about revoking a compromised SAS.

For more information about Azure AD integration in Azure Event Hubs, see [Authorize access to Event Hubs using Azure AD](#).

## Configuring for SAS authentication

You can configure the EventHubs shared access authorization rule on an Event Hubs namespace, or an entity (event hub instance or Kafka Topic in an event hub). Configuring a shared access authorization rule on a consumer group is currently not supported, but you can use rules configured on a namespace or entity to secure access to consumer group.

The following image shows how the authorization rules apply on sample entities.



In this example, the sample Event Hubs namespace (ExampleNamespace) has two entities: `eh1` and `topic1`. The authorization rules are defined both at the entity level and also at the namespace level.

The `manageRuleNS`, `sendRuleNS`, and `listenRuleNS` authorization rules apply to both event hub instance `eh1` and topic `t1`. The `listenRule-eh` and `sendRule-eh` authorization rules apply only to event hub instance `eh1` and `sendRuleT` authorization rule applies only to topic `topic1`.

When using `sendRuleNS` authorization rule, client applications can send to both `eh1` and `topic1`. When `sendRuleT` authorization rule is used, it enforces granular access to `topic1` only and hence client applications using this rule for access now cannot send to `eh1`, but only to `topic1`.

## Generate a Shared Access Signature token

Any client that has access to name of an authorization rule name and one of its signing keys can generate a SAS token. The token is generated by crafting a string in the following format:

- `se` – Token expiry instant. Integer reflecting seconds since epoch 00:00:00 UTC on 1 January 1970 (UNIX epoch) when the token expires
- `skn` – Name of the authorization rule, that is the SAS key name.
- `sr` – URI of the resource being accessed.
- `sig` – Signature.

The signature-string is the SHA-256 hash computed over the resource URI (scope as described in the previous section) and the string representation of the token expiry instant, separated by CRLF.

The hash computation looks similar to the following pseudo code and returns a 256-bit/32-byte hash value.

```
SHA-256('https://<yournamespace>.servicebus.windows.net/' + '\n' + 1438205742)
```

The token contains the non-hashed values so that the recipient can recompute the hash with the same parameters, verifying that the issuer is in possession of a valid signing key.

The resource URI is the full URI of the Service Bus resource to which access is claimed. For example,

`http://<namespace>.servicebus.windows.net/<entityPath>` or  
`sb://<namespace>.servicebus.windows.net/<entityPath>` that is, `http://contoso.servicebus.windows.net/eh1`.

The URI must be percent-encoded.

The shared access authorization rule used for signing must be configured on the entity specified by this URL, or by one of its hierarchical parents. For example, `http://contoso.servicebus.windows.net/eh1` or `http://contoso.servicebus.windows.net` in the previous example.

A SAS token is valid for all resources prefixed with the `<resourceURI>` used in the signature-string.

#### NOTE

You generate an access token for Event Hubs using shared access policy. For more information, see [Shared access authorization policy](#).

## Generating a signature(token) from a policy

Following section shows generating a SAS token using shared access signature policies,

#### NodeJS

```
function createSharedAccessToken(uri, saName, saKey) {
    if (!uri || !saName || !saKey) {
        throw "Missing required parameter";
    }
    var encoded = encodeURIComponent(uri);
    var now = new Date();
    var week = 60*60*24*7;
    var ttl = Math.round(now.getTime() / 1000) + week;
    var signature = encoded + '\n' + ttl;
    var signatureUTF8 = utf8.encode(signature);
    var hash = crypto.createHmac('sha256', saKey).update(signatureUTF8).digest('base64');
    return 'SharedAccessSignature sr=' + encoded + '&sig=' +
        encodeURIComponent(hash) + '&se=' + ttl + '&skn=' + saName;
```

#### JAVA

```

private static String GetSASToken(String resourceUri, String keyName, String key)
{
    long epoch = System.currentTimeMillis()/1000L;
    int week = 60*60*24*7;
    String expiry = Long.toString(epoch + week);

    String sasToken = null;
    try {
        String stringToSign = URLEncoder.encode(resourceUri, "UTF-8") + "\n" + expiry;
        String signature = getHMAC256(key, stringToSign);
        sasToken = "SharedAccessSignature sr=" + URLEncoder.encode(resourceUri, "UTF-8") + "&sig=" +
                   URLEncoder.encode(signature, "UTF-8") + "&se=" + expiry + "&skn=" + keyName;
    } catch (UnsupportedEncodingException e) {

        e.printStackTrace();
    }

    return sasToken;
}

public static String getHMAC256(String key, String input) {
    Mac sha256_HMAC = null;
    String hash = null;
    try {
        sha256_HMAC = Mac.getInstance("HmacSHA256");
        SecretKeySpec secret_key = new SecretKeySpec(key.getBytes(), "HmacSHA256");
        sha256_HMAC.init(secret_key);
        Encoder encoder = Base64.getEncoder();

        hash = new String(encoder.encode(sha256_HMAC.doFinal(input.getBytes("UTF-8"))));

    } catch (InvalidKeyException e) {
        e.printStackTrace();
    } catch (NoSuchAlgorithmException e) {
        e.printStackTrace();
    } catch (IllegalStateException e) {
        e.printStackTrace();
    } catch (UnsupportedEncodingException e) {
        e.printStackTrace();
    }
}

return hash;
}

```

## PHP

```

function generateSasToken($uri, $sasKeyName, $sasKeyValue)
{
    $targetUri = strtolower(rawurlencode(strtolower($uri)));
    $expires = time();
    $expiresInMins = 60;
    $week = 60*60*24*7;
    $expires = $expires + $week;
    $toSign = $targetUri . "\n" . $expires;
    $signature = rawurlencode(base64_encode(hash_hmac('sha256',
        $toSign, $sasKeyValue, TRUE)));

    $token = "SharedAccessSignature sr=" . $targetUri . "&sig=" . $signature . "&se=" . $expires . "&skn=" .
    $sasKeyName;
    return $token;
}

```

## C#

```

private static string createToken(string resourceUri, string keyName, string key)
{
    TimeSpan sinceEpoch = DateTime.UtcNow - new DateTime(1970, 1, 1);
    var week = 60 * 60 * 24 * 7;
    var expiry = Convert.ToString((int)sinceEpoch.TotalSeconds + week);
    string stringToSign = HttpUtility.UrlEncode(resourceUri) + "\n" + expiry;
    HMACSHA256 hmac = new HMACSHA256(Encoding.UTF8.GetBytes(key));
    var signature = Convert.ToBase64String(hmac.ComputeHash(Encoding.UTF8.GetBytes(stringToSign)));
    var sasToken = String.Format(CultureInfo.InvariantCulture, "SharedAccessSignature sr={0}&sig={1}&se={2}&swn={3}", HttpUtility.UrlEncode(resourceUri), HttpUtility.UrlEncode(signature), expiry, keyName);
    return sasToken;
}

```

## Authenticating Event Hubs publishers with SAS

An event publisher defines a virtual endpoint for an event hub. The publisher can only be used to send messages to an event hub and not receive messages.

Typically, an event hub employs one publisher per client. All messages that are sent to any of the publishers of an event hub are enqueued within that event hub. Publishers enable fine-grained access control.

Each Event Hubs client is assigned a unique token, which is uploaded to the client. The tokens are produced such that each unique token grants access to different unique publisher. A client that holds a token can only send to one publisher, and no other publisher. If multiple clients share the same token, then each of them shares the publisher.

All tokens are assigned with SAS keys. Typically, all tokens are signed with the same key. Clients aren't aware of the key, which prevents clients from manufacturing tokens. Clients operate on the same tokens until they expire.

For example, to define authorization rules scoped down to only sending/publishing to Event Hubs, you need to define a send authorization rule. This can be done at a namespace level or give more granular scope to a particular entity (event hubs instance or a topic). A client or an application that is scoped with such granular access is called, Event Hubs publisher. To do so, follow these steps:

1. Create a SAS key on the entity you want to publish to assign the **send** scope on it. For more information, see [Shared access authorization policies](#).
2. Generate a SAS token with an expiry time for a specific publisher by using the key generated in step1.

```

var sasToken = SharedAccessSignatureTokenProvider.GetPublisherSharedAccessSignature(
    new Uri("Service-Bus-URI"),
    "eventub-name",
    "publisher-name",
    "sas-key-name",
    "sas-key",
    TimeSpan.FromMinutes(30));

```

3. Provide the token to the publisher client, which can only send to the entity and the publisher that token grants access to.

Once the token expires, the client loses its access to send/publish to the entity.

#### NOTE

Although it's not recommended, it is possible to equip devices with tokens that grant access to an event hub or a namespace. Any device that holds this token can send messages directly to that event hub. Furthermore, the device cannot be blacklisted from sending to that event hub.

It's always recommended to give specific and granular scopes.

#### IMPORTANT

Once the tokens have been created, each client is provisioned with its own unique token.

When the client sends data into an event hub, it tags its request with the token. To prevent an attacker from eavesdropping and stealing the token, the communication between the client and the event hub must occur over an encrypted channel.

If a token is stolen by an attacker, the attacker can impersonate the client whose token has been stolen. Blocklisting a publisher, renders that client unusable until it receives a new token that uses a different publisher.

## Authenticating Event Hubs consumers with SAS

To authenticate back-end applications that consume from the data generated by Event Hubs producers, Event Hubs token authentication requires its clients to either have the **manage** rights or the **listen** privileges assigned to its Event Hubs namespace or event hub instance or topic. Data is consumed from Event Hubs using consumer groups. While SAS policy gives you granular scope, this scope is defined only at the entity level and not at the consumer level. It means that the privileges defined at the namespace level or the event hub instance or topic level will be applied to the consumer groups of that entity.

## Disabling Local/SAS Key authentication

For certain organizational security requirements, you may have to disable local/SAS key authentication completely and rely on the Azure Active Directory (Azure AD) based authentication which is the recommended way to connect with Azure Event Hubs. You can disable local/SAS key authentication at the Event Hubs namespace level using Azure portal or Azure Resource Manager template.

### Disabling Local/SAS Key authentication via the portal

You can disable local/SAS key authentication for a given Event Hubs namespace using the Azure portal.

As shown in the following image, in the namespace overview section, click on the *Local Authentication*.

The screenshot shows the Azure Event Hubs namespace overview page. At the top, there are buttons for '+ Event Hub', 'Delete', and 'Refresh'. Below that is a section titled 'Essentials' with a collapse arrow. Under 'Resource group (change)', it says 'TestPPG'. In the 'Status' row, 'Active' is listed. In the 'Location' row, 'East US' is listed. In the 'Subscription (change)' row, 'SB-DF-EH-Archive' is listed. In the 'Subscription ID' row, there is a placeholder ':'. In the 'Host name' row, there is a placeholder ':'. On the right side, there are several properties listed: 'Created' (Wednesday, September 1, 2021, 16:39:45 PDT), 'Updated' (Wednesday, September 1, 2021, 16:43:20 PDT), 'Zone Redundancy' (Not Enabled), 'Pricing tier' (Standard), 'Throughput Units' (1 unit), 'Auto-inflate throughput ...' (Disabled), and 'Local Authentication' (Disabled). The 'Local Authentication' row is highlighted with a red border.

And then select *Disabled* option and click *Ok* as shown below.

### Local Authentication

Use this setting to enable or disable local authentication (SAS key authentication).

Enabled  Disabled

**OK**

**Cancel**

## Disabling Local/SAS Key authentication using a template

You can disable local authentication for a given Event Hubs namespace by setting `disableLocalAuth` property to `true` as shown in the following Azure Resource Manager template(ARM Template).

```
"resources": [
  {
    "apiVersion": "[variables('ehVersion')]",
    "name": "[parameters('eventHubNamespaceName')]",
    "type": "Microsoft.EventHub/Namespace",
    "location": "[variables('location')]",
    "sku": {
      "name": "Standard",
      "tier": "Standard"
    },
    "resources": [
      {
        "apiVersion": "2017-04-01",
        "name": "[parameters('eventHubNamespaceName')]",
        "type": "Microsoft.EventHub/Namespace",
        "location": "[resourceGroup().location]",
        "sku": {
          "name": "Standard"
        },
        "properties": {
          "isAutoInflateEnabled": "true",
          "maximumThroughputUnits": "7",
          "disableLocalAuth": false
        },
        "resources": [
          {
            "apiVersion": "2017-04-01",
            "name": "[parameters('eventHubName')]",
            "type": "EventHubs",
            "dependsOn": [
              "[concat('Microsoft.EventHub/namespaces/', parameters('eventHubNamespaceName'))]"
            ],
            "properties": {
              "messageRetentionInDays": "[parameters('messageRetentionInDays')]",
              "partitionCount": "[parameters('partitionCount')]"
            }
          }
        ]
      }
    ]
  }
]
```

## Next steps

See the following articles:

- [Authorize using SAS](#)
- [Authorize using Azure role-based access control \(Azure RBAC\)](#)
- [Learn more about Event Hubs](#)

See the following related articles:

- [Authenticate requests to Azure Event Hubs from an application using Azure Active Directory](#)
- [Authenticate a managed identity with Azure Active Directory to access Event Hubs Resources](#)
- [Authorize access to Event Hubs resources using Azure Active Directory](#)
- [Authorize access to Event Hubs resources using Shared Access Signatures](#)

# Network security for Azure Event Hubs

11/26/2021 • 5 minutes to read • [Edit Online](#)

This article describes how to use the following security features with Azure Event Hubs:

- Service tags
- IP Firewall rules
- Network service endpoints
- Private endpoints

## Service tags

A service tag represents a group of IP address prefixes from a given Azure service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change, minimizing the complexity of frequent updates to network security rules. For more information about service tags, see [Service tags overview](#).

You can use service tags to define network access controls on [network security groups](#) or [Azure Firewall](#). Use service tags in place of specific IP addresses when you create security rules. By specifying the service tag name (for example, `EventHub`) in the appropriate *source* or *destination* field of a rule, you can allow or deny the traffic for the corresponding service.

SERVICE TAG	PURPOSE	CAN USE INBOUND OR OUTBOUND?	CAN BE REGIONAL?	CAN USE WITH AZURE FIREWALL?
EventHub	Azure Event Hubs.	Outbound	Yes	Yes

## IP firewall

By default, Event Hubs namespaces are accessible from internet as long as the request comes with valid authentication and authorization. With IP firewall, you can restrict it further to only a set of IPv4 addresses or IPv4 address ranges in [CIDR \(Classless Inter-Domain Routing\)](#) notation.

This feature is helpful in scenarios in which Azure Event Hubs should be only accessible from certain well-known sites. Firewall rules enable you to configure rules to accept traffic originating from specific IPv4 addresses. For example, if you use Event Hubs with [Azure Express Route](#), you can create a **firewall rule** to allow traffic from only your on-premises infrastructure IP addresses.

The IP firewall rules are applied at the Event Hubs namespace level. Therefore, the rules apply to all connections from clients using any supported protocol. Any connection attempt from an IP address that does not match an allowed IP rule on the Event Hubs namespace is rejected as unauthorized. The response does not mention the IP rule. IP filter rules are applied in order, and the first rule that matches the IP address determines the accept or reject action.

For more information, see [How to configure IP firewall for an event hub](#)

## Network service endpoints

The integration of Event Hubs with [Virtual Network \(VNet\) Service Endpoints](#) enables secure access to messaging capabilities from workloads such as virtual machines that are bound to virtual networks, with the network traffic path being secured on both ends.

Once configured to bind to at least one virtual network subnet service endpoint, the respective Event Hubs namespace no longer accepts traffic from anywhere but authorized subnets in virtual networks. From the virtual network perspective, binding an Event Hubs namespace to a service endpoint configures an isolated networking tunnel from the virtual network subnet to the messaging service.

The result is a private and isolated relationship between the workloads bound to the subnet and the respective Event Hubs namespace, in spite of the observable network address of the messaging service endpoint being in a public IP range. There is an exception to this behavior. Enabling a service endpoint, by default, enables the `denyall` rule in the [IP firewall](#) associated with the virtual network. You can add specific IP addresses in the IP firewall to enable access to the Event Hub public endpoint.

#### IMPORTANT

Virtual networks aren't supported in the [basic](#) and [premium](#) tiers.

### Advanced security scenarios enabled by VNet integration

Solutions that require tight and compartmentalized security, and where virtual network subnets provide the segmentation between the compartmentalized services, still need communication paths between services residing in those compartments.

Any immediate IP route between the compartments, including those carrying HTTPS over TCP/IP, carries the risk of exploitation of vulnerabilities from the network layer on up. Messaging services provide insulated communication paths, where messages are even written to disk as they transition between parties. Workloads in two distinct virtual networks that are both bound to the same Event Hubs instance can communicate efficiently and reliably via messages, while the respective network isolation boundary integrity is preserved.

That means your security sensitive cloud solutions not only gain access to Azure industry-leading reliable and scalable asynchronous messaging capabilities, but they can now use messaging to create communication paths between secure solution compartments that are inherently more secure than what is achievable with any peer-to-peer communication mode, including HTTPS and other TLS-secured socket protocols.

#### Bind event hubs to virtual networks

[Virtual network rules](#) are the firewall security feature that controls whether your Azure Event Hubs namespace accepts connections from a particular virtual network subnet.

Binding an Event Hubs namespace to a virtual network is a two-step process. You first need to create a [virtual Network service endpoint](#) on a virtual network's subnet and enable it for [Microsoft.EventHub](#) as explained in the [service endpoint overview](#) article. Once you have added the service endpoint, you bind the Event Hubs namespace to it with a [virtual network rule](#).

The virtual network rule is an association of the Event Hubs namespace with a virtual network subnet. While the rule exists, all workloads bound to the subnet are granted access to the Event Hubs namespace. Event Hubs itself never establishes outbound connections, doesn't need to gain access, and is therefore never granted access to your subnet by enabling this rule.

For more information, see [How to configure virtual network service endpoints for an event hub](#)

## Private endpoints

[Azure Private Link service](#) enables you to access Azure Services (for example, Azure Event Hubs, Azure Storage, and Azure Cosmos DB) and Azure hosted customer/partner services over a [private endpoint](#) in your virtual network.

A private endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. The private endpoint uses a private IP address from your VNet, effectively bringing the service into

your VNet. All traffic to the service can be routed through the private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are needed. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. You can connect to an instance of an Azure resource, giving you the highest level of granularity in access control.

**IMPORTANT**

This feature isn't supported in the **basic** tier.

For more information, see [How to configure private endpoints for an event hub](#)

## Next steps

See the following articles:

- [How to configure IP firewall for an event hub](#)
- [How to configure virtual network service endpoints for an event hub](#)
- [How to configure private endpoints for an event hub](#)

# Azure Policy Regulatory Compliance controls for Azure Event Hubs

11/26/2021 • 8 minutes to read • [Edit Online](#)

Regulatory Compliance in Azure Policy provides Microsoft created and managed initiative definitions, known as **built-ins**, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure Event Hubs. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

## IMPORTANT

Each control below is associated with one or more Azure Policy definitions. These policies may help you [assess compliance](#) with the control; however, there often is not a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves; this doesn't ensure you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards may change over time.

## Azure Security Benchmark

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Logging and Threat Detection	LT-4	Enable logging for Azure resources	<a href="#">Resource logs in Event Hub should be enabled</a>	5.0.0

## Azure Security Benchmark v1

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Network Security	1.1	Protect resources using Network Security Groups or Azure Firewall on your Virtual Network	<a href="#">Event Hub should use a virtual network service endpoint</a>	1.0.0
Logging and Monitoring	2.3	Enable audit logging for Azure resources	<a href="#">Resource logs in Event Hub should be enabled</a>	5.0.0

## CIS Microsoft Azure Foundations Benchmark 1.3.0

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - CIS Microsoft Azure Foundations Benchmark 1.3.0](#). For more information about this compliance standard, see [CIS Microsoft Azure Foundations Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Logging and Monitoring	5.3	Ensure that Diagnostic Logs are enabled for all services which support it.	<a href="#">Resource logs in Event Hub should be enabled</a>	5.0.0

## FedRAMP High

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP High](#). For more information about this compliance standard, see [FedRAMP High](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	<a href="#">Event Hub namespaces should use private link</a>	1.0.0
Access Control	AC-17	Remote Access	<a href="#">Event Hub namespaces should use private link</a>	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	<a href="#">Event Hub namespaces should use private link</a>	1.0.0
Audit and Accountability	AU-6 (4)	Central Review and Analysis	<a href="#">Resource logs in Event Hub should be enabled</a>	5.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Audit and Accountability	AU-6 (5)	Integration / Scanning and Monitoring Capabilities	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-12	Audit Generation	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-12 (1)	System-wide / Time-correlated Audit Trail	Resource logs in Event Hub should be enabled	5.0.0
System and Communications Protection	SC-7	Boundary Protection	Event Hub namespaces should use private link	1.0.0
System and Communications Protection	SC-7 (3)	Access Points	Event Hub namespaces should use private link	1.0.0
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Event Hub namespaces should use a customer-managed key for encryption	1.0.0

## FedRAMP Moderate

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - FedRAMP Moderate](#). For more information about this compliance standard, see [FedRAMP Moderate](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Event Hub namespaces should use private link	1.0.0
Access Control	AC-17	Remote Access	Event Hub namespaces should use private link	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Event Hub namespaces should use private link	1.0.0
Audit and Accountability	AU-12	Audit Generation	Resource logs in Event Hub should be enabled	5.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-7	Boundary Protection	Event Hub namespaces should use private link	1.0.0
System and Communications Protection	SC-7 (3)	Access Points	Event Hub namespaces should use private link	1.0.0
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Event Hub namespaces should use a customer-managed key for encryption	1.0.0

## HIPAA HITRUST 9.2

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - HIPAA HITRUST 9.2](#). For more information about this compliance standard, see [HIPAA HITRUST 9.2](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Segregation in Networks	0805.01m1Organizational.12 - 01.m	The organization's security gateways (e.g. firewalls) enforce security policies and are configured to filter traffic between domains, block unauthorized access, and are used to maintain segregation between internal wired, internal wireless, and external network segments (e.g., the Internet) including DMZs and enforce access control policies for each of the domains.	Event Hub should use a virtual network service endpoint	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
Segregation in Networks	0806.01m2Organizational.12356 - 01.m	The organization's network is logically and physically segmented with a defined security perimeter and a graduated set of controls, including subnetworks for publicly accessible system components that are logically separated from the internal network, based on organizational requirements; and traffic is controlled based on functionality required and classification of the data/systems based on a risk assessment and their respective security requirements.	<a href="#">Event Hub should use a virtual network service endpoint</a>	1.0.0
Segregation in Networks	0894.01m2Organizational.7 - 01.m	Networks are segregated from production-level networks when migrating physical servers, applications or data to virtualized servers.	<a href="#">Event Hub should use a virtual network service endpoint</a>	1.0.0
Audit Logging	1207.09aa2System.4 - 09.aa	Audit records are retained for 90 days and older audit records are archived for one year.	<a href="#">Resource logs in Event Hub should be enabled</a>	5.0.0
Network Controls	0863.09m2Organizational.910 - 09.m	The organization builds a firewall configuration that restricts connections between un-trusted networks and any system components in the covered information environment; and any changes to the firewall configuration are updated in the network diagram.	<a href="#">Event Hub should use a virtual network service endpoint</a>	1.0.0

## New Zealand ISM Restricted

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - New Zealand ISM Restricted](#). For more information about this compliance standard, see [New Zealand ISM Restricted](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control and Passwords	AC-17	16.6.9 Events to be logged	Resource logs in Event Hub should be enabled	5.0.0

## NIST SP 800-53 Rev. 4

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 4](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 4](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Event Hub namespaces should use private link	1.0.0
Access Control	AC-17	Remote Access	Event Hub namespaces should use private link	1.0.0
Access Control	AC-17 (1)	Automated Monitoring / Control	Event Hub namespaces should use private link	1.0.0
Audit and Accountability	AU-6 (4)	Central Review and Analysis	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-6 (5)	Integration / Scanning and Monitoring Capabilities	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-12	Audit Generation	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-12 (1)	System-wide / Time-correlated Audit Trail	Resource logs in Event Hub should be enabled	5.0.0
System and Communications Protection	SC-7	Boundary Protection	Event Hub namespaces should use private link	1.0.0
System and Communications Protection	SC-7 (3)	Access Points	Event Hub namespaces should use private link	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Event Hub namespaces should use a customer-managed key for encryption	1.0.0

## NIST SP 800-53 Rev. 5

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - NIST SP 800-53 Rev. 5](#). For more information about this compliance standard, see [NIST SP 800-53 Rev. 5](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Access Control	AC-4	Information Flow Enforcement	Event Hub namespaces should use private link	1.0.0
Access Control	AC-17	Remote Access	Event Hub namespaces should use private link	1.0.0
Access Control	AC-17 (1)	Monitoring and Control	Event Hub namespaces should use private link	1.0.0
Audit and Accountability	AU-6 (4)	Central Review and Analysis	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-6 (5)	Integrated Analysis of Audit Records	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-12	Audit Record Generation	Resource logs in Event Hub should be enabled	5.0.0
Audit and Accountability	AU-12 (1)	System-wide and Time-correlated Audit Trail	Resource logs in Event Hub should be enabled	5.0.0
System and Communications Protection	SC-7	Boundary Protection	Event Hub namespaces should use private link	1.0.0
System and Communications Protection	SC-7 (3)	Access Points	Event Hub namespaces should use private link	1.0.0

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY	POLICY VERSION
System and Communications Protection	SC-12	Cryptographic Key Establishment and Management	Event Hub namespaces should use a customer-managed key for encryption	1.0.0

## Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#).

# Availability and consistency in Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

This article provides information about availability and consistency supported by Azure Event Hubs.

## Availability

Azure Event Hubs spreads the risk of catastrophic failures of individual machines or even complete racks across clusters that span multiple failure domains within a datacenter. It implements transparent failure detection and failover mechanisms such that the service will continue to operate within the assured service-levels and typically without noticeable interruptions when such failures occur.

If an Event Hubs namespace has been created with [availability zones](#) enabled, the outage risk is further spread across three physically separated facilities, and the service has enough capacity reserves to instantly cope up with the complete, catastrophic loss of the entire facility. For more information, see [Azure Event Hubs - Geo-disaster recovery](#).

When a client application sends events to an event hub without specifying a partition, events are automatically distributed among partitions in your event hub. If a partition isn't available for some reason, events are distributed among the remaining partitions. This behavior allows for the greatest amount of up time. For use cases that require the maximum up time, this model is preferred instead of sending events to a specific partition.

## Consistency

In some scenarios, the ordering of events can be important. For example, you may want your back-end system to process an update command before a delete command. In this scenario, a client application sends events to a specific partition so that the ordering is preserved. When a consumer application consumes these events from the partition, they are read in order.

With this configuration, keep in mind that if the particular partition to which you are sending is unavailable, you will receive an error response. As a point of comparison, if you don't have an affinity to a single partition, the Event Hubs service sends your event to the next available partition.

Therefore, if high availability is most important, don't target a specific partition (using partition ID/key). Using partition ID/key downgrades the availability of an event hub to partition-level. In this scenario, you are making an explicit choice between availability (no partition ID/key) and consistency (pinning events to a specific partition). For detailed information about partitions in Event Hubs, see [Partitions](#).

## Appendix

### **Send events without specifying a partition**

We recommend sending events to an event hub without setting partition information to allow the Event Hubs service to balance the load across partitions. See the following quick starts to learn how to do so in different programming languages.

- [Send events using .NET](#)
- [Send events using Java](#)
- [Send events using JavaScript](#)
- [Send events using Python](#)

### **Send events to a specific partition**

In this section, you learn how to send events to a specific partition using different programming languages.

- [.NET](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)

To send events to a specific partition, create the batch using the [EventHubProducerClient.CreateBatchAsync](#) method by specifying either the `PartitionId` or the `PartitionKey` in [CreateBatchOptions](#). The following code sends a batch of events to a specific partition by specifying a partition key. Event Hubs ensures that all events sharing a partition key value are stored together and delivered in order of arrival.

```
var batchOptions = new CreateBatchOptions { PartitionKey = "cities" };
using var eventBatch = await producer.CreateBatchAsync(batchOptions);
```

You can also use the [EventHubProducerClient.SendAsync](#) method by specifying either `PartitionId` or `PartitionKey` in [SendEventOptions](#).

```
var sendEventOptions = new SendEventOptions { PartitionKey = "cities" };
// create the events array
producer.SendAsync(events, sendOptions)
```

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs service overview](#)
- [Event Hubs terminology](#)

# Scaling with Event Hubs

11/26/2021 • 6 minutes to read • [Edit Online](#)

There are two factors which influence scaling with Event Hubs.

- Throughput units (standard tier) or processing units (premium tier)
- Partitions

## Throughput units

The throughput capacity of Event Hubs is controlled by *throughput units*. Throughput units are pre-purchased units of capacity. A single throughput lets you:

- Ingress: Up to 1 MB per second or 1000 events per second (whichever comes first).
- Egress: Up to 2 MB per second or 4096 events per second.

Beyond the capacity of the purchased throughput units, ingress is throttled and a [ServerBusyException](#) is returned. Egress does not produce throttling exceptions, but is still limited to the capacity of the purchased throughput units. If you receive publishing rate exceptions or are expecting to see higher egress, be sure to check how many throughput units you have purchased for the namespace. You can manage throughput units on the **Scale** blade of the namespaces in the [Azure portal](#). You can also manage throughput units programmatically using the [Event Hubs APIs](#).

Throughput units are pre-purchased and are billed per hour. Once purchased, throughput units are billed for a minimum of one hour. Up to 40 throughput units can be purchased for an Event Hubs namespace and are shared across all event hubs in that namespace.

The **Auto-inflate** feature of Event Hubs automatically scales up by increasing the number of throughput units, to meet usage needs. Increasing throughput units prevents throttling scenarios, in which:

- Data ingress rates exceed set throughput units.
- Data egress request rates exceed set throughput units.

The Event Hubs service increases the throughput when load increases beyond the minimum threshold, without any requests failing with ServerBusy errors.

For more information about the auto-inflate feature, see [Automatically scale throughput units](#).

## Processing units

[Event Hubs Premium](#) provides superior performance and better isolation with in a managed multitenant PaaS environment. The resources in a Premium tier are isolated at the CPU and memory level so that each tenant workload runs in isolation. This resource container is called a *Processing Unit*(PU). You can purchase 1, 2, 4, 8 or 16 processing Units for each Event Hubs Premium namespace.

How much you can ingest and stream with a processing unit depends on various factors such as your producers, consumers, the rate at which you're ingesting and processing, and much more. One processing unit can approximately offer core capacity of ~5-10 MB/s ingress and 10-20 MB/s egress, given that we have sufficient partitions so that storage is not a throttling factor.

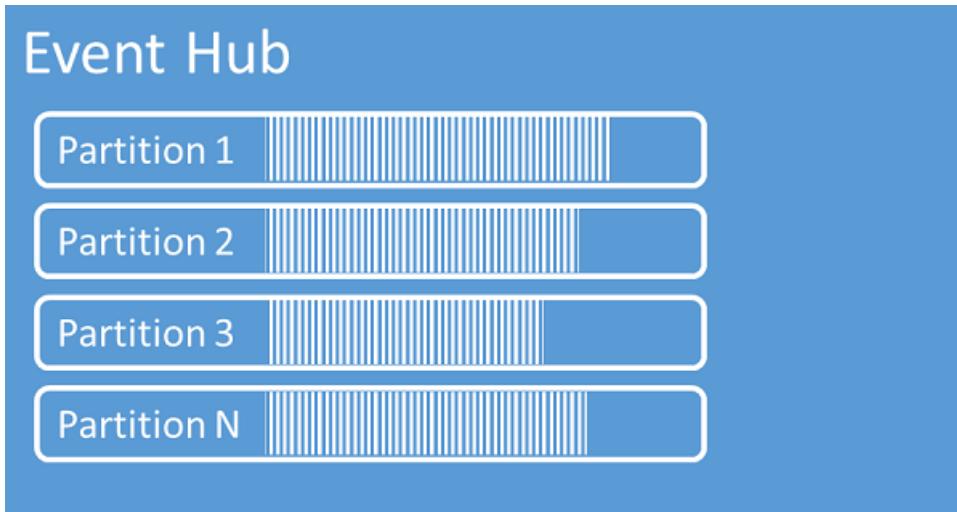
To learn about configuring PUs for a premium tier namespace, see [Configure processing units](#).

#### NOTE

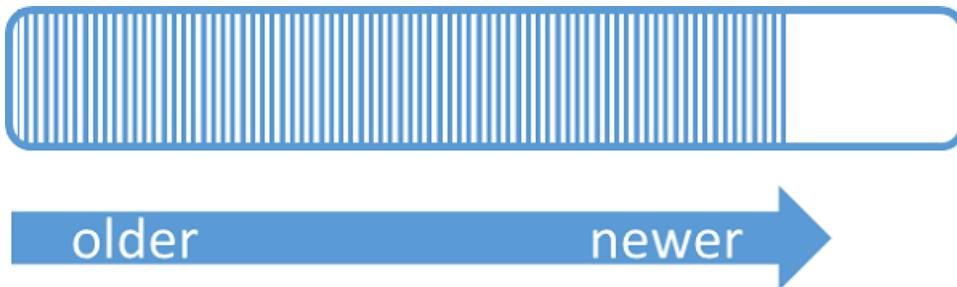
To learn more about quotas and limits, see [Azure Event Hubs - quotas and limits](#).

## Partitions

Event Hubs organizes sequences of events sent to an event hub into one or more partitions. As newer events arrive, they're added to the end of this sequence.



A partition can be thought of as a "commit log". Partitions hold event data that contains body of the event, a user-defined property bag describing the event, metadata such as its offset in the partition, its number in the stream sequence, and service-side timestamp at which it was accepted.



### Advantages of using partitions

Event Hubs is designed to help with processing of large volumes of events, and partitioning helps with that in two ways:

- Even though Event Hubs is a PaaS service, there's a physical reality underneath, and maintaining a log that preserves the order of events requires that these events are being kept together in the underlying storage and its replicas and that results in a throughput ceiling for such a log. Partitioning allows for multiple parallel logs to be used for the same event hub and therefore multiplying the available raw IO throughput capacity.
- Your own applications must be able to keep up with processing the volume of events that are being sent into an event hub. It may be complex and requires substantial, scaled-out, parallel processing capacity. The capacity of a single process to handle events is limited, so you need several processes. Partitions are how your solution feeds those processes and yet ensures that each event has a clear processing owner.

### Number of partitions

The number of partitions is specified at the time of creating an event hub. It must be between 1 and the maximum partition count allowed for each pricing tier. For the partition count limit for each tier, see [this article](#).

We recommend that you choose at least as many partitions as you expect that are required during the peak load

of your application for that particular event hub. You can't change the partition count for an event hub after its creation except for the event hub in a dedicated cluster and premium tier. The partition count for an event hub in a [dedicated Event Hubs cluster](#) can be [increased](#) after the event hub has been created, but the distribution of streams across partitions will change when it's done as the mapping of partition keys to partitions changes, so you should try hard to avoid such changes if the relative order of events matters in your application.

Setting the number of partitions to the maximum permitted value is tempting, but always keep in mind that your event streams need to be structured such that you can indeed take advantage of multiple partitions. If you need absolute order preservation across all events or only a handful of substreams, you may not be able to take advantage of many partitions. Also, many partitions make the processing side more complex.

It doesn't matter how many partitions are in an event hub when it comes to pricing. It depends on the number of pricing units ([throughput units \(TUs\)](#) for the standard tier, [processing units \(PUs\)](#) for the premium tier, and [capacity units \(CUs\)](#) for the dedicated tier) for the namespace or the dedicated cluster. For example, an event hub of the standard tier with 32 partitions or with 1 partition incur the exact same cost when the namespace is set to 1 TU capacity. Also, you can scale TUs or PUs on your namespace or CUs of your dedicated cluster independent of the partition count.

### Mapping of events to partitions

You can use a partition key to map incoming event data into specific partitions for the purpose of data organization. The partition key is a sender-supplied value passed into an event hub. It is processed through a static hashing function, which creates the partition assignment. If you don't specify a partition key when publishing an event, a round-robin assignment is used.

The event publisher is only aware of its partition key, not the partition to which the events are published. This decoupling of key and partition insulates the sender from needing to know too much about the downstream processing. A per-device or user unique identity makes a good partition key, but other attributes such as geography can also be used to group related events into a single partition.

Specifying a partition key enables keeping related events together in the same partition and in the exact order in which they arrived. The partition key is some string that is derived from your application context and identifies the interrelationship of the events. A sequence of events identified by a partition key is a *stream*. A partition is a multiplexed log store for many such streams.

#### NOTE

While you can send events directly to partitions, we don't recommend it, especially when high availability is important to you. It downgrades the availability of an event hub to partition-level. For more information, see [Availability and Consistency](#).

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Automatically scale throughput units for a standard tier namespace](#)
- [Configure processing units for a premium tier namespace](#)

# AMQP 1.0 in Azure Service Bus and Event Hubs protocol guide

11/26/2021 • 28 minutes to read • [Edit Online](#)

The Advanced Message Queueing Protocol 1.0 is a standardized framing and transfer protocol for asynchronously, securely, and reliably transferring messages between two parties. It is the primary protocol of Azure Service Bus Messaging and Azure Event Hubs.

AMQP 1.0 is the result of broad industry collaboration that brought together middleware vendors, such as Microsoft and Red Hat, with many messaging middleware users such as JP Morgan Chase representing the financial services industry. The technical standardization forum for the AMQP protocol and extension specifications is OASIS, and it has achieved formal approval as an international standard as ISO/IEC 19494:2014.

## Goals

This article summarizes the core concepts of the AMQP 1.0 messaging specification along extension specifications developed by the [OASIS AMQP Technical Committee](#) and explains how Azure Service Bus implements and builds on these specifications.

The goal is for any developer using any existing AMQP 1.0 client stack on any platform to be able to interact with Azure Service Bus via AMQP 1.0.

Common general-purpose AMQP 1.0 stacks, such as [Apache Qpid Proton](#) or [AMQP.NET Lite](#), implement all core AMQP 1.0 protocol elements like sessions or links. Those foundational elements are sometimes wrapped with a higher-level API; Apache Proton even offers two, the imperative Messenger API and the reactive Reactor API.

In the following discussion, we assume that the management of AMQP connections, sessions, and links and the handling of frame transfers and flow control are handled by the respective stack (such as Apache Proton-C) and do not require much if any specific attention from application developers. We abstractly assume the existence of a few API primitives like the ability to connect, and to create some form of *sender* and *receiver* abstraction objects, which then have some shape of `send()` and `receive()` operations, respectively.

When discussing advanced capabilities of Azure Service Bus, such as message browsing or management of sessions, those features are explained in AMQP terms, but also as a layered pseudo-implementation on top of this assumed API abstraction.

## What is AMQP?

AMQP is a framing and transfer protocol. Framing means that it provides structure for binary data streams that flow in either direction of a network connection. The structure provides delineation for distinct blocks of data, called *frames*, to be exchanged between the connected parties. The transfer capabilities make sure that both communicating parties can establish a shared understanding about when frames shall be transferred, and when transfers shall be considered complete.

Unlike earlier expired draft versions produced by the AMQP working group that are still in use by a few message brokers, the working group's final, and standardized AMQP 1.0 protocol does not prescribe the presence of a message broker or any particular topology for entities inside a message broker.

The protocol can be used for symmetric peer-to-peer communication, for interaction with message brokers that support queues and publish/subscribe entities, as Azure Service Bus does. It can also be used for interaction with messaging infrastructure where the interaction patterns are different from regular queues, as is the case

with Azure Event Hubs. An Event Hub acts like a queue when events are sent to it, but acts more like a serial storage service when events are read from it; it somewhat resembles a tape drive. The client picks an offset into the available data stream and is then served all events from that offset to the latest available.

The AMQP 1.0 protocol is designed to be extensible, enabling further specifications to enhance its capabilities. The three extension specifications discussed in this document illustrate this. For communication over existing HTTPS/WebSockets infrastructure, configuring the native AMQP TCP ports may be difficult. A binding specification defines how to layer AMQP over WebSockets. For interacting with the messaging infrastructure in a request/response fashion for management purposes or to provide advanced functionality, the AMQP management specification defines the required basic interaction primitives. For federated authorization model integration, the AMQP claims-based-security specification defines how to associate and renew authorization tokens associated with links.

## Basic AMQP scenarios

This section explains the basic usage of AMQP 1.0 with Azure Service Bus, which includes creating connections, sessions, and links, and transferring messages to and from Service Bus entities such as queues, topics, and subscriptions.

The most authoritative source to learn about how AMQP works is the [AMQP 1.0 specification](#), but the specification was written to precisely guide implementation and not to teach the protocol. This section focuses on introducing as much terminology as needed for describing how Service Bus uses AMQP 1.0. For a more comprehensive introduction to AMQP as well as a broader discussion of AMQP 1.0, you can review [this video course](#).

### Connections and sessions

AMQP calls the communicating programs *containers*; those contain *nodes*, which are the communicating entities inside of those containers. A queue can be such a node. AMQP allows for multiplexing, so a single connection can be used for many communication paths between nodes; for example, an application client can concurrently receive from one queue and send to another queue over the same network connection.



The network connection is thus anchored on the container. It is initiated by the container in the client role making an outbound TCP socket connection to a container in the receiver role, which listens for and accepts inbound TCP connections. The connection handshake includes negotiating the protocol version, declaring or negotiating the use of Transport Level Security (TLS/SSL), and an authentication/authorization handshake at the connection scope that is based on SASL.

Azure Service Bus or Azure Event Hubs requires the use of TLS at all times. It supports connections over TCP port 5671, whereby the TCP connection is first overlaid with TLS before entering the AMQP protocol handshake, and also supports connections over TCP port 5672 whereby the server immediately offers a mandatory upgrade of connection to TLS using the AMQP-prescribed model. The AMQP WebSockets binding creates a tunnel over TCP port 443 that is then equivalent to AMQP 5671 connections.

After setting up the connection and TLS, Service Bus offers two SASL mechanism options:

- SASL PLAIN is commonly used for passing username and password credentials to a server. Service Bus does not have accounts, but named [Shared Access Security rules](#), which confer rights and are associated with a key. The name of a rule is used as the user name and the key (as base64 encoded text) is used as the

password. The rights associated with the chosen rule govern the operations allowed on the connection.

- SASL ANONYMOUS is used for bypassing SASL authorization when the client wants to use the claims-based-security (CBS) model that is described later. With this option, a client connection can be established anonymously for a short time during which the client can only interact with the CBS endpoint and the CBS handshake must complete.

After the transport connection is established, the containers each declare the maximum frame size they are willing to handle, and after an idle timeout they'll unilaterally disconnect if there is no activity on the connection.

They also declare how many concurrent channels are supported. A channel is a unidirectional, outbound, virtual transfer path on top of the connection. A session takes a channel from each of the interconnected containers to form a bi-directional communication path.

Sessions have a window-based flow control model; when a session is created, each party declares how many frames it is willing to accept into its receive window. As the parties exchange frames, transferred frames fill that window and transfers stop when the window is full and until the window gets reset or expanded using the *flow performative* (*performative* is the AMQP term for protocol-level gestures exchanged between the two parties).

This window-based model is roughly analogous to the TCP concept of window-based flow control, but at the session level inside the socket. The protocol's concept of allowing for multiple concurrent sessions exists so that high priority traffic could be rushed past throttled normal traffic, like on a highway express lane.

Azure Service Bus currently uses exactly one session for each connection. The Service Bus maximum frame-size is 262,144 bytes (256-K bytes) for Service Bus Standard. It is 1048576 (100 MB) for Service Bus Premium and Event Hubs. Service Bus does not impose any particular session-level throttling windows, but resets the window regularly as part of link-level flow control (see [the next section](#)).

Connections, channels, and sessions are ephemeral. If the underlying connection collapses, connections, TLS tunnel, SASL authorization context, and sessions must be reestablished.

## AMQP outbound port requirements

Clients that use AMQP connections over TCP require ports 5671 and 5672 to be opened in the local firewall.

Along with these ports, it might be necessary to open additional ports if the [EnableLinkRedirect](#) feature is enabled. [EnableLinkRedirect](#) is a new messaging feature that helps skip one-hop while receiving messages, thus helping to boost throughput. The client would start communicating directly with the back-end service over port range 104XX as shown in the following image.

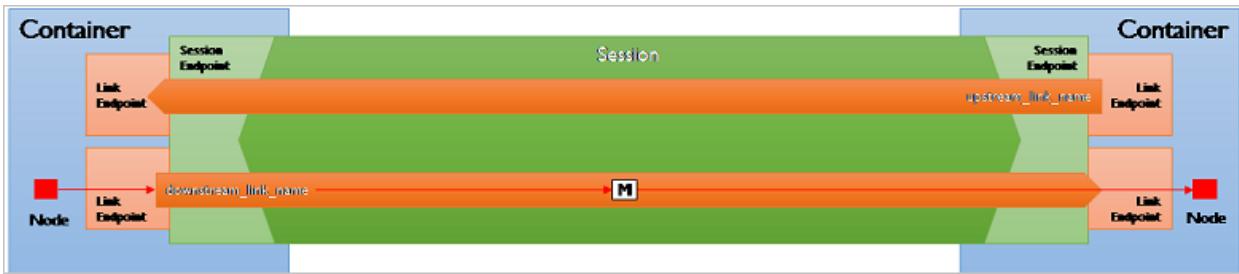
Source	Destination	Source Port	Destination Port	Protocol Name
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS

A .NET client would fail with a `SocketException` ("An attempt was made to access a socket in a way forbidden by its access permissions") if these ports are blocked by the firewall. The feature can be disabled by setting

`EnableAmqpLinkRedirect=false` in the connection string, which forces the clients to communicate with the remote service over port 5671.

## Links

AMQP transfers messages over links. A link is a communication path created over a session that enables transferring messages in one direction; the transfer status negotiation is over the link and bi-directional between the connected parties.



Links can be created by either container at any time and over an existing session, which makes AMQP different from many other protocols, including HTTP and MQTT, where the initiation of transfers and transfer path is an exclusive privilege of the party creating the socket connection.

The link-initiating container asks the opposite container to accept a link and it chooses a role of either sender or receiver. Therefore, either container can initiate creating unidirectional or bi-directional communication paths, with the latter modeled as pairs of links.

Links are named and associated with nodes. As stated in the beginning, nodes are the communicating entities inside a container.

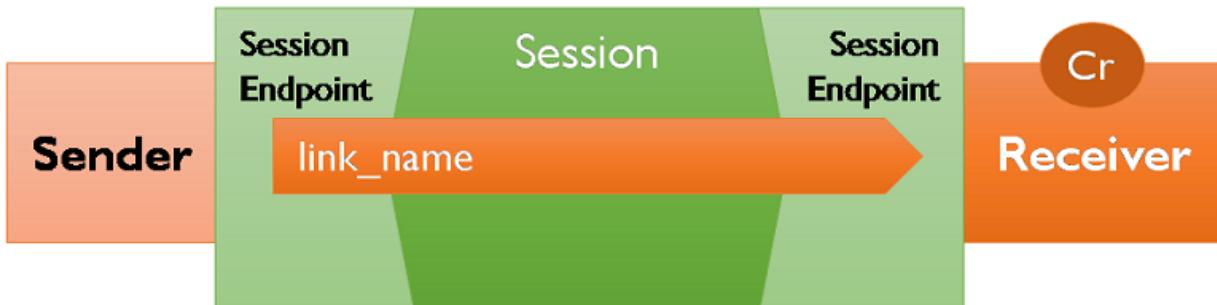
In Service Bus, a node is directly equivalent to a queue, a topic, a subscription, or a deadletter subqueue of a queue or subscription. The node name used in AMQP is therefore the relative name of the entity inside of the Service Bus namespace. If a queue is named `myqueue`, that's also its AMQP node name. A topic subscription follows the HTTP API convention by being sorted into a "subscriptions" resource collection and thus, a subscription `sub` on a topic `mytopic` has the AMQP node name `mytopic/subscriptions/sub`.

The connecting client is also required to use a local node name for creating links; Service Bus is not prescriptive about those node names and does not interpret them. AMQP 1.0 client stacks generally use a scheme to assure that these ephemeral node names are unique in the scope of the client.

## Transfers

Once a link has been established, messages can be transferred over that link. In AMQP, a transfer is executed with an explicit protocol gesture (the *transfer* performative) that moves a message from sender to receiver over a link. A transfer is complete when it is "settled", meaning that both parties have established a shared understanding of the outcome of that transfer.

```
TRANSFER(settled=false,  
         state=null, ...)
```



```
DISPOSITION(settled=true,  
            state=accepted, ...)
```

In the simplest case, the sender can choose to send messages "pre-settled," meaning that the client isn't interested in the outcome and the receiver does not provide any feedback about the outcome of the operation. This mode is supported by Service Bus at the AMQP protocol level, but not exposed in any of the client APIs.

The regular case is that messages are being sent unsettled, and the receiver then indicates acceptance or rejection using the *disposition* performative. Rejection occurs when the receiver cannot accept the message for any reason, and the rejection message contains information about the reason, which is an error structure defined by AMQP. If messages are rejected due to internal errors inside of Service Bus, the service returns extra information inside that structure that can be used for providing diagnostics hints to support personnel if you are filing support requests. You learn more details about errors later.

A special form of rejection is the *released* state, which indicates that the receiver has no technical objection to the transfer, but also no interest in settling the transfer. That case exists, for example, when a message is delivered to a Service Bus client, and the client chooses to "abandon" the message because it cannot perform the work resulting from processing the message; the message delivery itself is not at fault. A variation of that state is the *modified* state, which allows changes to the message as it is released. That state is not used by Service Bus at present.

The AMQP 1.0 specification defines a further disposition state called *received*, that specifically helps to handle link recovery. Link recovery allows reconstituting the state of a link and any pending deliveries on top of a new connection and session, when the prior connection and session were lost.

Service Bus does not support link recovery; if the client loses the connection to Service Bus with an unsettled message transfer pending, that message transfer is lost, and the client must reconnect, reestablish the link, and retry the transfer.

As such, Service Bus and Event Hubs support "at least once" transfer where the sender can be assured for the message having been stored and accepted, but do not support "exactly once" transfers at the AMQP level, where the system would attempt to recover the link and continue to negotiate the delivery state to avoid duplication of the message transfer.

To compensate for possible duplicate sends, Service Bus supports duplicate detection as an optional feature on queues and topics. Duplicate detection records the message IDs of all incoming messages during a user-defined time window, then silently drops all messages sent with the same message-IDs during that same window.

## Flow control

In addition to the session-level flow control model that previously discussed, each link has its own flow control model. Session-level flow control protects the container from having to handle too many frames at once, link-level flow control puts the application in charge of how many messages it wants to handle from a link and when.

Source	Destination	Source Port	Destination Port	Protocol Name
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS
EventHub	Client	10401 (0x28A1)	63006 (0xF61E)	TLS
Client	EventHub	63006 (0xF61E)	10401 (0x28A1)	TLS

On a link, transfers can only happen when the sender has enough *link credit*. Link credit is a counter set by the receiver using the *flow* performative, which is scoped to a link. When the sender is assigned link credit, it attempts to use up that credit by delivering messages. Each message delivery decrements the remaining link credit by 1. When the link credit is used up, deliveries stop.

When Service Bus is in the receiver role, it instantly provides the sender with ample link credit, so that messages can be sent immediately. As link credit is used, Service Bus occasionally sends a *flow* performative to the sender to update the link credit balance.

In the sender role, Service Bus sends messages to use up any outstanding link credit.

A "receive" call at the API level translates into a *flow* performative being sent to Service Bus by the client, and Service Bus consumes that credit by taking the first available, unlocked message from the queue, locking it, and transferring it. If there is no message readily available for delivery, any outstanding credit by any link established with that particular entity remains recorded in order of arrival, and messages are locked and transferred as they become available, to use any outstanding credit.

The lock on a message is released when the transfer is settled into one of the terminal states *accepted*, *rejected*, or *released*. The message is removed from Service Bus when the terminal state is *accepted*. It remains in Service Bus and is delivered to the next receiver when the transfer reaches any of the other states. Service Bus automatically moves the message into the entity's deadletter queue when it reaches the maximum delivery count allowed for the entity due to repeated rejections or releases.

Even though the Service Bus APIs do not directly expose such an option today, a lower-level AMQP protocol client can use the link-credit model to turn the "pull-style" interaction of issuing one unit of credit for each receive request into a "push-style" model by issuing a large number of link credits and then receive messages as they become available without any further interaction. Push is supported through the [MessagingFactory.PrefetchCount](#) or [MessageReceiver.PrefetchCount](#) property settings. When they are non-zero, the AMQP client uses it as the link credit.

In this context, it's important to understand that the clock for the expiration of the lock on the message inside the entity starts when the message is taken from the entity, not when the message is put on the wire. Whenever the client indicates readiness to receive messages by issuing link credit, it is therefore expected to be actively pulling messages across the network and be ready to handle them. Otherwise the message lock may have expired before the message is even delivered. The use of link-credit flow control should directly reflect the

immediate readiness to deal with available messages dispatched to the receiver.

In summary, the following sections provide a schematic overview of the performative flow during different API interactions. Each section describes a different logical operation. Some of those interactions may be "lazy," meaning they may only be performed when required. Creating a message sender may not cause a network interaction until the first message is sent or requested.

The arrows in the following table show the performative flow direction.

#### Create message receiver

CLIENT	SERVICE BUS
--> attach( name={link name}, handle={numeric handle}, role= <b>receiver</b> , source={entity name}, target={client link ID} )	Client attaches to entity as receiver
Service Bus replies attaching its end of the link	<-- attach( name={link name}, handle={numeric handle}, role= <b>sender</b> , source={entity name}, target={client link ID} )

#### Create message sender

CLIENT	SERVICE BUS
--> attach( name={link name}, handle={numeric handle}, role= <b>sender</b> , source={client link ID}, target={entity name} )	No action
No action	<-- attach( name={link name}, handle={numeric handle}, role= <b>receiver</b> , source={client link ID}, target={entity name} )

#### Create message sender (error)

CLIENT	SERVICE BUS
--> attach( name={link name}, handle={numeric handle}, role= <b>sender</b> , source={client link ID}, target={entity name} )	No action

CLIENT	SERVICE BUS
No action	<pre>&lt;-- attach(     name={link name},     handle={numeric handle},     role=receiver,     source=null,     target=null )  &lt;-- detach(     handle={numeric handle},     closed=true,     error={error info} )</pre>

#### Close message receiver/sender

CLIENT	SERVICE BUS
--> detach( handle={numeric handle}, closed=true )	No action
No action	<pre>&lt;-- detach(     handle={numeric handle},     closed=true )</pre>

#### Send (success)

CLIENT	SERVICE BUS
--> transfer( delivery-id={numeric handle}, delivery-tag={binary handle}, settled=false,,more=false, state=null, resume=false )	No action
No action	<pre>&lt;-- disposition(     role=receiver,     first={delivery ID},     last={delivery ID},     settled=true,     state=accepted )</pre>

#### Send (error)

CLIENT	SERVICE BUS
--> transfer( delivery-id={numeric handle}, delivery-tag={binary handle}, settled=false, more=false, state=null, resume=false )	No action
No action	<-- disposition( role=receiver, first={delivery ID}, last={delivery ID}, settled=true, state=rejected( error={error info} ) )

#### Receive

CLIENT	SERVICE BUS
--> flow( link-credit=1 )	No action
No action	< transfer( delivery-id={numeric handle}, delivery-tag={binary handle}, settled=false, more=false, state=null, resume=false )
--> disposition( role=receiver, first={delivery ID}, last={delivery ID}, settled=true, state=accepted )	No action

#### Multi-message receive

CLIENT	SERVICE BUS
--> flow( link-credit=3 )	No action

CLIENT	SERVICE BUS
No action	< transfer( delivery-id={numeric handle}, delivery-tag={binary handle}, settled=false, more=false, state=null, resume=false )
No action	< transfer( delivery-id={numeric handle+1}, delivery-tag={binary handle}, settled=false, more=false, state=null, resume=false )
No action	< transfer( delivery-id={numeric handle+2}, delivery-tag={binary handle}, settled=false, more=false, state=null, resume=false )
--> disposition( role=receiver, first={delivery ID}, last={delivery ID+2}, settled=true, state=accepted )	No action

## Messages

The following sections explain which properties from the standard AMQP message sections are used by Service Bus and how they map to the Service Bus API set.

Any property that application needs to defines should be mapped to AMQP's `application-properties` map.

FIELD NAME	USAGE	API NAME
durable	-	-
priority	-	-
ttl	Time to live for this message	TimeToLive
first-acquirer	-	-
delivery-count	-	DeliveryCount

## properties

FIELD NAME	USAGE	API NAME
message-id	Application-defined, free-form identifier for this message. Used for duplicate detection.	<a href="#">MessageId</a>
user-id	Application-defined user identifier, not interpreted by Service Bus.	Not accessible through the Service Bus API.
to	Application-defined destination identifier, not interpreted by Service Bus.	<a href="#">To</a>
subject	Application-defined message purpose identifier, not interpreted by Service Bus.	<a href="#">Label</a>
reply-to	Application-defined reply-path indicator, not interpreted by Service Bus.	<a href="#">ReplyTo</a>
correlation-id	Application-defined correlation identifier, not interpreted by Service Bus.	<a href="#">CorrelationId</a>
content-type	Application-defined content-type indicator for the body, not interpreted by Service Bus.	<a href="#">ContentType</a>
content-encoding	Application-defined content-encoding indicator for the body, not interpreted by Service Bus.	Not accessible through the Service Bus API.
absolute-expiry-time	Declares at which absolute instant the message expires. Ignored on input (header TTL is observed), authoritative on output.	<a href="#">ExpiresAtUtc</a>
creation-time	Declares at which time the message was created. Not used by Service Bus	Not accessible through the Service Bus API.
group-id	Application-defined identifier for a related set of messages. Used for Service Bus sessions.	<a href="#">SessionId</a>
group-sequence	Counter identifying the relative sequence number of the message inside a session. Ignored by Service Bus.	Not accessible through the Service Bus API.
reply-to-group-id	-	<a href="#">ReplyToSessionId</a>

#### Message annotations

There are few other service bus message properties, which are not part of AMQP message properties, and are passed along as `MessageAnnotations` on the message.

ANNOTATION MAP KEY	USAGE	API NAME
x-opt-scheduled-enqueue-time	Declares at which time the message should appear on the entity	ScheduledEnqueueTime
x-opt-partition-key	Application-defined key that dictates which partition the message should land in.	PartitionKey
x-opt-via-partition-key	Application-defined partition-key value when a transaction is to be used to send messages via a transfer queue.	ViaPartitionKey
x-opt-enqueued-time	Service-defined UTC time representing the actual time of enqueueing the message. Ignored on input.	EnqueuedTimeUtc
x-opt-sequence-number	Service-defined unique number assigned to a message.	SequenceNumber
x-opt-offset	Service-defined enqueued sequence number of the message.	EnqueuedSequenceNumber
x-opt-locked-until	Service-defined. The date and time until which the message will be locked in the queue/subscription.	LockedUntilUtc
x-opt-deadletter-source	Service-Defined. If the message is received from dead letter queue, the source of the original message.	DeadLetterSource

## Transaction capability

A transaction groups two or more operations together into an execution scope. By nature, such a transaction must ensure that all operations belonging to a given group of operations either succeed or fail jointly. The operations are grouped by an identifier `txnid`.

For transactional interaction, the client acts as a `transaction controller`, which controls the operations that should be grouped together. Service Bus Service acts as a `transactional resource` and performs work as requested by the `transaction controller`.

The client and service communicate over a `control link`, which is established by the client. The `declare` and `discharge` messages are sent by the controller over the control link to allocate and complete transactions respectively (they do not represent the demarcation of transactional work). The actual send/receive is not performed on this link. Each transactional operation requested is explicitly identified with the desired `txnid` and therefore may occur on any link on the Connection. If the control link is closed while there exist non-discharged transactions it created, then all such transactions are immediately rolled back, and attempts to perform further transactional work on them will lead to failure. Messages on control link must not be pre-settled.

Every connection has to initiate its own control link to be able to start and end transactions. The service defines a special target that functions as a `coordinator`. The client/controller establishes a control link to this target. Control link is outside the boundary of an entity, that is, same control link can be used to initiate and discharge transactions for multiple entities.

### Starting a transaction

To begin transactional work, the controller must obtain a `txnid` from the coordinator. It does this by sending a

`declare` type message. If the declaration is successful, the coordinator responds with a disposition outcome, which carries the assigned `txn-id`.

CLIENT (CONTROLLER)	DIRECTION	SERVICE BUS (COORDINATOR)
attach( name={link name}, ..., role=sender, target=Coordinator )	----->	
	<-----	attach( name={link name}, ..., target=Coordinator() )
transfer( delivery-id=0, ...) { AmqpValue (Declare())}	----->	
	<-----	disposition( first=0, last=0, state=Declared( txn-id={transaction ID} ))

#### Discharging a transaction

The controller concludes the transactional work by sending a `discharge` message to the coordinator. The controller indicates that it wishes to commit or roll back the transactional work by setting the `fail` flag on the discharge body. If the coordinator is unable to complete the discharge, the message is rejected with this outcome carrying the `transaction-error`.

Note: fail=true refers to Rollback of a transaction, and fail=false refers to Commit.

CLIENT (CONTROLLER)	DIRECTION	SERVICE BUS (COORDINATOR)
transfer( delivery-id=0, ...) { AmqpValue (Declare())}	----->	
	<-----	disposition( first=0, last=0, state=Declared( txn-id={transaction ID} ))
	... Transactional work on other links ...	

CLIENT (CONTROLLER)	DIRECTION	SERVICE BUS (COORDINATOR)
transfer( delivery-id=57, ...) { AmqpValue ( Discharge(txn-id=0, fail=false))}	----->	
	<-----	disposition( first=57, last=57, state=Accepted())

#### Sending a message in a transaction

All transactional work is done with the transactional delivery state `[transactional-state]` that carries the `txn-id`. In the case of sending messages, the `transactional-state` is carried by the message's transfer frame.

CLIENT (CONTROLLER)	DIRECTION	SERVICE BUS (COORDINATOR)
transfer( delivery-id=0, ...) { AmqpValue (Declare())}	----->	
	<-----	disposition( first=0, last=0, state=Declared( txnid={transaction ID} ))
transfer( handle=1, delivery-id=1, state= TransactionalState( txnid=0)) { payload }	----->	
	<-----	disposition( first=1, last=1, state=TransactionalState( txnid=0, outcome=Accepted()))

#### Disposing a message in a transaction

Message disposition includes operations like `Complete` / `Abandon` / `DeadLetter` / `Defer`. To perform these operations within a transaction, pass the `[transactional-state]` with the disposition.

CLIENT (CONTROLLER)	DIRECTION	SERVICE BUS (COORDINATOR)
transfer( delivery-id=0, ...) { AmqpValue (Declare())}	----->	
	<-----	disposition( first=0, last=0, state=Declared( txnid={transaction ID} ))

CLIENT (CONTROLLER)	DIRECTION	SERVICE BUS (COORDINATOR)
	<-----	transfer( handle=2, delivery-id=11, state=null) { payload }
disposition( first=11, last=11, state= <b>TransactionalState</b> ( txnid=0, outcome=Accepted()))	----->	

## Advanced Service Bus capabilities

This section covers advanced capabilities of Azure Service Bus that are based on draft extensions to AMQP, currently being developed in the OASIS Technical Committee for AMQP. Service Bus implements the latest versions of these drafts and adopts changes introduced as those drafts reach standard status.

### NOTE

Service Bus Messaging advanced operations are supported through a request/response pattern. The details of these operations are described in the article [AMQP 1.0 in Service Bus: request-response-based operations](#).

### AMQP management

The AMQP management specification is the first of the draft extensions discussed in this article. This specification defines a set of protocols layered on top of the AMQP protocol that allow management interactions with the messaging infrastructure over AMQP. The specification defines generic operations such as *create*, *read*, *update*, and *delete* for managing entities inside a messaging infrastructure and a set of query operations.

All those gestures require a request/response interaction between the client and the messaging infrastructure, and therefore the specification defines how to model that interaction pattern on top of AMQP: the client connects to the messaging infrastructure, initiates a session, and then creates a pair of links. On one link, the client acts as sender and on the other it acts as receiver, thus creating a pair of links that can act as a bi-directional channel.

LOGICAL OPERATION	CLIENT	SERVICE BUS
Create Request Response Path	--> attach( name={link name}, handle={numeric handle}, role=sender, source=null, target="myentity/\$management" )	No action
Create Request Response Path	No action	<-- attach( name={link name}, handle={numeric handle}, role=receiver, source=null, target="myentity" )

LOGICAL OPERATION	CLIENT	SERVICE BUS
Create Request Response Path	--> attach( name={link name}, handle={numeric handle}, role=receiver, source="myentity/\$management", target="myclient\$id" )	
Create Request Response Path	No action	<-- attach( name={link name}, handle={numeric handle}, role=sender, source="myentity", target="myclient\$id" )

Having that pair of links in place, the request/response implementation is straightforward: a request is a message sent to an entity inside the messaging infrastructure that understands this pattern. In that request message, the *reply-to* field in the *properties* section is set to the *target* identifier for the link onto which to deliver the response. The handling entity processes the request, and then delivers the reply over the link whose *target* identifier matches the indicated *reply-to* identifier.

The pattern obviously requires that the client container and the client-generated identifier for the reply destination are unique across all clients and, for security reasons, also difficult to predict.

The message exchanges used for the management protocol and for all other protocols that use the same pattern happen at the application level; they do not define new AMQP protocol-level gestures. That's intentional, so that applications can take immediate advantage of these extensions with compliant AMQP 1.0 stacks.

Service Bus does not currently implement any of the core features of the management specification, but the request/response pattern defined by the management specification is foundational for the claims-based-security feature and for nearly all of the advanced capabilities discussed in the following sections:

### Claims-based authorization

The AMQP Claims-Based-Authorization (CBS) specification draft builds on the management specification request/response pattern, and describes a generalized model for how to use federated security tokens with AMQP.

The default security model of AMQP discussed in the introduction is based on SASL and integrates with the AMQP connection handshake. Using SASL has the advantage that it provides an extensible model for which a set of mechanisms have been defined from which any protocol that formally leans on SASL can benefit. Among those mechanisms are "PLAIN" for transfer of usernames and passwords, "EXTERNAL" to bind to TLS-level security, "ANONYMOUS" to express the absence of explicit authentication/authorization, and a broad variety of additional mechanisms that allow passing authentication and/or authorization credentials or tokens.

AMQP's SASL integration has two drawbacks:

- All credentials and tokens are scoped to the connection. A messaging infrastructure may want to provide differentiated access control on a per-entity basis; for example, allowing the bearer of a token to send to queue A but not to queue B. With the authorization context anchored on the connection, it's not possible to use a single connection and yet use different access tokens for queue A and queue B.
- Access tokens are typically only valid for a limited time. This validity requires the user to periodically reacquire tokens and provides an opportunity to the token issuer to refuse issuing a fresh token if the user's access permissions have changed. AMQP connections may last for long periods of time. The SASL model only provides a chance to set a token at connection time, which means that the messaging infrastructure

either has to disconnect the client when the token expires or it needs to accept the risk of allowing continued communication with a client who's access rights may have been revoked in the interim.

The AMQP CBS specification, implemented by Service Bus, enables an elegant workaround for both of those issues: It allows a client to associate access tokens with each node, and to update those tokens before they expire, without interrupting the message flow.

CBS defines a virtual management node, named `$cbs`, to be provided by the messaging infrastructure. The management node accepts tokens on behalf of any other nodes in the messaging infrastructure.

The protocol gesture is a request/reply exchange as defined by the management specification. That means the client establishes a pair of links with the `$cbs` node and then passes a request on the outbound link, and then waits for the response on the inbound link.

The request message has the following application properties:

KEY	OPTIONAL	VALUE TYPE	VALUE CONTENTS
operation	No	string	<code>put-token</code>
type	No	string	The type of the token being put.
name	No	string	The "audience" to which the token applies.
expiration	Yes	timestamp	The expiry time of the token.

The `name` property identifies the entity with which the token shall be associated. In Service Bus it's the path to the queue, or topic/subscription. The `type` property identifies the token type:

TOKEN TYPE	TOKEN DESCRIPTION	BODY TYPE	NOTES
jwt	JSON Web Token (JWT)	AMQP Value (string)	
servicebus.windows.net:sastoken	Service Bus SAS Token	AMQP Value (string)	-

Tokens confer rights. Service Bus knows about three fundamental rights: "Send" enables sending, "Listen" enables receiving, and "Manage" enables manipulating entities. Service Bus SAS tokens refer to rules configured on the namespace or entity, and those rules are configured with rights. Signing the token with the key associated with that rule thus makes the token express the respective rights. The token associated with an entity using `put-token` permits the connected client to interact with the entity per the token rights. A link where the client takes on the `sender` role requires the "Send" right; taking on the `receiver` role requires the "Listen" right.

The reply message has the following *application-properties* values

KEY	OPTIONAL	VALUE TYPE	VALUE CONTENTS
status-code	No	int	HTTP response code [RFC2616].
status-description	Yes	string	Description of the status.

The client can call `put-token` repeatedly and for any entity in the messaging infrastructure. The tokens are

scoped to the current client and anchored on the current connection, meaning the server drops any retained tokens when the connection drops.

The current Service Bus implementation only allows CBS in conjunction with the SASL method "ANONYMOUS." A SSL/TLS connection must always exist prior to the SASL handshake.

The ANONYMOUS mechanism must therefore be supported by the chosen AMQP 1.0 client. Anonymous access means that the initial connection handshake, including creating of the initial session happens without Service Bus knowing who is creating the connection.

Once the connection and session is established, attaching the links to the `$cbs` node and sending the *put-token* request are the only permitted operations. A valid token must be set successfully using a *put-token* request for some entity node within 20 seconds after the connection has been established, otherwise the connection is unilaterally dropped by Service Bus.

The client is subsequently responsible for keeping track of token expiration. When a token expires, Service Bus promptly drops all links on the connection to the respective entity. To prevent problem occurring, the client can replace the token for the node with a new one at any time through the virtual `$cbs` management node with the same *put-token* gesture, and without getting in the way of the payload traffic that flows on different links.

### Send-via functionality

**Send-via / Transfer sender** is a functionality that lets service bus forward a given message to a destination entity through another entity. This feature is used to perform operations across entities in a single transaction.

With this functionality, you create a sender and establish the link to the `via-entity`. While establishing the link, additional information is passed to establish the true destination of the messages/transfers on this link. Once the attach has been successful, all the messages sent on this link are automatically forwarded to the *destination-entity* through *via-entity*.

Note: Authentication has to be performed for both *via-entity* and *destination-entity* before establishing this link.

CLIENT	DIRECTION	SERVICE BUS
attach( name={link name}, role=sender, source={client link ID}, target={via-entity}, properties=map [( com.microsoft:transfer- destination-address= {destination-entity} )] )	----->	
	<-----	attach( name={link name}, role=receiver, source={client link ID}, target={via-entity}, properties=map [( com.microsoft:transfer-destination- address= {destination-entity} )] )

## Next steps

To learn more about AMQP, see [Service Bus AMQP overview](#).

# AMQP errors in Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

This article provides some of the errors you receive when using AMQP with Azure Event Hubs. They are all standard behaviors of the service. You can avoid them by making send/receive calls on the connection/link, which automatically recreates the connection/link.

## Link is closed

You see the following error when the AMQP connection and link are active but no calls (for example, send or receive) are made using the link for 30 minutes. So, the link is closed. The connection is still open.

```
" AMQP:link: detach-forced: The link 'G2:7223832:user.tenant0.cud_00000000000-0000-0000-0000-00000000000' is force detached by the broker because of errors occurred in publisher(link164614). Detach origin: AmqpMessagePublisher.IdleTimerExpired: Idle timeout: 00:30:00.  
TrackingId:00000000000000000000000000000000_G2_B3,  
SystemTracker:mynamespace:Topic:MyTopic, Timestamp: 2/16/2018 11:10:40 PM "
```

## Connection is closed

You see the following error on the AMQP connection when all links in the connection have been closed because there was no activity (idle) and a new link hasn't been created in 5 minutes.

```
" Error{condition=amqp:connection:forced, description='The connection was inactive for more than the allowed 300000 milliseconds and is closed by container 'LinkTracker'.  
TrackingId:00000000000000000000000000000000_G21, SystemTracker:gateway5, Timestamp:2019-03-06T17:32:00', info=null} "
```

## Link isn't created

You see this error when a new AMQP connection is created but a link isn't created within 1 minute of the creation of the AMQP Connection.

```
" Error{condition=amqp:connection:forced, description='The connection was inactive for more than the allowed 60000 milliseconds and is closed by container 'LinkTracker'.  
TrackingId:00000000000000000000000000000000_G21, SystemTracker:gateway5, Timestamp:2019-03-06T18:41:51', info=null} "
```

## Next steps

To learn more about AMQP, see [AMQP 1.0 protocol guide](#).

# Get an Event Hubs connection string

11/26/2021 • 2 minutes to read • [Edit Online](#)

To use Event Hubs, you need to create an Event Hubs namespace. A namespace is a scoping container for multiple event hubs or Kafka topics. This namespace gives you a unique **FQDN**. Once a namespace is created, you can obtain the connection string required to communicate with Event Hubs.

The connection string for Azure Event Hubs has the following components embedded within it,

- FQDN = the FQDN of the EventHubs namespace you created (it includes the EventHubs namespace name followed by servicebus.windows.net)
- SharedAccessKeyName = the name you chose for your application's SAS keys
- SharedAccessKey = the generated value of the key.

The connection string template looks like

```
Endpoint=sb://<FQDN>;SharedAccessKeyName=<KeyName>;SharedAccessKey=<KeyValue>
```

An example connection string might look like

```
Endpoint=sb://dummynamespace.servicebus.windows.net;SharedAccessKeyName=DummyAccessKeyName;SharedAccessKey=5d0ntTRytoC24opYThisAsit3is2E
```

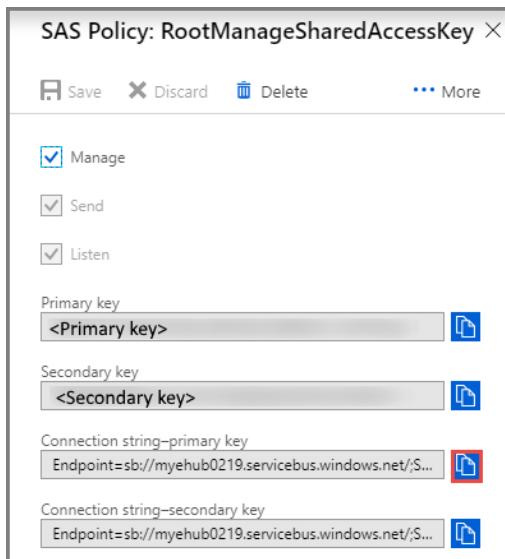
This article walks you through various ways of obtaining the connection string.

## Get connection string from the portal

1. Sign in to [Azure portal](#).
2. Select **All services** on the left navigational menu.
3. Select **Event Hubs** in the **Analytics** section.
4. In the list of event hubs, select your event hub.
5. On the **Event Hubs Namespace** page, select **Shared Access Policies** on the left menu.
6. Select a **shared access policy** in the list of policies. The default one is named: **RootManageSharedAccessPolicy**. You can add a policy with appropriate permissions (read, write), and use that policy.

The screenshot shows the Azure portal interface for managing an Event Hubs namespace. The top navigation bar includes 'Dashboard', 'myehubrg', 'DocsDemoNamespaceEHub', and a search bar. The main content area is titled 'DocsDemoNamespaceEHub | Shared access policies'. On the left, a sidebar lists various management options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings, and Shared access policies (which is highlighted with a red box). The main pane displays a table for shared access policies. It has columns for 'Policy' and 'Claims'. One row is shown, labeled 'RootManageSharedAccessKey', with 'Claims' listed as 'Manage, Send, Listen'. A red box highlights this row. A search bar at the top of the main pane also has a red box around it.

7. Select the **copy** button next to the **Connection string-primary key** field.



## Getting the connection string with Azure PowerShell

### NOTE

This article uses the Azure Az PowerShell module, which is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

You can use the [Get-AzEventHubKey](#) to get the connection string for the specific policy/rule name as shown below:

```
Get-AzEventHubKey -ResourceGroupName MYRESOURCEGROUP -NamespaceName MYEHUBNAMESPACE -AuthorizationRuleName RootManageSharedAccessKey
```

## Getting the connection string with Azure CLI

You can use the following to get the connection string for the namespace:

```
az eventhubs namespace authorization-rule keys list --resource-group MYRESOURCEGROUP --namespace-name MYEHUBNAMESPACE --name RootManageSharedAccessKey
```

Or you can use the following to get the connection string for an EventHub entity:

```
az eventhubs eventhub authorization-rule keys list --resource-group MYRESOURCEGROUP --namespace-name MYEHUBNAMESPACE --eventhub-name MYEHUB --name RootManageSharedAccessKey
```

For more information about Azure CLI commands for Event Hubs, see [Azure CLI for Event Hubs](#).

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)
- [Create an Event Hub](#)

# Configure processing units for a premium tier Azure Event Hubs namespace

11/26/2021 • 2 minutes to read • [Edit Online](#)

This article provides instructions for configuring processing units (PUs) for a premium tier Azure Event Hubs namespace. To learn more about the **premium** tier, see [Event Hubs Premium](#).

## Configure processing units when creating a namespace

When creating a **premium** tier namespace in the Azure portal, the auto-inflate feature is automatically enabled. You can configure the number of processing units (PUs) for your namespace (1, 2, 4, 8, or 16), which you can update after creating the namespace.

The screenshot shows the 'Create Namespace' wizard in the Azure portal. The 'Basics' tab is selected. In the 'PROJECT DETAILS' section, the subscription is set to 'Visual Studio Ultimate with MSDN' and the resource group is 'spehubrg'. In the 'INSTANCE DETAILS' section, the namespace name is 'sehubpremns', located in 'East US' with 'Premium' pricing tier and 1 processing unit. A note indicates that the region supports Availability zones. At the bottom, there are navigation links for 'Review + create', '< Previous', and 'Next: Tags >'.

[Review + create](#)

[< Previous](#)

[Next: Tags >](#)

# Configure processing units for an existing namespace

To update the number of PUs for an existing premium namespace, follow these steps:

1. On the **Event Hubs Namespace** page for your namespace, select **Scale** under **Settings** on the left menu.
2. Update the value for **processing units**, and select **Save**.

The screenshot shows the Azure portal interface for managing an Event Hubs namespace named "sehubpremns". The left sidebar lists various management options: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings (with Shared access policies and Scale selected), and Geo-Recovery. The right pane is titled "PROCESSING UNITS" and contains a descriptive text about controlling capacity using processing units, followed by a horizontal slider with a value of 1.

## Next steps

To learn more about processing units and Event Hubs premium tier, see the following articles.

- [Event Hubs Premium](#)
- [Event Hubs scalability](#)

# Add custom data to events in Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

Because an event consists mainly of an opaque set of bytes, it may be difficult for consumers of those events to make informed decisions about how to process them. To allow event publishers to offer better context for consumers, events may also contain custom metadata, in the form of a set of key-value pairs. One common scenario for the inclusion of metadata is to provide a hint about the type of data contained by an event, so that consumers understand its format and can deserialize it appropriately.

## NOTE

This metadata is not used by, or in any way meaningful to, the Event Hubs service; it exists only for coordination between event publishers and consumers.

The following sections show you how to add custom data to events in different programming languages.

## .NET

```
var eventBody = new BinaryData("Hello, Event Hubs!");
var eventData = new EventData(eventBody);
eventData.Properties.Add("EventType", "com.microsoft.samples.hello-event");
eventData.Properties.Add("priority", 1);
eventData.Properties.Add("score", 9.0);
```

For the full code sample, see [Publishing events with custom metadata](#).

## Java

```
EventData firstEvent = new EventData("EventData Sample 1".getBytes(UTF_8));
firstEvent.getProperties().put("EventType", "com.microsoft.samples.hello-event");
firstEvent.getProperties().put("priority", 1);
firstEvent.getProperties().put("score", 9.0);
```

For the full code sample, see [Publish events with custom metadata](#).

## Python

```
event_data = EventData('Message with properties')
event_data.properties = {'event-type': 'com.microsoft.samples.hello-event', 'priority': 1, "score": 9.0}
```

For the full code sample, see [Send Event Data batch with properties](#).

## JavaScript

```
let eventData = { body: "First event", properties: { "event-type": "com.microsoft.samples.hello-event",
"priority": 1, "score": 9.0 } };
```

## Next steps

See the following quickstarts and samples.

- Quickstarts: [.NET](#), [Java](#), [Python](#), [JavaScript](#)
- Samples on GitHub: [.NET](#), [Java](#), [Python](#), [JavaScript](#), [TypeScript](#)

# Dynamically add partitions to an event hub (Apache Kafka topic) in Azure Event Hubs

11/26/2021 • 5 minutes to read • [Edit Online](#)

Event Hubs provides message streaming through a partitioned consumer pattern in which each consumer only reads a specific subset, or partition, of the message stream. This pattern enables horizontal scale for event processing and provides other stream-focused features that are unavailable in queues and topics. A partition is an ordered sequence of events that is held in an event hub. As newer events arrive, they're added to the end of this sequence. For more information about partitions in general, see [Partitions](#)

You can specify the number of partitions at the time of creating an event hub. In some scenarios, you may need to add partitions after the event hub has been created. This article describes how to dynamically add partitions to an existing event hub.

## IMPORTANT

Dynamic additions of partitions is available only in **premium** and **dedicated** tiers of Event Hubs.

## NOTE

For Apache Kafka clients, an **event hub** maps to a **Kafka topic**. For more mappings between Azure Event Hubs and Apache Kafka, see [Kafka and Event Hubs conceptual mapping](#)

## Update the partition count

This section shows you how to update partition count of an event hub in different ways (PowerShell, CLI, and so on.).

### PowerShell

Use the [Set-AzureRmEventHub](#) PowerShell command to update partitions in an event hub.

```
Set-AzureRmEventHub -ResourceGroupName MyResourceGroupName -Namespace MyNamespaceName -Name MyEventHubName -partitionCount 12
```

### CLI

Use the [az eventhubs eventhub update](#) CLI command to update partitions in an event hub.

```
az eventhubs eventhub update --resource-group MyResourceGroupName --namespace-name MyNamespaceName --name MyEventHubName --partition-count 12
```

### Resource Manager template

Update value of the `partitionCount` property in the Resource Manager template and redeploy the template to update the resource.

```
{
    "apiVersion": "2017-04-01",
    "type": "Microsoft.EventHub/namespaces/eventhubs",
    "name": "[concat(parameters('namespaceName'), '/', parameters('eventHubName'))]",
    "location": "[parameters('location')]",
    "dependsOn": [
        "[resourceId('Microsoft.EventHub/namespaces', parameters('namespaceName'))]"
    ],
    "properties": {
        "messageRetentionInDays": 7,
        "partitionCount": 12
    }
}
```

## Apache Kafka

Use the [AlterTopics](#) API (for example, via [kafka-topics CLI tool](#)) to increase the partition count. For details, see [Modifying Kafka topics](#).

## Event Hubs clients

Let's look at how Event Hubs clients behave when the partition count is updated on an event hub.

When you add a partition to an existing even hub, the event hub client receives a [MessagingException](#) from the service informing the clients that entity metadata (entity is your event hub and metadata is the partition information) has been altered. The clients will automatically reopen the AMQP links, which would then pick up the changed metadata information. The clients then operate normally.

### Sender/producer clients

Event Hubs provides three sender options:

- **Partition sender** – In this scenario, clients send events directly to a partition. Although partitions are identifiable and events can be sent directly to them, we don't recommend this pattern. Adding partitions doesn't impact this scenario. We recommend that you restart applications so that they can detect newly added partitions.
- **Partition key sender** – in this scenario, clients sends the events with a key so that all events belonging to that key end up in the same partition. In this case, service hashes the key and routes to the corresponding partition. The partition count update can cause out-of-order issues because of hashing change. So, if you care about ordering, ensure that your application consumes all events from existing partitions before you increase the partition count.
- **Round-robin sender (default)** – In this scenario, the Event Hubs service round robins the events across partitions, and also uses a load-balancing algorithm. Event Hubs service is aware of partition count changes and will send to new partitions within seconds of altering partition count.

### Receiver/consumer clients

Event Hubs provides direct receivers and an easy consumer library called the [Event Processor Host \(old SDK\)](#) or [Event Processor \(new SDK\)](#).

- **Direct receivers** – The direct receivers listen to specific partitions. Their runtime behavior isn't affected when partitions are scaled out for an event hub. The application that uses direct receivers needs to take care of picking up the new partitions and assigning the receivers accordingly.
- **Event processor host** – This client doesn't automatically refresh the entity metadata. So, it wouldn't pick up on partition count increase. Recreating an event processor instance will cause an entity metadata fetch, which in turn will create new blobs for the newly added partitions. Pre-existing blobs won't be affected. Restarting all event processor instances is recommended to ensure that all instances are aware of the newly added partitions, and load-balancing is handled correctly among consumers.

If you're using the old version of .NET SDK ([WindowsAzure.ServiceBus](#)), the event processor host removes an existing checkpoint upon restart if partition count in the checkpoint doesn't match the partition count fetched from the service. This behavior may have an impact on your application.

## Apache Kafka clients

This section describes how Apache Kafka clients that use the Kafka endpoint of Azure Event Hubs behave when the partition count is updated for an event hub.

Kafka clients that use Event Hubs with the Apache Kafka protocol behave differently from event hub clients that use AMQP protocol. Kafka clients update their metadata once every `metadata.max.age.ms` milliseconds. You specify this value in the client configurations. The `librdkafka` libraries also use the same configuration. Metadata updates inform the clients of service changes including the partition count increases. For a list of configurations, see [Apache Kafka configurations for Event Hubs](#).

### Sender/producer clients

Producers always dictate that send requests contain the partition destination for each set of produced records. So, all produce partitioning is done on client-side with producer's view of broker's metadata. Once the new partitions are added to the producer's metadata view, they'll be available for producer requests.

### Consumer/receiver clients

When a consumer group member performs a metadata refresh and picks up the newly created partitions, that member initiates a group rebalance. Consumer metadata then will be refreshed for all group members, and the new partitions will be assigned by the allotted rebalance leader.

## Recommendations

- If you use partition key with your producer applications and depend on key hashing to ensure ordering in a partition, dynamically adding partitions isn't recommended.

#### IMPORTANT

While the existing data preserves ordering, partition hashing will be broken for messages hashed after the partition count changes due to addition of partitions.

- Adding partition to an existing topic or event hub instance is recommended in the following cases:
  - When you use the default method of sending events
  - Kafka default partitioning strategies, example – Sticky Assignor strategy

## Next steps

For more information about partitions, see [Partitions](#).

# Use Blob Storage as checkpoint store - Event Hubs on Azure Stack Hub

11/26/2021 • 2 minutes to read • [Edit Online](#)

If you're using Azure Blob Storage as the checkpoint store in an environment that supports a different version of Storage Blob SDK than the ones that are typically available on Azure, you'll need to use code to change the Storage service API version to the specific version supported by that environment. For example, if you're running [Event Hubs on an Azure Stack Hub version 2002](#), the highest available version for the Storage service is version 2017-11-09. In this case, you need to use code to target the Storage service API version to 2017-11-09. For an example on how to target a specific Storage API version, see these samples on GitHub:

- [.NET](#)
- [Java](#).
- [JavaScript or TypeScript](#)
- Python - [Synchronous, Asynchronous](#)

If you run Event Hubs receiver that uses Blob Storage as the checkpoint store without targeting the version that Azure Stack Hub supports, you'll receive the following error message:

```
The value for one of the HTTP headers is not in the correct format
```

## Sample error message in Python

For Python, an error of `azure.core.exceptions.HttpResponseError` is passed to the error handler `on_error(partition_context, error)` of `EventHubConsumerClient.receive()`. But, the method `receive()` doesn't raise an exception. `print(error)` will print the following exception information:

```
The value for one of the HTTP headers is not in the correct format.

RequestId:f048aee8-a90c-08ba-4ce1-e69dba759297
Time:2020-03-17T22:04:13.3559296Z
ErrorCode:InvalidHeaderValue
Error:None
HeaderName:x-ms-version
HeaderValue:2019-07-07
```

The logger will log two warnings like the following ones:

```
WARNING:azure.eventhub.extensions.checkpointstoreblobaio._blobstoragecsaio:  
An exception occurred during list_ownership for namespace '<namespace-name>.eventhub.  
<region>.azurestack.corp.microsoft.com' eventhub 'python-eh-test' consumer group '$Default'.  
  
Exception is HttpResponseError('The value for one of the HTTP headers is not in the correct  
format.\nRequestId:f048aee8-a90c-08ba-4ce1-e69dba759297\nTime:2020-03-  
17T22:04:13.3559296Z\nErrorCode:InvalidHeaderValue\nError:None\nHeaderName:x-ms-version\nHeaderValue:2019-  
07-07')  
  
WARNING:azure.eventhub.aio._eventprocessor.event_processor:EventProcessor instance '26d84102-45b2-48a9-b7f4-  
da8916f68214' of eventhub 'python-eh-test' consumer group '$Default'. An error occurred while load-balancing  
and claiming ownership.  
  
The exception is HttpResponseError('The value for one of the HTTP headers is not in the correct  
format.\nRequestId:f048aee8-a90c-08ba-4ce1-e69dba759297\nTime:2020-03-  
17T22:04:13.3559296Z\nErrorCode:InvalidHeaderValue\nError:None\nHeaderName:x-ms-version\nHeaderValue:2019-  
07-07'). Retrying after 71.45254944090853 seconds
```

## Next steps

See the following article learn about partitioning and checkpointing: [Balance partition load across multiple instances of your application](#)

# Deprecation of Azure Service Manager support for Azure Service Bus, Relay, and Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

Resource Manager, our next-generation cloud infrastructure stack, is fully replacing the "classic" Azure Service Management model (classic deployment model). As a result, classic deployment model REST APIs for Service Bus, Relay, and Event Hubs will be retired on December 1, 2021. This deprecation was first announced on a [Microsoft Tech Community announcement](#). For easy identification, these APIs have `management.core.windows.net` in their URI. Refer to the following table for a list of the deprecated APIs and their Azure Resource Manager API version that you should now use.

To continue using Service Bus, Relay, and Event Hubs, move to Resource Manager by November 30, 2021. We encourage all customers who are still using old APIs to make the switch soon to take advantage of the additional benefits of Resource Manager, which include resource grouping, tags, a streamlined deployment and management process, and fine-grained access control using Azure role-based access control (Azure RBAC).

For more information on Azure Resource Manager vs Azure Service Manager, see the [TechNet Blog](#).

For more information on Service Manager and Resource Manager APIs for Azure Service Bus, Relay and Event Hubs, see our REST API documentation:

- [Azure Service Bus](#)
- [Azure Event Hubs](#)
- [Azure Relay](#)

## Service Manager REST API - Resource Manager REST API

SERVICE MANAGER APIS (DEPRECATED)	RESOURCE MANAGER - SERVICE BUS API	RESOURCE MANAGER - EVENT HUB API	RESOURCE MANAGER - RELAY API
<b>Namespaces-</b> <a href="#">GetNamespaceAsync</a> <a href="#">Service Bus Get Namespace</a> <a href="#">Event Hub Get Namespace</a> <a href="#">Relay Get Namespace</a>  <code>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespaces/{namespace name}</code>	<code>get</code>	<code>get</code>	<code>get</code>
<b>ConnectionDetails-</b> <a href="#">GetConnectionDetails</a> Service Bus/Event Hub/Relay <a href="#">GetConnectionDetails</a>  <code>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespaces/{namespace name}/ConnectionDetails</code>	<code>listkeys</code>	<code>listkeys</code>	<code>listkeys</code>

Service Manager APIs (Deprecated)	Resource Manager - Service Bus API	Resource Manager - Event Hub API	Resource Manager - Relay API
Topics-GetTopicsAsync Service Bus	list		
	<pre>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}/topics? \$skip={skip}&amp;\$top={top}</pre>		
Queues-GetQueueAsync Service Bus	get		
	<pre>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}/queues/{queueName}</pre>		
Relays-GetRelaysAsync Get Relays			list
	<pre>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}/relays? \$skip={skip}&amp;\$top={top}</pre>		
NamespaceAuthorizationRules- GetNamespaceAuthorizationRuleAsync Service Bus/Event Hub/Relay GetNamespaceAuthRule	getauthorizationrule	getauthorizationrule	getauthorizationrule
	<pre>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}/authorizationrules?</pre>		
Namespaces- DeleteNamespaceAsync Service Bus Delete Namespace Event Hubs Delete Namespace Relays Delete Namespace	delete	delete	delete
	<pre>DELETE https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}</pre>		
MessagingSKUPlan- GetPlanAsync Service Bus/Event Hub/Relay Get Namespace	get	get	get
	<pre>GET https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}/MessagingPlan</pre>		
MessagingSKUPlan- UpdatePlanAsync Service Bus/Event Hub/Relay Get Namespace	createorupdate	createorupdate	createorupdate
	<pre>PUT https://management.core.windows.net/{subscription ID}/services/ServiceBus/Namespace/{namespace name}/MessagingPlan</pre>		

SERVICE MANAGER APIs (DEPRECATED)	RESOURCE MANAGER - SERVICE BUS API	RESOURCE MANAGER - EVENT HUB API	RESOURCE MANAGER - RELAY API
<b>NamespaceAuthorizationRules-</b> <b>UpdateNamespaceAuthorizationRuleAsync</b> Service Bus/Event Hub/Relay Get Namespace	<a href="#">createorupdate</a>	<a href="#">createorupdateauthorizationrule</a>	<a href="#">createorupdateauthorizationrule</a>
<b>NamespaceAuthorizationRules-</b> <b>CreateNamespaceAuthorizationRuleAsync</b>			
Service Bus/Event Hub/Relay	<a href="#">createorupdate</a>	<a href="#">createorupdateauthorizationrule</a>	<a href="#">createorupdateauthorizationrule</a>
<b>NamespaceProperties-</b> <b>GetNamespacePropertiesAsync</b> Service Bus Get Namespace Event Hubs Get Namespace Relay Get Namespace	<a href="#">get</a>	<a href="#">get</a>	<a href="#">get</a>
<b>RegionCodes-</b> <b>GetRegionCodesAsync</b> Service Bus/EventHub/Relay Get Namespace	<a href="#">listbysku</a>	<a href="#">listbysku</a>	
<b>NamespaceProperties-</b> <b>UpdateNamespacePropertyAsync</b> Service Bus/EventHub/Relay	<a href="#">createorupdate</a>	<a href="#">createorupdate</a>	<a href="#">createorupdate</a>
<b>EventHubsCrud-</b> <b>ListEventHubsAsync</b> <b>List Event Hubs</b>		<a href="#">list</a>	

SERVICE MANAGER APIs (DEPRECATED)	RESOURCE MANAGER - SERVICE BUS API	RESOURCE MANAGER - EVENT HUB API	RESOURCE MANAGER - RELAY API
<b>EventHubsCrud-</b> <b>GetEventHubAsync</b> <a href="#">Get Event Hubs</a>		<a href="#">get</a>	
<b>NamespaceAuthorizationRules-</b> <b>DeleteNamespaceAuthorizationRuleAsync</b> Service Bus/Event Hub/Relay	<a href="#">deleteauthorizationrule</a>	<a href="#">deleteauthorizationrule</a>	<a href="#">deleteauthorizationrule</a>
	<b>DELETE</b> <a href="https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}/AuthorizationRules/{rule name}">https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}/AuthorizationRules/{rule name}</a>		
<b>NamespaceAuthorizationRules-</b> <b>GetNamespaceAuthorizationRulesAsync</b> Service Bus/EventHub/Relay	<a href="#">listauthorizationrules</a>	<a href="#">listauthorizationrules</a>	<a href="#">listauthorizationrules</a>
	<b>GET</b> <a href="https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}/AuthorizationRules">https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}/AuthorizationRules</a>		
<b>NamespaceAvailability-</b> <b>IsNamespaceAvailable</b> Service Bus Namespace Availability	<a href="#">checknameavailability</a>	<a href="#">checknameavailability</a>	<a href="#">checknameavailability</a>
	<b>GET</b> <a href="https://management.core.windows.net/{subscriptionID}/services/ServiceBus/CheckNamespaceAvailability/?namespace=&lt;namespaceValue&gt;">https://management.core.windows.net/{subscriptionID}/services/ServiceBus/CheckNamespaceAvailability/?namespace=&lt;namespaceValue&gt;</a>		
<b>Namespaces-</b> <b>CreateOrUpdateNamespaceAsync</b> Service Bus/Event Hub/Relay	<a href="#">createorupdate</a>	<a href="#">createorupdate</a>	<a href="#">createorupdate</a>
	<b>PUT</b> <a href="https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}">https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}</a>		
<b>Topics-GetTopicAsync</b>	<a href="#">get</a>		
	<b>GET</b> <a href="https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}/topics/{topicPath}">https://management.core.windows.net/{subscriptionID}/services/ServiceBus/Namespaces/{namespace name}/topics/{topicPath}</a>		

## Service Manager PowerShell - Resource Manager PowerShell

SERVICE MANAGER POWERSHELL COMMAND (DEPRECATED)	NEW RESOURCE MANAGER COMMANDS	NEWER RESOURCE MANAGER COMMAND
--	----------------------------------	-----------------------------------

SERVICE MANAGER POWERSHELL COMMAND (DEPRECATED)	NEW RESOURCE MANAGER COMMANDS	NEWER RESOURCE MANAGER COMMAND
Get-AzureSBAuthorizationRule	Get-AzureRmServiceBusAuthorizationRule	Get-AzServiceBusAuthorizationRule
Get-AzureSBLocation	Get-AzureRmServiceBusGeoDRConfiguration	Get-AzServiceBusGeoDRConfiguration
Get-AzureSBNamespace	Get-AzureRmServiceBusNamespace	Get-AzServiceBusNamespace
New-AzureSBAuthorizationRule	New-AzureRmServiceBusAuthorizationRule	New-AzServiceBusAuthorizationRule
New-AzureSBNamespace	New-AzureRmServiceBusNamespace	New-AzServiceBusNamespace
Remove-AzureSBAuthorizationRule	Remove-AzureRmServiceBusAuthorizationRule	Remove-AzServiceBusAuthorizationRule
Remove-AzureSBNamespace	Remove-AzureRmServiceBusNamespace	Remove-AzServiceBusNamespace
Set-AzureSBAuthorizationRule	Set-AzureRmServiceBusAuthorizationRule	Set-AzServiceBusAuthorizationRule

## Next steps

See the following documentation:

- Latest REST API documentation
  - [Azure Service Bus](#)
  - [Azure Event Hubs](#)
  - [Azure Relay](#)
- Latest PowerShell documentation
  - [Azure Service Bus](#)
  - [Azure Event Hubs](#)
  - [Azure Event Grid](#)

# Exchange events between consumers and producers that use different protocols: AMQP, Kafka, and HTTPS

11/26/2021 • 10 minutes to read • [Edit Online](#)

Azure Event Hubs supports three protocols for consumers and producers: AMQP, Kafka, and HTTPS. Each one of these protocols has its own way of representing a message, so naturally the following question arises: if an application sends events to an Event Hub with one protocol and consumes them with a different protocol, what do the various parts and values of the event look like when they arrive at the consumer? This article discusses best practices for both producer and consumer to ensure that the values within an event are correctly interpreted by the consuming application.

The advice in this article specifically covers these clients, with the listed versions used in developing the code snippets:

- Kafka Java client (version 1.1.1 from <https://www.mvnrepository.com/artifact/org.apache.kafka/kafka-clients>)
- Microsoft Azure Event Hubs Client for Java (version 1.1.0 from <https://github.com/Azure/azure-event-hubs-java>)
- Microsoft Azure Event Hubs Client for .NET (version 2.1.0 from <https://github.com/Azure/azure-event-hubs-dotnet>)
- Microsoft Azure Service Bus (version 5.0.0 from <https://www.nuget.org/packages/WindowsAzure.ServiceBus>)
- HTTPS (supports producers only)

Other AMQP clients may behave slightly differently. AMQP has a well-defined type system, but the specifics of serializing language-specific types to and from that type system depends on the client, as does how the client provides access to the parts of an AMQP message.

## Event Body

All of the Microsoft AMQP clients represent the event body as an uninterpreted bag of bytes. A producing application passes a sequence of bytes to the client, and a consuming application receives that same sequence from the client. The interpretation of byte sequence happens within the application code.

When sending an event via HTTPS, the event body is the POSTed content, which is also treated as uninterpreted bytes. It is easy to achieve the same state in a Kafka producer or consumer by using the provided `ByteArraySerializer` and `ByteArrayDeserializer` as shown in the following code:

### Kafka byte[] producer

```
final Properties properties = new Properties();
// add other properties
properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, ByteArraySerializer.class.getName());

final KafkaProducer<Long, byte[]> producer = new KafkaProducer<Long, byte[]>(properties);

final byte[] eventBody = new byte[] { 0x01, 0x02, 0x03, 0x04 };
ProducerRecord<Long, byte[]> pr =
    new ProducerRecord<Long, byte[]>(myTopic, myPartitionId, myTimeStamp, eventBody);
```

### Kafka byte[] consumer

```

final Properties properties = new Properties();
// add other properties
properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, ByteArrayDeserializer.class.getName());

final KafkaConsumer<Long, byte[]> consumer = new KafkaConsumer<Long, byte[]>(properties);

ConsumerRecord<Long, byte[]> cr = /* receive event */
// cr.value() is a byte[] with values { 0x01, 0x02, 0x03, 0x04 }

```

This code creates a transparent byte pipeline between the two halves of the application and allows the application developer to manually serialize and deserialize in any way desired, including making deserialization decisions at runtime, for example based on type or sender information in user-set properties on the event.

Applications that have a single, fixed event body type may be able to use other Kafka serializers, and deserializers to transparently convert data. For example, consider an application, which uses JSON. The construction and interpretation of the JSON string happens at the application level. At the Event Hubs level, the event body is always a string, a sequence of bytes representing characters in the UTF-8 encoding. In this case, the Kafka producer or consumer can take advantage of the provided StringSerializer or StringDeserializer as shown in the following code:

### Kafka UTF-8 string producer

```

final Properties properties = new Properties();
// add other properties
properties.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());

final KafkaProducer<Long, String> producer = new KafkaProducer<Long, String>(properties);

final String exampleJson = "{\"name\":\"John\", \"number\":9001}";
ProducerRecord<Long, String> pr =
    new ProducerRecord<Long, String>(myTopic, myPartitionId, myTimeStamp, exampleJson);

```

### Kafka UTF-8 string consumer

```

final Properties properties = new Properties();
// add other properties
properties.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());

final KafkaConsumer<Long, String> consumer = new KafkaConsumer<Long, String>(properties);

ConsumerRecord<Long, Bytes> cr = /* receive event */
final String receivedJson = cr.value();

```

For the AMQP side, both Java and .NET provide built-in ways to convert strings to and from UTF-8 byte sequences. The Microsoft AMQP clients represent events as a class named `EventData`. The following examples show you how to serialize a UTF-8 string into an `EventData` event body in an AMQP producer, and how to deserialize an `EventData` event body into a UTF-8 string in an AMQP consumer.

### Java AMQP UTF-8 string producer

```

final String exampleJson = "{\"name\":\"John\", \"number\":9001}";
final EventData ed = EventData.create(exampleJson.getBytes(StandardCharsets.UTF_8));

```

### Java AMQP UTF-8 string consumer

```

EventData ed = /* receive event */
String receivedJson = new String(ed.getBytes(), StandardCharsets.UTF_8);

```

## C# .NET UTF-8 string producer

```
string exampleJson = "{\"name\":\"John\", \"number\":9001}";
EventData working = new EventData(Encoding.UTF8.GetBytes(exampleJson));
```

## C# .NET UTF-8 string consumer

```
EventData ed = /* receive event */

// getting the event body bytes depends on which .NET client is used
byte[] bodyBytes = ed.Body.Array; // Microsoft Azure Event Hubs Client for .NET
// byte[] bodyBytes = ed.GetBytes(); // Microsoft Azure Service Bus

string receivedJson = Encoding.UTF8.GetString(bodyBytes);
```

Because Kafka is open-source, the application developer can inspect the implementation of any serializer or deserializer and implement code, which produces or consumes a compatible sequence of bytes on the AMQP side.

## Event User Properties

User-set properties can be set and retrieved from both AMQP clients (in the Microsoft AMQP clients they are called properties) and Kafka (where they are called headers). HTTPS senders can set user properties on an event by supplying them as HTTP headers in the POST operation. However, Kafka treats both event bodies and event header values as byte sequences. Whereas in AMQP clients, property values have types, which are communicated by encoding the property values according to the AMQP type system.

HTTPS is a special case. At the point of sending, all property values are UTF-8 text. The Event Hubs service does a limited amount of interpretation to convert appropriate property values to AMQP-encoded 32-bit and 64-bit signed integers, 64-bit floating point numbers, and booleans. Any property value, which does not fit one of those types is treated as a string.

Mixing these approaches to property typing means that a Kafka consumer sees the raw AMQP-encoded byte sequence, including the AMQP type information. Whereas an AMQP consumer sees the untyped byte sequence sent by the Kafka producer, which the application must interpret.

For Kafka consumers that receive properties from AMQP or HTTPS producers, use the AmqpDeserializer class, which is modeled after the other deserializers in the Kafka ecosystem. It interprets the type information in the AMQP-encoded byte sequences to deserialize the data bytes into a Java type.

As a best practice, we recommend that you include a property in messages sent via AMQP or HTTPS. The Kafka consumer can use it to determine whether header values need AMQP deserialization. The value of the property is not important. It just needs a well-known name that the Kafka consumer can find in the list of headers and adjust its behavior accordingly.

### AMQP to Kafka part 1: create and send an event in C# (.NET) with properties

```
// Create an event with properties "MyStringProperty" and "MyIntegerProperty"
EventData working = new EventData(Encoding.UTF8.GetBytes("an event body"));
working.Properties.Add("MyStringProperty", "hello");
working.Properties.Add("MyIntegerProperty", 1234);

// BEST PRACTICE: include a property which indicates that properties will need AMQP deserialization
working.Properties.Add("AMQPheaders", 0);
```

### AMQP to Kafka part 2: use AmqpDeserializer to deserialize those properties in a Kafka consumer

```

final AmqpDeserializer amqpDeser = new AmqpDeserializer();

ConsumerRecord<Long, Bytes> cr = /* receive event */
final Header[] headers = cr.headers().toArray();

final Header headerNamedMyStringProperty = /* find header with key "MyStringProperty" */
final Header headerNamedMyIntegerProperty = /* find header with key "MyIntegerProperty" */
final Header headerNamedAMQPheaders = /* find header with key "AMQPheaders", or null if not found */

// BEST PRACTICE: detect whether AMQP deserialization is needed
if (headerNamedAMQPheaders != null) {
    // The deserialize() method requires no prior knowledge of a property's type.
    // It returns Object and the application can check the type and perform a cast later.
    Object propertyOfUnknownType = amqpDeser.deserialize("topicname", headerNamedMyStringProperty.value());
    if (propertyOfUnknownType instanceof String) {
        final String propertyString = (String)propertyOfUnknownType;
        // do work here
    }
    propertyOfUnknownType = amqpDeser.deserialize("topicname", headerNamedMyIntegerProperty.value());
    if (propertyOfUnknownType instanceof Integer) {
        final Integer propertyInt = (Integer)propertyOfUnknownType;
        // do work here
    }
} else {
    /* event sent via Kafka, interpret header values the Kafka way */
}

```

If the application knows the expected type for a property, there are deserialization methods that do not require a cast afterwards, but they throw an error if the property is not of the expected type.

### AMQP to Kafka part 3: a different way of using AmqpDeserializer in a Kafka consumer

```

// BEST PRACTICE: detect whether AMQP deserialization is needed
if (headerNamedAMQPheaders != null) {
    // Property "MyStringProperty" is expected to be of type string.
    try {
        final String propertyString = amqpDeser.deserializeString(headerNamedMyStringProperty.value());
        // do work here
    }
    catch (IllegalArgumentException e) {
        // property was not a string
    }

    // Property "MyIntegerProperty" is expected to be a signed integer type.
    // The method returns long because long can represent the value range of all AMQP signed integer types.
    try {
        final long propertyLong = amqpDeser.deserializeSignedInteger(headerNamedMyIntegerProperty.value());
        // do work here
    }
    catch (IllegalArgumentException e) {
        // property was not a signed integer
    }
} else {
    /* event sent via Kafka, interpret header values the Kafka way */
}

```

Going the other direction is more involved, because headers set by a Kafka producer are always seen by an AMQP consumer as raw bytes (type `org.apache.qpid.proton.amqp.Binary` for the Microsoft Azure Event Hubs Client for Java, or `System.Byte[]` for Microsoft's .NET AMQP clients). The easiest path is to use one of the Kafka-supplied serializers to generate the bytes for the header values on the Kafka producer side, and then write a compatible deserialization code on the AMQP consumer side.

As with AMQP-to-Kafka, the best practice that we recommend is to include a property in messages sent via

Kafka. The AMQP consumer can use the property to determine whether header values need deserialization. The value of the property is not important. It just needs a well-known name that the AMQP consumer can find in the list of headers and adjust its behavior accordingly. If the Kafka producer cannot be changed, it is also possible for the consuming application to check whether the property value is of a binary or byte type and attempt deserialization based on the type.

### Kafka to AMQP part 1: create and send an event from Kafka with properties

```
final String topicName = /* topic name */
final ProducerRecord<Long, String> pr = new ProducerRecord<Long, String>(topicName, /* other arguments */);
final Headers h = pr.headers();

// Set headers using Kafka serializers
IntegerSerializer intSer = new IntegerSerializer();
h.add("MyIntegerProperty", intSer.serialize(topicName, 1234));

LongSerializer longSer = new LongSerializer();
h.add("MyLongProperty", longSer.serialize(topicName, 5555555555L));

ShortSerializer shortSer = new ShortSerializer();
h.add("MyShortProperty", shortSer.serialize(topicName, (short)2222));

FloatSerializer floatSer = new FloatSerializer();
h.add("MyFloatProperty", floatSer.serialize(topicName, 1.125F));

DoubleSerializer doubleSer = new DoubleSerializer();
h.add("MyDoubleProperty", doubleSer.serialize(topicName, Double.MAX_VALUE));

StringSerializer stringSer = new StringSerializer();
h.add("MyStringProperty", stringSer.serialize(topicName, "hello world"));

// BEST PRACTICE: include a property which indicates that properties will need deserialization
h.add("RawHeaders", intSer.serialize(0));
```

### Kafka to AMQP part 2: manually deserialize those properties in C# (.NET)

```

EventData ed = /* receive event */

// BEST PRACTICE: detect whether manual deserialization is needed
if (ed.Properties.ContainsKey("RawHeaders"))
{
    // Kafka serializers send bytes in big-endian order, whereas .NET on x86/x64 is little-endian.
    // Therefore it is frequently necessary to reverse the bytes before further deserialization.

    byte[] rawbytes = ed.Properties["MyIntegerProperty"] as System.Byte[];
    if (BitConverter.IsLittleEndian)
    {
        Array.Reverse(rawbytes);
    }
    int myIntegerProperty = BitConverter.ToInt32(rawbytes, 0);

    rawbytes = ed.Properties["MyLongProperty"] as System.Byte[];
    if (BitConverter.IsLittleEndian)
    {
        Array.Reverse(rawbytes);
    }
    long myLongProperty = BitConverter.ToInt64(rawbytes, 0);

    rawbytes = ed.Properties["MyShortProperty"] as System.Byte[];
    if (BitConverter.IsLittleEndian)
    {
        Array.Reverse(rawbytes);
    }
    short myShortProperty = BitConverter.ToInt16(rawbytes, 0);

    rawbytes = ed.Properties["MyFloatProperty"] as System.Byte[];
    if (BitConverter.IsLittleEndian)
    {
        Array.Reverse(rawbytes);
    }
    float myFloatProperty = BitConverter.ToSingle(rawbytes, 0);

    rawbytes = ed.Properties["MyDoubleProperty"] as System.Byte[];
    if (BitConverter.IsLittleEndian)
    {
        Array.Reverse(rawbytes);
    }
    double myDoubleProperty = BitConverter.ToDouble(rawbytes, 0);

    rawbytes = ed.Properties["MyStringProperty"] as System.Byte[];
    string myStringProperty = Encoding.UTF8.GetString(rawbytes);
}

```

### Kafka to AMQP part 3: manually deserialize those properties in Java

```

final EventData ed = /* receive event */

// BEST PRACTICE: detect whether manual deserialization is needed
if (ed.getProperties().containsKey("RawHeaders")) {
    byte[] rawbytes =
        ((org.apache.qpid.proton.amqp.Binary)ed.getProperties().get("MyIntegerProperty")).getArray();
    int myIntegerProperty = 0;
    for (byte b : rawbytes) {
        myIntegerProperty <= 8;
        myIntegerProperty |= ((int)b & 0x00FF);
    }

    rawbytes = ((org.apache.qpid.proton.amqp.Binary)ed.getProperties().get("MyLongProperty")).getArray();
    long myLongProperty = 0;
    for (byte b : rawbytes) {
        myLongProperty <= 8;
        myLongProperty |= ((long)b & 0x00FF);
    }

    rawbytes = ((org.apache.qpid.proton.amqp.Binary)ed.getProperties().get("MyShortProperty")).getArray();
    short myShortProperty = (short)rawbytes[0];
    myShortProperty <= 8;
    myShortProperty |= ((short)rawbytes[1] & 0x00FF);

    rawbytes = ((org.apache.qpid.proton.amqp.Binary)ed.getProperties().get("MyFloatProperty")).getArray();
    int intbits = 0;
    for (byte b : rawbytes) {
        intbits <= 8;
        intbits |= ((int)b & 0x00FF);
    }
    float myFloatProperty = Float.intBitsToFloat(intbits);

    rawbytes = ((org.apache.qpid.proton.amqp.Binary)ed.getProperties().get("MyDoubleProperty")).getArray();
    long longbits = 0;
    for (byte b : rawbytes) {
        longbits <= 8;
        longbits |= ((long)b & 0x00FF);
    }
    double myDoubleProperty = Double.longBitsToDouble(longbits);

    rawbytes = ((org.apache.qpid.proton.amqp.Binary)ed.getProperties().get("MyStringProperty")).getArray();
    String myStringProperty = new String(rawbytes, StandardCharsets.UTF_8);
}

```

## Next steps

In this article, you learned how to stream into Event Hubs without changing your protocol clients or running your own clusters. To learn more about Event Hubs and Event Hubs for Kafka, see the following articles:

- [Learn about Event Hubs](#)
- [Learn about Event Hubs for Kafka](#)
- [Explore more samples on the Event Hubs for Kafka GitHub](#)
- Use [MirrorMaker](#) to stream events from Kafka on premises to Event Hubs on cloud.
- Learn how to stream into Event Hubs using [native Kafka applications](#), [Apache Flink](#), or [Akka Streams](#)

# .NET Programming guide for Azure Event Hubs (legacy Microsoft.Azure.EventHubs package)

11/26/2021 • 5 minutes to read • [Edit Online](#)

This article discusses some common scenarios in writing code using Azure Event Hubs. It assumes a preliminary understanding of Event Hubs. For a conceptual overview of Event Hubs, see the [Event Hubs overview](#).

## WARNING

This guide is for the old **Microsoft.Azure.EventHubs** package. We recommend that you [migrate](#) your code to use the latest [Azure.Messaging.EventHubs](#) package.

## Event publishers

You send events to an event hub either using HTTP POST or via an AMQP 1.0 connection. The choice of which to use and when depends on the specific scenario being addressed. AMQP 1.0 connections are metered as brokered connections in Service Bus and are more appropriate in scenarios with frequent higher message volumes and lower latency requirements, as they provide a persistent messaging channel.

When using the .NET managed APIs, the primary constructs for publishing data to Event Hubs are the [EventHubClient](#) and [EventData](#) classes. [EventHubClient](#) provides the AMQP communication channel over which events are sent to the event hub. The [EventData](#) class represents an event, and is used to publish messages to an event hub. This class includes the body, some metadata([Properties](#)), and header information([SystemProperties](#)) about the event. Other properties are added to the [EventData](#) object as it passes through an event hub.

## Get started

The .NET classes that support Event Hubs are provided in the [Microsoft.Azure.EventHubs](#) NuGet package. You can install using the Visual Studio Solution explorer, or the [Package Manager Console](#) in Visual Studio. To do so, issue the following command in the [Package Manager Console](#) window:

```
Install-Package Microsoft.Azure.EventHubs
```

## Create an event hub

You can use the Azure portal, Azure PowerShell, or Azure CLI to create Event Hubs. For details, see [Create an Event Hubs namespace and an event hub using the Azure portal](#).

## Create an Event Hubs client

The primary class for interacting with Event Hubs is [Microsoft.Azure.EventHubs.EventHubClient](#). You can instantiate this class using the [CreateFromConnectionString](#) method, as shown in the following example:

```
private const string EventHubConnectionString = "Event Hubs namespace connection string";
private const string EventHubName = "event hub name";

var connectionStringBuilder = new EventHubsConnectionStringBuilder(EventHubConnectionString)
{
    EntityPath = EventHubName

};

eventHubClient = EventHubClient.CreateFromConnectionString(connectionStringBuilder.ToString());
```

## Send events to an event hub

You send events to an event hub by creating an [EventHubClient](#) instance and sending it asynchronously via the [SendAsync](#) method. This method takes a single [EventData](#) instance parameter and asynchronously sends it to an event hub.

## Event serialization

The [EventData](#) class has [two overloaded constructors](#) that take a variety of parameters, bytes or a byte array, that represent the event data payload. When using JSON with [EventData](#), you can use [Encoding.UTF8.GetBytes\(\)](#) to retrieve the byte array for a JSON-encoded string. For example:

```
for (var i = 0; i < numMessagesToSend; i++)
{
    var message = $"Message {i}";
    Console.WriteLine($"Sending message: {message}");
    await eventHubClient.SendAsync(new EventData(Encoding.UTF8.GetBytes(message)));
}
```

## Partition key

### NOTE

If you aren't familiar with partitions, see [this article](#).

When sending event data, you can specify a value that is hashed to produce a partition assignment. You specify the partition using the [PartitionSender.PartitionID](#) property. However, the decision to use partitions implies a choice between availability and consistency. For more information, see [Availability and Consistency](#).

## Batch event send operations

Sending events in batches can help increase throughput. You can use the [CreateBatch](#) API to create a batch to which data objects can later be added for a [SendAsync](#) call.

A single batch must not exceed the 1 MB limit of an event. Additionally, each message in the batch uses the same publisher identity. It is the responsibility of the sender to ensure that the batch does not exceed the maximum event size. If it does, a client [Send](#) error is generated. You can use the helper method [EventHubClient.CreateBatch](#) to ensure that the batch does not exceed 1 MB. You get an empty [EventDataBatch](#) from the [CreateBatch](#) API and then use [TryAdd](#) to add events to construct the batch.

## Send asynchronously and send at scale

You send events to an event hub asynchronously. Sending asynchronously increases the rate at which a client is able to send events. [SendAsync](#) returns a [Task](#) object. You can use the [RetryPolicy](#) class on the client to control

client retry options.

## Event consumers

The [EventProcessorHost](#) class processes data from Event Hubs. You should use this implementation when building event readers on the .NET platform. [EventProcessorHost](#) provides a thread-safe, multi-process, safe runtime environment for event processor implementations that also provides checkpointing and partition lease management.

To use the [EventProcessorHost](#) class, you can implement [IEventProcessor](#). This interface contains four methods:

- [OpenAsync](#)
- [CloseAsync](#)
- [ProcessEventsAsync](#)
- [ProcessErrorAsync](#)

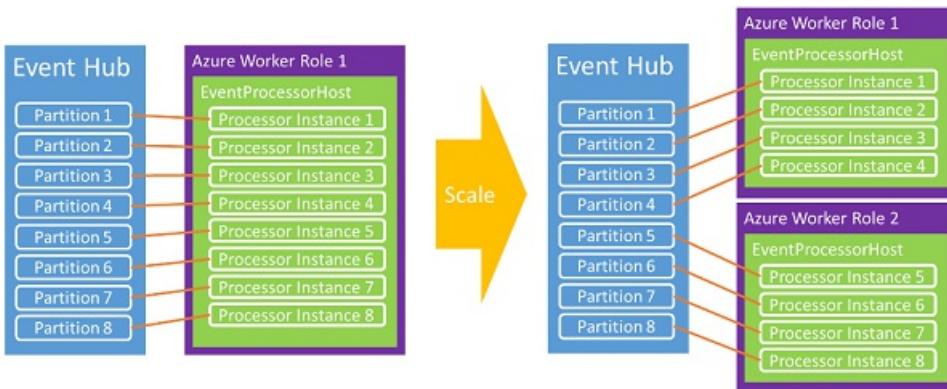
To start event processing, instantiate [EventProcessorHost](#), providing the appropriate parameters for your event hub. For example:

```
var eventProcessorHost = new EventProcessorHost(  
    EventHubName,  
    PartitionReceiver.DefaultConsumerGroupName,  
    EventHubConnectionString,  
    StorageConnectionString,  
    StorageContainerName);
```

Then, call [RegisterEventProcessorAsync](#) to register your [IEventProcessor](#) implementation with the runtime:

```
await eventProcessorHost.RegisterEventProcessorAsync<SimpleEventProcessor>();
```

At this point, the host attempts to acquire a lease on every partition in the event hub using a "greedy" algorithm. These leases last for a given timeframe and must then be renewed. As new nodes, worker instances in this case, come online, they place lease reservations and over time the load shifts between nodes as each attempts to acquire more leases.



Over time, an equilibrium is established. This dynamic capability enables CPU-based autoscaling to be applied to consumers for both scale-up and scale-down. Because Event Hubs does not have a direct concept of message counts, average CPU utilization is often the best mechanism to measure back end or consumer scale. If publishers begin to publish more events than consumers can process, the CPU increase on consumers can be used to cause an auto-scale on worker instance count.

The [EventProcessorHost](#) class also implements an Azure storage-based checkpointing mechanism. This mechanism stores the offset on a per partition basis, so that each consumer can determine what the last checkpoint from the previous consumer was. As partitions transition between nodes via leases, this is the

synchronization mechanism that facilitates load shifting.

## Publisher revocation

In addition to the advanced run-time features of Event Processor Host, the Event Hubs service enables [publisher revocation](#) in order to block specific publishers from sending event to an event hub. These features are useful if a publisher token has been compromised, or a software update is causing them to behave inappropriately. In these situations, the publisher's identity, which is part of their SAS token, can be blocked from publishing events.

### NOTE

Currently, only REST API supports this feature ([publisher revocation](#)).

## Next steps

To learn more about Event Hubs scenarios, visit these links:

- [Event Hubs API overview](#)
- [What is Event Hubs](#)
- [Availability and consistency in Event Hubs](#)
- [Event processor host API reference](#)

# Process data from your event hub using Azure Stream Analytics

11/26/2021 • 3 minutes to read • [Edit Online](#)

The Azure Stream Analytics service makes it easy to ingest, process, and analyze streaming data from Azure Event Hubs, enabling powerful insights to drive real-time actions. This integration allows you to quickly create a hot-path analytics pipeline. You can use the Azure portal to visualize incoming data and write a Stream Analytics query. Once your query is ready, you can move it into production in only a few clicks.

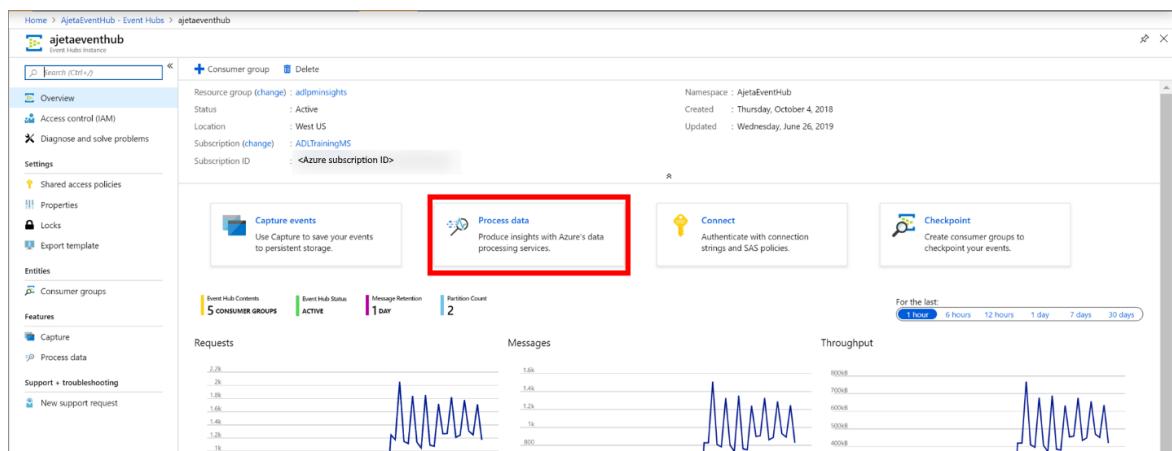
## Key benefits

Here are the key benefits of Azure Event Hubs and Azure Stream Analytics integration:

- **Preview data** – You can preview incoming data from an event hub in the Azure portal.
- **Test your query** – Prepare a transformation query and test it directly in the Azure portal. For the query language syntax, see [Stream Analytics Query Language](#) documentation.
- **Deploy your query to production** – You can deploy the query into production by creating and starting an Azure Stream Analytics job.

## End-to-end flow

1. Sign in to the [Azure portal](#).
2. Navigate to your **Event Hubs namespace** and then navigate to the **event hub**, which has the incoming data.
3. Select **Process Data** on the event hub page.



4. Select **Explore** on the Enable real-time insights from events tile.

The screenshot shows the 'ajetaeventhub - Process data' page in the Azure portal. On the left, there's a sidebar with options like Overview, Access control (IAM), Diagnose and solve problems, Settings (Shared access policies, Properties, Locks, Export template), Entities (Consumer groups), Features (Capture, Process data), and Support + troubleshooting. The 'Process data' option is highlighted with a red box. The main content area is titled 'Process Event Hub data with other Azure services' and includes a section 'Enable real time insights from events' with a gear icon and a list of bullet points: Preview Event Hub data, Analyze your data using SQL-like query, Deploy query by creating a new Azure Stream Analytics job. Below this is a button labeled 'Explore' which is also highlighted with a red box. To the right, there's a section 'More services coming soon!' with a triangle icon.

5. You see a query page with values already set for the following fields:

- Your **event hub** as an input for the query.
- Sample **SQL query** with SELECT statement.
- An **output alias** to refer to your query test results.

The screenshot shows the 'Query' page for the 'ajetaeventhub' Event Hub instance. The left sidebar shows 'Inputs (1)' with 'ajetaeventhub' selected and 'Outputs (1)' with 'outputalias' selected. The main area has a 'Test query' section containing a SQL script:

```

1 SELECT
2 *
3 INTO
4 [OutputAlias]
5 FROM
6 [ajetaeventhub]

```

Below the query editor are tabs for 'Input preview' and 'Test results'. A message 'We need your permission' is displayed, stating 'To preview your data, we need to create a shared access policy and a consumer group in your Event Hub instance.' A 'Create' button is highlighted with a red box. At the bottom, there's a note: '⚠ To preview your data, we need to create a shared access policy and a consumer group in your Event Hub instance.' and 'Ln 1, Col 1'.

#### NOTE

When you use this feature for the first time, this page asks for your permission to create a consumer group and a policy for your event hub to preview incoming data.

6. Select **Create** in the **Input preview** pane as shown in the preceding image.

7. You'll immediately see a snapshot of the latest incoming data in this tab.

- The serialization type in your data is automatically detected (JSON/CSV). You can manually change it as well to JSON/CSV/AVRO.
- You can preview incoming data in the table format or raw format.
- If your data shown isn't current, select **Refresh** to see the latest events.

Here is an example of data in the **table format**:

Home > AjetaEventHub - Event Hubs > ajetaeventhub - Process data > Query

**Query**  
Event Hub instance

Deploy query | Query language docs | UserVoice

Azure Stream Analytics lets you perform real-time analytics. Start by testing your query, then deploy your query as Azure Stream Analytics job. Learn more →

Inputs (1) Outputs (1)

```

▶ Test query
1 SELECT *
2
3 INTO [OutputAlias]
4 FROM [ajetaeventhub]
5
6

```

Input preview Test results

Showing events from ajetaeventhub 11:44:00 AM - 11:47:52 AM. Preview might not display a complete list of events.

View JSON Table Raw Refresh

R...	Sy...	Fl...	S...	C...	C...	C...	D...	Tl...	C...	C...	S...	Tr...	In...	O...	M...	C...	F...	ca...	Ev...	P...	Ev...
"MO"	"d0"	"182"	"UK"	"346..."	"345..."	"466..."	"201..."	null	1	921	null	null	"\$"	0	null	"443"	"141..."	null	"201..."	"201..."	
"MO"	"d0"	"942"	"US"	"345..."	"466..."	"466..."	"201..."	null	0	0	null	null	"\$"	0	null	"442"	"886..."	null	"201..."	"201..."	
"MO"	"d0"	"935"	"Aus..."	null	"466..."	"466..."	"F"	null	2	0	null	null	"a"	0	null	"621"	"886..."	null	"000..."	"201..."	
"MO"	"d0"	"933"	"Chi..."	"567..."	"466..."	"567..."	"466..."	"201..."	null	3	0	null	null	"V"	1	null	"400"	"886..."	null	"201..."	"201..."
"MO"	"d0"	"927"	"UK"	"567..."	"466..."	"789..."	"466..."	"201..."	null	2	0	null	null	"V"	1	null	"425"	"141..."	null	"201..."	"201..."
"MO"	"d0"	"925"	"US"	"789..."	"466..."	"123..."	"466..."	"201..."	null	1	0	null	null	"V"	0	null	"409"	"141..."	null	"201..."	"201..."
"MO"	"d0"	"923"	"Aus..."	"789..."	"466..."	"012..."	"466..."	"201..."	null	2	0	null	null	"V"	1	null	"621"	"141..."	null	"201..."	"201..."
"MO"	"d0"	"178"	"Aus..."	"789..."	"466..."	"567..."	"466..."	"201..."	null	1	934	null	null	"b"	1	null	"419"	"142..."	null	"201..."	"201..."
"MO"	"d0"	"916"	"UK"	"123..."	"466..."	"012..."	"466..."	"201..."	null	2	25	null	null	"V"	0	null	"300"	"886..."	null	"201..."	"201..."

Success

Here is an example of data in the raw format:

Home > AjetaEventHub - Event Hubs > ajetaeventhub - Process data > Query

**Query**  
Event Hub instance

Deploy query | Query language docs | UserVoice

Azure Stream Analytics lets you perform real-time analytics. Start by testing your query, then deploy your query as Azure Stream Analytics job. Learn more →

Inputs (1) Outputs (1)

```

▶ Test query
1 SELECT *
2
3 INTO [OutputAlias]
4 FROM [ajetaeventhub]
5
6

```

Input preview Test results

Showing events from ajetaeventhub 11:44:00 AM - 11:47:52 AM. Preview might not display a complete list of events.

View JSON Table Raw Refresh

```

1 [
2   {
3     "RecordType": "MO",
4     "SystemIdentity": "d0",
5     "FileName": "182",
6     "SwitchNum": "UK",
7     "CallingNum": "345690943",
8     "CallingIMST": "46692702346260",
9     "CalledNum": "345679188",
10    "CalledIMST": "466923300507919",
11    "Dates": "20190708",
12    "Times": null,
13    "TimeType": 1,
14    "CallPeriod": 921,
15    "CallingCellID": null,
16    "CalledCellID": null,
17    "ServiceType": "S",
18  }
19 ]

```

Success

8. Select **Test query** to see the snapshot of test results of your query in the **Test results** tab. You can also download the results.

The screenshot shows the Azure Stream Analytics Query editor. At the top, there's a navigation bar with 'Home > AjetaEventHub - Event Hubs > ajetaeventhub - Process data > Query'. Below the navigation is a toolbar with 'Query' (selected), 'Deploy query', 'Query language docs', and 'UserVoice'. A status bar at the bottom says 'Input preview Test results' and 'Showing 4 rows from outputalias'. The main area contains a query editor with the following SQL:

```

1 SELECT System.Timestamp AS WindowEnd, COUNT(*) AS FraudulentCalls
2 INTO "outputalias"
3 FROM "ajetaeventhub" CS1 TIMESTAMP BY CallRecTime
4 JOIN "ajetaeventhub" CS2 TIMESTAMP BY CallRecTime
5 ON CS1.CallingIMSI = CS2.CallingIMSI
6 AND DATEDIFF(ss, CS1, CS2) BETWEEN 0 AND 5
7 WHERE CS1.SwitchNum != CS2.SwitchNum
8 GROUP BY TumblingWindow(Duration(second, 1))

```

Below the query editor is a table titled 'fraudulentcalls' with columns 'windowend' and 'fraudulentcalls'. The data is as follows:

windowend	fraudulentcalls
'2019-07-08T18:56:02.000000Z'	1
'2019-07-08T18:56:03.000000Z'	2
'2019-07-08T18:56:05.000000Z'	2
'2019-07-08T18:56:06.000000Z'	2

At the bottom right of the editor, it says 'Ln 6, Col 37'.

9. Write your own query to transform the data. See [Stream Analytics Query Language reference](#).
10. Once you've tested the query and you want to move it in to production, select **Deploy query**. To deploy the query, create an Azure Stream Analytics job where you can set an output for your job, and start the job. To create a Stream Analytics job, specify a name for the job, and select **Create**.

The screenshot shows the 'New Stream Analytics job' dialog. It has four numbered callouts:

- 1**: Points to the 'Query' tab header.
- 2**: Points to the 'OK' button at the bottom left of the dialog.
- 3**: Points to the 'Job name' field containing 'ASAJobFromEventHub'.
- 4**: Points to the 'Create' button at the bottom right of the dialog.

The dialog includes fields for 'Subscription' (ADLTrainingMS), 'Resource group' (adlinsights), 'Location' (West US), 'Event Hub policy name' (ASAJobFromEventHub\_policy), and 'Event Hub consumer group' (ASAJobFromEventHub\_consumer\_group).

#### NOTE

We recommend that you create a consumer group and a policy for each new Azure Stream Analytics job that you create from the Event Hubs page. Consumer groups allow only five concurrent readers, so providing a dedicated consumer group for each job will avoid any errors that might arise from exceeding that limit. A dedicated policy allows you to rotate your key or revoke permissions without impacting other resources.

11. Your Stream Analytics job is now created where your query is the same that you tested, and input is your event hub.
12. To complete the pipeline, set the **output** of the query, and select **Start** to start the job.

## NOTE

Before starting the job, don't forget to replace the outputalias by the output name you created in Azure Stream Analytics.

The screenshot shows the Azure Stream Analytics job configuration page. The job name is 'ASAJobFromEventHub'. The 'Inputs' section shows one input named 'ajetaeventhub'. The 'Outputs' section shows one output named 'EventHubOutput' (labeled 1). The 'Query' section contains the following T-SQL code:

```
1 SELECT System.Timestamp AS WindowEnd, COUNT(*) AS FraudulentCalls
2 INTO "outputalias"
3 FROM "ajetaeventhub" CS1 TIMESTAMP BY CallRecTime
4 JOIN "ajetaeventhub" CS2 TIMESTAMP BY CallRecTime
5 ON CS1.CallingIMSI = CS2.CallingIMSI
6 AND DATEDIFF(ss, CS1, CS2) BETWEEN 0 AND 5
7 WHERE CS1.SwitchNum != CS2.SwitchNum
8 GROUP BY TumblingWindow(Duration(second, 1))
```

## Known limitations

While testing your query, the test results take approximately 6 seconds to load. We're working on improving the performance of testing. However, when deployed in production, Azure Stream Analytics will have subsecond latency.

## Streaming units

Your Azure Stream Analytics job defaults to three streaming units (SUs). To adjust this setting, select **Scale** on the left menu in the **Stream Analytics job** page in the Azure portal. To learn more about streaming units, see [Understand and adjust Streaming Units](#).

The screenshot shows the Azure Stream Analytics job configuration page for 'ajetacloudjob'. The 'Configure' section includes options for 'Storage account settings', 'Scale' (which is highlighted with a red box), and 'Locale'.

## Next steps

To learn more about Stream Analytics queries, see [Stream Analytics Query Language](#)

# Apache Kafka developer guide for Azure Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

This article provides links to articles that describe how to integrate your Apache Kafka applications with Azure Event Hubs.

## Overview

Event Hubs provides a Kafka endpoint that can be used by your existing Kafka based applications as an alternative to running your own Kafka cluster. Event Hubs works with many of your existing Kafka applications. For more information, see [Event Hubs for Apache Kafka](#)

## Quickstarts

You can find quickstarts in GitHub and in this content set that helps you quickly ramp up on Event Hubs for Kafka.

### Quickstarts in GitHub

See the following quickstarts in the [azure-event-hubs-for-kafka](#) repo:

CLIENT LANGUAGE/FRAMEWORK	DESCRIPTION
.NET	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in C# using .NET Core 2.0.</p> <p>This sample is based on <a href="#">Confluent's Apache Kafka .NET client</a>, modified for use with Event Hubs for Kafka.</p>
Java	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in Java.</p>
Node.js	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in Node.</p> <p>This sample uses the <a href="#">node-rdkafka</a> library.</p>
Python	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in python.</p> <p>This sample is based on <a href="#">Confluent's Apache Kafka Python client</a>, modified for use with Event Hubs for Kafka.</p>

CLIENT LANGUAGE/FRAMEWORK	DESCRIPTION
<a href="#">Go</a>	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in Go.</p> <p>This sample is based on <a href="#">Confluent's Apache Kafka Golang client</a>, modified for use with Event Hubs for Kafka.</p>
<a href="#">Sarama kafka Go</a>	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in Go using the <a href="#">Sarama Kafka client library</a>.</p>
<a href="#">Kafka</a>	<p>This quickstart will show how to create and connect to an Event Hubs Kafka endpoint using the CLI that comes bundled with the Apache Kafka distribution.</p>
<a href="#">Kafkacat</a>	<p>kafkacat is a non-JVM command-line consumer and producer based on librdkafka, popular due to its speed and small footprint. This quickstart contains a sample configuration and several simple sample kafkacat commands.</p>

## Quickstarts in DOCS

See the quickstart: [Data streaming with Event Hubs using the Kafka protocol](#) in this content set, which provides step-by-step instructions on how to stream into Event Hubs. You learn how to use your producers and consumers to talk to Event Hubs with just a configuration change in your applications.

## Tutorials

### Tutorials in GitHub

See the following tutorials on GitHub:

TUTORIAL	DESCRIPTION
<a href="#">Akka</a>	<p>This tutorial shows how to connect Akka Streams to Kafka-enabled Event Hubs without changing your protocol clients or running your own clusters. There are two separate tutorials using <a href="#">Java</a> and <a href="#">Scala</a> programming languages.</p>
<a href="#">Connect</a>	<p>This document will walk you through integrating Kafka Connect with Azure Event Hubs and deploying basic FileStreamSource and FileStreamSink connectors. While these connectors are not meant for production use, they demonstrate an end-to-end Kafka Connect Scenario where Azure Event Hubs masquerades as a Kafka broker.</p>
<a href="#">Filebeat</a>	<p>This document will walk you through integrating Filebeat and Event Hubs via Filebeat's Kafka output.</p>
<a href="#">Flink</a>	<p>This tutorial will show how to connect Apache Flink to Kafka-enabled Event Hubs without changing your protocol clients or running your own clusters.</p>

TUTORIAL	DESCRIPTION
<a href="#">FluentD</a>	This document will walk you through integrating Fluentd and Event Hubs using the <code>out_kafka</code> output plugin for Fluentd.
<a href="#">Interop</a>	This tutorial shows you how to exchange events between consumers and producers using different protocols.
<a href="#">Logstash</a>	This tutorial will walk you through integrating Logstash with Kafka-enabled Event Hubs using Logstash Kafka input/output plugins.
<a href="#">MirrorMaker</a>	This tutorial shows how an event hub and Kafka MirrorMaker can integrate an existing Kafka pipeline into Azure by mirroring the Kafka input stream in the Event Hubs service.
<a href="#">NiFi</a>	This tutorial will show how to connect Apache NiFi to an Event Hubs namespace.
<a href="#">OAuth</a>	Quickstarts show you how to create and connect to an Event Hubs Kafka endpoint using an example producer and consumer written in Go and Java programming languages.
<a href="#">Confluent's Schema Registry</a>	This tutorial will walk you through integrating Schema Registry and Event Hubs for Kafka.
<a href="#">Spark</a>	This tutorial will show how to connect your Spark application to an event hub without changing your protocol clients or running your own Kafka clusters.

## Tutorials in DOCS

Also, see the tutorial: [Process Apache Kafka for Event Hubs events using Stream analytics](#) in this content set, which shows how to stream data into Event Hubs and process it with Azure Stream Analytics.

## How-to guides

See the following How-to guides in our documentation:

ARTICLE	DESCRIPTION
<a href="#">Mirror a Kafka broker in an event hub</a>	Shows how to mirror a Kafka broker in an event hub using Kafka MirrorMaker.
<a href="#">Connect Apache Spark to an event hub</a>	Walks you through connecting your Spark application to Event Hubs for real-time streaming.
<a href="#">Connect Apache Flink to an event hub</a>	Shows you how to connect Apache Flink to an event hub without changing your protocol clients or running your own clusters.
<a href="#">Integrate Apache Kafka Connect with a event hub (Preview)</a>	Walks you through integrating Kafka Connect with an event hub and deploying basic FileStreamSource and FileStreamSink connectors.

ARTICLE	DESCRIPTION
<a href="#">Connect Akka Streams to an event hub</a>	Shows you how to connect Akka Streams to an event hub without changing your protocol clients or running your own clusters.
<a href="#">Use the Spring Boot Starter for Apache Kafka with Azure Event Hubs</a>	Demonstrates how to configure a Java-based Spring Cloud Stream Binder created with the Spring Boot Initializer to use Apache Kafka with Azure Event Hubs.

## Next steps

Review samples in the GitHub repo [azure-event-hubs-for-kafka](#) under quickstart and tutorials folders.

Also, see the following articles:

- [Apache Kafka troubleshooting guide for Event Hubs](#)
- [Frequently asked questions - Event Hubs for Apache Kafka](#)
- [Apache Kafka migration guide for Event Hubs](#)

# Migrate to Azure Event Hubs for Apache Kafka Ecosystems

11/26/2021 • 2 minutes to read • [Edit Online](#)

Azure Event Hubs exposes an Apache Kafka endpoint, which enables you to connect to Event Hubs using the Kafka protocol. By making minimal changes to your existing Kafka application, you can connect to Azure Event Hubs and reap the benefits of the Azure ecosystem. Event Hubs works with many of your existing Kafka applications, including MirrorMaker. For more information, see [Event Hubs for Apache Kafka](#)

## Pre-migration

### Create an Azure account

If you don't have an Azure subscription, create a [free account](#) before you begin.

### Create an Event Hubs namespace

Follow step-by-step instructions in the [Create an event hub](#) article to create an Event Hubs namespace and an event hub.

### Connection string

Follow steps from the [Get connection string from the portal](#) article. And, note down the connection string for later use.

### Fully qualified domain name (FQDN)

You may also need the FQDN that points to your Event Hub namespace. The FQDN can be found within your connection string as follows:

```
Endpoint=sb:// mynamespace.servicebus.windows.net /;SharedAccessKeyName=XXXXXX;SharedAccessKey=XXXXXX
```

If your Event Hubs namespace is deployed on a non-public cloud, your domain name may differ (for example, \*.servicebus.chinacloudapi.cn, \*.servicebus.usgovcloudapi.net, or \*.servicebus.cloudapi.de).

## Migration

### Update your Kafka client configuration

To connect to a Kafka-enabled Event Hub, you'll need to update the Kafka client configurations. If you're having trouble finding yours, try searching for where `bootstrap.servers` is set in your application.

Insert the following configs wherever makes sense in your application. Make sure to update the `bootstrap.servers` and `sasl.jaas.config` values to direct the client to your Event Hubs Kafka endpoint with the correct authentication.

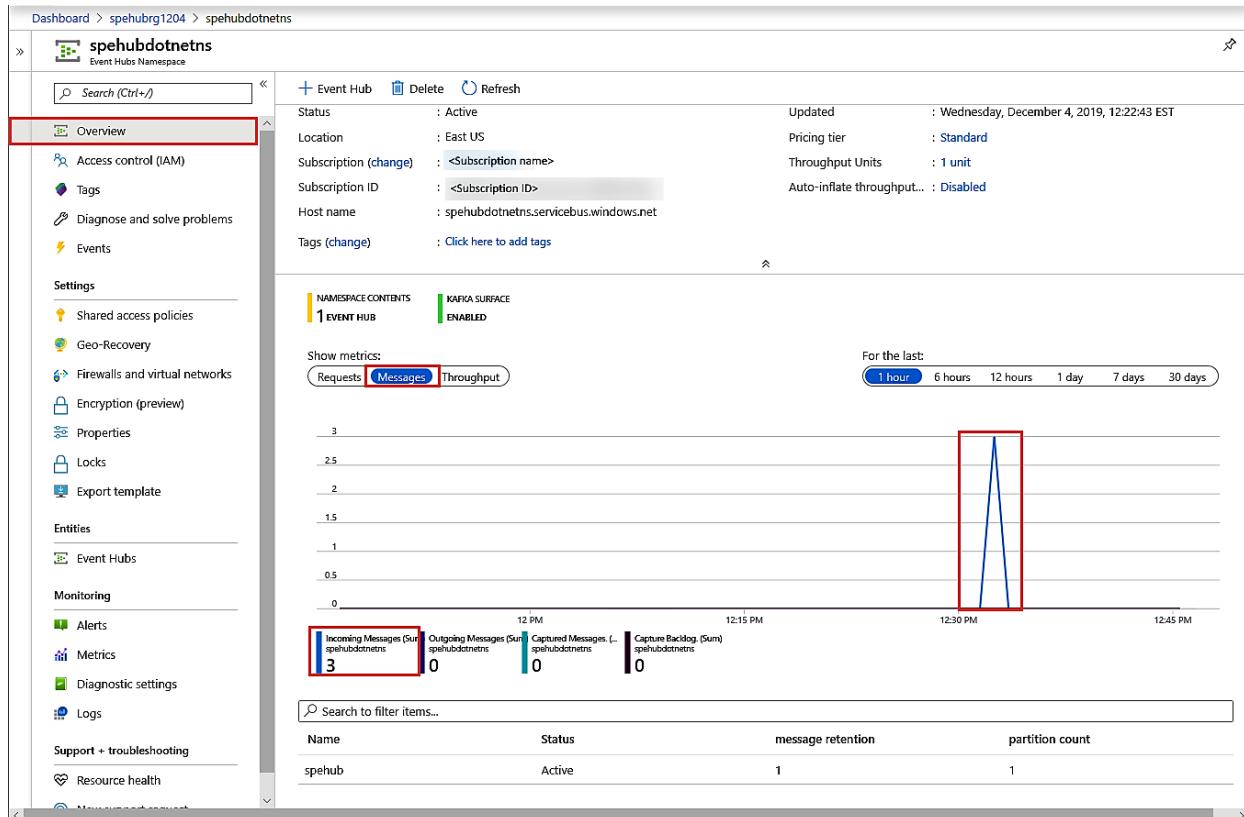
```
bootstrap.servers={MYNAMESPACE}.servicebus.windows.net:9093
request.timeout.ms=60000
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{CONNECTION STRING TO YOUR NAMESPACE}";
```

If `sasl.jaas.config` isn't a supported configuration in your framework, find the configurations that are used to set the SASL username and password and use them instead. Set the username to `$ConnectionString` and the

password to your Event Hubs connection string.

## Post-migration

Run your Kafka application that sends events to the event hub. Then, verify that the event hub receives the events using the Azure portal. On the **Overview** page of your Event Hubs namespace, switch to the **Messages** view in the **Metrics** section. Refresh the page to update the chart. It may take a few seconds for it to show that the messages have been received.



## Next steps

To learn more about Event Hubs and Event Hubs for Kafka, see the following articles:

- [Apache Kafka troubleshooting guide for Event Hubs](#)
- [Frequently asked questions - Event Hubs for Apache Kafka](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)
- [Recommended configurations](#)

# Apache Kafka troubleshooting guide for Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

This article provides troubleshooting tips for issues that you may run into when using Event Hubs for Apache Kafka.

## Server Busy exception

You may receive Server Busy exception because of Kafka throttling. With AMQP clients, Event Hubs immediately returns a **server busy** exception upon service throttling. It's equivalent to a "try again later" message. In Kafka, messages are delayed before being completed. The delay length is returned in milliseconds as `throttle_time_ms` in the produce/fetch response. In most cases, these delayed requests aren't logged as server busy exceptions on Event Hubs dashboards. Instead, the response's `throttle_time_ms` value should be used as an indicator that throughput has exceeded the provisioned quota.

If the traffic is excessive, the service has the following behavior:

- If produce request's delay exceeds request timeout, Event Hubs returns **Policy Violation** error code.
- If fetch request's delay exceeds request timeout, Event Hubs logs the request as throttled and responds with empty set of records and no error code.

[Dedicated clusters](#) don't have throttling mechanisms. You're free to consume all of your cluster resources.

## No records received

You may see consumers not getting any records and constantly rebalancing. In this scenario, consumers don't get any records and constantly rebalance. There's no exception or error when it happens, but the Kafka logs will show that the consumers are stuck trying to rejoin the group and assign partitions. There are a few possible causes:

- Make sure that your `request.timeout.ms` is at least the recommended value of 60000 and your `session.timeout.ms` is at least the recommended value of 30000. Having these settings too low could cause consumer timeouts, which then cause rebalances (which then cause more timeouts, which cause more rebalancing, and so on)
- If your configuration matches those recommended values, and you're still seeing constant rebalancing, feel free to open up an issue (make sure to include your entire configuration in the issue so that we can help debug)!

## Compression/Message format version issue

Kafka supports compression, and Event Hubs for Kafka currently doesn't. Errors that mention a message-format version (for example, `The message format version on the broker does not support the request.`) are caused when a client tries to send compressed Kafka messages to our brokers.

If compressed data is necessary, compressing your data before sending it to the brokers and decompressing after receiving is a valid workaround. The message body is just a byte array to the service, so client-side compression/decompression won't cause any issues.

## UnknownServerException

You may receive an UnknownServerException from Kafka client libraries similar to the following example:

```
org.apache.kafka.common.errors.UnknownServerException: The server experienced an unexpected error when processing the request
```

Open a ticket with Microsoft support. Debug-level logging and exception timestamps in UTC are helpful in debugging the issue.

## Other issues

Check the following items if you see issues when using Kafka on Event Hubs.

- **Firewall blocking traffic** - Make sure that port 9093 isn't blocked by your firewall.
- **TopicAuthorizationException** - The most common causes of this exception are:
  - A typo in the connection string in your configuration file, or
  - Trying to use Event Hubs for Kafka on a Basic tier namespace. The Event Hubs for Kafka feature isn't supported in the basic tier.
- **Kafka version mismatch** - Event Hubs for Kafka Ecosystems supports Kafka versions 1.0 and later. Some applications using Kafka version 0.10 and later could occasionally work because of the Kafka protocol's backwards compatibility, but we strongly recommend against using old API versions. Kafka versions 0.9 and earlier don't support the required SASL protocols and can't connect to Event Hubs.
- **Strange encodings on AMQP headers when consuming with Kafka** - when sending events to an event hub over AMQP, any AMQP payload headers are serialized in AMQP encoding. Kafka consumers don't deserialize the headers from AMQP. To read header values, manually decode the AMQP headers.  
Alternatively, you can avoid using AMQP headers if you know that you'll be consuming via Kafka protocol.  
For more information, see [this GitHub issue](#).
- **SASL authentication** - Getting your framework to cooperate with the SASL authentication protocol required by Event Hubs can be more difficult than meets the eye. See if you can troubleshoot the configuration using your framework's resources on SASL authentication.

## Limits

Apache Kafka vs. Event Hubs Kafka. For the most part, Azure Event Hubs' Kafka interface has the same defaults, properties, error codes, and general behavior that Apache Kafka does. The instances that these two explicitly differ (or where Event Hubs imposes a limit that Kafka doesn't) are listed below:

- The max length of the `group.id` property is 256 characters
- The max size of `offset.metadata.max.bytes` is 1024 bytes
- Offset commits are throttled to 4 calls/second per partition with a maximum internal log size of 1 MB

## Next steps

To learn more about Event Hubs and Event Hubs for Kafka, see the following articles:

- [Apache Kafka developer guide for Event Hubs](#)
- [Apache Kafka migration guide for Event Hubs](#)
- [Frequently asked questions - Event Hubs for Apache Kafka](#)
- [Recommended configurations](#)

# Use Apache Kafka MirrorMaker with Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

This tutorial shows how to mirror a Kafka broker into an Azure Event Hub using Kafka MirrorMaker. If you are hosting Apache Kafka on Kubernetes using the CNCF Strimzi operator, you can refer to the tutorial in [this blog post](#) to learn how to set up Kafka with Strimzi and Mirror Maker 2.



## NOTE

This sample is available on [GitHub](#)

## NOTE

This article contains references to the term *whitelist*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

In this tutorial, you learn how to:

- Create an Event Hubs namespace
- Clone the example project
- Set up a Kafka cluster
- Configure Kafka MirrorMaker
- Run Kafka MirrorMaker

## Introduction

This tutorial shows how an event hub and Kafka MirrorMaker can integrate an existing Kafka pipeline into Azure by "mirroring" the Kafka input stream in the Event Hubs service, which allows for integration of Apache Kafka streams using several [federation patterns](#).

An Azure Event Hubs Kafka endpoint enables you to connect to Azure Event Hubs using the Kafka protocol (that is, Kafka clients). By making minimal changes to a Kafka application, you can connect to Azure Event Hubs and enjoy the benefits of the Azure ecosystem. Event Hubs currently supports the protocol of Apache Kafka versions 1.0 and later.

You can use Apache Kafka's MirrorMaker 1 unidirectionally from Apache Kafka to Event Hubs. MirrorMaker 2 can be used in both directions, but the [MirrorCheckpointConnector](#) and [MirrorHeartbeatConnector](#) that are [configurable in MirrorMaker 2](#) must both be configured to point to the Apache Kafka broker and not to Event Hubs. This tutorial shows configuring MirrorMaker 1.

## Prerequisites

To complete this tutorial, make sure you have:

- Read through the [Event Hubs for Apache Kafka](#) article.
- An Azure subscription. If you do not have one, create a [free account](#) before you begin.
- [Java Development Kit \(JDK\) 1.7+](#)
  - On Ubuntu, run `apt-get install default-jdk` to install the JDK.
  - Be sure to set the `JAVA_HOME` environment variable to point to the folder where the JDK is installed.
- [Download](#) and [install](#) a Maven binary archive
  - On Ubuntu, you can run `apt-get install maven` to install Maven.
- [Git](#)
  - On Ubuntu, you can run `sudo apt-get install git` to install Git.

## Create an Event Hubs namespace

An Event Hubs namespace is required to send and receive from any Event Hubs service. See [Creating an event hub](#) for instructions to create a namespace and an event hub. Make sure to copy the Event Hubs connection string for later use.

## Clone the example project

Now that you have an Event Hubs connection string, clone the Azure Event Hubs for Kafka repository and navigate to the `mirror-maker` subfolder:

```
git clone https://github.com/Azure/azure-event-hubs-for-kafka.git
cd azure-event-hubs-for-kafka/tutorials/mirror-maker
```

## Set up a Kafka cluster

Use the [Kafka quickstart guide](#) to set up a cluster with the desired settings (or use an existing Kafka cluster).

## Configure Kafka MirrorMaker

Kafka MirrorMaker enables the "mirroring" of a stream. Given source and destination Kafka clusters, MirrorMaker ensures any messages sent to the source cluster are received by both the source and destination clusters. This example shows how to mirror a source Kafka cluster with a destination event hub. This scenario can be used to send data from an existing Kafka pipeline to Event Hubs without interrupting the flow of data.

For more detailed information on Kafka MirrorMaker, see the [Kafka Mirroring/MirrorMaker guide](#).

To configure Kafka MirrorMaker, give it a Kafka cluster as its consumer/source and an event hub as its producer/destination.

### Consumer configuration

Update the consumer configuration file `source-kafka.config`, which tells MirrorMaker the properties of the source Kafka cluster.

```
source-kafka.config
```

```
bootstrap.servers={SOURCE.KAFKA.IP.ADDRESS1}:{SOURCE.KAFKA.PORT1},{SOURCE.KAFKA.IP.ADDRESS2}:
{SOURCE.KAFKA.PORT2},etc
group.id=example-mirrormaker-group
exclude.internal.topics=true
client.id=mirror_maker_consumer
```

### Producer configuration

Now update the producer configuration file `mirror-eventhub.config`, which tells MirrorMaker to send the duplicated (or "mirrored") data to the Event Hubs service. Specifically, change `bootstrap.servers` and `sasl.jaas.config` to point to your Event Hubs Kafka endpoint. The Event Hubs service requires secure (SASL) communication, which is achieved by setting the last three properties in the following configuration:

```
mirror-eventhub.config
```

```
bootstrap.servers={YOUR.EVENTHUBS.FQDN}:9093
client.id=mirror_maker_producer

#Required for Event Hubs
sasl.mechanism=PLAIN
security.protocol=SASL_SSL
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net;/SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXX"
```

## Run Kafka MirrorMaker

Run the Kafka MirrorMaker script from the root Kafka directory using the newly updated configuration files.

Make sure to either copy the config files to the root Kafka directory, or update their paths in the following command.

```
bin/kafka-mirror-maker.sh --consumer.config source-kafka.config --num.streams 1 --producer.config mirror-
eventhub.config --whitelist=".*"
```

To verify that events are reaching the event hub, see the ingress statistics in the [Azure portal](#), or run a consumer against the event hub.

With MirrorMaker running, any events sent to the source Kafka cluster are received by both the Kafka cluster and the mirrored event hub. By using MirrorMaker and an Event Hubs Kafka endpoint, you can migrate an existing Kafka pipeline to the managed Azure Event Hubs service without changing the existing cluster or interrupting any ongoing data flow.

## Samples

See the following samples on GitHub:

- [Sample code for this tutorial on GitHub](#)
- [Azure Event Hubs Kafka MirrorMaker running on an Azure Container Instance](#)

## Next steps

To learn more about Event Hubs for Kafka, see the following articles:

- [Connect Apache Spark to an event hub](#)
- [Connect Apache Flink to an event hub](#)
- [Integrate Kafka Connect with an event hub](#)
- [Explore samples on our GitHub](#)
- [Connect Akka Streams to an event hub](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)

# Connect your Apache Spark application with Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

This tutorial walks you through connecting your Spark application to Event Hubs for real-time streaming. This integration enables streaming without having to change your protocol clients, or run your own Kafka or Zookeeper clusters. This tutorial requires Apache Spark v2.4+ and Apache Kafka v2.0+.

## NOTE

This sample is available on [GitHub](#)

In this tutorial, you learn how to:

- Create an Event Hubs namespace
- Clone the example project
- Run Spark
- Read from Event Hubs for Kafka
- Write to Event Hubs for Kafka

## Prerequisites

Before you start this tutorial, make sure that you have:

- Azure subscription. If you don't have one, [create a free account](#).
- [Apache Spark v2.4](#)
- [Apache Kafka v2.0](#)
- [Git](#)

## NOTE

The Spark-Kafka adapter was updated to support Kafka v2.0 as of Spark v2.4. In previous releases of Spark, the adapter supported Kafka v0.10 and later but relied specifically on Kafka v0.10 APIs. As Event Hubs for Kafka does not support Kafka v0.10, the Spark-Kafka adapters from versions of Spark prior to v2.4 are not supported by Event Hubs for Kafka Ecosystems.

## Create an Event Hubs namespace

An Event Hubs namespace is required to send and receive from any Event Hubs service. See [Creating an event hub](#) for instructions to create a namespace and an event hub. Get the Event Hubs connection string and fully qualified domain name (FQDN) for later use. For instructions, see [Get an Event Hubs connection string](#).

## Clone the example project

Clone the Azure Event Hubs repository and navigate to the `tutorials/spark` subfolder:

```
git clone https://github.com/Azure/azure-event-hubs-for-kafka.git
cd azure-event-hubs-for-kafka/tutorials/spark
```

## Read from Event Hubs for Kafka

With a few configuration changes, you can start reading from Event Hubs for Kafka. Update **BOOTSTRAP\_SERVERS** and **EH\_SASL** with details from your namespace and you can start streaming with Event Hubs as you would with Kafka. For the full sample code, see `sparkConsumer.scala` file on the GitHub.

```
//Read from your Event Hub!
val df = spark.readStream
  .format("kafka")
  .option("subscribe", TOPIC)
  .option("kafka.bootstrap.servers", BOOTSTRAP_SERVERS)
  .option("kafka.sasl.mechanism", "PLAIN")
  .option("kafka.security.protocol", "SASL_SSL")
  .option("kafka.sasl.jaas.config", EH_SASL)
  .option("kafka.request.timeout.ms", "60000")
  .option("kafka.session.timeout.ms", "30000")
  .option("kafka.group.id", GROUP_ID)
  .option("failOnDataLoss", "true")
  .load()

//Use dataframe like normal (in this example, write to console)
val df_write = df.writeStream
  .outputMode("append")
  .format("console")
  .start()
```

## Write to Event Hubs for Kafka

You can also write to Event Hubs the same way you write to Kafka. Don't forget to update your configuration to change **BOOTSTRAP\_SERVERS** and **EH\_SASL** with information from your Event Hubs namespace. For the full sample code, see `sparkProducer.scala` file on the GitHub.

```
df = /**Dataframe**/

//Write to your Event Hub!
df.writeStream
  .format("kafka")
  .option("topic", TOPIC)
  .option("kafka.bootstrap.servers", BOOTSTRAP_SERVERS)
  .option("kafka.sasl.mechanism", "PLAIN")
  .option("kafka.security.protocol", "SASL_SSL")
  .option("kafka.sasl.jaas.config", EH_SASL)
  .option("checkpointLocation", "./checkpoint")
  .start()
```

## Next steps

To learn more about Event Hubs and Event Hubs for Kafka, see the following articles:

- [Mirror a Kafka broker in an event hub](#)
- [Connect Apache Flink to an event hub](#)
- [Integrate Kafka Connect with an event hub](#)
- [Explore samples on our GitHub](#)

- [Connect Akka Streams to an event hub](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)

# Use Apache Flink with Azure Event Hubs for Apache Kafka

11/26/2021 • 3 minutes to read • [Edit Online](#)

This tutorial shows you how to connect Apache Flink to an event hub without changing your protocol clients or running your own clusters. For more information on Event Hubs' support for the Apache Kafka consumer protocol, see [Event Hubs for Apache Kafka](#).

In this tutorial, you learn how to:

- Create an Event Hubs namespace
- Clone the example project
- Run Flink producer
- Run Flink consumer

## NOTE

This sample is available on [GitHub](#)

## Prerequisites

To complete this tutorial, make sure you have the following prerequisites:

- Read through the [Event Hubs for Apache Kafka](#) article.
- An Azure subscription. If you do not have one, create a [free account](#) before you begin.
- [Java Development Kit \(JDK\) 1.7+](#)
  - On Ubuntu, run `apt-get install default-jdk` to install the JDK.
  - Be sure to set the `JAVA_HOME` environment variable to point to the folder where the JDK is installed.
- [Download and install](#) a Maven binary archive
  - On Ubuntu, you can run `apt-get install maven` to install Maven.
- [Git](#)
  - On Ubuntu, you can run `sudo apt-get install git` to install Git.

## Create an Event Hubs namespace

An Event Hubs namespace is required to send or receive from any Event Hubs service. See [Creating an event hub](#) for instructions to create a namespace and an event hub. Make sure to copy the Event Hubs connection string for later use.

## Clone the example project

Now that you have the Event Hubs connection string, clone the Azure Event Hubs for Kafka repository and navigate to the `flink` subfolder:

```
git clone https://github.com/Azure/azure-event-hubs-for-kafka.git
cd azure-event-hubs-for-kafka/tutorials/flink
```

## Run Flink producer

Using the provided Flink producer example, send messages to the Event Hubs service.

### Provide an Event Hubs Kafka endpoint

#### `producer.config`

Update the `bootstrap.servers` and `sasl.jaas.config` values in `producer/src/main/resources/producer.config` to direct the producer to the Event Hubs Kafka endpoint with the correct authentication.

```
bootstrap.servers={YOUR.EVENTHUBS.FQDN}:9093
client.id=FlinkExampleProducer
sasl.mechanism=PLAIN
security.protocol=SASL_SSL
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="$ConnectionString" \
    password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net;/SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXXXX"
```

### Run producer from the command line

To run the producer from the command line, generate the JAR and then run from within Maven (or generate the JAR using Maven, then run in Java by adding the necessary Kafka JAR(s) to the classpath):

```
mvn clean package
mvn exec:java -Dexec.mainClass="FlinkTestProducer"
```

The producer will now begin sending events to the event hub at topic `test` and printing the events to stdout.

### Run Flink consumer

Using the provided consumer example, receive messages from the event hub.

#### Provide an Event Hubs Kafka endpoint

##### consumer.config

Update the `bootstrap.servers` and `sasl.jaas.config` values in `consumer/src/main/resources/consumer.config` to direct the consumer to the Event Hubs Kafka endpoint with the correct authentication.

```
bootstrap.servers={YOUR.EVENTHUBS.FQDN}:9093
group.id=FlinkExampleConsumer
sasl.mechanism=PLAIN
security.protocol=SASL_SSL
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="$ConnectionString" \
    password="{YOUR.EVENTHUBS.CONNECTION.STRING}";
```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net;/SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXXXX"
```

### Run consumer from the command line

To run the consumer from the command line, generate the JAR and then run from within Maven (or generate the JAR using Maven, then run in Java by adding the necessary Kafka JAR(s) to the classpath):

```
mvn clean package
mvn exec:java -Dexec.mainClass="FlinkTestConsumer"
```

If the event hub has events (for example, if your producer is also running), then the consumer now begins receiving events from the topic `test`.

Check out [Flink's Kafka Connector Guide](#) for more detailed information about connecting Flink to Kafka.

## Next steps

To learn more about Event Hubs for Kafka, see the following articles:

- [Mirror a Kafka broker in an event hub](#)
- [Connect Apache Spark to an event hub](#)
- [Integrate Kafka Connect with an event hub](#)
- [Explore samples on our GitHub](#)
- [Connect Akka Streams to an event hub](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)

# Integrate Apache Kafka Connect support on Azure Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

Apache Kafka Connect is a framework to connect and import/export data from/to any external system such as MySQL, HDFS, and file system through a Kafka cluster. This tutorial walks you through using Kafka Connect framework with Event Hubs.

## WARNING

Use of the Apache Kafka Connect framework and its connectors is **not eligible for product support through Microsoft Azure**.

Apache Kafka Connect assumes for its dynamic configuration to be held in compacted topics with otherwise unlimited retention. Azure Event Hubs [does not implement compaction as a broker feature](#) and always imposes a time-based retention limit on retained events, rooting from the principle that Azure Event Hubs is a real-time event streaming engine and not a long-term data or configuration store.

While the Apache Kafka project might be comfortable with mixing these roles, Azure believes that such information is best managed in a proper database or configuration store.

Many Apache Kafka Connect scenarios will be functional, but these conceptual differences between Apache Kafka's and Azure Event Hubs' retention models may cause certain configurations not to work as expected.

This tutorial walks you through integrating Kafka Connect with an event hub and deploying basic FileStreamSource and FileStreamSink connectors. This feature is currently in preview. While these connectors are not meant for production use, they demonstrate an end-to-end Kafka Connect scenario where Azure Event Hubs acts as a Kafka broker.

## NOTE

This sample is available on [GitHub](#).

In this tutorial, you take the following steps:

- Create an Event Hubs namespace
- Clone the example project
- Configure Kafka Connect for Event Hubs
- Run Kafka Connect
- Create connectors

## Prerequisites

To complete this walkthrough, make sure you have the following prerequisites:

- Azure subscription. If you don't have one, [create a free account](#).
- [Git](#)
- Linux/MacOS
- Kafka release (version 1.1.1, Scala version 2.11), available from [kafka.apache.org](#)
- Read through the [Event Hubs for Apache Kafka](#) introduction article

## Create an Event Hubs namespace

An Event Hubs namespace is required to send and receive from any Event Hubs service. See [Creating an event hub](#) for instructions to create a namespace and an event hub. Get the Event Hubs connection string and fully qualified domain name (FQDN) for later use. For instructions, see [Get an Event Hubs connection string](#).

## Clone the example project

Clone the Azure Event Hubs repository and navigate to the tutorials/connect subfolder:

```
git clone https://github.com/Azure/azure-event-hubs-for-kafka.git
cd azure-event-hubs-for-kafka/tutorials/connect
```

## Configure Kafka Connect for Event Hubs

Minimal reconfiguration is necessary when redirecting Kafka Connect throughput from Kafka to Event Hubs. The following `connect-distributed.properties` sample illustrates how to configure Connect to authenticate and communicate with the Kafka endpoint on Event Hubs:

```
bootstrap.servers={YOUR.EVENTHUBS.FQDN}:9093 # e.g. namespace.servicebus.windows.net:9093
group.id=connect-cluster-group

# connect internal topic names, auto-created if not exists
config.storage.topic=connect-cluster-configs
offset.storage.topic=connect-cluster-offsets
status.storage.topic=connect-cluster-status

# internal topic replication factors - auto 3x replication in Azure Storage
config.storage.replication.factor=1
offset.storage.replication.factor=1
status.storage.replication.factor=1

rest.advertised.host.name=connect
offset.flush.interval.ms=10000

key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter

internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

# required EH Kafka security settings
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";

producer.security.protocol=SASL_SSL
producer.sasl.mechanism=PLAIN
producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";

consumer.security.protocol=SASL_SSL
consumer.sasl.mechanism=PLAIN
consumer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";

plugin.path={KAFKA.DIRECTORY}/libs # path to the libs directory within the Kafka release
```

### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net/;SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXXXXX"
```

## Run Kafka Connect

In this step, a Kafka Connect worker is started locally in distributed mode, using Event Hubs to maintain cluster state.

1. Save the above `connect-distributed.properties` file locally. Be sure to replace all values in braces.
2. Navigate to the location of the Kafka release on your machine.
3. Run `./bin/connect-distributed.sh /PATH/TO/connect-distributed.properties`. The Connect worker REST API is ready for interaction when you see `'INFO Finished starting connectors and tasks'`.

#### NOTE

Kafka Connect uses the Kafka AdminClient API to automatically create topics with recommended configurations, including compaction. A quick check of the namespace in the Azure portal reveals that the Connect worker's internal topics have been created automatically.

Kafka Connect internal topics **must use compaction**. The Event Hubs team is not responsible for fixing improper configurations if internal Connect topics are incorrectly configured.

### Create connectors

This section walks you through spinning up FileStreamSource and FileStreamSink connectors.

1. Create a directory for input and output data files.

```
mkdir ~/connect-quickstart
```

2. Create two files: one file with seed data from which the FileStreamSource connector reads, and another to which our FileStreamSink connector writes.

```
seq 1000 > ~/connect-quickstart/input.txt
touch ~/connect-quickstart/output.txt
```

3. Create a FileStreamSource connector. Be sure to replace the curly braces with your home directory path.

```
curl -s -X POST -H "Content-Type: application/json" --data '{"name": "file-source", "config": {"connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector", "tasks.max": "1", "topic": "connect-quickstart", "file": "{YOUR/HOME/PATH}/connect-quickstart/input.txt"}}'
http://localhost:8083/connectors
```

You should see the Event Hub `connect-quickstart` on your Event Hubs instance after running the above command.

4. Check status of source connector.

```
curl -s http://localhost:8083/connectors/file-source/status
```

Optionally, you can use [Service Bus Explorer](#) to verify that events have arrived in the `connect-quickstart` topic.

5. Create a FileStreamSink Connector. Again, make sure you replace the curly braces with your home directory path.

```
curl -X POST -H "Content-Type: application/json" --data '{"name": "file-sink", "config": {"connector.class": "org.apache.kafka.connect.file.FileStreamSinkConnector", "tasks.max": "1", "topics": "connect-quickstart", "file": "{YOUR/HOME/PATH}/connect-quickstart/output.txt"}}'
http://localhost:8083/connectors
```

6. Check the status of sink connector.

```
curl -s http://localhost:8083/connectors/file-sink/status
```

7. Verify that data has been replicated between files and that the data is identical across both files.

```
# read the file
cat ~/connect-quickstart/output.txt
# diff the input and output files
diff ~/connect-quickstart/input.txt ~/connect-quickstart/output.txt
```

### Cleanup

Kafka Connect creates Event Hub topics to store configurations, offsets, and status that persist even after the Connect cluster has been taken down. Unless this persistence is desired, it is recommended that these topics are deleted. You may also want to delete the `connect-quickstart` Event Hub that were created during the course of

this walkthrough.

## Next steps

To learn more about Event Hubs for Kafka, see the following articles:

- [Mirror a Kafka broker in an event hub](#)
- [Connect Apache Spark to an event hub](#)
- [Connect Apache Flink to an event hub](#)
- [Explore samples on our GitHub](#)
- [Connect Akka Streams to an event hub](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)

# Integrate Apache Kafka Connect support on Azure Event Hubs with Debezium for Change Data Capture

11/26/2021 • 8 minutes to read • [Edit Online](#)

**Change Data Capture (CDC)** is a technique used to track row-level changes in database tables in response to create, update, and delete operations. [Debezium](#) is a distributed platform that builds on top of Change Data Capture features available in different databases (for example, [logical decoding in PostgreSQL](#)). It provides a set of [Kafka Connect connectors](#) that tap into row-level changes in database table(s) and convert them into event streams that are then sent to [Apache Kafka](#).

## WARNING

Use of the Apache Kafka Connect framework as well as the Debezium platform and its connectors are **not eligible for product support through Microsoft Azure**.

Apache Kafka Connect assumes for its dynamic configuration to be held in compacted topics with otherwise unlimited retention. Event Hubs [does not implement compaction as a broker feature](#) and always imposes a time-based retention limit on retained events, rooting from the principle that Event Hubs is a real-time event streaming engine and not a long-term data or configuration store.

While the Apache Kafka project might be comfortable with mixing these roles, Azure believes that such information is best managed in a proper database or configuration store.

Many Apache Kafka Connect scenarios will be functional, but these conceptual differences between Apache Kafka's and Event Hubs' retention models may cause certain configurations not to work as expected.

This tutorial walks you through how to set up a change data capture based system on Azure using [Event Hubs](#) (for Kafka), [Azure DB for PostgreSQL](#) and Debezium. It will use the [Debezium PostgreSQL connector](#) to stream database modifications from PostgreSQL to Kafka topics in Event Hubs

## NOTE

This article contains references to the term *whitelist*, a term that Microsoft no longer uses. When the term is removed from the software, we'll remove it from this article.

In this tutorial, you take the following steps:

- Create an Event Hubs namespace
- Setup and configure Azure Database for PostgreSQL
- Configure and run Kafka Connect with Debezium PostgreSQL connector
- Test change data capture
- (Optional) Consume change data events with a `FileStreamSink` connector

## Pre-requisites

To complete this walk through, you'll require:

- Azure subscription. If you don't have one, [create a free account](#).
- Linux/MacOS
- Kafka release (version 1.1.1, Scala version 2.11), available from [kafka.apache.org](#)
- Read through the [Event Hubs for Apache Kafka](#) introduction article

## Create an Event Hubs namespace

An Event Hubs namespace is required to send and receive from any Event Hubs service. See [Creating an event hub](#) for instructions to create a namespace and an event hub. Get the Event Hubs connection string and fully qualified domain name (FQDN) for later use. For instructions, see [Get an Event Hubs connection string](#).

# Set up and configure Azure Database for PostgreSQL

Azure Database for PostgreSQL is a relational database service based on the community version of open-source PostgreSQL database engine, and is available in two deployment options: Single Server and Hyperscale (Citus). [Follow these instructions](#) to create an Azure Database for PostgreSQL server using the Azure portal.

## Setup and run Kafka Connect

This section will cover the following topics:

- Debezium connector installation
- Configuring Kafka Connect for Event Hubs
- Start Kafka Connect cluster with Debezium connector

### Download and setup Debezium connector

Follow the latest instructions in the [Debezium documentation](#) to download and set up the connector.

- Download the connector's plug-in archive. For example, to download version [1.2.0](#) of the connector, use this link - <https://repo1.maven.org/maven2/io/debezium/debezium-connector-postgres/1.2.0.Final/debezium-connector-postgres-1.2.0.Final-plugin.tar.gz>
- Extract the JAR files and copy them to the [Kafka Connect plugin.path](#).

### Configure Kafka Connect for Event Hubs

Minimal reconfiguration is necessary when redirecting Kafka Connect throughput from Kafka to Event Hubs. The following [connect-distributed.properties](#) sample illustrates how to configure Connect to authenticate and communicate with the Kafka endpoint on Event Hubs:

#### IMPORTANT

- Debezium will auto-create a topic per table and a bunch of metadata topics. Kafka **topic** corresponds to an Event Hubs instance (event hub). For Apache Kafka to Azure Event Hubs mappings, see [Kafka and Event Hubs conceptual mapping](#).
- There are different **limits** on number of event hubs in an Event Hubs namespace depending on the tier (Basic, Standard, Premium, or Dedicated). For these limits, See [Quotas](#).

```

bootstrap.servers={YOUR.EVENTHUBS.FQDN}:9093 # e.g. namespace.servicebus.windows.net:9093
group.id=connect-cluster-group

# connect internal topic names, auto-created if not exists
config.storage.topic=connect-cluster-configs
offset.storage.topic=connect-cluster-offsets
status.storage.topic=connect-cluster-status

# internal topic replication factors - auto 3x replication in Azure Storage
config.storage.replication.factor=1
offset.storage.replication.factor=1
status.storage.replication.factor=1

rest.advertised.host.name=connect
offset.flush.interval.ms=10000

key.converter=org.apache.kafka.connect.json.JsonConverter
value.converter=org.apache.kafka.connect.json.JsonConverter
internal.key.converter=org.apache.kafka.connect.json.JsonConverter
internal.value.converter=org.apache.kafka.connect.json.JsonConverter

internal.key.converter.schemas.enable=false
internal.value.converter.schemas.enable=false

# required EH Kafka security settings
security.protocol=SASL_SSL
sasl.mechanism=PLAIN
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";

producer.security.protocol=SASL_SSL
producer.sasl.mechanism=PLAIN
producer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";

consumer.security.protocol=SASL_SSL
consumer.sasl.mechanism=PLAIN
consumer.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
username="$ConnectionString" password="{YOUR.EVENTHUBS.CONNECTION.STRING}";

plugin.path={KAFKA.DIRECTORY}/libs # path to the libs directory within the Kafka release

```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net;/SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXXX"
```

#### Run Kafka Connect

In this step, a Kafka Connect worker is started locally in distributed mode, using Event Hubs to maintain cluster state.

1. Save the above `connect-distributed.properties` file locally. Be sure to replace all values in braces.
2. Navigate to the location of the Kafka release on your machine.
3. Run `./bin/connect-distributed.sh /PATH/TO/connect-distributed.properties` and wait for the cluster to start.

#### NOTE

Kafka Connect uses the Kafka AdminClient API to automatically create topics with recommended configurations, including compaction. A quick check of the namespace in the Azure portal reveals that the Connect worker's internal topics have been created automatically.

Kafka Connect internal topics **must use compaction**. The Event Hubs team is not responsible for fixing improper configurations if internal Connect topics are incorrectly configured.

#### Configure and start the Debezium PostgreSQL source connector

Create a configuration file (`pg-source-connector.json`) for the PostgreSQL source connector - replace the values as per your Azure PostgreSQL instance.

```
{
  "name": "todo-connector",
  "config": {
    "connector.class": "io.debezium.connector.postgresql.PostgresConnector",
    "database.hostname": "<replace with Azure PostgreSQL instance name>.postgres.database.azure.com",
    "database.port": "5432",
    "database.user": "<replace with database user name>",
    "database.password": "<replace with database password>",
    "database dbname": "postgres",
    "database.server.name": "my-server",
    "plugin.name": "wal2json",
    "table.whitelist": "public.todos"
  }
}
```

#### TIP

`database.server.name` attribute is a logical name that identifies and provides a namespace for the particular PostgreSQL database server/cluster being monitored.. For detailed info, check [Debezium documentation](#)

To create an instance of the connector, use the Kafka Connect REST API endpoint:

```
curl -X POST -H "Content-Type: application/json" --data @pg-source-connector.json
http://localhost:8083/connectors
```

To check the status of the connector:

```
curl -s http://localhost:8083/connectors/todo-connector/status
```

## Test change data capture

To see change data capture in action, you'll need to create/update/delete records in the Azure PostgreSQL database.

Start by connecting to your Azure PostgreSQL database (the example below uses `psql`)

```
psql -h <POSTGRES_INSTANCE_NAME>.postgres.database.azure.com -p 5432 -U <POSTGRES_USER_NAME> -W -d
<POSTGRES_DB_NAME> --set=sslmode=require

e.g.

psql -h my-postgres.postgres.database.azure.com -p 5432 -U testuser@my-postgres -W -d postgres --
set=sslmode=require
```

#### Create a table and insert records

```
CREATE TABLE todos (id SERIAL, description VARCHAR(50), todo_status VARCHAR(12), PRIMARY KEY(id));

INSERT INTO todos (description, todo_status) VALUES ('setup postgresql on azure', 'complete');
INSERT INTO todos (description, todo_status) VALUES ('setup kafka connect', 'complete');
INSERT INTO todos (description, todo_status) VALUES ('configure and install connector', 'in-progress');
INSERT INTO todos (description, todo_status) VALUES ('start connector', 'pending');
```

The connector should now spring into action and send change data events to an Event Hubs topic with the following name `my-server.public.todos`, assuming you have `my-server` as the value for `database.server.name` and `public.todos` is the table whose changes you're tracking (as per `table.whitelist` configuration)

#### Check Event Hubs topic

Let's introspect the contents of the topic to make sure everything is working as expected. The below example uses `kafkacat`, but you can also [create a consumer using any of the options listed here](#)

Create a file named `kafkacat.conf` with the following contents:

```
metadata.broker.list=<enter event hubs namespace>.servicebus.windows.net:9093
security.protocol=SASL_SSL
sasl.mechanisms=PLAIN
sasl.username=$ConnectionString
sasl.password=<enter event hubs connection string>
```

#### NOTE

Update `metadata.broker.list` and `sasl.password` attributes in `kafkacat.conf` as per Event Hubs information.

In a different terminal, start a consumer:

```
export KAFKACAT_CONFIG=kafkacat.conf
export BROKER=<enter event hubs namespace>.servicebus.windows.net:9093
export TOPIC=my-server.public.todos

kafkacat -b $BROKER -t $TOPIC -o beginning
```

You should see the JSON payloads representing the change data events generated in PostgreSQL in response to the rows you had added to the `todos` table. Here's a snippet of the payload:

```
{
  "schema": {...},
  "payload": {
    "before": null,
    "after": {
      "id": 1,
      "description": "setup postgresql on azure",
      "todo_status": "complete"
    },
    "source": {
      "version": "1.2.0.Final",
      "connector": "postgresql",
      "name": "fullfillment",
      "ts_ms": 1593018069944,
      "snapshot": "last",
      "db": "postgres",
      "schema": "public",
      "table": "todos",
      "txId": 602,
      "lsn": 184579736,
      "xmin": null
    },
    "op": "c",
    "ts_ms": 1593018069947,
    "transaction": null
  }
}
```

The event consists of the `payload` along with its `schema` (omitted for brevity). In `payload` section, notice how the create operation (`"op": "c"`) is represented - `"before": null` means that it was a newly `INSERT`ed row, `after` provides values for the columns in the row, `source` provides the PostgreSQL instance metadata from where this event was picked up and so on.

You can try the same with update or delete operations as well and introspect the change data events. For example, to update the task status for `configure and install connector` (assuming its `id` is `3`):

```
UPDATE todos SET todo_status = 'complete' WHERE id = 3;
```

## (Optional) Install FileStreamSink connector

Now that all the `todos` table changes are being captured in Event Hubs topic, you'll use the FileStreamSink connector (that is available by default in Kafka Connect) to consume these events.

Create a configuration file (`file-sink-connector.json`) for the connector - replace the `file` attribute as per your file system

```
{
  "name": "cdc-file-sink",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
    "tasks.max": "1",
    "topics": "my-server.public.todos",
    "file": "<enter full path to file e.g. /Users/foo/todos-cdc.txt>"
  }
}
```

To create the connector and check its status:

```
curl -X POST -H "Content-Type: application/json" --data @file-sink-connector.json
http://localhost:8083/connectors

curl http://localhost:8083/connectors/cdc-file-sink/status
```

Insert/update/delete database records and monitor the records in the configured output sink file:

```
tail -f /Users/foo/todos-cdc.txt
```

## Cleanup

Kafka Connect creates Event Hub topics to store configurations, offsets, and status that persist even after the Connect cluster has been taken down. Unless this persistence is desired, it's recommended that these topics are deleted. You may also want to delete the `my-server.public.todos` Event Hub that were created during this walk through.

## Next steps

To learn more about Event Hubs for Kafka, see the following articles:

- [Mirror a Kafka broker in an event hub](#)
- [Connect Apache Spark to an event hub](#)
- [Connect Apache Flink to an event hub](#)
- [Explore samples on our GitHub](#)
- [Connect Akka Streams to an event hub](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)

# Using Akka Streams with Event Hubs for Apache Kafka

11/26/2021 • 3 minutes to read • [Edit Online](#)

This tutorial shows you how to connect Akka Streams through the Event Hubs support for Apache Kafka without changing your protocol clients or running your own clusters.

In this tutorial, you learn how to:

- Create an Event Hubs namespace
- Clone the example project
- Run Akka Streams producer
- Run Akka Streams consumer

## NOTE

This sample is available on [GitHub](#)

## Prerequisites

To complete this tutorial, make sure you have the following prerequisites:

- Read through the [Event Hubs for Apache Kafka](#) article.
- An Azure subscription. If you do not have one, create a [free account](#) before you begin.
- [Java Development Kit \(JDK\) 1.8+](#)
  - On Ubuntu, run `apt-get install default-jdk` to install the JDK.
  - Be sure to set the JAVA\_HOME environment variable to point to the folder where the JDK is installed.
- [Download](#) and [install](#) a Maven binary archive
  - On Ubuntu, you can run `apt-get install maven` to install Maven.
- [Git](#)
  - On Ubuntu, you can run `sudo apt-get install git` to install Git.

## Create an Event Hubs namespace

An Event Hubs namespace is required to send or receive from any Event Hubs service. See [Create an event hub](#) for detailed information. Make sure to copy the Event Hubs connection string for later use.

## Clone the example project

Now that you have a Event Hubs connection string, clone the Azure Event Hubs for Kafka repository and navigate to the `akka` subfolder:

```
git clone https://github.com/Azure/azure-event-hubs-for-kafka.git
cd azure-event-hubs-for-kafka/tutorials/akka/java
```

## Run Akka Streams producer

Using the provided Akka Streams producer example, send messages to the Event Hubs service.

### Provide an Event Hubs Kafka endpoint

#### Producer application.conf

Update the `bootstrap.servers` and `sasl.jaas.config` values in `producer/src/main/resources/application.conf` to direct the producer to the Event Hubs Kafka endpoint with the correct authentication.

```

akka.kafka.producer {
    #Akka Kafka producer properties can be defined here

    # Properties defined by org.apache.kafka.clients.producer.ProducerConfig
    # can be defined in this configuration section.
    kafka-clients {
        bootstrap.servers="{YOUR.EVENTHUBS.FQDN}:9093"
        sasl.mechanism=PLAIN
        security.protocol=SASL_SSL
        sasl.jaas.config="org.apache.kafka.common.security.plain.PlainLoginModule required
username=\"$ConnectionString\" password=\"{YOUR.EVENTHUBS.CONNECTION.STRING}\";"}
    }
}

```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net;/SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXX"
```

#### Run producer from the command line

To run the producer from the command line, generate the JAR and then run from within Maven (or generate the JAR using Maven, then run in Java by adding the necessary Kafka JAR(s) to the classpath):

```

mvn clean package
mvn exec:java -Dexec.mainClass="AkkaTestProducer"

```

The producer begins sending events to the event hub at topic `test`, and prints the events to stdout.

## Run Akka Streams consumer

Using the provided consumer example, receive messages from the event hub.

#### Provide an Event Hubs Kafka endpoint

##### Consumer application.conf

Update the `bootstrap.servers` and `sasl.jaas.config` values in `consumer/src/main/resources/application.conf` to direct the consumer to the Event Hubs Kafka endpoint with the correct authentication.

```

akka.kafka.consumer {
    #Akka Kafka consumer properties defined here
    wakeup-timeout=60s

    # Properties defined by org.apache.kafka.clients.consumer.ConsumerConfig
    # defined in this configuration section.
    kafka-clients {
        request.timeout.ms=60000
        group.id=akka-example-consumer

        bootstrap.servers="{YOUR.EVENTHUBS.FQDN}:9093"
        sasl.mechanism=PLAIN
        security.protocol=SASL_SSL
        sasl.jaas.config="org.apache.kafka.common.security.plain.PlainLoginModule required
username=\"$ConnectionString\" password=\"{YOUR.EVENTHUBS.CONNECTION.STRING}\";"}
    }
}

```

#### IMPORTANT

Replace `{YOUR.EVENTHUBS.CONNECTION.STRING}` with the connection string for your Event Hubs namespace. For instructions on getting the connection string, see [Get an Event Hubs connection string](#). Here's an example configuration:

```
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required username="$ConnectionString"
password="Endpoint=sb://mynamespace.servicebus.windows.net;/SharedAccessKeyName=RootManageSharedAccessKey;SharedAccessKey=XXXXXXX"
```

#### Run consumer from the command line

To run the consumer from the command line, generate the JAR and then run from within Maven (or generate

the JAR using Maven, then run in Java by adding the necessary Kafka JAR(s) to the classpath):

```
mvn clean package  
mvn exec:java -Dexec.mainClass="AkkaTestConsumer"
```

If the event hub has events (for instance, if your producer is also running), then the consumer begins receiving events from topic `test`.

Check out the [Akka Streams Kafka Guide](#) for more detailed information about Akka Streams.

## Next steps

To learn more about Event Hubs for Kafka, see the following articles:

- [Mirror a Kafka broker in an event hub](#)
- [Connect Apache Spark to an event hub](#)
- [Connect Apache Flink to an event hub](#)
- [Integrate Kafka Connect with an event hub](#)
- [Explore samples on our GitHub](#)
- [Apache Kafka developer guide for Azure Event Hubs](#)

# Monitor Azure Event Hubs

11/26/2021 • 5 minutes to read • [Edit Online](#)

When you have critical applications and business processes relying on Azure resources, you want to monitor those resources for their availability, performance, and operation. This article describes the monitoring data generated by Azure Event Hubs and how to analyze and alert on this data with Azure Monitor.

## What is Azure Monitor?

Azure Event Hubs creates monitoring data using [Azure Monitor](#), which is a full stack monitoring service in Azure. Azure Monitor provides a complete set of features to monitor your Azure resources. It can also monitor resources in other clouds and on-premises.

Start with the article [Monitoring Azure resources with Azure Monitor](#), which describes the following concepts:

- What is Azure Monitor?
- Costs associated with monitoring
- Monitoring data collected in Azure
- Configuring data collection
- Standard tools in Azure for analyzing and alerting on monitoring data

The following sections build on this article by describing the specific data gathered for Azure Event Hubs. These sections also provide examples for configuring data collection and analyzing this data with Azure tools.

### TIP

To understand costs associated with Azure Monitor, see [Usage and estimated costs](#). To understand the time it takes for your data to appear in Azure Monitor, see [Log data ingestion time](#).

## Monitoring data from Azure Event Hubs

Azure Event Hubs collects the same kinds of monitoring data as other Azure resources that are described in [Monitoring data from Azure resources](#).

See [Azure Event Hubs monitoring data reference](#) for a detailed reference of the logs and metrics created by Azure Event Hubs.

## Collection and routing

Platform metrics and the activity log are collected and stored automatically, but can be routed to other locations by using a diagnostic setting.

Resource Logs aren't collected and stored until you create a diagnostic setting and route them to one or more locations.

See [Create diagnostic setting to collect platform logs and metrics in Azure](#) for the detailed process for creating a diagnostic setting using the Azure portal, CLI, or PowerShell. When you create a diagnostic setting, you specify which categories of logs to collect. The categories for Azure Event Hubs are listed in [Azure Event Hubs monitoring data reference](#).

If you use [Azure Storage](#) to store the diagnostic logging information, the information is stored in containers

named **insights-logs-operationlogs** and **insights-metrics-pt1m**. Sample URL for an operation log:

```
https://<Azure Storage account>.blob.core.windows.net/insights-logs-operationlogs/resourceId=/SUBSCRIPTIONS/<Azure subscription ID>/RESOURCEGROUPS/<Resource group name>/PROVIDERS/MICROSOFT.SERVICEBUS/NAMESPACES/<Namespace name>/y=<YEAR>/m=<MONTH-NUMBER>/d=<DAY-NUMBER>/h=<HOUR>/m=<MINUTE>/PT1H.json
```

The URL for a metric log is similar.

If you use **Azure Event Hubs** to store the diagnostic logging information, the information is stored in event hubs named **insights-logs-operationlogs** and **insights-metrics-pt1m**. You can also select your own event hub.

If you use **Log Analytics** to store the diagnostic logging information, the information is stored in tables named **AzureDiagnostics** and **AzureMetrics**.

#### IMPORTANT

Enabling these settings requires additional Azure services (storage account, event hub, or Log Analytics), which may increase your cost. To calculate an estimated cost, visit the [Azure pricing calculator](#).

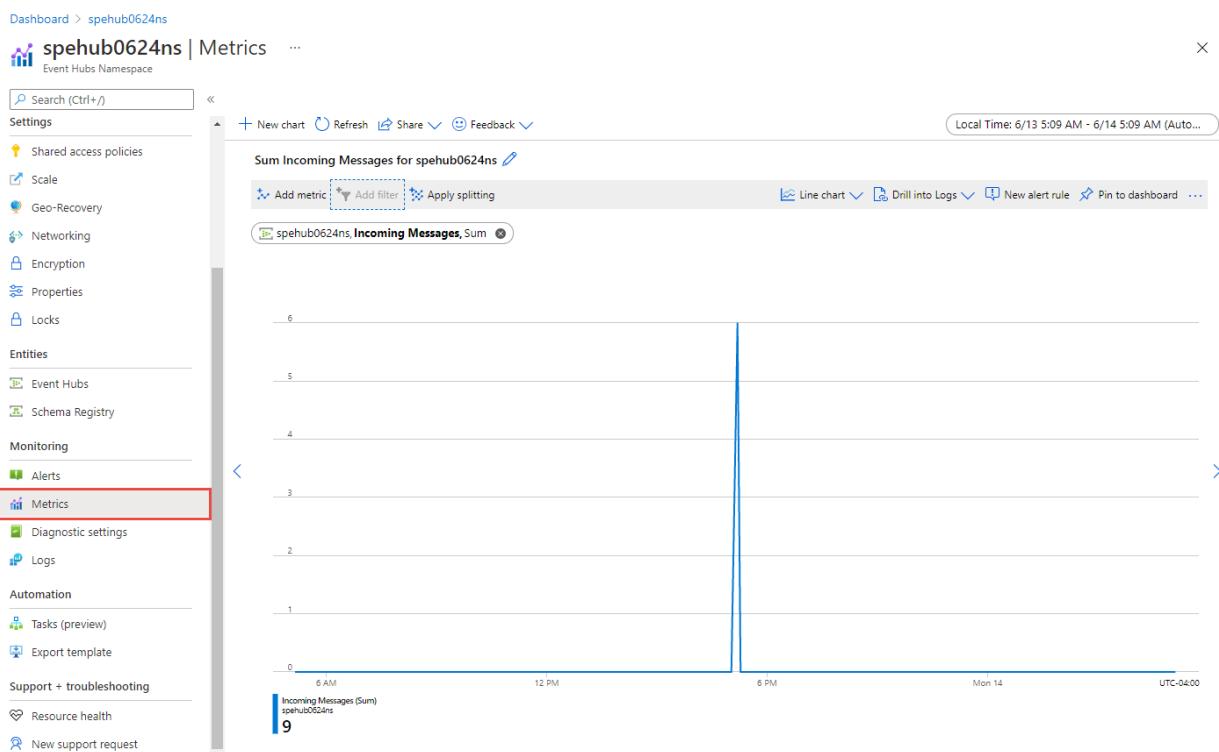
#### NOTE

When you enable metrics in a diagnostic setting, dimension information is not currently included as part of the information sent to a storage account, event hub, or log analytics.

The metrics and logs you can collect are discussed in the following sections.

## Analyzing metrics

You can analyze metrics for Azure Event Hubs, along with metrics from other Azure services, by selecting **Metrics** from the **Azure Monitor** section on the home page for your Event Hubs namespace. See [Getting started with Azure Metrics Explorer](#) for details on using this tool. For a list of the platform metrics collected, see [Monitoring Azure Event Hubs data reference metrics](#).



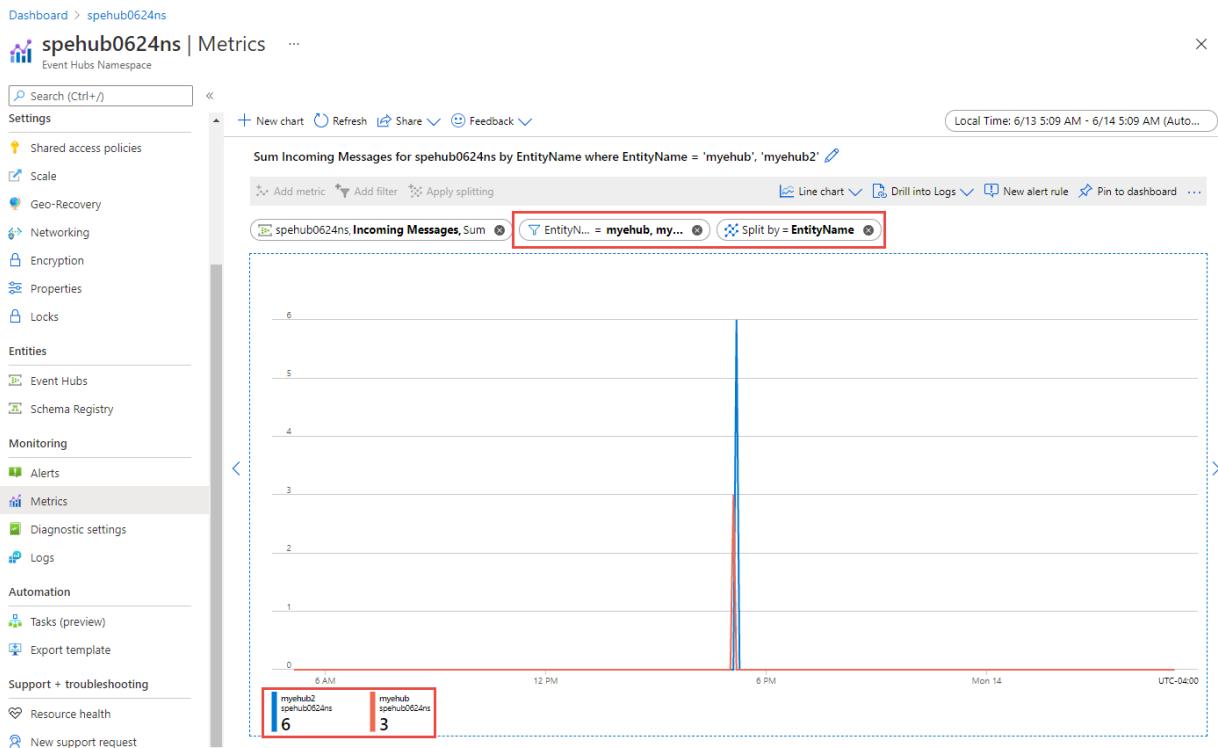
For reference, you can see a list of [all resource metrics supported in Azure Monitor](#).

### TIP

Azure Monitor metrics data is available for 90 days. However, when creating charts only 30 days can be visualized. For example, if you want to visualize a 90 day period, you must break it into three charts of 30 days within the 90 day period.

## Filtering and splitting

For metrics that support dimensions, you can apply filters using a dimension value. For example, add a filter with `EntityName` set to the name of an event hub. You can also split a metric by dimension to visualize how different segments of the metric compare with each other. For more information of filtering and splitting, see [Advanced features of Azure Monitor](#).



## Analyzing logs

Using Azure Monitor Log Analytics requires you to create a diagnostic configuration and enable [Send information to Log Analytics](#). For more information, see the [Collection and routing](#) section. Data in Azure Monitor Logs is stored in tables, with each table having its own set of unique properties. Azure Event Hubs stores data in the following tables: **AzureDiagnostics** and **AzureMetrics**.

### IMPORTANT

When you select **Logs** from the Azure Event Hubs menu, Log Analytics is opened with the query scope set to the current workspace. This means that log queries will only include data from that resource. If you want to run a query that includes data from other databases or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

For a detailed reference of the logs and metrics, see [Azure Event Hubs monitoring data reference](#).

## Sample Kusto queries

## IMPORTANT

When you select **Logs** from the Azure Event Hubs menu, Log Analytics is opened with the query scope set to the current Azure Event Hubs namespace. This means that log queries will only include data from that resource. If you want to run a query that includes data from other workspaces or data from other Azure services, select **Logs** from the **Azure Monitor** menu. See [Log query scope and time range in Azure Monitor Log Analytics](#) for details.

Following are sample queries that you can use to help you monitor your Azure Event Hubs resources:

- Get errors from the past 7 days

```
AzureDiagnostics  
| where TimeGenerated > ago(7d)  
| where ResourceProvider == "MICROSOFT.EVENTHUB"  
| where Category == "OperationalLogs"  
| summarize count() by "EventName"
```

- Get access attempts to a key vault that resulted in "key not found" error.

```
AzureDiagnostics  
| where ResourceProvider == "MICROSOFT.EVENTHUB"  
| where Category == "Error" and OperationName == "wrapkey"  
| project Message
```

- Get operations performed with a key vault to disable or restore the key.

```
AzureDiagnostics  
| where ResourceProvider == "MICROSOFT.EVENTHUB"  
| where Category == "info" and OperationName == "disable" or OperationName == "restore"  
| project Message
```

- Get capture failures and their duration in seconds

```
AzureDiagnostics  
| where ResourceProvider == "MICROSOFT.EVENTHUB"  
| where Category == "ArchiveLogs"  
| summarize count() by "failures", "durationInSeconds"
```

## Alerts

You can access alerts for Azure Event Hubs by selecting **Alerts** from the **Azure Monitor** section on the home page for your Event Hubs namespace. See [Create, view, and manage metric alerts using Azure Monitor](#) for details on creating alerts.

## Next steps

- For a reference of the logs and metrics, see [Monitoring Azure Event Hubs data reference](#).
- For details on monitoring Azure resources, see [Monitoring Azure resources with Azure Monitor](#).

# Send data from Windows Azure diagnostics extension to Azure Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

Azure diagnostics extension is an agent in Azure Monitor that collects monitoring data from the guest operating system and workloads of Azure virtual machines and other compute resources. This article describes how to send data from the Windows Azure Diagnostic extension (WAD) to [Azure Event Hubs](#) so you can forward to locations outside of Azure.

## Supported data

The data collected from the guest operating system that can be sent to Event Hubs includes the following. Other data sources collected by WAD, including IIS Logs and crash dumps, cannot be sent to Event Hubs.

- Event Tracing for Windows (ETW) events
- Performance counters
- Windows event logs, including application logs in the Windows event log
- Azure Diagnostics infrastructure logs

## Prerequisites

- Windows diagnostics extension 1.6 or higher. See [Azure Diagnostics extension configuration schema versions and history](#) for a version history and [Azure Diagnostics extension overview](#) for supported resources.
- Event Hubs namespace must always be provisioned. See [Get started with Event Hubs](#) for details.

## Configuration schema

See [Install and configure Windows Azure diagnostics extension \(WAD\)](#) for different options for enabling and configuring the diagnostics extension and [Azure Diagnostics configuration schema](#) for a reference of the configuration schema. The rest of this article will describe how to use this configuration to send data to an event hub.

Azure Diagnostics always sends logs and metrics to an Azure Storage account. You can configure one or more *data sinks* that send data to additional locations. Each sink is defined in the [SinksConfig element](#) of the public configuration with sensitive information in the private configuration. This configuration for event hubs uses the values in the following table.

PROPERTY	DESCRIPTION
Name	Descriptive name for the sink. Used in the configuration to specify which data sources to send to the sink.
Url	Url of the event hub in the form <event-hubs-namespace>.servicebus.windows.net/<event-hub-name>.
SharedAccessKeyName	Name of a shared access policy for the event hub that has at least <b>Send</b> authority.

PROPERTY	DESCRIPTION
SharedAccessKey	Primary or secondary key from the shared access policy for the event hub.

Example public and private configurations are shown below. This is a minimal configuration with a single performance counter and event log to illustrate how to configure and use the event hub data sink. See [Azure Diagnostics configuration schema](#) for a more complex example.

## Public configuration

```
{
  "WadCfg": {
    "DiagnosticMonitorConfiguration": {
      "overallQuotaInMB": 5120,
      "PerformanceCounters": {
        "scheduledTransferPeriod": "PT1M",
        "sinks": "myEventHub",
        "PerformanceCounterConfiguration": [
          {
            "counterSpecifier": "\Processor(_Total)\% Processor Time",
            "sampleRate": "PT3M"
          }
        ]
      },
      "WindowsEventLog": {
        "scheduledTransferPeriod": "PT1M",
        "sinks": "myEventHub",
        "DataSource": [
          {
            "name": "Application!*[System[(Level=1 or Level=2 or Level=3)]]"
          }
        ]
      }
    },
    "SinksConfig": {
      "Sink": [
        {
          "name": "myEventHub",
          "EventHub": {
            "Url": "https://diags-mycompany-ns.servicebus.windows.net/diageventhub",
            "SharedAccessKeyName": "SendRule"
          }
        }
      ]
    },
    "StorageAccount": "mystorageaccount",
  }
}
```

## Private configuration

```
{
  "storageAccountName": "mystorageaccount",
  "storageAccountKey": "{base64 encoded key}",
  "storageAccountEndPoint": "https://core.windows.net",
  "EventHub": {
    "Url": "https://diags-mycompany-ns.servicebus.windows.net/diageventhub",
    "SharedAccessKeyName": "SendRule",
    "SharedAccessKey": "{base64 encoded key}"
  }
}
```

## Configuration options

To send data to a data sink, you specify the `sinks` attribute on the data source's node. Where you place the `sinks` attribute determines the scope of the assignment. In the following example, the `sinks` attribute is defined to the `PerformanceCounters` node which will cause all child performance counters to be sent to the event hub.

```
"PerformanceCounters": {  
    "scheduledTransferPeriod": "PT1M",  
    "sinks": "MyEventHub",  
    "PerformanceCounterConfiguration": [  
        {  
            "counterSpecifier": "\Processor(_Total)\% Processor Time",  
            "sampleRate": "PT3M"  
        },  
        {  
            "counterSpecifier": "\Memory\Available MBytes",  
            "sampleRate": "PT3M"  
        },  
        {  
            "counterSpecifier": "\Web Service(_Total)\ISAPI Extension Requests/sec",  
            "sampleRate": "PT3M"  
        }  
    ]  
}
```

In the following example, the `sinks` attribute is applied directly to three counters which will cause only those performance counters to be sent to the event hub.

```
"PerformanceCounters": {  
    "scheduledTransferPeriod": "PT1M",  
    "PerformanceCounterConfiguration": [  
        {  
            "counterSpecifier": "\Processor(_Total)\% Processor Time",  
            "sampleRate": "PT3M",  
            "sinks": "MyEventHub"  
        },  
        {  
            "counterSpecifier": "\Memory\Available MBytes",  
            "sampleRate": "PT3M"  
        },  
        {  
            "counterSpecifier": "\Web Service(_Total)\ISAPI Extension Requests/sec",  
            "sampleRate": "PT3M"  
        },  
        {  
            "counterSpecifier": "\ASP.NET\Requests Rejected",  
            "sampleRate": "PT3M",  
            "sinks": "MyEventHub"  
        },  
        {  
            "counterSpecifier": "\ASP.NET\Requests Queued",  
            "sampleRate": "PT3M",  
            "sinks": "MyEventHub"  
        }  
    ]  
}
```

## Validating configuration

You can use a variety of methods to validate that data is being sent to the event hub. One straightforward method is to use Event Hubs capture as described in [Capture events through Azure Event Hubs in Azure Blob](#)

Storage or Azure Data Lake Storage.

## Troubleshoot Event Hubs sinks

- Look at the Azure Storage table **WADDiagnosticInfrastructureLogsTable** which contains logs and errors for Azure Diagnostics itself. One option is to use a tool such as [Azure Storage Explorer](#) to connect to this storage account, view this table, and add a query for TimeStamp in the last 24 hours. You can use the tool to export a .csv file and open it in an application such as Microsoft Excel. Excel makes it easy to search for calling-card strings, such as **EventHubs**, to see what error is reported.
- Check that your event hub is successfully provisioned. All connection info in the **PrivateConfig** section of the configuration must match the values of your resource as seen in the portal. Make sure that you have a SAS policy defined (*SendRule* in the example) in the portal and that *Send* permission is granted.

## Next steps

- [Event Hubs overview](#)
- [Create an event hub](#)
- [Event Hubs FAQ](#)

# Automatically scale up Azure Event Hubs throughput units (standard tier)

11/26/2021 • 3 minutes to read • [Edit Online](#)

Azure Event Hubs is a highly scalable data streaming platform. As such, Event Hubs usage often increases after starting to use the service. Such usage requires increasing the predetermined [throughput units \(TUs\)](#) to scale Event Hubs and handle larger transfer rates. The **Auto-inflate** feature of Event Hubs automatically scales up by increasing the number of TUs, to meet usage needs. Increasing TUs prevents throttling scenarios, in which:

- Data ingress rates exceed set TUs
- Data egress request rates exceed set TUs

The Event Hubs service increases the throughput when load increases beyond the minimum threshold, without any requests failing with ServerBusy errors.

## NOTE

To learn more about the [premium](#) tier, see [Event Hubs Premium](#).

## How Auto-inflate works in standard tier

Event Hubs traffic is controlled by TUs (standard tier). For the limits such as ingress and egress rates per TU, see [Event Hubs quotas and limits](#). Auto-inflate enables you to start small with the minimum required TUs you choose. The feature then scales automatically to the maximum limit of TUs you need, depending on the increase in your traffic. Auto-inflate provides the following benefits:

- An efficient scaling mechanism to start small and scale up as you grow.
- Automatically scale to the specified upper limit without throttling issues.
- More control over scaling, because you control when and how much to scale.

## NOTE

Auto-inflate does not *automatically* scale down the number of TUs when ingress or egress rates drop below the limits.

## Enable Auto-inflate on a namespace

You can enable or disable Auto-inflate on a standard tier Event Hubs namespace by using either [Azure portal](#) or an [Azure Resource Manager template](#).

For a premium Event Hubs namespace, the feature is automatically enabled. You can't disable it.

## NOTE

Basic tier Event Hubs namespaces do not support Auto-inflate.

## Use Azure portal

In the Azure portal, you can enable the feature when creating a standard Event Hubs namespace or after the

namespace is created. You can also set TUs for the namespace and specify maximum limit of TUs

You can enable the Auto-inflate feature **when creating an Event Hubs namespace**. The follow image shows you how to enable the auto-inflate feature for a standard tier namespace and configure TUs to start with and the maximum number of TUs.

The screenshot shows the 'Create Namespace' wizard in the 'Event Hubs' section, specifically the 'Basics' step. The 'PROJECT DETAILS' section includes fields for 'Subscription' (Visual Studio Ultimate with MSDN) and 'Resource group' ((New) spehubrg). The 'INSTANCE DETAILS' section includes fields for 'Namespace name' (spehubstandardns.servicebus.windows.net), 'Location' (East US), and 'Pricing tier' (Standard (20 Consumer groups, 1000 Brokered connections)). The 'Throughput Units' field is highlighted with a red border, showing a slider at 3 and a value of 20 in the input field. The 'Enable Auto-Inflate' checkbox is checked. At the bottom, there are navigation buttons: 'Review + create' (highlighted in blue), '< Previous', and 'Next: Tags >'.

With this option enabled, you can start small with your TUs and scale up as your usage needs increase. The upper limit for inflation doesn't immediately affect pricing, which depends on the number of TUs used per hour.

To enable the Auto-inflate feature and modify its settings for an existing, follow these steps:

1. On the **Event Hubs Namespace** page, select **Scale** under **Settings** on the left menu.
2. In the **Scale Settings** page, select the checkbox for **Enable** (if the autoscale feature wasn't enabled).

The screenshot shows the Azure portal interface for managing an Event Hubs Namespace. The left sidebar lists various settings: Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Settings (Shared access policies, Scale, Geo-Recovery, Networking). The 'Scale' option is highlighted with a red box. The main content area shows 'THROUGHPUT UNITS' with a slider set to 3 and a note about controlling capacity. Below it is the 'AUTO-INFLATE' section, which is enabled (checkbox checked) and has a slider set to 20, with a note indicating the range is min. 3, max. 20.

3. Enter the **maximum** number of throughput units or use the scrollbar to set the value.
4. (optional) Update the **minimum** number of throughput units at the top of this page.

#### NOTE

When you apply the auto-inflate configuration to increase throughput units, the Event Hubs service emits diagnostic logs that give you information about why and when the throughput increased. To enable diagnostic logging for an event hub, select **Diagnostic settings** on the left menu on the Event Hub page in the Azure portal. For more information, see [Set up diagnostic logs for an Azure event hub](#).

## Use an Azure Resource Manager template

You can enable Auto-inflate during an Azure Resource Manager template deployment. For example, set the `isAutoInflateEnabled` property to `true` and set `maximumThroughputUnits` to 10. For example:

```
"resources": [
  {
    "apiVersion": "2017-04-01",
    "name": "[parameters('namespaceName')]",
    "type": "Microsoft.EventHub/Namespaces",
    "location": "[variables('location')]",
    "sku": {
      "name": "Standard",
      "tier": "Standard"
    },
    "properties": {
      "isAutoInflateEnabled": true,
      "maximumThroughputUnits": 10
    },
    "resources": [
      {
        "apiVersion": "2017-04-01",
        "name": "[parameters('eventHubName')]",
        "type": "EventHubs",
        "dependsOn": [
          "[concat('Microsoft.EventHub/namespaces/', parameters('namespaceName'))]"
        ],
        "properties": {},
        "resources": [
          {
            "apiVersion": "2017-04-01",
            "name": "[parameters('consumerGroupName')]",
            "type": "ConsumerGroups",
            "dependsOn": [
              "[parameters('eventHubName')]"
            ],
            "properties": {}
          }
        ]
      }
    ]
  }
]
```

For the complete template, see the [Create Event Hubs namespace and enable inflate](#) template on GitHub.

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)

# Event Hubs management libraries

11/26/2021 • 2 minutes to read • [Edit Online](#)

You can use the Azure Event Hubs management libraries to dynamically provision Event Hubs namespaces and entities. This dynamic nature enables complex deployments and messaging scenarios, so that you can programmatically determine what entities to provision. These libraries are currently available for .NET.

## Supported functionality

- Namespace creation, update, deletion
- Event Hubs creation, update, deletion
- Consumer Group creation, update, deletion

## Prerequisites

To get started using the Event Hubs management libraries, you must authenticate with Azure Active Directory (AAD). AAD requires that you authenticate as a service principal, which provides access to your Azure resources. For information about creating a service principal, see one of these articles:

- [Use the Azure portal to create Active Directory application and service principal that can access resources](#)
- [Use Azure PowerShell to create a service principal to access resources](#)
- [Use Azure CLI to create a service principal to access resources](#)

These tutorials provide you with an `AppId` (Client ID), `TenantId`, and `ClientSecret` (authentication key), all of which are used for authentication by the management libraries. You must have **Owner** permissions for the resource group on which you want to run.

## Programming pattern

The pattern to manipulate any Event Hubs resource follows a common protocol:

1. Obtain a token from AAD using the `Microsoft.IdentityModel.Clients.ActiveDirectory` library.

```
var context = new AuthenticationContext($"https://login.microsoftonline.com/{tenantId}");

var result = await context.AcquireTokenAsync(
    "https://management.core.windows.net/",
    new ClientCredential(clientId, clientSecret)
);
```

2. Create the `EventHubManagementClient` object.

```
var creds = new TokenCredentials(token);
var ehClient = new EventHubManagementClient(creds)
{
    SubscriptionId = SettingsCache["SubscriptionId"]
};
```

3. Set the `CreateOrUpdate` parameters to your specified values.

```
var ehParams = new EventHubCreateOrUpdateParameters()
{
    Location = SettingsCache["DataCenterLocation"]
};
```

4. Execute the call.

```
await ehClient.EventHubs.CreateOrUpdateAsync(resourceGroupName, namespaceName, EventHubName,
ehParams);
```

## Next steps

- [.NET Management sample](#)
- [Microsoft.Azure.Management.EventHub Reference](#)

# Allow access to Azure Event Hubs namespaces from specific IP addresses or ranges

11/26/2021 • 6 minutes to read • [Edit Online](#)

By default, Event Hubs namespaces are accessible from internet as long as the request comes with valid authentication and authorization. With IP firewall, you can restrict it further to only a set of IPv4 addresses or IPv4 address ranges in [CIDR \(Classless Inter-Domain Routing\)](#) notation.

This feature is helpful in scenarios in which Azure Event Hubs should be only accessible from certain well-known sites. Firewall rules enable you to configure rules to accept traffic originating from specific IPv4 addresses. For example, if you use Event Hubs with [Azure Express Route](#), you can create a **firewall rule** to allow traffic from only your on-premises infrastructure IP addresses.

## IP firewall rules

The IP firewall rules are applied at the Event Hubs namespace level. So, the rules apply to all connections from clients using any supported protocol. Any connection attempt from an IP address that doesn't match an allowed IP rule on the Event Hubs namespace is rejected as unauthorized. The response doesn't mention the IP rule. IP filter rules are applied in order, and the first rule that matches the IP address determines the accept or reject action.

## Important points

- This feature isn't supported in the **basic** tier.
- Turning on firewall rules for your Event Hubs namespace blocks incoming requests by default, unless requests originate from a service operating from allowed public IP addresses. Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on. As an exception, you can allow access to Event Hubs resources from certain **trusted services** even when the IP filtering is enabled. For a list of trusted services, see [Trusted Microsoft services](#).
- Specify **at least one IP firewall rule or virtual network rule** for the namespace to allow traffic only from the specified IP addresses or subnet of a virtual network. If there are no IP and virtual network rules, the namespace can be accessed over the public internet (using the access key).

## Use Azure portal

This section shows you how to use the Azure portal to create IP firewall rules for an Event Hubs namespace.

1. Navigate to your **Event Hubs namespace** in the [Azure portal](#).
2. Select **Networking** under **Settings** on the left menu.

### WARNING

If you select the **Selected networks** option and don't add at least one IP firewall rule or a virtual network on this page, the namespace can be accessed via **public internet** (using the access key).

The screenshot shows the Azure portal interface for the 'Networking' settings of an Event Hubs Namespace named 'contosoehubsns'. On the left, a navigation menu lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Networking (which is highlighted with a red box). The main content area is titled 'Firewalls and virtual networks' and includes sections for 'Allow access from' (with 'Selected networks' selected), 'Virtual networks' (with options to add existing or create new), and 'Firewall' (with a note about bypassing for Microsoft services). A warning message at the bottom states: 'This can cause a disruption if this event hub is used by a trusted service and the service's network is not explicitly added.'

If you select the **All networks** option, the event hub accepts connections from any IP address (using the access key). This setting is equivalent to a rule that accepts the 0.0.0.0/0 IP address range.

The screenshot shows the Azure portal interface for the 'Networking' settings of an Event Hubs Namespace named 'DocsDemoNamespaceEH'. The left navigation menu shows various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Networking (which is highlighted with a red box). The main content area is titled 'Firewalls and virtual networks' and includes sections for 'Allow access from' (with 'All networks' selected) and a note stating 'All networks, including the internet, can access this Event Hubs Namespace.'

3. To restrict access to specific IP addresses, confirm that the **Selected networks** option is selected. In the **Firewall** section, follow these steps:
  - a. Select **Add your client IP address** option to give your current client IP the access to the namespace.
  - b. For **address range**, enter a specific IPv4 address or a range of IPv4 address in CIDR notation.

## WARNING

If you select the **Selected networks** option and don't add at least one IP firewall rule or a virtual network on this page, the namespace can be accessed over public internet (using the access key).

4. Specify whether you want to **allow trusted Microsoft services to bypass this firewall**. See [Trusted Microsoft services](#) for details.

5. Select **Save** on the toolbar to save the settings. Wait for a few minutes for the confirmation to show up on the portal notifications.

## NOTE

To restrict access to specific virtual networks, see [Allow access from specific networks](#).

## Trusted Microsoft services

When you enable the **Allow trusted Microsoft services to bypass this firewall** setting, the following services are granted access to your Event Hubs resources.

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure Event Grid	<p>Allows Azure Event Grid to send events to event hubs in your Event Hubs namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"><li>• Enable system-assigned identity for a topic or a domain</li><li>• Add the identity to the Azure Event Hubs Data Sender role on the Event Hubs namespace</li><li>• Then, configure the event subscription that uses an event hub as an endpoint to use the system-assigned identity.</li></ul> <p>For more information, see <a href="#">Event delivery with a managed identity</a></p>

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure Monitor (Diagnostic Settings and Action Groups)	Allows Azure Monitor to send diagnostic information and alert notifications to event hubs in your Event Hubs namespace. Azure Monitor can read from the event hub and also write data to the event hub.
Azure Stream Analytics	<p>Allows an Azure Stream Analytics job to read data from (<a href="#">input</a>) or write data to (<a href="#">output</a>) event hubs in your Event Hubs namespace.</p> <p><b>Important:</b> The Stream Analytics job should be configured to use a <b>managed identity</b> to access the event hub. For more information, see <a href="#">Use managed identities to access Event Hub from an Azure Stream Analytics job (Preview)</a>.</p>
Azure IoT Hub	<p>Allows IoT Hub to send messages to event hubs in your Event Hub namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"> <li>• Enable system-assigned identity for your IoT hub</li> <li>• Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace.</li> <li>• Then, configure the IoT Hub that uses an event hub as a custom endpoint to use the identity-based authentication.</li> </ul>
Azure API Management	<p>The API Management service allows you to send events to an event hub in your Event Hubs namespace.</p> <ul style="list-style-type: none"> <li>• You can trigger custom workflows by sending events to your event hub when an API is invoked by using the <a href="#">send-request policy</a>.</li> <li>• You can also treat an event hub as your backend in an API. For a sample policy, see <a href="#">Authenticate using a managed identity to access an event hub</a>. You also need to do the following steps:           <ol style="list-style-type: none"> <li>1. Enable system-assigned identity on the API Management instance. For instructions, see <a href="#">Use managed identities in Azure API Management</a>.</li> <li>2. Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace</li> </ol> </li> </ul>
Azure IoT Central	<p>Allows IoT Central to export data to event hubs in your Event Hub namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"> <li>• Enable system-assigned identity for your IoT Central application.</li> <li>• Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace.</li> <li>• Then, configure the Event Hubs <a href="#">export destination on your IoT Central application</a> to use identity-based authentication.</li> </ul>

## Use Resource Manager template

**IMPORTANT**

The Firewall feature isn't supported in the basic tier.

The following Resource Manager template enables adding an IP filter rule to an existing Event Hubs namespace.

**ipMask** in the template is a single IPv4 address or a block of IP addresses in CIDR notation. For example, in CIDR notation 70.37.104.0/24 represents the 256 IPv4 addresses from 70.37.104.0 to 70.37.104.255, with 24 indicating the number of significant prefix bits for the range.

When adding virtual network or firewalls rules, set the value of `defaultAction` to `Deny`.

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "eventhubNamespaceName": {
            "type": "string",
            "metadata": {
                "description": "Name of the Event Hubs namespace"
            }
        },
        "location": {
            "type": "string",
            "metadata": {
                "description": "Location for Namespace"
            }
        }
    },
    "variables": {
        "namespaceNetworkRuleSetName": "[concat(parameters('eventhubNamespaceName'), concat('/', 'default'))]",
    },
    "resources": [
        {
            "apiVersion": "2018-01-01-preview",
            "name": "[parameters('eventhubNamespaceName')]",
            "type": "Microsoft.EventHub/namespaces",
            "location": "[parameters('location')]",
            "sku": {
                "name": "Standard",
                "tier": "Standard"
            },
            "properties": { }
        },
        {
            "apiVersion": "2018-01-01-preview",
            "name": "[variables('namespaceNetworkRuleSetName')]",
            "type": "Microsoft.EventHub/namespaces/networkrulesets",
            "dependsOn": [
                "[concat('Microsoft.EventHub/namespaces/', parameters('eventhubNamespaceName'))]"
            ],
            "properties": {
                "virtualNetworkRules": [<YOUR EXISTING VIRTUAL NETWORK RULES>],
                "ipRules": [
                    {
                        "ipMask": "10.1.1.1",
                        "action": "Allow"
                    },
                    {
                        "ipMask": "11.0.0.0/24",
                        "action": "Allow"
                    }
                ],
                "trustedServiceAccessEnabled": false,
                "defaultAction": "Deny"
            }
        }
    ],
    "outputs": { }
}
```

To deploy the template, follow the instructions for [Azure Resource Manager](#).

**IMPORTANT**

If there are no IP and virtual network rules, all the traffic flows into the namespace even if you set the `defaultAction` to `deny`. The namespace can be accessed over the public internet (using the access key). Specify at least one IP rule or virtual network rule for the namespace to allow traffic only from the specified IP addresses or subnet of a virtual network.

## Next steps

For constraining access to Event Hubs to Azure virtual networks, see the following link:

- [Virtual Network Service Endpoints for Event Hubs](#)

# Allow access to Azure Event Hubs namespaces from specific virtual networks

11/26/2021 • 9 minutes to read • [Edit Online](#)

The integration of Event Hubs with [Virtual Network \(VNet\) Service Endpoints](#) enables secure access to messaging capabilities from workloads such as virtual machines that are bound to virtual networks, with the network traffic path being secured on both ends.

Once configured to bind to at least one virtual network subnet service endpoint, the respective Event Hubs namespace no longer accepts traffic from anywhere but authorized subnets in virtual networks. From the virtual network perspective, binding an Event Hubs namespace to a service endpoint configures an isolated networking tunnel from the virtual network subnet to the messaging service.

The result is a private and isolated relationship between the workloads bound to the subnet and the respective Event Hubs namespace, in spite of the observable network address of the messaging service endpoint being in a public IP range. There's an exception to this behavior. Enabling a service endpoint, by default, enables the `denyAll` rule in the [IP firewall](#) associated with the virtual network. You can add specific IP addresses in the IP firewall to enable access to the Event Hub public endpoint.

## Important points

- This feature isn't supported in the **basic** tier.
- Enabling virtual networks for your Event Hubs namespace blocks incoming requests by default, unless requests originate from a service operating from allowed virtual networks. Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on. As an exception, you can allow access to Event Hubs resources from certain **trusted services** even when virtual networks are enabled. For a list of trusted services, see [Trusted services](#).
- Specify **at least one IP rule or virtual network rule** for the namespace to allow traffic only from the specified IP addresses or subnet of a virtual network. If there are no IP and virtual network rules, the namespace can be accessed over the public internet (using the access key).

## Advanced security scenarios enabled by VNet integration

Solutions that require tight and compartmentalized security, and where virtual network subnets provide the segmentation between the compartmentalized services, still need communication paths between services residing in those compartments.

Any immediate IP route between the compartments, including those carrying HTTPS over TCP/IP, carries the risk of exploitation of vulnerabilities from the network layer on up. Messaging services provide insulated communication paths, where messages are even written to disk as they transition between parties. Workloads in two distinct virtual networks that are both bound to the same Event Hubs instance can communicate efficiently and reliably via messages, while the respective network isolation boundary integrity is preserved.

That means your security sensitive cloud solutions not only gain access to Azure industry-leading reliable and scalable asynchronous messaging capabilities, but they can now use messaging to create communication paths between secure solution compartments that are inherently more secure than what is achievable with any peer-to-peer communication mode, including HTTPS and other TLS-secured socket protocols.

## Bind event hubs to virtual networks

**Virtual network rules** are the firewall security feature that controls whether your Azure Event Hubs namespace accepts connections from a particular virtual network subnet.

Binding an Event Hubs namespace to a virtual network is a two-step process. You first need to create a **virtual Network service endpoint** on a virtual network's subnet and enable it for **Microsoft.EventHub** as explained in the [service endpoint overview](#) article. Once you've added the service endpoint, you bind the Event Hubs namespace to it with a **virtual network rule**.

The virtual network rule is an association of the Event Hubs namespace with a virtual network subnet. While the rule exists, all workloads bound to the subnet are granted access to the Event Hubs namespace. Event Hubs itself never establishes outbound connections, doesn't need to gain access, and is therefore never granted access to your subnet by enabling this rule.

## Use Azure portal

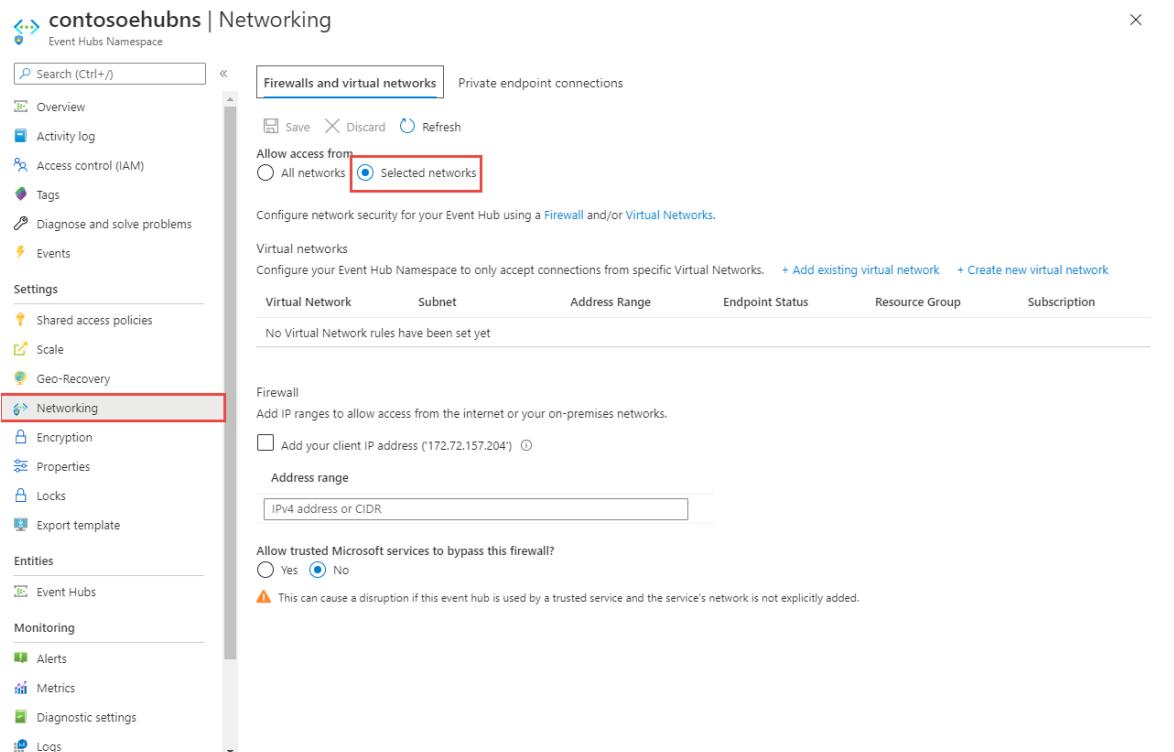
This section shows you how to use Azure portal to add a virtual network service endpoint. To limit access, you need to integrate the virtual network service endpoint for this Event Hubs namespace.

1. Navigate to your **Event Hubs namespace** in the [Azure portal](#).

2. Select **Networking** under **Settings** on the left menu.

### WARNING

If you select the **Selected networks** option and don't add at least one IP firewall rule or a virtual network on this page, the namespace can be accessed via **public internet** (using the access key).



The screenshot shows the Azure portal interface for managing the networking settings of an Event Hubs namespace named "contosoehubns". The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, Shared access policies, Scale, Geo-Recovery, and Networking. The Networking option is highlighted with a red box. The main content area is titled "Firewalls and virtual networks". It shows a dropdown menu "Allow access from" with two options: "All networks" (unchecked) and "Selected networks" (checked). Below this, there are sections for "Virtual networks" and "Firewall". The Firewall section contains fields for adding a client IP address ("Add your client IP address (172.72.157.204)") and defining an address range ("Address range" with a text input field "IPv4 address or CIDR"). At the bottom of the Firewall section, there is a note about allowing trusted Microsoft services to bypass the firewall, with radio buttons for "Yes" and "No".

If you select the **All networks** option, the event hub accepts connections from any IP address (using the access key). This setting is equivalent to a rule that accepts the 0.0.0.0/0 IP address range.

**DocsDemoNamespaceEH | Networking**

**Firewalls and virtual networks** Private endpoint connections (preview)

Allow access from  
 All networks  Selected networks

All networks, including the internet, can access this Event Hubs Namespace.

**Networking**

- Encryption
- Properties
- Locks
- Export template

- To restrict access to specific networks, select the **Selected Networks** option at the top of the page if it isn't already selected.
- In the **Virtual Network** section of the page, select **+Add existing virtual network\***. Select **+ Create new virtual network** if you want to create a new VNet.

**Firewalls and virtual networks** Private endpoint connections (preview)

Allow access from  
 All networks  Selected networks

Configure network security for your Event Hub using a [Firewall](#) and/or [Virtual Networks](#).

**Virtual networks**  
Configure your Event Hub Namespace to only accept connections from specific Virtual Networks:

Virtual Network	Subnet	Address Range	Endpoint Status	Resource Group	Subscription
No Virtual Network rules have been set yet					

**Firewall**  
Add IP ranges to allow access from the internet or your on-premises networks.

Add your client IP address (167.220.2.64)

**Address range**  
IPv4 address or CIDR

### WARNING

If you select the **Selected networks** option and don't add at least one IP firewall rule or a virtual network on this page, the namespace can be accessed over public internet (using the access key).

- Select the virtual network from the list of virtual networks, and then pick the **subnet**. You have to enable the service endpoint before adding the virtual network to the list. If the service endpoint isn't enabled, the portal will prompt you to enable it.

**Add networks**

**Subscription \***  
MDP\_492270

**Virtual networks \***  
docsdemovnet

**Subnets \***  
default (Service endpoint required)

**Info:** The following networks don't have service endpoints enabled for 'Microsoft.EventHub'. Enabling access will take up to 15 minutes to complete. After starting this operation, it is safe to leave and return later if you do not wish to wait.

Do not configure 'Microsoft.EventHub' service endpoint(s) at this time ⓘ

Virtual network	Service endpoint status	...
docsdemovnet	Not enabled	...

**Enable**

6. You should see the following successful message after the service endpoint for the subnet is enabled for **Microsoft.EventHub**. Select **Add** at the bottom of the page to add the network.

Add networks

**Subscription \***  
MDP\_492270

**Virtual networks \***  
docsdemovnet

**Subnets \***  
default

Enabling selected networks with service endpoints for 'Microsoft.EventHub' is complete.

Do not configure 'Microsoft.EventHub' service endpoint(s) at this time ⓘ

Virtual network	Service endpoint status
docsdemovnet	***
default	✓ Enabled

**Add**

#### NOTE

If you are unable to enable the service endpoint, you may ignore the missing virtual network service endpoint using the Resource Manager template. This functionality is not available on the portal.

7. Specify whether you want to **allow trusted Microsoft services to bypass this firewall**. See [Trusted Microsoft services](#) for details.

8. Select **Save** on the toolbar to save the settings. Wait for a few minutes for the confirmation to show up on the portal notifications.

Home > DocsDemoNamespaceEHub | Overview > DocsDemoNamespaceEHub | Networking

DocsDemoNamespaceEHub | Networking

Firewalls and virtual networks Private endpoint connections (preview)

**Save** **Discard** **Refresh**

Allow access from:

All networks  Selected networks

Configure network security for your Event Hub using a [Firewall](#) and/or [Virtual Networks](#).

Virtual networks

Configure your Event Hub Namespace to only accept connections from specific Virtual Networks. [+ Add existing virtual network](#) [+ Create new virtual network](#)

Virtual Network	Subnet	Address Range	Endpoint Status	Resource Group
docsdemovnet	1	172.27.0.0/16	docsdemorg	

Firewall

Add IP ranges to allow access from the internet or your on-premises networks.

Add your client IP address ('172.72.157.204') ⓘ

Address range

IPv4 address or CIDR

Allow trusted Microsoft services to bypass this firewall?

Yes  No

**⚠️** This can cause a disruption if this event hub is used by a trusted service and the service's network is not explicitly added.

**NOTE**

To restrict access to specific IP addresses or ranges, see [Allow access from specific IP addresses or ranges](#).

## Trusted Microsoft services

When you enable the **Allow trusted Microsoft services to bypass this firewall** setting, the following services are granted access to your Event Hubs resources.

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure Event Grid	<p>Allows Azure Event Grid to send events to event hubs in your Event Hubs namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"><li>• Enable system-assigned identity for a topic or a domain</li><li>• Add the identity to the Azure Event Hubs Data Sender role on the Event Hubs namespace</li><li>• Then, configure the event subscription that uses an event hub as an endpoint to use the system-assigned identity.</li></ul> <p>For more information, see <a href="#">Event delivery with a managed identity</a></p>
Azure Monitor (Diagnostic Settings and Action Groups)	Allows Azure Monitor to send diagnostic information and alert notifications to event hubs in your Event Hubs namespace. Azure Monitor can read from the event hub and also write data to the event hub.
Azure Stream Analytics	<p>Allows an Azure Stream Analytics job to read data from (<a href="#">input</a>) or write data to (<a href="#">output</a>) event hubs in your Event Hubs namespace.</p> <p><b>Important:</b> The Stream Analytics job should be configured to use a <b>managed identity</b> to access the event hub. For more information, see <a href="#">Use managed identities to access Event Hub from an Azure Stream Analytics job (Preview)</a>.</p>
Azure IoT Hub	Allows IoT Hub to send messages to event hubs in your Event Hub namespace. You also need to do the following steps:

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure API Management	<p>The API Management service allows you to send events to an event hub in your Event Hubs namespace.</p> <ul style="list-style-type: none"> <li>• You can trigger custom workflows by sending events to your event hub when an API is invoked by using the <a href="#">send-request policy</a>.</li> <li>• You can also treat an event hub as your backend in an API. For a sample policy, see <a href="#">Authenticate using a managed identity to access an event hub</a>. You also need to do the following steps:             <ol style="list-style-type: none"> <li>1. Enable system-assigned identity on the API Management instance. For instructions, see <a href="#">Use managed identities in Azure API Management</a>.</li> <li>2. Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace</li> </ol> </li> </ul>
Azure IoT Central	<p>Allows IoT Central to export data to event hubs in your Event Hub namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"> <li>• Enable system-assigned identity for your IoT Central application.</li> <li>• Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace.</li> <li>• Then, configure the Event Hubs <a href="#">export destination on your IoT Central application</a> to use identity-based authentication.</li> </ul>

## Use Resource Manager template

The following sample Resource Manager template adds a virtual network rule to an existing Event Hubs namespace. For the network rule, it specifies the ID of a subnet in a virtual network.

The ID is a fully qualified Resource Manager path for the virtual network subnet. For example,

```
/subscriptions/{id}/resourceGroups/{rg}/providers/Microsoft.Network/virtualNetworks/{vnet}/subnets/default
```

for the default subnet of a virtual network.

When adding virtual network or firewalls rules, set the value of `defaultAction` to `Deny`.

```
{
  "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
  "contentVersion": "1.0.0.0",
  "parameters": {
    "eventhubNamespaceName": {
      "type": "string",
      "metadata": {
        "description": "Name of the Event Hubs namespace"
      }
    },
    "virtualNetworkName": {
      "type": "string",
      "metadata": {
        "description": "Name of the Virtual Network Rule"
      }
    },
    "subnetName": {
      "type": "string",
      "metadata": {
        "description": "ID of the subnet in the virtual network"
      }
    }
  }
}
```

```

"subName": {
    "type": "string",
    "metadata": {
        "description": "Name of the Virtual Network Sub Net"
    }
},
"location": {
    "type": "string",
    "metadata": {
        "description": "Location for Namespace"
    }
}
},
"variables": {
    "namespaceNetworkRuleSetName": "[concat(parameters('eventhubNamespaceName'), concat('/', 'default'))]",
    "subNetId": "[resourceId('Microsoft.Network/virtualNetworks/subnets/', parameters('virtualNetworkName'), parameters('subnetName'))]"
},
"resources": [
{
    "apiVersion": "2018-01-01-preview",
    "name": "[parameters('eventhubNamespaceName')]",
    "type": "Microsoft.EventHub/namespaces",
    "location": "[parameters('location')]",
    "sku": {
        "name": "Standard",
        "tier": "Standard"
    },
    "properties": { }
},
{
    "apiVersion": "2017-09-01",
    "name": "[parameters('virtualNetworkName')]",
    "location": "[parameters('location')]",
    "type": "Microsoft.Network/virtualNetworks",
    "properties": {
        "addressSpace": {
            "addressPrefixes": [
                "10.0.0.0/23"
            ]
        },
        "subnets": [
            {
                "name": "[parameters('subnetName')]",
                "properties": {
                    "addressPrefix": "10.0.0.0/23",
                    "serviceEndpoints": [
                        {
                            "service": "Microsoft.EventHub"
                        }
                    ]
                }
            }
        ]
    }
},
{
    "apiVersion": "2018-01-01-preview",
    "name": "[variables('namespaceNetworkRuleSetName')]",
    "type": "Microsoft.EventHub/namespaces/networkruleset",
    "dependsOn": [
        "[concat('Microsoft.EventHub/namespaces/', parameters('eventhubNamespaceName'))]"
    ],
    "properties": {
        "virtualNetworkRules": [
            {
                "subnet": {
                    "id": "[concat(resourceId('Microsoft.Network/virtualNetworks/subnets', parameters('virtualNetworkName'), parameters('subnetName')), '/virtualNetworkRules')]"
                }
            }
        ]
    }
}
]
}

```

```
        "in": "[variables('subnetId')]",
    },
    "ignoreMissingVnetServiceEndpoint": false
}
],
"ipRules": [<YOUR EXISTING IP RULES>],
"trustedServiceAccessEnabled": false,
"defaultAction": "Deny"
}
}
],
"outputs": { }
}
```

To deploy the template, follow the instructions for [Azure Resource Manager](#).

#### IMPORTANT

If there are no IP and virtual network rules, all the traffic flows into the namespace even if you set the `defaultAction` to `deny`. The namespace can be accessed over the public internet (using the access key). Specify at least one IP rule or virtual network rule for the namespace to allow traffic only from the specified IP addresses or subnet of a virtual network.

## Next steps

For more information about virtual networks, see the following links:

- [Azure virtual network service endpoints](#)
- [Azure Event Hubs IP filtering](#)

# Allow access to Azure Event Hubs namespaces via private endpoints

11/26/2021 • 11 minutes to read • [Edit Online](#)

Azure Private Link Service enables you to access Azure Services (for example, Azure Event Hubs, Azure Storage, and Azure Cosmos DB) and Azure hosted customer/partner services over a **private endpoint** in your virtual network.

A private endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. The private endpoint uses a private IP address from your virtual network, effectively bringing the service into your virtual network. All traffic to the service can be routed through the private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are needed. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. You can connect to an instance of an Azure resource, giving you the highest level of granularity in access control.

For more information, see [What is Azure Private Link?](#)

## Important points

- This feature isn't supported in the **basic** tier.
- Enabling private endpoints can prevent other Azure services from interacting with Event Hubs. Requests that are blocked include those from other Azure services, from the Azure portal, from logging and metrics services, and so on. As an exception, you can allow access to Event Hubs resources from certain **trusted services** even when private endpoints are enabled. For a list of trusted services, see [Trusted services](#).
- Specify **at least one IP rule or virtual network rule** for the namespace to allow traffic only from the specified IP addresses or subnet of a virtual network. If there are no IP and virtual network rules, the namespace can be accessed over the public internet (using the access key).

## Add a private endpoint using Azure portal

### Prerequisites

To integrate an Event Hubs namespace with Azure Private Link, you'll need the following entities or permissions:

- An Event Hubs namespace.
- An Azure virtual network.
- A subnet in the virtual network. You can use the **default** subnet.
- Owner or contributor permissions for both the namespace and the virtual network.

Your private endpoint and virtual network must be in the same region. When you select a region for the private endpoint using the portal, it will automatically filter only virtual networks that are in that region. Your namespace can be in a different region.

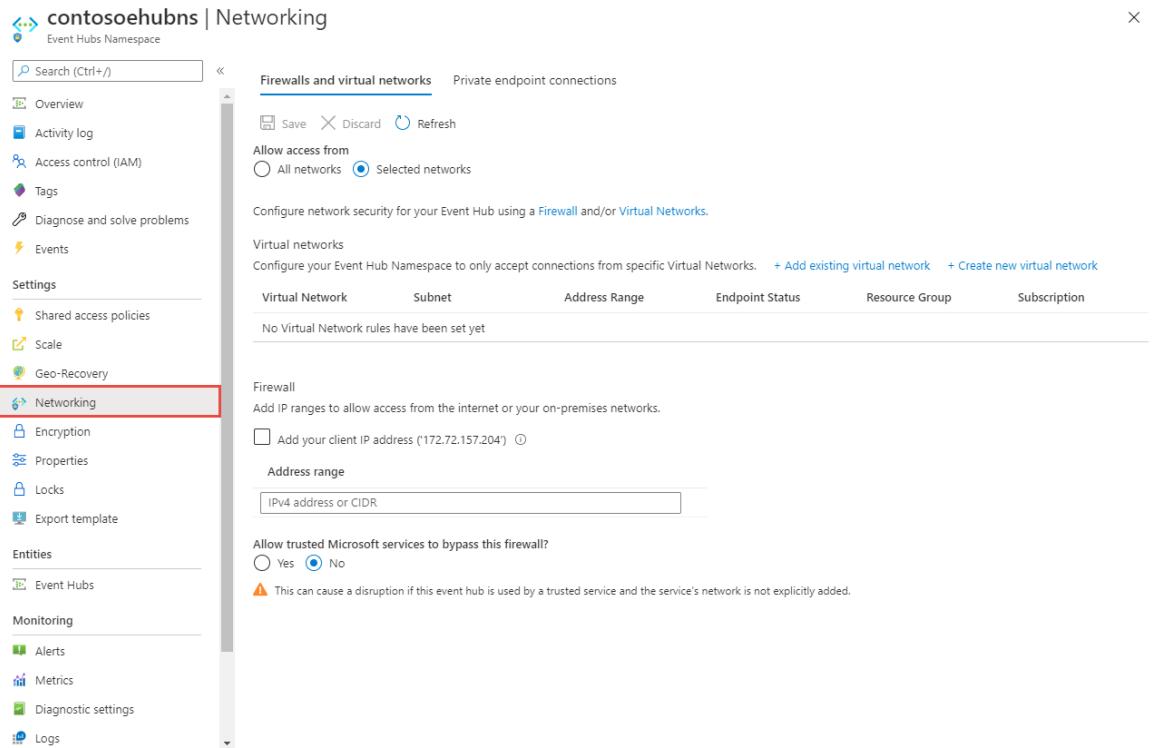
Your private endpoint uses a private IP address in your virtual network.

### Steps

If you already have an Event Hubs namespace, you can create a private link connection by following these steps:

1. Sign in to the [Azure portal](#).

2. In the search bar, type in **event hubs**.
3. Select the **namespace** from the list to which you want to add a private endpoint.
4. Select **Networking** under **Settings** on the left menu.



**WARNING**

By default, the **Selected networks** option is selected. If you don't specify an IP firewall rule or add a virtual network, the namespace can be accessed via public internet (using the access key).

5. Select the **Private endpoint connections** tab at the top of the page.
6. Select the **+ Private Endpoint** button at the top of the page.

The screenshot shows the Azure portal interface for managing an Event Hubs Namespace. On the left, a sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Events, and Networking (which is highlighted with a red box and labeled '1'). The main content area has tabs for Firewalls and virtual networks and Private endpoint connections (also highlighted with a red box and labeled '2'). Below these tabs is a toolbar with buttons for Private endpoint (labeled '3'), Approve, Reject, Remove, and Refresh. A search bar and a filter dropdown are also present. The main table displays connection states, with a message indicating 'No results.'

7. On the **Basics** page, follow these steps:

- Select the **Azure subscription** in which you want to create the private endpoint.
- Select the **resource group** for the private endpoint resource.
- Enter a **name** for the private endpoint.
- Select a **region** for the private endpoint. Your private endpoint must be in the same region as your virtual network, but can be in a different region from the private link resource that you are connecting to.
- Select **Next: Resource >** button at the bottom of the page.

The screenshot shows the 'Create a private endpoint' wizard on the 'Basics' step. At the top, it shows the navigation path: Home > DocsDemoNamespace2 | Overview > DocsDemoNamespace2 | Networking > Create a private endpoint. Below this is a title 'Create a private endpoint' with a numbered step indicator '1 Basics'.

The 'Project details' section contains fields for 'Subscription \*' (dropdown) and 'Resource group \*' (dropdown). The 'Resource group' dropdown is set to 'DocsDemo'.

The 'Instance details' section contains fields for 'Name \*' (text input: 'DocsDemoVnet') and 'Region \*' (dropdown: '(US) Central US').

At the bottom, there are navigation buttons: '< Previous' and 'Next : Resource >'.

8. On the **Resource** page, follow these steps:

a. For connection method, if you select **Connect to an Azure resource in my directory**, follow these steps:

- a. Select the **Azure subscription** in which your Event Hubs namespace exists.
- b. For **Resource type**, Select **Microsoft.EventHub/namespaces** for the Resource type.
- c. For **Resource**, select an Event Hubs namespace from the drop-down list.
- d. Confirm that the **Target subresource** is set to **namespace**.
- e. Select **Next: Configuration >** button at the bottom of the page.

The screenshot shows the 'Create a private endpoint' wizard on step 2: Resource. The 'Subscription' dropdown is set to 'DocsDemoNamespace2'. The 'Resource type' dropdown is set to 'Microsoft.EventHub/namespaces'. The 'Resource' dropdown is set to 'DocsDemoNamespace2'. The 'Target sub-resource' dropdown is set to 'namespace'. The 'Connection method' section shows 'Connect to an Azure resource in my directory' selected.

b. If you select **Connect to an Azure resource by resource ID or alias**, follow these steps:

- a. Enter the **resource ID or alias**. It can be the resource ID or alias that someone has shared with you. The easiest way to get the resource ID is to navigate to the Event Hubs namespace in the Azure portal and copy the portion of URI starting from `/subscriptions/`. See the following image for an example.
- b. For **Target sub-resource**, enter **namespace**. It's the type of the sub-resource that your private endpoint can access.
- c. (optional) Enter a **request message**. The resource owner sees this message while managing private endpoint connection.
- d. Then, select **Next: Configuration >** button at the bottom of the page.

Home > docsdemosbusns | Networking > Create a private endpoint

## Create a private endpoint

Basics  **Resource**  Configuration  Tags  Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method <small>i</small>	<input type="radio"/> Connect to an Azure resource in my directory. <input checked="" type="radio"/> Connect to an Azure resource by resource ID or alias.
Resource ID or alias <small>*</small> <small>i</small>	/subscriptions/ <Azure subscription ID> /resourcegroups/spsb... <input checked="" type="checkbox"/>
Target sub-resource <small>*</small> <small>i</small>	namespace <input checked="" type="checkbox"/>
Request message <small>i</small>	

< Previous Next : Configuration >

9. On the Configuration page, you select the subnet in a virtual network to where you want to deploy the private endpoint.
  - a. Select a **virtual network**. Only virtual networks in the currently selected subscription and location are listed in the drop-down list.
  - b. Select a **subnet** in the virtual network you selected.
  - c. Select **Next: Tags >** button at the bottom of the page.

Dashboard > DocsDemoNS | Networking > Create a private endpoint

## Create a private endpoint

Basics  Resource  **Configuration**  Tags  Review + create

**Networking**

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network <small>*</small> <small>i</small>	DocsDemoVNet <input type="button" value="▼"/>
Subnet <small>*</small> <small>i</small>	default (172.22.0.0/24) <input type="button" value="▼"/>

i If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

**Private DNS integration**

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone	<input checked="" type="radio"/> Yes <input type="radio"/> No
Private DNS Zone <small>*</small> <small>i</small>	(New) privatelink.servicebus.windows.net <input type="button" value="▼"/>

**Review + create** < Previous Next : Tags >

10. On the **Tags** page, create any tags (names and values) that you want to associate with the private endpoint resource. Then, select **Review + create** button at the bottom of the page.
11. On the **Review + create**, review all the settings, and select **Create** to create the private endpoint.

**Create a private endpoint**

✓ Validation passed

✓ Basics ✓ Resource ✓ Configuration ✓ Tags 5 Review + create

**Basics**

Subscription	Azure Messaging PM Playground
Resource group	DocsDemoRG
Region	East US
Name	DocsDemoEndpoint

**Resource**

Subscription ID	<Azure subscription ID and name>
Link type	Microsoft.EventHub/namespaces
Resource group	DocsDemoRG
Resource	DocsDemoNS
Target sub-resource	namespace

**Configuration**

Virtual network resource group	DocsDemoRG
Virtual network	DocsDemoVNet
Subnet	default (172.22.0.0/24)
Integrate with private DNS zone?	Yes
Private DNS zone resource group	DocsDemoRG
Private DNS zone	privatelink.servicebus.windows.net

**Tags**

None

**Create** < Previous Next > Download a template for automation

12. Confirm that you see the private endpoint connection you created shows up in the list of endpoints. In this example, the private endpoint is auto-approved because you connected to an Azure resource in your directory and you have sufficient permissions.

## Trusted Microsoft services

When you enable the **Allow trusted Microsoft services to bypass this firewall** setting, the following services are granted access to your Event Hubs resources.

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure Event Grid	<p>Allows Azure Event Grid to send events to event hubs in your Event Hubs namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"> <li>• Enable system-assigned identity for a topic or a domain</li> <li>• Add the identity to the Azure Event Hubs Data Sender role on the Event Hubs namespace</li> <li>• Then, configure the event subscription that uses an event hub as an endpoint to use the system-assigned identity.</li> </ul> <p>For more information, see <a href="#">Event delivery with a managed identity</a></p>
Azure Monitor (Diagnostic Settings and Action Groups)	<p>Allows Azure Monitor to send diagnostic information and alert notifications to event hubs in your Event Hubs namespace. Azure Monitor can read from the event hub and also write data to the event hub.</p>
Azure Stream Analytics	<p>Allows an Azure Stream Analytics job to read data from (<a href="#">input</a>) or write data to (<a href="#">output</a>) event hubs in your Event Hubs namespace.</p> <p><b>Important:</b> The Stream Analytics job should be configured to use a <b>managed identity</b> to access the event hub. For more information, see <a href="#">Use managed identities to access Event Hub from an Azure Stream Analytics job (Preview)</a>.</p>

TRUSTED SERVICE	SUPPORTED USAGE SCENARIOS
Azure IoT Hub	<p>Allows IoT Hub to send messages to event hubs in your Event Hub namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"> <li>• Enable system-assigned identity for your IoT hub</li> <li>• Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace.</li> <li>• Then, configure the IoT Hub that uses an event hub as a custom endpoint to use the identity-based authentication.</li> </ul>
Azure API Management	<p>The API Management service allows you to send events to an event hub in your Event Hubs namespace.</p> <ul style="list-style-type: none"> <li>• You can trigger custom workflows by sending events to your event hub when an API is invoked by using the <a href="#">send-request policy</a>.</li> <li>• You can also treat an event hub as your backend in an API. For a sample policy, see <a href="#">Authenticate using a managed identity to access an event hub</a>. You also need to do the following steps:           <ol style="list-style-type: none"> <li>1. Enable system-assigned identity on the API Management instance. For instructions, see <a href="#">Use managed identities in Azure API Management</a>.</li> <li>2. Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace</li> </ol> </li> </ul>
Azure IoT Central	<p>Allows IoT Central to export data to event hubs in your Event Hub namespace. You also need to do the following steps:</p> <ul style="list-style-type: none"> <li>• Enable system-assigned identity for your IoT Central application.</li> <li>• Add the identity to the <b>Azure Event Hubs Data Sender</b> role on the Event Hubs namespace.</li> <li>• Then, configure the Event Hubs <a href="#">export destination on your IoT Central application</a> to use identity-based authentication.</li> </ul>

To allow trusted services to access your namespace, switch to the **Firewalls and Virtual networks** tab on the **Networking** page, and select **Yes** for **Allow trusted Microsoft services to bypass this firewall?**.

## Add a private endpoint using PowerShell

The following example shows how to use Azure PowerShell to create a private endpoint connection. It doesn't create a dedicated cluster for you. Follow steps in [this article](#) to create a dedicated Event Hubs cluster.

```

$rgName = "<RESOURCE GROUP NAME>"
$vnetlocation = "<VIRTUAL NETWORK LOCATION>"
$vnetName = "<VIRTUAL NETWORK NAME>"
$subnetName = "<SUBNET NAME>"
$namespaceLocation = "<NAMESPACE LOCATION>"
$namespaceName = "<NAMESPACE NAME>"
$peConnectionName = "<PRIVATE ENDPOINT CONNECTION NAME>

# create resource group
New-AzResourceGroup -Name $rgName -Location $vnetlocation

# create virtual network
$virtualNetwork = New-AzVirtualNetwork `

    -ResourceGroupName $rgName `

    -Location $vnetlocation `

    -Name $vnetName `

    -AddressPrefix 10.0.0.0/16

# create subnet with endpoint network policy disabled
$subnetConfig = Add-AzVirtualNetworkSubnetConfig `

    -Name $subnetName `

    -AddressPrefix 10.0.0.0/24 `

    -PrivateEndpointNetworkPoliciesFlag "Disabled" `

    -VirtualNetwork $virtualNetwork

# update virtual network
$virtualNetwork | Set-AzVirtualNetwork

# create an event hubs namespace in a dedicated cluster
$namespaceResource = New-AzResource -Location $namespaceLocation `

    -ResourceName $namespaceName `

    -ResourceGroupName $rgName `

    -Sku @{name = "Standard"; capacity = 1} `

    -Properties @{clusterArmId = "/subscriptions/<SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventHub/clusters/<EVENT HUBS CLUSTER NAME>"} `

    -ResourceType "Microsoft.EventHub/namespaces" -ApiVersion "2018-01-01-preview"

# create private endpoint connection
$privateEndpointConnection = New-AzPrivateLinkServiceConnection `

    -Name $peConnectionName `

    -PrivateLinkServiceId $namespaceResource.ResourceId `

    -GroupId "namespace"

# get subnet object that you will use later
$virtualNetwork = Get-AzVirtualNetwork -ResourceGroupName $rgName -Name $vnetName
$subnet = $virtualNetwork | Select -ExpandProperty subnets `

    | Where-Object {$_ .Name -eq $subnetName}

# create a private endpoint
$privateEndpoint = New-AzPrivateEndpoint -ResourceGroupName $rgName `

    -Name $vnetName `

    -Location $vnetlocation `

    -Subnet $subnet `

    -PrivateLinkServiceConnection $privateEndpointConnection

(Get-AzResource -ResourceId $namespaceResource.ResourceId -ExpandProperties).Properties

```

## Configure the private DNS Zone

Create a private DNS zone for Event Hubs domain and create an association link with the virtual network:

```

$zone = New-AzPrivateDnsZone -ResourceGroupName $rgName ` 
    -Name "privatelink.servicebus.windows.net"

$link = New-AzPrivateDnsVirtualNetworkLink -ResourceGroupName $rgName ` 
    -ZoneName "privatelink.servicebus.windows.net" ` 
    -Name "mylink" ` 
    -VirtualNetworkId $virtualNetwork.Id

$networkInterface = Get-AzResource -ResourceId $privateEndpoint.NetworkInterfaces[0].Id -ApiVersion "2019-04-01"

foreach ($ipconfig in $networkInterface.properties.ipConfigurations) {
    foreach ($fqdn in $ipconfig.properties.privateLinkConnectionProperties.fqdns) {
        Write-Host "$($ipconfig.properties.privateIPAddress) $($fqdn)"
        $recordName = $fqdn.split('.').[0]
        $dnsZone = $fqdn.split('.').[1]
        New-AzPrivateDnsRecordSet -Name $recordName -RecordType A -ZoneName "privatelink.servicebus.windows.net" ` 
            -ResourceGroupName $rgName -Ttl 600 ` 
            -PrivateDnsRecords (New-AzPrivateDnsRecordConfig -IPv4Address $ipconfig.properties.privateIPAddress)
    }
}

```

## Manage private endpoints using Azure portal

When you create a private endpoint, the connection must be approved. If the resource for which you're creating a private endpoint is in your directory, you can approve the connection request provided you have sufficient permissions. If you're connecting to an Azure resource in another directory, you must wait for the owner of that resource to approve your connection request.

There are four provisioning states:

SERVICE ACTION	SERVICE CONSUMER PRIVATE ENDPOINT STATE	DESCRIPTION
None	Pending	Connection is created manually and is pending approval from the Private Link resource owner.
Approve	Approved	Connection was automatically or manually approved and is ready to be used.
Reject	Rejected	Connection was rejected by the private link resource owner.
Remove	Disconnected	Connection was removed by the private link resource owner, the private endpoint becomes informative and should be deleted for cleanup.

### Approve, reject, or remove a private endpoint connection

1. Sign in to the Azure portal.
2. In the search bar, type in **event hubs**.
3. Select the **namespace** that you want to manage.
4. Select the **Networking** tab.
5. Go to the appropriate section below based on the operation you want to: approve, reject, or remove.

## Approve a private endpoint connection

1. If there are any connections that are pending, you will see a connection listed with **Pending** in the provisioning state.
2. Select the **private endpoint** you wish to approve.
3. Select the **Approve** button.

Connection name	Connection state	Provisioning state	Private endpoint	Comments
c470c9bc-8e2b-44b2-845a-52e959be47d6	Approved	Succeeded	DocDemoVNET	Auto-Approved
74d5a6b2-2b7f-4cb6-979e-f770e625aa1a	Pending	Succeeded	private-endpoint-manual-eh	Please approve my connection, thanks.

4. On the **Approve connection** page, add a comment (optional), and select **Yes**. If you select **No**, nothing happens.
5. You should see the status of the private endpoint connection in the list changed to **Approved**.

## Reject a private endpoint connection

1. If there are any private endpoint connections you want to reject, whether it is a pending request or existing connection, select the connection and click the **Reject** button.

Connection name	Connection state	Provisioning state	Private endpoint	Comments
03439371-9032-4a11-b1ea-db88cc5d6337	Approved	Succeeded	DocsDemoVnet	Auto-Approved

2. On the **Reject connection** page, enter a comment (optional), and select **Yes**. If you select **No**, nothing happens.
3. You should see the status of the private endpoint connection in the list changed to **Rejected**.

## Remove a private endpoint connection

1. To remove a private endpoint connection, select it in the list, and select **Remove** on the toolbar.
2. On the **Delete connection** page, select **Yes** to confirm the deletion of the private endpoint. If you select **No**, nothing happens.
3. You should see the status changed to **Disconnected**. Then, you will see the endpoint disappear from the list.

## Validate that the private link connection works

You should validate that resources within the virtual network of the private endpoint are connecting to your Event Hubs namespace over a private IP address, and that they have the correct private DNS zone integration.

First, create a virtual machine by following the steps in [Create a Windows virtual machine in the Azure portal](#)

In the **Networking** tab:

1. Specify **Virtual network** and **Subnet**. You must select the Virtual Network on which you deployed the private endpoint.
2. Specify a **public IP** resource.
3. For **NIC network security group**, select **None**.
4. For **Load balancing**, select **No**.

Connect to the VM, open the command line, and run the following command:

```
nslookup <event-hubs-namespace-name>.servicebus.windows.net
```

You should see a result that looks like the following.

```
Non-authoritative answer:  
Name:    <event-hubs-namespace-name>.privatelink.servicebus.windows.net  
Address: 10.0.0.4 (private IP address associated with the private endpoint)  
Aliases: <event-hubs-namespace-name>.servicebus.windows.net
```

## Limitations and design considerations

**Pricing:** For pricing information, see [Azure Private Link pricing](#).

**Limitations:** This feature is available in all Azure public regions.

**Maximum number of private endpoints per Event Hubs namespace:** 120.

For more, see [Azure Private Link service: Limitations](#)

## Next steps

- Learn more about [Azure Private Link](#)
- Learn more about [Azure Event Hubs](#)

# Configure customer-managed keys for encrypting Azure Event Hubs data at rest

11/26/2021 • 15 minutes to read • [Edit Online](#)

Azure Event Hubs provides encryption of data at rest with Azure Storage Service Encryption (Azure SSE). The Event Hubs service uses Azure Storage to store the data. All the data that's stored with Azure Storage is encrypted using Microsoft-managed keys. If you use your own key (also referred to as Bring Your Own Key (BYOK) or customer-managed key), the data is still encrypted using the Microsoft-managed key, but in addition the Microsoft-managed key will be encrypted using the customer-managed key. This feature enables you to create, rotate, disable, and revoke access to customer-managed keys that are used for encrypting Microsoft-managed keys. Enabling the BYOK feature is a one time setup process on your namespace.

## IMPORTANT

- The BYOK capability is supported by **premium** and **dedicated** tiers of Event Hubs.
- The encryption can be enabled only for new or empty namespaces. If the namespace contains event hubs, the encryption operation will fail.

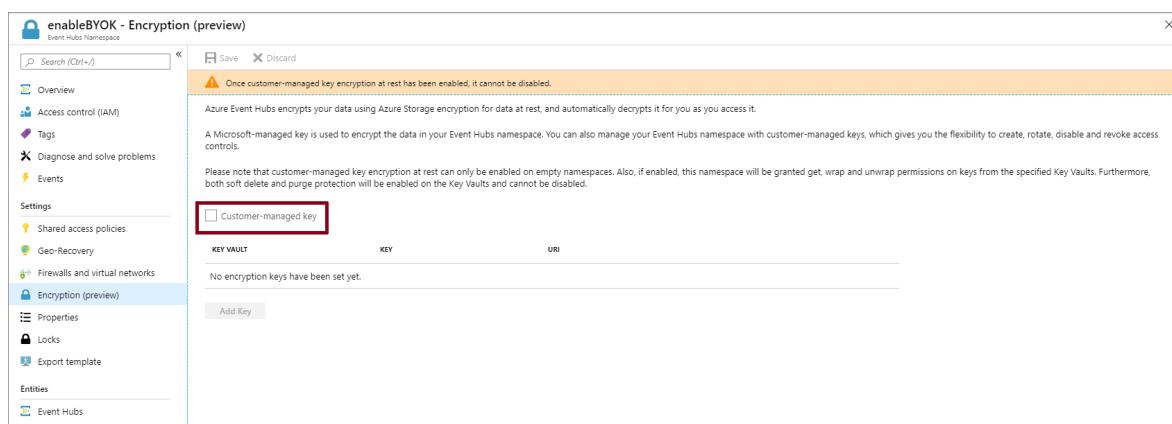
You can use Azure Key Vault to manage your keys and audit your key usage. You can either create your own keys and store them in a key vault, or you can use the Azure Key Vault APIs to generate keys. For more information about Azure Key Vault, see [What is Azure Key Vault?](#)

This article shows how to configure a key vault with customer-managed keys by using the Azure portal. To learn how to create a key vault using the Azure portal, see [Quickstart: Create an Azure Key Vault using the Azure portal](#).

## Enable customer-managed keys (Azure portal)

To enable customer-managed keys in the Azure portal, follow these steps. If you are using the dedicated tier, navigate to your Event Hubs Dedicated cluster first.

1. Select the namespace on which you want to enable BYOK.
2. On the **Settings** page of your Event Hubs namespace, select **Encryption**.
3. Select the **Customer-managed key encryption at rest** as shown in the following image.



## Set up a key vault with keys

After you enable customer-managed keys, you need to associate the customer managed key with your Azure Event Hubs namespace. Event Hubs supports only Azure Key Vault. If you enable the **Encryption with customer-managed key** option in the previous section, you need to have the key imported into Azure Key Vault. Also, the keys must have **Soft Delete** and **Do Not Purge** configured for the key. These settings can be configured using [PowerShell](#) or [CLI](#).

1. To create a new key vault, follow the Azure Key Vault [Quickstart](#). For more information about importing existing keys, see [About keys, secrets, and certificates](#).

#### IMPORTANT

Using customer-managed keys with Azure Event Hubs requires that the key vault have two required properties configured. They are: **Soft Delete** and **Do Not Purge**. These properties are enabled by default when you create a new key vault in the Azure portal. However, if you need to enable these properties on an existing key vault, you must use either PowerShell or Azure CLI.

2. To turn on both soft delete and purge protection when creating a vault, use the [az keyvault create](#) command.

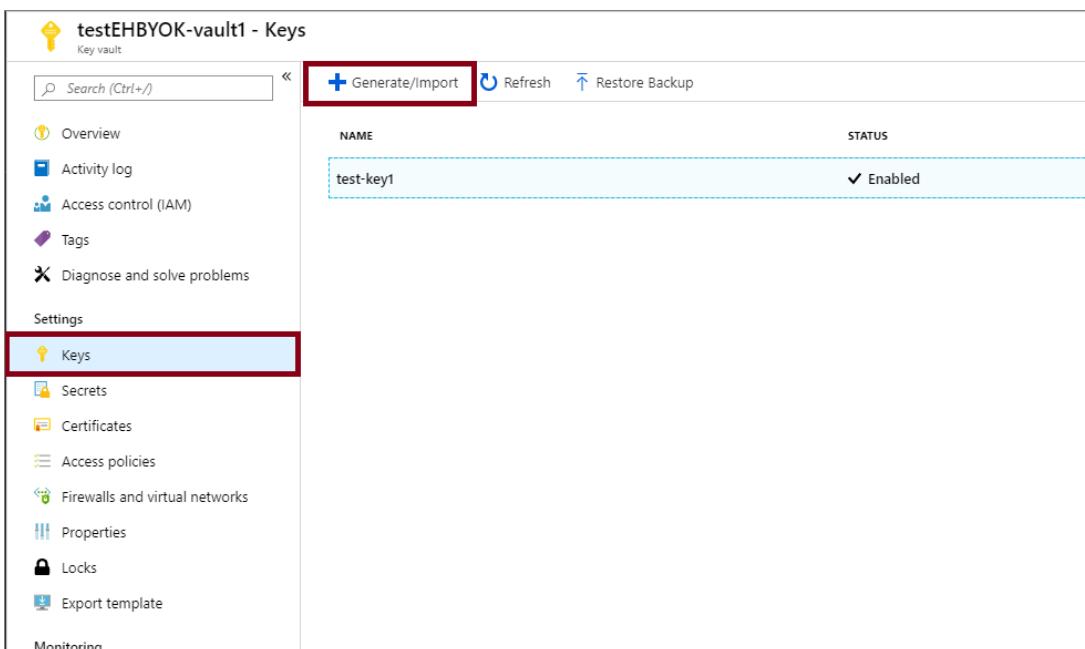
```
az keyvault create --name ContosoVault --resource-group ContosoRG --location westus --enable-soft-delete true --enable-purge-protection true
```

3. To add purge protection to an existing vault (that already has soft delete enabled), use the [az keyvault update](#) command.

```
az keyvault update --name ContosoVault --resource-group ContosoRG --enable-purge-protection true
```

4. Create keys by following these steps:

- a. To create a new key, select **Generate/Import** from the **Keys** menu under **Settings**.



- b. Set **Options** to **Generate** and give the key a name.

## Create a key

The screenshot shows the 'Create a key' interface. At the top, there's a dropdown menu labeled 'Generate'. Below it, a required field 'Name' contains 'test-key2', which is highlighted with a red border. Under 'Key Type', 'RSA' is selected. 'RSA Key Size' offers options 2048, 3072, and 4096, with 2048 selected. There are checkboxes for 'Set activation date?' and 'Set expiration date?', both of which are unchecked. An 'Enabled?' switch is set to 'Yes'. At the bottom is a blue 'Create' button.

- c. You can now select this key to associate with the Event Hubs namespace for encrypting from the drop-down list.

The screenshot shows the 'Select key from Azure Key Vault' dialog. It includes fields for 'Subscription' (set to 'Azure Messaging PM Playground') and 'Key vault' (set to 'testEHBVK-vault1'). A 'Create new key vault' button is present. The 'Key' dropdown shows 'test-key1' and a 'Create new key' button. At the bottom is a 'Select' button.

### NOTE

For redundancy, you can add up to 3 keys. In the event that one of the keys has expired, or is not accessible, the other keys will be used for encryption.

- d. Fill in the details for the key and click **Select**. This will enable the encryption of the Microsoft-managed key with your key (customer-managed key).

## Managed identities

There are two types of managed identities that you can assign to an Event Hubs namespace.

- **System-assigned:** You can enable a managed identity directly on an Event Hubs namespace. When you enable a system-assigned managed identity, an identity is created in Azure AD that's tied to the lifecycle of that Event Hubs namespace. So when the namespace is deleted, Azure automatically deletes the identity for you. By design, only that Azure resource (namespace) can use this identity to request tokens from Azure AD.
- **User-assigned:** You may also create a managed identity as a standalone Azure resource, which is called user-assigned identity. You can create a user-assigned managed identity and assign it to one or more Event Hubs namespaces. In the case of user-assigned managed identities, the identity is managed separately from the resources that use it. They are not tied to the lifecycle of the namespace. You can explicitly delete a user-assigned identity when you no longer need it.

For more information, see [What are managed identities for Azure resources?](#).

## Encrypt using system-assigned identities (template)

This section shows how to do the following tasks using **Azure Resource Manager templates**.

1. Create an **Event Hubs namespace** with a managed service identity.
2. Create a **key vault** and grant the service identity access to the key vault.
3. Update the Event Hubs namespace with the key vault information (key/value).

### Create an Event Hubs cluster and namespace with managed service identity

This section shows you how to create an Azure Event Hubs namespace with managed service identity by using an Azure Resource Manager template and PowerShell.

1. Create an Azure Resource Manager template to create an Event Hubs namespace with a managed service identity. Name the file: `CreateEventHubClusterAndNamespace.json`:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "clusterName": {
            "type": "string",
            "metadata": {
                "description": "Name for the Event Hub cluster."
            }
        },
        "namespaceName": {
            "type": "string",
            "metadata": {
                "description": "Name for the Namespace to be created in cluster."
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Specifies the Azure location for all resources."
            }
        }
    },
    "resources": [
        {
            "type": "Microsoft.EventHub/clusters",
            "apiVersion": "2018-01-01-preview",
            "name": "[parameters('clusterName')]",
            "location": "[parameters('location')]",
            "sku": {
                "name": "Dedicated",
                "capacity": 1
            }
        },
        {
            "type": "Microsoft.EventHub/namespaces",
            "apiVersion": "2018-01-01-preview",
            "name": "[parameters('namespaceName')]",
            "location": "[parameters('location')]",
            "identity": {
                "type": "SystemAssigned"
            },
            "sku": {
                "name": "Standard",
                "tier": "Standard",
                "capacity": 1
            },
            "properties": {
                "isAutoInflateEnabled": false,
                "maximumThroughputUnits": 0,
                "clusterArmId": "[resourceId('Microsoft.EventHub/clusters', parameters('clusterName'))]"
            },
            "dependsOn": [
                "[resourceId('Microsoft.EventHub/clusters', parameters('clusterName'))]"
            ]
        },
        {
            "outputs": {
                "EventHubNamespaceId": {
                    "type": "string",
                    "value": "[resourceId('Microsoft.EventHub/namespaces', parameters('namespaceName'))]"
                }
            }
        }
    ]
}
```

2. Create a template parameter file named: `CreateEventHubClusterAndNamespaceParams.json`.

**NOTE**

Replace the following values:

- `<EventHubsClusterName>` - Name of your Event Hubs cluster
- `<EventHubsNamespaceName>` - Name of your Event Hubs namespace
- `<Location>` - Location of your Event Hubs namespace

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "clusterName": {  
            "value": "<EventHubsClusterName>"  
        },  
        "namespaceName": {  
            "value": "<EventHubsNamespaceName>"  
        },  
        "location": {  
            "value": "<Location>"  
        }  
    }  
}
```

3. Run the following PowerShell command to deploy the template to create an Event Hubs namespace. Then, retrieve the ID of the Event Hubs namespace to use it later. Replace `{MyRG}` with the name of the resource group before running the command.

```
$outputs = New-AzResourceGroupDeployment -Name CreateEventHubClusterAndNamespace -ResourceGroupName  
{MyRG} -TemplateFile ./CreateEventHubClusterAndNamespace.json -TemplateParameterFile  
../CreateEventHubClusterAndNamespaceParams.json  
  
$EventHubNamespaceId = $outputs.Outputs["eventHubNamespaceId"].value
```

### Grant Event Hubs namespace identity access to key vault

1. Run the following command to create a key vault with **purge protection** and **soft-delete** enabled.

```
New-AzureRmKeyVault -Name {keyVaultName} -ResourceGroupName {RGName} -Location {location} -  
EnableSoftDelete -EnablePurgeProtection
```

(OR)

Run the following command to update an **existing key vault**. Specify values for resource group and key vault names before running the command.

```
($updatedKeyVault = Get-AzureRmResource -ResourceId (Get-AzureRmKeyVault -ResourceGroupName {RGName}  
-VaultName {keyVaultName}).ResourceId.Properties | Add-Member -MemberType "NoteProperty" -Name  
"enableSoftDelete" -Value "true"-Force | Add-Member -MemberType "NoteProperty" -Name  
"enablePurgeProtection" -Value "true" -Force
```

2. Set the key vault access policy so that the managed identity of the Event Hubs namespace can access key value in the key vault. Use the ID of the Event Hubs namespace from the previous section.

```
$identity = (Get-AzureRmResource -ResourceId $EventHubNamespaceId -ExpandProperties).Identity

Set-AzureRmKeyVaultAccessPolicy -VaultName {keyVaultName} -ResourceGroupName {RGName} -ObjectId
$identity.PrincipalId -PermissionsToKeys get,wrapKey,unwrapKey,list
```

## Encrypt data in Event Hubs namespace with customer-managed key from key vault

You have done the following steps so far:

1. Created a premium namespace with a managed identity.
2. Create a key vault and granted the managed identity access to the key vault.

In this step, you will update the Event Hubs namespace with key vault information.

1. Create a JSON file named **CreateEventHubClusterAndNamespace.json** with the following content:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "clusterName": {
            "type": "string",
            "metadata": {
                "description": "Name for the Event Hub cluster."
            }
        },
        "namespaceName": {
            "type": "string",
            "metadata": {
                "description": "Name for the Namespace to be created in cluster."
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Specifies the Azure location for all resources."
            }
        },
        "keyVaultUri": {
            "type": "string",
            "metadata": {
                "description": "URI of the KeyVault."
            }
        },
        "keyName": {
            "type": "string",
            "metadata": {
                "description": "KeyName."
            }
        }
    },
    "resources": [
        {
            "type": "Microsoft.EventHub/namespaces",
            "apiVersion": "2018-01-01-preview",
            "name": "[parameters('namespaceName')]",
            "location": "[parameters('location')]",
            "identity": {
                "type": "SystemAssigned"
            },
            "sku": {
                "name": "Standard",
                "tier": "Standard",
                "capacity": 1
            }
        }
    ]
}
```

```

        ],
        "properties": {
            "isAutoInflateEnabled": false,
            "maximumThroughputUnits": 0,
            "clusterArmId": "[resourceId('Microsoft.EventHub/clusters', parameters('clusterName'))]",
            "encryption": {
                "keySource": "Microsoft.KeyVault",
                "keyVaultProperties": [
                    {
                        "keyName": "[parameters('keyName')]",
                        "keyVaultUri": "[parameters('keyVaultUri')]"
                    }
                ]
            }
        }
    ]
}
}

```

2. Create a template parameter file: **UpdateEventHubClusterAndNamespaceParams.json**.

**NOTE**

Replace the following values:

- <EventHubsClusterName> - Name of your Event Hubs cluster.
- <EventHubsNamespaceName> - Name of your Event Hubs namespace
- <Location> - Location of your Event Hubs namespace
- <KeyVaultName> - Name of your key vault
- <KeyName> - Name of the key in the key vault

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "clusterName": {
            "value": "<EventHubsClusterName>"
        },
        "namespaceName": {
            "value": "<EventHubsNamespaceName>"
        },
        "location": {
            "value": "<Location>"
        },
        "keyName": {
            "value": "<KeyName>"
        },
        "keyVaultUri": {
            "value": "https://<KeyVaultName>.vault.azure.net"
        }
    }
}
```

3. Run the following PowerShell command to deploy the Resource Manager template. Replace {MyRG} with the name of your resource group before running the command.

```
New-AzResourceGroupDeployment -Name UpdateEventHubNamespaceWithEncryption -ResourceGroupName {MyRG} -TemplateFile ./UpdateEventHubClusterAndNamespace.json -TemplateParameterFile ./UpdateEventHubClusterAndNamespaceParams.json
```

# Encrypt using user-assigned identities (template)

1. Create a **user-assigned identity**.
2. Create a **key vault** and grant access to the user-assigned identity via access policies.
3. Create an Event Hubs **namespace** with the managed user-identity and the key vault information.

## Create a user-assigned identity

Follow instructions from the [Create a user-assigned managed identity](#) article to create a user-assigned identity. You can also create a user-assigned identity using [CLI](#), [PowerShell](#), [Azure Resource Manager template](#), and [REST](#).

### NOTE

You can assign up to 4 user identities to a namespace. These associations are deleted when the namespace is deleted or when you pass the `identity -> type` in the template to `None`.

## Create a key vault and grant access to user-assigned identity

1. Run the following command to create a key vault with purge protection and soft-delete enabled.

```
New-AzureRmKeyVault -Name "{keyVaultName}" -ResourceGroupName {RGName} -Location "{location}" -  
EnableSoftDelete -EnablePurgeProtection
```

(OR)

Run the following command to update an existing key vault. Specify values for resource group and key vault names before running the command.

```
($updatedKeyVault = Get-AzureRmResource -ResourceId (Get-AzureRmKeyVault -ResourceGroupName {RGName}  
-VaultName {keyVaultName}).ResourceId).Properties | Add-Member -MemberType "NoteProperty" -Name  
"enableSoftDelete" -Value "true"-Force | Add-Member -MemberType "NoteProperty" -Name  
"enablePurgeProtection" -Value "true" -Force
```

2. Get the **Service principal ID** for the user identity using the following PowerShell command. In the example, `ud1` is the user-assigned identity to be used for encryption.

```
$servicePrincipal=Get-AzADServicePrincipal -SearchString "ud1"
```

3. Grant the user-assigned identity access to the key vault by assigning an access policy.

```
Set-AzureRmKeyVaultAccessPolicy -VaultName {keyVaultName} -ResourceGroupName {RGName} -ObjectId  
$servicePrincipal.Id -PermissionsToKeys get,wrapKey,unwrapKey,list
```

### NOTE

You can add up to 3 keys but the user identity used for encryption should be the same for all keys. Currently, only single encryption identity is supported.

## Create an Event Hubs namespace with user identity and key vault information

This section gives you an example that shows you how to do the following tasks using an Azure Resource Manager template.

- Assign a user-managed identity to an Event Hubs namespace.

```

    "identity": {
        "type": "UserAssigned",
        "userAssignedIdentities": [
            "[parameters('identity').userAssignedIdentity]": {}
        ]
    },

```

- Enable encryption on the namespace by specifying a key from your key vault and the user-managed identity to access the key.

```

"encryption": {
    "keySource": "Microsoft.KeyVault",
    "keyVaultProperties": [
        {
            "keyName": "[parameters('keyName')]",
            "keyVaultUri": "[parameters('keyVaultUri')]",
            "identity": {
                "userAssignedIdentity": "
[parameters('identity').userAssignedIdentity]"
            }
        }
    ]
}

```

- Create a JSON file named **CreateEventHubsNamespaceWithUserIdentityAndEncryption.json** with the following content:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "clusterName": {
            "type": "string",
            "metadata": {
                "description": "Name for the Event Hub cluster."
            }
        },
        "namespaceName": {
            "type": "string",
            "metadata": {
                "description": "Name for the Namespace to be created in cluster."
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Specifies the Azure location for all resources."
            }
        },
        "keyVaultUri": {
            "type": "string",
            "metadata": {
                "description": "URI of the KeyVault."
            }
        },
        "keyName": {
            "type": "string",
            "metadata": {
                "description": "KeyName."
            }
        },
        "identity": {
            "type": "Object",
            "metadata": {
                "description": "Identity object"
            }
        }
    }
}
```

```

    "defaultValue": {
        "userAssignedIdentity": ""
    },
    "metadata": {
        "description": "user-assigned identity."
    }
}
},
"resources": [
{
    "type": "Microsoft.EventHub/clusters",
    "apiVersion": "2018-01-01-preview",
    "name": "[parameters('clusterName')]",
    "location": "[parameters('location')]",
    "sku": {
        "name": "Dedicated",
        "capacity": 1
    },
    {
        "type": "Microsoft.EventHub/namespaces",
        "apiVersion": "2021-01-01-preview",
        "name": "[parameters('namespaceName')]",
        "location": "[parameters('location')]",
        "sku": {
            "name": "Standard",
            "tier": "Standard",
            "capacity": 1
        },
        "identity": {
            "type": "UserAssigned",
            "userAssignedIdentities": {
                "[parameters('identity').userAssignedIdentity)": {}
            }
        },
        "properties": {
            "encryption": {
                "keySource": "Microsoft.KeyVault",
                "keyVaultProperties": [
                    {
                        "keyName": "[parameters('keyName')]",
                        "keyVaultUri": "[parameters('keyVaultUri')]",
                        "identity": {
                            "userAssignedIdentity": "[parameters('identity').userAssignedIdentity]"
                        }
                    }
                ]
            }
        }
    }
}
]
}

```

2. Create a template parameter file:

**CreateEventHubsNamespaceWithUserIdentityAndEncryptionParams.json.**

```
{
    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "namespaceName": {
            "value": "<EventHubsNamespaceName>"
        },
        "location": {
            "value": "<Location>"
        },
        "keyVaultUri": {
            "value": "https://<KeyVaultName>.vault.azure.net"
        },
        "keyName": {
            "value": "<KeyName>"
        },
        "identity": {
            "value": {
                "userAssignedIdentity": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<USER MANAGED IDENTITY NAME>"
            }
        }
    }
}
```

In the parameter file, replace placeholders with appropriate values.

PLACEHOLDER	VALUE
<EventHubsNamespaceName>	Name of the Event Hubs namespace.
<Location>	Location where you want the namespace to be created.
<KeyVaultName>	Name of the key vault.
<KeyName>	Name of the key in the key vault.
<AZURE SUBSCRIPTION ID>	Your Azure subscription ID.
<RESOURCE GROUP NAME>	Resource group of the user-managed identity.
<USER MANAGED IDENTITY NAME>	Name of the user-managed identity.

- Run the following PowerShell command to deploy the Resource Manager template. Replace `{MyRG}` with the name of your resource group before running the command.

```
New-AzResourceGroupDeployment -Name CreateEventHubsNamespaceWithEncryption -ResourceGroupName {MyRG}
-TemplateFile ./ CreateEventHubsNamespaceWithUserIdentityAndEncryption.json -TemplateParameterFile ./CreateEventHubsNamespaceWithUserIdentityAndEncryptionParams.json
```

## Use both user-assigned and system-assigned identities

A namespace can have both system-assigned and user-assigned identities at the same time. In this case, the `type` property would be `SystemAssigned, UserAssigned` as shown in the following example.

```

"identity": {
    "type": "SystemAssigned, UserAssigned",
    "userAssignedIdentities": {
        "/subscriptions/<SUBSCRIPTION ID>/resourceGroups/<RESOURCE
GROUP>/providers/Microsoft.ManagedIdentity/userAssignedIdentities/<userIdentity1>" : {}
    }
}

```

In this scenario, you can choose either the system-assigned identity or the user-assigned identity for encrypting the data at rest.

In the Resource Manager template, if you don't specify an `identity` attribute, the system-managed identity is used. Here's an example snippet.

```

"properties": {
    "encryption": {
        "keySource": "Microsoft.KeyVault",
        "keyVaultProperties": [
            {
                "keyName": "[parameters('keyName')]",
                "keyVaultUri": "[parameters('keyVaultUri')]"
            }
        ]
    }
}

```

See the following example for using the user-managed identity for the encryption. Notice the `identity` attribute is set to the user-managed identity.

```

"properties": {
    "encryption": {
        "keySource": "Microsoft.KeyVault",
        "keyVaultProperties": [
            {
                "keyName": "[parameters('keyName')]",
                "keyVaultUri": "[parameters('keyVaultUri')]",
                "identity": {
                    "userAssignedIdentity": "[parameters('identity').userAssignedIdentity]"
                }
            }
        ]
    }
}

```

## Enable infrastructure (or double) encryption of data

If you require a higher level of assurance that your data is secure, you can enable infrastructure level encryption which is also known as Double Encryption.

When infrastructure encryption is enabled, data in the Event Hubs namespace account is encrypted twice, once at the service level and once at the infrastructure level, using two different encryption algorithms and two different keys. Hence, infrastructure encryption of Event Hubs data protects against a scenario where one of the encryption algorithms or keys may be compromised.

You can enable infrastructure encryption by updating the Azure Resource Manager template with `requireInfrastructureEncryption` property in the above `CreateEventHubClusterAndNamespace.json` as shown below.

```

"properties": {
    "isAutoInflateEnabled": false,
    "maximumThroughputUnits": 0,
    "clusterArmId": "[resourceId('Microsoft.EventHub/clusters', parameters('clusterName'))]",
    "encryption": {
        "keySource": "Microsoft.KeyVault",
        "requireInfrastructureEncryption": true,
        "keyVaultProperties": [
            {
                "keyName": "[parameters('keyName')]",
                "keyVaultUri": "[parameters('keyVaultUri')]"
            }
        ]
    }
}

```

## Rotate, revoke, and cache encryption keys

### Rotate your encryption keys

You can rotate your key in the key vault by using the Azure Key Vaults rotation mechanism. Activation and expiration dates can also be set to automate key rotation. The Event Hubs service will detect new key versions and start using them automatically.

### Revoke access to keys

Revoking access to the encryption keys won't purge the data from Event Hubs. However, the data can't be accessed from the Event Hubs namespace. You can revoke the encryption key through access policy or by deleting the key. Learn more about access policies and securing your key vault from [Secure access to a key vault](#).

Once the encryption key is revoked, the Event Hubs service on the encrypted namespace will become inoperable. If the access to the key is enabled or the delete key is restored, Event Hubs service will pick the key so you can access the data from the encrypted Event Hubs namespace.

### Caching of keys

The Event Hubs instance (event hub) polls its listed encryption keys every 5 minutes. It caches and uses them until the next poll, which is after 5 minutes. As long as at least one key is available, the event hub is accessible. If all listed keys are inaccessible when it polls, all event hubs will become unavailable.

Here are more details:

- Every 5 minutes, the Event Hubs service polls all customer-managed keys listed in the namespace's record:
  - If a key has been rotated, the record is updated with the new key.
  - If a key has been revoked, the key is removed from the record.
  - If all keys have been revoked, the namespace's encryption status is set to **Revoked**. The data can't be accessed from the Event Hubs namespace.'

## Considerations when using geo-disaster recovery

### IMPORTANT

To enable Geo-DR on a namespace that's using the BYOK encryption, the secondary namespace for pairing must have a system-assigned or user-assigned managed identity enabled on it.

### Geo-disaster recovery - encryption with system-assigned identities

To enable encryption of Microsoft-managed key with a customer managed key, an [access policy](#) is set up for a system-assigned managed identity on the specified Azure KeyVault. This ensures controlled access to the Azure

KeyVault from the Azure Event Hubs namespace.

Due to this:

- If [Geo disaster recovery](#) is already enabled for the Event Hubs namespace and you are looking to enable customer managed key, then
  - Break the pairing.
  - [Set up the access policy](#) for the system-assigned managed identity for both the primary and secondary namespaces to the key vault.
  - Set up encryption on the primary namespace.
  - Re-pair the primary and secondary namespaces.
- If you are looking to enable Geo-DR on an Event Hubs namespace where customer-managed key is already set up, then follow these steps:
  - [Set up the access policy](#) for the managed identity for the secondary namespace to the key vault.
  - Pair the primary and secondary namespaces.

### **Geo-disaster recovery - encryption with user-assigned identities**

Here are a few recommendations:

1. Create managed identity and assign Key Vault permissions to your managed identity.
2. Add the identity as a user assigned identity, and enable encryption with the identity on both namespaces.
3. Pair namespaces together

Conditions for enabling Geo-DR and Encryption with User-Assigned Identities:

1. Secondary namespace must already have Encryption enabled with a User-Assigned identity if it is to be paired with a primary namespace that has Encryption enabled.
2. It is not possible to enable Encryption on an already paired primary, even if the secondary has a User-Assigned identity associated with the namespace.

## **Set up diagnostic logs**

Setting diagnostic logs for BYOK enabled namespaces gives you the required information about the operations. These logs can be enabled and later stream to an event hub or analyzed through log analytics or streamed to storage to perform customized analytics. To learn more about diagnostic logs, see [Overview of Azure Diagnostic logs](#). For the schema, see [Monitor data reference](#).

## **Next steps**

See the following articles:

- [Event Hubs overview](#)
- [Key Vault overview](#)

# Troubleshoot connectivity issues - Azure Event Hubs

11/26/2021 • 8 minutes to read • [Edit Online](#)

There are various reasons for client applications not able to connect to an event hub. The connectivity issues that you experience may be permanent or transient. If the issue happens all the time (permanent), you may want to check the connection string, your organization's firewall settings, IP firewall settings, network security settings (service endpoints, private endpoints, etc.), and more. For transient issues, upgrading to latest version of the SDK, running commands to check dropped packets, and obtaining network traces may help with troubleshooting the issues.

This article provides tips for troubleshooting connectivity issues with Azure Event Hubs.

## Troubleshoot permanent connectivity issues

If the application isn't able to connect to the event hub at all, follow steps from this section to troubleshoot the issue.

### Check if there is a service outage

Check for the Azure Event Hubs service outage on the [Azure service status site](#).

### Verify the connection string

Verify that the connection string you are using is correct. See [Get connection string](#) to get the connection string using the Azure portal, CLI, or PowerShell.

For Kafka clients, verify that producer.config or consumer.config files are configured properly. For more information, see [Send and receive messages with Kafka in Event Hubs](#).

### What ports do I need to open on the firewall?

You can use the following protocols with Azure Event Hubs to send and receive events:

- Advanced Message Queuing Protocol 1.0 (AMQP)
- Hypertext Transfer Protocol 1.1 with TLS (HTTPS)
- Apache Kafka

See the following table for the outbound ports you need to open to use these protocols to communicate with Azure Event Hubs.

Protocol	Ports	Details
AMQP	5671 and 5672	See <a href="#">AMQP protocol guide</a>
HTTPS	443	This port is used for the HTTP/REST API and for AMQP-over-WebSockets.
Kafka	9093	See <a href="#">Use Event Hubs from Kafka applications</a>

The HTTPS port is required for outbound communication also when AMQP is used over port 5671, because several management operations performed by the client SDKs and the acquisition of tokens from Azure Active Directory (when used) run over HTTPS.

The official Azure SDKs generally use the AMQP protocol for sending and receiving events from Event Hubs. The

AMQP-over-WebSockets protocol option runs over port TCP 443 just like the HTTP API, but is otherwise functionally identical with plain AMQP. This option has higher initial connection latency because of extra handshake round trips and slightly more overhead as tradeoff for sharing the HTTPS port. If this mode is selected, TCP port 443 is sufficient for communication. The following options allow selecting the plain AMQP or AMQP WebSockets mode:

LANGUAGE	OPTION
.NET	<code>EventHubConnectionOptions.TransportType</code> property with <code>EventHubsTransportType.AmqpTcp</code> or <code>EventHubsTransportType.AmqpWebSockets</code>
Java	<code>com.microsoft.azure.eventhubs.EventProcessorClientBuilder.transportType</code> with <code>AmqpTransportType.AMQP</code> or <code>AmqpTransportType.AMQP_WEB_SOCKETS</code>
Node	<code>EventHubConsumerClientOptions</code> has a <code>webSocketOptions</code> property.
Python	<code>EventHubConsumerClient.transport_type</code> with <code>TransportType.Amqp</code> or <code>TransportType.AmqpOverWebSocket</code>

### What IP addresses do I need to allow?

When you are working with Azure, sometimes you have to allow specific IP address ranges or URLs in your corporate firewall or proxy to access all Azure services you are using or trying to use. Verify that the traffic is allowed on IP addresses used by Event Hubs. For IP addresses used by Azure Event Hubs: see [Azure IP Ranges and Service Tags - Public Cloud](#).

Also, verify that the IP address for your namespace is allowed. To find the right IP addresses to allow for your connections, follow these steps:

1. Run the following command from a command prompt:

```
nslookup <YourNamespaceName>.servicebus.windows.net
```

2. Note down the IP address returned in `Non-authoritative answer`.

If you use the **zone redundancy** for your namespace, you need to do a few extra steps:

1. First, you run nslookup on the namespace.

```
nslookup <yournamespace>.servicebus.windows.net
```

2. Note down the name in the **non-authoritative answer** section, which is in one of the following formats:

```
<name>-s1.cloudapp.net
<name>-s2.cloudapp.net
<name>-s3.cloudapp.net
```

3. Run nslookup for each one with suffixes s1, s2, and s3 to get the IP addresses of all three instances running in three availability zones,

#### NOTE

The IP address returned by the `nslookup` command isn't a static IP address. However, it remains constant until the underlying deployment is deleted or moved to a different cluster.

### What client IPs are sending events to or receiving events from my namespace?

First, enable [IP filtering](#) on the namespace.

Then, Enable diagnostic logs for [Event Hubs virtual network connection events](#) by following instructions in the [Enable diagnostic logs](#). You'll see the IP address for which connection is denied.

```
{  
    "SubscriptionId": "0000000-0000-0000-0000-000000000000",  
    "NamespaceName": "namespace-name",  
    "IPAddress": "1.2.3.4",  
    "Action": "Deny Connection",  
    "Reason": "IPAddress doesn't belong to a subnet with Service Endpoint enabled.",  
    "Count": "65",  
    "ResourceId": "/subscriptions/0000000-0000-0000-0000-  
00000000000/resourcegroups/testrg/providers/microsoft.eventhub/namespaces/namespace-name",  
    "Category": "EventHubVNetConnectionEvent"  
}
```

#### IMPORTANT

Virtual network logs are generated only if the namespace allows access from [specific IP addresses](#) (IP filter rules). If you don't want to restrict access to your namespace using these features and still want to get virtual network logs to track IP addresses of clients connecting to the Event Hubs namespace, you could use the following workaround: Enable IP filtering, and add the total addressable IPv4 range (1.0.0.0/1 - 255.0.0.0/1). Event Hubs doesn't support IPv6 address ranges.

#### NOTE

Currently, it's not possible to determine the source IP of an individual message or event.

### Verify that EventHub service tag is allowed in your network security groups

If your application is running inside a subnet and there is an associated network security group, confirm whether the internet outbound is allowed or EventHub service tag is allowed. See [Virtual network service tags](#) and search for `EventHub`.

### Check if the application needs to be running in a specific subnet of a vnet

Confirm that your application is running in a virtual network subnet that has access to the namespace. If it's not, run the application in the subnet that has access to the namespace or add the IP address of the machine on which application is running to the [IP firewall](#).

When you create a virtual network service endpoint for an event hub namespace, the namespace accepts traffic only from the subnet that's bound to the service endpoint. There is an exception to this behavior. You can add specific IP addresses in the IP firewall to enable access to the Event Hub public endpoint. For more information, see [Network service endpoints](#).

### Check the IP Firewall settings for your namespace

Check that the public IP address of the machine on which the application is running isn't blocked by the IP firewall.

By default, Event Hubs namespaces are accessible from internet as long as the request comes with valid

authentication and authorization. With IP firewall, you can restrict it further to only a set of IPv4 addresses or IPv4 address ranges in [CIDR \(Classless Inter-Domain Routing\)](#) notation.

The IP firewall rules are applied at the Event Hubs namespace level. Therefore, the rules apply to all connections from clients using any supported protocol. Any connection attempt from an IP address that does not match an allowed IP rule on the Event Hubs namespace is rejected as unauthorized. The response does not mention the IP rule. IP filter rules are applied in order, and the first rule that matches the IP address determines the accept or reject action.

For more information, see [Configure IP firewall rules for an Azure Event Hubs namespace](#). To check whether you have IP filtering, virtual network, or certificate chain issues, see [Troubleshoot network related issues](#).

### Check if the namespace can be accessed using only a private endpoint

If the Event Hubs namespace is configured to be accessible only via private endpoint, confirm that the client application is accessing the namespace over the private endpoint.

[Azure Private Link service](#) enables you to access Azure Event Hubs over a **private endpoint** in your virtual network. A private endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. The private endpoint uses a private IP address from your virtual network, effectively bringing the service into your virtual network. All traffic to the service can be routed through the private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are needed. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. You can connect to an instance of an Azure resource, giving you the highest level of granularity in access control.

For more information, see [Configure private endpoints](#). See the **Validate that the private endpoint connection works** section to confirm that a private endpoint is used.

### Troubleshoot network-related issues

To troubleshoot network-related issues with Event Hubs, follow these steps:

Browse to or `wget https://<yournamespacename>.servicebus.windows.net/`. It helps with checking whether you have IP filtering or virtual network or certificate chain issues (most common when using Java SDK).

An example of successful message:

```
<feed xmlns="http://www.w3.org/2005/Atom"><title type="text">Publicly Listed Services</title><subtitle type="text">This is the list of publicly-listed services currently available.</subtitle><id>uuid:27fcde2-3a99-44b1-8f1e-3e92b52f0171;id=30</id><updated>2019-12-27T13:11:47Z</updated><generator>Service Bus 1.1</generator></feed>
```

An example of failure error message:

```
<Error>
  <Code>400</Code>
  <Detail>
    Bad Request. To know more visit https://aka.ms/sbResourceMgrExceptions. . TrackingId:b786d4d1-cbaf-47a8-a3d1-be689cda2a98_G22, SystemTracker:NoSystemTracker, Timestamp:2019-12-27T13:12:40
  </Detail>
</Error>
```

## Troubleshoot transient connectivity issues

If you are experiencing intermittent connectivity issues, go through the following sections for troubleshooting tips.

### Use the latest version of the client SDK

Some of the transient connectivity issues may have been fixed in the later versions of the SDK than what you are using. Ensure that you are using the latest version of client SDKs in your applications. SDKs are continuously improved with new/updated features and bug fixes, so always test with latest package. Check the release notes for issues that are fixed and features added/updated.

For information about client SDKs, see the [Azure Event Hubs - Client SDKs](#) article.

### Run the command to check dropped packets

When there are intermittent connectivity issues, run the following command to check if there are any dropped packets. This command will try to establish 25 different TCP connections every 1 second with the service. Then, you can check how many of them succeeded/failed and also see TCP connection latency. You can download the `psping` tool from [here](#).

```
.\psping.exe -n 25 -i 1 -q <yournamespacename>.servicebus.windows.net:5671 -nobanner
```

You can use equivalent commands if you're using other tools such as `tnc`, `ping`, and so on.

Obtain a network trace if the previous steps don't help and analyze it using tools such as [Wireshark](#). Contact [Microsoft Support](#) if needed.

### Service upgrades/restarts

Transient connectivity issues may occur because of backend service upgrades and restarts. When they occur, you may see the following symptoms:

- There may be a drop in incoming messages/requests.
- The log file may contain error messages.
- The applications may be disconnected from the service for a few seconds.
- Requests may be momentarily throttled.

If the application code utilizes SDK, the retry policy is already built in and active. The application will reconnect without significant impact to the application/workflow. Catching these transient errors, backing off and then retrying the call will ensure that your code is resilient to these transient issues.

## Next steps

See the following articles:

- [Troubleshoot authentication and authorization issues](#)

# Troubleshoot authentication and authorization issues - Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

The [Troubleshoot connectivity issues](#) article provides tips for troubleshooting connectivity issues with Azure Event Hubs. This article provides tips and recommendations for troubleshooting authentication and authorization issues with Azure Event Hubs.

## If you are using Azure Active Directory

If you are using Azure Active Directory (Azure AD) to authenticate and authorize with Azure Event Hubs, confirm that the identity accessing the event hub is a member of the right **Azure role** at the right **resource scope** (consumer group, event hub, namespace, resource group, or subscription).

### Azure roles

- [Azure Event Hubs Data owner](#) for complete access to Event Hubs resources.
- [Azure Event Hubs Data sender](#) for the send access.
- [Azure Event Hubs Data receiver](#) for the receive access.

For Schema Registry built-in roles, see [Schema Registry roles](#).

### Resource scopes

- **Consumer group:** At this scope, role assignment applies only to this entity. Currently, the Azure portal doesn't support assigning an Azure role to a security principal at this level.
- **Event hub:** Role assignment applies to the Event Hub entity and the consumer group under it.
- **Namespace:** Role assignment spans the entire topology of Event Hubs under the namespace and to the consumer group associated with it.
- **Resource group:** Role assignment applies to all the Event Hubs resources under the resource group.
- **Subscription:** Role assignment applies to all the Event Hubs resources in all of the resource groups in the subscription.

For more information, see the following articles:

- [Authenticate an application with Azure Active Directory to access Event Hubs resources](#)
- [Authorize access to Event Hubs resources using Azure Active Directory](#)

## If you are using Shared access signatures (SAS)

If you are using [SAS](#), follow these steps:

- Ensure that the SAS key you are using is correct. If not, use the right SAS key.
- Verify that the key has the right permissions (send, receive, or manage). If not, use a key that has the permission you need.
- Check if the key has expired. We recommend that you renew the SAS well before expiration. If there is clock skew between client and the Event Hubs service nodes, the authentication token might expire before client realizes it. Current implementation accounts clock skew up to 5 minutes, that is, client renews the token 5 minutes before it expires. Therefore, if the clock skew is bigger than 5 minutes the client can observe intermittent authentication failures.
- If **SAS start time** is set to **now**, you may see intermittent failures for the first few minutes due to clock skew

(differences in current time on different machines). Set the start time to be at least 15 minutes in the past or don't set it at all. The same generally applies to the expiry time as well.

For more information, see the following articles:

- [Authenticate using shared access signatures \(SAS\)](#).
- [Authorizing access to Event Hubs resources using Shared Access Signatures](#)

## Next steps

See the following articles:

- [Troubleshoot connectivity issues](#)

# Apache Kafka troubleshooting guide for Event Hubs

11/26/2021 • 4 minutes to read • [Edit Online](#)

This article provides troubleshooting tips for issues that you may run into when using Event Hubs for Apache Kafka.

## Server Busy exception

You may receive Server Busy exception because of Kafka throttling. With AMQP clients, Event Hubs immediately returns a **server busy** exception upon service throttling. It's equivalent to a "try again later" message. In Kafka, messages are delayed before being completed. The delay length is returned in milliseconds as `throttle_time_ms` in the produce/fetch response. In most cases, these delayed requests aren't logged as server busy exceptions on Event Hubs dashboards. Instead, the response's `throttle_time_ms` value should be used as an indicator that throughput has exceeded the provisioned quota.

If the traffic is excessive, the service has the following behavior:

- If produce request's delay exceeds request timeout, Event Hubs returns **Policy Violation** error code.
- If fetch request's delay exceeds request timeout, Event Hubs logs the request as throttled and responds with empty set of records and no error code.

[Dedicated clusters](#) don't have throttling mechanisms. You're free to consume all of your cluster resources.

## No records received

You may see consumers not getting any records and constantly rebalancing. In this scenario, consumers don't get any records and constantly rebalance. There's no exception or error when it happens, but the Kafka logs will show that the consumers are stuck trying to rejoin the group and assign partitions. There are a few possible causes:

- Make sure that your `request.timeout.ms` is at least the recommended value of 60000 and your `session.timeout.ms` is at least the recommended value of 30000. Having these settings too low could cause consumer timeouts, which then cause rebalances (which then cause more timeouts, which cause more rebalancing, and so on)
- If your configuration matches those recommended values, and you're still seeing constant rebalancing, feel free to open up an issue (make sure to include your entire configuration in the issue so that we can help debug)!

## Compression/Message format version issue

Kafka supports compression, and Event Hubs for Kafka currently doesn't. Errors that mention a message-format version (for example, `The message format version on the broker does not support the request.`) are caused when a client tries to send compressed Kafka messages to our brokers.

If compressed data is necessary, compressing your data before sending it to the brokers and decompressing after receiving is a valid workaround. The message body is just a byte array to the service, so client-side compression/decompression won't cause any issues.

## UnknownServerException

You may receive an UnknownServerException from Kafka client libraries similar to the following example:

```
org.apache.kafka.common.errors.UnknownServerException: The server experienced an unexpected error when processing the request
```

Open a ticket with Microsoft support. Debug-level logging and exception timestamps in UTC are helpful in debugging the issue.

## Other issues

Check the following items if you see issues when using Kafka on Event Hubs.

- **Firewall blocking traffic** - Make sure that port 9093 isn't blocked by your firewall.
- **TopicAuthorizationException** - The most common causes of this exception are:
  - A typo in the connection string in your configuration file, or
  - Trying to use Event Hubs for Kafka on a Basic tier namespace. The Event Hubs for Kafka feature isn't supported in the basic tier.
- **Kafka version mismatch** - Event Hubs for Kafka Ecosystems supports Kafka versions 1.0 and later. Some applications using Kafka version 0.10 and later could occasionally work because of the Kafka protocol's backwards compatibility, but we strongly recommend against using old API versions. Kafka versions 0.9 and earlier don't support the required SASL protocols and can't connect to Event Hubs.
- **Strange encodings on AMQP headers when consuming with Kafka** - when sending events to an event hub over AMQP, any AMQP payload headers are serialized in AMQP encoding. Kafka consumers don't deserialize the headers from AMQP. To read header values, manually decode the AMQP headers.  
Alternatively, you can avoid using AMQP headers if you know that you'll be consuming via Kafka protocol.  
For more information, see [this GitHub issue](#).
- **SASL authentication** - Getting your framework to cooperate with the SASL authentication protocol required by Event Hubs can be more difficult than meets the eye. See if you can troubleshoot the configuration using your framework's resources on SASL authentication.

## Limits

Apache Kafka vs. Event Hubs Kafka. For the most part, Azure Event Hubs' Kafka interface has the same defaults, properties, error codes, and general behavior that Apache Kafka does. The instances that these two explicitly differ (or where Event Hubs imposes a limit that Kafka doesn't) are listed below:

- The max length of the `group.id` property is 256 characters
- The max size of `offset.metadata.max.bytes` is 1024 bytes
- Offset commits are throttled to 4 calls/second per partition with a maximum internal log size of 1 MB

## Next steps

To learn more about Event Hubs and Event Hubs for Kafka, see the following articles:

- [Apache Kafka developer guide for Event Hubs](#)
- [Apache Kafka migration guide for Event Hubs](#)
- [Frequently asked questions - Event Hubs for Apache Kafka](#)
- [Recommended configurations](#)

# EventHubsException - .NET

11/26/2021 • 2 minutes to read • [Edit Online](#)

An **EventHubsException** is triggered when an operation specific to Event Hubs has caused an issue, including both errors within the service and specific to the client.

## Exception information

The exception includes the following contextual information to assist in understanding the context of the error and its relative severity.

- **IsTransient:** Identifies whether the exception is considered recoverable or not. In the case where it was considered transient, the appropriate retry policy has already been applied and retries were unsuccessful.
- **Reason:** Provides a set of well-known reasons for the failure that help to categorize and clarify the root cause. These reasons are intended to allow for applying exception filtering and other logic when inspecting the text of an exception message wouldn't be ideal. Some key failure reasons are:
  - **Client Closed:** Occurs when an Event Hub client that has already been closed or disposed. We recommend that you check the application code to ensure objects from the Event Hubs client library are created and closed in the intended scope.
  - **Service Timeout:** Indicates that the Event Hubs service didn't respond to an operation within the expected amount of time. This issue may have been caused by a transient network issue or service problem. The Event Hubs service may or may not have successfully completed the request; the status isn't known. It's recommended to attempt to verify the current state and retry if necessary.
  - **Quota Exceeded:** Indicates that there are too many active read operations for a single consumer group. This limit depends on the tier of the Event Hubs namespace, and moving to a higher tier may be needed. An alternative would be to create additional consumer groups and ensure that the number of consumer client reads for any group is within the limit. For more information, see [Azure Event Hubs quotas and limits](#).
  - **Message Size Exceeded:** Event data as a maximum size allowed for both an individual event and a batch of events. It includes the data of the event, and any associated metadata and system overhead. To resolve this error, reduce the number of events sent in a batch or reduce the size of data included in the message. Because size limits are subject to change, see to [Azure Event Hubs quotas and limits](#) for specifics.
  - **Consumer Disconnected:** A consumer client was disconnected by the Event Hub service from the Event Hub instance. It typically occurs when a consumer with a higher owner level asserts ownership over a partition and consumer group pairing.
  - **Resource Not Found:** The Event Hubs service couldn't find a resource such as an event hub, consumer group, or partition. It may have been deleted or that there's an issue with the Event Hubs service itself.

## Handling exceptions

You can react to a specific failure reason for the **EventHubException** in several ways. One way is to apply an exception filter clause as part of the catch block.

```
try
{
    // Read events using the consumer client
}
catch (EventHubsException ex) where
    (ex.Reason == EventHubsException.FailureReason.ConsumerDisconnected)
{
    // Take action based on a consumer being disconnected
}
```

## Next steps

There are other exceptions that are documented in the [legacy article](#). Some of them apply only to the legacy Event Hubs .NET client library.

# Event Hubs messaging exceptions - .NET (legacy)

11/26/2021 • 8 minutes to read • [Edit Online](#)

This section lists the .NET exceptions generated by .NET Framework APIs.

## IMPORTANT

Some of the exceptions listed in the article apply only to legacy Event Hubs .NET library. For example: Microsoft.ServiceBus.\* exceptions.

For information about the EventHubsException raised by the new .NET library, see [EventHubsException - .NET](#)

## Exception categories

The Event Hubs .NET APIs generate exceptions that can fall into the following categories, along with the associated action you can take to try to fix them:

- User coding error:

- [System.ArgumentException](#)
- [System.InvalidOperationException](#)
- [System.OperationCanceledException](#)
- [System.Runtime.Serialization.SerializationException](#)

General action: Try to fix the code before proceeding.

- Setup/configuration error:

- [Microsoft.ServiceBus.Messaging.MessagingEntityNotFoundException](#)
- [Microsoft.Azure.EventHubs.MessagingEntityNotFoundException](#)
- [System.UnauthorizedAccessException](#)

General action: Review your configuration and change if necessary.

- Transient exceptions:

- [Microsoft.ServiceBus.Messaging.MessagingException](#)
- [Microsoft.ServiceBus.Messaging.ServerBusyException](#)
- [Microsoft.Azure.EventHubs.ServerBusyException](#)
- [Microsoft.ServiceBus.Messaging.MessagingCommunicationException](#)

General action: Retry the operation or notify users.

- Other exceptions:

- [System.Transactions.TransactionException](#)
- [System.TimeoutException](#)
- [Microsoft.ServiceBus.Messaging.MessageLockLostException](#)
- [Microsoft.ServiceBus.Messaging.SessionLockLostException](#)

General action: Specific to the exception type; refer to the table in the following section.

## Exception types

The following table lists messaging exception types, and their causes, and notes suggested action you can take.

Exception Type	Description/Cause/Examples	Suggested Action	Note on Automatic/Immediate Retry
TimeoutException	The server didn't respond to the requested operation within the specified time, which is controlled by <a href="#">OperationTimeout</a> . The server may have completed the requested operation. This exception can happen because of network or other infrastructure delays.	Check the system state for consistency and retry if necessary. See <a href="#">TimeoutException</a> .	Retry might help in some cases; add retry logic to code.
InvalidOperationException	The requested user operation isn't allowed within the server or service. See the exception message for details. For example, <a href="#">Complete</a> generates this exception if the message was received in <a href="#">ReceiveAndDelete</a> mode.	Check the code and the documentation. Make sure the requested operation is valid.	Retry won't help.
OperationCanceledException	An attempt is made to invoke an operation on an object that has already been closed, aborted, or disposed. In rare cases, the ambient transaction is already disposed.	Check the code and make sure it doesn't invoke operations on a disposed object.	Retry won't help.
UnauthorizedAccessException	The <a href="#">TokenProvider</a> object couldn't acquire a token, the token is invalid, or the token doesn't contain the claims required to do the operation.	Make sure the token provider is created with the correct values. Check the configuration of the Access Control Service.	Retry might help in some cases; add retry logic to code.
ArgumentException ArgumentNullException ArgumentOutOfRangeException	One or more arguments supplied to the method are invalid. The URI supplied to <a href="#">NamespaceManager</a> or <a href="#">Create</a> contains path segment(s). The URI scheme supplied to <a href="#">NamespaceManager</a> or <a href="#">Create</a> is invalid. The property value is larger than 32 KB.	Check the calling code and make sure the arguments are correct.	Retry will not help.
Microsoft.ServiceBus.Messaging MessagingEntityNotFoundException	Entity associated with the operation does not exist or it has been deleted.	Make sure the entity exists.	Retry will not help.
Microsoft.Azure.EventHubs.MessagingEntityNotFoundException			

Exception Type	Description/Cause/Examples	Suggested Action	Note on Automatic/Immediate Retry
<a href="#">MessagingCommunicationException</a>	Client is not able to establish a connection to Event Hub.	Make sure the supplied host name is correct and the host is reachable.	Retry might help if there are intermittent connectivity issues.
<a href="#">Microsoft.ServiceBus.Messaging.ServerBusyException</a>	Service is not able to process the request at this time.	Client can wait for a period of time, then retry the operation. See <a href="#">ServerBusyException</a> .	Client may retry after certain interval. If a retry results in a different exception, check retry behavior of that exception.
<a href="#">MessagingException</a>	Generic messaging exception that may be thrown in the following cases: An attempt is made to create a <a href="#">QueueClient</a> using a name or path that belongs to a different entity type (for example, a topic). An attempt is made to send a message larger than 1 MB. The server or service encountered an error during processing of the request. See the exception message for details. This exception is usually a transient exception.	Check the code and ensure that only serializable objects are used for the message body (or use a custom serializer). Check the documentation for the supported value types of the properties and only use supported types. Check the <a href="#">IsTransient</a> property. If it is <b>true</b> , you can retry the operation.	Retry behavior is undefined and might not help.
<a href="#">MessagingEntityAlreadyExistsException</a>	Attempt to create an entity with a name that is already used by another entity in that service namespace.	Delete the existing entity or choose a different name for the entity to be created.	Retry will not help.
<a href="#">QuotaExceededException</a>	The messaging entity has reached its maximum allowable size. This exception can happen if the maximum number of receivers (which is 5) has already been opened on a per-consumer group level.	Create space in the entity by receiving messages from the entity or its subqueues. See <a href="#">QuotaExceededException</a>	Retry might help if messages have been removed in the meantime.
<a href="#">MessagingEntityDisabledException</a>	Request for a runtime operation on a disabled entity.	Activate the entity.	Retry might help if the entity has been activated in the interim.
<a href="#">Microsoft.ServiceBus.Messaging.MessageSizeExceededException</a>  <a href="#">Microsoft.Azure.EventHubs.MessageSizeExceededException</a>	A message payload exceeds the 1-MB limit. This 1-MB limit is for the total message, which can include system properties and any .NET overhead.	Reduce the size of the message payload, then retry the operation.	Retry will not help.

## QuotaExceededException

[QuotaExceededException](#) indicates that a quota for a specific entity has been exceeded.

This exception can happen if the maximum number of receivers (5) has already been opened on a per-consumer group level.

## Event Hubs

Event Hubs has a limit of 20 consumer groups per Event Hub. When you attempt to create more, you receive a [QuotaExceededException](#).

## TimeoutException

A [TimeoutException](#) indicates that a user-initiated operation is taking longer than the operation timeout.

For Event Hubs, the timeout is specified either as part of the connection string, or through [ServiceBusConnectionStringBuilder](#). The error message itself might vary, but it always contains the timeout value specified for the current operation.

Timeouts are expected to happen during or in-between maintenance operations such as Event Hubs service updates (or) OS updates on resources that run the service. During OS updates, entities are moved around and nodes are updated or rebooted, which can cause timeouts. For service level agreement (SLA) details for the Azure Event Hubs service, see [SLA for Event Hubs](#).

### Common causes

There are two common causes for this error: incorrect configuration, or a transient service error.

- **Incorrect configuration** The operation timeout might be too small for the operational condition. The default value for the operation timeout in the client SDK is 60 seconds. Check to see if your code has the value set to something too small. The condition of the network and CPU usage can affect the time it takes for a particular operation to complete, so the operation timeout should not be set to a small value.
- **Transient service error** Sometimes the Event Hubs service can experience delays in processing requests; for example, during periods of high traffic. In such cases, you can retry your operation after a delay, until the operation is successful. If the same operation still fails after multiple attempts, visit the [Azure service status site](#) to see if there are any known service outages.

## ServerBusyException

A [Microsoft.ServiceBus.Messaging.ServerBusyException](#) or [Microsoft.Azure.EventHubs.ServerBusyException](#) indicates that a server is overloaded. There are two relevant error codes for this exception.

### Error code 50002

This error can occur for one of two reasons:

- The load isn't evenly distributed across all partitions on the event hub, and one partition hits the local throughput unit limitation.

**Resolution:** Revising the partition distribution strategy or trying [EventHubClient.Send\(eventDataWithOutPartitionKey\)](#) might help.

- The Event Hubs namespace doesn't have sufficient throughput units (you can check the **Metrics** screen in the Event Hubs namespace window in the [Azure portal](#) to confirm). The portal shows aggregated (1 minute) information, but we measure the throughput in real time – so it's only an estimate.

**Resolution:** Increasing the throughput units on the namespace can help.

You can configure throughput units on the **Scale** page or **Overview** page of your Event Hubs **namespace** page in the Azure portal. Or, you can use [Auto-inflate](#), which automatically scales up by increasing the number of throughput units, to meet usage needs.

Throughput units (TUs) apply to all event hubs in an Event Hubs namespace. It means that you purchase TUs at the namespace level and are shared among the event hubs under that namespace. Each TU entitles the namespace to the following capabilities:

- Up to 1 MB per second of ingress events (events sent into an event hub), but no more than 1000 ingress events, management operations, or control API calls per second.
- Up to 2 MB per second of egress events (events consumed from an event hub), but no more than 4096 egress events.
- Up to 84 GB of event storage (enough for the default 24-hour retention period).

On the [Overview](#) page, in the **Show metrics** section, switch to the **Throughput** tab. Select the chart to open it in a larger window with 1-minute intervals on the x-axis. Look at the peak values and divide them by 60 to get incoming bytes/second or outgoing bytes/second. Use similar approach to calculate number of requests per second at peak times on the **Requests** tab.

If you see values higher than number of TUs \* limits (1 MB per second for ingress or 1000 requests for ingress/second, 2 MB per second for egress), increase the number of TUs by using the **Scale** (on the left menu) page of an Event Hubs namespace to manually scale higher or to use the [Auto-inflate](#) feature of Event Hubs. Note that auto-Inflate can only increase up to 20 TUS. To raise it to exactly 40 TUs, submit a [support request](#).

## Error code 50008

This error should rarely occur. It happens when the container running code for your namespace is low on CPU – not more than a few seconds before the Event Hubs load balancer begins.

**Resolution:** Limit calls to the `GetRuntimeInformation` method. Azure Event Hubs supports up to 50 calls per second per consumer group to the `GetRuntimeInfo` per second. You may receive an exception similar to the following one once the limit is reached:

```
ExceptionId: 0000000000-0000-0000-a48a-9c908fbe84f6-ServerBusyException: The request was terminated because the namespace 75248:aaa-default-eventhub-ns-prodb2b is being throttled. Error code : 50008. Please wait 10 seconds and try again.
```

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)
- [Create an Event Hub](#)
- [Event Hubs FAQ](#)

# Azure Event Hubs - Resource Manager exceptions

11/26/2021 • 3 minutes to read • [Edit Online](#)

This article lists exceptions generated when interacting with Azure Event Hubs using Azure Resource Manager - via templates or direct calls.

## IMPORTANT

This document is frequently updated. Please check back for updates.

The following sections provide various exceptions/errors that are surfaced through Azure Resource Manager.

## Error code: Conflict

ERROR CODE	ERROR SUBCODE	ERROR MESSAGE	DESCRIPTION	RECOMMENDATION
Conflict	40300	The maximum number of resources of type EventHub has been reached or exceeded. Actual: #, Max allowed: #	The namespace has reached its <a href="#">quota</a> for the number of Event Hubs it can contain.	Delete any unused or extraneous event hubs from the namespace or consider upgrading to a <a href="#">dedicated cluster</a> .
Conflict	none	Disaster recovery (DR) config can't be deleted because replication is in progress. Fail over or break pairing before attempting to delete the DR Config.	<a href="#">GeoDR replication</a> is in progress, so the config can't be deleted at this time.	To unblock deletion of the config, either wait until replication has completed, trigger a failover, or break the GeoDR pairing.
Conflict	none	Namespace update failed with conflict in backend.	Another operation is currently being done on this namespace.	Wait until the current operation completes, and then retry.

## Error code: 429

ERROR CODE	ERROR SUBCODE	ERROR MESSAGE	DESCRIPTION	RECOMMENDATION
429	none	Namespace provisioning in transition	Another operation is currently being done on this namespace.	Wait until the current operation completes, and then retry.
429	none	Disaster recovery operation in progress.	A <a href="#">GeoDR</a> operation is currently being done on this namespace or pairing.	Wait until the current GeoDR operation completes, and then retry.

## Error code: BadRequest

Error code	Error subcode	Error message	Description	Recommendation
BadRequest	40000	PartitionCount can't be changed for an event hub.	Basic, standard, or premium tier of Azure Event Hubs doesn't support changing partitions.	Create a new event hub with the wanted number of partitions in your basic, standard, or premium tier namespace. Partition scale-out is supported for <a href="#">dedicated clusters</a> .
BadRequest	40000	The value '#' for MessageRetentionInDays isn't valid for the Basic tier. the value can't exceed '1' day(s).	Basic tier Event Hubs namespaces only support message retention of up to 1 day.	If more than one day of message retention is wanted, <a href="#">create a standard Event Hubs namespace</a> .
BadRequest	none	The specified name isn't available.	Namespace names must be unique, and the specified name is already taken.	If you're the owner of the existing namespace with the specified name, you can delete it, which will cause data loss. Then, try again with the same name. If the namespace isn't safe to delete (or you aren't the owner), choose another namespace name.
BadRequest	none	The specified subscription has reached its quota of namespaces.	Your subscription has reached the <a href="#">quota</a> for the number of namespaces it can hold.	Consider deleting unused namespaces in this subscription, creating another subscription, or upgrading to a <a href="#">dedicated cluster</a> .
BadRequest	none	Can't update a namespace that is secondary	The namespace can't be updated because it's the secondary namespace in a <a href="#">GeoDR pairing</a> .	If appropriate, make the change to the primary namespace in this pairing instead. Otherwise break the GeoDR pairing to make the change.
BadRequest	none	Can't set Auto-Inflate in basic SKU	Auto-Inflate can't be enabled on basic tier Event Hubs namespaces.	To <a href="#">enable Auto Inflate</a> on a namespace, make sure it's of standard or premium tier.

Error code	Error subcode	Error message	Description	Recommendation
BadRequest	none	There isn't enough capacity to create the namespace. Contact your Event Hubs administrator.	The selected region is at capacity and more namespaces can't be created.	Select another region to house your namespace.
BadRequest	none	The operation can't be done on entity type 'ConsumerGroup' because the namespace 'namespace name' is using 'Basic' tier.	Basic tier Event Hubs namespaces have a quota(event-hubs-quotas.md) of one consumer group (the default). Creating more consumer groups isn't supported.	Continue using the default consumer group (\$Default), or if more are needed, consider using a standard or premium tier Event Hubs namespace instead.
BadRequest	none	The namespace 'namespace name' doesn't exist.	The namespace provided couldn't be found.	Double check that the namespace name is correct and can be found in your subscription. If it isn't, <a href="#">create an Event Hubs namespace</a> .
BadRequest	none	The location property of the resource doesn't match its containing Namespace.	Creating an event hub in a specific region failed because it didn't match the region of the namespace.	Try creating the event hub in the same region as the namespace.

## Error code: Internal server error

Error code	Error subcode	Error message	Description	Recommendation
Internal Server Error	none	Internal Server Error.	The Event Hubs service had an internal error.	Retry the failing operation. If the operation continues to fail, contact support.

# Move an Azure Event Hubs namespace to another region

11/26/2021 • 4 minutes to read • [Edit Online](#)

This article shows you how to export an Azure Resource Manager template for an existing Event Hubs namespace and then use the template to create a namespace with same configuration settings in another region. However, this process doesn't move events that aren't processed yet. You need to process the events from the original namespace before deleting it.

If you have other resources in the Azure resource group that contains the Event Hubs namespace, you may want to export the template at the resource group level so that all related resources can be moved to the new region in one step. The steps in this article show you how to export a **namespace** to the template. The steps for exporting a **resource group** to the template are similar.

## Prerequisites

- Ensure that the services and features that your account uses are supported in the target region.
- If you have **capture feature** enabled for event hubs in the namespace, move [Azure Storage or Azure Data Lake Store Gen 2](#) or [Azure Data Lake Store Gen 1](#) accounts before moving the Event Hubs namespace. You can also move the resource group that contains both Storage and Event Hubs namespaces to the other region by following steps similar to the ones described in this article.
- If the Event Hubs namespace is in an **Event Hubs cluster**, [move the dedicated cluster](#) to the **target region** before you go through steps in this article. You can also use the [quickstart template on GitHub](#) to create an Event Hubs cluster. In the template, remove the namespace portion of the JSON to create only the cluster.

## Prepare

To get started, export a Resource Manager template. This template contains settings that describe your Event Hubs namespace.

1. Sign in to the [Azure portal](#).
2. Select **All resources** and then select your Event Hubs namespace.
3. On the **Event Hubs Namespace** page, select **Export template** under **Automation** in the left menu.
4. Choose **Download** in the **Export template** page.

The screenshot shows the Azure portal interface for managing an Event Hubs Namespace. The left sidebar lists various service categories like Geo-Recovery, Networking, Encryption, Properties, Locks, Entities (Event Hubs), Schema Registry, Monitoring (Alerts, Metrics, Diagnostic settings, Logs), Automation, and Tasks (preview). The main content area shows the 'Export template' page for the namespace 'spehub1020ns'. At the top, there are buttons for Download, Add to library (preview), Deploy, and Visualize template. Below these are sections for Parameters (2), Variables (0), and Resources (11). The central part of the screen displays a JSON template with code lines numbered 1 through 23. The template defines a schema, content version, parameters (including a namespace name and storage account external ID), variables, and resources (an Event Hub namespace resource). The 'Export template' button at the bottom left of the main content area is highlighted with a red box.

5. Locate the .zip file that you downloaded from the portal, and unzip that file to a folder of your choice.

This zip file contains the .json files that include the template and scripts to deploy the template.

## Move

Deploy the template to create an Event Hubs namespace in the target region.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and select **Template deployment (deploy using custom templates)**.
3. Select **Build your own template in the editor**.
4. Select **Load file**, and then follow the instructions to load the **template.json** file that you downloaded in the last section.
5. Update the value of the `location` property to point to the new region. To obtain location codes, see [Azure locations](#). The code for a region is the region name with no spaces, for example, `West US` is equal to `westus`.
6. Select **Save** to save the template.
7. On the **Custom deployment** page, follow these steps:
  - a. Select an **Azure subscription**.
  - b. Select an existing **resource group** or create one. If the source namespace was in an Event Hubs cluster, select the resource group that contains cluster in the target region.
  - c. Select the target **location** or region. If you selected an existing resource group, this setting is read-only.
  - d. In the **SETTINGS** section, do the following steps:
    - a. Enter the new **namespace name**.

- b. If your source namespace was in an **Event Hubs cluster**, enter names of **resource group** and **Event Hubs cluster** as part of **external ID**.

```
/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<CLUSTER'S RESOURCE GROUP>/providers/Microsoft.EventHub/clusters/<CLUSTER NAME>
```

- c. If event hub in your namespace uses a Storage account for capturing events, specify the resource group name and the storage account for

`StorageAccounts_<original storage account name>_external` field.

```
/subscriptions/0000000000-0000-0000-000000000000/resourceGroups/<STORAGE'S RESOURCE GROUP>/providers/Microsoft.Storage/storageAccounts/<STORAGE ACCOUNT NAME>
```

- e. Select **Review + create** at the bottom of the page.  
f. On the **Review + create** page, review settings, and then select **Create**.

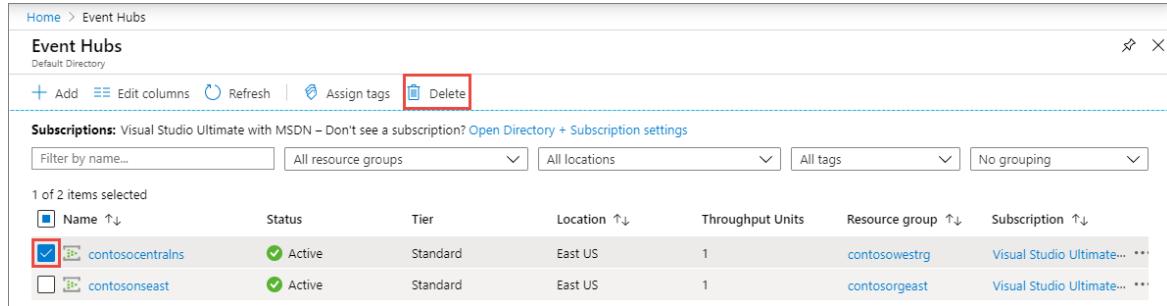
## Discard or clean up

After the deployment, if you want to start over, you can delete the target **Event Hubs namespace**, and repeat the steps described in the **Prepare** and **Move** sections of this article.

To commit the changes and complete the move of an Event Hubs namespace, delete the **Event Hubs namespace** in the original region. Make sure that you processed all the events in the namespace before deleting the namespace.

To delete an Event Hubs namespace (source or target) by using the Azure portal:

1. In the search window at the top of Azure portal, type **Event Hubs**, and select **Event Hubs** from search results. You see the Event Hubs namespaces in a list.
2. Select the target namespace to delete, and select **Delete** from the toolbar.



The screenshot shows the Azure Event Hubs management portal. At the top, there's a navigation bar with 'Home > Event Hubs'. Below it, a toolbar has a 'Delete' button highlighted with a red box. The main area is titled 'Subscriptions' and shows two items: 'contosocentrals' and 'contosonseast'. Both are listed as 'Active' with 'Standard' tier, located in 'East US' with 1 throughput unit. They belong to resource groups 'contosowestrg' and 'contosorgeast' respectively, under the 'Visual Studio Ultimate...' subscription. There are filter and grouping options at the top of the list.

Name	Status	Tier	Location	Throughput Units	Resource group	Subscription
contosocentrals	Active	Standard	East US	1	contosowestrg	Visual Studio Ultimate...
contosonseast	Active	Standard	East US	1	contosorgeast	Visual Studio Ultimate...

3. On the **Delete Namespace** page, confirm the deletion by typing the **namespace name**, and then select **Delete**.

## Next steps

In this tutorial, you moved an Azure Event Hubs namespace from one region to another and cleaned up the source resources. To learn more about moving resources between regions and disaster recovery in Azure, refer to:

- [Move resources to a new resource group or subscription](#)
- [Move Azure VMs to another region](#)

# Move an Azure Event Hubs dedicated cluster to another region

11/26/2021 • 3 minutes to read • [Edit Online](#)

This article shows you how to export an Azure Resource Manager template for an existing Event Hubs dedicated cluster and then use the template to create a cluster with same configuration settings in another region.

If you have other resources such as namespaces and event hubs in the Azure resource group that contains the Event Hubs cluster, you may want to export the template at the resource group level so that all related resources can be moved to the new region in one step. The steps in this article show you how to export an **Event Hubs cluster** to the template. The steps for exporting a **resource group** to the template are similar.

## Prerequisites

Ensure that the dedicated cluster can be created in the target region. The easiest way to find out is to use the Azure portal to try to [create an Event Hubs dedicated cluster](#). You see the list of regions that are supported at that point of time for creating the cluster.

## Prepare

To get started, export a Resource Manager template. This template contains settings that describe your Event Hubs dedicated cluster.

1. Sign in to the [Azure portal](#).
2. Select **All resources** and then select your Event Hubs dedicated cluster.
3. Select > **Settings** > **Export template**.
4. Choose **Download** in the **Export template** page.

The screenshot shows the 'Export template' page for a 'SPEventHubCluster' resource. The page has a header with 'Search (Ctrl+/' and a red box around the 'Download' button. Below the header are sections for 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', and 'Diagnose and solve problems'. On the left, there's a sidebar with 'Settings' (selected), 'Locks', 'Export template' (highlighted in grey), 'Monitoring' (with 'Alerts', 'Metrics', 'Diagnostic settings'), and 'Support + troubleshooting' (with 'New support request'). The main content area has tabs for 'Template', 'Parameters', and 'Scripts'. A note says 'To export related resources, select the resources from the Resource Group view then select the "Export template" option from the tool bar.' A checkbox 'Include parameters' is checked. The 'Template' tab shows a JSON code block:

```
$schema: "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
contentVersion: "1.0.0.0",
parameters: {
  "clusters_SPEventHubCluster_name": {
    "defaultValue": "SPEventHubCluster",
    "type": "String"
  }
},
variables: {},
resources: [
  {
    "type": "Microsoft.EventHub/clusters",
    "apiVersion": "2018-01-01-preview",
    "name": "[parameters('clusters_SPEventHubCluster_name')]",
    "location": "eastus",
    "sku": {
      "name": "Dedicated",
      "capacity": 1
    },
    "properties": {}
  }
]
```

5. Locate the .zip file that you downloaded from the portal, and unzip that file to a folder of your choice.

This zip file contains the json files that include the template and scripts to deploy the template.

## Move

Deploy the template to create an Event Hubs dedicated cluster in the target region.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and select **Template deployment (deploy using custom templates)**.
3. Select **Build your own template in the editor**.
4. Select **Load file**, and then follow the instructions to load the **template.json** file that you downloaded in the last section.
5. Update the value of the `location` property to point to the new region. To obtain location codes, see [Azure locations](#). The code for a region is the region name with no spaces, for example, `West US` is equal to `westus`.
6. Select **Save** to save the template.
7. On the **Custom deployment** page, follow these steps:
  - a. Select an **Azure subscription**.
  - b. Select an existing **resource group** or create one.
  - c. Select the target **location** or region. If you selected an existing resource group, this setting is read-only.
  - d. In the **SETTINGS** section, do the following steps:
    - a. Enter the new **cluster name**.

**Custom deployment**  
Deploy from a custom template

Select a template    Basics    Review + create

**Template**

Customized template 1 resource

Edit template    Edit parameters

**Deployment scope**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* <Azure subscription name>

Resource group \* (New) spehubclusterwestrg

Create new

**Parameters**

Region \* East US

Clusters\_SPEvent Hub Cluster\_name SPEventHubWestCluster

---

Review + create    < Previous    Next : Review + create >

- e. Select **Review + create** at the bottom of the page.
- f. On the **Review + create** page, review settings, and then select **Create**.

## Discard or clean up

After the deployment, if you want to start over, you can delete the target Event Hubs dedicated cluster, and

repeat the steps described in the [Prepare](#) and [Move](#) sections of this article.

To commit the changes and complete the move of an Event Hubs cluster, delete the **Event Hubs cluster** in the original region.

To delete an Event Hubs cluster (source or target) by using the Azure portal:

1. In the search window at the top of Azure portal, type **Event Hubs Clusters**, and select **Event Hubs Clusters** from search results. You see the Event Hubs cluster in a list.
2. Select the cluster to delete, and select **Delete** from the toolbar.
3. On the **Delete Cluster** page, confirm the deletion by typing the **cluster name**, and then select **Delete**.

## Next steps

In this tutorial, you learned how to move an Event Hubs dedicated cluster from one region to another.

See the [Move Event Hubs namespaces across regions](#) article for instructions on moving a namespace from one region to another region.

To learn more about moving resources between regions and disaster recovery in Azure, refer to:

- [Move resources to a new resource group or subscription](#)
- [Move Azure VMs to another region](#)

# Monitoring Azure Event Hubs data reference

11/26/2021 • 10 minutes to read • [Edit Online](#)

See [Monitoring Azure Event Hubs](#) for details on collecting and analyzing monitoring data for Azure Event Hubs.

## Metrics

This section lists all the automatically collected platform metrics collected for Azure Event Hubs. The resource provider for these metrics is **Microsoft.EventHub/clusters** or **Microsoft.EventHub/clusters**.

### Request metrics

Counts the number of data and management operations requests.

Metric Name	Exportable via Diagnostic Settings	Unit	Aggregation Type	Description	Dimensions
Incoming Requests	Yes	Count	Total	The number of requests made to the Event Hubs service over a specified period.	Entity name
Successful Requests	No	Count	Total	The number of successful requests made to the Event Hubs service over a specified period.	Entity name Operation Result
Throttled Requests	No	Count	Total	The number of requests that were throttled because the usage was exceeded.	Entity name Operation Result

The following two types of errors are classified as **user errors**:

1. Client-side errors (In HTTP that would be 400 errors).
2. Errors that occur while processing messages.

### Message metrics

Metric Name	Exportable via Diagnostic Settings	Unit	Aggregation Type	Description	Dimensions
-------------	------------------------------------	------	------------------	-------------	------------

Metric Name	Exportable via Diagnostic Settings	Unit	Aggregation Type	Description	Dimensions
Incoming Messages	Yes	Count	Total	The number of events or messages sent to Event Hubs over a specified period.	Entity name
Outgoing Messages	Yes	Count	Total	The number of events or messages received from Event Hubs over a specified period.	Entity name
Captured Messages	No	Count	Total	The number of captured messages.	Entity name
Incoming Bytes	Yes	Bytes	Total	Incoming bytes for an event hub over a specified period.	Entity name
Outgoing Bytes	Yes	Bytes	Total	Outgoing bytes for an event hub over a specified period.	Entity name
Size	No	Bytes	Average	Size of an event hub in bytes.	Entity name

#### NOTE

These values are point-in-time values. Incoming messages that were consumed immediately after that point-in-time may not be reflected in these metrics.

#### Capture metrics

Metric Name	Exportable via Diagnostic Settings	Unit	Aggregation Type	Description	Dimensions
Captured Messages	No	Count	Total	The number of captured messages.	Entity name
Captured Bytes	No	Bytes	Total	Captured bytes for an event hubs	Entity name
Capture Backlog	No	Count	Total	Capture backlog for an event hubs	Entity name

## Connection metrics

Metric Name	Exportable via Diagnostic Settings	Unit	Aggregation Type	Description	Dimensions
Active Connections	No	Count	Average	The number of active connections on a namespace and on an entity (event hub) in the namespace. Value for this metric is a point-in-time value. Connections that were active immediately after that point-in-time may not be reflected in the metric.	Entity name
Connections Opened	No	Count	Average	The number of open connections.	Entity name
Connections Closed	No	Count	Average	The number of closed connections.	Entity name

## Error metrics

Metric Name	Exportable via Diagnostic Settings	Unit	Aggregation Type	Description	Dimensions
Server Errors	No	Count	Total	The number of requests not processed because of an error in the Event Hubs service over a specified period.	Entity name Operation Result
User Errors	No	Count	Total	The number of requests not processed because of user errors over a specified period.	Entity name Operation Result
Quota Exceeded Errors	No	Count	Total	The number of errors caused by exceeding quotas over a specified period.	Entity name Operation Result

**NOTE**

Logic Apps creates epoch receivers and receivers may be moved from one node to another depending on the service load. During those moves, `ReceiverDisconnection` exceptions may occur. They are counted as user errors on the Event Hubs service side. Logic Apps may collect failures from Event Hubs clients so that you can view them in user logs.

## Metric dimensions

Azure Event Hubs supports the following dimensions for metrics in Azure Monitor. Adding dimensions to your metrics is optional. If you don't add dimensions, metrics are specified at the namespace level.

DIMENSION NAME	DESCRIPTION
Entity Name	Name of the event hub.

## Resource logs

Event Hubs captures diagnostic logs for the following categories:

CATEGORY	DESCRIPTION
Archive Logs	Captures information about <a href="#">Event Hubs Capture</a> operations, specifically, logs related to capture errors.
Operational Logs	Capture all management operations that are performed on the Azure Event Hubs namespace. Data operations aren't captured, because of the high volume of data operations that are conducted on Azure Event Hubs.
Auto scale logs	Captures auto-inflate operations done on an Event Hubs namespace.
Kafka coordinator logs	Captures Kafka coordinator operations related to Event Hubs.
Kafka user error logs	Captures information about Kafka APIs called on Event Hubs.
Event Hubs virtual network (VNet) connection event	Captures information about IP addresses and virtual networks sending traffic to Event Hubs.
Customer-managed key user logs	Captures operations related to customer-managed key.

All logs are stored in JavaScript Object Notation (JSON) format. Each entry has string fields that use the format described in the following sections.

### Archive logs schema

Archive log JSON strings include elements listed in the following table:

NAME	DESCRIPTION
<code>TaskName</code>	Description of the task that failed

NAME	DESCRIPTION
ActivityId	Internal ID, used for tracking
trackingId	Internal ID, used for tracking
resourceId	Azure Resource Manager resource ID
eventHub	Event hub full name (includes namespace name)
partitionId	Event Hub partition being written to
archiveStep	possible values: ArchiveFlushWriter, DestinationInit
startTime	Failure start time
failures	Number of times the failure occurred
durationInSeconds	Duration of failure
message	Error message
category	ArchiveLogs

The following code is an example of an archive log JSON string:

```
{
  "TaskName": "EventHubArchiveUserError",
  "ActivityId": "00000000-0000-0000-000000000000",
  "trackingId": "000000-0000-0000-0000000000000000",
  "resourceId": "/SUBSCRIPTIONS/00000000-0000-0000-0000-000000000000/RESOURCEGROUPS/<Resource Group Name>/PROVIDERS/MICROSOFT.EVENTHUB/NAMESPACES/<Event Hubs Namespace Name>",
  "eventHub": "<Event Hub full name>",
  "partitionId": "1",
  "archiveStep": "ArchiveFlushWriter",
  "startTime": "9/22/2016 5:11:21 AM",
  "failures": 3,
  "durationInSeconds": 360,
  "message": "Microsoft.WindowsAzure.Storage.StorageException: The remote server returned an error: (404) Not Found. ---> System.Net.WebException: The remote server returned an error: (404) Not Found.\r\n at Microsoft.WindowsAzure.Storage.Shared.Protocol.HttpResponseParsers.ProcessExpectedStatusCodeNoException[T] (HttpStatusCode expectedStatusCode, HttpStatusCode actualStatusCode, T retVal, StorageCommandBase`1 cmd, Exception ex)\r\n at Microsoft.WindowsAzure.Storage.Blob.CloudBlockBlob.<PutBlockImpl>b__3e(RESTCommand`1 cmd, HttpWebResponse resp, Exception ex, OperationContext ctx)\r\n at Microsoft.WindowsAzure.Storage.Core.Executor.Executor.EndGetResponse[T](IAsyncResult getResponseResult)\r\n --- End of inner exception stack trace ---\r\n at Microsoft.WindowsAzure.Storage.Core.Util.StorageAsyncResult`1.End()\r\n at Microsoft.WindowsAzure.Storage.Core.Util.AsyncExtensions.<>c__DisplayClass4.<CreateCallbackVoid>b__3(IAsyncResult ar)\r\n --- End of stack trace from previous location where exception was thrown ---\r\n at System.",
  "category": "ArchiveLogs"
}
```

## Operational logs schema

Operational log JSON strings include elements listed in the following table:

NAME	DESCRIPTION
ActivityId	Internal ID, used for tracking purposes
EventName	Operation name. For a list of values for this element, see the <a href="#">Event names</a>
resourceId	Azure Resource Manager resource ID
SubscriptionId	Subscription ID
EventTimeString	Operation time
EventProperties	Properties for the operation. This element provides more information about the event as shown in the following example.
Status	Operation status. The value can be either <b>Succeeded</b> or <b>Failed</b> .
Caller	Caller of operation (Azure portal or management client)
Category	OperationalLogs

The following code is an example of an operational log JSON string:

**Example:**

```
{
  "ActivityId": "00000000-0000-0000-0000-000000000000",
  "EventName": "Create EventHub",
  "resourceId": "/SUBSCRIPTIONS/00000000-0000-0000-0000-000000000000/RESOURCEGROUPS/<Resource Group Name>/PROVIDERS/MICROSOFT.EVENTHUB/NAMESPACES/<Event Hubs namespace name>",
  "SubscriptionId": "00000000-0000-0000-0000-000000000000",
  "EventTimeString": "9/28/2016 8:40:06 PM +00:00",
  "EventProperties": "{\"SubscriptionId\":\"00000000-0000-0000-0000-000000000000\", \"Namespace\":\"<Namespace Name>\", \"Via\":\"https://<Namespace Name>.servicebus.windows.net/f8096791adb448579ee83d30e006a13e/?api-version=2016-07\", \"TrackingId\":\"5ee74c9e-72b5-4e98-97c4-08a62e56e221_G1\"}",
  "Status": "Succeeded",
  "Caller": "ServiceBus Client",
  "category": "OperationalLogs"
}
```

#### Event names

Event name is populated as operation type + resource type from the following enumerations. For example,

Create Queue , Retrieve Event Hu , Or Delete Rule .

OPERATION TYPE	RESOURCE TYPE
----------------	---------------

OPERATION TYPE	RESOURCE TYPE
<ul style="list-style-type: none"> <li>• Create</li> <li>• Update</li> <li>• Delete</li> <li>• Retrieve</li> <li>• Unknown</li> </ul>	<ul style="list-style-type: none"> <li>• Namespace</li> <li>• Queue</li> <li>• Topic</li> <li>• Subscription</li> <li>• EventHub</li> <li>• EventHubSubscription</li> <li>• NotificationHub</li> <li>• NotificationHubTier</li> <li>• SharedAccessPolicy</li> <li>• UsageCredit</li> <li>• NamespacePnsCredentials</li> <li>• RuleConsumerGroup</li> </ul>

## Autoscale logs schema

Autoscale log JSON includes elements listed in the following table:

NAME	DESCRIPTION
TrackingId	Internal ID, which is used for tracing purposes
ResourceId	Azure Resource Manager resource ID.
Message	Informational message, which provides details about auto-inflate action. The message contains previous and current value of throughput unit for a given namespace and what triggered the inflate of the TU.

Here's an example autoscale event:

```
{
  "TrackingId": "fb1b3676-bb2d-4b17-85b7-be1c7aa1967e",
  "Message": "Scaled-up EventHub TUs (UpdateStartTimeUTC: 5/13/2021 7:48:36 AM, PreviousValue: 1,
UpdatedThroughputUnitValue: 2, AutoScaleReason: 'IncomingMessagesPerSecond reached 2170')",
  "ResourceId": "/subscriptions/0000000-0000-0000-0000-
000000000/resourcegroups/testrg/providers/microsoft.eventhub/namespaces/namespace-name"
}
```

## Kafka coordinator logs schema

Kafka coordinator log JSON includes elements listed in the following table:

NAME	DESCRIPTION
RequestId	Request ID, which is used for tracing purposes
ResourceId	Azure Resource Manager resource ID
Operation	Name of the operation that's done during the group coordination
ClientId	Client ID

NAME	DESCRIPTION
NamespaceName	Namespace name
SubscriptionId	Azure subscription ID
Message	Informational or warning message, which provides details about actions done during the group coordination.

#### Example

```
{
  "RequestId": "FE01001A89E30B020000000304620E2A_KafkaExampleConsumer#0",
  "Operation": "Join.Start",
  "ClientId": "KafkaExampleConsumer#0",
  "Message": "Start join group for new member namespace-name:c:$default:I:KafkaExampleConsumer#0-cc40856f7f3c4607915a571efe994e82, current group size: 0, API version: 2, session timeout: 10000ms, rebalance timeout: 30000ms.",
  "SubscriptionId": "0000000-0000-0000-0000-000000000000",
  "NamespaceName": "namespace-name",
  "ResourceId": "/subscriptions/0000000-0000-0000-0000-000000000000/resourcegroups/testrg/providers/microsoft.eventhub/namespaces/namespace-name",
  "Category": "KafkaCoordinatorLogs"
}
```

#### Kafka user error logs schema

Kafka user error log JSON includes elements listed in the following table:

NAME	DESCRIPTION
TrackingId	Tracking ID, which is used for tracing purposes.
NamespaceName	Namespace name
Eventhub	Event hub name
PartitionId	Partition ID
GroupId	Group ID
ClientId	Client ID
ResourceId	Azure Resource Manager resource ID.
Message	Informational message, which provides details about an error

#### Event Hubs virtual network connection event schema

Event Hubs virtual network (VNet) connection event JSON includes elements listed in the following table:

NAME	DESCRIPTION
SubscriptionId	Azure subscription ID

NAME	DESCRIPTION
NamespaceName	Namespace name
IPAddress	IP address of a client connecting to the Event Hubs service
Action	Action done by the Event Hubs service when evaluating connection requests. Supported actions are <b>Accept Connection</b> and <b>Deny Connection</b> .
Reason	Provides a reason why the action was done
Count	Number of occurrences for the given action
ResourceId	Azure Resource Manager resource ID.

Virtual network logs are generated only if the namespace allows access from **selected networks** or from **specific IP addresses** (IP filter rules). If you don't want to restrict the access to your namespace using these features and still want to get virtual network logs to track IP addresses of clients connecting to the Event Hubs namespace, you could use the following workaround. [Enable IP filtering](#), and add the total addressable IPv4 range (1.0.0.0/1 - 255.0.0.0/1). Event Hubs IP filtering doesn't support IPv6 ranges. You may see private endpoint addresses in the IPv6 format in the log.

#### Example

```
{
  "SubscriptionId": "0000000-0000-0000-0000-000000000000",
  "NamespaceName": "namespace-name",
  "IPAddress": "1.2.3.4",
  "Action": "Deny Connection",
  "Reason": "IPAddress doesn't belong to a subnet with Service Endpoint enabled.",
  "Count": "65",
  "ResourceId": "/subscriptions/0000000-0000-0000-0000-
  00000000000/resourcegroups/testrg/providers/microsoft.eventhub/namespaces/namespace-name",
  "Category": "EventHubVNetConnectionEvent"
}
```

#### Customer-managed key user logs schema

Customer-managed key user log JSON includes elements listed in the following table:

NAME	DESCRIPTION
Category	Type of category for a message. It's one of the following values: <b>error</b> and <b>info</b> . For example, if the key from your key vault is being disabled, then it would be an information category or if a key can't be unwrapped, it could fall under error.
ResourceId	Internal resource ID, which includes Azure subscription ID and namespace name
KeyVault	Name of the Key Vault resource
Key	Name of the Key Vault key that's used to encrypt the Event Hubs namespace.

NAME	DESCRIPTION
Version	Version of the Key Vault key.
Operation	The operation that's performed on the key in your key vault. For example, disable/enable the key, wrap, or unwrap.
Code	The code that's associated with the operation. Example: Error code, 404 means that key wasn't found.
Message	Message, which provides details about an error or informational message

Here's an example of the log for a customer managed key:

```
{
  "TaskName": "CustomerManagedKeyUserLog",
  "ActivityId": "11111111-1111-1111-1111-111111111111",
  "category": "error",
  "resourceId": "/SUBSCRIPTIONS/11111111-1111-1111-1111-111111111111/RESOURCEGROUPS/DEFAULT-EVENTHUB-CENTRALUS/PROVIDERS/MICROSOFT.EVENTHUB/NAMESPACES/FBETTATI-OPERA-EVENTHUB",
  "keyVault": "https://mykeyvault.vault-int.azure-int.net",
  "key": "mykey",
  "version": "11111111111111111111111111111111",
  "operation": "wrapKey",
  "code": "404",
  "message": "Key not found: ehbyok0/11111111111111111111111111111111"
}

{
  "TaskName": "CustomerManagedKeyUserLog",
  "ActivityId": "111111111111-1111-1111-111111111111",
  "category": "info",
  "resourceId": "/SUBSCRIPTIONS/11111111-1111-1111-1111-111111111111/RESOURCEGROUPS/DEFAULT-EVENTHUB-CENTRALUS/PROVIDERS/MICROSOFT.EVENTHUB/NAMESPACES/FBETTATI-OPERA-EVENTHUB",
  "keyVault": "https://mykeyvault.vault-int.azure-int.net",
  "key": "mykey",
  "version": "11111111111111111111111111111111",
  "operation": "disable" | "restore",
  "code": "",
  "message": ""
}
```

Following are the common errors codes to look for when BYOK encryption is enabled.

ACTION	ERROR CODE	RESULTING STATE OF DATA
Remove wrap/unwrap permission from a key vault	403	Inaccessible
Remove AAD role membership from an AAD principal that granted the wrap/unwrap permission	403	Inaccessible
Delete an encryption key from the key vault	404	Inaccessible

ACTION	ERROR CODE	RESULTING STATE OF DATA
Delete the key vault	404	Inaccessible (assumes soft-delete is enabled, which is a required setting.)
Changing the expiration period on the encryption key such that it's already expired	403	Inaccessible
Changing the NBF (not before) such that key encryption key isn't active	403	Inaccessible
Selecting the <b>Allow MSFT Services</b> option for the key vault firewall or otherwise blocking network access to the key vault that has the encryption key	403	Inaccessible
Moving the key vault to a different tenant	404	Inaccessible
Intermittent network issue or DNS/AAD/MSI outage		Accessible using cached data encryption key

## Azure Monitor Logs tables

Azure Event Hubs uses Kusto tables from Azure Monitor Logs. You can query these tables with Log Analytics. For a list of Kusto tables the service uses, see [Azure Monitor Logs table reference](#).

## Next steps

- For details on monitoring Azure Event Hubs, see [Monitoring Azure Event Hubs](#).
- For details on monitoring Azure resources, see [Monitoring Azure resources with Azure Monitor](#).

# Azure Event Hubs - Client SDKs

11/26/2021 • 2 minutes to read • [Edit Online](#)

This article provides following information for the SDKs supported by Azure Event Hubs:

- Location of package that you can use in your applications
- GitHub location where you can find source code, samples, readme, change log, reported issues, and also raise new issues
- Links to quickstart tutorials

## Client SDKs

The following table describes all the latest available Azure Event Hubs runtime clients. The core focus of these libraries is to **send and receive messages** from an event hub.

LANGUAGE	PACKAGE	REFERENCE
.NET Standard	<a href="#">Azure.Messaging.EventHubs</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
	<a href="#">Azure.Messaging.EventHubs.Processor</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
Java	<a href="#">azure-messaging-eventhubs</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
	<a href="#">azure-messaging-eventhubs-checkpointstore-blob</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
Python	<a href="#">azure-eventhub</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
	<a href="#">azure-eventhub-checkpointstoreblob-aio</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
JavaScript	<a href="#">azure/event-hubs</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
	<a href="#">azure/eventhubs-checkpointstore-blob</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>
Go	<a href="#">azure-event-hubs-go</a>	<ul style="list-style-type: none"><li>• <a href="#">GitHub location</a></li><li>• <a href="#">Tutorial</a></li></ul>

LANGUAGE	PACKAGE	REFERENCE
C	<a href="#">azure-event-hubs-c</a>	<ul style="list-style-type: none"> <li>• <a href="#">GitHub location</a></li> <li>• <a href="#">Tutorial</a></li> </ul>

The below table lists older Azure Event Hubs runtime clients. While these packages may receive critical bug fixes, they aren't in active development. We recommend using the latest SDKs listed in the above table instead.

LANGUAGE	PACKAGE	REFERENCE
.NET Standard	<a href="#">Microsoft.Azure.EventHubs (legacy)</a>	<ul style="list-style-type: none"> <li>• <a href="#">GitHub location</a></li> <li>• <a href="#">Tutorial</a></li> </ul>
	<a href="#">Microsoft.Azure.EventHubs.Processor (legacy)</a>	<ul style="list-style-type: none"> <li>• <a href="#">GitHub location</a></li> <li>• <a href="#">Tutorial</a></li> </ul>
.NET Framework	<a href="#">WindowsAzure.Messaging (legacy)</a>	
Java	<a href="#">azure-eventhubs (legacy)</a>	<ul style="list-style-type: none"> <li>• <a href="#">GitHub location</a></li> <li>• <a href="#">Tutorial</a></li> </ul>

## Management SDKs

Here's a list of currently available management-specific libraries. None of these libraries contain runtime operations, and are for the sole purpose of **managing Event Hubs entities**.

- [.NET Standard](#)
- [Java](#)
- [Python](#)
- [JavaScript](#)

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)
- [Create an Event Hub](#)
- [Event Hubs FAQ](#)

# Azure Event Hubs quotas and limits

11/26/2021 • 3 minutes to read • [Edit Online](#)

The following tables provide quotas and limits specific to [Azure Event Hubs](#). For information about Event Hubs pricing, see [Event Hubs pricing](#).

## Common limits for all tiers

The following limits are common across all tiers.

LIMIT	NOTES	VALUE
Size of an event hub name	-	256 characters
Size of a consumer group name	Kafka protocol doesn't require the creation of a consumer group.	Kafka: 256 characters AMQP: 50 characters
Number of non-epoch receivers per consumer group	-	5
Number of authorization rules per namespace	Subsequent requests for authorization rule creation are rejected.	12
Number of calls to the GetRuntimeInformation method	-	50 per second
Number of virtual networks (VNet)	-	128
Number of IP Config rules	-	128
Maximum length of a schema group name		50
Maximum length of a schema name		100
Size in bytes per schema		1 MB
Number of properties per schema group		1024
Size in bytes per schema group property key		256
Size in bytes per schema group property value		1024

## Basic vs. standard vs. premium vs. dedicated tiers

The following table shows limits that may be different for basic, standard, and dedicated tiers. In the table CU is

capacity unit, PU is processing unit, TU is throughput unit.

LIMIT	BASIC	STANDARD	PREMIUM	DEDICATED
Maximum size of Event Hubs publication	256 KB	1 MB	1 MB	1 MB
Number of consumer groups per event hub	1	20	100	1000 No limit per CU
Number of brokered connections per namespace	100	5,000	10000 per processing unit per PU	100, 000 per CU
Maximum retention period of event data	1 day	7 days	90 days 1 TB per PU	90 days 10 TB per CU
Maximum TUs or PUs or CUs	40 TUs	40 TUs	16 PUs	20 CUs
Number of partitions per event hub	32	32	100 per event hub, 200 per PU	1024 per event hub 2000 per CU
Number of namespaces per subscription	1000	1000	1000	1000 (50 per CU)
Number of event hubs per namespace	10	10	100 per PU	1000
Capture	N/A	Pay per hour	Included	Included
Size of the schema registry (namespace) in mega bytes	N/A	25	100	1024
Number of schema groups in a schema registry or namespace	N/A	1 - excluding the default group	100 1 MB per schema	1000 1 MB per schema
Number of schema versions across all schema groups	N/A	25	1000	10000
Throughput per unit	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	Ingress - 1 MB/s or 1000 events per second Egress – 2 MB/s or 4096 events per second	No limits per PU *	No limits per CU *

\* Depends on various factors such as resource allocation, number of partitions, storage, and so on.

**NOTE**

You can publish events individually or batched. The publication limit (according to SKU) applies regardless of whether it is a single event or a batch. Publishing events larger than the maximum threshold will be rejected.

The following table shows the list of features that are available (or not available) in a specific tier of Azure Event Hubs.

FEATURE	BASIC	STANDARD	PREMIUM	DEDICATED
Tenancy	Multi-tenant	Multi-tenant	Multi-tenant with resource isolation	Exclusive single tenant
Private link	N/A	Yes	Yes	Yes
Customer-managed key (Bring your own key)	N/A	N/A	Yes	Yes
Capture	N/A	Priced separately	Included	Included
Dynamic Partition scale out	N/A	N/A	Yes	Yes
Ingress events	Pay per million events	Pay per million events	Included	Included

**NOTE**

**Included** in the table means the feature is available and there's no separate charge for using it.

## Next steps

You can learn more about Event Hubs by visiting the following links:

- [Event Hubs overview](#)
- [Event Hubs Auto-inflate](#)
- [Event Hubs FAQ](#)

# Recommended configurations for Apache Kafka clients

11/26/2021 • 2 minutes to read • [Edit Online](#)

Here are the recommended configurations for using Azure Event Hubs from Apache Kafka client applications.

## Java client configuration properties

### Producer and consumer configurations

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>metadata.max.age.ms</code>	180000 (approximate)	< 240000	Can be lowered to pick up metadata changes sooner.
<code>connections.max.idle.ms</code>	180000	< 240000	Azure closes inbound TCP idle > 240,000 ms, which can result in sending on dead connections (shown as expired batches because of send timeout).

### Producer configurations only

Producer configs can be found [here](#).

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>max.request.size</code>	1000000	< 1046528	The service will close connections if requests larger than 1,046,528 bytes are sent. <i>This value must be changed and will cause issues in high-throughput produce scenarios.</i>
<code>retries</code>	> 0		May require increasing <code>delivery.timeout.ms</code> value, see documentation.
<code>request.timeout.ms</code>	30000 .. 60000	> 20000	EH will internally default to a minimum of 20,000 ms. <i>While requests with lower timeout values are accepted, client behavior isn't guaranteed.</i>

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>metadata.max.idle.ms</code>	180000	> 5000	Controls how long the producer will cache metadata for a topic that's idle. If the elapsed time since a topic was last produced exceeds the metadata idle duration, then the topic's metadata is forgotten and the next access to it will force a metadata fetch request.
<code>linger.ms</code>	> 0		For high throughput scenarios, linger value should be equal to the highest tolerable value to take advantage of batching.
<code>delivery.timeout.ms</code>			Set according to the formula ( $\text{request.timeout.ms} + \text{linger.ms}) * \text{retries}$
<code>compression.type</code>	<code>none</code>		Compression currently not supported..

## Consumer configurations only

Consumer configs can be found [here](#).

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>heartbeat.interval.ms</code>	3000		3000 is the default value and shouldn't be changed.
<code>session.timeout.ms</code>	30000	6000 .. 300000	Start with 30000, increase if seeing frequent rebalancing because of missed heartbeats.

## librdkafka configuration properties

The main `librdkafka` configuration file ([link](#)) contains extended descriptions for the properties below.

### Producer and consumer configurations

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>socket.keepalive.enable</code>	true		Necessary if connection is expected to idle. Azure will close inbound TCP idle > 240,000 ms.
<code>metadata.max.age.ms</code>	~ 180000	< 240000	Can be lowered to pick up metadata changes sooner.

## Producer configurations only

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>retries</code>	> 0		Default is 2. We recommend that you keep this value.
<code>request.timeout.ms</code>	30000 .. 60000	> 20000	EH will internally default to a minimum of 20,000 ms. librdkafka default value is 5000, which can be problematic. <i>While requests with lower timeout values are accepted, client behavior isn't guaranteed.</i>
<code>partitioner</code>	<code>consistent_random</code>	See librdkafka documentation	<code>consistent_random</code> is default and best. Empty and null keys are handled ideally for most cases.
<code>compression.codec</code>	<code>none</code>		Compression currently not supported.

## Consumer configurations only

PROPERTY	RECOMMENDED VALUES	PERMITTED RANGE	NOTES
<code>heartbeat.interval.ms</code>	3000		3000 is the default value and shouldn't be changed.
<code>session.timeout.ms</code>	30000	6000 .. 300000	Start with 30000, increase if seeing frequent rebalancing because of missed heartbeats.

## Further notes

Check the following table of common configuration-related error scenarios.

SYMPTOMS	PROBLEM	SOLUTION
Offset commit failures because of rebalancing	Your consumer is waiting too long in between calls to <code>poll()</code> and the service is kicking the consumer out of the group.	<p>You have several options:</p> <ul style="list-style-type: none"> <li>• Increase poll processing timeout ( <code>max.poll.interval.ms</code> )</li> <li>• Decrease message batch size to speed up processing</li> <li>• Improve processing parallelization to avoid blocking <code>consumer.poll()</code></li> </ul> <p>Applying some combination of the three is likely wisest.</p>

SYMPTOMS	PROBLEM	SOLUTION
Network exceptions at high produce throughput	Are you using Java client + default max.request.size? Your requests may be too large.	See Java configs above.

## Next steps

See [Azure subscription and service limits, quotas, and constraints](#) for quotas and limits of all Azure services.

# Azure Policy built-in definitions for Azure Event Hubs

11/26/2021 • 2 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy built-in policy definitions](#) for Azure Event Hubs. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Azure Event Hubs

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">All authorization rules except RootManageSharedAccessKey should be removed from Event Hub namespace</a>	Event Hub clients should not use a namespace level access policy that provides access to all queues and topics in a namespace. To align with the least privilege security model, you should create access policies at the entity level for queues and topics to provide access to only the specific entity	Audit, Deny, Disabled	<a href="#">1.0.1</a>
<a href="#">Authorization rules on the Event Hub instance should be defined</a>	Audit existence of authorization rules on Event Hub entities to grant least-privileged access	AuditIfNotExists, Disabled	<a href="#">1.0.0</a>
<a href="#">Configure Event Hub namespaces with private endpoints</a>	Private endpoints connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to Event Hub namespaces, you can reduce data leakage risks. Learn more at: <a href="https://docs.microsoft.com/azure/event-hubs/private-link-service">https://docs.microsoft.com/azure/event-hubs/private-link-service</a> .	DeployIfNotExists, Disabled	<a href="#">1.0.0</a>
<a href="#">Deploy Diagnostic Settings for Event Hub to Event Hub</a>	Deploys the diagnostic settings for Event Hub to stream to a regional Event Hub when any Event Hub which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	<a href="#">2.1.0</a>

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Deploy Diagnostic Settings for Event Hub to Log Analytics workspace</a>	Deploys the diagnostic settings for Event Hub to stream to a regional Log Analytics workspace when any Event Hub which is missing this diagnostic settings is created or updated.	DeployIfNotExists, Disabled	1.1.0
<a href="#">Event Hub namespaces should have double encryption enabled</a>	Enabling double encryption helps protect and safeguard your data to meet your organizational security and compliance commitments. When double encryption has been enabled, data in the storage account is encrypted twice, once at the service level and once at the infrastructure level, using two different encryption algorithms and two different keys.	Audit, Deny, Disabled	1.0.0
<a href="#">Event Hub namespaces should use a customer-managed key for encryption</a>	Azure Event Hubs supports the option of encrypting data at rest with either Microsoft-managed keys (default) or customer-managed keys. Choosing to encrypt data using customer-managed keys enables you to assign, rotate, disable, and revoke access to the keys that Event Hub will use to encrypt data in your namespace. Note that Event Hub only supports encryption with customer-managed keys for namespaces in dedicated clusters.	Audit, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
Event Hub namespaces should use private link	Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to Event Hub namespaces, data leakage risks are reduced. Learn more at: <a href="https://docs.microsoft.com/azure/event-hubs/private-link-service">https://docs.microsoft.com/azure/event-hubs/private-link-service</a> .	AuditIfNotExists, Disabled	1.0.0
Event Hub should use a virtual network service endpoint	This policy audits any Event Hub not configured to use a virtual network service endpoint.	AuditIfNotExists, Disabled	1.0.0
Resource logs in Event Hub should be enabled	Audit enabling of resource logs. This enables you to recreate activity trails to use for investigation purposes; when a security incident occurs or when your network is compromised	AuditIfNotExists, Disabled	5.0.0

## Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).