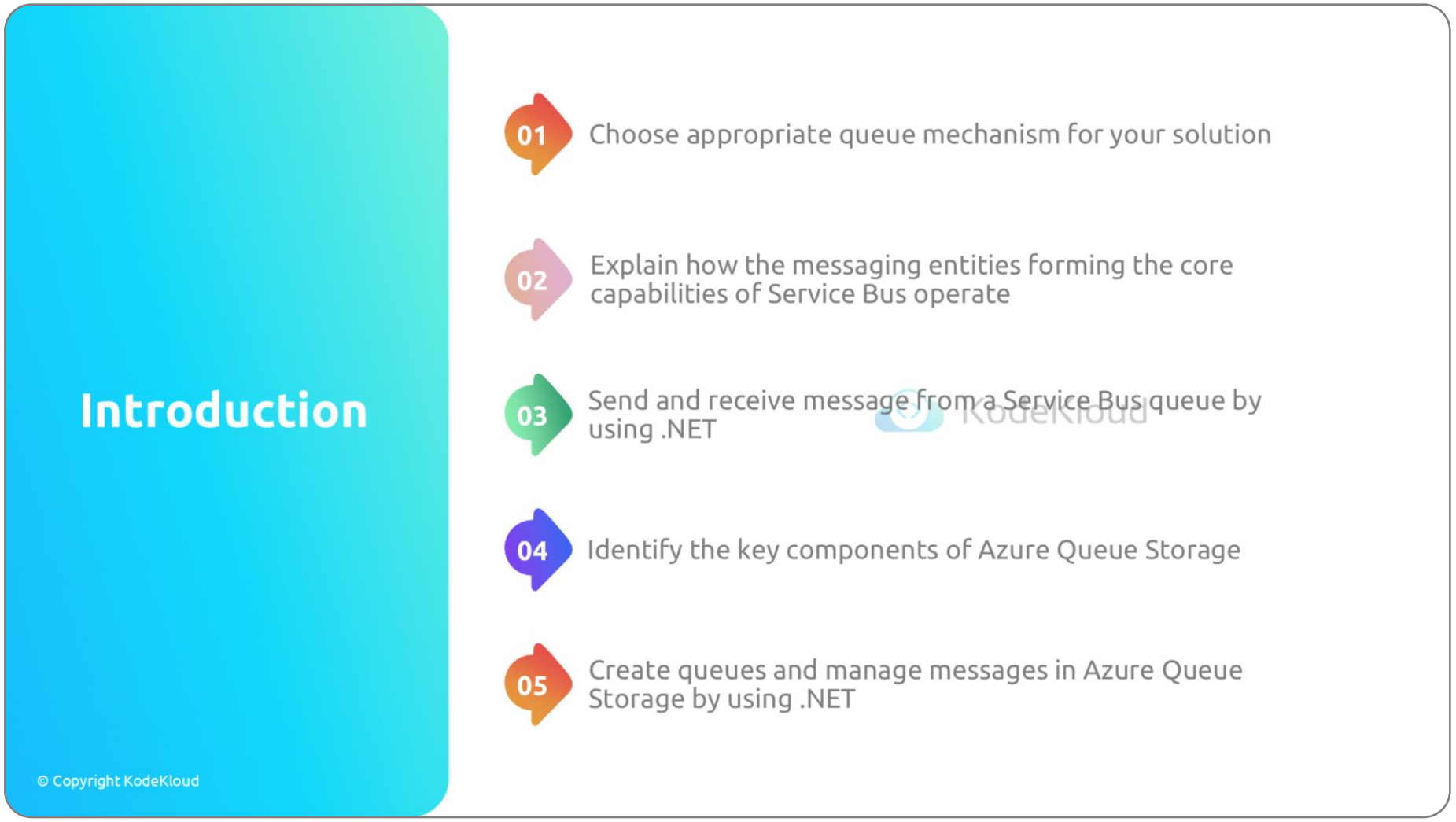
KodeKloud

Visit www.kodekloud.com to learn more.

Discovering Azure Message Queues



Welcome to the section on Developing Message-Based Solutions in Azure. We'll start by laying down the foundation of message-based architecture in cloud applications and how Azure supports this pattern through its various services.

Choosing the Appropriate Queue Mechanism:

We will Begin by understanding the different queue mechanisms available in Azure. We'll compare Azure Queue Storage and Azure Service Bus, discussing scenarios where one might be more suitable than the other. Remember, the choice hinges on factors like the need for ordering, duplicate detection, transaction support, and the size and rate of the messages.

Messaging Entities in Azure Service Bus:

We’ll then get into the nitty-gritty of Azure Service Bus, focusing on its core messaging entities: Queues, Topics, and Subscriptions. Each plays a key role in ensuring messages flow smoothly between different parts of your system.

Using .NET with Service Bus Queues:

For those of you developing with .NET, we'll explore how to send and receive messages using Service Bus Queues. We will cover hands-on examples and best practices for integrating Service Bus with your .NET applications.

Key Components of Azure Queue Storage:

The we will provide an overview of Azure Queue Storage's key components, emphasizing how it differs from Service Bus and when it's the best fit for your workloads. We'll talk about queues, messages, and visibility timeouts.

Managing Azure Queue Storage with .NET:

Finally, we'll wrap up this introduction by looking at how to create and manage queues in Azure Queue Storage using .NET. This will include practical demonstrations on pushing and pulling messages to ensure you have the skills to implement a robust message-based solution.

By the end of this module, you'll be confident in choosing and working with the right Azure messaging service to meet your application's needs.

Choosing a Message Queue Solution

Understanding Azure Queue Mechanisms

Azure offers two primary queue mechanisms for handling messaging within your applications: Service Bus queues and Storage queues. Understanding the distinctions between these two can help you choose the right solution based on your specific needs and application scenarios.

Service Bus vs Storage Queues

Service Bus queues	Storage queues
• Designed for complex, enterprise-level messaging	• Designed for simple, large-scale message queuing
• Supports queuing, publish/subscribe, transactions, forwarding, and dead-lettering	• Lacks advanced features but handles large volumes of messages
• E-commerce order processing where sequence and reliable delivery are critical	• Photo-sharing app for managing image processing tasks asynchronously
Go-to for sophisticated, feature-rich messaging needs	Simple message queuing scenarios

Service Bus Queues:

Service Bus queues are part of a broader Azure messaging infrastructure designed for more complex and enterprise-level messaging requirements.

They support advanced features like queuing, publish/subscribe, transactions, forwarding, and dead-lettering. Think of it as a sophisticated postal system that ensures your messages are delivered reliably, in order, and with various routing options.

As an example, imagine an e-commerce system where order processing must follow a strict sequence, and any failed

order needs to be rerouted or reviewed. Service Bus queues would be ideal for such scenarios, providing the necessary guarantees and features to handle these requirements.

Storage Queues:

Storage queues, on the other hand, are part of the Azure Storage infrastructure and are designed for simple, large-scale message queuing.

They are optimized for scenarios where you need to store a large number of messages without requiring the advanced features of Service Bus. It's like a basic message board where tasks are posted and picked up as needed, perfect for straightforward, high-volume workloads.

consider a photo-sharing app where users upload images that need to be processed asynchronously. Storage queues would be suitable for managing these image processing tasks due to their simplicity and scalability.

In summary, Service Bus queues are your go-to for sophisticated, feature-rich messaging needs, ensuring high reliability and complex routing. Meanwhile, Storage queues excel in high-volume, simple message queuing scenarios, providing a cost-effective and scalable solution.

Service Bus Queues – Considerations for Use

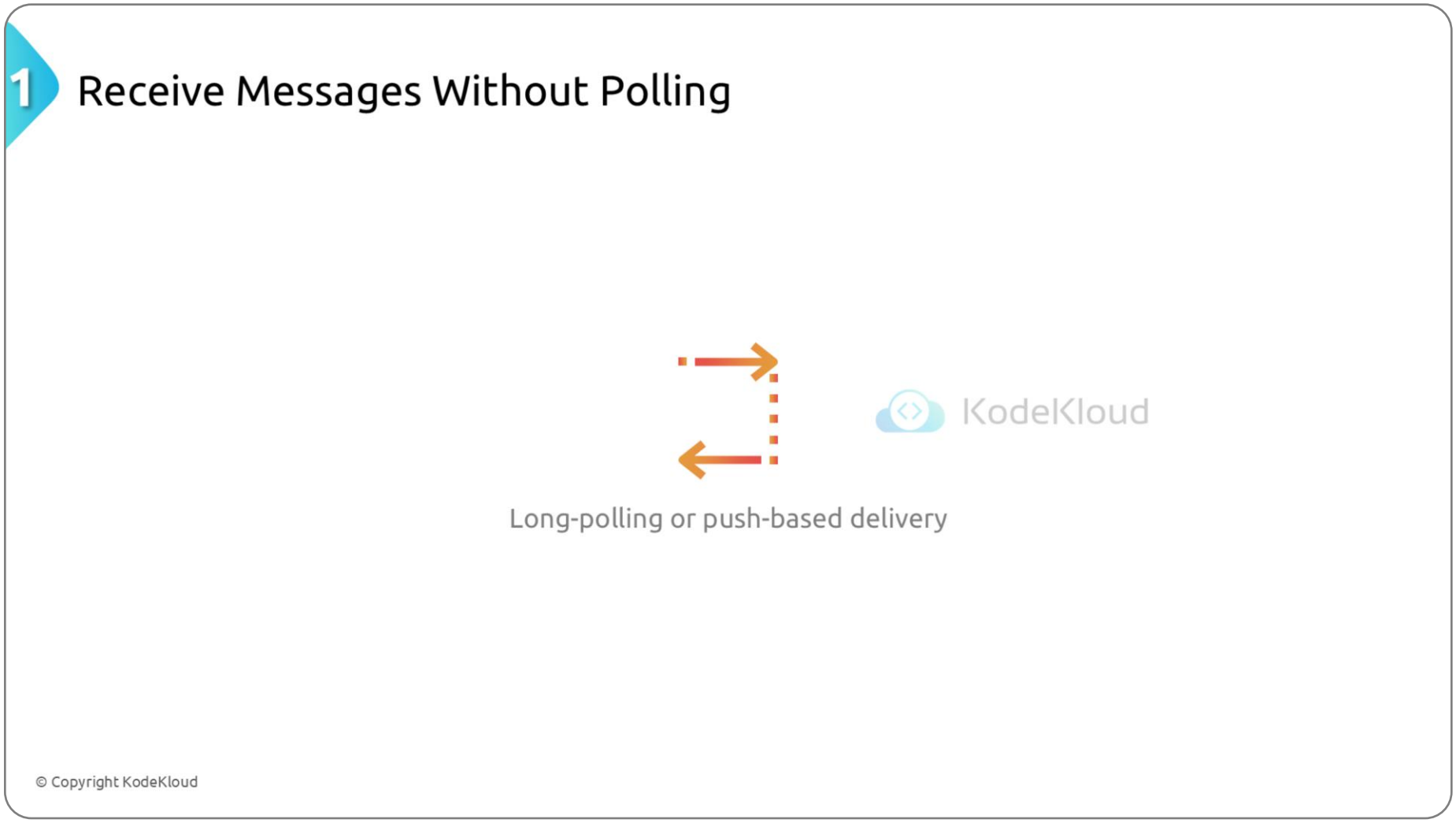
Receive messages without polling

KodeKloud

Let’s talk about some considerations for using Service Bus Queues:

1.) Receive Messages without Polling:
Explanation: Service Bus Queues support a feature called long-polling or push-based delivery, which allows your applications to receive messages without continuously polling the queue.
As an Analogy: Think of it like subscribing to notifications on your phone. Instead of checking your phone repeatedly for messages, notifications are pushed to you as they arrive.

Use Case: This is especially useful in scenarios where you need to respond to messages as soon as they are available, reducing unnecessary resource consumption.

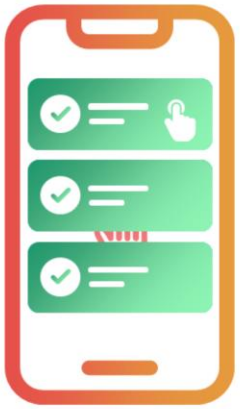


Let’s talk about some considerations for using Service Bus Queues:

1.) Receive Messages without Polling:
Explanation: Service Bus Queues support a feature called long-polling or push-based delivery, which allows your applications to receive messages without continuously polling the queue.
As an Analogy: Think of it like subscribing to notifications on your phone. Instead of checking your phone repeatedly for messages, notifications are pushed to you as they arrive.

Use Case: This is especially useful in scenarios where you need to respond to messages as soon as they are available, reducing unnecessary resource consumption.

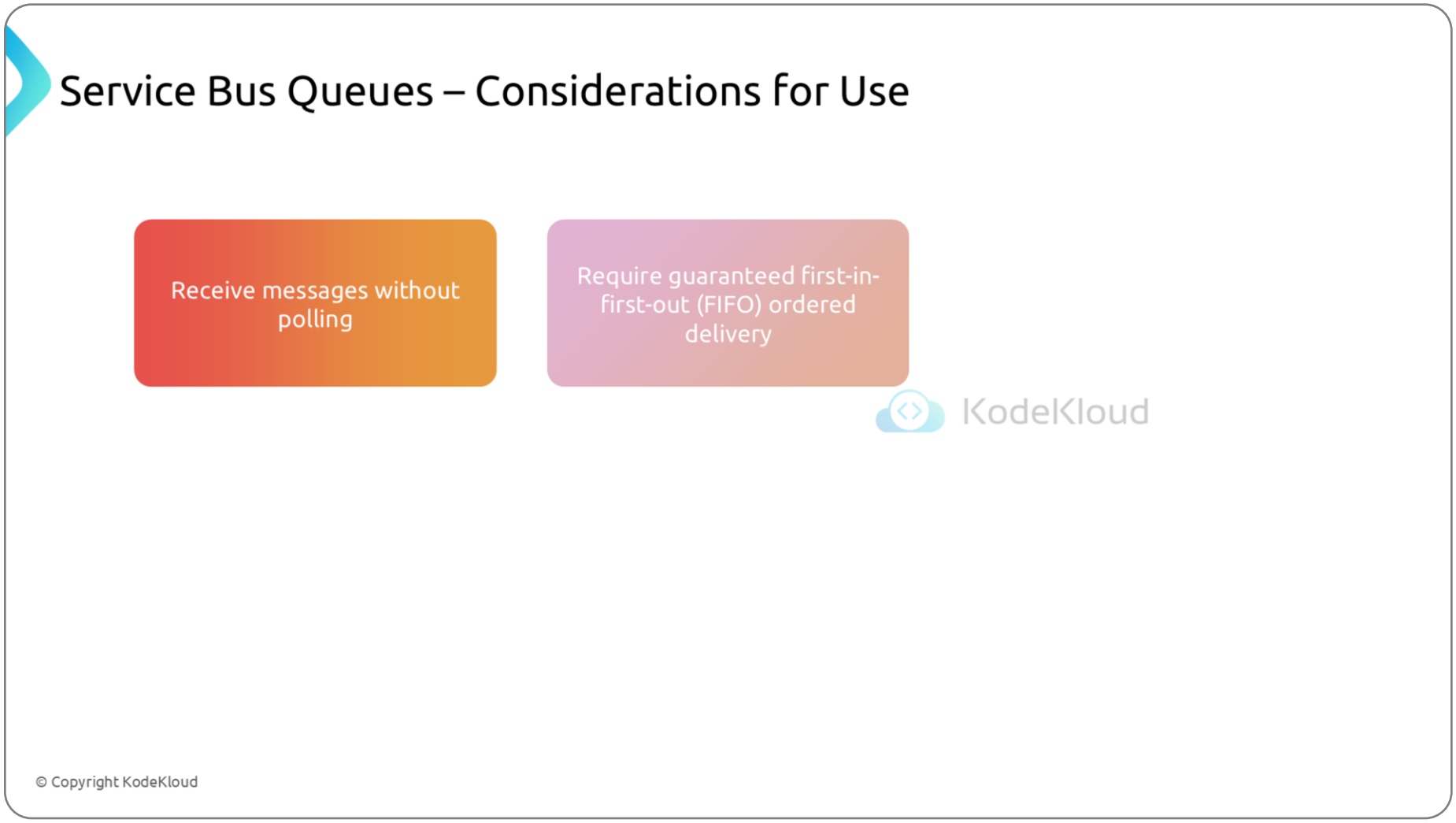
1 Receive Messages Without Polling



Let’s talk about some considerations for using Service Bus Queues:

1.) Receive Messages without Polling:
Explanation: Service Bus Queues support a feature called long-polling or push-based delivery, which allows your applications to receive messages without continuously polling the queue.
As an Analogy: Think of it like subscribing to notifications on your phone. Instead of checking your phone repeatedly for messages, notifications are pushed to you as they arrive.

Use Case: This is especially useful in scenarios where you need to respond to messages as soon as they are available, reducing unnecessary resource consumption.



Let’s talk about some considerations for using Service Bus Queues:

2.)Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery:
Explanation: Service Bus Queues ensure that messages are delivered and processed in the exact order they were sent. This is known as FIFO (First-In-First-Out) delivery.
Analogy: This is like a line at a movie theater. The first person in line is the first to get a ticket, ensuring fairness and order.
Use Case: This is critical for applications where the order of operations matters, such as processing financial transactions or

sequential task execution.

2.)Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery:
Explanation: Service Bus Queues ensure that messages are delivered and processed in the exact order they were sent. This is known as FIFO (First-In-First-Out) delivery.
Analogy: This is like a line at a movie theater. The first person in line is the first to get a ticket, ensuring fairness and order.
Use Case: This is critical for applications where the order of operations matters, such as processing financial transactions or sequential task execution.

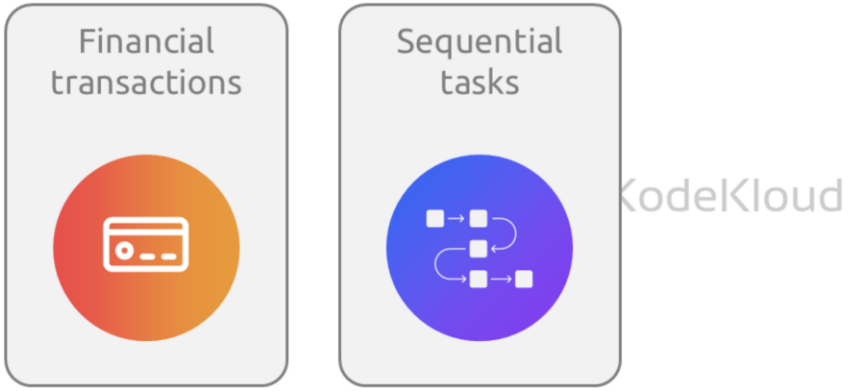
2.)Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery:
Explanation: Service Bus Queues ensure that messages are delivered and processed in the exact order they were sent. This is known as FIFO (First-In-First-Out) delivery.
Analogy: This is like a line at a movie theater. The first person in line is the first to get a ticket, ensuring fairness and order.
Use Case: This is critical for applications where the order of operations matters, such as processing financial transactions or sequential task execution.

2.)Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery:
Explanation: Service Bus Queues ensure that messages are delivered and processed in the exact order they were sent. This is known as FIFO (First-In-First-Out) delivery.
Analogy: This is like a line at a movie theater. The first person in line is the first to get a ticket, ensuring fairness and order.
Use Case: This is critical for applications where the order of operations matters, such as processing financial transactions or sequential task execution.

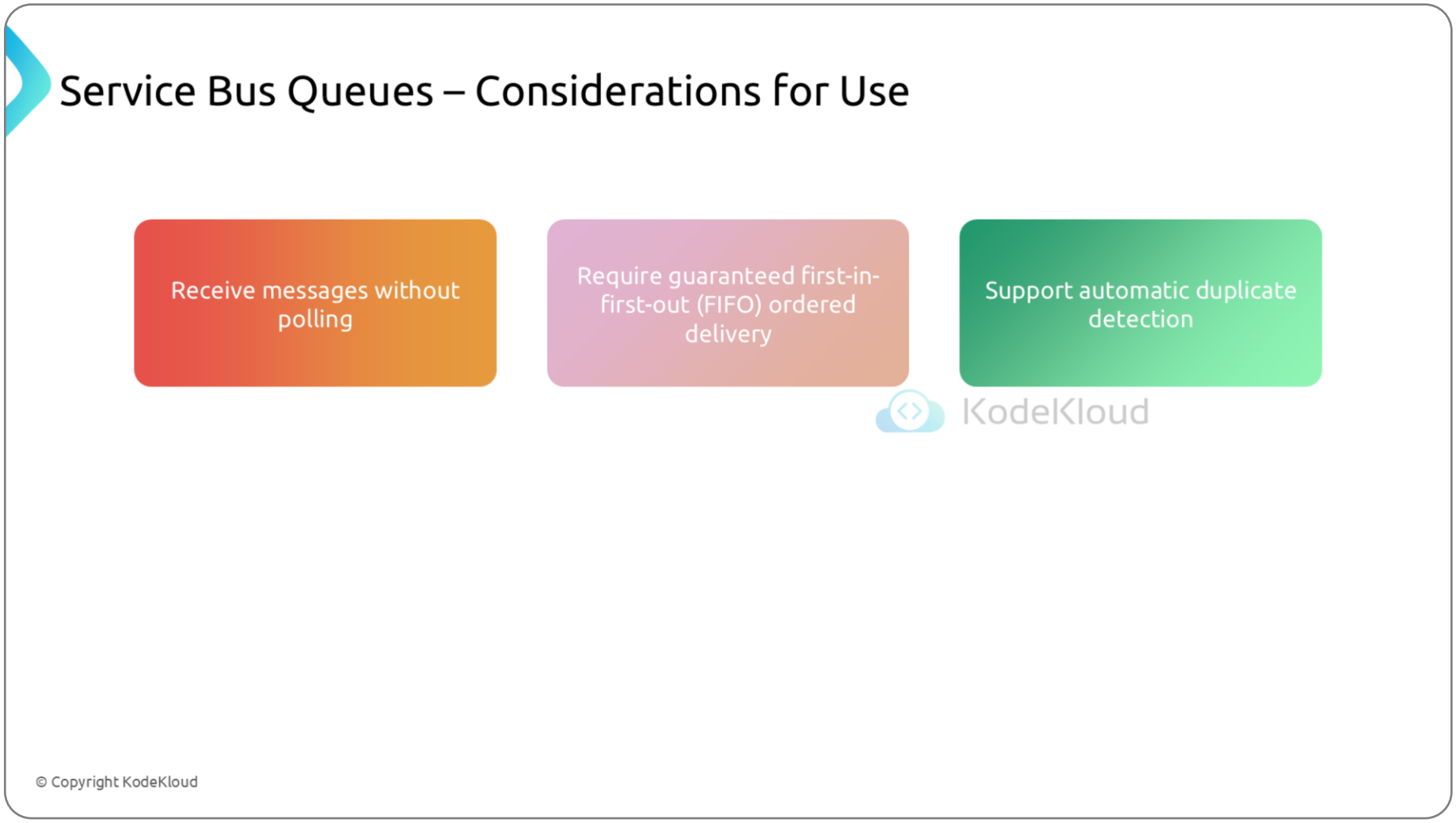
2 Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery

2.)Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery:
Explanation: Service Bus Queues ensure that messages are delivered and processed in the exact order they were sent. This is known as FIFO (First-In-First-Out) delivery.
Analogy: This is like a line at a movie theater. The first person in line is the first to get a ticket, ensuring fairness and order.
Use Case: This is critical for applications where the order of operations matters, such as processing financial transactions or sequential task execution.

2 Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery



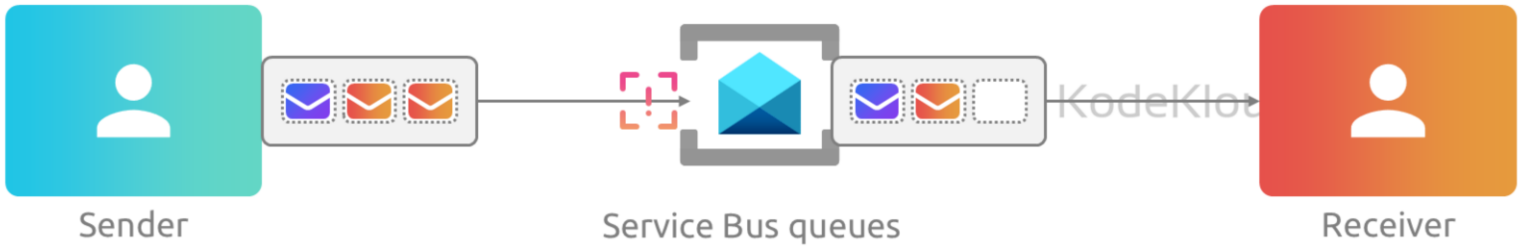
2.)Require Guaranteed First-In-First-Out (FIFO) Ordered Delivery:
Explanation: Service Bus Queues ensure that messages are delivered and processed in the exact order they were sent. This is known as FIFO (First-In-First-Out) delivery.
Analogy: This is like a line at a movie theater. The first person in line is the first to get a ticket, ensuring fairness and order.
Use Case: This is critical for applications where the order of operations matters, such as processing financial transactions or sequential task execution.



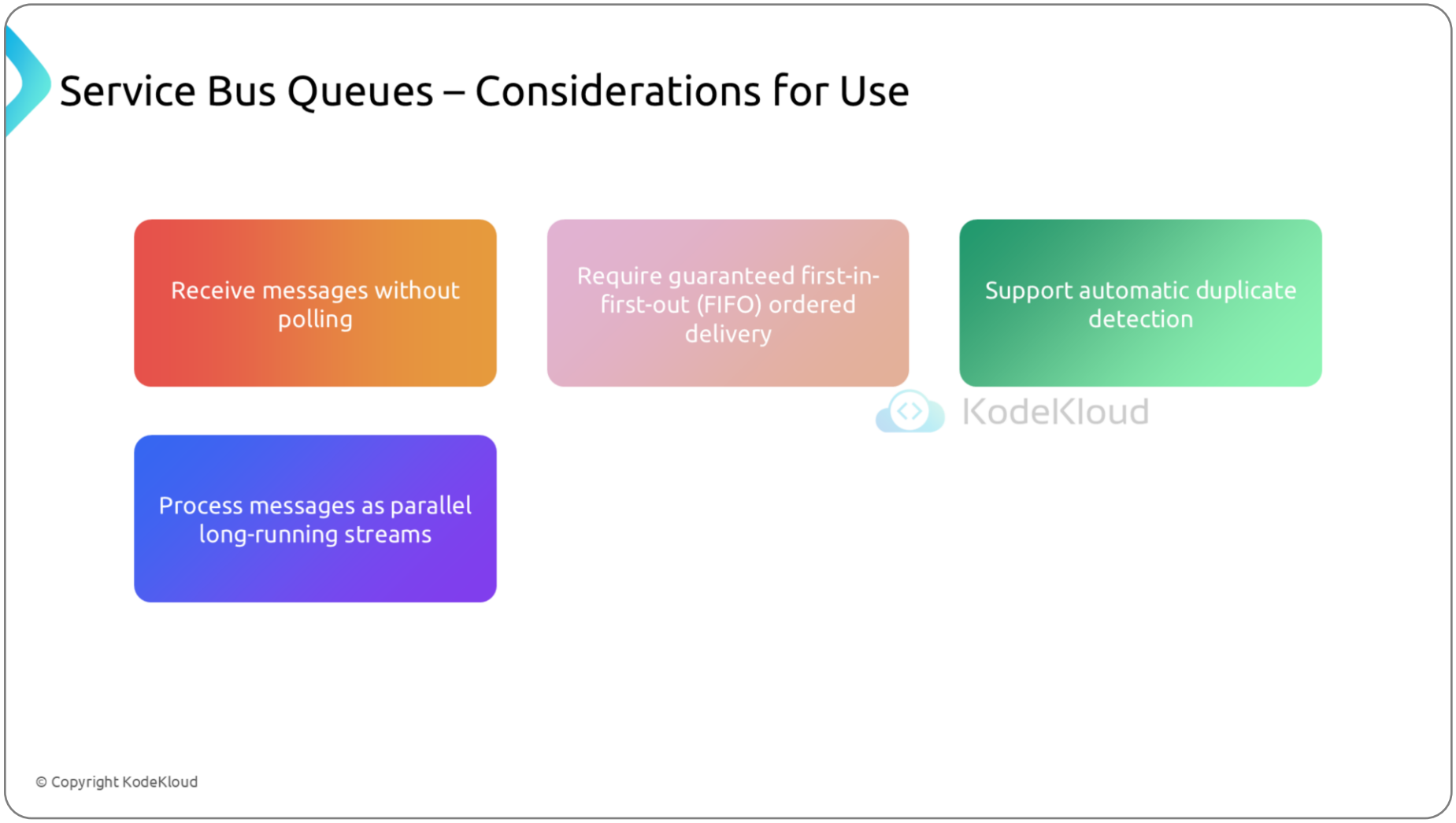
Let’s talk about some considerations for using Service Bus Queues:

3.) Support Automatic Duplicate Detection:
Explanation: Service Bus Queues can automatically detect and handle duplicate messages, ensuring that each message is processed only once.
Use Case: This feature is beneficial in scenarios where message duplication can cause issues, such as updating inventory levels in an e-commerce application.

3 Support Automatic Duplicate Detection

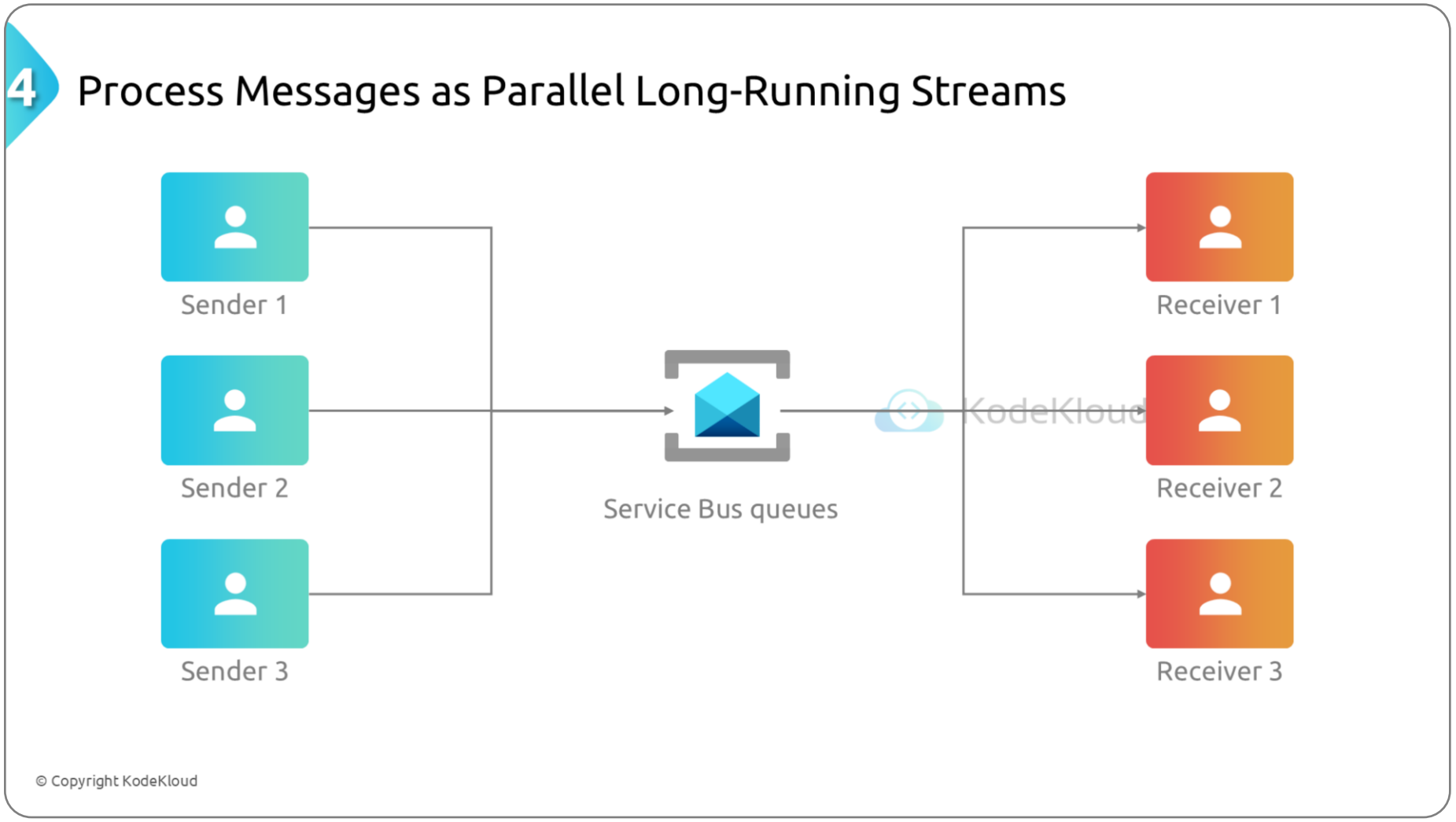


3.) Support Automatic Duplicate Detection:
Explanation: Service Bus Queues can automatically detect and handle duplicate messages, ensuring that each message is processed only once.
Use Case: This feature is beneficial in scenarios where message duplication can cause issues, such as updating inventory levels in an e-commerce application.



Let’s talk about some considerations for using Service Bus Queues:

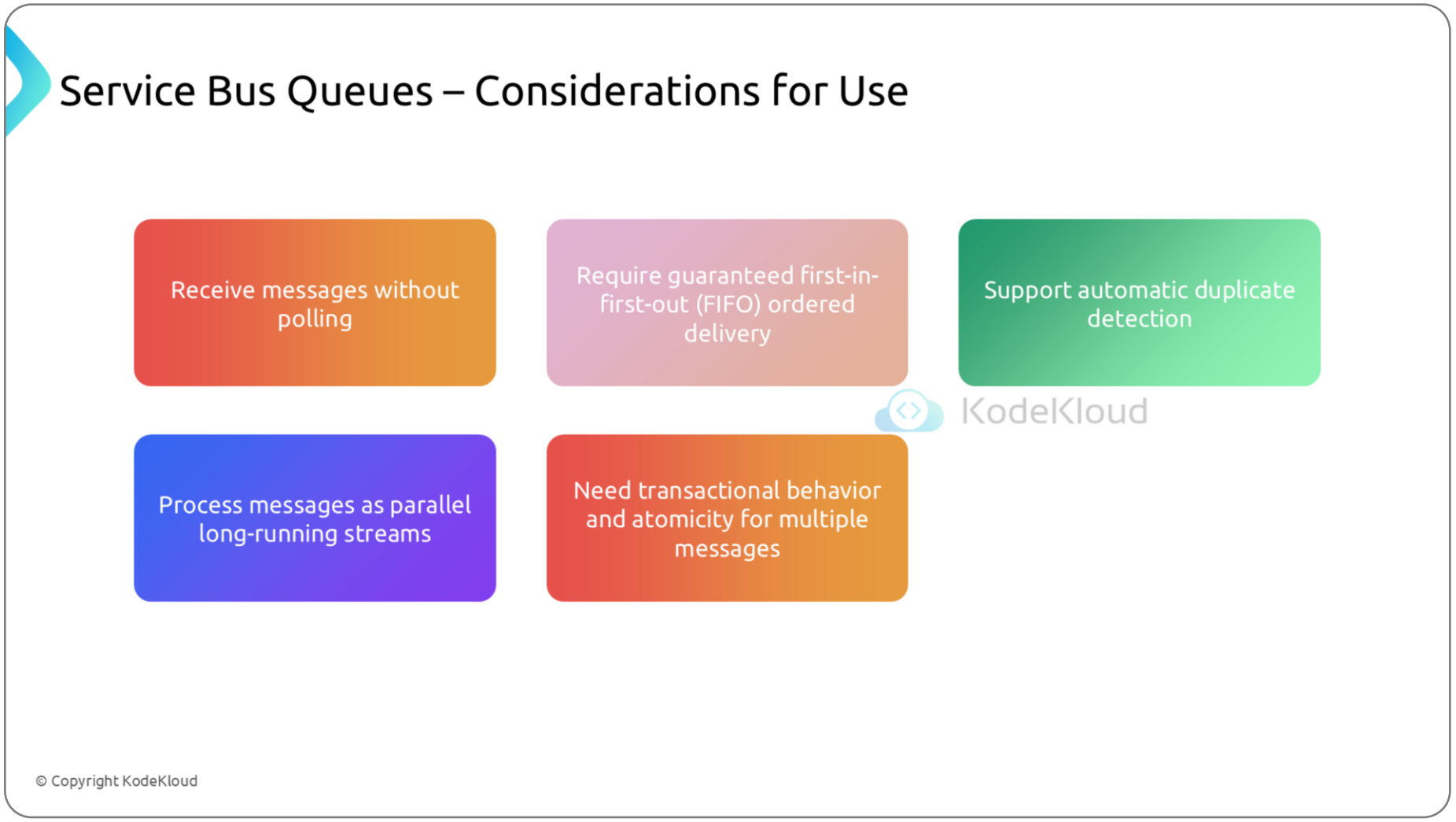
4.) Process Messages as Parallel Long-Running Streams:
Explanation: Service Bus Queues support parallel processing of messages, allowing multiple messages to be processed concurrently across different receivers.
Analogy: This is like multiple checkout counters in a supermarket. Each counter processes customers independently, speeding up the overall process.



4. Process Messages as Parallel Long-Running Streams:
Explanation: Service Bus Queues support parallel processing of messages, allowing multiple messages to be processed concurrently across different receivers.
Analogy: This is like multiple checkout counters in a supermarket. Each counter processes customers independently, speeding up the overall process.

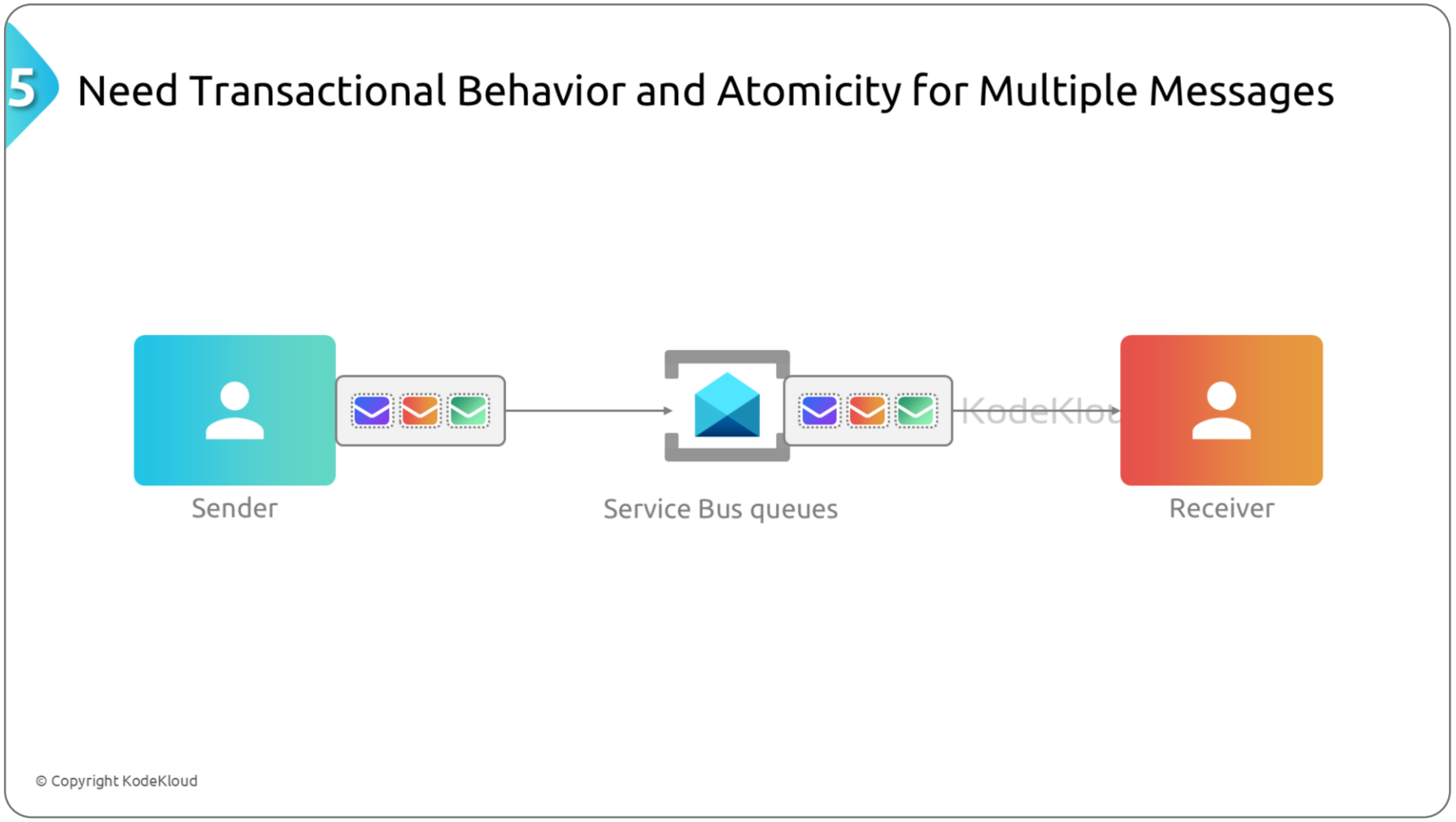
4 Process Messages as Parallel Long-Running Streams

4. Process Messages as Parallel Long-Running Streams:
Explanation: Service Bus Queues support parallel processing of messages, allowing multiple messages to be processed concurrently across different receivers.
Analogy: This is like multiple checkout counters in a supermarket. Each counter processes customers independently, speeding up the overall process.



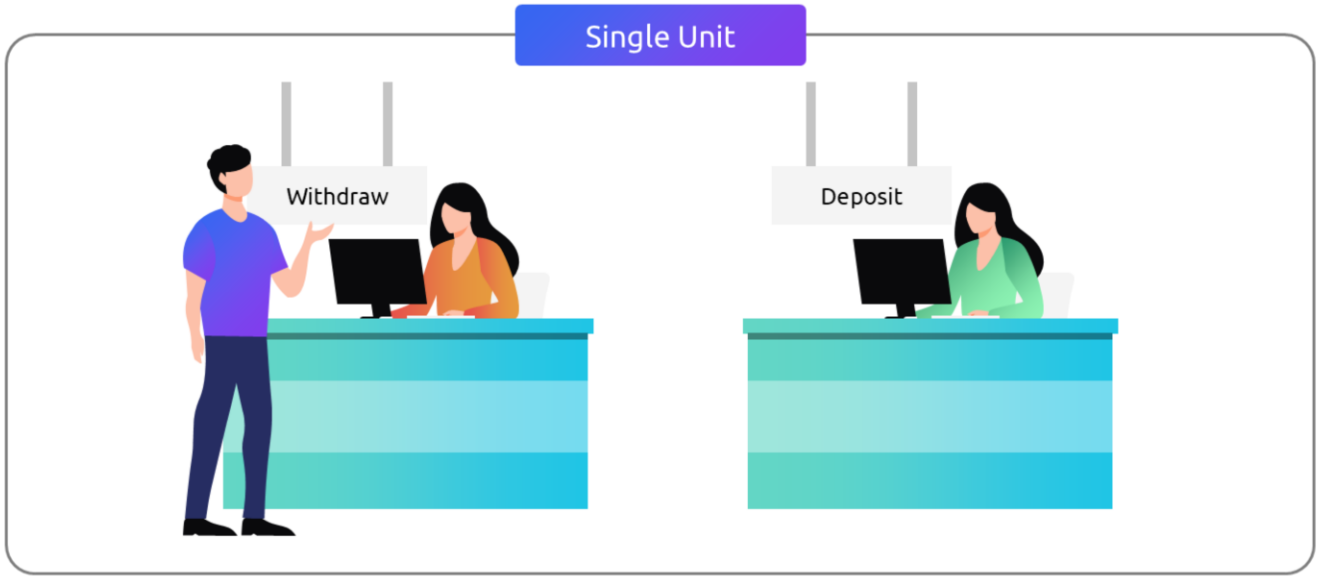
Let’s talk about some considerations for using Service Bus Queues:

5.)Need Transactional Behaviour and Atomicity for Multiple Messages:
Explanation: Service Bus Queues provide support for transactional operations, ensuring that multiple messages are processed as a single unit of work, maintaining atomicity.
Analogy: Think of it as a bank transaction where withdrawing from one account and depositing to another are treated as a single operation – either both succeed or both fail.



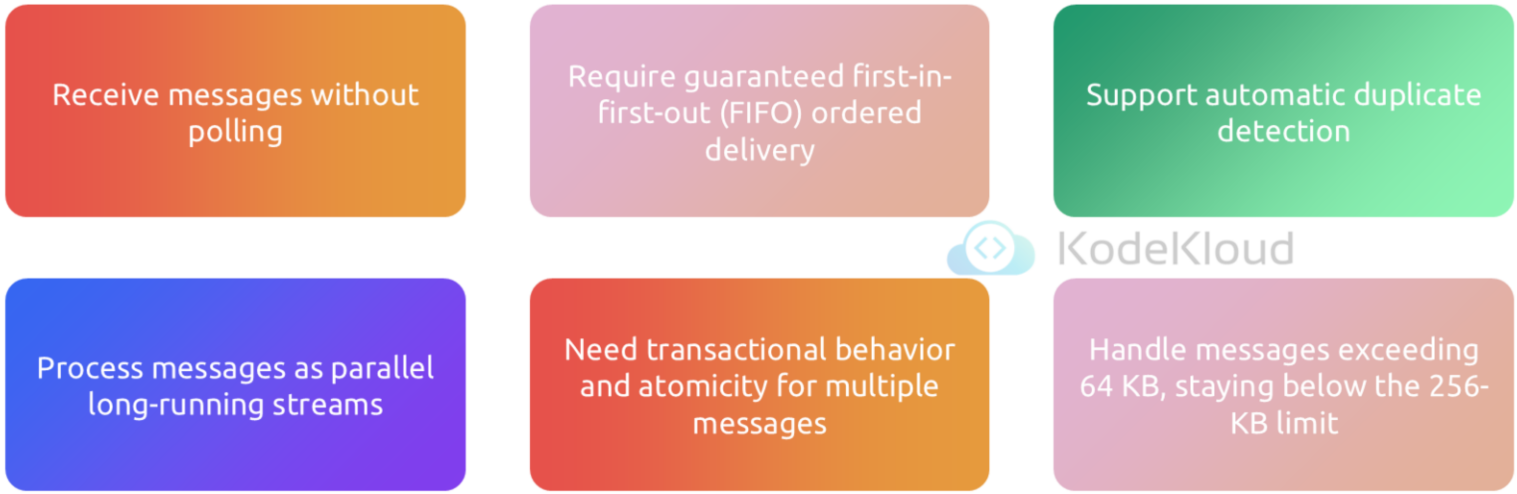
5.)Need Transactional Behaviour and Atomicity for Multiple Messages:
Explanation: Service Bus Queues provide support for transactional operations, ensuring that multiple messages are processed as a single unit of work, maintaining atomicity.
Analogy: Think of it as a bank transaction where withdrawing from one account and depositing to another are treated as a single operation – either both succeed or both fail.

5 Need Transactional Behavior and Atomicity for Multiple Messages



5.)Need Transactional Behaviour and Atomicity for Multiple Messages:
Explanation: Service Bus Queues provide support for transactional operations, ensuring that multiple messages are processed as a single unit of work, maintaining atomicity.
Analogy: Think of it as a bank transaction where withdrawing from one account and depositing to another are treated as a single operation – either both succeed or both fail.

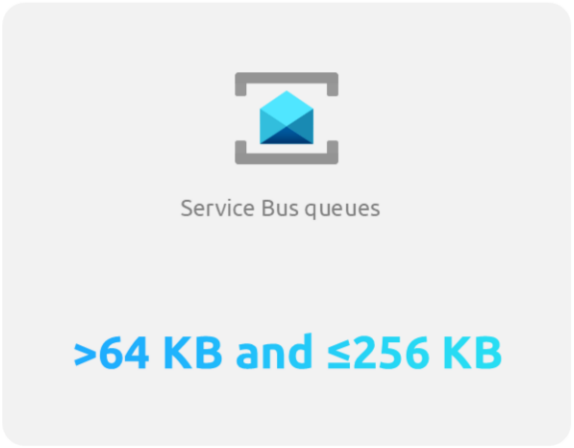
Service Bus Queues – Considerations for Use



Let’s talk about some considerations for using Service Bus Queues:

6.) Handle Messages Exceeding 64 KB, Staying Below the 256-KB Limit:
Explanation: Service Bus Queues can handle larger message sizes, accommodating messages that exceed 64 KB but stay within the 256 KB limit.

6 Handle Messages Exceeding 64 KB, Staying Below the 256-KB Limit



Handle Messages Exceeding 64 KB, Staying Below the 256-KB Limit:
Explanation: Service Bus Queues can handle larger message sizes, accommodating messages that exceed 64 KB but stay within the 256 KB limit.

Storage Queues – Considerations for Use

KodeKloud

Now, let's turn our focus to Azure Storage Queues and explore the key considerations for their use.

1.) Need to Store Over 80 GB of Messages in a Queue:
Explanation: One of the major advantages of Storage Queues is their ability to store a very large number of messages, exceeding 80 GB in size.
Use Case: This feature is beneficial for applications that need to queue a high volume of messages, such as logging data or handling large-scale data processing tasks.

Require Progress Tracking for Message Processing, Allowing Seamless Recovery After Worker Crashes:

Explanation: Storage Queues offer the ability to track the progress of message processing, which is crucial for ensuring that message processing can seamlessly recover in the event of worker crashes or failures.

Use Case: This is particularly useful for applications that need to ensure reliability and continuity in message processing, such as processing customer orders or managing background tasks.

Demand Server-Side Logs for All Transactions Executed Against Queues:

Explanation: Storage Queues provide server-side logging capabilities, allowing you to maintain a detailed log of all transactions executed against the queues.

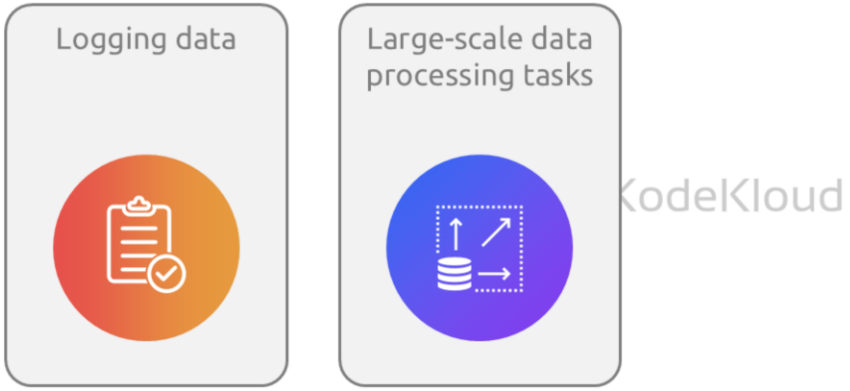
Analogy: Think of it as having a detailed logbook that records every transaction and interaction, providing transparency and accountability.

Use Case: This feature is essential for scenarios that require detailed auditing and tracking of message transactions, such as compliance reporting and security monitoring.

1 Need to Store Over 80 GB of Messages in a Queue

1.) Need to Store Over 80 GB of Messages in a Queue:
Explanation: One of the major advantages of Storage Queues is their ability to store a very large number of messages, exceeding 80 GB in size.
Use Case: This feature is beneficial for applications that need to queue a high volume of messages, such as logging data or handling large-scale data processing tasks.

1 Need to Store Over 80 GB of Messages in a Queue



1.) Need to Store Over 80 GB of Messages in a Queue:
Explanation: One of the major advantages of Storage Queues is their ability to store a very large number of messages, exceeding 80 GB in size.
Use Case: This feature is beneficial for applications that need to queue a high volume of messages, such as logging data or handling large-scale data processing tasks.

Now, let's turn our focus to Azure Storage Queues and explore the key considerations for their use.

1.) Need to Store Over 80 GB of Messages in a Queue:
Explanation: One of the major advantages of Storage Queues is their ability to store a very large number of messages, exceeding 80 GB in size.
Use Case: This feature is beneficial for applications that need to queue a high volume of messages, such as logging data or handling large-scale data processing tasks.

Require Progress Tracking for Message Processing, Allowing Seamless Recovery After Worker Crashes:

Explanation: Storage Queues offer the ability to track the progress of message processing, which is crucial for ensuring that message processing can seamlessly recover in the event of worker crashes or failures.

Use Case: This is particularly useful for applications that need to ensure reliability and continuity in message processing, such as processing customer orders or managing background tasks.

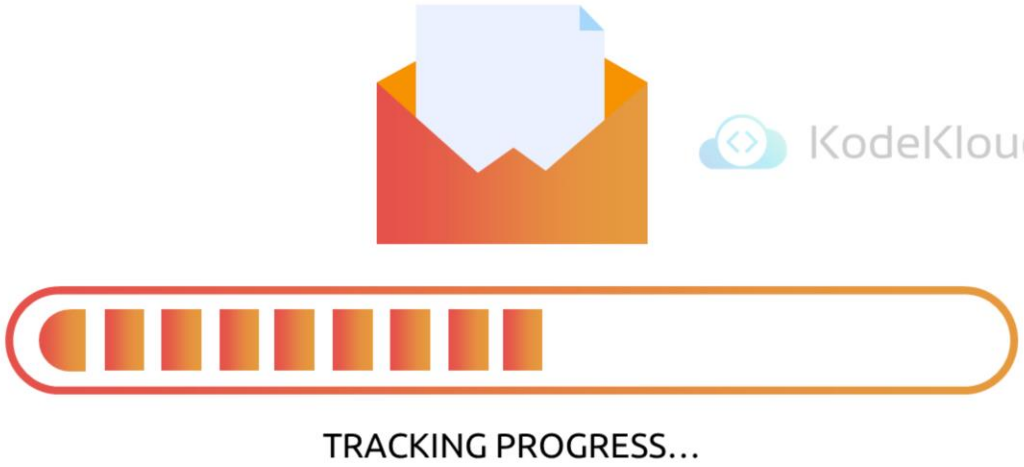
Demand Server-Side Logs for All Transactions Executed Against Queues:

Explanation: Storage Queues provide server-side logging capabilities, allowing you to maintain a detailed log of all transactions executed against the queues.

Analogy: Think of it as having a detailed logbook that records every transaction and interaction, providing transparency and accountability.

Use Case: This feature is essential for scenarios that require detailed auditing and tracking of message transactions, such as compliance reporting and security monitoring.

2 Require Progress Tracking for Message Processing, Allowing Seamless Recovery From Worker Crashes

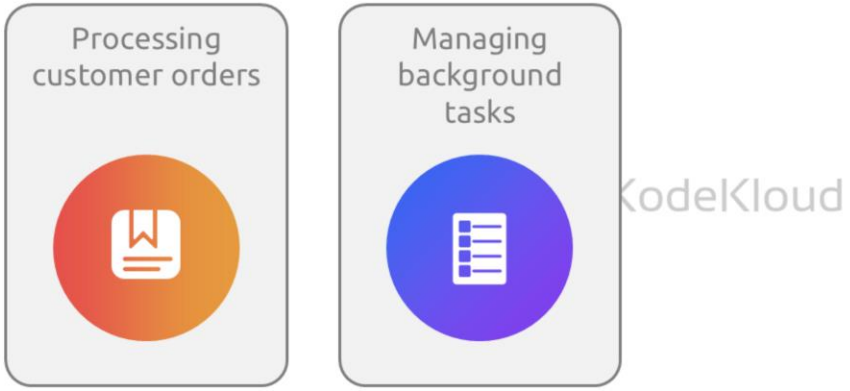


Require Progress Tracking for Message Processing, Allowing Seamless Recovery After Worker Crashes:

Explanation: Storage Queues offer the ability to track the progress of message processing, which is crucial for ensuring that message processing can seamlessly recover in the event of worker crashes or failures.

Use Case: This is particularly useful for applications that need to ensure reliability and continuity in message processing, such as processing customer orders or managing background tasks.

2 Require Progress Tracking for Message Processing, Allowing Seamless Recovery From Worker Crashes



Require Progress Tracking for Message Processing, Allowing Seamless Recovery After Worker Crashes:

Explanation: Storage Queues offer the ability to track the progress of message processing, which is crucial for ensuring that message processing can seamlessly recover in the event of worker crashes or failures.

Use Case: This is particularly useful for applications that need to ensure reliability and continuity in message processing, such as processing customer orders or managing background tasks.

Storage Queues – Considerations for Use

Now, let's turn our focus to Azure Storage Queues and explore the key considerations for their use.

1.) Need to Store Over 80 GB of Messages in a Queue:
Explanation: One of the major advantages of Storage Queues is their ability to store a very large number of messages, exceeding 80 GB in size.
Use Case: This feature is beneficial for applications that need to queue a high volume of messages, such as logging data or handling large-scale data processing tasks.

Require Progress Tracking for Message Processing, Allowing Seamless Recovery After Worker Crashes:

Explanation: Storage Queues offer the ability to track the progress of message processing, which is crucial for ensuring that message processing can seamlessly recover in the event of worker crashes or failures.

Use Case: This is particularly useful for applications that need to ensure reliability and continuity in message processing, such as processing customer orders or managing background tasks.

Demand Server-Side Logs for All Transactions Executed Against Queues:

Explanation: Storage Queues provide server-side logging capabilities, allowing you to maintain a detailed log of all transactions executed against the queues.

Analogy: Think of it as having a detailed logbook that records every transaction and interaction, providing transparency and accountability.

Use Case: This feature is essential for scenarios that require detailed auditing and tracking of message transactions, such as compliance reporting and security monitoring.

3

Demand Server-Side Logs for All Transactions Executed Against Queues

KodeKloud

Server-side logging capabilities

Demand Server-Side Logs for All Transactions Executed Against Queues:

Explanation: Storage Queues provide server-side logging capabilities, allowing you to maintain a detailed log of all transactions executed against the queues.

Analogy: Think of it as having a detailed logbook that records every transaction and interaction, providing transparency and accountability.

Use Case: This feature is essential for scenarios that require detailed auditing and tracking of message transactions, such as compliance reporting and security monitoring.

3

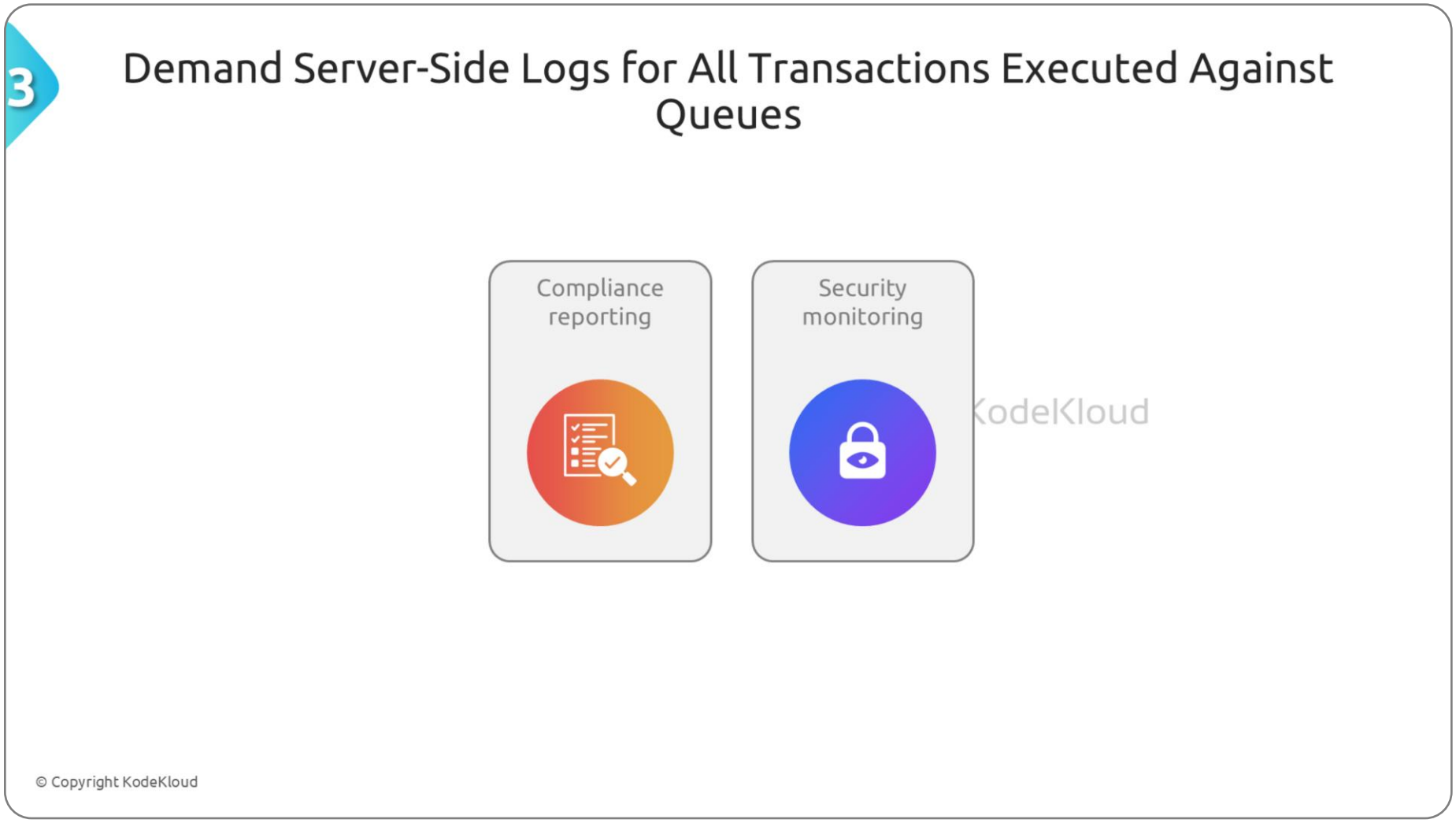
Demand Server-Side Logs for All Transactions Executed Against Queues

Demand Server-Side Logs for All Transactions Executed Against Queues:

Explanation: Storage Queues provide server-side logging capabilities, allowing you to maintain a detailed log of all transactions executed against the queues.

Analogy: Think of it as having a detailed logbook that records every transaction and interaction, providing transparency and accountability.

Use Case: This feature is essential for scenarios that require detailed auditing and tracking of message transactions, such as compliance reporting and security monitoring.



Demand Server-Side Logs for All Transactions Executed Against Queues:

Explanation: Storage Queues provide server-side logging capabilities, allowing you to maintain a detailed log of all transactions executed against the queues.

Analogy: Think of it as having a detailed logbook that records every transaction and interaction, providing transparency and accountability.

Use Case: This feature is essential for scenarios that require detailed auditing and tracking of message transactions, such as compliance reporting and security monitoring.

Basic vs Standard vs Premium

Service Bus comes in Basic, standard, and premium tiers. Here's how they compare:

Feature	Basic	Standard	Premium
Queues	✓	✓	✓
Scheduled messages	✓	✓	✓
Topics		✓	✓
Transactions		✓	✓
De-duplication		✓	✓
Sessions		✓	✓
ForwardTo/SendVia		✓	✓
Message Size	256 KB	256 KB	100 MB
Resource isolation			✓
Geo-Disaster Recovery (Geo-DR)			✓ *Requires additional Service Bus Premium namespaces in another region.
Java Messaging Service (JMS) 2.0 Support			✓
Availability Zones (AZ) support			✓

Let's explore the key differences between the Basic, Standard, and Premium tiers of Azure Service Bus, focusing on their unique features and when to use them.
First up, Queues. All tiers support queues, allowing you to store and retrieve messages in a first-in, first-out manner. Think of this like a waiting line where the first message in is the first message out.
Next, we have Scheduled Messages. This feature is also available across all tiers, enabling you to send messages at a specified future time. Imagine scheduling an email to be sent at a later date – it's the same concept here.

Moving on to Topics. These are available in the Standard and Premium tiers. Topics allow you to implement a publish/subscribe pattern, which is similar to subscribing to a newsletter where multiple subscribers can receive the same message.

Now, let's talk about Transactions. This feature is supported in both the Standard and Premium tiers and ensures that multiple operations are executed as a single unit, providing reliability. Think of this like a bank transaction where all steps must complete successfully or none at all.

De-duplication is another important feature, available in Standard and Premium tiers. This automatically removes duplicate messages, similar to a filter that ensures you only receive unique messages, preventing any repeats.

Sessions feature allows you to process related messages in a specific order. Imagine processing a series of related steps in a workflow

With ForwardTo/SendVia, you can forward messages to other queues or topics, providing flexibility in message routing.

When it comes to Message Size, the Basic and Standard tiers support messages up to 256 KB. However, the Premium tier supports larger messages up to 100 MB, making it suitable for applications that need to handle larger data payloads.

Resource Isolation is exclusive to the Premium tier. This ensures dedicated resources for your application, leading to predictable performance, which is crucial for high-availability applications.
Geo-Disaster Recovery (Geo-DR) is supported in the Premium tier, providing enhanced resiliency and cross-region disaster recovery. This means your application can withstand regional failures, ensuring business continuity.
The Java Messaging Service (JMS) 2.0 Support is available only in the Premium tier, which is ideal for advanced messaging scenarios requiring this protocol.

Finally, Availability Zones (AZ) Support is exclusive to the Premium tier as well, offering high availability and fault tolerance across different zones, ensuring robust disaster recovery capabilities.

For basic applications with minimal messaging needs, the Basic tier is cost-effective. Use this tier for simple queue-based applications without the need for advanced features. Microsoft does not recommend this tier for production workloads.

The Standard tier is suitable for applications requiring more advanced features like transactions, de-duplication, and sessions. This tier is ideal for applications needing reliable message processing but without the need for dedicated resources.

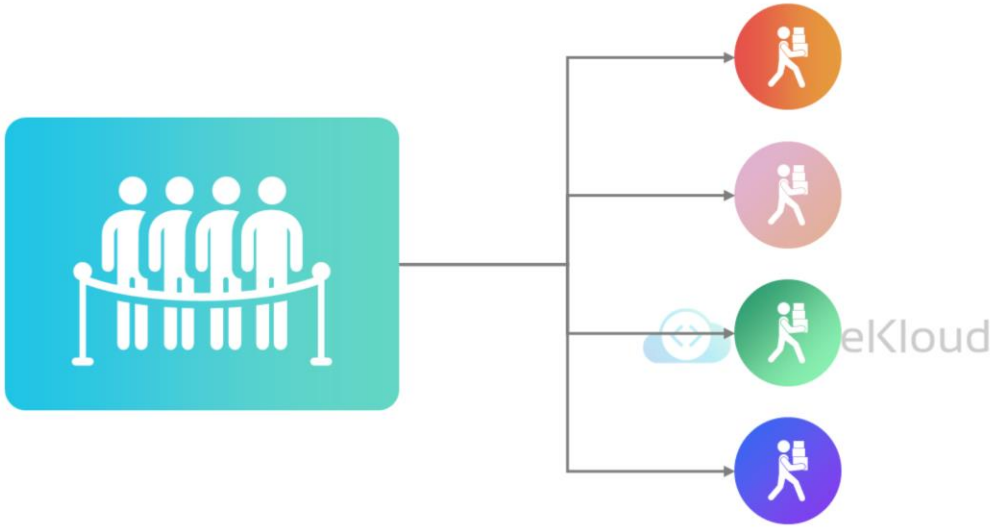
The Premium tier is best for mission-critical applications that demand high performance, large message support, and advanced features like resource isolation, Geo-Disaster Recovery, and JMS 2.0 support. Use this tier for applications where predictable performance and high availability are crucial.

Discovering Service Bus Queues, Topics, and Subscriptions

Queues

Azure Service Bus Queues provide a way to store and retrieve messages in a first-in, first-out (FIFO) manner. Imagine a waiting line where the first person in line is the first to be served. Queues are ideal for scenarios where you need to process messages sequentially and ensure that each message is handled only once. They offer reliable message delivery to one or more competing consumers, meaning multiple receivers can process messages from the same queue, but each message is processed by only one receiver.

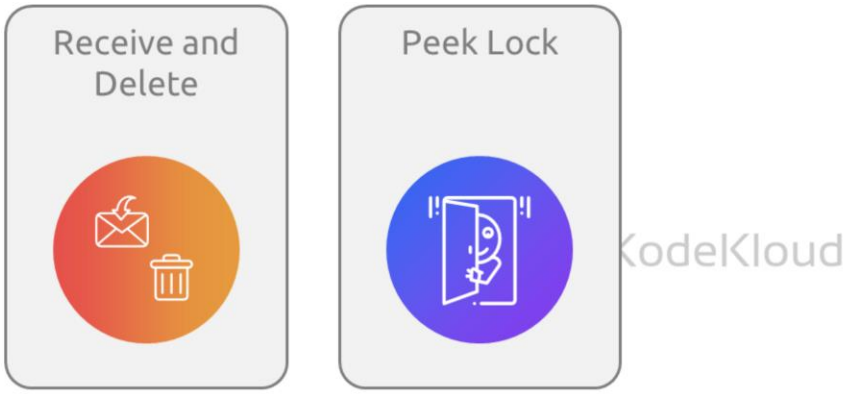
Queues



Queues offer First-In, First-Out (FIFO) message delivery to one or more competing consumers.

Azure Service Bus Queues provide a way to store and retrieve messages in a first-in, first-out (FIFO) manner. Imagine a waiting line where the first person in line is the first to be served. Queues are ideal for scenarios where you need to process messages sequentially and ensure that each message is handled only once. They offer reliable message delivery to one or more competing consumers, meaning multiple receivers can process messages from the same queue, but each message is processed by only one receiver.

Receive Modes

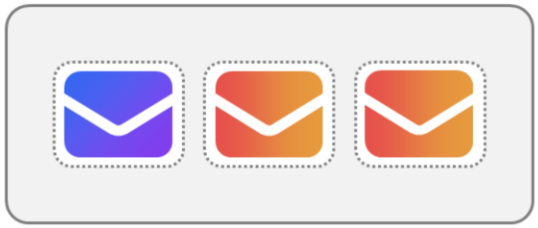


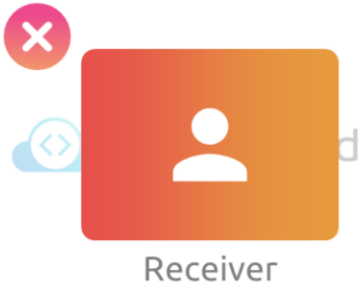
Here are two different modes in which Service Bus can receive messages: Receive and Delete and Peek Lock.
Receive and Delete:
In the Receive and Delete mode, when a receiver picks up a message, it is immediately marked as consumed and deleted from the queue. This mode is simple and efficient but does not provide any safety net. If a failure occurs after receiving the message but before processing it, the message is lost.
Example Scenario: This mode is suitable for non-critical applications where occasional message loss is acceptable, such as logging or telemetry data.

Peek Lock:
In the Peek Lock mode, the message is locked when received but not immediately deleted. The receiver has the opportunity to process the message and explicitly complete the processing. If the receiver fails to process the message, it can be reprocessed later.
Example Scenario: This mode is ideal for applications requiring high reliability and guaranteed message processing, such as order processing or financial transactions.

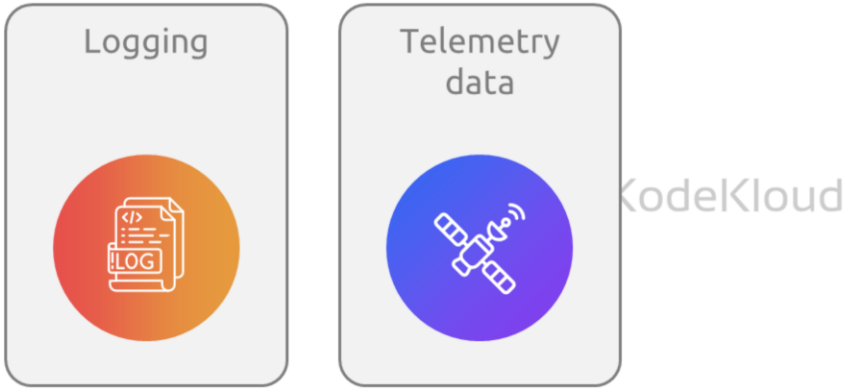
Note: This is the script for slide 39 to 43.

Receive and Delete

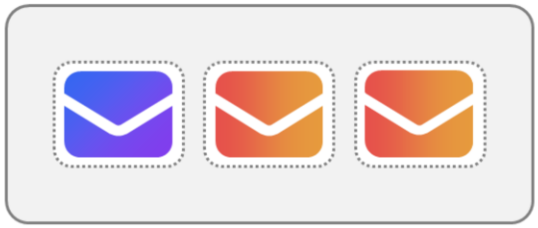


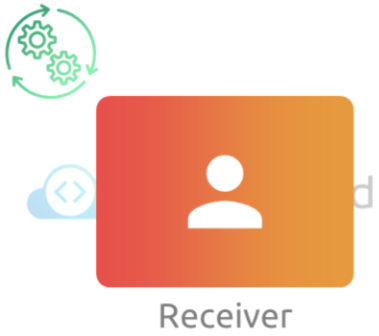


Receive and Delete

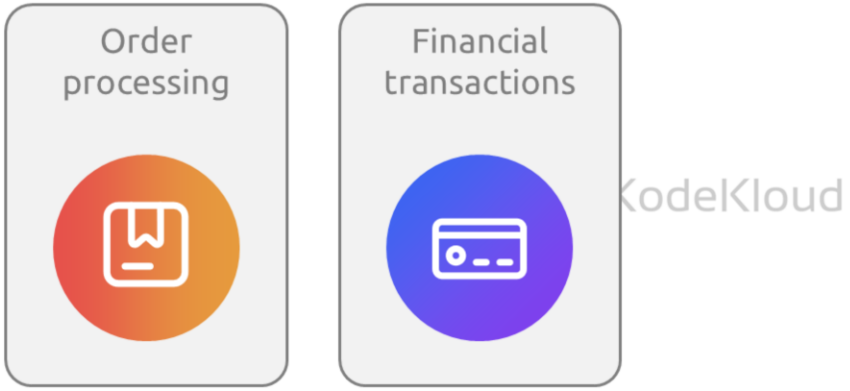


Peek Lock

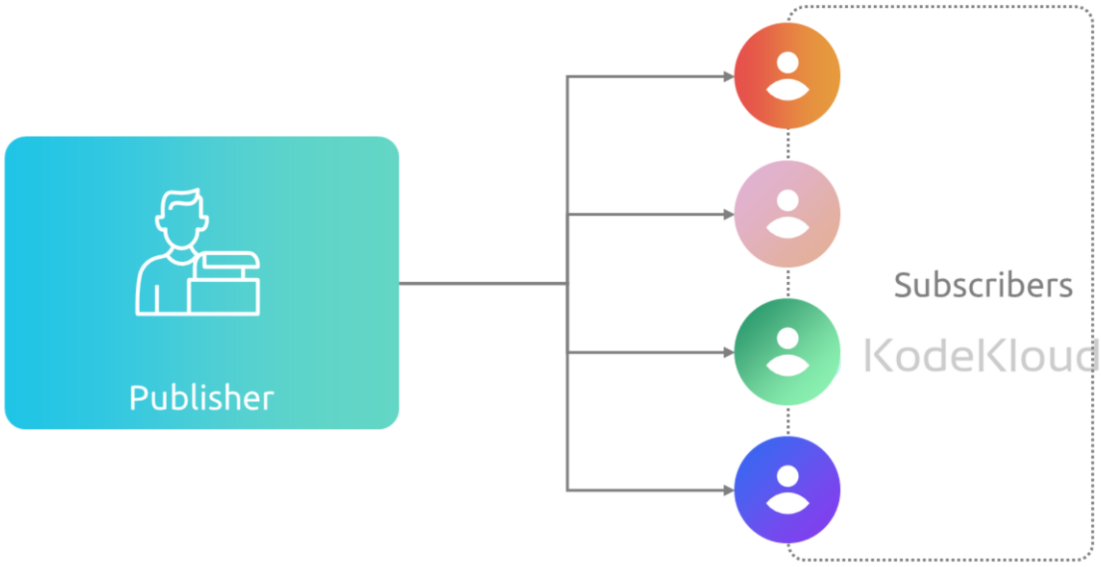




Peek Lock



Topics and Subscriptions

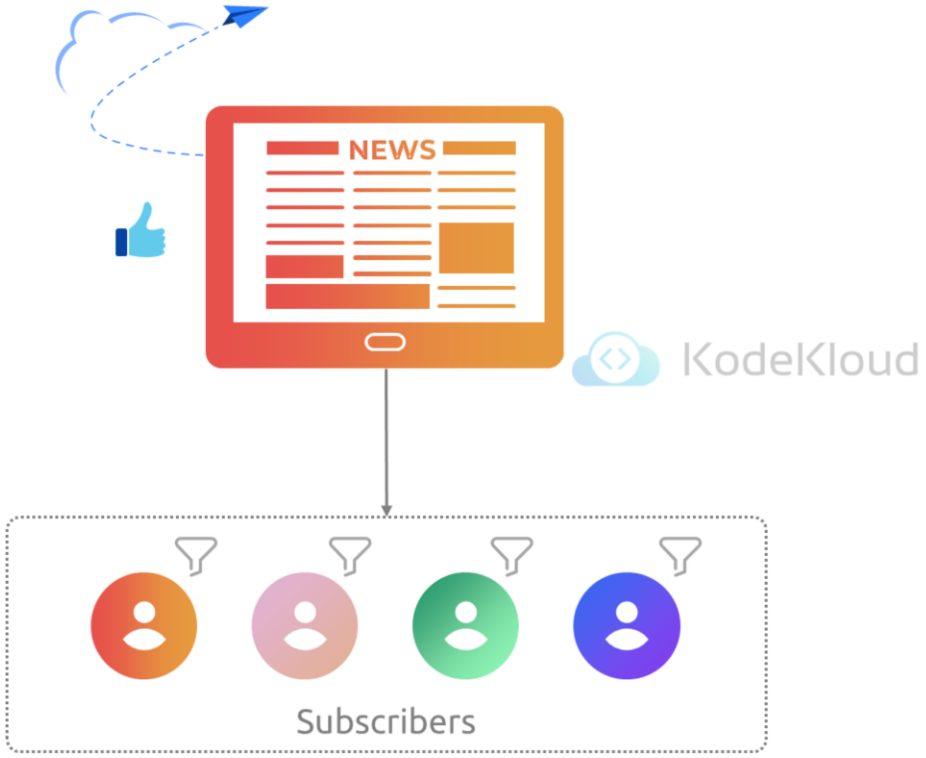


Provides a one-to-many form of communication in a publish-and-subscribe pattern

Topics:
Think of Topics as a broadcast station. When a message is sent to a Topic, it's like broadcasting a message that can be received by multiple listeners (subscribers). This is useful for scenarios where you want to send the same message to multiple recipients.
Example Scenario: Imagine a news broadcasting system. A news update (message) is sent to a Topic, and multiple users (subscribers) receive the update. This allows the same message to reach various endpoints efficiently.
Subscriptions:

Subscriptions are the individual listeners or receivers of the messages sent to a Topic. Each subscription can apply filters to receive only specific messages of interest. This ensures that each subscriber gets only the relevant messages, even if the Topic receives a broad range of messages.
Example Scenario: Consider a subscription-based news service. Users can subscribe to different topics like sports, technology, or politics. Each subscriber receives updates only for their chosen topics.

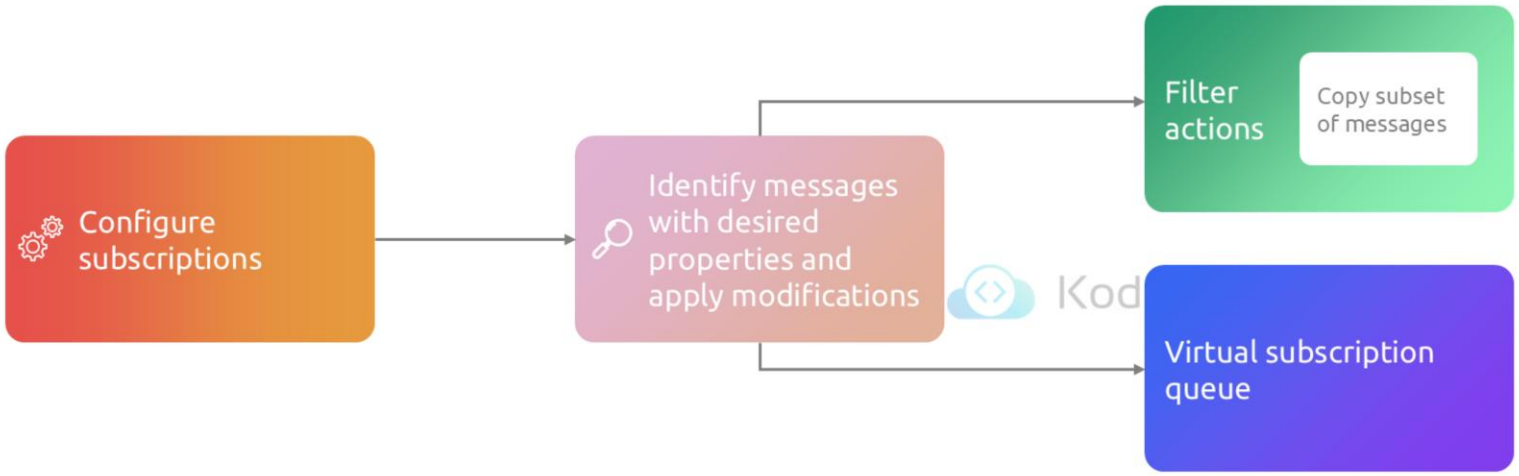
Topics and Subscriptions



Topics:
Think of Topics as a broadcast station. When a message is sent to a Topic, it's like broadcasting a message that can be received by multiple listeners (subscribers). This is useful for scenarios where you want to send the same message to multiple recipients.
Example Scenario: Imagine a news broadcasting system. A news update (message) is sent to a Topic, and multiple users (subscribers) receive the update. This allows the same message to reach various endpoints efficiently.
Subscriptions:

Subscriptions are the individual listeners or receivers of the messages sent to a Topic. Each subscription can apply filters to receive only specific messages of interest. This ensures that each subscriber gets only the relevant messages, even if the Topic receives a broad range of messages.
Example Scenario: Consider a subscription-based news service. Users can subscribe to different topics like sports, technology, or politics. Each subscriber receives updates only for their chosen topics.

Rules and Actions

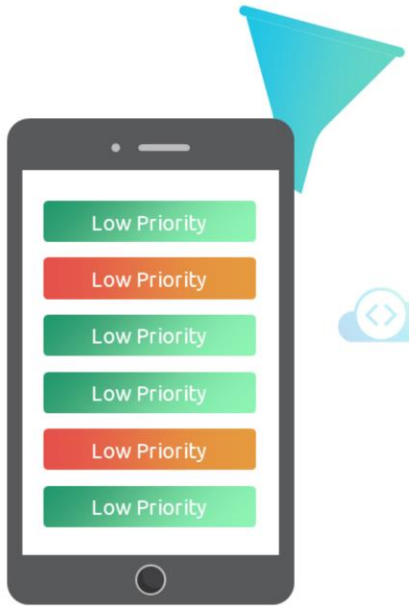


Configure subscriptions to identify messages with desired properties and apply modifications; use filter actions to copy a subset to the virtual subscription queue.

Configuring Subscriptions:
You can configure your subscriptions to identify messages with specific properties and apply necessary modifications. This allows you to ensure that only relevant messages are delivered to each subscription.
Example Scenario: Imagine you have a Topic that sends out various types of notifications. By configuring subscriptions with specific rules, you can ensure that a particular subscriber only receives error notifications, filtering out all other types.
Filter Actions:
Filter actions allow you to perform operations on the message, such as copying a subset of messages to a virtual

subscription queue. This is useful for creating a customized view of messages for different subscribers based on certain criteria.
Example Scenario: Consider an e-commerce system where you have different types of orders. You can configure a filter action to copy only high-priority orders to a specific subscription queue, ensuring they are processed first.

Rules and Actions



KodeKloud

Configuring Subscriptions:
You can configure your subscriptions to identify messages with specific properties and apply necessary modifications. This allows you to ensure that only relevant messages are delivered to each subscription.
Example Scenario: Imagine you have a Topic that sends out various types of notifications. By configuring subscriptions with specific rules, you can ensure that a particular subscriber only receives error notifications, filtering out all other types.
Filter Actions:
Filter actions allow you to perform operations on the message, such as copying a subset of messages to a virtual

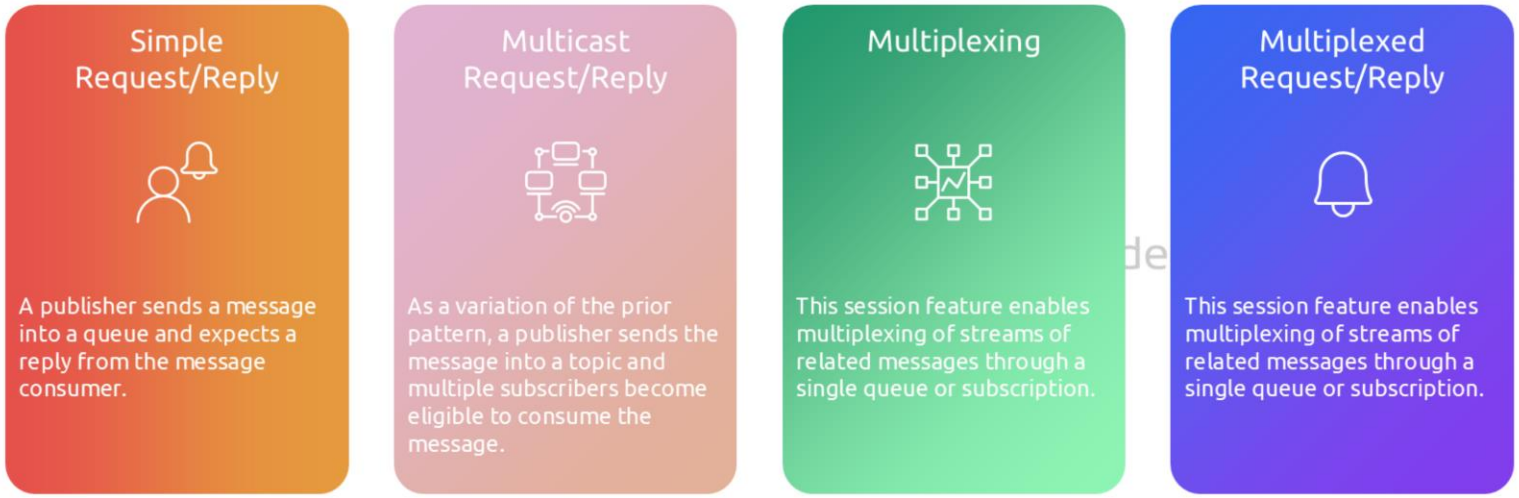
subscription queue. This is useful for creating a customized view of messages for different subscribers based on certain criteria.
Example Scenario: Consider an e-commerce system where you have different types of orders. You can configure a filter action to copy only high-priority orders to a specific subscription queue, ensuring they are processed first.

Exploring Service Bus Message Payloads and Serialization

Exploring Service Bus Message Payloads and Serialization

Exploring service best message payloads and serialization In this section we are diving deeper into two crucial aspects of messaging in Azure service best message routing and correlation patterns, as well as payload serialization Understanding these will help you design effective messaging systems that handle large volumes of data seamlessly

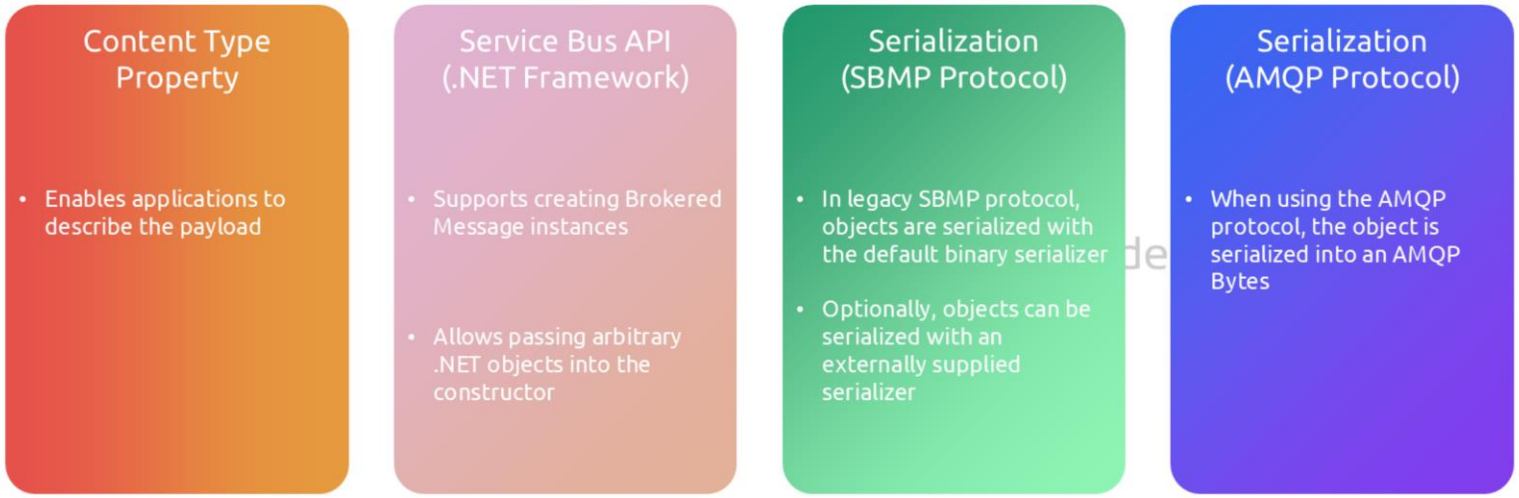
Message Routing and Correlation



Messages carry a payload and metadata. The metadata is in the form of key-value pair properties, and describes the payload.

A Service Bus message consists of a binary payload section that Service Bus never handles in any form on the service-side, and two sets of properties.

Payload Serialization



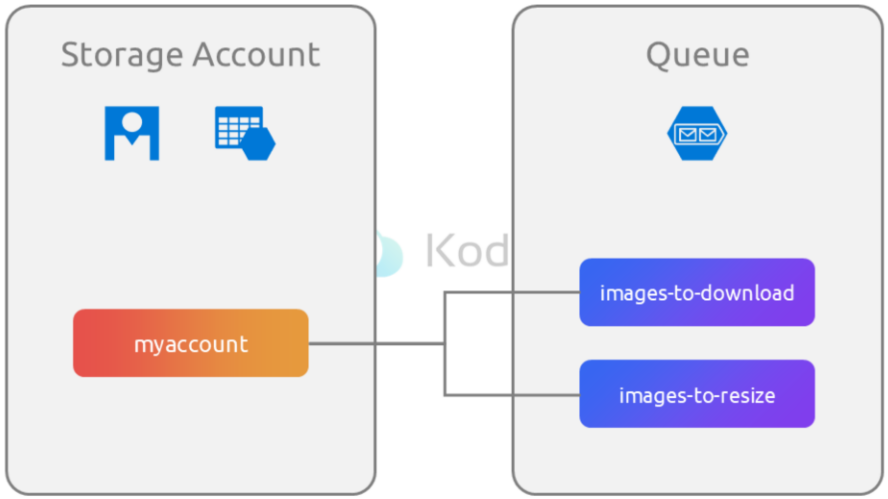
Messages carry a payload and metadata. The metadata is in the form of key-value pair properties, and describes the payload.

A Service Bus message consists of a binary payload section that Service Bus never handles in any form on the service-side, and two sets of properties.

Exploring Azure Queue Storage

Azure Queue Storage

- A service for storing large numbers of messages
- A queue message can be up to 64 KB in size
- Queue service contains the following components:
  - URL format
  - Storage
  - Queue
  - Message



The Queue service contains the following components:
URL format: Queues are addressable using the URL format https://<storage account>.queue.core.windows.net/<queue>. For example, the following URL addresses a queue in the diagram above https://myaccount.queue.core.windows.net/images-to-download
Storage account: All access to Azure Storage is done through a storage account.
Queue: A queue contains a set of messages. All messages must be in a queue. Note that the queue name must be all lowercase.

Message: A message, in any format, of up to 64 KB. For version 2017-07-29 or later, the maximum time-to-live can be any positive number, or -1 indicating that the message doesn't expire. If this parameter is omitted, the default time-to-live is seven days.

Creating and Managing Azure Queue Storage Queues and Messages by Using .NET

NuGet Packages

Azure.Core Library for .NET

- Provides shared primitives, abstractions, and helpers for modern .NET Azure SDK client libraries

Azure.Storage.Common Client Library for .NET

- Provides infrastructure shared by other Azure Storage client libraries

Azure.Storage.Queues Client Library for .NET

- Enables working with Azure Queue Storage for storing messages that may be accessed by a client

System.Configuration.ConfigurationManager Library for .NET

- Provides access to configuration files for client applications

Examples

Example

```
// The QueueClient class enables you to retrieve queues stored in Queue storage.

QueueClient queueClient = new QueueClient(connectionString, queueName);
```


There are more examples in the student and trainer handbooks.

Examples

Example

```
// This example shows how to create a queue if it does not already exist

// Get the connection string from app settings
string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

// Instantiate a QueueClient which will be used to create and manipulate the queue
QueueClient queueClient = new QueueClient(connectionString, queueName);

// Create the queue
queueClient.CreateIfNotExists();
```


There are more examples in the student and trainer handbooks.

Examples

Example

```
// To insert a message into an existing queue, call the SendMessage method.

// Get the connection string from app settings
string connectionString = ConfigurationManager.AppSettings["StorageConnectionString"];

// Instantiate a QueueClient which will be used to create and manipulate the queue
QueueClient queueClient = new QueueClient(connectionString, queueName);

// Create the queue if it doesn't already exist
queueClient.CreateIfNotExists();

if (queueClient.Exists())
{
    // Send a message to the queue
    queueClient.SendMessage(message);
}
```


There are more examples in the student and trainer handbooks.

KodeKloud

Visit www.kodekloud.com to learn more.