100 XP

Introduction

3 minutes

Azure Event Grid is deeply integrated with Azure services and can be integrated with third-party services. It simplifies event consumption and lowers costs by eliminating the need for constant polling. Event Grid efficiently and reliably routes events from Azure and non-Azure resources, and distributes the events to registered subscriber endpoints.

After completing this module, you'll be able to:

- Describe how Event Grid operates and how it connects to services and event handlers.
- Explain how Event Grid delivers events and how it handles errors.
- Implement authentication and authorization.
- Route custom events to web endpoint by using Azure CLI.

Next unit: Explore Azure Event Grid

Continue

100 XP

Explore Azure Event Grid

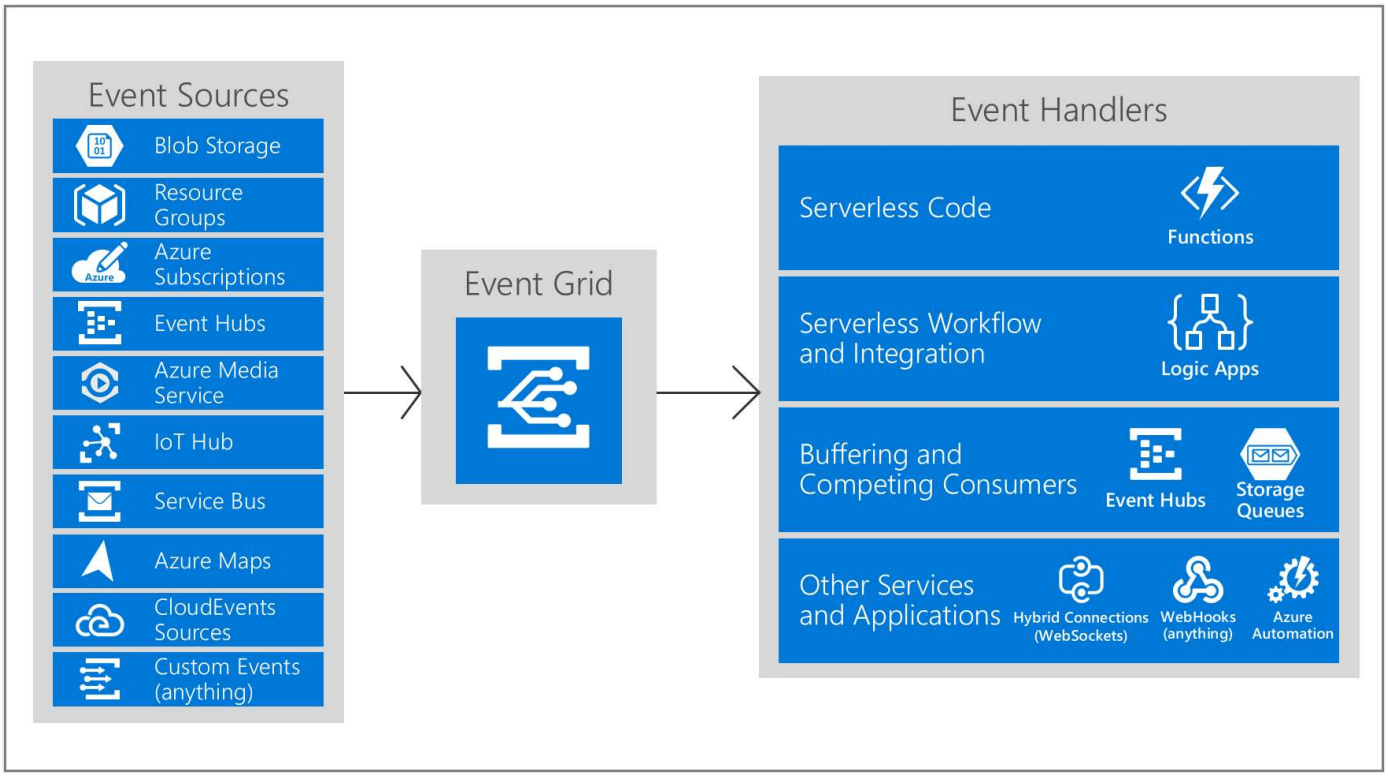
3 minutes

Azure Event Grid is an eventing backplane that enables event-driven, reactive programming. It uses the publish-subscribe model. Publishers emit events, but have no expectation about how the events are handled. Subscribers decide on which events they want to handle.

Event Grid allows you to easily build applications with event-based architectures. Event Grid has built-in support for events coming from Azure services, like storage blobs and resource groups. Event Grid also has support for your own events, using custom topics.

You can use filters to route specific events to different endpoints, multicast to multiple endpoints, and make sure your events are reliably delivered.

This image below shows how Event Grid connects sources and handlers, and isn't a comprehensive list of supported integrations.



Concepts in Azure Event Grid

There are five concepts in Azure Event Grid you need to understand to help you get started, and are described in more detail below:

- **Events -** What happened.
- **Event sources -** Where the event took place.
- **Topics -** The endpoint where publishers send events.
- **Event subscriptions -** The endpoint or built-in mechanism to route events, sometimes to more than one handler. Subscriptions are also used by handlers to intelligently filter incoming events.
- **Event handlers -** The app or service reacting to the event.

Events

An event is the smallest amount of information that fully describes something that happened in the system. Every event has common information like: source of the event, time the event took place, and unique identifier. Every event also has specific information that is only relevant to the specific type of event. For example, an event about a new file being created in Azure Storage has details about the file, such as the `lastTimeModified` value. Or, an Event Hubs event has the URL of the Capture file.

An event of size up to 64 KB is covered by General Availability (GA) Service Level Agreement (SLA). The support for an event of size up to 1 MB is currently in preview. Events over 64 KB are charged in 64-KB increments.

Event sources

An event source is where the event happens. Each event source is related to one or more event types. For example, Azure Storage is the event source for blob created events. IoT Hub is the event source for device created events. Your application is the event source for custom events that you define. Event sources are responsible for sending events to Event Grid.

Topics

The event grid topic provides an endpoint where the source sends events. The publisher creates the event grid topic, and decides whether an event source needs one topic or more than one

topic. A topic is used for a collection of related events. To respond to certain types of events, subscribers decide which topics to subscribe to.

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, but you can subscribe to them. To subscribe, you provide information about the resource you want to receive events from. As long as you have access to the resource, you can subscribe to its events.

Custom topics are application and third-party topics. When you create or are assigned access to a custom topic, you see that custom topic in your subscription.

Event subscriptions

A subscription tells Event Grid which events on a topic you're interested in receiving. When creating the subscription, you provide an endpoint for handling the event. You can filter the events that are sent to the endpoint. You can filter by event type, or subject pattern. Set an expiration for event subscriptions that are only needed for a limited time and you don't want to worry about cleaning up those subscriptions.

Event handlers

From an Event Grid perspective, an event handler is the place where the event is sent. The handler takes some further action to process the event. Event Grid supports several handler types. You can use a supported Azure service or your own webhook as the handler. Depending on the type of handler, Event Grid follows different mechanisms to guarantee the delivery of the event. For HTTP webhook event handlers, the event is retried until the handler returns a status code of `200 - OK`. For Azure Storage Queue, the events are retried until the Queue service successfully processes the message push into the queue.

Next unit: Discover event schemas

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Discover event schemas

3 minutes

Events consist of a set of four required string properties. The properties are common to all events from any publisher. The data object has properties that are specific to each publisher. For system topics, these properties are specific to the resource provider, such as Azure Storage or Azure Event Hubs.

Event sources send events to Azure Event Grid in an array, which can have several event objects. When posting events to an event grid topic, the array can have a total size of up to 1 MB. Each event in the array is limited to 1 MB. If an event or the array is greater than the size limits, you receive the response `413 Payload Too Large`. Operations are charged in 64 KB increments though. So, events over 64 KB will incur operations charges as though they were multiple events. For example, an event that is 130 KB would incur operations as though it were 3 separate events.

Event Grid sends the events to subscribers in an array that has a single event. You can find the JSON schema for the Event Grid event and each Azure publisher's data payload in the Event Schema store .

Event schema

The following example shows the properties that are used by all event publishers:

JSON Copy

```
[
  {
    "topic": string,
    "subject": string,
    "id": string,
    "eventType": string,
    "eventTime": string,
    "data":{
      object-unique-to-each-publisher
    },
    "dataVersion": string,
    "metadataVersion": string
```

```
  }
]
```

Event properties

All events have the same following top-level data:

Property	Type	Required	Description
topic	string	No. If not included, Event Grid will stamp onto the event. If included it must match the event grid topic Azure Resource Manager ID exactly.	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
subject	string	Yes	Publisher-defined path to the event subject.
eventType	string	Yes	One of the registered event types for this event source.
eventTime	string	Yes	The time the event is generated based on the provider's UTC time.
id	string	Yes	Unique identifier for the event.
data	object	No	Event data specific to the resource provider.
dataVersion	string	No. If not included it will be stamped with an empty value.	The schema version of the data object. The publisher defines the schema version.

Property	Type	Required	Description
metadataVersion	string	No. If not included, Event Grid will stamp onto the event. If included, must match the Event Grid Schema `metadataVersion` exactly (currently, only `1`).	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

For custom topics, the event publisher determines the data object. The top-level data should have the same fields as standard resource-defined events.

When publishing events to custom topics, create subjects for your events that make it easy for subscribers to know whether they're interested in the event. Subscribers use the subject to filter and route events. Consider providing the path for where the event happened, so subscribers can filter by segments of that path. The path enables subscribers to narrowly or broadly filter events. For example, if you provide a three segment path like `/A/B/C` in the subject, subscribers can filter by the first segment `/A` to get a broad set of events. Those subscribers get events with subjects like `/A/B/C` or `/A/D/E`. Other subscribers can filter by `/A/B` to get a narrower set of events.

Sometimes your subject needs more detail about what happened. For example, the **Storage Accounts** publisher provides the subject `/blobServices/default/containers/<container-name>/blobs/<file>` when a file is added to a container. A subscriber could filter by the path `/blobServices/default/containers/testcontainer` to get all events for that container but not other containers in the storage account. A subscriber could also filter or route by the suffix `.txt` to only work with text files.

CloudEvents v1.0 schema

In addition to its default event schema, Azure Event Grid natively supports events in the JSON implementation of CloudEvents v1.0 and HTTP protocol binding. CloudEvents is an open specification for describing event data.

CloudEvents simplifies interoperability by providing a common event schema for publishing, and consuming cloud based events. This schema allows for uniform tooling, standard ways of routing & handling events, and universal ways of deserializing the outer event schema. With a common schema, you can more easily integrate work across platforms.

Here is an example of an Azure Blob Storage event in CloudEvents format:

JSON Copy


```
{
    "specversion": "1.0",
    "type": "Microsoft.Storage.BlobCreated",
    "source": "/subscriptions/{subscription-id}/resourceGroups/{resource-
group}/providers/Microsoft.Storage/storageAccounts/{storage-account}",
    "id": "9aeb0fdf-c01e-0131-0922-9eb54906e209",
    "time": "2019-11-18T15:13:39.4589254Z",
    "subject": "blobServices/default/containers/{storage-container}/blobs/{new-
file}",
    "dataschema": "#",
    "data": {
        "api": "PutBlockList",
        "clientRequestId": "4c5dd7fb-2c48-4a27-bb30-5361b5de920a",
        "requestId": "9aeb0fdf-c01e-0131-0922-9eb549000000",
        "eTag": "0x8D76C39E4407333",
        "contentType": "image/png",
        "contentLength": 30699,
        "blobType": "BlockBlob",
        "url": "https://gridtesting.blob.core.windows.net/testcontainer/{new-file}",
        "sequencer": "000000000000000000000000000099240000000000c41c18",
        "storageDiagnostics": {
            "batchId": "681fe319-3006-00a8-0022-9e7cde000000"
        }
    }
}
```


A detailed description of the available fields, their types, and definitions in CloudEvents v1.0 is available here .

The headers values for events delivered in the CloudEvents schema and the Event Grid schema are the same except for `content-type`. For CloudEvents schema, that header value is `"content-type":"application/cloudevents+json; charset=utf-8"`. For Event Grid schema, that header value is `"content-type":"application/json; charset=utf-8"`.

You can use Event Grid for both input and output of events in CloudEvents schema. You can use CloudEvents for system events, like Blob Storage events and IoT Hub events, and custom events. It can also transform those events on the wire back and forth.

Next unit: Explore event delivery durability

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Explore event delivery durability

3 minutes

Event Grid provides durable delivery. It tries to deliver each event at least once for each matching subscription immediately. If a subscriber's endpoint doesn't acknowledge receipt of an event or if there is a failure, Event Grid retries delivery based on a fixed retry schedule and retry policy. By default, Event Grid delivers one event at a time to the subscriber, and the payload is an array with a single event.

> **Note**
>
> Event Grid doesn't guarantee order for event delivery, so subscribers may receive them out of order.

Retry schedule

When Event Grid receives an error for an event delivery attempt, EventGrid decides whether it should retry the delivery, dead-letter the event, or drop the event based on the type of the error.

If the error returned by the subscribed endpoint is a configuration-related error that can't be fixed with retries (for example, if the endpoint is deleted), EventGrid will either perform dead-lettering on the event or drop the event if dead-letter isn't configured.

The following table describes the types of endpoints and errors for which retry doesn't happen:

Endpoint Type	Error codes
Azure Resources	400 Bad Request, 413 Request Entity Too Large, 403 Forbidden

Endpoint Type	Error codes
Webhook	400 Bad Request, 413 Request Entity Too Large, 403 Forbidden, 404 Not Found, 401 Unauthorized

> **Important**
>
> If Dead-Letter isn't configured for an endpoint, events will be dropped when the above errors happen. Consider configuring Dead-Letter if you don't want these kinds of events to be dropped.

If the error returned by the subscribed endpoint isn't among the above list, Event Grid waits 30 seconds for a response after delivering a message. After 30 seconds, if the endpoint hasn't responded, the message is queued for retry. Event Grid uses an exponential backoff retry policy for event delivery.

If the endpoint responds within 3 minutes, Event Grid will attempt to remove the event from the retry queue on a best effort basis but duplicates may still be received. Event Grid adds a small randomization to all retry steps and may opportunistically skip certain retries if an endpoint is consistently unhealthy, down for a long period, or appears to be overwhelmed.

Retry policy

You can customize the retry policy when creating an event subscription by using the following two configurations. An event will be dropped if either of the limits of the retry policy is reached.

- **Maximum number of attempts -** The value must be an integer between 1 and 30. The default value is 30.
- **Event time-to-live (TTL) -** The value must be an integer between 1 and 1440. The default value is 1440 minutes

The example below shows setting the maximum number of attempts to 18 by using the Azure CLI.

Bash Copy

```
az eventgrid event-subscription create \
  -g gridResourceGroup \
  --topic-name <topic_name> \
  --name <event_subscription_name> \
  --endpoint <endpoint_URL> \
  --max-delivery-attempts 18
```

Output batching

You can configure Event Grid to batch events for delivery for improved HTTP performance in high-throughput scenarios. Batching is turned off by default and can be turned on per-subscription via the portal, CLI, PowerShell, or SDKs.

Batched delivery has two settings:

- **Max events per batch** - Maximum number of events Event Grid will deliver per batch. This number will never be exceeded, however fewer events may be delivered if no other events are available at the time of publish. Event Grid doesn't delay events to create a batch if fewer events are available. Must be between 1 and 5,000.
- **Preferred batch size in kilobytes** - Target ceiling for batch size in kilobytes. Similar to max events, the batch size may be smaller if more events aren't available at the time of publish. It's possible that a batch is larger than the preferred batch size *if* a single event is larger than the preferred size. For example, if the preferred size is 4 KB and a 10-KB event is pushed to Event Grid, the 10-KB event will still be delivered in its own batch rather than being dropped.

Delayed delivery

As an endpoint experiences delivery failures, Event Grid will begin to delay the delivery and retry of events to that endpoint. For example, if the first 10 events published to an endpoint fail, Event Grid will assume that the endpoint is experiencing issues and will delay all subsequent retries, and new deliveries, for some time - in some cases up to several hours.

The functional purpose of delayed delivery is to protect unhealthy endpoints and the Event Grid system. Without back-off and delay of delivery to unhealthy endpoints, Event Grid's retry policy and volume capabilities can easily overwhelm a system.

Dead-letter events

When Event Grid can't deliver an event within a certain time period or after trying to deliver the event a certain number of times, it can send the undelivered event to a storage account. This process is known as **dead-lettering**. Event Grid dead-letters an event when **one of the following** conditions is met.

- Event isn't delivered within the **time-to-live** period.
- The **number of tries** to deliver the event exceeds the limit.

If either of the conditions is met, the event is dropped or dead-lettered. By default, Event Grid doesn't turn on dead-lettering. To enable it, you must specify a storage account to hold undelivered events when creating the event subscription. You pull events from this storage account to resolve deliveries.

If Event Grid receives a 400 (Bad Request) or 413 (Request Entity Too Large) response code, it immediately schedules the event for dead-lettering. These response codes indicate delivery of the event will never succeed.

There is a five-minute delay between the last attempt to deliver an event and when it is delivered to the dead-letter location. This delay is intended to reduce the number of Blob storage operations. If the dead-letter location is unavailable for four hours, the event is dropped.

Custom delivery properties

Event subscriptions allow you to set up HTTP headers that are included in delivered events. This capability allows you to set custom headers that are required by a destination. You can set up to 10 headers when creating an event subscription. Each header value shouldn't be greater than 4,096 (4K) bytes. You can set custom headers on the events that are delivered to the following destinations:

- Webhooks
- Azure Service Bus topics and queues
- Azure Event Hubs
- Relay Hybrid Connections

Before setting the dead-letter location, you must have a storage account with a container. You provide the endpoint for this container when creating the event subscription.

Next unit: Control access to events

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Control access to events

3 minutes

Azure Event Grid allows you to control the level of access given to different users to do various management operations such as list event subscriptions, create new ones, and generate keys. Event Grid uses Azure role-based access control (Azure RBAC).

Built-in roles

Event Grid provides the following built-in roles:

Role	Description
Event Grid Subscription Reader	Lets you read Event Grid event subscriptions.
Event Grid Subscription Contributor	Lets you manage Event Grid event subscription operations.
Event Grid Contributor	Lets you create and manage Event Grid resources.
Event Grid Data Sender	Lets you send events to Event Grid topics.

The Event Grid Subscription Reader and Event Grid Subscription Contributor roles are for managing event subscriptions. They're important when implementing event domains because they give users the permissions they need to subscribe to topics in your event domain. These roles are focused on event subscriptions and don't grant access for actions such as creating topics.

The Event Grid Contributor role allows you to create and manage Event Grid resources.

Permissions for event subscriptions

If you're using an event handler that isn't a WebHook (such as an event hub or queue storage), you need write access to that resource. This permissions check prevents an unauthorized user from sending events to your resource.

You must have the **Microsoft.EventGrid/EventSubscriptions/Write** permission on the resource that is the event source. You need this permission because you're writing a new subscription at the scope of the resource. The required resource differs based on whether you're subscribing to a system topic or custom topic. Both types are described in this section.

Topic Type	Description
System topics	Need permission to write a new event subscription at the scope of the resource publishing the event. The format of the resource is: `/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/{resource-provider}/{resource-type}/{resource-name}`
Custom topics	Need permission to write a new event subscription at the scope of the event grid topic. The format of the resource is: `/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.EventGrid/topics/{topic-name}`

Next unit: Receive events by using webhooks

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Receive events by using webhooks

3 minutes

Webhooks are one of the many ways to receive events from Azure Event Grid. When a new event is ready, Event Grid service POSTs an HTTP request to the configured endpoint with the event in the request body.

Like many other services that support webhooks, Event Grid requires you to prove ownership of your Webhook endpoint before it starts delivering events to that endpoint. This requirement prevents a malicious user from flooding your endpoint with events.

When you use any of the three Azure services listed below, the Azure infrastructure automatically handles this validation:

- Azure Logic Apps with Event Grid Connector
- Azure Automation via webhook
- Azure Functions with Event Grid Trigger

Endpoint validation with Event Grid events

If you're using any other type of endpoint, such as an HTTP trigger based Azure function, your endpoint code needs to participate in a validation handshake with Event Grid. Event Grid supports two ways of validating the subscription.

- **Synchronous handshake**: At the time of event subscription creation, Event Grid sends a subscription validation event to your endpoint. The schema of this event is similar to any other Event Grid event. The data portion of this event includes a `validationCode` property. Your application verifies that the validation request is for an expected event subscription, and returns the validation code in the response synchronously. This handshake mechanism is supported in all Event Grid versions.

- **Asynchronous handshake**: In certain cases, you can't return the ValidationCode in response synchronously. For example, if you use a third-party service (like Zapier or IFTTT), you can't programmatically respond with the validation code.

Starting with version 2018-05-01-preview, Event Grid supports a manual validation handshake. If you're creating an event subscription with an SDK or tool that uses API version 2018-05-01-preview or later, Event Grid sends a `validationUrl` property in the data portion of the subscription validation event. To complete the handshake, find that URL in the event data and do a GET request to it. You can use either a REST client or your web browser.

The provided URL is valid for **5 minutes**. During that time, the provisioning state of the event subscription is `AwaitingManualAction`. If you don't complete the manual validation within 5 minutes, the provisioning state is set to `Failed`. You'll have to create the event subscription again before starting the manual validation.

This authentication mechanism also requires the webhook endpoint to return an HTTP status code of 200 so that it knows that the POST for the validation event was accepted before it can be put in the manual validation mode. In other words, if the endpoint returns 200 but doesn't return back a validation response synchronously, the mode is transitioned to the manual validation mode. If there's a GET on the validation URL within 5 minutes, the validation handshake is considered to be successful.

> **Note**
>
> Using self-signed certificates for validation isn't supported. Use a signed certificate from a commercial certificate authority (CA) instead.

Next unit: Filter events

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Filter events

3 minutes

When creating an event subscription, you have three options for filtering:

- Event types
- Subject begins with or ends with
- Advanced fields and operators

Event type filtering

By default, all event types for the event source are sent to the endpoint. You can decide to send only certain event types to your endpoint. For example, you can get notified of updates to your resources, but not notified for other operations like deletions. In that case, filter by the `Microsoft.Resources.ResourceWriteSuccess` event type. Provide an array with the event types, or specify `All` to get all event types for the event source.

The JSON syntax for filtering by event type is:

JSON Copy

```
"filter": {
  "includedEventTypes": [
    "Microsoft.Resources.ResourceWriteFailure",
    "Microsoft.Resources.ResourceWriteSuccess"
  ]
}
```

Subject filtering

For simple filtering by subject, specify a starting or ending value for the subject. For example, you can specify the subject ends with `.txt` to only get events related to uploading a text file to storage account. Or, you can filter the subject begins with

`/blobServices/default/containers/testcontainer` to get all events for that container but not other containers in the storage account.

The JSON syntax for filtering by subject is:

JSON Copy

```
"filter": {
  "subjectBeginsWith": "/blobServices/default/containers/mycontainer/log",
  "subjectEndsWith": ".jpg"
}
```

Advanced filtering

To filter by values in the data fields and specify the comparison operator, use the advanced filtering option. In advanced filtering, you specify the:

- operator type - The type of comparison.
- key - The field in the event data that you're using for filtering. It can be a number, boolean, or string.
- value or values - The value or values to compare to the key.

The JSON syntax for using advanced filters is:

JSON Copy

```
"filter": {
  "advancedFilters": [
    {
      "operatorType": "NumberGreaterThanOrEquals",
      "key": "Data.Key1",
      "value": 5
    },
    {
      "operatorType": "StringContains",
      "key": "Subject",
      "values": ["container1", "container2"]
    }
  ]
}
```

Next unit: Exercise: Route custom events to web endpoint by using Azure CLI

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Exercise: Route custom events to web endpoint by using Azure CLI

3 minutes

In this exercise you will learn how to:

- Enable an Event Grid resource provider
- Create a custom topic
- Create a message endpoint
- Subscribe to a custom topic
- Send an event to a custom topic

Prerequisites

- An **Azure account** with an active subscription. If you don't already have one, you can sign up for a free trial at https://azure.com/free .

Create a resource group

In this section you will open your terminal and create some variables that will be used throughout the rest of the exercise to make command entry, and unique resource name creation, a bit easier.

1. Launch the Cloud Shell: https://shell.azure.com
2. Select **Bash** as the shell.
3. Run the commands below to create the variables. Replace `<myLocation>` with a region near you.

Bash Copy

```
let rNum=$RANDOM*$RANDOM
myLocation=<myLocation>
myTopicName="az204-egtopic-${rNum}"
```

```
mySiteName="az204-egsite-${rNum}"
mySiteURL="https://${mySiteName}.azurewebsites.net"
```

4. Create a resource group for the new resources you will be creating.

Bash Copy

```
az group create --name az204-evgrid-rg --location $myLocation
```

Enable an Event Grid resource provider

> **Note**
>
> This step is only needed on subscriptions that haven't previously used Event Grid.

Register the Event Grid resource provider by using the `az provider register` command.

Bash Copy

```
az provider register --namespace Microsoft.EventGrid
```

It can take a few minutes for the registration to complete. To check the status run the command below.

Bash Copy

```
az provider show --namespace Microsoft.EventGrid --query "registrationState"
```

Create a custom topic

Create a custom topic by using the `az eventgrid topic create` command. The name must be unique because it is part of the DNS entry.

Bash Copy

```
az eventgrid topic create --name $myTopicName \
    --location $myLocation \
    --resource-group az204-evgrid-rg
```

Create a message endpoint

Before subscribing to the custom topic, we need to create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. The script below uses a pre-built web app that displays the event messages. The deployed solution includes an App Service plan, an App Service web app, and source code from GitHub. It also generates a unique name for the site.

1. Create a message endpoint. The deployment may take a few minutes to complete.

```bash
az deployment group create \
    --resource-group az204-evgrid-rg \
    --template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-viewer/main/azuredeploy.json" \
    --parameters siteName=$mySiteName hostingPlanName=viewerhost

echo "Your web app URL: ${mySiteURL}"
```

> **Note**
>
> This command may take a few minutes to complete.

2. In a new tab navigate to the URL generated at the end of the script above to ensure the web app is running. You should see the site with no messages currently displayed.

> **Tip**
>
> Leave the browser running, it is used to show updates.

Subscribe to a custom topic

You subscribe to an event grid topic to tell Event Grid which events you want to track and where to send those events.

1. Subscribe to a custom topic by using the `az eventgrid event-subscription create` command. The script below will grab the needed subscription ID from your account and use in the creation of the event subscription.

Bash Copy

```
endpoint="${mySiteURL}/api/updates"
subId=$(az account show --subscription "" | jq -r '.id')

az eventgrid event-subscription create \
    --source-resource-id "/subscriptions/$subId/resourceGroups/az204-evgrid-rg/providers/Microsoft.EventGrid/topics/$myTopicName" \
    --name az204ViewerSub \
    --endpoint $endpoint
```

2. View your web app again, and notice that a subscription validation event has been sent to it. Select the eye icon to expand the event data. Event Grid sends the validation event so the endpoint can verify that it wants to receive event data. The web app includes code to validate the subscription.

Send an event to your custom topic

Trigger an event to see how Event Grid distributes the message to your endpoint.

1. Retrieve URL and key for the custom topic.

Bash Copy

```
topicEndpoint=$(az eventgrid topic show --name $myTopicName -g az204-evgrid-rg --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name $myTopicName -g az204-evgrid-rg --query "key1" --output tsv)
```

2. Create event data to send. Typically, an application or Azure service would send the event data, we're creating data for the purposes of the exercise.

Bash Copy

```
event='[ {"id": "'"$RANDOM"'", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles", "eventTime": "'`date +%Y-%m-%dT%H:%M:%S%z`'", "data":{ "make": "Contoso", "model": "Monster"},"dataVersion": "1.0"} ]'
```

3. Use `curl` to send the event to the topic.

Bash Copy

```
curl -X POST -H "aeg-sas-key: $key" -d "$event" $topicEndpoint
```

4. View your web app to see the event you just sent. Select the eye icon to expand the event data. Event Grid sends the validation event so the endpoint can verify that it wants to receive event data. The web app includes code to validate the subscription.

JSON Copy

```
{
"id": "29078",
"eventType": "recordInserted",
"subject": "myapp/vehicles/motorcycles",
"eventTime": "2019-12-02T22:23:03+00:00",
"data": {
    "make": "Contoso",
    "model": "Northwind"
},
"dataVersion": "1.0",
"metadataVersion": "1",
"topic": "/subscriptions/{subscription-id}/resourceGroups/az204-evgrid-
rg/providers/Microsoft.EventGrid/topics/az204-egtopic-589377852"
}
```

Clean up resources

When you no longer need the resources in this exercise use the following command to delete the resource group and associated resources.

Bash Copy

```
az group delete --name az204-evgrid-rg --no-wait
```

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Knowledge check

3 minutes

Check your knowledge

1. Which of the following event schema properties requires a value?

- ○ Topic
- ◉ Data

  ✗ That's incorrect. A value is not required in this property.
- ○ Subject

  ✓ That's correct. The subject property specifies the publisher-defined path to the event subject and is required.

2. Which of the following Event Grid built-in roles is appropriate for managing Event Grid resources?

- ○ Event Grid Contributor

  ✓ That's correct. The Event Grid Contributor role has permissions to manage resources.
- ◉ Event Grid Subscription Contributor

  ✗ That's incorrect. The Event Grid Subscription Contributor role has permissions to manage subscription operations.
- ○ Event Grid Data Sender

Next unit: Summary

Continue

How are we doing? ☆ ☆ ☆ ☆ ☆