

Module 11: Develop message-based solutions



Topics

- Azure Service Bus
- Azure Queue Storage

Lesson 01: Azure Service Bus



Comparing cloud messaging options

| Requirement | Simple queuing | Eventing and PubSub | Big data streaming | Enterprise messaging |
|----------------------|--|--|--|--|
| Product | Queue storage | Event Grid | Event Hubs | Service Bus |
| Supported advantages | <ul style="list-style-type: none"> • Communication within an app • Individual message • Queue semantics / polling buffer • Simple and easy to use • Pay as you go | <ul style="list-style-type: none"> • Communication between apps / orgs • Individual message • Push semantics • Filtering and routing • Pay as you go • Fan out | <ul style="list-style-type: none"> • Many messages in a Stream (think in MBs) • Ease of use and operation • Low cost • Fan in • Strict ordering • Works with other tools | <ul style="list-style-type: none"> • Instantaneous consistency • Strict ordering • Java Messaging Service • Non-repudiation and security • Geo-replication and availability • Rich features (such as deduplication and scheduling) |
| Weaknesses | <ul style="list-style-type: none"> • Ordering of messaging • Instantaneous consistency | <ul style="list-style-type: none"> • Ordering of messaging • Instantaneous consistency | <ul style="list-style-type: none"> • Server-side cursor • Only once | <ul style="list-style-type: none"> • Cost • Simplicity |
| Type | Serverless | Serverless | Big data | Enterprise |

Azure Service Bus

- Enables your applications to interact in several different ways
- Uses a namespace as a scoping container for all messaging components
- The three communication mechanisms are:
 - Queues
 - Topics
 - Relays

Events vs. messaging services

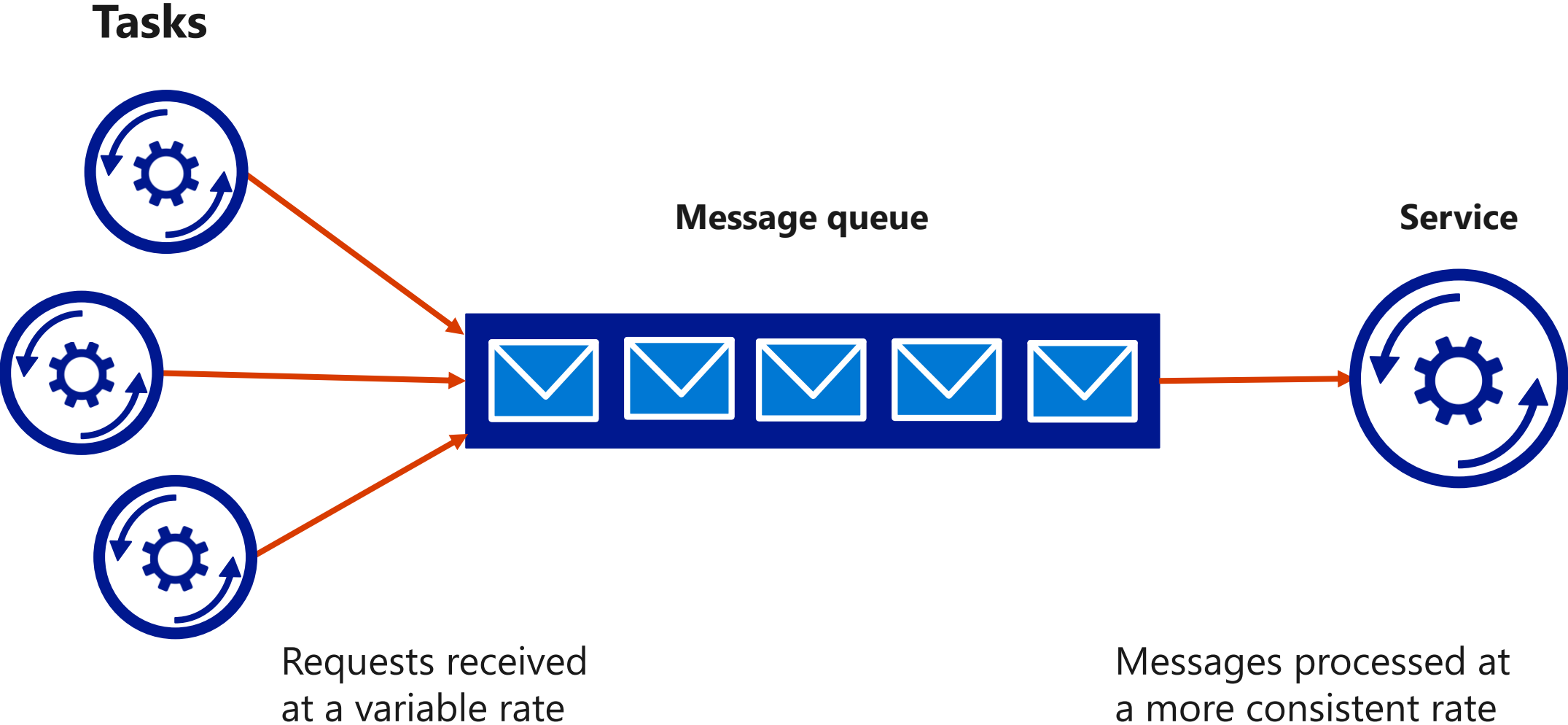
| Service | Purpose | Type | When to use |
|-------------|---------------------------------|-------------------------------|---|
| Event Grid | Reactive programming | Event distribution (discrete) | React to status changes |
| Event Hubs | Big data pipeline | Event streaming (series) | Telemetry and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

Queues

- Messages are sent to and received from queues
- Enables you to store messages until the receiving application is available to receive and process them
- Supports a brokered messaging communication model
- A general-purpose technology that can be used for a wide variety of scenarios

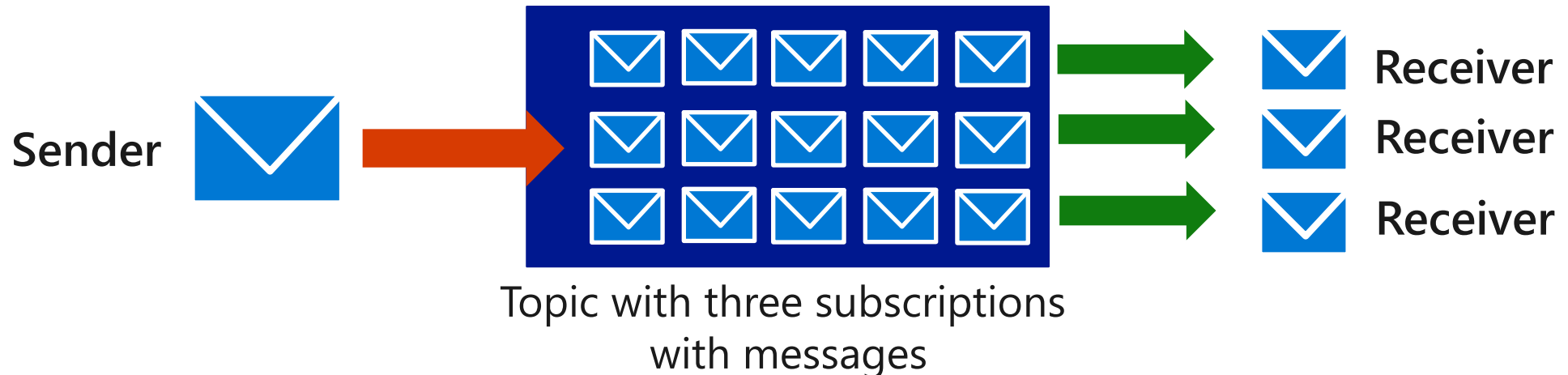


Queue-based load leveling



Topics and subscriptions

- Implements publish/subscribe (pub-sub) model
 - Receivers subscribe to a topic, and they can even filter down by interest
 - A sender publishes messages to the topic
 - Asynchronously, receivers get their own copy of the message
- Subscriptions are independent, which allows for many independent "taps" into a message stream



Messages, payloads, and serialization

- Messages carry multiple things
 - Metadata about the message itself (in key-value pairs)
 - Predefined Broker properties
 - The message binary payload
- Message payload is not visible to Service Bus at any point
 - Serializes as opaque, binary content
 - Can be deserialized by using client SDK libraries
 - Gives you the flexibility to explicitly define how you want to serialize content

Demonstration: Using .NET to send and receive messages from a Service Bus queue



Lesson 02: Azure Queue Storage



Azure Queue storage

- Service for storing messages in an Azure Storage account
 - Accessed using HTTP or HTTPS
 - Scalable to millions of messages
- Common uses of Queue storage include:
 - Creating a backlog of work to process asynchronously
 - Passing messages from an Azure web role to an Azure worker role

Components

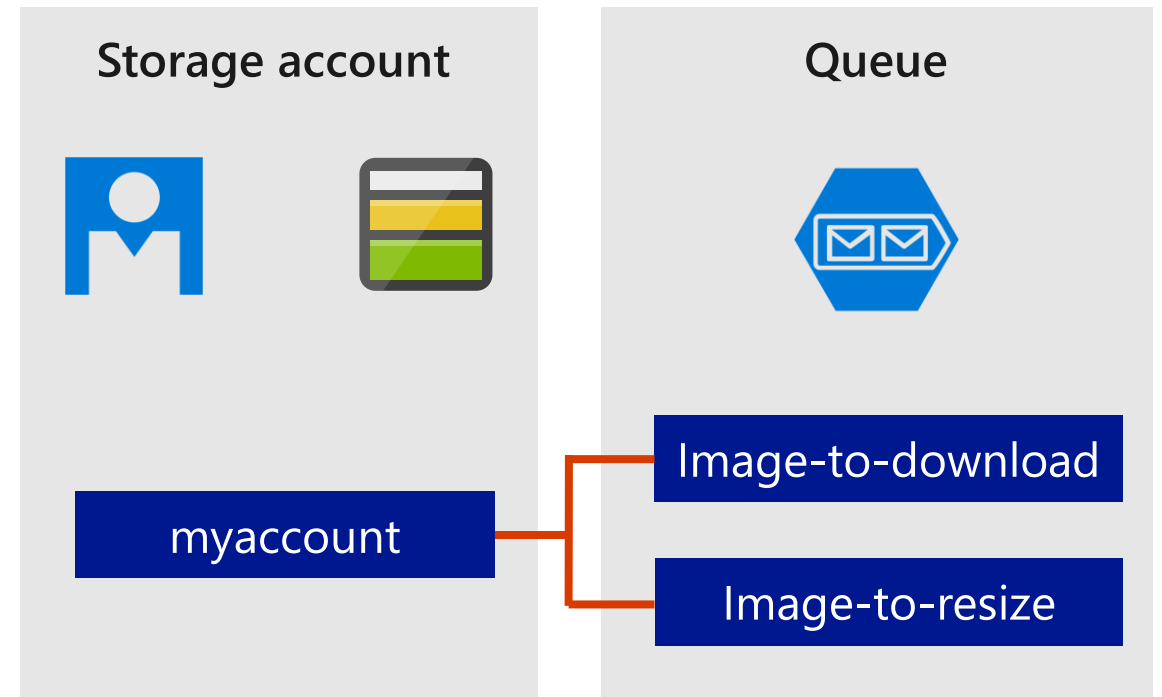
- URL format

`https://<storage-account>.queue.core.windows.net/<queue>`

- Storage account

- Queue

- Message



Queue service components

Code examples

```
// connection string in application's configuration file
<add key="StorageConnectionString"
value="DefaultEndpointsProtocol=https;AccountName=storagesample;AccountKey=GMuzNHj1B3S9
itqZJHHCnRkrokLkcSyW7yK9BRbGp0ENePunLPwBgpxV1Z/pVo9zpem/2xSHXkMqTHHLcx8XRA==" />
```

```
// create instance of CloudStorageAccount class
CloudStorageAccount account = CloudStorageAccount.Parse("StorageConnectionString");
```

```
// create queue client
CloudQueueClient queueClient = account.CreateCloudQueueClient();
```

```
// retrieve reference to queue
CloudQueue queue = queueClient.GetQueueReference("myqueue");
```

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



Code examples – create and get messages

```
// Create a message and add it to the queue.  
CloudQueueMessage message = new CloudQueueMessage("Hello, World");  
queue.AddMessage(message);  
  
// Peek at the next message  
CloudQueueMessage peekedMessage = queue.PeekMessage();  
  
// Fetch the queue attributes.  
queue.FetchAttributes();  
  
// Retrieve the cached approximate message count.  
int? cachedMessageCount = queue.ApproximateMessageCount;
```



Code examples – retrieve and change message

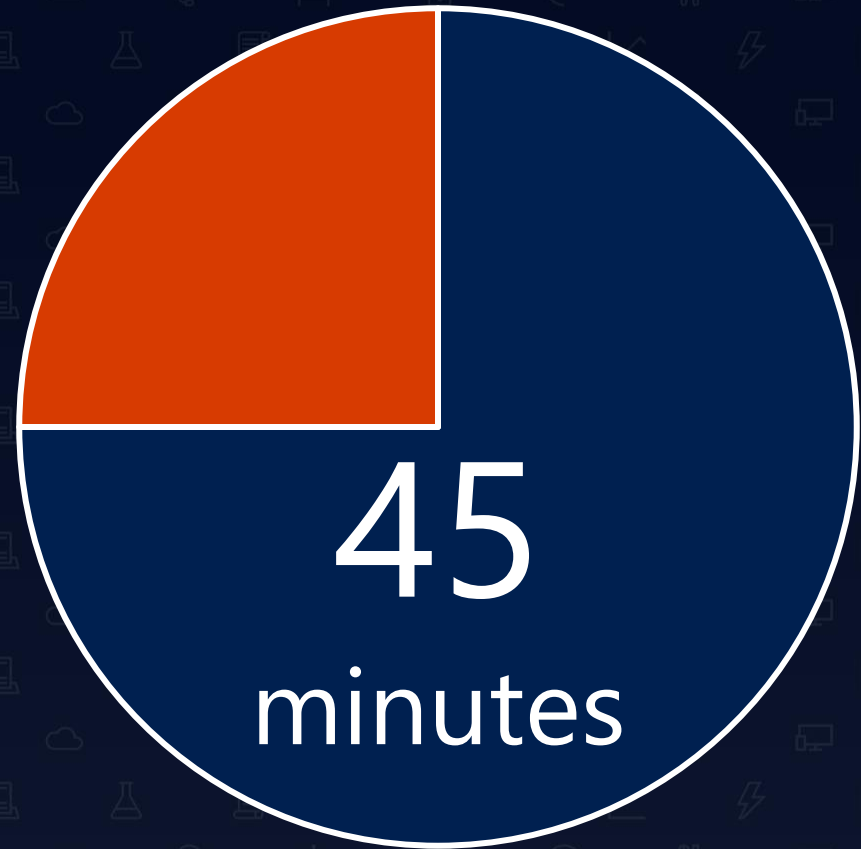
```
// Get the next message
CloudQueueMessage retrievedMessage = queue.GetMessage();

//Process the message in less than 30 seconds, and then delete the message
queue.DeleteMessage(retrievedMessage);

// Get the message from the queue and update the message contents.
CloudQueueMessage message = queue.GetMessage();
message.SetMessageContent("Updated contents.");
queue.UpdateMessage(
    message,
    TimeSpan.FromSeconds(60.0), // Make it invisible for another 60 seconds.
    MessageUpdateFields.Content | MessageUpdateFields.Visibility
);
```



Lab: Asynchronously processing messages by using Azure Storage queues



Lab: Asynchronously processing messages by using Azure Storage queues

Duration



Lab sign-in information

