Azure Event Hubs: A real-time data streaming platform with native Apache Kafka support

Azure Event Hubs is a native data-streaming service in the cloud that can stream millions of events per second, with low latency, from any source to any destination.

Event Hubs is compatible with Apache Kafka. It enables you to run existing Kafka workloads without any code changes.

Event

An event is a lightweight notification of a condition or a state change. The publisher of the event has no expectation about how the event is handled. The consumer of the event decides what to do with the notification. Events can be discrete units or part of a series.

Discrete events report change in a state and are actionable. To take the next step, the consumer only needs to know that something happened. The event data has information about what happened but doesn't have the data that triggered the event.

For example, an event notifies consumers that a file was created. It might have general information about the file, but it doesn't have the file itself. Discrete events are ideal for serverless solutions that need to scale.

A series of events reports a condition and are analyzable. The events are time-ordered and interrelated. The consumer needs the sequenced series of events to analyze what happened.

Message

A message is raw data produced by a service to be consumed or stored elsewhere. The message contains the data that triggered the message pipeline.

The publisher of the message has an expectation about how the consumer handles the message. A contract exists between the two sides. For example, the publisher sends a message with the raw data, and expects the consumer to create a file from that data and send a response when the work is done.

Azure Event Grid

Azure Event Grid is a highly scalable, fully managed Pub Sub message distribution service that offers flexible message consumption patterns using the Message Queueing Telemetry Transport (MQTT) and HTTP protocols.

With Azure Event Grid, you can build data pipelines with device data, integrate applications, and build event-driven serverless architectures.

The service provides an eventing backbone that enables event-driven and reactive programming. It uses the publish-subscribe model. Publishers emit events, but have no expectation about how the events are handled. Subscribers decide on which events they want to handle.

Event Grid is deeply integrated with other Azure services and can be integrated with third-party services. It simplifies event consumption and lowers costs by eliminating the need for constant polling. Event Grid efficiently and reliably routes events from Azure and non-Azure resources. It distributes the events to registered subscriber endpoints. The event message has the information you need to react to changes in services and applications.

It has the following characteristics:

- Dynamically scalable
- Low cost
- Serverless
- At least once delivery of an event

Event Grid is offered in two editions: **Azure Event Grid**, a fully managed PaaS service on Azure, and **Event Grid on Kubernetes with Azure Arc**, which lets you use Event Grid on your Kubernetes cluster wherever that is deployed, on-premises or on the cloud.

Azure Event Hubs

Azure Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second. It facilitates the capture, retention, and replay of telemetry and event stream data.

The data can come from many concurrent sources. Event Hubs allows telemetry and event data to be made available to various stream-processing infrastructures and analytics services. It's available either as data streams or bundled event batches.

This service provides a single solution that enables rapid data retrieval for real-time processing, and repeated replay of stored raw data. It can capture the streaming data into a file for processing and analysis. It has the following characteristics:

- Low latency
- Can receive and process millions of events per second
- At least once delivery of an event

Azure Service Bus

Service Bus is a fully managed enterprise message broker with message queues and publish-subscribe topics. The service is intended for enterprise applications that require transactions, ordering, duplicate detection, and instantaneous consistency.

Service Bus enables cloud-native applications to provide reliable state transition management for business processes. When handling high-value messages that can't be lost or duplicated, use Azure Service Bus. This service also facilitates highly secure communication across hybrid cloud solutions and can connect existing on-premises systems to cloud solutions.

Service Bus is a brokered messaging system. It stores messages in a "broker" (for example, a queue) until the consuming party is ready to receive the messages. It has the following characteristics:

Reliable asynchronous message delivery (enterprise messaging as a service) that requires polling. If you're using Service Bus and you need to receive messages without having to poll the queue, you can achieve it by using a **long polling** receive operation using the TCP-based protocols that Service Bus supports.

- Advanced messaging features like first-in and first-out (FIFO), batching/sessions, transactions, dead-lettering, temporal control, routing and filtering, and duplicate detection
- At least once delivery of a message
- Optional ordered delivery of messages

Exchange events between consumers and producers that use different protocols: AMQP, Kafka, and HTTPS

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?

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.