Connect your services together

Azure lets you create applications composed of various components: web site front-ends, back-end services, and triggered functions that perform compute-on-demand services. Azure also includes various communication strategies to let these various components pass data to each other. Learn how to leverage these communication services to create scalable, efficient solutions out of testable components.

Choose a messaging model in Azure to loosely connect your services

Communication strategies in Azure (APIs)

In the terminology of distributed applications, **messages** have the following characteristics:

- A message contains raw data, produced by one component, that will be consumed by another component.
- A message contains the data itself, not just a reference to that data.
- The sending component expects the message content to be processed in a certain way by the destination component. The integrity of the overall system may depend on both sender and receiver doing a specific job.

Events are lighter weight than messages, and are most often used for broadcast communications. The components sending the event are known as **publishers**, and receivers are known as **subscribers**.

With events, receiving components will generally decide in which communications they are interested, and will "subscribe" to those events. The subscription is managed by an intermediary, like Azure Event Grid or Azure Event Hubs. When publishers send an event, the intermediary will route that event to interested subscribers. This pattern is known as a "publish-subscribe architecture." It's not the only way to deal with events, but it is the most common.

Events have the following characteristics:

- An event is a lightweight notification that indicates that something happened.
- The event may be sent to multiple receivers, or to none at all.
- Events are often intended to "fan out," or have a large number of subscribers for each publisher.
- The publisher of the event has no expectation about the action a receiving component takes.
- Some events are discrete units and unrelated to other events.
- Some events are part of a related and ordered series.

Choose a message-based delivery with queues

Queue storage is a service that uses Azure Storage to store large numbers of messages that can be securely accessed from anywhere in the world using a simple REST-based interface. Queues can contain millions of messages, limited only by the capacity of the storage account that owns it

Service Bus is a message broker system intended for enterprise applications. These apps often utilize multiple communication protocols, have different data contracts, higher security requirements, and can include both cloud and on-premises services. Service Bus is built on top of a dedicated messaging infrastructure designed for exactly these scenarios.

Both of these services are based on the idea of a "queue" which holds sent messages until the target is ready to receive them.

Choose Service Bus Topics if

- you need multiple receivers to handle each message

Choose Service Bus queues if:

- You need an At-Most-Once delivery guarantee.
- You need a FIFO guarantee.
- You need to group messages into transactions.
- You want to receive messages without polling the queue.
- You need to provide a role-based access model to the queues.
- You need to handle messages larger than 64 KB but less than 256 KB.
- Your queue size will not grow larger than 80 GB.
- You would like to be able to publish and consume batches of messages.

Queue storage isn't quite as feature-rich, but if you don't need any of those features, it can be a simpler choice. In addition, it's the best solution if your app has any of the following requirements.

Choose Queue storage if:

- You need an audit trail of all messages that pass through the queue.
- You expect the queue to exceed 80 GB in size.
- You want to track progress for processing a message inside of the queue.

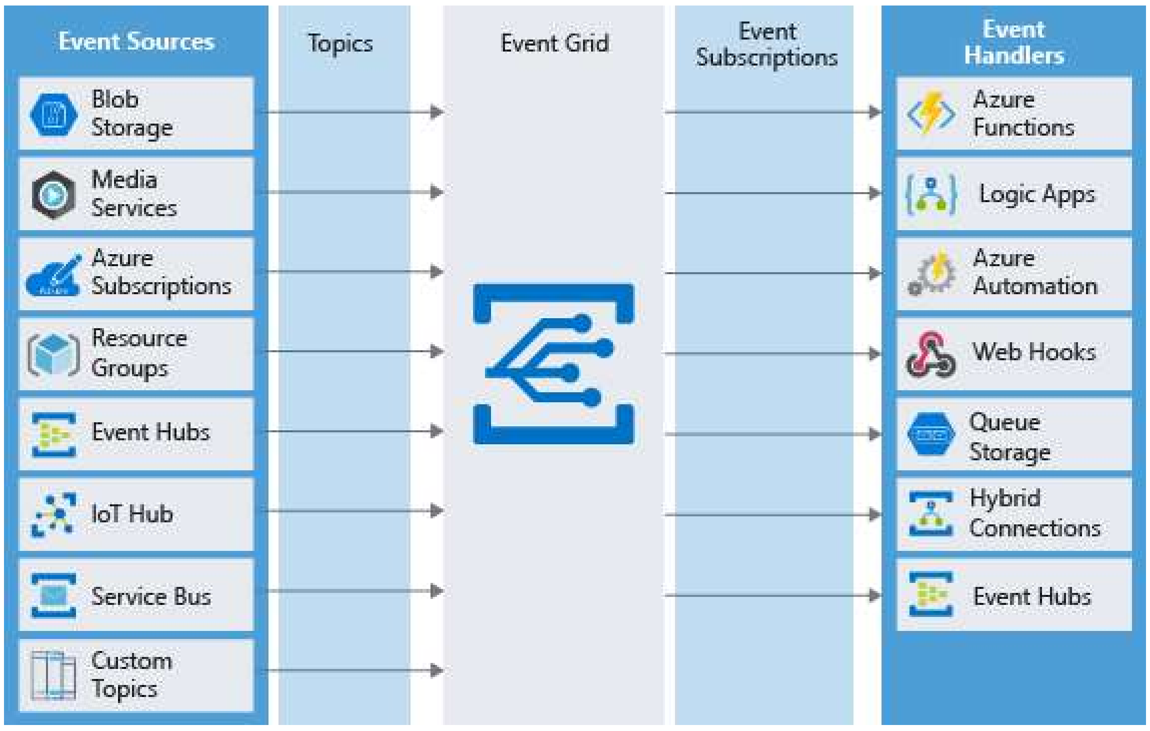
A queue is a simple, temporary storage location for messages sent between the components of a distributed application. Use a queue to organize messages and gracefully handle unpredictable surges in demand.

Use Storage queues when you want a simple and easy-to-code queue system. For more advanced needs, use Service Bus queues. If you have multiple destinations for a single message, but need queue-like behavior, use topics.

Choose a event-based delivery

Azure Event Grid is a fully-managed event routing service running on top of Azure Service Fabric. Event Grid distributes events from different sources, such as Azure Blob storage accounts or Azure Media Services, to different handlers, such as Azure Functions or Webhooks. Event Grid was created to make it easier to build event-based and serverless applications on Azure.

Event Grid supports most Azure services as a publisher or subscriber and can be used with third-party services. It provides a dynamically scalable, low-cost, messaging system that allows publishers to notify subscribers about a status change. The following illustration shows Azure Event Grid receiving messages from multiple sources and distributing them to event handlers based on subscription.



Events are the data messages passing through Event Grid that describe what has taken place. Each event is self-contained, can be up to 64 KB, and contains several pieces of information based on a schema defined by Event Grid

Field	**Description**
topic	The full resource path to the event source. Event Grid provides this value.
subject	Publisher-defined path to the event subject.
id	The unique identifier for event.
eventType	One of the registered event types for this event source. This is a value you can create filters against, e.g. CustomerCreated, BlobDeleted, HttpRequestReceived...

eventTime	The time the event was generated based on the provider's UTC time.
data	Specific information that is relevant to the type of event. For example, an event about a new file being created in Azure Storage has details about the file, such as the lastTimeModified value. Or, an Event Hubs event has the URL of the Capture file. This field is optional.
dataVersion	The schema version of the data object. The publisher defines the schema version.
metadataVersion	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

Use Event Grid when you need these features:

- **Simplicity**: It is straightforward to connect sources to subscribers in Event Grid.
- **Advanced filtering**: Subscriptions have close control over the events they receive from a topic.
- **Fan-out**: You can subscribe to an unlimited number of endpoints to the same events and topics.
- **Reliability**: Event Grid retries event delivery for up to 24 hours for each subscription.
- **Pay-per-event**: Pay only for the number of events that you transmit.

Event Grid is a simple but versatile event distribution system. Use it to deliver discrete events to subscribers, which will receive those events reliably and quickly. We have one more messaging model to examine - what if we want to deliver a large stream of events? In this scenario, Event Grid isn't a great solution because it's designed for one-event-at-a-time delivery. Instead, we need to turn to another Azure service: Event Hubs.

Event Hubs is an intermediary for the publish-subscribe communication pattern. Unlike Event Grid, however, it is optimized for extremely high throughput, a large number of publishers, security, and resiliency.

Choose Event Hubs if:
- You need to support authenticating a large number of publishers.
- You need to save a stream of events to Data Lake or Blob storage.
- You need aggregation or analytics on your event stream.
- You need reliable messaging or resiliency.

Event Hubs lets you build a big data pipeline capable of processing millions of events per second with low latency. It can handle data from concurrent sources and route it to a variety of stream-processing infrastructures and analytics services. It enables real-time processing and supports repeated replay of stored raw data.

Summary

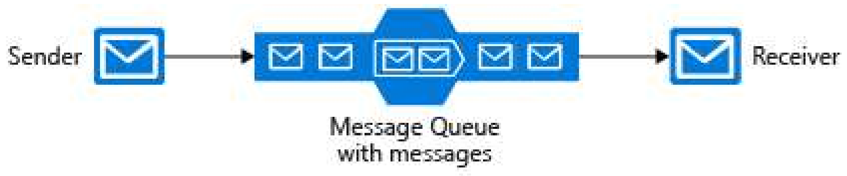
1. Suppose you have a distributed application with a web service that authenticates users. When a user logs on, the web service notifies all the client applications so they

can display that user's status as "Online". Is the login notification an example of a message or an event?
Event
The login notification is an event: it contains only a simple piece of status data and there is no expectation by the authentication service for the client applications to react to the notice in any particular way.

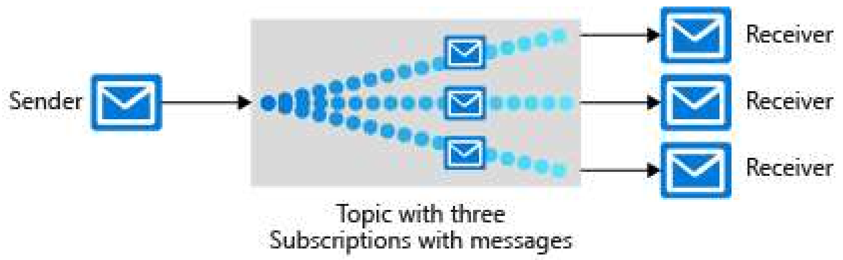
2. Suppose you have a distributed application with a web service that lets users manage their account. Users can sign up, edit their profile, and delete their account. When a user deletes their account, your web service notifies your data layer so the user's data will be removed from the database. Is the delete-account notification an example of a message or an event?
Message
The delete-account notification is a message. The key factor is that the web service has an expectation about how the data layer will process the message: the data layer must remove the user's data from the database for the system to function correctly. Note that the message itself contains only simple information so this aspect of the communication could be considered an event; however, the fact that the web service requires the data layer to handle the notification in a specific way is sufficient to make this a message.

Implement message-based communication workflows with Azure Service Bus

A **queue** is a simple temporary storage location for messages. A sending component adds a message to the queue. A destination component picks up the message at the front of the queue. Under ordinary circumstances, each message is received by only one receiver.



A **topic** is similar to a queue but can have multiple subscriptions. This means that multiple destination components can subscribe to a single topic, so each message is delivered to multiple receivers. Subscriptions can also filter the messages in the topic to receive only messages that are relevant. Subscriptions provide the same decoupled communications as queues and respond to high demand in the same way. Use a topic if you want each message to be delivered to more than one destination component.



A **relay** is an object that performs synchronous, two-way communication between applications. Unlike queues and topics, it is not a temporary storage location for messages. Instead, it provides bidirectional, unbuffered connections across network boundaries such as firewalls. Use a relay when you want direct communications between components as if they were located on the same network segment but separated by network security devices.

There are two Azure features that include message queues: Service Bus and Azure Storage accounts. As a general guide, storage queues are simpler to use but are less sophisticated and flexible than Service Bus queues.

Key advantages of Service Bus queues include:

- Supports larger messages sizes of 256 KB (standard tier) or 1MB (premium tier) per message versus 64 KB
- Supports both at-least-once and at-most-once delivery - choose between a very small chance that a message is lost or a very small chance it is handled twice
- Guarantees first-in-first-out (FIFO) order - messages are handled in the same order they are added (although FIFO is the normal operation of a queue, it is not guaranteed for every message)

- Can group multiple messages into a transaction - if one message in the transaction fails to be delivered, all messages in the transaction will not be delivered
- Supports role-based security
- Does not require destination components to continuously poll the queue

Advantages of storage queues:
- Supports unlimited queue size (versus 80-GB limit for Service Bus queues)
- Maintains a log of all messages

Choose Service Bus queues if:
- You need an at-most-once delivery guarantee
- You need a FIFO guarantee
- You need to group messages into transactions
- You want to receive messages without polling the queue
- You need to provide role-based access to the queues
- You need to handle messages larger than 64 KB but smaller than 256 KB
- Your queue size will not grow larger than 80 GB
- You would like to be able to publish and consume batches of messages

Choose queue storage if:
- You need a simple queue with no particular additional requirements
- You need an audit trail of all messages that pass through the queue
- You expect the queue to exceed 80 GB in size
- You want to track progress for processing a message inside of the queue

Filters can be one of three types:
- **Boolean Filters**. The TrueFilter ensures that all messages sent to the topic are delivered to the current subscription. The FalseFilter ensures that none of the messages are delivered to the current subscription. (This effectively blocks or switches off the subscription.)
- **SQL Filters**. A SQL filter specifies a condition by using the same syntax as a WHERE clause in a SQL query. Only messages that return True when evaluated against this subscription will be delivered to the subscribers.
- **Correlation Filters**. A correlation filter holds a set of conditions that are matched against the properties of each message. If the property in the filter and the property on the message have the same value, it is considered a match.

Summary

1. Which of the following queues should you use if you need first-in-first-out order and support for transactions?

Azure Service Bus queues

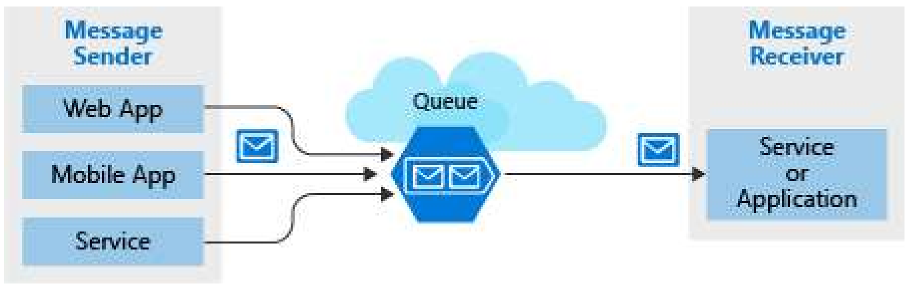
Azure Service Bus queues handle messages in the same order they're added and also support transactions. This means that if one message in a transaction fails to be added to the queue, all messages in the transaction will not be added.

2. Suppose you're sending a message with Azure Service Bus and you want multiple components to receive it. Which Azure Service Bus exchange feature should you use?
Topic
A topic allows multiple destination components to subscribe. This means that each message can be delivered to multiple receivers.

3. True or false: you can add a message to an Azure Service Bus queue that is 2 MB in size.
False
An Azure Service Bus queue message must be larger than 64 KB but smaller than 256 KB.

Communicate between applications with Azure Queue storage

Azure Queue storage is an Azure service that implements cloud-based queues. Each queue maintains a list of messages. Application components access a queue using a REST API or an Azure-supplied client library. Typically, you will have one or more sender components and one or more receiver components. Sender components add messages to the queue. Receiver components retrieve messages from the front of the queue for processing. The following illustration shows multiple sender applications adding messages to the Azure Queue and one receiver application retrieving the messages.



Why use queues?

A queue increases resiliency by temporarily storing waiting messages. At times of low or normal demand, the size of the queue remains small because the destination component removes messages from the queue faster than they are added. At times of high demand, the queue may increase in size, but messages are not lost. The destination component can catch up and empty the queue as demand returns to normal.

A single queue can be up to 500 TB in size, so it can potentially store millions of messages. The target throughput for a single queue is 2000 messages per second, allowing it to handle high-volume scenarios.

A queue must be part of a storage account. You can create a storage account using the Azure CLI (or PowerShell), or Azure portal. The portal is easiest because it's all guided and prompts you for each piece of information.

Every queue has a name that you assign during creation. The name must be unique within your storage account but doesn't need to be globally unique (unlike the storage account name). The combination of your storage account name and your queue name uniquely identifies a queue.

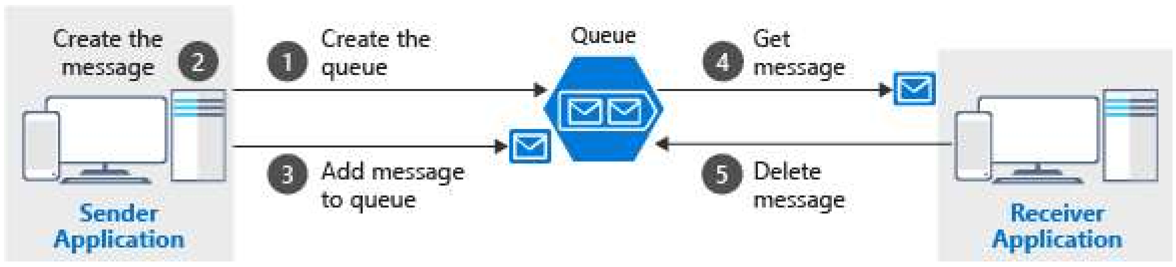
Authorization Type	Description

Azure Active Directory	You can use role-based authentication and identify specific clients based on AAD credentials.
Shared Key	Sometimes referred to as an account key, this is an encrypted key signature associated with the storage account. Every storage account has two of these keys that can be passed with each request to authenticate access. Using this approach is like using a root password - it provides *full access* to the storage account.
Shared access signature	A shared access signature (SAS) is a generated URI that grants limited access to objects in your storage account to clients. You can restrict access to specific resources, permissions, and scope to a data range to automatically turn off access after a period of time.

What is a message?

A message in a queue is a byte array of up to 64 KB. Message contents are not interpreted at all by any Azure component.If you want to create a structured message, you could format the message content using XML or JSON. Your code is responsible for generating and interpreting your custom format.

Queues hold messages - packets of data whose shape is known to the sender application and receiver application. The sender creates the queue and adds a message. The receiver retrieves a message, processes it, and then deletes the message from the queue. The following illustration shows a typical flow of this process.



Notice that get and delete are separate operations. This arrangement handles potential failures in the receiver and implements a concept called at-least-once delivery. After the receiver gets a message, that message remains in the queue but is invisible for 30 seconds. If the receiver crashes or experiences a power failure during processing, then it will never delete the message from the queue. After 30 seconds, the message will reappear in the queue and another instance of the receiver can process it to completion.

While the total queue size can be up to 500 TB, the individual messages in it can only be up to 64 KB in size (48 KB when using Base64 encoding). If you need a larger payload you can combine queues and blobs – passing the URL to the actual data (stored as a Blob) in the message. This approach would allow you to enqueue up to 200 GB for a single item.

Summary

1. Suppose you work for a government agency that plans the long-term expansion of the highway system. You receive traffic data from thousands of sensors and analyze it to make your recommendations. The amount of incoming data varies throughout the day; for example, it spikes during the morning and evening commuting hours. True or false: a server-side architecture consisting of an Azure Queue connected to a single virtual machine is a reasonable choice for this workload?
True
The queue will handle spikes in traffic and ensure no data is lost. If the VM cannot keep up with the flow of incoming messages, it will process the message backlog during low-traffic times.
2. What information uniquely identifies a queue?
Storage account name and queue name
Storage account names must be globally unique. Queue names must be unique within their containing storage account. This means the combination of storage account name and queue name uniquely identifies a queue.
3. True or false: when a client programmatically retrieves a message from a queue, the message is automatically deleted from the queue?
False
By design, messages are not automatically deleted from a queue after they are retrieved for processing. This helps ensure that every message is processed to completion. If a consumer application crashes during processing, the message is still available to be processed by a different instance of the consumer app.

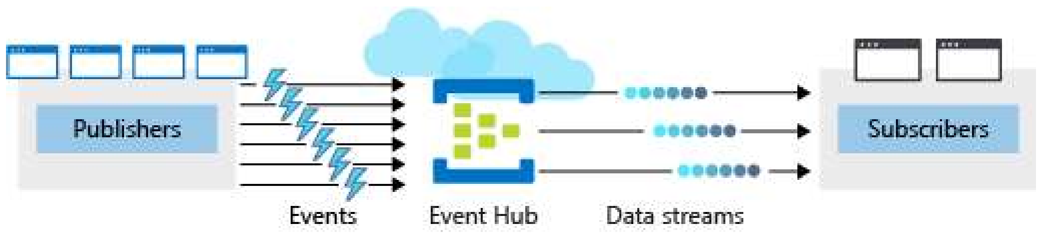
Enable reliable messaging for Big Data applications using Azure Event Hubs

Connect sending and receiving applications with Event Hubs so you can handle extremely high loads without losing data.
Azure Event Hubs can receive and process a large number of transactions. It can also be configured to scale dynamically, when required, to handle increased throughput. In this module, you'll learn how to connect Event Hubs to your application and reliably process large transaction volumes.

What is an Azure Event Hub?

Azure Event Hubs is a cloud-based, event-processing service can receive and process millions of events per second. Event Hubs acts as a front door for an event pipeline, to receive incoming data and stores this data until processing resources are available.

An entity that sends data to the Event Hubs is called a publisher, and an entity that reads data from the Event Hubs is called a consumer or a subscriber. Azure Event Hubs sits between these two entities to divide the production (from the publisher) and consumption (to a subscriber) of an event stream. This decoupling helps to manage scenarios where the rate of event production is much higher than the consumption. The following illustration shows the role of an Event Hub.



Events

An event is a small packet of information (a datagram) that contains a notification. Events can be published individually, or in batches, but a single publication (individual or batch) can't exceed 1 MB.

Publishers and subscribers

Event publishers are any application or device that can send out events using either HTTPS or Advanced Message Queuing Protocol (AMQP) 1.0.

For publishers that send data frequently, AMQP has better performance. However, it has a higher initial session overhead, because a persistent bidirectional socket and transport-level security (TLS) or SSL/TLS has to be set up first.

For more intermittent publishing, HTTPS is the better option. Though HTTPS requires additional overhead for each request, there isn't the session initialization overhead.

Consumer groups

An Event Hub consumer group represents a specific view of an Event Hub data stream. By using separate consumer groups, multiple subscriber applications can process an event stream independently, and without affecting other applications. However, the use of many consumer groups isn't a requirement, and for many applications, the single default consumer group is sufficient.

The following parameters are required to create an Event Hub:

- **Event Hub name -** Event Hub name that is unique within your subscription and:
  - Is between 1 and 50 characters long
  - Contains only letters, numbers, periods, hyphens, and underscores
  - Starts and ends with a letter or number
- **Partition Count -** The number of partitions required in an Event Hub (between 2 and 32). The partition count should be directly related to the expected number of concurrent consumers and can't be changed after the hub has been created. The partition separates the message stream so that consumer or receiver applications only need to read a specific subset of the data stream. If not defined, this value defaults to 4.
- **Message Retention -** The number of days (between 1 and 7) that messages will remain available if the data stream needs to be replayed for any reason. If not defined, this value defaults to 7.

You can also optionally configure an Event Hub to stream data to an Azure Blob storage or Azure Data Lake Store account.

To configure an application to send messages to an Event Hub, you must provide the following information, so that the application can create connection credentials:

- Event Hub namespace name
- Event Hub name
- Shared access policy name
- Primary shared access key

To configure an application to receive messages from an Event Hub, provide the following information, so that the application can create connection credentials:

- Event Hub namespace name
- Event Hub name
- Shared access policy name
- Primary shared access key
- Storage account name
- Storage account connection string
- Storage account container name

If you have a receiver application that stores messages in Azure Blob Storage, you'll also need to configure a storage account.

Azure Event Hubs provides big data applications the capability to process large volume of data. It can also scale out during exceptionally high-demand periods as and when required. Azure Event Hubs decouples the sending and receiving messages to manage the data processing. This helps eliminate the risk of overwhelming consumer application and data loss because of any unplanned interruptions.

Example repo: https://github.com/Azure/azure-event-hubs

Summary

1. Applications that publish messages to Azure Event Hub very frequently will get the best performance using Advanced Message Queuing Protocol (AMQP) because it establishes a persistent socket.

True

Publishers can use either HTTPS or AMQP. AMQP opens a socket and can send multiple messages over that socket.

2. By default, how many partitions will a new Event Hub have?

4

Event Hubs default to 4 partitions. Partitions are the buckets within an Event Hub. Each publication will go into only one partition. Each consumer group may read from one or more than one partition.

3. What is the maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub?

1 MB

The maximum size for a single publication (individual or batch) that is allowed by Azure Event Hub is 1 MB.