Azure Queues

Practical Tutorial On Messaging

Lab 08

By Joan Imrich, Nishava, Inc.

Deep Azure @McKesson

Goals of Lab 08

Objectives (key concepts & take away):

Understand Azure Messaging Pub/Sub models

Demos on Azure Queue, Azure Service Bus Queue & Topics

- Exposed via RESTful Web Services
- Messaging in context of various storage abstractions
  - Azure Storage Service
  - REST API and the Storage Client

How many messaging services in Microsoft Azure? (7)


Azure Storage Queue (way back in the early days)

Azure Storage Queue is the "original" Message Queue service, since the initial General Availability of Azure back in 2010


Azure Service Bus Queue (next came ESB)


Azure Service Bus Topic

Azure Service Bus Queue & Topics (the "Swiss Army Knife" service for all other generic messaging tasks) enables a single endpoint to receive multiple types of messages, and then multiple subscriptions can be setup within the service to enable a pub / sub messaging model for specific message types.


Azure IoT Hub

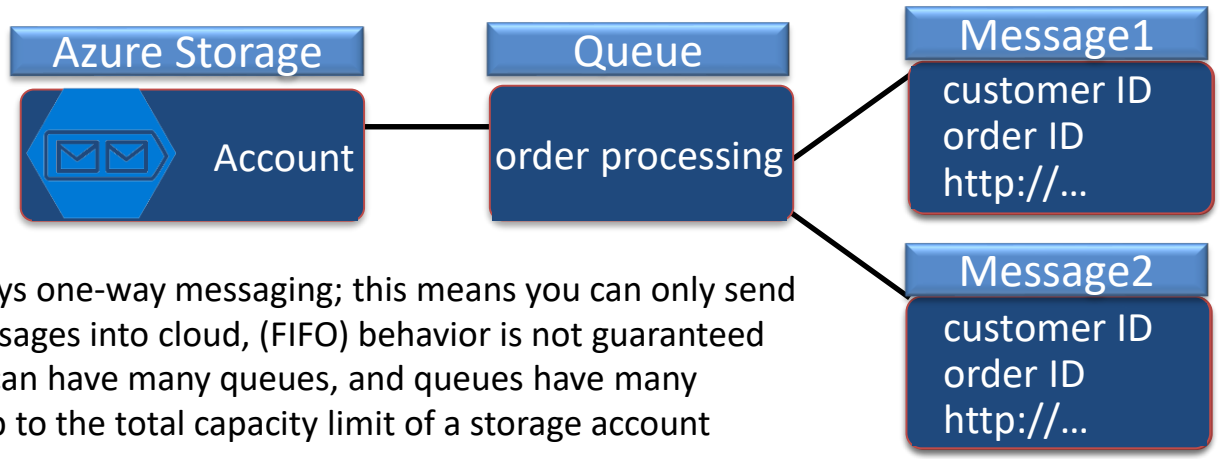

Azure Event Hubs


Azure Event Grid

The expanded messaging service fleet consists of the **Service Bus Message Broker**, the **Event Hubs Streaming** platform, and the new **Event Grid** event distribution service. Those services, which are all focused on moving datagrams, are complemented by Azure Relay that enables bi-directional, point-to-point network connection bridging.


Azure Notification Hubs

Azure Storage Queue Data Model



- Model deploys one-way messaging; this means you can only send events / messages into cloud, (FIFO) behavior is not guaranteed
- An Account can have many queues, and queues have many messages, up to the total capacity limit of a storage account

Storage Account is the highest level of the **namespace** for accessing queues and their messages. Users receive a 256-bit secret key once the account is created. This secret key is used to authenticate user requests to the storage system (HMAC SHA256 signature for the request is created using this secret key).

Messages can be In any format, of up to 64 KB, message data is returned as base64 encoded

- To store larger data, use Azure Blob store or Azure Table store, and then store the blob/entity name in the message
- message data can be binary, but when you get the messages back from the store, the response is in XML format
- There is no guaranteed return order of the messages from a queue, and a message may be returned more than once

Parameters used by Azure Queue Service are

MessageID: A GUID value that identifies the message in the queue 14

VisibilityTimeout: An integer value that specifies the message's visibility timeout in seconds (maximum value is 2 hours, default message visibility timeout is 30 seconds)

PopReceipt: A string which is returned for every message retrieved getting a message. This string, along with the MessageID, is required in order to delete a message from the Queue

MessageTTL: This specifies the time-to-live interval for the message, in seconds (maximum default time-to-live allowed is 7 days. If a message is not deleted from a queue within its TTL, then it is garbage collected and deleted

Azure Storage Queue Services

Azure Storage Queues are a great option to start with for using a cloud-based Message Queue service. It's fairly simple to integrate, and it's also fairly cost-effective, among a other key features. Azure Storage Queues are a great option to start with for using a cloud-based Message Queue service.

- Simple
- Cost-effective
- Support large volumes of messages
- one-way messaging; this means you can only send events / messages into the cloud

When messages are delivered to consumers, Azure Storage Queues operates in a "at least once" message delivery model. While this guarantees messages will be delivered, it also means that applications need to be designed to be tolerant of the possibility that messages are received multiple times. Generally, this doesn't happen, but if a receiver fails to process the message without error, then the message will remain in the queue to be delivered again.

Azure Service Bus Queue & Topics

Azure Service Bus is a set of features in Microsoft Azure that are centered around inter and intra-application messaging. This is the reason the service is called a "Bus" is because it is a generic, cloud-based messaging system for connecting just about anything - applications, services, and devices - wherever they are. A **Queue** serves two purposes. First, it is a synchronization point where workers (a process running on an instance) can coordinate job assignments. Second, queue serves as a decoupling mechanism to coordinate data flow between different stages.

Within Azure Service Bus are 2 message queue services: Queues and Topics.

- a more robust message queue service than what is provided by Azure Storage Queues
- It offers a First In, First Out (FIFO) message delivery model when sending messages to one or more competing consumers.
- same general features and scalability of Azure Service Bus Queues, but adds on a few more advanced message queue features, implementing a Publish / Subscribe messaging model.

This enables a single endpoint to receive multiple types of messages, and then multiple subscriptions can be setup within the service to "subscribe" to specific message types.

These subscriptions are implemented using filters that ensure messages are delivered only to those subscriptions that are filtering for them; whether a message goes to just a single subscription or is even copied out to multiple subscriptions simultaniously.

- Supports "at most" and "at least once" message delivery models
- one-way messaging; this means you can only send events / messages into the cloud
- Supports larger message volumes than Storage Queues
- Supports Publish / Subscribe messaging model
- Supports advanced messaging features, like:
  - Route-based messaging
  - Sessions
  - Transactions

Comparing Fundamental Queue Capabilities

	Storage Queues	Service Bus Queues
Ordering guarantee	No For more information, see the first note in the "Additional Information" section.	Yes - First-In-First-Out (FIFO) (through the use of messaging sessions)
Delivery guarantee	At-Least-Once	At-Least-Once, At-Most-Once
Atomic operation support	No	Yes
Receive behavior	Non-blocking (completes immediately if no new message is found)	Blocking with/without timeout (offers long polling, or the "Comet technique") Non-blocking (through the use of .NET managed API only)
Push-style API	No	Yes OnMessage and OnMessage sessions .NET API.
Receive mode	Peek & Lease	Peek & Lock, Receive & Delete
Exclusive access mode	Lease-based	Lock-based
Lease/Lock duration	30 seconds (default) 7 days (maximum) (You can renew or release a message lease using the UpdateMessage API.)	60 seconds (default) You can renew a message lock using the RenewLock API.
Lease/Lock precision	Message level (each message can have a different timeout value, which you can then update as needed while processing the message, by using the UpdateMessage API)	Queue level (each queue has a lock precision applied to all of its messages, but you can renew the lock using the RenewLock API.)
Batched receive	Yes (explicitly specifying message count when retrieving messages, up to a maximum of 32 messages)	Yes (implicitly enabling a pre-fetch property or explicitly through the use of transactions)
Batched send	No	Yes (through the use of transactions or client-side batching)

REF: https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted

Azure IoT Hub

Azure Event Hubs works really great for high volume throughput of event data, but it's not the greatest for Internet of Things (IoT). For this reason, Microsoft added the Azure IoT Hub service to the Azure platform, and it's built on the foundation of Azure Event Hubs with additional capabilities built specifically for the Internet of Things.

Azure IoT Hub service offers a high volume event ingress service in the cloud and access to the events sent through the service as an event stream

- Connect and manage BILLIONS of IoT devices
- Establish reliable, two-way communication between cloud and device
- Register, manage, and secure IoT devices individually with the service

- adds specific features for managing and securing large amounts of individual IoT devices as they connect to and send events through the service
- Adds cursor access for receivers to traverse up and down the stream
- adds additional IoT specific features for managing and securing large amounts of individual IoT devices
- **enables bi-directional or two-way communication between the IoT devices and the cloud.**

Other messaging services like Storage Queue, Service Bus, and Event Hubs are all just one-way messaging; this means you can only send events / messages into the cloud.Azure IoT Hub integrates command and control messaging that allows applications to send messages from the cloud directly to specific devices. This is important with IoT devices since there are instructions that often need to be sent to these devices such as configuration changes, instructions to install new firmware, and many other scenarios.

Azure Event Hubs

Azure Event Hubs is a slightly different type of message queue service when compared to Service Bus Queues, Service Bus Topics, and Azure Storage Queues. Instead of working on a 1 message in, then 1 message out at a time, Azure Event Hubs works as the "front door" to an event stream pipeline, often called an *event ingestor.* An event ingestor is a component or service that sits between event publishers and event consumers to decouple the production of an event stream from the consumption of those events. And enables it to handle a much larger scale of high throughput messages.

Azure Event Hubs can handle the ingress of millions of messages, called "Events" per second in near real-time, capabilities are built around event processing scenarios

- Stream millions of events per second
- Process events in real-time and batch on the same stream
- one-way messaging; this means you can only send events / messages into the cloud

The **messages that flow into Azure Event Hubs are referred to as "Events**". The reason for this is that it's meant to be used in a high volume, high ingress system that needs to send, receive, and process MILLIONS of events per second.

The way the events are handled and sent to receivers with Azure Event Hubs isn't one at a time like with Service Bus Queues/Topics or Storage Queues. Azure Event Hubs actually exposes the events to receivers as an **Event Stream** that allows those receivers to read, access, process those event messages using a cursor that enables access bulk or batch access up and down the event stream.

Azure Event Grid

Azure Event Grid is a messaging service built to enable event-based architectures like Microservices and Event-Driven systems to be built more easily. This is a messaging service that's built with a few target uses in mind. It is similar to Azure Service Bus Topics in that it enables a publish / subscribe messaging model. However, the similarity pretty much ends there.

Azure Event Grid is a new messaging service that is built specifically for the cloud and for event-driven architectures. It really is a "next evolution of cloud" service, similar to Azure Functions and Logic Apps with Serverless, and takes cloud-based message queues and event stream much further than the other services offer with features:

- Fully managed event routing service
- Built to support event-driven and serverless applications

Within Azure Subscription, you provision an Event Grid Topic which is the endpoint that will receive Events. **In Event Grid, they are called Events, not Messages** since it's a event-based messaging system. From there, you setup Subscriptions on the Topic that will receive Events. These Subscriptions are setup as an Endpoint using WebHooks or other Azure Services that will receive the Events sent to the Topic.

Azure Notification Hubs

Azure Notification Hubs is used to send Native Mobile Push Notifications from any service (within Azure, on-premises, or elsewhere) to native apps running on mobile devices (Windows, iOS, or Android) … rather than sending messages between applications or with a micro services architecture.

Azure Notification Hubs is a service that's designed so you have a single endpoint to send notifications to, then the Notification Hubs service automatically figures out how to send those notifications to specific mobile devices you're targeting. This service provides a nice abstraction layer, since developers do NOT need to worry about the differences of how to send push notifications to different platforms and their Push Notification Services; Microsoft (Windows), Apple (iOS), and Google (Android).

Just to clarify on what Mobile Push Notifications are… The push notifications that Azure Notification Hubs sends are NOT SMS messages or Text messages. SMS and Text messages are sent through the mobile telephone carrier. Native Push Notifications are sent through a Push Notification Service (PNS) provider for the specific mobile platform the device is running. This is the same infrastructure used to send the native, mobile push notifications you see popup with popular native apps on your smartphone or other devices, like Facebook, Twitter, Instagram, etc.

L8 DEMOS

Demo1: [10min] Azure Power Shell Messages
- Basic, Queue Storage with AzureRM Modules

Demo2: [20min] Azure Storage Queue Services
- Intermediate, VS Web App C#
- Create/Delete Queue, Add/Peek/Read/Delete Message, Get queue Length

Demo3: [30min] Azure Service Bus Queues
- Advanced, VS Web App C#
- Pub/Sub Message Queues, multi-tier app Customer Orders

Deep Azure Assignment 8

Code samples were collected from various Microsoft & GitHub repositories

https://azure.microsoft.com/en-us/resources/samples/?sort=0

References for HW8 Samples by Anudeep Sharma:

- **Getting Started with Service - Service Bus Queue Basic - in .Net**

Getting started on managing Service Bus Queues with basic features in C#
https://azure.microsoft.com/en-us/resources/samples/service-bus-dotnet-manage-queue-with-basic-features/

- **Getting Started with Service - Service Bus Queue Advance Features - in .Net**

Getting started on managing Service Bus Queues with advanced features in C# - sessions, dead-lettering, de-duplication and auto-deletion of idle entries
https://azure.microsoft.com/en-us/resources/samples/service-bus-dotnet-manage-queue-with-advanced-features/

Misc. / Healthcare Industry-specific Cortana Intelligence solutions

- **Getting Started with Service - Service Bus With Claim Based Authorization - in .Net**

Getting started on managing Service Bus with claims based authorization in C#
https://azure.microsoft.com/en-us/resources/samples/service-bus-dotnet-manage-with-claims-based-authorization/

- **Spark with Kafka (preview) on HDInsight (Event Hubs – See Arch Diagram Below)**

Learn how to use Spark Structured Streaming to read data from Apache Kafka on Azure HDInsight.
https://gallery.cortanaintelligence.com/Tutorial/Spark-with-Kafka-preview-on-HDInsight

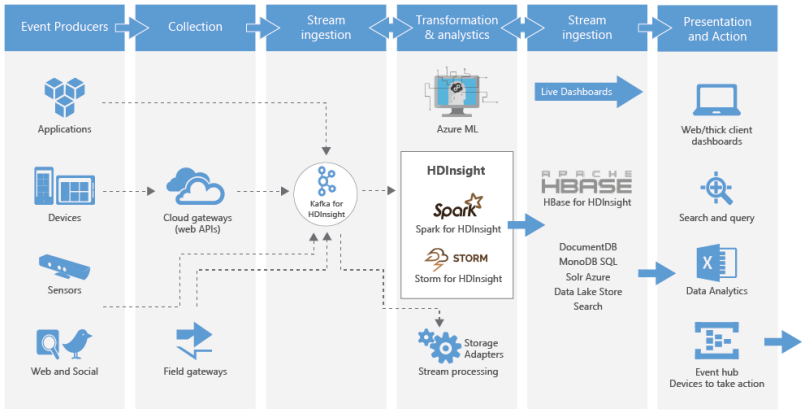
Misc.

Predicting Length of Stay in Hospitals
https://gallery.cortanaintelligence.com/Solution/Predicting-Length-of-Stay-in-Hospitals-1

Population Health Management for Healthcare
https://gallery.cortanaintelligence.com/Solution/Population-Health-Management-for-Healthcare-6

Heart Disease Prediction
https://gallery.cortanaintelligence.com/Experiment/Heart-Disease-Prediction-2

Diagnosing Schizophrenia: A Second Opinion for Doctors
https://gallery.cortanaintelligence.com/Experiment/Diagnosing-Schizophrenia-A-Second-Opinion-for-Doctors



Demo1: Azure Storage Queue Services

Perform Azure Queue storage operations with Azure PowerShell

- Azure Queue storage is a service for storing large numbers of messages that can be accessed using HTTP or HTTPS exposed via RESTful Web Services

Demo 1 covers common Queue storage operations, You learn basic program commands with PowerShell using Azure resource manger (AzureRM.*) and the .NET storage client library:

- Create a queue
- Retrieve a queue
- Add a message
- Read a message
- Delete a message
- Delete a queue

See Reference Guide **Lab08_AzureRMqueueDemo1.pdf** in Course Site Week 8
https://canvas.instructure.com/courses/1227361/pages/week-8

REF: https://docs.microsoft.com/en-us/azure/storage/queues/storage-powershell-how-to-use-queues

Demo2: VS Azure Storage Queue Services

Azure Storage:

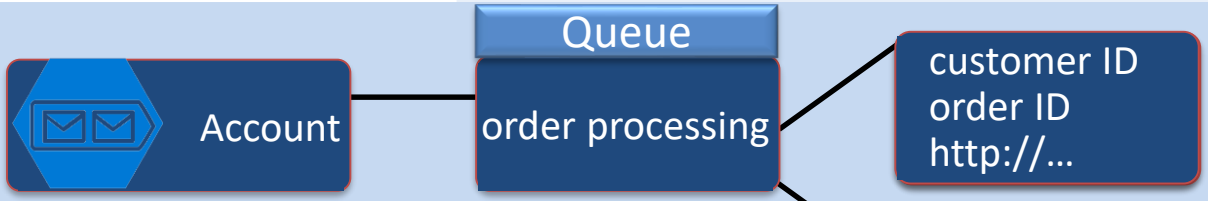
- Storage Account Service for messages that can be accessed via authenticated calls using HTTP or HTTPS

Queue:

- Contains a set of messages
- Simple asynchronous Dispatch
- Passes messages from web to worker role
- Queue name must be all lowercase

Message:

- In any format, of up to 64 KB
- TTL maximum time that a message can remain in the queue is 7 days



- one-way messaging; this means you can only send events / messages into cloud

queue can contain millions of messages, up to the total capacity limit of a storage account

Operations on Messages
Microsoft Azure Storage provides REST operations for working with messages in message queues.

Demo 2 expands on Queue Services, you learn how to create web app in VS with C# code and display status in browser :

- Create queue
- Get queue length
- Delete queue
- Get/Set Metadata
- ListQueues

- Add Message
- PutMessage– Reads message and hides for time period
- GetMessages – Reads one or more messages and hides them
- PeekMessages – Reads one or more messages w/o hiding them
- DeleteMessage – Permanently deletes messages from queue
- Clear Messages
- Update Messages

See Reference Guide **Lab08_VSqueueDemo2.pdf** in Course Site Week 8
https://canvas.instructure.com/courses/1227361/pages/week-8
REF: https://docs.microsoft.com/en-us/azure/visual-studio/vs-storage-aspnet-getting-started-queues

Demo3: VS Azure Service Bus Queues

Azure Service Bus Queues implementing a multi-tier application with Azure SDK for .NET, via Visual Studio (C# Code)

- In this tutorial you'll build and run the multi-tier application in an Azure cloud service. The front end is an ASP.NET MVC web role and the back end is a worker-role that uses a Service Bus queue. You can create the same multi-tier application with the front end as a web project, that is deployed to an Azure website instead of a cloud service. You can also try out the .NET on-premises/cloud hybrid application tutorial.

Demo 3 walks you through the steps to create an application that uses multiple Azure resources running in your local environment, you will learn the following:

- Use Visual Studio to develop for Azure
- How to create a multi-tier application in Azure using web and worker roles
- How to communicate between tiers using Service Bus queues

See Reference Guide **Lab08_VSbusDemo3.pdf** in Course Site Week 8
https://canvas.instructure.com/courses/1227361/pages/week-8

REF: https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-dotnet-multi-tier-app-using-service-bus-queues
REF: https://docs.microsoft.com/en-us/azure/storage/common/storage-use-emulator

Queue Services

- The Queue Service provides reliable, persistent messaging within and between services. The REST API for the Queue service exposes two resources: **queues and messages**.
- **Queues** support user-defined metadata in the form of name-value pairs specified as headers on a request operation. If you need to store messages larger than 64KB, you can store message data as a blob or in a table, and then store a reference to the data as a message in a queue.
- **A queue can contain an unlimited number of messages, each of which can be up to 64KB in size using version 2011-08-18 or newer.** For previous versions, the maximum size of a message is 8KB. Messages are generally added to the end of the queue and retrieved from the front of the queue, although first in, first out (FIFO) behavior is not guaranteed.
- Each storage account may have an unlimited number of message queues that are named uniquely within the account. Each message queue may contain an unlimited number of messages. **The maximum size for a message is limited to 64KB for version 2011-08-18 and 8KB for previous versions.**
- When a **message** is read from the queue, the consumer is expected to process the message and then delete it. After the message is read, it is made invisible to other consumers for a specified interval. If the message has not yet been deleted at the time the interval expires, its visibility is restored, so that another consumer may process it.
- For more information about the Queue service, see Queue Service REST API.

https://docs.microsoft.com/en-us/rest/api/storageservices/queue-service-rest-api

Service Bus Queue's Reliable Delivery

Loosely Coupled Workflow With Queues Enables Workflow Between Roles

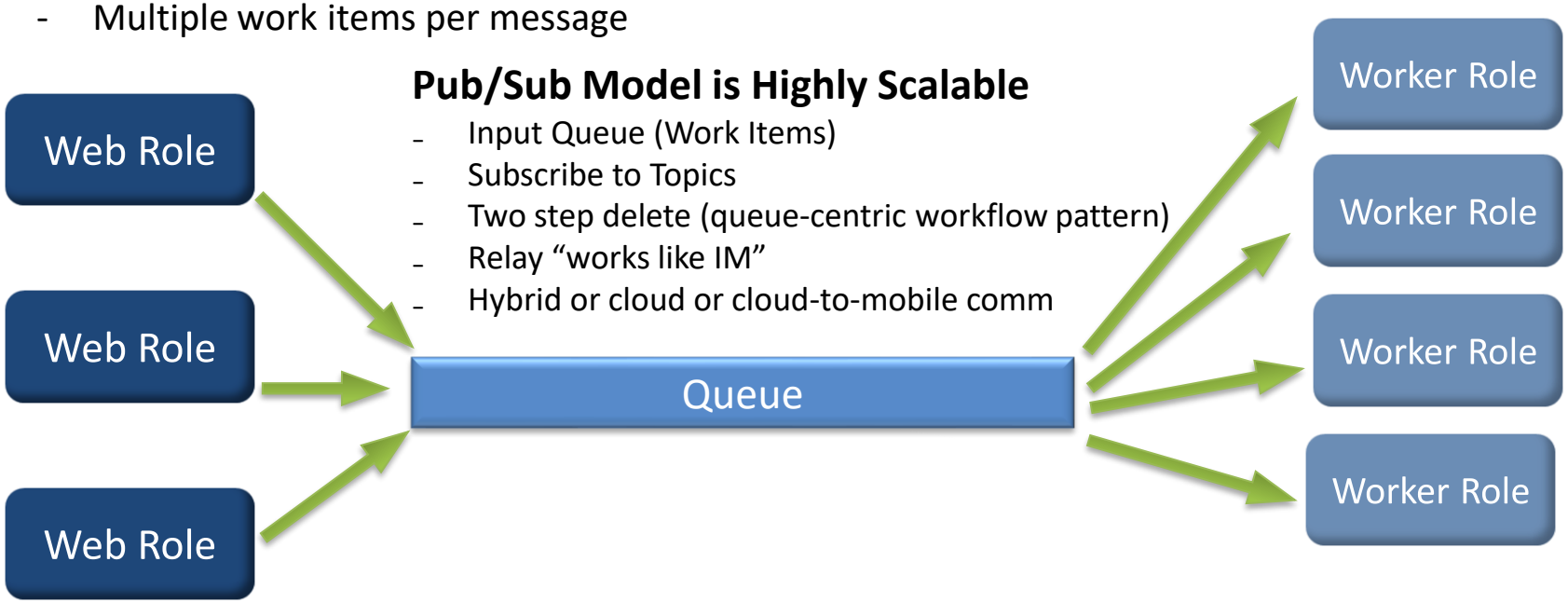
- Load work in a queue, Producer can forget about message once it is in queue
- Many workers consume the queue
- For extreme throughput
  - Use multiple queues
  - Read messages in batches
  - Multiple work items per message

Guarantee Delivery/Processing Of Messages (Two-step Consumption)

- Worker Dequeues message and it is marked as Invisible for a specified "Invisibility Time"
- Worker Deletes message when finished processing
- If Worker role crashes, message becomes visible for another Worker to process

Pub/Sub Model is Highly Scalable

- Input Queue (Work Items)
- Subscribe to Topics
- Two step delete (queue-centric workflow pattern)
- Relay "works like IM"
- Hybrid or cloud or cloud-to-mobile comm



Service Bus Messaging

Message Handling

Topics & Subscriptions

Session and Workflow Management Features - Sessions enable you to save and retrieve the application processing state (by using SetState and GetState). SessionID property on a message enables receivers to listen on a specific session ID and receive messages that share the specified session identifier (removes dup's)

Error and Transaction Handling

- Dead lettering,is only supported by Service Bus queues, can be useful for isolating messages that cannot be processed successfully by the receiving application or when messages cannot reach their destination due to an expired time-to-live (TTL) property. The TTL value specifies how long a message remains in the queue. With Service Bus, the message will be moved to a special queue called $DeadLetterQueue when the TTL period expires.
- To find "poison" messages in Storage queues, when dequeuing a message the application examines the DequeueCount property of the message. If **DequeueCount** is greater than a given threshold, the application moves the message to an application-defined "dead letter" queue.
- **Deadletter Queues** - The DeadletterQueue sample shows how to use the deadletter queue for setting aside messages that can't be processed, and how to receive from deadletter queue to inspect, repair, resubmit messages.
- **Time To Live** - The TimeToLive example shows the basic functionality of the TimeToLive option for messages as well as handling of the deadletter queue where messages can optionally be stored by the system as they expire.
- **Atomic Transactions** - Service Bus supports wrapping AtomicTransactions scopes around a range of operations, allowing for such groups of operations to either jointly succeed or fail
- **Durable Senders** - The DurableSender sample makes client applications robust against frequent network link failures.
- **Geo Replication** - The GeoReplication sample illustrates how to route messages through two distinct entities, possibly located in different namespaces in differentr datacenters, to limit the application's availability risk.

Partitioned Entities

Windows Communication Foundation (WCF) Binding

https://github.com/Azure/azure-service-bus/blob/master/samples/DotNet/Microsoft.ServiceBus.Messaging/README.md
https://github.com/Azure/azure-service-bus/blob/master/samples/Java/readme.md

APPENDIX

MISC. MESSAGING INTEROPERABILITY SLIDES

Terminology

	MS Azure	Apache Kafka	AWS
Producer	Event Publishers	Producer	Producer
Consumer	Event Consumers	Consumer	Amazon KinesisStreams Applications
Stream	Event Hub	Topic	Stream
Partition	Partition	Partition	Shard
Index	Partition Key	Offset	Sequence Number
Coordinator	Worker	Consumer Groups built into client libraries(ZooKeeper)	Amazon Kinesis ClientLibrary/MultiLangDaemon(DynamoDB)
Consumer Group	Consumer Group	Consumer Group	Application

Terminology

Queue Names

- Every queue within an account must have a unique name. The queue name must be a valid DNS name, and cannot be changed once created. Queue names must confirm to the following rules:
- A queue name must start with a letter or number, and can only contain letters, numbers, and the dash (-) character.
- The first and last letters in the queue name must be alphanumeric. The dash (-) character cannot be the first or last character. Consecutive dash characters are not permitted in the queue name.
- All letters in a queue name must be lowercase.
- A queue name must be from 3 through 63 characters long.

Azure Service Bus Messaging

- Service Bus Messaging, which includes entities such as queues and topics, combines enterprise messaging capabilities with rich publish-subscribe semantics at cloud scale. Service Bus Messaging is used as the communication backbone for many sophisticated cloud solutions.
- https://docs.microsoft.com/en-us/azure/service-bus-messaging/service-bus-azure-and-service-bus-queues-compared-contrasted

- Azure supports two types of queue mechanisms: **Storage queues** and **Service Bus queues**.
- **Storage queues**, which are part of the Azure storage infrastructure, feature a simple REST-based GET/PUT/PEEK interface, providing reliable, persistent messaging within and between services.
- **Service Bus queues** are part of a broader Azure messaging infrastructure that supports queuing as well as publish/subscribe, and more advanced integration patterns. For more information about Service Bus queues/topics/subscriptions, see the overview of Service Bus.
- While both queuing technologies exist concurrently, Storage queues were introduced first, as a dedicated queue storage mechanism built on top of Azure Storage services.
- Service Bus queues are built on top of the broader "messaging" infrastructure designed to integrate applications or application components that may span multiple communication protocols, data contracts, trust domains, and/or network environments.

Service Bus APIs

REST operation groups

The Service Bus REST API provides operations for working with the following resources.
Azure Resource Manager APIs perform operations on Service Bus entities, such as namespaces, queues, and topics. They are useful in scenarios that enable global authentication, rather than at the namespace or entity level

Operation group	Description
Namespaces	Provides operations for managing Service Bus namespaces.
Queues	Operations for managing Service Bus queues.
Subscriptions	Operations for managing Service Bus topic subscriptions.
Topics	Operations for managing Service Bus topics.

https://docs.microsoft.com/en-us/rest/api/servicebus/

https://github.com/Azure/azure-service-bus/blob/master/samples/DotNet/Microsoft.ServiceBus.Messaging/README.md