**Practicals**

1. Implement a Service Bus topic and queue
2. Write code that uses Service Bus queues
3. Write code that uses Service Bus topics

**Choose whether to use messages or events**

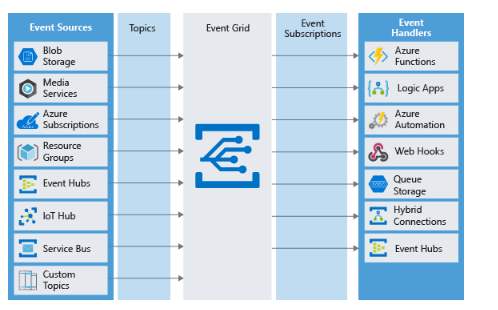
1. The first thing to understand about a communication is whether it sends **messages** or **events**
2. Questions:
   1. Does the sending component expect the processing to be processed in a particular way by the destination component?
   2. Does the communication include the data and payload?
   3. If yes to both use a Message.
   4. If no to both then use an Event
   5. Then choose the technology:
   6. Azure Queue Storage
      1. Azure Event Hubs
      2. Azure Event Grids
      3. Azure Service Bus
3. What is a Message
   1. A message contains raw data, produced by one component that will be consumed by another component.
   2. A message contains the data itself, not just a reference to that data
   3. The sending component expects the message content to be processed in a certain way by the destination component.
4. What is an Event
   1. **Events** are lighter weight than messages, and are most often used for broadcast communications
   2. The components sending the event are known as **publishers**, and receivers are known as **subscribers**
   3. The **subscription** is managed by an **intermediary**, like **Azure** **Event** **Grid** or **Azure** **Event** **Hubs**
   4. When **publishers** send an event, the **intermediary** will route that event to interested **subscribers**
   5. This pattern is known as a "**publish**-**subscribe** **architecture**"
   6. Event Characteristics:
      1. An event is a lightweight notification
      2. The event may be sent to multiple receivers, or to none
      3. Events are often intended to "fan out"
      4. The publisher of the event has no expectation about the action a receiving component takes
      5. Some events are discrete units
      6. Some events are part of a related and ordered series
   7. Message – Expects
   8. Events – Does not Expect (Discrete)
5. How to choose messages or events
   1. Before you choose, you must analyze your application's architecture and all its use cases, to identify all the different purposes where its components have to communicate with each other
   2. **Events** are more likely to be used for **broadcasts** and are often **ephemeral**, meaning a communication might **not** be handled by any **receiver**
   3. **Messages** are more likely to be used where the distributed application requires a **guarantee** that the communication will be **processed**
   4. For each communication, consider the following question: **Does the sending component expect the communication to be processed in a particular way by the destination component?**

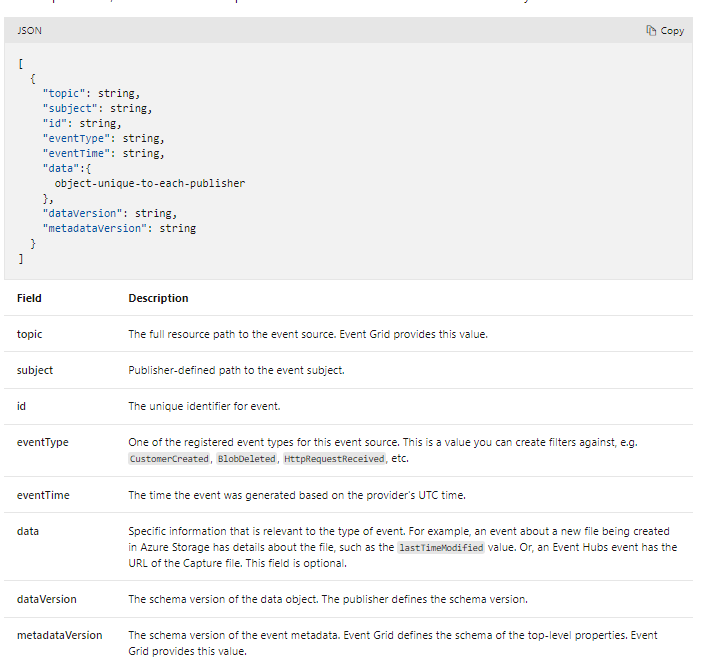
# Choose a message-based delivery with queues

1. What is Azure Queue Storage?
   1. Is a service that uses Azure Storage to store large numbers of messages that can be securely accessed from anywhere in the world
   2. Using a simple REST-based interface.
   3. Can contain millions of messages
2. What is Azure Service Bus Queues?
   1. Is a message broker system intended for enterprise applications
   2. Service Bus is built on top of a dedicated messaging infrastructure
   3. Both of these services are based on the idea of a queue, which holds sent messages until the target is ready to receive them
3. What are Azure Service Bus Topics?
   1. Azure Service Bus topics are like queues, but can have multiple subscribers
   2. When a message is sent to a topic instead of a queue, multiple components can be triggered to do their work.
   3. Internally, topics use queues
   4. When you post to a topic, the message is copied and dropped into the queue for each subscription
4. Benefits of queues
   1. Increased reliability
      1. Queues are used by distributed applications as a temporary storage location for messages
      2. The source component can add a message to the queue and destination components can retrieve the message
      3. Queues increase the reliability because messages can wait until a destination component is ready to process them
   2. Message delivery guarantees
      1. Delivery Approaches:
         1. At least once delivery – Guaranteed to arrive
         2. At most once delivery – May not arrive
         3. First in first out - Precisely the correct order
   3. Transactional support
      1. For example, we won't be in business long if the credit card message is not delivered, and all our orders are fulfilled without payment!
      2. You can avoid these kinds of problems by grouping the two messages into a transaction
      3. Message transactions succeed or fail as a single unit
5. Which service should I choose?
   1. Use **Azure** **Storage** **Queues** or **Azure** **Service** **Bus?**
   2. Both of which can be used to **store** and **deliver** **messages**
   3. Use Service Bus topics if you:
      1. Need multiple receivers to handle each message
   4. Use Service Bus Queues if you:
      1. Need At most once delivery guarantee
      2. Need FIFO
      3. Need to group messages into transactions
      4. Receive messages without polling the queue
      5. Provide role based access model into queues
      6. Handle messages larger than 64kb and smaller than 256kb
      7. Queue size will not grow larger than 80gb
      8. Want to publish and consume batches of messages
   5. Use queue storage if you:
      1. Need an audit trail of all messages
      2. Expect the queue to exceed 80gb
      3. Want to track progress of processing a message inside a queue
6. A queue is a simple, temporary storage location for messages sent between the components of a distributed application.
7. Use a queue to organize messages and gracefully handle unpredictable surges in demand
8. Use Storage queues when you want a simple and easy-to-code queue system
9. For more advanced needs, use Service Bus queues
10. If you have multiple destinations for a single message, but need queue-like behaviour, use Service Bus topics.

# Choose Azure Event Grid

1. Many applications use a publish-subscribe model to notify distributed components that something happened
2. What is Azure Event Grid?
   1. Is a fully-managed event routing service running on top of Azure Service Fabric
   2. Event Grid distributes events from different source
   3. Event Grid was created to make it easier to build event-based and serverless applications on Azure
   4. It is  dynamically scalable, low-cost, messaging system
   5. There are several concepts in Azure Event Grid that connect a source to a subscriber:
      1. Events: What happened
      2. Event Source: Where the event took place
      3. Topics: Endpoint where publishers send events
      4. Event Subscriptions: The endpoint or built-in mechanism to route events
      5. Event Handlers: The app or service reacting to the event
   6. The following illustration shows an Azure Event Grid positioned between multiple event sources and multiple event handlers



1. What is an event?
   1. Events are the data messages passing through Event Grid that describe what has taken place
   2. Example:
2. What is an event source?
   1. Event sources are responsible for sending events to Event Grid.
   2. Each event source is related to one or more event types
   3. For example, Azure Storage is the event source for blob created events. IoT Hub is the event source for device created events
   4. A publisher to Event Hub is the user or organization that decides to send events to Event Grid
   5. For example, Microsoft publishes events for several Azure services.
   6. Organizations that host services outside of Azure can publish events through Event Grid
3. What is an event topic?
   1. Event topics categorize events into groups
   2. Topics are represented by a public endpoint and are where the event source sends events to
   3. Topics are divided into **system** topics, and **custom** topics
   4. **System Topics**: Are built-in topics provided by Azure services. You don't see system topics in your Azure subscription because the publisher owns the topics. To subscribe, you provide information about the resource you want to receive events from
   5. **Custom Topics**: Custom topics are application and third-party topics.
4. **What is an Event Subscription?**
   * 1. Event Subscriptions define which events on a topic an event handler wants to receive
5. **What is an event handler?**
   1. An event handler (sometimes referred to as an event "subscriber"),  is any component (application or resource) that can **receive** **events** from **Event** **Grid**
6. **Type of Event Sources**
   1. **Azure subscriptions and Resource Groups**: Subscriptions and resource groups generate events related to management operations in Azure
   2. **Container Registry**: Generates events when images in the registry are added
   3. **Event Hub**: Events from a variety of data sources
   4. **Service Bus**: Service bus can generate events to Event Grid when there are active messages with no active listeners
   5. **Storage Accounts**: Storage accounts can generate events when users add blobs, files, table entries, or queue messages
   6. **Media Services**: Generate events when an encoding job is started or completed on a video file
   7. **Azure IoT Hub**: Gathers telemetry from IoT devices. It can generate events whenever such communications arrive
   8. **Custom Events**:  Custom events can be generated using the REST API, or with the Azure SDK on Java, GO, .NET, Node, Python, and Ruby
7. Event Handlers
   1. The following object types in Azure can receive and handle events from Event Grid:
      1. Azure Functions
      2. Webhooks
      3. Azure Logic Apps
      4. Microsoft Power Automate
8. Should you use an Event Grid?
   1. Use Event Grid when you need these features:
      1. Simplicity
      2. Advanced Filtering
      3. Fan-Out: You can subscribe to an unlimited number of endpoints to the same events and topics
      4. Reliability
      5. Pay per event

# Choose Azure Event Hubs

1. There are certain applications that produce a massive number of events from almost as many sources
2. The term BIG DATA applies
3. What is Azure Event Hubs?
   1. Is an intermediary for the publish-subscribe communication pattern
   2. Unlike [Event Grid](https://azure.microsoft.com/services/event-grid/), however, it is optimized for extremely high throughput, a large number of publishers, security, and resiliency
4. Event Hub Services:
   1. Partitions: As Event Hubs receives communications, it divides them into partitions. Partitions are buffers into which the communications are saved
   2. Capture: Event Hubs can send all your events immediately to Azure [Data Lake](https://azure.microsoft.com/services/storage/data-lake-storage/)s
   3. Authentication: All publishers are authenticated and issued a token. This means Event Hubs can accept events from external devices and mobile apps, without worrying that fraudulent data from pranksters

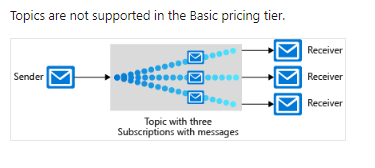
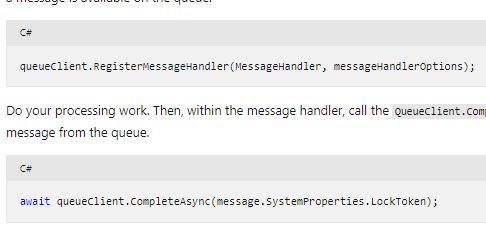
## Using Event Hubs

1. Event Hubs has support for pipelining event streams to other Azure services
2. Which Service should I choose?
   1. Just like our queue choice, selecting between these two events delivery services can seem tricky at first. Both support At Least Once semantics.
3. Choose Event Hubs if:
   1. You need to support authenticating a large number of publishers
   2. You need to save a stream of events to Data Lake or Blob storage
   3. You need aggregation or analytics on your event stream
   4. You need reliable messaging or resiliency
4. Otherwise, if you need a simple event publish-subscribe infrastructure, with trusted publishers (for instance, your own web server), you should choose Event Grid
5. Event Hubs lets you build a big data pipeline capable of processing millions of events per second with low latency
6. It can handle data from concurrent sources and route it to a variety of stream-processing infrastructures and analytics services

# Choose a messaging platform

1. Decide between messages and events
   1. Messages and events are both **datagrams**: packages of data sent from one component to another
   2. Messages
      1. A message generally contains the data itself, not just a reference
      2. The sending application needs to know exactly what data to include to avoid sending too much data
      3. For example, a web front end or mobile app would send a message to the back-end processing components. In the back end, steps like routing to the store near the customer and charging the credit card would take place
   3. Events
      1. An event triggers a notification that something has occurred
      2. Events are "lighter" than messages and are most often used for broadcast communications
      3. Our pizza chain would likely use events for notifications to users about status changes
      4. Status change events could be sent to **Azure** **Event** **Grid**, then on to **Azure** **Functions**, and to **Azure** **Notification** **Hubs** for a completely **serverless** solution
      5. **Service** **Bus** is designed to **handle** **messages**. If you want to send events, you would likely choose **Event** **Grid**

## Service Bus topics, queues, and relays

1. Azure Service Bus can exchange messages in three different ways: queues, topics, and relays.
2. What is a queue?
   1. A **queue** is a simple temporary storage location for messages.  A destination component picks up the message at the front of the queue. Under ordinary circumstances, each message is received by only one receiver.
   2. During peak times, messages may come in faster than destination components can handle them
   3. When demand drops, destination components can catch up and the queue shortens
   4. A queue responds to high demand like this without needing to add resources to the system
   5. **However**, for messages that need to be handled relatively **quickly**, adding **additional** **instances** of your destination component can allow them to **share** the **load**
3. What is a topic?
   1. A **topic** is similar to a queue but can have multiple subscriptions
   2. This means that multiple destination components can subscribe to a single topic, so each message is delivered to multiple receivers
   3. Subscriptions can also filter the messages in the topic to receive only messages that are relevant
4. What is a Relay
   1. A **relay** is an object that performs synchronous, two-way communication between applications
   2. Unlike **queues** and **topics**, it is not a **temporary** **storage** **location** for messages
   3. Instead, it provides bidirectional, unbuffered connections across network boundaries such as firewalls
5. Service Bus queues and storage queues
   1. There are two Azure features that include message queues: **Service** **Bus** and **Azure** **Storage** **accounts**
   2. Key advantages of Service Bus queues include: (Repetition)
      1. Supports larger messages sizes of 256 KB (Standard tier) or 1mb (Premium tier).
      2. Supports both at-most-once and at-least-once delivery
      3. Guarantees FIFO order
      4. Can group multiple messages into a transaction
      5. Supports role-based security
      6. Does not require destination components to continuously poll the queue
   3. Advantages of storage queues:
      1. Supports unlimited queue size (versus 80-GB limit for Service Bus queues)
      2. Maintains a log of all messages
6. How to choose a communications technology
   1. Is the **communication** an **event**? If so, consider using **Event** **Grid** or **Event** **Hubs**
   2. Should a **single** **message** be delivered to more than one destination? If so, use a **Service** **Bus** topic. Otherwise, use a **queue**.
   3. Choose Service Bus queues if:
      1. You need an at-most-once delivery guarantee
      2. FIFO
      3. Group messages into one transaction
      4. Receive messages without polling the queue
      5. Provide role-based access to the queues
      6. Handle messages larger than 64 KB but smaller than 256 KB
      7. Queue size will not grow larger than 80 GB
      8. You would like to be able to publish and consume batches of messages
   4. Choose Queue Storage if:
      1. You need a simple queue with no particular additional requirements
      2. You need a simple queue with no particular additional requirements
      3. You need a simple queue with no particular additional requirements
      4. You want to track progress
   5. **Event** **Grid** is designed for events, which notify recipients only of an event and do not contain the raw data
   6. **Azure** **Event** **Hubs** is designed for high-flow analytics types of events
   7. **Azure** **Service** **Bus** and storage queues are for messages
   8. If your requirements are simple, if you want to send each message to only one destination, or if you want to write code as quickly as possible, a **storage** **queue**
   9. Otherwise, **Service** **Bus** queues provide many more options and flexibility
7. Write code that uses Service Bus queues
   1. Distributed applications use queues, such as Service Bus queues, as temporary storage locations for messages that are awaiting delivery to a destination component
   2. Microsoft.Azure.ServiceBus NuGet package
      1. To make it easy to write code that sends and receives messages through Service Bus, Microsoft provides a library of .NET classes, **Microsoft.Azure.ServiceBus** NuGet package
      2. The most important class in this library for queues is the QueueClient class
   3. Connection strings and keys
      1. Source components and destination components both need two pieces of information to connect to a queue in a Service Bus namespace:
         1. The location of the Service Bus namespace, also known as an endpoint
         2. An access key
      2. Both of these pieces of information are provided to the QueueClient object in the form of a connection string
   4. Call methods asynchronously
      1. When sending a message to a queue, for example, use the QueueClient.SendAsync() method with the await keyword
   5. Write code that sends to queues
   6. Receive messages from the queue
      1. To receive messages, you must first register a message handler
      2. This is the method in your code that will be invoked when a message is available on the queue

# Exercise:

# Configure a connection string to a Service Bus namespace

# To access a Service Bus namespace and use a queue, you must configure two pieces of information in your console apps:

* 1. Endpoint for your namespace
  2. Shared access key for authentication

1. Both of these values can be obtained from the Azure portal in the form of a complete connection string
2. To run code after editing: dotnet run -p ./privatemessagesender

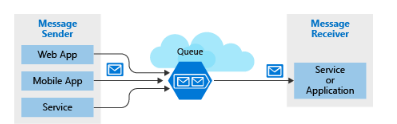
# Write code that uses Service Bus topics

1. To allow multiple components to receive the same message, we'll use an Azure Service Bus topic
2. Code with topics versus code with queues
   1. If you want every message sent to be delivered to all subscribing components, you'll use topics
   2. Writing code that uses topics is a way to replace queues
   3. However, you'll use the TopicClient class instead of the QueueClient class to send messages and the SubscriptionClient class to receive messages
3. Set filters on subscriptions
   1. If you want to control that specific messages sent to the topic are delivered to particular subscriptions, you can place filters on each subscription in the topic
   2. Filters can be:
      1. **Boolean** **Filters**: The **TrueFilter** ensures that all messages sent to the topic are delivered to the current subscription. The **FalseFilter** ensures that none of the messages are delivered to the current subscription
      2. **SQL** **Filters**: A **SQL** filter specifies a condition by using the same syntax as a **WHERE** clause in a **SQL** **query**
      3. **Correlation** **Filters**: A correlation filter holds a **set** of **conditions** that are **matched** against the properties of each message
   3. **SQL** **filters** are the **most** **flexible**, but they're also the **most** computationally **expensive** and could **slow** **down** our **Service** **Bus** **throughput**
4. TopicClient example



**Create the Azure storage infrastructure**

1. A queue can be used to eliminate the direct link between the front-end apps and your middle-tier web service
2. **What is Azure Queue storage?**
   1. Azure Queue storage is an Azure service that implements cloud-based queues
   2. Each queue maintains a list of messages
   3. Application components access a queue using a REST API or an Azure-supplied client library
   4. Sender component: Adds message to queue
   5. Receiver component: Retrieves messages



* 1. Pricing is based on queue size and number of operations. Larger queues cost more

## Why use queues?

1. A queue increases resiliency by temporarily storing waiting messages
2. Low demand = small queue, msg may be lost. Due to msg coming in fast
3. High demand = large queue, msg won’t be lost
4. A single queue can be up to 500 TB in size, so it can potentially store millions of messages
5. Queue scale automatically
6. The **Autoscale** feature is available on Azure **virtual** **machine** **scale** **sets**, **cloud** **services**, Azure **App** **Service** **plans**, and **App** **Service** **environments**.

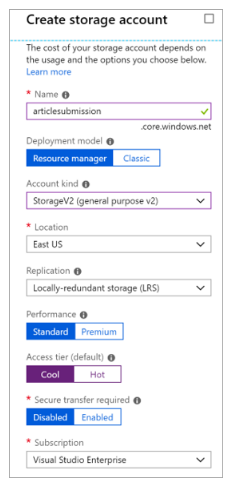
## What is a message?

1. A message in a queue is a byte array of up to 64 KB
2. You could format the message content using XML or JSON
3. Example:



## Creating a storage account

1. A queue must be part of a storage account
2. You can create a storage account using the **Azure** **CLI** (or **PowerShell**), or **Azure** **portal**



### Settings for queues

1. Settings to consider:
   1. Queues are only available as part of Azure general-purpose storage accounts (v1 or v2). Not Blob storage
   2. **Access tier:** Applies to blob storage and not queues
   3. Choose location close to source or destination
   4. Data is always replicated to multiple servers. Either **Locally** **Redundant** **Storage** – Low Cost (**LRS**) or **Geo**-**Redundant** **Storage** (**GRS**).
   5. Performance tier – How messages are stored. Standard uses magnetic drives. Premium uses Solid State Drives.
   6. Require secure transfer if sensitive information may pass through the queue

## Exercise - Create a storage account

1. Use the **az storage account create** command to create the storage account
2. Command parameters:
   1. –**name**: sets the name
   2. –**g**: supplies the resource group
   3. –**kind**: sets the storage account type
   4. –**sku**: sets the replication and storage type
   5. –**l**: sets the location
3. What was used in the example:
   1. az storage account create --name storage288cm -g learn-a5af18d4-edde-4e9a-9395-a1a269cdfc79 --kind StorageV2 --sku Standard\_LRS

## Identify a queue

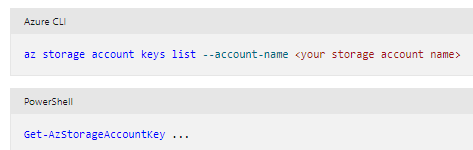
1. To access a queue you need 3 pieces of information:
   1. Storage account name
   2. Queue name
   3. Authorization token

## Queue identity

1. Every queue has a name that you assign during creation
2. The name must be **unique** within your storage account but **doesn't** need to be **globally** **unique**

## Access authorization

1. Every request to a queue must be authorized
2. Options to choose from:
   1. Azure Active Directory: role based authentication or AAD credentials
   2. Shared key: AKA account key.
   3. Shared access signature: SAS, generated URI that grants limited access
3. Retrieve the account key
   1. Your account key is available in the **Settings** > **Access** **keys**
   2. Retrieve keys using these commands:

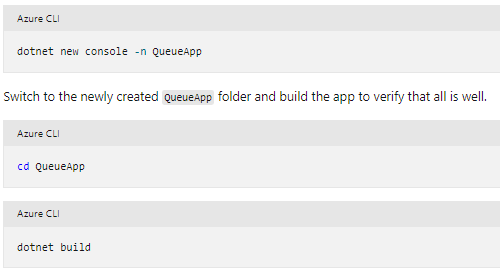


## Access queues

1. You access a queue using a REST API
2. For example: http://<storage account>.queue.core.windows.net/<queue name>
3. An Authorization header must be included with every request. The value can be any of the **three** **authorization** **styles**
4. **Use the Azure Storage Client Library for .NET**
   1. Library provided by Microsoft that formulates REST requests and parses REST responses for you
   2. Reduces the amount of code you need to write
   3. The client library uses a connection string to establish your connection
   4. Your connection string is available in the Settings section of your Storage Account in the Azure portal, or through the Azure CLI and PowerShell
   5. Example:
      1. string connectionString = "DefaultEndpointsProtocol=https;AccountName=<your storage account name>;AccountKey=<your key>;EndpointSuffix=core.windows.net"

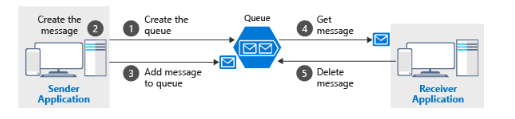
## Exercise - Identify a queue

1. Use the **dotnet** **new** command to create a new console app with the name **QueueApp**



## Programmatically access a queue

1. Queues hold messages - packets of data whose shape is known to the sender application and receiver application
2. Sender creates the queue and adds a message
3. Receiver retrieves a message, processes it, and then deletes the message from the queue



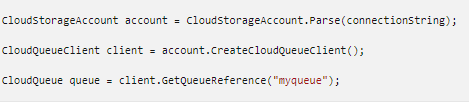
1. Notice that **get** and **delete** are separate operations
2. Implements a concept called **at**-**least**-**once** **delivery**
3. If the receiver crashes or experiences a power failure during processing, then it will never delete the message from the queue

## The Azure Storage Client Library for .NET

1. Provides the following to interact with:
   1. CloudStorageAccount
   2. CloudQueueClient
   3. CloudQueue
   4. CloudQueueMessage
2. You will use these classes to get programmatic access to your queue

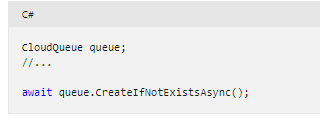
## How to connect to a queue

1. To connect to a queue, you first create a **CloudStorageAccount** with your connection string
2. The resulting object can then create a **CloudQueueClient**, which in turn can open a **CloudQueue** instance



## How to create a queue

1. You will use a common pattern for queue creation
2. Sender application should always be responsible for creating the queue
3. To make the creation simple, the client library exposes a **CreateIfNotExistsAsync** method

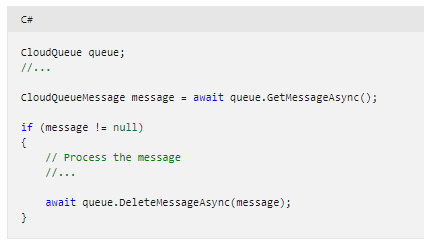


## How to send a message

1. To send a message, you instantiate a **CloudQueueMessage** object



## How to receive and delete a message



Carry on here

Exercise - Add a message to the queue