Choose a messaging platform

8 minutes

There are many communications platforms that can help improve the reliability of a distributed application, including several within Azure. Each of these tools serves a different purpose; let's review each tool in Azure to help choose the right one.

The architecture of our pizza ordering and tracking application requires several components: a website, data storage, back-end service, etc. We can bind the components of our application together in many different ways, and a single application can take advantage of multiple techniques.

We need to decide which techniques to use in the Contoso Slices application. The first step is to evaluate each place where there is communication between multiple parts. Some components *must* run in a timely manner for our application to be doing its job at all. Some may be important, but not time-critical. Finally, other components, like our mobile app notifications, are a bit more optional.

Here, you will learn about the communications platforms available in Azure so that you can choose the right one for each requirement in your application.

Decide between messages and events

Messages and events are both **datagrams**: packages of data sent from one component to another. They are different in ways that at first seem subtle, but can make significant differences in how you architect your application.

Messages

In the terminology of distributed applications, the defining characteristic of a message is that the overall integrity of the application may rely on messages being received. You can think of sending a message as one component passing the baton of a workflow to a different component. The entire workflow may be a vital business process, and the message is the mortar that holds the components together.

A message generally contains the data itself, not just a reference (like an ID or URL) to data. Sending the data as part of the datagram is less brittle than sending a reference. The messaging architecture guarantees delivery of the message, and because no additional lookups are required, the message is reliably handled. However, the sending application needs

to know exactly what data to include to avoid sending too much data, which requires the receiving component to do unnecessary work. In this sense, the sender and receiver of a message are often coupled by a strict data contract.

In Contoso Slices' new architecture, when a pizza order is entered, the company would likely use messages. The 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.

Events

An event triggers a notification that something has occurred. Events are "lighter" than messages and are most often used for broadcast communications.

Events have the following characteristics:

- The event may be sent to multiple receivers, or to none at all
- Events are often intended to "fan out," or have a large number of subscribers for each publisher
- The publisher of the event has no expectation about the action a receiving component takes

Our pizza chain would likely use events for notifications to users about status changes. 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.

This difference between events and messages is fundamental because communications platforms are generally designed to handle one or the other. Service Bus is designed to handle messages. If you want to send events, you would likely choose Event Grid.

Azure also has Azure Event Hubs, but it is most often used for a specific type of high-flow stream of communications used for analytics. For example, if we had networked sensors on our pizza ovens, we could use Event Hubs coupled with Azure Stream Analytics to watch for patterns in the temperature changes that might indicate an unwanted fire or component wear.

Service Bus topics, queues, and relays

Azure Service Bus can exchange messages in three different ways: queues, topics, and relays.

What is a queue?

A **queue** is a simple temporary storage location for messages. A sending component adds a message to the queue. A destination component picks up the message at the front of the

queue. Under ordinary circumstances, each message is received by only one receiver.

Queues decouple the source and destination components to insulate destination components from high demand.

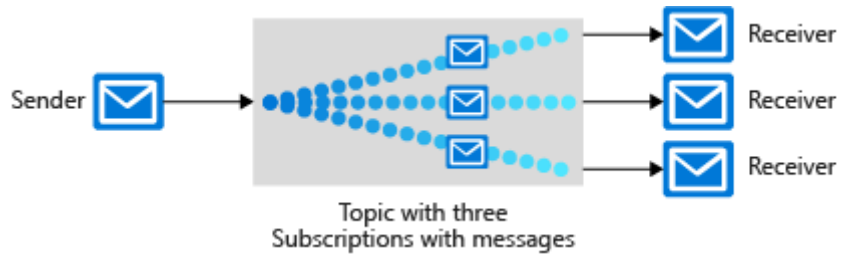
During peak times, messages may come in faster than destination components can handle them. Because source components have no direct connection to the destination, the source is unaffected and the queue will grow. Destination components will remove messages from the queue as they are able to handle them. When demand drops, destination components can catch up and the queue shortens.

A queue responds to high demand like this without needing to add resources to the system. However, for messages that need to be handled relatively quickly, adding additional instances of your destination component can allow them to share the load. Each message would be handled by only one instance. This is an effective way to scale your entire application while only adding resources to the components that actually need it.

What is a topic?

A **topic** is similar to a queue but can have multiple subscriptions. This means that multiple destination components can subscribe to a single topic, so each message is delivered to multiple receivers. Subscriptions can also filter the messages in the topic to receive only messages that are relevant. Subscriptions provide the same decoupled communications as queues and respond to high demand in the same way. Use a topic if you want each message to be delivered to more than one destination component.

Topics are not supported in the Basic pricing tier.



What is a relay?

A **relay** is an object that performs synchronous, two-way communication between applications. Unlike queues and topics, it is not a temporary storage location for messages. Instead, it provides bidirectional, unbuffered connections across network boundaries such as

firewalls. Use a relay when you want direct communications between components as if they were located on the same network segment but separated by network security devices.

> ⓘ **Note**
>
> Although relays are part of Azure Service Bus, they do not implement loosely coupled messaging workflows and are not considered further in this module.

Service Bus queues and storage queues

There are two Azure features that include message queues: Service Bus and Azure Storage accounts. As a general guide, storage queues are simpler to use but are less sophisticated and flexible than Service Bus queues.

Key advantages of Service Bus queues include:

- Supports larger messages sizees of 256 KB (standard tier) or 1MB (premium tier) per message versus 64 KB
- Supports both at-least-once and at-most-once delivery - choose between a very small chance that a message is lost or a very small chance it is handled twice
- Guarantees **first-in-first-out (FIFO)** order - messages are handled in the same order they are added (although FIFO is the normal operation of a queue, it is not guaranteed for every message)
- Can group multiple messages into a transaction - if one message in the transaction fails to be delivered, all messages in the transaction will not be delivered
- Supports role-based security
- Does not require destination components to continuously poll the queue

Advantages of storage queues:

- Supports unlimited queue size (versus 80-GB limit for Service Bus queues)
- Maintains a log of all messages

How to choose a communications technology

We've seen the different concepts and the implementations Azure provides. Let's discuss what our decision process should look like for each of our communications.

Consider the following questions:

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.

If you decide that you need a queue:

Choose Service Bus queues if:

- You need an at-most-once delivery guarantee
- You need a FIFO guarantee
- You need to group messages into transactions
- You want to receive messages without polling the queue
- You need to provide role-based access to the queues
- You need to handle messages larger than 64 KB but smaller than 256 KB
- Your queue size will not grow larger than 80 GB
- You would like to be able to publish and consume batches of messages

Choose queue storage if:

- You need a simple queue with no particular additional requirements
- You need an audit trail of all messages that pass through the queue
- You expect the queue to exceed 80 GB in size
- You want to track progress for processing a message inside of the queue

Although the components of a distributed application can communicate directly, you can often increase the reliability of that communication by using an intermediate communication platform such as Azure Service Bus or Azure Event Grid.

Event Grid is designed for events, which notify recipients only of an event and do not contain the raw data associated with that event. Azure Event Hubs is designed for high-flow analytics types of events. Azure Service Bus and storage queues are for messages, which can be used for binding the core pieces of any application workflow.

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 may be the best option. Otherwise, Service Bus queues provide many more options and flexibility.

If you want to send messages to multiple subscribers, use a Service Bus topic.

Next unit: Exercise - Implement a Service Bus topic and queue

Continue >