An application might need to transfer configuration data or files to other applications or services. When there are multiple applications and services that run together depending on each other, the proper management of this data transfer can be an issue of serious concern.

This data transfer between Apps and services and users are very vital and needs to be maintained properly. Azure Service Bus is a messenger which helps in this data transfer.

The contents about Azure Service Bus include:

* [What is Azure Service Bus?](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no1)
  + [Azure Service Bus Overview](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no2)
* [Azure Service Bus: Concepts](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no3)
  + [What is an Azure Service Bus Queue?](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no4)
  + [Azure Service Bus Topics and Subscriptions](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no5)
  + [Namespaces](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no6)
* [Azure Service Bus: Standard vs Premium Messaging](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no7)
* [Azure Storage Queues v/s Azure Service Bus Queues](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no8)
* [Compare Azure Messaging Services](https://intellipaat.com/blog/tutorial/microsoft-azure-tutorial/azure-service-bus/#no9)
* **What is Azure Service Bus?**
* Microsoft Azure Service Bus is a message broker service that helps in the transfer of messages. It has queues, topics and subscriptions, namespaces, etc. that helps in the efficient delivery of messages.  It is fully managed which means that the users don’t need to get in there and configure it.
* Topics and Message queues allows for the processing of message transfer between apps and services with versatility which allows the apps to function properly and tackle challenges in data transfer.
* The Service Bus also allows applications and services to work independently providing them the benefits of having to use less resources, and lesser points of failure.

### Azure Service Bus Overview

The data transferred between various applications and services can be any type of information, including data encoded in standard formats like structured Plain Text, JSON, XML, etc. Azure Service Bus serves as a messenger for these data transfers.

The common messaging scenarios in which Azure Service Bus works are:

**Messaging**: Exchanging data related to business, e-commerce, application data, etc. An e-commerce app would need to constantly communicate with the logistics service, the users, the product catalogs, etc to maintain proper function. The communication formats in these cases are different from one another and Service Bus provides different ways to process them.

**Decouple applications**: Improve application and service scalability and reliability. The load is distributed evenly so that the producer and consumer need not be online at the same time and traffic surges do not overwhelm a service.

Applications working on a global scale would be dealing with humongous amounts of data transfer which is vital for one another to perform. If the applications are coupled together, a fault from one side can stop the entire system from working. Service Bus allows the apps to work independently by taking care of the communication between them which increases their efficiency.

**Load Balancing**: Allow many customers to read at the same time. The request coming in from all over the world would create the need for the apps and services to transfer and process large amounts of data. If an app is doing this on its own, it can put a heavy load on its computing resources. Service Bus allows messages to be processed in an orderly fashion depending on the requirements which can balance the load on the processing.

**Transactions**: This allows you to perform many operations within the same atomic transaction. Multiple actions might need to be taken from a single message. Service Bus allows the messages to be processed any number of times to take necessary actions from a single transaction.

**Message sessions**: Certain messages need not be delivered urgently, Service Bus allows the messages to be scheduled and delivered to destinations based on filters that can be specified by the users.

Bottom of Form

## Azure Service Bus: Concepts

Message-oriented technologies such as message queuing and long-term publish or subscribe messaging are supported by Azure Service Bus.

The following are the key principles of Azure Service Bus:

The messaging entities at the basis of Service Bus’s messaging capabilities include queues, topics and subscriptions, and rules/actions.

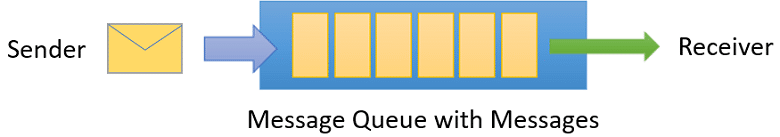
### What is an Azure Service Bus Queue?

* Queues allow one-way communication.
* Each queue acts as a broker that stores sent messages until they are received.
* A single recipient receives every single message.

Queues provide messages to one or more competing customers in a First In, First Out (FIFO) fashion. Messages are typically received by receivers and processed in the order in which they were added to the queue.

A message is received and processed by only one message consumer. Message producers (senders) and consumers (receivers) are not required to transmit and receive messages at the same time since they are stored.

This allows the messenger and consumer to be decoupled. Otherwise both of them would have to work at the same time so that each of them can send or receive messages properly. But here since they are stored in queues, this is not required.



**Load-leveling**: This allows producers and consumers to transmit and receive messages at various rates. The system load fluctuates over time in many applications. The processing time required for each unit of work, on the other hand, is usually constant. By using a queue to connect message producers and consumers, the consuming application just has to be able to handle average traffic rather than peak load.

The different receiving modes available in Queues are:

**Receive and delete**: When Service Bus receives a request from a consumer in this mode, it marks the message as consumed and sends it to the consumer application. This is the most basic model. It’s best for instances where the application cannot handle processing a broken message.

Here the messenger marks all the sent messages as consumed as soon as it leaves and can work on the remaining messages. The consumer will get the message and the messenger need not worry about what happens to the message after it was sent. This also means that the messenger application will not be slowed by what happens to the message.

The downside of this mode is that since the message is marked as consumed, if it does not reach the consumer, it can get lost.

**Peek lock**: This mode is for when the receivers need to process the message exclusively and make the broker aware of that. The message is put on a lock for one particular receiver to process while others can take the unlocked messages.

The receive operation becomes two-stage in this mode, allowing applications that can’t tolerate missing messages to continue.

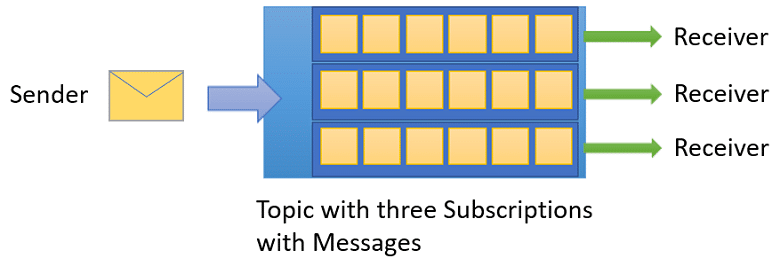
* Finds the next message to be consumed, locks it to prevent it from being received by other consumers, and then returns it to the application.
* The application then asks the Service Bus service to complete the second stage of the receive procedure after it has finished processing the message. The message is then marked as consumed by the service.

Azure Service Bus Topics and Subscriptions

* Topics help in one-way communication
* Topics also help you filter the received message to match the specific criteria.

A queue allows a single consumer to handle a message. Topics and subscriptions, in contrast to queues, allows a one-to-many mode of communication in a publish and subscribe pattern. It can be used to scale a large number of recipients. Each published message is made available to everyone who has subscribed to the topic.

A publisher publishes a message to a subject and based on the filter rules specified on these subscriptions, one or more subscribers receive a copy of the message. These filtr rules can be specified as per the requirements of the receiver.



#### Rules and Actions

Messages with specific qualities need to be processed in different ways based on context. For this, consumers can subscribe to messages that have certain features.

Subscription filters are used to accomplish this filtering. Filter actions are the term for such changes. When you create a subscription, you may include a filter expression that operates on the message’s characteristics. The properties can be both system properties (for example, Label) and user properties (for example, StoreName).

### Namespaces

All messaging components are contained in a namespace (queues and topics). A single namespace can contain several queues and topics; therefore, namespaces are frequently used as application containers.

A Service Bus namespace is made up of dozens of virtual machines that can be controlled. It can span up to three [Azure](https://intellipaat.com/blog/what-is-microsoft-azure/) availability zones if desired. As a result, you receive all of the benefits of running the message broker at a massive scale in terms of availability and resilience. You also don’t have to be concerned with underlying complexities.

## Azure Service Bus: Standard vs Premium Messaging

The main differences between the standard and premium tiers of Service Bus are:

|  |  |
| --- | --- |
| Premium | Standard |
| High throughput | Variable throughput |
| Predictable performance | Variable latency |
| Fixed   pricing | Pay as you go variable pricing |
| Ability to scale workload up and down | N/A |
| Message size up to 1 MB | Message size up to 256 KB |

**Courses you may like**

## Azure Storage Queues v/s Azure Service Bus Queues

The Azure Storage infrastructure includes storage queues. They give you the ability to store a big number of messages. HTTP or HTTPS calls allow you to access messages from anywhere in the world. A queue message can have a maximum size of 64 KB and it can hold millions of messages, up to the storage account’s overall capacity limit.

Service Bus queues are part of the Azure messaging infrastructure, which also includes queuing, publish/subscribe, and more advanced integration patterns. They’re made to connect applications or parts of applications that span different communication protocols.

Consider using Azure Storage Queues if:

* In a queue, your program must hold over 80 terabytes of data.
* Your application wishes to keep track of how far a message in the queue has progressed. It comes in handy if the service handling a message crashes. The information can then be used by another worker to pick up where the previous one left off.
* You’ll need server-side logs for every transaction made against your queues.

Consider using Azure Service Bus Queues if:

* Your approach requires a first-in-first-out (FIFO) ordered delivery from the queue.
* You wish to process messages in parallel long-running streams in your program.
* Your application can handle messages up to 64 KB in size, but it won’t likely get close to the 256-KB maximum.
* The size of your queue will not exceed 80 GB.
* Your solution would have to be able to send and receive messages in batches.

### Capacity and Quota

|  |  |  |
| --- | --- | --- |
| Comparison Criteria | Storage queues | Service Bus queues |
| Maximum queue size | 500 TB | 1 GB to 80 GB |
| Maximum message size | 64 KB | 256 KB or 1 MB |
| Maximum message TTL | Infinite | TimeSpan.Max |
| Maximum number of queues | Unlimited | 10000 |
| Maximum number of concurrent clients | Unlimited | 5000 |

## Compare Messaging Services

Azure provides three services to help with an event or message delivery across a solution. These services are:

* Azure Event Grid
* Azure Event Hubs
* Azure Service Bus

### Events v/s Messages

**Event**: A notification of a condition or state change is an event. The event’s publisher has no expectations for how the event will be handled. The event’s consumer chooses what to do with the notification.

Events can be singular or part of a sequence. Discrete events are actionable since they report state changes. To go to the next phase, the customer merely needs to be aware that anything has occurred. The event data contains information about what occurred, but not the data that caused the event to occur.

An event, for example, alerts consumers that a file has been produced. It may provide generic information about the file, but it does not contain the actual file.

**Messages**: A message is unprocessed data produced by the service and intended for consumption or storage elsewhere. The data that triggered the message pipeline is contained in the message. The communication’s publisher has a presumption about how the message will be interpreted by the recipient. Both parties have signed a contract.

For example, the publisher may send a message containing raw data, expecting the consumer to generate a file from it and respond once the operation is completed.

#### Azure Event Grid

Event Grid is an eventing service that allows for reactive programming based on events. The publish-subscribe model is used. Publishers send out events with no notion of how they will be treated. Subscribers choose the events they want to be notified about.

Event Grid is tightly connected with Azure services and can be used in conjunction with third-party applications. It reduces expenses and simplifies event consumption by reducing the need for frequent polling.

Event Grid routes events from Azure and non-Azure resources efficiently and reliably. It sends the events to the endpoints of registered subscribers. The information you need to react to changes in services and applications is contained in the event message. Event Grid isn’t a data pipeline, and it doesn’t send data.

The following are its characteristics:

* Scalable in real-time
* Cost-effective
* Serverless
* At least once, an event will be delivered.

#### Azure Event Hubs

Azure Event Hubs is a platform for large data streaming and event intake. It has the capability of receiving and processing millions of events per second. It makes capturing, storing and replaying telemetry and event stream data easier.

Data might originate from a variety of sources at the same time. Telemetry and event data can be made available to a variety of stream-processing infrastructures and analytics services using Event Hubs.

It comes in two forms: data streams and packaged event batches. This service offers a unified solution for rapid data retrieval for real-time processing as well as a recurrent replay of stored raw data. It can save streaming data to a file for analysis and processing.

The following are its characteristics:

* Latency is low.
* Thousands of events per second can be received and processed.
* At least once, an event will be delivered.

#### Comparison of Services

|  |  |  |  |
| --- | --- | --- | --- |
| Service | Purpose | Type | When to use |
| Event Grid | Reactive programming | Event distribution (discrete) | React to status changes |
| Event Hubs | Big data pipeline | Event streaming (series) | Telemetry and distributed data streaming |
| Service Bus | High-value enterprise messaging | Message | Order processing and financial transactions |