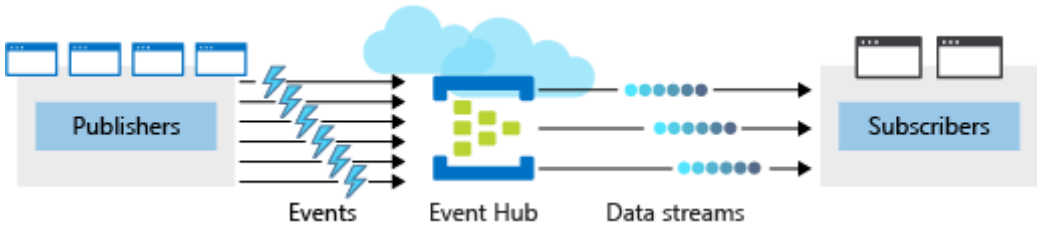
Agenda: Event Hub

- Overview
- Key Architecture Components
- Sample Event Producer and Consumer in C#

Overview

https://azure.microsoft.com/en-in/services/event-hubs/

- Azure Event Hubs is a fully managed cloud-based (PaaS) big data streaming platform, event ingestion service (is a feature within Service Bus) and can receive and process **millions of events** per second in **near real-time**.
- 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.



- 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 a complete **big data pipeline**.
- Although telemetry data flowing in from **IoT-enabled devices and machines** is a major data source for Event Hubs, the highest volume of data comes from **application logging** (e.g. status updates) and **application performance telemetry**.

A stream, not a queue

Event Hubs works with a ***stream*** of data, rather than a queue. The service also works within **a limited length** of recording volume whose size you can specify based on your needs. Accordingly, Event Hubs is constantly adding new data and removing old data from the stream under a FIFO approach.

Events

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

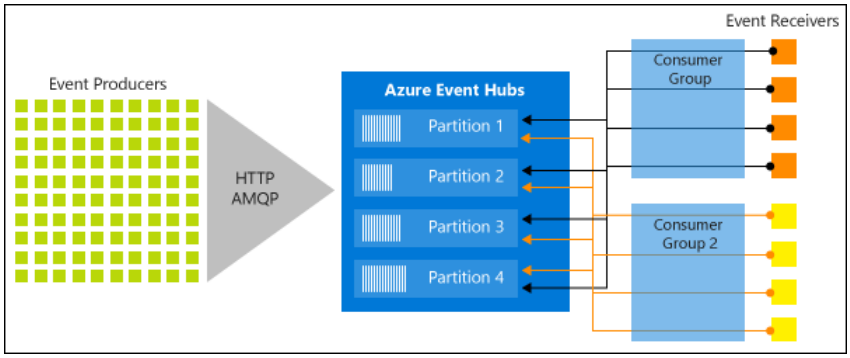
Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Why choose Event Hubs

- **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.
- **Simple:** Build real-time data pipelines with just a couple of clicks.
- **Secure:** Protect your real-time data. Event Hubs is certified by CSA STAR, ISO, SOC, GxP, HIPAA, HITRUST and PCI.
- **Scalable:** You can adjust throughput dynamically based on your usage needs and pay only for what you use. Internally Event Hubs implements a **partitioning pattern** to allow it to scale to deal with huge bursts of events and to retain events for a longer period of time.
- **Open Protocol**: Event Hubs provides simple interfaces such as AMQP, HTTP and Apache Kafka to make it easy for apps to send messages to an Event Hub.

Key architecture components

The following figure shows the Event Hubs stream processing architecture:



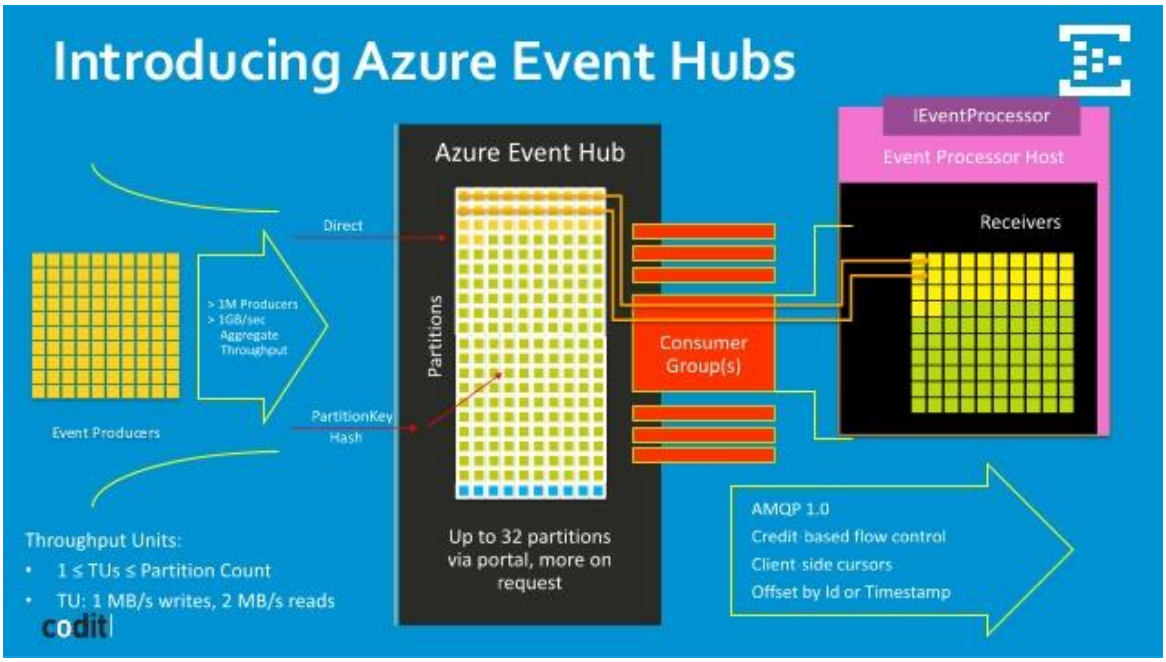
- **Event producers**: Any entity that sends data to an event hub. Event publishers can publish events using HTTPS or AMQP 1.0 or Apache Kafka.
- **Partitions**: Each event hub is divided into entirely separate channels, known as *partitions. Event producers* do not need to categorize the event data they send to an event hub. They need only send it. By default, every event hub will distribute incoming data evenly between its partitions for "**the best availability and performance**." Each consumer only reads a specific subset, or partition, of the data stream. Partitions are buffers into which the data is saved. Because of these buffers, an event isn't missed just because a subscriber is busy or even offline.
- **Consumer groups**: An Event Hub **consumer group** represents a specific **view** (state, position, or offset) 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. If you only need one receiver to read the stream then you can use the **default** consumer group, but if you need multiple receivers to read the stream concurrently but at **their own rate** then each receiver would use its own consumer group. A receiver will also manage an index (or offset) which is its own pointer to where in the stream of messages it is reading.

- **Throughput units**: Pre-purchased units of capacity that control the throughput capacity of Event Hubs. A single throughput unit allows 1 MB per second or 1000 events per second of ingress and twice that amount of egress. Standard Event Hubs can be configured with 1-40 throughput units, and you can purchase more with a quota increase support request. Usage beyond your purchased throughput units is throttled.
- **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.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Walkthrough in C#



Create Event Hub 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.

1. Azure Portal → All Services → Event Hubs → + Add
2. Provide the details as required → sku = standard, . . . → Create

Create an Event Hub

1. Select Event Hubs Namespace → **Event Hubs** → click **+ Event Hub**.
2. Provide name = myeventhub . . .
3. **Partition Count:** Partitions are data organization mechanism that relates to the downstream parallelism required in consuming applications. The number of partitions in an event hub directly relates to the number of concurrent readers you expect to have. (1 – 32)
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 defaults to *7.*

Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

5. **Capture:** 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, with the added flexibility of specifying a time or size interval.
    - Capture 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.
    - 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.

Programming Event Producer

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

3. Create a .NET Core Console Application (To send events)
4. Add NuGet Package: **Azure.Messaging.EventHubs**
5. Write Code to send messages to event hub.

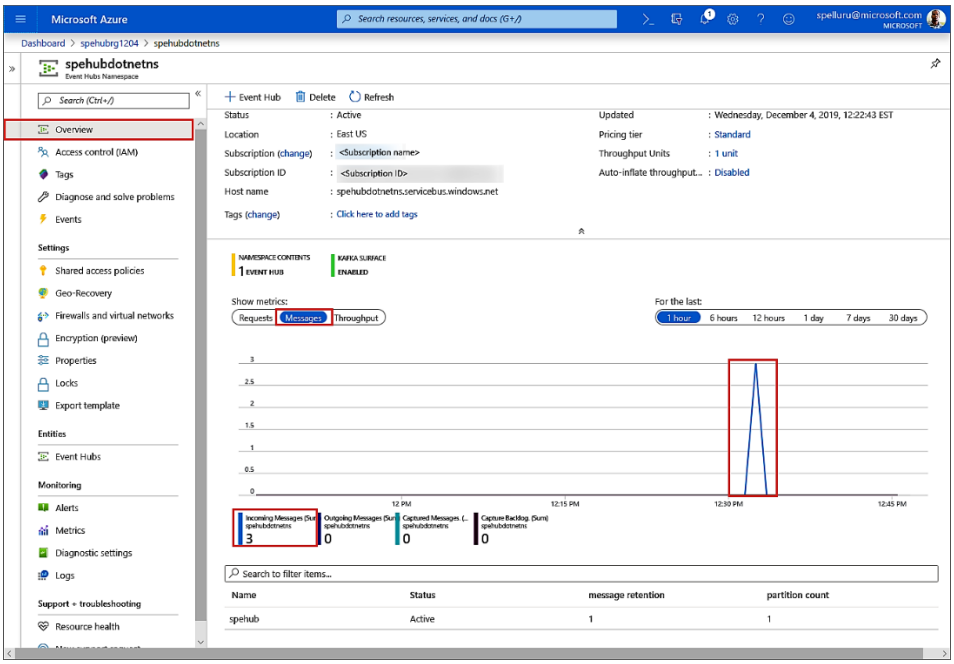
```
using System;
using System.Text;
using System.Threading.Tasks;
using Azure.Messaging.EventHubs;
using Azure.Messaging.EventHubs.Producer;

class Program
{
   private const string connectionString = "<EVENT HUBS NAMESPACE - CONNECTION STRING>";
   private const string eventHubName = "<EVENT HUB NAME>";
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
  static async Task Main()
  {
    // Create a producer client that you can use to send events to an event hub
    await using (var producerClient = new EventHubProducerClient(connectionString, eventHubName))
    {
      while (true)
      { // Create a batch of events
        using EventDataBatch eventBatch = await producerClient.CreateBatchAsync();

        // Add events to the batch. An event is a represented by a collection of bytes and metadata.
        string time = DateTime.Now.ToLongTimeString();
        eventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes("First event at " + time)));
        eventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes("Second event " + time)));
        eventBatch.TryAdd(new EventData(Encoding.UTF8.GetBytes("Third event " + time)));

        // Use the producer client to send the batch of events to the event hub
        await producerClient.SendAsync(eventBatch);
        Console.WriteLine("A batch of 3 events has been published at " + time + "...Press enter for next batch");
        Console.ReadLine();
      }
    }
  }

}
```

6. Run the application and post the messages.
7. 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 may take a few seconds for it to show that the messages have been received.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Programming Events Receiver

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.

For checkpointing, the Event Processor Host requires a storage account.

8. Create a **Storage Account** for Event Processor Host.
9. Create a container by **name="evthubcontainer"**

Create a .NET Core Console Application

10. Add NuGet Package: **Azure.Storage.Blobs, Azure.Messaging.EventHubs and Azure.Messaging.EventHubs.Processor**
11. Write Code to send messages to event hub.

```
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;
class Program
{
   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>";

   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
      BlobContainerClient storageClient = new BlobContainerClient(blobStorageConnectionString, blobContainerNa
me);
       // Create an event processor client to process events in the event hub
      EventProcessorClient processor = new EventProcessorClient(storageClient, consumerGroup, ehubNamespaceC
onnectionString, eventHubName);
       // Register handlers for processing events and handling errors
      processor.ProcessEventAsync += ProcessEventHandler;
      processor.ProcessErrorAsync += ProcessErrorHandler;

      // Start the processing
      await processor.StartProcessingAsync();
       // Wait for 10 seconds for the events to be processed
      //await Task.Delay(TimeSpan.FromSeconds(10));
      Console.WriteLine("Press Enter to stop...");
      Console.ReadLine();
       // Stop the processing
      await processor.StopProcessingAsync();
   }
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    static async Task ProcessEventHandler(ProcessEventArgs eventArgs)
    {
        // Write the body of the event to the console window
        Console.WriteLine("\tRecevied 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;
    }
}
```

Run the receiver application.

12. You should see a message that the event has been received.

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

Choosing between Azure Messaging Services

Event vs 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 sender sends a message with the raw data, and expects the reciever to create a file from that data and **send a response** when the work is done.

  Action in future - **Eg: CreateOrder, UpdateEmployee, Approve Leave Request**

  Sender – Receiver (One to One)

- 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.

  Action in Past - **Eg: OrderCreated, EmployeeUpdated, LeaveRejected**

  Publisher & Subscriber (One to Many)

Service	Purpose	Type	When to use
Queue Storage	Simple Queue. Pull based.	Message	Communication within an app. Simple and easy to use. Supports large volume of storage (500TB)
Service Bus	High-value enterprise messaging providing Instantaneous consistency. Partial Pull Based.	Message	Order processing and financial transactions. Rich features like FIFO, batching/sessions, transactions, dead-lettering, deduplication and scheduling. Supports Geo-replication and availability. 80 GB (Premium) 5GB (Standard)
Event Grid	Reactive programming. Events and PubSub Push based.	Event Distribution (Discrete) (Individual Messages)	React to status changes of services (Near Realtime). Deeply integrated with Azure Services along with third party services. Supports dead-lettering and Guaranteed Delivery. Fan out. (Single Source and Multiple Recievers) Low cost - Consumption Pricing.
Event Hub	Big data streaming (Millions of events per second) Capture, Rention and replay. Fast Pull	Event streaming (series) (think in MBs)	Starting point in an event processing pipeline. Telemetry and distributed data streaming. Supports repeated replay/strict ordering of stored raw data. Can capture streaming data into a file for processing and analysis. Fan In – Fan Out (Multiple Source and Multiple Receivers) Low cost. Consumption Pricing.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- Event Grid uses a publish-subscribe model. Publishers emit events, but have no expectation (unlike Service Bus) about which events are handled. Subscribers decide which events they want to handle.
- Event Grid isn't a data pipeline, and **doesn't deliver the actual object** that was updated.

Event Grid vs Event Hub

- Azure Event Hubs is a more suitable solution when we need a service that can receive and process millions of events per second and provide low-latency event processing. It can handle data from concurrent sources and route it to a variety of stream-processing infrastructure and analytics services. Azure Event Hubs are used more for telemetry scenarios.
- On the other hand, Azure Event Grid is ideal for reactive scenarios, like when an item has been shipped or an item has been added or updated on storage. We have to take into account also its native integrations with Functions, Logic Apps and Webhooks. Moreover, Event Grid is cheaper than Event Hubs and more suitable when we don’t have to deal with big data.

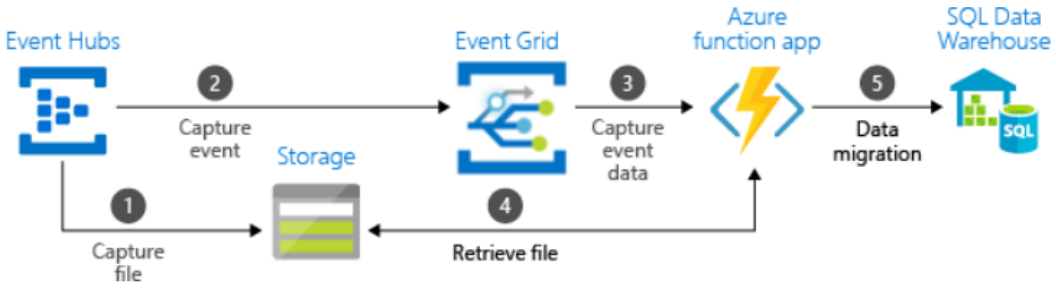
In some cases, you use the services side by side to fulfill distinct roles.

For example, an ecommerce site can use

- **Service Bus to process the order**.
- Event Hubs to capture site telemetry.
- Event Grid to respond to events like an item was shipped.

In other cases, you link them together to form an event and data pipeline. You use Event Grid to respond to events in the other Azure services. For an example of using Event Grid with Event Hubs to migrate data to a data warehouse.

The following image shows the workflow for streaming the data.



Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

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

Deccansoft Software Services H.No: 153, A/4, Balamrai, Secunderabad-500003 TELANGANA, INDIA.
http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)