Message Queuing

Message queuing is a communication system used by different processes. In its simple form it is a first-in, first-out data structure of messages (of various formats). The data is being published by publishing processes (publishers) and retrieved by consuming processes (consumers). It is designed to handle a large number of messages and can have additional features.

For the requirements of this projects the queue should have these additional features:

- Publishers should be able to share their data only with subscribers of their own choice
- Consumers should not be able to access the data they are not authorised to access
- All of the data sent over the queue should be stored in the main NoSQL database

Microsoft Event Hubs

Event Hubs reads messages from writing services/applications/… They partition event messages in partitions as opposed to a single messaging queue (number of partitions = number of concurrent readers) They also allow consumers only access to certain partitions of messaging stream. They are of high scalability and low latency

Technical overview:

1. 2-32 partitions (4 is default), number is unchangeable after creation, upper limit of 32 can be changed contacting the service bus team
2. While partitions are identifiable and can be sent to directly, it is best to avoid sending data to specific partitions. Instead, you can use higher level constructs - EVENT PUBLISHER:
    a. Any entity that sends events or data to an Event Hub is an event publisher. Event publishers can publish events using either HTTPS or AMQP 1.0. Choice of which depends on requirements. HTTPS has lower latency for sessions with less publishings, whereas AMQP is more efficient for longer sessions with frequent publishing.
    b. Event publishers use a Shared Access Signature (SAS) token to identify themselves to an Event Hub, and can have a unique identity, or use a common SAS token, depending on the requirements of the scenario. (more about SAS)
    c. Event publisher’s tasks:
        i. Acquire an SAS token:
        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, Service Bus 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.
        ii. Publish an event (using HTTPS/AMQP):
        Events can be published individually or batched. Upper limit of one

publishing is 256KB. Publisher should be unaware of partitions, instead they should use a partition key.

d. Partition Key:
   A partition key is a value that is used to map incoming event data into specific partitions for the purposes of data organization. The partition key is a sender-supplied value passed into an Event Hub. If you don't specify a partition key when publishing an event, a round robin assignment is used. Hashing function ensures that events sharing the same partition key are delivered in order to the same partition.
e. Event Consumers - any entity that reads from an event hub. There should only one active reader per partition at all time.
   i. 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 is performed by another, separate consumer group. You can only access partitions through a consumer group. There is always a default consumer group in an Event Hub, and you can create up to 20 consumer groups for a Standard tier Event Hub.
   ii. 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 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.
   iii. Checkpointing:
      The system which the consumer within the consumer group marks a position in the partition sequence. Each reader must keep track of its checkpoint. When reconnecting, the consumer starts reading from the position previously posted by the last reader connected. Used to mark events ‘complete’. It is possible to reach events older than the offset specified by checkpoint.
   iv. Consumers’ responsibilities:
      1. Connect to a partition within the consumer group.
         IMPORTANT: it must be the only connected reader within that group.
      2. Read events. It is on consumer to manage offsets.

3. Capacity and scalability:

    a. Pre-purchased throughout points are used to scale-up the Event hubs capacity.
    b. Per Throughout:
    Ingress: Up to 1 MB per second or 1000 events per second.
    Egress: Up to 2 MB per second.
    Exceeding ingress results in quota exceeded exception. Egress does not throw exceptions, but is likely to result in publishers' exceptions.
4. Security:
    a. Event Hubs enables granular control over event publishers through publisher policies. Publisher policies are a set of run-time features designed to facilitate large numbers of independent event publishers.
    b. Publisher names do not need to be created 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.

Azure Queue

Unlike Event Hubs, Azure Queue is simply a queue of messages, that mostly works on a first-in first-out basis (but it does not guarantee it). It does not provide all of the features that Event Hubs do, but it is a straightforward and easy to use.

Technical Overview:

1. Storage Account:
    a. A storage account is needed to create queues
    b. Storage account supports multiple queues (alongside with tables, blobs and files)
2. Publishing/Consuming:
    a. HTTPS/HTTPS protocols are used
    b. The publishing/consuming calls must be authenticated by the key provided with the storage account
    c. Default RESTful API is provided for publishing/retrieving messages
3. Capacity:
    a. Mazimum size of the queue 200TB
    b. Maximum size of messages is 64 KB
    c. Maximum time TTL 7 Days

Technology Used and Possible Alternative

The decision was made to proceed using Azure Queue. While the Event Hubs provide all of the features specified for the needs of the project, their capability in terms of ingress/egress is far beyond the scope of the project, which does not justify their costliness. Furthermore, all of the additional features like partitioning and consumer groups additionally increase the cost.

On the other hand, the Azure Queues could with proper implementation simulate those functions of Event Hubs at a lower cost. Its ingress/egress capabilities are sufficient to handle even the amount of messages expected.

A possible alternative to the Azure Queue implementation would be an open source tool rabbitMQ. While its capabilities are similar to the capabilities of Azure Queue. However, we have decided to use Azure Queues as they are easy to manage within the Azure portal.

Additional Links for More Information

1. Event Hubs:
    a. Azure Event Hubs page: https://azure.microsoft.com/en-gb/documentation/learning-paths/event-hubs/
    b. Shared Access Signature Authentication with Service Bus: https://azure.microsoft.com/en-gb/documentation/articles/service-bus-shared-access-signature-authentication/
    c. Pricing page: https://azure.microsoft.com/en-gb/pricing/details/event-hubs/
2. Azure Queue:
    a. Specification: https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted
    b. Implementation in .NET and other languages: https://docs.microsoft.com/en-us/azure/storage/storage-dotnet-how-to-use-queues#create-an-azure-storage-account
3. RabbitMQ Website: https://www.rabbitmq.com/