# Azure Service Bus

You have identified the following scenarios for message exchange between the mobile app and the web service:

1. Messages that relate to individual sales must be sent to the web service instance in the user's region.
2. Messages that relate to sales performance must be sent to all instances of the web service.

You have decided to implement a Service Bus queue for the first use case and a Service Bus topic for the second use case.

## Create a Service Bus namespace

Azure portal -> Create a resource -> Service Bus -> Create->

| **Setting** | **Value** | **Description** |
| --- | --- | --- |
| Subscription | Concierge subscription | The subscription in which this new app is created. |
| Resource group | [Sandbox resource group name] | The name of the resource group in which to create your Service Bus namespace. |
| Namespace name | [Globally unique name] | Enter a name that is unique in Azure. If you want to use the format *salesteamapp*<*Company*><*year*>, your namespace name would look like the example *salesteamappContoso2022*. |
| Location | Select from the dropdown | Choose from the free *sandbox regions* listed after this table. |
| Pricing tier | Standard | The recommended pricing tier for this exercise. |

-> Review + create -> Create -> Go to resource.

### Get Connection String for the storage account by following steps:

-> Service Bus namespace -> Left menu -> Shared access policies

-> RootManageSharedAccessKey -> Primary connection string -> Copy to clipboard.

## Create a Service Bus queue

Service Bus Namespace -> Left menu Entities -> Queues -> [+ Queue] button

-> Create queue pane -> “Name” = salesmessages -> Create.

## Create a Service Bus topic

Service Bus Namespace -> Left menu Entities -> Topics -> [+ Topic] button

-> Create topic pane

-> “Name” = salesperformancemessages

“Enable partitioning” = checked -> select Create.

## Create subscriptions for Service Bus topic

Salesperformancemessages -> [+ Subscription] button -> Create subscription pane

-> “Name” = Americas, “Max delivery count” = 100 -> Create.

Salesperformancemessages -> [+ Subscription] button -> Create subscription pane

-> “Name” = EuropeAndAsia, “Max delivery count” = 100 -> Create.

## Write code to send and receive messages by using a queue / a topic

### NuGet package required:

**Azure.Messaging.ServiceBus**

[Find the code here](https://github.com/AjinkyaApte88/General/tree/main/mslearn-service-bus/implement-message-workflows-with-service-bus/src/start).

**Service Bus queues and topics are excellent tools you can use to increase the resilience of communications within a distributed application. By acting as temporary storage locations, Service Bus queues and topics remove the requirement for direct communication between components, and they smoothly handle peaks in demand.**

**Consider using Service Bus queues and topics when you have a component that can communicate with another component in a loosely coupled configuration.**

# Azure Queue storage

Direct communication between the components of a distributed application can be problematic because it might be disrupted when network bandwidth is low or when demand is high.

We've seen this in our system: the web portal calls a web service, which works great if the service responds in a timely manner. High traffic causes problems and so the plan is to use a queue to eliminate the direct link between the front-end apps and your middle-tier web service.

## Create a storage account

| **Parameter** | **Value** |
| --- | --- |
| Name | Sets the name. Remember that storage accounts use the name to generate a public URL - so it must be unique. In addition, the account name must be between 3 and 24 characters, and be composed of numbers and lowercase letters only. We recommend you use the prefix **articles** with a random number suffix, but you can use whatever you like. |
| Resource group | Supplies the **Resource Group**. Use *[sandbox resource group name]* as the value. |
| Account Type | Sets the **Storage Account type**: *StorageV2* to create a general-purpose V2.account. |
| Redundancy | Sets the **Replication and Storage type**. It defaults to *Standard\_RAGRS (Global Redundancy)*.  Let's use *Standard\_LRS* (locally redundant within the datacenter). |
| location | Sets the **Location** independent of the resource group owner. It's optional, but you can use it to place the queue in a different region than the resource group. Place it close to you. |

-> Review + create -> Create -> Go to resource.

### Get Connection String for the storage account by following steps:

Azure storage account -> Left menu -> Access keys -> Connection String -> Show

-> Copy to clipboard

## Programmatically create and access a queue

### NuGet package required:

**Azure.Storage.Queues**

[Find the code here](https://github.com/AjinkyaApte88/General/tree/main/mslearn-storage-queues/start).

**Using the Azure.Storage.Queues package for .NET can help to make a distributed application more reliable and resilient to failures and periods of high demand.**

# Azure Event Hubs

Big data apps must be able to process increased throughput by scaling out to meet increased transaction volumes.

## Azure Event Hubs

### What is Azure Event Hubs?

Azure Event Hubs is a cloud-based, event-processing service that can receive and process millions of events per second. Event Hubs acts as a front door for an event pipeline, to receive incoming data and stores this data until processing resources are available.

### Events

An event is a small packet of information (a datagram) that contains a notification.

Events can be published individually or in batches, but a **single publication (individual or batch) cannot exceed 1 MB.**

### Publishers and subscribers

Event publishers are any app or device that can send out events using either HTTPS, Advanced Message Queuing Protocol (AMQP) 1.0, or Apache Kafka.

1. **For publishers that send data frequently**, **AMQP** has better performance. However, it has a higher initial session overhead, because a persistent bidirectional socket and transport-level security (TLS), or SSL/TLS, has to be set up first.
2. **For more intermittent publishing**, **HTTPS** is the better option. Though HTTPS requires more overhead for each request, there's no session initialization overhead.
3. Event Hubs provides an **endpoint compatible with the Apache Kafka producer and consumer APIs** that can be **used by most existing Apache Kafka client applications** as an **alternative to running your own Apache Kafka cluster**.

Event Hubs supports Apache Kafka's producer and consumer APIs clients at version 1.0 and above. For more information, see Event Hubs for Apache Kafka.

**Event subscribers are apps that use one of two supported programmatic methods to receive and process events from an event hub.**

* **EventHubReceiver** - A simple method that provides limited management options.
* **EventProcessorHost** - An efficient method that we'll use later in this module.

## Create an Event Hub using the Azure CLI

### Create an Event Hubs namespace

az configure --defaults group=[sandbox Resource Group] location=westus2

NS\_NAME=ehubns-$RANDOM

az eventhubs namespace create --name $NS\_NAME

### Fetch the connection string and primary key for your namespace

az eventhubs namespace authorization-rule keys list \

--name RootManageSharedAccessKey \

--namespace-name $NS\_NAME

The return output is in JSON:

* primaryConnectionString
* primaryKey

### Create an event hub

**(default - 4 partitions)**

HUB\_NAME=hubname-$RANDOM

az eventhubs eventhub create --name $HUB\_NAME --namespace-name $NS\_NAME

az eventhubs eventhub show --namespace-name $NS\_NAME --name $HUB\_NAME

## View the event hub in the Azure portal

Azure portal -> Event Hubs -> your namespace -> Left menu > Entities -> Event Hubs

-> your event hub.

## Send or receive messages through an Event Hub

NuGet Package:

**Azure.Messaging.EventHubs**

## Evaluate the performance of the deployed Event Hub using the Azure portal

Azure portal -> your Event Hub -> Overview page -> view the message counts.

## Test Event Hub resilience

1. Send messages to the Event Hub using EventHubProducer.
2. Set “EVENT HUB STATUS”, as Disabled.
3. Wait for a minimum of five minutes.
4. Set “EVENT HUB STATUS”, as Active.
5. Run the EventProcessor.
6. -> Event Hub Namespace -> Left menu pane -> Monitoring -> Metrics.
7. -> Metric dropdown list -> Incoming Messages -> in the taskbar -> Add metric.
8. -> select Last 24 hours (Automatic) -> Last 30 minutes -> Apply.

**You will see that though the messages were sent before the Event Hub was taken offline for a period, all 100 messages were successfully transmitted.**

# Event Grid : React to state changes in your Azure services

In a complex cloud environment, you might need to respond to events from many different sources automatically and manually.

Suppose you work on an operations team for a large healthcare organization. You have a system with virtual machines under strict regulatory and change control. You want to be notified by email of any changes to these virtual machines' configuration in Azure.

To accomplish this notification, you'll use Azure Event Grid to receive virtual-machine events and Azure Logic Apps to send emails that alert your team of any changes.

### What is Event Grid?

Event Grid aggregates all your events and provides routing from any source to any destination.

### Capabilities

It's native to Azure, with the ability to be extended and customized. Some of the main advantages are:

* **It's simple.** Point and click in the Azure portal to add and collect your events from Azure resources.
* **It can filter events.** Thus, handlers receive only relevant events.
* **It supports multiple subscribers.** Attach multiple handlers to a single event from a single source.
* **It's reliable.** Take advantage of 24-hour retries to ensure events are delivered.
* **Its throughput is high.** Handle a high volume of events, in the range of millions per second.
* **It has built-in events.** Use built-in events to get started quickly and easily.
* **It supports custom events.** Use Event Grid to reliably deliver events for your custom components.

### Event sources

Azure offers many event sources or publishers. For example, Azure Storage is the event source for blob-created events.

Sources can be configured from anywhere, and include on-premises custom applications or virtual machines within your Azure account. A source allows a single mechanism for event management through all your systems, whether they're in an on-premises datacenter or with other cloud providers.

### Event handlers

There are event handlers for many services in Azure, and more are being added all the time. You can also use a webhook handler to call a custom endpoint outside Azure.

Some examples of event handlers within Azure are:

* Azure Functions
* Azure Logic Apps
* Azure Automation
* Azure Event Hubs
* Azure Service Bus

### Topics

Topics provide the core mechanism for managing the various events being raised throughout your system's lifecycle.

You can create as many topics as you need in Event Grid to provide the granularity you need to manage events in your system.

### Event subscriptions

When the topics have been defined, you can subscribe to them. Subscriptions convey which events on a topic you're interested in receiving. You can then filter these events by type or subject.

### Configure connectors and triggers

After creating an event handler that deals with an event, you can configure the connectors and triggers associated to it.

E.g. –

1. Azure Function - You can use EventGridTrigger.
2. Using the Logic Apps Designer, you can configure the Event Grid connector on the logic app to look for any events of a certain topic.

**Build logic inside the Event Handlers to respond to the event.**

**(Anything like Send Email Notification etc. as per requirement.)**