

# Contents

[Azure Event Grid Documentation](#)

[Switch to Event Grid on Kubernetes documentation](#)

[Switch to Event Grid on IoT Edge documentation](#)

[Overview](#)

[What is Event Grid?](#)

[Compare messaging services](#)

[Release notes](#)

[Quickstarts](#)

[Storage events](#)

[Portal](#)

[Azure CLI](#)

[Azure PowerShell](#)

[ARM template](#)

[Custom events](#)

[Portal](#)

[Azure CLI](#)

[PowerShell](#)

[Azure Functions as event handler](#)

[Queue storage as event handler](#)

[Event Hubs as event handler](#)

[Container Registry events](#)

[Handle Azure Communication Services events](#)

[Azure Cache for Redis events](#)

[Tutorials](#)

[Email when VM changes](#)

[Resize uploaded images](#)

[Trigger Automation runbook](#)

[Email when IoT Hub device disconnects](#)

[Handle Service Bus events via Event Grid](#)

[Azure Logic Apps](#)

[Azure Functions](#)

[Stream data from Event Hubs](#)

[Route Media Services events](#)

[Route to Hybrid Connection](#)

[Samples](#)

[Azure CLI](#)

[Create custom topic](#)

[Subscribe to Azure subscription](#)

[Subscribe to Blob storage](#)

[Subscribe to custom topic](#)

[Subscribe to resource group](#)

[Subscribe and filter events for resource group](#)

[Azure PowerShell](#)

[Create custom topic](#)

[Subscribe to Azure subscription](#)

[Subscribe to Blob storage](#)

[Subscribe to custom topic](#)

[Subscribe to resource group](#)

[Subscribe and filter events for resource group](#)

[Resource Manager templates](#)

[Code samples](#)

[Concepts](#)

[Event Grid terminology](#)

[Custom topics](#)

[System topics](#)

[Overview of system topics](#)

[Event sources \(publishers\)](#)

[Azure App Configuration](#)

[Azure App Service](#)

[Azure Blob Storage](#)

[Azure Communication Services](#)

[Azure Container Registry](#)

[Azure Event Hubs](#)

[Azure IoT Hub](#)

[Azure Key Vault](#)

[Azure Machine Learning](#)

[Azure Maps](#)

[Azure Media Services](#)

[Azure Policy](#)

[Azure resource groups](#)

[Azure Service Bus](#)

[Azure SignalR](#)

[Azure subscriptions](#)

[Azure Cache for Redis](#)

[Partner Events](#)

[Overview of Partner Events](#)

[Event sources \(publishers\)](#)

[Auth0](#)

[Partner onboarding overview](#)

[Event Domains](#)

[Event schema formats](#)

[Event Grid event schema](#)

[Cloud event schema](#)

[Event handlers](#)

[Webhooks](#)

[Azure Functions](#)

[Event Hubs](#)

[Service Bus](#)

[Relay hybrid connections](#)

[Storage queues](#)

[Event delivery](#)

[Event filtering](#)

[Delivery and retry](#)

- [Webhook event delivery](#)
- [Custom delivery properties](#)
- [Disaster recovery](#)
  - [Geo disaster recovery](#)
- [Security](#)
  - [Security baseline](#)
  - [Security controls by Azure Policy](#)
  - [Authenticate event delivery to event handlers](#)
  - [Authenticate publishing clients](#)
  - [Authorize access to Event Grid resources](#)
  - [Network security](#)
- [Monitor and troubleshoot](#)
  - [Diagnostic logs](#)
  - [Metrics](#)
- [Use cases](#)
  - [Use cases for domains](#)
- [How-to guides](#)
  - [Create, view, and manage system topics](#)
    - [Azure portal](#)
    - [Azure CLI](#)
    - [Azure Resource Manager template](#)
  - [Partner Events](#)
    - [Onboard as a partner](#)
    - [Subscribe to Auth0 events](#)
  - [Get access keys for topics or domains](#)
  - [Post to custom topic](#)
  - [Receive events at HTTP endpoint](#)
  - [Set dead-letter location and retry policy](#)
  - [Filter events](#)
  - [Query event subscriptions](#)
  - [Subscribe through portal](#)
  - [Map custom fields to schema](#)

[Use CloudEvents schema](#)

[Manage topics with Event Domains](#)

[Implement client-side failover](#)

[Track asynchronous REST operations](#)

[Move](#)

[Move system topics across regions](#)

[Move custom topics across regions](#)

[Move domains across regions](#)

[Secure](#)

[Publish events securely over a private link](#)

[Deliver events using managed identity](#)

[Enable managed identity for a custom topic or a domain](#)

[Enable managed identity for a system topic](#)

[Grant managed identity the access to Event Grid destination](#)

[Create an event subscription that uses the managed identity](#)

[Deliver events securely over a private link](#)

[Configure IP firewall](#)

[Secure Webhook with Azure Active Directory](#)

[Troubleshoot](#)

[Troubleshoot Event Grid issues](#)

[Enable diagnostic logs](#)

[Troubleshoot errors](#)

[Troubleshoot network connectivity issues](#)

[Troubleshoot subscription validation](#)

[View metrics](#)

[Create alerts](#)

[Reference](#)

[Azure CLI](#)

[PowerShell](#)

[REST](#)

[.NET](#)

[Java](#)

- [Python](#)
- [Node.js](#)
- [Quotas and limits](#)
- [Subscription schema](#)
- [Template resources](#)
  - [Event subscriptions](#)
  - [Topics](#)
- [SDKs](#)
- [Azure Policy built-ins](#)
- [Resources](#)
  - [Build your skills with Microsoft Learn](#)
  - [Azure Roadmap](#)
  - [Pricing](#)
  - [Pricing Calculator](#)
  - [Stack Overflow](#)

# Tutorial: Publish, subscribe to events locally

5/25/2021 • 6 minutes to read • [Edit Online](#)

This article walks you through all the steps needed to publish and subscribe to events using Event Grid on IoT Edge.

## NOTE

To learn about Azure Event Grid topics and subscriptions, see [Event Grid Concepts](#).

## Prerequisites

In order to complete this tutorial, you will need:

- **Azure subscription** - Create a [free account](#) if you don't already have one.
- **Azure IoT Hub and IoT Edge device** - Follow the steps in the quickstart for [Linux](#) or [Windows](#) devices if you don't already have one.

## Deploy Event Grid IoT Edge module

There are several ways to deploy modules to an IoT Edge device and all of them work for Azure Event Grid on IoT Edge. This article describes the steps to deploy Event Grid on IoT Edge from the Azure portal.

## NOTE

In this tutorial, you will deploy the Event Grid module without persistence. It means that any topics and subscriptions you create in this tutorial will be deleted when you redeploy the module. For more information on how to setup persistence, see the following articles: [Persist state in Linux](#) or [Persist state in Windows](#). For production workloads, we recommend that you install the Event Grid module with persistence.

### Select your IoT Edge device

1. Sign in to the [Azure portal](#)
2. Navigate to your IoT Hub.
3. Select **IoT Edge** from the menu in the **Automatic Device Management** section.
4. Click on the ID of the target device from the list of devices
5. Select **Set Modules**. Keep the page open. You will continue with the steps in the next section.

### Configure a deployment manifest

A deployment manifest is a JSON document that describes which modules to deploy, how data flows between the modules, and desired properties of the module twins. The Azure portal has a wizard that walks you through creating a deployment manifest, instead of building the JSON document manually. It has three steps: **Add modules**, **Specify routes**, and **Review deployment**.

### Add modules

1. In the **Deployment Modules** section, select **Add**
2. From the types of modules in the drop-down list, select **IoT Edge Module**
3. Provide the name, image, container create options of the container:

- **Name:** eventgridmodule
- **Image URI:** `mcr.microsoft.com/azure-event-grid/iotedge:latest`
- **Container Create Options:**

**NOTE**

Always check the version of your image to ensure it has the features you require. If you have a previous version of the container image already pulled on your machine, you'll need to specify the version tag you want, or delete the existing image before pulling again with the `:latest` tag. For more on image versions and tags, see [release notes](#).

```
{
  "Env": [
    "inbound__clientAuth__clientCert__enabled=false"
  ],
  "HostConfig": {
    "PortBindings": {
      "4438/tcp": [
        {
          "HostPort": "4438"
        }
      ]
    }
  }
}
```

4. Click **Save**

5. Continue to the next section to add the Azure Event Grid Subscriber module before deploying them together.

**IMPORTANT**

In this tutorial, you will deploy the Event Grid module with client authentication disabled. For production workloads, we recommend that you enable the client authentication. For more information on how to configure Event Grid module securely, see [Security and authentication](#).

If you are using an Azure VM as an edge device, add an inbound port rule to allow inbound traffic on the port 4438. For instructions on adding the rule, see [How to open ports to a VM](#).

## Deploy Event Grid Subscriber IoT Edge module

This section shows you how to deploy another IoT module which would act as an event handler to which events can be delivered.

### Add modules

1. In the **Deployment Modules** section, select **Add** again.
2. From the types of modules in the drop-down list, select **IoT Edge Module**
3. Provide the name, image, and container create options of the container:
  - **Name:** subscriber
  - **Image URI:** `mcr.microsoft.com/azure-event-grid/iotedge-samplesubscriber:latest`
  - **Container Create Options:** None
4. Click **Save**

5. Click **Next** to continue to the routes section

### Setup routes

Keep the default routes, and select **Next** to continue to the review section

### Submit the deployment request

1. The review section shows you the JSON deployment manifest that was created based on your selections in the previous section. Confirm that you see both the modules: **eventgridmodule** and **subscriber** listed in the JSON.
2. Review your deployment information, then select **Submit**. After you submit the deployment, you return to the **device** page.
3. In the **Modules section**, verify that both **eventgrid** and **subscriber** modules are listed. And, verify that the **Specified in deployment** and **Reported by device** columns are set to **Yes**.

It may take a few moments for the module to be started on the device and then reported back to IoT Hub. Refresh the page to see an updated status.

## Create a topic

As a publisher of an event, you need to create an event grid topic. In Azure Event Grid, a topic refers to an endpoint where publishers can send events to.

1. Create `topic.json` with the following content. For details about the payload, see our [API documentation](#).

```
{  
  "name": "sampleTopic1",  
  "properties": {  
    "inputschema": "eventGridSchema"  
  }  
}
```

2. Run the following command to create an event grid topic. Confirm that you see the HTTP status code is

`200 OK`.

```
curl -k -H "Content-Type: application/json" -X PUT -d @topic.json https://<your-edge-device-public-ip-here>:4438/topics/sampleTopic1?api-version=2019-01-01-preview
```

3. Run the following command to verify topic was created successfully. HTTP Status Code of 200 OK should be returned.

```
curl -k -H "Content-Type: application/json" -X GET -g https://<your-edge-device-public-ip-here>:4438/topics/sampleTopic1?api-version=2019-01-01-preview
```

Sample output:

```
[  
  {  
    "id": "/iotHubs/eg-iot-edge-hub/devices/eg-edge-  
device/modules/eventgridmodule/topics/sampleTopic1",  
    "name": "sampleTopic1",  
    "type": "Microsoft.EventGrid/topics",  
    "properties": {  
      "endpoint": "https://<edge-vm-ip>:4438/topics/sampleTopic1/events?api-version=2019-01-01-  
preview",  
      "inputSchema": "EventGridSchema"  
    }  
  }  
]
```

## Create an event subscription

Subscribers can register for events published to a topic. To receive any event, you'll need to create an Event Grid subscription for a topic of interest.

### NOTE

If you need to guarantee pending events are persisted in the event of a device restart, you'll need to enable persistence for the event subscription. For more information on how to setup persistence, see the following articles: [Persist state in Linux](#) or [Persist state in Windows](#).

1. Create `subscription.json` with the following content. For details about the payload, see our [API documentation](#)

```
{  
  "properties": {  
    "destination": {  
      "endpointType": "WebHook",  
      "properties": {  
        "endpointUrl": "https://subscriber:4430"  
      }  
    }  
  }  
}
```

### NOTE

The `endpointType` property specifies that the subscriber is a [Webhook](#). The `endpointUrl` specifies the URL at which the subscriber is listening for events. This URL corresponds to the Azure Subscriber sample you deployed earlier.

2. Run the following command to create a subscription for the topic. Confirm that you see the HTTP status code is `200 OK`.

```
curl -k -H "Content-Type: application/json" -X PUT -g -d @subscription.json https://<your-edge-  
device-public-ip-here>:4438/topics/sampleTopic1/eventSubscriptions/sampleSubscription1?api-  
version=2019-01-01-preview
```

3. Run the following command to verify subscription was created successfully. HTTP Status Code of 200 OK should be returned.

```
curl -k -H "Content-Type: application/json" -X GET -g https://<your-edge-device-public-ip-here>:4438/topics/sampleTopic1/eventSubscriptions/sampleSubscription1?api-version=2019-01-01-preview
```

Sample output:

```
{  
    "id": "/iotHubs/eg-iot-edge-hub/devices/eg-edge-device/modules/eventgridmodule/topics/sampleTopic1/eventSubscriptions/sampleSubscription1",  
    "type": "Microsoft.EventGrid/eventSubscriptions",  
    "name": "sampleSubscription1",  
    "properties": {  
        "Topic": "sampleTopic1",  
        "destination": {  
            "endpointType": "WebHook",  
            "properties": {  
                "endpointUrl": "https://subscriber:4430"  
            }  
        }  
    }  
}
```

## Publish an event

1. Create event.json with the following content. For details about the payload, see our [API documentation](#).

```
[  
    {  
        "id": "eventId-func-0",  
        "eventType": "recordInserted",  
        "subject": "myapp/vehicles/motorcycles",  
        "eventTime": "2019-07-28T21:03:07+00:00",  
        "dataVersion": "1.0",  
        "data": {  
            "make": "Ducati",  
            "model": "Monster"  
        }  
    }  
]
```

2. Run the following command to publish an event.

```
curl -k -H "Content-Type: application/json" -X POST -g -d @event.json https://<your-edge-device-public-ip-here>:4438/topics/sampleTopic1/events?api-version=2019-01-01-preview
```

## Verify event delivery

1. SSH or RDP into your IoT Edge VM.
2. Check the subscriber logs:

On Windows, run the following command:

```
docker -H npipe:///./pipe/iotedge_moby_engine container logs subscriber
```

On Linux, run the following command:

```
sudo docker logs subscriber
```

Sample output:

```
Received Event:  
{  
  "id": "eventId-func-0",  
  "topic": "sampleTopic1",  
  "subject": "myapp/vehicles/motorcycles",  
  "eventType": "recordInserted",  
  "eventTime": "2019-07-28T21:03:07+00:00",  
  "dataVersion": "1.0",  
  "metadataVersion": "1",  
  "data": {  
    "make": "Ducati",  
    "model": "Monster"  
  }  
}
```

## Cleanup resources

- Run the following command to delete the topic and all its subscriptions.

```
curl -k -H "Content-Type: application/json" -X DELETE https://<your-edge-device-public-ip-here>:4438/topics/sampleTopic1?api-version=2019-01-01-preview
```

- Delete the subscriber module from your IoT Edge device.

## Next steps

In this tutorial, you created an event grid topic, subscription, and published events. Now that you know the basic steps, see the following articles:

- To troubleshoot issues with using Azure Event Grid on IoT Edge, see [Troubleshooting guide](#).
- Create/update subscription with [filters](#).
- Enable persistence of Event Grid module on [Linux](#) or [Windows](#)
- Follow [documentation](#) to configure client authentication
- Forward events to Azure Functions in the cloud by following this [tutorial](#)
- [React to Blob Storage events on IoT Edge](#)
- [Monitor topics and subscriptions on the edge](#)

# What is Azure Event Grid?

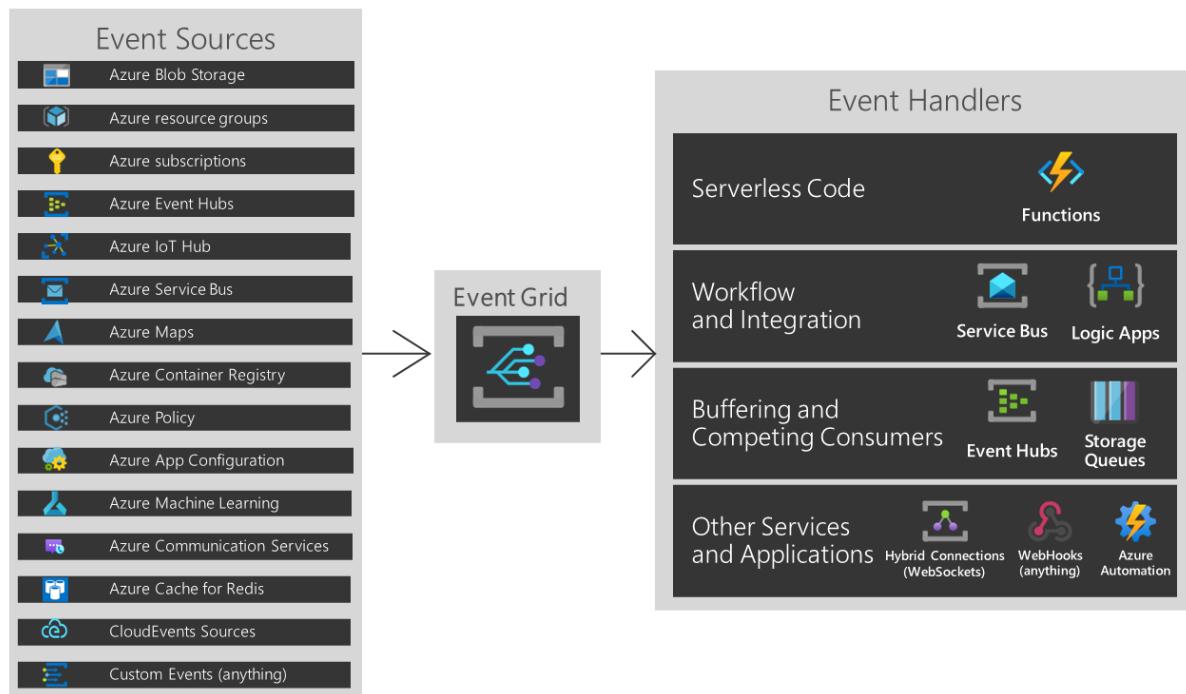
3/29/2021 • 4 minutes to read • [Edit Online](#)

Azure Event Grid allows you to easily build applications with event-based architectures. First, select the Azure resource you would like to subscribe to, and then give the event handler or WebHook endpoint to send the event to. 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.

Azure Event Grid is deployed to maximize availability by natively spreading across multiple fault domains in every region, and across availability zones (in regions that support them). For a list of regions that are supported by Event Grid, see [Products available by region](#).

This article provides an overview of Azure Event Grid. If you want to get started with Event Grid, see [Create and route custom events with Azure Event Grid](#).



## NOTE

This image shows how Event Grid connects sources and handlers, and isn't a comprehensive list of supported integrations. For a list of all supported event sources, see the following section.

## Event sources

Currently, the following Azure services support sending events to Event Grid. For more information about a source in the list, select the link.

- [Azure App Configuration](#)
- [Azure Blob Storage](#)
- [Azure Communication Services](#)

- [Azure Container Registry](#)
- [Azure Event Hubs](#)
- [Azure IoT Hub](#)
- [Azure Key Vault](#)
- [Azure Machine Learning](#)
- [Azure Maps](#)
- [Azure Media Services](#)
- [Azure Policy](#)
- [Azure resource groups](#)
- [Azure Service Bus](#)
- [Azure SignalR](#)
- [Azure subscriptions](#)
- [Azure Cache for Redis](#)

## Event handlers

For full details on the capabilities of each handler as well as related articles, see [event handlers](#). Currently, the following Azure services support handling events from Event Grid:

- [Azure Automation](#)
- [Azure Functions](#)
- [Event Hubs](#)
- [Relay Hybrid Connections](#)
- [Logic Apps](#)
- [Power Automate \(Formerly known as Microsoft Flow\)](#)
- [Service Bus](#)
- [Queue Storage](#)
- [WebHooks](#)

## Concepts

There are five concepts in Azure Event Grid that let you get going:

- **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.

For more information about these concepts, see [Concepts in Azure Event Grid](#).

## Capabilities

Here are some of the key features of Azure Event Grid:

- **Simplicity** - Point and click to aim events from your Azure resource to any event handler or endpoint.
- **Advanced filtering** - Filter on event type or event publish path to make sure event handlers only receive relevant events.
- **Fan-out** - Subscribe several endpoints to the same event to send copies of the event to as many places as needed.

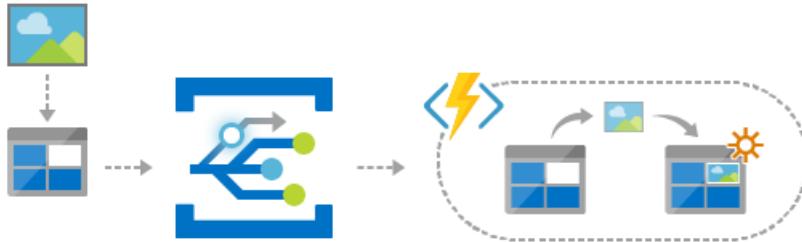
- **Reliability** - 24-hour retry with exponential backoff to make sure events are delivered.
- **Pay-per-event** - Pay only for the amount you use Event Grid.
- **High throughput** - Build high-volume workloads on Event Grid.
- **Built-in Events** - Get up and running quickly with resource-defined built-in events.
- **Custom Events** - Use Event Grid to route, filter, and reliably deliver custom events in your app.

For a comparison of Event Grid, Event Hubs, and Service Bus, see [Choose between Azure services that deliver messages](#).

## What can I do with Event Grid?

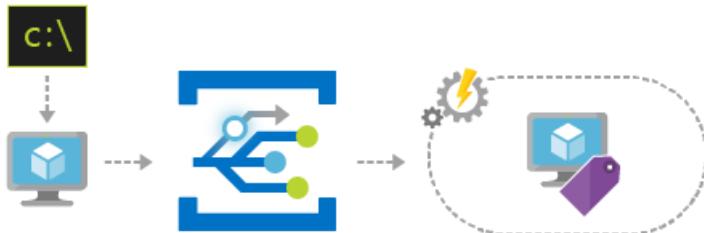
Azure Event Grid provides several features that vastly improve serverless, ops automation, and [integration](#) work:

### Serverless application architectures



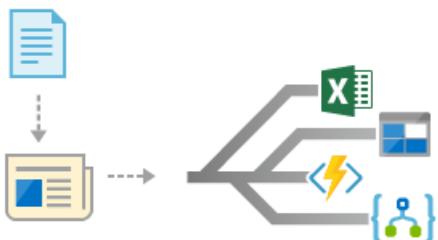
Event Grid connects data sources and event handlers. For example, use Event Grid to trigger a serverless function that analyzes images when added to a blob storage container.

### Ops Automation



Event Grid allows you to speed automation and simplify policy enforcement. For example, use Event Grid to notify Azure Automation when a virtual machine or database in Azure SQL is created. Use the events to automatically check that service configurations are compliant, put metadata into operations tools, tag virtual machines, or file work items.

### Application integration



Event Grid connects your app with other services. For example, create a custom topic to send your app's event data to Event Grid, and take advantage of its reliable delivery, advanced routing, and direct integration with Azure. Or, you can use Event Grid with Logic Apps to process data anywhere, without writing code.

## How much does Event Grid cost?

Azure Event Grid uses a pay-per-event pricing model, so you only pay for what you use. The first 100,000 operations per month are free. Operations are defined as event ingress, subscription delivery attempts, management calls, and filtering by subject suffix. For details, see the [pricing page](#).

## Next steps

- [Route Storage Blob events](#)

Respond to storage blob events by using Event Grid.

- [Create and subscribe to custom events](#)

Jump right in and start sending your own custom events to any endpoint using the Azure Event Grid quickstart.

- [Using Logic Apps as an Event Handler](#)

A tutorial on building an app using Logic Apps to react to events pushed by Event Grid.

- [Stream big data into a data warehouse](#)

A tutorial that uses Azure Functions to stream data from Event Hubs to Azure Synapse Analytics.

- [Event Grid REST API reference](#)

Provides reference content for managing Event Subscriptions, routing, and filtering.

# Choose between Azure messaging services - Event Grid, Event Hubs, and Service Bus

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure offers three services that assist with delivering event messages throughout a solution. These services are:

- [Event Grid](#)
- [Event Hubs](#)
- [Service Bus](#)

Although they have some similarities, each service is designed for particular scenarios. This article describes the differences between these services, and helps you understand which one to choose for your application. In many cases, the messaging services are complementary and can be used together.

## Event vs. message services

There's an important distinction to note between services that deliver an event and services that deliver a message.

### Event

An event is a lightweight notification of a condition or a state change. The publisher of the event has no expectation about how the event is handled. The consumer of the event decides what to do with the notification. Events can be discrete units or part of a series.

Discrete events report state change and are actionable. To take the next step, the consumer only needs to know that something happened. The event data has information about what happened but doesn't have the data that triggered the event. For example, an event notifies consumers that a file was created. It may have general information about the file, but it doesn't have the file itself. Discrete events are ideal for [serverless](#) solutions that need to scale.

Series events report a condition and are analyzable. The events are time-ordered and interrelated. The consumer needs the sequenced series of events to analyze what happened.

### Message

A message is raw data produced by a service to be consumed or stored elsewhere. The message contains the data that triggered the message pipeline. The publisher of the message has an expectation about how the consumer handles the message. A contract exists between the two sides. For example, the publisher sends a message with the raw data, and expects the consumer to create a file from that data and send a response when the work is done.

## Comparison of 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	Purpose	Type	When to Use
Service Bus	High-value enterprise messaging	Message	Order processing and financial transactions

## Event Grid

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

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. It distributes the events to registered subscriber endpoints. The event message has the information you need to react to changes in services and applications. Event Grid isn't a data pipeline, and doesn't deliver the actual object that was updated.

Event Grid supports dead-lettering for events that aren't delivered to an endpoint.

It has the following characteristics:

- dynamically scalable
- low cost
- serverless
- at least once delivery

## Event Hubs

Azure Event Hubs is a big data pipeline. It facilitates the capture, retention, and replay of telemetry and event stream data. The data can come from many concurrent sources. Event Hubs allows telemetry and event data to be made available to a variety of stream-processing infrastructures and analytics services. It is available either as data streams or bundled event batches. This service provides a single solution that enables rapid data retrieval for real-time processing as well as repeated replay of stored raw data. It can capture the streaming data into a file for processing and analysis.

It has the following characteristics:

- low latency
- capable of receiving and processing millions of events per second
- at least once delivery

## Service Bus

Service Bus is intended for traditional enterprise applications. These enterprise applications require transactions, ordering, duplicate detection, and instantaneous consistency. Service Bus enables [cloud-native](#) applications to provide reliable state transition management for business processes. When handling high-value messages that cannot be lost or duplicated, use Azure Service Bus. Service Bus also facilitates highly secure communication across hybrid cloud solutions and can connect existing on-premises systems to cloud solutions.

Service Bus is a brokered messaging system. It stores messages in a "broker" (for example, a queue) until the consuming party is ready to receive the messages.

It has the following characteristics:

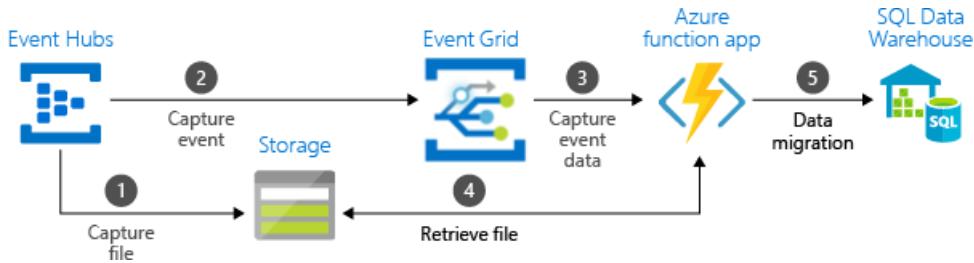
- reliable asynchronous message delivery (enterprise messaging as a service) that requires polling
- advanced messaging features like FIFO, batching/sessions, transactions, dead-lettering, temporal control, routing and filtering, and duplicate detection
- at least once delivery

- optional in-order delivery

## Use the services together

In some cases, you use the services side by side to fulfill distinct roles. For example, an e-commerce site can use Service Bus to process the order, Event Hubs to capture site telemetry, and Event Grid to respond to events like an item was shipped.

In other cases, you link them together to form an event and data pipeline. You use Event Grid to respond to events in the other services. For an example of using Event Grid with Event Hubs to migrate data to a data warehouse, see [Stream big data into a data warehouse](#). The following image shows the workflow for streaming the data.



## Next steps

See the following articles:

- [Asynchronous messaging options in Azure](#)
- [Events, Data Points, and Messages - Choosing the right Azure messaging service for your data.](#)
- [Storage queues and Service Bus queues - compared and contrasted](#)
- To get started with Event Grid, see [Create and route custom events with Azure Event Grid](#).
- To get started with Event Hubs, see [Create an Event Hubs namespace and an event hub using the Azure portal](#).
- To get started with Service Bus, see [Create a Service Bus namespace using the Azure portal](#).

# What's new in Azure Event Grid?

4/27/2021 • 3 minutes to read • [Edit Online](#)

Get notified about when to revisit this page for updates by copying and pasting this URL:

`https://docs.microsoft.com/api/search/rss?search=%22Release+notes++Azure+Event+Grid%22&locale=en-us` into your  feed reader.

Azure Event Grid receives improvements on an ongoing basis. To stay up to date with the most recent developments, this article provides you with information about:

- The latest releases
- Known issues
- Bug fixes
- Deprecated functionality
- Plans for changes

## 6.1.0-preview (2020-10)

- [Managed identities for system topics](#)
- [Custom delivery properties](#)
- [Storage queue - message time-to-live \(TTL\)](#)
- [Advanced filtering improvements](#)
  - Support filtering on array data in incoming events
  - Allow filtering on CloudEvents extensions context attributes
  - New operators
    - StringNotContains
    - StringNotBeginsWith
    - StringNotEndsWith
    - NumberInRange
    - NumberNotInRange
    - IsNullOrUndefined
    - IsNotNull
- [Allow Event Grid schema to CloudEvents 1.0 schema transformations for custom topics and domains](#)

## 6.0.0 (2020-06)

- Add support to new generally available (GA) service API version 2020-06-01.
- The new features that became GA:
  - [Input mappings](#)
  - [Custom input schema](#)
  - [Cloud event V10 schema](#)
  - [Service Bus topic as destination](#)
  - [Azure function as destination](#)
  - [Webhook batching](#)
  - [Secure webhook \(Azure Active Directory support\)](#)
  - [Ip filtering](#)

- [Private Link Service support](#)
- [Event delivery schema](#)

## 5.3.2-preview (2020-05)

- This release includes additional bug fixes to enhance quality.
- As version 5.3.1-preview, this release corresponds to the 2020-04-01-Preview API version, which includes the following new functionalities:
  - [Support for IP Filtering when publishing events to domains and topics](#)
  - [Partner topics](#)
  - [System topics as tracked resources in Azure portal](#)
  - [Event delivery with managed service identity](#)
  - [Private Link Service support](#)

## 5.3.1-preview (2020-04)

- This release includes various bug fixes to enhance quality.
- As version 5.3.0-preview, this release corresponds to the 2020-04-01-Preview API version, which includes the following new functionalities:
  - [Support for IP Filtering when publishing events to domains and topics](#)
  - [Partner topics](#)
  - [System topics as tracked resources in Azure portal](#)
  - [Event delivery with managed service identity](#)
  - [Private Link Service support](#)

## 5.3.0-preview (2020-03)

- We introduce new features on top of features already added in version 5.2.0-preview.
- As version 5.2.0-preview, this release corresponds to the 2020-04-01-Preview API version.
- It adds supports to the following new functionalities:
  - [Support for IP Filtering when publishing events to domains and topics](#)
  - [Partner topics](#)
  - [System topics as tracked resources in Azure portal](#)
  - [Event delivery with managed service identity](#)
  - [Private Link Service support](#)

## 5.2.0-preview (2020-01)

- This release corresponds to the 2020-04-01-Preview API version.
- It adds supports to the following new functionality:
  - [Support for IP Filtering when publishing events to domains and topics](#)

## 5.0.0 (2019-05)

- This release corresponds to the `2019-06-01` API version.
- It adds support to the following new functionalities:
  - [Domains](#)
  - Pagination and search filter for resources list operations. For an example, see [Topics - List By Subscription](#).
  - [Service Bus queue as destination](#)

- Advanced filtering

## 4.1.0-preview (2019-03)

- This release corresponds to the 2019-02-01-preview API version.
- It adds support to the following new functionalities:
  - Pagination and search filter for resources list operations. For an example, see [Topics - List By Subscription](#).
  - [Manual create/delete of domain topics](#)
  - [Service Bus Queue as destination](#)

## 4.0.0 (2018-12)

- This release corresponds to the `2019-01-01` stable API version.
- It supports General Availability (GA) of the following functionalities related to event subscriptions:
  - [Dead Letter destination](#)
  - [Azure Storage queue as destination](#)
  - [Azure Relay - Hybrid Connection as destination](#)
  - [Manual handshake validation](#)
  - [Support for retry policies](#)
- Features that are still in preview (such as [Event Grid domains](#) or [advanced filters support](#) can still be accessed using the 3.0.1-preview version of the SDK."

## 3.0.1-preview (2018-10)

- Taking dependency on [10.0.3 version of Newtonsoft NuGet package](#).

## 3.0.0-preview (2018-10)

- This release is a preview SDK for the new features introduced in 2018-09-15-preview API version. - This release includes support for:
  - [Domain and domain topics](#)
  - Introducing [expiration date for event subscription](#)
  - Introducing [advanced filtering](#) for event subscriptions
  - The stable version of the SDK targeting the `2018-01-01` API version continues to exist as version 1.3.0

## Next steps

# Quickstart: Route Blob storage events to web endpoint with the Azure portal

11/2/2020 • 5 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure portal to create a Blob storage account, subscribe to events for that blob storage, and trigger an event to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

When you're finished, you see that the event data has been sent to the web app.

## Create a storage account

1. Sign in to [Azure portal](#).
  2. To create a Blob storage, select **Create a resource**.
  3. Select for **Storage** to filter the available options, and select **Storage account - blob, file, table, queue**.

The screenshot shows the Azure portal's 'New' blade. On the left, there's a sidebar with a 'Create a resource' button (1) highlighted with a red box. Below it is a list of services under categories like Favorites, Recent, and Virtual machines. In the main area, the 'Storage' category (2) is selected and highlighted with a red box. The 'Storage account - blob, file, table, queue' service (3) is also highlighted with a red box. Other visible services include Azure Stack Edge / Data Box Gateway, Data Lake Storage Gen1, Azure Data Box, Backup and Site Recovery, AltaVault AVA-c4, version 4.4.1 (preview), Cloudian HyperCloud for Azure (preview), Veeam Cloud Connect for the Enterprise (preview), and Web.

To subscribe to events, create either a general-purpose v2 storage account or a Blob storage account.

4. On the **Create storage account** page, do the following steps:

- a. Select your Azure subscription.
- b. For **Resource group**, create a new resource group or select an existing one.
- c. Enter the name for your storage account.
- d. Select **Review + create**.

The screenshot shows the 'Create storage account' wizard in the Microsoft Azure portal. The 'Basics' tab is selected. The 'Project details' section shows a subscription set to 'Visual Studio Ultimate with MSDN' and a resource group named '(New) egridgroup'. The 'Instance details' section includes fields for 'Storage account name' (set to 'spegstorage'), 'Location' (set to '(US) East US'), 'Performance' (set to 'Standard'), 'Account kind' (set to 'StorageV2 (general purpose v2)'), 'Replication' (set to 'Read-access geo-redundant storage (RA-GRS)'), and 'Access tier (default)' (set to 'Hot'). At the bottom, there are 'Review + create' and 'Next : Networking >' buttons.

- e. On the **Review + create** page, review the settings, and select **Create**.

**NOTE**

Only storage accounts of kind **StorageV2 (general purpose v2)** and **BlobStorage** support event integration. **Storage (genral purpose v1)** does *not* support integration with Event Grid.

## Create a message endpoint

Before subscribing to the events for the Blob storage, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

1. Select **Deploy to Azure** to deploy the solution to your subscription.



2. On the **Custom deployment** page, do the following steps:

- a. For **Resource group**, select the resource group that you created when creating the storage account. It will be easier for you to clean up after you are done with the tutorial by deleting the resource group.

- b. For **Site Name**, enter a name for the web app.
- c. For **Hosting plan name**, enter a name for the App Service plan to use for hosting the web app.
- d. Select the check box for **I agree to the terms and conditions stated above**.
- e. Select **Purchase**.

The screenshot shows the Microsoft Azure 'Custom deployment' interface. In the 'BASICS' section, the 'Subscription' is set to 'Visual Studio Ultimate with MSDN', 'Resource group' is 'egridgroup', and 'Location' is '(US) East US'. In the 'SETTINGS' section, the 'Site Name' is 'spegridsite', 'Hosting Plan Name' is 'spegridsiteplan', 'Sku' is 'F1', 'Repo URL' is 'https://github.com/Azure-Samples/azure-event-grid-viewer.git', 'Branch' is 'master', and 'Location' is '[resourceGroup().location]'. Under 'TERMS AND CONDITIONS', there is a checked checkbox for 'I agree to the terms and conditions stated above'. A large blue 'Purchase' button is at the bottom.

3. The deployment may take a few minutes to complete. Select Alerts (bell icon) in the portal, and then select **Go to resource group**.

The screenshot shows the Microsoft Azure dashboard with a 'Notifications' overlay. A yellow callout '1' points to the bell icon in the top right corner of the dashboard header. A yellow callout '2' points to the 'Go to resource group' button in the notifications list, which contains a message: 'Deployment succeeded' and 'Deployment 'Microsoft.Template' to resource group 'egridgroup' was successful.' There is also a 'Pin to dashboard' button next to it.

4. On the **Resource group** page, in the list of resources, select the web app that you created. You also see the App Service plan and the storage account in this list.

The screenshot shows the Azure portal's Resource Groups page. A resource group named 'egridgroup' is selected. It contains three resources: 'spegridsite' (App Service), 'spegridsiteplan' (App Service plan), and 'spegstorage' (Storage account). The 'spegridsite' resource is highlighted with a red box.

5. On the App Service page for your web app, select the URL to navigate to the web site. The URL should be in this format: `https://<your-site-name>.azurewebsites.net`.

The screenshot shows the Azure portal's App Services page for the 'spegridsite' service. The URL 'https://spegridsite.azurewebsites.net' is highlighted with a red box.

6. Confirm that you see the site but no events have been posted to it yet.

The screenshot shows a web browser displaying the 'Azure Event Grid Viewer' page. The URL 'spegridsite.azurewebsites.net' is visible in the address bar. The page content is currently empty, indicating no events have been posted.

## Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

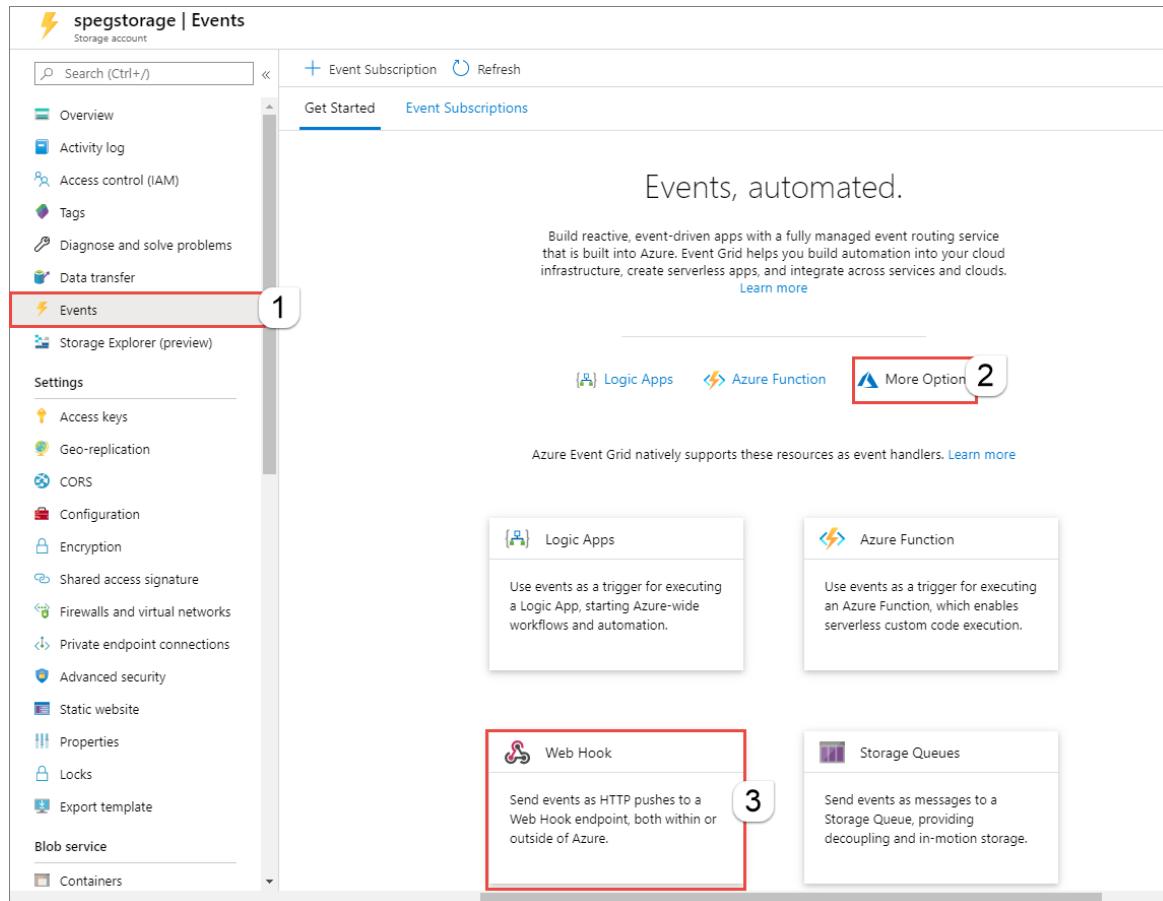
1. Select **Subscriptions** on the left menu.
2. Select the subscription you're using for Event Grid.
3. On the left menu, under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

## Subscribe to the Blob storage

You subscribe to a topic to tell Event Grid which events you want to track, and where to send the events.

1. In the portal, navigate to your Azure Storage account that you created earlier. On the left menu, select **All resources** and select your storage account.
2. On the **Storage account** page, select **Events** on the left menu.
3. Select **More Options**, and **Web Hook**. You are sending events to your viewer app using a web hook for the endpoint.



4. On the **Create Event Subscription** page, do the following steps:

- a. Enter a **name** for the event subscription.
- b. Enter a **name** for the **system topic**. To learn about system topics, see [Overview of system topics](#).

All services > egridgroup > spegridstorage2 | Events >

## Create Event Subscription

Event Grid

Basic Filters Additional Features Advanced Editor

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**EVENT SUBSCRIPTION DETAILS**

Name: spegridsubscription ✓

Event Schema: Event Grid Schema

**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type: Storage account

Source Resource: spegridstorage2

System Topic Name: spegridsystopic1 ✓

**EVENT TYPES**

Pick which event types get pushed to your destination. [Learn more](#)

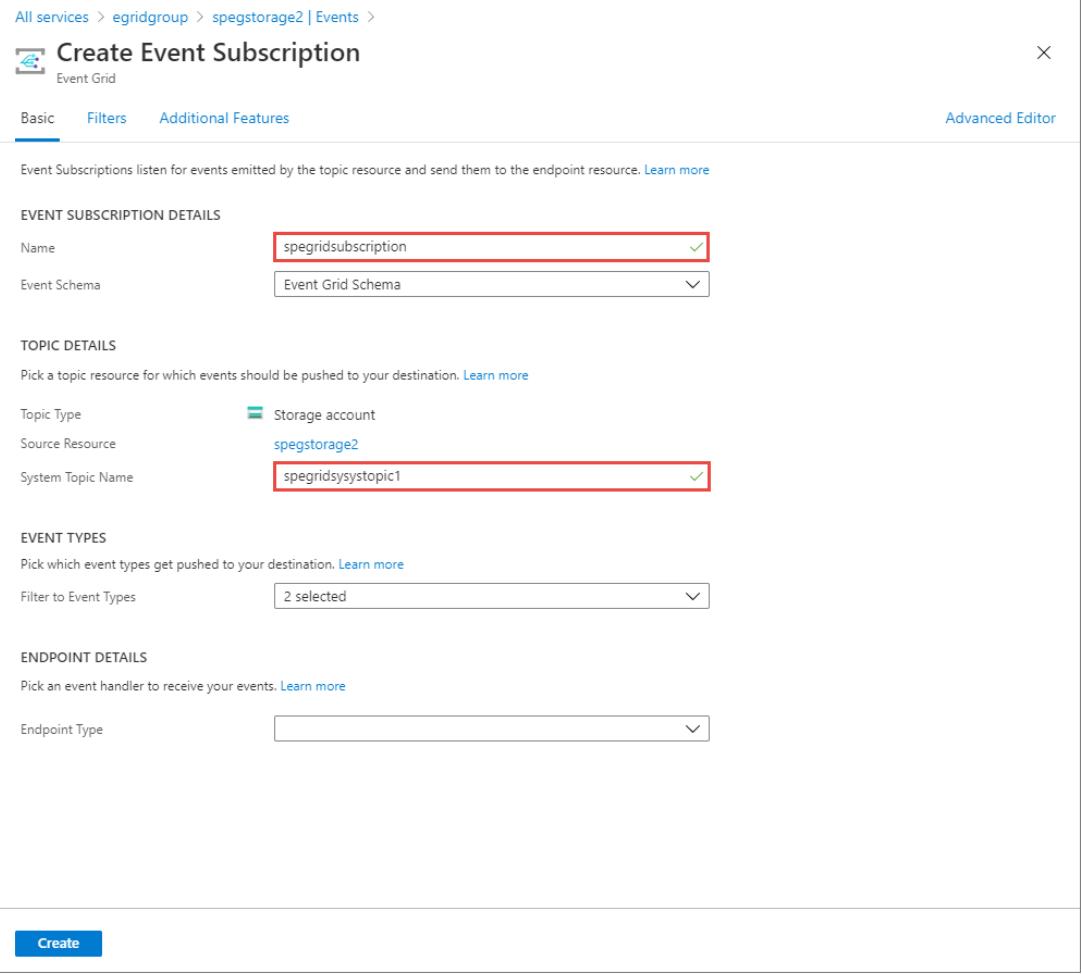
Filter to Event Types: 2 selected

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type: (dropdown menu)

**Create**



c. Select Web Hook for Endpoint type.

## Create Event Subscription



Event Grid

Basic    Filters    Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

### EVENT SUBSCRIPTION DETAILS

Name

spegridsubscription



Event Schema

Event Grid Schema



### TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type

Storage account

Source Resource

spegridstorage2

System Topic Name

spegridsystopic1



### EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

2 selected



### ENDPOINT DETAILS

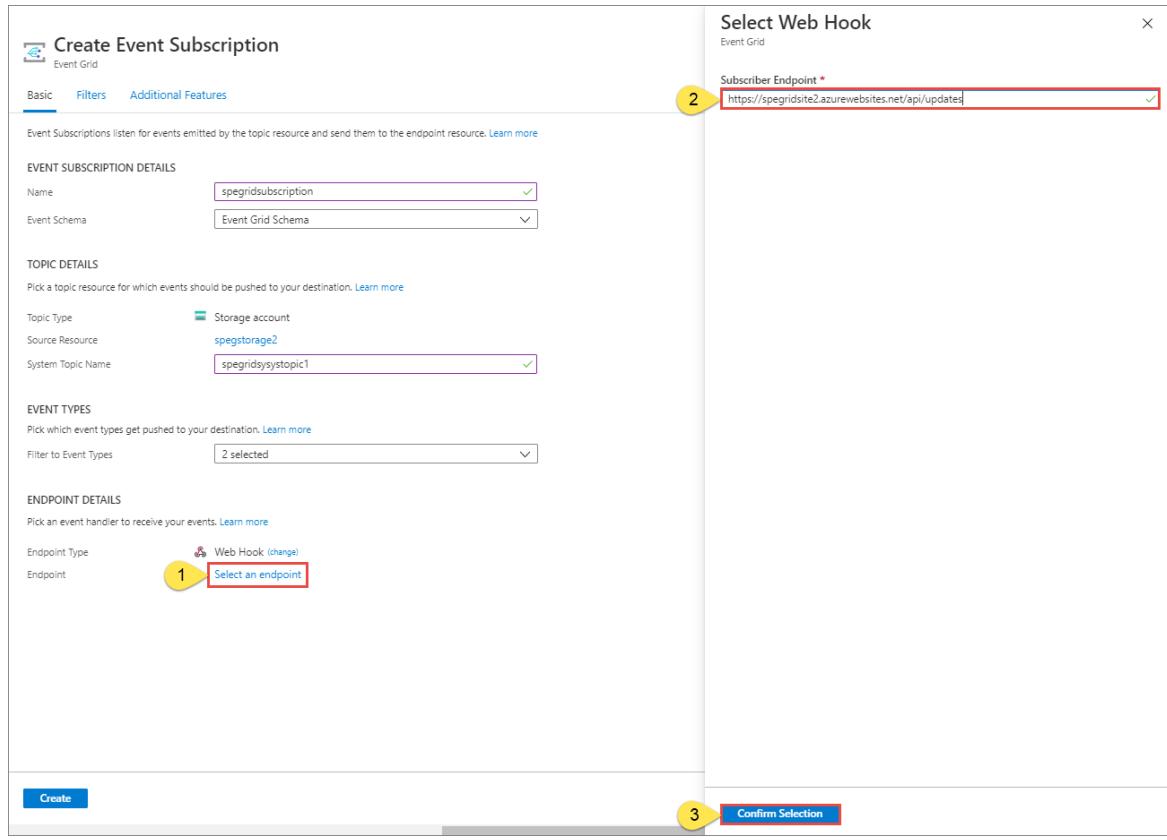
Pick an event handler to receive your events. [Learn more](#)

Endpoint Type

- Azure Function
- Web Hook**
- Storage Queues
- Event Hubs
- Hybrid Connections
- Service Bus Queue
- Service Bus Topic

**Create**

5. For **Endpoint**, click **Select an endpoint**, and enter the URL of your web app and add `api/updates` to the home page URL (for example: `https://spegridsite.azurewebsites.net/api/updates`), and then select **Confirm Selection**.



6. Now, on the **Create Event Subscription** page, select **Create** to create the event subscription.

# Create Event Subscription



Event Grid

Basic    Filters    Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

## EVENT SUBSCRIPTION DETAILS

Name

spegridsubscription



Event Schema

Event Grid Schema



## TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type

Storage account

Source Resource

spegridstorage2

System Topic Name

spegridsysytopic1



## EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

2 selected



## ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type

Web Hook [\(change\)](#)

Endpoint

<https://spegridsite2.azurewebsites.net/api/updates> [\(change\)](#)

**Create**

7. 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.



## Azure Event Grid Viewer

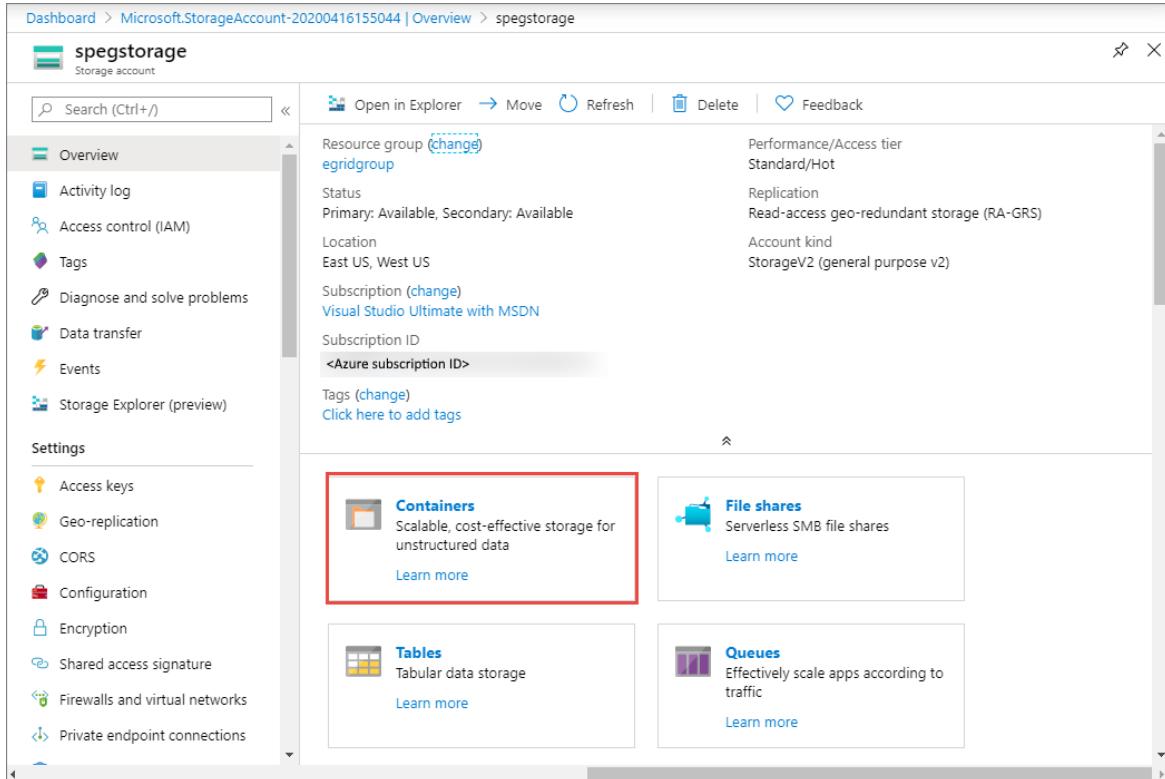
Event Type	Subject
 Microsoft.EventGrid.SubscriptionValidationEvent	
	<pre>[{   "id": "d48934f6-e17a-4fc4-9697-510ed0c7dd63",   "topic": "/subscriptions/{subscription-id}/resourceGroups/eventgroup/providers/Microsoft.Storage/storageAccounts/cosmosdb",   "subject": "",   "data": {     "validationCode": "569F8F44-A944-4840-BF8C-1600FAC184D2",     "validationUrl": "https://rp-westcentralus.eventgrid.azure.net/events/subscriptions/demosubscription/validate?id=52018-06-25T22:23:55.3663623Z&amp;apiVersion=2018-05-01-preview&amp;token=LA9LJwgfvrbFtc7GLh%2fLX29Dm3l2ukVDiamV2QNKI%3d"   },   "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",   "eventTime": "2018-06-25T22:23:55.5405971Z",   "metadataVersion": "1",   "dataVersion": "2" }]</pre>

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint.

## Send an event to your endpoint

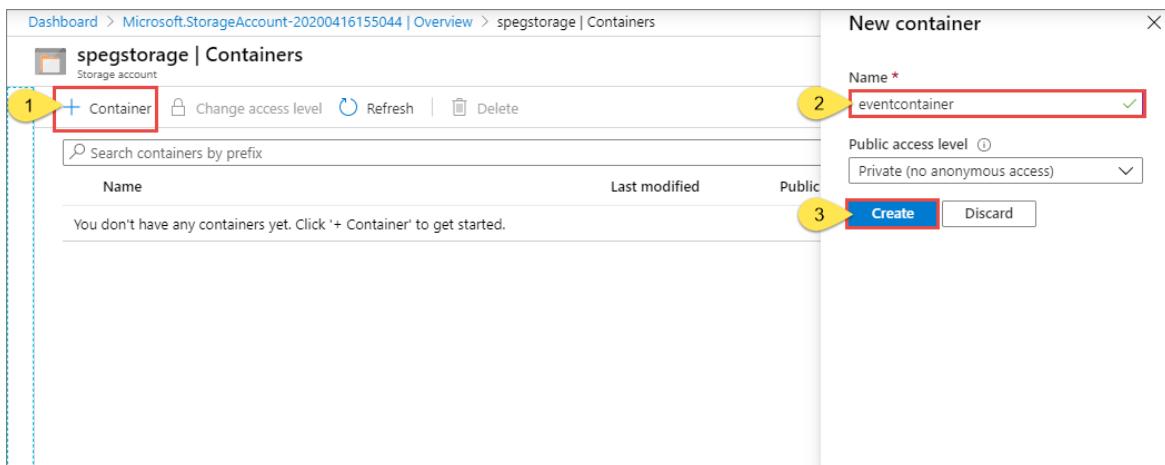
You trigger an event for the Blob storage by uploading a file. The file doesn't need any specific content. The article assumes you have a file named testfile.txt, but you can use any file.

1. In the Azure portal, navigate to your Blob storage account, and select **Containers** on the **Overview** page.



The screenshot shows the Azure Storage Account Overview page for the account 'spegstorage'. The left sidebar contains navigation links: Dashboard, Microsoft.StorageAccount-20200416155044 | Overview, spegstorage, Search (Ctrl+/,) <>, Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Data transfer, Events, Storage Explorer (preview), Settings, Access keys, Geo-replication, CORS, Configuration, Encryption, Shared access signature, Firewalls and virtual networks, Private endpoint connections. The main content area displays account details: Resource group (change) egridgroup, Status Primary: Available, Secondary: Available, Location East US, West US, Subscription (change) Visual Studio Ultimate with MSDN, Subscription ID <Azure subscription ID>, Tags (change) Click here to add tags. Below these details are four service cards: Containers (Scalable, cost-effective storage for unstructured data, Learn more), File shares (Serverless SMB file shares, Learn more), Tables (Tabular data storage, Learn more), and Queues (Effectively scale apps according to traffic, Learn more). The 'Containers' card is highlighted with a red box.

2. Select **+ Container**. Give your container a name, and use any access level, and select **Create**.



3. Select your new container.

The screenshot shows the 'Containers' section of the Azure Storage Account 'spegstorage'. The newly created container 'eventcontainer' is listed in the table. A yellow arrow labeled '1' points to the checkbox next to 'eventcontainer'. A yellow arrow labeled '2' points to the 'Name' column for 'eventcontainer'. A yellow arrow labeled '3' points to the 'Create' button.

4. To upload a file, select **Upload**. On the **Upload blob** page, browse and select a file that you want to upload for testing, and then select **Upload** on that page.

The screenshot shows the 'eventcontainer' blobs page. A yellow arrow labeled '1' points to the 'Upload' button. The page also displays the authentication method as 'Access key' and the location as 'eventcontainer'.

5. Browse to your test file and upload it.

6. You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. The message is in the JSON format and it contains an array with one or more events. In the following example, the JSON message contains an array with one event. View your web app and notice that a **blob created** event was received.

# Azure Event Grid Viewer

Event Type Subject

Microsoft.Storage.BlobCreated	/blobServices/default/containers/eventcontainer/blobs/Casual Dove Gray Cabinets.jpg
{ "topic": "/subscriptions/<subscription ID> "resourceGroup": "resourceGroups/egridgroup/providers/Microsoft.Storage/storageAccounts/pegridstorage", "subject": "blobServices/default/containers/eventcontainer/blobs/Casual Dove Gray Cabinets.jpg", "eventType": "Microsoft.Storage.BlobCreated", "eventTime": "2020-04-16T20:54:11.604813Z", "id": "307919c9-601e-0007-6131-145ea206776e", "data": { "api": "PutBlob", "clientRequestId": "57a2d749-4a62-4bdc-a253-7cd45b11b04a", "requestId": "307919c9-601e-0007-6131-145ea2000000", "eTag": "0x8D7E24850B92502", "contentType": "image/jpeg", "contentLength": 79588, "blobType": "BlockBlob", "url": "https://pegridstorage.blob.core.windows.net/eventcontainer/Casual Dove Gray Cabinets.jpg", "sequencer": "00000000000000000000000000000000A670000000000257c02", "storageDiagnostics": { "batchId": "eabfc0a-5006-000f-0031-14b122000000" } }, "dataVersion": "", "metadataVersion": "1" }	
Microsoft.EventGrid.SubscriptionValidationEvent	

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this article. Otherwise, delete the resources you created in this article.

Select the resource group, and select **Delete resource group**.

## Next steps

Now that you know how to create custom topics and event subscriptions, learn more about what Event Grid can help you do:

- About Event Grid
  - Route Blob storage events to a custom web endpoint
  - Monitor virtual machine changes with Azure Event Grid and Logic Apps
  - Stream big data into a data warehouse

# Quickstart: Route storage events to web endpoint with Azure CLI

3/23/2021 • 5 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure CLI to subscribe to Blob storage events, and trigger the event to view the result.

Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

When you complete the steps described in this article, you see that the event data has been sent to the web app.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Use the Bash environment in Azure Cloud Shell.

 Launch Cloud Shell

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
    - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
    - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).

- Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires version 2.0.70 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the [az group create](#) command.

The following example creates a resource group named `<resource_group_name>` in the *westcentralus* location. Replace `<resource_group_name>` with a unique name for your resource group.

```
az group create --name <resource_group_name> --location westcentralus
```

## Create a storage account

Blob storage events are available in general-purpose v2 storage accounts and Blob storage accounts. **General-purpose v2** storage accounts support all features for all storage services, including Blobs, Files, Queues, and Tables. A **Blob storage account** is a specialized storage account for storing your unstructured data as blobs (objects) in Azure Storage. Blob storage accounts are like general-purpose storage accounts and share all the great durability, availability, scalability, and performance features that you use today including 100% API consistency for block blobs and append blobs. For more information, see [Azure storage account overview](#).

Replace `<storage_account_name>` with a unique name for your storage account, and `<resource_group_name>` with the resource group you created earlier.

```
az storage account create \
--name <storage_account_name> \
--location westcentralus \
--resource-group <resource_group_name> \
--sku Standard_LRS \
--kind BlobStorage \
--access-tier Hot
```

## Create a message endpoint

Before subscribing to the topic, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

Replace `<your-site-name>` with a unique name for your web app. The web app name must be unique because it's part of the DNS entry.

```
sitename=<your-site-name>

az deployment group create \
--resource-group <resource_group_name> \
--template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-
viewer/master/azuredeploy.json" \
--parameters siteName=$sitename hostingPlanName=viewerhost
```

The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: `https://<your-site-name>.azurewebsites.net`

You should see the site with no messages currently displayed.

## Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

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

It might take a moment for the registration to finish. To check the status, run:

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

When `registrationState` is `Registered`, you're ready to continue.

## Subscribe to your storage account

You subscribe to a topic to tell Event Grid which events you want to track and where to send those events. The following example subscribes to the storage account you created, and passes the URL from your web app as the endpoint for event notification. Replace `<event_subscription_name>` with a name for your event subscription. For `<resource_group_name>` and `<storage_account_name>`, use the values you created earlier.

The endpoint for your web app must include the suffix `/api/updates/`.

```
storageid=$(az storage account show --name <storage_account_name> --resource-group <resource_group_name> --query id --output tsv)
endpoint=https://$sitename.azurewebsites.net/api/updates

az eventgrid event-subscription create \
--source-resource-id $storageid \
--name <event_subscription_name> \
--endpoint $endpoint
```

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.



## Azure Event Grid Viewer

### Event Type



Microsoft.EventGrid.SubscriptionValidationEvent

```
[{
  "id": "803f7b19-15d9-4753-aad4-feaf6b535adc",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/gridresourcegroup/providers/microsoft.eventgrid/",
  "subject": "",
  "data": {
    "validationCode": "2F6D57C8-97A3-4073-A4AF-3EF5DE46A8B4"
  },
  "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
  "eventTime": "2018-05-24T17:17:44.3039911Z",
  "metadataVersion": "1",
  "dataVersion": "2"
}]
```

## Trigger an event from Blob storage

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's configure the name and key for the storage account, then we create a container, then create and upload a file. Again, use the values for `<storage_account_name>` and `<resource_group_name>` you created earlier.

```
export AZURE_STORAGE_ACCOUNT=<storage_account_name>
export AZURE_STORAGE_KEY="$(az storage account keys list --account-name <storage_account_name> --resource-group <resource_group_name> --query "[0].value" --output tsv)"

az storage container create --name testcontainer

touch testfile.txt
az storage blob upload --file testfile.txt --container-name testcontainer --name testfile.txt
```

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.

```
[{
  "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx/resourceGroups/myrg/providers/Microsoft.Storage/storageAccounts/myblobstorageaccount",
  "subject": "/blobServices/default/containers/testcontainer/blobs/testfile.txt",
  "eventType": "Microsoft.Storage.BlobCreated",
  "eventTime": "2017-08-16T20:33:51.059575Z",
  "id": "4d96b1d4-0001-00b3-58ce-16568c064fab",
  "data": {
    "api": "PutBlockList",
    "clientRequestId": "d65ca2e2-a168-4155-b7a4-2c925c18902f",
    "requestId": "4d96b1d4-0001-00b3-58ce-16568c000000",
    "eTag": "0x8D4E4E61AE038AD",
    "contentType": "text/plain",
    "contentLength": 0,
    "blobType": "BlockBlob",
    "url": "https://myblobstorageaccount.blob.core.windows.net/testcontainer/testblob1.txt",
    "sequencer": "000000000000EB0000000000046199",
    "storageDiagnostics": {
      "batchId": "dffea416-b46e-4613-ac19-0371c0c5e352"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Clean up resources

If you plan to continue working with this storage account and event subscription, do not clean up the resources created in this article. If you do not plan to continue, use the following command to delete the resources you created in this article.

Replace <resource\_group\_name> with the resource group you created above.

```
az group delete --name <resource_group_name>
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about Blob storage Events and what Event Grid can help you do:

- [Reacting to Blob storage events](#)
- [About Event Grid](#)

# Quickstart: Route storage events to web endpoint with PowerShell

11/2/2020 • 4 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use Azure PowerShell to subscribe to Blob storage events, trigger an event, and view the result.

Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

When you're finished, you see that the event data has been sent to the web app.

# Setup

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

This article requires that you're running the latest version of Azure PowerShell. If you need to install or upgrade, see [Install and configure Azure PowerShell](#).

## Sign in to Azure

Sign in to your Azure subscription with the `Connect-AzAccount` command and follow the on-screen directions to authenticate.

```
Connect-AzAccount
```

This example uses `westus2` and stores the selection in a variable for use throughout.

```
$location = "westus2"
```

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the [New-AzResourceGroup](#) command.

The following example creates a resource group named `gridResourceGroup` in the `westus2` location.

```
$resourceGroup = "gridResourceGroup"  
New-AzResourceGroup -Name $resourceGroup -Location $location
```

## Create a storage account

Blob storage events are available in general-purpose v2 storage accounts and Blob storage accounts. **General-purpose v2** storage accounts support all features for all storage services, including Blobs, Files, Queues, and Tables. A **Blob storage account** is a specialized storage account for storing your unstructured data as blobs (objects) in Azure Storage. Blob storage accounts are like general-purpose storage accounts and share all the great durability, availability, scalability, and performance features that you use today including 100% API consistency for block blobs and append blobs. For more information, see [Azure storage account overview](#).

Create a Blob storage account with LRS replication using [New-AzStorageAccount](#), then retrieve the storage account context that defines the storage account to be used. When acting on a storage account, you reference the context instead of repeatedly providing the credentials. This example creates a storage account called `gridstorage` with locally redundant storage (LRS).

### NOTE

Storage account names are in a global name space so you need to append some random characters to the name provided in this script.

```
$storageName = "gridstorage"  
$storageAccount = New-AzStorageAccount -ResourceGroupName $resourceGroup `  
    -Name $storageName `  
    -Location $location `  
    -SkuName Standard_LRS `  
    -Kind BlobStorage `  
    -AccessTier Hot  
  
$ctx = $storageAccount.Context
```

## Create a message endpoint

Before subscribing to the topic, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

Replace `<your-site-name>` with a unique name for your web app. The web app name must be unique because it's part of the DNS entry.

```
$sitename=<your-site-name>

New-AzResourceGroupDeployment ` 
    -ResourceGroupName $resourceGroup ` 
    -TemplateUri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-
viewer/master/azuredeploy.json" ` 
    -siteName $sitename ` 
    -hostingPlanName viewerhost
```

The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: <https://<your-site-name>.azurewebsites.net>

You should see the site with no messages currently displayed.

## Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider. Run the following command:

```
Register-AzResourceProvider -ProviderNamespace Microsoft.EventGrid
```

It may take a moment for the registration to finish. To check the status, run:

```
Get-AzResourceProvider -ProviderNamespace Microsoft.EventGrid
```

When `RegistrationStatus` is `Registered`, you're ready to continue.

## Subscribe to your storage account

You subscribe to a topic to tell Event Grid which events you want to track. The following example subscribes to the storage account you created, and passes the URL from your web app as the endpoint for event notification. The endpoint for your web app must include the suffix `/api/updates/`.

```
$storageId = (Get-AzStorageAccount -ResourceGroupName $resourceGroup -AccountName $storageName).Id
$endpoint="https://$sitename.azurewebsites.net/api/updates"

New-AzEventGridSubscription ` 
    -EventSubscriptionName gridBlobQuickStart ` 
    -Endpoint $endpoint ` 
    -ResourceId $storageId
```

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.



## Azure Event Grid Viewer

### Event Type



Microsoft.EventGrid.SubscriptionValidationEvent

```
[{
  "id": "803f7b19-15d9-4753-aad4-feaf6b535adc",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/gridresourcegroup/providers/microsoft.eventgrid/",
  "subject": "",
  "data": {
    "validationCode": "2F6D57C8-97A3-4073-A4AF-3EF5DE46A8B4"
  },
  "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
  "eventTime": "2018-05-24T17:17:44.3039911Z",
  "metadataVersion": "1",
  "dataVersion": "2"
}]
```

## Trigger an event from Blob storage

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's create a container and an object. Then, let's upload the object into the container.

```
$containerName = "gridcontainer"
New-AzStorageContainer -Name $containerName -Context $ctx

echo $null >> gridTestFile.txt

Set-AzStorageBlobContent -File gridTestFile.txt -Container $containerName -Context $ctx -Blob
gridTestFile.txt
```

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.

```
[{
  "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-
xxxxxxxxxxx/resourceGroups/myrg/providers/Microsoft.Storage/storageAccounts/myblobstorageaccount",
  "subject": "/blobServices/default/containers/gridcontainer/blobs/gridTestFile.txt",
  "eventType": "Microsoft.Storage.BlobCreated",
  "eventTime": "2017-08-16T20:33:51.059575Z",
  "id": "4d96b1d4-0001-00b3-58ce-16568c064fab",
  "data": {
    "api": "PutBlockList",
    "clientRequestId": "Azure-Storage-PowerShell-d65ca2e2-a168-4155-b7a4-2c925c18902f",
    "requestId": "4d96b1d4-0001-00b3-58ce-16568c000000",
    "eTag": "0x8D4E4E61AE038AD",
    "contentType": "application/octet-stream",
    "contentLength": 0,
    "blobType": "BlockBlob",
    "url": "https://myblobstorageaccount.blob.core.windows.net/gridcontainer/gridTestFile.txt",
    "sequencer": "000000000000EB0000000000046199",
    "storageDiagnostics": {
      "batchId": "dffea416-b46e-4613-ac19-0371c0c5e352"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Clean up resources

If you plan to continue working with this storage account and event subscription, don't clean up the resources created in this article. If you don't plan to continue, use the following command to delete the resources you created in this article.

```
Remove-AzResourceGroup -Name $resourceGroup
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about Blob storage Events and what Event Grid can help you do:

- [Reacting to Blob storage events](#)
- [About Event Grid](#)

# Quickstart: Route Blob storage events to web endpoint by using an ARM template

4/27/2021 • 4 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use an Azure Resource Manager template (ARM template) to create a Blob storage account, subscribe to events for that blob storage, and trigger an event to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

An [ARM template](#) is a JavaScript Object Notation (JSON) file that defines the infrastructure and configuration for your project. The template uses declarative syntax. In declarative syntax, you describe your intended deployment without writing the sequence of programming commands to create the deployment.

If your environment meets the prerequisites and you're familiar with using ARM templates, select the **Deploy to Azure** button. The template will open in the Azure portal.

 [Deploy to Azure](#)

## Prerequisites

If you don't have an Azure subscription, create a [free account](#) before you begin.

### Create a message endpoint

Before subscribing to the events for the Blob storage, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

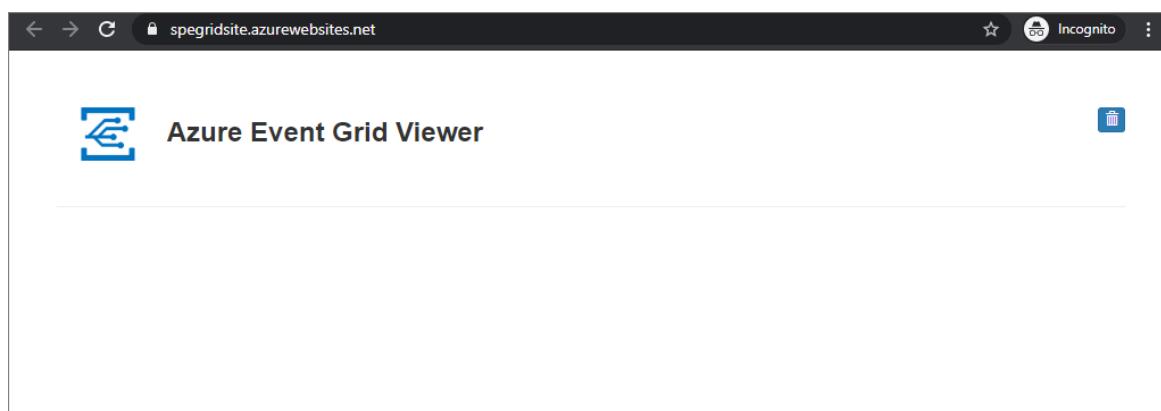
1. Select **Deploy to Azure** to deploy the solution to your subscription. In the Azure portal, provide values for the parameters.

[Deploy to Azure](#)

2. The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to:

`https://<your-site-name>.azurewebsites.net`

3. You see the site but no events have been posted to it yet.



## Review the template

The template used in this quickstart is from [Azure Quickstart Templates](#).

```
{  
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",  
    "contentVersion": "1.0.0.0",  
    "parameters": {  
        "storageName": {  
            "type": "string",  
            "defaultValue": "[concat('storage', uniqueString(resourceGroup().id))]",  
            "metadata": {  
                "description": "Provide a unique name for the Blob Storage account."  
            }  
        },  
        "location": {  
            "type": "string",  
            "defaultValue": "[resourceGroup().location]",  
            "metadata": {  
                "description": "Provide a location for the Blob Storage account that supports Event Grid."  
            }  
        },  
        "eventSubName": {  
            "type": "string",  
            "defaultValue": "subToStorage",  
            "metadata": {  
                "description": "Provide a name for the Event Grid subscription."  
            }  
        },  
        "endpoint": {  
            "type": "string",  
            "metadata": {  
                "description": "Provide the URL for the WebHook to receive events. Create your own endpoint for  
events."  
            }  
        },  
        "systemTopicName": {  
            "type": "String",  
            "defaultValue": "mystoragesystemtopic",  
            "metadata": {  
                "description": "Provide a name for the system topic."  
            }  
        }  
    },  
    "resources": [  
        {  
            "type": "Microsoft.Storage/storageAccounts",  
            "apiVersion": "2019-06-01",  
            "name": "[parameters('storageName')]",  
            "location": "[parameters('location')]",  
            "sku": {  
                "name": "Standard_LRS"  
            },  
            "kind": "StorageV2",  
            "properties": {  
                "accessTier": "Hot"  
            }  
        },  
        {  
            "type": "Microsoft.EventGrid/systemTopics",  
            "apiVersion": "2020-04-01-preview",  
            "name": "[parameters('systemTopicName')]",  
            "location": "[parameters('location')]",  
            "dependsOn": [  
                "[parameters('storageName')]"  
            ],  
            "properties": {  
                "topicFilter": "  
                    {  
                        \"type\": \"blob\",  
                        \"blobContainerNames\": \"\",  
                        \"blobPrefix\": \"\",  
                        \"blobStatus\": \"active\"  
                    }  
                "  
            }  
        }  
    ]  
}
```

```

        "source": "[resourceId('Microsoft.Storage/storageAccounts', parameters('storageName'))]",
        "topicType": "Microsoft.Storage.StorageAccounts"
    },
},
{
    "type": "Microsoft.EventGrid/systemTopics/eventSubscriptions",
    "apiVersion": "2020-04-01-preview",
    "name": "[concat(parameters('systemTopicName'), '/', parameters('eventSubName'))]",
    "dependsOn": [
        "[resourceId('Microsoft.EventGrid/systemTopics', parameters('systemTopicName'))]"
    ],
    "properties": {
        "destination": {
            "properties": {
                "endpointUrl": "[parameters('endpoint')]"
            },
            "endpointType": "WebHook"
        },
        "filter": {
            "includedEventTypes": [
                "Microsoft.Storage.BlobCreated",
                "Microsoft.Storage.BlobDeleted"
            ]
        }
    }
}
]
}

```

Two Azure resources are defined in the template:

- **Microsoft.Storage/storageAccounts**: create an Azure Storage account.
- **Microsoft.EventGrid/systemTopics**: create a system topic with the specified name for the storage account.
- **Microsoft.EventGrid/systemTopics/eventSubscriptions**: create an Azure Event Grid subscription for the system topic.

## Deploy the template

1. Select the following link to sign in to Azure and open a template. The template creates a key vault and a secret.



2. Specify the **endpoint**: provide the URL of your web app and add `api/updates` to the home page URL.

3. Select **Purchase** to deploy the template.

The Azure portal is used here to deploy the template. You can also use the Azure PowerShell, Azure CLI, and REST API. To learn other deployment methods, see [Deploy templates](#).

### NOTE

You can find more Azure Event Grid template samples [here](#).

## Validate the deployment

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.

Event Type	Subject
 Microsoft.EventGrid.SubscriptionValidationEvent	<pre>[{"id": "d48934f6-e17a-4fc4-9697-510ed0c7dd63", "topic": "/subscriptions/{subscription-id}/resourceGroups/eventgroup/providers/Microsoft.Storage/storageAccounts/{storage-account}", "subject": "", "data": { "validationCode": "569F8F44-A944-4840-BF8C-1600FAC184D2", "validationUrl": "https://rp-westcentralus.eventgrid.azure.net/eventsubscriptions/demosubscription/validate?i=2018-06-25T22:23:55.3663623Z&amp;apiVersion=2018-05-01-preview&amp;token=LA9LJwgfvvrbFtci7GLh%2fLX29Dm3l2ukVDiamV2QNKI%3d", }, "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent", "eventTime": "2018-06-25T22:23:55.5405971Z", "metadataVersion": "1", "dataVersion": "2"}]</pre>

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint.

You trigger an event for the Blob storage by uploading a file. The file doesn't need any specific content. The articles assumes you have a file named testfile.txt, but you can use any file.

When you upload the file to the Azure Blob storage, Event Grid sends a message to the endpoint you configured when subscribing. The message is in the JSON format and it contains an array with one or more events. In the following example, the JSON message contains an array with one event. View your web app and notice that a blob created event was received.

## Clean up resources

When no longer needed, [delete the resource group](#).

## Next steps

For more information about Azure Resource Manager templates, see the following articles:

- [Azure Resource Manager documentation](#)
- [Define resources in Azure Resource Manager templates](#)
- [Azure Quickstart templates](#)
- [Azure Event Grid templates](#).

# Quickstart: Route custom events to web endpoint with the Azure portal and Event Grid

4/22/2021 • 7 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure portal to create a custom topic, subscribe to the custom topic, and trigger the event to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

## Prerequisites

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

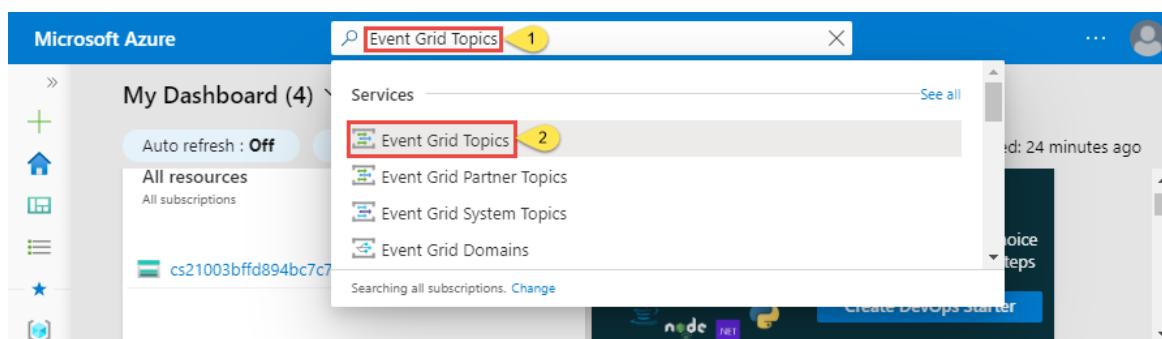
1. Select **Subscriptions** on the left menu.
2. Select the subscription you're using for Event Grid.
3. On the left menu, under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

## Create a custom topic

An event grid topic provides a user-defined endpoint that you post your events to.

1. Sign in to [Azure portal](#).
2. In the search bar at the topic, type **Event Grid Topics**, and then select **Event Grid Topics** from the drop-down list.



3. On the **Event Grid Topics** page, select **+ Add** on the toolbar.

Dashboard >

## Event Grid Topics

Default Directory

[+ Add](#) [Manage view](#) [Refresh](#) [Export to CSV](#) | [Assign tags](#) | [Feedback](#) [Leave preview](#)

Filter by name... Subscription == all Resource group == all Location == all Add filter

Showing 0 to 0 of 0 records.

No grouping

Name ↑↓	Type ↑↓	Resource group ↑↓	Location ↑↓	Subscription ↑↓
---------	---------	-------------------	-------------	-----------------



No Event Grid topics to display

Try changing your filters if you don't see what you're looking for.

[Learn more](#)

[Create Event Grid topics](#)

4. On the **Create Topic** page, follow these steps:

- a. Select your **Azure subscription**.
- b. Select an existing resource group or select **Create new**, and enter a **name** for the **resource group**.
- c. Provide a unique **name** for the custom topic. The topic name must be unique because it's represented by a DNS entry. Don't use the name shown in the image. Instead, create your own name - it must be between 3-50 characters and contain only values a-z, A-Z, 0-9, and "-".
- d. Select a **location** for the event grid topic.
- e. Select **Review + create** at the bottom of the page.

## Create Topic

Event Grid

[X](#)

Basics   Advanced   Tags   Review + create

**PROJECT DETAILS**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Visual Studio Ultimate with MSDN
Resource group *	(New) contosoegridrg
	<a href="#">Create new</a>

**TOPIC DETAILS**

Enter required settings for this topic.

Name *	contosocustomtopic
Location *	East US

---

[Review + create](#)   [< Previous](#)   [Next: Advanced >](#)

- f. On the **Review + create** tab of the **Create topic** page, select **Create**.

Dashboard > Event Grid Topics >

## Create Topic

Event Grid

[X](#)

Validation succeeded.

Basics   Advanced   Tags   **Review + create**

Event Grid Topic  
by Microsoft

**Basics**

Name	contosocustomtopic
Subscription	Visual Studio Ultimate with MSDN
Resource group	contosoegridrg
Location	East US

**Schema**

Event Schema	Event Grid Schema
--------------	-------------------

**Identity**

Enable system assigned identity	Disabled
---------------------------------	----------

---

[Create](#)   [< Previous](#)   [Next >](#)

5. After the deployment succeeds, type **Event Grid Topics** in the search bar again, and select **Event Grid Topics** from the drop-down list as you did before.

6. Select the topic you created from the list.

The screenshot shows the 'Event Grid Topics' list page in the Azure portal. At the top, there are filters for 'Subscription == all', 'Resource group == all', 'Location == all', and a 'Filter by name...' input field. Below the filters, it says 'Showing 1 to 1 of 1 records.' A table lists one topic: 'contosocustomtopic' (Event Grid Topic), which is part of the 'contosoegridrg' resource group, located in 'East US', and associated with 'Visual Studio Ultimate with MSDN'. The 'Name' column has an upward arrow, and the 'Subscription' column has a downward arrow.

7. You see the **Event Grid Topic** page for your topic. Keep this page open. You use it later in the quickstart.

The screenshot shows the 'contosocustomtopic' details page in the Azure portal. On the left, a sidebar lists navigation options: Overview, Activity log, Access control (IAM), Tags, Settings (with sub-options like Access keys, Networking, Identity, Locks, and Export template), Monitoring (with sub-options like Alerts, Metrics, Diagnostic settings, and Logs), and Support + troubleshooting (with sub-option New support request). The main content area shows the topic's properties: Resource group (contosoegridrg), Status (Active), Location (East US), Topic Endpoint (https://contosocustomtopic.eastus-1.eventgrid.azure.net/api/eve...), Pricing tier (Basic), Subscription (Visual Studio Ultimate with MSDN), Subscription ID (<Your Azure subscription ID>), and Tags (Click here to add tags). Below the properties is a large icon of a computer monitor with circuit board elements. A message at the bottom states 'No Event Subscriptions Found!' and provides instructions for enabling event-based programming. A 'Create one' button is at the bottom right.

## Create a message endpoint

Before you create a subscription for the custom topic, create an endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

1. In the article page, select **Deploy to Azure** to deploy the solution to your subscription. In the Azure portal, provide values for the parameters.

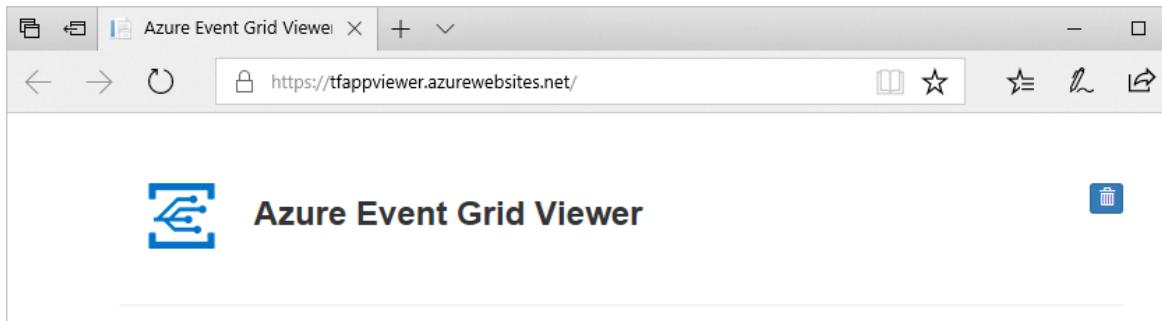


2. The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to:

`https://<your-site-name>.azurewebsites.net`

If the deployment fails, check the error message. It may be because the web site name is already taken. Deploy the template again and choose a different name for the site.

3. You see the site but no events have been posted to it yet.



## Subscribe to custom topic

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

1. Now, on the **Event Grid Topic** page for your custom topic, select **+ Event Subscription** on the toolbar.

A screenshot of the Azure portal showing the "Event Grid Topic" blade for a topic named "contosocustomtopic". The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Tags, Settings (Access keys, Networking, Identity, Locks, Export template), Monitoring (Alerts, Metrics, Diagnostic settings, Logs), and Help. The main content area has a toolbar with "Event Subscription" (highlighted with a red box), Delete, Refresh, and a search bar. Below the toolbar is a section titled "Essentials" featuring a "Create one" button, a "No Event Subscriptions Found!" message, and a callout about enabling event-based programming. A large "Event Grid" icon is prominently displayed.

2. On the **Create Event Subscription** page, follow these steps:

- Enter a **name** for the event subscription.
- Select **Web Hook** for the **Endpoint type**.
- Choose **Select an endpoint**.

# Create Event Subscription



Event Grid

Basic    Filters    Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

## EVENT SUBSCRIPTION DETAILS

Name

1

contosotopicsubscription



Event Schema

Event Grid Schema



## TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type

Event Grid Topic

Source Resource

contosocustomtopic

## EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

Add Event Type

## ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type

2

Web Hook (change)

Endpoint

3

Select an endpoint

**Create**

- d. For the web hook endpoint, provide the URL of your web app and add `api/updates` to the home page URL. Select **Confirm Selection**.

## Select Web Hook



Event Grid

Subscriber Endpoint \*

`https://contosoeventviewer.azurewebsites.net/api/updates`



**Confirm Selection**

- e. Back on the **Create Event Subscription** page, select **Create**.

3. 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.

The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a header with the title "Azure Event Grid Viewer". Below the header, there are two columns: "Event Type" and "Subject". Under "Event Type", there's a small blue square icon followed by the text "Microsoft.EventGrid.SubscriptionValidationEvent". In the "Subject" column, there is a long URL starting with "https://rp-eastus.eventgrid.azure.net/events/...". Below these columns is a large text area containing JSON event data. The JSON is as follows:

```
[{
  "id": "<Event ID>",
  "topic": "/subscriptions/<Subscription ID>",
  "subject": "",
  "data": {
    "validationCode": "<Validation code>",
    "validationUrl": "https://rp-eastus.eventgrid.azure.net/events/...validate?id=<Validation ID>&t=2019-03-28T01:03:57.087990Z&apiVersion=2018-09-15-preview&token=<Token>",
    "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
    "eventTime": "2019-03-28T01:03:57.087990Z",
    "metadataVersion": "1",
    "dataVersion": "2"
  }
}]
```

## Send an event to your topic

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. Use either Azure CLI or PowerShell to send a test event to your custom topic. Typically, an application or Azure service would send the event data.

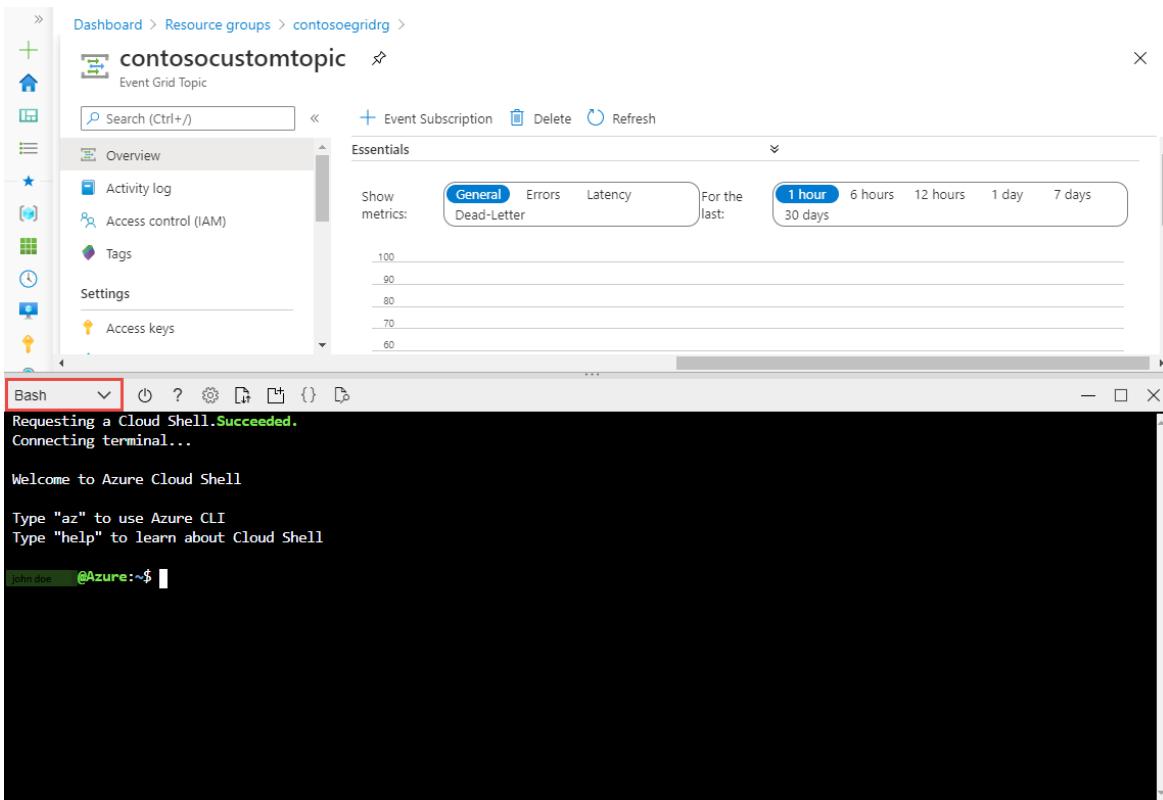
The first example uses Azure CLI. It gets the URL and key for the custom topic, and sample event data. Use your custom topic name for <topic name>. It creates sample event data. The `data` element of the JSON is the payload of your event. Any well-formed JSON can go in this field. You can also use the subject field for advanced routing and filtering. CURL is a utility that sends HTTP requests.

### Azure CLI

1. In the Azure portal, select **Cloud Shell**. The Cloud Shell opens in the bottom pane of the web browser.

The screenshot shows the Azure portal interface with the Cloud Shell integrated. At the top, there's a navigation bar with "Microsoft Azure" and a search bar. A yellow arrow points to the Cloud Shell icon (a terminal window) in the top-right corner of the navigation bar. The main area shows the "contosocustomtopic" Event Grid Topic overview page. On the left, there's a sidebar with navigation links like "Dashboard", "Resource groups", and "contosocustomtopic". The "Overview" tab is selected. The main content area displays metrics for the topic, including a chart showing event counts over time and a table of event types and their counts. At the bottom, there's a "Event Types" table with one row visible: "contosotopicsubs..." under "Name" and "WebHook" under "Endpoint".

2. Select **Bash** in the top-left corner of the Cloud Shell window.



- Run the following command to get the **endpoint** for the topic: After you copy and paste the command, update the **topic name** and **resource group name** before you run the command. You'll publish sample events to this topic endpoint.

```
endpoint=$(az eventgrid topic show --name <topic name> -g <resource group name> --query "endpoint" --output tsv)
```

- Run the following command to get the **key** for the custom topic: After you copy and paste the command, update the **topic name** and **resource group name** before you run the command. It's the primary key of the Event Grid topic. To get this key from the Azure portal, switch to the **Access keys** tab of the **Event Grid Topic** page. To be able post an event to a custom topic, you need the access key.

```
key=$(az eventgrid topic key list --name <topic name> -g <resource group name> --query "key1" --output tsv)
```

- Copy the following statement with the event definition, and press **ENTER**.

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

- Run the following **Curl** command to post the event: In the command, **aeg-sas-key** header is set to the access key you got earlier.

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

## Azure PowerShell

The second example uses PowerShell to do similar steps.

- In the Azure portal, select **Cloud Shell** (alternatively go to <https://shell.azure.com/>). The Cloud Shell opens in the bottom pane of the web browser.

The screenshot shows the Azure Event Grid Topic 'contosocustomtopic' dashboard. The left sidebar contains navigation links for Activity log, Access control (IAM), Tags, Settings (Access keys, Networking, Identity, Locks, Export template), Monitoring (Alerts, Metrics, Diagnostic settings, Logs), and Support + troubleshooting (New support request). The main area has tabs for Overview, Event Subscription, Delete, and Refresh. Under Overview, there's a metrics chart showing General, Errors, and Latency for the last 1 hour, 6 hours, 12 hours, 1 day, or 7 days. Below the chart is a timeline from 9:45 PM to 10:15 PM UTC-04:00 showing event types: Published Events (Sum) contosocustomtopic, Publish Failed Event... contosocustomtopic, Matched Events (Sum) contosocustomtopic, Delivery Failed Even... contosocustomtopic, and Dropped Events (Sum) contosocustomtopic. A search bar at the bottom allows filtering items.

2. In the Cloud Shell, select **PowerShell** in the top-left corner of the Cloud Shell window. See the sample Cloud Shell window image in the Azure CLI section.
3. Set the following variables. After you copy and paste each command, update the **topic name** and **resource group name** before you run the command:

#### Resource group:

```
$resourceGroupName = "<resource group name>"
```

#### Event Grid topic name:

```
$topicName = "<topic name>"
```

4. Run the following commands to get the **endpoint** and the **keys** for the topic:

```
$endpoint = (Get-AzEventGridTopic -ResourceGroupName $resourceGroupName -Name $topicName).Endpoint
$keys = Get-AzEventGridTopicKey -ResourceGroupName $resourceGroupName -Name $topicName
```

5. Prepare the event. Copy and run the statements in the Cloud Shell window.

```

$eventID = Get-Random 99999

#Date format should be SortableDateTimePattern (ISO 8601)
$eventDate = Get-Date -Format s

#Construct body using Hashtable
$htbody = @{
    id= $eventID
    eventType="recordInserted"
    subject="myapp/vehicles/motorcycles"
    eventTime= $eventDate
    data= @{
        make="Ducati"
        model="Monster"
    }
    dataVersion="1.0"
}

#Use ConvertTo-Json to convert event body from Hashtable to JSON Object
#Append square brackets to the converted JSON payload since they are expected in the event's JSON
payload syntax
$body = "[`n+(ConvertTo-Json $htbody)+"]"

```

## 6. Use the **Invoke-WebRequest** cmdlet to send the event.

```
Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}
```

### Verify in the Event Grid Viewer

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.

Event Type	Subject
recordInserted	myapp/vehicles/motorcycles
<pre>{   "subject": "myapp/vehicles/motorcycles",   "id": 49042,   "data": {     "make": "Ducati",     "model": "Monster"   },   "eventType": "recordInserted",   "dataVersion": "1.0",   "metadataVersion": "1",   "eventTime": "2020-06-17T02:56:14",   "topic": "/subscriptions/&lt;Azure subscription ID&gt;/resourceGroups/contosoegridrg/providers/Microsoft.EventGrid/topics/contosocustomtopic" }</pre>	
recordInserted	myapp/vehicles/motorcycles
Microsoft.EventGrid.SubscriptionValidationEvent	

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this article. Otherwise, delete the resources you created in this article.

1. Select **Resource Groups** on the left menu. If you don't see it on the left menu, select **All Services** on the left menu, and select **Resource Groups**.

The screenshot shows the Azure Resource Groups page. On the left, there's a sidebar with options like 'Create a resource', 'Home', 'Dashboard', 'All services', and 'FAVORITES' which includes 'Resource groups' (which is selected and highlighted with a red box). The main area is titled 'Resource groups' and shows a table with two items:

NAME	SUBSCRIPTION	LOCATION
cloud-shell-storage-eastus	Visual Studio Ultimate with MSDN	East US
myeventgrid	Visual Studio Ultimate with MSDN	East US

2. Select the resource group to launch the **Resource Group** page.
3. Select **Delete resource group** on the toolbar.
4. Confirm deletion by entering the name of the resource group, and select **Delete**.

The other resource group you see in the image was created and used by the Cloud Shell window. Delete it if you don't plan to use the Cloud Shell window later.

## Next steps

Now that you know how to create custom topics and event subscriptions, learn more about what Event Grid can help you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

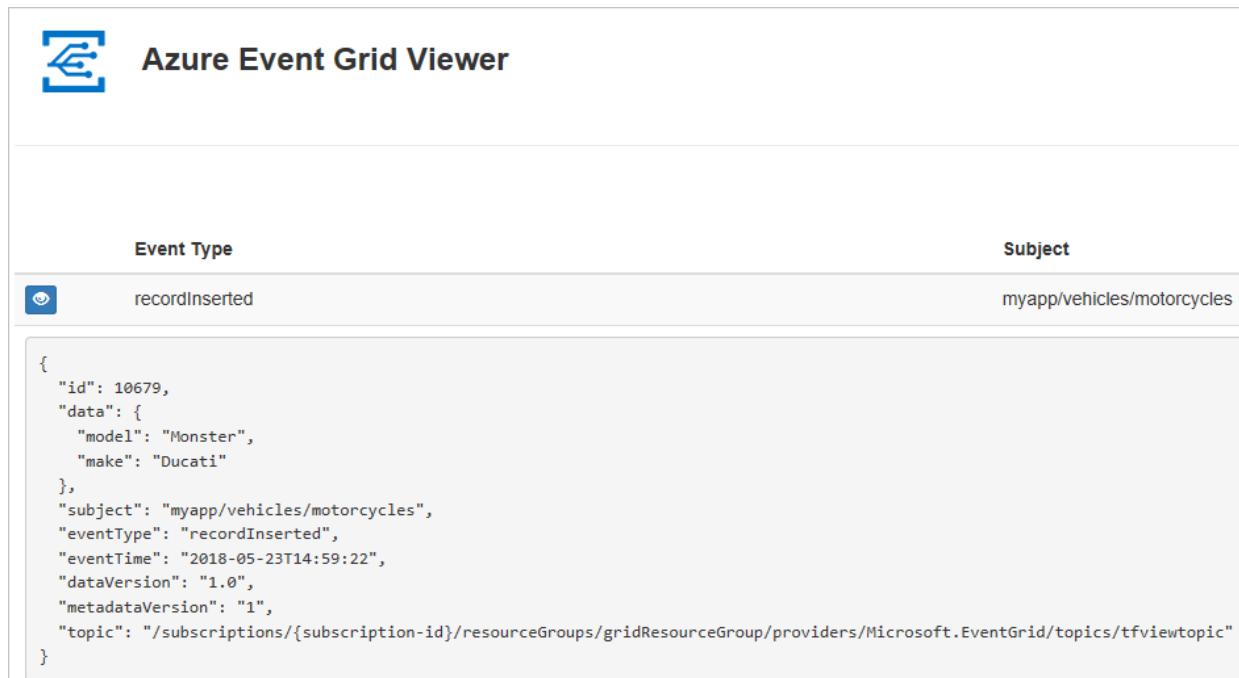
# Quickstart: Route custom events to web endpoint with Azure CLI and Event Grid

4/22/2021 • 5 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure CLI to create a custom topic, subscribe to the custom topic, and trigger the event to view the result.

Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

When you're finished, you see that the event data has been sent to the web app.



The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a blue header bar with the title "Azure Event Grid Viewer". Below the header, there's a table with two columns: "Event Type" and "Subject". Under "Event Type", there's a small icon of a document with a plus sign and the text "recordInserted". Under "Subject", there's the URL "myapp/vehicles/motorcycles". Below the table, the raw JSON event data is displayed:

```
{  
  "id": 10679,  
  "data": {  
    "model": "Monster",  
    "make": "Ducati"  
  },  
  "subject": "myapp/vehicles/motorcycles",  
  "eventType": "recordInserted",  
  "eventTime": "2018-05-23T14:59:22",  
  "dataVersion": "1.0",  
  "metadataVersion": "1",  
  "topic": "/subscriptions/{subscription-id}/resourceGroups/gridResourceGroup/providers/Microsoft.EventGrid/topics/tfviewtopic"  
}
```

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires version 2.0.70 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the [az group create](#) command.

The following example creates a resource group named *gridResourceGroup* in the *westus2* location.

```
az group create --name gridResourceGroup --location westus2
```

## Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

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

It might take a moment for the registration to finish. To check the status, run:

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

When `registrationState` is `Registered`, you're ready to continue.

## Create a custom topic

An event grid topic provides a user-defined endpoint that you post your events to. The following example creates the custom topic in your resource group. Replace `<your-topic-name>` with a unique name for your topic. The custom topic name must be unique because it's part of the DNS entry. Additionally, it must be between 3-50 characters and contain only values a-z, A-Z, 0-9, and "-".

```
topicname=<your-topic-name>
az eventgrid topic create --name $topicname -l westus2 -g gridResourceGroup
```

## Create a message endpoint

Before subscribing to the custom topic, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

Replace `<your-site-name>` with a unique name for your web app. The web app name must be unique because it's part of the DNS entry.

```
sitename=<your-site-name>

az deployment group create \
--resource-group gridResourceGroup \
--template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-
viewer/master/azuredeploy.json" \
--parameters siteName=$sitename hostingPlanName=viewerhost
```

The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: <https://<your-site-name>.azurewebsites.net>

You should see the site with no messages currently displayed.

## Subscribe to a custom topic

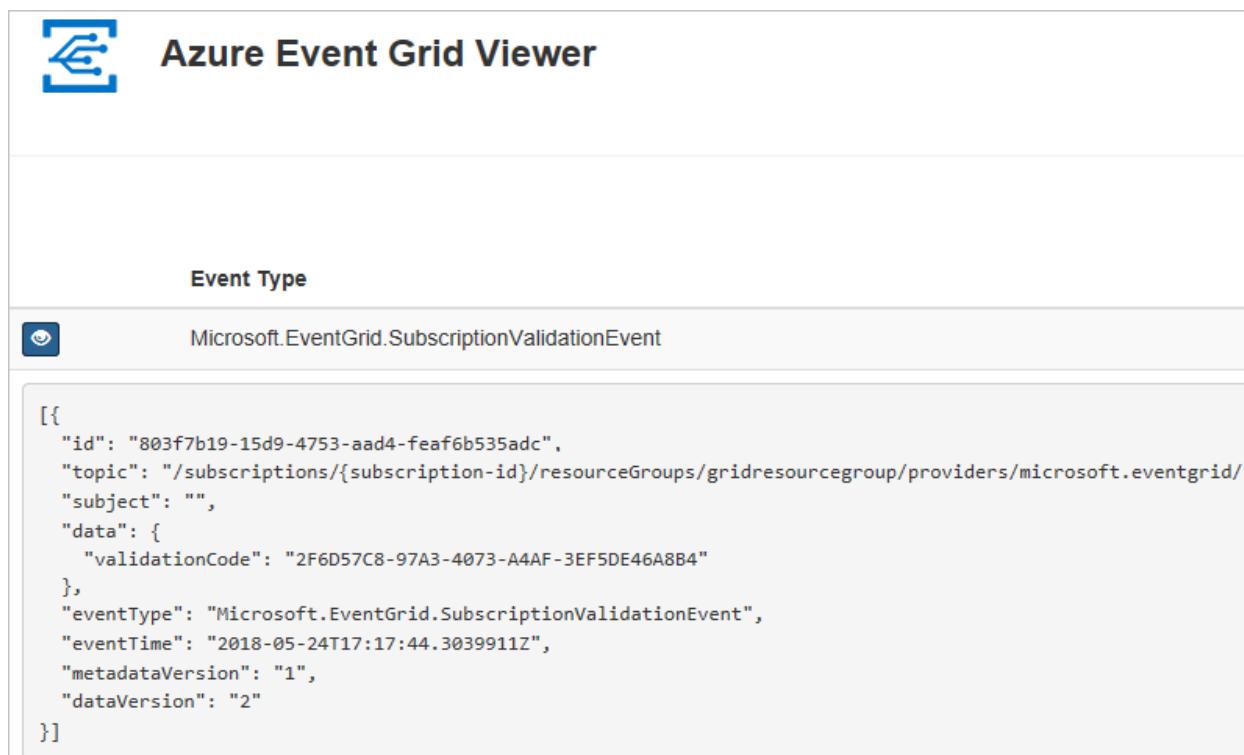
You subscribe to an event grid topic to tell Event Grid which events you want to track and where to send those events. The following example subscribes to the custom topic you created, and passes the URL from your web app as the endpoint for event notification.

The endpoint for your web app must include the suffix `/api/updates/`.

```
endpoint=https://$sitename.azurewebsites.net/api/updates

az eventgrid event-subscription create \
--source-resource-id "/subscriptions/{subscription-id}/resourceGroups/{resource-
group}/providers/Microsoft.EventGrid/topics/$topicname" \
--name demoViewerSub \
--endpoint $endpoint
```

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.



The screenshot shows the Azure Event Grid Viewer interface. At the top, there is a logo and the text "Azure Event Grid Viewer". Below this, there is a table with a single row. The first column contains an eye icon, and the second column contains the text "Microsoft.EventGrid.SubscriptionValidationEvent". Underneath the table, there is a JSON representation of the event data:

```
[{
  "id": "803f7b19-15d9-4753-aad4-feaf6b535adc",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/gridresourcegroup/providers/microsoft.eventgrid/",
  "subject": "",
  "data": {
    "validationCode": "2F6D57C8-97A3-4073-A4AF-3EF5DE46A8B4"
  },
  "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
  "eventTime": "2018-05-24T17:17:44.3039911Z",
  "metadataVersion": "1",
  "dataVersion": "2"
}]
```

## Send an event to your custom topic

Let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's get the URL and key for the custom topic.

```
endpoint=$(az eventgrid topic show --name $topicname -g gridResourceGroup --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name $topicname -g gridResourceGroup --query "key1" --output tsv)
```

To simplify this article, you use sample event data to send to the custom topic. Typically, an application or Azure service would send the event data. The following example creates sample event data:

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

The `data` element of the JSON is the payload of your event. Any well-formed JSON can go in this field. You can also use the subject field for advanced routing and filtering.

CURL is a utility that sends HTTP requests. In this article, use CURL to send the event to the topic.

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

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.

```
[{
  "id": "1807",
  "eventType": "recordInserted",
  "subject": "myapp/vehicles/motorcycles",
  "eventTime": "2017-08-10T21:03:07+00:00",
  "data": {
    "make": "Ducati",
    "model": "Monster"
  },
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventGrid/topics/{topic}"
}]
```

## Clean up resources

If you plan to continue working with this event or the event viewer app, don't clean up the resources created in this article. Otherwise, use the following command to delete the resources you created in this article.

```
az group delete --name gridResourceGroup
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about what Event Grid can help you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

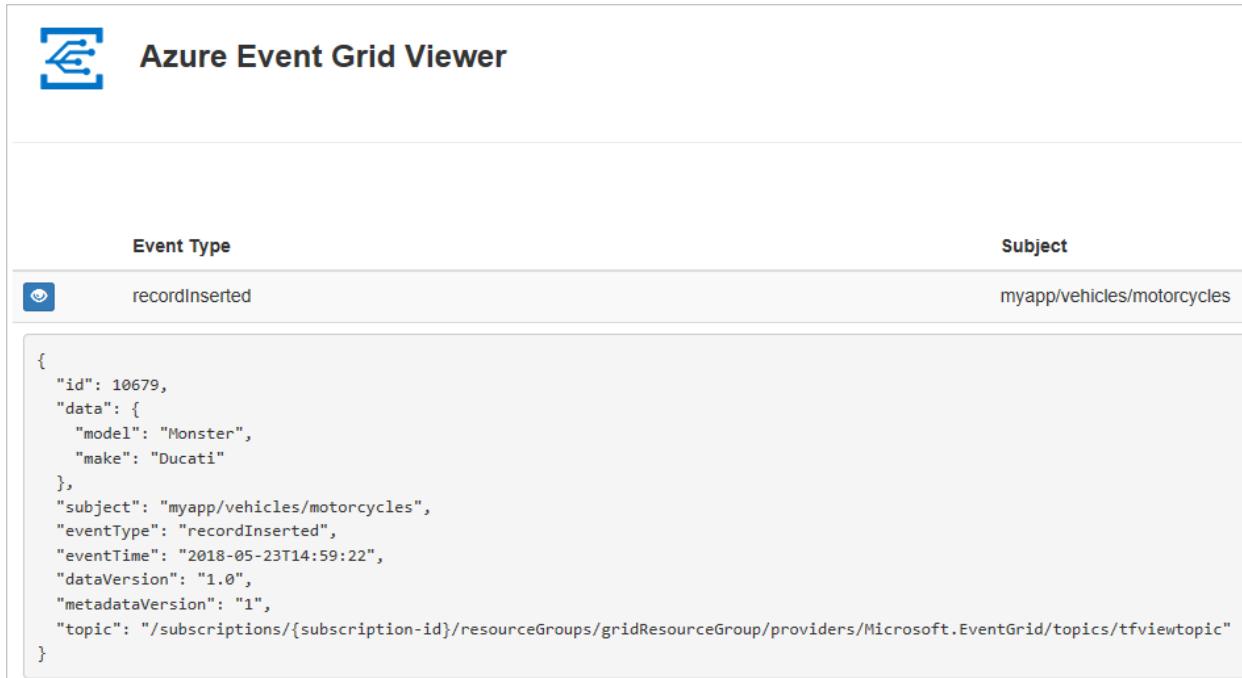
- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

# Quickstart: Route custom events to web endpoint with PowerShell and Event Grid

4/22/2021 • 5 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this article, you use the Azure PowerShell to create a custom topic, subscribe to the topic, and trigger the event to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this article, you send the events to a web app that collects and displays the messages.

When you're finished, you see that the event data has been sent to the web app.



The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a header with the title "Azure Event Grid Viewer" and a small icon. Below the header, there's a table with two columns: "Event Type" and "Subject". Under "Event Type", there's a blue circular icon with a white question mark. Next to it, the text "recordInserted" is displayed. Under "Subject", the text "myapp/vehicles/motorcycles" is shown. Below the table, there's a large code block representing the event payload in JSON format:

```
{  
  "id": 10679,  
  "data": {  
    "model": "Monster",  
    "make": "Ducati"  
  },  
  "subject": "myapp/vehicles/motorcycles",  
  "eventType": "recordInserted",  
  "eventTime": "2018-05-23T14:59:22",  
  "dataVersion": "1.0",  
  "metadataVersion": "1",  
  "topic": "/subscriptions/{subscription-id}/resourceGroups/gridResourceGroup/providers/Microsoft.EventGrid/topics/tfviewtopic"  
}
```

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

This article requires that you are running the latest version of Azure PowerShell. If you need to install or upgrade, see [Install and configure Azure PowerShell](#).

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the [New-AzResourceGroup](#) command.

The following example creates a resource group named *gridResourceGroup* in the *westus2* location.

```
New-AzResourceGroup -Name gridResourceGroup -Location westus2
```

## Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider. Run the following command:

```
Register-AzResourceProvider -ProviderNamespace Microsoft.EventGrid
```

It may take a moment for the registration to finish. To check the status, run:

```
Get-AzResourceProvider -ProviderNamespace Microsoft.EventGrid
```

When `RegistrationStatus` is `Registered`, you're ready to continue.

## Create a custom topic

An event grid topic provides a user-defined endpoint that you post your events to. The following example creates the custom topic in your resource group. Replace `<your-topic-name>` with a unique name for your topic. The topic name must be unique because it's part of the DNS entry. Additionally, it must be between 3-50 characters and contain only values a-z, A-Z, 0-9, and "-".

```
$topicname=<your-topic-name>

New-AzEventGridTopic -ResourceGroupName gridResourceGroup -Location westus2 -Name $topicname
```

## Create a message endpoint

Before subscribing to the topic, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you deploy 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.

Replace `<your-site-name>` with a unique name for your web app. The web app name must be unique because it's part of the DNS entry.

```
$sitename=<your-site-name>

New-AzResourceGroupDeployment ` 
-ResourceGroupName gridResourceGroup ` 
-TemplateUri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-
viewer/master/azuredeploy.json" ` 
-siteName $sitename ` 
-hostingPlanName viewerhost
```

The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to: <https://<your-site-name>.azurewebsites.net>

You should see the site with no messages currently displayed.

## Subscribe to a topic

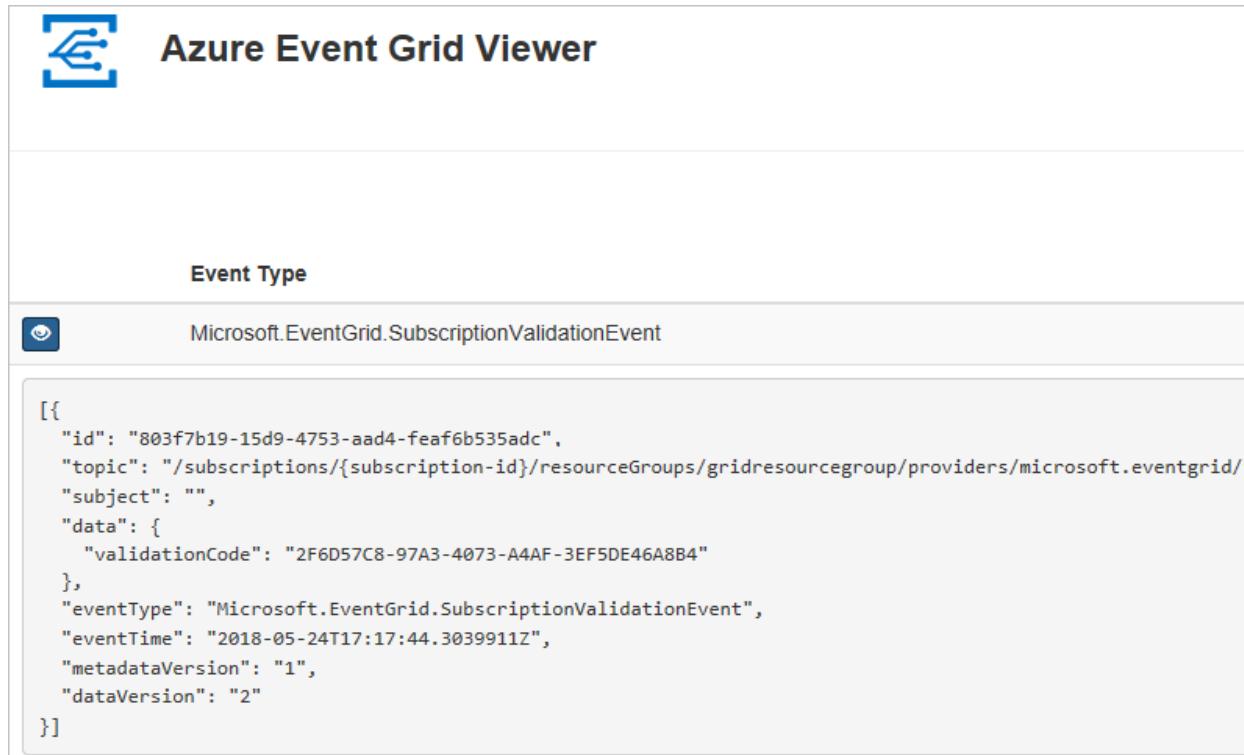
You subscribe to a topic to tell Event Grid which events you want to track and where to send those events. The following example subscribes to the topic you created, and passes the URL from your web app as the endpoint for event notification.

The endpoint for your web app must include the suffix `/api/updates/`.

```
$endpoint="https://$sitename.azurewebsites.net/api/updates"

New-AzEventGridSubscription ` 
    -EventSubscriptionName demoViewerSub ` 
    -Endpoint $endpoint ` 
    -ResourceGroupName gridResourceGroup ` 
    -TopicName $topicname
```

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.



The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a logo and the title "Azure Event Grid Viewer". Below the title, there's a section titled "Event Type" with a blue eye icon. Underneath, it says "Microsoft.EventGrid.SubscriptionValidationEvent". A JSON object is displayed below this:

```
[{"id": "803f7b19-15d9-4753-aad4-feaf6b535adc", "topic": "/subscriptions/{subscription-id}/resourceGroups/gridresourcegroup/providers/microsoft.eventgrid", "subject": "", "data": { "validationCode": "2F6D57C8-97A3-4073-A4AF-3EF5DE46A8B4" }, "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent", "eventTime": "2018-05-24T17:17:44.3039911Z", "metadataVersion": "1", "dataVersion": "2"}]
```

## Send an event to your topic

Let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's get the URL and key for the topic.

```
$endpoint = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name $topicname).Endpoint
$key = Get-AzEventGridTopicKey -ResourceGroupName gridResourceGroup -Name $topicname
```

To simplify this article, let's set up sample event data to send to the custom topic. Typically, an application or Azure service would send the event data. The following example uses Hashtable to construct the event's data `htbody` and then converts it to well-formed JSON payload object `$body`:

```

$eventID = Get-Random 99999

#Date format should be SortableDateTimePattern (ISO 8601)
$eventDate = Get-Date -Format s

#Construct body using Hashtable
$htbody = @{
    id= $eventID
    eventType="recordInserted"
    subject="myapp/vehicles/motorcycles"
    eventTime= $eventDate
    data= @{
        make="Ducati"
        model="Monster"
    }
    dataVersion="1.0"
}

#Use ConvertTo-Json to convert event body from Hashtable to JSON Object
#Append square brackets to the converted JSON payload since they are expected in the event's JSON payload
syntax
$body = "[`n+(ConvertTo-Json $htbody)+`n]"

```

If you view `$body`, you see the full event. The `data` element of the JSON is the payload of your event. Any well-formed JSON can go in this field. You can also use the `subject` field for advanced routing and filtering.

Now, send the event to your topic.

```
Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}
```

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. View your web app to see the event you just sent.

```
[{
    "id": "1807",
    "eventType": "recordInserted",
    "subject": "myapp/vehicles/motorcycles",
    "eventTime": "2018-01-25T15:58:13",
    "data": {
        "make": "Ducati",
        "model": "Monster"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1",
    "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventGrid/topics/{topic}"
}]
```

## Clean up resources

If you plan to continue working with this event or the event viewer app, don't clean up the resources created in this article. Otherwise, use the following command to delete the resources you created in this article.

```
Remove-AzResourceGroup -Name gridResourceGroup
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about what Event Grid can help

you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

# Quickstart: Route custom events to an Azure Function with Event Grid

4/22/2021 • 7 minutes to read • [Edit Online](#)

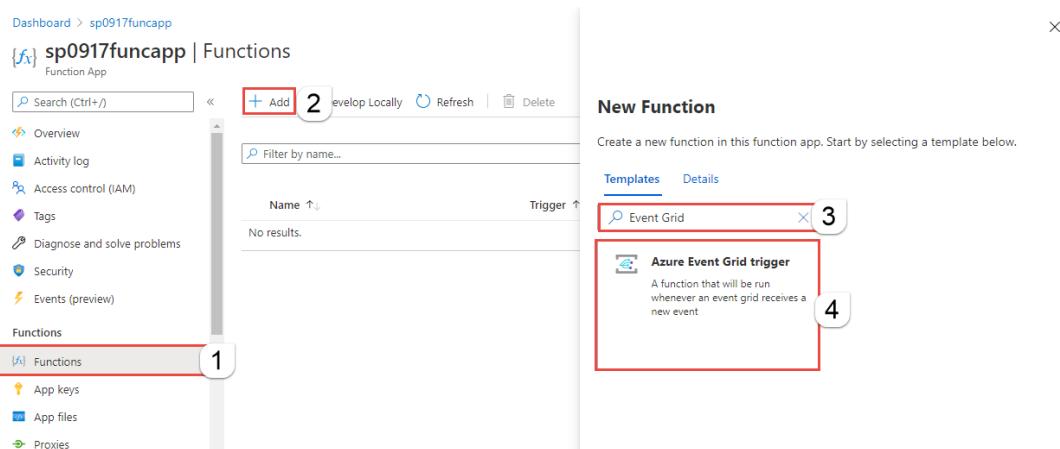
Azure Event Grid is an eventing service for the cloud. Azure Functions is one of the supported event handlers. In this article, you use the Azure portal to create a custom topic, subscribe to the custom topic, and trigger the event to view the result. You send the events to an Azure Function.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create Azure Function

Before subscribing to the custom topic, create a function to handle the events.

1. Create a function app using instructions from [Create a function app](#).
2. Create a function using the **Event Grid Trigger**. Select If this is your first time using this trigger, you may have to click 'Install' to install the extension.
  - a. On the **Function App** page, select **Functions** on the left menu, search for **Event Grid** in templates, and then select **Azure Event Grid trigger**.



3. On the **New Function** page, enter a name for the function, and select **Create Function**.

## New Function

Create a new function in this function app. Start by selecting a template below.

[Templates](#) [Details](#)

New Function\*

HandleGridEventsFunc

**Create Function**

4. Use the **Code + Test** page to see the existing code for the function and update it.

# Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

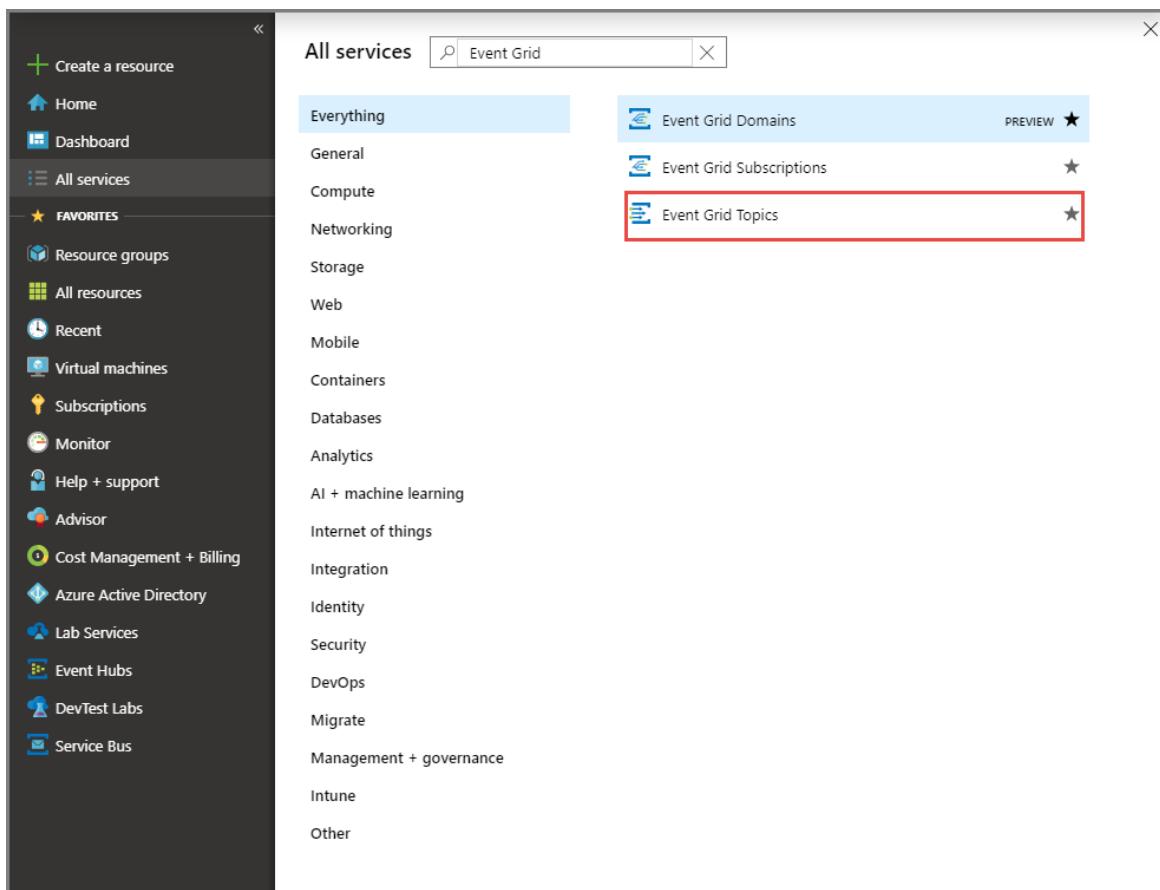
1. Select **Subscriptions** on the left menu.
2. Select the subscription you're using for Event Grid.
3. On the left menu, under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

## Create a custom topic

An event grid topic provides a user-defined endpoint that you post your events to.

1. Sign in to [Azure portal](#).
2. Select **All services** on the left navigational menu, search for **Event Grid**, and select **Event Grid Topics**.



3. On the **Event Grid Topics** page, select **+ Add** on the toolbar.

Dashboard > Event Grid Topics

## Event Grid Topics

Default Directory

**+ Add** **Edit columns** **Refresh** **Assign tags**

**Subscriptions:** Visual Studio Ultimate with MSDN – Don't see a subscription? [Open](#)  
[Directory + Subscription settings](#)

Filter by name... All resource groups All locations All tags No grouping

0 items

<input type="checkbox"/> NAME ↑↓	TYPE ↑↓	RESOURCE G... ↑↓	LOCATION ↑↓	SUBSCRIPTI... ↑↓
				

No Event Grid topics to display  
Try changing your filters if you don't see what you're looking for.

**Create Event Grid topics**

4. On the **Create Topic** page, follow these steps:

- a. Provide a unique **name** for the custom topic. The topic name must be unique because it's represented by a DNS entry. Don't use the name shown in the image. Instead, create your own name - it must be between 3-50 characters and contain only values a-z, A-Z, 0-9, and "-".
- b. Select your Azure **subscription**.
- c. Select the same resource group from the previous steps.
- d. Select a **location** for the event grid topic.
- e. Keep the default value **Event Grid Schema** for the **Event Schema** field.

 **Create Topic**

Event Grid

\* Name  
myegridtopic 

\* Subscription  
Visual Studio Ultimate with MSDN 

\* Resource group  
(New) myegridrg   
[Create new](#)

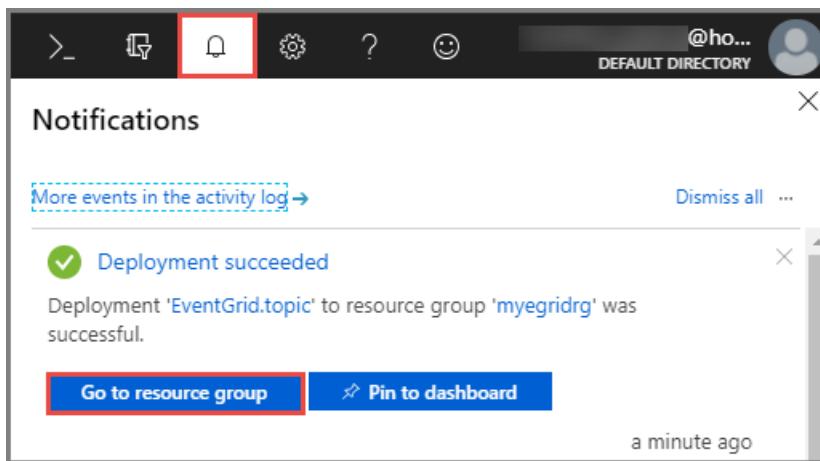
\* Location  
East US 

Event Schema  
Event Grid Schema 

**Create**

f. Select **Create**.

5. After the custom topic has been created, you see the successful notification. Select **Go to resource group**.



6. On the **Resource Group** page, select the event grid topic.

The screenshot shows the Azure Resource Group 'myegridrg' dashboard. On the left, there's a sidebar with navigation links like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Resource costs, Deployments, Policies, Properties, Locks, Export template, Monitoring, Insights (preview), Alerts, Metrics, Diagnostic settings, Advisor recommendations, Support + troubleshooting, and New support request. The main area displays the 'Subscription (change)' as 'Visual Studio Ultimate with MSDN' and 'Subscription ID' as '<Your Azure subscription ID>'. It also shows 'Tags (change)' with a link to 'Click here to add tags'. Below this is a table titled 'myegridtopic' with one item: 'myegridtopic' (Event Grid Topic) located in East US. The table has columns for NAME, TYPE, and LOCATION.

7. You see the **Event Grid Topic** page for your event grid. Keep this page open. You use it later in the quickstart.

The screenshot shows the 'myegridtopic' Event Grid Topic page. The sidebar includes Overview, Activity log, Access control (IAM), Tags, Settings (with Access keys, Locks, and Export template), Monitoring (Metrics), Support + troubleshooting, and New support request. The main area shows 'Resource group (change)' as 'myegridrg', 'Status' as Active, 'Location' as East US, and 'Subscription (change)' as 'Visual Studio Ultimate with MSDN'. It also shows 'Topic Endpoint' as <https://myegridtopic.eastus-1.eventgrid.azure.net/api/events>. Below this is a large blue icon of a circuit board. A message says 'No Event Subscriptions Found!' followed by instructions to enable event-based programming using Event Subscriptions to connect resources. A 'Create one' button is at the bottom.

## Subscribe to custom topic

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

1. Now, on the **Event Grid Topic** page for your custom topic, select **+ Event Subscription** on the toolbar.

The screenshot shows the Azure portal interface for an Event Grid Topic named 'myegridtopic'. On the left, there's a sidebar with links for Overview, Activity log, Access control (IAM), and Tags. The main panel displays basic information: Resource group (myegridrg), Status (Active), Location (East US). At the top right, there are buttons for Event Subscription (highlighted with a red box), Delete, and Refresh.

2. On the **Create Event Subscription** page, follow these steps:

- Enter a **name** for the event subscription.
- Select **Azure Function** for the **Endpoint type**.
- Choose **Select an endpoint**.

The screenshot shows the 'Create Event Subscription' page. The 'Basic' tab is selected. In the 'EVENT SUBSCRIPTION DETAILS' section, the 'Name' field contains 'mynewsubscription' with a green checkmark. The 'Event Schema' dropdown is set to 'Event Grid Schema'. In the 'TOPIC DETAILS' section, 'Topic Type' is 'Event Grid Topic' and 'Topic Resource' is 'connector-test'. In the 'EVENT TYPES' section, there is a 'Filter to Event Types' input and an 'Add Event Type' button. In the 'ENDPOINT DETAILS' section, 'Endpoint Type' is 'Azure Function (change)' and 'Endpoint' is 'Select an endpoint'.

- For the function endpoint, select the Azure Subscription and Resource Group your Function App is in and then select the Function App and function you created earlier. Select **Confirm Selection**.

## Select Azure Function

Event Grid



### Subscription

Azure Messaging PM Playground



### Resource group

babanisa



### Function app \*

eventhandlerapp



### Slot \* ⓘ

Production



### Function \* ⓘ

EventGridTrigger1



- e. This step is optional, but recommended for production scenarios. On the **Create Event Subscription** page, switch to the **Advanced Features** tab, and set values for **Max events per batch** and **Preferred batch size in kilobytes**.

Batching can give you high-throughput. For **Max events per batch**, set maximum number of events that a subscription will include in a batch. Preferred batch size sets the preferred upper bound of batch size in kilo bytes, but can be exceeded if a single event is larger than this threshold.

 **Create Event Subscription**

Event Grid

[Basic](#) [Filters](#) [Additional Features](#)

---

**DEAD-LETTERING**

Save events that cannot be delivered to storage. [Learn more](#)

Enable dead-lettering

**RETRY POLICIES**

Customize how many times and for how long event delivery will be retried. [Learn more](#)

Configure Retry Policies

**EVENT SUBSCRIPTION EXPIRATION TIME**

Set a time at which the event subscription will automatically be deleted.

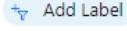
Enable expiration time

**BATCHING**

Batching can be used to improve efficiency at high-throughput. Max events sets the maximum number of events that a subscription will include in a batch. Preferred batch size sets the preferred upper bound of batch size in kb, but can be exceeded if a single event is larger than this threshold. [Learn more](#)

Max events per batch *	1
Preferred batch size in kilobytes *	64

**LABELS**

 Add Label

---

**Create**

- f. On the Create Event Subscription page, select **Create**.

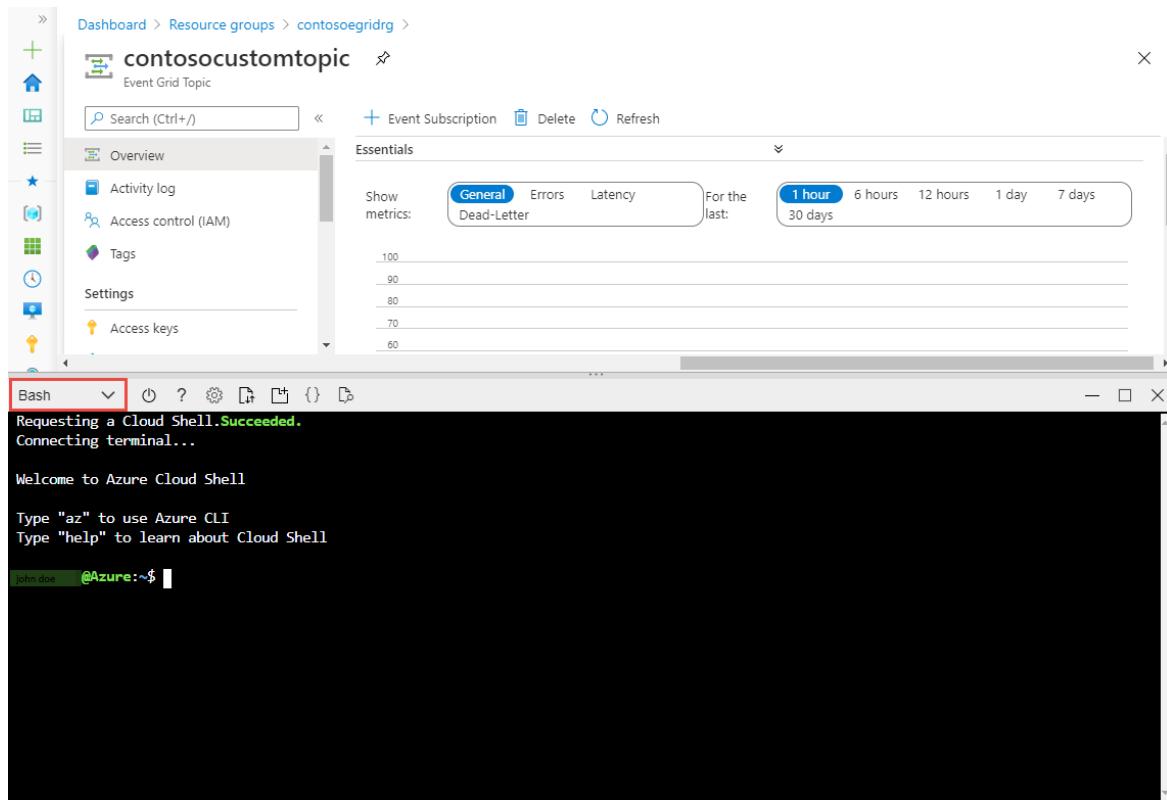
## Send an event to your topic

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. Use either Azure CLI or PowerShell to send a test event to your custom topic. Typically, an application or Azure service would send the event data.

The first example uses Azure CLI. It gets the URL and key for the custom topic, and sample event data. Use your custom topic name for `<topic_name>`. It creates sample event data. The `data` element of the JSON is the payload of your event. Any well-formed JSON can go in this field. You can also use the subject field for advanced routing and filtering. CURL is a utility that sends HTTP requests.

### Azure CLI

1. In the Azure portal, select **Cloud Shell**. Select **Bash** in the top-left corner of the Cloud Shell window.



- Run the following command to get the **endpoint** for the topic: After you copy and paste the command, update the **topic name** and **resource group name** before you run the command.

```
endpoint=$(az eventgrid topic show --name <topic name> -g <resource group name> --query "endpoint" --output tsv)
```

- Run the following command to get the **key** for the custom topic: After you copy and paste the command, update the **topic name** and **resource group name** before you run the command.

```
key=$(az eventgrid topic key list --name <topic name> -g <resource group name> --query "key1" --output tsv)
```

- Copy the following statement with the event definition, and press **ENTER**.

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

- Run the following **Curl** command to post the event:

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

## Azure PowerShell

The second example uses PowerShell to perform similar steps.

- In the Azure portal, select **Cloud Shell** (alternatively go to <https://shell.azure.com/>). Select **PowerShell** in the top-left corner of the Cloud Shell window. See the sample **Cloud Shell** window image in the Azure CLI section.
- Set the following variables. After you copy and paste each command, update the **topic name** and **resource group name** before you run the command:

```
$resourceGroupName = <resource group name>
$topicName = <topic name>
```

3. Run the following commands to get the **endpoint** and the **keys** for the topic:

```
$endpoint = (Get-AzEventGridTopic -ResourceGroupName $resourceGroupName -Name $topicName).Endpoint
$keys = Get-AzEventGridTopicKey -ResourceGroupName $resourceGroupName -Name $topicName
```

4. Prepare the event. Copy and run the statements in the Cloud Shell window.

```
$eventID = Get-Random 99999

#Date format should be SortableDateTimePattern (ISO 8601)
$eventDate = Get-Date -Format s

#Construct body using Hashtable
$htbody = @{
    id= $eventID
    eventType="recordInserted"
    subject="myapp/vehicles/motorcycles"
    eventTime= $eventDate
    data= @{
        make="Ducati"
        model="Monster"
    }
    dataVersion="1.0"
}

#Use ConvertTo-Json to convert event body from Hashtable to JSON Object
#Append square brackets to the converted JSON payload since they are expected in the event's JSON
payload syntax
$body = "[+(ConvertTo-Json $htbody)+]"
```

5. Use the **Invoke-WebRequest** cmdlet to send the event.

```
Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}
```

### Verify in the Event Grid Viewer

You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. Navigate to your Event Grid triggered function and open the logs. You should see a copy of the data payload of the event in the logs. If you don't make sure you open the logs window first, or hit reconnect, and then try sending a test event again.

The screenshot shows the Azure Functions portal interface. On the left, the navigation bar includes 'eventhandlerapp - EventGridTrigger1', 'Function Apps', 'Azure Messaging PM Playground', 'eventhandlerapp', 'Functions', 'EventGridTrigger1', 'Integrate', 'Manage', 'Monitor', 'Proxies', and 'Slots'. The main area shows the 'run.csx' file content:

```

1 # "Microsoft.Azure.EventGrid"
2 using Microsoft.Azure.EventGrid.Models;
3
4 public static void Run(EventGridEvent eventGridEvent, ILogger log)
5 {
6     log.LogInformation(eventGridEvent.Data.ToString());
7 }

```

Below the code editor is a 'Logs' section with the following log entries:

```

2019-11-18T14:43:47.779 [Information] Executing 'Functions:EventGridTrigger1' (Reason='EventGrid trigger fired at 2019-11-18T14:43:47.779Z', Id=3e28733b-2571-4985-ab09-de39ad0e5527)
2019-11-18T14:43:47.779 [Information] {"make": "Ducati", "model": "Monster"}
2019-11-18T14:43:47.779 [Information] Executed 'Functions:EventGridTrigger1' (Succeeded, Id=3e28733b-2571-4985-ab09-de39ad0e5527)

```

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this article. Otherwise, delete the resources you created in this article.

1. Select **Resource Groups** on the left menu. If you don't see it on the left menu, select **All Services** on the left menu, and select **Resource Groups**.
2. Select the resource group to launch the **Resource Group** page.
3. Select **Delete resource group** on the toolbar.
4. Confirm deletion by entering the name of the resource group, and select **Delete**.

The screenshot shows the Azure portal's 'Resource groups' page. The left sidebar has a 'FAVORITES' section with 'Resource groups' highlighted. The main area shows a table of resource groups:

NAME	SUBSCRIPTION	LOCATION	...
cloud-shell-storage-eastus	Visual Studio Ultimate with MSDN	East US	...
myegridrg	Visual Studio Ultimate with MSDN	East US	...

The other resource group you see in the image was created and used by the Cloud Shell window. Delete it if you don't plan to use the Cloud Shell window later.

## Next steps

Now that you know how to create topics and event subscriptions, learn more about what Event Grid can help you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

# Quickstart: Route custom events to Azure Queue storage with Azure CLI and Event Grid

4/22/2021 • 5 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. Azure Queue storage is one of the supported event handlers. In this article, you use the Azure CLI to create a custom topic, subscribe to the custom topic, and trigger the event to view the result. You send the events to the Queue storage.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).  
[Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).
- This article requires version 2.0.56 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.
- If you are using Azure PowerShell on your local machine instead of using Cloud Shell in the Azure portal, ensure that you have Azure PowerShell version 1.1.0 or greater. Download the latest version of Azure PowerShell on your Windows machine from [Azure downloads - Command-line tools](#).

This article gives you commands for using Azure CLI.

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the [az group create](#) command.

The following example creates a resource group named *gridResourceGroup* in the *westus2* location.

```
az group create --name gridResourceGroup --location westus2
```

## Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

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

It might take a moment for the registration to finish. To check the status, run:

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

When `registrationState` is `Registered`, you're ready to continue.

## Create a custom topic

An event grid topic provides a user-defined endpoint that you post your events to. The following example creates the custom topic in your resource group. Replace `<topic_name>` with a unique name for your custom topic. The event grid topic name must be unique because it's represented by a DNS entry.

```
az eventgrid topic create --name <topic_name> -l westus2 -g gridResourceGroup
```

## Create Queue storage

Before subscribing to the custom topic, let's create the endpoint for the event message. You create a Queue storage for collecting the events.

```
storagename=<unique-storage-name>
queuename="eventqueue"

az storage account create -n $storagename -g gridResourceGroup -l westus2 --sku Standard_LRS
az storage queue create --name $queuename --account-name $storagename
```

## Subscribe to a custom topic

You subscribe to a custom topic to tell Event Grid which events you want to track. The following example subscribes to the custom topic you created, and passes the resource ID of the Queue storage for the endpoint. With Azure CLI, you pass the Queue storage ID as the endpoint. The endpoint is in the format:

```
/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.Storage/storageAccounts/<storage-name>/queueservices/default/queues/<queue-name>
```

The following script gets the resource ID of the storage account for the queue. It constructs the ID for the queue storage, and subscribes to an event grid topic. It sets the endpoint type to `storagequeue` and uses the queue ID for the endpoint.

```
storageid=$(az storage account show --name $storagename --resource-group gridResourceGroup --query id --output tsv)
queueid="$storageid/queueservices/default/queues/$queuename"
topicid=$(az eventgrid topic show --name <topic_name> -g gridResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--source-resource-id $topicid \
--name <event_subscription_name> \
--endpoint-type storagequeue \
--endpoint $queueid \
--expiration-date "<yyyy-mm-dd>"
```

The account that creates the event subscription must have write access to the queue storage. Notice that an [expiration date](#) is set for the subscription.

If you use the REST API to create the subscription, you pass the ID of the storage account and the name of the queue as a separate parameter.

```
"destination": {  
    "endpointType": "storagequeue",  
    "properties": {  
        "queueName": "eventqueue",  
        "resourceId": "/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.Storage/storageAccounts/<storage-name>"  
    }  
    ...  
}
```

## Send an event to your custom topic

Let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's get the URL and key for the custom topic. Again, use your custom topic name for <topic\_name>.

```
endpoint=$(az eventgrid topic show --name <topic_name> -g gridResourceGroup --query "endpoint" --output tsv)  
key=$(az eventgrid topic key list --name <topic_name> -g gridResourceGroup --query "key1" --output tsv)
```

To simplify this article, you use sample event data to send to the custom topic. Typically, an application or Azure service would send the event data. CURL is a utility that sends HTTP requests. In this article, use CURL to send the event to the custom topic. The following example sends three events to the event grid topic:

```
for i in 1 2 3  
do  
    event='[ {"id": """$RANDOM""", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles",  
    "eventTime": "'`date +%Y-%m-%dT%H:%M:%S%z`'", "data":{ "make": "Ducati", "model": "Monster"}, "dataVersion":  
    "1.0"} ]'  
    curl -X POST -H "aeg-sas-key: $key" -d "$event" $endpoint  
done
```

Navigate to the Queue storage in the portal, and notice that Event Grid sent those three events to the queue.

Messages		
eventqueue		
Refresh  Add message  Dequeue message  Clear queue		
<input type="text"/> <i>Search to filter items...</i>		
ID	MESSAGE TEXT	INSERTION TIME
7949b019-8a55-49...	{ "id": "22448", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles",...}	Wed, 25 Apr 2018 18:57:57 GMT
7b03ea47-b06d-42...	{ "id": "3641", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles", "...}	Wed, 25 Apr 2018 18:57:57 GMT
7dab886e-3b45-4...	{ "id": "10310", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles",...}	Wed, 25 Apr 2018 18:57:57 GMT

## NOTE

If you use an [Azure Queue storage trigger for Azure Functions](#) for a queue that receives messages from Event Grid, you may see the following error message on the function execution:

```
The input is not a valid Base-64 string as it contains a non-base 64 character, more than two padding characters, or an illegal character among the padding characters.
```

The reason is that when you use an [Azure Queue storage trigger](#), Azure Functions expect a **base64 encoded string**, but Event Grid sends messages to a storage queue in a plain text format. Currently, it's not possible to configure the queue trigger for Azure Functions to accept plain text.

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this article. Otherwise, use the following command to delete the resources you created in this article.

```
az group delete --name gridResourceGroup
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about what Event Grid can help you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

# Quickstart: Route custom events to Azure Event Hubs with Azure CLI and Event Grid

4/22/2021 • 4 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. Azure Event Hubs is one of the supported event handlers. In this article, you use the Azure CLI to create a custom topic, subscribe to the custom topic, and trigger the event to view the result. You send the events to an event hub.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the `az group create` command.

The following example creates a resource group named `gridResourceGroup` in the `westus2` location.

```
az group create --name gridResourceGroup --location westus2
```

## Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

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

It might take a moment for the registration to finish. To check the status, run:

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

When `registrationState` is `Registered`, you're ready to continue.

## Create a Custom Topic

An event grid topic provides a user-defined endpoint that you post your events to. The following example creates the custom topic in your resource group. Replace `<your-topic-name>` with a unique name for your custom topic. The custom topic name must be unique because it's represented by a DNS entry.

```
topicname=<your-topic-name>
az eventgrid topic create --name $topicname -l westus2 -g gridResourceGroup
```

## Create event hub

Before subscribing to the custom topic, let's create the endpoint for the event message. You create an event hub for collecting the events.

```
namespace=<unique-namespace-name>
hubname=demohub

az eventhubs namespace create --name $namespace --resource-group gridResourceGroup
az eventhubs eventhub create --name $hubname --namespace-name $namespace --resource-group gridResourceGroup
```

## Subscribe to a custom topic

You subscribe to an event grid topic to tell Event Grid which events you want to track. The following example subscribes to the custom topic you created, and passes the resource ID of the event hub for the endpoint. The endpoint is in the format:

```
/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.EventHub/namespaces/<namespace-name>/eventhubs/<hub-name>
```

The following script gets the resource ID for the event hub, and subscribes to an event grid topic. It sets the endpoint type to `eventhub` and uses the event hub ID for the endpoint.

```
hubid=$(az eventhubs eventhub show --name $hubname --namespace-name $namespace --resource-group gridResourceGroup --query id --output tsv)
topicid=$(az eventgrid topic show --name $topicname -g gridResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--source-resource-id $topicid \
--name subtoeventhub \
--endpoint-type eventhub \
--endpoint $hubid
```

The account that creates the event subscription must have write access to the event hub.

## Send an event to your custom topic

Let's trigger an event to see how Event Grid distributes the message to your endpoint. First, let's get the URL and key for the custom topic.

```
endpoint=$(az eventgrid topic show --name $topicname -g gridResourceGroup --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name $topicname -g gridResourceGroup --query "key1" --output tsv)
```

To simplify this article, you use sample event data to send to the custom topic. Typically, an application or Azure service would send the event data. CURL is a utility that sends HTTP requests. In this article, use CURL to send the event to the custom topic. The following example sends three events to the event grid topic:

```
for i in 1 2 3
do
    event='[ {"id": """$RANDOM""", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles",
    "eventTime": "'`date +%Y-%m-%dT%H:%M:%S%z`'", "data":{ "make": "Ducati", "model": "Monster"}, "dataVersion":
    "1.0"} ]'
    curl -X POST -H "aeg-sas-key: $key" -d "$event" $endpoint
done
```

Navigate to the event hub in the portal, and notice that Event Grid sent those three events to the event hub.

The screenshot shows the Azure portal interface for an Event Hubs instance named 'demohub'. On the left, there's a navigation sidebar with links like Overview, Access control (IAM), Diagnose and solve problems, Properties, Locks, Automation script, Consumer groups, Capture, and Support + Troubleshooting. The main content area displays the 'gridResourceGroup' consumer group details, including Resource group (gridResourceGroup), Status (Active), Namespace (demohubspacetf09), Location (West US 2), and Subscription (Third Intern...). It also shows EVENT HUB CONTENTS (1 CONSUMER GROUP), EVENT HUB STATUS (ACTIVE), MESSAGE RETENTION (7 DAYS), and PARTITION COUNT (4). Below this, there are two line charts: 'Requests' and 'Messages'. The 'Requests' chart shows a sharp spike at 4:30 PM with a value of 10. The 'Messages' chart shows a similar spike at 4:30 PM with a value of 3. At the bottom, there are summary counts for incoming and outgoing messages: INCOMING REQUESTS (10), SUCCESSFUL REQUESTS (10), SERVER ERRORS (0), OUTGOING MESSAGES (3), and CAPTURED MESSAGES (0).

Typically, you create an application that retrieves the events from the event hub. To create an application that gets messages from an event hub, see:

- [Get started receiving messages with the Event Processor Host in .NET Standard](#)
- [Receive events from Azure Event Hubs using Java](#)
- [Receive events from Event Hubs using Apache Storm](#)

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this article. Otherwise, use the following command to delete the resources you created in this article.

```
az group delete --name gridResourceGroup
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about what Event Grid can help you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

# Quickstart: Send events from private container registry to Event Grid

4/21/2021 • 6 minutes to read • [Edit Online](#)

Azure Event Grid is a fully managed event routing service that provides uniform event consumption using a publish-subscribe model. In this quickstart, you use the Azure CLI to create a container registry, subscribe to registry events, then deploy a sample web application to receive the events. Finally, you trigger container image `push` and `delete` events and view the event payload in the sample application.

After you complete the steps in this article, events sent from your container registry to Event Grid appear in the sample web app:

Event Type	Subject
Microsoft.ContainerRegistry.ImageDeleted	myimage
Microsoft.ContainerRegistry.ImagePushed	myimage:v1
Microsoft.EventGrid.SubscriptionValidationEvent	

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- Use the Bash environment in [Azure Cloud Shell](#).  
[Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.

- The Azure CLI commands in this article are formatted for the **Bash** shell. If you're using a different shell like PowerShell or Command Prompt, you may need to adjust line continuation characters or variable assignment lines accordingly. This article uses variables to minimize the amount of command editing required.

## Create a resource group

An Azure resource group is a logical container in which you deploy and manage your Azure resources. The following `az group create` command creates a resource group named *myResourceGroup* in the *eastus* region. If you want to use a different name for your resource group, set `RESOURCE_GROUP_NAME` to a different value.

```
RESOURCE_GROUP_NAME=myResourceGroup

az group create --name $RESOURCE_GROUP_NAME --location eastus
```

## Create a container registry

Next, deploy a container registry into the resource group with the following commands. Before you run the `az acr create` command, set `ACR_NAME` to a name for your registry. The name must be unique within Azure, and is restricted to 5-50 alphanumeric characters.

```
ACR_NAME=<acrName>

az acr create --resource-group $RESOURCE_GROUP_NAME --name $ACR_NAME --sku Basic
```

Once the registry has been created, the Azure CLI returns output similar to the following:

```
{
  "adminUserEnabled": false,
  "creationDate": "2018-08-16T20:02:46.569509+00:00",
  "id": "/subscriptions/<Subscription
ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/myregistry",
  "location": "eastus",
  "loginServer": "myregistry.azurecr.io",
  "name": "myregistry",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "sku": {
    "name": "Basic",
    "tier": "Basic"
  },
  "status": null,
  "storageAccount": null,
  "tags": {},
  "type": "Microsoft.ContainerRegistry/registries"
}
```

## Create an event endpoint

In this section, you use a Resource Manager template located in a GitHub repository to deploy a pre-built sample web application to Azure App Service. Later, you subscribe to your registry's Event Grid events and specify this app as the endpoint to which the events are sent.

To deploy the sample app, set `SITE_NAME` to a unique name for your web app, and execute the following commands. The site name must be unique within Azure because it forms part of the fully qualified domain

name (FQDN) of the web app. In a later section, you navigate to the app's FQDN in a web browser to view your registry's events.

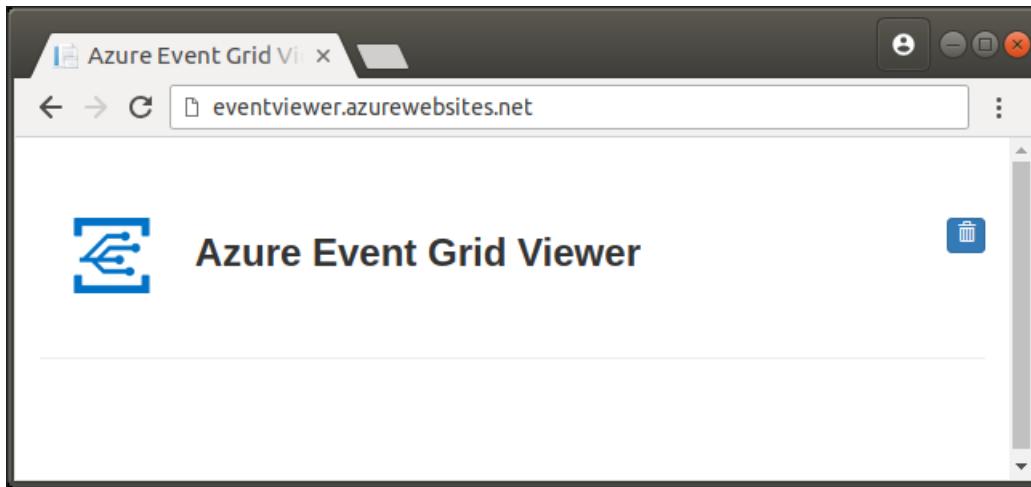
```
SITE_NAME=<your-site-name>

az deployment group create \
    --resource-group $RESOURCE_GROUP_NAME \
    --template-uri "https://raw.githubusercontent.com/Azure-Samples/azure-event-grid-
viewer/master/azuredeploy.json" \
    --parameters siteName=$SITE_NAME hostingPlanName=$SITE_NAME-plan
```

Once the deployment has succeeded (it might take a few minutes), open a browser and navigate to your web app to make sure it's running:

```
http://<your-site-name>.azurewebsites.net
```

You should see the sample app rendered with no event messages displayed:



## Enable the Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you might need to register the Event Grid resource provider. Run the following command to register the provider:

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

It might take a moment for the registration to finish. To check the status, run:

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

When `registrationState` is `Registered`, you're ready to continue.

## Subscribe to registry events

In Event Grid, you subscribe to a *topic* to tell it which events you want to track, and where to send them. The following [az eventgrid event-subscription create](#) command subscribes to the container registry you created, and specifies your web app's URL as the endpoint to which it should send events. The environment variables you populated in earlier sections are reused here, so no edits are required.

```

ACR_REGISTRY_ID=$(az acr show --name $ACR_NAME --query id --output tsv)
APP_ENDPOINT=https://$SITE_NAME.azurewebsites.net/api/updates

az eventgrid event-subscription create \
--name event-sub-acr \
--source-resource-id $ACR_REGISTRY_ID \
--endpoint $APP_ENDPOINT

```

When the subscription is completed, you should see output similar to the following:

```
{
  "destination": {
    "endpointBaseUrl": "https://eventgridviewer.azurewebsites.net/api/updates",
    "endpointType": "WebHook",
    "endpointUrl": null
  },
  "filter": {
    "includedEventTypes": [
      "All"
    ],
    "isSubjectCaseSensitive": null,
    "subjectBeginsWith": "",
    "subjectEndsWith": ""
  },
  "id": "/subscriptions/<Subscription
ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/myregistry/providers/Mic
rosoft.EventGrid/eventSubscriptions/event-sub-acr",
  "labels": null,
  "name": "event-sub-acr",
  "provisioningState": "Succeeded",
  "resourceGroup": "myResourceGroup",
  "topic": "/subscriptions/<Subscription
ID>/resourceGroups/myresourcegroup/providers/microsoft.containerregistry/registries/myregistry",
  "type": "Microsoft.EventGrid/eventSubscriptions"
}
}
```

## Trigger registry events

Now that the sample app is up and running and you've subscribed to your registry with Event Grid, you're ready to generate some events. In this section, you use ACR Tasks to build and push a container image to your registry. ACR Tasks is a feature of Azure Container Registry that allows you to build container images in the cloud, without needing the Docker Engine installed on your local machine.

### Build and push image

Execute the following Azure CLI command to build a container image from the contents of a GitHub repository. By default, ACR Tasks automatically pushes a successfully built image to your registry, which generates the `ImagePushed` event.

```
az acr build --registry $ACR_NAME --image myimage:v1 -f Dockerfile https://github.com/Azure-Samples/acr-
build-helloworld-node.git#main
```

You should see output similar to the following while ACR Tasks builds and then pushes your image. The following sample output has been truncated for brevity.

```
Sending build context to ACR...
Queued a build with build ID: aa2
Waiting for build agent...
2018/08/16 22:19:38 Using acb_vol_27a2afa6-27dc-4ae4-9e52-6d6c8b7455b2 as the home volume
2018/08/16 22:19:38 Setting up Docker configuration...
2018/08/16 22:19:39 Successfully set up Docker configuration
2018/08/16 22:19:39 Logging in to registry: myregistry.azurecr.io
2018/08/16 22:19:55 Successfully logged in
Sending build context to Docker daemon 94.72kB
Step 1/5 : FROM node:9-alpine
...

```

To verify that the built image is in your registry, execute the following command to view the tags in the "myimage" repository:

```
az acr repository show-tags --name $ACR_NAME --repository myimage
```

The "v1" tag of the image you built should appear in the output, similar to the following:

```
[  
  "v1"  
]
```

## Delete the image

Now, generate an `ImageDeleted` event by deleting the image with the `az acr repository delete` command:

```
az acr repository delete --name $ACR_NAME --image myimage:v1
```

You should see output similar to the following, asking for confirmation to delete the manifest and associated images:

```
This operation will delete the manifest  
'sha256:f15fa9d0a69081ba93eee308b0e475a54fac9c682196721e294b2bc20ab23a1b' and all the following images:  
'myimage:v1'.  
Are you sure you want to continue? (y/n):
```

## View registry events

You've now pushed an image to your registry and then deleted it. Navigate to your Event Grid Viewer web app, and you should see both `ImageDeleted` and `ImagePushed` events. You might also see a subscription validation event generated by executing the command in the [Subscribe to registry events](#) section.

The following screenshot shows the sample app with the three events, and the `ImageDeleted` event is expanded to show its details.

The screenshot shows a browser window titled "Azure Event Grid" displaying the "Azure Event Grid Viewer". The URL in the address bar is "eventviewer.azurewebsites.net". The page content includes a header with a gear icon and the text "Azure Event Grid Viewer". Below this is a table with two columns: "Event Type" and "Subject". A single event entry is shown:

Event Type	Subject
Microsoft.ContainerRegistry.ImageDeleted	myimage

The "Event Type" row contains a small blue eye icon. The "Subject" row contains a blue trash can icon.

Below the table is a large JSON object representing the event data:

```
{  
  "id": "0e22195d-048f-4a26-99a7-075071a641cb",  
  "topic": "/subscriptions/<Subscription ID>/resourceGroups/myResourceGroup/providers/Microsoft.ContainerRegistry/registries/myregistry",  
  "subject": "myimage",  
  "eventType": "Microsoft.ContainerRegistry.ImageDeleted",  
  "eventTime": "2018-08-16T22:40:32.5983564Z",  
  "data": "{\"id\":\"0e22195d-048f-4a26-99a7-075071a641cb\", \"timestamp\":\"2018-08-16T22:40:31.619754049Z\", \"action\":\"delete\", \"target\":{\"mediaType\":\"application/vnd.dockerdistribution.manifest.v2+json\", \"digest\":\"sha256:f15fa9d0a69081ba93eee308b0e475a54fac9c682196721e294b2bc20ab23a1b\", \"repository\":\"myimage\"}, \"request\":{\"id\":\"1cac1c8f-fa71-4264-b457-8561f3910487\", \"host\":\"myregistry.azurecr.io\", \"method\":\"DELETE\", \"useragent\":\"python-requests/2.19.1\"}}",  
  "dataVersion": "1.0",  
  "metadataVersion": "1"  
}
```

Below this are two more event entries:

Event Type	Subject
Microsoft.ContainerRegistry.ImagePushed	myimage:v1
Microsoft.EventGrid.SubscriptionValidationEvent	

Congratulations! If you see the `ImagePushed` and `ImageDeleted` events, your registry is sending events to Event Grid, and Event Grid is forwarding those events to your web app endpoint.

## Clean up resources

Once you're done with the resources you created in this quickstart, you can delete them all with the following Azure CLI command. When you delete a resource group, all of the resources it contains are permanently deleted.

**WARNING:** This operation is irreversible. Be sure you no longer need any of the resources in the group before running the command.

```
az group delete --name $RESOURCE_GROUP_NAME
```

## Event Grid event schema

You can find the Azure Container Registry event message schema reference in the Event Grid documentation:

[Azure Event Grid event schema for Container Registry](#)

## Next steps

In this quickstart, you deployed a container registry, built an image with ACR Tasks, deleted it, and have consumed your registry's events from Event Grid with a sample application. Next, move on to the ACR Tasks tutorial to learn more about building container images in the cloud, including automated builds on base image update:

[Build container images in the cloud with ACR Tasks](#)

# Quickstart: Handle SMS events for Delivery Reports and Inbound Messages

4/29/2021 • 3 minutes to read • [Edit Online](#)

## IMPORTANT

Phone number availability is currently restricted to Azure subscriptions that have a billing address in the United States. For more information, visit the [Phone number types](#) documentation.

Get started with Azure Communication Services by using Azure Event Grid to handle Communication Services SMS events.

## About Azure Event Grid

Azure Event Grid is a cloud-based eventing service. In this article, you'll learn how to subscribe to events for [communication service events](#), and trigger an event to view the result. Typically, you send events to an endpoint that processes the event data and takes actions. In this article, we'll send the events to a web app that collects and displays the messages.

## Prerequisites

- An Azure account with an active subscription. [Create an account for free](#).
- An Azure Communication Service resource. Further details can be found in the [Create an Azure Communication Resource](#) quickstart.
- An SMS enabled telephone number. [Get a phone number](#).

## Setting up

### Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider following the steps below:

In the Azure portal:

1. Select **Subscriptions** on the left menu.
2. Select the subscription you're using for Event Grid.
3. On the left menu, under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

### Event Grid Viewer deployment

For this quickstart, we will use the [Azure Event Grid Viewer Sample](#) to view events in near-real time. This will provide the user with the experience of a real-time feed. In addition, the payload of each event should be available for inspection as well.

# Subscribe to the SMS events using web hooks

In the portal, navigate to your Azure Communication Services Resource that you created. Inside the Communication Service resource, select **Events** from the left menu of the **Communication Services** page.

The screenshot shows the Azure Communication Services | Events page. The top navigation bar includes links for Microsoft Azure (Preview), Search resources, services, and docs, and a user profile for Casey.Jensen@contoso.com. The main content area has a breadcrumb trail: All services > All resources > Azure Communication Services. The title is "Azure Communication Services | Events". On the left, a sidebar lists various options: Overview, Activity log, Access control (IAM), Tags, QuickStart, Automation (selected), Events (highlighted with a red box), Export template, Resource explorer, Settings, Properties, Locks, Tools, Keys, and Phone Numbers. At the top center, there is a search bar, a "Event Subscription" button (also highlighted with a red box), and a "Refresh" button. Below the search bar, there are two tabs: "Get Started" and "Event Subscriptions" (the latter is underlined). A large text area says "Events, automated." followed by a description of Event Grid's capabilities. To the right, there are sections for "Example Scenarios" (with examples like resizing an image or copying files) and "Azure Event Grid natively supports these resources as event handlers" (with examples for Logic Apps and Azure Functions).

Press **Add Event Subscription** to enter the creation wizard.

On the **Create Event Subscription** page, Enter a **name** for the event subscription.

You can subscribe to specific events to tell Event Grid which of the SMS events you want to track, and where to send the events. Select the events you'd like to subscribe to from the dropdown menu. For SMS you'll have the option to choose **SMS Received** and **SMS Delivery Report Received**.

If you're prompted to provide a **System Topic Name**, feel free to provide a unique string. This field has no impact on your experience and is used for internal telemetry purposes.

Check out the full list of [events supported by Azure Communication Services](#).



# Create Event Subscription

Event Grid

Basic

Filters

Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

## EVENT SUBSCRIPTION DETAILS

Name \*

SMSEventsSubscription



Event Schema

Event Grid Schema



## TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type

Communication Service

Source Resource

SMSApplication

System Topic Name \* ⓘ

SMSTopic



## EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

2 selected

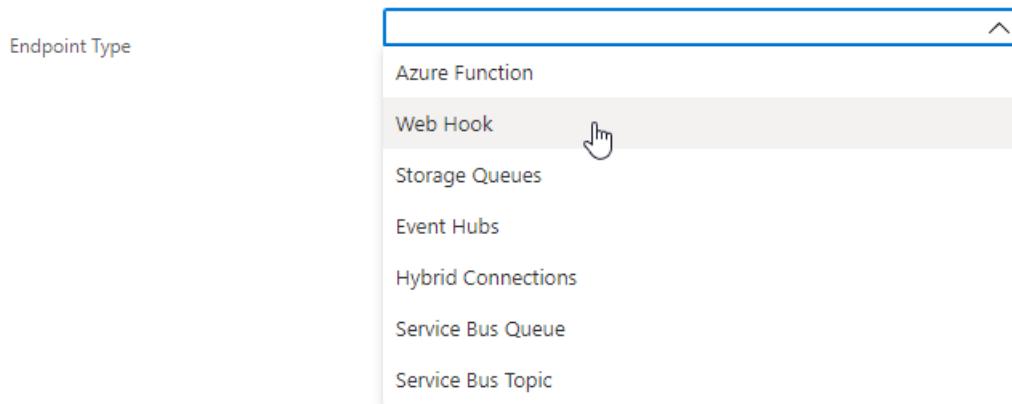


- SMS Received
- SMS Delivery Report Received
- Chat Message Received
- Chat Message Edited
- Chat Message Deleted
- Chat Thread Created With User
- Chat Thread With User Deleted
- Chat Thread Properties Updated Per User
- Chat Member Added To Thread With User
- Chat Member Removed From Thread With User

Select Web Hook for Endpoint type.

## ENDPOINT DETAILS

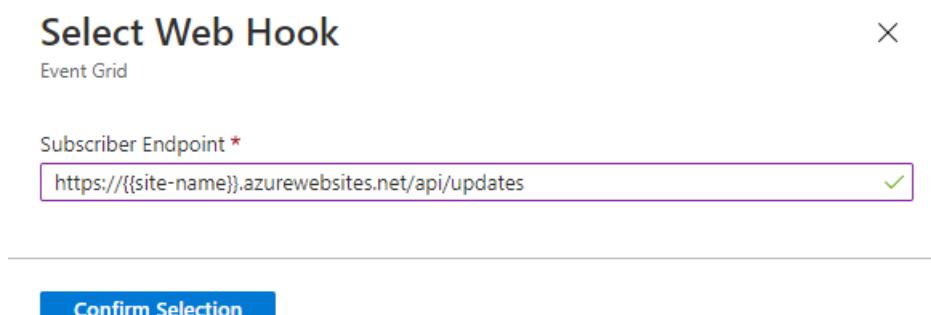
Pick an event handler to receive your events. [Learn more](#)



For Endpoint, click **Select an endpoint**, and enter the URL of your web app.

In this case, we will use the URL from the [Azure Event Grid Viewer Sample](#) we set up earlier in the quickstart. The URL for the sample will be in the format: `https://{{site-name}}.azurewebsites.net/api/updates`

Then select **Confirm Selection**.



## Viewing SMS events

### Triggering SMS events

To view event triggers, we must generate events in the first place.

- `SMS Received` events are generated when the Communication Services phone number receives a text message. To trigger an event, just send a message from your phone to the phone number attached to your Communication Services resource.
- `SMS Delivery Report Received` events are generated when you send an SMS to a user using a Communication Services phone number. To trigger an event, you are required to enable `Delivery Report` in the options of the [sent SMS](#). Try sending a message to your phone with `Delivery Report`. Completing this action incurs a small cost of a few USD cents or less in your Azure account.

Check out the full list of [events supported by Azure Communication Services](#).

### Receiving SMS events

Once you complete either action above you will notice that `SMS Received` and `SMS Delivery Report Received` events are sent to your endpoint. These events will show up in the [Azure Event Grid Viewer Sample](#) we set up at the beginning. You can press the eye icon next to the event to see the entire payload. Events will look like this:



Event Type	Subject
Microsoft.Communication.SMSReceived	/phonenumber/+14255550123

```
{
  "id": "0000000-0000-0000-000000000000",
  "topic": "/subscriptions/0000000-0000-0000-0000-000000000000/resourcegroups/sanitized-rg/providers/microsoft.communication/communicationservices/acs-re
source",
  "subject": "/phonenumber/+14255550123",
  "data": {
    "messageId": "0000000-0000-0000-000000000000",
    "from": "+14255550122",
    "to": "+14255550123",
    "message": "Weekly Promotion!",
    "receivedTimestamp": "2021-03-17T02:39:04.01Z"
  },
  "eventType": "Microsoft.Communication.SMSReceived",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-03-17T02:39:04Z"
}
```

Event Type	Subject
Microsoft.Communication.SMSDeliveryReportReceived	/phonenumber/+14255550123

```
{
  "id": "0000000-0000-0000-000000000000",
  "topic": "/subscriptions/0000000-0000-0000-0000-000000000000/resourcegroups/sanitized-rg/providers/microsoft.communication/communicationservices/acs-re
source",
  "subject": "/phonenumber/+14255550123",
  "data": {
    "messageId": "0000000-0000-0000-000000000000",
    "from": "+14255550122",
    "to": "+14255550123",
    "deliveryStatus": "Delivered",
    "deliveryStatusDetails": "No error.",
    "receivedTimestamp": "2021-03-17T02:39:14.7786146Z",
    "deliveryAttempts": [
      {
        "timestamp": "2021-03-17T02:39:04.0136264Z",
        "segmentsSucceeded": 1,
        "segmentsFailed": 0
      }
    ],
    "tag": "marketing"
  },
  "eventType": "Microsoft.Communication.SMSDeliveryReportReceived",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-03-17T02:39:14Z"
}
```

Learn more about the [event schemas](#) and other eventing concepts.

## Clean up resources

If you want to clean up and remove a Communication Services subscription, you can delete the resource or resource group. Deleting the resource group also deletes any other resources associated with it. Learn more about [cleaning up resources](#).

## Next steps

In this quickstart, you learned how to consume SMS events. You can receive SMS messages by creating an Event Grid subscription.

[Send SMS](#)

You may also want to:

- [Learn about event handling concepts](#)
- [Learn about Event Grid](#)

# Quickstart: Route Azure Cache for Redis events to web endpoint with the Azure portal

4/15/2021 • 7 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. In this quickstart, you'll use the Azure portal to create an Azure Cache for Redis instance, subscribe to events for that instance, trigger an event, and view the results. Typically, you send events to an endpoint that processes the event data and takes actions. However, to simplify this quickstart, you'll send events to a web app that will collect and display the messages.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

When you're finished, you'll see that the event data has been sent to the web app.

The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a header with a logo and the text "Azure Event Grid Viewer". Below the header, there's a table with two columns: "Event Type" and "Subject". Under "Event Type", there's a small icon followed by the text "Microsoft.Cache.ScalingCompleted". Under "Subject", there's the text "ScalingCompleted". Below the table, there's a large code block containing JSON event data. The JSON data includes fields like "id", "eventType", "topic", "data", "subject", "dataVersion", "metadataVersion", and "eventTime". The "data" field contains detailed information about the scaling completed event, including the timestamp "2020-12-21T20:51:37.2941616+00:00" and status "Succeeded".

Event Type	Subject
Microsoft.Cache.ScalingCompleted	ScalingCompleted

```
{  
  "id": "00000000-0000-0000-0000-000000000000",  
  "eventType": "Microsoft.Cache.ScalingCompleted",  
  "topic": "/subscriptions/  
  /resourceGroups/  
  /providers/Microsoft.Cache/Redis/azure-ca  
che",  
  "data": {  
    "name": "ScalingCompleted",  
    "timestamp": "2020-12-21T20:51:37.2941616+00:00",  
    "status": "Succeeded"  
  },  
  "subject": "ScalingCompleted",  
  "dataVersion": "1.0",  
  "metadataVersion": "1",  
  "eventTime": "2020-12-21T20:51:37.2941616+00:00"  
}
```

## Create an Azure Cache for Redis cache instance

1. To create a cache, sign in to the [Azure portal](#) and select **Create a resource**.

The screenshot shows the Azure portal homepage. At the top, there's a navigation bar with icons for back, forward, refresh, and a search bar containing the URL "https://ms.portal.azure.com/#home". Below the navigation bar, there's a "New" button and a search bar with the placeholder "Search resources, services, and docs (G+)". On the left, there's a sidebar with options like "Create a resource" (which is highlighted with a red box), "Home", "Dashboard", "All services", and "FAVORITES". On the right, there are several service icons: "Azure Cache for Redis", "Resource groups", "Subscriptions", "Virtual machines", and "App Services".

2. On the New page, select **Databases** and then select **Azure Cache for Redis**.

# New

 Search the Marketplace

Azure Marketplace [See all](#)

Featured [See all](#)

Get started



Azure SQL Managed Instance  
[Quickstarts + tutorials](#)

Recently created



SQL Database  
[Quickstarts + tutorials](#)

AI + Machine Learning



Azure Synapse Analytics (formerly  
SQL DW)  
[Quickstarts + tutorials](#)

Analytics

Blockchain



Azure Database for MariaDB  
[Learn more](#)

Compute



Azure Database for MySQL  
[Quickstarts + tutorials](#)

Containers

Databases



Azure Database for PostgreSQL  
[Quickstarts + tutorials](#)

Developer Tools



Azure Cosmos DB  
[Quickstarts + tutorials](#)

Identity



Azure Database Migration Service  
[Learn more](#)

Integration

Internet of Things



SQL Server 2017 Enterprise Windows  
Server 2016  
[Learn more](#)

IT & Management Tools

Media



Azure Cache for Redis  
[Quickstarts + tutorials](#)

Migration

Mixed Reality



Azure Cache for Redis  
[Quickstarts + tutorials](#)

Monitoring & Diagnostics

Networking



Azure Database Migration Service  
[Learn more](#)

Security

Software as a Service (SaaS)



Azure Database Migration Service  
[Learn more](#)

Storage

Web

- On the **New Redis Cache** page, configure the settings for your new cache.

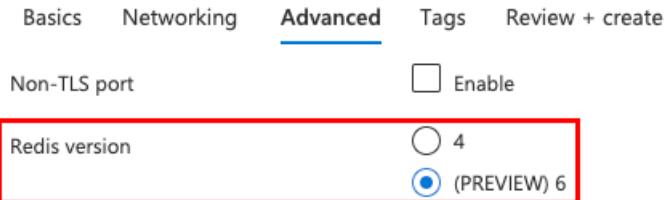
SETTING	SUGGESTED VALUE	DESCRIPTION
DNS name	Enter a globally unique name.	The cache name must be a string between 1 and 63 characters that contains only numbers, letters, or hyphens. The name must start and end with a number or letter, and can't contain consecutive hyphens. Your cache instance's <i>host name</i> will be <i>&lt;DNS name&gt;.redis.cache.windows.net</i> .
Subscription	Drop down and select your subscription.	The subscription under which to create this new Azure Cache for Redis instance.

SETTING	SUGGESTED VALUE	DESCRIPTION
Resource group	Drop down and select a resource group, or select <b>Create new</b> and enter a new resource group name.	Name for the resource group in which to create your cache and other resources. By putting all your app resources in one resource group, you can easily manage or delete them together.
Location	Drop down and select a location.	Select a <a href="#">region</a> near other services that will use your cache.
Pricing tier	Drop down and select a <a href="#">Pricing tier</a> .	The pricing tier determines the size, performance, and features that are available for the cache. For more information, see <a href="#">Azure Cache for Redis Overview</a> .

4. Select the **Networking** tab or click the **Networking** button at the bottom of the page.
5. In the **Networking** tab, select your connectivity method.
6. Select the **Next: Advanced** tab or click the **Next: Advanced** button on the bottom of the page.
7. In the **Advanced** tab for a basic or standard cache instance, select the enable toggle if you want to enable a non-TLS port. You can also select which Redis version you would like use, either 4 or (PREVIEW) 6.

[Home](#) > [Azure Cache for Redis](#) >

## New Redis Cache



8. In the **Advanced** tab for premium cache instance, configure the settings for non-TLS port, clustering, and data persistence. You can also select which Redis version you would like use, either 4 or (PREVIEW) 6.
9. Select the **Next: Tags** tab or click the **Next: Tags** button at the bottom of the page.
10. Optionally, in the **Tags** tab, enter the name and value if you wish to categorize the resource.
11. Select **Review + create**. You're taken to the Review + create tab where Azure validates your configuration.
12. After the green Validation passed message appears, select **Create**.

It takes a while for the cache to create. You can monitor progress on the [Azure Cache for Redis Overview](#) page. When **Status** shows as **Running**, the cache is ready to use.

## Create a message endpoint

Before subscribing to the events for the cache instance, let's create the endpoint for the event message. Typically, the endpoint takes actions based on the event data. To simplify this quickstart, you'll deploy 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.

1. Select **Deploy to Azure** in GitHub README to deploy the solution to your subscription.



2. On the **Custom deployment** page, do the following steps:

- a. For **Resource group**, select the resource group that you created when creating the cache instance. It will be easier for you to clean up after you are done with the tutorial by deleting the resource group.
- b. For **Site Name**, enter a name for the web app.
- c. For **Hosting plan name**, enter a name for the App Service plan to use for hosting the web app.
- d. Select the check box for **I agree to the terms and conditions stated above**.
- e. Select **Purchase**.

SETTING	SUGGESTED VALUE	DESCRIPTION
Subscription	Drop down and select your subscription.	The subscription under which to create this web app.
Resource group	Drop down and select a resource group, or select <b>Create new</b> and enter a new resource group name.	By putting all your app resources in one resource group, you can easily manage or delete them together.
Site Name	Enter a name for your web app.	This value cannot be empty.
Hosting plan name	Enter a name for the App Service plan to use for hosting the web app.	This value cannot be empty.

3. Select Alerts (bell icon) in the portal, and then select **Go to resource group**.



## Notifications



[More events in the activity log →](#)

[Dismiss all](#)



Deployment succeeded

Deployment 'Microsoft.Template-20201221134916' to resource group 'cache' was successful.

[Go to resource group](#)

[Pin to dashboard](#)

a minute ago

4. On the **Resource group** page, in the list of resources, select the web app that you created. You'll also see the App Service plan and the cache instance in this list.
5. On the **App Service** page for your web app, select the URL to navigate to the web site. The URL should be in this format: `https://<your-site-name>.azurewebsites.net`.

6. Confirm that you see the site but no events have been posted to it yet.



## Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

1. Select **Subscriptions** on the left menu.
2. Select the subscription you're using for Event Grid.
3. On the left menu, under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

## Subscribe to the Azure Cache for Redis instance

In this step, you'll subscribe to a topic to tell Event Grid which events you want to track, and where to send the events.

1. In the portal, navigate to your cache instance that you created earlier.
2. On the **Azure Cache for Redis** page, select **Events** on the left menu.
3. Select **Web Hook**. You are sending events to your viewer app using a web hook for the endpoint.

The screenshot shows the Azure Cache for Redis 'Events' page. The left sidebar has a 'Events' link highlighted with a red box. The main content area displays several examples of how Event Grid can be used:

- Logic Apps**: Use events as a trigger for executing a Logic App, starting Azure-wide workflows and automation.
- Azure Function**: Use events as a trigger for executing an Azure Function, which enables serverless custom code execution.
- Web Hook** (highlighted with a red box): Send events as HTTP pushes to a Web Hook endpoint, both within or outside of Azure.
- Storage Queues**: Send events as messages to a Storage Queue, providing decoupling and in-motion storage.

4. On the **Create Event Subscription** page, enter the following:

SETTING	SUGGESTED VALUE	DESCRIPTION
<b>Name</b>	Enter a name for the event subscription.	The value must be between 3 and 64 characters long. It can only contain letters, numbers, and dashes.
<b>Event Types</b>	Drop down and select which event type(s) you want to get pushed to your destination. For this quickstart, we'll be scaling our cache instance.	Patching, scaling, import and export are the available options.
<b>Endpoint Type</b>	Select <b>Web Hook</b> .	Event handler to receive your events.
<b>Endpoint</b>	Click <b>Select an endpoint</b> , and enter the URL of your web app and add <code>api/updates</code> to the home page URL (for example: <code>https://cache.azurewebsites.net/api/updates</code> ), and then select <b>Confirm Selection</b> .	This is the URL of your web app that you created earlier.

5. Now, on the **Create Event Subscription** page, select **Create** to create the event subscription.

6. 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.

The screenshot shows the Azure Event Grid Viewer interface. At the top, there's a logo and the text "Azure Event Grid Viewer". On the right side, there's a small blue square icon with a white symbol. Below the header, there are two columns: "Event Type" and "Subject". Under "Event Type", there's a small blue eye icon followed by the text "Microsoft.EventGrid.SubscriptionValidationEvent". In the "Subject" column, there's a long URL: "https://rp-eastus.eventgrid.azure.net:553/events/subscriptions/event-grid-cache/validate?id=4F775990-2FED-4CAF-9E1C-FA898AFD134F&t=2020-12-21T19:59:12.6625048Z&apiVersion=2020-04-01-preview&token=". Below the table, there's a large code block containing JSON data:

```
[{"id": "", "topic": "/subscriptions/ /resourceGroups/ /providers/Microsoft.Cache/Redis/azure-ca", "subject": "", "data": { "validationCode": "", "validationUrl": "https://rp-eastus.eventgrid.azure.net:553/events/subscriptions/event-grid-cache/validate?id=4F775990-2FED-4CAF-9E1C-FA898AFD134F&t=2020-12-21T19:59:12.6625048Z&apiVersion=2020-04-01-preview&token": "" }, "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent", "eventTime": "2020-12-21T19:59:12.6625048Z", "metadataVersion": "1", "dataVersion": "2" }]
```

## Send an event to your endpoint

Now, let's trigger an event to see how Event Grid distributes the message to your endpoint. We'll be scaling your Azure Cache for Redis instance.

1. In the Azure portal, navigate to your Azure Cache for Redis instance and select **Scale** on the left menu.
2. Select the desired pricing tier from the **Scale** page and click **Select**.

You can scale to a different pricing tier with the following restrictions:

- You can't scale from a higher pricing tier to a lower pricing tier.
  - You can't scale from a **Premium** cache down to a **Standard** or a **Basic** cache.
  - You can't scale from a **Standard** cache down to a **Basic** cache.
- You can scale from a **Basic** cache to a **Standard** cache but you can't change the size at the same time. If you need a different size, you can do a subsequent scaling operation to the desired size.
- You can't scale from a **Basic** cache directly to a **Premium** cache. First, scale from **Basic** to **Standard** in one scaling operation, and then from **Standard** to **Premium** in a subsequent scaling operation.
- You can't scale from a larger size down to the C0 (250 MB) size.

While the cache is scaling to the new pricing tier, a **Scaling** status is displayed in the **Azure Cache for Redis** blade. When scaling is complete, the status changes from **Scaling** to **Running**.

3. You've triggered the event, and Event Grid sent the message to the endpoint you configured when subscribing. The message is in the JSON format and it contains an array with one or more events. In the following example, the JSON message contains an array with one event. View your web app and notice that a **ScalingCompleted** event was received.

Event Type	Subject
Microsoft.Cache.ScalingCompleted	ScalingCompleted
<pre>{   "id": "00000000-0000-0000-0000-000000000000",   "eventType": "Microsoft.Cache.ScalingCompleted",   "topic": "/subscriptions/ che",   "data": {     "name": "ScalingCompleted",     "timestamp": "2020-12-21T20:51:37.2941616+00:00",     "status": "Succeeded"   },   "subject": "ScalingCompleted",   "dataversion": "1.0",   "metadataVersion": "1",   "eventTime": "2020-12-21T20:51:37.2941616+00:00" }</pre>	

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this quickstart. Otherwise, delete the resources you created in this quickstart.

Select the resource group, and select **Delete resource group**.

## Next steps

Now that you know how to create custom topics and event subscriptions, learn more about what Event Grid can help you do:

- Reacting to Azure Cache for Redis events
  - About Event Grid

# Tutorial: Monitor virtual machine changes by using Azure Event Grid and Logic Apps

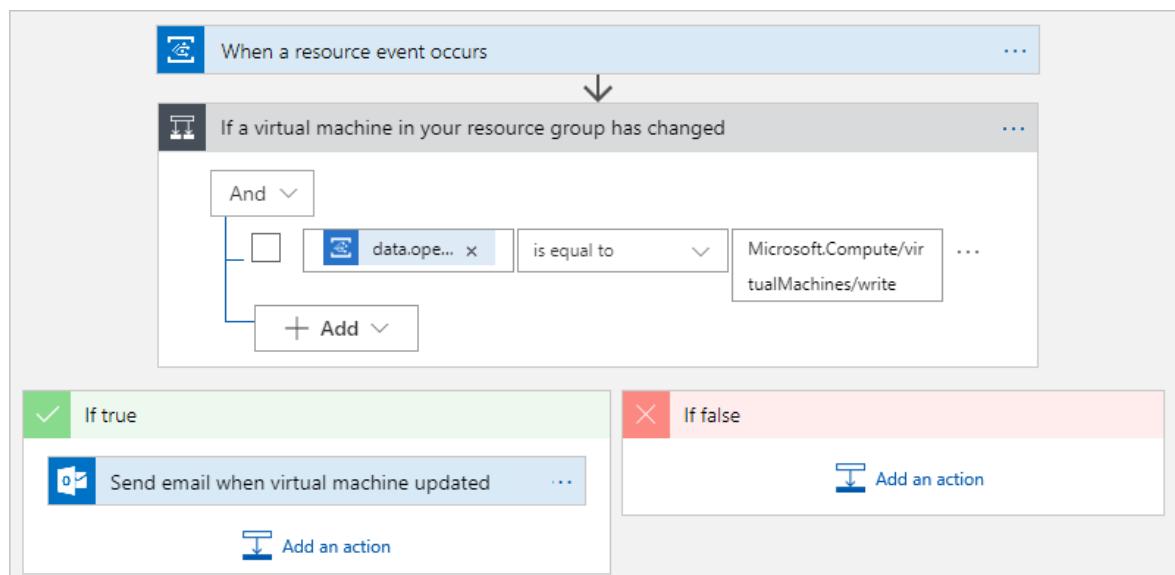
4/22/2021 • 11 minutes to read • [Edit Online](#)

To monitor and respond to specific events that happen in Azure resources or third-party resources, you can automate and run tasks as a workflow by creating a [logic app](#) that uses minimal code. These resources can publish events to an [Azure event grid](#). In turn, the event grid pushes those events to subscribers that have queues, webhooks, or [event hubs](#) as endpoints. As a subscriber, your logic app can wait for those events from the event grid before running automated workflows to perform tasks.

For example, here are some events that publishers can send to subscribers through the Azure Event Grid service:

- Create, read, update, or delete a resource. For example, you can monitor changes that might incur charges on your Azure subscription and affect your bill.
- Add or remove a person from an Azure subscription.
- Your app performs a particular action.
- A new message appears in a queue.

This tutorial creates a logic app that monitors changes to a virtual machine, and sends emails about those changes. When you create a logic app with an event subscription for an Azure resource, events flow from that resource through an event grid to the logic app. The tutorial walks you through building this logic app:



In this tutorial, you learn how to:

- Create a logic app that monitors events from an event grid.
- Add a condition that specifically checks for virtual machine changes.
- Send email when your virtual machine changes.

## Prerequisites

- An Azure subscription. If you don't have an Azure subscription, [sign up for a free Azure account](#).
- An email account from an email provider supported by Logic Apps for sending notifications, such as Office 365 Outlook, Outlook.com, or Gmail. For other providers, [review the connectors list here](#).

This tutorial uses an Office 365 Outlook account. If you use a different email account, the general steps

stay the same, but your UI might appear slightly different.

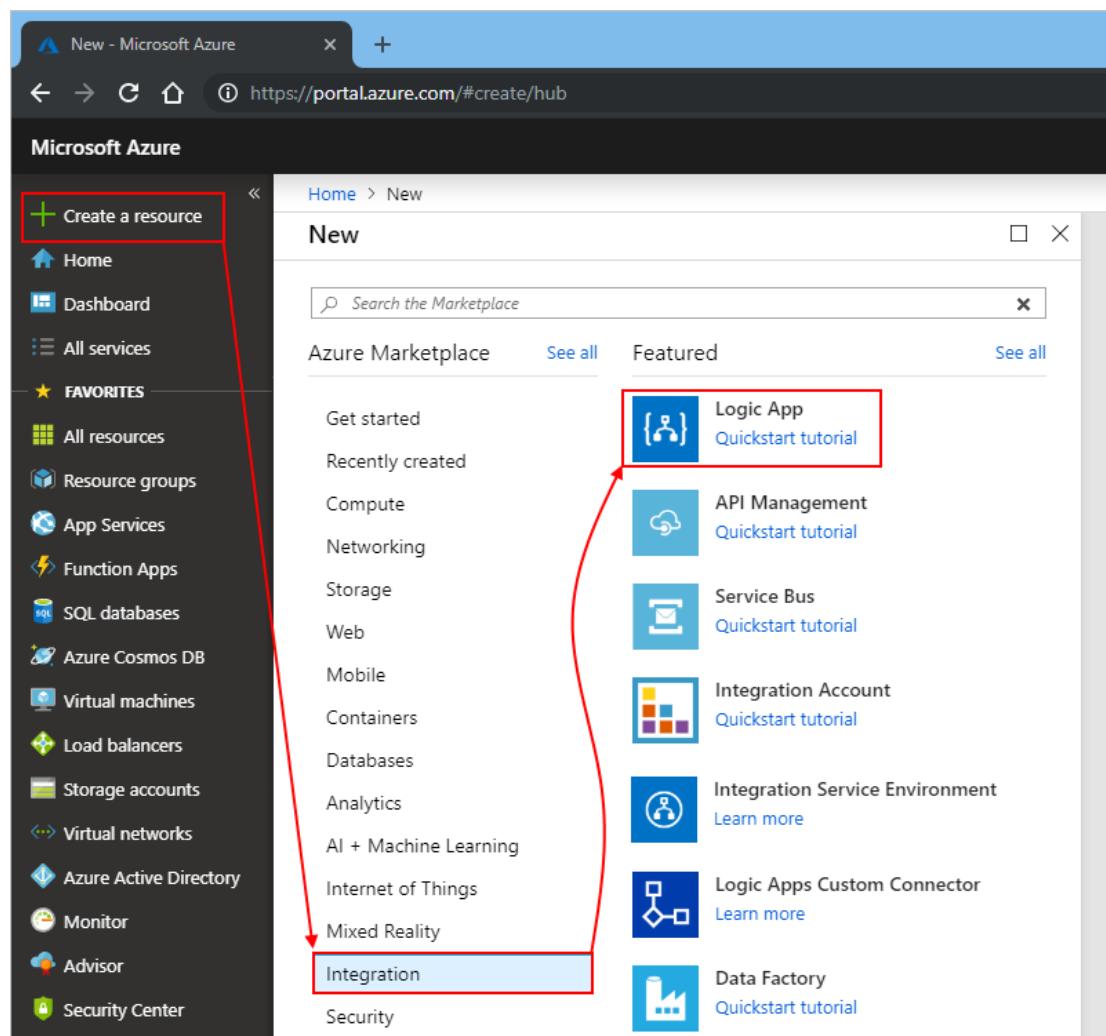
#### IMPORTANT

If you want to use the Gmail connector, only G-Suite business accounts can use this connector without restriction in logic apps. If you have a Gmail consumer account, you can use this connector with only specific Google-approved services, or you can [create a Google client app to use for authentication with your Gmail connector](#). For more information, see [Data security and privacy policies for Google connectors in Azure Logic Apps](#).

- A [virtual machine](#) that's alone in its own Azure resource group. If you haven't already done so, create a virtual machine through the [Create a VM tutorial](#). To make the virtual machine publish events, you [don't need to do anything else](#).

## Create blank logic app

1. Sign in to the [Azure portal](#) with your Azure account credentials.
2. From the main Azure menu, select **Create a resource > Integration > Logic App**.



3. Under **Logic App**, provide information about your logic app resource. When you're done, select **Create**.

**Logic App**

Create

Name \*

Subscription \*

Resource group \* ⓘ

Create new  Use existing

Location \*

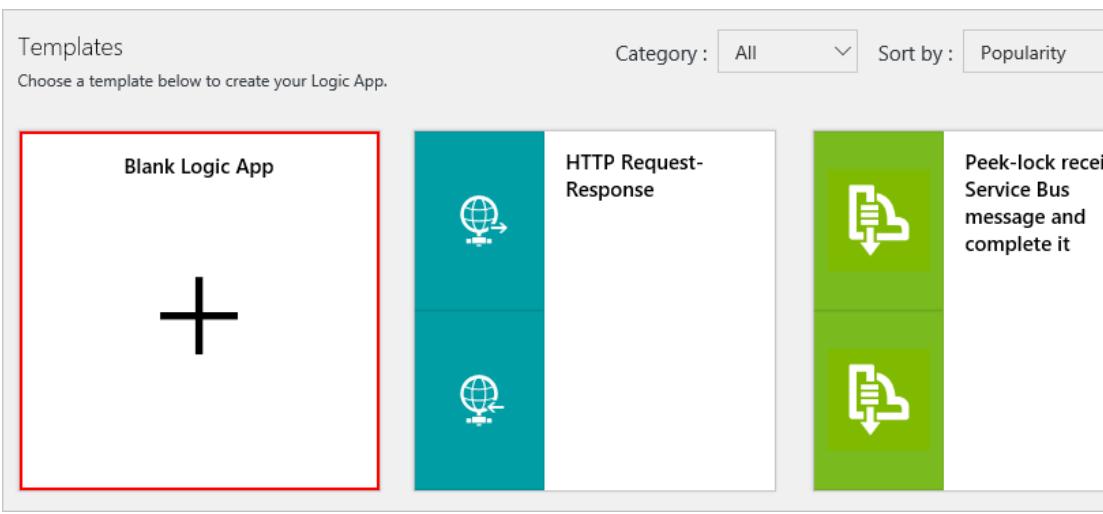
Log Analytics ⓘ

 You can add triggers and actions to your Logic App after creation.

**Create**    Automation options

PROPERTY	REQUIRED	VALUE	DESCRIPTION
Name	Yes	<logic-app-name>	Provide a unique name for your logic app.
Subscription	Yes	<Azure-subscription-name>	Select the same Azure subscription for all the services in this tutorial.
Resource group	Yes	<Azure-resource-group>	The Azure resource group name for your logic app, which you can select for all the services in this tutorial.
Location	Yes	<Azure-region>	Select the same region for all services in this tutorial.

4. After Azure deploys your logic app, the Logic Apps Designer shows a page with an introduction video and commonly used triggers. Scroll past the video and triggers.
5. Under **Templates**, select **Blank Logic App**.



The Logic Apps Designer now shows you the *triggers* that you can use to start your logic app. Every logic app must start with a trigger, which fires when a specific event happens or when a specific condition is met. Each time the trigger fires, Azure Logic Apps creates a workflow instance that runs your logic app.

## Add an Event Grid trigger

Now add the Event Grid trigger, which you use to monitor the resource group for your virtual machine.

1. On the designer, in the search box, enter `event grid` as your filter. From the triggers list, select the **When a resource event occurs** trigger.

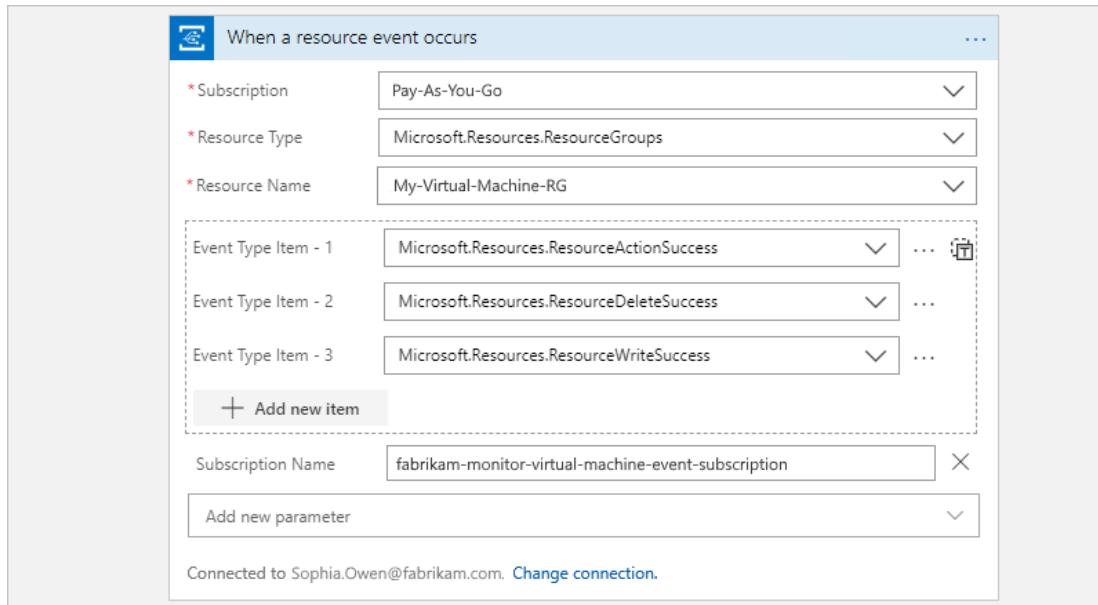
Trigger Type	Description
Request	
Azure Event Grid	
Azure Event Grid Publish	
<b>When a resource event occurs</b>	Azure Event Grid
When an Event Grid resource event occurs	Request

2. When prompted, sign in to Azure Event Grid with your Azure account credentials. In the **Tenant** list, which shows the Azure Active Directory tenant that's associated with your Azure subscription, check that the correct tenant appears, for example:

**NOTE**

If you're signed in with a personal Microsoft account, such as @outlook.com or @hotmail.com, the Event Grid trigger might not appear correctly. As a workaround, select [Connect with Service Principal](#), or authenticate as a member of the Azure Active Directory that's associated with your Azure subscription, for example, *username@emailoutlook.onmicrosoft.com*.

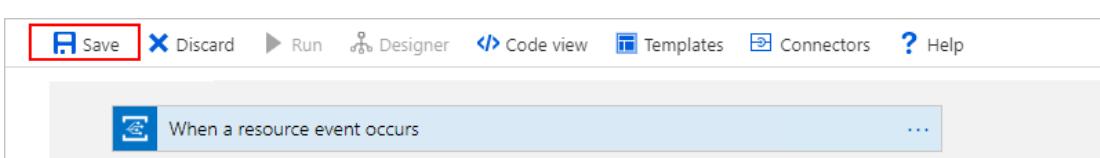
- Now subscribe your logic app to events from the publisher. Provide the details about your event subscription as described in the following table, for example:



PROPERTY	REQUIRED	VALUE	DESCRIPTION
Subscription	Yes	<event-publisher-Azure-subscription-name>	Select the name for the Azure subscription that's associated with the <i>event publisher</i> . For this tutorial, select the Azure subscription name for your virtual machine.
Resource Type	Yes	<event-publisher-Azure-resource-type>	Select the Azure resource type for the event publisher. For more information about Azure resource types, see <a href="#">Azure resource providers and types</a> . For this tutorial, select the <code>Microsoft.Resources.ResourceGroups</code> value to monitor Azure resource groups.
Resource Name	Yes	<event-publisher-Azure-resource-name>	Select the Azure resource name for the event publisher. This list varies based on the resource type that you selected. For this tutorial, select the name for the Azure resource group that includes your virtual machine.

PROPERTY	REQUIRED	VALUE	DESCRIPTION
Event Type Item	No	<event-types>	<p>Select one or more specific event types to filter and send to your event grid. For example, you can optionally add these event types to detect when resources are changed or deleted:</p> <ul style="list-style-type: none"> <li>- Microsoft.Resources.ResourceActionSuccess</li> <li>- Microsoft.Resources.ResourceDeleteSuccess</li> <li>- Microsoft.Resources.ResourceWriteSuccess</li> </ul> <p>For more information, see these topics:</p> <ul style="list-style-type: none"> <li>- <a href="#">Azure Event Grid event schema for resource groups</a></li> <li>- <a href="#">Understand event filtering</a></li> <li>- <a href="#">Filter events for Event Grid</a></li> </ul>
To add optional properties, select <b>Add new parameter</b> , and then select the properties that you want.	No	{see descriptions}	<p>* <b>Prefix Filter:</b> For this tutorial, leave this property empty. The default behavior matches all values. However, you can specify a prefix string as a filter, for example, a path and a parameter for a specific resource.</p> <p>* <b>Suffix Filter:</b> For this tutorial, leave this property empty. The default behavior matches all values. However, you can specify a suffix string as a filter, for example, a file name extension, when you want only specific file types.</p> <p>* <b>Subscription Name:</b> For this tutorial, you can provide a unique name for your event subscription.</p>

- Save your logic app. On the designer toolbar, select **Save**. To collapse and hide an action's details in your logic app, select the action's title bar.



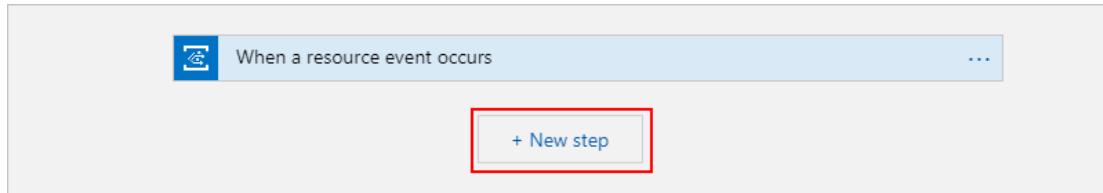
When you save your logic app with an event grid trigger, Azure automatically creates an event subscription for your logic app to your selected resource. So when the resource publishes an event to the event grid, that event grid automatically pushes the event to your logic app. This event triggers your logic app, then creates and runs an instance of the workflow that you define in these next steps.

Your logic app is now live and listens to events from the event grid, but doesn't do anything until you add actions to the workflow.

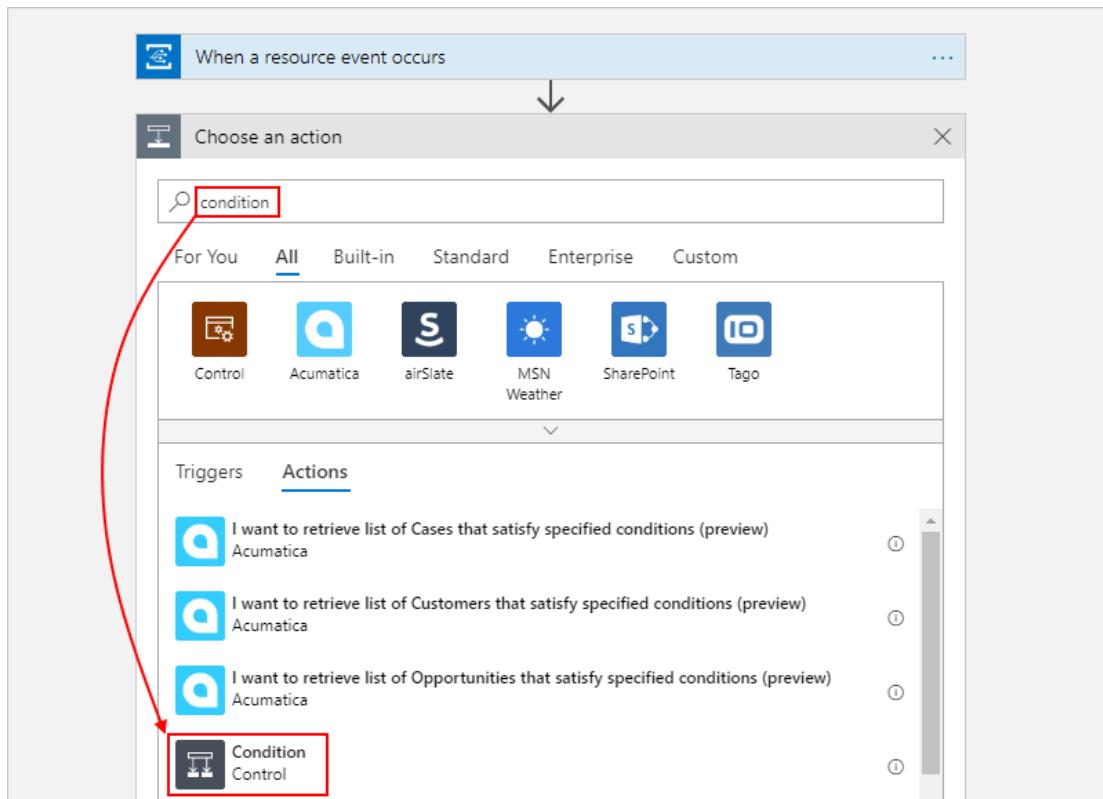
## Add a condition

If you want to your logic app to run only when a specific event or operation happens, add a condition that checks for the `Microsoft.Compute/virtualMachines/write` operation. When this condition is true, your logic app sends you email with details about the updated virtual machine.

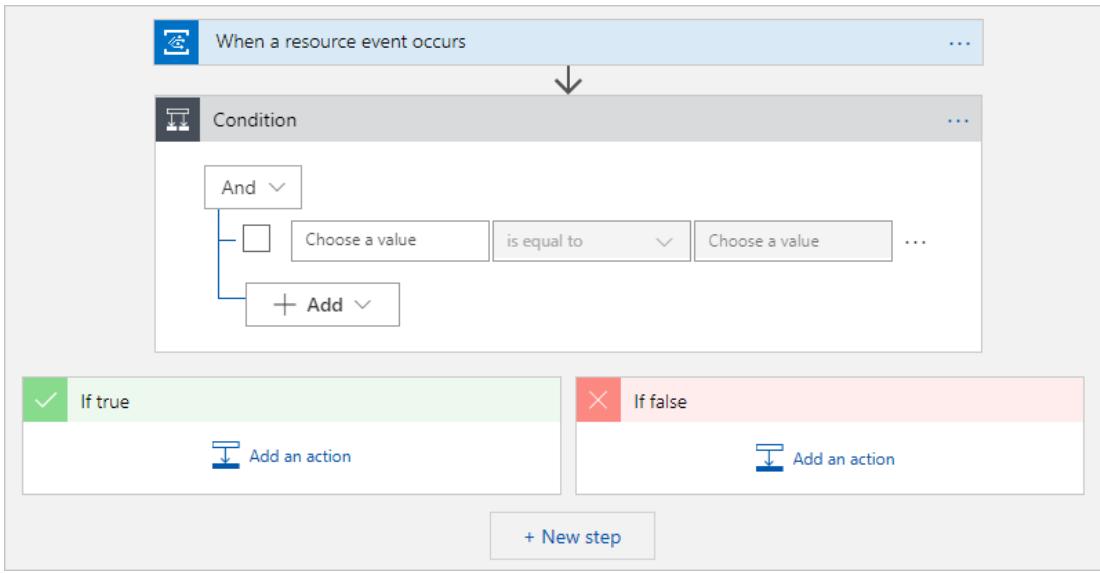
1. In Logic App Designer, under the event grid trigger, select **New step**.



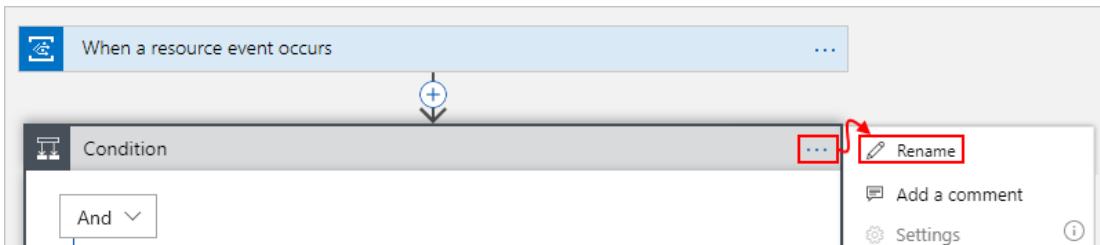
2. Under **Choose an action**, in the search box, enter `condition` as your filter. From the actions list, select the **Condition** action.



The Logic App Designer adds an empty condition to your workflow, including action paths to follow based whether the condition is true or false.

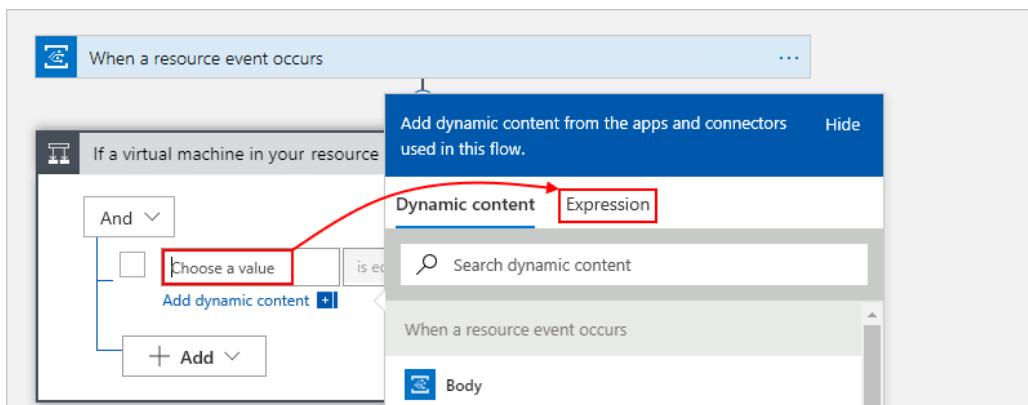


3. Rename the condition title to `If a virtual machine in your resource group has changed`. On the condition's title bar, select the ellipses (...) button, and select **Rename**.



4. Create a condition that checks the event `body` for a `data` object where the `operationName` property is equal to the `Microsoft.Compute/virtualMachines/write` operation. Learn more about [Event Grid event schema](#).

- On the first row under **And**, click inside the left box. In the dynamic content list that appears, select **Expression**.



- In the expression editor, enter this expression, which returns the operation name from the trigger, and select **OK**:

```
triggerBody()['data']['operationName']
```

For example:

The screenshot shows the Logic App designer with a condition step selected. A modal window is open, titled 'Add an expression to do basic things like access, convert, and compare values. [Learn more](#)', with the 'Expression' tab selected. Inside, the expression `triggerBody()['data']['operationName']` is displayed, with the entire code block highlighted by a red box. Below the expression, there are tabs for 'String functions' and 'See more'.

- In the middle box, keep the operator is **equal to**.
- In the right box, enter this value, which is the specific operation that you want to monitor:

`Microsoft.Compute/virtualMachines/write`

Your finished condition now looks like this example:

The screenshot shows the Logic App designer with the condition step completed. The expression `triggerBody()['data']['operationName']` has been resolved to `data.operationName`, which is now displayed in the condition's right-hand value field. The condition step is now fully configured.

If you switch from design view to code view and back to design view, the expression that you specified in the condition resolves to the **data.operationName** token:

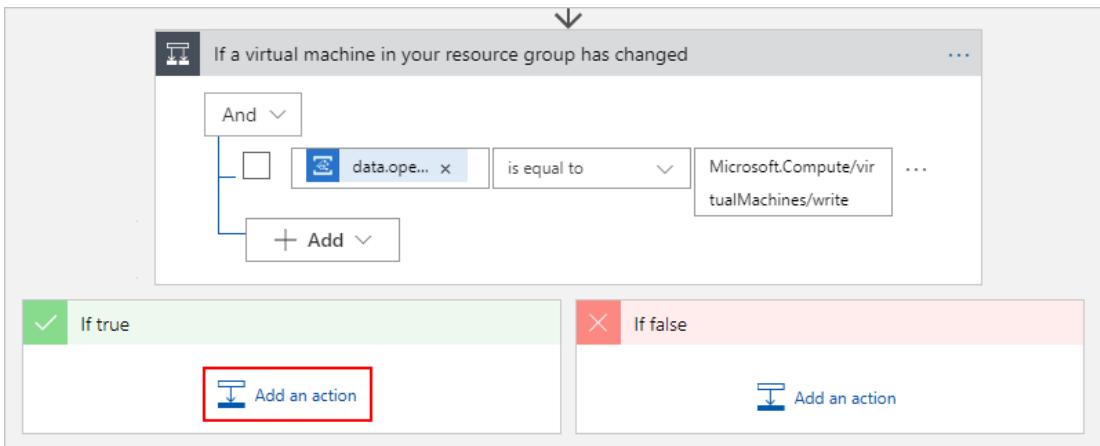
The screenshot shows the Logic App designer with the condition step in code view. The expression `triggerBody()['data']['operationName']` has been resolved to `data.operationName`, which is now displayed in the condition's right-hand value field. The condition step is now fully configured.

- Save your logic app.

## Send email notifications

Now add an *action* so that you can receive an email when the specified condition is true.

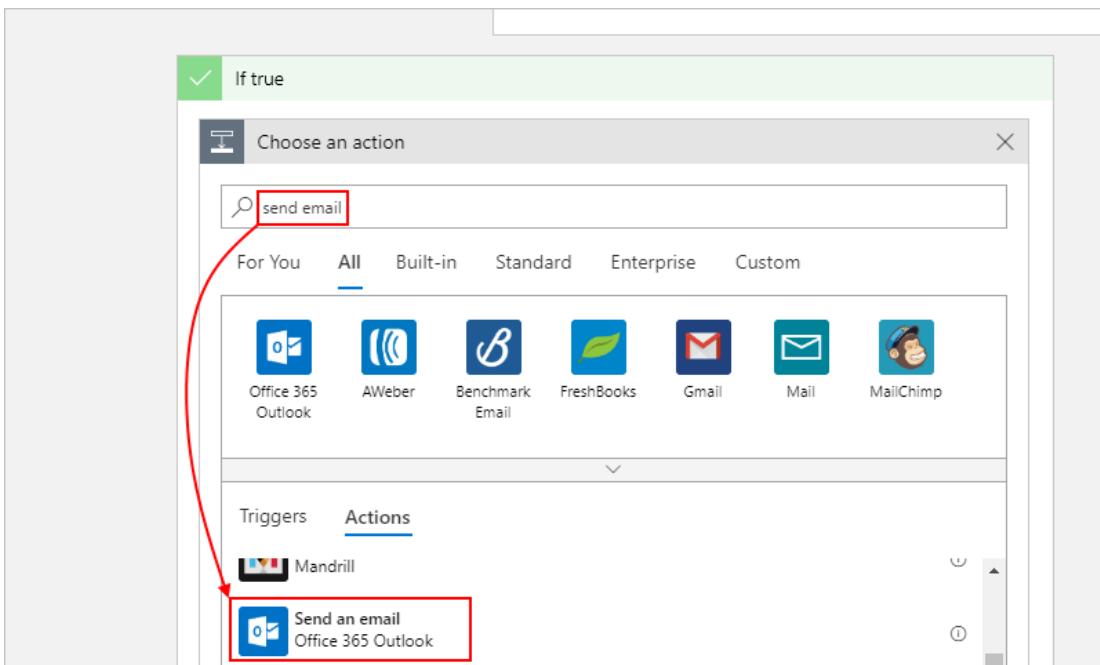
1. In the condition's **If true** box, select **Add an action**.



2. Under **Choose an action**, in the search box, enter **send an email** as your filter. Based on your email provider, find and select the matching connector. Then select the "send email" action for your connector. For example:

- For an Azure work or school account, select the Office 365 Outlook connector.
- For personal Microsoft accounts, select the Outlook.com connector.
- For Gmail accounts, select the Gmail connector.

This tutorial continues with the Office 365 Outlook connector. If you use a different provider, the steps remain the same, but your UI might appear slightly different.



3. If you don't already have a connection for your email provider, sign in to your email account when you're asked for authentication.
4. Rename the send email action to this title: **Send email when virtual machine updated**
5. Provide information about the email as specified in the following table:

If true

**Send email when virtual machine updated**

\* To Sophia.Owen@fabrikam.com

\* Subject Resource updated:

Add dynamic content +

\* Body Specify the body of the mail

Add new parameter

Connected to Sophia.Owen@fabrikam.com. [Change connection.](#)

**Add an action**

**Dynamic content** Expression

Search dynamic content

When a resource event occurs

- Event Time Time of the event.
- Event Type Type of the event.
- ID ID for the event.
- Subject** Subject of the event.

#### TIP

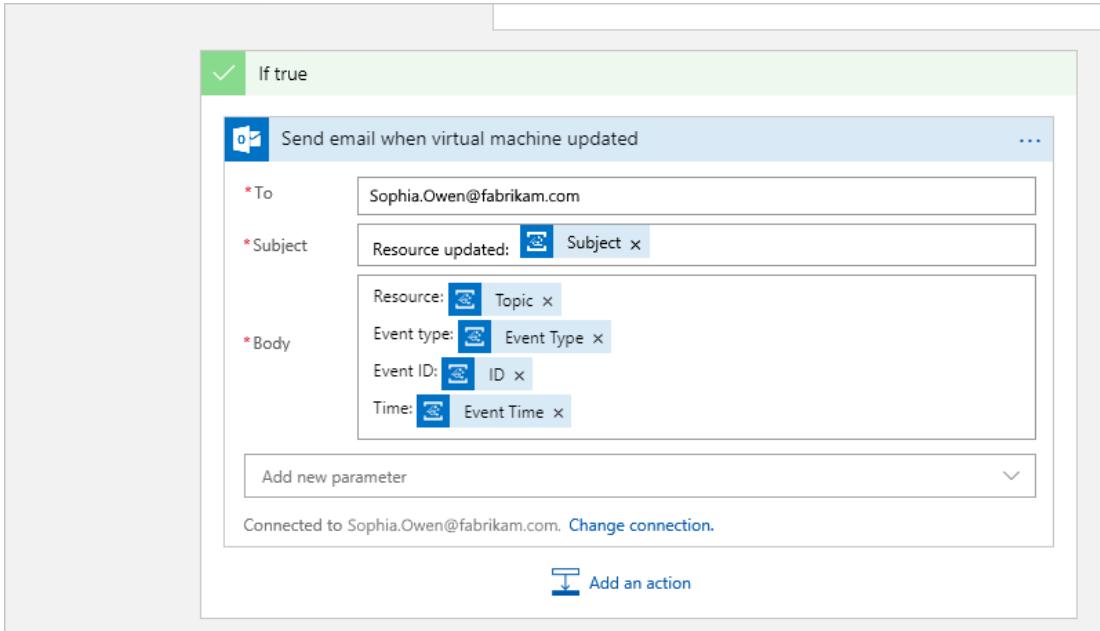
To select output from the previous steps in your workflow, click inside an edit box so that the dynamic content list appears, or select **Add dynamic content**. For more results, select **See more** for each section in the list. To close the dynamic content list, select **Add dynamic content** again.

PROPERTY	REQUIRED	VALUE	DESCRIPTION
To	Yes	<recipient@domain>	Enter the recipient's email address. For testing purposes, you can use your own email address.
Subject	Yes	Resource updated: Subject	Enter the content for the email's subject. For this tutorial, enter the specified text, and select the event's <b>Subject</b> field. Here, your email subject includes the name for the updated resource (virtual machine).
Body	Yes	Resource: Topic Event type: Event Type Event ID: ID Time: Event Time	Enter the content for the email's body. For this tutorial, enter the specified text and select the event's <b>Topic</b> , <b>Event Type</b> , <b>ID</b> , and <b>Event Time</b> fields so that your email includes the resource that fired the event, event type, event timestamp, and event ID for the update. For this tutorial, the resource is the Azure resource group selected in the trigger.  To add blank lines in your content, press Shift + Enter.

**NOTE**

If you select a field that represents an array, the designer automatically adds a **For each** loop around the action that references the array. That way, your logic app performs that action on each array item.

Now, your email action might look like this example:



And your finished logic app might look like this example:

The screenshot shows the Azure Logic App designer interface. At the top, there is a trigger card titled "When a resource event occurs". It has three configuration fields: "Subscription" set to "Pay-As-You-Go", "Resource Type" set to "Microsoft.Resources.ResourceGroups", and "Resource Name" set to "My-Virtual-Machine-RG". Below the trigger is a dashed box labeled "Event Type Item - 1" with a "Add new item" button. Underneath the trigger is a "Subscription Name" field containing "fabrikam-monitor-virtual-machine-event-subscription" with a delete icon. A "Connected to Sophia.Owen@fabrikam.com" message with a "Change connection" link is also present.

A downward arrow points from the trigger to the next card, which is titled "If a virtual machine in your resource group has changed". This card contains a condition "And" followed by a comparison: "data.operation" is equal to "Microsoft.Compute/virtualMachines/write". There is also a "+ Add" button.

The logic app then branches into two paths:

- If true:** A green box containing the action "Send email when virtual machine updated". It has fields for "To" (Sophia.Owen@fabrikam.com), "Subject" (Resource updated: [Topic]), and "Body" (Resource: [Topic], Event type: [Event Type], Event ID: [ID], Time: [Event Time]). Below the body fields is an "Add new parameter" button. A "Connected to Sophia.Owen@fabrikam.com" message with a "Change connection" link is at the bottom.
- If false:** A red box containing a placeholder "Add an action" button.

6. Save your logic app. To collapse and hide each action's details in your logic app, select the action's title bar.

Your logic app is now live, but waits for changes to your virtual machine before doing anything. To test your logic app now, continue to the next section.

## Test your logic app workflow

1. To check that your logic app is getting the specified events, update your virtual machine.

For example, you can resize your virtual machine in the Azure portal or [resize your VM with Azure PowerShell](#).

After a few moments, you should get an email. For example:

Reply Reply All Forward IM  
**SO** Tue 8/15/2017 1:56 PM  
 Sophia Owen  
 Resource updated: /subscriptions/00000000000000000000/resourcegroups  
   /My-Virtual-Machine-RG/providers/Microsoft.Compute/virtualMachines/fabrikam-vm-test  
 To: Sophia Owen  
 This message was sent with Low importance.  
 Resource: /subscriptions/00000000000000000000/resourcegroups/My-Virtual-Machine-RG  
 Event type: Microsoft.Resources.ResourceWriteSuccess  
 Event ID: 63fd13ed-70a7-4954-8532-0e39a32f1e20  
 Time: 2017-08-15T20:55:33.8571295Z

2. To review the runs and trigger history for your logic app, on your logic app menu, select **Overview**. To view more details about a run, select the row for that run.

STATUS	START TIME	IDENTIFIER	DURATION
Succeeded	8/15/2017 ...	08586987723215706036981692672	1.02 Seconds
Succeeded	8/15/2017 ...	08586987723316430403123174944	2.02 Seconds
Succeeded	8/15/2017 ...	08586987763320045914957257322	467 Milliseconds

3. To view the inputs and outputs for each step, expand the step that you want to review. This information can help you diagnose and debug problems in your logic app.

The screenshot shows two adjacent Azure Logic App pages. On the left is the 'Runs history' page for 'MyLogicApp', displaying a list of recent runs with columns for 'Start time' and 'Duration'. The most recent run is selected. On the right is the 'Logic app run' details page for the same run. It shows the logic app's workflow: it triggers when a resource event occurs (0s delay) and checks if a virtual machine in a resource group has a specific status (1s delay). If true, it sends an email. The 'If true' section is highlighted in green, and the 'If false' section is highlighted in red.

Congratulations, you've created and run a logic app that monitors resource events through an event grid and emails you when those events happen. You also learned how easily you can create workflows that automate processes and integrate systems and cloud services.

You can monitor other configuration changes with event grids and logic apps, for example:

- A virtual machine gets Azure role-based access control (Azure RBAC) rights.
- Changes are made to a network security group (NSG) on a network interface (NIC).
- Disks for a virtual machine are added or removed.
- A public IP address is assigned to a virtual machine NIC.

## Clean up resources

This tutorial uses resources and performs actions that incur charges on your Azure subscription. So when you're done with the tutorial and testing, make sure that you disable or delete any resources where you don't want to incur charges.

- To stop running your logic app without deleting your work, disable your app. On your logic app menu, select **Overview**. On the toolbar, select **Disable**.

The screenshot shows the 'Overview' page for a logic app named '<logic-app-name>'. The toolbar includes 'Run Trigger', 'Refresh', 'Edit', 'Delete', and 'Disable' (which is highlighted with a red box). Below the toolbar, there are sections for 'Essentials' (Summary, Activity log), 'Actions' (Trigger), and a 'Actions' button. The 'Overview' tab is also highlighted with a red box.

### TIP

If you don't see the logic app menu, try returning to the Azure dashboard, and reopen your logic app.

- To permanently delete your logic app, on the logic app menu, select **Overview**. On the toolbar, select **Delete**. Confirm that you want to delete your logic app, and select **Delete**.

## Next steps

- [Create and route custom events with Event Grid](#)

See the following samples to learn about publishing events to and consuming events from Event Grid using different programming languages.

- [Azure Event Grid samples for .NET](#)
- [Azure Event Grid samples for Java](#)
- [Azure Event Grid samples for Python](#)
- [Azure Event Grid samples for JavaScript](#)
- [Azure Event Grid samples for TypeScript](#)

# Tutorial: Automate resizing uploaded images using Event Grid

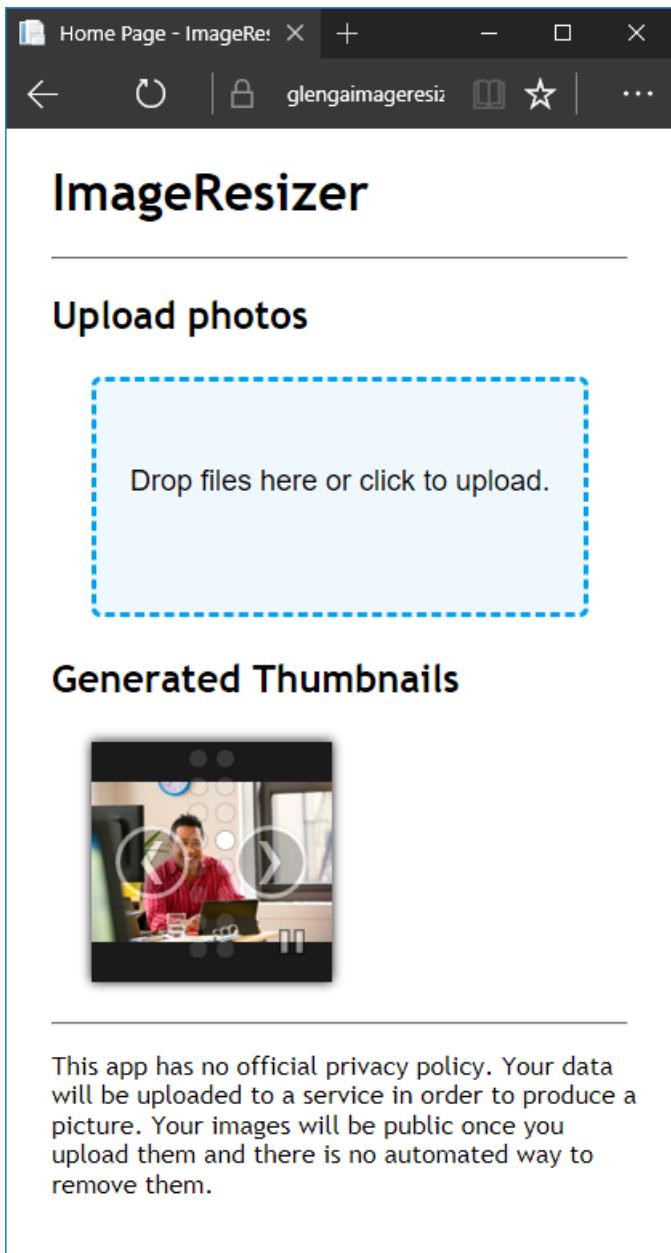
5/25/2021 • 9 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. Event Grid enables you to create subscriptions to events raised by Azure services or third-party resources.

This tutorial is part two of a series of Storage tutorials. It extends the [previous Storage tutorial](#) to add serverless automatic thumbnail generation using Azure Event Grid and Azure Functions. Event Grid enables [Azure Functions](#) to respond to [Azure Blob storage](#) events and generate thumbnails of uploaded images. An event subscription is created against the Blob storage create event. When a blob is added to a specific Blob storage container, a function endpoint is called. Data passed to the function binding from Event Grid is used to access the blob and generate the thumbnail image.

You use the Azure CLI and the Azure portal to add the resizing functionality to an existing image upload app.

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)



In this tutorial, you learn how to:

- Create an Azure Storage account
- Deploy serverless code using Azure Functions
- Create a Blob storage event subscription in Event Grid

## Prerequisites

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

To complete this tutorial:

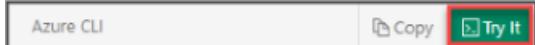
You must have completed the previous Blob storage tutorial: [Upload image data in the cloud with Azure Storage](#).

You need an [Azure subscription](#). This tutorial doesn't work with the free subscription.

# Use Azure Cloud Shell

Azure hosts Azure Cloud Shell, an interactive shell environment that you can use through your browser. You can use either Bash or PowerShell with Cloud Shell to work with Azure services. You can use the Cloud Shell preinstalled commands to run the code in this article without having to install anything on your local environment.

To start Azure Cloud Shell:

OPTION	EXAMPLE/LINK
Select Try It in the upper-right corner of a code block. Selecting Try It doesn't automatically copy the code to Cloud Shell.	
Go to <a href="https://shell.azure.com">https://shell.azure.com</a> , or select the Launch Cloud Shell button to open Cloud Shell in your browser.	
Select the Cloud Shell button on the menu bar at the upper right in the <a href="#">Azure portal</a> .	

To run the code in this article in Azure Cloud Shell:

1. Start Cloud Shell.
2. Select the **Copy** button on a code block to copy the code.
3. Paste the code into the Cloud Shell session by selecting **Ctrl+Shift+V** on Windows and Linux or by selecting **Cmd+Shift+V** on macOS.
4. Select **Enter** to run the code.

If you choose to install and use the CLI locally, this tutorial requires the Azure CLI version 2.0.14 or later. Run `az --version` to find the version. If you need to install or upgrade, see [Install Azure CLI](#).

If you are not using Cloud Shell, you must first sign in using `az login`.

If you've not previously registered the Event Grid resource provider in your subscription, make sure it's registered.

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

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

## Create an Azure Storage account

Azure Functions requires a general storage account. In addition to the Blob storage account you created in the previous tutorial, create a separate general storage account in the resource group by using the [az storage account create](#) command. Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.

1. Set a variable to hold the name of the resource group that you created in the previous tutorial.

```
resourceGroupName="myResourceGroup"
```

```
$resourceGroupName="myResourceGroup"
```

- Set a variable to hold the location for resources to be created.

```
location="eastus"
```

```
$location="eastus"
```

- Set a variable for the name of the new storage account that Azure Functions requires.

```
functionstorage="<name of the storage account to be used by the function>"
```

```
$functionstorage="<name of the storage account to be used by the function>"
```

- Create the storage account for the Azure function.

```
az storage account create --name $functionstorage --location $location \
--resource-group $resourceGroupName --sku Standard_LRS --kind StorageV2
```

```
az storage account create --name $functionstorage --location $location \
--resource-group $resourceGroupName --sku Standard_LRS --kind StorageV2
```

## Create a function app

You must have a function app to host the execution of your function. The function app provides an environment for serverless execution of your function code. Create a function app by using the [az functionapp create](#) command.

In the following command, provide your own unique function app name. The function app name is used as the default DNS domain for the function app, and so the name needs to be unique across all apps in Azure.

- Specify a name for the function app that's to be created.

```
functionapp="<name of the function app>"
```

```
$functionapp="<name of the function app>"
```

- Create the Azure function.

```
az functionapp create --name $functionapp --storage-account $functionstorage \
--resource-group $resourceGroupName --consumption-plan-location $location \
--functions-version 2
```

```
az functionapp create --name $functionapp --storage-account $functionstorage \
--resource-group $resourceGroupName --consumption-plan-location $location \
--functions-version 2
```

Now configure the function app to connect to the Blob storage account you created in the [previous tutorial](#).

## Configure the function app

The function needs credentials for the Blob storage account, which are added to the application settings of the function app using the `az functionapp config appsettings set` command.

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

```
storageConnectionString=$(az storage account show-connection-string --resource-group $resourceGroupName \
--name $blobStorageAccount --query connectionString --output tsv)

az functionapp config appsettings set --name $functionapp --resource-group $resourceGroupName \
--settings AzureWebJobsStorage=$storageConnectionString THUMBNAIL_CONTAINER_NAME=thumbnails \
THUMBNAIL_WIDTH=100 FUNCTIONS_EXTENSION_VERSION=~2
```

```
$storageConnectionString=$(az storage account show-connection-string --resource-group $resourceGroupName \
--name $blobStorageAccount --query connectionString --output tsv)

az functionapp config appsettings set --name $functionapp --resource-group $resourceGroupName \
--settings AzureWebJobsStorage=$storageConnectionString THUMBNAIL_CONTAINER_NAME=thumbnails \
THUMBNAIL_WIDTH=100 FUNCTIONS_EXTENSION_VERSION=~2
```

The `FUNCTIONS_EXTENSION_VERSION=~2` setting makes the function app run on version 2.x of the Azure Functions runtime.

You can now deploy a function code project to this function app.

## Deploy the function code

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

The sample C# resize function is available on [GitHub](#). Deploy this code project to the function app by using the `az functionapp deployment source config` command.

```
az functionapp deployment source config --name $functionapp --resource-group $resourceGroupName \
--branch master --manual-integration \
--repo-url https://github.com/Azure-Samples/function-image-upload-resize
```

```
az functionapp deployment source config --name $functionapp --resource-group $resourceGroupName \
--branch master --manual-integration \
--repo-url https://github.com/Azure-Samples/function-image-upload-resize
```

The image resize function is triggered by HTTP requests sent to it from the Event Grid service. You tell Event Grid that you want to get these notifications at your function's URL by creating an event subscription. For this tutorial you subscribe to blob-created events.

The data passed to the function from the Event Grid notification includes the URL of the blob. That URL is in turn passed to the input binding to obtain the uploaded image from Blob storage. The function generates a thumbnail image and writes the resulting stream to a separate container in Blob storage.

This project uses `EventGridTrigger` for the trigger type. Using the Event Grid trigger is recommended over

generic HTTP triggers. Event Grid automatically validates Event Grid Function triggers. With generic HTTP triggers, you must implement the [validation response](#).

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

To learn more about this function, see the [function.json and run.csx files](#).

The function project code is deployed directly from the public sample repository. To learn more about deployment options for Azure Functions, see [Continuous deployment for Azure Functions](#).

## Create an event subscription

An event subscription indicates which provider-generated events you want sent to a specific endpoint. In this case, the endpoint is exposed by your function. Use the following steps to create an event subscription that sends notifications to your function in the Azure portal:

1. In the [Azure portal](#), at the top of the page search for and select **Function App** and choose the function app that you just created. Select **Functions** and choose the **Thumbnail** function.

A screenshot of the Azure portal's Functions blade. At the top, there's a search bar and several action buttons: Add, Develop Locally, Refresh, Enable, Disable, and Delete. Below the search bar is a filter bar with a search icon and the text 'Filter by name...'. The main area shows a list of functions under the 'Functions' category. The 'Thumbnail' function is listed and highlighted with a red box. Other items in the list include 'App keys', 'App files', and 'Proxies'. The list is sorted by Name (Thumbnail) and Trigger (EventGrid), and is currently Enabled.

2. Select **select Integration** then choose the **Event Grid Trigger** and select **Create Event Grid subscription**.

A screenshot of the Azure portal's Integration blade for the 'Thumbnail' function. On the left, there's a sidebar with 'Overview', 'Developer' (Code + Test, Integration, Monitor, Function Keys), and 'Inputs' (No inputs defined). The 'Integration' section is selected and highlighted with a red box. It contains a 'Trigger' section with the 'Event Grid Trigger (eventGridEvent)' option highlighted with a red box. On the right, there's an 'Edit Trigger' pane with 'Save', 'Discard', and 'Delete' buttons. It shows the 'Binding Type' as 'Event Grid Trigger', the 'Event Trigger parameter name' as 'eventGridEvent', and a 'Create Event Grid subscription' button highlighted with a red box.

3. Use the event subscription settings as specified in the table.

# Create Event Subscription



Event Grid

[Basic](#)   [Filters](#)   [Additional Features](#)

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

## EVENT SUBSCRIPTION DETAILS

Name

imageresizesub



Event Schema

Event Grid Schema



## TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type



Storage account

Source Resource

spblogstorageaccount0608 [\(change\)](#)

System Topic Name

imagestoragesystopic



## EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

2 selected



## ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type

Azure Function [\(change\)](#)

Endpoint

[Thumbnail](#) [\(change\)](#)[Create](#)

SETTING	SUGGESTED VALUE	DESCRIPTION
Name	imageresizersub	Name that identifies your new event subscription.
Topic type	Storage accounts	Choose the Storage account event provider.
Subscription	Your Azure subscription	By default, your current Azure subscription is selected.
Resource group	myResourceGroup	Select <b>Use existing</b> and choose the resource group you have been using in this tutorial.
Resource	Your Blob storage account	Choose the Blob storage account you created.
System Topic Name	imagestoragesystopic	Specify a name for the system topic. To learn about system topics, see <a href="#">System topics overview</a> .

SETTING	SUGGESTED VALUE	DESCRIPTION
Event types	Blob created	Uncheck all types other than <b>Blob created</b> . Only event types of <code>Microsoft.Storage.BlobCreated</code> are passed to the function.
Endpoint type	autogenerated	Pre-defined as <b>Azure Function</b> .
Endpoint	autogenerated	Name of the function. In this case, it's <b>Thumbnail</b> .

4. Switch to the **Filters** tab, and do the following actions:

- a. Select **Enable subject filtering** option.
- b. For **Subject begins with**, enter the following value :  
`/blobServices/default/containers/images/`.

**Create Event Subscription**  
Event Grid

**Basic**   **Filters**   **Additional Features**

**SUBJECT FILTERS**  
Apply filters to the subject of each event. Only events with matching subjects get delivered. [Learn more](#)

Enable subject filtering

Subject Begins With

Subject Ends With

Case-sensitive subject matching

**ADVANCED FILTERS**  
Filter on attributes of each event. Only events that match all filters get delivered. Up to 5 filters can be specified. All string comparisons are case-insensitive. [Learn more](#)

Valid keys for currently selected event schema:

- id, topic, subject, eventtype, dataversion
- Custom properties at most one level inside the data payload, using "." as the nesting separator. (e.g. data, data.key are valid, data.key.key is not)

KEY	OPERATOR	VALUE
No results		
<a href="#">Add new filter</a>		

**Create**

5. Select **Create** to add the event subscription. This creates an event subscription that triggers the `Thumbnail` function when a blob is added to the `images` container. The function resizes the images and adds them to the `thumbnails` container.

Now that the backend services are configured, you test the image resize functionality in the sample web app.

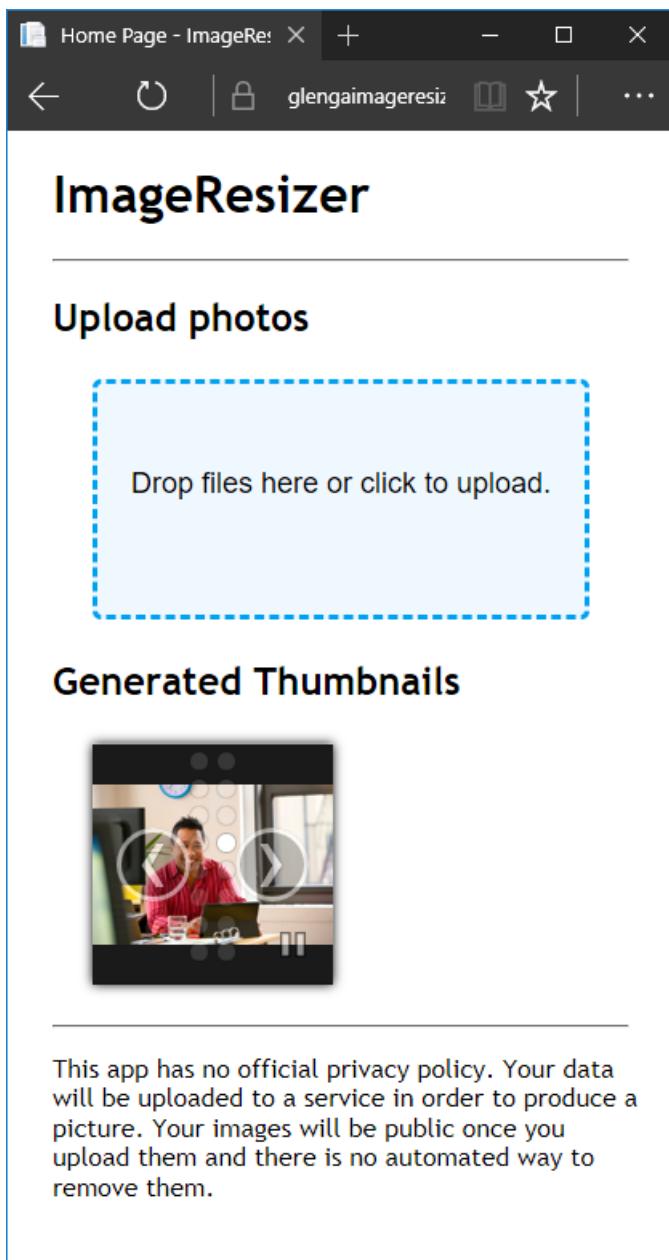
## Test the sample app

To test image resizing in the web app, browse to the URL of your published app. The default URL of the web app is `https://<web_app>.azurewebsites.net`.

- [.NET v12 SDK](#)
- [Node.js v10 SDK](#)

Click the **Upload photos** region to select and upload a file. You can also drag a photo to this region.

Notice that after the uploaded image disappears, a copy of the uploaded image is displayed in the **Generated Thumbnails** carousel. This image was resized by the function, added to the *thumbnails* container, and downloaded by the web client.



## Next steps

In this tutorial, you learned how to:

- Create a general Azure Storage account
- Deploy serverless code using Azure Functions
- Create a Blob storage event subscription in Event Grid

Advance to part three of the Storage tutorial series to learn how to secure access to the storage account.

[Secure access to an applications data in the cloud](#)

- To learn more about Event Grid, see [An introduction to Azure Event Grid](#).

- To try another tutorial that features Azure Functions, see [Create a function that integrates with Azure Logic Apps.](#)

# Tutorial: Integrate Azure Automation with Event Grid and Microsoft Teams

11/2/2020 • 3 minutes to read • [Edit Online](#)

In this tutorial, you learn how to:

- Import an Event Grid sample runbook.
- Create an optional Microsoft Teams webhook.
- Create a webhook for the runbook.
- Create an Event Grid subscription.
- Create a VM that triggers the runbook.

If you don't have an Azure subscription, create a [free account](#) before you begin.

## Prerequisites

### IMPORTANT

Using this Azure feature from PowerShell requires the `AzureRM` module installed. This is an older module only available for Windows PowerShell 5.1 that no longer receives new features. The `Az` and `AzureRM` modules are **not** compatible when installed for the same versions of PowerShell. If you need both versions:

1. [Uninstall the Az module](#) from a PowerShell 5.1 session.
2. [Install the AzureRM module](#) from a PowerShell 5.1 session.
3. [Download and install PowerShell Core 6.x or later](#).
4. [Install the Az module](#) in a PowerShell Core session.

To complete this tutorial, an [Azure Automation account](#) is required to hold the runbook that is triggered from the Azure Event Grid subscription.

- The `AzureRM.Tags` module needs to be loaded in your Automation Account, see [How to import modules in Azure Automation](#) to learn how to import modules into Azure Automation.

## Import an Event Grid sample runbook

1. Select your Automation account, and select the [Runbooks](#) page.

NAME	AUTHORING STATUS	LAST MODIFIED	TAGS
AzureAutomationTutorial	✓ Published	8/14/2018, 9:27 AM	
AzureAutomationTutorialPython2	✓ Published	8/14/2018, 9:27 AM	
AzureAutomationTutorialScript	✓ Published	8/14/2018, 9:27 AM	
AzureClassicAutomationTutorial	✓ Published	8/14/2018, 9:27 AM	
AzureClassicAutomationTutorial...	✓ Published	8/14/2018, 9:27 AM	

2. Select the **Browse gallery** button.
3. Search for **Event Grid**, and select **Integrating Azure Automation with Event grid**.

**Integrating Azure Automation with Event grid**  
PowerShell Runbook  
This sample Automation runbook integrates with Azure event grid subscriptions to get notified when a write command is performed against an Azure VM. The runbook adds a cost tag to the VM if it doesn't exist. It also sends an optional notification to a Microsoft Tag: [Azure Automation](#), [Microsoft Azure Virtual Machines](#), [Event Grid](#)  
Created by: SC Automation Product Team  
417 downloads  
Last updated: 11/28/2017

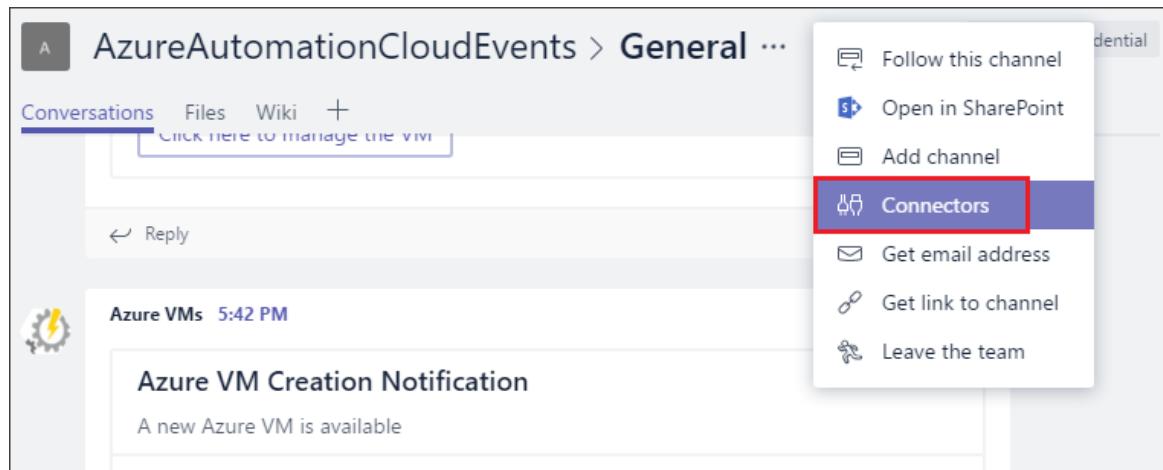
4. Select **Import** and name it **Watch-VMWrite**.
5. After it has imported, select **Edit** to view the runbook source.
6. Update the line 74 in the script to use `Tag` instead of `Tags`.

```
Update-AzureRmVM -ResourceGroupName $VMResourceGroup -VM $VM -Tag $Tag | Write-Verbose
```

7. Select the **Publish** button.

## Create an optional Microsoft Teams webhook

1. In Microsoft Teams, select **More Options** next to the channel name, and then select **Connectors**.



2. Scroll through the list of connectors to **Incoming Webhook**, and select **Add**.
3. Enter **AzureAutomationIntegration** for the name, and select **Create**.
4. Copy the webhook URL to the clipboard, and save it. The webhook URL is used to send information to Microsoft Teams.
5. Select **Done** to save the webhook.

## Create a webhook for the runbook

1. Open the Watch-VMWrite runbook.
2. Select **Webhooks**, and select the **Add Webhook** button.
3. Enter **WatchVMEventGrid** for the name. Copy the URL to the clipboard, and save it.

Two overlapping windows are shown. The left window is titled 'Add Webhook' and the right one is 'Create a new webhook'. Both windows have a header with a star, a close button, and a gear icon. The 'Create a new webhook' window is active. It contains a warning message: 'For security, after creating a webhook its URL can't be viewed. Make sure to copy it before pressing "OK", and to store it securely.' Below this are fields: 'Name' set to 'WatchVMEventGrid' with a green checkmark; 'Enabled' set to 'Yes'; 'Expires' set to '2018-12-06 10:12:36 AM'; and a 'URL' field containing 'https://s5events.azure-automation.net...' with a copy icon highlighted by a red box.

4. Select **Configure parameters and run settings**, and enter the Microsoft Teams webhook URL for **CHANNELURL**. Leave **WEBHOOKDATA** blank.

The screenshot shows the 'Add Webhook' configuration interface. On the left, under 'Webhook', the 'WatchVMEventGrid' runbook is selected. Under 'Parameters and run settings', 'Configure parameters and run settings' is chosen. On the right, the 'Parameters' tab is active, displaying the 'WEBHOOKDATA' parameter with a value of 'No value' and the 'CHANNELURL' parameter with a value of 'https://outlook.office.com/webhook/52af4b29'. A 'Run Settings' section is also visible.

- Select **Create** to create the Automation runbook webhook.

## Create an Event Grid subscription

- On the **Automation Account** overview page, select **Event grid**.

The screenshot shows the 'Event grid' blade in the Azure portal. The left sidebar lists 'Variables', 'RELATED RESOURCES' (with 'Event grid' highlighted), 'Start/Stop VM', 'ACCOUNT SETTINGS' (Properties, Keys, Pricing, Source control, Run as accounts), and a search bar. The main area displays a blue circuit board icon and a message: 'No Event Subscriptions Found! Consider modifying your search parameters above.' It includes a note about enabling event-based programming and a 'Create one' button.

- Click **+ Event Subscription**.
- Configure the subscription with the following information:
  - For **Topic Type**, select **Azure Subscriptions**.
  - Uncheck the **Subscribe to all event types** check box.
  - Enter **AzureAutomation** for the name.
  - In the **Defined Event Types** drop-down, uncheck all options except **Resource Write Success**.

### NOTE

Azure Resource Manager does not currently differentiate between Create and Update, so implementing this tutorial for all Microsoft.Resources.ResourceWriteSuccess events in your Azure Subscription could result in a high volume of calls.

- For **Endpoint Type**, select **Webhook**.
- Click **Select an endpoint**. On the **Select Web Hook** page that opens up, paste the webhook url you created for the Watch-VMWrite runbook.

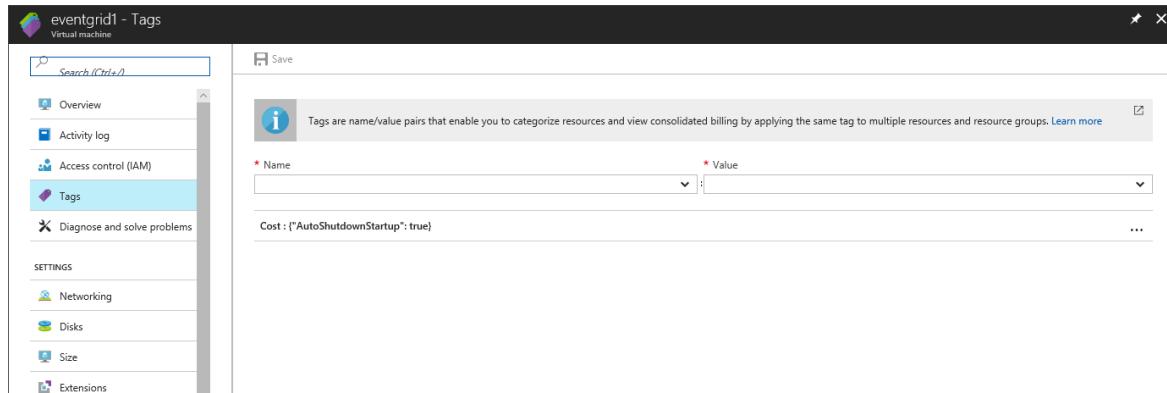
- g. Under **FILTERS**, enter the subscription and resource group where you want to look for the new VMs created. It should look like:

```
/subscriptions/<subscription-id>/resourcegroups/<resource-group-name>/providers/Microsoft.Compute/virtualMachines
```

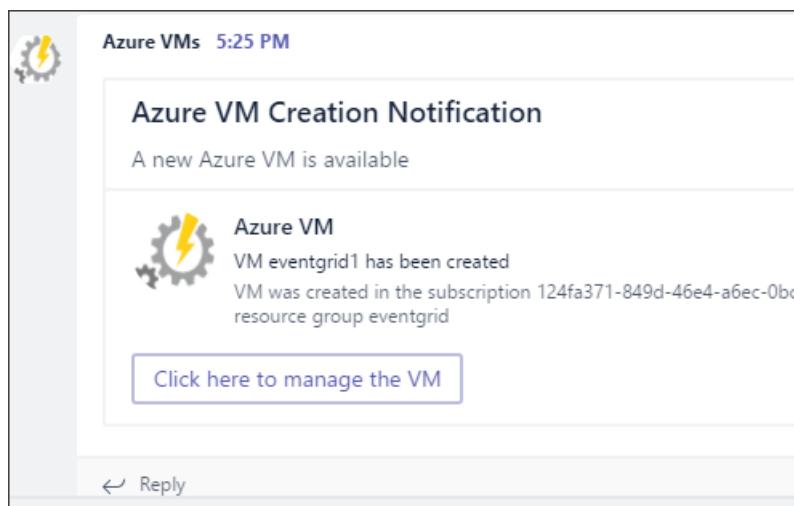
4. Select **Create** to save the Event Grid subscription.

## Create a VM that triggers the runbook

1. Create a new VM in the resource group you specified in the Event Grid subscription prefix filter.
2. The Watch-VMWrite runbook should be called and a new tag added to the VM.



3. A new message is sent to the Microsoft Teams channel.



## Next steps

In this tutorial, you set up integration between Event Grid and Automation. You learned how to:

- Import an Event Grid sample runbook.
- Create an optional Microsoft Teams webhook.
- Create a webhook for the runbook.
- Create an Event Grid subscription.
- Create a VM that triggers the runbook.

[Create and route custom events with Event Grid](#)

# Tutorial: Send email notifications about Azure IoT Hub events using Event Grid and Logic Apps

3/5/2021 • 7 minutes to read • [Edit Online](#)

Azure Event Grid enables you to react to events in IoT Hub by triggering actions in your downstream business applications.

This article walks through a sample configuration that uses IoT Hub and Event Grid. At the end, you have an Azure logic app set up to send a notification email every time a device connects or disconnects to your IoT hub. Event Grid can be used to get timely notification about critical devices disconnecting. Metrics and Diagnostics can take several (i.e. 20 or more -- though we don't want to put a number on it) minutes to show up in logs/alerts. That might be unacceptable for critical infrastructure.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Prerequisites

- An email account from any email provider that is supported by Azure Logic Apps, such as Office 365 Outlook or Outlook.com. This email account is used to send the event notifications.
- Use the Bash environment in [Azure Cloud Shell](#).

 [Launch Cloud Shell](#)

- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the [az login](#) command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run [az version](#) to find the version and dependent libraries that are installed. To upgrade to the latest version, run [az upgrade](#).

## Create an IoT hub

You can quickly create a new IoT hub using the Azure Cloud Shell terminal in the portal.

1. Sign in to the [Azure portal](#).
2. On the upper right of the page, select the Cloud Shell button.



3. Run the following command to create a new resource group:

```
az group create --name {your resource group name} --location westus
```

4. Run the following command to create an IoT hub:

```
az iot hub create --name {your iot hub name} --resource-group {your resource group name} --sku S1
```

5. Minimize the Cloud Shell terminal. You will return to the shell later in the tutorial.

## Create a logic app

Next, create a logic app and add an HTTP event grid trigger that processes requests from IoT hub.

### Create a logic app resource

1. In the [Azure portal](#), select **Create a resource**, then type "logic app" in the search box and select return. Select **Logic App** from the results.

The screenshot shows the Azure Marketplace interface. On the left, there's a sidebar with categories like Get Started, AI + Machine Learning, Analytics, Blockchain, Compute, Containers, Databases, Developer Tools, and DevOps. The 'Get Started' category is currently selected. In the main area, there's a search bar with 'logic app' typed in, and filters for Pricing: All, Operating System: All, and Publisher: All. Below the search bar, it says 'Showing All Results'. There are three cards displayed: 1. 'Alert Logic Enterprise - BYOL' by Alert Logic, which is not selected. 2. 'Logic App' by Microsoft, which is highlighted with a red box. 3. 'Sumo Logic for Azure Web Apps' by Sumo Logic. At the bottom right of the card list, there are two icons: a blue square with a white plus sign and a blue square with a white heart.

2. On the next screen, select **Create**.
3. Give your logic app a name that's unique in your subscription, then select the same subscription, resource group, and location as your IoT hub.

Home > New > Marketplace > Logic App >

## Logic App

\* Basics Tags Review + create

**Project details**

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

Resource group \*  [Create new](#)

**Instance details**

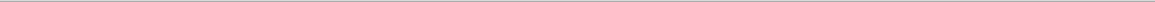
Logic App name \*  ✓

Select the location  Region  Integration Service Environment

Location \*

Log Analytics [ⓘ](#)

[Review + create](#) [< Previous : Basics](#) [Next : Tags >](#) [Download a template for automation ⓘ](#)



4. Select **Review + create**.
5. Verify your settings and then select **Create**.
6. Once the resource is created, select **Go to resource**.
7. In the Logic Apps Designer, page down to see **Templates**. Choose **Blank Logic App** so that you can build your logic app from scratch.

### Select a trigger

A trigger is a specific event that starts your logic app. For this tutorial, the trigger that sets off the workflow is receiving a request over HTTP.

1. In the connectors and triggers search bar, type **HTTP**.
2. Scroll through the results and select **Request - When an HTTP request is received** as the trigger.

The screenshot shows the Microsoft Flow interface with a search bar at the top containing 'http'. Below the search bar, there are tabs: 'For You', 'All', 'Built-in', 'Standard', 'Enterprise', and 'Custom'. Under the 'All' tab, there is a grid of trigger icons. The 'When a HTTP request is received' trigger, which has a globe icon and the text 'Request', is highlighted with a red box. Other triggers include 'Office 365 Outlook', 'SharePoint', 'FTP', 'Aquaforest PDF', 'Azure Blob Storage', 'Azure DevOps', 'Azure Kusto', 'Cloudmersive Data ...', 'Content Moderator', 'Data8 Data Enrichment', 'Encodian', 'HTTP', and 'HTTP with Azure AD'. Below the grid, there are two tabs: 'Triggers' (selected) and 'Actions'.

Trigger	Description
On new high severity alerts (preview)	Microsoft Graph Security
Trigger fires when recommendations are available (preview)	ModuleQ
When a HTTP request is received	Request
Inbound SMS Notification (preview)	TxtSync
Outbound SMS Notification (preview)	TxtSync

3. Select Use sample payload to generate schema.

The screenshot shows the configuration page for the 'When a HTTP request is received' trigger. At the top, it says 'When a HTTP request is received'. Below that, there is a 'HTTP POST URL' field with the placeholder 'URL will be generated after save' and a '... More' button. Underneath is a 'Request Body JSON Schema' section with a large text area. At the bottom of this section is a red-bordered button labeled 'Use sample payload to generate schema'. Below this button is another button labeled 'Add new parameter' with a dropdown arrow.

4. Paste the *Device connected event schema* JSON into the text box, then select **Done**:

This event publishes when a device is connected to an IoT hub.

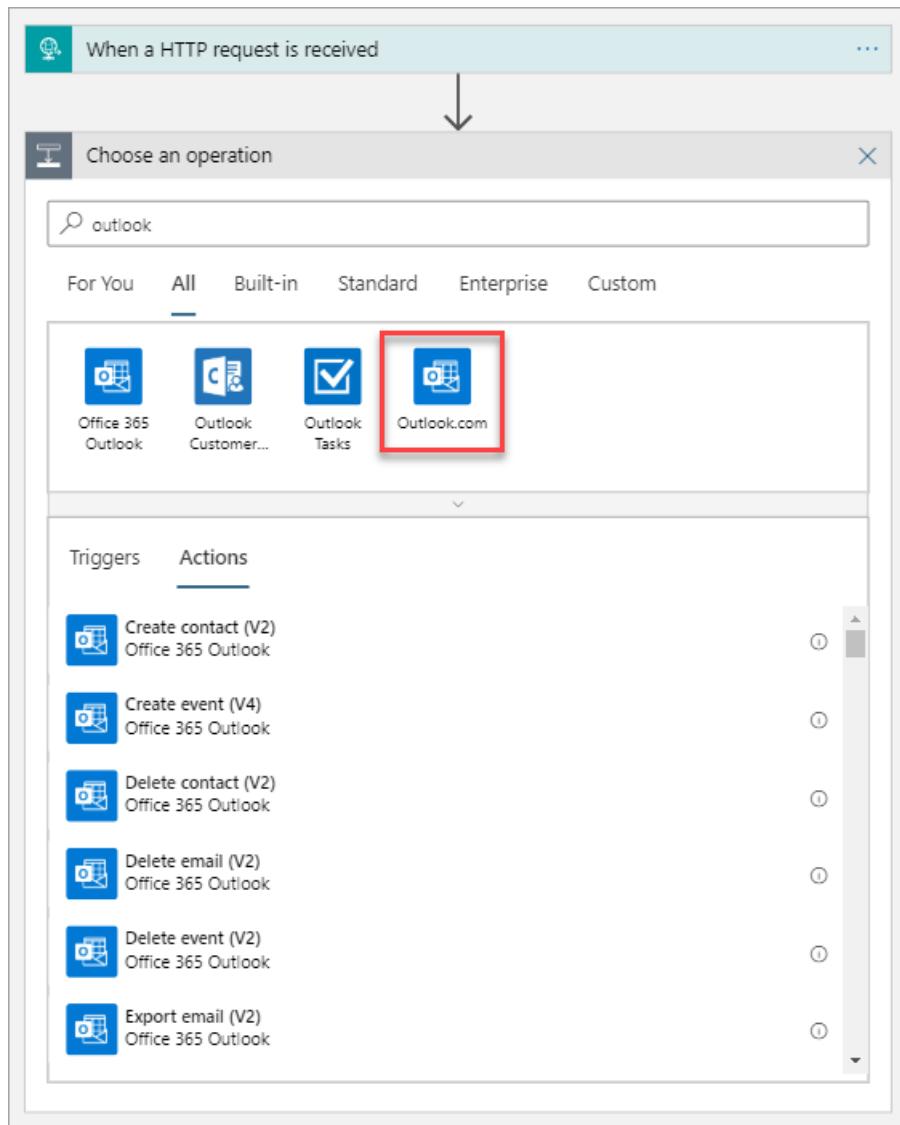
## NOTE

You may receive a pop-up notification that says, **Remember to include a Content-Type header set to application/json in your request**. You can safely ignore this suggestion, and move on to the next section.

## Create an action

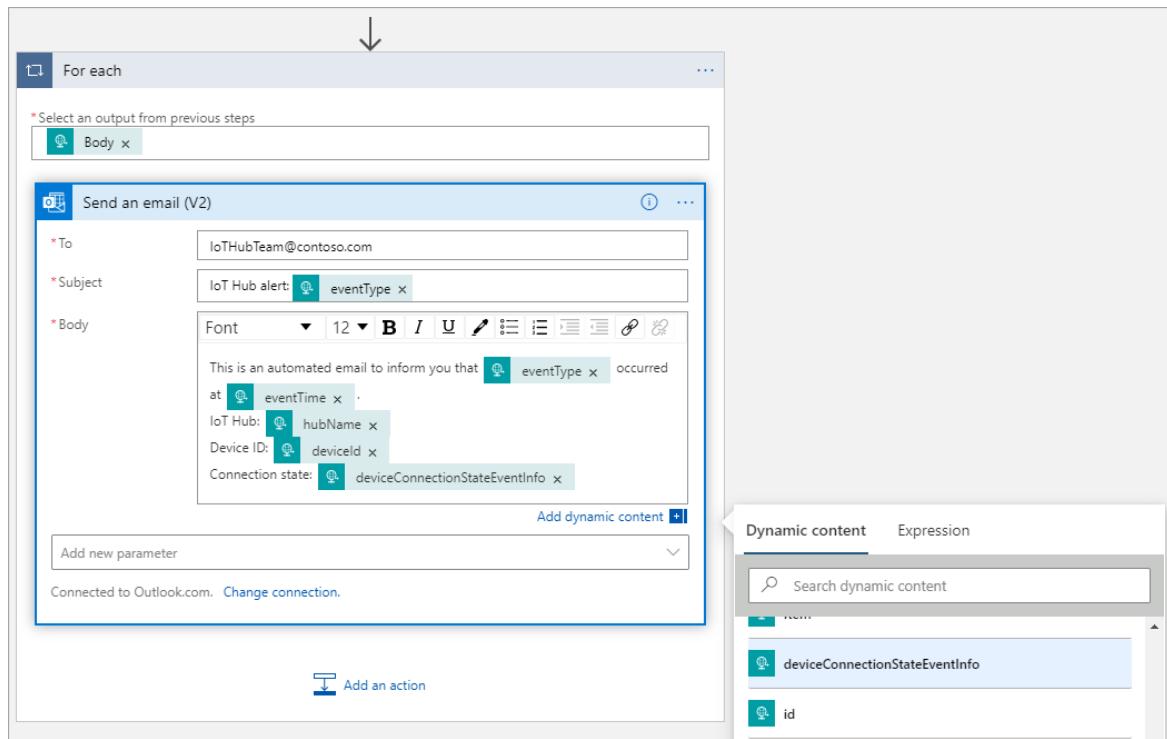
Actions are any steps that occur after the trigger starts the logic app workflow. For this tutorial, the action is to send an email notification from your email provider.

1. Select **New step**. This opens a window to **Choose an action**.
  2. Search for **Outlook**.
  3. Based on your email provider, find and select the matching connector. This tutorial uses **Outlook.com**.  
The steps for other email providers are similar.



4. Select the **Send an email (V2)** action.
5. Select **Sign in** and sign in to your email account. Select **Yes** to let the app access your info.
6. Build your email template.
  - **To:** Enter the email address to receive the notification emails. For this tutorial, use an email account that you can access for testing.
  - **Subject:** Fill in the text for the subject. When you click on the Subject text box, you can select dynamic content to include. For example, this tutorial uses `IoT Hub alert: {eventType}`. If you can't see Dynamic content, select the **Add dynamic content** hyperlink -- this toggles it on and off.
  - **Body:** Write the text for your email. Select JSON properties from the selector tool to include dynamic content based on event data. If you can't see the Dynamic content, select the **Add dynamic content** hyperlink under the **Body** text box. If it doesn't show you the fields you want, click *more* in the Dynamic content screen to include the fields from the previous action.

Your email template may look like this example:

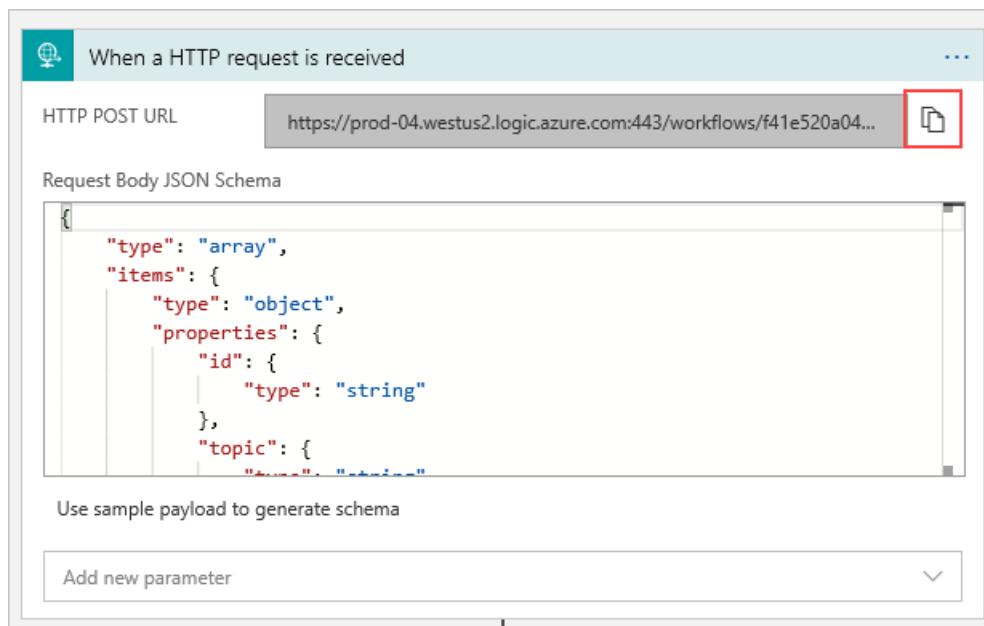


7. Select **Save** in the Logic Apps Designer.

#### Copy the HTTP URL

Before you leave the Logic Apps Designer, copy the URL that your logic app is listening to for a trigger. You use this URL to configure Event Grid.

1. Expand the **When a HTTP request is received** trigger configuration box by clicking on it.
2. Copy the value of **HTTP POST URL** by selecting the copy button next to it.



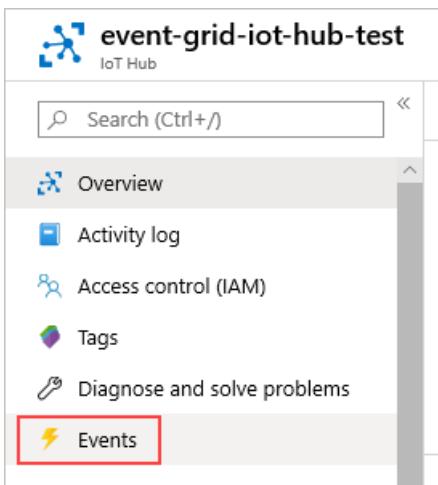
3. Save this URL so that you can refer to it in the next section.

## Configure subscription for IoT Hub events

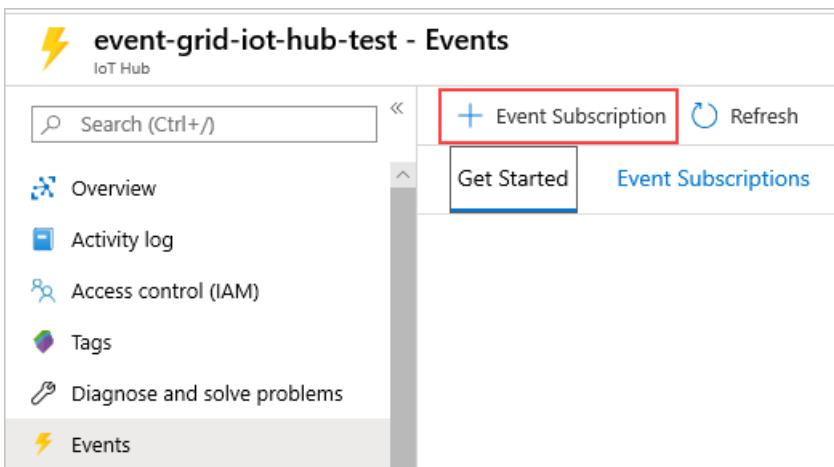
In this section, you configure your IoT Hub to publish events as they occur.

1. In the Azure portal, navigate to your IoT hub. You can do this by selecting **Resource groups**, then select the resource group for this tutorial, and then select your IoT hub from the list of resources.

2. Select Events.



3. Select Event subscription.



4. Create the event subscription with the following values:

a. In the **EVENT SUBSCRIPTION DETAILS** section:

- Provide a **name** for the event subscription.
- Select **Event Grid Schema** for **Event Schema**.

b. In the **TOPIC DETAILS** section:

- Confirm that the **Topic type** is set to **IoT Hub**.
- Confirm that the name of the IoT hub is set as the value for the **Source Resource** field.
- Enter a name for the **system topic** that will be created for you. To learn about system topics, see [Overview of system topics](#).

c. In the **EVENT TYPES** section:

- Select the **Filter to Event Types** drop-down.
- Deselect the **Device Created** and **Device Deleted** checkboxes, leaving only the **Device Connected** and **Device Disconnected** checkboxes selected.

**EVENT TYPES**

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

**ENDPOINT DETAILS**

Pick an event handler to receive your events [Learn more](#)

Endpoint Type

Device Created  
 Device Deleted  
 Device Connected  
 Device Disconnected  
 Device Telemetry

d. In the **ENDPOINT DETAILS** section:

- a. Select **Endpoint Type** as **Web Hook**.
- b. Click **select an endpoint**, paste the URL that you copied from your logic app, and confirm selection.

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type

Endpoint

When you're done, the pane should look like the following example:

**Create Event Subscription**

[Basic](#) [Filters](#) [Additional Features](#) [Advanced Editor](#)

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**EVENT SUBSCRIPTION DETAILS**

Name  ✓  
Event Schema

**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type   
Source Resource philmeaEGHub  
System Topic Name  ✓

**EVENT TYPES**

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type   
Endpoint <https://prod-34.westus.logic.azure.com:443/workflows/bf864...> [\(change\)](#)

5. Select **Create**.

# Simulate a new device connecting and sending telemetry

Test your logic app by quickly simulating a device connection using the Azure CLI.

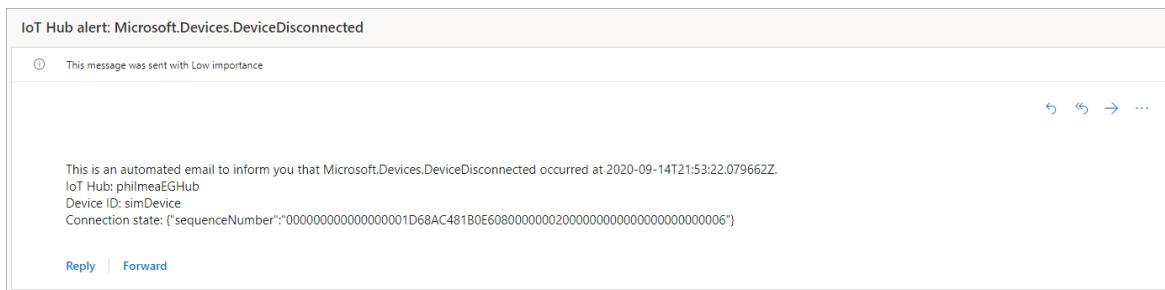
1. Select the Cloud Shell button to re-open your terminal.
2. Run the following command to create a simulated device identity:

```
az iot hub device-identity create --device-id simDevice --hub-name {YourIoTHubName}
```

3. Run the following command to simulate connecting your device to IoT Hub and sending telemetry:

```
az iot device simulate -d simDevice -n {YourIoTHubName}
```

4. When the simulated device connects to IoT Hub, you will receive an email notifying you of a "DeviceConnected" event.
5. When the simulation completes, you will receive an email notifying you of a "DeviceDisconnected" event.



## Clean up resources

This tutorial used resources that incur charges on your Azure subscription. When you're finished trying out the tutorial and testing your results, disable or delete resources that you don't want to keep.

To delete all of the resources created in this tutorial, delete the resource group.

1. Select **Resource groups**, then select the resource group you created for this tutorial.
2. On the Resource group pane, select **Delete resource group**. You are prompted to enter the resource group name, and then you can delete it. All of the resources contained therein are also removed.

## Next steps

- Learn more about [Reacting to IoT Hub events by using Event Grid to trigger actions](#).
- [Learn how to order device connected and disconnected events](#)
- Learn about what else you can do with [Event Grid](#).

For a complete list of supported Logic App connectors, see the

[Connectors overview](#).

# Tutorial: Respond to Azure Service Bus events received via Azure Event Grid by using Azure Logic Apps

3/5/2021 • 4 minutes to read • [Edit Online](#)

In this tutorial, you learn how to respond to Azure Service Bus events that are received via Azure Event Grid by using Azure Logic Apps.

## Prerequisites

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create a Service Bus namespace

Follow instructions in this tutorial: [Quickstart: Use the Azure portal to create a Service Bus topic and subscriptions to the topic](#) to do the following tasks:

- Create a **premium** Service Bus namespace.
- Get the connection string.
- Create a Service Bus topic.
- Create a subscription to the topic. You need only one subscription in this tutorial, so no need to create subscriptions S2 and S3.

## Send messages to the Service Bus topic

In this step, you use a sample application to send messages to the Service Bus topic you created in the previous step.

1. Clone the [GitHub azure-service-bus repository](#).
2. In Visual Studio, go to the `\samples\DotNet\Azure.Messaging.ServiceBus\ServiceBusEventGridIntegration` folder, and then open the `SBEVENTGRIDINTEGRATION.SLN` file.
3. In the Solution Explorer window, expand the **MessageSender** project, and select **Program.cs**.
4. Replace `<SERVICE BUS NAMESPACE - CONNECTION STRING>` with the connection string to your Service Bus namespace and `<TOPIC NAME>` with the name of the topic.

```
const string ServiceBusConnectionString = "<SERVICE BUS NAMESPACE - CONNECTION STRING>";
const string TopicName = "<TOPIC NAME>";
```

5. Build and run the program to send 5 test messages (`const int numberOfMessages = 5;`) to the Service Bus topic.



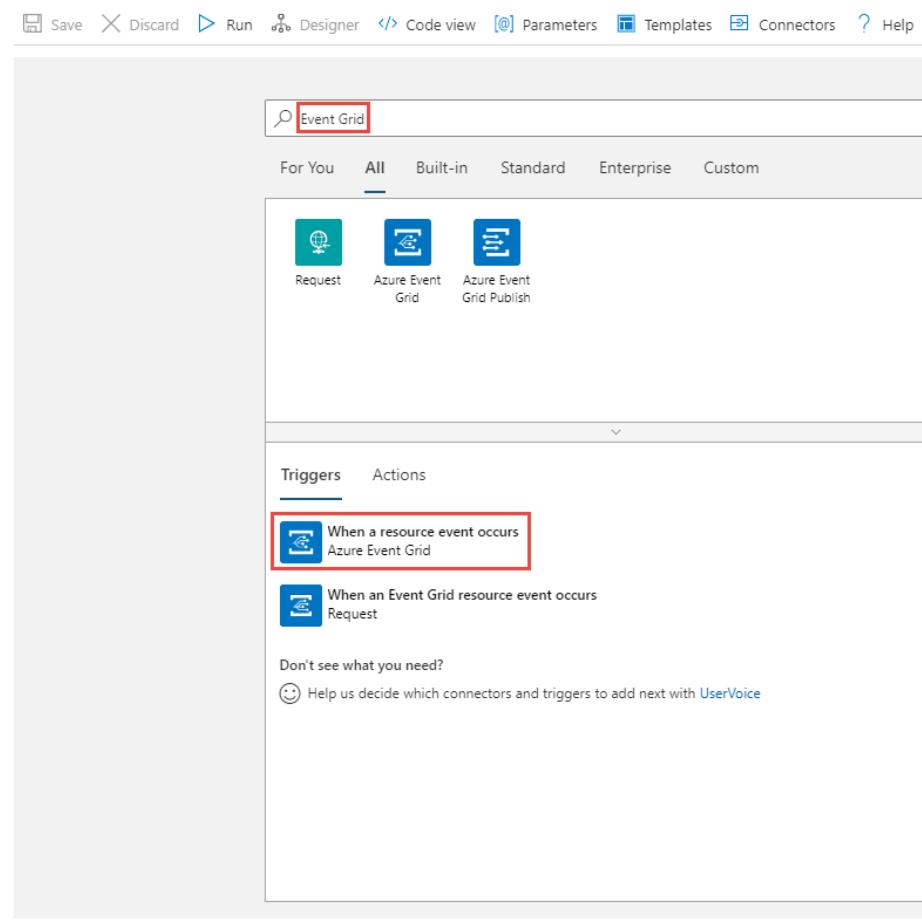
```
C:\WINDOWS\system32\cmd.exe
=====
Press any key to exit after sending the message.
=====
Sending message: Message 1
Sending message: Message 2
Sending message: Message 3
Sending message: Message 4
Sending message: Message 5
```

## Receive messages by using Logic Apps

In this step, you create an Azure logic app that receives Service Bus events via Azure Event Grid.

1. Create a logic app in the Azure portal.
  - a. Select **+ Create a resource**, select **Integration**, and then select **Logic App**.
  - b. On the **Logic App - Create** page, enter a name for the logic app.
  - c. Select your Azure subscription.
  - d. Select **Use existing** for the **Resource group**, and select the resource group that you used for other resources (like Azure function, Service Bus namespace) that you created earlier.
  - e. Select the **Location** for the logic app.
  - f. Select **Review + Create**.
  - g. On the **Review + Create** page, select **Create** to create the logic app.
2. On the **Logic Apps Designer** page, select **Blank Logic App** under **Templates**.
3. On the designer, do the following steps:
  - a. Search for **Event Grid**.
  - b. Select **When a resource event occurs - Azure Event Grid**.

## Logic Apps Designer



4. Select **Sign in**, enter your Azure credentials, and select **Allow Access**.

5. On the **When a resource event occurs** page, do the following steps:

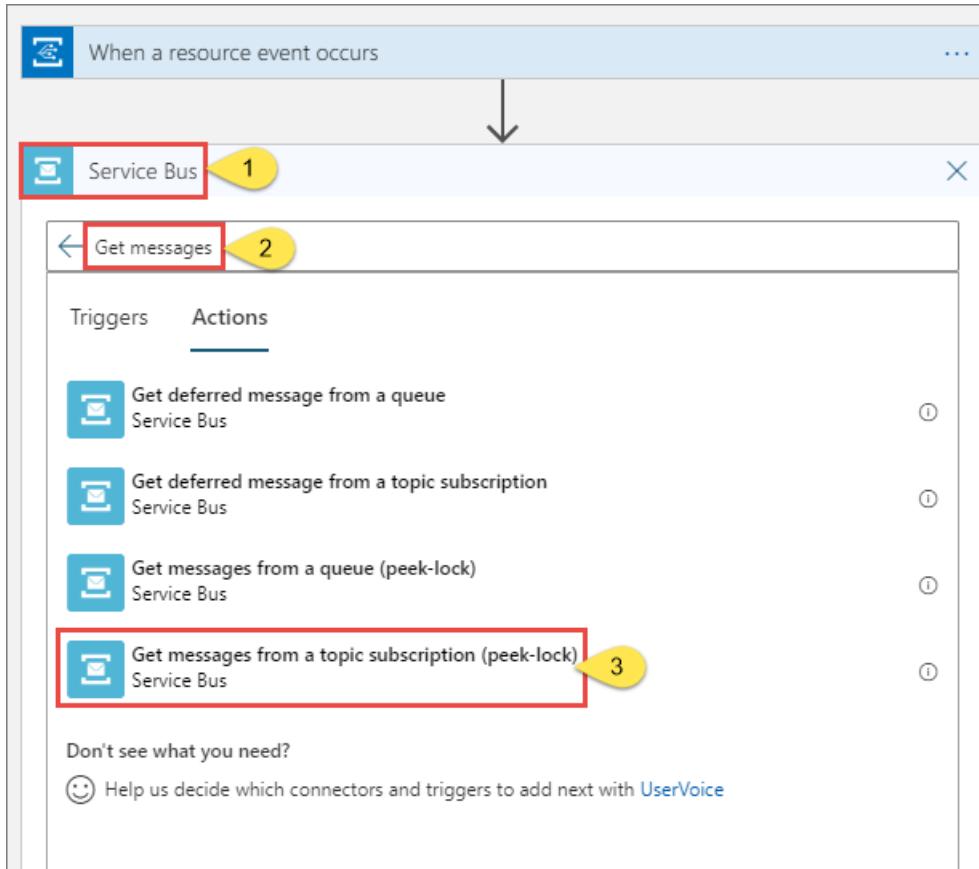
- Select your Azure subscription.
- For **Resource Type**, select **Microsoft.ServiceBus.Namespaces**.
- For **Resource Name**, select your Service Bus namespace.
- Select **Add new parameter**, and select **Suffix Filter**.
- For **Suffix Filter**, enter the name of your second Service Bus topic subscription.

This screenshot shows the configuration dialog for the 'When a resource event occurs' trigger. It has fields for Subscription (Visual Studio Ultimate with MSDN), Resource Type (Microsoft.ServiceBus.Namespaces), and Resource Name (spsbusgridns). Below these is a section for Event Type Item - 1, which is currently empty. There is a '+ Add new item' button. At the bottom, there is a Suffix Filter field containing 'S1' and an 'Add new parameter' dropdown.

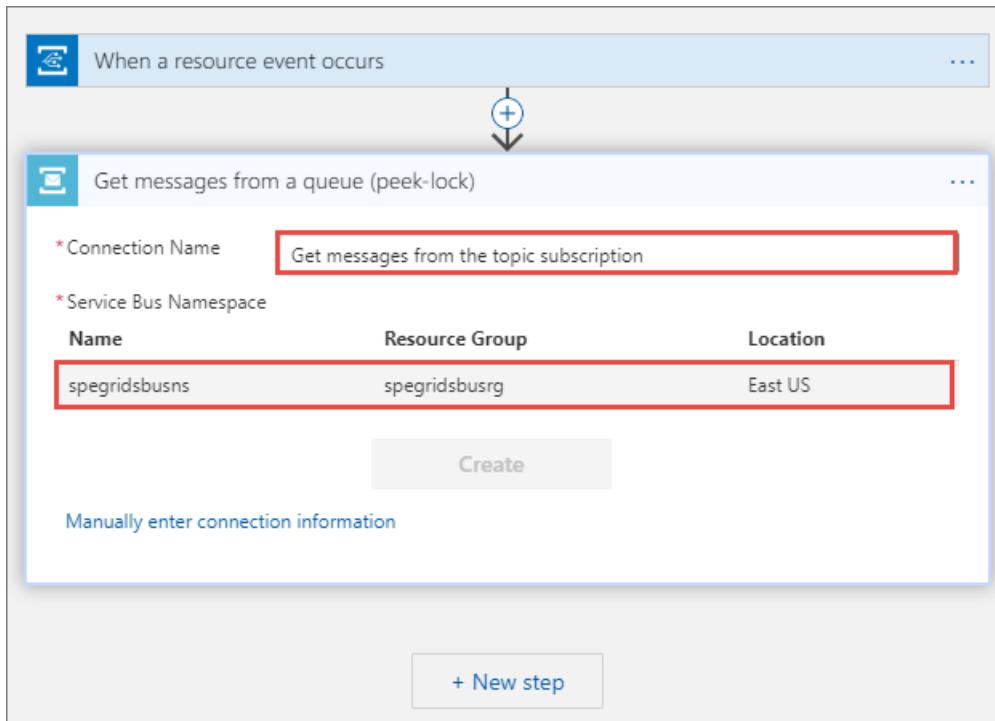
6. Select **+ New Step** in the designer, and do the following steps:

- Search for **Service Bus**.
- Select **Service Bus** in the list.

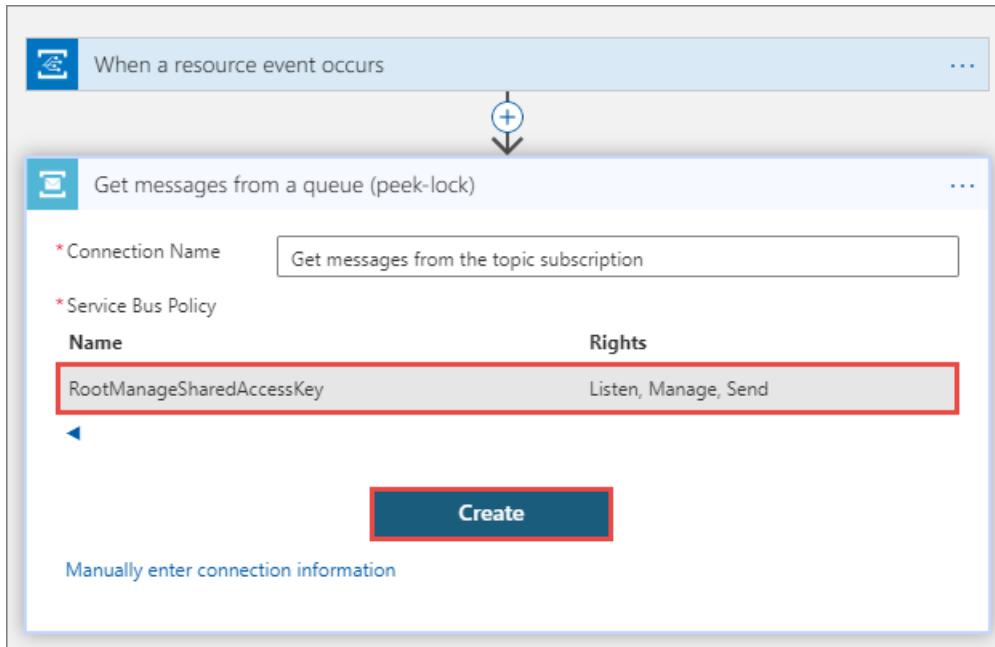
- c. Select for Get messages in the Actions list.
- d. Select Get messages from a topic subscription (peek-lock).



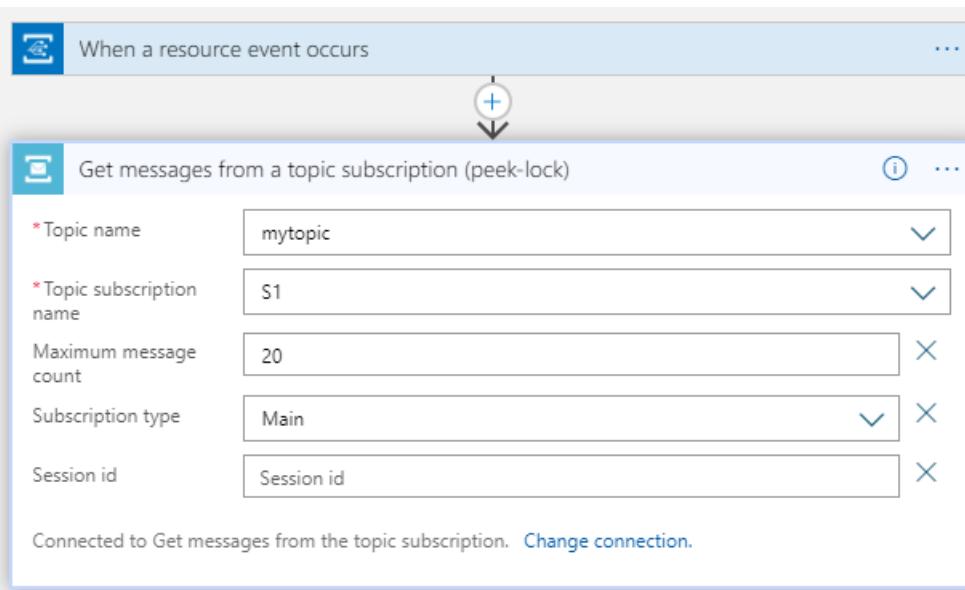
- e. Enter a name for the connection. For example: **Get messages from the topic subscription**, and select the Service Bus namespace.



- f. Select **RootManageSharedAccessKey**, and then select **Create**.

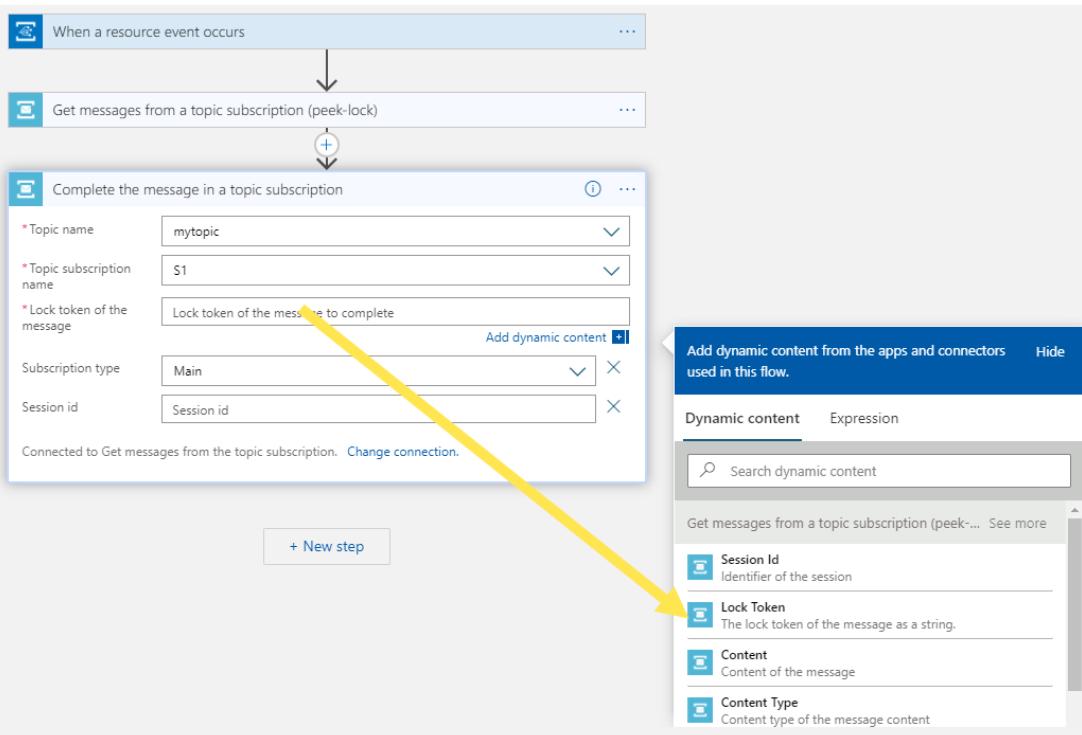


g. Select your **topic** and **subscription**.



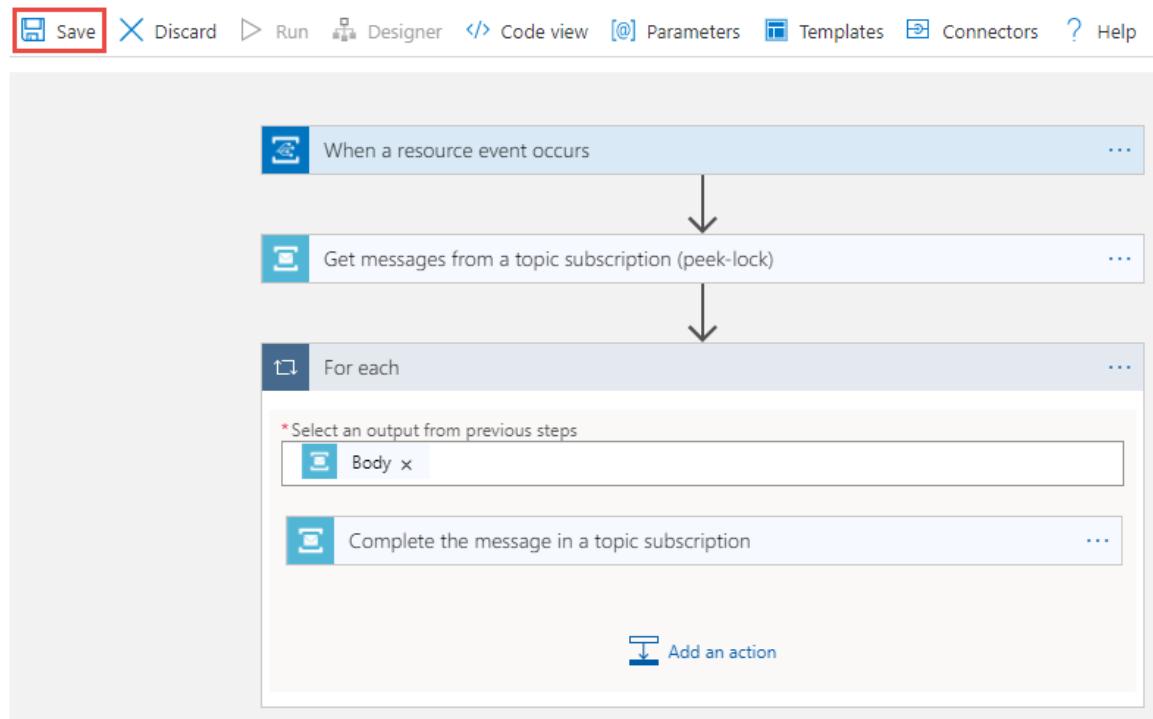
7. Select **+ New step**, and do the following steps:

- a. Select **Service Bus**.
- b. Select **Complete the message in a topic subscription** from the list of actions.
- c. Select your **Service Bus topic**.
- d. Select the **second subscription** to the topic.
- e. For **Lock token of the message**, select **Lock Token** from the **Dynamic content**.



8. Select **Save** on the toolbar on the Logic Apps Designer to save the logic app.

## Logic Apps Designer



9. If you haven't already sent test messages to the topic, follow instructions in the [Send messages to the Service Bus topic](#) section to send messages to the topic.

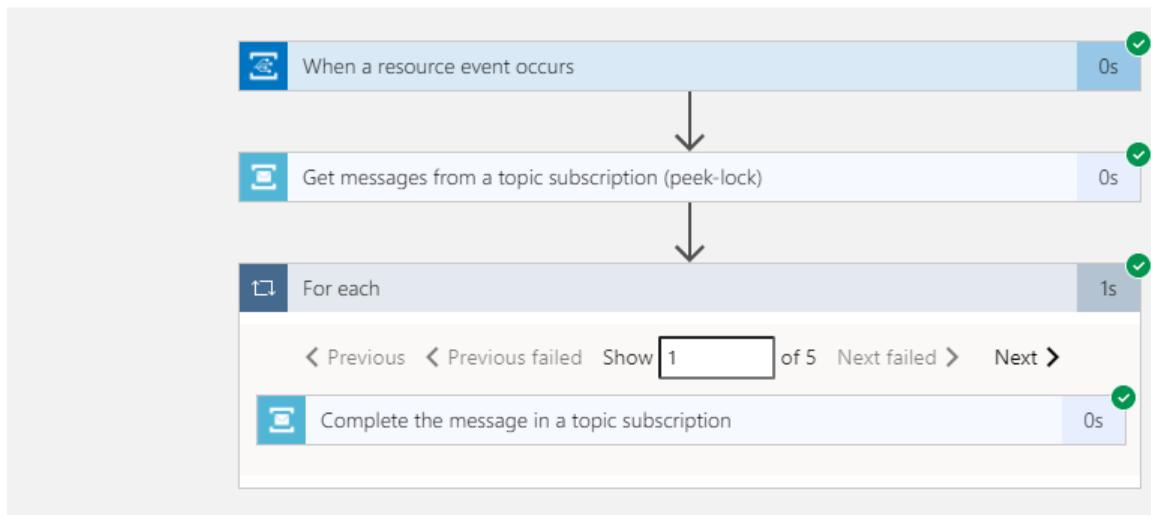
10. Switch to the **Overview** page of your logic app. You see the logic app runs in the **Runs history** for the messages sent. It could take a few minutes before you see the logic app runs. Select **Refresh** on the toolbar to refresh the page.

11. Select a logic app run to see the details. Notice that it processed 5 messages in the for loop.

## Logic app run

08585987260681522168663319286CU82

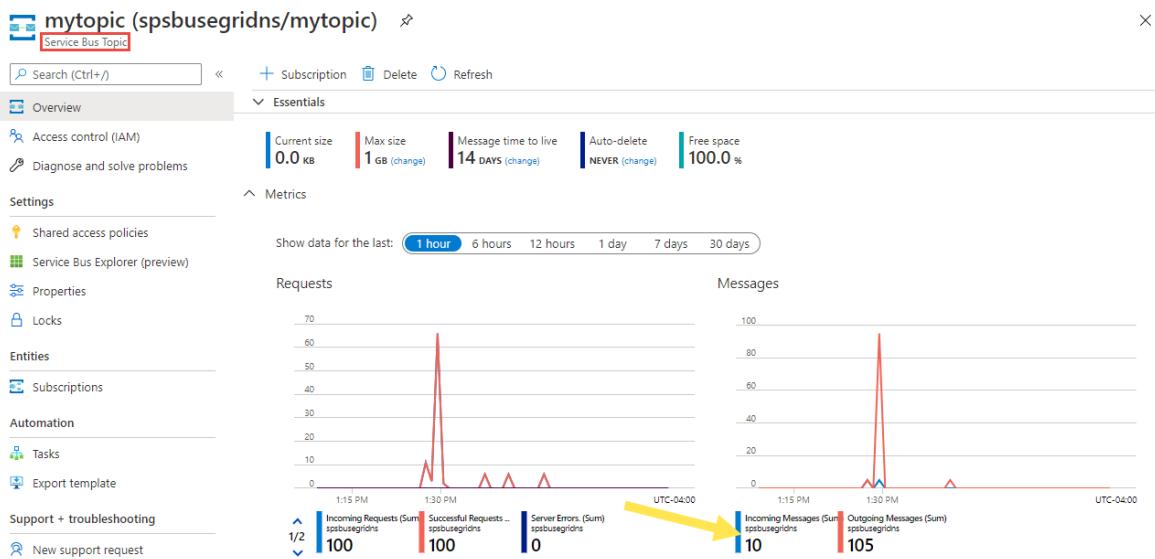
[Run Details](#) [Resubmit](#) [Cancel Run](#) [Info](#)



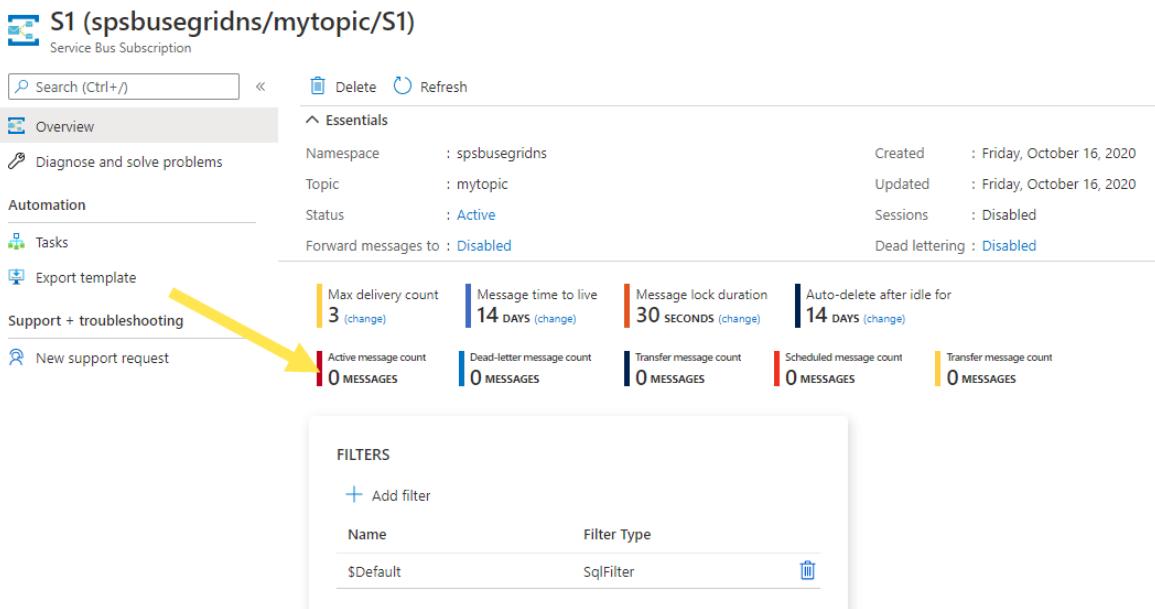
## Troubleshoot

If you don't see any invocations after waiting and refreshing for sometime, follow these steps:

1. Confirm that the messages reached the Service Bus topic. See the **incoming messages** counter on the **Service Bus Topic** page. In this case, I ran the **MessageSender** application twice, so I see 10 messages (5 messages for each run).



2. Confirm that there are no active messages at the Service Bus subscription. If you don't see any events on this page, verify that the **Service Bus Subscription** page doesn't show any Active message count. If the number for this counter is greater than zero, the messages at the subscription aren't forwarded to the handler function (event subscription handler) for some reason. Verify that you've set up the event subscription properly.



3. You also see delivered events on the **Events** page of the Service Bus namespace.

The screenshot shows the 'Events' page for a Service Bus Namespace named 'spsbusegridns'. The left sidebar includes links for Geo-Recovery, Networking, Encryption, Properties, Locks, Queues, Topics, Monitoring, Alerts, Metrics, Diagnostic settings, Logs, Tasks, Events (selected), Export template, Support + troubleshooting, Resource health, and New support request.

The main area displays event metrics for a topic named 'spsbusnssytopic'. The chart shows two spikes of delivered events at 1:30 PM and 1:45 PM. Below the chart, a table lists event types and their counts:

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
spsbusegridsubscription	AzureFunction	S1	--	Microsoft.ServiceBus.ActiveMessagesAvailableWithNoList...

4. You can also see that the events are delivered on the **Event Subscription** page. You can get to this page by selecting the event subscription on the **Events** page.

The screenshot shows the 'Event Subscription' page for 'spsbusegridsubscription'. It includes a 'Save' and 'Delete' button. The top section shows the 'TOPIC' as 'Service Bus Namespace' with name 'spsbusegridns'. The 'Metrics' tab is selected, showing a chart with two spikes of delivered events at 1:30 PM and 1:45 PM. A yellow arrow points to the 'Delivered Events (Sum)' bar, which shows a value of 2. Below the chart, the 'ENDPOINT' is listed as 'Azure Function' with name 'EventGridTriggerFunction' and its icon.

## Next steps

- Learn more about [Azure Event Grid](#).
- Learn more about [Azure Functions](#).
- Learn more about the [Logic Apps feature of Azure App Service](#).
- Learn more about [Azure Service Bus](#).

# Tutorial: Respond to Azure Service Bus events received via Azure Event Grid by using Azure Functions

3/5/2021 • 6 minutes to read • [Edit Online](#)

In this tutorial, you learn how to respond to Azure Service Bus events that are received via Azure Event Grid by using Azure Functions and Azure Logic Apps.

In this tutorial, you learn how to:

- Create a Service Bus namespace
- Prepare a sample application to send messages
- Send messages to the Service Bus topic
- Receive messages by using Logic Apps
- Set up a test function on Azure
- Connect the function and namespace via Event Grid
- Receive messages by using Azure Functions

## Prerequisites

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Create a Service Bus namespace

Follow instructions in this tutorial: [Quickstart: Use the Azure portal to create a Service Bus topic and subscriptions to the topic](#) to do the following tasks:

- Create a **premium** Service Bus namespace.
- Get the connection string.
- Create a Service Bus topic.
- Create a subscription to the topic. You need only one subscription in this tutorial, so no need to create subscriptions S2 and S3.

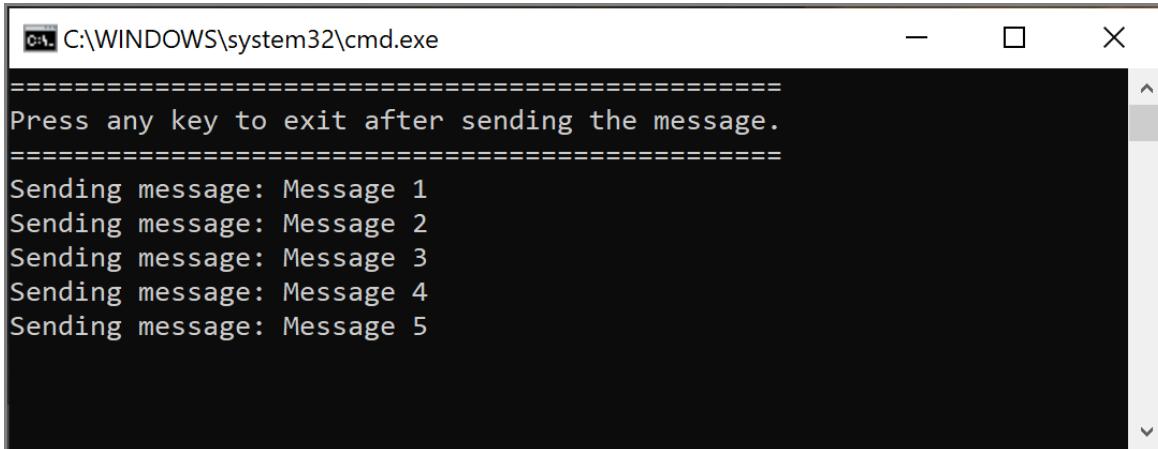
## Send messages to the Service Bus topic

In this step, you use a sample application to send messages to the Service Bus topic you created in the previous step.

1. Clone the [GitHub azure-service-bus repository](#).
2. In Visual Studio, go to the `\samples\DotNet\Azure.Messaging.ServiceBus\ServiceBusEventGridIntegration` folder, and then open the `SBEVENTGRIDINTEGRATION.SLN` file.
3. In the Solution Explorer window, expand the **MessageSender** project, and select `Program.cs`.
4. Replace `<SERVICE BUS NAMESPACE - CONNECTION STRING>` with the connection string to your Service Bus namespace and `<TOPIC NAME>` with the name of the topic.

```
const string ServiceBusConnectionString = "<SERVICE BUS NAMESPACE - CONNECTION STRING>";
const string TopicName = "<TOPIC NAME>";
```

5. Build and run the program to send 5 test messages (`const int numberOfMessages = 5;`) to the Service Bus topic.



```
C:\WINDOWS\system32\cmd.exe
=====
Press any key to exit after sending the message.
=====
Sending message: Message 1
Sending message: Message 2
Sending message: Message 3
Sending message: Message 4
Sending message: Message 5
```

## Additional prerequisites

Install [Visual Studio 2019](#) and include the **Azure development** workload. This workload includes **Azure Function Tools** that you need to create, build, and deploy Azure Functions projects in Visual Studio.

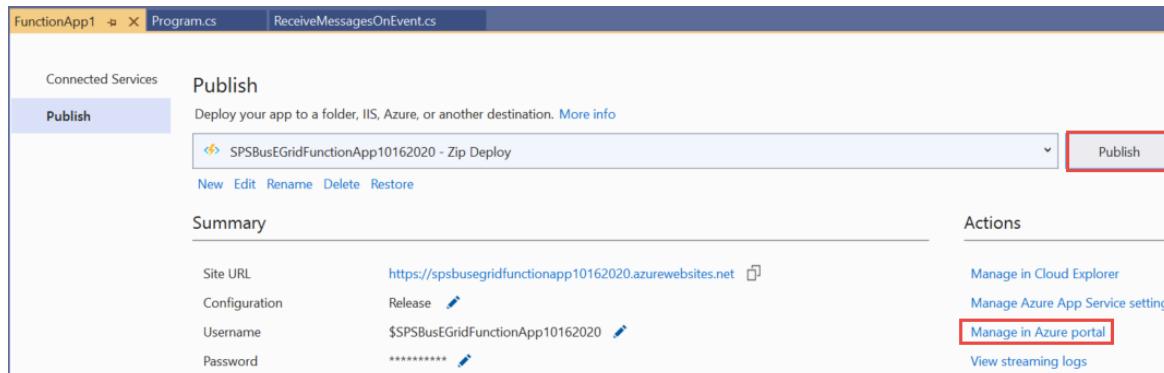
## Deploy the function app

### NOTE

To learn more about creating and deploying an Azure Functions app, see [Develop Azure Functions using Visual Studio](#)

1. Open `ReceiveMessagesOnEvent.cs` file from the `FunctionApp1` project of the `SBEVENTGRIDINTEGRATION.SLN` solution.
2. Replace `<SERVICE BUS NAMESPACE - CONNECTION STRING>` with the connection string to your Service Bus namespace. It should be the same as the one you used in the `Program.cs` file of the `MessageSender` project in the same solution.
3. Right-click `FunctionApp1`, and select **Publish**.
4. On the **Publish** page, select **Start**. These steps may be different from what you see, but the process of publishing should be similar.
5. In the **Publish** wizard, on the **Target** page, select **Azure** for **Target**.
6. On the **Specific target** page, select **Azure Function App (Windows)**.
7. On the **Functions instance** page, select **Create a new Azure function**.
8. On the **Function App (Windows)** page, follow these steps:
  - a. Enter a **name** for the function app.
  - b. Select an **Azure subscription**.
  - c. Select an existing **resource group** or create a new resource group. For this tutorial, select the resource group that has the Service Bus namespace.
  - d. Select a **plan type** for App Service.

- e. Select a **location**. Select the same location as the Service Bus namespace.
- f. Select an existing **Azure Storage** or select **New** to create a new Storage account to be used by the Functions app.
- g. Select **Create** to create the Functions app.
9. Back on the **Functions instance** page of the **Publish** wizard, select **Finish**.
10. On the **Publish** page in Visual Studio, select **Publish** to publish the Functions app to Azure.



11. In the **Output** window, see the messages from build and publish, and confirm that they both succeeded.
12. Now, on the **Publish** page, select **Manage in Azure portal**.
13. In the Azure portal, on the **Function App** page, select **Functions** in the left menu, and confirm that you see two functions:

**SPSBusEGridFunctionApp10162020 | Functions**

Function App

Search (Ctrl+ /) < + Add </> Develop Locally Refresh Delete

Your app is currently in read only mode because you are running from a package file. To make any changes update the content in your zip file and WEBSITE\_RUN\_FROM\_PACKAGE app setting.

Filter by name...

Name ↑	Trigger ↑	Status ↑
EventGridTriggerFunction	EventGrid	Enabled
HTTPTriggerFunction	HTTP	Enabled

**NOTE**

The `EventGridTriggerFunction` uses an [Event Grid trigger](#) and the `HTTPTriggerFunction` uses a [HTTP trigger](#).

14. Select **EventGridTriggerFunction** from the list. We recommend that you use the Event Grid trigger with Azure Functions as it has a few benefits over using the HTTP trigger. For details, see [Azure function as an event handler for Event Grid events](#).
15. On the **Function** page for the **EventGridTriggerFunction**, select **Monitor** on the left menu.
16. Select **Configure** to configure Application Insights to capture invocation log. You use this page to monitor function executions upon receiving Service Bus events from Event Grid.
17. On the **Application Insights** page, enter a name for the resource, select a **location** for the resource,

and then select OK.

18. Select the function **EventGridTriggerFunction** at the top (breadcrumb menu) to navigate back to the **Function** page.
19. Confirm that you are on the **Monitor** page.

The screenshot shows the Azure Function Monitor page for the function 'EventGridTriggerFunction'. The left sidebar includes links for Overview, Developer (Code + Test, Integration, Monitor, Function Keys), and Logs. The main area has tabs for 'Invocations' (selected) and 'Logs'. A message states 'Results may be delayed for up to 5 minutes.' Below are 'Success Count' (0) and 'Error Count' (0) for the last 30 days. The 'Invocation Traces' section shows a table with columns: Date (UTC), Success, Result Code, Duration (ms), and Operation Id. A note says 'No results'.

Keep this page open in a tab your web browser. You'll refresh this page to see invocations for this function later.

## Connect the function and namespace via Event Grid

In this section, you tie together the function and the Service Bus namespace by using the Azure portal.

To create an Azure Event Grid subscription, follow these steps:

1. In the Azure portal, go to your namespace and then, in the left pane, select **Events**. Your namespace window opens, with two Event Grid subscriptions displayed in the right pane.

The screenshot shows the Azure Service Bus Namespace 'spsbusegridns' Events page. The left sidebar includes links for Networking, Encryption, Properties, Locks, Entities (Queues, Topics), Monitoring (Alerts, Metrics, Diagnostic settings, Logs), Automation (Tasks, Events selected), Support + troubleshooting, Resource health, and New support request. The main area features a heading 'Events, automated.' and a description of Event Grid's capabilities. It shows 'Example Scenarios' like resizing an image, copying files, and processing file content through cognitive services. It also lists supported resources: Logic Apps and Azure Functions. Both are described as using events as triggers for executing workflows or custom code.

2. Select **+** **Event Subscription** on the toolbar.
3. On the **Create Event Subscription** page, do the following steps:
  - a. Enter a **name** for the subscription.
  - b. Enter a **name** for the **system topic**. System topics are topics created for Azure resources such as Azure Storage account and Azure Service Bus. To learn more about system topics, see [System topics overview](#).
  - c. Select **Azure Function** for **Endpoint Type**, and click **Select an endpoint**.

**Create Event Subscription**

Event Grid

Basic    Filters    Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**EVENT SUBSCRIPTION DETAILS**

Name *	spsbusgridsubscription	✓
Event Schema	Event Grid Schema	▼

**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type	Service Bus Namespace	
Source Resource	spsbusgridns	
System Topic Name * ⓘ	spsbusnssystopic	✓

**EVENT TYPES**

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types	2 selected	▼
-----------------------	------------	---

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type *	Azure Function (change)
Endpoint	Select an endpoint

**Create**

- d. On the **Select Azure Function** page, select the subscription, resource group, function app, slot, and the function, and then select **Confirm selection**.

## Select Azure Function

X

Event Grid

Subscription

▼

Resource group

▼

Function app \*

▼

Slot \* ⓘ

▼

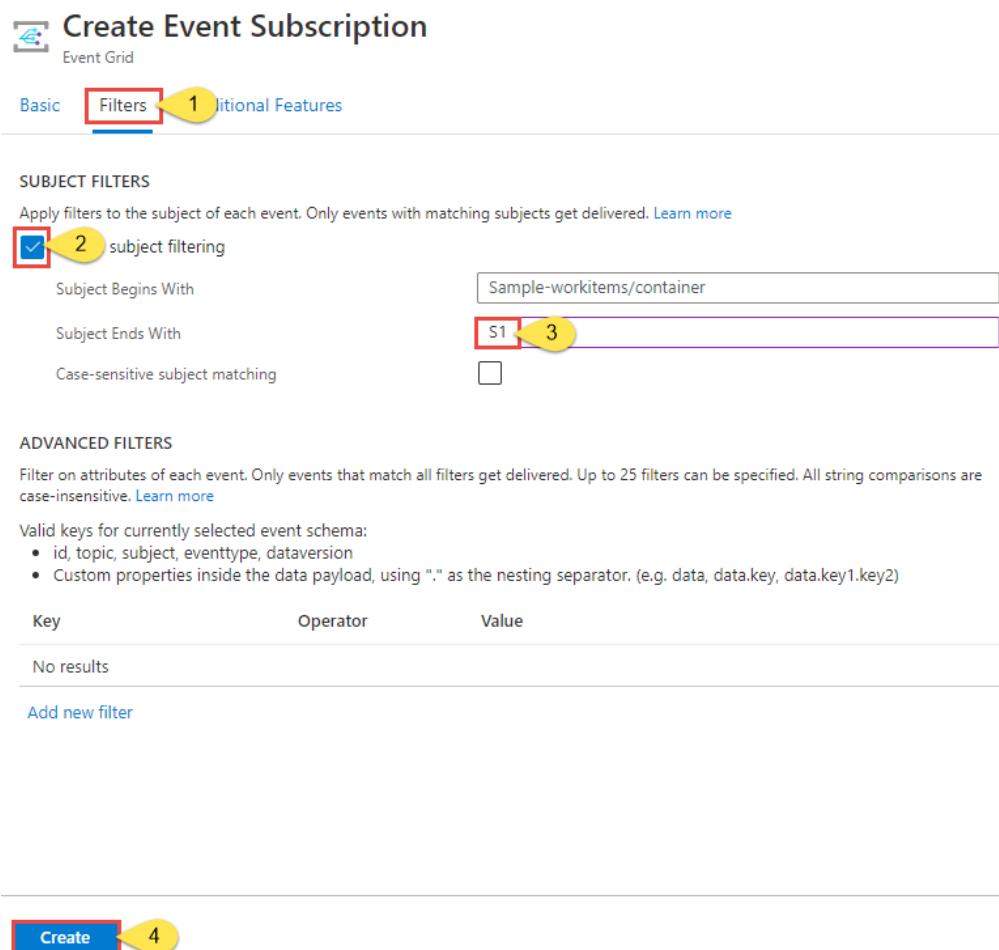
Function \* ⓘ

▼

**Confirm Selection**

e. On the Create Event Subscription page, switch to the Filters tab, and do the following tasks:

- a. Select Enable subject filtering
- b. Enter the name of the subscription to the Service Bus topic (S1) you created earlier.
- c. Select the Create button.

The screenshot shows the 'Create Event Subscription' page for 'Event Grid'. The 'Filters' tab is selected, indicated by a red box and yellow circle labeled '1'. The 'Basic' tab is shown below it. The 'SUBJECT FILTERS' section contains a checked checkbox labeled '2' with a red box and yellow circle, followed by 'subject filtering'. It includes three input fields: 'Subject Begins With' (value: 'Sample-workitems/container'), 'Subject Ends With' (value: 'S1'), and 'Case-sensitive subject matching' (unchecked). The 'ADVANCED FILTERS' section is present but empty. At the bottom, a 'Create' button is highlighted with a red box and yellow circle labeled '4'.

**Create Event Subscription**

Event Grid

Basic **Filters** 1 Additional Features

SUBJECT FILTERS

Apply filters to the subject of each event. Only events with matching subjects get delivered. [Learn more](#)

2 subject filtering

Subject Begins With

Subject Ends With  3

Case-sensitive subject matching

ADVANCED FILTERS

Filter on attributes of each event. Only events that match all filters get delivered. Up to 25 filters can be specified. All string comparisons are case-insensitive. [Learn more](#)

Valid keys for currently selected event schema:

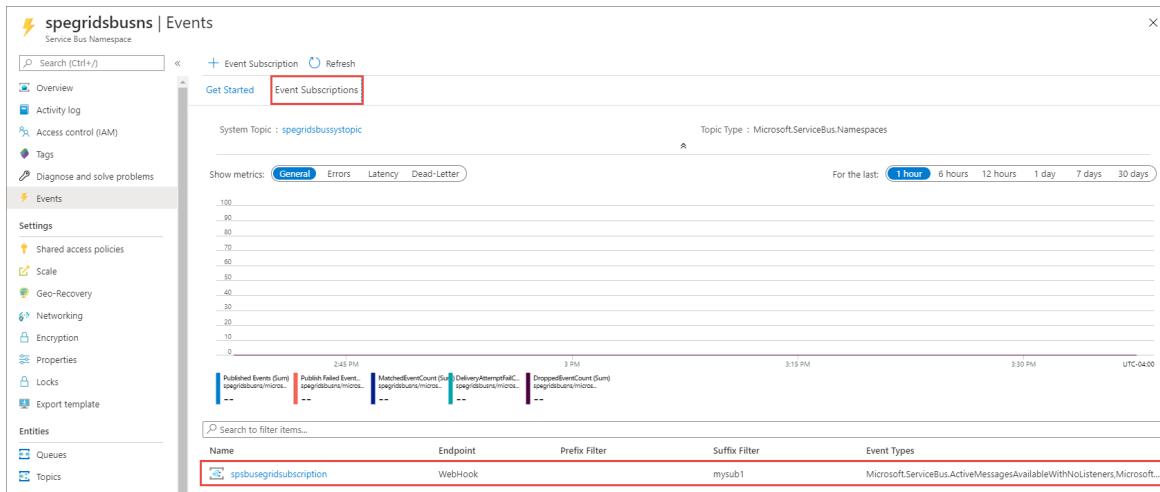
- id, topic, subject, eventtype, dataversion
- Custom properties inside the data payload, using ":" as the nesting separator. (e.g. data, data.key, data.key1.key2)

Key	Operator	Value
No results		

[Add new filter](#)

**Create** 4

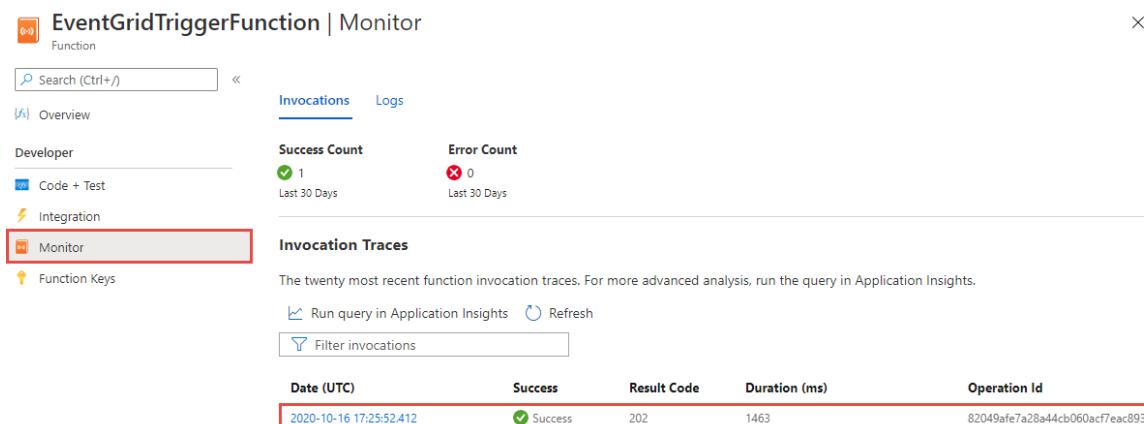
- Switch to the **Event Subscriptions** tab of the **Events** page and confirm that you see the event subscription in the list.



## Monitor the Functions app

The messages you sent to the Service Bus topic earlier are forwarded to the subscription (S1). Event Grid forwards the messages at the subscription to the Azure function. In this step of the tutorial, you confirm the function was invoked and view the logged informational messages.

- On the page for your Azure function app, switch to the **Monitor** tab if it isn't already active. You should see an entry for each message posted to the Service Bus topic. If you don't see them, refresh the page after waiting for a few minutes.



- Select the invocation from the list to see the details.

Invocation Details			X
<a href="#">Run query in Application Insights</a>			
Timestamp	Message	Type	
2020-10-16 17:25:52.618	Executing 'EventGridTriggerFunction' (Reason='EventGrid trigger fired at 2020-10-16T17:25:52.5347045+00:00', Id=44a2b0dc-9e1a-4153-899e-11c35b419a2d)	Information	
2020-10-16 17:25:52.628	C# Event Grid trigger function processed a request.	Information	
2020-10-16 17:25:52.628	{"namespaceName": "spsubgriddns", "requestUri": "https://spsubgriddns.servicebus.windows.net/mytopic/subscriptions/s1/messages/head", "entityType": "subscriber", "queueName": null, "topicName": "mytopic", "subscriptionName": "s1"}	Information	
2020-10-16 17:25:52.668	Topic: mytopic Subscription: s1	Information	
2020-10-16 17:25:53.448	Received: Message 1 from subscription: s1	Information	
2020-10-16 17:25:53.609	Received: Message 2 from subscription: s1	Information	
2020-10-16 17:25:53.661	Received: Message 3 from subscription: s1	Information	
2020-10-16 17:25:53.722	Received: Message 4 from subscription: s1	Information	
2020-10-16 17:25:53.771	Received: Message 5 from subscription: s1	Information	
2020-10-16 17:25:53.832	Executed 'EventGridTriggerFunction' (Succeeded, Id=44a2b0dc-9e1a-4153-899e-11c35b419a2d, Duration=1299ms)	Information	

You can also use the **Logs** tab of the **Monitor** page to see the logging information as the messages are sent. There could be some delay, so give it a few minutes to see the logged messages.

```

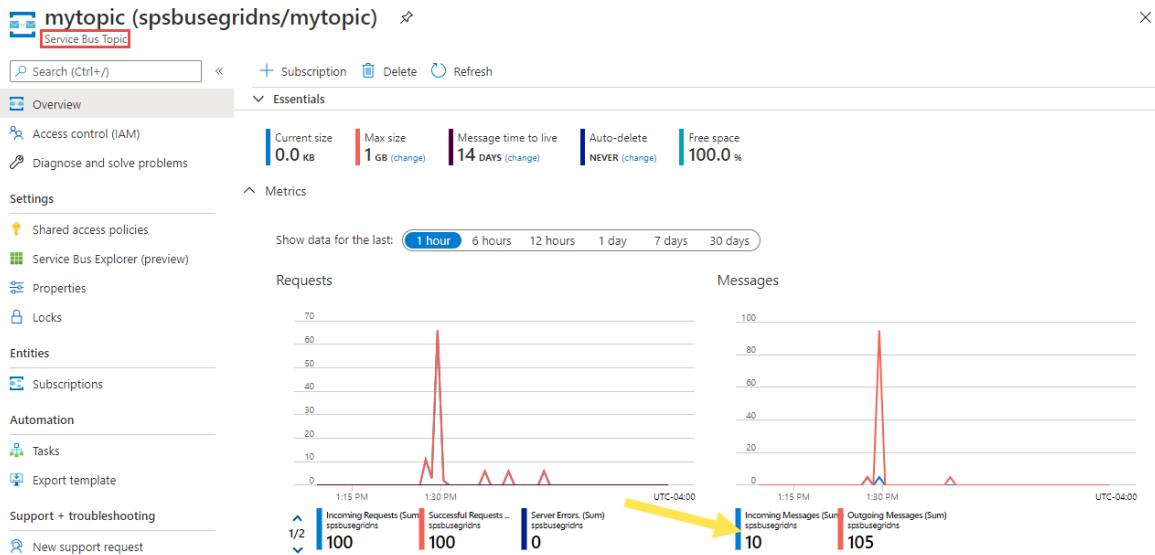
Connected!
2020-10-16T18:22:15 [Information] Welcome, you are now connected to log-streaming service. The default timeout is 2 hours. Change the timeout with the App Setting SCM_LOGSTREAM_TIMEOUT (in seconds).
2020-10-16T18:22:48.020 [Information] Executing "EventGridTriggerFunction" (Reason:"EventGrid trigger fired at 2020-10-16T18:22:49.000+00:00", Id=cfc00c03-25b2-4b18-a222-4e874d666133)
2020-10-16T18:22:49.030 [Information] # Event Grid trigger function processed a request.
2020-10-16T18:22:49.030 [Information] {"namespaceName": "spsbusegridns", "requestId": "https://spsbusegridns.servicebus.windows.net/mytopic/subscriptions/s1/messages/head", "entityType": "subscriber", "queueName": null, "topicName": "mytopic", "subscriptionName": "s1"}
2020-10-16T18:22:49.030 [Information] Topic mytopic Subscription s1
2020-10-16T18:22:49.720 [Information] Received: Message 1 from subscription: s1
2020-10-16T18:22:49.840 [Information] Received: Message 2 from subscription: s1
2020-10-16T18:22:49.907 [Information] Received: Message 3 from subscription: s1
2020-10-16T18:22:50.012 [Information] Received: Message 4 from subscription: s1
2020-10-16T18:22:50.058 [Information] Received: Message 5 from subscription: s1
2020-10-16T18:22:50.235 [Information] Executed "EventGridTriggerFunction" (Succeeded, Id=cfc00c03-25b2-4b18-a222-4e874d666133, Duration=123ms)

```

## Troubleshoot

If you don't see any function invocations after waiting and refreshing for sometime, follow these steps:

1. Confirm that the messages reached the Service Bus topic. See the **incoming messages** counter on the **Service Bus Topic** page. In this case, I ran the **MessageSender** application twice, so I see 10 messages (5 messages for each run).



2. Confirm that there are **no active messages** at the Service Bus subscription. If you don't see any events on this page, verify that the **Service Bus Subscription** page doesn't show any **Active message count**. If the number for this counter is greater than zero, the messages at the subscription aren't forwarded to the handler function (event subscription handler) for some reason. Verify that you've set up the event subscription properly.

S1 (spsbusegridns/mytopic/S1)

Service Bus Subscription

**Overview**

Diagnose and solve problems

Automation

Tasks

Export template

Support + troubleshooting

New support request

**Essentials**

Namespace	: spsbusegridns	Created	: Friday, October 16, 2020
Topic	: mytopic	Updated	: Friday, October 16, 2020
Status	: Active	Sessions	: Disabled
Forward messages to	: Disabled	Dead lettering	: Disabled

Max delivery count: 3 (change)   Message time to live: 14 DAYS (change)   Message lock duration: 30 SECONDS (change)   Auto-delete after idle for: 14 DAYS (change)

Active message count: 0 MESSAGES   Dead-letter message count: 0 MESSAGES   Transfer message count: 0 MESSAGES   Scheduled message count: 0 MESSAGES   Transfer message count: 0 MESSAGES

**FILTERS**

+ Add filter

Name	Filter Type
\$Default	SqlFilter

3. You also see **delivered events** on the **Events** page of the Service Bus namespace.

spsbusegridns | Events

Service Bus Namespace

**Events**

Geo-Recovery

Networking

Encryption

Properties

Locks

Entities

Queues

Topics

Monitoring

Alerts

Metrics

Diagnostic settings

Logs

Automation

Tasks

**Event Subscriptions**

Get Started

System Topic : spsbusnssystopic   Topic Type : Microsoft.ServiceBus.Namespaces

Show metrics: General Errors Latency Dead-letter For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

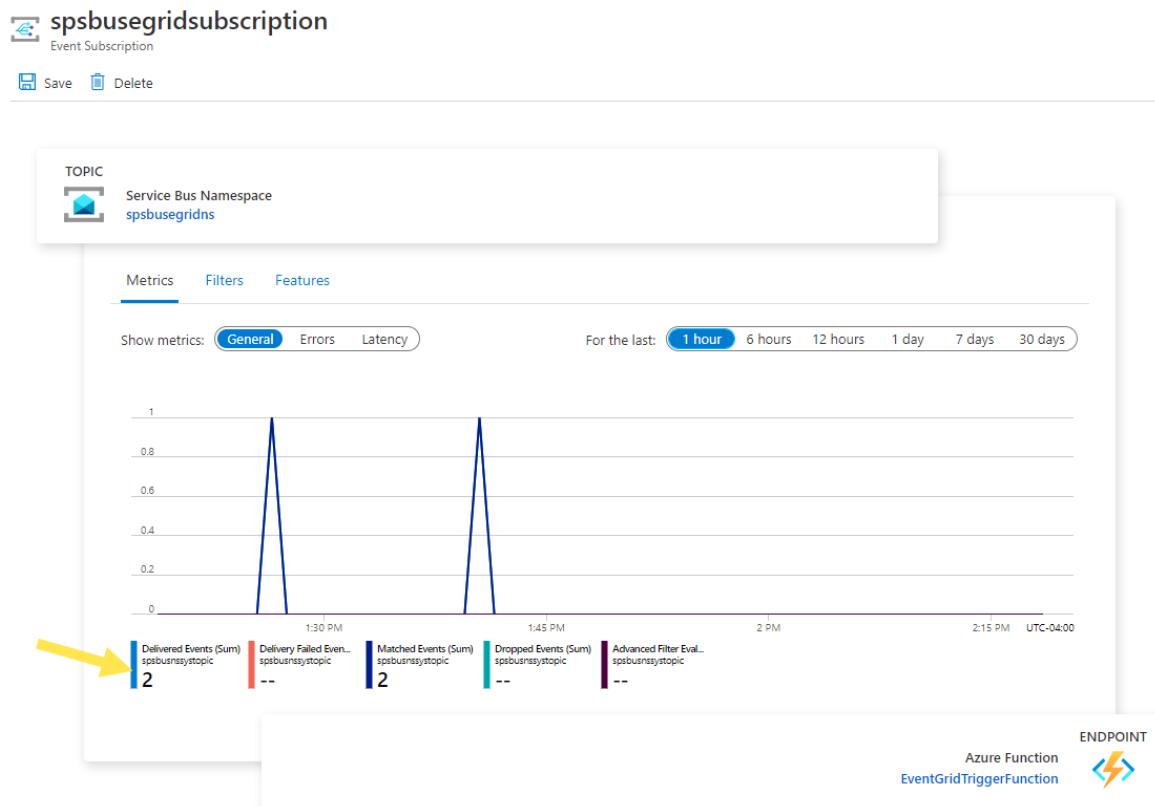
UTC-04:00

Published Events (Sum) spsbusnssystopic 2  
Published Failed Events (Sum) spsbusnssystopic 2  
Max Delivery Count (Sum) spsbusnssystopic 130 PM  
Delivered Events (Sum) spsbusnssystopic 2  
1:45 PM  
Dead Lettered Events (Sum) spsbusnssystopic --  
Delivery Failed Events (Sum) spsbusnssystopic --  
Dropped Events (Sum) spsbusnssystopic --  
Advanced Filter Evaluation UTC-04:00

**Event Subscriptions**

Name	Endpoint	Prefilter	Suffix Filter	Event Types
spsbusegridsubscription	AzureFunction	S1		Microsoft.ServiceBus.ActiveMessagesAvailableWithNoList...

4. You can also see that the events are delivered on the **Event Subscription** page. You can get to this page by selecting the event subscription on the **Events** page.



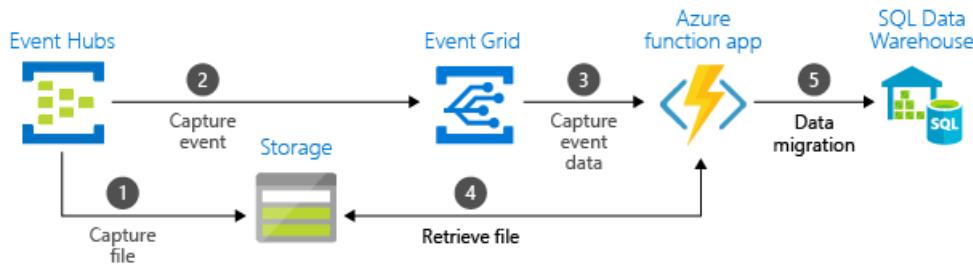
## Next steps

- Learn more about [Azure Event Grid](#).
- Learn more about [Azure Functions](#).
- Learn more about the [Logic Apps feature of Azure App Service](#).
- Learn more about [Azure Service Bus](#).

# Tutorial: Stream big data into a data warehouse

4/27/2021 • 10 minutes to read • [Edit Online](#)

Azure Event Grid is an intelligent event routing service that enables you to react to notifications or events from apps and services. For example, it can trigger an Azure Function to process Event Hubs data that's captured to a Blob storage or Data Lake Storage. This [sample](#) shows you how to use Event Grid and Azure Functions to migrate captured Event Hubs data from blob storage to Azure Synapse Analytics, specifically a dedicated SQL pool.



This diagram depicts the workflow of the solution you build in this tutorial:

1. Data sent to an Azure event hub is captured in an Azure blob storage.
2. When the data capture is complete, an event is generated and sent to an Azure event grid.
3. The event grid forwards this event data to an Azure function app.
4. The function app uses the blob URL in the event data to retrieve the blob from the storage.
5. The function app migrates the blob data to an Azure Synapse Analytics.

In this article, you take the following steps:

- Deploy the required infrastructure for the tutorial
- Publish code to a Functions App
- Create an Event Grid subscription
- Stream sample data into Event Hubs
- Verify captured data in Azure Synapse Analytics

## Prerequisites

To complete this tutorial, you must have:

- An Azure subscription. If you don't have an Azure subscription, create a [free account](#) before you begin.
- [Visual studio 2019](#) with workloads for: .NET desktop development, Azure development, ASP.NET and web development, Node.js development, and Python development.
- Download the [EventHubsCaptureEventGridDemo sample project](#) to your computer.
  - WindTurbineDataGenerator – A simple publisher that sends sample wind turbine data to a capture-enabled event hub
  - FunctionDWDumper – An Azure Function that receives an Event Grid notification when an Avro file is captured to the Azure Storage blob. It receives the blob's URI path, reads its contents, and pushes this data to Azure Synapse Analytics (dedicated SQL pool).

## Deploy the infrastructure

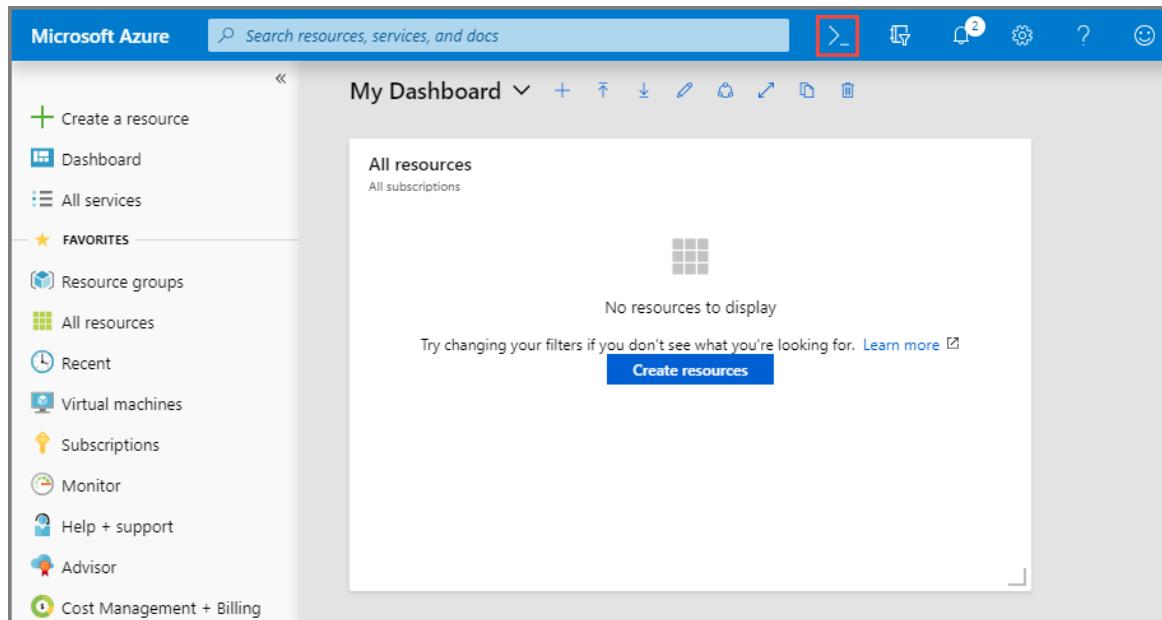
In this step, you deploy the required infrastructure with a [Resource Manager template](#). When you deploy the

template, the following resources are created:

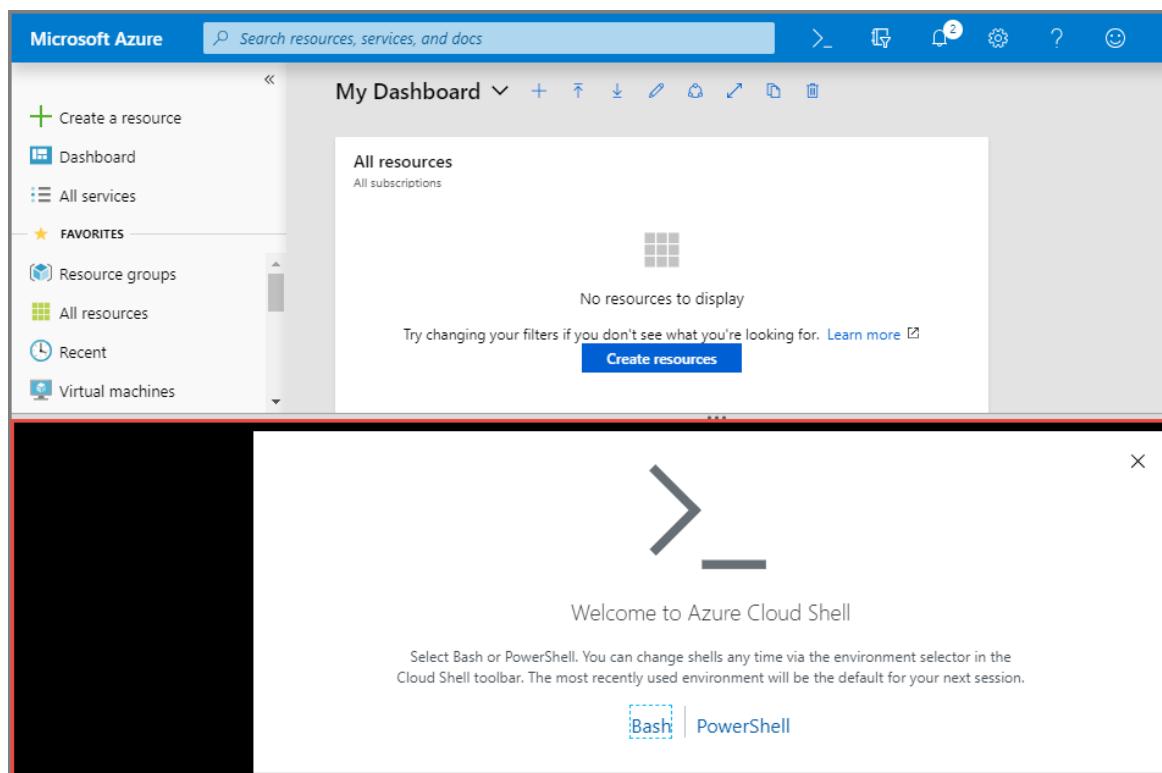
- Event hub with the Capture feature enabled.
- Storage account for the captured files.
- App service plan for hosting the function app
- Function app for processing the event
- SQL Server for hosting the data warehouse
- Azure Synapse Analytics (dedicated SQL pool) for storing the migrated data

## Use Azure CLI to deploy the infrastructure

1. Sign in to the [Azure portal](#).
2. Select **Cloud Shell** button at the top.

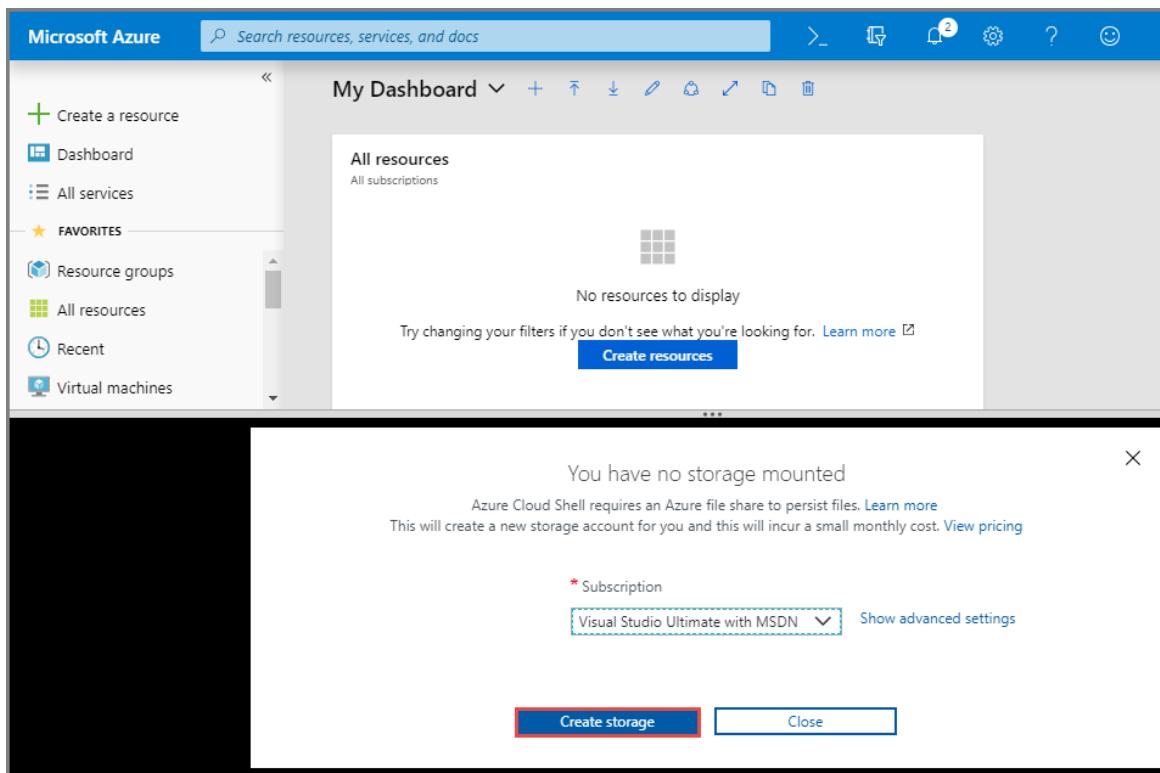


3. You see the Cloud Shell opened at the bottom of the browser.

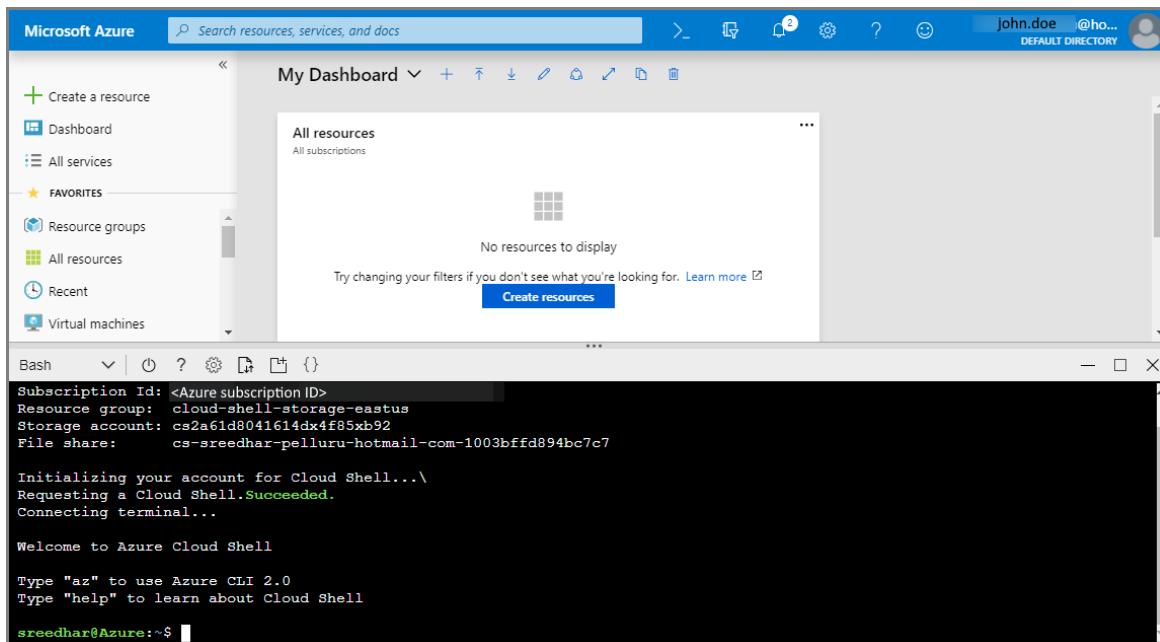


4. In the Cloud Shell, if you see an option to select between **Bash** and **PowerShell**, select **Bash**.

5. If you are using the Cloud Shell for the first time, create a storage account by selecting **Create storage**.  
Azure Cloud Shell requires an Azure storage account to store some files.



6. Wait until the Cloud Shell is initialized.



7. Create an Azure resource group by running the following CLI command:

- Copy and paste the following command into the Cloud Shell window. Change the resource group name and location if you want.

```
az group create -l eastus -n rgDataMigration
```

- Press **ENTER**.

Here is an example:

```
user@Azure:~$ az group create -l eastus -n rgDataMigration
{
  "id": "/subscriptions/00000000-0000-0000-0000-000000000000/resourceGroups/rgDataMigration",
  "location": "eastus",
  "managedBy": null,
  "name": "rgDataMigration",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null
}
```

8. Deploy all the resources mentioned in the previous section (event hub, storage account, functions app, Azure Synapse Analytics) by running the following CLI command:

- Copy and paste the command into the Cloud Shell window. Alternatively, you may want to copy/paste into an editor of your choice, set values, and then copy the command to the Cloud Shell.

#### IMPORTANT

Specify values for the following entities before running the command:

- Name of the resource group you created earlier.
- Name for the event hub namespace.
- Name for the event hub. You can leave the value as it is (hubdatamigration).
- Name for the SQL server.
- Name of the SQL user and password.
- Name for the database.
- Name of the storage account.
- Name for the function app.

```
az deployment group create \
  --resource-group rgDataMigration \
  --template-uri https://raw.githubusercontent.com/Azure/azure-docs-json-
samples/master/event-grid/EventHubsDataMigration.json \
  --parameters eventHubNamespaceName=<event-hub-namespace> eventHubName=hubdatamigration
sqlServerName=<sql-server-name> sqlServerUserName=<user-name> sqlServerPassword=<password>
sqlServerDatabaseName=<database-name> storageName=<unique-storage-name> functionAppName=<app-
name>
```

- Press **ENTER** in the Cloud Shell window to run the command. This process may take a while since you are creating a bunch of resources. In the result of the command, ensure that there have been no failures.

9. Close the Cloud Shell by selecting the **Cloud Shell** button in the portal (or) X button in the top-right corner of the Cloud Shell window.

#### Verify that the resources are created

- In the Azure portal, select **Resource groups** on the left menu.
- Filter the list of resource groups by entering the name of your resource group in the search box.
- Select your resource group in the list.

All services >

## Resource groups

Default Directory

+ Add Manage view Refresh Export to CSV Open query Assign tags Feedback

Filter by name... Subscription == all Location == all Add filter

Showing 1 to 3 of 3 records.

Name	Subscription	Location
cloud-shell-storage-eastus	Visual Studio Ultimate with MSDN	East US
NetworkWatcherRG	Visual Studio Ultimate with MSDN	East US
rgDataMigration	Visual Studio Ultimate with MSDN	East US

No grouping List view

< Page 1 > of 1

4. Confirm that you see the following resources in the resource group:

rgDataMigration

Resource group

Search (Ctrl+J)

+ Add Edit columns Delete resource group Refresh Export to CSV Open query ...

Overview

Activity log Access control (IAM) Tags Events

Settings Deployments Policies Properties Locks

Cost Management Cost analysis Cost alerts (preview)

Essentials

Filter by name... Type == all Location == all Add filter

Showing 1 to 6 of 6 records. Show hidden types No grouping List view

Name	Type	Location
spehubfuncapp1207	Function App	East US
spehubfuncapp1207Plan	App Service plan	East US
spehubns1207	Event Hubs Namespace	East US
spehubsqldb1207 (spehubsqlsvr1207/spehubsql1207)	Dedicated SQL pool (formerly SQL DW)	East US
spehubsqlsvr1207	SQL server	East US
spehubstorage1207	Storage account	East US

### Create a table in Azure Synapse Analytics

Create a table in your data warehouse by running the [CreateDataWarehouseTable.sql](#) script. To run the script, you can use Visual Studio or the Query Editor in the portal. The following steps show you how to use the Query Editor:

1. In the list of resources in the resource group, select your **Dedicated SQL pool**.
2. On the **Dedicated SQL pool** page, in the **Common Tasks** section on the left menu, select **Query editor (preview)**.

The screenshot shows the Azure portal interface for a Dedicated SQL pool (formerly SQL DW). The left sidebar lists various management options like Dynamic Data Masking, Firewalls and virtual networks, Security Center, and Transparent data encryption. Below that is a 'Common Tasks' section with View streaming jobs, Load Data, and Query editor (preview), which is highlighted with a red box. Other tasks include Build dashboards + reports, Model and cache data, and Open in Visual Studio. The main content area has a 'Essentials' section with Notifications (1), Features (4), and Tasks (6). A prominent message box says 'High throughput streaming available!' with the subtext 'Ingest streaming data into a table from multiple input sources for near real-time analytics.'

3. Enter the name of **user** and **password** for the SQL server, and select **OK**. If you see a message about allowing your client to access the SQL server, follow these steps:
  - a. Select the link: **Set server firewall**.
  - b. On the **Firewall settings** page, select **Add client IP** on the toolbar, and then select **Save** on the toolbar.
  - c. Select **OK** on the success message.
  - d. Navigate back to the **Dedicated SQL pool** page, and select **Query editor (preview)** on the left menu.
  - e. Enter **user** and **password**, and then select **OK**.
4. In the query window, copy and run the following SQL script:

```
CREATE TABLE [dbo].[Fact_WindTurbineMetrics] (
    [DeviceId] nvarchar(50) COLLATE SQL_Latin1_General_CI_AS NULL,
    [MeasureTime] datetime NULL,
    [GeneratedPower] float NULL,
    [WindSpeed] float NULL,
    [TurbineSpeed] float NULL
)
WITH (CLUSTERED COLUMNSTORE INDEX, DISTRIBUTION = ROUND_ROBIN);
```

The screenshot shows the Azure Data Studio interface. On the left, there's a sidebar with a connection named "spegridehubsqldb (sqluser)" and a note about limited object explorer. Below it are links for Tables, Views, and Stored Procedures. The main area is titled "Query 1" and contains a SQL script to create a table:

```

1  CREATE TABLE [dbo].[Fact_WindTurbineMetrics] (
2      [DeviceId] nvarchar(50) COLLATE SQL_Latin1_General_CI_AS NULL,
3      [MeasureTime] datetime NULL,
4      [GeneratedPower] float NULL,
5      [WindSpeed] float NULL,
6      [TurbineSpeed] float NULL
7  )
8  WITH (CLUSTERED COLUMNSTORE INDEX, DISTRIBUTION = ROUND_ROBIN);

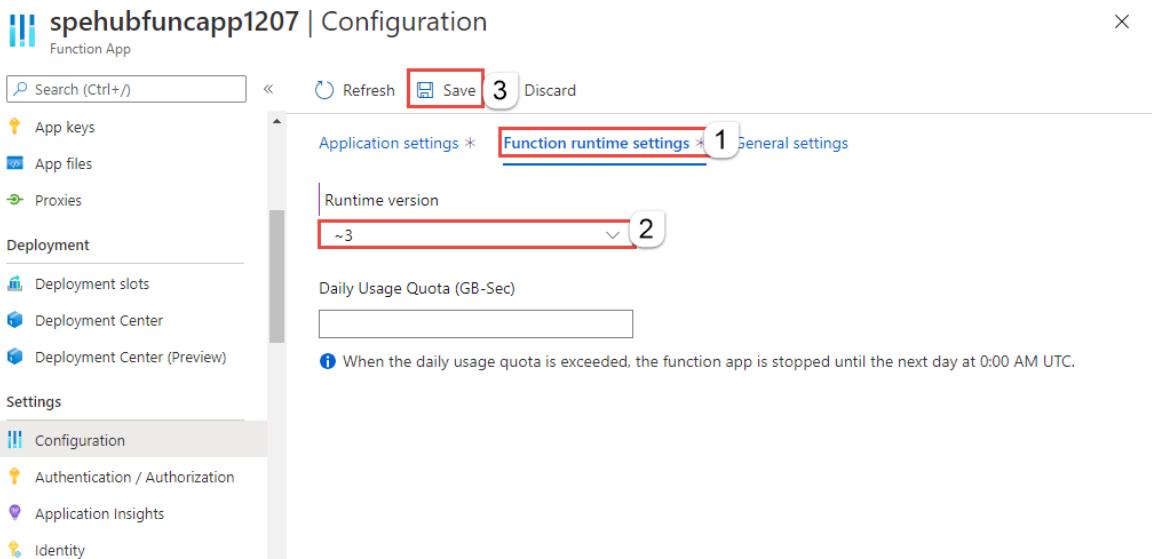
```

Below the query editor, there are tabs for "Results" and "Messages". The "Messages" tab shows the message "Query succeeded: Affected rows: 0". At the bottom, a yellow bar indicates "Query succeeded | 1s".

5. Keep this tab or window open so that you can verify that the data is created at the end of the tutorial.

### Update the function runtime version

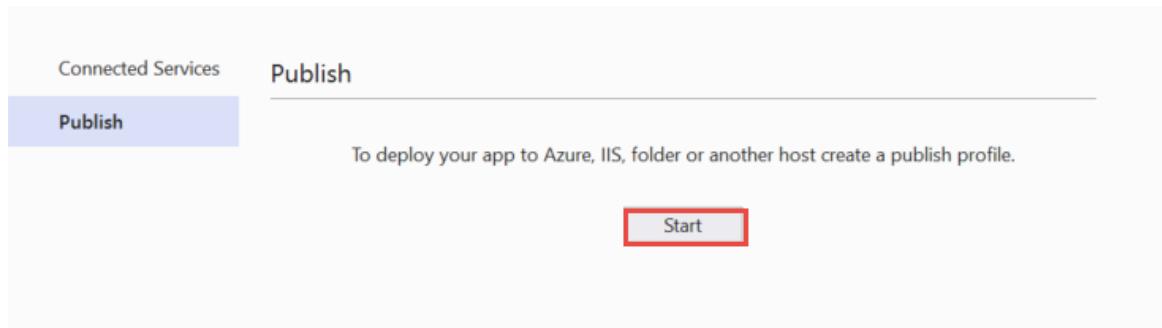
1. Open another tab in the web browser, and navigate to [Azure portal](#).
2. In the Azure portal, select **Resource groups** on the left menu.
3. Select the resource group in which the function app exists.
4. Select the **function app** in the list of resources in the resource group.
5. Select **Configuration** under **Settings** on the left menu.
6. Switch to the **Function runtime settings** tab in the right pane.
7. Update the **runtime version** to **~3**.



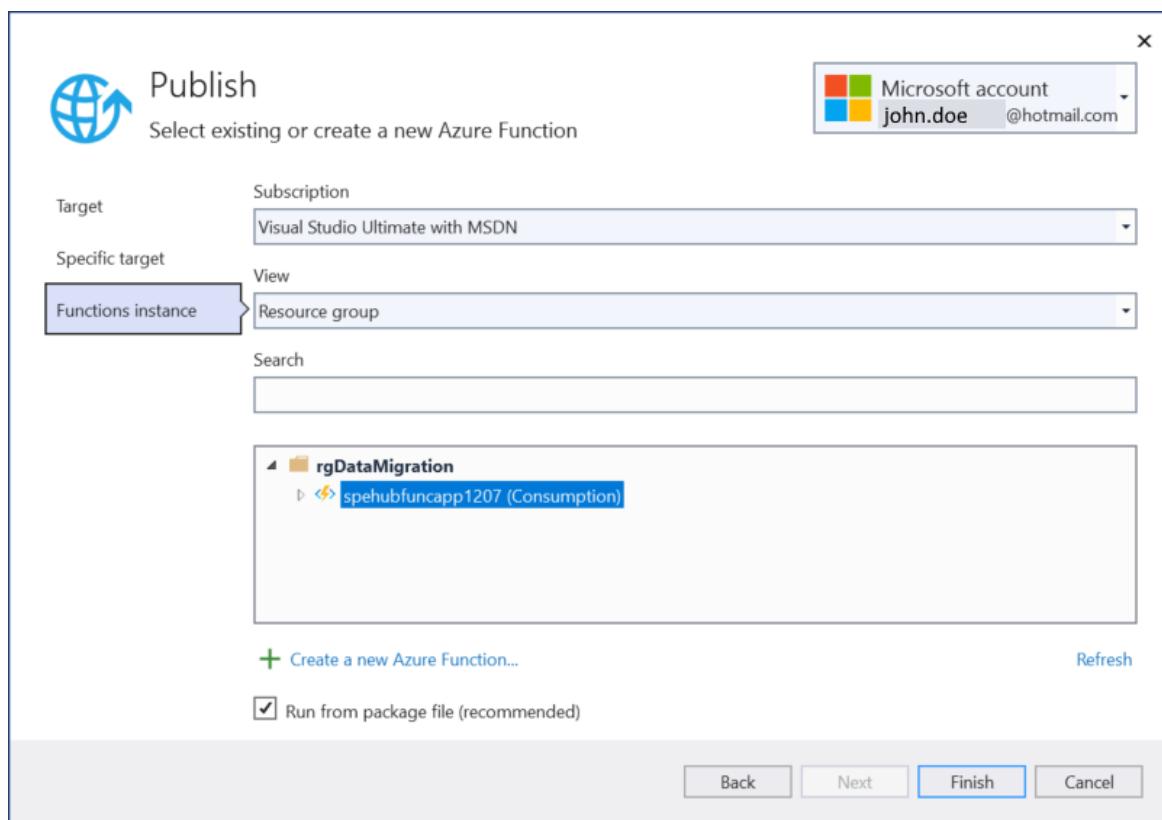
8. Select **Save** on the toolbar.
9. On the **Save changes** confirmation popup, select **Continue**.

# Publish the Azure Functions app

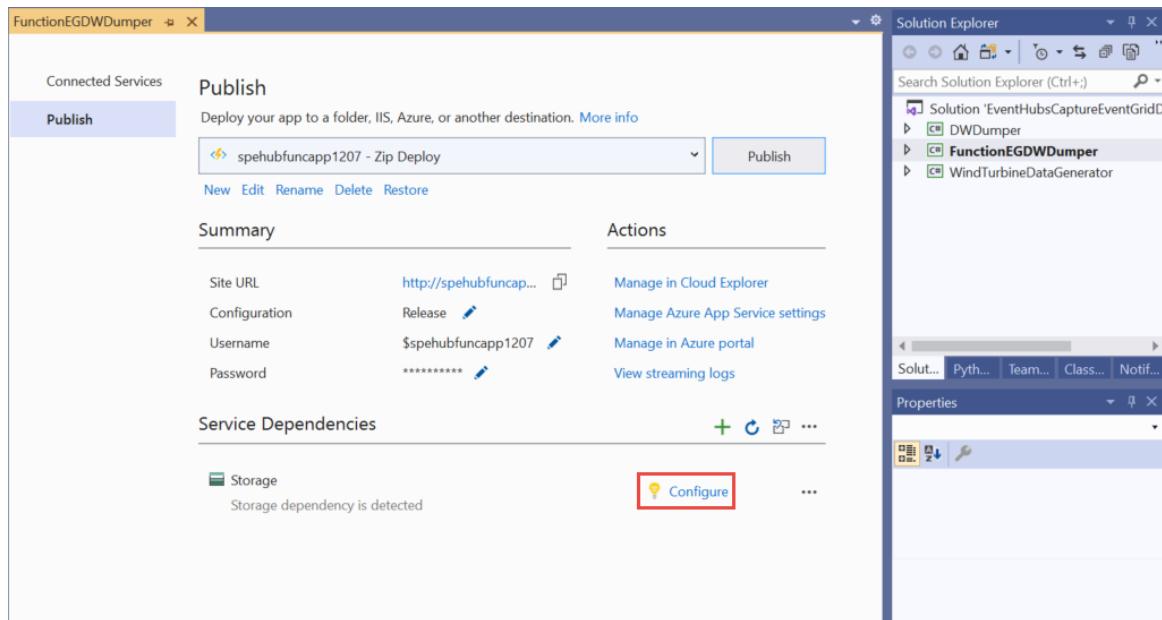
1. Launch Visual Studio.
2. Open the `EventHubsCaptureEventGridDemo.sln` solution that you downloaded from the [GitHub](#) as part of the prerequisites. You can find it in the `/samples/e2e/EventHubsCaptureEventGridDemo` folder.
3. In Solution Explorer, right-click `FunctionEGDWDumper` project, and select **Publish**.
4. If you see the following screen, select **Start**.



5. In the **Publish** dialog box, select **Azure** for **Target**, and select **Next**.
6. Select **Azure Function App (Windows)**, and select **Next**.
7. On the **Functions instance** tab, select your Azure subscription, expand the resource group, and select your function app, and then select **Finish**. You need to sign into your Azure account if you haven't already done so.

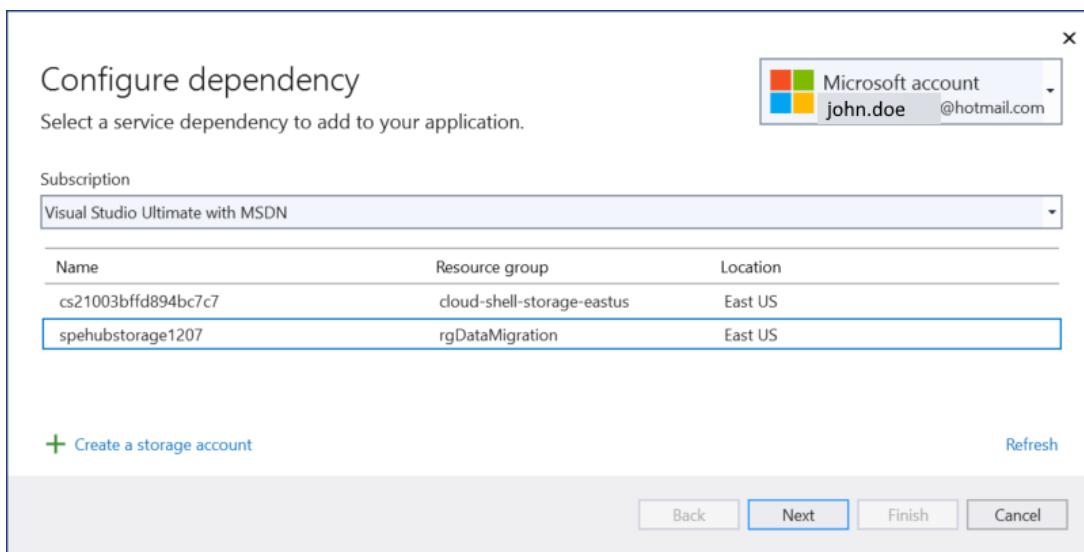


8. On the **Publish** page, in the **Service Dependencies** section, select **Configure for Storage**.

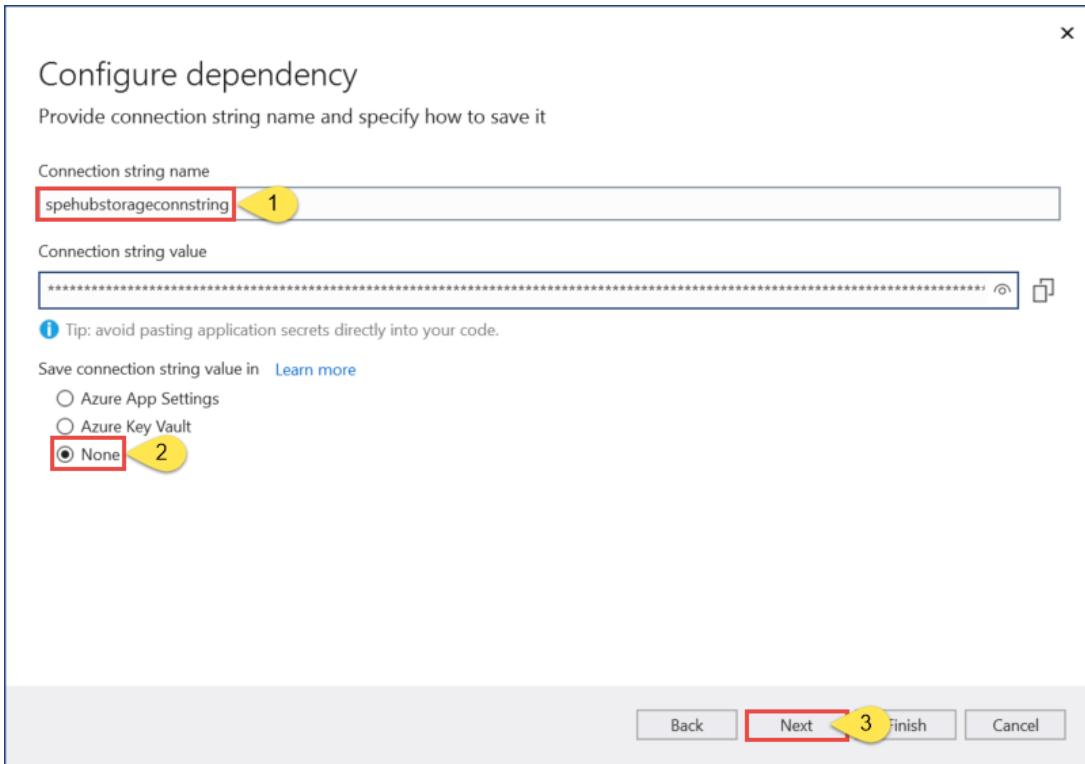


9. On the **Configure dependency** page, follow these steps:

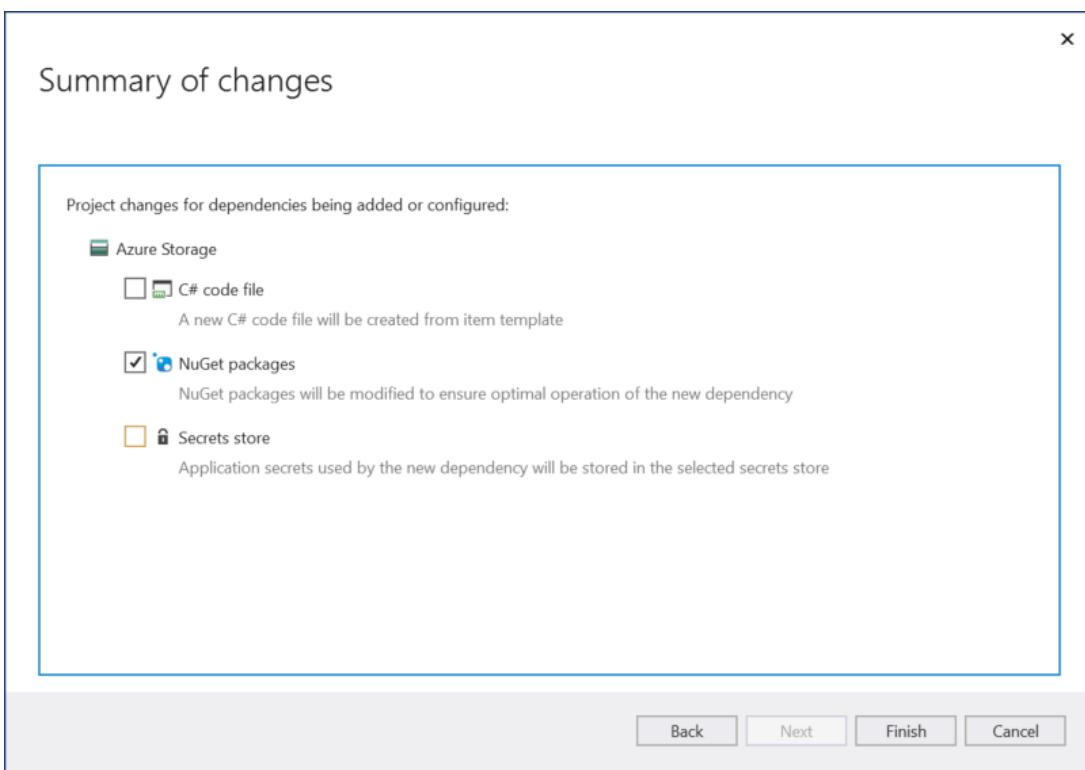
- a. select the **storage account** you created earlier, and then select **Next**.



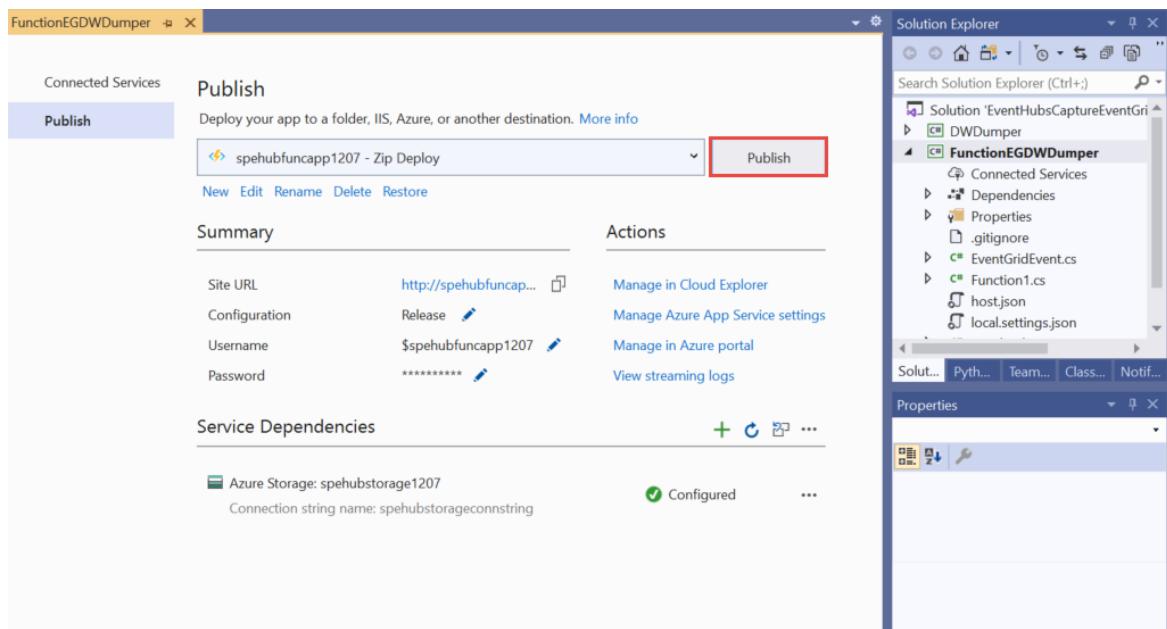
- b. Specify a name for the connection string, and select None for the Save connection string option, and then select **Next**.



- c. Clear the **C# code file** and **Secrets store** option, and then select **Finish**.



10. When Visual Studio has configured the profile, select **Publish**.



- In the tab that has the **Azure Function** page open, select **Functions** on the left menu. Confirm that the **EventGridTriggerMigrateData** function shows up in the list. If you don't see it, try publishing from Visual Studio again, and then refresh the page in the portal.

The screenshot shows the 'Functions' blade for the 'spehubfuncapp1207' function app. A red box highlights the 'Functions' link in the left sidebar. The main area displays a table of functions, with the first row ('EventGridTriggerMigrateData') highlighted by a red box. The table columns are Name, Trigger, and Status. The status for this function is 'Enabled'. A warning message in the center states: 'Your app is currently in read only mode because you are running from a package file. To make any changes update the content in your zip file and WEBSITE\_RUN\_FROM\_PACKAGE app setting.'

After publishing the function, you're ready to subscribe to the event.

## Subscribe to the event

- In a new tab or new window of a web browser, navigate to the [Azure portal](#).
- In the Azure portal, select **Resource groups** on the left menu.
- Filter the list of resource groups by entering the name of your resource group in the search box.
- Select your resource group in the list.
- Select the **Event Hubs namespace** from the list of resources.
- On the **Event Hubs Namespace** page, select **Events** on the left menu, and then select **+ Event Subscription** on the toolbar.

The screenshot shows the Azure portal interface for an Event Hubs Namespace named 'spehubns1207'. The left sidebar has a red box around the 'Events' item under the 'Settings' section. The main content area has a red box around the 'Event Subscriptions' tab. The page title is 'spehubns1207 | Events' and the sub-page title is 'Event Subscriptions'. A large heading 'Events, automated.' is displayed. Below it, a description states: 'Build reactive, event-driven apps with a fully managed event routing service that is built into Azure. Event Grid helps you build automation into your cloud infrastructure, create serverless apps, and integrate across services and clouds.' A 'Learn more' link is present. To the right, there's a section titled 'Example Scenarios' with three items: 'Execute code to resize an image whenever one is uploaded', 'Copy files with a certain extension to a different file store', and 'Process file content through a cognitive service and generate a metadata file'. At the bottom, a note says 'Azure Event Grid natively supports these resources as event handlers.' followed by a 'Learn more' link.

7. On the **Create Event Subscription** page, follow these steps:

- a. Enter a name for the **event subscription**.
- b. Enter a name for the **system topic**. A system topic provides an endpoint for the sender to send events. For more information, see [System topics](#)
- c. For **Endpoint Type**, select **Azure Function**.
- d. For **Endpoint**, select the link.
- e. On the **Select Azure Function** page, follow these steps if they aren't automatically filled.
  - a. Select the Azure subscription that has the Azure function.
  - b. Select the resource group for the function.
  - c. Select the function app.
  - d. Select the deployment slot.
  - e. Select the function **EventGridTriggerMigrateData**.
- f. On the **Select Azure Function** page, select **Confirm Selection**.
- g. Then, back on the **Create Event Subscription** page, select **Create**.

**Select Azure Function**

Subscription: Visual Studio Ultimate with MSDN

Resource group: rgDataMigration

Function app: spehubfuncapp1207

Slot: Production

Function: EventGridTriggerMigrateData

**Create Event Subscription**

**Event Grid**

**Basic Filters Additional Features**

**EVENT SUBSCRIPTION DETAILS**

Name: CaptureEventFuncSubscription

Event Schema: Event Grid Schema

**TOPIC DETAILS**

Pick a topic resource for which events should be pushed to your destination. Learn more

Topic Type: Event Hubs Namespace

Source Resource: spehubs1207

System Topic Name: spehubstoragesystopic

**EVENT TYPES**

Pick which event types get pushed to your destination. Learn more

Filter to Event Types: Capture File Created

**ENDPOINT DETAILS**

Pick an event handler to receive your events. Learn more

Endpoint Type: Azure Function (change)

Endpoint: Select an endpoint

**Create**

**Confirm Selection**

8. Verify that the event subscription is created. Switch to the **Event Subscriptions** tab on the **Events** page for the Event Hubs namespace.

**spehubs1207 | Events**

Event Hubs Namespace

**Event Subscription**

**Get Started Event Subscriptions**

**Essentials**

Show metrics: General Errors Latency Dead-Letter For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

Metrics chart:

- Published Events (Sum): 5
- Matched Events (Sum): 4
- Delivered Events (Sum): 2
- Dead Lettered Events (Sum): 1
- Delivery Failed Events (Sum): --
- Dropped Events (Sum): --

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
CaptureEventFuncSubscription	AzureFunction			Microsoft.EventHub.CaptureFileCreated

9. Select the App Service plan (not the App Service) in the list of resources in the resource group.

## Run the app to generate data

You've finished setting up your event hub, dedicate SQL pool (formerly SQL Data Warehouse), Azure function app, and event subscription. Before running an application that generates data for event hub, you need to configure a few values.

- In the Azure portal, navigate to your resource group as you did earlier.
- Select the Event Hubs namespace.
- In the **Event Hubs Namespace** page, select **Shared access policies** on the left menu.

4. Select **RootManageSharedAccessKey** in the list of policies.

**spehubns1207 | Shared access policies**

Event Hubs Namespace

Search (Ctrl+ /) << Add

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Events

Settings

Shared access policies

Scale

Geo-Recovery

Search to filter items...

Policy	Claims
RootManageSharedAccessKey	Manage, Send, Listen

5. Select the copy button next to the **Connection string-primary key** text box.

6. Go back to your Visual Studio solution.

7. Right-click **WindTurbineDataGenerator** project, and select Set as Startup project.

8. In the WindTurbineDataGenerator project, open **program.cs**.

9. Replace <EVENT HUBS NAMESPACE CONNECTION STRING> with the connection string you copied from the portal.

10. Replace <EVENT HUB NAME> with the name of the event hub.

```
private const string EventHubConnectionString =
"Endpoint=sb://demomigrationnamespace.servicebus.windows.net/...";
private const string EventHubName = "hubdatamigration";
```

11. Build the solution. Run the `WindTurbineGenerator.exe` application.

12. After a couple of minutes, in the other browser tab where you have the query window open, query the table in your data warehouse for the migrated data.

```
select * from [dbo].[Fact_WindTurbineMetrics]
```

The screenshot shows the Azure SQL Database Query editor interface. The left sidebar contains navigation links for connection strings, properties, locks, security, auditing, data discovery & classification, dynamic data masking, firewalls, virtual networks, security center, and transparent data encryption. Below these are common tasks like viewing streaming jobs, loading data, using the query editor (which is currently selected), building dashboards, modeling data, and opening in Visual Studio. The monitoring section includes query activity and alerts.

The main area displays a query titled "Query 1" with the following content:

```
1 select * from [Fact_WindTurbineMetrics]
```

A message in the object explorer states: "Showing limited object explorer here. For full capability please open SSDT."

The results pane shows a table with the following data:

DeviceId	MeasureTime	GeneratedPower	WindSpeed	TurbineSpeed
Turbine_0	2020-12-07T21:28:56.393...	47.5	285	5.7000028610229
Turbine_57	2020-12-07T21:29:11.137...	47.5	285	5.7000028610229
Turbine_0	2020-12-07T21:29:13.1363...	42.5	255	5.1000038146973
Turbine_0	2020-12-07T21:29:13.277...	27.5	165	3.3000019073486
Turbine_57	2020-12-07T21:29:24.303...	47.5	285	5.7000028610229
Turbine_14	2020-12-07T21:29:37.263...	15	90	1.8000007152557

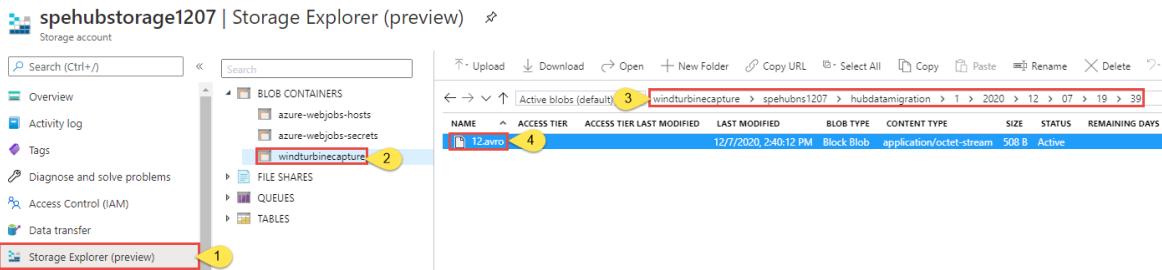
The status bar at the bottom indicates "Query succeeded | 0s".

# Monitor the solution

This section helps you with monitoring or troubleshooting the solution.

## View captured data in the storage account

1. Navigate to the resource group and select the storage account used for capturing event data.
2. On the **Storage account** page, select **Storage Explorer (preview)** on the left menu.
3. Expand **BLOB CONTAINERS**, and select **windturbinecapture**.
4. Open the folder named same as your **Event Hubs namespace** in the right pane.
5. Open the folder named same as your event hub (**hubdatamigration**).
6. Drill through the folders and you see the AVRO files. Here's an example:



## Verify that the Event Grid trigger invoked the function

1. Navigate to the resource group and select the function app.
2. Select **Functions** on the left menu.
3. Select the **EventGridTriggerMigrateData** function from the list.
4. On the **Function** page, select **Monitor** on the left menu.
5. Select **Configure** to configure application insights to capture invocation logs.
6. Create a new **Application Insights** resource or use an existing resource.
7. Navigate back to the **Monitor** page for the function.
8. Confirm that the client application (**WindTurbineDataGenerator**) that's sending the events is still running. If not, run the app.
9. Wait for a few minutes (5 minutes or more) and select the **Refresh** button to see function invocations.

The screenshot shows the Azure Function Monitor interface for the 'EventGridTriggerMigrateData' function. The top navigation bar includes a search bar, a back arrow, and tabs for 'Invocations' (which is selected), 'Logs', and 'Overview'. On the left, there's a sidebar with icons for 'Developer' (Code + Test, Integration), 'Monitor' (selected), and 'Function Keys'. The main content area is titled 'Invocation Traces' and displays a table of recent invocations. The table has columns for Date (UTC), Success, Result Code, and Duration (ms). All listed invocations are successful (202 status code) and took between 5 and 902 ms.

Date (UTC)	Success	Result Code	Duration (ms)
2020-12-07 21:50:12.881	✓ Success	202	11
2020-12-07 21:50:12.326	✓ Success	202	7
2020-12-07 21:49:12.576	✓ Success	202	7
2020-12-07 21:49:12.306	✓ Success	202	11
2020-12-07 21:48:12.549	✓ Success	202	5
2020-12-07 21:48:12.499	✓ Success	202	16
2020-12-07 21:47:22.007	✓ Success	202	902
2020-12-07 21:47:22.007	✓ Success	202	909

#### 10. Select an invocation to see details.

Event Grid distributes event data to the subscribers. The following example shows event data generated when data streaming through an event hub is captured in a blob. In particular, notice the `fileUrl` property in the `data` object points to the blob in the storage. The function app uses this URL to retrieve the blob file with captured data.

```
{
  "topic": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourcegroups/rgDataMigration/providers/Microsoft.EventHub/namespaces/spehubns1207",
  "subject": "hubdatamigration",
  "eventType": "Microsoft.EventHub.CaptureFileCreated",
  "id": "4538f1a5-02d8-4b40-9f20-36301ac976ba",
  "data": {
    "fileUrl":
      "https://spehubstorage1207.blob.core.windows.net/windturbinecapture/spehubns1207/hubdatamigration/0/2020/12/07/21/49/12.avro",
    "fileType": "AzureBlockBlob",
    "partitionId": "0",
    "sizeInBytes": 473444,
    "eventCount": 2800,
    "firstSequenceNumber": 55500,
    "lastSequenceNumber": 58299,
    "firstEnqueueTime": "2020-12-07T21:49:12.556Z",
    "lastEnqueueTime": "2020-12-07T21:50:11.534Z"
  },
  "dataVersion": "1",
  "metadataVersion": "1",
  "eventTime": "2020-12-07T21:50:12.7065524Z"
}
```

#### Verify that the data is stored in the dedicated SQL pool

In the browser tab where you have the query window open, query the table in your dedicated SQL pool for the migrated data.

The screenshot shows the Azure SQL Database Query editor (preview) interface. The left sidebar contains navigation links for Connection strings, Properties, Locks, Security, Auditing, Data Discovery & Classification, Dynamic Data Masking, Firewalls and virtual networks, Security Center, and Transparent data encryption. Under Common Tasks, the 'Query editor (preview)' link is selected. The main area has tabs for 'Query 1' and 'Results'. The 'Query 1' tab shows the T-SQL command: 'select \* from [Fact\_WindTurbineMetrics]'. The 'Results' tab displays a table with six rows of data:

DeviceId	MeasureTime	GeneratedPower	WindSpeed	TurbineSpeed
Turbine_0	2020-12-07T21:28:56.393...	47.5	285	5.7000028610229
Turbine_57	2020-12-07T21:29:11.137...	47.5	285	5.7000028610229
Turbine_0	2020-12-07T21:29:01.363...	42.5	255	5.1000038146973
Turbine_0	2020-12-07T21:29:13.277...	27.5	165	3.3000019073486
Turbine_57	2020-12-07T21:29:24.303...	47.5	285	5.7000028610229
Turbine_14	2020-12-07T21:29:37.263...	15	90	1.8000007152557

A message at the bottom of the results pane says 'Query succeeded | 0s'.

## Next steps

- To learn about differences in the Azure messaging services, see [Choose between Azure services that deliver messages](#).
- For an introduction to Event Grid, see [About Event Grid](#).
- For an introduction to Event Hubs Capture, see [Enable Event Hubs Capture using the Azure portal](#).
- For more information about setting up and running the sample, see [Event Hubs Capture and Event Grid sample](#).

# Tutorial: Route custom events to Azure Relay Hybrid Connections with Azure CLI and Event Grid

4/21/2021 • 4 minutes to read • [Edit Online](#)

Azure Event Grid is an eventing service for the cloud. Azure Relay Hybrid Connections is one of the supported event handlers. You use hybrid connections as the event handler when you need to process events from applications that don't have a public endpoint. These applications might be within your corporate enterprise network. In this article, you use the Azure CLI to create a custom topic, subscribe to the custom topic, and trigger the event to view the result. You send the events to the hybrid connection.

## Prerequisites

- This article assumes you already have a hybrid connection and a listener application. To get started with hybrid connections, see [Get started with Relay Hybrid Connections - .NET](#) or [Get started with Relay Hybrid Connections - Node](#).
- Use the Bash environment in [Azure Cloud Shell](#).  
[Launch Cloud Shell](#)
- If you prefer, [install](#) the Azure CLI to run CLI reference commands.
  - If you're using a local installation, sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal. For additional sign-in options, see [Sign in with the Azure CLI](#).
  - When you're prompted, install Azure CLI extensions on first use. For more information about extensions, see [Use extensions with the Azure CLI](#).
  - Run `az version` to find the version and dependent libraries that are installed. To upgrade to the latest version, run `az upgrade`.
- This article requires version 2.0.56 or later of the Azure CLI. If using Azure Cloud Shell, the latest version is already installed.

## Create a resource group

Event Grid topics are Azure resources, and must be placed in an Azure resource group. The resource group is a logical collection into which Azure resources are deployed and managed.

Create a resource group with the `az group create` command.

The following example creates a resource group named `gridResourceGroup` in the `westus2` location.

```
az group create --name gridResourceGroup --location westus2
```

## Create a custom topic

An event grid topic provides a user-defined endpoint that you post your events to. The following example creates the custom topic in your resource group. Replace `<topic_name>` with a unique name for your custom topic. The event grid topic name must be unique because it's represented by a DNS entry.

```
az eventgrid topic create --name <topic_name> -l westus2 -g gridResourceGroup
```

## Subscribe to a custom topic

You subscribe to an event grid topic to tell Event Grid which events you want to track. The following example subscribes to the custom topic you created, and passes the resource ID of the hybrid connection for the endpoint. The hybrid connection ID is in the format:

```
/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Relay/namespaces/<relay-namespace>/hybridConnections/<hybrid-connection-name>
```

The following script gets the resource ID of the relay namespace. It constructs the ID for the hybrid connection, and subscribes to an event grid topic. The script sets the endpoint type to `hybridconnection` and uses the hybrid connection ID for the endpoint.

```
relayname=<namespace-name>
relayrg=<resource-group-for-relay>
hybridname=<hybrid-name>

relayid=$(az resource show --name $relayname --resource-group $relayrg --resource-type Microsoft.Relay/namespaces --query id --output tsv)
hybridid="$relayid/hybridConnections/$hybridname"
topicid=$(az eventgrid topic show --name <topic_name> -g gridResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--source-resource-id $topicid \
--name <event_subscription_name> \
--endpoint-type hybridconnection \
--endpoint $hybridid \
--expiration-date "<yyyy-mm-dd>"
```

Notice that an [expiration date](#) is set for the subscription.

## Create application to process events

You need an application that can retrieve events from the hybrid connection. The [Microsoft Azure Event Grid Hybrid Connection Consumer sample for C#](#) performs that operation. You've already finished the prerequisite steps.

1. Make sure you have Visual Studio 2019 or later.
2. Clone the repository to your local machine.
3. Load HybridConnectionConsumer project in Visual Studio.
4. In Program.cs, replace `<relayConnectionString>` and `<hybridConnectionName>` with the relay connection string and hybrid connection name that you created.
5. Compile and run the application from Visual Studio.

## Send an event to your topic

Let's trigger an event to see how Event Grid distributes the message to your endpoint. This article shows how to use Azure CLI to trigger the event. Alternatively, you can use [Event Grid publisher application](#).

First, let's get the URL and key for the custom topic. Again, use your custom topic name for `<topic_name>`.

```
endpoint=$(az eventgrid topic show --name <topic_name> -g gridResourceGroup --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name <topic_name> -g gridResourceGroup --query "key1" --output tsv)
```

To simplify this article, you use sample event data to send to the custom topic. Typically, an application or Azure service would send the event data. CURL is a utility that sends HTTP requests. In this article, use CURL to send the event to the custom topic.

```
event='[ {"id": """$RANDOM""", "eventType": "recordInserted", "subject": "myapp/vehicles/motorcycles",
"eventTime": "'`date +%Y-%m-%dT%H:%M:%S%z`'", "data":{ "make": "Ducati", "model": "Monster"}, "dataVersion":
"1.0"} ]'
curl -X POST -H "aeg-sas-key: $key" -d "$event" $endpoint
```

Your listener application should receive the event message.

## Clean up resources

If you plan to continue working with this event, don't clean up the resources created in this article. Otherwise, use the following command to delete the resources you created in this article.

```
az group delete --name gridResourceGroup
```

## Next steps

Now that you know how to create topics and event subscriptions, learn more about what Event Grid can help you do:

- [About Event Grid](#)
- [Route Blob storage events to a custom web endpoint](#)
- [Monitor virtual machine changes with Azure Event Grid and Logic Apps](#)
- [Stream big data into a data warehouse](#)

# Azure CLI samples for Event Grid

11/2/2020 • 2 minutes to read • [Edit Online](#)

The following table includes links to Azure CLI samples for Event Grid.

## Event Grid subscriptions

- [Subscribe to Azure subscription](#) - Subscribes to events for an Azure subscription.
- [Subscribe to Blob storage](#) - Subscribes to events for a Blob storage account.
- [Subscribe to custom topic](#) - Subscribes to events for a custom topic.
- [Subscribe to resource group](#) - Subscribes to events for a resource group.
- [Subscribe to resource group and filter for a resource](#) - Subscribes to events for a resource group and filters events for a resource.

## Event Grid topics

- [Create custom topic](#) - Creates an Event Grid custom topic, and returns the endpoint and key.

# Create Event Grid custom topic with Azure CLI

4/21/2021 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid custom topic.

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Sample script

```
#!/bin/bash

# Give your custom topic a unique name
myTopic=demoContosoTopic

# Provide name for resource group
myResourceGroup=demoResourceGroup

# Select the Azure subscription to contain the custom topic.
az account set --subscription "<name or ID of the subscription>"

# Create resource group
az group create --name $myResourceGroup --location westus2

# Create custom topic
az eventgrid topic create --resource-group $myResourceGroup --name $myTopic --location westus2

# Retrieve endpoint and key to use when publishing to the topic
endpoint=$(az eventgrid topic show --name $myTopic -g $myResourceGroup --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name $myTopic -g $myResourceGroup --query "key1" --output tsv)

echo $endpoint
echo $key
```

## Script explanation

This script uses the following command to create the custom topic. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az eventgrid topic create</a>	Create an Event Grid custom topic.

## Next steps

- For information about querying subscriptions, see [Query Event Grid subscriptions](#).
- For more information on the Azure CLI, see [Azure CLI documentation](#).

# Subscribe to events for an Azure subscription with Azure CLI

4/22/2021 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for an Azure subscription.

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

The preview sample script requires the Event Grid extension. To install, run `az extension add --name eventgrid`.

## Sample script - stable

```
#!/bin/bash

# Provide an endpoint for handling the events.
myEndpoint=<endpoint URL>

# Select the Azure subscription you want to subscribe to.
az account set --subscription "<name or ID of the subscription>"

# Subscribe to the Azure subscription. The command creates the subscription for the currently selected Azure
# subscription.
az eventgrid event-subscription create --name demoSubscriptionToAzureSub --endpoint $myEndpoint
```

## Sample script - preview extension

```
#!/bin/bash

# You must have the latest version of the Event Grid preview extension.
# If you have not installed previously:
# az extension add -n eventgrid
# If you have installed previously:
# az extension update -n eventgrid

# Provide an endpoint for handling the events.
myEndpoint=<endpoint URL>

# Select the Azure subscription you want to subscribe to.
az account set --subscription "<name or ID of the subscription>"

# Get the subscription ID
subID=$(az account show --query id --output tsv)

# Subscribe to the Azure subscription. The command creates the subscription for the currently selected Azure
# subscription.
az eventgrid event-subscription create --name demoSubscriptionToAzureSub --endpoint $myEndpoint --source-
resource-id /subscriptions/$subID
```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az eventgrid event-subscription create</a>	Create an Event Grid subscription.
<a href="#">az eventgrid event-subscription create - extension version</a>	Create an Event Grid subscription.

## Next steps

- For information about querying subscriptions, see [Query Event Grid subscriptions](#).
- For more information on the Azure CLI, see [Azure CLI documentation](#).

# Subscribe to events for a Blob storage account with Azure CLI

4/22/2021 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a Blob storage account.

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Sample script

```
#!/bin/bash

# You must have the latest version of the Azure CLI

# Provide a unique name for the Blob storage account.
storageName=<Name of the storage account>

# Provide an endpoint for handling the events.
myEndpoint=<endpoint URL>

# Provide the name of the resource group to contain the storage account.
myResourceGroup=<resource group name>

# Select the Azure subscription to contain the storage account and event subscription.
az account set --subscription "<name or ID of the subscription>"

# Create resource group
az group create --name $myResourceGroup --location westus2

# Create the Blob storage account.
az storage account create \
    --name $storageName \
    --location westus2 \
    --resource-group $myResourceGroup \
    --sku Standard_LRS \
    --kind BlobStorage \
    --access-tier Hot

# Get the resource ID of the Blob storage account.
storageid=$(az storage account show --name $storageName --resource-group $myResourceGroup --query id --output tsv)

# Subscribe to the Blob storage account.
az eventgrid event-subscription create \
    --source-resource-id $storageid \
    --name demoSubToStorage \
    --endpoint $myEndpoint
```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to

command-specific documentation.

COMMAND	NOTES
<code>az eventgrid event-subscription create</code>	Create an Event Grid subscription.
<code>az eventgrid event-subscription create - extension version</code>	Create an Event Grid subscription.

## Next steps

- For information about querying subscriptions, see [Query Event Grid subscriptions](#).
- For more information on the Azure CLI, see [Azure CLI documentation](#).

# Subscribe to events for a custom topic with Azure CLI

4/22/2021 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a custom topic.

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

The preview sample script requires the Event Grid extension. To install, run `az extension add --name eventgrid`.

## Sample script - stable

```
#!/bin/bash

# Provide the name of the topic you are subscribing to
myTopic=demoContosoTopic

# Provide the name of the resource group containing the custom topic
resourceGroup=demoResourceGroup

# Provide an endpoint for handling the events.
myEndpoint="<endpoint URL>

# Select the Azure subscription that contains the custom topic.
az account set --subscription "<name or ID of the subscription>"

# Subscribe to the custom event. Include the resource group that contains the custom topic.
az eventgrid event-subscription create \
--resource-group $resourceGroup \
--topic-name $myTopic \
--name demoSubscription \
--endpoint $myEndpoint
```

## Sample script - preview extension

```

#!/bin/bash

# You must have the latest version of the Event Grid preview extension.
# If you have not installed previously:
# az extension add -n eventgrid
# If you have installed previously:
# az extension update -n eventgrid

# Provide the name of the topic you are subscribing to
myTopic=demoContosoTopic

# Provide the name of the resource group containing the custom topic
resourceGroup=demoResourceGroup

# Provide an endpoint for handling the events.
myEndpoint="

```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az eventgrid event-subscription create</a>	Create an Event Grid subscription.
<a href="#">az eventgrid event-subscription create - extension version</a>	Create an Event Grid subscription.

## Next steps

- For information about querying subscriptions, see [Query Event Grid subscriptions](#).
- For more information on the Azure CLI, see [Azure CLI documentation](#).

# Subscribe to events for a resource group with Azure CLI

4/22/2021 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a resource group.

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

The preview sample script requires the Event Grid extension. To install, run `az extension add --name eventgrid`.

## Sample script - stable

```
#!/bin/bash

# Provide an endpoint for handling the events.
myEndpoint=<endpoint URL>

# Provide the name of the resource group to subscribe to.
myResourceGroup=<resource group name>

# Select the Azure subscription that contains the resource group.
az account set --subscription "<name or ID of the subscription>"

# Subscribe to the resource group. Provide the name of the resource group you want to subscribe to.
az eventgrid event-subscription create \
--name demoSubscriptionToResourceGroup \
--resource-group $myResourceGroup \
--endpoint $myEndpoint
```

## Sample script - preview extension

```

#!/bin/bash

# Provide an endpoint for handling the events.
myEndpoint=""

# Provide the name of the resource group to subscribe to.
myResourceGroup=<resource group name>

# Select the Azure subscription that contains the resource group.
az account set --subscription "<name or ID of the subscription>"

# Get resource ID of the resource group.
resourceGroupID=$(az group show --name $myResourceGroup --query id --output tsv)

# Subscribe to the resource group. Provide the name of the resource group you want to subscribe to.
az eventgrid event-subscription create \
--name demoSubscriptionToResourceGroup \
--source-resource-id $resourceGroupID \
--endpoint $myEndpoint

```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az eventgrid event-subscription create</a>	Create an Event Grid subscription.
<a href="#">az eventgrid event-subscription create - extension version</a>	Create an Event Grid subscription.

## Next steps

- For information about querying subscriptions, see [Query Event Grid subscriptions](#).
- For more information on the Azure CLI, see [Azure CLI documentation](#).

# Subscribe to events for a resource group and filter for a resource with Azure CLI

4/22/2021 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a resource group. It uses a filter to get only events for a specified resource in the resource group.

To run this sample, install the latest version of the [Azure CLI](#). To start, run `az login` to create a connection with Azure.

Samples for the Azure CLI are written for the `bash` shell. To run this sample in Windows PowerShell or Command Prompt, you may need to change elements of the script.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

The preview sample script requires the Event Grid extension. To install, run `az extension add --name eventgrid`.

## Sample script - stable

```
#!/bin/bash

# Provide an endpoint for handling the events.
myEndpoint="<endpoint URL>

# Select the Azure subscription that contains the resource group.
az account set --subscription "<name or ID of the subscription>"

# Get the resource ID to filter events
resourceId=$(az resource show --name demoSecurityGroup --resource-group myResourceGroup --resource-type Microsoft.Network/networkSecurityGroups --query id --output tsv)

# Subscribe to the resource group. Provide the name of the resource group you want to subscribe to.
az eventgrid event-subscription create \
--name demoSubscriptionToResourceGroup \
--resource-group myResourceGroup \
--endpoint $myEndpoint \
--subject-begins-with $resourceId
```

## Sample script - preview extension

```

#!/bin/bash

# Provide an endpoint for handling the events.
myEndpoint=<endpoint URL>

# Provide the name of the custom topic to create
topicName=<your-topic-name>

# Provide the name of the resource group to contain the custom topic
myResourceGroup=demoResourceGroup

# Select the Azure subscription that contains the resource group.
az account set --subscription "<name or ID of the subscription>"

# Create the resource group
az group create -n $myResourceGroup -l eastus2

# Create custom topic
az eventgrid topic create --name $topicName -l eastus2 -g $myResourceGroup

# Get resource ID of custom topic
topicid=$(az eventgrid topic show --name $topicName -g $myResourceGroup --query id --output tsv)

# Subscribe to the custom topic. Filter based on a value in the event data.
az eventgrid event-subscription create \
--source-resource-id $topicid \
-n demoAdvancedFilterSub \
--advanced-filter data.color stringin blue red green \
--endpoint $endpointURL

```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">az eventgrid event-subscription create</a>	Create an Event Grid subscription.
<a href="#">az eventgrid event-subscription create - extension version</a>	Create an Event Grid subscription.

## Next steps

- For information about querying subscriptions, see [Query Event Grid subscriptions](#).
- For more information on the Azure CLI, see [Azure CLI documentation](#).

# Azure PowerShell samples for Event Grid

11/2/2020 • 2 minutes to read • [Edit Online](#)

The following table includes links to Azure PowerShell samples for Event Grid.

## Event Grid subscriptions

- [Subscribe to Azure subscription](#) - Subscribes to events for an Azure subscription.
- [Subscribe to Blob storage](#) - Subscribes to events for a Blob storage account.
- [Subscribe to custom topic](#) - Subscribes to events for a custom topic.
- [Subscribe to resource group](#) - Subscribes to events for a resource group.
- [Subscribe to resource group and filter for a resource](#) - Subscribes to events for a resource group and filters events for a resource.

## Event Grid topics

- [Create custom topic](#) - Creates an Event Grid custom topic, and returns the endpoint and key.

# Create Event Grid custom topic with PowerShell

11/2/2020 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid custom topic.

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Sample script

```
# Give your custom topic a unique name
$myTopic = "<your-custom-topic-name>"

# Provide a name for resource group to create. It will contain the custom event.
$myResourceGroup = "<resource-group-name>"

# Create resource group
New-AzResourceGroup -Name $myResourceGroup -Location westus2

# Create custom topic
New-AzEventGridTopic -ResourceGroupName $myResourceGroup -Name $myTopic -Location westus2

# Retrieve endpoint and key to use when publishing to the topic
$endpoint = (Get-AzEventGridTopic -ResourceGroupName $myResourceGroup -Name $myTopic).Endpoint
$key = (Get-AzEventGridTopicKey -ResourceGroupName $myResourceGroup -Name $myTopic).Key1

$endpoint
$key
```

## Script explanation

This script uses the following command to create the custom topic. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">New-AzEventGridTopic</a>	Create an Event Grid custom topic.

## Next steps

- For an introduction to managed applications, see [Azure Managed Application overview](#).
- For more information on PowerShell, see [Azure PowerShell documentation](#).

# Subscribe to events for an Azure subscription with PowerShell

11/2/2020 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for an Azure subscription.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Sample script - stable

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

```
# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"
$myEndpoint = "<your-endpoint-URL>

# Select the Azure subscription you want to subscribe to. You need this command only if the
# current subscription is not the one you wish to subscribe to.
Set-AzContext -Subscription "<subscription-name-or-ID>

# Subscribe to the Azure subscription. The command creates the subscription for the currently selected Azure
subscription.
New-AzEventGridSubscription -Endpoint $myEndpoint -EventSubscriptionName demoSubscriptionToAzureSub
```

## Sample script - preview module

This preview sample script requires the Event Grid module. To install, run

```
Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery
```

### IMPORTANT

Using this Azure feature from PowerShell requires the [AzureRM](#) module installed. This is an older module only available for Windows PowerShell 5.1 that no longer receives new features. The [Az](#) and [AzureRM](#) modules are **not** compatible when installed for the same versions of PowerShell. If you need both versions:

1. [Uninstall the Az module](#) from a PowerShell 5.1 session.
2. [Install the AzureRM module](#) from a PowerShell 5.1 session.
3. [Download and install PowerShell Core 6.x or later](#).
4. [Install the Az module](#) in a PowerShell Core session.

```

# You must have the latest version of the Event Grid PowerShell module.
# To install:
# Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery

# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"
$myEndpoint = "<your-endpoint-URL>"

# Get the subscription ID
$subID = (Get-AzureRmSubscription -SubscriptionName "<subscription-name>").Id

# Subscribe to the Azure subscription. The command creates the subscription for the currently selected Azure
# subscription.
New-AzureRmEventGridSubscription -ResourceId "/subscriptions/$subID" -Endpoint $myEndpoint -
EventSubscriptionName demoSubscriptionToAzureSub

```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">New-AzEventGridSubscription</a>	Create an Event Grid subscription.

## Next steps

- For an introduction to managed applications, see [Azure Managed Application overview](#).
- For more information on PowerShell, see [Azure PowerShell documentation](#).

# Subscribe to events for a Blob storage account with PowerShell

11/2/2020 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a Blob storage account.

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Sample script - stable

```
# Provide a unique name for the Blob storage account.  
$storageName = "<your-unique-storage-name>"  
  
# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"  
$myEndpoint = "<your-endpoint-URL>"  
  
# Provide the name of the resource group to create. It will contain the storage account.  
$myResourceGroup="<resource-group-name>"  
  
# Create resource group  
New-AzResourceGroup -Name $myResourceGroup -Location westus2  
  
# Create the Blob storage account.  
New-AzStorageAccount -ResourceGroupName $myResourceGroup `  
    -Name $storageName `  
    -Location westus2 `  
    -SkuName Standard_LRS `  
    -Kind BlobStorage `  
    -AccessTier Hot  
  
# Get the resource ID of the Blob storage account.  
$storageId = (Get-AzStorageAccount -ResourceGroupName $myResourceGroup -AccountName $storageName).Id  
  
# Subscribe to the Blob storage account.  
New-AzEventGridSubscription `  
    -EventSubscriptionName demoSubToStorage `  
    -Endpoint $myEndpoint `  
    -ResourceId $storageId
```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">New-AzEventGridSubscription</a>	Create an Event Grid subscription.

## Next steps

- For an introduction to managed applications, see [Azure Managed Application overview](#).
- For more information on PowerShell, see [Azure PowerShell documentation](#).

# Subscribe to events for a custom topic with PowerShell

11/2/2020 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a custom topic.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

The preview sample script requires the Event Grid module. To install, run

```
Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery
```

## Sample script - stable

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

```
# Provide the name of the topic you are subscribing to
$myTopic = "<your-custom-topic-name>

# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"
$myEndpoint = "<your-endpoint-URL>

# Provide a name for resource group to create. It will contain the custom event.
$myResourceGroup = "<resource-group-name>

# Create resource group
New-AzResourceGroup -Name $myResourceGroup -Location westus2

# Create custom topic
New-AzEventGridTopic -ResourceGroupName $myResourceGroup -Name $myTopic -Location westus2

# Subscribe to the custom event. Include the resource group that contains the custom topic.
New-AzEventGridSubscription `

    -EventSubscriptionName demoSubscription `

    -Endpoint $myEndpoint `

    -ResourceGroupName $myResourceGroup `

    -TopicName $myTopic
```

## Sample script - preview module

## IMPORTANT

Using this Azure feature from PowerShell requires the `AzureRM` module installed. This is an older module only available for Windows PowerShell 5.1 that no longer receives new features. The `Az` and `AzureRM` modules are **not** compatible when installed for the same versions of PowerShell. If you need both versions:

1. [Uninstall the Az module](#) from a PowerShell 5.1 session.
2. [Install the AzureRM module](#) from a PowerShell 5.1 session.
3. [Download and install PowerShell Core 6.x or later](#).
4. [Install the Az module](#) in a PowerShell Core session.

```
# You must have the latest version of the Event Grid PowerShell module.  
# To install:  
# Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery  
  
# Provide the name of the topic you are subscribing to  
$myTopic = "<your-custom-topic-name>"  
  
# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"  
$myEndpoint = "<your-endpoint-URL>"  
  
# Provide the name of the resource group to create. It will contain the custom topic.  
$myResourceGroup = "<resource-group-name>"  
  
# Create resource group  
New-AzResourceGroup -Name $myResourceGroup -Location westus2  
  
# Create custom topic and get its resource ID.  
$topicID = (New-AzEventGridTopic -ResourceGroupName $myResourceGroup -Name $myTopic -Location westus2).Id  
  
# Subscribe to the custom event. Include the resource group that contains the custom topic.  
New-AzEventGridSubscription `  
    -ResourceId $topicID `  
    -EventSubscriptionName demoSubscription `  
    -Endpoint $myEndpoint
```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">New-AzEventGridSubscription</a>	Create an Event Grid subscription.

## Next steps

- For an introduction to managed applications, see [Azure Managed Application overview](#).
- For more information on PowerShell, see [Azure PowerShell documentation](#).

# Subscribe to events for a resource group with PowerShell

11/2/2020 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a resource group.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

The preview sample script requires the Event Grid module. To install, run

```
Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery
```

## Sample script - stable

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

```
# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"
$myEndpoint = "<your-endpoint-URL>"

# Provide the name of the resource group to create and subscribe to.
$myResourceGroup="<resource-group-name>"

# Create resource group
New-AzResourceGroup -Name $myResourceGroup -Location westus2

# Subscribe to the resource group. Provide the name of the resource group you want to subscribe to.
New-AzEventGridSubscription ` 
    -Endpoint $myEndpoint ` 
    -EventSubscriptionName demoSubscriptionToResourceGroup ` 
    -ResourceGroupName $myResourceGroup
```

## Sample script - preview module

### IMPORTANT

Using this Azure feature from PowerShell requires the [AzureRM](#) module installed. This is an older module only available for Windows PowerShell 5.1 that no longer receives new features. The [Az](#) and [AzureRM](#) modules are **not** compatible when installed for the same versions of PowerShell. If you need both versions:

1. [Uninstall the Az module](#) from a PowerShell 5.1 session.
2. [Install the AzureRM module](#) from a PowerShell 5.1 session.
3. [Download and install PowerShell Core 6.x or later](#).
4. [Install the Az module](#) in a PowerShell Core session.

```

# You must have the latest version of the Event Grid PowerShell module.
# To install:
# Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery

# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"
$myEndpoint = "<your-endpoint-URL>"

# Provide the name of the resource group to create and subscribe to.
$myResourceGroup = "<resource-group-name>"

# Create resource group
$resourceGroupID = (New-AzResourceGroup -Name $myResourceGroup -Location westus2).ResourceId

# Subscribe to the resource group. Provide the name of the resource group you want to subscribe to.
New-AzEventGridSubscription `

    -ResourceId $resourceGroupID `

    -Endpoint $myEndpoint `

    -EventSubscriptionName demoSubscriptionToResourceGroup

```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">New-AzEventGridSubscription</a>	Create an Event Grid subscription.

## Next steps

- For an introduction to managed applications, see [Azure Managed Application overview](#).
- For more information on PowerShell, see [Azure PowerShell documentation](#).

# Subscribe to events for a resource group and filter for a resource with PowerShell

11/2/2020 • 2 minutes to read • [Edit Online](#)

This script creates an Event Grid subscription to the events for a resource group. It uses a filter to get only events for a specified resource in the resource group.

If you don't have an [Azure subscription](#), create a [free account](#) before you begin.

## Sample script - stable

### NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

```
# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"
$myEndpoint = "<your-endpoint-URL>"

# Provide the name of the resource group to create. It will contain the network security group.
# You will subscribe to events for this resource group.
$myResourceGroup = "<resource-group-name>"

# Provide a name for the network security group to create.
$nsgName = "<your-nsg-name>"

# Create the resource group
New-AzResourceGroup -Name $myResourceGroup -Location westus2

# Create a network security group. You will filter events to only those that are related to this resource.
New-AzNetworkSecurityGroup -Name $nsgName -ResourceGroupName $myResourceGroup -Location westus2

# Get the resource ID to filter events. The name of the network security group must not be the same as the other resource names.
$resourceId = (Get-AzResource -ResourceName $nsgName -ResourceGroupName $myResourceGroup).ResourceId

# Subscribe to the resource group. Provide the name of the resource group you want to subscribe to.
New-AzEventGridSubscription ` 
    -Endpoint $myEndpoint ` 
    -EventSubscriptionName demoSubscriptionToResourceGroup ` 
    -ResourceGroupName $myResourceGroup ` 
    -SubjectBeginsWith $resourceId
```

## Sample script - preview module

## IMPORTANT

Using this Azure feature from PowerShell requires the `AzureRM` module installed. This is an older module only available for Windows PowerShell 5.1 that no longer receives new features. The `Az` and `AzureRM` modules are **not** compatible when installed for the same versions of PowerShell. If you need both versions:

1. [Uninstall the Az module](#) from a PowerShell 5.1 session.
2. [Install the AzureRM module](#) from a PowerShell 5.1 session.
3. [Download and install PowerShell Core 6.x or later](#).
4. [Install the Az module](#) in a PowerShell Core session.

The preview sample script requires the Event Grid module. To install, run

```
Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery
```

```
# You must have the latest version of the Event Grid PowerShell module.  
# To install:  
# Install-Module -Name AzureRM.EventGrid -AllowPrerelease -Force -Repository PSGallery  
  
# Provide an endpoint for handling the events. Must be formatted "https://your-endpoint-URL"  
$myEndpoint = "<your-endpoint-URL>"  
  
# Provide the name of the custom topic to create  
$topicName = "<your-topic-name>"  
  
# Provide the name of the resource group to create. It will contain the custom topic.  
$myResourceGroup= "<resource-group-name>"  
  
# Create the resource group  
New-AzResourceGroup -Name $myResourceGroup -Location westus2  
  
# Create custom topic  
New-AzEventGridTopic -ResourceGroupName $myResourceGroup -Location westus2 -Name $topicName  
  
# Get resource ID of custom topic  
$topicid = (Get-AzEventGridTopic -ResourceGroupName $myResourceGroup -Name $topicName).Id  
  
# Set the operator type, field and values for the filtering  
$AdvFilter1=@{operator="StringIn"; key="Data.color"; Values=@('blue', 'red', 'green')}  
  
# Subscribe to the custom topic. Filter based on a value in the event data.  
New-AzEventGridSubscription `  
    -ResourceId $topicid `  
    -EventSubscriptionName demoSubWithFilter `  
    -Endpoint $myEndpoint `  
    -AdvancedFilter $($AdvFilter1)
```

## Script explanation

This script uses the following command to create the event subscription. Each command in the table links to command-specific documentation.

COMMAND	NOTES
<a href="#">New-AzEventGridSubscription</a>	Create an Event Grid subscription.

## Next steps

- For an introduction to managed applications, see [Azure Managed Application overview](#).

- For more information on PowerShell, see [Azure PowerShell documentation](#).

# Azure Resource Manager templates for Event Grid

5/11/2021 • 2 minutes to read • [Edit Online](#)

For the JSON syntax and properties to use in a template, see [Microsoft.EventGrid resource types](#). The following table includes links to Azure Resource Manager templates for Event Grid.

## Event Grid subscriptions

- [Custom topic and subscription with WebHook endpoint](#) - Deploys an Event Grid custom topic. Creates a subscription to that custom topic that uses a WebHook endpoint.
- [Custom topic subscription with EventHub endpoint](#) - Creates an Event Grid subscription to a custom topic. The subscription uses an Event Hub for the endpoint.
- [Azure subscription or resource group subscription](#) - Subscribes to events for a resource group or Azure subscription. The resource group you specify as the target during deployment is the source of events. The subscription uses a WebHook for the endpoint.
- [Blob storage account and subscription](#) - Deploys an Azure Blob storage account and subscribes to events for that storage account.

## Next steps

See the following samples:

- [PowerShell samples](#)
- [CLI samples](#)

# Concepts in Azure Event Grid

3/5/2021 • 5 minutes to read • [Edit Online](#)

This article describes the main concepts in Azure Event Grid.

## 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.

The maximum allowed size for an event is 1 MB. Events over 64 KB are charged in 64-KB increments. For the properties that are sent in an event, see [Azure Event Grid event schema](#).

## Publishers

A publisher is the user or organization that decides to send events to Event Grid. Microsoft publishes events for several Azure services. You can publish events from your own application. Organizations that host services outside of Azure can publish events through Event Grid.

## 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.

For information about implementing any of the supported Event Grid sources, see [Event sources in Azure 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 such as Azure Storage, Azure Event Hubs, and Azure Service Bus. You can create system topics in your Azure subscription and subscribe to them. For more information, see [Overview of system topics](#).

**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. For more information, see [Custom topics](#). When designing your application, you have flexibility when deciding how many topics to create. For large solutions, create a custom topic for each category of related events. For example, consider an application that sends events related to modifying user accounts and processing orders. It's unlikely any event handler wants both categories of events. Create two custom topics and let event handlers subscribe to the one that interests them. For small solutions, you might prefer to send all events to a single topic. Event subscribers can filter for the event types they want.

There is another type of topic: **partner topic**. The [Partner Events](#) feature allows a third-party SaaS provider to publish events from its services to make them available to consumers who can subscribe to those events. The SaaS provider exposes a topic type, a **partner topic**, that subscribers use to consume events. It also offers a clean pub-sub model by separating concerns and ownership of resources that are used by event publishers and subscribers.

## 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. For more information, see [Event Grid subscription schema](#).

For examples of creating subscriptions, see:

- [Azure CLI samples for Event Grid](#)
- [Azure PowerShell samples for Event Grid](#)
- [Azure Resource Manager templates for Event Grid](#)

For information about getting your current event grid subscriptions, see [Query Event Grid subscriptions](#).

## Event subscription expiration

The event subscription is automatically expired after that date. 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. For example, when creating an event subscription to test a scenario, you might want to set an expiration.

For an example of setting an expiration, see [Subscribe with advanced filters](#).

## 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.

For information about implementing any of the supported Event Grid handlers, see [Event handlers in Azure Event Grid](#).

## Security

Event Grid provides security for subscribing to topics, and publishing topics. When subscribing, you must have adequate permissions on the resource or event grid topic. When publishing, you must have a SAS token or key authentication for the topic. For more information, see [Event Grid security and authentication](#).

## Event delivery

If Event Grid can't confirm that an event has been received by the subscriber's endpoint, it redelivers the event. For more information, see [Event Grid message delivery and retry](#).

## Batching

When using a custom topic, events must always be published in an array. This can be a batch of one for low-

throughput scenarios, however, for high volume use cases, it's recommended that you batch several events together per publish to achieve higher efficiency. Batches can be up to 1 MB and the maximum size of an event is 1 MB.

## Next steps

- For an introduction to Event Grid, see [About Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Custom topics in Azure Event Grid

5/11/2021 • 2 minutes to read • [Edit Online](#)

An 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.

**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.

When designing your application, you have flexibility when deciding how many topics to create. For large solutions, create a **custom topic for each category of related events**. For example, consider an application that sends events related to modifying user accounts and processing orders. It's unlikely any event handler wants both categories of events. Create two custom topics and let event handlers subscribe to the one that interests them. For small solutions, you might prefer to send all events to a single topic. Event subscribers can filter for the event types they want.

## Event schema

For a detailed overview of event schema, see [Azure Event Grid event schema](#). For custom topics, the event publisher determines the **data** object. The top-level data should have the same fields as standard resource-defined events.

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

The following sections provide links to tutorials to create custom topics using Azure portal, CLI, PowerShell, and Azure Resource Manager (ARM) templates.

## Azure portal tutorials

TITLE	DESCRIPTION
<a href="#">Quickstart: create and route custom events with the Azure portal</a>	Shows how to use the portal to send custom events.
<a href="#">Quickstart: route custom events to Azure Queue storage</a>	Describes how to send custom events to a Queue storage.
<a href="#">How to: post to custom topic</a>	Shows how to post an event to a custom topic.

## Azure CLI tutorials

TITLE	DESCRIPTION
<a href="#">Quickstart: create and route custom events with Azure CLI</a>	Shows how to use Azure CLI to send custom events.
<a href="#">Azure CLI: create Event Grid custom topic</a>	Sample script that creates a custom topic. The script retrieves the endpoint and a key.
<a href="#">Azure CLI: subscribe to events for a custom topic</a>	Sample script that creates a subscription for a custom topic. It sends events to a WebHook.

## Azure PowerShell tutorials

TITLE	DESCRIPTION
<a href="#">Quickstart: create and route custom events with Azure PowerShell</a>	Shows how to use Azure PowerShell to send custom events.
<a href="#">PowerShell: create Event Grid custom topic</a>	Sample script that creates a custom topic. The script retrieves the endpoint and a key.
<a href="#">PowerShell: subscribe to events for a custom topic</a>	Sample script that creates a subscription for a custom topic. It sends events to a WebHook.

## ARM template tutorials

TITLE	DESCRIPTION
<a href="#">Resource Manager template: custom topic and WebHook endpoint</a>	A Resource Manager template that creates a custom topic and subscription for that custom topic. It sends events to a WebHook.
<a href="#">Resource Manager template: custom topic and Event Hubs endpoint</a>	A Resource Manager template that creates a subscription for a custom topic. It sends events to an Azure Event Hubs.

## Next steps

See the following articles:

- [System topics](#)
- [Domains](#)

# System topics in Azure Event Grid

3/29/2021 • 3 minutes to read • [Edit Online](#)

A system topic in Event Grid represents one or more events published by Azure services such as Azure Storage and Azure Event Hubs. For example, a system topic may represent **all blob events** or only **blob created** and **blob deleted** events published for a **specific storage account**. In this example, when a blob is uploaded to the storage account, the Azure Storage service publishes a **blob created** event to the system topic in Event Grid, which then forwards the event to topic's **subscribers** that receive and process the event.

## NOTE

Only Azure services can publish events to system topics. Therefore, you don't get an endpoint or access keys that you can use to publish events like you do for custom topics or domains.

## Azure services that support system topics

Here is the current list of Azure services that support creation of system topics on them.

- [Azure App Configuration](#)
- [Azure App Service](#)
- [Azure Blob Storage](#)
- [Azure Communication Services](#)
- [Azure Container Registry](#)
- [Azure Event Hubs](#)
- [Azure IoT Hub](#)
- [Azure Key Vault](#)
- [Azure Machine Learning](#)
- [Azure Maps](#)
- [Azure Media Services](#)
- [Azure Policy](#)
- [Azure resource groups](#)
- [Azure Service Bus](#)
- [Azure SignalR](#)
- [Azure subscriptions](#)
- [Azure Cache for Redis](#)

## System topics as Azure resources

In the past, a system topic was implicit and wasn't exposed for simplicity. System topics are now visible as Azure resources and provide the following capabilities:

- [View system topics in the Azure portal](#)
- Export Resource Manager templates for system topics and event subscriptions in the Azure portal
- [Set up diagnostic logs for system topics](#)
- Set up alerts on publish and delivery failures

## Lifecycle of system topics

You can create a system topic in two ways:

- Create an [event subscription on an Azure resource as an extension resource](#), which automatically creates a system topic with the name in the format: <Azure resource name>-<GUID>. The system topic created in this way is automatically deleted when the last event subscription for the topic is deleted.
- Create a system topic for an Azure resource, and then create an event subscription for that system topic. When you use this method, you can specify a name for the system topic. The system topic isn't deleted automatically when the last event subscription is deleted. You need to manually delete it.

When you use the Azure portal, you are always using this method. When you create an event subscription using the [Events page of an Azure resource](#), the system topic is created first and then the subscription for the topic is created. You can explicitly create a system topic first by using the [Event Grid System Topics page](#) and then create a subscription for that topic.

When you use [CLI](#), [REST](#), or [Azure Resource Manager template](#), you can choose either of the above methods. We recommend that you create a system topic first and then create a subscription on the topic, as this is the latest way of creating system topics.

The system topic creation fails if you have set up Azure policies in such a way that the Event Grid service can't create it. For example, you may have a policy that allows creation of only certain types of resources (for example: Azure Storage, Azure Event Hubs, etc.) in the subscription.

## Location and resource group for a system topic

For Azure event sources that are in a specific region/location, system topic is created in the same location as the Azure event source. For example, if you create an event subscription for an Azure blob storage in East US, the system topic is created in East US. For global Azure event sources such as Azure subscriptions, resource groups, or Azure Maps, Event Grid creates the system topic in **global** location.

In general, system topic is created in the same resource group that the Azure event source is in. For event subscriptions created at Azure subscription scope, system topic is created in the **Default-EventGrid** resource group in the **West US 2** region. If the resource group doesn't exist, Azure Event Grid creates it before creating the system topic.

## Next steps

See the following articles:

- [Create, view, and manage system topics by using Azure portal](#).
- [Create, view, and manage Event Grid system topics by using Azure CLI](#)
- [Create Event Grid system topics by using Azure Resource Manager templates](#)

# Azure App Configuration as an Event Grid source

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article provides the properties and schema for Azure App Configuration events. For an introduction to event schemas, see [Azure Event Grid event schema](#). It also gives you a list of quick starts and tutorials to use Azure App Configuration as an event source.

## Available event types

Azure App Configuration emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.AppConfiguration.KeyValueModified	Raised when a key-value is created or replaced.
Microsoft.AppConfiguration.KeyValueDeleted	Raised when a key-value is deleted.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of a key-value modified event:

```
[{  
    "id": "84e17ea4-66db-4b54-8050-df8f7763f87b",  
    "topic": "/subscriptions/00000000-0000-0000-0000-  
0000000000/resourceGroups/testrg/providers/microsoft.appconfiguration/configurationstores/contoso",  
    "subject": "https://contoso.azureconfig.io/kv/Foo?label=FizzBuzz",  
    "data": {  
        "key": "Foo",  
        "label": "FizzBuzz",  
        "etag": "FnUEExLaj2moIi4tJX9AXn9sakm0"  
    },  
    "eventType": "Microsoft.AppConfiguration.KeyValueModified",  
    "eventTime": "2019-05-31T20:05:03Z",  
    "dataVersion": "1",  
    "metadataVersion": "1"  
}]
```

The schema for a key-value deleted event is similar:

```
[{
  "id": "84e17ea4-66db-4b54-8050-df8f7763f87b",
  "topic": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/testrg/providers/microsoft.appconfiguration/configurationstores/contoso",
  "subject": "https://contoso.azureedge.net/kv/Foo?label=FizzBuzz",
  "data": {
    "key": "Foo",
    "label": "FizzBuzz",
    "etag": "FnUExLaj2moIi4tJX9Axn9sakm0"
  },
  "eventType": "Microsoft.AppConfiguration.KeyValueDeleted",
  "eventTime": "2019-05-31T20:05:03Z",
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	App Configuration event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
----------	------	-------------

PROPERTY	TYPE	DESCRIPTION
<code>key</code>	string	The key of the key-value that was modified or deleted.
<code>label</code>	string	The label, if any, of the key-value that was modified or deleted.
<code>etag</code>	string	For <code>KeyValueModified</code> the etag of the new key-value. For <code>KeyValueDeleted</code> the etag of the key-value that was deleted.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">React to Azure App Configuration events by using Event Grid</a>	Overview of integrating Azure App Configuration with Event Grid.
<a href="#">Use Event Grid for data change notifications</a>	Learn how to use Azure App Configuration event subscriptions to send key-value modification events to a web endpoint.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).
- For an introduction to working with Azure App Configuration events, see [Use Event Grid for data change notifications](#).

# Azure App Service as an Event Grid source

3/7/2021 • 8 minutes to read • [Edit Online](#)

This article provides the properties and schema for Azure App Service events. For an introduction to event schemas, see [Azure Event Grid event schema](#). It also gives you a list of quick starts and tutorials to use Azure App Service as an event source.

## Available event types

Azure App Service emits the following event types

EVENT TYPE	DESCRIPTION
Microsoft.Web/sites.BackupOperationStarted	Triggered when a backup has started
Microsoft.Web/sites.BackupOperationCompleted	Triggered when a backup has completed
Microsoft.Web/sites.BackupOperationFailed	Triggered when a backup has failed
Microsoft.Web/sites.RestoreOperationStarted	Triggered when a restoration from a backup has started
Microsoft.Web/sites.RestoreOperationCompleted	Triggered when a restoration from a backup has completed
Microsoft.Web/sites.RestoreOperationFailed	Triggered when a restoration from a backup has failed
Microsoft.Web/sites.SlotSwapStarted	Triggered when a slot swap has started
Microsoft.Web/sites.SlotSwapCompleted	Triggered when a slot swap has completed
Microsoft.Web/sites.SlotSwapFailed	Triggered when a slot swap has failed
Microsoft.Web/sites.SlotSwapWithPreviewStarted	Triggered when a slot swap with preview has started
Microsoft.Web/sites.SlotSwapWithPreviewCancelled	Triggered when a slot swap with preview has been canceled
Microsoft.Web/sites.AppUpdated.Restarted	Triggered when a site has been restarted
Microsoft.Web/sites.AppUpdated.Stopped	Triggered when a site has been stopped
Microsoft.Web/sites.AppUpdated.ChangedAppSettings	Triggered when a site's app settings have changed
Microsoft.Web/serverfarms.AppServicePlanUpdated	Triggered when an App Service Plan is updated

## Properties common to all events

- [Event Grid event schema](#)
- [Cloud event schema](#)

When an event is triggered, the Event Grid service sends data about that event to subscribing endpoint. This

This section contains an example of what that data would look like for each event. Each event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Blob storage event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

## Example events

### **BackupOperationStarted, BackupOperationCompleted, BackupOperationFailed**

- [Event Grid event schema](#)
- [Cloud event schema](#)

```
{
  "id": "7c5d6de5-eb70-4de2-b788-c52a544e68b8",
  "topic": "/subscriptions/<id>/resourceGroups/<rg>/providers/Microsoft.Web/sites/<site-name>",
  "subject": "/Microsoft.Web/sites/<site-name>",
  "eventType": "Microsoft.Web.BackupOperationStarted",
  "eventTime": "2020-01-28T18:26:51.7194887Z",
  "data": {
    "appEventTypeDetail": {
      "action": "Started"
    },
    "name": "<site-name>",
    "clientRequestId": "None",
    "correlationRequestId": "None",
    "requestId": "292f499d-04ee-4066-994d-c2df57b99198",
    "address": "None",
    "verb": "None"
  },
  "dataVersion": "1",
  "metaDataVersion": "1"
}
```

The data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>appEventTypeDetail</code>	object	Detail of action on the app
<code>action</code>	string	Type of action of the operation
<code>name</code>	string	name of the web site that had this event
<code>clientRequestId</code>	string	The client request ID generated by the app service for the site API operation that triggered this event
<code>correlationRequestId</code>	string	The correlation request ID generated by the app service for the site API operation that triggered this event
<code>requestId</code>	string	The request ID generated by the app service for the site API operation that triggered this event
<code>address</code>	string	HTTP request URL of this operation
<code>verb</code>	string	HTTP verb of this operation

#### **RestoreOperationStarted, RestoreOperationCompleted, RestoreOperationFailed**

- [Event Grid event schema](#)
- [Cloud event schema](#)

```
{  
  "id": "7c5d6de5-eb70-4de2-b788-c52a544e68b8",  
  "topic": "/subscriptions/<id>/resourceGroups/<rg>/providers/Microsoft.Web/sites/<site-name>",  
  "subject": "/Microsoft.Web/sites/<site-name>",  
  "eventType": "Microsoft.Web.RestoreOperationStarted",  
  "eventTime": "2020-01-28T18:26:51.7194887Z",  
  "data": {  
    "appEventTypeDetail": {  
      "action": "Started"  
    },  
    "name": "<site-name>",  
    "clientRequestId": "None",  
    "correlationRequestId": "None",  
    "requestId": "292f499d-04ee-4066-994d-c2df57b99198",  
    "address": "None",  
    "verb": "POST"  
  },  
  "dataVersion": "1",  
  "metaDataVersion": "1"  
}
```

The data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>appEventTypeDetail</code>	object	Detail of action on the app
<code>action</code>	string	Type of action of the operation
<code>name</code>	string	name of the web site that had this event
<code>clientRequestId</code>	string	The client request ID generated by the app service for the site API operation that triggered this event
<code>correlationRequestId</code>	string	The correlation request ID generated by the app service for the site API operation that triggered this event
<code>requestId</code>	string	The request ID generated by the app service for the site API operation that triggered this event
<code>address</code>	string	HTTP request URL of this operation
<code>verb</code>	string	HTTP verb of this operation

### SlotSwapStarted, SlotSwapCompleted, SlotSwapFailed

- [Event Grid event schema](#)
- [Cloud event schema](#)

```
{
  "id": "7c5d6de5-eb70-4de2-b788-c52a544e68b8",
  "topic": "/subscriptions/<id>/resourceGroups/<rg>/providers/Microsoft.Web/sites/<site-name>",
  "subject": "/Microsoft.Web/sites/<site-name>",
  "eventType": "Microsoft.Web.SlotSwapStarted",
  "eventTime": "2020-01-28T18:26:51.7194887Z",
  "data": {
    "appEventTypeDetail": null,
    "name": "<site-name>",
    "clientRequestId": "922f4841-20d9-4dd6-8c5b-23f0d85e5592",
    "correlationRequestId": "9ac46505-2b8a-4e06-834c-05ffbe2e8c3a",
    "requestId": "765117aa-eaf8-4bd2-a644-1dbf69c7b0fd",
    "address": "/websystems/WebSites/web/subscriptions/<id>/webspaces/<webspace>/sites/<site-name>/slots?Command=SWAP&targetSlot=production",
    "verb": "POST",
    "sourceSlot": "staging",
    "targetSlot": "production"
  },
  "dataVersion": "1",
  "metaDataVersion": "1"
}
```

The data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>appEventTypeDetail</code>	object	Detail of action on the app

PROPERTY	TYPE	DESCRIPTION
<code>action</code>	string	Type of action of the operation
<code>name</code>	string	name of the web site that had this event
<code>clientRequestId</code>	string	The client request ID generated by the app service for the site API operation that triggered this event
<code>correlationRequestId</code>	string	The correlation request ID generated by the app service for the site API operation that triggered this event
<code>requestId</code>	string	The request ID generated by the app service for the site API operation that triggered this event
<code>address</code>	string	HTTP request URL of this operation
<code>verb</code>	string	HTTP verb of this operation
<code>sourceSlot</code>	string	The source slot of the swap

### SlotSwapWithPreviewStarted, SlotSwapWithPreviewCancelled

- [Event Grid event schema](#)
- [Cloud event schema](#)

```
{
  "id": "7c5d6de5-eb70-4de2-b788-c52a544e68b8",
  "topic": "/subscriptions/<id>/resourceGroups/<rg>/providers/Microsoft.Web/sites/<site-name>",
  "subject": "/Microsoft.Web/sites/<site-name>",
  "eventType": "Microsoft.Web.SlotSwapWithPreviewStarted",
  "eventTime": "2020-01-28T18:26:51.7194887Z",
  "data": {
    "appEventTypeDetail": null,
    "name": "<site-name>",
    "clientRequestId": "922f4841-20d9-4dd6-8c5b-23f0d85e5592",
    "correlationRequestId": "9ac46505-2b8a-4e06-834c-05ffbe2e8c3a",
    "requestId": "765117aa-eaf8-4bd2-a644-1dbf69c7b0fd",
    "address": "/websystems/WebSites/web/subscriptions/<id>/webspaces/<webspace>/sites/<site-name>/slots?Command=SWAP&targetSlot=production",
    "verb": "POST",
    "sourceSlot": "staging",
    "targetSlot": "production"
  },
  "dataVersion": "1",
  "metaDataVersion": "1"
}
```

The data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>appEventTypeDetail</code>	object	Detail of action on the app

PROPERTY	TYPE	DESCRIPTION
<code>action</code>	string	Type of action of the operation
<code>name</code>	string	name of the web site that had this event
<code>clientRequestId</code>	string	The client request ID generated by the app service for the site API operation that triggered this event
<code>correlationRequestId</code>	string	The correlation request ID generated by the app service for the site API operation that triggered this event
<code>requestId</code>	string	The request ID generated by the app service for the site API operation that triggered this event
<code>address</code>	string	HTTP request URL of this operation
<code>verb</code>	string	HTTP verb of this operation

### AppUpdated.Restarted, AppUpdated.Stopped, AppUpdated.ChangedAppSettings

- [Event Grid event schema](#)
- [Cloud event schema](#)

```
{
  "id": "b74ea56b-2a3f-4de5-a5d7-38e60c81cf23",
  "topic": "/subscriptions/<id>/resourceGroups/<group>/providers/Microsoft.Web/sites/<site-name>",
  "subject": "/Microsoft.Web/sites/<site-name>",
  "eventType": "Microsoft.Web.AppUpdated",
  "eventTime": "2020-01-28T18:22:30.2760952Z",
  "data": {
    "appEventTypeDetail": {
      "action": "Stopped"
    },
    "name": "<site-name>",
    "clientRequestId": "64a5e0aa-7cee-4ff1-9093-b9197b820014",
    "correlationRequestId": "25bb36a5-8f6c-4f04-b615-e9a0ee045756",
    "requestId": "f2e8eb3f-b190-42de-b99e-6acefe587374",
    "address": "/websystems/WebSites/web/subscriptions/<id>/webspaces/<webspace>/sites/<site-name>/stop",
    "verb": "POST"
  },
  "dataVersion": "1",
  "metaDataVersion": "1"
}
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>appEventTypeDetail</code>	object	Detail of action on the app
<code>action</code>	string	Type of action of the operation

PROPERTY	TYPE	DESCRIPTION
<code>name</code>	string	name of the web site that had this event
<code>clientRequestId</code>	string	The client request ID generated by the app service for the site API operation that triggered this event
<code>correlationRequestId</code>	string	The correlation request ID generated by the app service for the site API operation that triggered this event
<code>requestId</code>	string	The request ID generated by the app service for the site API operation that triggered this event
<code>address</code>	string	HTTP request URL of this operation
<code>verb</code>	string	HTTP verb of this operation

## Serverfarms.AppServicePlanUpdated

- [Event Grid event schema](#)
- [Cloud event schema](#)

```
{
  "id": "56501672-9150-40e1-893a-18420c7fdbf7",
  "topic": "/subscriptions/<id>/resourceGroups/<rg>/providers/Microsoft.Web/serverfarms/<serverfarm-name>",
  "subject": "/Microsoft.Web/serverfarms/<plan-name>",
  "eventType": "Microsoft.Web.AppServicePlanUpdated",
  "eventTime": "2020-01-28T18:22:23.5516004Z",
  "data": {
    "serverFarmEventTypeDetail": {
      "stampKind": "Public",
      "action": "Updated",
      "status": "Started"
    },
    "serverFarmId": "0",
    "sku": {
      "name": "P1v2",
      "tier": "PremiumV2",
      "size": "P1v2",
      "family": "Pv2",
      "capacity": 1
    },
    "clientRequestId": "8f880321-a991-45c7-b743-6ff63fe4c004",
    "correlationRequestId": "1995c3be-ba7f-4ccf-94af-516df637ec8a",
    "requestId": "b973a8e6-6949-4783-b44c-ac778be831bb",
    "address": "/websystems/WebSites/serverfarms/subscriptions/<id>/webspaces/<webspace-id>/serverfarms/<plan-name>/async",
    "verb": "PUT"
  },
  "dataVersion": "1",
  "metaDataVersion": "1"
}
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>appServicePlanEventTypeDetail</code>	object	Detail of action on the app service plan
<code>stampKind</code>	string	Kind of environment where app service plan is
<code>action</code>	string	Type of action on the app service plan
<code>status</code>	string	Status of the operation on the app service plan
<code>sku</code>	object	sku of the app service plan
<code>name</code>	string	name of the app service plan
<code>Tier</code>	string	tier of the app service plan
<code>Size</code>	string	size of the app service plan
<code>Family</code>	string	family of app service plan
<code>Capacity</code>	string	capacity of app service plan
<code>action</code>	string	Type of action of the operation
<code>name</code>	string	name of the web site that had this event
<code>clientRequestId</code>	string	The client request ID generated by the app service for the site API operation that triggered this event
<code>correlationRequestId</code>	string	The correlation request ID generated by the app service for the site API operation that triggered this event
<code>requestId</code>	string	The request ID generated by the app service for the site API operation that triggered this event
<code>address</code>	string	HTTP request URL of this operation
<code>verb</code>	string	HTTP verb of this operation

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#)

# Azure Blob Storage as an Event Grid source

6/8/2021 • 12 minutes to read • [Edit Online](#)

This article provides the properties and schema for blob storage events. For an introduction to event schemas, see [Azure Event Grid event schema](#). It also gives you a list of quick starts and tutorials to use Azure Blob Storage as an event source.

## NOTE

Only storage accounts of kind **StorageV2** (general purpose v2), **BlockBlobStorage**, and **BlobStorage** support event integration. **Storage** (general purpose v1) does *not* support integration with Event Grid.

## Available event types

### List of events for Blob REST APIs

These events are triggered when a client creates, replaces, or deletes a blob by calling Blob REST APIs.

## NOTE

The `$logs` and `$blobchangefeed` containers aren't integrated with Event Grid, so activity in these containers will not generate events. Also, using the `dfs` endpoint `(abfss://URI)` for non-hierarchical namespace enabled accounts will not generate events, but the blob endpoint `(wasb:// URI)` will generate events.

EVENT NAME	DESCRIPTION
<b>Microsoft.Storage.BlobCreated</b>	Triggered when a blob is created or replaced. Specifically, this event is triggered when clients use the <code>PutBlob</code> , <code>PutBlockList</code> , or <code>CopyBlob</code> operations that are available in the Blob REST API and when the Block Blob is completely committed. If clients use the <code>copyBlob</code> operation on accounts that have the <b>hierarchical namespace</b> feature enabled on them, the <code>CopyBlob</code> operation works a little differently. In that case, the <b>Microsoft.Storage.BlobCreated</b> event is triggered when the <code>CopyBlob</code> operation is <b>initiated</b> and not when the Block Blob is completely committed.
<b>Microsoft.Storage.BlobDeleted</b>	Triggered when a blob is deleted. Specifically, this event is triggered when clients call the <code>DeleteBlob</code> operation that is available in the Blob REST API.
<b>Microsoft.Storage.BlobTierChanged</b>	Triggered when the blob access tier is changed. Specifically, when clients call the <code>Set Blob Tier</code> operation that is available in the Blob REST API, this event is triggered after the tier change completes.

### List of the events for Azure Data Lake Storage Gen 2 REST APIs

These events are triggered if you enable a hierarchical namespace on the storage account, and clients use Azure Data Lake Storage Gen2 REST APIs. For more information about Azure Data Lake Storage Gen2, see [Introduction](#)

to Azure Data Lake Storage Gen2.

EVENT NAME	DESCRIPTION
<b>Microsoft.Storage.BlobCreated</b>	Triggered when a blob is created or replaced. Specifically, this event is triggered when clients use the <code>CreateFile</code> and <code>FlushWithClose</code> operations that are available in the Azure Data Lake Storage Gen2 REST API.
<b>Microsoft.Storage.BlobDeleted</b>	Triggered when a blob is deleted. Specifically, This event is also triggered when clients call the <code>DeleteFile</code> operation that is available in the Azure Data Lake Storage Gen2 REST API.
<b>Microsoft.Storage.BlobRenamed</b>	Triggered when a blob is renamed. Specifically, this event is triggered when clients use the <code>RenameFile</code> operation that is available in the Azure Data Lake Storage Gen2 REST API.
<b>Microsoft.Storage.DirectoryCreated</b>	Triggered when a directory is created. Specifically, this event is triggered when clients use the <code>CreateDirectory</code> operation that is available in the Azure Data Lake Storage Gen2 REST API.
<b>Microsoft.Storage.DirectoryRenamed</b>	Triggered when a directory is renamed. Specifically, this event is triggered when clients use the <code>RenameDirectory</code> operation that is available in the Azure Data Lake Storage Gen2 REST API.
<b>Microsoft.Storage.DirectoryDeleted</b>	Triggered when a directory is deleted. Specifically, this event is triggered when clients use the <code>DeleteDirectory</code> operation that is available in the Azure Data Lake Storage Gen2 REST API.

#### NOTE

For Azure Data Lake Storage Gen2, if you want to ensure that the **Microsoft.Storage.BlobCreated** event is triggered only when a Block Blob is completely committed, filter the event for the `FlushWithClose` REST API call. This API call triggers the **Microsoft.Storage.BlobCreated** event only after data is fully committed to a Block Blob. To learn how to create a filter, see [Filter events for Event Grid](#).

## Example event

When an event is triggered, the Event Grid service sends data about that event to subscribing endpoint. This section contains an example of what that data would look like for each blob storage event.

- [Event Grid event schema](#)
- [Cloud event schema](#)

### Microsoft.Storage.BlobCreated event

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/test-container/blobs/new-file.txt",
  "eventType": "Microsoft.Storage.BlobCreated",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "PutBlockList",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "eTag": "\"0x8D4BCC2E4835CD0\"",
    "contentType": "text/plain",
    "contentLength": 524288,
    "blobType": "BlockBlob",
    "url": "https://my-storage-account.blob.core.windows.net/testcontainer/new-file.txt",
    "sequencer": "000000000000442000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

### Microsoft.Storage.BlobCreated event (Data Lake Storage Gen2)

If the blob storage account has a hierarchical namespace, the data looks similar to the previous example with an exception of these changes:

- The `dataVersion` key is set to a value of `2`.
- The `data.api` key is set to the string `CreateFile` or `FlushWithClose`.
- The `contentOffset` key is included in the data set.

#### NOTE

If applications use the `PutBlockList` operation to upload a new blob to the account, the data won't contain these changes.

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/my-file-system/blobs/new-file.txt",
  "eventType": "Microsoft.Storage.BlobCreated",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "CreateFile",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "eTag": "\\"0x8D4BCC2E4835CD0\\",
    "contentType": "text/plain",
    "contentLength": 0,
    "contentOffset": 0,
    "blobType": "BlockBlob",
    "url": "https://my-storage-account.dfs.core.windows.net/my-file-system/new-file.txt",
    "sequencer": "0000000000004420000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "2",
  "metadataVersion": "1"
}]
```

## Microsoft.Storage.BlobDeleted event

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/testcontainer/blobs/file-to-delete.txt",
  "eventType": "Microsoft.Storage.BlobDeleted",
  "eventTime": "2017-11-07T20:09:22.5674003Z",
  "id": "4c2359fe-001e-00ba-0e04-58586806d298",
  "data": {
    "api": "DeleteBlob",
    "requestId": "4c2359fe-001e-00ba-0e04-585868000000",
    "contentType": "text/plain",
    "blobType": "BlockBlob",
    "url": "https://my-storage-account.blob.core.windows.net/testcontainer/file-to-delete.txt",
    "sequencer": "000000000000281000000000002F5CA",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Microsoft.Storage.BlobDeleted event (Data Lake Storage Gen2)

If the blob storage account has a hierarchical namespace, the data looks similar to the previous example with an exception of these changes:

- The `dataVersion` key is set to a value of `2`.
- The `data.api` key is set to the string `DeleteFile`.
- The `url` key contains the path `dfs.core.windows.net`.

## NOTE

If applications use the `DeleteBlob` operation to delete a blob from the account, the data won't contain these changes.

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/my-file-system/blobs/file-to-delete.txt",
  "eventType": "Microsoft.Storage.BlobDeleted",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "DeleteFile",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "contentType": "text/plain",
    "blobType": "BlockBlob",
    "url": "https://my-storage-account.dfs.core.windows.net/my-file-system/file-to-delete.txt",
    "sequencer": "000000000000442000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "2",
  "metadataVersion": "1"
}]
```

## Microsoft.Storage.BlobTierChanged event

```
{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/testcontainer/blobs/Auto.jpg",
  "eventType": "Microsoft.Storage.BlobTierChanged",
  "id": "0fdefc06-b01e-0034-39f6-4016610696f6",
  "data": {
    "api": "SetBlobTier",
    "clientRequestId": "68be434c-1a0d-432f-9cd7-1db90bff83d7",
    "requestId": "0fdefc06-b01e-0034-39f6-401661000000",
    "contentType": "image/jpeg",
    "contentLength": 105891,
    "blobType": "BlockBlob",
    "url": "https://my-storage-account.blob.core.windows.net/testcontainer/Auto.jpg",
    "sequencer": "00000000000000000000000000000089A400000000018d6ea",
    "storageDiagnostics": {
      "batchId": "3418f7a9-7006-0014-00f6-406dc6000000"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1",
  "eventTime": "2021-05-04T15:00:00.8350154Z"
}
```

## Microsoft.Storage.BlobRenamed event

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/my-file-system/blobs/my-renamed-file.txt",
  "eventType": "Microsoft.Storage.BlobRenamed",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "RenameFile",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "destinationUrl": "https://my-storage-account.dfs.core.windows.net/my-file-system/my-renamed-file.txt",
    "sourceUrl": "https://my-storage-account.dfs.core.windows.net/my-file-system/my-original-file.txt",
    "sequencer": "000000000000442000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

## Microsoft.Storage.DirectoryCreated event

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/my-file-system/blobs/my-new-directory",
  "eventType": "Microsoft.Storage.DirectoryCreated",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "CreateDirectory",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "url": "https://my-storage-account.dfs.core.windows.net/my-file-system/my-new-directory",
    "sequencer": "000000000000442000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

## Microsoft.Storage.DirectoryRenamed event

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/my-file-system/blobs/my-renamed-directory",
  "eventType": "Microsoft.Storage.DirectoryRenamed",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "RenameDirectory",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "destinationUrl": "https://my-storage-account.dfs.core.windows.net/my-file-system/my-renamed-directory",
    "sourceUrl": "https://my-storage-account.dfs.core.windows.net/my-file-system/my-original-directory",
    "sequencer": "0000000000004420000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

## Microsoft.Storage.DirectoryDeleted event

```
[{
  "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/my-storage-account",
  "subject": "/blobServices/default/containers/my-file-system/blobs/directory-to-delete",
  "eventType": "Microsoft.Storage.DirectoryDeleted",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "api": "DeleteDirectory",
    "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
    "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
    "url": "https://my-storage-account.dfs.core.windows.net/my-file-system/directory-to-delete",
    "recursive": "true",
    "sequencer": "0000000000004420000000000028963",
    "storageDiagnostics": {
      "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
    }
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
topic	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.

PROPERTY	TYPE	DESCRIPTION
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Blob storage event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>api</code>	string	The operation that triggered the event.
<code>clientRequestId</code>	string	a client-provided request ID for the storage API operation. This ID can be used to correlate to Azure Storage diagnostic logs using the "client-request-id" field in the logs, and can be provided in client requests using the "x-ms-client-request-id" header. See <a href="#">Log Format</a> .
<code>requestId</code>	string	Service-generated request ID for the storage API operation. Can be used to correlate to Azure Storage diagnostic logs using the "request-id-header" field in the logs and is returned from initiating API call in the 'x-ms-request-id' header. See <a href="#">Log Format</a> .
<code>eTag</code>	string	The value that you can use to run operations conditionally.
<code>contentType</code>	string	The content type specified for the blob.
<code>contentLength</code>	integer	The size of the blob in bytes.
<code>blobType</code>	string	The type of blob. Valid values are either "BlockBlob" or "PageBlob".

PROPERTY	TYPE	DESCRIPTION
<code>contentOffset</code>	number	The offset in bytes of a write operation taken at the point where the event-triggering application completed writing to the file. Appears only for events triggered on blob storage accounts that have a hierarchical namespace.
<code>destinationUrl</code>	string	The url of the file that will exist after the operation completes. For example, if a file is renamed, the <code>destinationUrl</code> property contains the url of the new file name. Appears only for events triggered on blob storage accounts that have a hierarchical namespace.
<code>sourceUrl</code>	string	The url of the file that exists before the operation is done. For example, if a file is renamed, the <code>sourceUrl</code> contains the url of the original file name before the rename operation. Appears only for events triggered on blob storage accounts that have a hierarchical namespace.
<code>url</code>	string	<p>The path to the blob.            If the client uses a Blob REST API, then the url has this structure:</p> <pre>&lt;storage-account-name&gt;.blob.core.windows.net\&lt;container-name&gt;\&lt;file-name&gt;</pre> <p>If the client uses a Data Lake Storage REST API, then the url has this structure:</p> <pre>&lt;storage-account-name&gt;.dfs.core.windows.net/&lt;file-system-name&gt;/&lt;file-name&gt;</pre>
<code>recursive</code>	string	<code>True</code> to run the operation on all child directories; otherwise <code>False</code> . Appears only for events triggered on blob storage accounts that have a hierarchical namespace.
<code>sequencer</code>	string	An opaque string value representing the logical sequence of events for any particular blob name. Users can use standard string comparison to understand the relative sequence of two events on the same blob name.
<code>storageDiagnostics</code>	object	Diagnostic data occasionally included by the Azure Storage service. When present, should be ignored by event consumers.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Quickstart: route Blob storage events to a custom web endpoint with Azure CLI</a>	Shows how to use Azure CLI to send blob storage events to a WebHook.
<a href="#">Quickstart: route Blob storage events to a custom web endpoint with PowerShell</a>	Shows how to use Azure PowerShell to send blob storage events to a WebHook.
<a href="#">Quickstart: create and route Blob storage events with the Azure portal</a>	Shows how to use the portal to send blob storage events to a WebHook.
<a href="#">Azure CLI: subscribe to events for a Blob storage account</a>	Sample script that subscribes to event for a Blob storage account. It sends the event to a WebHook.
<a href="#">PowerShell: subscribe to events for a Blob storage account</a>	Sample script that subscribes to event for a Blob storage account. It sends the event to a WebHook.
<a href="#">Resource Manager template: Create Blob storage and subscription</a>	Deploys an Azure Blob storage account and subscribes to events for that storage account. It sends events to a WebHook.
<a href="#">Overview: reacting to Blob storage events</a>	Overview of integrating Blob storage with Event Grid.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).
- For an introduction to working with blob storage events, see [Route Blob storage events - Azure CLI](#).

# Event Handling in Azure Communication Services

3/19/2021 • 9 minutes to read • [Edit Online](#)

Azure Communication Services integrates with [Azure Event Grid](#) to deliver real-time event notifications in a reliable, scalable and secure manner. The purpose of this article is to help you configure your applications to listen to Communication Services events. For example, you may want to update a database, create a work item and deliver a push notification whenever an SMS message is received by a phone number associated with your Communication Services resource.

Azure Event Grid is a fully managed event routing service, which uses a publish-subscribe model. Event Grid has built-in support for Azure services like [Azure Functions](#) and [Azure Logic Apps](#). It can deliver event alerts to non-Azure services using webhooks. For a complete list of the event handlers that Event Grid supports, see [An introduction to Azure Event Grid](#).

## NOTE

To learn more about how data residency relates to event handling, visit the [Data Residency conceptual documentation](#)

## Events types

Event grid uses [event subscriptions](#) to route event messages to subscribers.

Azure Communication Services emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.Communication.SMSReceived	Published when an SMS is received by a phone number associated with the Communication Service.
Microsoft.Communication.SMSDeliveryReportReceived	Published when a delivery report is received for an SMS sent by the Communication Service.
Microsoft.Communication.ChatMessageReceived	Published when a message is received for a user in a chat thread that she is member of.
Microsoft.Communication.ChatMessageEdited	Published when a message is edited in a chat thread that the user is member of.
Microsoft.Communication.ChatMessageDeleted	Published when a message is deleted in a chat thread that the user is member of.
Microsoft.Communication.ChatThreadCreatedWithUser	Published when the user is added as member at the time of creation of a chat thread.
Microsoft.Communication.ChatThreadWithUserDeleted	Published when a chat thread is deleted which the user is member of.
Microsoft.Communication.ChatThreadPropertiesUpdatedPer User	Published when a chat thread's properties are updated that the user is member of.

EVENT TYPE	DESCRIPTION
Microsoft.Communication.ChatMemberAddedToThreadWithUser	Published when the user is added as member to a chat thread.
Microsoft.Communication.ChatMemberRemovedFromThreadWithUser	Published when the user is removed from a chat thread.
Microsoft.Communication.ChatParticipantAddedToThreadWithUser	Published for a user when a new participant is added to a chat thread, that the user is part of.
Microsoft.Communication.ChatParticipantRemovedFromThreadWithUser	Published for a user when a participant is removed from a chat thread, that the user is part of.
Microsoft.Communication.ChatThreadCreated	Published when a chat thread is created
Microsoft.Communication.ChatThreadDeleted	Published when a chat thread is deleted
Microsoft.Communication.ChatThreadParticipantAdded	Published when a new participant is added to a chat thread
Microsoft.Communication.ChatThreadParticipantRemoved	Published when a new participant is removed from a chat thread.
Microsoft.Communication.ChatMessageReceivedInThread	Published when a message is received in a chat thread
Microsoft.Communication.ChatThreadPropertiesUpdated	Published when a chat thread's properties like topic are updated.
Microsoft.Communication.ChatMessageEditedInThread	Published when a message is edited in a chat thread
Microsoft.Communication.ChatMessageDeletedInThread	Published when a message is deleted in a chat thread

You can use the Azure portal or Azure CLI to subscribe to events emitted by your Communication Services resource. Get started with handling events by looking at [How to handle SMS Events in Communication Services](#)

## Event subjects

The `subject` field of all Communication Services events identifies the user, phone number or entity that is targeted by the event. Common prefixes are used to allow simple [Event Grid Filtering](#).

SUBJECT PREFIX	COMMUNICATION SERVICE ENTITY
<code>phonenumbers/</code>	PSTN phone number
<code>user/</code>	Communication Services User
<code>thread/</code>	Chat thread.

The following example shows a filter for all SMS messages and delivery reports sent to all 555 area code phone numbers owned by a Communication Services resource:

```

"filter": {
    "includedEventTypes": [
        "Microsoft.Communication.SMSReceived",
        "Microsoft.Communication.SMSDeliveryReportReceived"
    ],
    "subjectBeginsWith": "phononenumber/1555",
}

```

## Sample event responses

When an event is triggered, the Event Grid service sends data about that event to subscribing endpoints.

This section contains an example of what that data would look like for each event.

### **Microsoft.Communication.SMSDeliveryReportReceived event**

```

[{
    "id": "Outgoing_202009180022138813a09b-0cbf-4304-9b03-1546683bb910",
    "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
    "subject": "/phononenumber/15555555555",
    "data": {
        "MessageId": "Outgoing_202009180022138813a09b-0cbf-4304-9b03-1546683bb910",
        "From": "15555555555",
        "To": "+15555555555",
        "DeliveryStatus": "Delivered",
        "DeliveryStatusDetails": "No error.",
        "ReceivedTimestamp": "2020-09-18T00:22:20.2855749Z",
        "DeliveryAttempts": [
            {
                "Timestamp": "2020-09-18T00:22:14.9315918Z",
                "SegmentsSucceeded": 1,
                "SegmentsFailed": 0
            }
        ]
    },
    "eventType": "Microsoft.Communication.SMSDeliveryReportReceived",
    "dataVersion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2020-09-18T00:22:20Z"
}]

```

### **Microsoft.Communication.SMSReceived event**

```

[{
    "id": "Incoming_20200918002745d29ebbea-3341-4466-9690-0a03af35228e",
    "topic": "/subscriptions/50ad1522-5c2c-4d9a-a6c8-67c11ecb75b8/resourcegroups/acse2e/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
    "subject": "/phononenumber/15555555555",
    "data": {
        "MessageId": "Incoming_20200918002745d29ebbea-3341-4466-9690-0a03af35228e",
        "From": "15555555555",
        "To": "15555555555",
        "Message": "Great to connect with ACS events ",
        "ReceivedTimestamp": "2020-09-18T00:27:45.32Z"
    },
    "eventType": "Microsoft.Communication.SMSReceived",
    "dataVersion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2020-09-18T00:27:47Z"
}]

```

## **Microsoft.Communication.ChatMessageReceived event**

```
[{
  "id": "02272459-badb-4e2e-b538-4cb8a2f71da6",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/sender/{rawId}/recipient/{rawId}",
  "data": {
    "messageBody": "Welcome to Azure Communication Services",
    "messageId": "1613694358927",
    "senderId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724",
    "senderCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724"
      }
    },
    "senderDisplayName": "Jhon",
    "composeTime": "2021-02-19T00:25:58.927Z",
    "type": "Text",
    "version": 1613694358927,
    "recipientId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d05-83fe-084822000f6d",
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d05-83fe-084822000f6d",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d05-83fe-084822000f6d"
      }
    },
    "transactionId": "oh+LGB2dUUadMcTAdRWQxQ.1.1.1.1827536918.1.7",
    "threadId": "19:6e5d6ca1d75044a49a36a7965ec4a906@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatMessageReceived",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-19T00:25:59.9436666Z"
}
]
```

## **Microsoft.Communication.ChatMessageEdited event**

```
[{
  "id": "93fc1037-b645-4eb0-a0f2-d7bb3ba6e060",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/sender/{rawId}/recipient/{rawId}",
  "data": {
    "editTime": "2021-02-19T00:28:20.784Z",
    "messageBody": "Let's Chat about new communication services.",
    "messageId": "1613694357917",
    "senderId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724",
    "senderCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724"
      }
    },
    "senderDisplayName": "Bob(Admin)",
    "composeTime": "2021-02-19T00:25:57.917Z",
    "type": "Text",
    "version": 1613694500784,
    "recipientId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d60-83fe-084822000f6f",
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d60-83fe-084822000f6f",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d60-83fe-084822000f6f"
      }
    },
    "transactionId": "1mL4XZH2gEecu/alk9t0tw.2.1.2.1.1833042153.1.7",
    "threadId": "19:6e5d6ca1d75044a49a36a7965ec4a906@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatMessageEdited",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-19T00:28:21.7456718Z"
}]
```

## Microsoft.Communication.ChatMessageDeleted event

```
[{
  "id": "23cfcc13-33f2-4ae1-8d23-b5015b05302b",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/sender/{rawId}/recipient/{rawId}",
  "data": {
    "deleteTime": "2021-02-19T00:43:10.14Z",
    "messageId": "1613695388152",
    "senderId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d07-83fe-084822000f6e",
    "senderCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d07-83fe-084822000f6e",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d07-83fe-084822000f6e"
      }
    },
    "senderDisplayName": "Bob(Admin)",
    "composeTime": "2021-02-19T00:43:08.152Z",
    "type": "Text",
    "version": 1613695390361,
    "recipientId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d60-83fe-084822000f6f",
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d60-83fe-084822000f6f",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d60-83fe-084822000f6f"
      }
    },
    "transactionId": "fFs4InlBn00/0WyhfQZVSQ.1.1.2.1.1867776045.1.4",
    "threadId": "19:48899258eec941e7a281e03edc8f4964@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatMessageDeleted",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-19T00:43:10.9982947Z"
}
]
```

## Microsoft.Communication.ChatThreadCreatedWithUser event

```
[{
  "id": "eba02b2d-37bf-420e-8656-3a42ef74c435",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{groupName}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/createdBy/rawId/recipient/rawId",
  "data": {
    "createdBy": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286d-e1fe-0848220013b9",
    "createdByCommunicationIdentifier": {
      "rawId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286d-e1fe-0848220013b9",
      "communicationUser": {
        "id": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286d-e1fe-0848220013b9"
      }
    },
    "properties": {
      "topic": "Chat about new communication services"
    },
    "members": [
      {
        "displayName": "Bob",
        "memberId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286d-e1fe-0848220013b9"
      },
      {
        "displayName": "John",
        "memberId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-289b-07fd-0848220015ea"
      }
    ],
    "participants": [
      {
        "displayName": "Bob",
        "participantCommunicationIdentifier": {
          "rawId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286d-e1fe-0848220013b9",
          "communicationUser": {
            "id": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286d-e1fe-0848220013b9"
          }
        }
      },
      {
        "displayName": "John",
        "participantCommunicationIdentifier": {
          "rawId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-289b-07fd-0848220015ea",
          "communicationUser": {
            "id": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-289b-07fd-0848220015ea"
          }
        }
      }
    ],
    "createTime": "2021-02-18T23:47:26.91Z",
    "version": 1613692046910,
    "recipientId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286e-84f5-08482200181c",
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286e-84f5-08482200181c",
      "communicationUser": {
        "id": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-576c-286e-84f5-08482200181c"
      }
    },
    "transactionId": "zbZt+9h/N0em+XCW2QvyIA.1.1.1.1.1737228330.0.1737490483.1.6",
    "threadId": "19:1d594fb1eeb14566903cbc5decb5bf5b@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatThreadCreatedWithUser",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-18T23:47:34.7437103Z"
}]
}
```

## Microsoft.Communication.ChatThreadWithUserDeleted event

```
[{
  "id": "f5d6750c-c6d7-4da8-bb05-6f3fcac6c7295",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/deletedBy/{rawId}/recipient/{rawId}",
  "data": {
    "deletedBy": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-5772-6473-83fe-084822000e21",
    "deletedByCommunicationIdentifier": {
      "rawId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-5772-6473-83fe-084822000e21",
      "communicationUser": {
        "id": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-5772-6473-83fe-084822000e21"
      }
    },
    "deleteTime": "2021-02-18T23:57:51.5987591Z",
    "createTime": "2021-02-18T23:54:15.683Z",
    "version": 1613692578672,
    "recipientId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-5772-647b-e1fe-084822001416",
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-5772-647b-e1fe-084822001416",
      "communicationUser": {
        "id": "8:acs:3d703c91-9657-4b3f-b19c-ef9d53f99710_00000008-5772-647b-e1fe-084822001416"
      }
    },
    "transactionId": "mrliWVUndEmLwkZbeS5KoA.1.1.2.1.1761607918.1.6",
    "threadId": "19:5870b8f021d74fd786bf5aeb095da291@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatThreadWithUserDeleted",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-18T23:57:52.1597234Z"
}]
```

## Microsoft.Communication.ChatParticipantAddedToThreadWithUser event

```
[{
  "id": "049a5a7f-6cd7-43c1-b352-df9e9e6146d1",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/participantAdded/{rawId}/recipient/{rawId}",
  "data": {
    "time": "2021-02-25T06:37:29.9232485Z",
    "addedByCommunicationIdentifier": {
      "rawId": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8767-1655-373a0d00885d",
      "communicationUser": {
        "id": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8767-1655-373a0d00885d"
      }
    },
    "participantAdded": {
      "displayName": "John Smith",
      "participantCommunicationIdentifier": {
        "rawId": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8785-1655-373a0d00885f",
        "communicationUser": {
          "id": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8785-1655-373a0d00885f"
        }
      }
    },
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8781-1655-373a0d00885e",
      "communicationUser": {
        "id": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8781-1655-373a0d00885e"
      }
    },
    "createTime": "2021-02-25T06:37:17.371Z",
    "version": 1614235049907,
    "transactionId": "q7rr9by6m0CiGiQxKdS01w.1.1.1.1473446055.1.6",
    "threadId": "19:f1400e1c542f4086a606b52ad20cd0bd@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatParticipantAddedToThreadWithUser",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-25T06:37:31.4880091Z"
}]
```

### **Microsoft.Communication.ChatParticipantRemovedFromThreadWithUser event**

```
[{
  "id": "e8a4df24-799d-4c53-94fd-1e05703a4549",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/participantRemoved/{rawId}/recipient/{rawId}",
  "data": {
    "time": "2021-02-25T06:40:20.3564556Z",
    "removedByCommunicationIdentifier": {
      "rawId": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8767-1655-373a0d00885d",
      "communicationUser": {
        "id": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8767-1655-373a0d00885d"
      }
    },
    "participantRemoved": {
      "displayName": "Bob",
      "participantCommunicationIdentifier": {
        "rawId": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8785-1655-373a0d00885f",
        "communicationUser": {
          "id": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8785-1655-373a0d00885f"
        }
      }
    },
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8781-1655-373a0d00885e",
      "communicationUser": {
        "id": "8:acs:0a420b29-555c-4f6b-841e-de8059893bb9_00000008-77c9-8781-1655-373a0d00885e"
      }
    },
    "createTime": "2021-02-25T06:37:17.371Z",
    "version": 1614235220325,
    "transactionId": "usv74GQ5zU+JmWv/bQ+qfg.1.1.1.1480065078.1.5",
    "threadId": "19:f1400e1c542f4086a606b52ad20cd0bd@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatParticipantRemovedFromThreadWithUser",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-25T06:40:24.2244945Z"
}]
```

## Microsoft.Communication.ChatThreadPropertiesUpdatedPerUser event

```
[{
  "id": "d57342ff-264e-4a5e-9c54-ef05b7d50082",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/editedBy/{rawId}/recipient/{rawId}",
  "data": {
    "editedBy": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d07-83fe-084822000f6e",
    "editedByCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d07-83fe-084822000f6e",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7d07-83fe-084822000f6e"
      }
    },
    "editTime": "2021-02-19T00:28:28.739028Z",
    "properties": {
      "topic": "Communication in Azure"
    },
    "createTime": "2021-02-19T00:28:25.864Z",
    "version": 1613694508719,
    "recipientId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724",
    "recipientCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-578d-7caf-07fd-084822001724"
      }
    },
    "transactionId": "WLXPrnJ/I0+LTj2cwMrNMQ.1.1.1.1833369763.1.4",
    "threadId": "19:2cc3504c41244d7483208a4f58a1f188@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatThreadPropertiesUpdatedPerUser",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-19T00:28:29.559726Z"
}]
}
```

## Microsoft.Communication.ChatMemberAddedToThreadWithUser event

```
[{
  "id": "4abd2b49-d1a9-4fcc-9cd7-170fa5d96443",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/memberAdded/{rawId}/recipient/{rawId}",
  "data": {
    "time": "2020-09-18T00:47:13.1867087Z",
    "addedBy": "8:acs:5354158b-17b7-489c-9380-95d8821ff76b_00000005-3e5f-1bc6-f40f-343a0d0003f1",
    "memberAdded": {
      "displayName": "John Smith",
      "memberId": "8:acs:5354158b-17b7-489c-9380-95d8821ff76b_00000005-3e5f-1bc6-f40f-343a0d0003fe"
    },
    "createTime": "2020-09-18T00:46:41.559Z",
    "version": 1600390033176,
    "recipientId": "8:acs:5354158b-17b7-489c-9380-95d8821ff76b_00000005-3e5f-1bc6-f40f-343a0d0003f0",
    "transactionId": "pVIjw/pHEEKUOUJ2DAA15A.1.1.1.1.1818361951.1.1",
    "threadId": "19:6d20c2f921cd402ead7d1b31b0d030cd@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatMemberAddedToThreadWithUser",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2020-09-18T00:47:13.2342692Z"
}]
}
```

## Microsoft.Communication.ChatMemberRemovedFromThreadWithUser event

```
[{
  "id": "b3701976-1ea2-4d66-be68-4ec4fc1b4b96",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/Microsoft.Communication/communicationServices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/memberRemoved/{rawId}/recipient/{rawId}",
  "data": {
    "time": "2020-09-18T00:47:51.1461742Z",
    "removedBy": "8:acs:5354158b-17b7-489c-9380-95d8821ff76b_00000005-3e5f-1bc6-f40f-343a0d0003f1",
    "memberRemoved": {
      "displayName": "John",
      "memberId": "8:acs:5354158b-17b7-489c-9380-95d8821ff76b_00000005-3e5f-1bc6-f40f-343a0d0003fe"
    },
    "createTime": "2020-09-18T00:46:41.559Z",
    "version": 1600390071131,
    "recipientId": "8:acs:5354158b-17b7-489c-9380-95d8821ff76b_00000005-3e5f-1bc6-f40f-343a0d0003f0",
    "transactionId": "G9Y+UbjVmEuxAG304bEyvw.1.1.1.1819803816.1.1",
    "threadId": "19:6d20c2f921cd402ead7d1b31b0d030cd@thread.v2"
  },
  "eventType": "Microsoft.Communication.ChatMemberRemovedFromThreadWithUser",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2020-09-18T00:47:51.2244511Z"
}]
```

### **Microsoft.Communication.ChatThreadCreated event**

```
[ {
  "id": "a607ac52-0974-4d3c-bfd8-6f708a26f509",
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/createdBy/{rawId}",
  "data": {
    "createdByCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453"
      }
    },
    "properties": {
      "topic": "Talk about new Thread Events in communication services"
    },
    "participants": [
      {
        "displayName": "Bob",
        "participantCommunicationIdentifier": {
          "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453",
          "communicationUser": {
            "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453"
          }
        }
      },
      {
        "displayName": "Scott",
        "participantCommunicationIdentifier": {
          "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38e6-07fd-084822002467",
          "communicationUser": {
            "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38e6-07fd-084822002467"
          }
        }
      },
      {
        "displayName": "Shawn",
        "participantCommunicationIdentifier": {
          "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38f6-83fe-084822002337",
          "communicationUser": {
            "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38f6-83fe-084822002337"
          }
        }
      },
      {
        "displayName": "Anthony",
        "participantCommunicationIdentifier": {
          "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38e3-e1fe-084822002c35",
          "communicationUser": {
            "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38e3-e1fe-084822002c35"
          }
        }
      }
    ],
    "createTime": "2021-02-20T00:31:54.365+00:00",
    "version": 1613781114365,
    "threadId": "19:e07c8ddc5bab4c059ea9f11d29b544b6@thread.v2",
    "transactionId": "gK6+kgANy001wchlVKVTJg.1.1.1.921436178.1"
  },
  "eventType": "Microsoft.Communication.ChatThreadCreated",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-20T00:31:54.5369967Z"
}]]
```

## Microsoft.Communication.ChatThreadPropertiesUpdated event

```
[{
  "id": "cf867580-9caf-45be-b49f-ab1cbfcaa59f",
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/editedBy/{rawId}",
  "data": {
    "editedByCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5c9e-9e35-07fd-084822002264",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5c9e-9e35-07fd-084822002264"
      }
    },
    "editTime": "2021-02-20T00:04:07.7152073+00:00",
    "properties": {
      "topic": "Talk about new Thread Events in communication services"
    },
    "createTime": "2021-02-20T00:00:40.126+00:00",
    "version": 1613779447695,
    "threadId": "19:9e8eefe67b3c470a8187b4c2b00240bc@thread.v2",
    "transactionId": "GBE9MB2a40KEWzexIg0D3A.1.1.1.856359041.1"
  },
  "eventType": "Microsoft.Communication.ChatThreadPropertiesUpdated",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-20T00:04:07.8410277Z"
}
]
```

## Microsoft.Communication.ChatThreadDeleted event

```
[{
  {
    "id": "1dbd5237-4823-4fed-980c-8d27c17cf5b0",
    "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
    "subject": "thread/{thread-id}/deletedBy/{rawId}",
    "data": {
      "deletedByCommunicationIdentifier": {
        "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5c9e-a300-07fd-084822002266",
        "communicationUser": {
          "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5c9e-a300-07fd-084822002266"
        }
      },
      "deleteTime": "2021-02-20T00:00:42.109802+00:00",
      "createTime": "2021-02-20T00:00:39.947+00:00",
      "version": 1613779241389,
      "threadId": "19:c9e9f3060b884e448671391882066ac3@thread.v2",
      "transactionId": "KibptDpcLEeEFn1R7cI3QA.1.1.2.1.848298005.1"
    },
    "eventType": "Microsoft.Communication.ChatThreadDeleted",
    "dataVersion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2021-02-20T00:00:42.5428002Z"
  }
]
```

## Microsoft.Communication.ChatThreadParticipantAdded event

```
[
{
  "id": "3024eb5d-1d71-49d1-878c-7dc3165433d9",
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/participantadded/{rawId}",
  "data": {
    "time": "2021-02-20T00:54:42.8622646+00:00",
    "addedByCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453"
      }
    },
    "participantAdded": {
      "displayName": "Bob",
      "participantCommunicationIdentifier": {
        "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38f3-88f7-084822002454",
        "communicationUser": {
          "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38f3-88f7-084822002454"
        }
      }
    },
    "createTime": "2021-02-20T00:31:54.365+00:00",
    "version": 1613782482822,
    "threadId": "19:e07c8ddc5bab4c059ea9f11d29b544b6@thread.v2",
    "transactionId": "9q6c07i4FkaZ+5RRVzshVw.1.1.1.1.974913783.1"
  },
  "eventType": "Microsoft.Communication.ChatThreadParticipantAdded",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-20T00:54:43.9866454Z"
}
]
]
```

## **Microsoft.Communication.ChatThreadParticipantRemoved event**

```
[
{
  "id": "6ed810fd-8776-4b13-81c2-1a0c4f791a07",
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/participantremoved/{rawId}",
  "data": {
    "time": "2021-02-20T00:56:18.1118825+00:00",
    "removedByCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453"
      }
    },
    "participantRemoved": {
      "displayName": "Shawn",
      "participantCommunicationIdentifier": {
        "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38e6-07fd-084822002467",
        "communicationUser": {
          "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38e6-07fd-084822002467"
        }
      }
    },
    "createTime": "2021-02-20T00:31:54.365+00:00",
    "version": 1613782578096,
    "threadId": "19:e07c8ddc5bab4c059ea9f11d29b544b6@thread.v2",
    "transactionId": "z6Cq8IGRr0aEF6C0uy7wSA.1.1.1.1.978649284.1"
  },
  "eventType": "Microsoft.Communication.ChatThreadParticipantRemoved",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-20T00:56:18.856721Z"
}
]
]
```

### **Microsoft.Communication.ChatMessageReceivedInThread event**

```
[
{
  "id": "4f614f97-c451-4b82-a8c9-1e30c3bfcda1",
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/sender/8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cdb-4916-07fd-084822002624",
  "data": {
    "messageBody": "Talk about new Thread Events in communication services",
    "messageId": "1613783230064",
    "type": "Text",
    "version": "1613783230064",
    "senderDisplayName": "Bob",
    "senderCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cdb-4916-07fd-084822002624",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cdb-4916-07fd-084822002624"
      }
    },
    "composeTime": "2021-02-20T01:07:10.064+00:00",
    "threadId": "19:5b3809e80e4a439d92c3316e273f4a2b@thread.v2",
    "transactionId": "foMkntkKS00/MhMlIE5Aag.1.1.1.1.1004077250.1"
  },
  "eventType": "Microsoft.Communication.ChatMessageReceivedInThread",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-20T01:07:10.5704596Z"
}
]
]
```

## Microsoft.Communication.ChatMessageEditedInThread event

```
[
{
  "id": "7b8dc01e-2659-41fa-bc8c-88a967714510",
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
  "subject": "thread/{thread-id}/sender/{rawId}",
  "data": {
    "editTime": "2021-02-20T00:59:10.464+00:00",
    "messageBody": "8efffb181-1eb2-4a58-9d03-ed48a461b19b",
    "messageId": "1613782685964",
    "type": "Text",
    "version": "1613782750464",
    "senderDisplayName": "Scott",
    "senderCommunicationIdentifier": {
      "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453",
      "communicationUser": {
        "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453"
      }
    },
    "composeTime": "2021-02-20T00:58:05.964+00:00",
    "threadId": "19:e07c8ddc5bab4c059ea9f11d29b544b6@thread.v2",
    "transactionId": "H8Gpj3NkIU6bXlWw8WPvhQ.2.1.2.1.985333801.1"
  },
  "eventType": "Microsoft.Communication.ChatMessageEditedInThread",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2021-02-20T00:59:10.7600061Z"
}
]
]
```

## Microsoft.Communication.ChatMessageDeletedInThread event

```
[
  {
    "id": "17d9c39d-0c58-4ed8-947d-c55959f57f75",
    "topic": "/subscriptions/{subscription-id}/resourcegroups/{group-name}/providers/microsoft.communication/communicationservices/{communication-services-resource-name}",
    "subject": "thread/{thread-id}/sender/{rawId}",
    "data": {
      "deleteTime": "2021-02-20T00:59:10.464+00:00",
      "messageId": "1613782685440",
      "type": "Text",
      "version": "1613782814333",
      "senderDisplayName": "Scott",
      "senderCommunicationIdentifier": {
        "rawId": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453",
        "communicationUser": {
          "id": "8:acs:109f0644-b956-4cd9-87b1-71024f6e2f44_00000008-5cbb-38a0-88f7-084822002453"
        }
      },
      "composeTime": "2021-02-20T00:58:05.44+00:00",
      "threadId": "19:e07c8ddc5bab4c059ea9f11d29b544b6@thread.v2",
      "transactionId": "HqU6PeK5AkCRSpW8eAbL0A.1.1.2.1.987824181.1"
    },
    "eventType": "Microsoft.Communication.ChatMessageDeletedInThread",
    "dataVersion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2021-02-20T01:00:14.8518034Z"
  }
]
```

## Quickstarts and how-tos

TITLE	DESCRIPTION
<a href="#">How do handle SMS Events in Communication Services</a>	Handling all SMS events received by your Communication Service using WebHook.

## Tutorials

TITLE	DESCRIPTION
<a href="#">Quickstart: Handle SMS events</a>	Shows how to subscribe to SMS events using Event Grid.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For an introduction to Azure Event Grid Concepts, see [Concepts in Event Grid?](#)
- For an introduction to Azure Event Grid SystemTopics, see [System topics in Azure Event Grid?](#)

# Azure Container Registry as an Event Grid source

3/5/2021 • 4 minutes to read • [Edit Online](#)

This article provides the properties and schema for Container Registry events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

Azure Container Registry emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.ContainerRegistry.ImagePushed	Raised when an image is pushed.
Microsoft.ContainerRegistry.ImageDeleted	Raised when an image is deleted.
Microsoft.ContainerRegistry.ChartPushed	Raised when a Helm chart is pushed.
Microsoft.ContainerRegistry.ChartDeleted	Raised when a Helm chart is deleted.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of an image pushed event:

```
[{
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.ContainerRegistry/registries/<name>",
  "subject": "aci-helloworld:v1",
  "eventType": "ImagePushed",
  "eventTime": "2018-04-25T21:39:47.6549614Z",
  "data": {
    "id": "31c51664-e5bd-416a-a5df-e5206bc47ed0",
    "timestamp": "2018-04-25T21:39:47.276585742Z",
    "action": "push",
    "target": {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "size": 3023,
      "digest": "sha256:213bbc182920ab41e18edc2001e06abcca6735d87782d9cef68abd83941cf0e5",
      "length": 3023,
      "repository": "aci-helloworld",
      "tag": "v1"
    },
    "request": {
      "id": "7c66f28b-de19-40a4-821c-6f5f6c0003a4",
      "host": "demo.azurecr.io",
      "method": "PUT",
      "useragent": "docker/18.03.0-ce go/go1.9.4 git-commit/0520e24 os/windows arch/amd64 UpstreamClient(Docker-Client/18.03.0-ce \\\\"(windows\\\\\\))\""
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

The schema for an image deleted event is similar:

```
[{
  "id": "f06e3921-301f-42ec-b368-212f7d5354bd",
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.ContainerRegistry/registries/<name>",
  "subject": "aci-helloworld",
  "eventType": "ImageDeleted",
  "eventTime": "2018-04-26T17:56:01.8211268Z",
  "data": {
    "id": "f06e3921-301f-42ec-b368-212f7d5354bd",
    "timestamp": "2018-04-26T17:56:00.996603117Z",
    "action": "delete",
    "target": {
      "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
      "digest": "sha256:213bbc182920ab41e18edc2001e06abcca6735d87782d9cef68abd83941cf0e5",
      "repository": "aci-helloworld"
    },
    "request": {
      "id": "aeda5b99-4197-409f-b8a8-ff539edb7de2",
      "host": "demo.azurecr.io",
      "method": "DELETE",
      "useragent": "python-requests/2.18.4"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

The schema for a chart pushed event is similar to the schema for an imaged pushed event, but it doesn't include a request object:

```
[{
  "id": "ea3a9c28-5b17-40f6-a500-3f02b6829277",
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.ContainerRegistry/registries/<name>",
  "subject": "mychart:1.0.0",
  "eventType": "Microsoft.ContainerRegistry.ChartPushed",
  "eventTime": "2019-03-12T22:16:31.5164086Z",
  "data": {
    "id": "ea3a9c28-5b17-40f6-a500-3f02b6829277",
    "timestamp": "2019-03-12T22:16:31.0087496+00:00",
    "action": "chart_push",
    "target": {
      "mediaType": "application/vnd.acr.helm.chart",
      "size": 25265,
      "digest": "sha256:7f060075264b5ba7c14c23672698152ae6a3ebac1c47916e4efe19cd624d5fab",
      "repository": "repo",
      "tag": "mychart-1.0.0.tgz",
      "name": "mychart",
      "version": "1.0.0"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

The schema for a chart deleted event is similar to the schema for an imaged deleted event, but it doesn't include a request object:

```
[{
  "id": "39136b3a-1a7e-416f-a09e-5c85d5402fca",
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.ContainerRegistry/registries/<name>",
  "subject": "mychart:1.0.0",
  "eventType": "Microsoft.ContainerRegistry.ChartDeleted",
  "eventTime": "2019-03-12T22:42:08.7034064Z",
  "data": {
    "id": "ea3a9c28-5b17-40f6-a500-3f02b6829277",
    "timestamp": "2019-03-12T22:42:08.3783775+00:00",
    "action": "chart_delete",
    "target": {
      "mediaType": "application/vnd.acr.helm.chart",
      "size": 25265,
      "digest": "sha256:7f060075264b5ba7c14c23672698152ae6a3ebac1c47916e4efe19cd624d5fab",
      "repository": "repo",
      "tag": "mychart-1.0.0.tgz",
      "name": "mychart",
      "version": "1.0.0"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Blob storage event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>id</code>	string	The event ID.
<code>timestamp</code>	string	The time at which the event occurred.
<code>action</code>	string	The action that encompasses the provided event.
<code>target</code>	object	The target of the event.
<code>request</code>	object	The request that generated the event.

The target object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>mediaType</code>	string	The MIME type of the referenced object.
<code>size</code>	integer	The number of bytes of the content. Same as Length field.

PROPERTY	TYPE	DESCRIPTION
<code>digest</code>	string	The digest of the content, as defined by the Registry V2 HTTP API Specification.
<code>length</code>	integer	The number of bytes of the content. Same as Size field.
<code>repository</code>	string	The repository name.
<code>tag</code>	string	The tag name.
<code>name</code>	string	The chart name.
<code>version</code>	string	The chart version.

The request object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>id</code>	string	The ID of the request that initiated the event.
<code>addr</code>	string	The IP or hostname and possibly port of the client connection that initiated the event. This value is the RemoteAddr from the standard http request.
<code>host</code>	string	The externally accessible hostname of the registry instance, as specified by the http host header on incoming requests.
<code>method</code>	string	The request method that generated the event.
<code>useragent</code>	string	The user agent header of the request.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Quickstart: send container registry events</a>	Shows how to use Azure CLI to send Container Registry events.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Azure Event Hubs as an Event Grid source

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article provides the properties and schema for event hubs events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

Event Hubs emits the `Microsoft.EventHub.CaptureFileCreated` event type when a capture file is created.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

This sample event shows the schema of an event hubs event raised when the capture feature stores a file:

```
[  
  {  
    "topic":  
      "/subscriptions/<guid>/resourcegroups/rgDataMigrationSample/providers/Microsoft.EventHub/namespaces/tfdatami  
gratens",  
    "subject": "eventhubs/hubdatamigration",  
    "eventType": "Microsoft.EventHub.CaptureFileCreated",  
    "eventTime": "2017-08-31T19:12:46.0498024Z",  
    "id": "14e87d03-6fbf-4bb2-9a21-92bd1281f247",  
    "data": {  
      "fileUrl":  
        "https://tf0831datamigrate.blob.core.windows.net/windturbinecapture/tfdatamigratens/hubdatamigration/1/2017/  
08/31/19/11/45.avro",  
      "fileType": "AzureBlockBlob",  
      "partitionId": "1",  
      "sizeInBytes": 249168,  
      "eventCount": 1500,  
      "firstSequenceNumber": 2400,  
      "lastSequenceNumber": 3899,  
      "firstEnqueueTime": "2017-08-31T19:12:14.674Z",  
      "lastEnqueueTime": "2017-08-31T19:12:44.309Z"  
    },  
    "dataVersion": "",  
    "metadataVersion": "1"  
  }  
]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
----------	------	-------------

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field is not writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Event hub event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>fileUrl</code>	string	The path to the capture file.
<code>fileType</code>	string	The file type of the capture file.
<code>partitionId</code>	string	The shard ID.
<code>sizeInBytes</code>	integer	The file size.
<code>eventCount</code>	integer	The number of events in the file.
<code>firstSequenceNumber</code>	integer	The smallest sequence number from the queue.
<code>lastSequenceNumber</code>	integer	The last sequence number from the queue.
<code>firstEnqueueTime</code>	string	The first time from the queue.
<code>lastEnqueueTime</code>	string	The last time from the queue.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Tutorial: stream big data into a data warehouse</a>	When Event Hubs creates a Capture file, Event Grid sends an event to a function app. The app retrieves the Capture file and migrates data to a data warehouse.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).
- For information about handling event hubs events, see [Stream big data into a data warehouse](#).

# Azure IoT Hub as an Event Grid source

3/5/2021 • 7 minutes to read • [Edit Online](#)

This article provides the properties and schema for Azure IoT Hub events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

Azure IoT Hub emits the following event types:

Event Type	Description
Microsoft.Devices.DeviceCreated	Published when a device is registered to an IoT hub.
Microsoft.Devices.DeviceDeleted	Published when a device is deleted from an IoT hub.
Microsoft.Devices.DeviceConnected	Published when a device is connected to an IoT hub.
Microsoft.Devices.DeviceDisconnected	Published when a device is disconnected from an IoT hub.
Microsoft.Devices.DeviceTelemetry	Published when a telemetry message is sent to an IoT hub.

## Example event

- Event Grid event schema
  - Cloud event schema

The schema for DeviceConnected and DeviceDisconnected events have the same structure. This sample event shows the schema of an event raised when a device is connected to an IoT hub:

The DeviceTelemetry event is raised when a telemetry event is sent to an IoT Hub. A sample schema for this event is shown below.

```
[{
  "id": "9af86784-8d40-fe2g-8b2a-bab65e106785",
  "topic": "/SUBSCRIPTIONS/<subscription ID>/RESOURCEGROUPS/<resource group name>/PROVIDERS/MICROSOFT.DEVICES/IOTHUBS/<hub name>",
  "subject": "devices/LogicAppTestDevice",
  "eventType": "Microsoft.Devices.DeviceTelemetry",
  "eventTime": "2019-01-07T20:58:30.48Z",
  "data": {
    "body": {
      "Weather": {
        "Temperature": 900
      },
      "Location": "USA"
    },
    "properties": {
      "Status": "Active"
    },
    "systemProperties": {
      "iothub-content-type": "application/json",
      "iothub-content-encoding": "utf-8",
      "iothub-connection-device-id": "d1",
      "iothub-connection-auth-method": ""
    }
  },
  "scope": {
    "device": {
      "type": "sas",
      "issuer": "iothub",
      "acceptingIpFilterRule": null,
      "iothub-connection-auth-generation-id": "123455432199234570",
      "iothub-queuedtime": "2019-01-07T20:58:30.48Z",
      "iothub-message-source": "Telemetry"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

The schema for DeviceCreated and DeviceDeleted events have the same structure. This sample event shows the schema of an event raised when a device is registered to an IoT hub:

```
[{
  "id": "56afc886-767b-d359-d59e-0da7877166b2",
  "topic": "/SUBSCRIPTIONS/<subscription ID>/RESOURCEGROUPS/<resource group name>/PROVIDERS/MICROSOFT.DEVICES/IOTHUBS/<hub name>",
  "subject": "devices/LogicAppTestDevice",
  "eventType": "Microsoft.Devices.DeviceCreated",
  "eventTime": "2018-01-02T19:17:44.4383997Z",
  "data": {
    "twin": {
      "deviceId": "LogicAppTestDevice",
      "etag": "AAAAAAAAAAE=",
      "deviceEtag": "null",
      "status": "enabled",
      "statusUpdateTime": "0001-01-01T00:00:00",
      "connectionState": "Disconnected",
      "lastActivityTime": "0001-01-01T00:00:00",
      "cloudToDeviceMessageCount": 0,
      "authenticationType": "sas",
      "x509Thumbprint": {
        "primaryThumbprint": null,
        "secondaryThumbprint": null
      },
      "version": 2,
      "properties": {
        "desired": {
          "$metadata": {
            "$lastUpdated": "2018-01-02T19:17:44.4383997Z"
          },
          "$version": 1
        },
        "reported": {
          "$metadata": {
            "$lastUpdated": "2018-01-02T19:17:44.4383997Z"
          },
          "$version": 1
        }
      },
      "hubName": "egtesthub1",
      "deviceId": "LogicAppTestDevice"
    },
    "dataVersion": "1",
    "metadataVersion": "1"
  }
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

All events contain the same top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>id</code>	string	Unique identifier for the event.
<code>topic</code>	string	Full resource path to the event source. This field is not writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.

PROPERTY	TYPE	DESCRIPTION
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>data</code>	object	IoT Hub event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

For all IoT Hub events, the data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>hubName</code>	string	Name of the IoT Hub where the device was created or deleted.
<code>deviceId</code>	string	The unique identifier of the device. This case-sensitive string can be up to 128 characters long, and supports ASCII 7-bit alphanumeric characters plus the following special characters: <div style="border: 1px solid black; padding: 2px; display: inline-block;"> - : . + % _ # * ? ! ( ) , = @ ;  \$ ' </div>

The contents of the data object are different for each event publisher.

For **Device Connected** and **Device Disconnected** IoT Hub events, the data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>moduleId</code>	string	The unique identifier of the module. This field is output only for module devices. This case-sensitive string can be up to 128 characters long, and supports ASCII 7-bit alphanumeric characters plus the following special characters: <div style="border: 1px solid black; padding: 2px; display: inline-block;"> - : . + % _ # * ? ! ( ) , = @ ;  \$ ' </div>
<code>deviceConnectionStateEventInfo</code>	object	Device connection state event information

PROPERTY	TYPE	DESCRIPTION
sequenceNumber	string	A number which helps indicate order of device connected or device disconnected events. Latest event will have a sequence number that is higher than the previous event. This number may change by more than 1, but is strictly increasing. See <a href="#">how to use sequence number</a> .

For **Device Telemetry** IoT Hub event, the data object contains the device-to-cloud message in [IoT hub message format](#) and has the following properties:

PROPERTY	TYPE	DESCRIPTION
body	string	The content of the message from the device.
properties	string	Application properties are user-defined strings that can be added to the message. These fields are optional.
system properties	string	<b>System properties</b> help identify contents and source of the messages. Device telemetry message must be in a valid JSON format with the contentType set to JSON and contentEncoding set to UTF-8 in the message system properties. If this is not set, then IoT Hub will write the messages in base 64 encoded format.

For **Device Created** and **Device Deleted** IoT Hub events, the data object contains the following properties:

PROPERTY	TYPE	DESCRIPTION
twin	object	Information about the device twin, which is the cloud representation of application device metadata.
deviceID	string	The unique identifier of the device twin.
etag	string	A validator for ensuring consistency of updates to a device twin. Each etag is guaranteed to be unique per device twin.
deviceEtag	string	A validator for ensuring consistency of updates to a device registry. Each deviceEtag is guaranteed to be unique per device registry.
status	string	Whether the device twin is enabled or disabled.

PROPERTY	TYPE	DESCRIPTION
<code>statusUpdateTime</code>	string	The ISO8601 timestamp of the last device twin status update.
<code>connectionState</code>	string	Whether the device is connected or disconnected.
<code>lastActivityTime</code>	string	The ISO8601 timestamp of the last activity.
<code>cloudToDeviceMessageCount</code>	integer	Count of cloud to device messages sent to this device.
<code>authenticationType</code>	string	Authentication type used for this device: either <code>SAS</code> , <code>SelfSigned</code> , or <code>CertificateAuthority</code> .
<code>x509Thumbprint</code>	string	The thumbprint is a unique value for the x509 certificate, commonly used to find a particular certificate in a certificate store. The thumbprint is dynamically generated using the SHA1 algorithm, and does not physically exist in the certificate.
<code>primaryThumbprint</code>	string	Primary thumbprint for the x509 certificate.
<code>secondaryThumbprint</code>	string	Secondary thumbprint for the x509 certificate.
<code>version</code>	integer	An integer that is incremented by one each time the device twin is updated.
<code>desired</code>	object	A portion of the properties that can be written only by the application back-end, and read by the device.
<code>reported</code>	object	A portion of the properties that can be written only by the device, and read by the application back-end.
<code>lastUpdated</code>	string	The ISO8601 timestamp of the last device twin property update.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Send email notifications about Azure IoT Hub events using Logic Apps</a>	A logic app sends a notification email every time a device is added to your IoT Hub.
<a href="#">React to IoT Hub events by using Event Grid to trigger actions</a>	Overview of integrating IoT Hub with Event Grid.

TITLE	DESCRIPTION
Order device connected and device disconnected events	Shows how to order device connection state events.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- To learn about how IoT Hub and Event Grid work together, see [React to IoT Hub events by using Event Grid to trigger actions](#).

# Azure Key Vault as Event Grid source

6/8/2021 • 3 minutes to read • [Edit Online](#)

This article provides the properties and schema for events in [Azure Key Vault](#). For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

An Azure Key Vault account generates the following event types:

Event Full Name	Event Display Name	Description
Microsoft.KeyVault.CertificateNewVersionCreated	Certificate New Version Created	Triggered when a new certificate or new certificate version is created.
Microsoft.KeyVault.CertificateNearExpiry	Certificate Near Expiry	Triggered when the current version of certificate is about to expire. (The event is triggered 30 days before the expiration date.)
Microsoft.KeyVault.CertificateExpired	Certificate Expired	Triggered when the certificate is expired.
Microsoft.KeyVault.KeyNewVersionCreated	Key New Version Created	Triggered when a new key or new key version is created.
Microsoft.KeyVault.KeyNearExpiry	Key Near Expiry	Triggered when the current version of a key is about to expire. (The event is triggered 30 days before the expiration date.)
Microsoft.KeyVault.KeyExpired	Key Expired	Triggered when a key is expired.
Microsoft.KeyVault.SecretNewVersionCreated	Secret New Version Created	Triggered when a new secret or new secret version is created.
Microsoft.KeyVault.SecretNearExpiry	Secret Near Expiry	Triggered when the current version of a secret is about to expire. (The event is triggered 30 days before the expiration date.)
Microsoft.KeyVault.SecretExpired	Secret Expired	Triggered when a secret is expired.
Microsoft.KeyVault.VaultAccessPolicyChanged	Vault Access Policy Changed	Triggered when an access policy on Key Vault changed. It includes a scenario when Key Vault permission model is changed to/from Azure role-based access control.

## Event examples

- [Event Grid event schema](#)

- [Cloud event schema](#)

The following example show schema for `Microsoft.KeyVault.SecretNewVersionCreated`:

```
[
  {
    "id": "00eccf70-95a7-4e7c-8299-2eb17ee9ad64",
    "topic": "/subscriptions/{subscription-id}/resourceGroups/sample-
rg/providers/Microsoft.KeyVault/vaults/sample-kv",
    "subject": "newsecret",
    "eventType": "Microsoft.KeyVault.SecretNewVersionCreated",
    "eventTime": "2019-07-25T01:08:33.1036736Z",
    "data": {
      "Id": "https://sample-kv.vault.azure.net/secrets/newsecret/ee059b2bb5bc48398a53b168c6cdcb10",
      "VaultName": "sample-kv",
      "ObjectType": "Secret",
      "ObjectName": "newsecret",
      "Version": " ee059b2bb5bc48398a53b168c6cdcb10",
      "NBF": "1559081980",
      "EXP": "1559082102"
    },
    "dataVersion": "1",
    "metadataVersion": "1"
  }
]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	App Configuration event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.

PROPERTY	TYPE	DESCRIPTION
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>id</code>	string	The ID of the object that triggered this event
<code>VaultName</code>	string	The key vault name of the object that triggered this event
<code>ObjectType</code>	string	The type of the object that triggered this event
<code>ObjectName</code>	string	The name of the object that triggered this event
<code>Version</code>	string	The version of the object that triggered this event
<code>NBF</code>	number	The not-before date in seconds since 1970-01-01T00:00:00Z of the object that triggered this event
<code>EXP</code>	number	The expiration date in seconds since 1970-01-01T00:00:00Z of the object that triggered this event

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Monitoring Key Vault events with Azure Event Grid</a>	Overview of integrating Key Vault with Event Grid.
<a href="#">Tutorial: Create and monitor Key Vault events with Event Grid</a>	Learn how to set up Event Grid notifications for Key Vault.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#).
- For more information about how to create an Azure Event Grid subscription, see [Event Grid subscription schema](#).
- For more information about Key Vault, see [What is Azure Key Vault?](#)

# Azure Machine Learning as an Event Grid source

3/5/2021 • 6 minutes to read • [Edit Online](#)

This article provides the properties and schema for machine learning workspace events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

Azure Machine Learning emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.MachineLearningServices.ModelRegistered	Raised when a new Model or Model version has been successfully registered.
Microsoft.MachineLearningServices.ModelDeployed	Raised when Model(s) have been successfully deployed to an Endpoint.
Microsoft.MachineLearningServices.RunCompleted	Raised when a Run has been successfully completed.
Microsoft.MachineLearningServices.DatasetDriftDetected	Raised when a Dataset drift monitor detects drift.
Microsoft.MachineLearningServices.RunStatusChanged	Raised when a run status changes.

## Example events

When an event is triggered, the Event Grid service sends data about that event to subscribing endpoint. This section contains an example of what that data would look like for each event.

- [Event Grid event schema](#)
- [Cloud event schema](#)

### **Microsoft.MachineLearningServices.ModelRegistered event**

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearningServices/workspaces/{workspace-name}",
  "subject": "models/sklearn_regression_model:20",
  "eventType": "Microsoft.MachineLearningServices.ModelRegistered",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "ModelName": "sklearn_regression_model",
    "ModelVersion": 20,
    "ModelTags": {
      "area": "diabetes",
      "type": "regression"
    },
    "ModelProperties": {
      "type": "test"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Microsoft.MachineLearningServices.ModelDeployed event

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearningServices/workspaces/{workspace-name}",
  "subject": "endpoints/my-sklearn-service",
  "eventType": "Microsoft.MachineLearningServices.ModelDeployed",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "ServiceName": "my-sklearn-service",
    "ServiceComputeType": "ACI",
    "ModelIds": "sklearn_regression_model:1,sklearn_regression_model:2",
    "ServiceTags": {
      "area": "diabetes",
      "type": "regression"
    },
    "ServiceProperties": {
      "type": "test"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Microsoft.MachineLearningServices.RunCompleted event

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearningServices/workspaces/{workspace-name}",
  "subject": "experiments/0fa9dfaa-cba3-4fa7-b590-23e48548f5c1/runs/AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5",
  "eventType": "Microsoft.MachineLearningServices.RunCompleted",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "experimentId": "0fa9dfaa-cba3-4fa7-b590-23e48548f5c1",
    "experimentName": "automl-local-regression",
    "runId": "AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5",
    "runType": null,
    "runTags": {},
    "runProperties": {
      "runTemplate": "automl_child",
      "pipeline_id": "5adc0a4fe02504a586f09a4fcbb241f9a4012062",
      "pipeline_spec": "{\"objects\": [{\"class_name\": \"StandardScaler\", \"module\": \"sklearn.preprocessing\", \"param_args\": [], \"param_kwargs\": {\"with_mean\": true, \"with_std\": false}, \"prepared_kwargs\": {}}, {"spec_class\": \"preproc\", \"class_name\": \"LassoLars\", \"module\": \"sklearn.linear_model\", \"param_args\": [], \"param_kwargs\": {\"alpha\": 0.001, \"normalize\": true}, \"prepared_kwargs\": {}, \"spec_class\": \"sklearn\"}], \"pipeline_id\": \"5adc0a4fe02504a586f09a4fcbb241f9a4012062\"}",
      "training_percent": "100",
      "predicted_cost": "0.062226144097381045",
      "iteration": "5",
      "run_template": "automl_child",
      "run_preprocessor": "StandardScalerWrapper",
      "run_algorithm": "LassoLars",
      "conda_env_data_location": "aml://artifact/ExperimentRun/dcid.AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5/outputs/conda_env_v_1_0_0.yml",
      "model_name": "AutoMLad912b2d65",
      "scoring_data_location": "aml://artifact/ExperimentRun/dcid.AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5/outputs/scoring_file_v_1_0_0.py",
      "model_data_location": "aml://artifact/ExperimentRun/dcid.AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5/outputs/model.pkl"
    }
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Microsoft.MachineLearningServices.DatasetDriftDetected event

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearningServices/workspaces/{workspace-name}",
  "subject": "datadrifts/{}/runs/{}",
  "eventType": "Microsoft.MachineLearningServices.DatasetDriftDetected",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "DataDriftId": "01d29aa4-e6a4-470a-9ef3-66660d21f8ef",
    "DataDriftName": "myDriftMonitor",
    "RunId": "01d29aa4-e6a4-470a-9ef3-66660d21f8ef_1571590300380",
    "BaseDatasetId": "3c56d136-0f64-4657-a0e8-5162089a88a3",
    "TargetDatasetId": "d7e74d2e-c972-4266-b5fb-6c9c182d2a74",
    "DriftCoefficient": 0.83503490684792081,
    "StartTime": "2019-07-04T00:00:00+00:00",
    "EndTime": "2019-07-05T00:00:00+00:00"
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

## Microsoft.MachineLearningServices.RunStatusChanged event

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group-name}/providers/Microsoft.MachineLearningServices/workspaces/{workspace-name}",
  "subject": "experiments/0fa9dcaa-cba3-4fa7-b590-23e48548f5c1/runs/AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5",
  "eventType": "Microsoft.MachineLearningServices.RunStatusChanged",
  "eventTime": "2017-06-26T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "experimentId": "0fa9dcaa-cba3-4fa7-b590-23e48548f5c1",
    "experimentName": "automl-local-regression",
    "runId": "AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5",
    "runType": null,
    "runTags": {},
    "runProperties": {
      "runTemplate": "automl_child",
      "pipeline_id": "5adc0a4fe02504a586f09a4fcbb241f9a4012062",
      "pipeline_spec": "{\"objects\": [{\"class_name\": \"StandardScaler\", \"module\": \"sklearn.preprocessing\", \"param_args\": [], \"param_kwargs\": {\"with_mean\": true, \"with_std\": false}, \"prepared_kwargs\": {}, \"spec_class\": \"preproc\"}, {\"class_name\": \"LassoLars\", \"module\": \"sklearn.linear_model\", \"param_args\": [], \"param_kwargs\": {\"alpha\": 0.001, \"normalize\": true}, \"prepared_kwargs\": {}, \"spec_class\": \"sklearn\"}], \"pipeline_id\": \"5adc0a4fe02504a586f09a4fcbb241f9a4012062\"}",
      "training_percent": "100",
      "predicted_cost": "0.062226144097381045",
      "iteration": "5",
      "run_template": "automl_child",
      "run_preprocessor": "StandardScalerWrapper",
      "run_algorithm": "LassoLars",
      "conda_env_data_location": "aml://artifact/ExperimentRun/dcid.AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5/outputs/conda_env_v_1_0_0.yml",
      "model_name": "AutoMLad912b2d65",
      "scoring_data_location": "aml://artifact/ExperimentRun/dcid.AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5/outputs/scoring_file_v_1_0_0.py",
      "model_data_location": "aml://artifact/ExperimentRun/dcid.AutoML_ad912b2d-6467-4f32-a616-dbe4af6dd8fc_5/outputs/model.pkl"
    },
    "runStatus": "failed"
  },
  "dataVersion": "",
  "metadataVersion": "1"
}]
```

### Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.

PROPERTY	TYPE	DESCRIPTION
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Blob storage event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties for each event type:

#### **Microsoft.MachineLearningServices.ModelRegistered**

PROPERTY	TYPE	DESCRIPTION
<code>ModelName</code>	string	The name of the model that was registered.
<code>ModelVersion</code>	string	The version of the model that was registered.
<code>ModelTags</code>	object	The tags of the model that was registered.
<code>ModelProperties</code>	object	The properties of the model that was registered.

#### **Microsoft.MachineLearningServices.ModelDeployed**

PROPERTY	TYPE	DESCRIPTION
<code>ServiceName</code>	string	The name of the deployed service.
<code>ServiceComputeType</code>	string	The compute type (for example, ACI, AKS) of the deployed service.
<code>ModelIds</code>	string	A comma-separated list of model IDs. The IDs of the models deployed in the service.
<code>ServiceTags</code>	object	The tags of the deployed service.
<code>ServiceProperties</code>	object	The properties of the deployed service.

#### **Microsoft.MachineLearningServices.RunCompleted**

PROPERTY	TYPE	DESCRIPTION
<code>experimentId</code>	string	The ID of the experiment that the run belongs to.
<code>experimentName</code>	string	The name of the experiment that the run belongs to.
<code>runId</code>	string	The ID of the Run that was completed.
<code>runType</code>	string	The Run Type of the completed Run.
<code>runTags</code>	object	The tags of the completed Run.
<code>runProperties</code>	object	The properties of the completed Run.

### **Microsoft.MachineLearningServices.DatasetDriftDetected**

PROPERTY	TYPE	DESCRIPTION
<code>DataDriftId</code>	string	The ID of the data drift monitor that triggered the event.
<code>DataDriftName</code>	string	The name of the data drift monitor that triggered the event.
<code>RunId</code>	string	The ID of the Run that detected data drift.
<code>BaseDatasetId</code>	string	The ID of the base Dataset used to detect drift.
<code>TargetDatasetId</code>	string	The ID of the target Dataset used to detect drift.
<code>DriftCoefficient</code>	double	The coefficient result that triggered the event.
<code>StartTime</code>	datetime	The start time of the target dataset time series that resulted in drift detection.
<code>EndTime</code>	datetime	The end time of the target dataset time series that resulted in drift detection.

### **Microsoft.MachineLearningServices.RunStatusChanged**

PROPERTY	TYPE	DESCRIPTION
<code>experimentId</code>	string	The ID of the experiment that the run belongs to.
<code>experimentName</code>	string	The name of the experiment that the run belongs to.

PROPERTY	TYPE	DESCRIPTION
<code>runId</code>	string	The ID of the Run that was completed.
<code>runType</code>	string	The Run Type of the completed Run.
<code>runTags</code>	object	The tags of the completed Run.
<code>runProperties</code>	object	The properties of the completed Run.
<code>runStatus</code>	string	The status of the Run.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Consume Azure Machine Learning events</a>	Overview of integrating Azure Machine Learning with Event Grid.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#)
- For an introduction to using Azure Event Grid with Azure Machine Learning, see [Consume Azure Machine Learning events](#)
- For an example of using Azure Event Grid with Azure Machine Learning, see [Create event driven machine learning workflows](#)

# Azure Maps as an Event Grid source

3/5/2021 • 5 minutes to read • [Edit Online](#)

This article provides the properties and schema for Azure Maps events. For an introduction to event schemas, see [Azure Event Grid event schema](#). It also gives you a list of quick starts and tutorials to use Azure Maps as an event source.

## Available event types

An Azure Maps account emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.Maps.GeofenceEntered	Raised when coordinates received have moved from outside of a given geofence to within
Microsoft.Maps.GeofenceExited	Raised when coordinates received have moved from within a given geofence to outside
Microsoft.Maps.GeofenceResult	Raised every time a geofencing query returns a result, regardless of the state

## Example events

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of a **GeofenceEntered** event

```
{
  "id": "7f8446e2-1ac7-4234-8425-303726ea3981",
  "topic": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Maps/accounts/{accountName}",
  "subject": "/spatial/geofence/udid/{udid}/id/{eventId}",
  "data": {
    "geometries": [
      {
        "deviceId": "device_1",
        "udId": "1a13b444-4acf-32ab-ce4e-9ca4af20b169",
        "geometryId": "2",
        "distance": -999.0,
        "nearestLat": 47.618786,
        "nearestLon": -122.132151
      }
    ],
    "expiredGeofenceGeometryId": [
    ],
    "invalidPeriodGeofenceGeometryId": [
    ],
    "eventType": "Microsoft.Maps.GeofenceEntered",
    "eventTime": "2018-11-08T00:54:17.6408601Z",
    "metadataVersion": "1",
    "dataVersion": "1.0"
  }
}
```

The following example show schema for **GeofenceResult**

```
{
  "id": "451675de-a67d-4929-876c-5c2bf0b2c000",
  "topic": "/subscriptions/{subscriptionId}/resourceGroups/{resourceGroup}/providers/Microsoft.Maps/accounts/{accountName}",
  "subject": "/spatial/geofence/udid/{udid}/id/{eventId}",
  "data": {
    "geometries": [
      {
        "deviceId": "device_1",
        "udId": "1a13b444-4acf-32ab-ce4e-9ca4af20b169",
        "geometryId": "1",
        "distance": 999.0,
        "nearestLat": 47.609833,
        "nearestLon": -122.148274
      },
      {
        "deviceId": "device_1",
        "udId": "1a13b444-4acf-32ab-ce4e-9ca4af20b169",
        "geometryId": "2",
        "distance": 999.0,
        "nearestLat": 47.621954,
        "nearestLon": -122.131841
      }
    ],
    "expiredGeofenceGeometryId": [
    ],
    "invalidPeriodGeofenceGeometryId": [
    ],
    "eventType": "Microsoft.Maps.GeofenceResult",
    "eventTime": "2018-11-08T00:52:08.0954283Z",
    "metadataVersion": "1",
    "dataVersion": "1.0"
  }
}
```

# Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Geofencing event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>apiCategory</code>	string	API category of the event.
<code>apiName</code>	string	API name of the event.
<code>issues</code>	object	Lists issues occurred during processing. If any issues are returned, then there will be no geometries returned with the response.
<code>responseCode</code>	number	HTTP response code
<code>geometries</code>	object	Lists the fence geometries that contain the coordinate position or overlap the searchBuffer around the position.

The error object is returned when an error occurs in the Maps API. The error object has the following properties:

PROPERTY	TYPE	DESCRIPTION
error	ErrorDetails	This object is returned when an error occurs in the Maps API

The ErrorDetails object is returned when an error occurs in the Maps API. The ErrorDetails object has the following properties:

PROPERTY	TYPE	DESCRIPTION
code	string	The HTTP status code.
message	string	If available, a human readable description of the error.
innererror	InnerError	If available, an object containing service-specific information about the error.

The InnerError is an object containing service-specific information about the error. The InnerError object has the following properties:

PROPERTY	TYPE	DESCRIPTION
code	string	The error message.

The geometries object lists geometry IDs of the geofences that have expired relative to the user time in the request. The geometries object has geometry items with the following properties:

PROPERTY	TYPE	DESCRIPTION
deviceid	string	ID of device.
distance	string	Distance from the coordinate to the closest border of the geofence. Positive means the coordinate is outside of the geofence. If the coordinate is outside of the geofence, but more than the value of searchBuffer away from the closest geofence border, then the value is 999. Negative means the coordinate is inside of the geofence. If the coordinate is inside the polygon, but more than the value of searchBuffer away from the closest geofencing border, then the value is -999. A value of 999 means that there is great confidence the coordinate is well outside the geofence. A value of -999 means that there is great confidence the coordinate is well within the geofence.

PROPERTY	TYPE	DESCRIPTION
<code>geometryId</code>	string	The unique ID identifies the geofence geometry.
<code>nearestLat</code>	number	Latitude of the nearest point of the geometry.
<code>nearestLon</code>	number	Longitude of the nearest point of the geometry.
<code>udId</code>	string	The unique ID returned from user upload service when uploading a geofence. Won't be included in geofencing post API.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>expiredGeofenceGeometryId</code>	string[]	Lists of the geometry ID of the geofence that is expired relative to the user time in the request.
<code>geometries</code>	geometries[]	Lists the fence geometries that contain the coordinate position or overlap the searchBuffer around the position.
<code>invalidPeriodGeofenceGeometryId</code>	string[]	Lists of the geometry ID of the geofence that is in invalid period relative to the user time in the request.
<code>isEventPublished</code>	boolean	True if at least one event is published to the Azure Maps event subscriber, false if no event is published to the Azure Maps event subscriber.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">React to Azure Maps events by using Event Grid</a>	Overview of integrating Azure Maps with Event Grid.
<a href="#">Tutorial: Set up a geofence</a>	This tutorial walks you through the basics steps to set up geofence by using Azure Maps. You use Azure Event Grid to stream the geofence results and set up a notification based on the geofence results.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Azure Media Services as an Event Grid source

4/2/2021 • 17 minutes to read • [Edit Online](#)

This article provides the schemas and properties for Media Services events.

## Job-related event types

Media Services emits the **Job-related** event types described below. There are two categories for the **Job-related** events: "Monitoring Job State Changes" and "Monitoring Job Output State Changes".

You can register for all of the events by subscribing to the `JobStateChange` event. Or, you can subscribe for specific events only (for example, final states like `JobErrored`, `JobFinished`, and `JobCanceled`).

### Monitoring Job state changes

EVENT TYPE	DESCRIPTION
<code>Microsoft.Media.JobStateChange</code>	Get an event for all Job State changes.
<code>Microsoft.Media.JobScheduled</code>	Get an event when Job transitions to scheduled state.
<code>Microsoft.Media.JobProcessing</code>	Get an event when Job transitions to processing state.
<code>Microsoft.Media.JobCanceling</code>	Get an event when Job transitions to canceling state.
<code>Microsoft.Media.JobCanceled</code>	Get an event when Job transitions to canceled state. This is a final state that includes Job outputs.
<code>Microsoft.Media.JobErrored</code>	Get an event when Job transitions to error state. This is a final state that includes Job outputs.

See [Schema examples](#) that follow.

### Monitoring job output state changes

A job may contain multiple job outputs (if you configured the transform to have multiple job outputs.) If you want to track the details of the individual job output, listen for a job output change event.

Each **Job** is going to be at a higher level than **JobOutput**, thus job output events get fired inside of a corresponding job.

The error messages in `JobFinished`, `JobCanceled`, `JobError` output the aggregated results for each job output – when all of them are finished. Whereas the job output events fire as each task finishes. For example, if you have an encoding output, followed by a Video Analytics output, you would get two events firing as job output events before the final `JobFinished` event fires with the aggregated data.

EVENT TYPE	DESCRIPTION
<code>Microsoft.Media.JobOutputStateChange</code>	Get an event for all Job output State changes.
<code>Microsoft.Media.JobOutputScheduled</code>	Get an event when Job output transitions to scheduled state.

EVENT TYPE	DESCRIPTION
Microsoft.Media.JobOutputProcessing	Get an event when Job output transitions to processing state.
Microsoft.Media.JobOutputCanceling	Get an event when Job output transitions to canceling state.
Microsoft.Media.JobOutputFinished	Get an event when Job output transitions to finished state.
Microsoft.Media.JobOutputCanceled	Get an event when Job output transitions to canceled state.
Microsoft.Media.JobOutputErrored	Get an event when Job output transitions to error state.

See [Schema examples](#) that follow.

## Monitoring job output progress

EVENT TYPE	DESCRIPTION
Microsoft.Media.JobOutputProgress	This event reflects the job processing progress, from 0% to 100%. The service attempts to send an event if there has been 5% or greater increase in the progress value or it has been more than 30 seconds since the last event (heartbeat). The progress value isn't guaranteed to start at 0%, or to reach 100%, nor is it guaranteed to increase at a constant rate over time. This event should not be used to determine that the processing has been completed – you should instead use the state change events.

See [Schema examples](#) that follow.

## Live event types

Media Services also emits the **Live** event types described below. There are two categories for the **Live** events: stream-level events and track-level events.

### Stream-level events

Stream-level events are raised per stream or connection. Each event has a `streamId` parameter that identifies the connection or stream. Each stream or connection has one or more tracks of different types. For example, one connection from an encoder may have one audio track and four video tracks. The stream event types are:

EVENT TYPE	DESCRIPTION
Microsoft.Media.LiveEventConnectionRejected	Encoder's connection attempt is rejected.
Microsoft.Media.LiveEventEncoderConnected	Encoder establishes connection with live event.
Microsoft.Media.LiveEventEncoderDisconnected	Encoder disconnects.

See [Schema examples](#) that follow.

### Track-level events

Track-level events are raised per track.

**NOTE**

All track-level events are raised after a live encoder is connected.

The track-level event types are:

EVENT TYPE	DESCRIPTION
Microsoft.Media.LiveEventIncomingDataChunkDropped	Media server drops data chunk because it's too late or has an overlapping timestamp (timestamp of new data chunk is less than the end time of the previous data chunk).
Microsoft.Media.LiveEventIncomingStreamReceived	Media server receives first data chunk for each track in the stream or connection.
Microsoft.Media.LiveEventIncomingStreamsOutOfSync	Media server detects audio and video streams are out of sync. Use as a warning because user experience may not be impacted.
Microsoft.Media.LiveEventIncomingVideoStreamsOutOfSync	Media server detects any of the two video streams coming from external encoder are out of sync. Use as a warning because user experience may not be impacted.
Microsoft.Media.LiveEventIngestHeartbeat	Published every 20 seconds for each track when live event is running. Provides ingest health summary.  After the encoder was initially connected, the heartbeat event continues to emit every 20 sec whether the encoder is still connected or not.
Microsoft.Media.LiveEventTrackDiscontinuityDetected	Media server detects discontinuity in the incoming track.

See [Schema examples](#) that follow.

## Event schema examples

### JobStateChange

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **JobStateChange** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
    "eventType": "Microsoft.Media.JobStateChange",
    "eventTime": "2018-04-20T21:26:13.8978772",
    "id": "b9d38923-9210-4c2b-958f-0054467d4dd7",
    "data": {
      "previousState": "Processing",
      "state": "Finished"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>previousState</code>	string	The state of the job before the event.
<code>state</code>	string	The new state of the job being notified in this event. For example, "Scheduled: The job is ready to start" or "Finished: The job is finished".

Where the Job state can be one of the values: *Queued, Scheduled, Processing, Finished, Error, Canceled, Canceling*

#### NOTE

*Queued* is only going to be present in the `previousState` property but not in the `state` property.

### JobScheduled, JobProcessing, JobCanceling

- [Event Grid event schema](#)
- [Cloud event schema](#)

For each non-final Job state change (such as JobScheduled, JobProcessing, JobCanceling), the example schema looks similar to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mEDIAServices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobProcessing",
  "eventTime": "2018-10-12T16:12:18.0839935",
  "id": "a0a6efc8-f647-4fc2-be73-861fa25ba2db",
  "data": {
    "previousState": "Scheduled",
    "state": "Processing",
    "correlationData": {
      "testKey1": "testValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

### JobFinished, JobCanceled, JobErrored

For each final Job state change (such as JobFinished, JobCanceled, JobErrored), the example schema looks similar to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mEDIAServices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobFinished",
  "eventTime": "2018-10-12T16:25:56.4115495",
  "id": "9e07e83a-dd6e-466b-a62f-27521b216f2a",
  "data": {
    "outputs": [
      {
        "@odata.type": "#Microsoft.Media.JobOutputAsset",
        "assetName": "output-7640689F",
        "error": null,
        "label": "VideoAnalyzerPreset_0",
        "progress": 100,
        "state": "Finished"
      }
    ],
    "previousState": "Processing",
    "state": "Finished",
    "correlationData": {
      "testKey1": "testValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
outputs	Array	Gets the Job outputs.

### JobOutputStateChange

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **JobOutputStateChange** event:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobOutputStateChange",
  "eventTime": "2018-10-12T16:25:56.0242854",
  "id": "dde85f46-b459-4775-b5c7-befe8e32cf90",
  "data": {
    "previousState": "Processing",
    "output": {
      "@odata.type": "#Microsoft.Media.JobOutputAsset",
      "assetName": "output-7640689F",
      "error": null,
      "label": "VideoAnalyzerPreset_0",
      "progress": 100,
      "state": "Finished"
    },
    "jobCorrelationData": {
      "testKey1": "testValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

## **JobOutputScheduled, JobOutputProcessing, JobOutputFinished, JobOutputCanceling, JobOutputCanceled, JobOutputErrored**

For each JobOutput state change, the example schema looks similar to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/<job-id>",
  "eventType": "Microsoft.Media.JobOutputProcessing",
  "eventTime": "2018-10-12T16:12:18.0061141",
  "id": "f1fd5338-1b6c-4e31-83c9-cd7c88d2aedb",
  "data": {
    "previousState": "Scheduled",
    "output": {
      "@odata.type": "#Microsoft.Media.JobOutputAsset",
      "assetName": "output-7640689F",
      "error": null,
      "label": "VideoAnalyzerPreset_0",
      "progress": 0,
      "state": "Processing"
    },
    "jobCorrelationData": {
      "testKey1": "testValue1",
      "testKey2": "testValue2"
    }
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

## **JobOutputProgress**

The example schema looks similar to the following:

```
[{
  "topic": "/subscriptions/<subscription-id>/resourceGroups/belowGroup/providers/Microsoft.Media/mediaServices/<account-name>",
  "subject": "transforms/VideoAnalyzerTransform/jobs/job-5AB6DE32",
  "eventType": "Microsoft.Media.JobOutputProgress",
  "eventTime": "2018-12-10T18:20:12.1514867",
  "id": "00000000-0000-0000-0000-000000000000",
  "data": {
    "jobCorrelationData": {
      "TestKey1": "TestValue1",
      "testKey2": "testValue2"
    },
    "label": "VideoAnalyzerPreset_0",
    "progress": 86
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

## LiveEventConnectionRejected

The following example shows the schema of the **LiveEventConnectionRejected** event:

```
[{
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaServices/<account-name>",
    "subject": "/LiveEvents/MyLiveEvent1",
    "eventType": "Microsoft.Media.LiveEventConnectionRejected",
    "eventTime": "2018-01-16T01:57:26.005121Z",
    "id": "b303db59-d5c1-47eb-927a-3650875fded1",
    "data": {
      "streamId": "Mystream1",
      "ingestUrl": "http://abc.ingest.isml",
      "encoderIp": "118.238.251.xxx",
      "encoderPort": 52859,
      "resultCode": "MPE_INGEST_CODEC_NOT_SUPPORTED"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
}]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>streamId</code>	string	Identifier of the stream or connection. Encoder or customer is responsible to add this ID in the ingest URL.
<code>ingestUrl</code>	string	Ingest URL provided by the live event.
<code>encoderIp</code>	string	IP of the encoder.
<code>encoderPort</code>	string	Port of the encoder from where this stream is coming.

PROPERTY	TYPE	DESCRIPTION
resultCode	string	The reason the connection was rejected. The result codes are listed in the following table.

You can find the error result codes in [live Event error codes](#).

### LiveEventEncoderConnected

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **LiveEventEncoderConnected** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventEncoderConnected",
    "eventTime": "2018-08-07T23:08:09.1710643",
    "id": "<id>",
    "data": {
      "ingestUrl": "http://mle1-amsts03mediaacctgndos-ts031.channel.media.azure-test.net:80/ingest.isml",
      "streamId": "15864-stream0",
      "encoderIp": "131.107.147.xxx",
      "encoderPort": "27485"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
streamId	string	Identifier of the stream or connection. Encoder or customer is responsible for providing this ID in the ingest URL.
ingestUrl	string	Ingest URL provided by the live event.
encoderIp	string	IP of the encoder.
encoderPort	string	Port of the encoder from where this stream is coming.

### LiveEventEncoderDisconnected

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **LiveEventEncoderDisconnected** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventEncoderDisconnected",
    "eventTime": "2018-08-07T23:08:09.1710872",
    "id": "<id>",
    "data": {
      "ingestUrl": "http://mle1-amsts03mediaacctgndos-ts031.channel.media.azure-test.net:80/ingest.isml",
      "streamId": "15864-stream0",
      "encoderIp": "131.107.147.xxx",
      "encoderPort": "27485",
      "resultCode": "S_OK"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>streamId</code>	string	Identifier of the stream or connection. Encoder or customer is responsible to add this ID in the ingest URL.
<code>ingestUrl</code>	string	Ingest URL provided by the live event.
<code>encoderIp</code>	string	IP of the encoder.
<code>encoderPort</code>	string	Port of the encoder from where this stream is coming.
<code>resultCode</code>	string	The reason for the encoder disconnecting. It could be graceful disconnect or from an error. The result codes are listed in the following table.

You can find the error result codes in [live Event error codes](#).

The graceful disconnect result codes are:

RESULT CODE	DESCRIPTION
S_OK	Encoder disconnected successfully.
MPE_CLIENT_TERMINATED_SESSION	Encoder disconnected (RTMP).
MPE_CLIENT_DISCONNECTED	Encoder disconnected (FMP4).
MPI_REST_API_CHANNEL_RESET	Channel reset command is received.
MPI_REST_API_CHANNEL_STOP	Channel stop command received.
MPI_REST_API_CHANNEL_STOP	Channel undergoing maintenance.

RESULT CODE	DESCRIPTION
MPI_STREAM_HIT_EOF	EOF stream is sent by the encoder.

### LiveEventIncomingDataChunkDropped

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **LiveEventIncomingDataChunkDropped** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaServices/<account-name>",
    "subject": "/LiveEvents/MyLiveEvent1",
    "eventType": "Microsoft.Media.LiveEventIncomingDataChunkDropped",
    "eventTime": "2018-01-16T01:57:26.005121Z",
    "id": "03da9c10-fde7-48e1-80d8-49936f2c3e7d",
    "data": {
      "trackType": "Video",
      "trackName": "Video",
      "bitrate": 300000,
      "timestamp": 36656620000,
      "timescale": 10000000,
      "resultCode": "FragmentDrop_OverlapTimestamp"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>trackType</code>	string	Type of the track (Audio / Video).
<code>trackName</code>	string	Name of the track.
<code>bitrate</code>	integer	Bitrate of the track.
<code>timestamp</code>	string	Timestamp of the data chunk dropped.
<code>timescale</code>	string	Timescale of the timestamp.
<code>resultCode</code>	string	Reason of the data chunk drop. <code>FragmentDrop_OverlapTimestamp</code> or <code>FragmentDrop_NonIncreasingTimestamp</code> .

### LiveEventIncomingStreamReceived

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **LiveEventIncomingStreamReceived** event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventIncomingStreamReceived",
    "eventTime": "2018-08-07T23:08:10.5069288Z",
    "id": "7f939a08-320c-47e7-8250-43dcfc04ab4d",
    "data": {
      "ingestUrl": "http://mle1-amsts03mediaacctgndos-ts031.channel.media.azure-test.net:80/ingest.isml/Streams(15864-stream0)15864-stream0",
      "trackType": "video",
      "trackName": "video",
      "bitrate": 2962000,
      "encoderIp": "131.107.147.xxx",
      "encoderPort": "27485",
      "timestamp": "15336831655032322",
      "duration": "20000000",
      "timescale": "10000000"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>trackType</code>	string	Type of the track (Audio / Video).
<code>trackName</code>	string	Name of the track (either provided by the encoder or, in case of RTMP, server generates in <i>TrackType_Bitrate</i> format).
<code>bitrate</code>	integer	Bitrate of the track.
<code>ingestUrl</code>	string	Ingest URL provided by the live event.
<code>encoderIp</code>	string	IP of the encoder.
<code>encoderPort</code>	string	Port of the encoder from where this stream is coming.
<code>timestamp</code>	string	First timestamp of the data chunk received.
<code>timescale</code>	string	Timescale in which timestamp is represented.

## LiveEventIncomingStreamsOutOfSync

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the `LiveEventIncomingStreamsOutOfSync` event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventIncomingStreamsOutOfSync",
    "eventTime": "2018-08-10T02:26:20.6269183Z",
    "id": "b9d38923-9210-4c2b-958f-0054467d4dd7",
    "data": {
      "minLastTimestamp": "319996",
      "typeOfStreamWithMinLastTimestamp": "Audio",
      "maxLastTimestamp": "366000",
      "typeOfStreamWithMaxLastTimestamp": "Video",
      "timescaleOfMinLastTimestamp": "10000000",
      "timescaleOfMaxLastTimestamp": "10000000"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>minLastTimestamp</code>	string	Minimum of last timestamps among all the tracks (audio or video).
<code>typeOfTrackWithMinLastTimestamp</code>	string	Type of the track (audio or video) with minimum last timestamp.
<code>maxLastTimestamp</code>	string	Maximum of all the timestamps among all the tracks (audio or video).
<code>typeOfTrackWithMaxLastTimestamp</code>	string	Type of the track (audio or video) with maximum last timestamp.
<code>timescaleOfMinLastTimestamp</code>	string	Gets the timescale in which "MinLastTimestamp" is represented.
<code>timescaleOfMaxLastTimestamp</code>	string	Gets the timescale in which "MaxLastTimestamp" is represented.

## LiveEventIncomingVideoStreamsOutOfSync

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the `LiveEventIncomingVideoStreamsOutOfSync` event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaServices/<account-name>",
    "subject": "/LiveEvents/LiveEvent1",
    "eventType": "Microsoft.Media.LiveEventIncomingVideoStreamsOutOfSync",
    "eventTime": "2018-01-16T01:57:26.005121Z",
    "id": "6dd4d862-d442-40a0-b9f3-fc14bcf6d750",
    "data": {
      "firstTimestamp": "2162058216",
      "firstDuration": "2000",
      "secondTimestamp": "2162057216",
      "secondDuration": "2000",
      "timescale": "10000000"
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>firstTimestamp</code>	string	Timestamp received for one of the tracks/quality levels of type video.
<code>firstDuration</code>	string	Duration of the data chunk with first timestamp.
<code>secondTimestamp</code>	string	Timestamp received for some other track/quality level of the type video.
<code>secondDuration</code>	string	Duration of the data chunk with second timestamp.
<code>timescale</code>	string	Timescale of timestamps and duration.

## LiveEventIngestHeartbeat

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the `LiveEventIngestHeartbeat` event:

```
[
  {
    "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",
    "subject": "liveEvent/mle1",
    "eventType": "Microsoft.Media.LiveEventIngestHeartbeat",
    "eventTime": "2018-08-07T23:17:57.4610506",
    "id": "7f450938-491f-41e1-b06f-c6cd3965d786",
    "data": {
      "trackType": "audio",
      "trackName": "audio",
      "bitrate": 160000,
      "incomingBitrate": 155903,
      "lastTimestamp": "15336837535253637",
      "timescale": "10000000",
      "overlapCount": 0,
      "discontinuityCount": 0,
      "nonincreasingCount": 0,
      "unexpectedBitrate": false,
      "state": "Running",
      "healthy": true
    },
    "dataVersion": "1.0",
    "metadataVersion": "1"
  }
]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>trackType</code>	string	Type of the track (Audio / Video).
<code>trackName</code>	string	Name of the track (either provided by the encoder or, in case of RTMP, server generates in <code>TrackType_Bitrate</code> format).
<code>bitrate</code>	integer	Bitrate of the track.
<code>incomingBitrate</code>	integer	Calculated bitrate based on data chunks coming from encoder.
<code>lastTimestamp</code>	string	Latest timestamp received for a track in last 20 seconds.
<code>timescale</code>	string	Timescale in which timestamps are expressed.
<code>overlapCount</code>	integer	Number of data chunks had overlapped timestamps in last 20 seconds.
<code>discontinuityCount</code>	integer	Number of discontinuities observed in last 20 seconds.
<code>nonIncreasingCount</code>	integer	Number of data chunks with timestamps in the past were received in last 20 seconds.

PROPERTY	TYPE	DESCRIPTION
<code>unexpectedBitrate</code>	bool	If expected and actual bitrates differ by more than allowed limit in last 20 seconds. It's true if and only if, <code>incomingBitrate &gt;= 2 * bitrate</code> OR <code>incomingBitrate &lt;= bitrate / 2</code> OR <code>IncomingBitrate = 0</code> .
<code>state</code>	string	State of the live event.
<code>healthy</code>	bool	Indicates whether ingest is healthy based on the counts and flags. Healthy is true if <code>overlapCount = 0</code> && <code>discontinuityCount = 0</code> && <code>nonIncreasingCount = 0</code> && <code>unexpectedBitrate = false</code> .

## LiveEventTrackDiscontinuityDetected

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of the **LiveEventTrackDiscontinuityDetected** event:

```
[  
 {  
   "topic": "/subscriptions/<subscription-id>/resourceGroups/<rg-name>/providers/Microsoft.Media/mediaservices/<account-name>",  
   "subject": "liveEvent/mle1",  
   "eventType": "Microsoft.Media.LiveEventTrackDiscontinuityDetected",  
   "eventTime": "2018-08-07T23:18:06.1270405Z",  
   "id": "5f4c510d-5be7-4bef-baf0-64b828be9c9b",  
   "data": {  
     "trackName": "video",  
     "previousTimestamp": "15336837615032322",  
     "trackType": "video",  
     "bitrate": 2962000,  
     "newTimestamp": "15336837619774273",  
     "discontinuityGap": "575284",  
     "timescale": "10000000"  
   },  
   "dataVersion": "1.0",  
   "metadataVersion": "1"  
 }  
 ]
```

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>trackType</code>	string	Type of the track (Audio / Video).
<code>trackName</code>	string	Name of the track (either provided by the encoder or, in case of RTMP, server generates in <code>TrackType_Bitrate</code> format).
<code>bitrate</code>	integer	Bitrate of the track.

PROPERTY	TYPE	DESCRIPTION
previousTimestamp	string	Timestamp of the previous fragment.
newTimestamp	string	Timestamp of the current fragment.
discontinuityGap	string	Gap between above two timestamps.
timescale	string	Timescale in which both timestamp and discontinuity gap are represented.

## Common event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
topic	string	The event grid topic. This property has the resource ID for the Media Services account.
subject	string	The resource path for the Media Services channel under the Media Services account. Concatenating the topic and subject give you the resource ID for the job.
eventType	string	One of the registered event types for this event source. For example, "Microsoft.Media.JobStateChange".
eventTime	string	The time the event is generated based on the provider's UTC time.
id	string	Unique identifier for the event.
data	object	Media Services event data.
dataVersion	string	The schema version of the data object. The publisher defines the schema version.
metadataVersion	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

## Next steps

[Register for job state change events](#)

## See also

- EventGrid .NET SDK that includes Media Service events
- Definitions of Media Services events
- Live Event error codes

# Azure Policy as an Event Grid source

3/29/2021 • 3 minutes to read • [Edit Online](#)

This article provides the properties and schema for [Azure Policy](#) events. For an introduction to event schemas, see [Azure Event Grid event schema](#). It also gives you a list of quick starts and tutorials to use Azure Policy as an event source.

## Available event types

Azure Policy emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.PolicyInsights.PolicyStateCreated	Raised when a policy compliance state is created.
Microsoft.PolicyInsights.PolicyStateChanged	Raised when a policy compliance state is changed.
Microsoft.PolicyInsights.PolicyStateDeleted	Raised when a policy compliance state is deleted.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of a policy state created event:

```
[{  
    "id": "5829794FCB5075FCF585476619577B5A5A30E52C84842CBD4E2AD73996714C4C",  
    "topic": "/subscriptions/<SubscriptionID>",  
    "subject":  
        "/subscriptions/<SubscriptionID>/resourceGroups/<ResourceGroup>/providers/<ProviderNamespace>/<ResourceType>  
        /<ResourceName>",  
        "data": {  
            "timestamp": "2021-03-27T18:37:42.4496956Z",  
            "policyAssignmentId": "<policy-assignment-  
            scope>/providers/microsoft.authorization/policyassignments/<policy-assignment-name>",  
            "policyDefinitionId": "<policy-definition-  
            scope>/providers/microsoft.authorization/policydefinitions/<policy-definition-name>",  
            "policyDefinitionReferenceId": "",  
            "complianceState": "NonCompliant",  
            "subscriptionId": "<subscription-id>",  
            "complianceReasonCode": ""  
        },  
        "eventType": "Microsoft.PolicyInsights.PolicyStateCreated",  
        "eventTime": "2021-03-27T18:37:42.5241536Z",  
        "dataVersion": "1",  
        "metadataVersion": "1"  
    }]  
]
```

The schema for a policy state changed event is similar:

```
[{  
    "id": "5829794FCB5075FCF585476619577B5A5A30E52C84842CBD4E2AD73996714C4C",  
    "topic": "/subscriptions/<SubscriptionID>",  
    "subject":  
        "/subscriptions/<SubscriptionID>/resourceGroups/<ResourceGroup>/providers/<ProviderNamespace>/<ResourceType>  
        /<ResourceName>",  
        "data": {  
            "timestamp": "2021-03-27T18:37:42.4496956Z",  
            "policyAssignmentId": "<policy-assignment-  
            scope>/providers/microsoft.authorization/policyassignments/<policy-assignment-name>",  
            "policyDefinitionId": "<policy-definition-  
            scope>/providers/microsoft.authorization/policydefinitions/<policy-definition-name>",  
            "policyDefinitionReferenceId": "",  
            "complianceState": "NonCompliant",  
            "subscriptionId": "<subscription-id>",  
            "complianceReasonCode": ""  
        },  
        "eventType": "Microsoft.PolicyInsights.PolicyStateChanged",  
        "eventTime": "2021-03-27T18:37:42.5241536Z",  
        "dataVersion": "1",  
        "metadataVersion": "1"  
    }]  
]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
topic	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.

PROPERTY	TYPE	DESCRIPTION
<code>subject</code>	string	The fully qualified ID of the resource that the compliance state change is for, including the resource name and resource type. Uses the format, <code>/subscriptions/&lt;SubscriptionID&gt;/resourceGroups/&lt;ResourceGroup&gt;/providers/&lt;Provide</code>
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Azure Policy event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>timestamp</code>	string	The time (in UTC) that the resource was scanned by Azure Policy. For ordering events, use this property instead of the top-level <code>eventTime</code> or <code>time</code> properties.
<code>policyAssignmentId</code>	string	The resource ID of the policy assignment.
<code>policyDefinitionId</code>	string	The resource ID of the policy definition.
<code>policyDefinitionReferenceId</code>	string	The reference ID for the policy definition inside the initiative definition, if the policy assignment is for an initiative. May be empty.
<code>complianceState</code>	string	The compliance state of the resource with respect to the policy assignment.
<code>subscriptionId</code>	string	The subscription ID of the resource.
<code>complianceReasonCode</code>	string	The compliance reason code. May be empty.

## Next steps

- For a walkthrough routing Azure Policy state change events, see [Use Event Grid for policy state change notifications](#).
- For an overview of integrating Azure Policy with Event Grid, see [React to Azure Policy events by using Event Grid](#).
- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Azure resource group as an Event Grid source

5/11/2021 • 8 minutes to read • [Edit Online](#)

This article provides the properties and schema for resource group events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

Azure subscriptions and resource groups emit the same event types. The event types are related to resource changes or actions. The primary difference is that resource groups emit events for resources within the resource group, and Azure subscriptions emit events for resources across the subscription.

Resource events are created for PUT, PATCH, POST, and DELETE operations that are sent to `management.azure.com`. GET operations don't create events. Operations sent to the data plane (like `myaccount.blob.core.windows.net`) don't create events. The action events provide event data for operations like listing the keys for a resource.

When you subscribe to events for a resource group, your endpoint receives all events for that resource group. The events can include event you want to see, such as updating a virtual machine, but also events that maybe aren't important to you, such as writing a new entry in the deployment history. You can receive all events at your endpoint and write code that processes the events you want to handle. Or, you can set a filter when creating the event subscription.

To programmatically handle events, you can sort events by looking at the `operationName` value. For example, your event endpoint might only process events for operations that are equal to `Microsoft.Compute/virtualMachines/write` or `Microsoft.Storage/storageAccounts/write`.

The event subject is the resource ID of the resource that is the target of the operation. To filter events for a resource, provide that resource ID when creating the event subscription. To filter by a resource type, use a value in following format:

`/subscriptions/<subscription-id>/resourcegroups/<resource-group>/providers/Microsoft.Compute/virtualMachines`

## Available event types

Resource groups emit management events from Azure Resource Manager, such as when a VM is created or a storage account is deleted.

EVENT TYPE	DESCRIPTION
<code>Microsoft.Resources.ResourceActionCancel</code>	Raised when action on resource is canceled.
<code>Microsoft.Resources.ResourceActionFailure</code>	Raised when action on resource fails.
<code>Microsoft.Resources.ResourceActionSuccess</code>	Raised when action on resource succeeds.
<code>Microsoft.Resources.ResourceDeleteCancel</code>	Raised when delete operation is canceled. This event happens when a template deployment is canceled.
<code>Microsoft.Resources.ResourceDeleteFailure</code>	Raised when delete operation fails.
<code>Microsoft.Resources.ResourceDeleteSuccess</code>	Raised when delete operation succeeds.
<code>Microsoft.Resources.ResourceWriteCancel</code>	Raised when create or update operation is canceled.

EVENT TYPE	DESCRIPTION
Microsoft.Resources.ResourceWriteFailure	Raised when create or update operation fails.
Microsoft.Resources.ResourceWriteSuccess	Raised when create or update operation succeeds.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema for a `ResourceWriteSuccess` event. The same schema is used for `ResourceWriteFailure` and `ResourceWriteCancel` events with different values for `eventType`.

```
[{
  "subject": "/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
  "eventType": "Microsoft.Resources.ResourceWriteSuccess",
  "eventTime": "2018-07-19T18:38:04.611735Z",
  "id": "4db48cba-50a2-455a-93b4-de41a3b5b7f6",
  "data": {
    "authorization": {
      "scope": "/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
      "action": "Microsoft.Storage/storageAccounts/write",
      "evidence": {
        "role": "Subscription Admin"
      }
    },
    "claims": {
      "aud": "{audience-claim}",
      "iss": "{issuer-claim}",
      "iat": "{issued-at-claim}",
      "nbf": "{not-before-claim}",
      "exp": "{expiration-claim}",
      "_claim_names": "{\"groups\": \"src1\"}",
      "_claim_sources": "{\"src1\": {\"endpoint\": \"{URI}\"}}",
      "http://schemas.microsoft.com/claims/authnclassreference": "1",
      "aio": "{token}",
      "http://schemas.microsoft.com/claims/authnmethodsreferences": "rsa,mfa",
      "appid": "{ID}",
      "appidacr": "2",
      "http://schemas.microsoft.com/2012/01/devicecontext/claims/identifier": "{ID}",
      "e_exp": "{expiration}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname": "{last-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "{first-name}",
      "ipaddr": "{IP-address}",
      "name": "{full-name}",
      "http://schemas.microsoft.com/identity/claims/objectidentifier": "{ID}",
      "onprem_sid": "{ID}",
      "puid": "{ID}",
      "http://schemas.microsoft.com/identity/claims/scope": "user_impersonation",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "{ID}",
      "http://schemas.microsoft.com/identity/claims/tenantid": "{ID}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "{user-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn": "{user-name}",
      "uti": "{ID}",
      "ver": "1.0"
    },
    "correlationId": "{ID}",
    "resourceProvider": "Microsoft.Storage",
    "resourceUri": "/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
    "operationName": "Microsoft.Storage/storageAccounts/write",
    "status": "Succeeded",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}"
  },
  "dataVersion": "2",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}"
}]
}
```

The following example shows the schema for a **ResourceDeleteSuccess** event. The same schema is used for **ResourceDeleteFailure** and **ResourceDeleteCancel** events with different values for `eventType`.

```
[{
  "subject": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
  "eventType": "Microsoft.Resources.ResourceDeleteSuccess",
  "eventTime": "2018-07-19T19:24:12.763881Z",
  "id": "19a69642-1aad-4a96-a5ab-8d05494513ce",
  "data": {
    "authorization": {
      "scope": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
      "action": "Microsoft.Storage/storageAccounts/delete",
      "evidence": {
        "role": "Subscription Admin"
      }
    },
    "claims": {
      "aud": "{audience-claim}",
      "iss": "{issuer-claim}",
      "iat": "{issued-at-claim}",
      "nbf": "{not-before-claim}",
      "exp": "{expiration-claim}",
      "_claim_names": "{\"groups\": \"src1\"}",
      "_claim_sources": "{\"src1\": {\"endpoint\": \"{URI}\"}}",
      "http://schemas.microsoft.com/claims/authnclassreference": "1",
      "aio": "{token}",
      "http://schemas.microsoft.com/claims/authnmethodsreferences": "rsa,mfa",
      "appid": "{ID}",
      "appidacr": "2",
      "http://schemas.microsoft.com/2012/01/devicecontext/claims/identifier": "{ID}",
      "e_exp": "262800",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname": "{last-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "{first-name}",
      "ipaddr": "{IP-address}",
      "name": "{full-name}",
      "http://schemas.microsoft.com/identity/claims/objectidentifier": "{ID}",
      "onprem_sid": "{ID}",
      "puid": "{ID}",
      "http://schemas.microsoft.com/identity/claims/scope": "user_impersonation",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "{ID}",
      "http://schemas.microsoft.com/identity/claims/tenantid": "{ID}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "{user-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn": "{user-name}",
      "uti": "{ID}",
      "ver": "1.0"
    },
    "correlationId": "{ID}",
    "httpRequest": {
      "clientRequestId": "{ID}",
      "clientIpAddress": "{IP-address}",
      "method": "DELETE",
      "url": "https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}?api-version=2018-02-01"
    },
    "resourceProvider": "Microsoft.Storage",
    "resourceUri": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
    "operationName": "Microsoft.Storage/storageAccounts/delete",
    "status": "Succeeded",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}"
  },
  "dataVersion": "2",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}"
}]
```

The following example shows the schema for a **ResourceActionSuccess** event. The same schema is used for **ResourceActionFailure** and **ResourceActionCancel** events with different values for `eventType`.

```
[{
  "subject": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey",
  "eventType": "Microsoft.Resources.ResourceActionSuccess",
  "eventTime": "2018-10-08T22:46:22.6022559Z",
  "id": "{ID}",
  "data": {
    "authorization": {
      "scope": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey",
      "action": "Microsoft.EventHub/namespaces/AuthorizationRules/listKeys/action",
      "evidence": {
        "role": "Contributor",
        "roleAssignmentScope": "/subscriptions/{subscription-id}",
        "roleAssignmentId": "{ID}",
        "roleDefinitionId": "{ID}",
        "principalId": "{ID}",
        "principalType": "ServicePrincipal"
      }
    },
    "claims": {
      "aud": "{audience-claim}",
      "iss": "{issuer-claim}",
      "iat": "{issued-at-claim}",
      "nbf": "{not-before-claim}",
      "exp": "{expiration-claim}",
      "aio": "{token}",
      "appid": "{ID}",
      "appidacr": "2",
      "http://schemas.microsoft.com/identity/claims/identityprovider": "{URL}",
      "http://schemas.microsoft.com/identity/claims/objectidentifier": "{ID}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "{ID}",
      "http://schemas.microsoft.com/identity/claims/tenantid": "{ID}",
      "uti": "{ID}",
      "ver": "1.0"
    },
    "correlationId": "{ID}",
    "httpRequest": {
      "clientRequestId": "{ID}",
      "clientIpAddress": "{IP-address}",
      "method": "POST",
      "url": "https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey/listKeys?api-version=2017-04-01"
    },
    "resourceProvider": "Microsoft.EventHub",
    "resourceUri": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey",
    "operationName": "Microsoft.EventHub/namespaces/AuthorizationRules/listKeys/action",
    "status": "Succeeded",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}"
  },
  "dataVersion": "2",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}"
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
topic	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
subject	string	Publisher-defined path to the event subject.
eventType	string	One of the registered event types for this event source.
eventTime	string	The time the event is generated based on the provider's UTC time.
id	string	Unique identifier for the event.
data	object	Resource group event data.
dataVersion	string	The schema version of the data object. The publisher defines the schema version.
metadataVersion	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
authorization	object	The requested authorization for the operation.
claims	object	The properties of the claims. For more information, see <a href="#">JWT specification</a> .
correlationId	string	An operation ID for troubleshooting.
httpRequest	object	The details of the operation. This object is only included when updating an existing resource or deleting a resource.
resourceProvider	string	The resource provider for the operation.
resourceUri	string	The URI of the resource in the operation.
operationName	string	The operation that was taken.

PROPERTY	TYPE	DESCRIPTION
<code>status</code>	string	The status of the operation.
<code>subscriptionId</code>	string	The subscription ID of the resource.
<code>tenantId</code>	string	The tenant ID of the resource.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Tutorial: monitor virtual machine changes with Azure Event Grid and Logic Apps</a>	A logic app monitors changes to a virtual machine and sends emails about those changes.
<a href="#">Azure CLI: subscribe to events for a resource group</a>	Sample script that subscribes to events for a resource group. It sends events to a WebHook.
<a href="#">Azure CLI: subscribe to events for a resource group and filter for a resource</a>	Sample script that subscribes to events for a resource group and filters events for one resource.
<a href="#">PowerShell: subscribe to events for a resource group</a>	Sample script that subscribes to events for a resource group. It sends events to a WebHook.
<a href="#">PowerShell: subscribe to events for a resource group and filter for a resource</a>	Sample script that subscribes to events for a resource group and filters events for one resource.
<a href="#">Resource Manager template: resource subscription</a>	Subscribes to events for an Azure subscription or resource group. It sends events to a WebHook.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Azure Service Bus as an Event Grid source

3/5/2021 • 5 minutes to read • [Edit Online](#)

This article provides the properties and schema for Service Bus events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

Service Bus emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.ServiceBus.ActiveMessagesAvailableWithNoListeners	Raised when there are active messages in a Queue or Subscription and no receivers listening.
Microsoft.ServiceBus.DeadletterMessagesAvailableWithNoListeners	Raised when there are active messages in a Dead Letter Queue and no active listeners.
Microsoft.ServiceBus.ActiveMessagesAvailablePeriodicNotifications	Raised periodically if there are active messages in a Queue or Subscription, even if there are active listeners on that specific Queue or Subscription.
Microsoft.ServiceBus.DeadletterMessagesAvailablePeriodicNotifications	Raised periodically if there are messages in the dead-letter entity of a Queue or Subscription, even if there are active listeners on the dead-letter entity of that specific Queue or Subscription.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

### Active Messages Available With No Listeners

This event is generated if you have active messages in a queue or a subscription and there are no receivers listening.

```
[{
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{your-
    rg}/providers/Microsoft.ServiceBus/namespaces/{your-service-bus-namespace}",
  "subject": "topics/{your-service-bus-topic}/subscriptions/{your-service-bus-subscription}",
  "eventType": "Microsoft.ServiceBus.ActiveMessagesAvailableWithNoListeners",
  "eventTime": "2018-02-14T05:12:53.4133526Z",
  "id": "dede87b0-3656-419c-acaf-70c95ddc60f5",
  "data": {
    "namespaceName": "YOUR SERVICE BUS NAMESPACE WILL SHOW HERE",
    "requestUri": "https://'{your-service-bus-namespace}.servicebus.windows.net/{your-
      topic}/subscriptions/{your-service-bus-subscription}/messages/head",
    "entityType": "subscriber",
    "queueName": "QUEUE NAME IF QUEUE",
    "topicName": "TOPIC NAME IF TOPIC",
    "subscriptionName": "SUBSCRIPTION NAME"
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

#### **Deadletter Messages Available With No Listeners**

The schema for a dead letter queue event is similar. You get at least one event per dead-letter queue that has messages and no active receivers.

```
[{
  "topic": "/subscriptions/{subscription-id}/resourcegroups/{your-
    rg}/providers/Microsoft.ServiceBus/namespaces/{your-service-bus-namespace}",
  "subject": "topics/{your-service-bus-topic}/subscriptions/{your-service-bus-subscription}",
  "eventType": "Microsoft.ServiceBus.DeadletterMessagesAvailableWithNoListeners",
  "eventTime": "2018-02-14T05:12:53.4133526Z",
  "id": "dede87b0-3656-419c-acaf-70c95ddc60f5",
  "data": {
    "namespaceName": "YOUR SERVICE BUS NAMESPACE WILL SHOW HERE",
    "requestUri": "https://'{your-service-bus-namespace}.servicebus.windows.net/{your-
      topic}/subscriptions/{your-service-bus-subscription}/$deadletterqueue/messages/head",
    "entityType": "subscriber",
    "queueName": "QUEUE NAME IF QUEUE",
    "topicName": "TOPIC NAME IF TOPIC",
    "subscriptionName": "SUBSCRIPTION NAME"
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

#### **Active Messages Available Periodic Notifications**

This event is generated periodically if you have active messages on the specific queue or subscription, even if there are active listeners on that specific queue or subscription.

```
[{
  "topic": "/subscriptions/<subscription
id>/resourcegroups/DemoGroup/providers/Microsoft.ServiceBus/namespaces/<YOUR SERVICE BUS NAMESPACE WILL SHOW
HERE>",
  "subject": "topics/<service bus topic>/subscriptions/<service bus subscription>",
  "eventType": "Microsoft.ServiceBus.ActiveMessagesAvailablePeriodicNotifications",
  "eventTime": "2018-02-14T05:12:53.4133526Z",
  "id": "dede87b0-3656-419c-acaf-70c95ddc60f5",
  "data": {
    "namespaceName": "YOUR SERVICE BUS NAMESPACE WILL SHOW HERE",
    "requestUri": "https://YOUR-SERVICE-BUS-NAMESPACE-WILL-SHOW-HERE.servicebus.windows.net/TOPIC-
NAME/subscriptions/SUBSCRIPTIONNAME/$deadletterqueue/messages/head",
    "entityType": "subscriber",
    "queueName": "QUEUE NAME IF QUEUE",
    "topicName": "TOPIC NAME IF TOPIC",
    "subscriptionName": "SUBSCRIPTION NAME"
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

#### Deadletter Messages Available Periodic Notifications

This event is generated periodically if you have deadletter messages on the specific queue or subscription, even if there are active listeners on the deadletter entity of that specific queue or subscription.

```
[{
  "topic": "/subscriptions/<subscription
id>/resourcegroups/DemoGroup/providers/Microsoft.ServiceBus/namespaces/<YOUR SERVICE BUS NAMESPACE WILL SHOW
HERE>",
  "subject": "topics/<service bus topic>/subscriptions/<service bus subscription>",
  "eventType": "Microsoft.ServiceBus.DeadletterMessagesAvailablePeriodicNotifications",
  "eventTime": "2018-02-14T05:12:53.4133526Z",
  "id": "dede87b0-3656-419c-acaf-70c95ddc60f5",
  "data": {
    "namespaceName": "YOUR SERVICE BUS NAMESPACE WILL SHOW HERE",
    "requestUri": "https://YOUR-SERVICE-BUS-NAMESPACE-WILL-SHOW-HERE.servicebus.windows.net/TOPIC-
NAME/subscriptions/SUBSCRIPTIONNAME/$deadletterqueue/messages/head",
    "entityType": "subscriber",
    "queueName": "QUEUE NAME IF QUEUE",
    "topicName": "TOPIC NAME IF TOPIC",
    "subscriptionName": "SUBSCRIPTION NAME"
  },
  "dataVersion": "1",
  "metadataVersion": "1"
}]
```

#### Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field is not writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.

PROPERTY	TYPE	DESCRIPTION
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Blob storage event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>namespaceName</code>	string	The Service Bus namespace the resource exists in.
<code>requestUri</code>	string	The URI to the specific queue or subscription emitting the event.
<code>entityType</code>	string	The type of Service Bus entity emitting events (queue or subscription).
<code>queueName</code>	string	The queue with active messages if subscribing to a queue. Value null if using topics / subscriptions.
<code>topicName</code>	string	The topic the Service Bus subscription with active messages belongs to. Value null if using a queue.
<code>subscriptionName</code>	string	The Service Bus subscription with active messages. Value null if using a queue.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Tutorial: Azure Service Bus to Azure Event Grid integration examples</a>	Event Grid sends messages from Service Bus topic to function app and logic app.
<a href="#">Azure Service Bus to Event Grid integration</a>	Overview of integrating Service Bus with Event Grid.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).
- For details on using Azure Event Grid with Service Bus, see the [Service Bus to Event Grid integration overview](#).
- Try [receiving Service Bus events with Functions or Logic Apps](#).

# Azure Event Grid event schema for SignalR Service

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article provides the properties and schema for SignalR Service events. For an introduction to event schemas, see [Azure Event Grid event schema](#). It also gives you a list of quick starts and tutorials to use Azure SignalR as an event source.

## Available event types

SignalR Service emits the following event types:

EVENT TYPE	DESCRIPTION
Microsoft.SignalRService.ClientConnectionConnected	Raised when a client connection connected.
Microsoft.SignalRService.ClientConnectionDisconnected	Raised when a client connection disconnected.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema of a client connection connected event:

```
[{  
    "topic": "/subscriptions/{subscription-id}/resourceGroups/signalr-  
rg/providers/Microsoft.SignalRService/SignalR/signalr-resource",  
    "subject": "/hub/chat",  
    "eventType": "Microsoft.SignalRService.ClientConnectionConnected",  
    "eventTime": "2019-06-10T18:41:00.9584103Z",  
    "id": "831e1650-001e-001b-66ab-eeb76e069631",  
    "data": {  
        "timestamp": "2019-06-10T18:41:00.9584103Z",  
        "hubName": "chat",  
        "connectionId": "crH0uxVSvP61p5wkFY1x1A",  
        "userId": "user-eymwo23"  
    },  
    "dataVersion": "1.0",  
    "metadataVersion": "1"  
}]
```

The schema for a client connection disconnected event is similar:

```
[{
  "topic": "/subscriptions/{subscription-id}/resourceGroups/signalr-
    rg/providers/Microsoft.SignalRService/SignalR/signalr-resource",
  "subject": "/hub/chat",
  "eventType": "Microsoft.SignalRService.ClientConnectionDisconnected",
  "eventTime": "2019-06-10T18:41:00.9584103Z",
  "id": "831e1650-001e-001b-66ab-eeb76e069631",
  "data": {
    "timestamp": "2019-06-10T18:41:00.9584103Z",
    "hubName": "chat",
    "connectionId": "crH0uxVSvP61p5wkFY1x1A",
    "userId": "user-eymwo23",
    "errorMessage": "Internal server error."
  },
  "dataVersion": "1.0",
  "metadataVersion": "1"
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	SignalR Service event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
----------	------	-------------

PROPERTY	TYPE	DESCRIPTION
<code>timestamp</code>	string	The time the event is generated based on the provider's UTC time.
<code>hubName</code>	string	The hub that the client connection belongs to.
<code>connectionId</code>	string	The unique identifier for the client connection.
<code>userId</code>	string	The user identifier defined in claim.
<code>errorMessage</code>	string	The error that causes the connection disconnected.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">React to Azure SignalR Service events by using Event Grid</a>	Overview of integrating Azure SignalR Service with Event Grid.
<a href="#">How to send Azure SignalR Service events to Event Grid</a>	Shows how to send Azure SignalR Service events to an application through Event Grid.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Azure subscription as an Event Grid source

3/5/2021 • 8 minutes to read • [Edit Online](#)

This article provides the properties and schema for Azure subscription events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

Azure subscriptions and resource groups emit the same event types. The event types are related to resource changes or actions. The primary difference is that resource groups emit events for resources within the resource group, and Azure subscriptions emit events for resources across the subscription.

Resource events are created for PUT, PATCH, POST, and DELETE operations that are sent to `management.azure.com`. GET operations don't create events. Operations sent to the data plane (like `myaccount.blob.core.windows.net`) don't create events. The action events provide event data for operations like listing the keys for a resource.

When you subscribe to events for an Azure subscription, your endpoint receives all events for that subscription. The events can include event you want to see, such as updating a virtual machine, but also events that maybe aren't important to you, such as writing a new entry in the deployment history. You can receive all events at your endpoint and write code that processes the events you want to handle. Or, you can set a filter when creating the event subscription.

To programmatically handle events, you can sort events by looking at the `operationName` value. For example, your event endpoint might only process events for operations that are equal to `Microsoft.Compute/virtualMachines/write` or `Microsoft.Storage/storageAccounts/write`.

The event subject is the resource ID of the resource that is the target of the operation. To filter events for a resource, provide that resource ID when creating the event subscription. To filter by a resource type, use a value in following format:

`/subscriptions/<subscription-id>/resourcegroups/<resource-group>/providers/Microsoft.Compute/virtualMachines`

## Available event types

Azure subscriptions emit management events from Azure Resource Manager, such as when a VM is created or a storage account is deleted.

EVENT TYPE	DESCRIPTION
<code>Microsoft.Resources.ResourceActionCancel</code>	Raised when action on resource is canceled.
<code>Microsoft.Resources.ResourceActionFailure</code>	Raised when action on resource fails.
<code>Microsoft.Resources.ResourceActionSuccess</code>	Raised when action on resource succeeds.
<code>Microsoft.Resources.ResourceDeleteCancel</code>	Raised when delete operation is canceled. This event happens when a template deployment is canceled.
<code>Microsoft.Resources.ResourceDeleteFailure</code>	Raised when delete operation fails.
<code>Microsoft.Resources.ResourceDeleteSuccess</code>	Raised when delete operation succeeds.
<code>Microsoft.Resources.ResourceWriteCancel</code>	Raised when create or update operation is canceled.

EVENT TYPE	DESCRIPTION
Microsoft.Resources.ResourceWriteFailure	Raised when create or update operation fails.
Microsoft.Resources.ResourceWriteSuccess	Raised when create or update operation succeeds.

## Example event

- [Event Grid event schema](#)
- [Cloud event schema](#)

The following example shows the schema for a `ResourceWriteSuccess` event. The same schema is used for `ResourceWriteFailure` and `ResourceWriteCancel` events with different values for `eventType`.

```
[{
  "subject": "/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
  "eventType": "Microsoft.Resources.ResourceWriteSuccess",
  "eventTime": "2018-07-19T18:38:04.611735Z",
  "id": "4db48cba-50a2-455a-93b4-de41a3b5b7f6",
  "data": {
    "authorization": {
      "scope": "/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
      "action": "Microsoft.Storage/storageAccounts/write",
      "evidence": {
        "role": "Subscription Admin"
      }
    },
    "claims": {
      "aud": "{audience-claim}",
      "iss": "{issuer-claim}",
      "iat": "{issued-at-claim}",
      "nbf": "{not-before-claim}",
      "exp": "{expiration-claim}",
      "_claim_names": "{\"groups\": \"src1\"}",
      "_claim_sources": "{\"src1\": {\"endpoint\": \"{URI}\"}}",
      "http://schemas.microsoft.com/claims/authnclassreference": "1",
      "aio": "{token}",
      "http://schemas.microsoft.com/claims/authnmethodsreferences": "rsa,mfa",
      "appid": "{ID}",
      "appidacr": "2",
      "http://schemas.microsoft.com/2012/01/devicecontext/claims/identifier": "{ID}",
      "e_exp": "{expiration}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname": "{last-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "{first-name}",
      "ipaddr": "{IP-address}",
      "name": "{full-name}",
      "http://schemas.microsoft.com/identity/claims/objectidentifier": "{ID}",
      "onprem_sid": "{ID}",
      "puid": "{ID}",
      "http://schemas.microsoft.com/identity/claims/scope": "user_impersonation",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "{ID}",
      "http://schemas.microsoft.com/identity/claims/tenantid": "{ID}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "{user-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn": "{user-name}",
      "uti": "{ID}",
      "ver": "1.0"
    },
    "correlationId": "{ID}",
    "resourceProvider": "Microsoft.Storage",
    "resourceUri": "/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
    "operationName": "Microsoft.Storage/storageAccounts/write",
    "status": "Succeeded",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}"
  },
  "dataVersion": "2",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}"
}]
}
```

The following example shows the schema for a **ResourceDeleteSuccess** event. The same schema is used for **ResourceDeleteFailure** and **ResourceDeleteCancel** events with different values for `eventType`.

```
[{
  "subject": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
  "eventType": "Microsoft.Resources.ResourceDeleteSuccess",
  "eventTime": "2018-07-19T19:24:12.763881Z",
  "id": "19a69642-1aad-4a96-a5ab-8d05494513ce",
  "data": {
    "authorization": {
      "scope": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
      "action": "Microsoft.Storage/storageAccounts/delete",
      "evidence": {
        "role": "Subscription Admin"
      }
    },
    "claims": {
      "aud": "{audience-claim}",
      "iss": "{issuer-claim}",
      "iat": "{issued-at-claim}",
      "nbf": "{not-before-claim}",
      "exp": "{expiration-claim}",
      "_claim_names": "{\"groups\": \"src1\"}",
      "_claim_sources": "{\"src1\": {\"endpoint\": \"{URI}\"}}",
      "http://schemas.microsoft.com/claims/authnclassreference": "1",
      "aio": "{token}",
      "http://schemas.microsoft.com/claims/authnmethodsreferences": "rsa,mfa",
      "appid": "{ID}",
      "appidacr": "2",
      "http://schemas.microsoft.com/2012/01/devicecontext/claims/identifier": "{ID}",
      "e_exp": "262800",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/surname": "{last-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/givenname": "{first-name}",
      "ipaddr": "{IP-address}",
      "name": "{full-name}",
      "http://schemas.microsoft.com/identity/claims/objectidentifier": "{ID}",
      "onprem_sid": "{ID}",
      "puid": "{ID}",
      "http://schemas.microsoft.com/identity/claims/scope": "user_impersonation",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "{ID}",
      "http://schemas.microsoft.com/identity/claims/tenantid": "{ID}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name": "{user-name}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/upn": "{user-name}",
      "uti": "{ID}",
      "ver": "1.0"
    },
    "correlationId": "{ID}",
    "httpRequest": {
      "clientRequestId": "{ID}",
      "clientIpAddress": "{IP-address}",
      "method": "DELETE",
      "url": "https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}?api-version=2018-02-01"
    },
    "resourceProvider": "Microsoft.Storage",
    "resourceUri": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}",
    "operationName": "Microsoft.Storage/storageAccounts/delete",
    "status": "Succeeded",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}"
  },
  "dataVersion": "2",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}"
}]
```

The following example shows the schema for a **ResourceActionSuccess** event. The same schema is used for **ResourceActionFailure** and **ResourceActionCancel** events with different values for `eventType`.

```
[{
  "subject": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey",
  "eventType": "Microsoft.Resources.ResourceActionSuccess",
  "eventTime": "2018-10-08T22:46:22.6022559Z",
  "id": "{ID}",
  "data": {
    "authorization": {
      "scope": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey",
      "action": "Microsoft.EventHub/namespaces/AuthorizationRules/listKeys/action",
      "evidence": {
        "role": "Contributor",
        "roleAssignmentScope": "/subscriptions/{subscription-id}",
        "roleAssignmentId": "{ID}",
        "roleDefinitionId": "{ID}",
        "principalId": "{ID}",
        "principalType": "ServicePrincipal"
      }
    },
    "claims": {
      "aud": "{audience-claim}",
      "iss": "{issuer-claim}",
      "iat": "{issued-at-claim}",
      "nbf": "{not-before-claim}",
      "exp": "{expiration-claim}",
      "aio": "{token}",
      "appid": "{ID}",
      "appidacr": "2",
      "http://schemas.microsoft.com/identity/claims/identityprovider": "{URL}",
      "http://schemas.microsoft.com/identity/claims/objectidentifier": "{ID}",
      "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier": "{ID}",
      "http://schemas.microsoft.com/identity/claims/tenantid": "{ID}",
      "uti": "{ID}",
      "ver": "1.0"
    },
    "correlationId": "{ID}",
    "httpRequest": {
      "clientRequestId": "{ID}",
      "clientIpAddress": "{IP-address}",
      "method": "POST",
      "url": "https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey/listKeys?api-version=2017-04-01"
    },
    "resourceProvider": "Microsoft.EventHub",
    "resourceUri": "/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.EventHub/namespaces/{namespace}/AuthorizationRules/RootManageSharedAccessKey",
    "operationName": "Microsoft.EventHub/namespaces/AuthorizationRules/listKeys/action",
    "status": "Succeeded",
    "subscriptionId": "{subscription-id}",
    "tenantId": "{tenant-id}"
  },
  "dataVersion": "2",
  "metadataVersion": "1",
  "topic": "/subscriptions/{subscription-id}"
}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
topic	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.
subject	string	Publisher-defined path to the event subject.
eventType	string	One of the registered event types for this event source.
eventTime	string	The time the event is generated based on the provider's UTC time.
id	string	Unique identifier for the event.
data	object	Subscription event data.
dataVersion	string	The schema version of the data object. The publisher defines the schema version.
metadataVersion	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
authorization	object	The requested authorization for the operation.
claims	object	The properties of the claims. For more information, see <a href="#">JWT specification</a> .
correlationId	string	An operation ID for troubleshooting.
httpRequest	object	The details of the operation. This object is only included when updating an existing resource or deleting a resource.
resourceProvider	string	The resource provider for the operation.
resourceUri	string	The URI of the resource in the operation.
operationName	string	The operation that was taken.

PROPERTY	TYPE	DESCRIPTION
<code>status</code>	string	The status of the operation.
<code>subscriptionId</code>	string	The subscription ID of the resource.
<code>tenantId</code>	string	The tenant ID of the resource.

## Tutorials and how-tos

TITLE	DESCRIPTION
<a href="#">Tutorial: Azure Automation with Event Grid and Microsoft Teams</a>	Create a virtual machine, which sends an event. The event triggers an Automation runbook that tags the virtual machine, and triggers a message that is sent to a Microsoft Teams channel.
<a href="#">How to: subscribe to events through portal</a>	Use the portal to subscribe to events for an Azure subscription.
<a href="#">Azure CLI: subscribe to events for an Azure subscription</a>	Sample script that creates an Event Grid subscription to an Azure subscription and sends events to a WebHook.
<a href="#">PowerShell: subscribe to events for an Azure subscription</a>	Sample script that creates an Event Grid subscription to an Azure subscription and sends events to a WebHook.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#).
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Azure Cache for Redis as an Event Grid source

3/5/2021 • 3 minutes to read • [Edit Online](#)

This article provides the properties and schema for Azure Cache for Redis events. For an introduction to event schemas, see [Azure Event Grid event schema](#).

## Available event types

These events are triggered when a client exports, imports, or scales by calling Azure Cache for Redis REST APIs. Patching event is triggered by Redis update.

EVENT NAME	DESCRIPTION
Microsoft.Cache.ExportRDBCompleted	Triggered when cache data is exported.
Microsoft.Cache.ImportRDBCompleted	Triggered when cache data is imported.
Microsoft.Cache.PatchingCompleted	Triggered when patching is completed.
Microsoft.Cache.ScalingCompleted	Triggered when scaling is completed.

## Example event

When an event is triggered, the Event Grid service sends data about that event to subscribing endpoint. This section contains an example of what that data would look like for each Azure Cache for Redis event.

- [Event Grid event schema](#)
- [Cloud event schema](#)

### Microsoft.Cache.PatchingCompleted event

```
[{"id": "9b87886d-21a5-4af5-8e3e-10c4b8dac73b", "eventType": "Microsoft.Cache.PatchingCompleted", "topic": "/subscriptions/{subscription_id}/resourceGroups/{resource_group_name}/providers/Microsoft.Cache/Redis/{cache_name}", "data": { "name": "PatchingCompleted", "timestamp": "2020-12-09T21:50:19.9995668+00:00", "status": "Succeeded"}, "subject": "PatchingCompleted", "dataversion": "1.0", "metadataVersion": "1", "eventTime": "2020-12-09T21:50:19.9995668+00:00"}]
```

### Microsoft.Cache.ImportRDBCompleted event

```
[{
  "id": "9b87886d-21a5-4af5-8e3e-10c4b8dac73b",
  "eventType": "Microsoft.Cache.ImportRDBCompleted",
  "topic": "/subscriptions/{subscription_id}/resourceGroups/{resource_group_name}/providers/Microsoft.Cache/Redis/{cache_name}",
  "data": {
    "name": "ImportRDBCompleted",
    "timestamp": "2020-12-09T21:50:19.9995668+00:00",
    "status": "Succeeded",
    "subject": "ImportRDBCompleted",
    "dataversion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2020-12-09T21:50:19.9995668+00:00"}]
```

## Microsoft.Cache.ExportRDBCompleted event

```
[{
  "id": "9b87886d-21a5-4af5-8e3e-10c4b8dac73b",
  "eventType": "Microsoft.Cache.ExportRDBCompleted",
  "topic": "/subscriptions/{subscription_id}/resourceGroups/{resource_group_name}/providers/Microsoft.Cache/Redis/{cache_name}",
  "data": {
    "name": "ExportRDBCompleted",
    "timestamp": "2020-12-09T21:50:19.9995668+00:00",
    "status": "Succeeded",
    "subject": "ExportRDBCompleted",
    "dataversion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2020-12-09T21:50:19.9995668+00:00"}]
```

## Microsoft.Cache.ScalingCompleted

```
[{
  "id": "9b87886d-21a5-4af5-8e3e-10c4b8dac73b",
  "eventType": "Microsoft.Cache.ScalingCompleted",
  "topic": "/subscriptions/{subscription_id}/resourceGroups/{resource_group_name}/providers/Microsoft.Cache/Redis/{cache_name}",
  "data": {
    "name": "ScalingCompleted",
    "timestamp": "2020-12-09T21:50:19.9995668+00:00",
    "status": "Succeeded",
    "subject": "ScalingCompleted",
    "dataversion": "1.0",
    "metadataVersion": "1",
    "eventTime": "2020-12-09T21:50:19.9995668+00:00"}]
```

## Event properties

- [Event Grid event schema](#)
- [Cloud event schema](#)

An event has the following top-level data:

PROPERTY	TYPE	DESCRIPTION
<code>topic</code>	string	Full resource path to the event source. This field isn't writeable. Event Grid provides this value.

PROPERTY	TYPE	DESCRIPTION
<code>subject</code>	string	Publisher-defined path to the event subject.
<code>eventType</code>	string	One of the registered event types for this event source.
<code>eventTime</code>	string	The time the event is generated based on the provider's UTC time.
<code>id</code>	string	Unique identifier for the event.
<code>data</code>	object	Azure Cache for Redis event data.
<code>dataVersion</code>	string	The schema version of the data object. The publisher defines the schema version.
<code>metadataVersion</code>	string	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

The data object has the following properties:

PROPERTY	TYPE	DESCRIPTION
<code>timestamp</code>	string	The time at which the event occurred.
<code>name</code>	string	The name of the event.
<code>status</code>	string	The status of the event. Failed or succeeded.

## Quickstarts

If you want to try Azure Cache for Redis events, see any of these quickstart articles:

IF YOU WANT TO USE THIS TOOL:	SEE THIS ARTICLE:
Azure portal	<a href="#">Quickstart: Route Azure Cache for Redis events to web endpoint with the Azure portal</a>
PowerShell	<a href="#">Quickstart: Route Azure Cache for Redis events to web endpoint with PowerShell</a>
Azure CLI	<a href="#">Quickstart: Route Azure Cache for Redis events to web endpoint with Azure CLI</a>

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Partner Events in Azure Event Grid (preview)

3/5/2021 • 5 minutes to read • [Edit Online](#)

The **Partner Events** feature allows a third-party SaaS provider to publish events from its services so that consumers can subscribe to those events. This feature offers a first-party experience to third-party event sources by exposing a [topic](#) type, a **partner topic**. Subscribers create subscriptions to this topic to consume events. It also provides a clean pub-sub model by separating concerns and ownership of resources that are used by event publishers and subscribers.

## NOTE

If you're new at using Event Grid, see [overview](#), [concepts](#), and [event handlers](#).

## What is Partner Events to a publisher?

To an event publisher, the Partner Events feature allows publishers to do the following tasks:

- Onboard their event sources to Event Grid
- Create a namespace (endpoint) to which they can publish events
- Create partner topics in Azure that subscribers own and use to consume events

## What is Partner Events to a subscriber?

To a subscriber, the Partner Events feature allows them to create partner topics in Azure to consume events from third-party event sources. Event consumption is realized by creating event subscriptions that send (push) events to a subscriber's event handler.

## Why should I use Partner Events?

You may want to use the Partner Events if you've one or more of the following requirements.

### For publishers

- You want a mechanism to make your events available on Azure. Your users can filter and route those events by using partner topics and event subscriptions that they own and manage. You could use other integration approaches such as [topics](#) and [domains](#). But, they wouldn't allow for a clean separation of resource (partner topics) ownership, management, and billing between publishers and subscribers. Also, this approach provides more intuitive user experience that makes it easy to discover and use partner topics.
- You want to publish events to a single endpoint, the namespace's endpoint. And, you want the ability to filter those events so that only a subset of them is available.
- You want to have visibility into metrics related to published events.
- You want to use [Cloud Events 1.0](#) schema for your events.

### For subscribers

- You want to subscribe to events from [third-party publishers](#) and handle the events using event handlers that are on Azure or elsewhere.
- You want to take advantage of the rich set of routing features and [destinations/event handlers](#) to process events from third-party sources.
- You want to implement loosely coupled architectures where your subscriber/event handler is unaware of the existence of the message broker used.

- You need a resilient push delivery mechanism with send-retry support and at-least once semantics.
- You want to use [Cloud Events 1.0](#) schema for your events.

## Available third-party event publishers

A third-party event publisher must go through an [onboarding process](#) before a subscriber can start consuming its events.

If you're a subscriber and would like a third-party service to expose its events through Event Grid,

### Auth0

Auth0 is the first partner publisher available. You can create an [Auth0 partner topic](#) to connect your Auth0 and Azure accounts. This integration allows you to react to, log, and monitor Auth0 events in real time. To try it out, see [Integrate Azure Event Grid with Auto0](#)

If you would like a third-party service to expose its events through Event Grid, submit the idea on the [User Voice portal](#).

## Resources managed by event publishers

Event publishers create and manage the following resources:

### Partner registration

A registration holds general information related to a publisher. It defines a type of partner topic that shows in the Azure portal as an option when users try to create a partner topic. A publisher may expose more than one or more partner topic types to fit the needs of its subscribers. That is, a publisher may create separate registrations (partner topic types) for events from different services. For example, for the human resources (HR) service, publisher may define a partner topic for events such as employee joined, employee promoted, and employee left the company.

Keep in mind the following points:

- Only Azure-approved partner registrations are visible.
- Registrations are global. That is, they aren't associated to a particular Azure region.
- A registration is an optional resource. But, we recommend that you (as a publisher) create a registration. It allows users to discover your topics on the [Create Partner Topic](#) page in the [Azure portal](#). Then, user can select event types (for example, employee joined, employee left, and so on.) while creating event subscriptions.

### Namespace

Like [custom topics](#) and [domains](#), a partner namespace is a regional endpoint to publish events. It's through namespaces that publishers create and manage event channels. A namespace also functions as the container resource for event channels.

### Event Channels

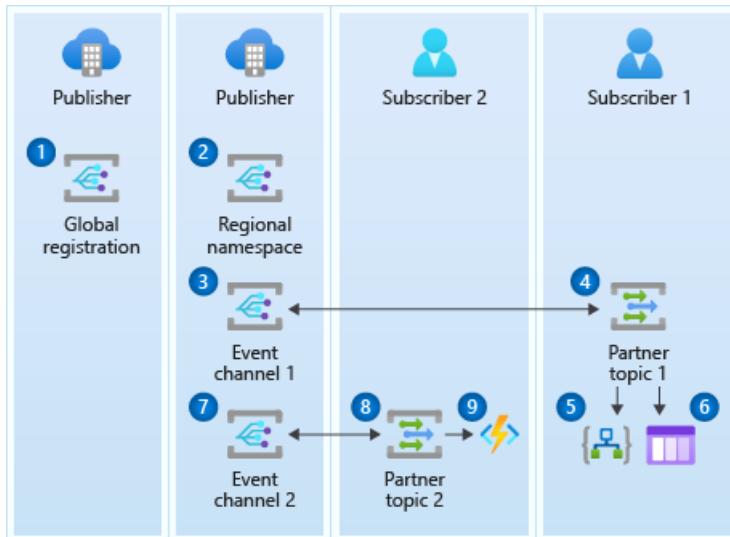
An event channel is a mirrored resource to a partner topic. When a publisher creates an event channel in the publisher's Azure subscription, it also creates a partner topic under a subscriber's Azure subscription. The operations done against an event channel (except GET) will be applied to the corresponding subscriber partner topic, even deletion. However, only partner topics are the kind of resources on which subscriptions and event delivery can be configured.

## Resources managed by subscribers

Subscribers can use partner topics defined by a publisher and it's the only type of resource they see and manage. Once a partner topic is created, a subscriber user can create event subscriptions defining filter rules to

[destinations/event handlers](#). To subscribers, a partner topic and its associated event subscriptions provide the same rich capabilities as [custom topics](#) and its related subscription(s) do with one notable difference: partner topics support only the [Cloud Events 1.0 schema](#), which provides a richer set of capabilities than other supported schemas.

The following image shows the flow of control plane operations.



1. Publisher creates a **partner registration**. Partner registrations are global. That is, they aren't associated with a particular Azure region. This step is optional.
2. Publisher creates a **partner namespace** in a specific region.
3. When Subscriber 1 tries to create a partner topic, an **event channel**, Event Channel 1, is created in the publisher's Azure subscription first.
4. Then, a **partner topic**, Partner Topic 1, is created in the subscriber's Azure subscription. The subscriber needs to activate the partner topic.
5. Subscriber 1 creates an **Azure Logic Apps subscription** to Partner Topic 1.
6. Subscriber 1 creates an **Azure Blob Storage subscription** to Partner Topic 1.
7. When Subscriber 2 tries to create a partner topic, another **event channel**, Event Channel 2, is created in the publisher's Azure subscription first.
8. Then, the **partner topic**, Partner Topic 2, is created in the second subscriber's Azure subscription. The subscriber needs to activate the partner topic.
9. Subscriber 2 creates an **Azure Functions subscription** to Partner Topic 2.

## Pricing

Partner topics are charged by the number of operations done when using Event Grid. For more information on all types of operations that are used as the basis for billing and detailed price information, see [Event Grid pricing](#).

## Limits

See [Event Grid Service limits](#) for detailed information about the limits in place for partner topics.

## Next steps

- [Auth0 partner topic](#)
- [How to use the Auth0 partner topic](#)
- [Become an Event Grid partner](#)

# Auth0 partner topics

11/2/2020 • 2 minutes to read • [Edit Online](#)



Auth0, the identity platform for application builders, provides developers and enterprises with the building blocks they need to secure their applications.

The Auth0 partner topic allows you to use events that are emitted by Auth0's system to accomplish a number of tasks. Engage with users in meaningful ways after the authentication or automate security and infrastructure tasks.

The integration allows you to stream your Auth0 log events with high reliability into Azure. There, you can consume the events with your favorite Azure resources. This integration allows you to react to events, gain insights, monitor for security issues, and interact with other powerful data pipelines.

For organizations that use Auth0 and Azure, this integration allows you to seamlessly integrate data across your entire stack.

## Available event types

The full list of available Auth0 event types and their descriptions is available on [this website](#).

## Use cases

### Engage with your users

Delivering a strong user experience is critical to reducing churn and keeping your users. Deliver more customized application experiences by using Auth0 events with Azure Functions and Azure Logic Apps.

### Understand user behavior

Understand when users access your product, where they're signed in, and what devices they use. Develop an understanding of the product areas that matter most by keeping track of these signals. These signals help you determine:

- What browsers and devices to support.
- What languages to localize your app in.
- When your peak traffic times are.

### Manage user data

Keeping and auditing your user actions is crucial for maintaining security and following industry regulations. The ability to edit, remove, or export user data is increasingly important to following privacy laws, such as the European Union's General Data Protection Regulation (GDPR).

### Secure your application

Combining security monitoring and incident response procedures is important when you protect a distributed system. For this reason, it's important to keep all the data in one place and monitor the entire stack.

## Next steps

- [Partner topics overview](#)
- [How to use the Auth0 partner topic](#)
- [Auth0 documentation](#)
- [Become an Event Grid partner](#)

# Partner onboarding overview (Azure Event Grid)

4/22/2021 • 5 minutes to read • [Edit Online](#)

This article describes how to privately use the Azure Event Grid partner resources and how to become a publicly available partner topic type.

You don't need special permission to begin using the Event Grid resource types associated with publishing events as an Event Grid partner. In fact, you can use them today to publish events privately to your own Azure subscriptions and to test out the resource model if you're considering becoming a partner.

## NOTE

For step-by-step instruction on how to onboard as an Event Grid partner by using the Azure portal, see [How to onboard as an Event Grid partner \(Azure portal\)](#).

## How Partner Events work

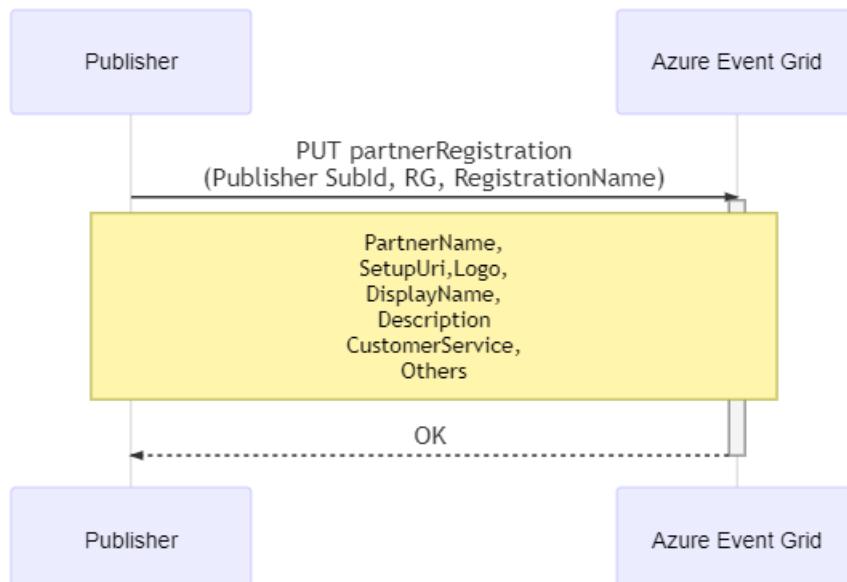
The Partner Events feature take the existing architecture that Event Grid already uses to publish events from Azure resources, such as Azure Storage and Azure IoT Hub, and makes those tools publicly available for anyone to use. Using these tools is by default private to your Azure subscription only. To make your events publicly available, fill out the form and [contact the Event Grid team](#).

The Partner Events feature allow you to publish events to Azure Event Grid for multitenant consumption.

## Onboarding and event publishing overview

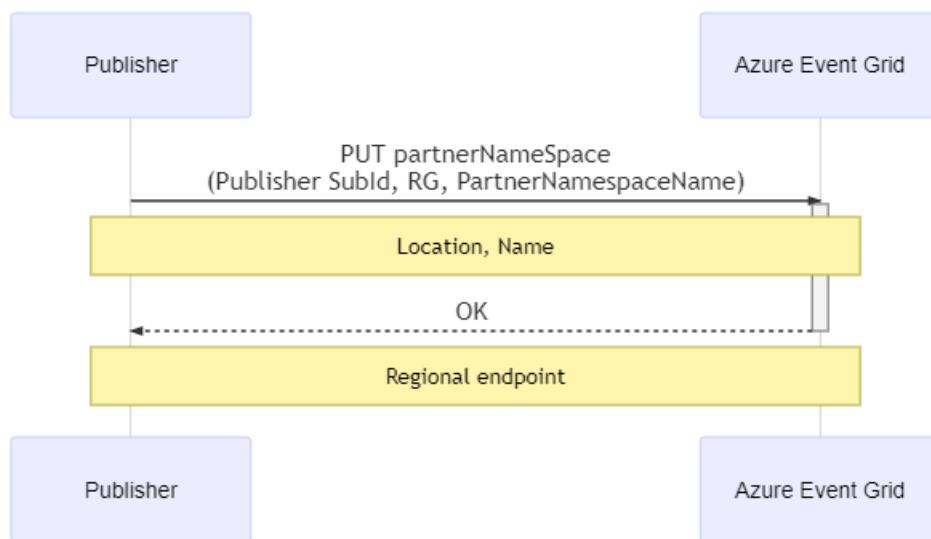
### Partner flow

1. Create an Azure tenant if you don't already have one.
2. Use the Azure CLI to create a new Event Grid `partnerRegistration`. This resource includes information such as display name, description, setup URI, and so on.



3. Create one or more partner namespaces in each region where you want to publish events. The Event Grid service provisions a publishing endpoint (for example,

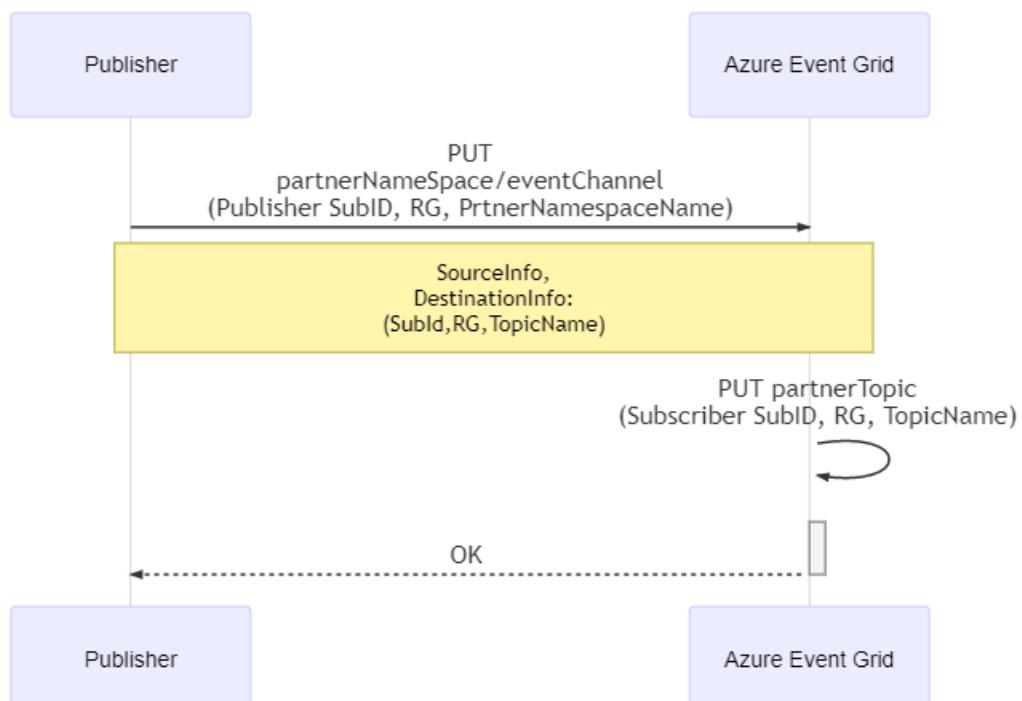
<https://contoso.westus-1.eventgrid.azure.net/api/events>) and access keys.



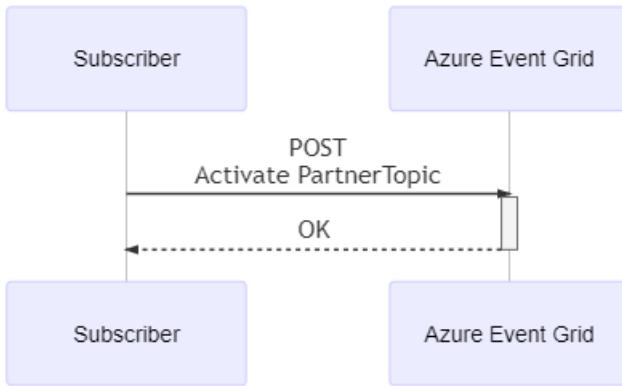
4. Provide a way for customers to register in your system that they want a partner topic.
5. Contact the Event Grid team to let them know you want your partner topic type to become public.

#### Customer flow

1. Your customer visits the Azure portal to note the Azure subscription ID and resource group they want the partner topic created in.
2. The customer requests a partner topic via your system. In response, you create an event tunnel to your partner namespace.
3. Event Grid creates a **Pending** partner topic in the customer's Azure subscription and resource group.



4. The customer activates the partner topic via the Azure portal. Events may now flow from your service to the customer's Azure subscription.



## Resource model

The following resource model is for Partner Events.

### Partner registrations

- Resource: `partnerRegistrations`
- Used by: Partners
- Description: Captures the global metadata of the software as a service (SaaS) partner (for example, name, display name, description, setup URI).

Creating or updating a partner registration is a self-serve operation for the partners. This self-serve ability enables partners to build and test the complete end-to-end flow.

Only Microsoft-approved partner registrations are discoverable by customers.

- Scope: Created in the partner's Azure subscription. Metadata is visible to customers after it's made public.

### Partner namespaces

- Resource: `partnerNamespaces`
- Used by: Partners
- Description: Provides a regional resource for publishing customer events to. Each partner namespace has a publishing endpoint and auth keys. The namespace is also how the partner requests a partner topic for a given customer and lists active customers.
- Scope: Lives in the partner's subscription.

### Event channel

- Resource: `partnerNamespaces/eventChannels`
- Used by: Partners
- Description: The event channels are a mirror of the customer's partner topic. By creating an event channel and specifying the customer's Azure subscription and resource group in the metadata, you signal to Event Grid to create a partner topic for the customer. Event Grid issues an Azure Resource Manager call to create a corresponding partner topic in the customer's subscription. The partner topic is created in a pending state. There's a one-to-one link between each event channel and partner topic.
- Scope: Lives in the partner's subscription.

### Partner topics

- Resource: `partnerTopics`
- Used by: Customers
- Description: Partner topics are similar to custom topics and system topics in Event Grid. Each partner topic is associated with a specific source (for example, `Contoso:myaccount`) and a specific partner topic

type (for example, Contoso). Customers create event subscriptions on the partner topic to route events to various event handlers.

Customers can't directly create this resource. The only way to create a partner topic is through a partner operation that creates an event channel.

- Scope: Lives in the customer's subscription.

### Partner topic types

- Resource: `partnerTopicTypes`
- Used by: Customers
- Description: Partner topic types are tenant-wide resource types that enable customers to discover the list of approved partner topic types. The URL looks like  
<https://management.azure.com/providers/Microsoft.EventGrid/partnerTopicTypes>
- Scope: Global

## Publish events to Event Grid

When you create a partner namespace in an Azure region, you get a regional endpoint and corresponding auth keys. Publish batches of events to this endpoint for all customer event channels in that namespace. Based on the source field in the event, Azure Event Grid maps each event with the corresponding partner topics.

### Event schema: CloudEvents v1.0

Publish events to Azure Event Grid by using the CloudEvents 1.0 schema. Event Grid supports both structured mode and batched mode. CloudEvents 1.0 is the only supported event schema for partner namespaces.

### Example flow

1. The publishing service does an HTTP POST to  
`https://contoso.westus2-1.eventgrid.azure.net/api/events?api-version=2018-01-01`.
2. In the request, include a header value named `aeg-sas-key` that contains a key for authentication. This key is provisioned during the creation of the partner namespace. For example, a valid header value is `aeg-sas-key: VXbGWce53249Mt8wuotr0GPmyJ/nDT4hgdEj9DpBeRr38arnnm5OFg==`.
3. Set the `Content-Type` header to `"application/cloudevents-batch+json; charset=UTF-8a"`.
4. Run an HTTP POST query to the publishing URL with a batch of events that correspond to that region. For example:

```
[
{
  "specversion" : "1.0-rc1",
  "type" : "com.contoso.ticketcreated",
  "source" : "com.contoso.account1",
  "subject" : "tickets/123",
  "id" : "A234-1234-1234",
  "time" : "2019-04-05T17:31:00Z",
  "comexampleextension1" : "value",
  "comexampleothervalue" : 5,
  "datacontenttype" : "application/json",
  "data" : {
    object-unique-to-each-publisher
  }
},
{
  "specversion" : "1.0-rc1",
  "type" : "com.contoso.ticketclosed",
  "source" : "https://contoso.com/account2",
  "subject" : "tickets/456",
  "id" : "A234-1234-1234",
  "time" : "2019-04-05T17:31:00Z",
  "comexampleextension1" : "value",
  "comexampleothervalue" : 5,
  "datacontenttype" : "application/json",
  "data" : {
    object-unique-to-each-publisher
  }
}
]
```

After posting to the partner namespace endpoint, you receive a response. The response is a standard HTTP response code. Some common responses are:

RESULT	RESPONSE
Success	200 OK
Event data has incorrect format	400 Bad Request
Invalid access key	401 Unauthorized
Incorrect endpoint	404 Not Found
Array or event exceeds size limits	413 Payload Too Large

## References

- [Swagger](#)
- [ARM template](#)
- [ARM template schema](#)
- [REST APIs](#)
- [CLI extension](#)

## SDKs

- [.NET](#)
- [Python](#)
- [Java](#)

- [Ruby](#)
- [JS](#)
- [Go](#)

## Next steps

- [Partner topics overview](#)
- [Partner topics onboarding form](#)
- [Auth0 partner topic](#)
- [How to use the Auth0 partner topic](#)

# Understand event domains for managing Event Grid topics

5/26/2021 • 4 minutes to read • [Edit Online](#)

This article describes how to use event domains to manage the flow of custom events to your various business organizations, customers, or applications. Use event domains to:

- Manage multitenant eventing architectures at scale.
- Manage your authorization and authentication.
- Partition your topics without managing each individually.
- Avoid individually publishing to each of your topic endpoints.

## Event domain overview

An event domain is a management tool for large numbers of Event Grid topics related to the same application. You can think of it as a meta-topic that can have thousands of individual topics.

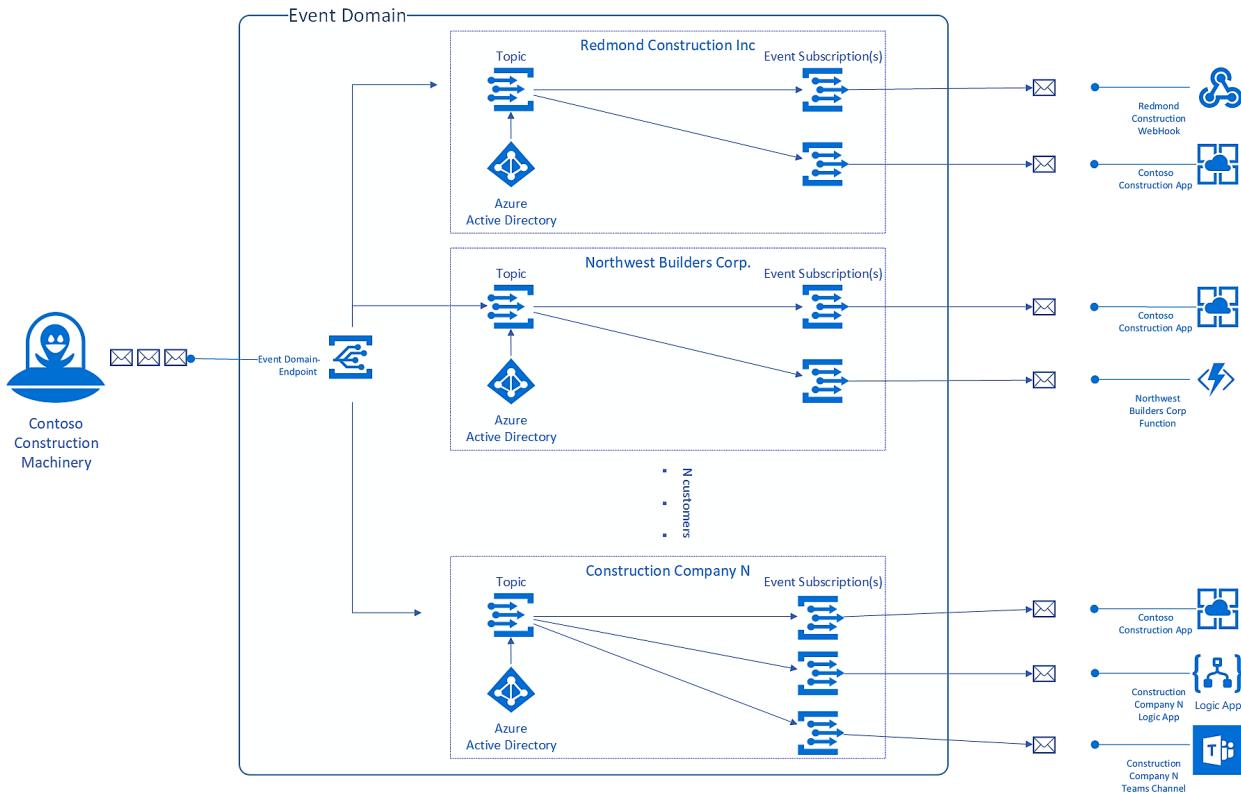
Event domains provide you the same architecture used by Azure services like Storage and IoT Hub to publish their events. They allow you to publish events to thousands of topics. Domains also give you authorization and authentication control over each topic so you can partition your tenants.

## Example use case

Event domains are most easily explained using an example. Let's say you run Contoso Construction Machinery, where you manufacture tractors, digging equipment, and other heavy machinery. As a part of running the business, you push real-time information to customers about equipment maintenance, systems health, and contract updates. All of this information goes to various endpoints including your app, customer endpoints, and other infrastructure that customers have set up.

Event domains allow you to model Contoso Construction Machinery as a single eventing entity. Each of your customers is represented as a topic within the domain. Authentication and authorization are handled using Azure Active Directory. Each of your customers can subscribe to their topic and get their events delivered to them. Managed access through the event domain ensures they can only access their topic.

It also gives you a single endpoint, which you can publish all of your customer events to. Event Grid will take care of making sure each topic is only aware of events scoped to its tenant.



## Access management

With a domain, you get fine grain authorization and authentication control over each topic via Azure role-based access control (Azure RBAC). You can use these roles to restrict each tenant in your application to only the topics you wish to grant them access to.

Azure RBAC in event domains works the same way [managed access control](#) works in the rest of Event Grid and Azure. Use Azure RBAC to create and enforce custom role definitions in event domains.

### Built in roles

Event Grid has two built-in role definitions to make Azure RBAC easier for working with event domains. These roles are [EventGrid EventSubscription Contributor \(Preview\)](#) and [EventGrid EventSubscription Reader \(Preview\)](#). You assign these roles to users who need to subscribe to topics in your event domain. You scope the role assignment to only the topic that users need to subscribe to.

For information about these roles, see [Built-in roles for Event Grid](#).

## Subscribing to topics

Subscribing to events on a topic within an event domain is the same as [creating an Event Subscription on a custom topic](#) or subscribing to an event from an Azure service.

### IMPORTANT

Domain topic is considered an **auto-managed** resource in Event Grid. You can create an event subscription at the [domain scope](#) without creating the domain topic. In this case, Event Grid automatically creates the domain topic on your behalf. Of course, you can still choose to create the domain topic manually. This behavior allows you to worry about one less resource when dealing with a huge number of domain topics. When the last subscription to a domain topic is deleted, the domain topic is also deleted irrespective of whether the domain topic was manually created or auto-created.

### Domain scope subscriptions

Event domains also allow for domain-scope subscriptions. An event subscription on an event domain will

receive all events sent to the domain regardless of the topic the events are sent to. Domain scope subscriptions can be useful for management and auditing purposes.

## Publishing to an event domain

When you create an event domain, you're given a publishing endpoint similar to if you had created a topic in Event Grid.

To publish events to any topic in an Event Domain, push the events to the domain's endpoint the [same way you would for a custom topic](#). The only difference is that you must specify the topic you'd like the event to be delivered to.

For example, publishing the following array of events would send event with `"id": "1111"` to topic `foo` while the event with `"id": "2222"` would be sent to topic `bar`:

```
[{
    "topic": "foo",
    "id": "1111",
    "eventType": "maintenanceRequested",
    "subject": "myapp/vehicles/diggers",
    "eventTime": "2018-10-30T21:03:07+00:00",
    "data": {
        "make": "Contoso",
        "model": "Small Digger"
    },
    "dataVersion": "1.0"
},
{
    "topic": "bar",
    "id": "2222",
    "eventType": "maintenanceCompleted",
    "subject": "myapp/vehicles/tractors",
    "eventTime": "2018-10-30T21:04:12+00:00",
    "data": {
        "make": "Contoso",
        "model": "Big Tractor"
    },
    "dataVersion": "1.0"
}]
```

Event domains handle publishing to topics for you. Instead of publishing events to each topic you manage individually, you can publish all of your events to the domain's endpoint. Event Grid makes sure each event is sent to the correct topic.

## Limits and quotas

Here are the limits and quotas related to event domains:

- 100,000 topics per event domain
- 100 event domains per Azure subscription
- 500 event subscriptions per topic in an event domain
- 50 domain scope subscriptions
- 5,000 events per second ingestion rate (into a domain)

If these limits don't suit you, open a support ticket or send an email to [askgrid@microsoft.com](mailto:askgrid@microsoft.com).

## Pricing

Event domains use the same [operations pricing](#) that all other features in Event Grid use.

Operations work the same in event domains as they do in custom topics. Each ingress of an event to an event domain is an operation, and each delivery attempt for an event is an operation.

## Next steps

- To learn about setting up event domains, creating topics, creating event subscriptions, and publishing events, see [Manage event domains](#).

# Azure Event Grid event schema

6/9/2021 • 3 minutes to read • [Edit Online](#)

This article describes the properties and schema that are present for all events. 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. This behavior may change in the future.

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:

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

For example, the schema published for an Azure Blob storage event is:

```
[
  {
    "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/xstoretestaccount",
    "subject": "/blobServices/default/containers/oc2d2817345i200097container/blobs/oc2d2817345i20002296blob",
    "eventType": "Microsoft.Storage.BlobCreated",
    "eventTime": "2017-06-26T18:41:00.9584103Z",
    "id": "831e1650-001e-001b-66ab-eeb76e069631",
    "data": {
      "api": "PutBlockList",
      "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
      "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
      "eTag": "0x8D4BCC2E4835CD0",
      "contentType": "application/octet-stream",
      "contentLength": 524288,
      "blobType": "BlockBlob",
      "url": "https://oc2d2817345i60006.blob.core.windows.net/oc2d2817345i200097container/oc2d2817345i20002296blob",
      "sequencer": "000000000000442000000000028963",
      "storageDiagnostics": {
        "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
      }
    },
    "dataVersion": "",
    "metadataVersion": "1"
  }
]
```

## Event properties

All events have the same following top-level data:

PROPERTY	TYPE	REQUIRED	DESCRIPTION
topic	string	No, but if included, must match the Event Grid topic Azure Resource Manager ID exactly. If not included, Event Grid will stamp onto the event.	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.

PROPERTY	TYPE	REQUIRED	DESCRIPTION
dataVersion	string	No, but will be stamped with an empty value.	The schema version of the data object. The publisher defines the schema version.
metadataVersion	string	Not required, but if included, must match the Event Grid Schema <code>metadataVersion</code> exactly (currently, only <code>1</code> ). If not included, Event Grid will stamp onto the event.	The schema version of the event metadata. Event Grid defines the schema of the top-level properties. Event Grid provides this value.

To learn about the properties in the data object, see the event source:

- [Azure Policy](#)
- [Azure subscriptions \(management operations\)](#)
- [Container Registry](#)
- [Blob storage](#)
- [Event Hubs](#)
- [IoT Hub](#)
- [Media Services](#)
- [Resource groups \(management operations\)](#)
- [Service Bus](#)
- [Azure SignalR](#)
- [Azure Machine Learning](#)

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.

## Next steps

- For an introduction to Azure Event Grid, see [What is Event Grid?](#)
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# CloudEvents v1.0 schema with Azure Event Grid

11/2/2020 • 2 minutes to read • [Edit Online](#)

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.

CloudEvents is being built by several [collaborators](#), including Microsoft, through the [Cloud Native Computing Foundation](#). It's currently available as version 1.0.

This article describes CloudEvents schema with Event Grid.

## Sample event using CloudEvents schema

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

```
{  
    "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": "0000000000000000000000000000000099240000000000c41c18",  
        "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"
```

## Event Grid for CloudEvents

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.

INPUT SCHEMA	OUTPUT SCHEMA
CloudEvents format	CloudEvents format
Event Grid format	CloudEvents format
Event Grid format	Event Grid format

For all event schemas, Event Grid requires validation when publishing to an event grid topic and when creating an event subscription. For more information, see [Event Grid security and authentication](#).

## Next steps

See [How to use CloudEvents v1.0 schema with Event Grid](#).

# Event handlers in Azure Event Grid

11/2/2020 • 2 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes some further action to process the event. Several Azure services are automatically configured to handle events. You can also use any webhook for handling events. The webhook doesn't need to be hosted in Azure to handle events. Event Grid only supports HTTPS webhook endpoints.

## Supported event handlers

Here are the supported event handlers:

- [Webhooks](#). Azure Automation runbooks and Logic Apps are supported via webhooks.
- [Azure functions](#)
- [Event hubs](#)
- [Relay hybrid connections](#)
- [Service Bus queues and topics](#)
- [Storage queues](#)

## Next steps

- For an introduction to Event Grid, see [About Event Grid](#).

# Webhooks, Automation runbooks, Logic Apps as event handlers for Azure Event Grid events

11/2/2020 • 2 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes some further action to process the event. Several Azure services are automatically configured to handle events. You can also use any WebHook for handling events. The WebHook doesn't need to be hosted in Azure to handle events. Event Grid only supports HTTPS Webhook endpoints.

## NOTE

- Azure Automation runbooks and logic apps are supported as event handlers via webhooks.
- Even though you can use **Webhook** as an **endpoint type** to configure an Azure function as an event handler, use **Azure Function** as an endpoint type. For more information, see [Azure function as an event handler](#).

## Webhooks

See the following articles for an overview and examples of using webhooks as event handlers.

TITLE	DESCRIPTION
<a href="#">Quickstart: create and route custom events with - Azure CLI, PowerShell, and portal.</a>	Shows how to send custom events to a WebHook.
<a href="#">Quickstart: route Blob storage events to a custom web endpoint with - Azure CLI, PowerShell, and portal.</a>	Shows how to send blob storage events to a WebHook.
<a href="#">Quickstart: send container registry events</a>	Shows how to use Azure CLI to send Container Registry events.
<a href="#">Overview: receive events to an HTTP endpoint</a>	Describes how to validate an HTTP endpoint to receive events from an Event Subscription, and receive and deserialize events.

## Azure Automation

You can process events by using Azure Automation runbooks. Processing of events by using automated runbooks is supported via webhooks. You create a webhook for the runbook and then use the webhook handler. See the following tutorial for an example:

TITLE	DESCRIPTION
<a href="#">Tutorial: Azure Automation with Event Grid and Microsoft Teams</a>	Create a virtual machine, which sends an event. The event triggers an Automation runbook that tags the virtual machine, and triggers a message that is sent to a Microsoft Teams channel.

## Logic Apps

Use Logic Apps to implement business processes to process Event Grid events. You don't create a webhook explicitly in this scenario. The webhook is created for you automatically when you configure the logic app to handle events from Event Grid. See the following tutorials for examples:

TITLE	DESCRIPTION
<a href="#">Tutorial: Monitor virtual machine changes with Azure Event Grid and Logic Apps</a>	A logic app monitors changes to a virtual machine and sends emails about those changes.
<a href="#">Tutorial: Send email notifications about Azure IoT Hub events using Logic Apps</a>	A logic app sends a notification email every time a device is added to your IoT hub.
<a href="#">Tutorial: Respond to Azure Service Bus events received via Azure Event Grid by using Azure Functions and Azure Logic Apps</a>	Event Grid sends messages from Service Bus topic to function app and logic app.

## REST example (for PUT)

```
{  
  "properties":  
  {  
    "destination":  
    {  
      "endpointType": "WebHook",  
      "properties":  
      {  
        "endpointUrl": "<WEB HOOK URL>",  
        "maxEventsPerBatch": 1,  
        "preferredBatchSizeInKilobytes": 64  
      }  
    },  
    "eventDeliverySchema": "EventGridSchema"  
  }  
}
```

## Next steps

See the [Event handlers](#) article for a list of supported event handlers.

# Use a function as an event handler for Event Grid events

3/17/2021 • 3 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes an action to process the event. Several Azure services are automatically configured to handle events and **Azure Functions** is one of them.

To use a function in Azure as a handler for events, follow one of these approaches:

- Use [Event Grid trigger](#). Specify **Azure Function** as the **endpoint type**. Then, specify the function app and the function that will handle events.
- Use [HTTP trigger](#). Specify **Web Hook** as the **endpoint type**. Then, specify the URL for the function that will handle events.

We recommend that you use the first approach (Event Grid trigger) as it has the following advantages over the second approach:

- Event Grid automatically validates Event Grid triggers. With generic HTTP triggers, you must implement the [validation response](#) yourself.
- Event Grid automatically adjusts the rate at which events are delivered to a function triggered by an Event Grid event based on the perceived rate at which the function can process events. This rate match feature averts delivery errors that stem from the inability of a function to process events as the function's event processing rate can vary over time. To improve efficiency at high throughput, enable batching on the event subscription. For more information, see [Enable batching](#).

## NOTE

Currently, you can't use an Event Grid trigger for a function app when the event is delivered in the **CloudEvents** schema. Instead, use an HTTP trigger.

## Tutorials

TITLE	DESCRIPTION
<a href="#">Quickstart: Handle events with function</a>	Sends a custom event to a function for processing.
<a href="#">Tutorial: automate resizing uploaded images using Event Grid</a>	Users upload images through web app to storage account. When a storage blob is created, Event Grid sends an event to the function app, which resizes the uploaded image.
<a href="#">Tutorial: stream big data into a data warehouse</a>	When Event Hubs creates a Capture file, Event Grid sends an event to a function app. The app retrieves the Capture file and migrates data to a data warehouse.
<a href="#">Tutorial: Azure Service Bus to Azure Event Grid integration examples</a>	Event Grid sends messages from Service Bus topic to a function app and a logic app.

## REST example (for PUT)

```
{  
  "properties":  
  {  
    "destination":  
    {  
      "endpointType": "AzureFunction",  
      "properties":  
      {  
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Web/sites/<FUNCTION APP NAME>/functions/<FUNCTION NAME>",  
        "maxEventsPerBatch": 10,  
        "preferredBatchSizeInKilobytes": 6400  
      }  
    },  
    "eventDeliverySchema": "EventGridSchema"  
  }  
}
```

## Enable batching

For a higher throughput, enable batching on the subscription. If you are using the Azure portal, you can set maximum events per batch and the preferred batch size in kilo bytes at the time of creating a subscription or after the creation.

You can configure batch settings using the Azure portal, PowerShell, CLI, or Resource Manager template.

### Azure portal

At the time creating a subscription in the UI, on the **Create Event Subscription** page, switch to the **Advanced Features** tab, and set values for **Max events per batch** and **Preferred batch size in kilobytes**.

## Create Event Subscription



Event Grid

Basic    Filters    Additional Features

### DEAD-LETTERING

Save events that cannot be delivered to storage. [Learn more](#)

Enable dead-lettering

### RETRY POLICIES

Customize how many times and for how long event delivery will be retried. [Learn more](#)

Configure Retry Policies

### EVENT SUBSCRIPTION EXPIRATION TIME

Set a time at which the event subscription will automatically be deleted.

Enable expiration time

### BATCHING

Batching can be used to improve efficiency at high-throughput. Max events sets the maximum number of events that a subscription will include in a batch. Preferred batch size sets the preferred upper bound of batch size in kb, but can be exceeded if a single event is larger than this threshold. [Learn more](#)

Max events per batch \*

1

Preferred batch size in kilobytes \*

64

### LABELS

Add Label

**Create**

You can update these values for an existing subscription on the **Features** tab of the **Event Grid Topic** page.

 **spfuncsub**  
Event Subscription

**TOPIC**  
 Event Grid Topic  
[spegrid0917topic](#)

**Metrics** **Filters** **Features** Features

**RETRY POLICIES**  
Customize how many times and for how long event delivery will be retried. [Learn more](#)

Max Event Delivery Attempts:

Event Time to Live:  Days  Hours  Minutes  Seconds

**DEAD-LETTERING**  
Save events that cannot be delivered to storage. [Learn more](#)

Enable dead-lettering

**EVENT SUBSCRIPTION EXPIRATION TIME**  
Set a time at which the event subscription will automatically be deleted.

Enable expiration time

**BATCHING**  
Batching can be used to improve efficiency at high-throughput. Max events sets the maximum number of events that a subscription will include in a batch. Preferred batch size sets the preferred upper bound of batch size in kb, but can be exceeded if a single event is larger than this threshold. [Learn more](#)

**Max events per batch \***

**Preferred batch size in kilobytes \***

## Azure Resource Manager template

You can set `maxEventsPerBatch` and `preferredBatchSizeInKilobytes` in an Azure Resource Manager template. For more information, see [Microsoft.EventGrid eventSubscriptions template reference](#).

## Azure CLI

You can use the `az eventgrid event-subscription create` or `az eventgrid event-subscription update` command to configure batch-related settings using the following parameters: `--max-events-per-batch` or `--preferred-batch-size-in-kilobytes`.

## Azure PowerShell

You can use the `New-AzEventGridSubscription` or `Update-AzEventGridSubscription` cmdlet to configure batch-related settings using the following parameters: `-MaxEventsPerBatch` or `-PreferredBatchSizeInKiloBytes`.

### NOTE

When you use Event Grid Trigger, the Event Grid service fetches the client secret for the target Azure function, and uses it to deliver events to the Azure function. If you protect your azure function with an Azure Active Directory application, you have to take the generic web hook approach and use the HTTP Trigger.

## Next steps

See the [Event handlers](#) article for a list of supported event handlers.

# Event hub as an event handler for Azure Event Grid events

5/11/2021 • 2 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes an action to process the event. Several Azure services are automatically configured to handle events and **Azure Event Hubs** is one of them.

Use **Event Hubs** when your solution gets events from Event Grid faster than it can process the events. Once the events are in an event hub, your application can process events from the event hub at its own schedule. You can scale your event processing to handle the incoming events.

## Tutorials

See the following examples:

TITLE	DESCRIPTION
<a href="#">Quickstart: Route custom events to Azure Event Hubs with Azure CLI</a>	Sends a custom event to an event hub for processing by an application.
<a href="#">Resource Manager template: Create an Event Grid custom topic and send events to an event hub</a>	A Resource Manager template that creates a subscription for a custom topic. It sends events to an Azure Event Hubs.

## Message headers

These are the properties you receive in the message headers:

PROPERTY NAME	DESCRIPTION
aeg-subscription-name	Name of the event subscription.
aeg-delivery-count	Number of attempts made for the event.
aeg-event-type	Type of the event.  It can be one of the following values: <ul style="list-style-type: none"><li>• SubscriptionValidation</li><li>• Notification</li><li>• SubscriptionDeletion</li></ul>
aeg-metadata-version	Metadata version of the event.  For <b>Event Grid event schema</b> , this property represents the metadata version and for <b>cloud event schema</b> , it represents the <b>spec version</b> .

PROPERTY NAME	DESCRIPTION
aeg-data-version	Data version of the event.  For <b>Event Grid event schema</b> , this property represents the data version and for <b>cloud event schema</b> , it doesn't apply.
aeg-output-event-id	ID of the Event Grid event.

## REST examples (for PUT)

### Event hub

```
{
  "properties": {
    {
      "destination": {
        {
          "endpointType": "EventHub",
          "properties": {
            {
              "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventHub/namespaces/<EVENT HUBS NAMESPACE NAME>/eventhubs/<EVENT HUB NAME>"
            }
          },
          "eventDeliverySchema": "EventGridSchema"
        }
      }
    }
  }
}
```

### Event hub - delivery with managed identity

```
{
  "properties": {
    "deliveryWithResourceIdentity": {
      {
        "identity": {
          {
            "type": "SystemAssigned"
          },
          "destination": {
            {
              "endpointType": "EventHub",
              "properties": {
                {
                  "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventHub/namespaces/<EVENT HUBS NAMESPACE NAME>/eventhubs/<EVENT HUB NAME>"
                }
              }
            },
            "eventDeliverySchema": "EventGridSchema"
          }
        }
      }
    }
  }
}
```

## Next steps

See the [Event handlers](#) article for a list of supported event handlers.

# Service Bus queues and topics as event handlers for Azure Event Grid events

5/14/2021 • 3 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes some further action to process the event. Several Azure services are automatically configured to handle events and **Azure Service Bus** is one of them.

You can use a Service queue or topic as a handler for events from Event Grid.

## Service Bus queues

### NOTE

Session enabled queues are not supported as event handlers for Azure Event Grid events

You can route events in Event Grid directly to Service Bus queues for use in buffering or command & control scenarios in enterprise applications.

In the Azure portal, while creating an event subscription, select **Service Bus Queue** as endpoint type and then click **select an endpoint** to choose a Service Bus queue.

### Using CLI to add a Service Bus queue handler

For Azure CLI, the following example subscribes and connects an event grid topic to a Service Bus queue:

```
az eventgrid event-subscription create \
  --name <my-event-subscription> \
  --source-resource-id
/subscriptions/{SubID}/resourceGroups/{RG}/providers/Microsoft.EventGrid/topics/topic1 \
  --endpoint-type servicebusqueue \
  --endpoint
/subscriptions/{SubID}/resourceGroups/TestRG/providers/Microsoft.ServiceBus/namespaces/ns1/queues/queue1
```

## Service Bus topics

You can route events in Event Grid directly to Service Bus topics to handle Azure system events with Service Bus topics, or for command & control messaging scenarios.

In the Azure portal, while creating an event subscription, select **Service Bus Topic** as endpoint type and then click **select and endpoint** to choose a Service Bus topic.

### Using CLI to add a Service Bus topic handler

For Azure CLI, the following example subscribes and connects an event grid topic to a Service Bus topic:

```

az eventgrid event-subscription create \
--name <my-event-subscription> \
--source-resource-id
/subscriptions/{SubID}/resourceGroups/{RG}/providers/Microsoft.EventGrid/topics/topic1 \
--endpoint-type servicebustopic \
--endpoint
/subscriptions/{SubID}/resourceGroups/TestRG/providers/Microsoft.ServiceBus/namespaces/ns1/topics/topic1

```

## Message headers

These are the properties you receive in the message headers:

PROPERTY NAME	DESCRIPTION
aeg-subscription-name	Name of the event subscription.
aeg-delivery-count	Number of attempts made for the event.
aeg-event-type	Type of the event.  It can be one of the following values: <ul style="list-style-type: none"> <li>• SubscriptionValidation</li> <li>• Notification</li> <li>• SubscriptionDeletion</li> </ul>
aeg-metadata-version	Metadata version of the event.  For <b>Event Grid event schema</b> , this property represents the metadata version and for <b>cloud event schema</b> , it represents the <b>spec version</b> .
aeg-data-version	Data version of the event.  For <b>Event Grid event schema</b> , this property represents the data version and for <b>cloud event schema</b> , it doesn't apply.
aeg-output-event-id	ID of the Event Grid event.

When sending an event to a Service Bus queue or topic as a brokered message, the `messageid` of the brokered message is an internal system ID.

The internal system ID for the message will be maintained across redelivery of the event so that you can avoid duplicate deliveries by turning on **duplicate detection** on the service bus entity. We recommend that you enable duration of the duplicate detection on the Service Bus entity to be either the time-to-live (TTL) of the event or max retry duration, whichever is longer.

## REST examples (for PUT)

### Service Bus queue

```
{
  "properties":
  {
    "destination":
    {
      "endpointType": "ServiceBusQueue",
      "properties":
      {
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.ServiceBus/namespaces/<SERVICE BUS NAMESPACE NAME>/queues/<SERVICE BUS QUEUE NAME>",
        "eventDeliverySchema": "EventGridSchema"
      }
    }
  }
}
```

## Service Bus queue - delivery with managed identity

```
{
  "properties": {
    "deliveryWithResourceIdentity":
    {
      "identity":
      {
        "type": "SystemAssigned"
      },
      "destination":
      {
        "endpointType": "ServiceBusQueue",
        "properties":
        {
          "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.ServiceBus/namespaces/<SERVICE BUS NAMESPACE NAME>/queues/<SERVICE BUS QUEUE NAME>",
          "eventDeliverySchema": "EventGridSchema"
        }
      }
    }
  }
}
```

## Service Bus topic

```
{
  "properties":
  {
    "destination":
    {
      "endpointType": "ServiceBusTopic",
      "properties":
      {
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.ServiceBus/namespaces/<SERVICE BUS NAMESPACE NAME>/topics/<SERVICE BUS TOPIC NAME>",
        "eventDeliverySchema": "EventGridSchema"
      }
    }
  }
}
```

## Service Bus topic - delivery with managed identity

```
{  
  "properties":  
  {  
    "deliveryWithResourceIdentity":  
    {  
      "identity":  
      {  
        "type": "SystemAssigned"  
      },  
      "destination":  
      {  
        "endpointType": "ServiceBusTopic",  
        "properties":  
        {  
          "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP  
NAME>/providers/Microsoft.ServiceBus/namespaces/<SERVICE BUS NAMESPACE NAME>/topics/<SERVICE BUS TOPIC  
NAME>"  
        }  
      }  
    },  
    "eventDeliverySchema": "EventGridSchema"  
  }  
}
```

## Next steps

See the [Event handlers](#) article for a list of supported event handlers.

# Relay Hybrid connection as an event handler for Azure Event Grid events

11/2/2020 • 2 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes some further action to process the event. Several Azure services are automatically configured to handle events and **Azure Relay** is one of them.

Use **Azure Relay Hybrid Connections** to send events to applications that are within an enterprise network and don't have a publicly accessible endpoint.

## Tutorials

See the following tutorial for an example of using an Azure Relay hybrid connection as an event handler.

TITLE	DESCRIPTION
<a href="#">Tutorial: send events to hybrid connection</a>	Sends a custom event to an existing hybrid connection for processing by a listener application.

## REST example (for PUT)

```
{  
  "properties":  
  {  
    "destination":  
    {  
      "endpointType": "HybridConnection",  
      "properties":  
      {  
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Relay/namespaces/<RELAY NAMESPACE NAME>/hybridconnections/<HYBRID CONNECTION NAME>"  
      }  
    },  
    "eventDeliverySchema": "EventGridSchema"  
  }  
}
```

## Next steps

See the [Event handlers](#) article for a list of supported event handlers.

# Storage queue as an event handler for Azure Event Grid events

6/8/2021 • 2 minutes to read • [Edit Online](#)

An event handler is the place where the event is sent. The handler takes some further action to process the event. Several Azure services are automatically configured to handle events and **Azure Queue Storage** is one of them.

Use **Queue Storage** to receive events that need to be pulled. You might use Queue storage when you have a long running process that takes too long to respond. By sending events to Queue storage, the app can pull and process events on its own schedule.

## Tutorials

See the following tutorial for an example of using Queue storage as an event handler.

TITLE	DESCRIPTION
<a href="#">Quickstart: route custom events to Azure Queue storage with Azure CLI and Event Grid</a>	Describes how to send custom events to a Queue storage.

## REST examples (for PUT)

### Storage queue as the event handler

```
{  
  "properties":  
  {  
    "destination":  
    {  
      "endpointType": "StorageQueue",  
      "properties":  
      {  
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Storage/storageAccounts/<STORAGE ACCOUNT NAME>",  
        "queueName": "<QUEUE NAME>"  
      }  
    },  
    "eventDeliverySchema": "EventGridSchema"  
  }  
}
```

### Storage queue as the event handler - delivery with managed identity

```
{
  "properties":
  {
    "deliveryWithResourceIdentity":
    {
      "identity":
      {
        "type": "SystemAssigned"
      },
      "destination":
      {
        "endpointType": "StorageQueue",
        "properties":
        {
          "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Storage/storageAccounts/<STORAGE ACCOUNT NAME>",
          "queueName": "<QUEUE NAME>"
        }
      }
    },
    "eventDeliverySchema": "EventGridSchema"
  }
}
```

### **Storage queue as the event handler with a deadletter destination**

```
{
  "name": "",
  "properties":
  {
    "destination":
    {
      "endpointType": "StorageQueue",
      "properties":
      {
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Storage/storageAccounts/<DESTINATION STORAGE>",
        "queueName": "queue1"
      }
    },
    "eventDeliverySchema": "EventGridSchema",
    "deadLetterDestination":
    {
      "endpointType": "StorageBlob",
      "properties":
      {
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Storage/storageAccounts/<DEADLETTER STORAGE>",
        "blobContainerName": "test"
      }
    }
  }
}
```

### **Storage queue as the event handler with a deadletter destination - managed identity**

```
{  
  "properties":  
  {  
    "destination":  
    {  
      "endpointType": "StorageQueue",  
      "properties":  
      {  
        "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Storage/storageAccounts/<DESTINATION STORAGE>",  
        "queueName": "queue1"  
      }  
    },  
    "eventDeliverySchema": "EventGridSchema",  
    "deadLetterWithResourceIdentity":  
    {  
      "identity":  
      {  
        "type": "SystemAssigned"  
      },  
      "deadLetterDestination":  
      {  
        "endpointType": "StorageBlob",  
        "properties":  
        {  
          "resourceId": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Storage/storageAccounts/<DEADLETTER STORAGE>",  
          "blobContainerName": "test"  
        }  
      }  
    }  
  }  
}
```

## Next steps

See the [Event handlers](#) article for a list of supported event handlers.

# Understand event filtering for Event Grid subscriptions

6/8/2021 • 15 minutes to read • [Edit Online](#)

This article describes the different ways to filter which events are sent to your endpoint. 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:

```
"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.

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 property to filter and route events. Consider adding 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. 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.

The JSON syntax for filtering by subject is:

```
"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, string, or an array.
- values - The value or values to compare to the key.

## Key

Key is the field in the event data that you're using for filtering. It can be one of the following types:

- Number
- Boolean
- String
- Array. You need to set the `enableAdvancedFilteringOnArrays` property to true to use this feature.

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

For events in the **Event Grid schema**, use the following values for the key: `ID`, `Topic`, `Subject`, `EventType`, `DataVersion`, or event data (like `data.key1`).

For events in **Cloud Events schema**, use the following values for the key: `eventid`, `source`, `eventtype`, `eventtypeversion`, or event data (like `data.key1`).

For **custom input schema**, use the event data fields (like `data.key1`). To access fields in the data section, use the `.` (dot) notation. For example, `data.sitename`, `data.appEventTypeDetail.action` to access `sitename` or `action` for the following sample event.

```
"data": {  
    "appEventTypeDetail": {  
        "action": "Started"  
    },  
    "siteName": "<site-name>",  
    "clientRequestId": "None",  
    "correlationRequestId": "None",  
    "requestId": "292f499d-04ee-4066-994d-c2df57b99198",  
    "address": "None",  
    "verb": "None"  
},
```

## Values

The values can be: number, string, boolean, or array

## Operators

The available operators for **numbers** are:

## NumberIn

The NumberIn operator evaluates to true if the **key** value is one of the specified **filter** values. In the following example, it checks whether the value of the `counter` attribute in the `data` section is 5 or 1.

```
"advancedFilters": [{  
    "operatorType": "NumberIn",  
    "key": "data.counter",  
    "values": [  
        5,  
        1  
    ]  
}]
```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a, b, c]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH filter IN (a, b, c)  
    FOR_EACH key IN (v1, v2, v3)  
        IF filter == key  
            MATCH
```

## NumberNotIn

The NumberNotIn evaluates to true if the **key** value is **not** any of the specified **filter** values. In the following example, it checks whether the value of the `counter` attribute in the `data` section isn't 41 and 0.

```
"advancedFilters": [{  
    "operatorType": "NumberNotIn",  
    "key": "data.counter",  
    "values": [  
        41,  
        0  
    ]  
}]
```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a, b, c]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH filter IN (a, b, c)  
    FOR_EACH key IN (v1, v2, v3)  
        IF filter == key  
            FAIL_MATCH
```

## NumberLessThan

The NumberLessThan operator evaluates to true if the **key** value is **less than** the specified **filter** value. In the following example, it checks whether the value of the `counter` attribute in the `data` section is less than 100.

```
"advancedFilters": [{
    "operatorType": "NumberLessThan",
    "key": "data.counter",
    "value": 100
}]
```

If the key is an array, all the values in the array are checked against the filter value. Here's the pseudo code with the key: `[v1, v2, v3]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH key IN (v1, v2, v3)
IF key < filter
    MATCH
```

## NumberGreaterThan

The NumberGreaterThan operator evaluates to true if the **key** value is **greater than** the specified **filter** value. In the following example, it checks whether the value of the `counter` attribute in the `data` section is greater than 20.

```
"advancedFilters": [{
    "operatorType": "NumberGreaterThan",
    "key": "data.counter",
    "value": 20
}]
```

If the key is an array, all the values in the array are checked against the filter value. Here's the pseudo code with the key: `[v1, v2, v3]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH key IN (v1, v2, v3)
IF key > filter
    MATCH
```

## NumberLessThanOrEquals

The NumberLessThanOrEquals operator evaluates to true if the **key** value is **less than or equal** to the specified **filter** value. In the following example, it checks whether the value of the `counter` attribute in the `data` section is less than or equal to 100.

```
"advancedFilters": [{
    "operatorType": "NumberLessThanOrEquals",
    "key": "data.counter",
    "value": 100
}]
```

If the key is an array, all the values in the array are checked against the filter value. Here's the pseudo code with the key: `[v1, v2, v3]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH key IN (v1, v2, v3)
IF key <= filter
    MATCH
```

## NumberGreaterThanOrEquals

The NumberGreaterThanOrEqual operator evaluates to true if the **key** value is **greater than or equal** to the specified **filter** value. In the following example, it checks whether the value of the `counter` attribute in the `data` section is greater than or equal to 30.

```
"advancedFilters": [{}  
  "operatorType": "NumberGreaterThanOrEqual",  
  "key": "data.counter",  
  "value": 30  
]
```

If the key is an array, all the values in the array are checked against the filter value. Here's the pseudo code with the key: `[v1, v2, v3]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH key IN (v1, v2, v3)  
  IF key >= filter  
    MATCH
```

## NumberInRange

The NumberInRange operator evaluates to true if the **key** value is in one of the specified **filter ranges**. In the following example, it checks whether the value of the `key1` attribute in the `data` section is in one of the two ranges: 3.14159 - 999.95, 3000 - 4000.

```
{  
  "operatorType": "NumberInRange",  
  "key": "data.key1",  
  "values": [[3.14159, 999.95], [3000, 4000]]  
}
```

The `values` property is an array of ranges. In the previous example, it's an array of two ranges. Here's an example of an array with one range to check.

**Array with one range:**

```
{  
  "operatorType": "NumberInRange",  
  "key": "data.key1",  
  "values": [[3000, 4000]]  
}
```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: an array of ranges. In this pseudo code, `a` and `b` are low and high values of each range in the array. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH (a,b) IN filter.Values  
  FOR_EACH key IN (v1, v2, v3)  
    IF key >= a AND key <= b  
      MATCH
```

## NumberNotInRange

The NumberNotInRange operator evaluates to true if the **key** value is **not** in any of the specified **filter ranges**.

In the following example, it checks whether the value of the `key1` attribute in the `data` section is in one of the two ranges: 3.14159 - 999.95, 3000 - 4000. If it's, the operator returns false.

```
{  
    "operatorType": "NumberNotInRange",  
    "key": "data.key1",  
    "values": [[3.14159, 999.95], [3000, 4000]]  
}
```

The `values` property is an array of ranges. In the previous example, it's an array of two ranges. Here's an example of an array with one range to check.

**Array with one range:**

```
{  
    "operatorType": "NumberNotInRange",  
    "key": "data.key1",  
    "values": [[3000, 4000]]  
}
```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: an array of ranges. In this pseudo code, `a` and `b` are low and high values of each range in the array. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH (a,b) IN filter.Values  
    FOR_EACH key IN (v1, v2, v3)  
        IF key >= a AND key <= b  
            FAIL_MATCH
```

The available operator for **booleans** is:

## BoolEquals

The `BoolEquals` operator evaluates to true if the `key` value is the specified boolean value `filter`. In the following example, it checks whether the value of the `isEnabled` attribute in the `data` section is `true`.

```
"advancedFilters": [{  
    "operatorType": "BoolEquals",  
    "key": "data.isEnabled",  
    "value": true  
}]
```

If the key is an array, all the values in the array are checked against the filter boolean value. Here's the pseudo code with the key: `[v1, v2, v3]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH key IN (v1, v2, v3)  
    IF filter == key  
        MATCH
```

The available operators for **strings** are:

## StringContains

The **StringContains** evaluates to true if the **key** value **contains** any of the specified **filter** values (as substrings). In the following example, it checks whether the value of the `key1` attribute in the `data` section contains one of the specified substrings: `microsoft` or `azure`. For example, `azure data factory` has `azure` in it.

```
"advancedFilters": [{}  
    "operatorType": "StringContains",  
    "key": "data.key1",  
    "values": [  
        "microsoft",  
        "azure"  
    ]  
]
```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH filter IN (a, b, c)  
    FOR_EACH key IN (v1, v2, v3)  
        IF key CONTAINS filter  
            MATCH
```

## StringNotContains

The **StringNotContains** operator evaluates to true if the **key** does **not contain** the specified **filter** values as substrings. If the key contains one of the specified values as a substring, the operator evaluates to false. In the following example, the operator returns true only if the value of the `key1` attribute in the `data` section doesn't have `contoso` and `fabrikam` as substrings.

```
"advancedFilters": [{}  
    "operatorType": "StringNotContains",  
    "key": "data.key1",  
    "values": [  
        "contoso",  
        "fabrikam"  
    ]  
]
```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```
FOR_EACH filter IN (a, b, c)  
    FOR_EACH key IN (v1, v2, v3)  
        IF key CONTAINS filter  
            FAIL_MATCH
```

See [Limitations](#) section for current limitation of this operator.

## StringBeginsWith

The **StringBeginsWith** operator evaluates to true if the **key** value **begins with** any of the specified **filter** values. In the following example, it checks whether the value of the `key1` attribute in the `data` section begins with `event` or `grid`. For example, `event hubs` begins with `event`.

```

"advancedFilters": [
    "operatorType": "StringBeginsWith",
    "key": "data.key1",
    "values": [
        "event",
        "message"
    ]
}

```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```

FOR_EACH filter IN (a, b, c)
    FOR_EACH key IN (v1, v2, v3)
        IF key BEGINS_WITH filter
            MATCH

```

## StringNotBeginsWith

The **StringNotBeginsWith** operator evaluates to true if the **key** value does **not begin with** any of the specified **filter** values. In the following example, it checks whether the value of the `key1` attribute in the `data` section doesn't begin with `event` or `message`.

```

"advancedFilters": [
    "operatorType": "StringNotBeginsWith",
    "key": "data.key1",
    "values": [
        "event",
        "message"
    ]
}

```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```

FOR_EACH filter IN (a, b, c)
    FOR_EACH key IN (v1, v2, v3)
        IF key BEGINS_WITH filter
            FAIL_MATCH

```

## StringEndsWith

The **StringEndsWith** operator evaluates to true if the **key** value **ends with** one of the specified **filter** values. In the following example, it checks whether the value of the `key1` attribute in the `data` section ends with `jpg` or `jpeg` or `png`. For example, `eventgrid.png` ends with `png`.

```

"advancedFilters": [
    "operatorType": "StringEndsWith",
    "key": "data.key1",
    "values": [
        "jpg",
        "jpeg",
        "png"
    ]
}

```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```

FOR_EACH filter IN (a, b, c)
    FOR_EACH key IN (v1, v2, v3)
        IF key ENDS_WITH filter
            MATCH

```

## StringNotEndsWith

The **StringNotEndsWith** operator evaluates to true if the **key** value does **not end with** any of the specified **filter** values. In the following example, it checks whether the value of the `key1` attribute in the `data` section doesn't end with `jpg` or `jpeg` or `png`.

```

"advancedFilters": [
    "operatorType": "StringNotEndsWith",
    "key": "data.key1",
    "values": [
        "jpg",
        "jpeg",
        "png"
    ]
}

```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```

FOR_EACH filter IN (a, b, c)
    FOR_EACH key IN (v1, v2, v3)
        IF key ENDS_WITH filter
            FAIL_MATCH

```

## StringIn

The **StringIn** operator checks whether the **key** value **exactly matches** one of the specified **filter** values. In the following example, it checks whether the value of the `key1` attribute in the `data` section is `contoso` or `fabrikam` or `factory`.

```

"advancedFilters": [
    "operatorType": "StringIn",
    "key": "data.key1",
    "values": [
        "contoso",
        "fabrikam",
        "factory"
    ]
}

```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```

FOR_EACH filter IN (a, b, c)
    FOR_EACH key IN (v1, v2, v3)
        IF filter == key
            MATCH

```

## StringNotIn

The **StringNotIn** operator checks whether the **key** value **does not match** any of the specified **filter** values. In the following example, it checks whether the value of the `key1` attribute in the `data` section isn't `aws` and `bridge`.

```

"advancedFilters": [
    "operatorType": "StringNotIn",
    "key": "data.key1",
    "values": [
        "aws",
        "bridge"
    ]
}

```

If the key is an array, all the values in the array are checked against the array of filter values. Here's the pseudo code with the key: `[v1, v2, v3]` and the filter: `[a,b,c]`. Any key values with data types that don't match the filter's data type are ignored.

```

FOR_EACH filter IN (a, b, c)
    FOR_EACH key IN (v1, v2, v3)
        IF filter == key
            FAIL_MATCH

```

All string comparisons aren't case-sensitive.

### NOTE

If the event JSON doesn't contain the advanced filter key, filter is evaluated as **not matched** for the following operators: NumberGreaterThan, NumberGreaterThanOrEqual, NumberLessThan, NumberLessThanOrEqual, NumberIn, BoolEquals, StringContains, StringNotContains, StringBeginsWith, StringNotBeginsWith, StringEndsWith, StringNotEndsWith, StringIn.

The filter is evaluated as **matched** for the following operators: NumberNotIn, StringNotIn.

## IsNullOrUndefined

The `IsNullOrUndefined` operator evaluates to true if the key's value is `NULL` or `undefined`.

```
{  
    "operatorType": "IsNullOrUndefined",  
    "key": "data.key1"  
}
```

In the following example, `key1` is missing, so the operator would evaluate to true.

```
{  
    "data":  
    {  
        "key2": 5  
    }  
}
```

In the following example, `key1` is set to null, so the operator would evaluate to true.

```
{  
    "data":  
    {  
        "key1": null  
    }  
}
```

If `key1` has any other value in these examples, the operator would evaluate to false.

## IsNotNull

The `IsNotNull` operator evaluates to true if the key's value isn't `NULL` or `undefined`.

```
{  
    "operatorType": "IsNotNull",  
    "key": "data.key1"  
}
```

## OR and AND

If you specify a single filter with multiple values, an **OR** operation is performed, so the value of the key field must be one of these values. Here is an example:

```
"advancedFilters": [  
    {  
        "operatorType": "StringContains",  
        "key": "Subject",  
        "values": [  
            "/providers/microsoft.devtestlab/",  
            "/providers/Microsoft.Compute/virtualMachines/"  
        ]  
    }  
]
```

If you specify multiple different filters, an **AND** operation is done, so each filter condition must be met. Here's an example:

```

"advancedFilters": [
    {
        "operatorType": "StringContains",
        "key": "Subject",
        "values": [
            "/providers/microsoft.devtestlab/"
        ]
    },
    {
        "operatorType": "StringContains",
        "key": "Subject",
        "values": [
            "/providers/Microsoft.Compute/virtualMachines/"
        ]
    }
]

```

## CloudEvents

For events in the [CloudEvents schema](#), use the following values for the key: `eventid`, `source`, `eventtype`, `eventtypeversion`, or event data (like `data.key1`).

You can also use [extension context attributes in CloudEvents 1.0](#). In the following example, `comexampleextension1` and `comexampleothervalue` are extension context attributes.

```

{
    "specversion" : "1.0",
    "type" : "com.example.someevent",
    "source" : "/mycontext",
    "id" : "C234-1234-1234",
    "time" : "2018-04-05T17:31:00Z",
    "subject": null,
    "comexampleextension1" : "value",
    "comexampleothervalue" : 5,
    "datacontenttype" : "application/json",
    "data" : {
        "appinfoA" : "abc",
        "appinfoB" : 123,
        "appinfoC" : true
    }
}

```

Here's an example of using an extension context attribute in a filter.

```

"advancedFilters": [
    {
        "operatorType": "StringBeginsWith",
        "key": "comexampleothervalue",
        "values": [
            "5",
            "1"
        ]
    }
]

```

## Limitations

Advanced filtering has the following limitations:

- 5 advanced filters and 25 filter values across all the filters per event grid subscription
- 512 characters per string value

- Five values for `in` and `not in` operators
- The `StringNotContains` operator is currently not available in the portal.
- Keys with `.` (dot) character in them. For example: `http://schemas.microsoft.com/claims/authnclassreference` or `john.doe@contoso.com`. Currently, there's no support for escape characters in keys.

The same key can be used in more than one filter.

## Next steps

- To learn about filtering events with PowerShell and Azure CLI, see [Filter events for Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Event Grid message delivery and retry

4/2/2021 • 8 minutes to read • [Edit Online](#)

This article describes how Azure Event Grid handles events when delivery isn't acknowledged.

Event Grid provides durable delivery. It delivers each message **at least once** for each subscription. Events are sent to the registered endpoint of each subscription immediately. If an endpoint doesn't acknowledge receipt of an event, Event Grid retries delivery of the event.

## NOTE

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

## Batched event delivery

Event Grid defaults to sending each event individually to subscribers. The subscriber receives an array with a single event. You can configure Event Grid to batch events for delivery for improved HTTP performance in high-throughput scenarios.

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.

Batched delivery is configured on a per-event subscription basis via the portal, CLI, PowerShell, or SDKs.

### Azure portal:

#### BATCHING

Batching can be used to improve efficiency at high-throughput. Max events sets the maximum number of events that a subscription will include in a batch. Preferred batch size sets the preferred upper bound of batch size in kb, but can be exceeded if a single event is larger than this threshold. [Learn more](#)

Max events per batch \*

1

Preferred batch size in kilobytes \*

64

### Azure CLI

When creating an event subscription, use the following parameters:

- **max-events-per-batch** - Maximum number of events in a batch. Must be a number between 1 and 5000.
- **preferred-batch-size-in-kilobytes** - Preferred batch size in kilobytes. Must be a number between 1 and 1024.

```

storageid=$(az storage account show --name <storage_account_name> --resource-group <resource_group_name> --
query id --output tsv)
endpoint=https://$sitename.azurewebsites.net/api/updates

az eventgrid event-subscription create \
--resource-id $storageid \
--name <event_subscription_name> \
--endpoint $endpoint \
--max-events-per-batch 1000 \
--preferred-batch-size-in-kilobytes 512

```

For more information on using Azure CLI with Event Grid, see [Route storage events to web endpoint with Azure CLI](#).

## Retry schedule and duration

When EventGrid 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
Webhook	400 Bad Request, 413 Request Entity Too Large, 403 Forbidden, 404 Not Found, 401 Unauthorized

### NOTE

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, EventGrid performs the retry using policies described below:

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. Event Grid retries delivery on the following schedule on a best effort basis:

- 10 seconds
- 30 seconds
- 1 minute
- 5 minutes
- 10 minutes
- 30 minutes
- 1 hour
- 3 hours
- 6 hours

- Every 12 hours up to 24 hours

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.

For deterministic behavior, set the event time-to-live and max delivery attempts in the [subscription retry policies](#).

By default, Event Grid expires all events that aren't delivered within 24 hours. You can [customize the retry policy](#) when creating an event subscription. You provide the maximum number of delivery attempts (default is 30) and the event time-to-live (default is 1440 minutes).

## 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 has exceeded 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.

Event Grid sends an event to the dead-letter location when it has tried all of its retry attempts. 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.

The time-to-live expiration is checked ONLY at the next scheduled delivery attempt. So, even if time-to-live expires before the next scheduled delivery attempt, event expiry is checked only at the time of the next delivery and then subsequently dead-lettered.

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.

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. The endpoint is in the format of:

```
/subscriptions/<subscription-id>/resourceGroups/<resource-group-name>/providers/Microsoft.Storage/storageAccounts/<storage-name>/blobServices/default/containers/<container-name>
```

You might want to be notified when an event has been sent to the dead-letter location. To use Event Grid to

respond to undelivered events, [create an event subscription](#) for the dead-letter blob storage. Every time your dead-letter blob storage receives an undelivered event, Event Grid notifies your handler. The handler responds with actions you wish to take for reconciling undelivered events. For an example of setting up a dead-letter location and retry policies, see [Dead letter and retry policies](#).

## Delivery event formats

This section gives you examples of events and dead-lettered events in different delivery schema formats (Event Grid schema, CloudEvents 1.0 schema, and custom schema). For more information about these formats, see [Event Grid schema](#) and [Cloud Events 1.0 schema](#) articles.

### Event Grid schema

#### Event

```
{
  "id": "93902694-901e-008f-6f95-7153a806873c",
  "eventTime": "2020-08-13T17:18:13.164726Z",
  "eventType": "Microsoft.Storage.BlobCreated",
  "dataVersion": "",
  "metadataVersion": "1",
  "topic": "/subscriptions/00000000-0000-0000-0000-
000000000000/resourceGroups/rgwithoutpolicy/providers/Microsoft.Storage/storageAccounts/myegteststgfoo",
  "subject": "/blobServices/default/containers/deadletter/blobs/myBlobFile.txt",
  "data": {
    "api": "PutBlob",
    "clientRequestId": "c0d879ad-88c8-4bbe-8774-d65888dc2038",
    "requestId": "93902694-901e-008f-6f95-7153a8000000",
    "eTag": "0x8D83FACDC0C3402",
    "contentType": "text/plain",
    "contentLength": 0,
    "blobType": "BlockBlob",
    "url": "https://myegteststgfoo.blob.core.windows.net/deadletter/myBlobFile.txt",
    "sequencer": "0000000000000000000000000000000015508000000000005101c",
    "storageDiagnostics": { "batchId": "cfb32f79-3006-0010-0095-711faa000000" }
  }
}
```

#### Dead-letter event

```
{
  "id": "93902694-901e-008f-6f95-7153a806873c",
  "eventTime": "2020-08-13T17:18:13.164726Z",
  "eventType": "Microsoft.Storage.BlobCreated",
  "dataVersion": "",
  "metadataVersion": "1",
  "topic": "/subscriptions/0000000000-0000-0000-0000-
0000000000/resourceGroups/rgwithoutpolicy/providers/Microsoft.Storage/storageAccounts/myegteststgfoo",
  "subject": "/blobServices/default/containers/deadletter/blobs/myBlobFile.txt",
  "data": {
    "api": "PutBlob",
    "clientRequestId": "c0d879ad-88c8-4bbe-8774-d65888dc2038",
    "requestId": "93902694-901e-008f-6f95-7153a8000000",
    "eTag": "0x8D83FACDC0C3402",
    "contentType": "text/plain",
    "contentLength": 0,
    "blobType": "BlockBlob",
    "url": "https://myegteststgfoo.blob.core.windows.net/deadletter/myBlobFile.txt",
    "sequencer": "0000000000000000000000000000000015508000000000005101c",
    "storageDiagnostics": { "batchId": "cfb32f79-3006-0010-0095-711faa000000" }
  },
  "deadLetterReason": "MaxDeliveryAttemptsExceeded",
  "deliveryAttempts": 1,
  "lastDeliveryOutcome": "NotFound",
  "publishTime": "2020-08-13T17:18:14.0265758Z",
  "lastDeliveryAttemptTime": "2020-08-13T17:18:14.0465788Z"
}
}
```

## CloudEvents 1.0 schema

### Event

```
{
  "id": "caee971c-3ca0-4254-8f99-1395b394588e",
  "source": "mysource",
  "dataversion": "1.0",
  "subject": "mySubject",
  "type": "fooEventType",
  "datacontenttype": "application/json",
  "data": {
    "prop1": "value1",
    "prop2": 5
  }
}
```

### Dead-letter event

```
{
  "id": "caee971c-3ca0-4254-8f99-1395b394588e",
  "source": "mysource",
  "dataversion": "1.0",
  "subject": "mySubject",
  "type": "fooEventType",
  "datacontenttype": "application/json",
  "data": {
    "prop1": "value1",
    "prop2": 5
  },
  "deadletterreason": "MaxDeliveryAttemptsExceeded",
  "deliveryattempts": 1,
  "lastdeliveryoutcome": "NotFound",
  "publishtime": "2020-08-13T21:21:36.4018726Z",
}
}
```

## Custom schema

### Event

```
{
  "prop1": "my property",
  "prop2": 5,
  "myEventType": "fooEventType"
}
```

### Dead-letter event

```
{
  "id": "8bc07e6f-0885-4729-90e4-7c3f052bd754",
  "eventTime": "2020-08-13T18:11:29.4121391Z",
  "eventType": "myEventType",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "topic": "/subscriptions/00000000000-0000-0000-0000-
00000000000/resourceGroups/rgwithoutpolicy/providers/Microsoft.EventGrid/topics/myCustomSchemaTopic",
  "subject": "subjectDefault",

  "deadLetterReason": "MaxDeliveryAttemptsExceeded",
  "deliveryAttempts": 1,
  "lastDeliveryOutcome": "NotFound",
  "publishTime": "2020-08-13T18:11:29.4121391Z",
  "lastDeliveryAttemptTime": "2020-08-13T18:11:29.4277644Z",

  "data": {
    "prop1": "my property",
    "prop2": 5,
    "myEventType": "fooEventType"
  }
}
```

## Message delivery status

Event Grid uses HTTP response codes to acknowledge receipt of events.

### Success codes

Event Grid considers **only** the following HTTP response codes as successful deliveries. All other status codes are considered failed deliveries and will be retried or deadlettered as appropriate. Upon receiving a successful status code, Event Grid considers delivery complete.

- 200 OK
- 201 Created
- 202 Accepted
- 203 Non-Authoritative Information
- 204 No Content

### Failure codes

All other codes not in the above set (200-204) are considered failures and will be retried (if needed). Some have specific retry policies tied to them outlined below, all others follow the standard exponential back-off model. It's important to keep in mind that due to the highly parallelized nature of Event Grid's architecture, the retry behavior is non-deterministic.

STATUS CODE	RETRY BEHAVIOR
400 Bad Request	Not retried
401 Unauthorized	Retry after 5 minutes or more for Azure Resources Endpoints
403 Forbidden	Not retried
404 Not Found	Retry after 5 minutes or more for Azure Resources Endpoints
408 Request Timeout	Retry after 2 minutes or more
413 Request Entity Too Large	Not retried
503 Service Unavailable	Retry after 30 seconds or more
All others	Retry after 10 seconds or more

## 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

For more information, see [Custom delivery properties](#).

## Next steps

- To view the status of event deliveries, see [Monitor Event Grid message delivery](#).
- To customize event delivery options, see [Dead letter and retry policies](#).

# Webhook Event delivery

3/30/2021 • 4 minutes to read • [Edit Online](#)

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.

1. **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.
2. **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.

### Validation details

- At the time of event subscription creation/update, Event Grid posts a subscription validation event to the target endpoint.
- The event contains a header value "aeg-event-type: SubscriptionValidation".
- The event body has the same schema as other Event Grid events.
- The eventType property of the event is `Microsoft.EventGrid.SubscriptionValidationEvent`.
- The data property of the event includes a `validationCode` property with a randomly generated string. For example, "validationCode: acb13...".
- The event data also includes a `validationUrl` property with a URL for manually validating the subscription.
- The array contains only the validation event. Other events are sent in a separate request after you echo back the validation code.
- The EventGrid DataPlane SDKs have classes corresponding to the subscription validation event data and subscription validation response.

An example `SubscriptionValidationEvent` is shown in the following example:

```
[  
 {  
   "id": "2d1781af-3a4c-4d7c-bd0c-e34b19da4e66",  
   "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
   "subject": "",  
   "data": {  
     "validationCode": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6",  
     "validationUrl": "https://rp-eastus2.eventgrid.azure.net:553/events/subscriptions/estest/validate?  
id=512d38b6-c7b8-40c8-89fe-f46f9e9622b6&t=2018-04-26T20:30:54.4538837Z&apiVersion=2018-05-01-  
preview&token=1A1A1A1A"  
   },  
   "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",  
   "eventTime": "2018-01-25T22:12:19.4556811Z",  
   "metadataVersion": "1",  
   "dataVersion": "1"  
 }  
 ]
```

To prove endpoint ownership, echo back the validation code in the `validationResponse` property, as shown in the following example:

```
{  
   "validationResponse": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"  
}
```

You must return an HTTP 200 OK response status code. HTTP 202 Accepted is not recognized as a valid Event Grid subscription validation response. The HTTP request must complete within 30 seconds. If the operation doesn't finish within 30 seconds, then the operation will be canceled and it may be reattempted after 5 seconds. If all the attempts fail, then it will be treated as validation handshake error.

Or, you can manually validate the subscription by sending a GET request to the validation URL. The event subscription stays in a pending state until validated. The validation Url uses port 553. If your firewall rules block port 553 then rules may need to be updated for successful manual handshake.

For an example of handling the subscription validation handshake, see a [C# sample](#).

## Endpoint validation with CloudEvents v1.0

If you are already familiar with Event Grid, you may be aware of Event Grid's endpoint validation handshake for preventing abuse. CloudEvents v1.0 implements its own [abuse protection semantics](#) using the HTTP OPTIONS method. You can read more about it [here](#). When using the CloudEvents schema for output, Event Grid uses with the CloudEvents v1.0 abuse protection in place of the Event Grid validation event mechanism.

## Next steps

See the following article to learn how to troubleshoot event subscription validations:

[Troubleshoot event subscription validations](#)

# Custom delivery properties

4/22/2021 • 2 minutes to read • [Edit Online](#)

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

When creating an event subscription in the Azure portal, you can use the **Delivery Properties** tab to set custom HTTP headers. This page lets you set fixed and dynamic header values.

## Setting static header values

To set headers with a fixed value, provide the name of the header and its value in the corresponding fields:

The screenshot shows the 'Create Event Subscription' dialog for 'Event Grid'. The 'Delivery Properties' tab is selected and highlighted with a red border. Below it, a section titled 'PROPERTIES SENT ON EVENT DELIVERY' provides instructions for defining headers. A table is used to set static header values, with rows for 'category' and another unnamed row. The 'category' row has 'Static' selected in the Type dropdown, 'orders' in the Value input, and an unchecked 'Is secret?' checkbox. The second row also has 'Static' selected in the Type dropdown and an unchecked 'Is secret?' checkbox.

Header name	Type	Value	Secret
category	Static	orders	<input type="checkbox"/> Is secret?
	Static		<input type="checkbox"/> Is secret?

You may want check **Is secret?** when providing sensitive data. Sensitive data won't be displayed on the Azure portal.

## Setting dynamic header values

You can set the value of a header based on a property in an incoming event. Use JsonPath syntax to refer to an incoming event's property value to be used as the value for a header in outgoing requests. For example, to set the value of a header named **Channel** using the value of the incoming event property **system** in the event data, configure your event subscription in the following way:

## Create Event Subscription

Event Grid

Basic Filters Additional Features **Delivery Properties**

### PROPERTIES SENT ON EVENT DELIVERY

Define headers that are included with the request sent to the destination. For example, you may want to set a fixed value for the Authorization header to be sent with the request. Alternatively, you can set the value as a JsonPath reference to an existing envelope property or to any property in the event data. For more information, consult [Event Grid documentation](#).

Valid dynamic header source fields for currently selected event schema:

- id, topic, subject, eventtype, dataversion
- Custom properties inside the data payload, using "." as the nesting separator. (e.g. data, data.key, data.key1.key2)

Header name	Type	Value	Secret
channel	Dynamic	data.system	<input type="checkbox"/> Is secret? 
	Static		<input type="checkbox"/> Is secret?

## Examples

This section gives you a few examples of using delivery properties.

### Setting the Authorization header with a bearer token (non-normative example)

Set a value to an Authorization header to identify the request with your Webhook handler. An Authorization header can be set if you aren't [protecting your Webhook with Azure Active Directory](#).

HEADER NAME	HEADER TYPE	HEADER VALUE
Authorization	Static	BEARER S1AV32hkKG...

Outgoing requests should now contain the header set on the event subscription:

```
GET /home.html HTTP/1.1
Host: acme.com
User-Agent: <user-agent goes here>
Authorization: BEARER S1AV32hkKG...
```

### NOTE

Defining authorization headers is a sensible option when your destination is a Webhook. It should not be used for [functions subscribed with a resource id](#), Service Bus, Event Hubs, and Hybrid Connections as those destinations support their own authentication schemes when used with Event Grid.

## Service Bus example

Azure Service Bus supports the use of a [BrokerProperties HTTP header](#) to define message properties when sending single messages. The value of the `BrokerProperties` header should be provided in the JSON format. For example, if you need to set message properties when sending a single message to Service Bus, set the header in the following way:

HEADER NAME	HEADER TYPE	HEADER VALUE

HEADER NAME	HEADER TYPE	HEADER VALUE
BrokerProperties	Static	BrokerProperties: { "MessageId": "701332E1-B37B-4D29-AA0A-E367906C206E", "TimeToLive" : 90}

## Event Hubs example

If you need to publish events to a specific partition within an event hub, define a [BrokerProperties HTTP header](#) on your event subscription to specify the partition key that identifies the target event hub partition.

HEADER NAME	HEADER TYPE	HEADER VALUE
BrokerProperties	Static	BrokerProperties: {"PartitionKey": "0000000000-0000-0000-0000-000000000000"}

## Configure time to live on outgoing events to Azure Storage Queues

For the Azure Storage Queues destination, you can only configure the time-to-live the outgoing message will have once it has been delivered to an Azure Storage queue. If no time is provided, the message's default time to live is 7 days. You can also set the event to never expire.

The screenshot shows the 'Create Event Subscription' interface. The 'Delivery Properties' tab is active. A red box highlights the 'STORAGE QUEUE MESSAGE TTL' section, which contains a note about defining the time-to-live for storage queues and a 'Never expires' checkbox. Below this, there's a 'Storage queue message time to live' input field with a 7-day setting.

## Next steps

For more information about event delivery, see the following article:

- [Delivery and retry](#)
- [Webhook event delivery](#)
- [Event filtering](#)

# Server-side geo disaster recovery in Azure Event Grid

3/5/2021 • 2 minutes to read • [Edit Online](#)

Event Grid now has an automatic geo disaster recovery (GeoDR) of meta-data not only for new, but all existing domains, topics, and event subscriptions. If an entire Azure region goes down, Event Grid will already have all of your event-related infrastructure metadata synced to a paired region. Your new events will begin to flow again with no intervention by you.

Disaster recovery is measured with two metrics:

- Recovery Point Objective (RPO): the minutes or hours of data that may be lost.
- Recovery Time Objective (RTO): the minutes or hours the service may be down.

Event Grid's automatic failover has different RPOs and RTOs for your metadata (event subscriptions etc.) and data (events). If you need different specification from the following ones, you can still implement your own [client-side fail over using the topic health apis](#).

## Recovery point objective (RPO)

- **Metadata RPO:** zero minutes. Anytime a resource is created in Event Grid, it's instantly replicated across regions. When a failover occurs, no metadata is lost.
- **Data RPO:** If your system is healthy and caught up on existing traffic at the time of regional failover, the RPO for events is about 5 minutes.

## Recovery time objective (RTO)

- **Metadata RTO:** Though generally it happens much more quickly, within 60 minutes, Event Grid will begin to accept create/update/delete calls for topics and subscriptions.
- **Data RTO:** Like metadata, it generally happens much more quickly, however within 60 minutes, Event Grid will begin accepting new traffic after a regional failover.

### NOTE

The cost for metadata GeoDR on Event Grid is: \$0.

## Next steps

If you want to implement your own client-side failover logic, see [# Build your own disaster recovery for custom topics in Event Grid](#)

# Azure Policy Regulatory Compliance controls for Azure Event Grid

5/14/2021 • 2 minutes to read • [Edit Online](#)

Regulatory Compliance in Azure Policy provides Microsoft created and managed initiative definitions, known as **built-ins**, for the **compliance domains** and **security controls** related to different compliance standards. This page lists the **compliance domains** and **security controls** for Azure Event Grid. You can assign the built-ins for a **security control** individually to help make your Azure resources compliant with the specific standard.

The title of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Policy Version** column to view the source on the [Azure Policy GitHub repo](#).

## IMPORTANT

Each control below is associated with one or more Azure Policy definitions. These policies may help you [assess compliance](#) with the control; however, there often is not a one-to-one or complete match between a control and one or more policies. As such, **Compliant** in Azure Policy refers only to the policies themselves; this doesn't ensure you're fully compliant with all requirements of a control. In addition, the compliance standard includes controls that aren't addressed by any Azure Policy definitions at this time. Therefore, compliance in Azure Policy is only a partial view of your overall compliance status. The associations between controls and Azure Policy Regulatory Compliance definitions for these compliance standards may change over time.

## Azure Security Benchmark

The [Azure Security Benchmark](#) provides recommendations on how you can secure your cloud solutions on Azure. To see how this service completely maps to the Azure Security Benchmark, see the [Azure Security Benchmark mapping files](#).

To review how the available Azure Policy built-ins for all Azure services map to this compliance standard, see [Azure Policy Regulatory Compliance - Azure Security Benchmark](#).

DOMAIN	CONTROL ID	CONTROL TITLE	POLICY (AZURE PORTAL)	POLICY VERSION (GITHUB)
Network Security	NS-2	Connect private networks together	<a href="#">Azure Event Grid domains should use private link</a>	1.0.2
Network Security	NS-2	Connect private networks together	<a href="#">Azure Event Grid topics should use private link</a>	1.0.2
Network Security	NS-3	Establish private network access to Azure services	<a href="#">Azure Event Grid domains should use private link</a>	1.0.2
Network Security	NS-3	Establish private network access to Azure services	<a href="#">Azure Event Grid topics should use private link</a>	1.0.2

## Next steps

- Learn more about [Azure Policy Regulatory Compliance](#).
- See the built-ins on the [Azure Policy GitHub repo](#).

# Authenticate event delivery to event handlers (Azure Event Grid)

4/21/2021 • 2 minutes to read • [Edit Online](#)

This article provides information on authenticating event delivery to event handlers. It also shows how to secure the webhook endpoints that are used to receive events from Event Grid using Azure Active Directory (Azure AD) or a shared secret.

## Use system-assigned identities for event delivery

You can enable a system-assigned managed identity for a topic or domain and use the identity to forward events to supported destinations such as Service Bus queues and topics, event hubs, and storage accounts.

Here are the steps:

1. Create a topic or domain with a system-assigned identity, or update an existing topic or domain to enable identity.
2. Add the identity to an appropriate role (for example, Service Bus Data Sender) on the destination (for example, a Service Bus queue).
3. When you create event subscriptions, enable the usage of the identity to deliver events to the destination.

For detailed step-by-step instructions, see [Event delivery with a managed identity](#).

## Authenticate event delivery to webhook endpoints

The following sections describe how to authenticate event delivery to webhook endpoints. You need to use a validation handshake mechanism irrespective of the method you use. See [Webhook event delivery](#) for details.

### Using Azure Active Directory (Azure AD)

You can secure the webhook endpoint that's used to receive events from Event Grid by using Azure AD. You'll need to create an Azure AD application, create a role and service principal in your application authorizing Event Grid, and configure the event subscription to use the Azure AD application. Learn how to [Configure Azure Active Directory with Event Grid](#).

### Using client secret as a query parameter

You can also secure your webhook endpoint by adding query parameters to the webhook destination URL specified as part of creating an Event Subscription. Set one of the query parameters to be a client secret such as an [access token](#) or a shared secret. Event Grid service includes all the query parameters in every event delivery request to the webhook. The webhook service can retrieve and validate the secret. If the client secret is updated, event subscription also needs to be updated. To avoid delivery failures during this secret rotation, make the webhook accept both old and new secrets for a limited duration before updating the event subscription with the new secret.

As query parameters could contain client secrets, they are handled with extra care. They are stored as encrypted and are not accessible to service operators. They are not logged as part of the service logs/traces. When retrieving the Event Subscription properties, destination query parameters aren't returned by default. For example: `--include-full-endpoint-url` parameter is to be used in Azure [CLI](#).

For more information on delivering events to webhooks, see [Webhook event delivery](#)

**IMPORTANT**

Azure Event Grid only supports **HTTPS** webhook endpoints.

## Endpoint validation with CloudEvents v1.0

If you're already familiar with Event Grid, you might be aware of the endpoint validation handshake for preventing abuse. CloudEvents v1.0 implements its own [abuse protection semantics](#) by using the **HTTP OPTIONS** method. To read more about it, see [HTTP 1.1 Web Hooks for event delivery - Version 1.0](#). When you use the CloudEvents schema for output, Event Grid uses the CloudEvents v1.0 abuse protection in place of the Event Grid validation event mechanism. For more information, see [Use CloudEvents v1.0 schema with Event Grid](#).

## Next steps

See [Authenticate publishing clients](#) to learn about authenticating clients publishing events to topics or domains.

# Authenticate publishing clients (Azure Event Grid)

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article provides information on authenticating clients that publish events to Azure Event Grid topics or domains using **access key** or **Shared Access Signature (SAS)** token. We recommend using SAS token, but key authentication provides simple programming, and is compatible with many existing webhook publishers.

## Authenticate using an access key

Access key authentication is the simplest form of authentication. You can pass the access key as a HTTP header or a URL query parameter.

### Access key in a HTTP header

Pass the access key as a value for the HTTP header: `aeg-sas-key`.

```
aeg-sas-key: XXXXXXXXXXXXXXXXXXXX0GXXX/nDT4hgdEj9DpBeRr38arnnm50Fg==
```

### Access key as a query parameter

You can also specify `aeg-sas-key` as a query parameter.

```
https://<yourtopic>.<region>.eventgrid.azure.net/api/events?aeg-sas-key=XXXXXXXX53249XX8XXXX0GXXX/nDT4hgdEj9DpBeRr38arnnm50Fg==
```

For instructions on how to get access keys for a topic or domain, see [Get access keys](#).

## Authenticate using a SAS token

SAS tokens for an Event Grid resource include the resource, expiration time, and a signature. The format of the SAS token is: `r={resource}&e={expiration}&s={signature}`.

The resource is the path for the event grid topic to which you're sending events. For example, a valid resource path is: `https://<yourtopic>.<region>.eventgrid.azure.net/api/events`. To see all the supported API versions, see [Microsoft.EventGrid resource types](#).

First, programmatically generate a SAS token and then use the `aeg-sas-token` header or `Authorization SharedAccessSignature` header to authenticate with Event Grid.

### Generate SAS token programmatically

The following example creates a SAS token for use with Event Grid:

```

static string BuildSharedAccessSignature(string resource, DateTime expirationUtc, string key)
{
    const char Resource = 'r';
    const char Expiration = 'e';
    const char Signature = 's';

    string encodedResource = HttpUtility.UrlEncode(resource);
    var culture = CultureInfo.CreateSpecificCulture("en-US");
    var encodedExpirationUtc = HttpUtility.UrlEncode(expirationUtc.ToString(culture));

    string unsignedSas = $"{Resource}={encodedResource}&{Expiration}={encodedExpirationUtc}";
    using (var hmac = new HMACSHA256(Convert.FromBase64String(key)))
    {
        string signature = Convert.ToBase64String(hmac.ComputeHash(Encoding.UTF8.GetBytes(unsignedSas)));
        string encodedSignature = HttpUtility.UrlEncode(signature);
        string signedSas = $"{unsignedSas}&{Signature}={encodedSignature}";

        return signedSas;
    }
}

```

```

def generate_sas_token(uri, key, expiry=3600):
    ttl = datetime.datetime.utcnow() + datetime.timedelta(seconds=expiry)
    encoded_resource = urllib.parse.quote_plus(uri)
    encoded_expiration_utc = urllib.parse.quote_plus(ttl.isoformat())

    unsigned_sas = f'r={encoded_resource}&e={encoded_expiration_utc}'
    signature = b64encode(HMAC(b64decode(key), unsigned_sas.encode('utf-8'), sha256).digest())
    encoded_signature = urllib.parse.quote_plus(signature)

    token = f'r={encoded_resource}&e={encoded_expiration_utc}&s={encoded_signature}'

    return token

```

## Using aeg-sas-token header

Here's an example of passing the SAS token as a value for the `aeg-sas-token` header.

```

aeg-sas-token:
r=https%3a%2f%2fmytopic.eventgrid.azure.net%2fapi%2fevents&e=6%2f15%2f2017+6%3a20%3a15+PM&s=XXXXXXXXXXXXXX%2f
BPjdDL0rc6THPy3tDcGHw1zP40ajQ%3d

```

## Using Authorization header

Here's an example of passing the SAS token as a value for the `Authorization` header.

```

Authorization: SharedAccessSignature
r=https%3a%2f%2fmytopic.eventgrid.azure.net%2fapi%2fevents&e=6%2f15%2f2017+6%3a20%3a15+PM&s=XXXXXXXXXXXXXX%2f
BPjdDL0rc6THPy3tDcGHw1zP40ajQ%3d

```

## Next steps

See [Event delivery authentication](#) to learn about authentication with event handlers to deliver events.

# Authorizing access to Event Grid resources

5/11/2021 • 3 minutes to read • [Edit Online](#)

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).

## NOTE

EventGrid doesn't support Azure RBAC for publishing events to Event Grid topics or domains. Use a Shared Access Signature (SAS) key or token to authenticate clients that publish events. For more information, see [Authenticate publishing clients](#).

## Operation types

For a list of operation supported by Azure Event Grid, run the following Azure CLI command:

```
az provider operation show --namespace Microsoft.EventGrid
```

The following operations return potentially secret information, which gets filtered out of normal read operations. It's recommended that you restrict access to these operations.

- Microsoft.EventGrid/eventSubscriptions/getFullUrl/action
- Microsoft.EventGrid/topics/listKeys/action
- Microsoft.EventGrid/topics/regenerateKey/action

## Built-in roles

Event Grid provides the following three built-in roles.

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.

ROLE	DESCRIPTION
<a href="#">Event Grid Subscription Reader</a>	Lets you read Event Grid event subscriptions.
<a href="#">Event Grid Subscription Contributor</a>	Lets you manage Event Grid event subscription operations.
<a href="#">Event Grid Contributor</a>	Lets you create and manage Event Grid resources.

## NOTE

Select links in the first column to navigate to an article that provides more details about the role. For instructions on how to assign users or groups to RBAC roles, see [this article](#).

## Custom roles

If you need to specify permissions that are different than the built-in roles, you can create custom roles.

The following are sample Event Grid role definitions that allow users to take different actions. These custom roles are different from the built-in roles because they grant broader access than just event subscriptions.

**EventGridReadOnlyRole.json:** Only allow read-only operations.

```
{
  "Name": "Event grid read only role",
  "Id": "7C0B6B59-A278-4B62-BA19-411B70753856",
  "IsCustom": true,
  "Description": "Event grid read only role",
  "Actions": [
    "Microsoft.EventGrid/*/read"
  ],
  "NotActions": [
  ],
  "AssignableScopes": [
    "/subscriptions/<Subscription Id>"
  ]
}
```

**EventGridNoDeleteListKeysRole.json:** Allow restricted post actions but disallow delete actions.

```
{
  "Name": "Event grid No Delete Listkeys role",
  "Id": "B9170838-5F9D-4103-A1DE-60496F7C9174",
  "IsCustom": true,
  "Description": "Event grid No Delete Listkeys role",
  "Actions": [
    "Microsoft.EventGrid/*/write",
    "Microsoft.EventGrid/eventSubscriptions/getFullUrl/action",
    "Microsoft.EventGrid/topics/listkeys/action",
    "Microsoft.EventGrid/topics/regenerateKey/action"
  ],
  "NotActions": [
    "Microsoft.EventGrid/*/delete"
  ],
  "AssignableScopes": [
    "/subscriptions/<Subscription id>"
  ]
}
```

**EventGridContributorRole.json:** Allows all event grid actions.

```
{
  "Name": "Event grid contributor role",
  "Id": "4BA6FB33-2955-491B-A74F-53C9126C9514",
  "IsCustom": true,
  "Description": "Event grid contributor role",
  "Actions": [
    "Microsoft.EventGrid/*/write",
    "Microsoft.EventGrid/*/delete",
    "Microsoft.EventGrid/topics/listkeys/action",
    "Microsoft.EventGrid/topics/regenerateKey/action",
    "Microsoft.EventGrid/eventSubscriptions/getFullUrl/action"
  ],
  "NotActions": [],
  "AssignableScopes": [
    "/subscriptions/<Subscription id>"
  ]
}
```

You can create custom roles with [PowerShell](#), [Azure CLI](#), and [REST](#).

### Encryption at rest

All events or data written to disk by the Event Grid service is encrypted by a Microsoft-managed key ensuring that it's encrypted at rest. Additionally, the maximum period of time that events or data retained is 24 hours in adherence with the [Event Grid retry policy](#). Event Grid will automatically delete all events or data after 24 hours, or the event time-to-live, whichever is less.

## 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.

### System topics (Azure service publishers)

For system topics, if you are not the owner or contributor of the source resource, you 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}
```

For example, to subscribe to an event on a storage account named **myacct**, you need the **Microsoft.EventGrid/EventSubscriptions/Write** permission on:

```
/subscriptions/####/resourceGroups/testrg/providers/Microsoft.Storage/storageAccounts/myacct
```

### Custom topics

For custom topics, you 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}
```

For example, to subscribe to a custom topic named **mytopic**, you need the **Microsoft.EventGrid/EventSubscriptions/Write** permission on:

```
/subscriptions/####/resourceGroups/testrg/providers/Microsoft.EventGrid/topics/mytopic
```

## Next steps

- For an introduction to Event Grid, see [About Event Grid](#)

# Network security for Azure Event Grid resources

3/30/2021 • 5 minutes to read • [Edit Online](#)

This article describes how to use the following security features with Azure Event Grid:

- Service tags for egress
- IP Firewall rules for ingress
- Private endpoints for ingress

## Service tags

A service tag represents a group of IP address prefixes from a given Azure service. Microsoft manages the address prefixes encompassed by the service tag and automatically updates the service tag as addresses change, minimizing the complexity of frequent updates to network security rules. For more information about service tags, see [Service tags overview](#).

You can use service tags to define network access controls on [network security groups](#) or [Azure Firewall](#). Use service tags in place of specific IP addresses when you create security rules. By specifying the service tag name (for example, `AzureEventGrid`) in the appropriate *source* or *destination* field of a rule, you can allow or deny the traffic for the corresponding service.

SERVICE TAG	PURPOSE	CAN USE INBOUND OR OUTBOUND?	CAN BE REGIONAL?	CAN USE WITH AZURE FIREWALL?
AzureEventGrid	Azure Event Grid.	Both	No	No

## IP firewall

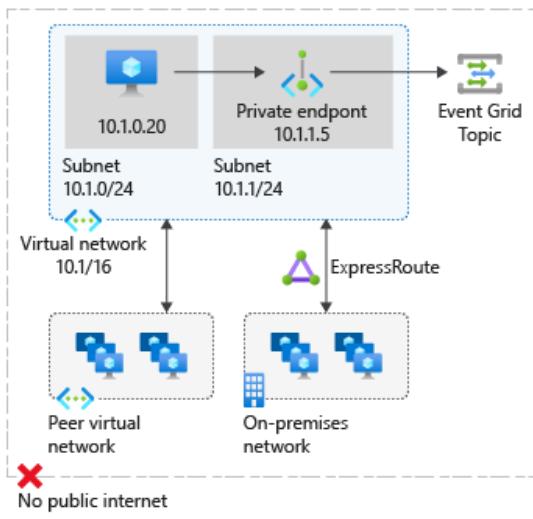
Azure Event Grid supports IP-based access controls for publishing to topics and domains. With IP-based controls, you can limit the publishers to a topic or domain to only a set of approved set of machines and cloud services. This feature complements the [authentication mechanisms](#) supported by Event Grid.

By default, topic and domain are accessible from the internet as long as the request comes with valid authentication and authorization. With IP firewall, you can restrict it further to only a set of IP addresses or IP address ranges in [CIDR \(Classless Inter-Domain Routing\)](#) notation. Publishers originating from any other IP address will be rejected and will receive a 403 (Forbidden) response.

For step-by-step instructions to configure IP firewall for topics and domains, see [Configure IP firewall](#).

## Private endpoints

You can use [private endpoints](#) to allow ingress of events directly from your virtual network to your topics and domains securely over a [private link](#) without going through the public internet. A private endpoint is a special network interface for an Azure service in your VNet. When you create a private endpoint for your topic or domain, it provides secure connectivity between clients on your VNet and your Event Grid resource. The private endpoint is assigned an IP address from the IP address range of your VNet. The connection between the private endpoint and the Event Grid service uses a secure private link.



Using private endpoints for your Event Grid resource enables you to:

- Secure access to your topic or domain from a VNet over the Microsoft backbone network as opposed to the public internet.
- Securely connect from on-premises networks that connect to the VNet using VPN or Express Routes with private-peering.

When you create a private endpoint for a topic or domain in your VNet, a consent request is sent for approval to the resource owner. If the user requesting the creation of the private endpoint is also an owner of the resource, this consent request is automatically approved. Otherwise, the connection is in **pending** state until approved. Applications in the VNet can connect to the Event Grid service over the private endpoint seamlessly, using the same connection strings and authorization mechanisms that they would use otherwise. Resource owners can manage consent requests and the private endpoints, through the **Private endpoints** tab for the resource in the Azure portal.

### Connect to private endpoints

Publishers on a VNet using the private endpoint should use the same connection string for the topic or domain as clients connecting to the public endpoint. DNS resolution automatically routes connections from the VNet to the topic or domain over a private link. Event Grid creates a [private DNS zone](#) attached to the VNet with the necessary update for the private endpoints, by default. However, if you're using your own DNS server, you may need to make additional changes to your DNS configuration.

### DNS changes for private endpoints

When you create a private endpoint, the DNS CNAME record for the resource is updated to an alias in a subdomain with the prefix `privatelink`. By default, a private DNS zone is created that corresponds to the private link's subdomain.

When you resolve the topic or domain endpoint URL from outside the VNet with the private endpoint, it resolves to the public endpoint of the service. The DNS resource records for 'topicA', when resolved from **outside the VNet** hosting the private endpoint, will be:

NAME	TYPE	VALUE
topicA.westus.eventgrid.azure.net	CNAME	topicA.westus.privatelink.eventgrid.azure.net
topicA.westus.privatelink.eventgrid.azure.net	CNAME	<Azure traffic manager profile>

You can deny or control access for a client outside the VNet through the public endpoint using the [IP firewall](#).

When resolved from the VNet hosting the private endpoint, the topic or domain endpoint URL resolves to the private endpoint's IP address. The DNS resource records for the topic 'topicA', when resolved from **inside the VNet** hosting the private endpoint, will be:

NAME	TYPE	VALUE
topicA.westus.eventgrid.azure.net	CNAME	topicA.westus.privateLink.eventgrid.azure.net
topicA.westus.privateLink.eventgrid.azure.net		10.0.0.5

This approach enables access to the topic or domain using the same connection string for clients on the VNet hosting the private endpoints, and clients outside the VNet.

If you're using a custom DNS server on your network, clients can resolve the FQDN for the topic or domain endpoint to the private endpoint IP address. Configure your DNS server to delegate your private link subdomain to the private DNS zone for the VNet, or configure the A records for `topicOrDomainName.regionName.privateLink.eventgrid.azure.net` with the private endpoint IP address.

The recommended DNS zone name is `privatelink.eventgrid.azure.net`.

### Private endpoints and publishing

The following table describes the various states of the private endpoint connection and the effects on publishing:

CONNECTION STATE	SUCCESSFULLY PUBLISH (YES/NO)
Approved	Yes
Rejected	No
Pending	No
Disconnected	No

For publishing to be successful, the private endpoint connection state should be **approved**. If a connection is rejected, it can't be approved using the Azure portal. The only possibility is to delete the connection and create a new one instead.

## Pricing and quotas

**Private endpoints** is available in both basic and premium tiers of Event Grid. Event Grid allows up to 64 private endpoint connections to be created per topic or domain.

**IP Firewall** feature is available in both basic and premium tiers of Event Grid. We allow up to 16 IP Firewall rules to be created per topic or domain.

## Next steps

You can configure IP firewall for your Event Grid resource to restrict access over the public internet from only a select set of IP Addresses or IP Address ranges. For step-by-step instructions, see [Configure IP firewall](#).

You can configure private endpoints to restrict access from only from selected virtual networks. For step-by-step instructions, see [Configure private endpoints](#).

To troubleshoot network connectivity issues, see [Troubleshoot network connectivity issues](#)

# Diagnostic logs for Azure Event Grid topics/domains

4/30/2021 • 2 minutes to read • [Edit Online](#)

Diagnostic settings allow Event Grid users to capture and view **publish and delivery failure** logs in either a Storage account, an event hub, or a Log Analytics Workspace. This article provides schema for the logs and an example log entry.

## Schema for publish/delivery failure logs

PROPERTY NAME	DATA TYPE	DESCRIPTION
Time	DateTime	The time when the log entry was generated <b>Example value:</b> 01-29-2020 09:52:02.700
EventSubscriptionName	String	The name of the event subscription <b>Example value:</b> "EVENTSUB1" This property exists only for delivery failure logs.
Category	String	The log category name. <b>Example values:</b> "DeliveryFailures" or "PublishFailures"
OperationName	String	The name of the operation caused the failure. <b>Example Values:</b> "Deliver" for delivery failures.
Message	String	The log message for the user explaining the reason for the failure and other additional details.
ResourceId	String	The resource ID for the topic/domain resource <b>Example Values:</b> /SUBSCRIPTIONS/SAMPLE-SUBSCRIPTION-ID/RESOURCEGROUPS/SAMPLE-RESOURCEGROUP/PROVIDERS/MICROSOFT.EVENTGRID/TOPICS

## Example

```
{  
    "time": "2019-11-01T00:17:13.4389048Z",  
    "resourceId": "/SUBSCRIPTIONS/SAMPLE-SUBSCRIPTION-ID /RESOURCEGROUPS/SAMPLE-RESOURCEGROUP-  
NAME/PROVIDERS/MICROSOFT.EVENTGRID/TOPICS/SAMPLE-TOPIC-NAME ",  
    "eventSubscriptionName": "SAMPLEDESTINATION",  
    "category": "DeliveryFailures",  
    "operationName": "Deliver",  
    "message": "Message:outcome=NotFound, latencyInMs=2635, id=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx,  
systemId=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxx, state=FilteredFailingDelivery, deliveryTime=11/1/2019 12:17:10  
AM, deliveryCount=0, probationCount=0, deliverySchema=EventGridEvent,  
eventSubscriptionDeliverySchema=EventGridEvent, fields=InputEvent, EventSubscriptionId, DeliveryTime, State,  
Id, DeliverySchema, LastDeliveryAttemptTime, SystemId, fieldCount=, requestExpiration=1/1/0001 12:00:00 AM,  
delivered=False publishTime=11/1/2019 12:17:10 AM, eventTime=11/1/2019 12:17:09 AM, eventType=Type,  
deliveryTime=11/1/2019 12:17:10 AM, filteringState=FilteredWithRpc, inputSchema=EventGridEvent,  
publisher=DIAGNOSTICLOGSTEST-EASTUS.EASTUS-1.EVENTGRID.AZURE.NET, size=363, fields=Id, PublishTime,  
SerializedBody, EventType, Topic, Subject, FilteringHashCode, SystemId, Publisher, FilteringTopic,  
TopicCategory, DataVersion, MetadataVersion, InputSchema, EventTime, fieldCount=15,  
url=sb://diagnosticlogstesting-eastus.servicebus.windows.net/, deliveryResponse=NotFound: The messaging  
entity 'sb://diagnosticlogstesting-eastus.servicebus.windows.net/eh-diagnosticlogstest' could not be found.  
TrackingId:c98c5af6-11f0-400b-8f56-c605662fb849_G14, SystemTracker:diagnosticlogstesting-  
eastus.servicebus.windows.net:eh-diagnosticlogstest, Timestamp:2019-11-01T00:17:13, referenceId:  
ac141738a9a54451b12b4cc31a10dedc_G14:"  
}
```

The possible values of `Outcome` are `Aborted`, `TimedOut`, `GenericError`, and `Busy`. Event Grid logs any information it receives from the event handler in the `message`. For example, for `GenericError`, it logs the HTTP status code, error code, and the error message.

## Next steps

To learn how to enable diagnostic logs for topics or domains, see [Enable diagnostic logs](#).

# Metrics supported by Azure Event Grid

3/18/2021 • 5 minutes to read • [Edit Online](#)

This article provides lists of Event Grid metrics that are categorized by namespaces.

## System topics

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
AdvancedFilterEvaluationCount	Yes	Advanced Filter Evaluations	Count	Total	Total advanced filters evaluated across event subscriptions for this topic.	EventSubscriptionName
DeadLetteredCount	Yes	Dead Lettered Events	Count	Total	Total dead lettered events matching to this event subscription	DeadLetterReason, EventSubscriptionName
DeliveryAttemptFailCount	No	Delivery Failed Events	Count	Total	Total events failed to deliver to this event subscription	Error, ErrorType, EventSubscriptionName
DeliverySuccessCount	Yes	Delivered Events	Count	Total	Total events delivered to this event subscription	EventSubscriptionName
DestinationProcessingDurationInMs	No	Destination Processing Duration	Milliseconds	Average	Destination processing duration in milliseconds	EventSubscriptionName
DroppedEventCount	Yes	Dropped Events	Count	Total	Total dropped events matching to this event subscription	DropReason, EventSubscriptionName
MatchedEventCount	Yes	Matched Events	Count	Total	Total events matched to this event subscription	EventSubscriptionName

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
PublishFailCount	Yes	Publish Failed Events	Count	Total	Total events failed to publish to this topic	ErrorType, Error
PublishSuccessCount	Yes	Published Events	Count	Total	Total events published to this topic	No Dimensions
PublishSuccessLatencyInMs	Yes	Publish Success Latency	Milliseconds	Total	Publish success latency in milliseconds	No Dimensions
UnmatchedEventCount	Yes	Unmatched Events	Count	Total	Total events not matching any of the event subscriptions for this topic	No Dimensions

## Custom topics

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
AdvancedFilterEvaluationCount	Yes	Advanced Filter Evaluations	Count	Total	Total advanced filters evaluated across event subscriptions for this topic.	EventSubscriptionName
DeadLetteredCount	Yes	Dead Lettered Events	Count	Total	Total dead lettered events matching to this event subscription	DeadLetterReason, EventSubscriptionName
DeliveryAttemptFailCount	No	Delivery Failed Events	Count	Total	Total events failed to deliver to this event subscription	Error, ErrorType, EventSubscriptionName
DeliverySuccessCount	Yes	Delivered Events	Count	Total	Total events delivered to this event subscription	EventSubscriptionName

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
DestinationProcessingDurationInMs	No	Destination Processing Duration	Milliseconds	Average	Destination processing duration in milliseconds	EventSubscriptionName
DroppedEventCount	Yes	Dropped Events	Count	Total	Total dropped events matching to this event subscription	DropReason, EventSubscriptionName
MatchedEventCount	Yes	Matched Events	Count	Total	Total events matched to this event subscription	EventSubscriptionName
PublishFailCount	Yes	Publish Failed Events	Count	Total	Total events failed to publish to this topic	ErrorType, Error
PublishSuccessCount	Yes	Published Events	Count	Total	Total events published to this topic	No Dimensions
PublishSuccessLatencyInMs	Yes	Publish Success Latency	Milliseconds	Total	Publish success latency in milliseconds	No Dimensions
UnmatchedEventCount	Yes	Unmatched Events	Count	Total	Total events not matching any of the event subscriptions for this topic	No Dimensions

## Domains

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
AdvancedFilterEvaluationCount	Yes	Advanced Filter Evaluations	Count	Total	Total advanced filters evaluated across event subscriptions for this topic.	Topic, EventSubscriptionName, DomainEventSubscriptionName

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
DeadLetteredCount	Yes	Dead Lettered Events	Count	Total	Total dead lettered events matching to this event subscription	Topic, EventSubscriptionName, DomainEvent SubscriptionName, DeadLetterReason
DeliveryAttemptFailCount	No	Delivery Failed Events	Count	Total	Total events failed to deliver to this event subscription	Topic, EventSubscriptionName, DomainEvent SubscriptionName, Error, ErrorType
DeliverySuccessCount	Yes	Delivered Events	Count	Total	Total events delivered to this event subscription	Topic, EventSubscriptionName, DomainEvent SubscriptionName
DestinationProcessingDurationInMs	No	Destination Processing Duration	Milliseconds	Average	Destination processing duration in milliseconds	Topic, EventSubscriptionName, DomainEvent SubscriptionName
DroppedEventCount	Yes	Dropped Events	Count	Total	Total dropped events matching to this event subscription	Topic, EventSubscriptionName, DomainEvent SubscriptionName, DropReason
MatchedEventCount	Yes	Matched Events	Count	Total	Total events matched to this event subscription	Topic, EventSubscriptionName, DomainEvent SubscriptionName
PublishFailCount	Yes	Publish Failed Events	Count	Total	Total events failed to publish to this topic	Topic, ErrorType, Error
PublishSuccessCount	Yes	Published Events	Count	Total	Total events published to this topic	Topic

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
PublishSuccessLatencyInMs	Yes	Publish Success Latency	Milliseconds	Total	Publish success latency in milliseconds	No Dimensions

## Event subscriptions

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
DeadLetteredCount	Yes	Dead Lettered Events	Count	Total	Total dead lettered events matching to this event subscription	DeadLetterReason
DeliveryAttemptFailCount	No	Delivery Failed Events	Count	Total	Total events failed to deliver to this event subscription	Error, ErrorType
DeliverySuccessCount	Yes	Delivered Events	Count	Total	Total events delivered to this event subscription	No Dimensions
DestinationProcessingDurationInMs	No	Destination Processing Duration	Milliseconds	Average	Destination processing duration in milliseconds	No Dimensions
DroppedEventCount	Yes	Dropped Events	Count	Total	Total dropped events matching to this event subscription	DropReason
MatchedEventCount	Yes	Matched Events	Count	Total	Total events matched to this event subscription	No Dimensions

## Extension topics

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
PublishFailCount	Yes	Publish Failed Events	Count	Total	Total events failed to publish to this topic	ErrorType, Error
PublishSuccessCount	Yes	Published Events	Count	Total	Total events published to this topic	No Dimensions
PublishSuccessLatencyInMs	Yes	Publish Success Latency	Milliseconds	Total	Publish success latency in milliseconds	No Dimensions
UnmatchedEventCount	Yes	Unmatched Events	Count	Total	Total events not matching any of the event subscriptions for this topic	No Dimensions

## Partner namespaces

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
DeadLetteredCount	Yes	Dead Lettered Events	Count	Total	Total dead lettered events matching to this event subscription	DeadLetterReason, EventSubscriptionName
DeliveryAttemptFailCount	No	Delivery Failed Events	Count	Total	Total events failed to deliver to this event subscription	Error, ErrorType, EventSubscriptionName
DeliverySuccessCount	Yes	Delivered Events	Count	Total	Total events delivered to this event subscription	EventSubscriptionName
DestinationProcessingDurationInMs	No	Destination Processing Duration	Milliseconds	Average	Destination processing duration in milliseconds	EventSubscriptionName

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
DroppedEventCount	Yes	Dropped Events	Count	Total	Total dropped events matching to this event subscription	DropReason, EventSubscriptionName
MatchedEventCount	Yes	Matched Events	Count	Total	Total events matched to this event subscription	EventSubscriptionName
PublishFailCount	Yes	Publish Failed Events	Count	Total	Total events failed to publish to this topic	ErrorType, Error
PublishSuccessCount	Yes	Published Events	Count	Total	Total events published to this topic	No Dimensions
PublishSuccessLatencyInMs	Yes	Publish Success Latency	Milliseconds	Total	Publish success latency in milliseconds	No Dimensions
UnmatchedEventCount	Yes	Unmatched Events	Count	Total	Total events not matching any of the event subscriptions for this topic	No Dimensions

## Partner topics

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
AdvancedFilterEvaluationCount	Yes	Advanced Filter Evaluations	Count	Total	Total advanced filters evaluated across event subscriptions for this topic.	EventSubscriptionName
DeadLetterCount	Yes	Dead Lettered Events	Count	Total	Total dead lettered events matching to this event subscription	DeadLetterReason, EventSubscriptionName

METRIC	EXPORTABLE VIA DIAGNOSTIC SETTINGS?	METRIC DISPLAY NAME	UNIT	AGGREGATION TYPE	DESCRIPTION	DIMENSIONS
DeliveryAttemptFailCount	No	Delivery Failed Events	Count	Total	Total events failed to deliver to this event subscription	Error, ErrorType, EventSubscriptionName
DeliverySuccessCount	Yes	Delivered Events	Count	Total	Total events delivered to this event subscription	EventSubscriptionName
DestinationProcessingDurationInMs	No	Destination Processing Duration	Milliseconds	Average	Destination processing duration in milliseconds	EventSubscriptionName
DroppedEventCount	Yes	Dropped Events	Count	Total	Total dropped events matching to this event subscription	DropReason, EventSubscriptionName
MatchedEventCount	Yes	Matched Events	Count	Total	Total events matched to this event subscription	EventSubscriptionName
PublishFailCount	Yes	Publish Failed Events	Count	Total	Total events failed to publish to this topic	ErrorType, Error
PublishSuccessCount	Yes	Published Events	Count	Total	Total events published to this topic	No Dimensions
UnmatchedEventCount	Yes	Unmatched Events	Count	Total	Total events not matching any of the event subscriptions for this topic	No Dimensions

## Next steps

See the following article: [Diagnostic logs](#)

# Use cases for event domains in Azure Event Grid

3/5/2021 • 2 minutes to read • [Edit Online](#)

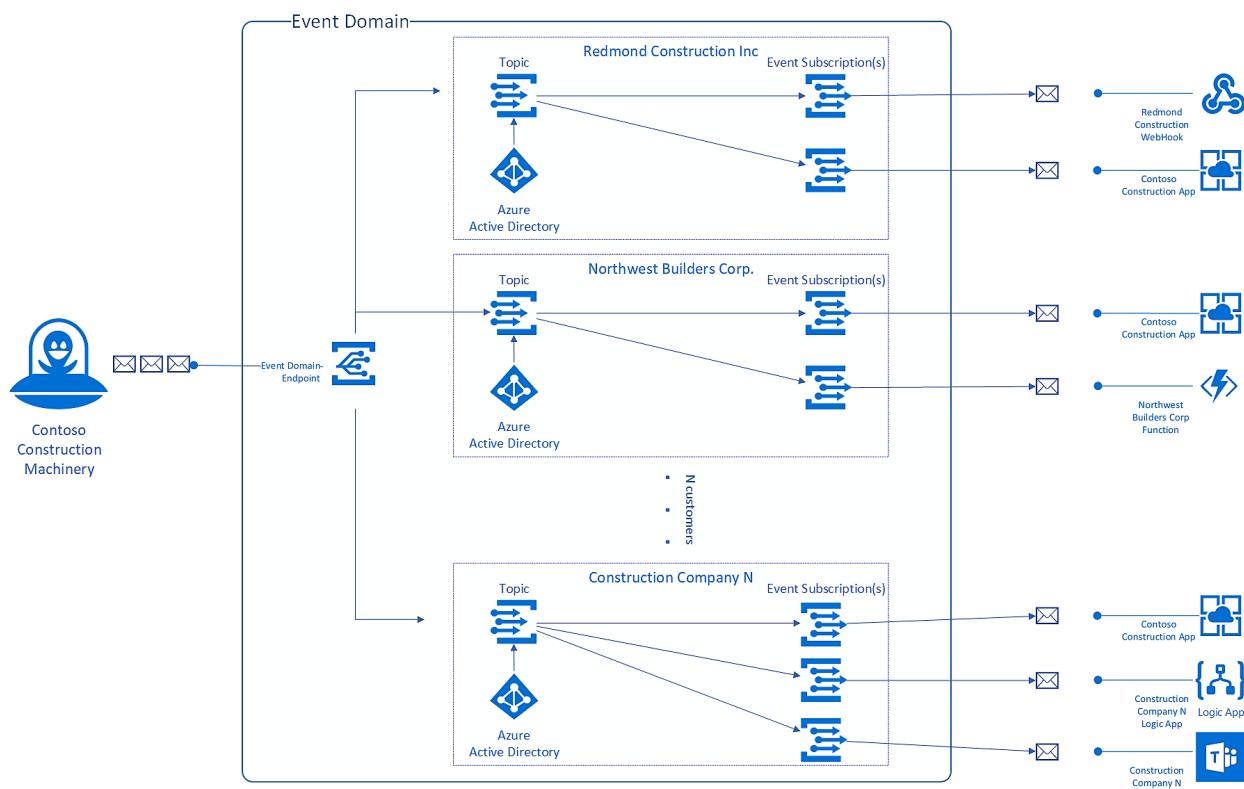
This article describes a few use cases for using event domains in Azure Event Grid.

## Use case 1

Event domains are most easily explained using an example. Let's say you run Contoso Construction Machinery, where you manufacture tractors, digging equipment, and other heavy machinery. As a part of running the business, you push real-time information to customers about equipment maintenance, systems health, and contract updates. All of this information goes to various endpoints including your app, customer endpoints, and other infrastructure that customers have set up.

Event domains allow you to model Contoso Construction Machinery as a single eventing entity. Each of your customers is represented as a topic within the domain. Authentication and authorization are handled using Azure Active Directory. Each of your customers can subscribe to their topic and get their events delivered to them. Managed access through the event domain ensures they can only access their topic.

It also gives you a single endpoint, which you can publish all of your customer events to. Event Grid will take care of making sure each topic is only aware of events scoped to its tenant.

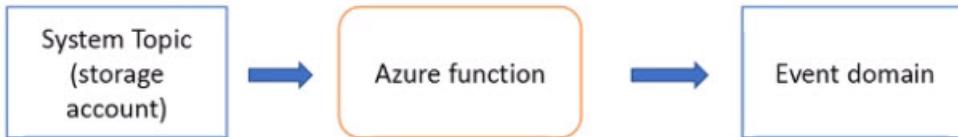


## Use case 2

There is a limit of 500 event subscriptions when using system topics. If you need more than 500 event subscriptions for a system topic, you could use domains.

Suppose, you have created a system topic for an Azure Blob Storage and you need to create more than 500 subscriptions to the topic, but it's not possible because of the limitation (500 subscriptions per topic). In this case, you could use the following solution that uses an event domain.

1. Create a domain, which can support up to 100,000 topics and each domain topic can have 500 event subscriptions. This model would give you  $500 * 100,000$  event subscriptions.
2. Create an Azure function subscription for the Azure Storage system topic. When function receives events from Azure storage, it can enrich and publish events to an appropriate domain topic. For example, the function could parse the blob file name to determine the target domain topic and publish the event to domain topic.



## Next steps

To learn about setting up event domains, creating topics, creating event subscriptions, and publishing events, see [Manage event domains](#).

# Create, view, and manage Event Grid system topics in the Azure portal

11/2/2020 • 3 minutes to read • [Edit Online](#)

This article shows you how to create and manage system topics using the Azure portal. For an overview of system topics, see [System topics](#).

## Create a system topic

You can create a system topic for an Azure resource (Storage account, Event Hubs namespace, etc.) in two ways:

- Using the **Events** page of a resource, for example, Storage Account or Event Hubs Namespace. When you use the **Events** page in the Azure portal to create an event subscription for an event raised by an Azure source (for example: Azure Storage account), the portal creates a system topic for the Azure resource and then creates a subscription for the system topic. You specify the name of the system topic if you are creating an event subscription on the Azure resource for the first time. From the second time onwards, the system topic name is displayed for you in the read-only mode. See [Quickstart: Route Blob storage events to web endpoint with the Azure portal](#) for detailed steps.
- Using the **Event Grid System Topics** page. The following steps are for creating a system topic using the **Event Grid System Topics** page.

1. Sign in to [Azure portal](#).
2. In the search box at the top, type **Event Grid System Topics**, and then press **ENTER**.

Event Grid System Topics

Services

- Event Grid System Topics
- Event Grid Partner Topics
- Event Grid Topics
- Recent
- Event Grid Domains
- Event Grid Subscriptions
- Event Hubs
- SendGrid Accounts
- Management groups
- Event Hubs Clusters

Resources

No results were found.

Resource Groups

No results were found.

Documentation

- [Azure Event Grid concepts | Microsoft Docs](#)
- [System topics in Azure Event Grid | Microsoft Docs](#)
- [What is Azure Event Grid? | Microsoft Docs](#)
- [Quickstart: Send custom events with Event Grid and Azure ...](#)

Marketplace

Searching all subscriptions. [Change](#)

- On the Event Grid System Topics page, select + Add on the toolbar.

Event Grid System Topics

+ Add Manage view Refresh Export to CSV Assign tags Feedback Leave preview

Name	Topic Type	Location	Resource group	Subscription
contosohub-systemtopic	MICROSOFT.EVENTHUB.NAMESPACES	East US	egridgroup	Visual Studio Ultimate with MSDN
resource-group-egridgroup-syst...	MICROSOFT.RESOURCES.RESOURCEGROUPS	global	egridgroup	Visual Studio Ultimate with MSDN

- On the Create Event Grid System Topic page, do the following steps:

- Select the **topic type**. In the following example, **Storage Accounts** option is selected.
- Select the **Azure subscription** that has your storage account resource.
- Select the **resource group** that has the storage account.
- Select the **storage account**.
- Enter a **name** for the system topic to be created.

**NOTE**

You can use this system topic name to search metrics and diagnostic logs.

f. Select **Review + create**.

 **Create Event Grid System Topic**  
Event Grid

Basics Tags Review + create

**TOPIC DETAILS**

System topic resource is associated with an existing azure resource which allows customer to subscribe events emitted by that resource. System topic resource is created in the same subscription and resource group as the source.

Topic Type: Storage Accounts (Blob & GPv2) ▾

Subscription \*: Visual Studio Ultimate with MSDN ▾

Resource group \*: egridgroup ▾

Resource \*: spegstorage2 ▾

**SYSTEM TOPIC DETAILS**

Enter required settings for this system topic.

Name \*: spegstorage-system-topic ✓

**Review + create** < Previous Next: Tags >

g. Review settings and select **Create**.

The screenshot shows the 'Create Event Grid System Topic' wizard. At the top, there's a green success message: 'Validation succeeded.' Below it, the tabs 'Basics', 'Tags', and 'Review + create' are shown, with 'Review + create' being the active tab. The topic details are listed under 'Event Grid System Topic by Microsoft'. Under the 'Basics' section, the following values are set:

Name	spegstorage-system-topic
Subscription	Visual Studio Ultimate with MSDN
Resource group	egridgroup
Source Resource	spegstorage2
Location	eastus

At the bottom, there are three buttons: 'Create' (highlighted in blue), '< Previous', and 'Next >'.

- h. After the deployment succeeds, select **Go to resource** to see the **Event Grid System Topic** page for the system topic you created.

The screenshot shows the 'spegstorage-system-topic' Overview page in the Azure portal. The left sidebar includes 'Overview', 'Activity log', 'Access control (IAM)', 'Tags', 'Properties', 'Locks', 'Export template', 'Metrics', 'Diagnostic settings', 'Support + troubleshooting', and 'New support request'. The main area displays the following information:

- Resource group:** egridgroup
- Status:** Active
- Location:** East US
- Subscription:** Visual Studio Ultimate with MSDN
- Subscription ID:** <Azure subscription ID>
- Tags:** Click here to add tags

Below this, there's a chart showing metrics over the last hour, with a legend for General, Errors, and Latency. The chart shows values from 0 to 100. At the bottom, event logs are displayed with columns for Name, Endpoint, Prefix Filter, Suffix Filter, and Event Types. The logs show:

Published Events (Sum)	Published Failed Events (Sum)	Matched Events (Sum)	Delivery Failed Events (Sum)	Dropped Events (Sum)
1:00 PM	1:00 PM	1:00 PM	1:00 PM	1:00 PM
spegridtopic-sys...	spegridtopic-sys...	spegridtopic-sys...	spegridtopic-sys...	spegridtopic-sys...
--	--	--	--	--

## View all system topics

Follow these steps to view all existing Event Grid system topics.

1. Sign in to [Azure portal](#).
2. In the search box at the top, type **Event Grid System Topics**, and then press **ENTER**.

The screenshot shows the Microsoft Azure portal interface. The top navigation bar has 'Microsoft Azure' on the left and a search bar containing 'Event Grid System Topics' with a red border around it. On the far right of the top bar are three icons: a three-dot menu, a user profile, and a close button.

The main content area has a sidebar on the left with various service icons. The main title is 'Event Grid'. Below it are sections for 'Default Directory', 'Add', 'Manage', and a 'Filter by name...' input field. A 'Showing 0 to 0 of 0 results' message is displayed. There is a 'Name ↑↓' sorting option.

The main content area is divided into two main sections: 'Services' and 'Resources'.

**Services** section:

- Event Grid System Topics (highlighted with a red border)
- Event Grid Partner Topics
- Event Grid Topics
- Recent
- Event Grid Domains
- Event Grid Subscriptions
- Event Hubs
- SendGrid Accounts
- Management groups
- Event Hubs Clusters

**Resources** section:

No results were found.

**Resource Groups** section:

No results were found.

**Documentation** section:

- [Azure Event Grid concepts | Microsoft Docs](#)
- [System topics in Azure Event Grid | Microsoft Docs](#)
- [What is Azure Event Grid? | Microsoft Docs](#)
- [Quickstart: Send custom events with Event Grid and Azure ...](#)

**Marketplace** section:

Searching all subscriptions. [Change](#)

3. On the **Event Grid System Topics** page, you see all the system topics.

The screenshot shows the 'Event Grid System Topics' page. At the top, there's a header with the title 'Event Grid System Topics' and a close button. Below the header are buttons for 'Add', 'Manage view', 'Refresh', 'Export to CSV', 'Assign tags', 'Feedback', and 'Leave preview'. There are also filter buttons for 'Subscription', 'Resource group', 'Location', and 'Add filter', and a dropdown for 'No grouping'.

The main area displays a table with the following data:

<input type="checkbox"/>	Name ↑↓	Topic Type ↑↓	Location ↑↓	Resource group ↑↓	Subscription ↑↓
<input type="checkbox"/>	contosohub-systemtopic	MICROSOFT.EVENTHUB.NAMESPACES	East US	egridgroup	Visual Studio Ultimate with MSDN ...
<input type="checkbox"/>	resource-group-egridgroup-sys...	MICROSOFT.RESOURCES.RESOURCEGROUPS	global	egridgroup	Visual Studio Ultimate with MSDN ...
<input type="checkbox"/>	spegridssytopic1	MICROSOFT.STORAGE.STORAGEACCOUNTS	East US	egridgroup	Visual Studio Ultimate with MSDN ...

4. Select a **system topic** from the list to see details about it.

This page shows you details about the system topic such as the following information:

- Source. Name of the resource on which the system topic was created.
- Source type. Type of the resource. For example: `Microsoft.Storage.StorageAccounts`, `Microsoft.EventHub.Namespaces`, `Microsoft.Resources.ResourceGroups` and so on.
- Any subscriptions created for the system topic.

This page allows operations such as the following ones:

- Create an event subscription Select **+ Event Subscription** on the toolbar.
- Delete an event subscription. Select **Delete** on the toolbar.
- Add tags for the system topic. Select **Tags** on the left menu, and specify tag names and values.

## Delete a system topic

1. Follow instructions from the [View system topics](#) section to view all system topics, and select the system topic that you want to delete from the list.
2. On the **Event Grid System Topic** page, select **Delete** on the toolbar.

3. On the confirmation page, select **OK** to confirm the deletion. It deletes the system topic and also all the event subscriptions for the system topic.

## Create an event subscription

1. Follow instructions from the [View system topics](#) section to view all system topics, and select the system topic that you want to delete from the list.
2. On the **Event Grid System Topic** page, select **+ Event Subscription** from the toolbar.

The screenshot shows the Azure portal interface for managing an Event Grid System Topic. The top navigation bar includes a search bar, a back arrow, and a refresh button. Below the navigation is a toolbar with 'Event Subscription' (highlighted with a red box), 'Delete', and 'Refresh' buttons. The main content area displays the topic details: Resource group (eGRIDgroup), Status (Active), Location (East US), Source (spegridstorage2), and Source Type (Microsoft.Storage.StorageAccounts). There are also links for Activity log, Access control (IAM), and Tags.

3. Confirm that the **Topic Type**, **Source Resource**, and **Topic Name** are automatically populated. Enter a name, select an **Endpoint Type**, and specify the **endpoint**. Then, select **Create** to create the event subscription.

The screenshot shows the 'Create Event Subscription' blade. The 'Basic' tab is selected. In the 'EVENT SUBSCRIPTION DETAILS' section, the 'Name' field contains 'spegridssytopicsubscription'. The 'Event Schema' dropdown is set to 'Event Grid Schema'. In the 'TOPIC DETAILS' section, the 'Topic Type' is 'Storage account', 'Source Resource' is 'spegridstorage2', and 'Topic Name' is 'spegridssytopic1'. Under 'EVENT TYPES', there is a dropdown labeled '2 selected'. In the 'ENDPOINT DETAILS' section, the 'Endpoint Type' is 'Web Hook' and the 'Endpoint' is 'https://spegridssite2.azurewebsites.net/api/uploads'. At the bottom of the blade is a blue 'Create' button.

## Next steps

See the [System topics in Azure Event Grid](#) section to learn more about system topics and topic types supported by Azure Event Grid.

# Create, view, and manage Event Grid system topics using Azure CLI

4/22/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to create and manage system topics using Azure CLI. For an overview of system topics, see [System topics](#).

## Install extension for Azure CLI

For Azure CLI, you need the [Event Grid extension](#).

In Cloud Shell:

- If you've installed the extension previously, update it: `az extension update -n eventgrid`
- If you haven't installed the extension previously, install it: `az extension add -n eventgrid`

For a local installation:

1. [Install the Azure CLI](#). Make sure that you have the latest version, by checking with `az --version`.
2. Uninstall previous versions of the extension: `az extension remove -n eventgrid`
3. Install the eventgrid extension with `az extension add -n eventgrid`

## Create a system topic

- To create a system topic on an Azure source first and then create an event subscription for that topic, see the following reference topics:
  - [az eventgrid system-topic create](#)

```
# Get the ID of the Azure source (for example: Azure Storage account)
storageid=$(az storage account show \
    --name <AZURE STORAGE ACCOUNT NAME> \
    --resource-group <AZURE RESOURCE GROUP NAME> \
    --query id --output tsv)

# Create the system topic on the Azure source (example: Azure Storage account)
az eventgrid system-topic create \
    -g <AZURE RESOURCE GROUP NAME> \
    --name <SPECIFY SYSTEM TOPIC NAME> \
    --location <LOCATION> \
    --topic-type microsoft.storage.storageaccounts \
    --source $storageid
```

For a list of `topic-type` values that you can use to create a system topic, run the following command. These topic type values represent the event sources that support creation of system topics. Please ignore `Microsoft.EventGrid.Topics` and `Microsoft.EventGrid.Domains` from the list.

```
az eventgrid topic-type list --output json | grep -w id
```

- [az eventgrid system-topic event-subscription create](#)

```
az eventgrid system-topic event-subscription create --name <SPECIFY EVENT SUBSCRIPTION NAME> \
-g rg1 --system-topic-name <SYSTEM TOPIC NAME> \
--endpoint <ENDPOINT URL>
```

- To create a system topic (implicitly) when creating an event subscription for an Azure source, use the [az eventgrid event-subscription create](#) method. Here's an example:

```
storageid=$(az storage account show --name <AZURE STORAGE ACCOUNT NAME> --resource-group <AZURE RESOURCE GROUP NAME> --query id --output tsv)
endpoint=<ENDPOINT URL>

az eventgrid event-subscription create \
--source-resource-id $storageid \
--name <EVENT SUBSCRIPTION NAME> \
--endpoint $endpoint
```

For a tutorial with step-by-step instructions, see [Subscribe to storage account](#).

## View all system topics

To view all system topics and details of a selected system topic, use the following commands:

- [az eventgrid system-topic list](#)

```
az eventgrid system-topic list
```

- [az eventgrid system-topic show](#)

```
az eventgrid system-topic show -g <AZURE RESOURCE GROUP NAME> -n <SYSTEM TOPIC NAME>
```

## Delete a system topic

To delete a system topic, use the following command:

- [az eventgrid system-topic delete](#)

```
az eventgrid system-topic delete -g <AZURE RESOURCE GROUP NAME> --name <SYSTEM TOPIC NAME>
```

## Next steps

See the [System topics in Azure Event Grid](#) section to learn more about system topics and topic types supported by Azure Event Grid.

# Create system topics in Azure Event Grid using Resource Manager templates

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article shows you how to create and manage system topics using Resource Manager templates. For an overview of system topics, see [System topics](#).

## Create system topic first and then create event subscription

To create a system topic on an Azure source first and then create an event subscription for that topic, you can use a template like:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageName": {
            "type": "string",
            "defaultValue": "[concat('storage', uniqueString(resourceGroup().id))]",
            "metadata": {
                "description": "Provide a unique name for the Blob Storage account."
            }
        },
        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Provide a location for the Blob Storage account that supports Event Grid."
            }
        },
        "eventSubName": {
            "type": "string",
            "defaultValue": "subToStorage",
            "metadata": {
                "description": "Provide a name for the Event Grid subscription."
            }
        },
        "endpoint": {
            "type": "string",
            "metadata": {
                "description": "Provide the URL for the WebHook to receive events. Create your own endpoint for events."
            }
        },
        "systemTopicName": {
            "type": "String",
            "defaultValue": "mystoragesystemtopic",
            "metadata": {
                "description": "Provide a name for the system topic."
            }
        }
    },
    "resources": [
        {
            "name": "[parameters('storageName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2017-10-01",
            "sku": {
                "name": "Standard_LRS"
            }
        }
    ]
}
```

```

    },
    "kind": "StorageV2",
    "location": "[parameters('location')]",
    "tags": {},
    "properties": {
        "accessTier": "Hot"
    }
},
{
    "name": "[parameters('systemTopicName')]",
    "type": "Microsoft.EventGrid/systemTopics",
    "apiVersion": "2020-04-01-preview",
    "location": "[parameters('location')]",
    "dependsOn": [
        "[parameters('storageName')]"
    ],
    "properties": {
        "source": "[resourceId('Microsoft.Storage/storageAccounts', parameters('storageName'))]",
        "topicType": "Microsoft.Storage.StorageAccounts"
    }
},
{
    "type": "Microsoft.EventGrid/systemTopics/eventSubscriptions",
    "apiVersion": "2020-04-01-preview",
    "name": "[concat(parameters('systemTopicName'), '/', parameters('eventSubName'))]",
    "dependsOn": [
        "[resourceId('Microsoft.EventGrid/systemTopics', parameters('systemTopicName'))]"
    ],
    "properties": {
        "destination": {
            "properties": {
                "endpointUrl": "[parameters('endpoint')]"
            },
            "endpointType": "WebHook"
        },
        "filter": {
            "includedEventTypes": [
                "Microsoft.Storage.BlobCreated",
                "Microsoft.Storage.BlobDeleted"
            ]
        }
    }
}
]
}

```

See [Route Blob storage events to web endpoint by using Azure Resource Manager template](#) for instructions on creating system topics and subscription for them using Resource Manager templates.

## Create system topic while creating an event subscription

To create a system topic implicitly, while creating an event subscription on an Azure source, you could use the following template:

```
{
    "$schema": "https://schema.management.azure.com/schemas/2019-04-01/deploymentTemplate.json#",
    "contentVersion": "1.0.0.0",
    "parameters": {
        "storageName": {
            "type": "string",
            "defaultValue": "[concat('storage', uniqueString(resourceGroup().id))]",
            "metadata": {
                "description": "Provide a unique name for the Blob Storage account."
            }
        },
        "location": [

```

```

        "location": {
            "type": "string",
            "defaultValue": "[resourceGroup().location]",
            "metadata": {
                "description": "Provide a location for the Blob Storage account that supports Event Grid."
            }
        },
        "eventSubName": {
            "type": "string",
            "defaultValue": "subToStorage",
            "metadata": {
                "description": "Provide a name for the Event Grid subscription."
            }
        },
        "endpoint": {
            "type": "string",
            "metadata": {
                "description": "Provide the URL for the WebHook to receive events. Create your own endpoint for events."
            }
        }
    },
    "resources": [
        {
            "name": "[parameters('storageName')]",
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2017-10-01",
            "sku": {
                "name": "Standard_LRS"
            },
            "kind": "StorageV2",
            "location": "[parameters('location')]",
            "tags": {},
            "properties": {
                "accessTier": "Hot"
            }
        },
        {
            "type": "Microsoft.Storage/storageAccounts/providers/eventSubscriptions",
            "name": "[concat(parameters('storageName'), '/Microsoft.EventGrid/',
parameters('eventSubName'))]",
            "apiVersion": "2018-01-01",
            "dependsOn": [
                "[parameters('storageName')]"
            ],
            "properties": {
                "destination": {
                    "endpointType": "WebHook",
                    "properties": {
                        "endpointUrl": "[parameters('endpoint')]"
                    }
                },
                "filter": {
                    "subjectBeginsWith": "",
                    "subjectEndsWith": "",
                    "isSubjectCaseSensitive": false,
                    "includedEventTypes": [
                        "All"
                    ]
                }
            }
        }
    ]
}

```

## Next steps

See the [System topics in Azure Event Grid](#) section to learn more about system topics and topic types supported by Azure Event Grid.

# Onboard as an Azure Event Grid partner using the Azure portal

3/5/2021 • 6 minutes to read • [Edit Online](#)

This article describes the way third-party SaaS providers, also known as event publishers or partners, are onboarded to Event Grid to be able to publish events from their services and how those events are consumed by end users.

## IMPORTANT

If you are not familiar with Partner Events, see [Partner Events overview](#) for a detailed introduction of key concepts that are critical to understand and follow the steps in this article.

## Onboarding process for event publishers (partners)

In a nutshell, enabling your service's events to be consumed by users typically involves the following process:

1. **Communicate your interest** in becoming a partner to the Event Grid service team before proceeding with the next steps.
2. Create a partner topic type by creating a **registration**.
3. Create a **namespace**.
4. Create an **event channel** and **partner topic** (single step).
5. Test the Partner Events functionality end to end.

For step #4, you should decide what kind of user experience you want to provide. You have the following options:

- Provide your own solution, typically a web graphical user interface (GUI) experience, hosted under your domain using our SDK and/or REST API to create an event channel and its corresponding partner topic. With this option, you can ask the user for the subscription and resource group under which you'll create a partner topic.
- Use Azure portal or CLI to create the event channel and associated partner topic. With this option, you must have get in the user's Azure subscription some way and resource group under which you'll create a partner topic.

This article shows you how to onboard as an Azure Event Grid partner using the Azure portal.

## NOTE

Registering a partner topic type is an optional step. To help you decide if you should create a partner topic type, see [Resources managed by event publisher](#).

## Communicate your interest in becoming a partner

Fill out [this form](#) and contact the Event Grid team at [GridPartner@microsoft.com](mailto:GridPartner@microsoft.com). We'll have a conversation with you providing detailed information on Partner Events' use cases, personas, onboarding process, functionality, pricing, and more.

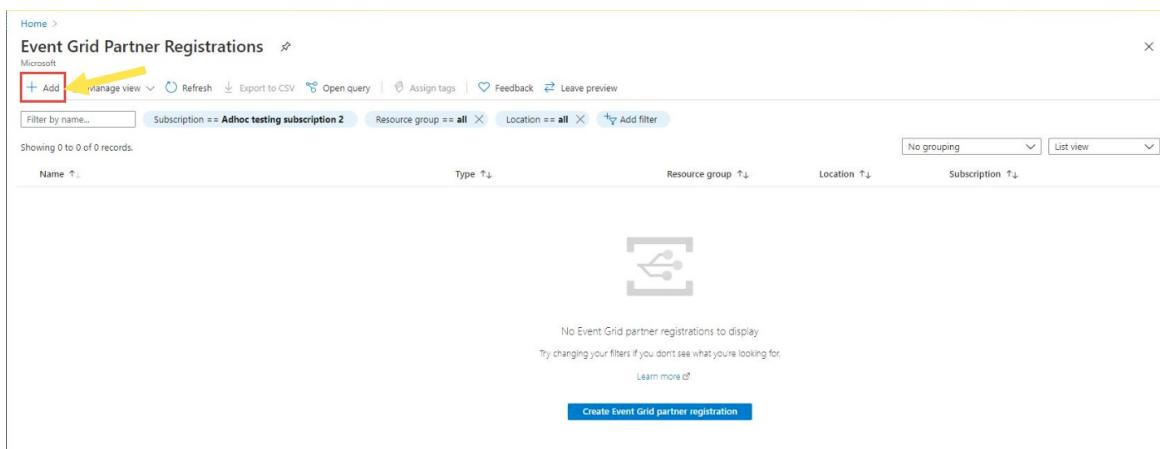
# Prerequisites

To complete the remaining steps, make sure you have:

- An Azure subscription. If you don't have one, [create a free account](#) before you begin.
- An Azure [tenant](#).

## Register a partner topic type (optional)

1. Sign in to the [Azure portal](#).
2. Select **All services** from the left navigation pane, then type in **Event Grid Partner Registrations** in the search bar, and select it.
3. On the **Event Grid Partner Registrations** page, select **+ Add** on the toolbar.



The screenshot shows the 'Event Grid Partner Registrations' blade in the Azure portal. At the top, there's a toolbar with various icons and a search bar. Below the toolbar, there are filter options for 'Subscription', 'Resource group', 'Location', and 'Add filter'. A message at the top says 'Showing 0 to 0 of 0 records.' In the center, there's a large icon of a USB port. Below the icon, a message reads 'No Event Grid partner registrations to display. Try changing your filters if you don't see what you're looking for.' At the bottom, there's a blue button labeled 'Create Event Grid partner registration'.

4. On the **Create Partner Topic Type Registrations - Basics** page, enter the following information:
  - a. In the **Project details** section, follow these steps:
    - a. Select your **Azure subscription**.
    - b. Select an existing Azure **resource group** or create a new resource group.
  - b. In the **Registration details** section, follow these steps:
    - a. For **Registration name**, enter a name for the registration.
    - b. For **Organization name**, enter the name of your organization.
  - c. In the **Partner resource type** section, enter details about your resource type that will be displayed on the **partner topic create** page:
    - a. For **Partner resource type name**, enter the name for the resource type. This will be the type of partner topic that will be created in your Azure subscription.
    - b. For **Display name**, enter a user-friendly display name for the partner topic (resource) type.
    - c. Enter a **description for the resource type**.
  - d. Enter a **description for the scenario**. It should explain the ways or scenarios in which the partner topics for your resources can be used.

## Create Partner Registration



Event Grid

Basics

Customer Service

Tags

Review + create

### Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

Azure Messaging PM Playground

Resource group \*

Create new

### Registration Details

Enter required settings for this registration.

Registration name \*

Organization name \*

### Partner Resource Type

Enter details about your resource type that will be displayed on the partner topic create page.

Partner resource type name \* ⓘ

Display name \* ⓘ

Description \*

Description of your resource type (256 char. max) to be shown on the Partner Topic gallery

Scenario description \*

Max 2048 characters. This is the description that will be shown on the create page for your partner topic. This should explain the ways or scenarios in which the partner topics can be used.

[Review + create](#)

< Previous

Next: Customer Service >

5. Select **Next: Custom Service** at the bottom of the page. On the **Customer Service** tab of the **Create Partner Registration** page, enter information that subscriber users will use to contact you in case of a problem with the event source:

- a. Enter the **Phone number**.
- b. Enter **extension** for the phone number.
- c. Enter a support web site **URL**.

## Create Partner Registration



Basics    Customer Service    Tags    Review + create

### Customer Service Details

Enter details for where customers can receive service for your product.

Phone number ⓘ

Ext. ⓘ

URL ⓘ

**Review + create**

< Previous

Next: Tags >

6. Select **Next: Tags** at the bottom of the page.

7. On the **Tags** page, configure the following values.

a. Enter a **name** and a **value** for the tag you want to add. This step is **optional**.

b. Select **Review + create** at the bottom of the page to create the registration (partner topic type).

## Create a partner namespace

1. In the Azure portal, select **All services** from the left navigational menu, then type **Event Grid Partner Namespaces** in the search bar, and then select it from the list.

2. On the **Event Grid Partner Namespaces** page, select **+ Add** on the toolbar.

The screenshot shows the 'Event Grid Partner Namespaces' page in the Azure portal. At the top, there's a toolbar with various icons like 'Manage view', 'Refresh', 'Export to CSV', 'Open query', 'Assign tags', 'Feedback', and 'Leave preview'. Below the toolbar, there are filter options: 'Filter by name...', 'Subscription == Adhoc testing subscription 2', 'Resource group == all', 'Location == all', and 'Add filter'. A message says 'Showing 0 to 0 of 0 records.' In the center, there's a large 'Create Event Grid partner namespace' button with a plus sign icon. Above the button, there's a note: 'No Event Grid partner namespaces to display. Try changing your filters if you don't see what you're looking for.' At the bottom of the page, there's a 'Learn more' link.

3. On the **Create Partner Namespace - Basics** page, specify the following information.

a. In the **Project details** section, do the following steps:

a. Select an **Azure subscription**.

b. Select an existing **resource group** or create a resource group.

b. In the **Namespace details** section, do the following steps:

a. Enter a **name** for the namespace.

b. Select a **location** for the namespace.

c. In the **Registration details** section, do the following steps to associate the namespace with a

partner registration.

- a. Select the **subscription** in which the partner registration exists.
- b. Select the **resource group** that contains the partner registration.
- c. Select the **partner registration** from the drop-down list.
- d. Select **Next: Tags** at the bottom of the page.

Home > New > Marketplace > Event Grid Partner Namespace >

## Create Partner Namespace



Event Grid

**Basics** Tags Review + create

### Project Details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

Azure Messaging PM Playground



Resource group \*

Resource group \*

Create new



### Namespace Details

Enter required settings for this namespace.

Name \*

Location \*

Central US



### Registration Details

Choose the partner registration that should be associated with this partner namespace.

Subscription \*

Azure Messaging PM Playground



Resource group \*

Resource group \*



Partner Registration



**Review + create**

< Previous

Next: Tags >

4. On the **Tags** page, add tags (optional).

- a. Enter a **name** and a **value** for the tag you want to add. This step is **optional**.
- b. Select **Review + create** at the bottom of the page.

5. On the **Review + create** page, review the details, and select **Create**.

## Create an event channel

### IMPORTANT

You'll need to request from your user an Azure subscription, resource group, location, and partner topic name to create a partner topic that your user will own and manage.

1. Go to the **Overview** page of the namespace you created.

The screenshot shows the Azure portal interface for a partner namespace named 'partnerNamespace123'. The left sidebar contains navigation links like Overview, Activity log, Access control (IAM), Tags, Settings, Monitoring, Alerts, Metrics, Diagnostic settings, Logs, Support + troubleshooting, and New support request. The main content area displays details for an event channel: Resource group (change) is 'amh', Status is 'Active', Location is 'centraluseup', Subscription (change) is '<Azure subscription name>', Subscription ID is '<Azure subscription ID>', and Tags (change) are 'key1 : value1', 'key2 : value2', and 'key3 : value3'. On the right, there's a table with columns Name, Source, Destination, Expiration Time, and Partner Topic State. A search bar at the top says 'Search to filter items...'.

partner-namespace-overview.png

2. Select **+ Event Channel** on the toolbar.

3. On the **Create Event Channel - Basics** page, specify the following information.

- a. In the **Channel details** section, do these steps:

- a. For **Event channel name**, enter a name for the event channel.
- b. Enter the **source**. See [Cloud Events 1.0 specifications](#) to get an idea of a suitable value for the source. Also, see [this Cloud Events schema example](#).

- b. In the **Destination details** section, enter details for the destination partner topic that will be created for this event channel.

- a. Enter the **ID of the subscription** in which the partner topic will be created.
- b. Enter the **name of the resource group** in which the partner topic resource will be created.
- c. Enter a **name for the partner topic**.

- c. Select **Next: Filters** at the bottom of the page.

The screenshot shows the 'Create Event Channel' page in the Azure portal. The title is 'Create Event Channel' under 'Event Grid'. Below it, there are tabs: Basics (which is selected and highlighted in blue), Filters, Additional Features, and Review + create. The 'Channel Details' section has a note: 'Enter required settings for this channel.' It contains two input fields: 'Event Channel name \*' and 'Source \*'. The 'Destination Details' section has a note: 'Enter the details for the destination partner topic that will be created for this event channel.' It contains three input fields: 'Destination subscription ID \*', 'Destination resource group \*', and 'Partner Topic name \*'. At the bottom, there are navigation buttons: 'Review + create' (in a blue box), '< Previous', and 'Next: Filters >'.

4. On the **Filters** page, add filters. do the following steps:

- Filter on attributes of each event. Only events that match all filters get delivered. Up to 25 filters can be specified. Comparisons are case-insensitive. Valid keys used for filters vary based on the event schema. In the following example, `eventid`, `source`, `eventtype`, and `eventtypeversion` can be used for keys. You can also use custom properties inside the data payload, using the `.` as the nesting operator. For example: `data`, `data.key`, `data.key1.key2`.

- Select **Next: Additional features** at the bottom of the page.

The screenshot shows the 'Create Event Channel' interface. At the top, there's a breadcrumb navigation: Home > partnerNamespace123 >. Below it is the title 'Create Event Channel' with a 'Event Grid' icon. The main content area has tabs: Basics, Filters (which is selected and highlighted in blue), Additional Features, and Review + create. Under the 'Filters' tab, there's a section titled 'Publisher Filters' with the sub-instruction: 'Filter on attributes of each event. Only events that match all filters get delivered. Up to 25 filters can be specified. All string comparisons are case-insensitive. [Learn more](#)'. It lists 'Valid keys for currently selected event schema': `eventid`, `source`, `eventtype`, `eventtypeversion`. Below this is a table with columns 'Key', 'Operator', and 'Value'. A message 'No results' is displayed. There's a blue link 'Add new filter'. At the bottom of the page, there are three buttons: 'Review + create' (in blue), '< Previous', and 'Next: Additional Features >'.

create-event-channel-filters-page.png

5. On the **Additional features** page, you can set an **expiration time** and **description** for the partner topic.

- The **expiration time** is the time at which the topic and its associated event channel will be automatically deleted if not activated by the customer. A default of seven days is used in case a time isn't provided. Select the checkbox to specify your own expiration time.
- As this topic is a resource that's not created by the user, a **description** can help the user with understanding the nature of this topic. A generic description will be provided if none is set. Select the checkbox to set your own partner topic description.
- Select **Next: Review + create**.

## Create Event Channel



Event Grid

Basics

Filters

Additional Features

Review + create

### Expiration Time

Set a time at which the topic and its associated event channel will be automatically deleted if not activated by the customer. A default of 7 days is used in case a time is not provided.

Set your own expiration time

### Description for Partner Topic

Set a description for the partner topic. As this topic is a resource that is not created by the user a description can help the user understand the nature of this topic. A generic description will be provided if none is set.

Set your own partner topic description

[Review + create](#)

< Previous

Next: Review + create >

6. On the **Review + create**, review the settings, and select **Create** to create the event channel.

## Next steps

- [Partner topics overview](#)
- [Partner topics onboarding form](#)
- [Auth0 partner topic](#)
- [How to use the Auth0 partner topic](#)

# Integrate Azure Event Grid with Auth0

11/2/2020 • 2 minutes to read • [Edit Online](#)

This article describes how to connect your Auth0 and Azure accounts by creating an Event Grid partner topic.

See the [Auth0 event type codes](#) for a full list of the events that Auth0 supports

## Send events from Auth0 to Azure Event Grid

To send Auth0 events to Azure:

1. Enable Event Grid resource provider
2. Set up an Auth0 Partner Topic in the Auth0 Dashboard
3. Activate the Partner Topic in Azure
4. Subscribe to events from Auth0

For more information about these concepts, see Event Grid [concepts](#).

### Enable Event Grid resource provider

Unless you've used Event Grid before, you'll need to register the Event Grid resource provider. If you've used Event Grid before, skip to the next section.

In the Azure portal:

1. Select Subscriptions on the left menu
2. Select the subscription you're using for Event Grid
3. On the left menu, under Settings, select Resource providers
4. Find Microsoft.EventGrid
5. Select Register
6. Refresh to make sure the status changes to Registered

### Set up an Auth0 Partner Topic

Part of the integration process is to set up Auth0 for use as an event source (this step happens in your [Auth0 Dashboard](#)).

1. Log in to the [Auth0 Dashboard](#).
2. Navigate to Logs > Streams.
3. Click + Create Stream.
4. Select Azure Event Grid and enter a unique name for your new stream.
5. Create the event source by providing your Azure Subscription ID, Azure Region, and a Resource Group name.
6. Click Save.

### Activate your Auth0 Partner Topic in Azure

Activating the Auth0 topic in Azure allows events to flow from Auth0 to Azure.

1. Log in to the Azure portal.
2. Search `Partner Topics` at the top and click `Event Grid Partner Topics` under services.
3. Click on the topic that matches the stream you created in your Auth0 Dashboard.
4. Confirm the `Source` field matches your Auth0 account.
5. Click Activate.

## Subscribe to Auth0 events

### Create an event handler

To test your Partner Topic, you'll need an event handler. Go to your Azure subscription and spin up a service that is supported as an [event handler](#) such as an [Azure Function](#).

### Subscribe to your Auth0 Partner Topic

Subscribing to your Auth0 Partner Topic allows you to tell Event Grid where you want your Auth0 events to go.

1. On the Partner Topic blade for your Auth0 integration, select + Event Subscription at the top.
2. On the Create Event Subscription page:
  - a. Enter a name for the event subscription.
  - b. Select the Azure service or Webhook you created for the Endpoint type.
  - c. Follow the instructions for the particular service.
  - d. Click Create.

## Testing

Your Auth0 Partner Topic integration with Azure should be ready to use.

### Verify the integration

To verify that the integration is working as expected:

1. Log in to the Auth0 Dashboard.
2. Navigate to Logs > Streams.
3. Click on your Event Grid stream.
4. Once on the stream, click on the Health tab. The stream should be active and as long as you don't see any errors, the stream is working.

Try [invoking any of the Auth0 actions that trigger an event to be published](#) to see events flow.

## Delivery attempts and retries

Auth0 events are delivered to Azure via a streaming mechanism. Each event is sent as it is triggered in Auth0. If Event Grid is unable to receive the event, Auth0 will retry up to three times to deliver the event. Otherwise, Auth0 will log the failure to deliver in its system.

## Next steps

- [Auth0 Partner Topic](#)
- [Partner topics overview](#)
- [Become an Event Grid partner](#)

# Get access keys for Event Grid resources (topics or domains)

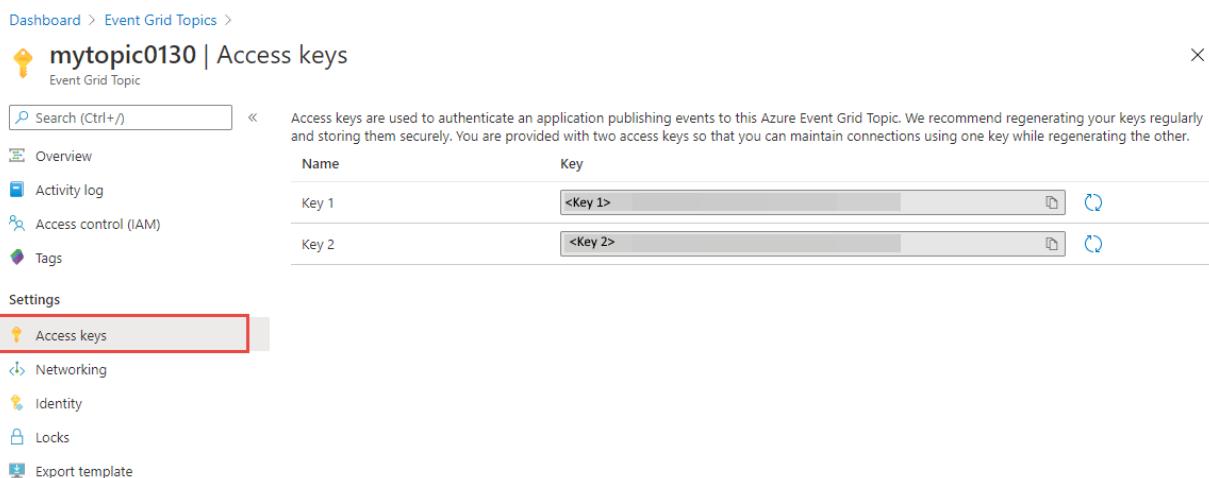
5/28/2021 • 2 minutes to read • [Edit Online](#)

Access keys are used to authenticate an application publishing events to Azure Event Grid resources (topics and domains). We recommend regenerating your keys regularly and storing them securely. You are provided with two access keys so that you can maintain connections using one key while regenerating the other.

This article describes how to get access keys for an Event Grid resource (topic or domain) using Azure portal, PowerShell, or CLI.

## Azure portal

In the Azure portal, switch to **Access keys** tab of the **Event Grid Topic** or **Event Grid Domain** page for your topic or domain.



Dashboard > Event Grid Topics >

mytopic0130 | Access keys

Event Grid Topic

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Settings

Access keys

Name Key

Key 1 <Key 1>

Key 2 <Key 2>

X

## Azure PowerShell

Use the [Get-AzEventGridTopicKey](#) command to get access keys for topics.

```
Get-AzEventGridTopicKey -ResourceGroup <RESOURCE GROUP NAME> -Name <TOPIC NAME>
```

Use [Get-AzEventGridDomainKey](#) command to get access keys for domains.

```
Get-AzEventGridDomainKey -ResourceGroup <RESOURCE GROUP NAME> -Name <DOMAIN NAME>
```

## Azure CLI

Use the [az eventgrid topic key list](#) to get access keys for topics.

```
az eventgrid topic key list --resource-group <RESOURCE GROUP NAME> --name <TOPIC NAME>
```

Use [az eventgrid domain key list](#) to get access keys for domains.

```
az eventgrid domain key list --resource-group <RESOURCE GROUP NAME> --name <DOMAIN NAME>
```

## Next steps

See the following article: [Authenticate publishing clients](#).

# Post to custom topic for Azure Event Grid

5/28/2021 • 2 minutes to read • [Edit Online](#)

This article describes how to post an event to a custom topic. It shows the format of the post and event data. The [Service Level Agreement \(SLA\)](#) only applies to posts that match the expected format.

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Endpoint

When sending the HTTP POST to a custom topic, use the URI format:

```
https://<topic-endpoint>?api-version=2018-01-01 .
```

For example, a valid URI is:

```
https://exampletopic.westus2-1.eventgrid.azure.net/api/events?api-version=2018-01-01 .
```

To get the endpoint for a custom topic with Azure CLI, use:

```
az eventgrid topic show --name <topic-name> -g <topic-resource-group> --query "endpoint"
```

To get the endpoint for a custom topic with Azure PowerShell, use:

```
(Get-AzEventGridTopic -ResourceGroupName <topic-resource-group> -Name <topic-name>).Endpoint
```

In the request, include a header value named `aeg-sas-key` that contains a key for authentication.

For example, a valid header value is `aeg-sas-key: VXbGWce53249Mt8wuotr0GPmyJ/nDT4hgdEj9DpBeRr38arnnm50Fg==`.

To get the key for a custom topic with Azure CLI, use:

```
az eventgrid topic key list --name <topic-name> -g <topic-resource-group> --query "key1"
```

To get the key for a custom topic with PowerShell, use:

```
(Get-AzEventGridTopicKey -ResourceGroupName <topic-resource-group> -Name <topic-name>).Key1
```

## Event data

For custom topics, the top-level data contains the same fields as standard resource-defined events. One of those properties is a `data` property that contains properties unique to the custom topic. As event publisher, you determine the properties for that data object. Use the following schema:

```
[
  {
    "id": string,
    "eventType": string,
    "subject": string,
    "eventTime": string-in-date-time-format,
    "data":{
      object-unique-to-each-publisher
    },
    "dataVersion": string
  }
]
```

For a description of these properties, see [Azure Event Grid event schema](#). When posting events to an event grid topic, the array can have a total size of up to 1 MB. The maximum allowed size for an event is also 1 MB. Events over 64 KB are charged in 64-KB increments.

For example, a valid event data schema is:

```
[{
  "id": "1807",
  "eventType": "recordInserted",
  "subject": "myapp/vehicles/motorcycles",
  "eventTime": "2017-08-10T21:03:07+00:00",
  "data": {
    "make": "Ducati",
    "model": "Monster"
  },
  "dataVersion": "1.0"
}]
```

## Response

After posting to the topic endpoint, you receive a response. The response is a standard HTTP response code. Some common responses are:

RESULT	RESPONSE
Success	200 OK
Event data has incorrect format	400 Bad Request
Invalid access key	401 Unauthorized
Incorrect endpoint	404 Not Found
Array or event exceeds size limits	413 Payload Too Large

For errors, the message body has the following format:

```
{  
  "error": {  
    "code": "<HTTP status code>",  
    "message": "<description>",  
    "details": [  
      {"code": "<HTTP status code>",  
       "message": "<description>"  
    ]  
  }  
}
```

## Next steps

- For information about monitoring event deliveries, see [Monitor Event Grid message delivery](#).
- For more information about the authentication key, see [Event Grid security and authentication](#).
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Receive events to an HTTP endpoint

5/25/2021 • 8 minutes to read • [Edit Online](#)

This article describes how to [validate an HTTP endpoint](#) to receive events from an Event Subscription and then receive and deserialize events. This article uses an Azure Function for demonstration purposes, however the same concepts apply regardless of where the application is hosted.

## NOTE

It is recommended that you use an [Event Grid Trigger](#) when triggering an Azure Function with Event Grid. It provides an easier and quicker integration between Event Grid and Azure Functions. However, please note that Azure Functions' Event Grid trigger does not support the scenario where the hosted code needs to control the HTTP status code returned to Event Grid. Given this limitation, your code running on an Azure Function would not be able to return a 5XX error to initiate an event delivery retry by Event Grid, for example.

## Prerequisites

You need a function app with an HTTP triggered function.

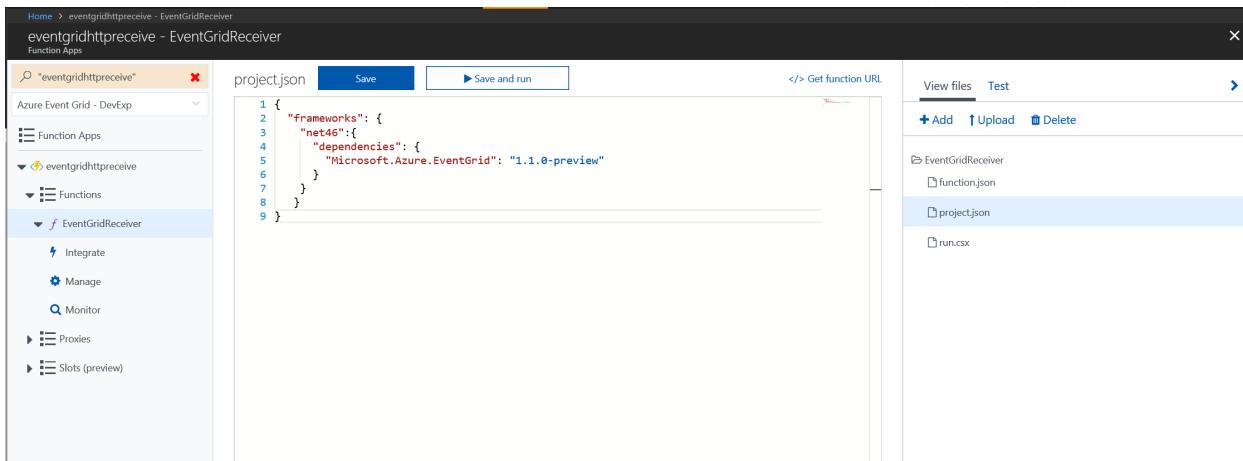
## Add dependencies

If you're developing in .NET, [add a dependency](#) to your function for the `Microsoft.Azure.EventGrid` NuGet package. The examples in this article require version 1.4.0 or later.

SDKs for other languages are available via the [Publish SDKs](#) reference. These packages have the models for native event types such as `EventGridEvent`, `StorageBlobCreatedEventData`, and `EventHubCaptureFileCreatedEventData`.

Click on the "View Files" link in your Azure Function (right most pane in the Azure functions portal), and create a file called `project.json`. Add the following contents to the `project.json` file and save it:

```
{
  "frameworks": {
    "net46": {
      "dependencies": {
        "Microsoft.Azure.EventGrid": "2.0.0"
      }
    }
  }
}
```



## Endpoint validation

The first thing you want to do is handle `Microsoft.EventGrid.SubscriptionValidationEvent` events. Every time someone subscribes to an event, Event Grid sends a validation event to the endpoint with a `validationCode` in the data payload. The endpoint is required to echo this back in the response body to [prove the endpoint is valid and owned by you](#). If you're using an [Event Grid Trigger](#) rather than a WebHook triggered Function, endpoint validation is handled for you. If you use a third-party API service (like [Zapier](#) or [IFTTT](#)), you might not be able to programmatically echo the validation code. For those services, you can manually validate the subscription by using a validation URL that is sent in the subscription validation event. Copy that URL in the `validationUrl` property and send a GET request either through a REST client or your web browser.

In C#, the `DeserializeEventGridEvents()` function deserializes the Event Grid events. It deserializes the event data into the appropriate type, such as `StorageBlobCreatedEventData`. Use the `Microsoft.Azure.EventGrid.EventType` class to get supported event types and names.

To programmatically echo the validation code, use the following code. You can find related samples at [Event Grid Consumer example](#).

```

using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Microsoft.Azure.EventGrid.Models;
using Microsoft.Azure.EventGrid;
namespace Function1
{
    public static class Function1
    {
        [FunctionName("Function1")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Anonymous, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation("C# HTTP trigger function processed a request.");
            string response = string.Empty;
            string requestContent = await new StreamReader(req.Body).ReadToEndAsync();
            log.LogInformation($"Received events: {requestContent}");

            EventGridSubscriber eventGridSubscriber = new EventGridSubscriber();

            EventGridEvent[] eventGridEvents =
            eventGridSubscriber.DeserializeEventGridEvents(requestContent);

            foreach (EventGridEvent eventGridEvent in eventGridEvents)
            {
                if (eventGridEvent.Data is SubscriptionValidationEventData)
                {
                    var eventData = (SubscriptionValidationEventData)eventGridEvent.Data;
                    log.LogInformation($"Got SubscriptionValidation event data, validation code: {eventData.ValidationCode}, topic: {eventGridEvent.Topic}");
                    // Do any additional validation (as required) and then return back the below response

                    var responseData = new SubscriptionValidationResponse()
                    {
                        ValidationResponse = eventData.ValidationCode
                    };
                    return new OkObjectResult(responseData);
                }
            }
            return new OkObjectResult(response);
        }
    }
}

```

```

module.exports = function (context, req) {
    context.log('JavaScript HTTP trigger function begun');
    var validationEventType = "Microsoft.EventGrid.SubscriptionValidationEvent";

    for (var events in req.body) {
        var body = req.body[events];
        // Deserialize the event data into the appropriate type based on event type
        if (body.data && body.eventType == validationEventType) {
            context.log("Got SubscriptionValidation event data, validation code: " +
body.data.validationCode + " topic: " + body.topic);

            // Do any additional validation (as required) and then return back the below response
            var code = body.data.validationCode;
            context.res = { status: 200, body: { "ValidationResponse": code } };
        }
    }
    context.done();
};


```

## Test validation response

Test the validation response function by pasting the sample event into the test field for the function:

```
[{
    "id": "2d1781af-3a4c-4d7c-bd0c-e34b19da4e66",
    "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "subject": "",
    "data": {
        "validationCode": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"
    },
    "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
    "eventTime": "2018-01-25T22:12:19.4556811Z",
    "metadataVersion": "1",
    "dataVersion": "1"
}]
```

When you click Run, the Output should be 200 OK and

`{"validationResponse": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"}` in the body:

[Input](#) [Output](#)

---

Provide parameters to test the HTTP request. Results can be found in the Output tab.

HTTP method [①](#)

POST



Key

master (Host key)



Query

[+ Add parameter](#)

---

Headers

[+ Add header](#)

---

Body

```
1 [ {  
2     "id": "2d1781af-3a4c-4d7c-bd0c-e34b19da4e66",  
3     "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",  
4     "subject": "",  
5     "data": {  
6         "validationCode": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"  
7     },  
8     "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",  
9     "eventTime": "2018-01-25T22:12:19.4556811Z",  
10    "metadataVersion": "1",  
11    "dataVersion": "1"  
12 } ]
```

---

[Input](#) [Output](#)

---

HTTP response code

200 OK

HTTP response content

```
{  
    "validationResponse": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"  
}
```

## Handle Blob storage events

Now, let's extend the function to handle `Microsoft.Storage.BlobCreated`:

```

using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Microsoft.Azure.EventGrid;
using Microsoft.Azure.EventGrid.Models;

namespace FunctionApp1
{
    public static class Function1
    {
        [FunctionName("Function1")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation($"C# HTTP trigger function begun");
            string response = string.Empty;

            string requestContent = await new StreamReader(req.Body).ReadToEndAsync();
            log.LogInformation($"Received events: {requestContent}");

            EventGridSubscriber eventGridSubscriber = new EventGridSubscriber();

            EventGridEvent[] eventGridEvents =
                eventGridSubscriber.DeserializeEventGridEvents(requestContent);

            foreach (EventGridEvent eventGridEvent in eventGridEvents)
            {
                if (eventGridEvent.Data is SubscriptionValidationEventData)
                {
                    var eventData = (SubscriptionValidationEventData)eventGridEvent.Data;
                    log.LogInformation($"Got SubscriptionValidation event data, validation code: {eventData.ValidationCode}, topic: {eventGridEvent.Topic}");
                    // Do any additional validation (as required) and then return back the below response

                    var responseData = new SubscriptionValidationResponse()
                    {
                        ValidationResponse = eventData.ValidationCode
                    };

                    return new OkObjectResult(responseData);
                }
            }

            return new OkObjectResult(response);
        }
    }
}

```

```

module.exports = function (context, req) {
    context.log('JavaScript HTTP trigger function begun');
    var validationEventType = "Microsoft.EventGrid.SubscriptionValidationEvent";
    var storageBlobCreatedEvent = "Microsoft.Storage.BlobCreated";

    for (var events in req.body) {
        var body = req.body[events];
        // Deserialize the event data into the appropriate type based on event type
        if (body.data && body.eventType == validationEventType) {
            context.log("Got SubscriptionValidation event data, validation code: " +
body.data.validationCode + " topic: " + body.topic);

            // Do any additional validation (as required) and then return back the below response
            var code = body.data.validationCode;
            context.res = { status: 200, body: { "ValidationResponse": code } };
        }

        else if (body.data && body.eventType == storageBlobCreatedEvent) {
            var blobCreatedEventData = body.data;
            context.log("Relaying received blob created event payload:" +
JSON.stringify(blobCreatedEventData));
        }
    }
    context.done();
};


```

## Test Blob Created event handling

Test the new functionality of the function by putting a [Blob storage event](#) into the test field and running:

```

[{
    "topic": "/subscriptions/{subscription-
id}/resourceGroups/Storage/providers/Microsoft.Storage/storageAccounts/xstoretestaccount",
    "subject": "/blobServices/default/containers/testcontainer/blobs/testfile.txt",
    "eventType": "Microsoft.Storage.BlobCreated",
    "eventTime": "2017-06-26T18:41:00.9584103Z",
    "id": "831e1650-001e-001b-66ab-eeb76e069631",
    "data": {
        "api": "PutBlockList",
        "clientRequestId": "6d79dbfb-0e37-4fc4-981f-442c9ca65760",
        "requestId": "831e1650-001e-001b-66ab-eeb76e000000",
        "eTag": "0x8D4BCC2E4835CD0",
        "contentType": "text/plain",
        "contentLength": 524288,
        "blobType": "BlockBlob",
        "url": "https://example.blob.core.windows.net/testcontainer/testfile.txt",
        "sequencer": "0000000000004420000000000028963",
        "storageDiagnostics": {
            "batchId": "b68529f3-68cd-4744-baa4-3c0498ec19f0"
        }
    },
    "dataVersion": "",
    "metadataVersion": "1"
}]

```

You should see the blob URL output in the function log:

Logs

```
2018-02-08T21:36:04.549 Function started (Id=d68d407c-9061-4c64-b57e-e8fda4234daf)
2018-02-08T21:36:04.549 Got BlobCreated event data, blob URI https://example.blob.core.windows.net/testcontainer/testfile.txt
2018-02-08T21:36:04.549 Function completed (Success, Id=d68d407c-9061-4c64-b57e-e8fda4234daf, Duration=7ms)
```

You can also test by creating a Blob storage account or General Purpose V2 (GPv2) Storage account, [adding and event subscription](#), and setting the endpoint to the function URL:

run.csx

Save

Run

</> Get function URL

```
1 using System.Net;
2 using Newtonsoft.Json;
3 using Newtonsoft.Json.Linq;
4 using Newtonsoft.Json.Serialization;
5 using Microsoft.Azure.EventGrid.Models;
6
7 class SubscriptionValidationEventData
8 {
9     public string ValidationCode { get; set; }
10 }
11
12 class SubscriptionValidationResponseData
13 {
14     public string ValidationResponse { get; set; }
15 }
16
17 class ContosoItemReceivedEventData
18 {
19     public string ItemUri { get; set; }
20     public int Count { get; set; }
21 }
22
23 public static async Task<HttpResponseMessage> Run(HttpRequestMessage req, TraceWriter
24 {
25     string response = string.Empty;
----- Subsequent code omitted for brevity -----
```

## Handle Custom events

Finally, let's extend the function once more so that it can also handle custom events.

In C#, the SDK supports mapping an event type name to the event data type. Use the

`AddOrUpdateCustomEventMapping()` function to map the custom event.

Add a check for your event `Contoso.Items.ItemReceived`. Your final code should look like:

```
using System.IO;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Azure.WebJobs;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.Logging;
using Microsoft.Azure.EventGrid.Models;
using Microsoft.Azure.EventGrid;
using Newtonsoft.Json;

namespace FunctionApp1
{
    class ContosoItemReceivedEventData
```

```

        [JsonProperty("itemSku")]
        public string ItemSku { get; set; }
    }
    public static class Function1
    {
        [FunctionName("Function1")]
        public static async Task<IActionResult> Run(
            [HttpTrigger(AuthorizationLevel.Function, "get", "post", Route = null)] HttpRequest req,
            ILogger log)
        {
            log.LogInformation($"C# HTTP trigger function begun");
            string response = string.Empty;

            string requestContent = await new StreamReader(req.Body).ReadToEndAsync();
            log.LogInformation($"Received events: {requestContent}");

            EventGridSubscriber eventGridSubscriber = new EventGridSubscriber();
            eventGridSubscriber.AddOrUpdateCustomEventMapping("Contoso.Items.ItemReceived",
typeof(ContosoItemReceivedEventData));
            EventGridEvent[] eventGridEvents =
eventGridSubscriber.DeserializeEventGridEvents(requestContent);

            foreach (EventGridEvent eventGridEvent in eventGridEvents)
            {
                if (eventGridEvent.Data is SubscriptionValidationEventData)
                {
                    var eventData = (SubscriptionValidationEventData)eventGridEvent.Data;
                    log.LogInformation($"Got SubscriptionValidation event data, validation code:
{eventData.ValidationCode}, topic: {eventGridEvent.Topic}");
                    // Do any additional validation (as required) and then return back the below response

                    var responseData = new SubscriptionValidationResponse()
                    {
                        ValidationResponse = eventData.ValidationCode
                    };

                    return new OkObjectResult(responseData);
                }
                else if (eventGridEvent.Data is StorageBlobCreatedEventData)
                {
                    var eventData = (StorageBlobCreatedEventData)eventGridEvent.Data;
                    log.LogInformation($"Got BlobCreated event data, blob URI {eventData.Url}");
                }
                else if (eventGridEvent.Data is ContosoItemReceivedEventData)
                {
                    var eventData = (ContosoItemReceivedEventData)eventGridEvent.Data;
                    log.LogInformation($"Got ContosoItemReceived event data, item SKU {eventData.ItemSku}");
                }
            }

            return new OkObjectResult(response);
        }
    }
}

```

```

module.exports = function (context, req) {
    context.log('JavaScript HTTP trigger function begun');
    var validationEventType = "Microsoft.EventGrid.SubscriptionValidationEvent";
    var storageBlobCreatedEvent = "Microsoft.Storage.BlobCreated";
    var customEventType = "Contoso.Items.ItemReceived";

    for (var events in req.body) {
        var body = req.body[events];
        // Deserialize the event data into the appropriate type based on event type
        if (body.data && body.eventType == validationEventType) {
            context.log("Got SubscriptionValidation event data, validation code: " +
body.data.validationCode + " topic: " + body.topic);

            // Do any additional validation (as required) and then return back the below response
            var code = body.data.validationCode;
            context.res = { status: 200, body: { "ValidationResponse": code } };
        }

        else if (body.data && body.eventType == storageBlobCreatedEvent) {
            var blobCreatedEventData = body.data;
            context.log("Relaying received blob created event payload:" +
JSON.stringify(blobCreatedEventData));
        }

        else if (body.data && body.eventType == customEventType) {
            var payload = body.data;
            context.log("Relaying received custom payload:" + JSON.stringify(payload));
        }
    }
    context.done();
};


```

## Test custom event handling

Finally, test that your function can now handle your custom event type:

```
[{
    "subject": "Contoso/foo/bar/items",
    "eventType": "Contoso.Items.ItemReceived",
    "eventTime": "2017-08-16T01:57:26.005121Z",
    "id": "602a88ef-0001-00e6-1233-1646070610ea",
    "data": {
        "itemSku": "Standard"
    },
    "dataVersion": "",
    "metadataVersion": "1"
}]
```

You can also test this functionality live by [sending a custom event with CURL from the Portal](#) or by [posting to a custom topic](#) using any service or application that can POST to an endpoint such as [Postman](#). Create a custom topic and an event subscription with the endpoint set as the Function URL.

## Message headers

These are the properties you receive in the message headers:

PROPERTY NAME	DESCRIPTION
aeg-subscription-name	Name of the event subscription.
aeg-delivery-count	Number of attempts made for the event.

PROPERTY NAME	DESCRIPTION
aeg-event-type	<p>Type of the event.</p> <p>It can be one of the following values:</p> <ul style="list-style-type: none"> <li>• SubscriptionValidation</li> <li>• Notification</li> <li>• SubscriptionDeletion</li> </ul>
aeg-metadata-version	<p>Metadata version of the event.</p> <p>For <b>Event Grid event schema</b>, this property represents the metadata version and for <b>cloud event schema</b>, it represents the <b>spec version</b>.</p>
aeg-data-version	<p>Data version of the event.</p> <p>For <b>Event Grid event schema</b>, this property represents the data version and for <b>cloud event schema</b>, it doesn't apply.</p>
aeg-output-event-id	ID of the Event Grid event.

## Next steps

- Explore the [Azure Event Grid Management and Publish SDKs](#)
- Learn how to [post to a custom topic](#)
- Try one of the in-depth Event Grid and Functions tutorials such as [resizing images uploaded to Blob storage](#)

# Set dead-letter location and retry policy

5/28/2021 • 4 minutes to read • [Edit Online](#)

When creating an event subscription, you can customize the settings for event delivery. This article shows you how to set up a dead letter location and customize the retry settings. For information about these features, see [Event Grid message delivery and retry](#).

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## NOTE

To learn about message delivery, retries, and dead-lettering, see the conceptual article: [Event Grid message delivery and retry](#).

## Set dead-letter location

To set a dead letter location, you need a storage account for holding events that can't be delivered to an endpoint. The examples get the resource ID of an existing storage account. They create an event subscription that uses a container in that storage account for the dead-letter endpoint.

## NOTE

- Create a storage account and a blob container in the storage before running commands in this article.
- The Event Grid service creates blobs in this container. The names of blobs will have the name of the Event Grid subscription with all the letters in upper case. For example, if the name of the subscription is My-Blob-Subscription, names of the dead letter blobs will have MY-BLOB-SUBSCRIPTION (myblobcontainer/MY-BLOB-SUBSCRIPTION/2019/8/5/11111111-1111-1111-1111-111111111111.json). This behavior is to protect against differences in case handling between Azure services.

## Azure CLI

```
containername=testcontainer

topicid=$(az eventgrid topic show --name demoTopic -g gridResourceGroup --query id --output tsv)
storageid=$(az storage account show --name demoStorage --resource-group gridResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--source-resource-id $topicid \
--name <event_subscription_name> \
--endpoint <endpoint_URL> \
--deadletter-endpoint $storageid/blobServices/default/containers/$containername
```

To turn off dead-lettering, rerun the command to create the event subscription but don't provide a value for `deadletter-endpoint`. You don't need to delete the event subscription.

**NOTE**

If you are using Azure CLI on your local machine, use Azure CLI version 2.0.56 or greater. For instructions on installing the latest version of Azure CLI, see [Install the Azure CLI](#).

## PowerShell

```
$containername = "testcontainer"

$topicid = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name demoTopic).Id
$storageid = (Get-AzStorageAccount -ResourceGroupName gridResourceGroup -Name demostorage).Id

New-AzEventGridSubscription ` 
-ResourceId $topicid ` 
-EventSubscriptionName <event_subscription_name> ` 
-Endpoint <endpoint_URL> ` 
-DeadLetterEndpoint "$storageid/blobServices/default/containers/$containername"
```

To turn off dead-lettering, rerun the command to create the event subscription but don't provide a value for `DeadLetterEndpoint`. You don't need to delete the event subscription.

**NOTE**

If you are using Azure Powershell on your local machine, use Azure PowerShell version 1.1.0 or greater. Download and install the latest Azure PowerShell from [Azure downloads](#).

## Set retry policy

When creating an Event Grid subscription, you can set values for how long Event Grid should try to deliver the event. By default, Event Grid tries for 24 hours (1440 minutes), or 30 times. You can set either of these values for your event grid subscription. The value for event time-to-live must be an integer from 1 to 1440. The value for max retries must be an integer from 1 to 30.

You can't configure the [retry schedule](#).

### Azure CLI

To set the event time-to-live to a value other than 1440 minutes, use:

```
az eventgrid event-subscription create \
-g gridResourceGroup \
--topic-name <topic_name> \
--name <event_subscription_name> \
--endpoint <endpoint_URL> \
--event-ttl 720
```

To set the max retries to a value other than 30, use:

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

#### NOTE

If you set both `event-ttl` and `max-deliver-attempts`, Event Grid uses the first to expire to determine when to stop event delivery. For example, if you set 30 minutes as time-to-live (TTL) and 10 max delivery attempts. When an event isn't delivered after 30 minutes (or) isn't delivered after 10 attempts, whichever happens first, the event is dead-lettered.

## PowerShell

To set the event time-to-live to a value other than 1440 minutes, use:

```
$topicid = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name demoTopic).Id

New-AzEventGridSubscription `

-ResourceId $topicid `

-EventSubscriptionName <event_subscription_name> `

-Endpoint <endpoint_URL> `

-EventTtl 720
```

To set the max retries to a value other than 30, use:

```
$topicid = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name demoTopic).Id

New-AzEventGridSubscription `

-ResourceId $topicid `

-EventSubscriptionName <event_subscription_name> `

-Endpoint <endpoint_URL> `

-MaxDeliveryAttempt 18
```

#### NOTE

If you set both `event-ttl` and `max-deliver-attempts`, Event Grid uses the first to expire to determine when to stop event delivery. For example, if you set 30 minutes as time-to-live (TTL) and 10 max delivery attempts. When an event isn't delivered after 30 minutes (or) isn't delivered after 10 attempts, whichever happens first, the event is dead-lettered.

## Next steps

- For a sample application that uses an Azure Function app to process dead letter events, see [Azure Event Grid Dead Letter Samples for .NET](#).
- For information about event delivery and retries, [Event Grid message delivery and retry](#).
- For an introduction to Event Grid, see [About Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Filter events for Event Grid

5/28/2021 • 5 minutes to read • [Edit Online](#)

This article shows how to filter events when creating an Event Grid subscription. To learn about the options for event filtering, see [Understand event filtering for Event Grid subscriptions](#).

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Filter by event type

When creating an Event Grid subscription, you can specify which [event types](#) to send to the endpoint. The examples in this section create event subscriptions for a resource group but limit the events that are sent to `Microsoft.Resources.ResourceWriteFailure` and `Microsoft.Resources.ResourceWriteSuccess`. If you need more flexibility when filtering events by event types, see [Filter by advanced operators and data fields](#).

For PowerShell, use the `-IncludedEventType` parameter when creating the subscription.

```
$includedEventTypes = "Microsoft.Resources.ResourceWriteFailure", "Microsoft.Resources.ResourceWriteSuccess"

New-AzEventGridSubscription `

-EventSubscriptionName demoSubToResourceGroup `

-ResourceGroupName myResourceGroup `

-Endpoint <endpoint-URL> `

-IncludedEventType $includedEventTypes
```

For Azure CLI, use the `--included-event-types` parameter. The following example uses Azure CLI in a Bash shell:

```
includedEventTypes="Microsoft.Resources.ResourceWriteFailure Microsoft.Resources.ResourceWriteSuccess"

az eventgrid event-subscription create \
--name demoSubToResourceGroup \
--resource-group myResourceGroup \
--endpoint <endpoint-URL> \
--included-event-types $includedEventTypes
```

For a Resource Manager template, use the `includedEventTypes` property.

```

"resources": [
  {
    "type": "Microsoft.EventGrid/eventSubscriptions",
    "name": "[parameters('eventSubName')]",
    "apiVersion": "2018-09-15-preview",
    "properties": {
      "destination": {
        "endpointType": "WebHook",
        "properties": {
          "endpointUrl": "[parameters('endpoint')]"
        }
      },
      "filter": {
        "subjectBeginsWith": "",
        "subjectEndsWith": "",
        "isSubjectCaseSensitive": false,
        "includedEventTypes": [
          "Microsoft.Resources.ResourceWriteFailure",
          "Microsoft.Resources.ResourceWriteSuccess"
        ]
      }
    }
  }
]

```

## Filter by subject

You can filter events by the subject in the event data. You can specify a value to match for the beginning or end of the subject. If you need more flexibility when filtering events by subject, see Filter by advanced operators and data fields.

In the following PowerShell example, you create an event subscription that filters by the beginning of the subject. You use the `-SubjectBeginsWith` parameter to limit events to ones for a specific resource. You pass the resource ID of a network security group.

```

$resourceId = (Get-AzResource -ResourceName demoSecurityGroup -ResourceGroupName myResourceGroup).ResourceId

New-AzEventGridSubscription ` 
  -Endpoint <endpoint-URL> ` 
  -EventSubscriptionName demoSubscriptionToResourceGroup ` 
  -ResourceGroupName myResourceGroup ` 
  -SubjectBeginsWith $resourceId

```

The next PowerShell example creates a subscription for a blob storage. It limits events to ones with a subject that ends in `.jpg`.

```

$storageId = (Get-AzStorageAccount -ResourceGroupName myResourceGroup -AccountName $storageName).Id

New-AzEventGridSubscription ` 
  -EventSubscriptionName demoSubToStorage ` 
  -Endpoint <endpoint-URL> ` 
  -ResourceId $storageId ` 
  -SubjectEndsWith ".jpg"

```

In the following Azure CLI example, you create an event subscription that filters by the beginning of the subject. You use the `--subject-begins-with` parameter to limit events to ones for a specific resource. You pass the resource ID of a network security group.

```
resourceId=$(az resource show --name demoSecurityGroup --resource-group myResourceGroup --resource-type Microsoft.Network/networkSecurityGroups --query id --output tsv)

az eventgrid event-subscription create \
--name demoSubscriptionToResourceGroup \
--resource-group myResourceGroup \
--endpoint <endpoint-URL> \
--subject-begins-with $resourceId
```

The next Azure CLI example creates a subscription for a blob storage. It limits events to ones with a subject that ends in `.jpg`.

```
storageid=$(az storage account show --name $storageName --resource-group myResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--resource-id $storageid \
--name demoSubToStorage \
--endpoint <endpoint-URL> \
--subject-ends-with ".jpg"
```

In the following Resource Manager template example, you create an event subscription that filters by the beginning of the subject. You use the `subjectBeginsWith` property to limit events to ones for a specific resource. You pass the resource ID of a network security group.

```
"resources": [
{
    "type": "Microsoft.EventGrid/eventSubscriptions",
    "name": "[parameters('eventSubName')]",
    "apiVersion": "2018-09-15-preview",
    "properties": {
        "destination": {
            "endpointType": "WebHook",
            "properties": {
                "endpointUrl": "[parameters('endpoint')]"
            }
        },
        "filter": {
            "subjectBeginsWith": "[resourceId('Microsoft.Network/networkSecurityGroups','demoSecurityGroup')]",
            "subjectEndsWith": "",
            "isSubjectCaseSensitive": false,
            "includedEventTypes": [ "All" ]
        }
    }
}]
```

The next Resource Manager template example creates a subscription for a blob storage. It limits events to ones with a subject that ends in `.jpg`.

```
"resources": [
  {
    "type": "Microsoft.Storage/storageAccounts/providers/eventSubscriptions",
    "name": "[concat(parameters('storageName'), '/Microsoft.EventGrid/', parameters('eventSubName'))]",
    "apiVersion": "2018-09-15-preview",
    "properties": {
      "destination": {
        "endpointType": "WebHook",
        "properties": {
          "endpointUrl": "[parameters('endpoint')]"
        }
      },
      "filter": {
        "subjectEndsWith": ".jpg",
        "subjectBeginsWith": "",
        "isSubjectCaseSensitive": false,
        "includedEventTypes": [ "All" ]
      }
    }
  }
]
```

## Filter by operators and data

For more flexibility in filtering, you can use operators and data properties to filter events.

### Subscribe with advanced filters

To learn about the operators and keys that you can use for advanced filtering, see [Advanced filtering](#).

These examples create a custom topic. They subscribe to the custom topic and filter by a value in the data object. Events that have the color property set to blue, red, or green are sent to the subscription.

For Azure CLI, use:

```
topicName=<your-topic-name>
endpointURL=<endpoint-URL>

az group create -n gridResourceGroup -l eastus2
az eventgrid topic create --name $topicName -l eastus2 -g gridResourceGroup

topicid=$(az eventgrid topic show --name $topicName -g gridResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--source-resource-id $topicid \
-n demoAdvancedSub \
--advanced-filter data.color stringin blue red green \
--endpoint $endpointURL \
--expiration-date "<yyyy-mm-dd>"
```

Notice that an [expiration date](#) is set for the subscription.

For PowerShell, use:

```

$topicName = <your-topic-name>
$endpointURL = <endpoint-URL>

New-AzResourceGroup -Name gridResourceGroup -Location eastus2
New-AzEventGridTopic -ResourceGroupName gridResourceGroup -Location eastus2 -Name $topicName

$topicid = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name $topicName).Id

$expDate = '<mm/dd/yyyy hh:mm:ss>' | Get-Date
$AdvFilter1=@{operator="StringIn"; key="Data.color"; Values=('@('blue', 'red', 'green'))}

New-AzEventGridSubscription ` 
    -ResourceId $topicid ` 
    -EventSubscriptionName <event_subscription_name> ` 
    -Endpoint $endpointURL ` 
    -ExpirationDate $expDate ` 
    -AdvancedFilter @($AdvFilter1)

```

## Test filter

To test the filter, send an event with the color field set to green. Because green is one of the values in the filter, the event is delivered to the endpoint.

For Azure CLI, use:

```

topicEndpoint=$(az eventgrid topic show --name $topicName -g gridResourceGroup --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name $topicName -g gridResourceGroup --query "key1" --output tsv)

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

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

```

For PowerShell, use:

```

$endpoint = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name $topicName).Endpoint
$keys = Get-AzEventGridTopicKey -ResourceGroupName gridResourceGroup -Name $topicName

$eventID = Get-Random 99999
$eventDate = Get-Date -Format s

$htbody = @{
    id= $eventID
    eventType="recordInserted"
    subject="myapp/vehicles/cars"
    eventTime= $eventDate
    data= @{
        model="SUV"
        color="green"
    }
    dataVersion="1.0"
}

$body = "[ "+(ConvertTo-Json $htbody)+" ]"

Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}

```

To test a scenario where the event isn't sent, send an event with the color field set to yellow. Yellow isn't one of the values specified in the subscription, so the event isn't delivered to your subscription.

For Azure CLI, use:

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

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

For PowerShell, use:

```
$htbody = @{  
    id= $eventID  
    eventType="recordInserted"  
    subject="myapp/vehicles/cars"  
    eventTime= $eventDate  
    data= @{  
        model="SUV"  
        color="yellow"  
    }  
    dataVersion="1.0"  
}  
  
$body = "[ "+(ConvertTo-Json $htbody)+""]"  
  
Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}
```

## Next steps

- For information about monitoring event deliveries, see [Monitor Event Grid message delivery](#).
- For more information about the authentication key, see [Event Grid security and authentication](#).
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Query Event Grid subscriptions

5/28/2021 • 3 minutes to read • [Edit Online](#)

This article describes how to list the Event Grid subscriptions in your Azure subscription. When querying your existing Event Grid subscriptions, it's important to understand the different types of subscriptions. You provide different parameters based on the type of subscription you want to get.

## NOTE

This article has been updated to use the Azure Az PowerShell module. The Az PowerShell module is the recommended PowerShell module for interacting with Azure. To get started with the Az PowerShell module, see [Install Azure PowerShell](#). To learn how to migrate to the Az PowerShell module, see [Migrate Azure PowerShell from AzureRM to Az](#).

## Resource groups and Azure subscriptions

Azure subscriptions and resource groups aren't Azure resources. Therefore, event grid subscriptions to resource groups or Azure subscriptions do not have the same properties as event grid subscriptions to Azure resources. Event grid subscriptions to resource groups or Azure subscriptions are considered global.

To get event grid subscriptions for an Azure subscription and its resource groups, you don't need to provide any parameters. Make sure you've selected the Azure subscription you want to query. The following examples don't get event grid subscriptions for custom topics or Azure resources.

For Azure CLI, use:

```
az account set -s "My Azure Subscription"  
az eventgrid event-subscription list
```

For PowerShell, use:

```
Set-AzContext -Subscription "My Azure Subscription"  
Get-AzEventGridSubscription
```

To get event grid subscriptions for an Azure subscription, provide the topic type of **Microsoft.Resources.Subscriptions**.

For Azure CLI, use:

```
az eventgrid event-subscription list --topic-type-name "Microsoft.Resources.Subscriptions" --location global
```

For PowerShell, use:

```
Get-AzEventGridSubscription -TopicTypeName "Microsoft.Resources.Subscriptions"
```

To get event grid subscriptions for all resource groups within an Azure subscription, provide the topic type of **Microsoft.Resources.ResourceGroups**.

For Azure CLI, use:

```
az eventgrid event-subscription list --topic-type-name "Microsoft.Resources.ResourceGroups" --location global
```

For PowerShell, use:

```
Get-AzEventGridSubscription -TopicTypeName "Microsoft.Resources.ResourceGroups"
```

To get event grid subscriptions for a specified resource group, provide the name of the resource group as a parameter.

For Azure CLI, use:

```
az eventgrid event-subscription list --resource-group myResourceGroup --location global
```

For PowerShell, use:

```
Get-AzEventGridSubscription -ResourceGroupName myResourceGroup
```

## Custom topics and Azure resources

Event grid custom topics are Azure resources. Therefore, you query event grid subscriptions for custom topics and other resources, like Blob storage account, in the same way. To get event grid subscriptions for custom topics, you must provide parameters that identify the resource or identify the location of the resource. It's not possible to broadly query event grid subscriptions for resources across your Azure subscription.

To get event grid subscriptions for custom topics and other resources in a location, provide the name of the location.

For Azure CLI, use:

```
az eventgrid event-subscription list --location westus2
```

For PowerShell, use:

```
Get-AzEventGridSubscription -Location westus2
```

To get subscriptions to custom topics for a location, provide the location and the topic type of **Microsoft.EventGrid.Topics**.

For Azure CLI, use:

```
az eventgrid event-subscription list --topic-type-name "Microsoft.EventGrid.Topics" --location "westus2"
```

For PowerShell, use:

```
Get-AzEventGridSubscription -TopicTypeName "Microsoft.EventGrid.Topics" -Location westus2
```

To get subscriptions to storage accounts for a location, provide the location and the topic type of **Microsoft.Storage.StorageAccounts**.

For Azure CLI, use:

```
az eventgrid event-subscription list --topic-type "Microsoft.Storage.StorageAccounts" --location westus2
```

For PowerShell, use:

```
Get-AzEventGridSubscription -TopicTypeName "Microsoft.Storage.StorageAccounts" -Location westus2
```

To get event grid subscriptions for a custom topic, provide the name of the custom topic and the name of its resource group.

For Azure CLI, use:

```
az eventgrid event-subscription list --topic-name myCustomTopic --resource-group myResourceGroup
```

For PowerShell, use:

```
Get-AzEventGridSubscription -TopicName myCustomTopic -ResourceGroupName myResourceGroup
```

To get event grid subscriptions for a particular resource, provide the resource ID.

For Azure CLI, use:

```
resourceid=$(az resource show -n mystorage -g myResourceGroup --resource-type "Microsoft.Storage/storageaccounts" --query id --output tsv)
az eventgrid event-subscription list --resource-id $resourceid
```

For PowerShell, use:

```
$resourceid = (Get-AzResource -Name mystorage -ResourceGroupName myResourceGroup).ResourceId
Get-AzEventGridSubscription -ResourceId $resourceid
```

## Next steps

- For information about event delivery and retries, [Event Grid message delivery and retry](#).
- For an introduction to Event Grid, see [About Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Subscribe to events through portal

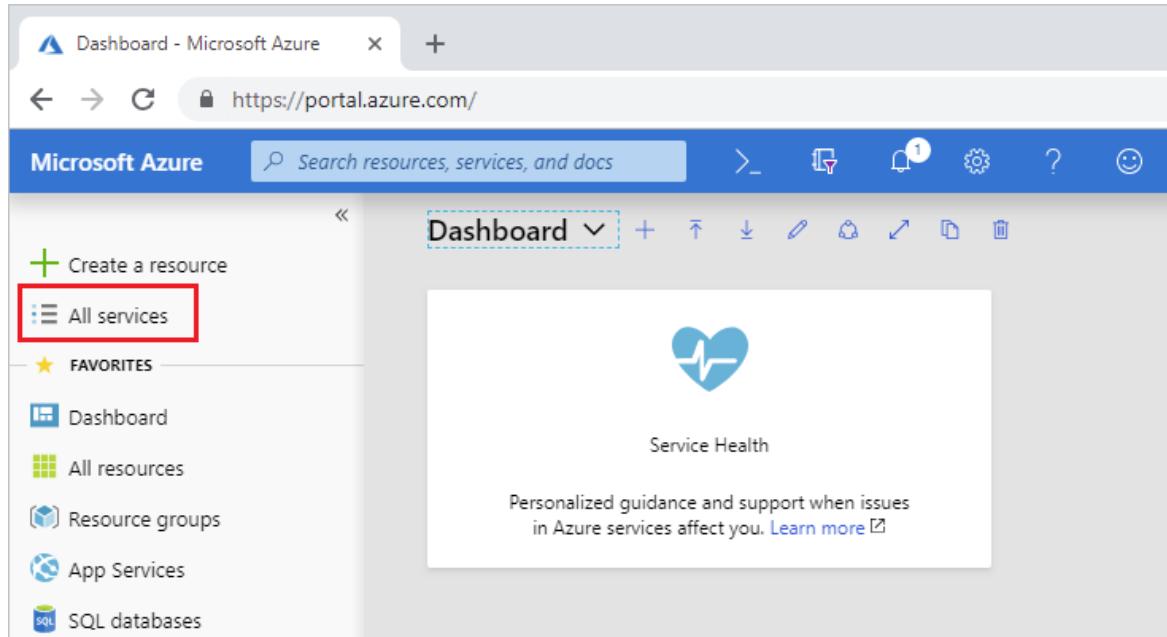
11/2/2020 • 2 minutes to read • [Edit Online](#)

This article describes how to create Event Grid subscriptions through the portal.

## Create event subscriptions

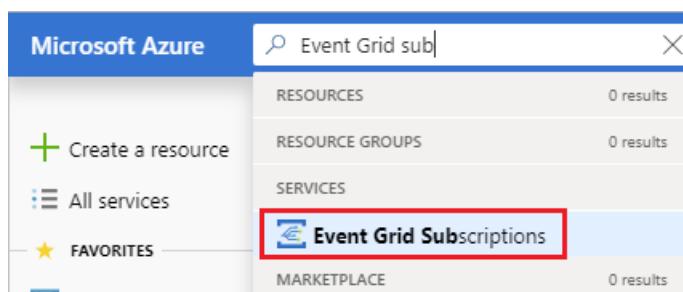
To create an Event Grid subscription for any of the supported [event sources](#), use the following steps. This article shows how to create an Event Grid subscription for an Azure subscription.

1. Select All services.



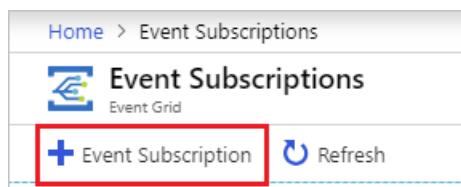
The screenshot shows the Microsoft Azure portal dashboard. On the left, there's a sidebar with various service links: 'Create a resource', 'All services' (which is highlighted with a red box), 'FAVORITES', 'Dashboard', 'All resources', 'Resource groups', 'App Services', and 'SQL databases'. The main area is titled 'Dashboard' with a 'Service Health' section featuring a heart icon and a message about personalized guidance for Azure issues. A search bar at the top says 'Search resources, services, and docs'.

2. Search for **Event Grid Subscriptions** and select it from the available options.



The screenshot shows a search results page in the Microsoft Azure portal. The search term 'Event Grid sub' is entered in the search bar. The results are categorized into 'RESOURCES', 'RESOURCE GROUPS', 'SERVICES', and 'MARKETPLACE'. Under the 'SERVICES' category, the 'Event Grid Subscriptions' option is highlighted with a red box. It has a small icon of a gear and a plus sign.

3. Select **+** Event Subscription.



The screenshot shows the 'Event Subscriptions' page. At the top, there's a breadcrumb trail: 'Home > Event Subscriptions'. Below that is a section titled 'Event Subscriptions' with a sub-section 'Event Grid'. At the bottom of the page, there's a large red button with a plus sign and the text 'Event Subscription'. To the right of this button is a 'Refresh' button with a circular arrow icon.

4. Select the type of subscription you want to create. For example, to subscribe to events for your Azure subscription, select **Azure Subscriptions** and the target subscription.

Home > Event Subscriptions > Create Event Subscription

## Create Event Subscription

Event Grid

Basic Additional Features Advanced Editor

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

**TOPIC DETAILS**

Pick a topic resource for which events should be generated and pushed. [Learn more](#)

Topic Types: Azure Subscriptions

Subscription:

5. To subscribe to all event types for this event source, keep the **Subscribe to all event types** option checked. Otherwise, select the event types for this subscription.

**EVENT TYPES**

Pick which event types get pushed to your destination. [Learn more](#)

**Subscribe to all event types**

Defined Event Types: 9 selected

- Resource Write Success
- Resource Write Failure
- Resource Write Cancel
- Resource Delete Success
- Resource Delete Failure
- Resource Delete Cancel
- Resource Action Success
- Resource Action Failure
- Resource Action Cancel

**ENDPOINT DETAILS**

Pick an event handler to receive your events

Endpoint Type: Web Hook ([change](#))

Endpoint: <https://contoso.com/api> ([change](#))

**EVENT SUBSCRIPTION DETAILS**

Name: demoSub

Event Schema: Event Grid Schema

6. Provide additional details about the event subscription, such as the endpoint for handling events and a subscription name.

**ENDPOINT DETAILS**

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type: Web Hook ([change](#))

Endpoint: <https://contoso.com/api> ([change](#))

**EVENT SUBSCRIPTION DETAILS**

Name: demoSub

Event Schema: Event Grid Schema

7. To enable dead lettering and customize retry policies, select **Additional Features**.

## Create Event Subscription

Event Grid

Basic Additional Features

8. Select a container to use for storing events that aren't delivered, and set how retries are sent.

**FILTERS AND LABELS**

Apply filters to the subject of each event. Only events with matching subjects get delivered. [Learn more](#)

Subject Begins With

Subject Ends With

Labels

**DEAD-LETTERING**

Save events that cannot be delivered to storage. [Learn more](#)

Enable dead-lettering

Storage Subscription

Storage Blob Container

**RETRY POLICIES**

Customize how many times and for how long event delivery will be retried. [Learn more](#)

Configure Retry Policies

Events will be dead-lettered or discarded after either the number of retry attempts is exceeded or the retry interval has elapsed.

Number of Retry Attempts <input type="text" value="10"/>				
Retry Interval	Days <input type="text" value="0"/>	Hours <input type="text" value="2"/>	Minutes <input type="text" value="0"/>	Seconds <input type="text" value="0"/>

9. When done, select **Create**.

## Create subscription on resource

Some event sources support creating an event subscription through the portal interface for that resource. Select the event source, and look for **Events** in left pane.

contosostorage1031  
Storage account

Search (Ctrl+ /)

- Overview
- Activity log
- Access control (IAM)
- Tags
- Diagnose and solve problems
- Events
- Storage Explorer (preview)

The portal presents you with options for creating an event subscription that is relevant to that source.

## Next steps

- For information about event delivery and retries, [Event Grid message delivery and retry](#).

- For an introduction to Event Grid, see [About Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Map custom fields to Event Grid schema

5/28/2021 • 4 minutes to read • [Edit Online](#)

If your event data doesn't match the expected [Event Grid schema](#), you can still use Event Grid to route event to subscribers. This article describes how to map your schema to the Event Grid schema.

## Original event schema

Let's suppose you have an application that sends events in the following format:

```
[  
 {  
   "myEventTypeField": "Created",  
   "resource": "Users/example/Messages/1000",  
   "resourceData": {"someDataField1": "SomeDataFieldValue"}  
 }  
]
```

Although that format doesn't match the required schema, Event Grid enables you to map your fields to the schema. Or, you can receive the values in the original schema.

## Create custom topic with mapped fields

When creating a custom topic, specify how to map fields from your original event to the event grid schema. There are three values you use to customize the mapping:

- The **input schema** value specifies the type of schema. The available options are CloudEvents schema, custom event schema, or Event Grid schema. The default value is Event Grid schema. When creating custom mapping between your schema and the event grid schema, use custom event schema. When events are in the CloudEvents format, use CloudEvents schema.
- The **mapping default values** property specifies default values for fields in the Event Grid schema. You can set default values for `subject`, `eventtype`, and `dataversion`. Typically, you use this parameter when your custom schema doesn't include a field that corresponds to one of those three fields. For example, you can specify that data version is always set to **1.0**.
- The **mapping fields** value maps fields from your schema to the event grid schema. You specify values in space-separated key/value pairs. For the key name, use the name of the event grid field. For the value, use the name of your field. You can use key names for `id`, `topic`, `eventtime`, `subject`, `eventtype`, and `dataversion`.

To create a custom topic with Azure CLI, use:

```
az eventgrid topic create \  
 -n demotopic \  
 -l eastus2 \  
 -g myResourceGroup \  
 --input-schema customeventschema \  
 --input-mapping-fields eventType=myEventTypeField \  
 --input-mapping-default-values subject=DefaultSubject dataVersion=1.0
```

For PowerShell, use:

```
New-AzEventGridTopic ` 
-ResourceGroupName myResourceGroup ` 
-Name demotopic ` 
-Location eastus2 ` 
-InputSchema CustomEventSchema ` 
-InputMappingField @{eventType="myEventTypeField"} ` 
-InputMappingDefaultValue @{subject="DefaultSubject"; dataVersion="1.0" }
```

## Subscribe to event grid topic

When subscribing to the custom topic, you specify the schema you would like to use for receiving the events. You specify the CloudEvents schema, custom event schema, or Event Grid schema. The default value is Event Grid schema.

The following example subscribes to an event grid topic and uses the Event Grid schema. For Azure CLI, use:

```
topicid=$(az eventgrid topic show --name demoTopic -g myResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--source-resource-id $topicid \
--name eventsub1 \
--event-delivery-schema eventgridschema \
--endpoint <endpoint_URL>
```

The next example uses the input schema of the event:

```
az eventgrid event-subscription create \
--source-resource-id $topicid \
--name eventsub2 \
--event-delivery-schema custominputschema \
--endpoint <endpoint_URL>
```

The following example subscribes to an event grid topic and uses the Event Grid schema. For PowerShell, use:

```
$topicid = (Get-AzEventGridTopic -ResourceGroupName myResourceGroup -Name demoTopic).Id

New-AzEventGridSubscription ` 
-ResourceId $topicid ` 
-EventSubscriptionName eventsub1 ` 
-EndpointType webhook ` 
-Endpoint <endpoint-url> ` 
-DeliverySchema EventGridSchema
```

The next example uses the input schema of the event:

```
New-AzEventGridSubscription ` 
-ResourceId $topicid ` 
-EventSubscriptionName eventsub2 ` 
-EndpointType webhook ` 
-Endpoint <endpoint-url> ` 
-DeliverySchema CustomInputSchema
```

## Publish event to topic

You're now ready to send an event to the custom topic, and see the result of the mapping. The following script to post an event in the [example schema](#):

For Azure CLI, use:

```
endpoint=$(az eventgrid topic show --name demotopic -g myResourceGroup --query "endpoint" --output tsv)
key=$(az eventgrid topic key list --name demotopic -g myResourceGroup --query "key1" --output tsv)

event='[ { "myEventTypeField": "Created", "resource": "Users/example/Messages/1000", "resourceData": {"someDataField1": "SomeDataFieldValue"} } ]'

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

For PowerShell, use:

```
$endpoint = (Get-AzEventGridTopic -ResourceGroupName myResourceGroup -Name demotopic).Endpoint
$keys = Get-AzEventGridTopicKey -ResourceGroupName myResourceGroup -Name demotopic

$hhtbody = @{
    myEventTypeField="Created"
    resource="Users/example/Messages/1000"
    resourceData= @{
        someDataField1="SomeDataFieldValue"
    }
}

$body = "[ "+(ConvertTo-Json $htbody)+" ]"
Invoke-WebRequest -Uri $endpoint -Method POST -Body $body -Headers @{"aeg-sas-key" = $keys.Key1}
```

Now, look at your WebHook endpoint. The two subscriptions delivered events in different schemas.

The first subscription used event grid schema. The format of the delivered event is:

```
{
  "id": "aa5b8e2a-1235-4032-be8f-5223395b9eae",
  "eventTime": "2018-11-07T23:59:14.7997564Z",
  "eventType": "Created",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "topic": "/subscriptions/<subscription-
id>/resourceGroups/myResourceGroup/providers/Microsoft.EventGrid/topics/demotopic",
  "subject": "DefaultSubject",
  "data": {
    "myEventTypeField": "Created",
    "resource": "Users/example/Messages/1000",
    "resourceData": {
      "someDataField1": "SomeDataFieldValue"
    }
  }
}
```

These fields contain the mappings from the custom topic. **myEventTypeField** is mapped to **EventType**. The default values for **DataVersion** and **Subject** are used. The **Data** object contains the original event schema fields.

The second subscription used the input event schema. The format of the delivered event is:

```
{  
    "myEventTypeField": "Created",  
    "resource": "Users/example/Messages/1000",  
    "resourceData": {  
        "someDataField1": "SomeFieldValue"  
    }  
}
```

Notice that the original fields were delivered.

## Next steps

- For information about event delivery and retries, [Event Grid message delivery and retry](#).
- For an introduction to Event Grid, see [About Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Use CloudEvents v1.0 schema with Event Grid

5/28/2021 • 4 minutes to read • [Edit Online](#)

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 and handling events, and universal ways of deserializing the outer event schema. With a common schema, you can more easily integrate work across platforms.

CloudEvents is being built by several [collaborators](#), including Microsoft, through the [Cloud Native Computing Foundation](#). It's currently available as version 1.0.

This article describes how to use the CloudEvents schema with Event Grid.

## CloudEvent schema

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

```
{  
    "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": "0000000000000000000000000000000099240000000000c41c18",  
        "storageDiagnostics": {  
            "batchId": "681fe319-3006-00a8-0022-9e7cde000000"  
        }  
    }  
}
```

For a detailed description of the available fields, their types, and definitions, see [CloudEvents v1.0](#).

The headers values for events delivered in the CloudEvents schema and the Event Grid schema are the same except for `content-type`. For the CloudEvents schema, that header value is

```
"content-type": "application/cloudevents+json; charset=utf-8"
```

For the Event Grid schema, that header value is

```
"content-type": "application/json; charset=utf-8"
```

## Configure Event Grid for CloudEvents

You can use Event Grid for both input and output of events in the CloudEvents schema. The following table describes the possible transformations:

EVENT GRID RESOURCE	INPUT SCHEMA	DELIVERY SCHEMA
System topics	Event Grid schema	Event Grid schema or CloudEvents schema
Custom topics/domains	Event Grid schema	Event Grid schema or CloudEvents schema
Custom topics/domains	CloudEvents schema	CloudEvents schema
Custom topics/domains	Custom schema	Custom schema, Event Grid schema, or CloudEvents schema
Partner topics	CloudEvents schema	CloudEvents schema

For all event schemas, Event Grid requires validation when you're publishing to an Event Grid topic and when you're creating an event subscription.

For more information, see [Event Grid security and authentication](#).

## Input schema

You set the input schema for a custom topic when you create the custom topic.

For the Azure CLI, use:

```
az eventgrid topic create \
--name <topic_name> \
-l westcentralus \
-g gridResourceGroup \
--input-schema cloudeventschemasv1_0
```

For PowerShell, use:

```
New-AzEventGridTopic ` 
-ResourceGroupName gridResourceGroup ` 
-Location westcentralus ` 
-Name <topic_name> ` 
-InputSchema CloudEventSchemaV1_0
```

## Output schema

You set the output schema when you create the event subscription.

For the Azure CLI, use:

```
topicID=$(az eventgrid topic show --name <topic-name> -g gridResourceGroup --query id --output tsv)

az eventgrid event-subscription create \
--name <event_subscription_name> \
--source-resource-id $topicID \
--endpoint <endpoint_URL> \
--event-delivery-schema cloudeventschemasv1_0
```

For PowerShell, use:

```
$topicid = (Get-AzEventGridTopic -ResourceGroupName gridResourceGroup -Name <topic-name>).Id

New-AzEventGridSubscription `

-ResourceId $topicid `

-EventSubscriptionName <event_subscription_name> `

-Endpoint <endpoint_URL> `

-DeliverySchema CloudEventSchemaV1_0
```

Currently, you can't use an Event Grid trigger for an Azure Functions app when the event is delivered in the CloudEvents schema. Use an HTTP trigger. For examples of implementing an HTTP trigger that receives events in the CloudEvents schema, see [Using CloudEvents with Azure Functions](#).

## Endpoint validation with CloudEvents v1.0

If you're already familiar with Event Grid, you might be aware of the endpoint validation handshake for preventing abuse. CloudEvents v1.0 implements its own [abuse protection semantics](#) by using the HTTP OPTIONS method. To read more about it, see [HTTP 1.1 Web Hooks for event delivery - Version 1.0](#). When you use the CloudEvents schema for output, Event Grid uses the CloudEvents v1.0 abuse protection in place of the Event Grid validation event mechanism.

## Use with Azure Functions

The [Azure Functions Event Grid binding](#) doesn't natively support CloudEvents, so HTTP-triggered functions are used to read CloudEvents messages. When you use an HTTP trigger to read CloudEvents, you have to write code for what the Event Grid trigger does automatically:

- Sends a validation response to a [subscription validation request](#)
- Invokes the function once per element of the event array contained in the request body

For information about the URL to use for invoking the function locally or when it runs in Azure, see the [HTTP trigger binding reference documentation](#).

The following sample C# code for an HTTP trigger simulates Event Grid trigger behavior. Use this example for events delivered in the CloudEvents schema.

```

[FunctionName("HttpTrigger")]
public static async Task<HttpResponseMessage> Run([HttpTrigger(AuthorizationLevel.Anonymous, "post",
"options", Route = null)]HttpRequestMessage req, ILogger log)
{
    log.LogInformation("C# HTTP trigger function processed a request.");
    if (req.Method == HttpMethod.Options)
    {
        // If the request is for subscription validation, send back the validation code

        var response = req.CreateResponse(HttpStatusCode.OK);
        response.Headers.Add("Webhook-Allowed-Origin", "eventgrid.azure.net");

        return response;
    }

    var requestmessage = await req.Content.ReadAsStringAsync();
    var message = JToken.Parse(requestmessage);

    // The request isn't for subscription validation, so it's for an event.
    // CloudEvents schema delivers one event at a time.
    log.LogInformation($"Source: {message["source"]}");
    log.LogInformation($"Time: {message["eventTime"]}");
    log.LogInformation($"Event data: {message["data"].ToString()}");

    return req.CreateResponse(HttpStatusCode.OK);
}

```

The following sample JavaScript code for an HTTP trigger simulates Event Grid trigger behavior. Use this example for events delivered in the CloudEvents schema.

```

module.exports = function (context, req) {
    context.log('JavaScript HTTP trigger function processed a request.');

    if (req.method == "OPTIONS") {
        // If the request is for subscription validation, send back the validation code

        context.log('Validate request received');
        context.res = {
            status: 200,
            headers: {
                'Webhook-Allowed-Origin': 'eventgrid.azure.net',
            },
        };
    }
    else
    {
        var message = req.body;

        // The request isn't for subscription validation, so it's for an event.
        // CloudEvents schema delivers one event at a time.
        var event = JSON.parse(message);
        context.log('Source: ' + event.source);
        context.log('Time: ' + event.eventTime);
        context.log('Data: ' + JSON.stringify(event.data));
    }

    context.done();
};

```

## Next steps

- For information about monitoring event deliveries, see [Monitor Event Grid message delivery](#).
- We encourage you to test, comment on, and [contribute to CloudEvents](#).

- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Manage topics and publish events using event domains

5/28/2021 • 4 minutes to read • [Edit Online](#)

This article shows how to:

- Create an Event Grid domain
- Subscribe to event grid topics
- List keys
- Publish events to a domain

To learn about event domains, see [Understand event domains for managing Event Grid topics](#).

## Create an Event Domain

To manage large sets of topics, create an event domain.

- [Azure CLI](#)
- [PowerShell](#)

```
az eventgrid domain create \
-g <my-resource-group> \
--name <my-domain-name> \
-l <location>
```

Successful creation returns the following values:

```
{
  "endpoint": "https://<my-domain-name>.westus2-1.eventgrid.azure.net/api/events",
  "id": "/subscriptions/<sub-id>/resourceGroups/<my-resource-
group>/providers/Microsoft.EventGrid/domains/<my-domain-name>",
  "inputSchema": "EventGridSchema",
  "inputSchemaMapping": null,
  "location": "westus2",
  "name": "<my-domain-name>",
  "provisioningState": "Succeeded",
  "resourceGroup": "<my-resource-group>",
  "tags": null,
  "type": "Microsoft.EventGrid/domains"
}
```

Note the `endpoint` and `id` as they're required to manage the domain and publish events.

## Manage access to topics

Managing access to topics is done via [role assignment](#). Role assignment uses Azure role-based access control to limit operations on Azure resources to authorized users at a certain scope.

Event Grid has two built-in roles, which you can use to assign particular users access on various topics within a domain. These roles are `EventGrid EventSubscription Contributor (Preview)`, which allows for creation and deletion of subscriptions, and `EventGrid EventSubscription Reader (Preview)`, which only allows for listing of

event subscriptions.

- [Azure CLI](#)
- [PowerShell](#)

The following Azure CLI command limits `alice@contoso.com` to creating and deleting event subscriptions only on topic `demotopic1`:

```
az role assignment create \
--assignee alice@contoso.com \
--role "EventGrid EventSubscription Contributor (Preview)" \
--scope /subscriptions/<sub-id>/resourceGroups/<my-resource-group>/providers/Microsoft.EventGrid/domains/<my-domain-name>/topics/demotopic1
```

For more information about managing access for Event Grid operations, see [Event Grid security and authentication](#).

## Create topics and subscriptions

The Event Grid service automatically creates and manages the corresponding topic in a domain based on the call to create an event subscription for a domain topic. There's no separate step to create a topic in a domain. Similarly, when the last event subscription for a topic is deleted, the topic is deleted as well.

Subscribing to a topic in a domain is the same as subscribing to any other Azure resource. For the source resource ID, specify the event domain ID returned when creating the domain earlier. To specify the topic you want to subscribe to, add `/topics/<my-topic>` to the end of the source resource ID. To create a domain scope event subscription that receives all events in the domain, specify the event domain ID without specifying any topics.

Typically, the user you granted access to in the preceding section would create the subscription. To simplify this article, you create the subscription.

- [Azure CLI](#)
- [PowerShell](#)

```
az eventgrid event-subscription create \
--name <event-subscription> \
--source-resource-id "/subscriptions/<sub-id>/resourceGroups/<my-resource-group>/providers/Microsoft.EventGrid/domains/<my-domain-name>/topics/demotopic1" \
--endpoint https://contoso.azurewebsites.net/api/updates
```

If you need a test endpoint to subscribe your events to, you can always deploy a [pre-built web app](#) that displays the incoming events. You can send your events to your test website at

`https://<your-site-name>.azurewebsites.net/api/updates`.



Permissions that are set for a topic are stored in Azure Active Directory and must be deleted explicitly. Deleting an event subscription won't revoke a user's access to create event subscriptions if they've written access on a topic.

## Publish events to an Event Grid Domain

Publishing events to a domain is the same as [publishing to a custom topic](#). However, instead of publishing to the custom topic, you publish all events to the domain endpoint. In the JSON event data, you specify the topic you

wish the events to go to. The following array of events would result in event with `"id": "1111"` to topic `demotopic1` while event with `"id": "2222"` would be sent to topic `demotopic2`:

```
[{
  "topic": "demotopic1",
  "id": "1111",
  "eventType": "maintenanceRequested",
  "subject": "myapp/vehicles/diggers",
  "eventTime": "2018-10-30T21:03:07+00:00",
  "data": {
    "make": "Contoso",
    "model": "Small Digger"
  },
  "dataVersion": "1.0"
},
{
  "topic": "demotopic2",
  "id": "2222",
  "eventType": "maintenanceCompleted",
  "subject": "myapp/vehicles/tractors",
  "eventTime": "2018-10-30T21:04:12+00:00",
  "data": {
    "make": "Contoso",
    "model": "Big Tractor"
  },
  "dataVersion": "1.0"
}]
```

- [Azure CLI](#)
- [PowerShell](#)

To get the domain endpoint with Azure CLI, use

```
az eventgrid domain show \
-g <my-resource-group> \
-n <my-domain>
```

To get the keys for a domain, use:

```
az eventgrid domain key list \
-g <my-resource-group> \
-n <my-domain>
```

And then use your favorite method of making an HTTP POST to publish your events to your Event Grid domain.

## Search lists of topics or subscriptions

To search and manage large number of topics or subscriptions, Event Grid's APIs support listing and pagination.

### Using CLI

For example, the following command lists all the topics with name containing `mytopic`.

```
az eventgrid topic list --odata-query "contains(name, 'mytopic')"
```

For more information about this command, see [az eventgrid topic list](#).

## Next steps

- For more information on high-level concepts in Event domains and why they're useful, see the [conceptual overview of Event Domains](#).

# Build your own disaster recovery for custom topics in Event Grid

4/22/2021 • 7 minutes to read • [Edit Online](#)

Disaster recovery focuses on recovering from a severe loss of application functionality. This tutorial will walk you through how to set up your eventing architecture to recover if the Event Grid service becomes unhealthy in a particular region.

In this tutorial, you'll learn how to create an active-passive failover architecture for custom topics in Event Grid. You'll accomplish failover by mirroring your topics and subscriptions across two regions and then managing a failover when a topic becomes unhealthy. The architecture in this tutorial fails over all new traffic. It's important to be aware, with this setup, events already in flight won't be recovered until the compromised region is healthy again.

## NOTE

Event Grid supports automatic geo disaster recovery (GeoDR) on the server side now. You can still implement client-side disaster recovery logic if you want a greater control on the failover process. For details about automatic GeoDR, see [Server-side geo disaster recovery in Azure Event Grid](#).

## Create a message endpoint

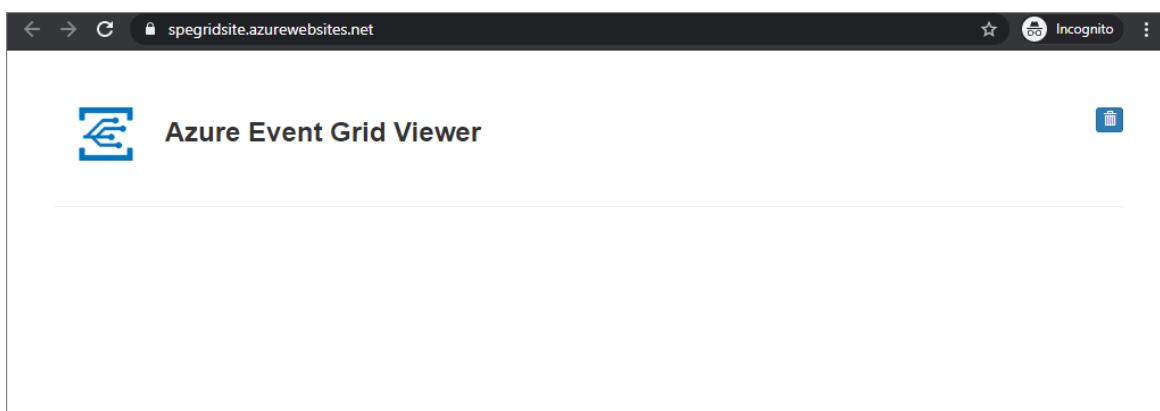
To test your failover configuration, you'll need an endpoint to receive your events at. The endpoint isn't part of your failover infrastructure, but will act as our event handler to make it easier to test.

To simplify testing, deploy 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.

1. Select **Deploy to Azure** to deploy the solution to your subscription. In the Azure portal, provide values for the parameters.



2. The deployment may take a few minutes to complete. After the deployment has succeeded, view your web app to make sure it's running. In a web browser, navigate to:  
`https://<your-site-name>.azurewebsites.net` Make sure to note this URL as you'll need it later.
3. You see the site but no events have been posted to it yet.



# Enable Event Grid resource provider

If you haven't previously used Event Grid in your Azure subscription, you may need to register the Event Grid resource provider.

In the Azure portal:

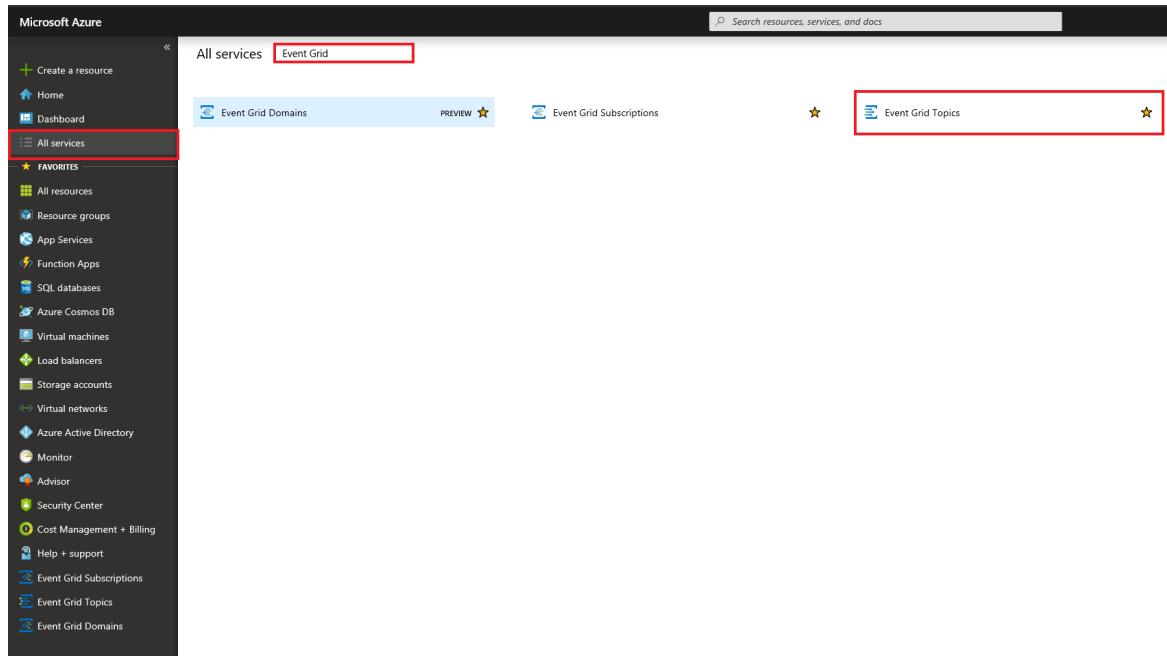
1. Select **Subscriptions** on the left menu.
2. Select the subscription you're using for Event Grid.
3. On the left menu, under **Settings**, select **Resource providers**.
4. Find **Microsoft.EventGrid**.
5. If not registered, select **Register**.

It may take a moment for the registration to finish. Select **Refresh** to update the status. When **Status** is **Registered**, you're ready to continue.

## Create your primary and secondary topics

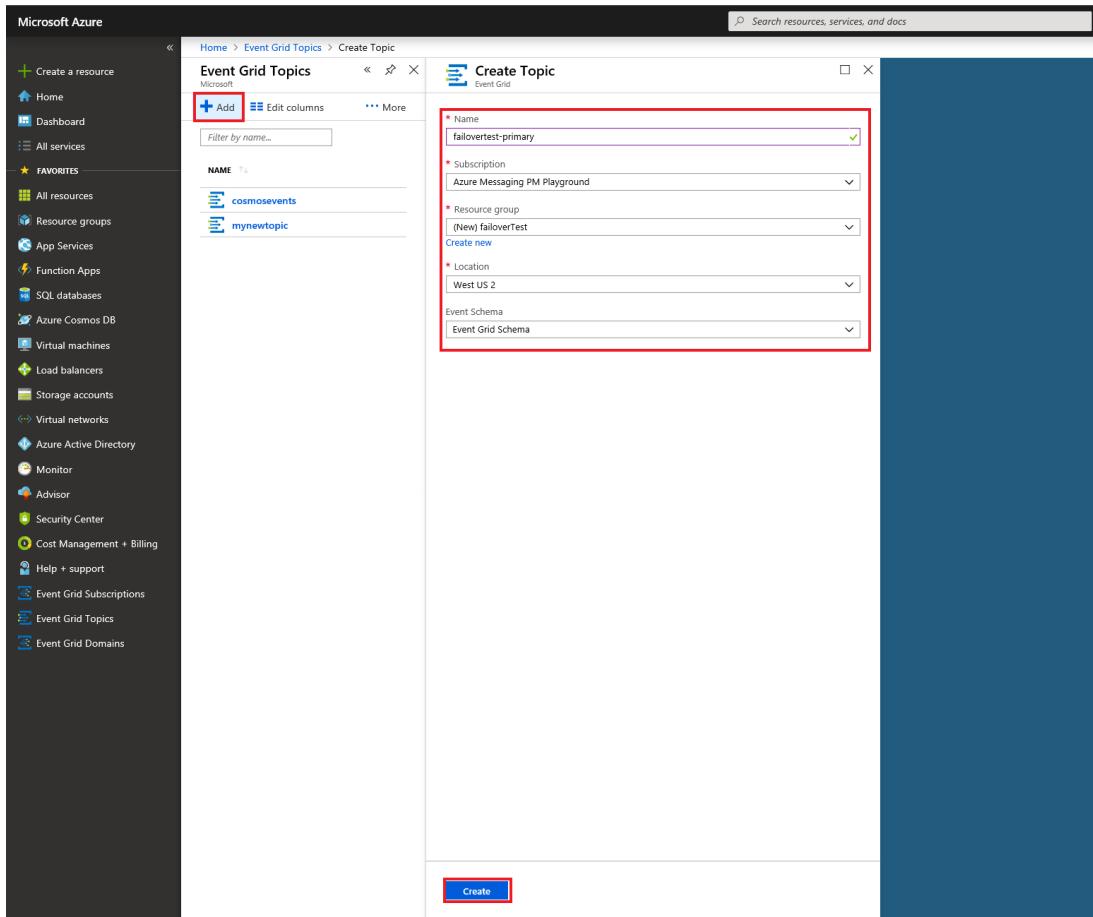
First, create two Event Grid topics. These topics will act as your primary and secondary. By default, your events will flow through your primary topic. If there is a service outage in the primary region, your secondary will take over.

1. Sign in to the [Azure portal](#).
2. From the upper left corner of the main Azure menu, choose **All services** > search for **Event Grid** > select **Event Grid Topics**.



Select the star next to Event Grid Topics to add it to resource menu for easier access in the future.

3. In the Event Grid Topics Menu, select **+ADD** to create your primary topic.
  - Give the topic a logical name and add "-primary" as a suffix to make it easy to track.
  - This topic's region will be your primary region.



- Once the Topic has been created, navigate to it and copy the **Topic Endpoint**. you'll need the URI later.

- Get the access key for the topic, which you'll also need later. Click on **Access keys** in the resource menu and copy Key 1.

- In the Topic blade, click **+Event Subscription** to create a subscription connecting your subscribing the event receiver website you made in the pre-requisites to the tutorial.

- Give the event subscription a logical name and add "-primary" as a suffix to make it easy to track.
- Select Endpoint Type Web Hook.
- Set the endpoint to your event receiver's event URL, which should look something like:

`https://<your-event-reciever>.azurewebsites.net/api/updates`

The screenshot shows the Azure portal interface for creating an Event Subscription. On the left is the navigation sidebar with various service icons. The main area shows the 'Create Event Subscription' page under 'Event Grid'. The 'Basic' tab is selected. In the 'EVENT SUBSCRIPTION DETAILS' section, the 'Name' field contains 'mytestsubscription-primary' with a green checkmark. Under 'TOPIC DETAILS', 'Topic Type' is 'Event Grid Topic' and 'Topic Resource' is 'failovertest-primary'. In the 'EVENT TYPES' section, there is a checked checkbox for 'Subscribe to all event types'. The 'ENDPOINT DETAILS' section shows 'Endpoint Type' as 'Web Hook' and the 'Endpoint' URL as 'https://myeventreciever.azurewebsites.net/api/updates'. Both the 'Endpoint Type' and 'Endpoint' fields are highlighted with red boxes. At the bottom is a blue 'Create' button.

7. Repeat the same flow to create your secondary topic and subscription. This time, replace the "-primary" suffix with "-secondary" for easier tracking. Finally, make sure you put it in a different Azure Region. While you can put it anywhere you want, it's recommended that you use the [Azure Paired Regions](#). Putting the secondary topic and subscription in a different region ensures that your new events will flow even if the primary region goes down.

You should now have:

- An event receiver website for testing.
- A primary topic in your primary region.
- A primary event subscription connecting your primary topic to the event receiver website.
- A secondary topic in your secondary region.

- A secondary event subscription connecting your primary topic to the event receiver website.

## Implement client-side failover

Now that you have a regionally redundant pair of topics and subscriptions setup, you're ready to implement client-side failover. There are several ways to accomplish it, but all failover implementations will have a common feature: if one topic is no longer healthy, traffic will redirect to the other topic.

### Basic client-side implementation

The following sample code is a simple .NET publisher that will always attempt to publish to your primary topic first. If it doesn't succeed, it will then failover the secondary topic. In either case, it also checks the health api of the other topic by doing a GET on `https://<topic-name>.<topic-region>.eventgrid.azure.net/api/health`. A healthy topic should always respond with **200 OK** when a GET is made on the `/api/health` endpoint.

#### NOTE

The following sample code is only for demonstration purposes and is not intended for production use.

```
using System;
using System.Net.Http;
using System.Collections.Generic;
using Microsoft.Azure.EventGrid;
using Microsoft.Azure.EventGrid.Models;
using Newtonsoft.Json;

namespace EventGridFailoverPublisher
{
    // This captures the "Data" portion of an EventGridEvent on a custom topic
    class FailoverEventData
    {
        [JsonProperty(PropertyName = "teststatus")]
        public string TestStatus { get; set; }
    }

    class Program
    {
        static void Main(string[] args)
        {
            // TODO: Enter the endpoint each topic. You can find this topic endpoint value
            // in the "Overview" section in the "Event Grid Topics" blade in Azure Portal..
            string primaryTopic = "https://<primary-topic-name>.<primary-topic-
region>.eventgrid.azure.net/api/events";

            string secondaryTopic = "https://<secondary-topic-name>.<secondary-topic-
region>.eventgrid.azure.net/api/events";

            // TODO: Enter topic key for each topic. You can find this in the "Access Keys" section in the
            // "Event Grid Topics" blade in Azure Portal.
            string primaryTopicKey = "<your-primary-topic-key>";
            string secondaryTopicKey = "<your-secondary-topic-key>";

            string primaryTopicHostname = new Uri(primaryTopic).Host;
            string secondaryTopicHostname = new Uri(secondaryTopic).Host;

            Uri primaryTopicHealthProbe = new Uri("https://" + primaryTopicHostname + "/api/health");
            Uri secondaryTopicHealthProbe = new Uri("https://" + secondaryTopicHostname + "/api/health");

            var httpClient = new HttpClient();

            try
            {
                TopicCredentials topicCredentials = new TopicCredentials(primaryTopicKey);
                EventGridClient client = new EventGridClient(topicCredentials);
```

```

        client.PublishEventsAsync(primaryTopicHostname, GetEventsList()).GetAwaiter().GetResult();
        Console.WriteLine("Published events to primary Event Grid topic.");

        HttpResponseMessage health = httpClient.GetAsync(secondaryTopicHealthProbe).Result;
        Console.WriteLine("\n\nSecondary Topic health " + health);
    }
    catch (Microsoft.Rest.Azure.CloudException e)
    {
        TopicCredentials topicCredentials = new TopicCredentials(secondaryTopicKey);
        EventGridClient client = new EventGridClient(topicCredentials);

        client.PublishEventsAsync(secondaryTopicHostname, GetEventsList()).GetAwaiter().GetResult();
        Console.WriteLine("Published events to secondary Event Grid topic. Reason for primary topic
failure:\n\n" + e);

        HttpResponseMessage health = httpClient.GetAsync(primaryTopicHealthProbe).Result;
        Console.WriteLine("\n\nPrimary Topic health " + health);
    }

    Console.ReadLine();
}

static IList<EventGridEvent> GetEventsList()
{
    List<EventGridEvent> eventsList = new List<EventGridEvent>();

    for (int i = 0; i < 5; i++)
    {
        eventsList.Add(new EventGridEvent()
        {
            Id = Guid.NewGuid().ToString(),
            EventType = "Contoso.Failover.Test",
            Data = new FailoverEventData()
            {
                TestStatus = "success"
            },
            EventTime = DateTime.Now,
            Subject = "test" + i,
            DataVersion = "2.0"
        });
    }

    return eventsList;
}
}
}
}

```

## Try it out

Now that you have all of your components in place, you can test out your failover implementation. Run the above sample in Visual Studio code, or your favorite environment. Replace the following four values with the endpoints and keys from your topics:

- `primaryTopic` - the endpoint for your primary topic.
- `secondaryTopic` - the endpoint for your secondary topic.
- `primaryTopicKey` - the key for your primary topic.
- `secondaryTopicKey` - the key for your secondary topic.

Try running the event publisher. You should see your test events land in your Event Grid viewer like below.



Event Type	Subject
	Contoso.Failover.Test

To make sure your failover is working, you can change a few characters in your primary topic key to make it no longer valid. Try running the publisher again. You should still see new events appear in your Event Grid viewer, however when you look at your console, you'll see that they are now being published via the secondary topic.

### Possible extensions

There are many ways to extend this sample based on your needs. For high-volume scenarios, you may want to regularly check the topic's health api independently. That way, if a topic were to go down, you don't need to check it with every single publish. Once you know a topic isn't healthy, you can default to publishing to the secondary topic.

Similarly, you may want to implement fallback logic based on your specific needs. If publishing to the closest data center is critical for you to reduce latency, you can periodically probe the health api of a topic that has failed over. Once it's healthy again, you'll know it's safe to fallback to the closer data center.

## Next steps

- Learn how to [receive events at an http endpoint](#)
- Discover how to [route events to Hybrid Connections](#)
- Learn about [disaster recovery using Azure DNS and Traffic Manager](#)

# Track asynchronous Azure operations

3/31/2021 • 4 minutes to read • [Edit Online](#)

Some Azure REST operations run asynchronously because the operation can't be completed quickly. This article describes how to track the status of asynchronous operations through values returned in the response.

## Status codes for asynchronous operations

An asynchronous operation initially returns an HTTP status code of either:

- 201 (Created)
- 202 (Accepted)

When the operation successfully completes, it returns either:

- 200 (OK)
- 204 (No Content)

Refer to the [REST API documentation](#) to see the responses for the operation you're executing.

After getting the 201 or 202 response code, you're ready to monitor the status of the operation.

## URL to monitor status

There are two different ways to monitor the status the asynchronous operation. You determine the correct approach by examining the header values that are returned from your original request. First, look for:

- `Azure-AsyncOperation` - URL for checking the ongoing status of the operation. If your operation returns this value, use it to track the status of the operation.
- `Retry-After` - The number of seconds to wait before checking the status of the asynchronous operation.

If `Azure-AsyncOperation` isn't one of the header values, then look for:

- `Location` - URL for determining when an operation has completed. Only use this value only when `Azure-AsyncOperation` isn't returned.
- `Retry-After` - The number of seconds to wait before checking the status of the asynchronous operation.

## Azure-AsyncOperation request and response

If you have a URL from the `Azure-AsyncOperation` header value, send a GET request to that URL. Use the value from `Retry-After` to schedule how often to check the status. You'll get a response object that indicates the status of the operation. A different response is returned when checking the status of the operation with the `Location` URL. For more information about the response from a location URL, see [Create storage account \(202 with Location and Retry-After\)](#).

The response properties can vary but always include the status of the asynchronous operation.

```
{  
    "status": "{status-value}"  
}
```

The following example shows other values that might be returned from the operation:

```
{
  "id": "{resource path from GET operation}",
  "name": "{operation-id}",
  "status" : "Succeeded | Failed | Canceled | {resource provider values}",
  "startTime": "2017-01-06T20:56:36.002812+00:00",
  "endTime": "2017-01-06T20:56:56.002812+00:00",
  "percentComplete": {double between 0 and 100 },
  "properties": {
    /* Specific resource provider values for successful operations */
  },
  "error" : {
    "code": "{error code}",
    "message": "{error description}"
  }
}
```

The error object is returned when the status is Failed or Canceled. All other values are optional. The response you receive may look different than the example.

## provisioningState values

Operations that create, update, or delete (PUT, PATCH, DELETE) a resource typically return a `provisioningState` value. When an operation has completed, one of following three values is returned:

- Succeeded
- Failed
- Canceled

All other values indicate the operation is still running. The resource provider can return a customized value that indicates its state. For example, you may receive **Accepted** when the request is received and running.

## Example requests and responses

### Start virtual machine (202 with Azure-AsyncOperation)

This example shows how to determine the status of [start operation for virtual machines](#). The initial request is in the following format:

```
POST  
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Compute/virtualMachines/{vm-name}/start?api-version=2019-12-01
```

It returns status code 202. Among the header values, you see:

```
Azure-AsyncOperation : https://management.azure.com/subscriptions/{subscription-id}/providers/Microsoft.Compute/locations/{region}/operations/{operation-id}?api-version=2019-12-01
```

To check the status of the asynchronous operation, sending another request to that URL.

```
GET  
https://management.azure.com/subscriptions/{subscription-id}/providers/Microsoft.Compute/locations/{region}/operations/{operation-id}?api-version=2019-12-01
```

The response body contains the status of the operation:

```
{  
  "startTime": "2017-01-06T18:58:24.7596323+00:00",  
  "status": "InProgress",  
  "name": "9a062a88-e463-4697-bef2-fe039df73a02"  
}
```

## Deploy resources (201 with Azure-AsyncOperation)

This example shows how to determine the status of [deployments operation for deploying resources](#) to Azure. The initial request is in the following format:

```
PUT  
https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/microsoft.resources/deployments/{deployment-name}?api-version=2020-06-01
```

It returns status code 201. The body of the response includes:

```
"provisioningState": "Accepted",
```

Among the header values, you see:

```
Azure-AsyncOperation: https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Resources/deployments/{deployment-name}/operationStatuses/{operation-id}?api-version=2020-06-01
```

To check the status of the asynchronous operation, sending another request to that URL.

```
GET  
https://management.azure.com/subscriptions/{subscription-id}/resourcegroups/{resource-group}/providers/Microsoft.Resources/deployments/{deployment-name}/operationStatuses/{operation-id}?api-version=2020-06-01
```

The response body contains the status of the operation:

```
{  
  "status": "Running"  
}
```

When the deployment is finished, the response contains:

```
{  
  "status": "Succeeded"  
}
```

## Create storage account (202 with Location and Retry-After)

This example shows how to determine the status of the [create operation for storage accounts](#). The initial request is in the following format:

```
PUT  
https://management.azure.com/subscriptions/{subscription-id}/resourceGroups/{resource-group}/providers/Microsoft.Storage/storageAccounts/{storage-name}?api-version=2019-06-01
```

And the request body contains properties for the storage account:

```
{  
    "location": "South Central US",  
    "properties": {},  
    "sku": {  
        "name": "Standard_LRS"  
    },  
    "kind": "Storage"  
}
```

It returns status code 202. Among the header values, you see the following two values:

```
Location: https://management.azure.com/subscriptions/{subscription-id}/providers/Microsoft.Storage/operations/{operation-id}?monitor=true&api-version=2019-06-01  
Retry-After: 17
```

After waiting for number of seconds specified in Retry-After, check the status of the asynchronous operation by sending another request to that URL.

```
GET  
https://management.azure.com/subscriptions/{subscription-id}/providers/Microsoft.Storage/operations/{operation-id}?monitor=true&api-version=2019-06-01
```

If the request is still running, you receive a status code 202. If the request has completed, you receive a status code 200, and the body of the response contains the properties of the storage account that has been created.

## Next steps

- For documentation about each REST operation, see [REST API documentation](#).
- For information about deploying templates through the Resource Manager REST API, see [Deploy resources with Resource Manager templates and Resource Manager REST API](#).

# Move Azure Event Grid system topics to another region

5/25/2021 • 5 minutes to read • [Edit Online](#)

You might want to move your resources to another region for a number of reasons. For example, to take advantage of a new Azure region, to meet internal policy and governance requirements, or in response to capacity planning requirements.

Here are the high-level steps covered in this article:

- **Export the resource group** that contains the Azure Storage account and its associated system topic to a Resource Manager template. You can also export a template only for the system topic. If you go this route, remember to move the Azure event source (in this example, an Azure Storage account) to the other region before moving the system topic. Then, in the exported template for the system topic, update the external ID for the storage account in the target region.
- **Modify the template** to add the `endpointUrl` property to point to a webhook that subscribes to the system topic. When the system topic is exported, its subscription (in this case, it's a webhook) is also exported to the template, but the `endpointUrl` property isn't included. So, you need to update it to point to the endpoint that subscribes to the topic. Also, update the value of the `location` property to the new location or region. For other types of event handlers, you need to update only the location.
- **Use the template to deploy resources** to the target region. You'll specify names for the storage account and the system topic to be created in the target region.
- **Verify the deployment.** Verify that the webhook is invoked when you upload a file to the blob storage in the target region.
- To complete the move, delete resources (event source and system topic) from the source region.

## Prerequisites

- Complete the [Quickstart: Route Blob storage events to web endpoint with the Azure portal](#) in the source region. This step is **optional**. Do it to test steps in this article. Keep the storage account in a separate resource group from the App Service and App Service plan.
- Ensure that the Event Grid service is available in the target region. See [Products available by region](#).

## Prepare

To get started, export a Resource Manager template for the resource group that contains the system event source (Azure Storage account) and its associated system topic.

1. Sign in to the [Azure portal](#).
2. Select **Resource groups** on the left menu. Then, select the resource group that contains the event source for which the system topic was created. In the following example, it's the **Azure Storage** account. The resource group contains the storage account and its associated system topic.

The screenshot shows the Azure Resource Groups portal. On the left, there's a navigation menu with options like Overview, Activity log, Access control (IAM), Tags, Events, Settings, Quickstart, Deployments, Policies, Properties, Locks, and Export template. The 'Export template' option is highlighted with a yellow circle labeled '1'. In the main content area, there's a search bar and a toolbar with Add, Edit columns, Delete resource group, Refresh, Move, Export to CSV, Open query, and more. Below the toolbar is a 'Essentials' blade titled 'Filter by name...'. It shows two records: 'spcontosostorage' (Storage account, Type: Storage account, Location: East US) and 'spcontosostorage-systopic' (Event Grid System Topic, Type: Event Grid System Topic, Location: East US). There are also filter and sorting options.

3. On the left menu, select **Export template** under **Settings**, and then, select **Download** on the toolbar.

The screenshot shows the 'Export template' blade for the 'spegridr' resource group. On the left, there's a sidebar with 'Export template' (highlighted with a red box and yellow circle '1') and other options like Cost Management, Cost analysis, Cost alerts (preview), Budgets, and Advisor recommendations. The main area has a toolbar with Download (highlighted with a yellow circle '2'), Add to library (preview), Deploy, and a help icon. Below the toolbar, it says 'To export related resources, select the resources from the Resource Group view then select the "Export template" option from the tool bar.' Under 'Template', there are tabs for Template, Parameters, and Scripts. The 'Template' tab shows a JSON code editor with deployment template code. The 'Parameters' and 'Resources' sections are also visible.

4. Locate the .zip file that you downloaded from the portal, and unzip that file to a folder of your choice. This zip file contains template and parameters JSON files.

5. Open the **template.json** in an editor of your choice.

6. URL for the Webhook isn't exported to the template. So, do the following steps:

- In the template file, search for **WebHook**.
- In the **Properties** section, add a comma ( , ) character at the end of the last line. In this example, it's `"preferredBatchSizeInKilobytes": 64,`.
- Add the `endpointUrl` property with the value set to your Webhook URL as shown in the following example.

```
"destination": {
    "properties": {
        "maxEventsPerBatch": 1,
        "preferredBatchSizeInKilobytes": 64,
        "endpointUrl": "https://mysite.azurewebsites.net/api/updates"
    },
    "endpointType": "WebHook"
}
```

#### NOTE

For other types of event handlers, all properties are exported to the template. You only need to update the `location` property to the target region as shown in the next step.

7. Update `location` for the **storage account** resource to the target region or location. To obtain location codes, see [Azure locations](#). The code for a region is the region name with no spaces, for example, `West US` is equal to `westus`.

```
"type": "Microsoft.Storage/storageAccounts",
"apiVersion": "2019-06-01",
"name": "[parameters('storageAccounts_spegridstorage080420_name')]",
"location": "westus",
```

8. Repeat the step to update `location` for the **system topic** resource in the template.

```
"type": "Microsoft.EventGrid/systemTopics",
"apiVersion": "2020-04-01-preview",
"name": "[parameters('systemTopics_spegridsystopic080420_name')]",
"location": "westus",
```

9. Save the template.

## Recreate

Deploy the template to create a storage account and a system topic for the storage account in the target region.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and then press **ENTER**.
3. Select **Template deployment**.
4. Select **Create**.
5. Select **Build your own template in the editor**.
6. Select **Load file**, and then follow the instructions to load the **template.json** file that you downloaded in the last section.
7. Select **Save** to save the template.
8. On the **Custom deployment** page, follow these steps.
  - a. Select an **Azure subscription**.
  - b. Select an existing **resource group** in the target region or create one.
  - c. For **Region**, select the target region. If you selected an existing resource group, this setting is read-only.
  - d. For the **system topic name**, enter a name for the system topic that will be associated with the storage account.
  - e. For the **storage account name**, enter a name for the storage account to be created in the target region.

## Custom deployment

Deploy from a custom template

Select a template    Basics    Review + create

### Template



Edit template

Edit parameters

### Deployment scope

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* ⓘ

<Azure subscription name>

Resource group \* ⓘ

(New) spegridrg0827

Create new

### Parameters

Region \* ⓘ

West US

System

Topics\_spegridssystopic080420\_name

spegridwestsstopic

Storage

Accounts\_spegridstorage080420\_name

spegridweststorage

Review + create

< Previous

Next : Review + create >

f. Select **Review + create** at the bottom of the page.

g. On the **Review + create** page, review settings, and select **Create**.

## Verify

1. After the deployment succeeds, select **Goto resource group**.
2. On the **Resource group** page, verify that the event source (in this example, Azure Storage account) and the system topic are created.
3. Upload a file to a container in the Azure Blob storage, and verify that the webhook has received the event. For more information, see [Send an event to your endpoint](#).

## Discard or clean up

To complete the move, delete the resource group that contains the storage account and its associated system topic in the source region.

If you want to start over, delete the resource group in the target region, and repeat steps in the [Prepare](#) and [Recreate](#) sections of this article.

To delete a resource group (source or target) by using the Azure portal:

1. In the search window at the top of Azure portal, type **Resource groups**, and select **Resource groups** from search results.
2. Select the resource group to delete, and select **Delete** from the toolbar.

The screenshot shows the Azure portal interface for a resource group named 'spegridrg'. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Events, Settings (Quickstart, Deployments, Policies, Properties, Locks, Export template), and a search bar. The main content area displays subscription information (Visual Studio Ultimate with MSDN) and deployment details (1 succeeded). A table lists resources: 'spcontosostorage' (Storage account, East US) and 'spcontosostorage-systopic' (Event Grid System Topic, East US). At the top right, there are buttons for Add, Edit columns, Delete resource group (highlighted with a red box), Refresh, Move, Export to CSV, Open query, and more.

3. On the confirmation page, enter the name of the resource group, and select **Delete**.

## Next steps

You learned how to move an Azure event source and its associated system topic from one region to another region. See the following articles for moving custom topics, domains, and partner namespaces across regions.

- [Move custom topics across regions](#).
- [Move domains across regions](#).

To learn more about moving resources between regions and disaster recovery in Azure, see the following article: [Move resources to a new resource group or subscription](#).

# Move Azure Event Grid custom topics to another region

5/25/2021 • 4 minutes to read • [Edit Online](#)

You might want to move your resources to another region for a number of reasons. For example, to take advantage of a new Azure region, to meet internal policy and governance requirements, or in response to capacity planning requirements.

Here are the high-level steps covered in this article:

- **Export the custom topic** resource to an Azure Resource Manager template.

#### IMPORTANT

Only the custom topic is exported to the template. Any subscriptions for the topic aren't exported.

- **Use the template to deploy the custom topic** to the target region.
- **Create subscriptions manually** in the target region. When you exported the custom topic to a template in the current region, only the topic is exported. Subscriptions aren't included in the template, so create them manually after the custom topic is created in the target region.
- **Verify the deployment.** Verify that the custom topic is created in the target region.
- To complete the move, delete custom topic from the source region.

## Prerequisites

- Complete the [Quickstart: Route custom events to web endpoint](#) in the source region. Do this step so that you can test steps in this article.
- Ensure that the Event Grid service is available in the target region. See [Products available by region](#).

## Prepare

To get started, export a Resource Manager template for the custom topic.

1. Sign in to the [Azure portal](#).
2. In the search bar, type **Event Grid topics**, and select **Event Grid Topics** from the results list.

The screenshot shows the Microsoft Azure dashboard with a search bar at the top containing "Event Grid Topics". Below the search bar, there's a "Services" section with a "See all" link. Under "Event Grid Topics", there are several items listed: Event Grid Partner Topics, Event Grid System Topics, Event Grid Domains, Event Grid Subscriptions, Event Grid Partner Namespaces, Event Grid Partner Registrations, Recent, Event Hubs, and SendGrid Accounts. On the left sidebar, under "All resources", there are several subscriptions listed: mytopic0130, spegridsite080420, spegriddomain, ehubprivateendpoint, and spegridvnet.

3. Select the **topic** that you want to export to a Resource Manager template.

The screenshot shows the "Event Grid Topics" page. At the top, there are filter options: Subscription (2 of 28 selected), Resource group (all), Location (all), and a "No grouping" button. Below the filters, it says "Showing 1 to 1 of 1 records." A table displays one record: mytopic0130, which is an Event Grid Topic located in the myegridrg resource group and the East US location. The "mytopic0130" row is highlighted with a red box.

4. On the **Event Grid Topic** page, select **Export Template** under **Settings** on the left menu, and then select **Download** on the toolbar.

The screenshot shows the "mytopic0130 | Export template" dialog. On the left, there's a sidebar with "Overview", "Activity log", "Access control (IAM)", "Tags", "Settings" (with "Access keys", "Networking", "Identity", "Locks" listed), and "Export template" highlighted with a red box. The main area has a "Download" button highlighted with a red box. Below the download button, there's a note: "To export related resources, select the resources from the Resource Group view then select the "Export template" option from the tool bar." There are tabs for "Template", "Parameters", and "Scripts". The "Template" tab shows a JSON code block:

```

1  {
2    "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
3    "contentVersion": "1.0.0.0",
4    "parameters": {
5      "topics_mytopic0130_name": {
6        "defaultValue": "mytopic0130",
7        "type": "String"
8      }
9    },
10   "variables": {},
11   "resources": [
12     {
13       "type": "Microsoft.EventGrid/topics",
14       "apiVersion": "2020-06-01",
15       "name": "[parameters('topics_mytopic0130_name')]",
16       "location": "eastus",
17       "properties": {
18         "inputSchema": "EventGridSchema",
19         "publicNetworkAccess": "Enabled"
20       }
21     }
22   ]
23 }

```

### IMPORTANT

Only the topic is exported to the template. Subscriptions for the topic aren't exported. So, you need to create subscriptions for the topic after you move the topic to the target region.

5. Locate the .zip file you downloaded from the portal, and unzip that file to a folder of your choice. This zip file contains template and parameters JSON files.
6. Open the **template.json** in an editor of your choice.
7. Update `location` for the **topic** resource to the target region or location. To obtain location codes, see [Azure locations](#). The code for a region is the region name with no spaces, for example, `West US` is equal to `westus`.

```
"type": "Microsoft.EventGrid/topics",
"apiVersion": "2020-06-01",
"name": "[parameters('topics_mytopic0130_name')]",
"location": "westus"
```

8. Save the template.

## Recreate

Deploy the template to create a custom topic in the target region.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and then press **ENTER**.
3. Select **Template deployment**.
4. Select **Create**.
5. Select **Build your own template in the editor**.
6. Select **Load file**, and then follow the instructions to load the **template.json** file that you downloaded in the last section.
7. Select **Save** to save the template.
8. On the **Custom deployment** page, follow these steps:
  - a. Select an **Azure subscription**.
  - b. Select an existing **resource group** in the target region or create one.
  - c. For **Region**, select the target region. If you selected an existing resource group, this setting is read-only.
  - d. For the **topic name**, enter a new name for the topic.
  - e. Select **Review + create** at the bottom of the page.

## Custom deployment

Deploy from a custom template

Select a template    **Basics**    Review + create

### Template

 Customized template  1 resource

 Edit template

 Edit parameters

### Deployment scope

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \* 

<Azure subscription name>

Resource group \* 

(New) spegridwestcustomrg

[Create new](#)

### Parameters

Region \* 

West US

Topics\_mytopic0130\_name

mytopicwest0827

[Review + create](#)

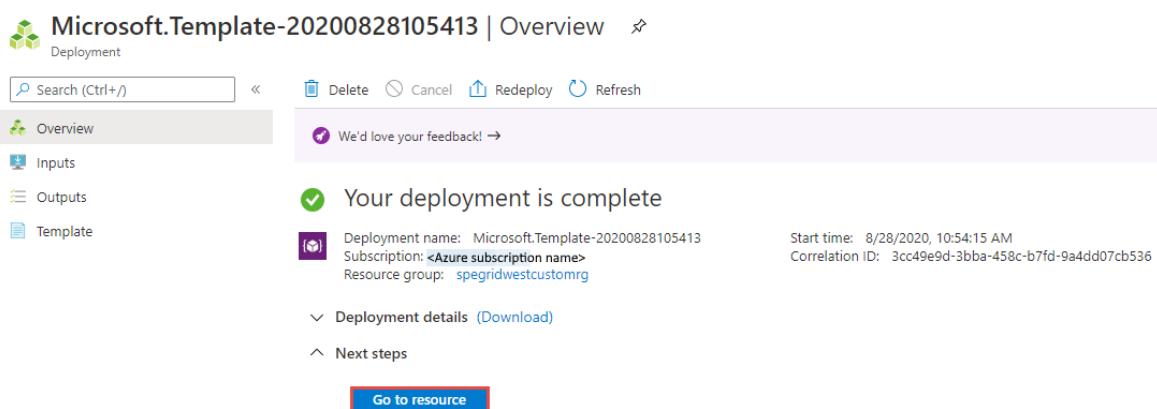
[< Previous](#)

[Next : Review + create >](#)

- f. On the **Review + create** page, review settings, and select **Create**.

## Verify

1. After the deployment succeeds, select **Go to resource**.

 Microsoft.Template-20200828105413 | Overview

Deployment

Search (Ctrl+ /)    Refresh

Overview  We'd love your feedback! →

Inputs

Outputs

Template

✓ Your deployment is complete

Deployment name: Microsoft.Template-20200828105413  
Subscription: <Azure subscription name>  
Resource group: spegridwestcustomrg

Start time: 8/28/2020, 10:54:15 AM  
Correlation ID: 3cc49e9d-3bba-458c-b7fd-9a4dd07cb536

Deployment details (Download)

Next steps

[Go to resource](#)

2. Confirm that you see the **Event Grid Topic** page for the custom topic.

3. Follow steps in the [Route custom events to a web endpoint](#) to send events to the topic. Verify that the webhook event handler is invoked.

## Discard or clean up

To complete the move, delete the custom topic in the source region.

If you want to start over, delete the topic in the target region, and repeat steps in the [Prepare](#) and [Recreate](#) sections of this article.

To delete a custom topic by using the Azure portal:

1. In the search window at the top of Azure portal, type **Event Grid Topics**, and select **Event Grid Topics** from search results.
2. Select the topic to delete, and select **Delete** from the toolbar.
3. On the confirmation page, enter the name of the resource group, and select **Delete**.

To delete the resource group that contains the custom topic by using the Azure portal:

1. In the search window at the top of Azure portal, type **Resource groups**, and select **Resource groups** from search results.
2. Select the resource group to delete, and select **Delete** from the toolbar.
3. On the confirmation page, enter the name of the resource group, and select **Delete**.

## Next steps

You learned how to move an Event Grid custom topic from one region to another region. See the following articles for moving system topics, domains, and partner namespaces across regions.

- [Move system topics across regions](#).
- [Move domains across regions](#).

To learn more about moving resources between regions and disaster recovery in Azure, see the following article: [Move resources to a new resource group or subscription](#).

# Move Azure Event Grid domains to another region

5/25/2021 • 3 minutes to read • [Edit Online](#)

You might want to move your resources to another region for a number of reasons. For example, to take advantage of a new Azure region, to meet internal policy and governance requirements, or in response to capacity planning requirements.

Here are the high-level steps covered in this article:

- **Export the domain resource to an Azure Resource Manager template.**

## IMPORTANT

The domain resource and topics in the domain are exported to the template. Subscriptions to domain topics aren't exported.

- **Use the template to deploy the domain to the target region.**
- **Create subscriptions for domain topics manually** in the target region. When you exported the domain to a template in the current region, subscriptions for domain topics aren't exported. So, create them after the domain and domain topics are created in the target region.
- **Verify the deployment.** Send an event to a domain topic in the domain and verify the event handler associated with the subscription is invoked.
- To complete the move, delete domain from the source region.

## Prerequisites

- Ensure that the Event Grid service is available in the target region. See [Products available by region](#).

## Prepare

To get started, export a Resource Manager template for the domain.

1. Sign in to the [Azure portal](#).
2. In the search bar, type **Event Grid Domains**, and select **Event Grid Domains** from the results list.

3. Select the **domain** that you want to export to a Resource Manager template.

## Event Grid Domains

Microsoft

Showing 1 to 1 of 1 records.

Name	Type	Resource group
<a href="#">spegriddomain</a>	Event Grid Domain	spegriddomainrg

- On the Event Grid Domain page, select Export Template under Settings on the left menu, and then select Download on the toolbar.

spegriddomain | Export template

Event Grid Domain

Search (Ctrl+ /) < [Download](#) Add to library (preview) Deploy

To export related resources, select the resources from the Resource Group view then select the "Export template" option from the tool bar.

Include parameters

Template Parameters Scripts

Settings

- Access keys
- Networking
- Identity
- Locks
- [Export template](#)

Monitoring

- Alerts
- Metrics
- Diagnostic settings
- Logs

Support + troubleshooting

New support request

```
8     }
9     },
10    "variables": {},
11    "resources": [
12      {
13        "type": "Microsoft.EventGrid/domains",
14        "apiVersion": "2020-06-01",
15        "name": "[parameters('domains_spegriddomain_name')]",
16        "location": "eastus",
17        "properties": {
18          "inputSchema": "EventGridSchema",
19          "publicNetworkAccess": "Enabled"
20        }
21      },
22      {
23        "type": "Microsoft.EventGrid/domains/topics",
24        "apiVersion": "2020-06-01",
25        "name": "[concat(parameters('domains_spegriddomain_name'), '/spdomaintopic1')]",
26        "dependsOn": [
27          "[resourceId('Microsoft.EventGrid/domains', parameters('domains_spegriddomain_name'))]"
28        ]
29      },
30      {
31        "type": "Microsoft.EventGrid/domains/topics",
32        "apiVersion": "2020-06-01",
33        "name": "[concat(parameters('domains_spegriddomain_name'), '/spdomaintopic2')]",
34        "dependsOn": [
35          "[resourceId('Microsoft.EventGrid/domains', parameters('domains_spegriddomain_name'))]"
36        ]
37      }
38    ]
39  ]
```

### IMPORTANT

Domain and domain topics are exported. Subscriptions for domain topics aren't exported. So, you need to create subscriptions for domain topics after you move domain topics.

- Locate the .zip file that you downloaded from the portal, and unzip that file to a folder of your choice. This zip file contains template and parameters JSON files.
- Open the **template.json** in an editor of your choice.
- Update `location` for the **domain** resource to the target region or location. To obtain location codes, see [Azure locations](#). The code for a region is the region name with no spaces, for example, `West US` is equal to `westus`.

```
"type": "Microsoft.EventGrid/domains",
"apiVersion": "2020-06-01",
"name": "[parameters('domains_spegriddomain_name')]",
"location": "westus",
```

- Save the template.

## Recreate

Deploy the template to create the domain and domain topics in the target region.

1. In the Azure portal, select **Create a resource**.
2. In **Search the Marketplace**, type **template deployment**, and then press **ENTER**.
3. Select **Template deployment**.
4. Select **Create**.
5. Select **Build your own template in the editor**.
6. Select **Load file**, and then follow the instructions to load the **template.json** file that you downloaded in the last section.
7. Select **Save** to save the template.
8. On the **Custom deployment** page, follow these steps:
  - a. Select an Azure **subscription**.
  - b. Select an existing **resource group** in the target region or create one.
  - c. For **Region**, select the target region. If you selected an existing resource group, this setting is read-only.
  - d. For the **domain name**, enter a new name for the domain.
  - e. Select **Review + create**.

### Custom deployment

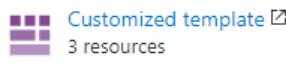
Deploy from a custom template

Select a template

Basics

Review + create

#### Template



Customized template

3 resources

Edit template

Edit parameters

#### Deployment scope

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription \*

<Azure subscription name>

Resource group \*

(New) spegriddomainwestrg

[Create new](#)

#### Parameters

Region \*

West US

Domains\_spegriddomain\_name

spdomainwest0828

[Review + create](#)

[< Previous](#)

[Next : Review + create >](#)

- f. After the validation of the template succeeds, select **Create** at the bottom of the page to deploy the resource.
- g. After the deployment succeeds, select **Go to resource group** to navigate to the resource group page. Confirm that there's a domain in the resource group. Select the domain. Confirm that there

are domain topics in the domain.

## Discard or clean up

To complete the move, delete the domain in the source region.

If you want to start over, delete the domain in the target region, and repeat steps in the [Prepare](#) and [Recreate](#) sections of this article.

To delete a domain by using the Azure portal:

1. In the search window at the top of Azure portal, type **Event Grid Domains**, and select **Event Grid Domains** from search results.
2. Select the domain to delete, and select **Delete** from the toolbar.
3. On the confirmation page, enter the name of the resource group, and select **Delete**.

To delete the resource group that contains the domain by using the Azure portal:

1. In the search window at the top of Azure portal, type **Resource groups**, and select **Resource groups** from search results.
2. Select the resource group to delete, and select **Delete** from the toolbar.
3. On the confirmation page, enter the name of the resource group, and select **Delete**.

## Next steps

You learned how to move an Event Grid domain from one region to another region. See the following articles for moving system topics, custom topics, and partner namespaces across regions.

- [Move system topics across regions](#).
- [Move custom topics across regions](#).

To learn more about moving resources between regions and disaster recovery in Azure, see the following article: [Move resources to a new resource group or subscription](#).

# Configure private endpoints for Azure Event Grid topics or domains

5/28/2021 • 11 minutes to read • [Edit Online](#)

You can use [private endpoints](#) to allow ingress of events directly from your virtual network to your topics and domains securely over a [private link](#) without going through the public internet. The private endpoint uses an IP address from the VNet address space for your topic or domain. For more conceptual information, see [Network security](#).

This article describes how to configure private endpoints for topics or domains.

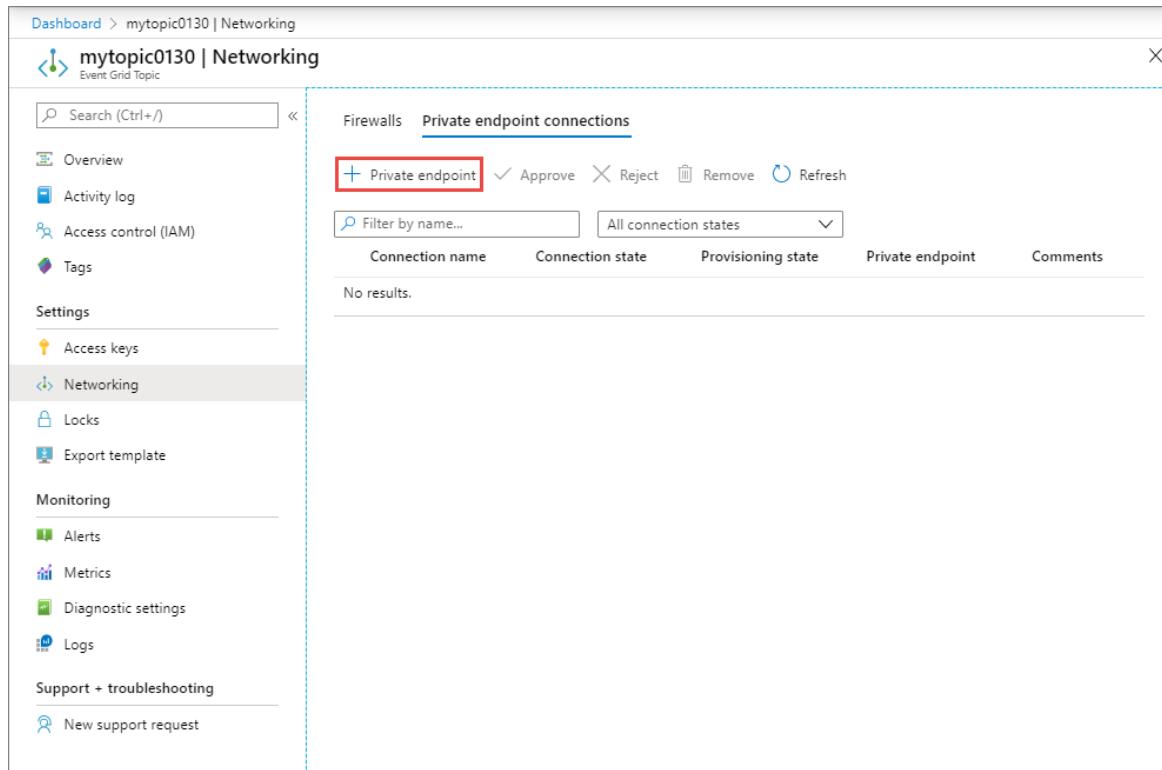
## Use Azure portal

This section shows you how to use the Azure portal to create a private endpoint for a topic or a domain.

### NOTE

The steps shown in this section are mostly for topics. You can use similar steps to create private endpoints for [domains](#).

1. Sign in to the [Azure portal](#) and navigate to your topic or domain.
2. Switch to the **Networking** tab of your topic page. Select **+ Private endpoint** on the toolbar.



3. On the **Basics** page, follow these steps:
  - a. Select an **Azure subscription** in which you want to create the private endpoint.
  - b. Select an **Azure resource group** for the private endpoint.
  - c. Enter a **name** for the endpoint.
  - d. Select the **region** for the endpoint. Your private endpoint must be in the same region as your virtual

network, but can in a different region from the private link resource (in this example, an event grid topic).

- e. Then, select **Next: Resource >** button at the bottom of the page.

The screenshot shows the 'Create a private endpoint' wizard on the 'Basics' step. At the top, there's a note: 'Changes you make on this tab may affect any configuration you've done on other tabs. Review all options prior to creating the private endpoint.' Below are sections for 'Project details' and 'Instance details'. In 'Project details', 'Subscription' is set to '<Subscription name>' and 'Resource group' is set to 'myegridrg'. In 'Instance details', 'Name' is 'myprivateendpoint' and 'Region' is '(US) East US 2'. At the bottom, there are '< Previous' and 'Next : Resource >' buttons.

4. On the **Resource** page, follow these steps:

- a. For connection method, if you select **Connect to an Azure resource in my directory**, follow these steps. This example shows how to connect to an Azure resource in your directory.
  - a. Select the **Azure subscription** in which your **topic/domain** exists.
  - b. For **Resource type**, Select **Microsoft.EventGrid/topics** or **Microsoft.EventGrid/domains** for the **Resource type**.
  - c. For **Resource**, select an topic/domain from the drop-down list.
  - d. Confirm that the **Target subresource** is set to **topic** or **domain** (based on the resource type you selected).
- e. Select **Next: Configuration >** button at the bottom of the page.

The screenshot shows the 'Create a private endpoint' wizard on the 'Resource' step. It includes a note about Private Link options. Under 'Connection method', 'Connect to an Azure resource in my directory' is selected. The 'Subscription' dropdown is set to '<Subscription name>'. The 'Resource type' dropdown is set to 'Microsoft.EventGrid/topics'. The 'Resource' dropdown is set to 'mytopic0130'. The 'Target sub-resource' dropdown is set to 'topic'. At the bottom, there are '< Previous' and 'Next : Configuration >' buttons.

- b. If you select **Connect to a resource using a resource ID or an alias**, follow these steps:

- a. Enter the ID of the resource. For example:

```
/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>
```

- b. For **Resource**, enter **topic** or domain.  
c. (optional) Add a request message.  
d. Select **Next: Configuration >** button at the bottom of the page.

Create a private endpoint

✓ Basics ② **Resource** ③ Configuration ④ Tags ⑤ Review + create

Private Link offers options to create private endpoints for different Azure resources, like your private link service, a SQL server, or an Azure storage account. Select which resource you would like to connect to using this private endpoint. [Learn more](#)

Connection method ⓘ  Connect to an Azure resource in my directory.  Connect to an Azure resource by resource ID or alias.

Resource ID or alias \* ⓘ  ✓

Target sub-resource \* ⓘ  ✓

Request message ⓘ

< Previous Next : Configuration >

5. On the **Configuration** page, you select the subnet in a virtual network to where you want to deploy the private endpoint.
- a. Select a **virtual network**. Only virtual networks in the currently selected subscription and location are listed in the drop-down list.  
b. Select a **subnet** in the virtual network you selected.  
c. Select **Next: Tags >** button at the bottom of the page.

## Create a private endpoint

✓ Basics ✓ Resource 3 Configuration 4 Tags 5 Review + create

### Networking

To deploy the private endpoint, select a virtual network subnet. [Learn more](#)

Virtual network \* ⓘ

spegridvnet



Subnet \* ⓘ

default (172.17.0.0/24)



ⓘ If you have a network security group (NSG) enabled for the subnet above, it will be disabled for private endpoints on this subnet only. Other resources on the subnet will still have NSG enforcement.

### Private DNS integration

To connect privately with your private endpoint, you need a DNS record. We recommend that you integrate your private endpoint with a private DNS zone. You can also utilize your own DNS servers or create DNS records using the host files on your virtual machines. [Learn more](#)

Integrate with private DNS zone ⓘ

Yes No

**Review + create**

< Previous

Next : Tags >

6. On the **Tags** page, create any tags (names and values) that you want to associate with the private endpoint resource. Then, select **Review + create** button at the bottom of the page.
7. On the **Review + create**, review all the settings, and select **Create** to create the private endpoint.

## Create a private endpoint

✓ Validation passed

✓ Basics ✓ Resource ✓ Configuration ✓ Tags 5 Review + create

### Basics

Subscription <Subscription name>  
Resource group myegridrg  
Region East US  
Name mytopicprivateendpoint

### Resource

Subscription ID <Subscription ID>  
Link type Microsoft.EventGrid/topics  
Resource group myegridrg  
Resource mytopic0130  
Target sub-resource topic

### Configuration

Virtual network resource group spegridrg  
Virtual network spegridvnet  
Subnet default (172.17.0.0/24)  
Integrate with private DNS zone? No

### Tags

None

[Create](#)

[< Previous](#)

[Next >](#)

[Download a template for automation](#)

## Manage private link connection

When you create a private endpoint, the connection must be approved. If the resource for which you're creating a private endpoint is in your directory, you can approve the connection request provided you have sufficient permissions. If you're connecting to an Azure resource in another directory, you must wait for the owner of that resource to approve your connection request.

There are four provisioning states:

SERVICE ACTION	SERVICE CONSUMER PRIVATE ENDPOINT STATE	DESCRIPTION
None	Pending	Connection is created manually and is pending approval from the private Link resource owner.
Approve	Approved	Connection was automatically or manually approved and is ready to be used.
Reject	Rejected	Connection was rejected by the private link resource owner.
Remove	Disconnected	Connection was removed by the private link resource owner, the private endpoint becomes informative and should be deleted for cleanup.

SERVICE ACTION	SERVICE CONSUMER PRIVATE ENDPOINT STATE	DESCRIPTION

## How to manage a private endpoint connection

The following sections show you how to approve or reject a private endpoint connection.

1. Sign in to the [Azure portal](#).
2. In the search bar, type in **Event Grid topics** or **Event Grid domains**.
3. Select the **topic** or **domain** that you want to manage.
4. Select the **Networking** tab.
5. If there are any connections that are pending, you'll see a connection listed with **Pending** in the provisioning state.

### To approve a private endpoint

You can approve a private endpoint that's in the pending state. To approve, follow these steps:

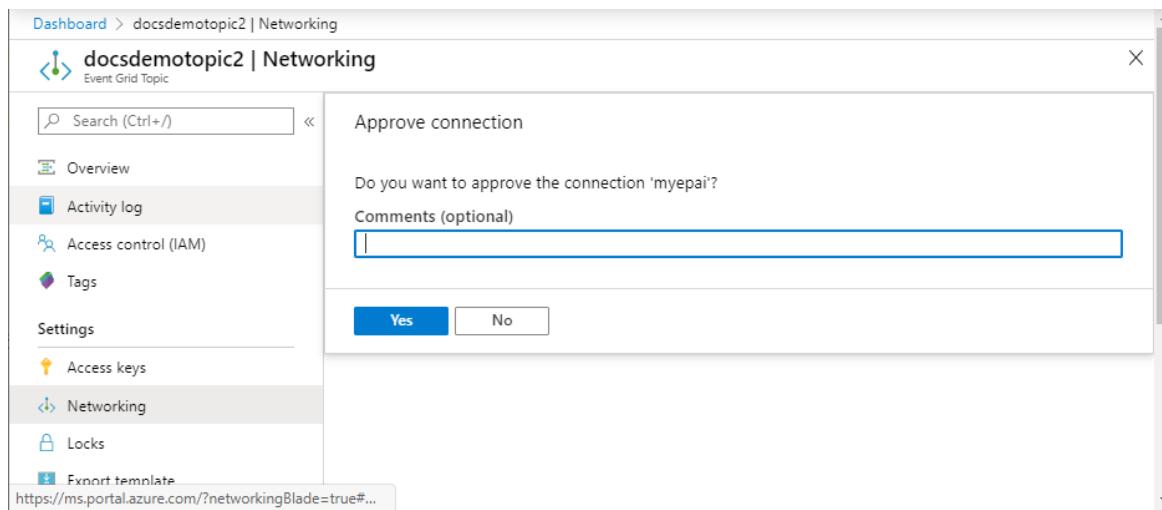
#### NOTE

The steps shown in this section are mostly for topics. You can use similar steps to approve private endpoints for domains.

1. Select the **private endpoint** you wish to approve, and select **Approve** on the toolbar.

Connection name	Connection state	Provisioning state	Comments
myepai	Pending	Succeeded	myepai

2. On the **Approve connection** dialog box, enter a comment (optional), and select **Yes**.



### 3. Confirm that you see the status of the endpoint as **Approved**.

The screenshot shows the 'Private endpoint connections' section. The table data is as follows:

Connection name	Connection state	Provisioning state	Private endpoint	Comments
myepai	Approved	Succeeded	<a href="#">myepai</a>	

## To reject a private endpoint

You can reject a private endpoint that's in the pending state or approved state. To reject, follow these steps:

### NOTE

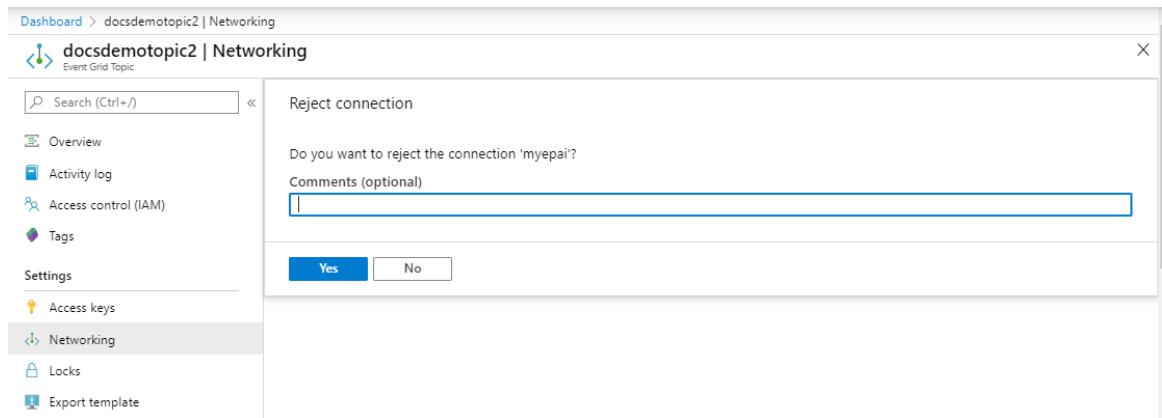
The steps shown in this section are for topics. You can use similar steps to reject private endpoints for **domains**.

### 1. Select the private endpoint you wish to reject, and select **Reject** on the toolbar.

The screenshot shows the 'Private endpoint connections' section. The table data is as follows:

Connection name	Connection state	Provisioning state	Private endpoint	Comments
demoprivateendpoint	Pending	Succeeded	<a href="#">demoprivateendpoint</a>	

### 2. On the **Reject connection** dialog box, enter a comment (optional), and select **Yes**.



3. Confirm that you see the status of the endpoint as **Rejected**.

Connection name	Connection state	Provisioning state	Private endpoint	Comments
myepai	Rejected	Succeeded	<a href="#">myepai</a>	

#### NOTE

You can't approve a private endpoint in the Azure portal once it's rejected.

## Use Azure CLI

To create a private endpoint, use the [az network private-endpoint create](#) method as shown in the following example:

```
az network private-endpoint create \
--resource-group <RESOURECE GROUP NAME> \
--name <PRIVATE ENDPOINT NAME> \
--vnet-name <VIRTUAL NETWORK NAME> \
--subnet <SUBNET NAME> \
--private-connection-resource-id "/subscriptions/<SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<TOPIC NAME>" \
--connection-name <PRIVATE LINK SERVICE CONNECTION NAME> \
--location <LOCATION> \
--group-ids topic
```

For descriptions of the parameters used in the example, see documentation for [az network private-endpoint create](#). A few points to note in this example are:

- For `private-connection-resource-id`, specify the resource ID of the **topic** or **domain**. The preceding example uses the type: `topic`.
- for `group-ids`, specify `topic` or `domain`. In the preceding example, `topic` is used.

To delete a private endpoint, use the [az network private-endpoint delete](#) method as shown in the following example:

```
az network private-endpoint delete --resource-group <RESOURSE GROUP NAME> --name <PRIVATE ENDPOINT NAME>
```

#### NOTE

The steps shown in this section are for topics. You can use similar steps to create private endpoints for **domains**.

## Prerequisites

Update the Azure Event Grid extension for CLI by running the following command:

```
az extension update -n eventgrid
```

If the extension isn't installed, run the following command to install it:

```
az extension add -n eventgrid
```

## Create a private endpoint

To create a private endpoint, use the [az network private-endpoint create](#) method as shown in the following example:

```
az network private-endpoint create \
--resource-group <RESOURSE GROUP NAME> \
--name <PRIVATE ENDPOINT NAME> \
--vnet-name <VIRTUAL NETWORK NAME> \
--subnet <SUBNET NAME> \
--private-connection-resource-id "/subscriptions/<SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<TOPIC NAME>" \
--connection-name <PRIVATE LINK SERVICE CONNECTION NAME> \
--location <LOCATION> \
--group-ids topic
```

For descriptions of the parameters used in the example, see documentation for [az network private-endpoint create](#). A few points to note in this example are:

- For `private-connection-resource-id`, specify the resource ID of the **topic** or **domain**. The preceding example uses the type: `topic`.
- For `group-ids`, specify `topic` or `domain`. In the preceding example, `topic` is used.

To delete a private endpoint, use the [az network private-endpoint delete](#) method as shown in the following example:

```
az network private-endpoint delete --resource-group <RESOURSE GROUP NAME> --name <PRIVATE ENDPOINT NAME>
```

#### NOTE

The steps shown in this section are for topics. You can use similar steps to create private endpoints for **domains**.

## Sample script

Here's a sample script that creates the following Azure resources:

- Resource group
- Virtual network

- Subnet in the virtual network
- Azure Event Grid topic
- Private endpoint for the topic

**NOTE**

The steps shown in this section are for topics. You can use similar steps to create private endpoints for domains.

```

subscriptionID=<AZURE SUBSCRIPTION ID>
resourceGroupName=<RESOURCE GROUP NAME>
location=<LOCATION>
vNetName=<VIRTUAL NETWORK NAME>
subNetName=<SUBNET NAME>
topicName = "<TOPIC NAME>"
connectionName=<ENDPOINT CONNECTION NAME>
endpointName=<ENDPOINT NAME>

# resource ID of the topic. replace <SUBSCRIPTION ID>, <RESOURCE GROUP NAME>, and <TOPIC NAME>
topicResourceID="/subscriptions/<SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP
NAME>/providers/Microsoft.EventGrid/topics/<TOPIC NAME>"

# select subscription
az account set --subscription $subscriptionID

# create resource group
az group create --name $resourceGroupName --location $location

# create vnet
az network vnet create \
    --resource-group $resourceGroupName \
    --name $vNetName \
    --address-prefix 10.0.0.0/16

# create subnet
az network vnet subnet create \
    --resource-group $resourceGroupName \
    --vnet-name $vNetName \
    --name $subNetName \
    --address-prefixes 10.0.0.0/24

# disable private endpoint network policies for the subnet
az network vnet subnet update \
    --resource-group $resourceGroupName \
    --vnet-name $vNetName \
    --name $subNetName \
    --disable-private-endpoint-network-policies true

# create event grid topic. update <LOCATION>
az eventgrid topic create \
    --resource-group $resourceGroupName \
    --name $topicName \
    --location $location

# verify that the topic was created.
az eventgrid topic show \
    --resource-group $resourceGroupName \
    --name $topicName

# create private endpoint for the topic you created
az network private-endpoint create
    --resource-group $resourceGroupName \
    --name $endpointName \
    --vnet-name $vNetName \
    --subnet $subNetName \
    --private-connection-resource-id $topicResourceID \
    --connection-name $connectionName \
    --location $location \
    --group-ids topic

# get topic
az eventgrid topic show \
    --resource-group $resourceGroupName \
    --name $topicName

```

## Approve a private endpoint

The following sample CLI snippet shows you how to approve a private endpoint connection.

```
az eventgrid topic private-endpoint-connection approve \
--resource-group $resourceGroupName \
--topic-name $topicName \
--name $endpointName \
--description "connection approved"
```

## Reject a private endpoint

The following sample CLI snippet shows you how to reject a private endpoint connection.

```
az eventgrid topic private-endpoint-connection reject \
--resource-group $resourceGroupName \
--topic-name $topicName \
--name $endpointName \
--description "Connection rejected"
```

## Disable public network access

By default, public network access is enabled for an Event Grid topic or domain. To allow access via private endpoints only, disable public network access by running the following command:

```
az eventgrid topic update \
--resource-group $resourceGroupName \
--name $topicName \
--public-network-access disabled
```

# Use PowerShell

This section shows you how to create a private endpoint for a topic or domain using PowerShell.

## Prerequisite

Follow instructions from [How to: Use the portal to create an Azure AD application and service principal that can access resources](#) to create an Azure Active Directory application and note down the values for **Directory (tenant) ID**, **Application (Client) ID**, and **Application (client) secret**.

## Prepare token and headers for REST API calls

Run the following prerequisite commands to get an authentication token to use with REST API calls and authorization and other header information.

```
$body = "grant_type=client_credentials&client_id=<CLIENT ID>&client_secret=<CLIENT SECRET>&resource=https://management.core.windows.net"

# get authentication token
$Token = Invoke-RestMethod -Method Post ` 
    -Uri https://login.microsoftonline.com/<TENANT ID>/oauth2/token ` 
    -Body $body ` 
    -ContentType 'application/x-www-form-urlencoded'

# set authorization and content-type headers
$headers = @{}
$headers.Add("Authorization","$(($Token.token_type) " + " " + $($Token.access_token))")
```

## Create a subnet with endpoint network policies disabled

```
# create resource group
New-AzResourceGroup -ResourceGroupName <RESOURCE GROUP NAME> -Location <LOCATION>

# create virtual network
$virtualNetwork = New-AzVirtualNetwork ` 
    -ResourceGroupName <RESOURCE GROUP NAME> ` 
    -Location <LOCATION> ` 
    -Name <VIRTUAL NETWORK NAME> ` 
    -AddressPrefix 10.0.0.0/16

# create subnet with endpoint network policy disabled
$subnetConfig = Add-AzVirtualNetworkSubnetConfig ` 
    -Name example-privateLinksubnet ` 
    -AddressPrefix 10.0.0.0/24 ` 
    -PrivateEndpointNetworkPoliciesFlag "Disabled" ` 
    -VirtualNetwork $virtualNetwork

# update virtual network
$virtualNetwork | Set-AzVirtualNetwork
```

## Create an event grid topic with a private endpoint

### NOTE

The steps shown in this section are for topics. You can use similar steps to create private endpoints for [domains](#).

```

$body = @{"location"=<LOCATION>; "properties"=@{"publicNetworkAccess"="disabled"} } | ConvertTo-Json

# create topic
Invoke-RestMethod -Method 'Put' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
    -Headers $Headers ` 
    -Body $body

# verify that the topic was created
$topic=Invoke-RestMethod -Method 'Get' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
    -Headers $Headers

# create private link service connection
$privateEndpointConnection = New-AzPrivateLinkServiceConnection ` 
    -Name "<PRIVATE LINK SERVICE CONNECTION NAME>" ` 
    -PrivateLinkServiceId $topic.Id ` 
    -GroupId "topic"

# get virtual network info
$virtualNetwork = Get-AzVirtualNetwork ` 
    -ResourceGroupName <RESOURCE GROUP NAME> ` 
    -Name <VIRTUAL NETWORK NAME>

# get subnet info
$subnet = $virtualNetwork | Select -ExpandProperty subnets ` 
    | Where-Object {$_ .Name -eq <SUBNET NAME> }

# now, you are ready to create a private endpoint
$privateEndpoint = New-AzPrivateEndpoint -ResourceGroupName <RESOURCE GROUP NAME> ` 
    -Name <PRIVATE ENDPOINT NAME> ` 
    -Location <LOCATION> ` 
    -Subnet $subnet ` 
    -PrivateLinkServiceConnection $privateEndpointConnection

# verify that the endpoint was created
Invoke-RestMethod -Method 'Get' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>/privateEndpointConnections?api-version=2020-06-01" ` 
    -Headers $Headers ` 
    | ConvertTo-Json -Depth 5

```

When you verify that the endpoint was created, you should see the result similar to the following:

```
{
  "value": [
    {
      "properties": {
        "privateEndpoint": {
          "id": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.Network/privateEndpoints/<PRIVATE ENDPOINT NAME>"
        },
        "groupIds": [
          "topic"
        ],
        "privateLinkServiceConnectionState": {
          "status": "Approved",
          "description": "Auto-approved",
          "actionsRequired": "None"
        },
        "provisioningState": "Succeeded"
      },
      "id": "/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>/privateEndpointConnections/<PRIVATE ENDPOINT NAME>.<GUID>",
      "name": "myConnection",
      "type": "Microsoft.EventGrid/topics/privateEndpointConnections"
    }
  ]
}
```

## Approve a private endpoint connection

The following sample PowerShell snippet shows you how to approve a private endpoint.

### NOTE

The steps shown in this section are for topics. You can use similar steps to approve private endpoints for **domains**.

```
$approvedBody =
@{ "properties"=@{ "privateLinkServiceConnectionState"=@{ "status"="approved"; "description"="connection approved"; "actionsRequired"="none" } } } | ConvertTo-Json

# approve endpoint connection
Invoke-RestMethod -Method 'Put'
  -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>/privateEndpointConnections/<PRIVATE ENDPOINT NAME>.<GUID>?api-version=2020-06-01"
  -Headers $Headers
  -Body $approvedBody

# confirm that the endpoint connection was approved
Invoke-RestMethod -Method 'Get'
  -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>/privateEndpointConnections/<PRIVATE ENDPOINT NAME>.<GUID>?api-version=2020-06-01"
  -Headers $Headers
```

## Reject a private endpoint connection

The following example shows you how to reject a private endpoint using PowerShell. You can get the GUID for the private endpoint from the result of the previous GET command.

## NOTE

The steps shown in this section are for topics. You can use similar steps to reject private endpoints for [domains](#).

```
$rejectedBody =  
@{ "properties"=@{ "privateLinkServiceConnectionState"=@{ "status"="rejected"; "description"="connection  
rejected"; "actionsRequired"="none" }}} | ConvertTo-Json  
  
# reject private endpoint  
Invoke-RestMethod -Method 'Put' `  
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP  
NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>/privateEndpointConnections/<PRIVATE  
ENDPOINT NAME>.<GUID>?api-version=2020-06-01" `  
    -Headers $Headers `  
    -Body $rejectedBody  
  
# confirm that endpoint was rejected  
Invoke-RestMethod -Method 'Get'  
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP  
NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>/privateEndpointConnections/<PRIVATE  
ENDPOINT NAME>.<GUID>?api-version=2020-06-01" `  
    -Headers $Headers
```

You can approve the connection even after it's rejected via API. If you use Azure portal, you can't approve an endpoint that has been rejected.

## Next steps

- To learn about how to configure IP firewall settings, see [Configure IP firewall for Azure Event Grid topics or domains](#).
- To troubleshoot network connectivity issues, see [Troubleshoot network connectivity issues](#)

# Assign a system-managed identity to an Event Grid custom topic or domain

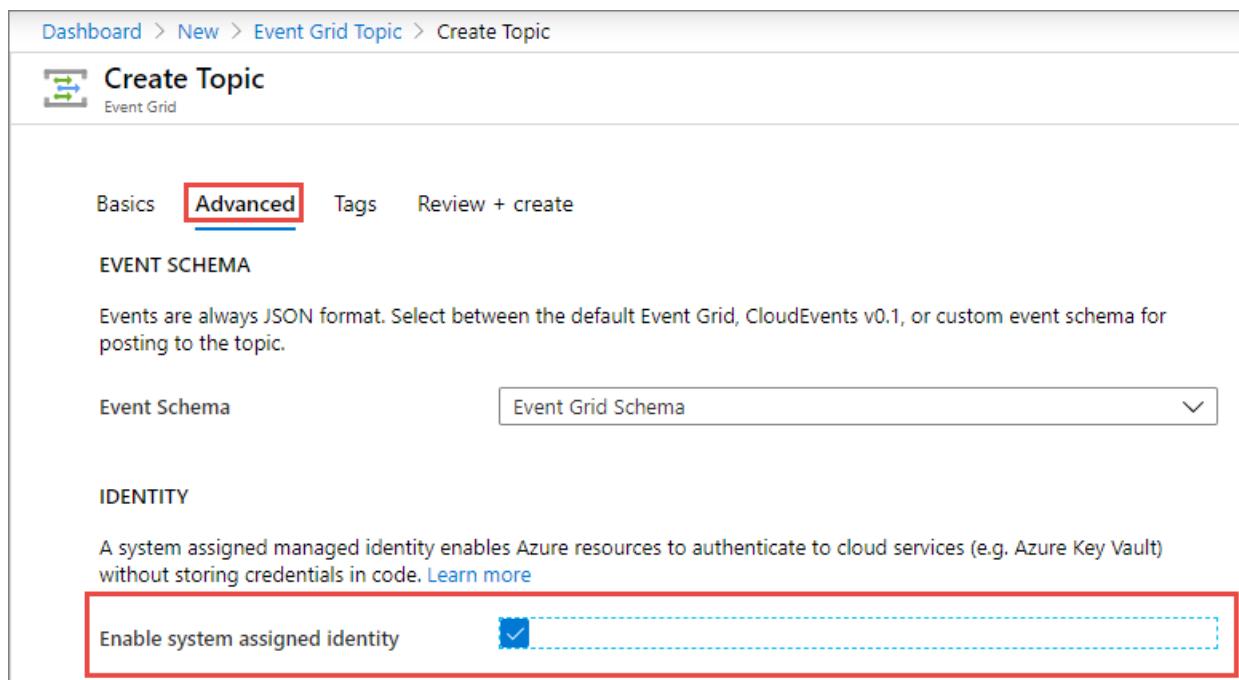
4/2/2021 • 2 minutes to read • [Edit Online](#)

This article shows you how to enable a system-managed identity for an Event Grid custom topic or a domain. To learn about managed identities, see [What are managed identities for Azure resources](#).

## Enable identity at the time of creation

### Using Azure portal

You can enable system-assigned identity for a custom topic or a domain while creating it in the Azure portal. The following image shows how to enable a system-managed identity for a custom topic. Basically, you select the option **Enable system assigned identity** on the **Advanced** page of the topic creation wizard. You'll see this option on the **Advanced** page of the domain creation wizard too.



The screenshot shows the 'Create Topic' wizard in the Azure portal. The 'Advanced' tab is selected. In the 'Event Schema' section, 'Event Schema' is chosen. In the 'Identity' section, the 'Enable system assigned identity' checkbox is checked and highlighted with a red box. A tooltip for 'Learn more' is visible near the checkbox.

### Using Azure CLI

You can also use the Azure CLI to create a custom topic or domain with a system-assigned identity. Use the `az eventgrid topic create` command with the `--identity` parameter set to `systemassigned`. If you don't specify a value for this parameter, the default value `noidentity` is used.

```
# create a custom topic with a system-assigned identity
az eventgrid topic create -g <RESOURCE GROUP NAME> --name <TOPIC NAME> -l <LOCATION> --identity
systemassigned
```

Similarly, you can use the `az eventgrid domain create` command to create a domain with a system-managed identity.

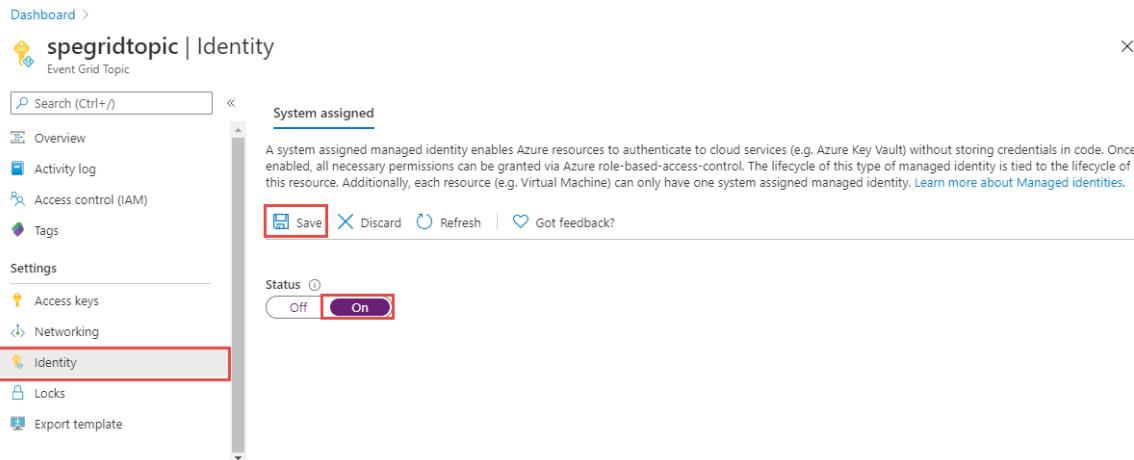
## Enable identity for an existing custom topic or domain

In this section, you learn how to enable a system-managed identity for an existing custom topic or domain.

## Using Azure portal

The following procedure shows you how to enable system-managed identity for a custom topic. The steps for enabling an identity for a domain are similar.

1. Go to the [Azure portal](#).
2. Search for **event grid topics** in the search bar at the top.
3. Select the **custom topic** for which you want to enable the managed identity.
4. Switch to the **Identity** tab.
5. Turn **on** the switch to enable the identity.
6. Select **Save** on the toolbar to save the setting.



You can use similar steps to enable an identity for an event grid domain.

## Use the Azure CLI

Use the `az eventgrid topic update` command with `--identity` set to `systemassigned` to enable system-assigned identity for an existing custom topic. If you want to disable the identity, specify `noidentity` as the value.

```
# Update the topic to assign a system-assigned identity.  
az eventgrid topic update -g $rg --name $topicname --identity systemassigned --sku basic
```

The command for updating an existing domain is similar (`az eventgrid domain update`).

## Next steps

Add the identity to an appropriate role (for example, Service Bus Data Sender) on the destination (for example, a Service Bus queue). For detailed steps, see [Grant managed identity the access to Event Grid destination](#).

# Assign a system-managed identity to an Event Grid system topic

4/2/2021 • 2 minutes to read • [Edit Online](#)

In this article, you learn how to enable system-managed identity for an existing Event Grid system topic. To learn about managed identities, see [What are managed identities for Azure resources](#).

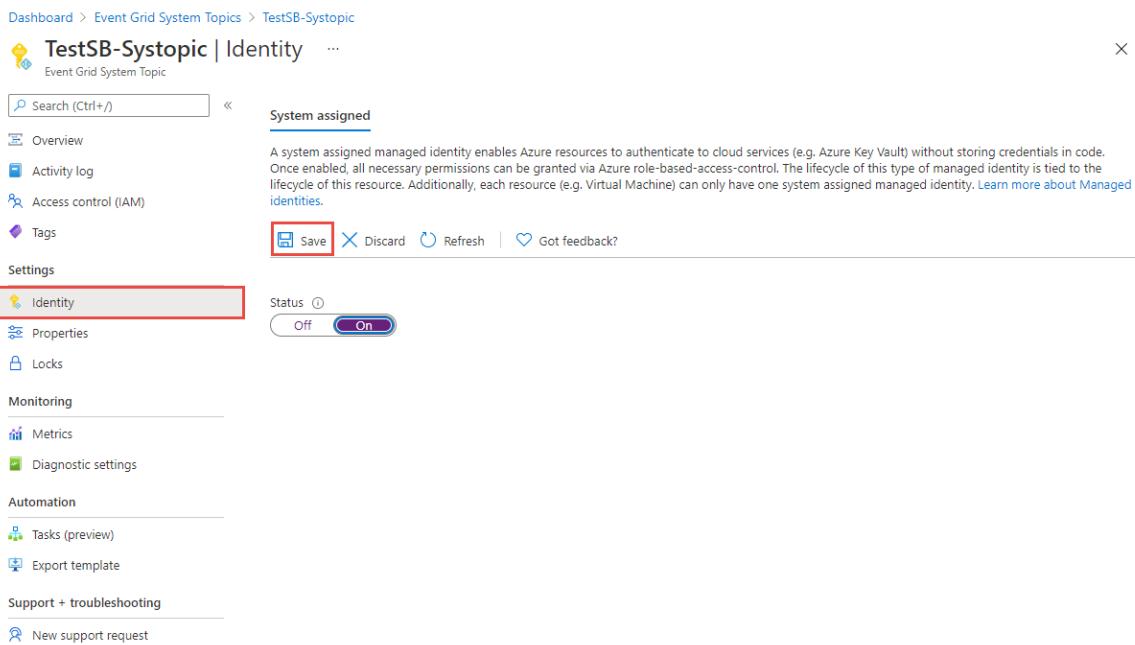
## IMPORTANT

Currently, you can't enable a system-managed identity when creating a new system topic, that is, when creating an event subscription on an Azure resource that supports system topics.

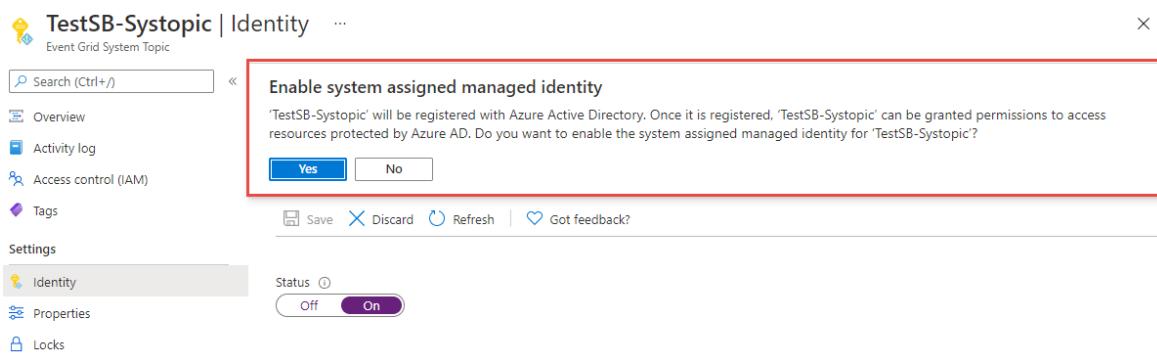
## Use Azure portal

The following procedure shows you how to enable system-managed identity for a system topic.

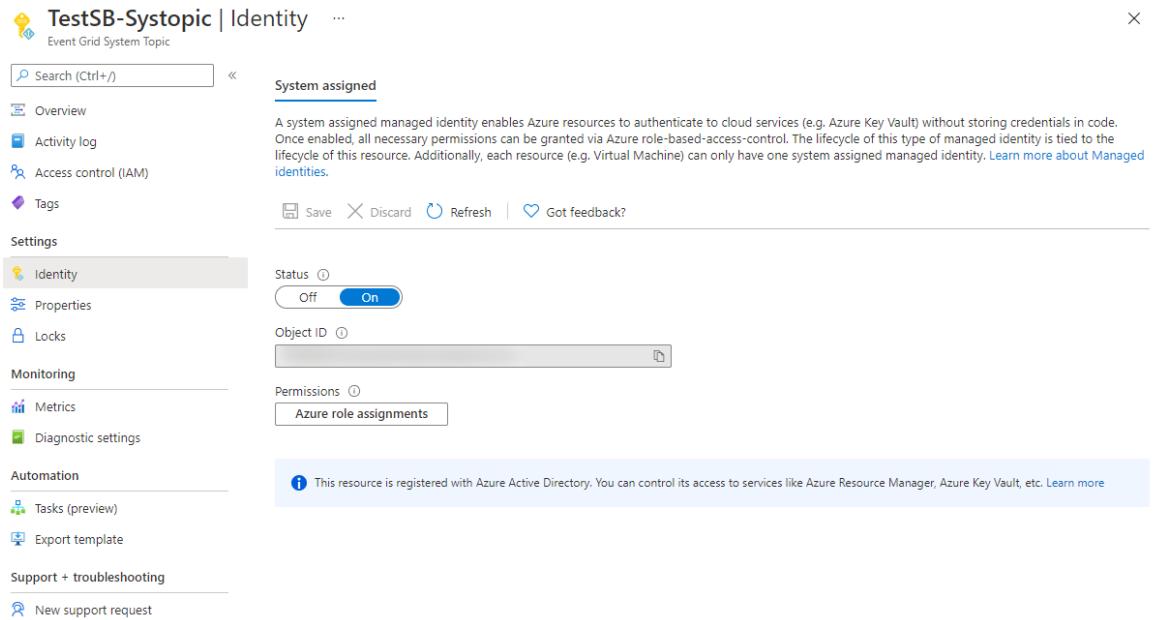
1. Go to the [Azure portal](#).
2. Search for **event grid system topics** in the search bar at the top.
3. Select the **system topic** for which you want to enable the managed identity.
4. Select **Identity** on the left menu. You don't see this option for a system topic that's in the global location.
5. Turn **on** the switch to enable the identity.
6. Select **Save** on the toolbar to save the setting.



7. Select **Yes** on the confirmation message.



8. Confirm that you see the object ID of the system-assigned managed identity and see a link to assign roles.



## Global Azure sources

You can enable system-managed identity only for the regional Azure resources. You can't enable it for system topics associated with global Azure resources such as Azure subscriptions, resource groups, or Azure Maps. The system topics for these global sources are also not associated with a specific region. You don't see the **Identity** page for the system topic whose location is set to **Global**.

The screenshot shows the 'Event Grid System Topic' page for 'PPGTestSub-Systopic'. The 'Location' field is highlighted with a red border and contains the value 'Global'. Other fields shown include 'Resource group (change)' set to 'PPGTest', 'Status' set to 'Active', and 'Event Subscription' and 'Delete' buttons.

## Next steps

Add the identity to an appropriate role (for example, Service Bus Data Sender) on the destination (for example, a Service Bus queue). For detailed steps, see [Grant managed identity the access to Event Grid destination](#).

# Grant managed identity the access to Event Grid destination

4/2/2021 • 4 minutes to read • [Edit Online](#)

This section describes how to add the identity for your system topic, custom topic, or domain to an Azure role.

## Prerequisites

Assign a system-assigned managed identity by using instructions from the following articles:

- [Custom topics or domains](#)
- [System topics](#)

## Supported destinations and Azure roles

After you enable identity for your event grid custom topic or domain, Azure automatically creates an identity in Azure Active Directory. Add this identity to appropriate Azure roles so that the custom topic or domain can forward events to supported destinations. For example, add the identity to the **Azure Event Hubs Data Sender** role for an Azure Event Hubs namespace so that the event grid custom topic can forward events to event hubs in that namespace.

Currently, Azure event grid supports custom topics or domains configured with a system-assigned managed identity to forward events to the following destinations. This table also gives you the roles that the identity should be in so that the custom topic can forward the events.

DESTINATION	AZURE ROLE
Service Bus queues and topics	<a href="#">Azure Service Bus Data Sender</a>
Azure Event Hubs	<a href="#">Azure Event Hubs Data Sender</a>
Azure Blob storage	<a href="#">Storage Blob Data Contributor</a>
Azure Queue storage	<a href="#">Storage Queue Data Message Sender</a>

## Use the Azure portal

You can use the Azure portal to assign the custom topic or domain identity to an appropriate role so that the custom topic or domain can forward events to the destination.

The following example adds a managed identity for an event grid custom topic named **msitesttopic** to the **Azure Service Bus Data Sender** role for a Service Bus namespace that contains a queue or topic resource. When you add to the role at the namespace level, the event grid custom topic can forward events to all entities within the namespace.

1. Go to your **Service Bus namespace** in the [Azure portal](#).
2. Select **Access Control** in the left pane.
3. Select **Add** in the **Add a role assignment** section.
4. On the **Add a role assignment** page, do the following steps:

- a. Select the role. In this case, it's **Azure Service Bus Data Sender**.
- b. Select the **identity** for your event grid custom topic or domain.
- c. Select **Save** to save the configuration.

The steps are similar for adding an identity to other roles mentioned in the table.

## Use the Azure CLI

The example in this section shows you how to use the Azure CLI to add an identity to an Azure role. The sample commands are for event grid custom topics. The commands for event grid domains are similar.

### Get the principal ID for the custom topic's system identity

First, get the principal ID of the custom topic's system-managed identity and assign the identity to appropriate roles.

```
topic_pid=$(az ad sp list --display-name "<TOPIC NAME>" --query [].objectId -o tsv)
```

### Create a role assignment for event hubs at various scopes

The following CLI example shows how to add a custom topic's identity to the **Azure Event Hubs Data Sender** role at the namespace level or at the event hub level. If you create the role assignment at the namespace level, the custom topic can forward events to all event hubs in that namespace. If you create a role assignment at the event hub level, the custom topic can forward events only to that specific event hub.

```
role="Azure Event Hubs Data Sender"
namespaceresourceid=$(az eventhubs namespace show -n <EVENT HUBS NAMESPACE NAME> -g <RESOURCE GROUP of EVENT HUB> --query "{I:id}" -o tsv)
eventhubresourceid=$(az eventhubs eventhub show -n <EVENT HUB NAME> --namespace-name <EVENT HUBS NAMESPACE NAME> -g <RESOURCE GROUP of EVENT HUB> --query "{I:id}" -o tsv)

# create role assignment for the whole namespace
az role assignment create --role "$role" --assignee "$topic_pid" --scope "$namespaceresourceid"

# create role assignment scoped to just one event hub inside the namespace
az role assignment create --role "$role" --assignee "$topic_pid" --scope "$eventhubresourceid"
```

### Create a role assignment for a Service Bus topic at various scopes

The following CLI example shows how to add an event grid custom topic's identity to the **Azure Service Bus Data Sender** role at the namespace level or at the Service Bus topic level. If you create the role assignment at the namespace level, the event grid topic can forward events to all entities (Service Bus queues or topics) within that namespace. If you create a role assignment at the Service Bus queue or topic level, the event grid custom topic can forward events only to that specific Service Bus queue or topic.

```
role="Azure Service Bus Data Sender"
namespaceresourceid=$(az servicebus namespace show -n $RG\SB -g "$RG" --query "{I:id}" -o tsv)
sbustopicresourceid=$(az servicebus topic show -n topic1 --namespace-name $RG\SB -g "$RG" --query "{I:id}" -o tsv)

# create role assignment for the whole namespace
az role assignment create --role "$role" --assignee "$topic_pid" --scope "$namespaceresourceid"

# create role assignment scoped to just one hub inside the namespace
az role assignment create --role "$role" --assignee "$topic_pid" --scope "$sbustopicresourceid"
```

## Next steps

Now that you have assigned a system-assigned identity to your system topic, custom topic, or domain, and added the identity to appropriate roles on destinations, see [Deliver events using the managed identity](#) on delivering events to destinations using the identity.

# Event delivery with a managed identity

3/26/2021 • 5 minutes to read • [Edit Online](#)

This article describes how to use a [managed service identity](#) for an Azure event grid system topic, custom topic, or domain. Use it to forward events to supported destinations such as Service Bus queues and topics, event hubs, and storage accounts.

## Prerequisites

1. Assign a system-assigned identity to a system topic, a custom topic, or a domain.
  - For custom topics and domains, see [Enable managed identity for custom topics and domains](#).
  - For system topics, see [Enable managed identity for system topics](#)
2. Add the identity to an appropriate role (for example, Service Bus Data Sender) on the destination (for example, a Service Bus queue). For detailed steps, see [Add identity to Azure roles on destinations](#)

### NOTE

Currently, it's not possible to deliver events using [private endpoints](#). For more information, see the [Private endpoints](#) section at the end of this article.

## Create event subscriptions that use an identity

After you have an event grid custom topic or system topic or domain with a system-managed identity and have added the identity to the appropriate role on the destination, you're ready to create subscriptions that use the identity.

### Use the Azure portal

When you create an event subscription, you see an option to enable the use of a system-assigned identity for an endpoint in the **ENDPOINT DETAILS** section.

## Create Event Subscription

Event Grid

[Basic](#) [Filters](#) [Additional Features](#)

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

### EVENT SUBSCRIPTION DETAILS

Name

es1



Event Schema

Event Grid Schema



### TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type Event Grid Topic

Topic Resource msiiii

### EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types

Add Event Type

### ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type Service Bus Topic (change)Endpoint [Select an endpoint](#)

Use system assigned identity



You can also enable using a system-assigned identity to be used for dead-lettering on the **Additional Features** tab.

Dashboard > msiiii > Create Event Subscription

## Create Event Subscription

Event Grid

Basic Filters Additional Features Advanced Editor

**DEAD-LETTERING**

Save events that cannot be delivered to storage. [Learn more](#)

Enable dead-lettering

Storage Subscription: SB-DF-Sub1

Storage Blob Container: Select Container

Use system assigned identity for dead-lettering:

**RETRY POLICIES**

Customize how many times and for how long event delivery will be retried. [Learn more](#)

Configure Retry Policies

**EVENT SUBSCRIPTION EXPIRATION TIME**

Set a time at which the event subscription will automatically be deleted.

Enable expiration time

**LABELS**

+ Add Label

## Use the Azure CLI - Service Bus queue

In this section, you learn how to use the Azure CLI to enable the use of a system-assigned identity to deliver events to a Service Bus queue. The identity must be a member of the **Azure Service Bus Data Sender** role. It must also be a member of the **Storage Blob Data Contributor** role on the storage account that's used for dead-lettering.

### Define variables

First, specify values for the following variables to be used in the CLI command.

```
subid=<AZURE SUBSCRIPTION ID>
rg = "<RESOURCE GROUP of EVENT GRID CUSTOM TOPIC>"
topicname = "<EVENT GRID TOPIC NAME>

# get the service bus queue resource id
queueid=$(az servicebus queue show --namespace-name <SERVICE BUS NAMESPACE NAME> --name <QUEUE NAME> --
resource-group <RESOURCE GROUP NAME> --query id --output tsv)
sb_esname = "<Specify a name for the event subscription>"
```

### Create an event subscription by using a managed identity for delivery

This sample command creates an event subscription for an event grid custom topic with an endpoint type set to **Service Bus queue**.

```
az eventgrid event-subscription create
--source-resource-id
/subscriptions/$subid/resourceGroups/$rg/providers/Microsoft.EventGrid/topics/$topicname
--delivery-identity-endpoint-type servicebusqueue
--delivery-identity systemassigned
--delivery-identity-endpoint $queueid
-n $sb_esname
```

### Create an event subscription by using a managed identity for delivery and dead-lettering

This sample command creates an event subscription for an event grid custom topic with an endpoint type set to **Service Bus queue**. It also specifies that the system-managed identity is to be used for dead-lettering.

```

storageid=$(az storage account show --name demoStorage --resource-group gridResourceGroup --query id --
output tsv)
deadletterendpoint="$storageid/blobServices/default/containers/<BLOB CONTAINER NAME>"

az eventgrid event-subscription create
--source-resource-id
/subscriptions/$subid/resourceGroups/$rg/providers/Microsoft.EventGrid/topics/$topicname
--delivery-identity-endpoint-type servicebusqueue
--delivery-identity systemassigned
--delivery-identity-endpoint $queueid
--deadletter-identity-endpoint $deadletterendpoint
--deadletter-identity systemassigned
-n $sb_esnameq

```

## Use the Azure CLI - Event Hubs

In this section, you learn how to use the Azure CLI to enable the use of a system-assigned identity to deliver events to an event hub. The identity must be a member of the **Azure Event Hubs Data Sender** role. It must also be a member of the **Storage Blob Data Contributor** role on the storage account that's used for dead-lettering.

### Define variables

```

subid=<AZURE SUBSCRIPTION ID>
rg = "<RESOURCE GROUP of EVENT GRID CUSTOM TOPIC>"
topicname = "<EVENT GRID CUSTOM TOPIC NAME>

hubid=$(az eventhubs eventhub show --name <EVENT HUB NAME> --namespace-name <NAMESPACE NAME> --resource-
group <RESOURCE GROUP NAME> --query id --output tsv)
eh_esname = "<SPECIFY EVENT SUBSCRIPTION NAME>"
```

### Create an event subscription by using a managed identity for delivery

This sample command creates an event subscription for an event grid custom topic with an endpoint type set to **Event Hubs**.

```

az eventgrid event-subscription create
--source-resource-id
/subscriptions/$subid/resourceGroups/$rg/providers/Microsoft.EventGrid/topics/$topicname
--delivery-identity-endpoint-type eventhub
--delivery-identity systemassigned
--delivery-identity-endpoint $hubid
-n $sbq_esname
```

### Create an event subscription by using a managed identity for delivery + deadletter

This sample command creates an event subscription for an event grid custom topic with an endpoint type set to **Event Hubs**. It also specifies that the system-managed identity is to be used for dead-lettering.

```

storageid=$(az storage account show --name demoStorage --resource-group gridResourceGroup --query id --
output tsv)
deadletterendpoint="$storageid/blobServices/default/containers/<BLOB CONTAINER NAME>"

az eventgrid event-subscription create
--source-resource-id
/subscriptions/$subid/resourceGroups/$rg/providers/Microsoft.EventGrid/topics/$topicname
--delivery-identity-endpoint-type servicebusqueue
--delivery-identity systemassigned
--delivery-identity-endpoint $hubid
--deadletter-identity-endpoint $eh_deadletterendpoint
--deadletter-identity systemassigned
-n $eh_esname
```

## Use the Azure CLI - Azure Storage queue

In this section, you learn how to use the Azure CLI to enable the use of a system-assigned identity to deliver events to an Azure Storage queue. The identity must be a member of the **Storage Queue Data Message Sender** role on the storage account. It must also be a member of the **Storage Blob Data Contributor** role on the storage account that's used for dead-lettering.

### Define variables

```
subid=<AZURE SUBSCRIPTION ID>
rg = "<RESOURCE GROUP of EVENT GRID CUSTOM TOPIC>"
topicname = "<EVENT GRID CUSTOM TOPIC NAME>

# get the storage account resource id
storageid=$(az storage account show --name <STORAGE ACCOUNT NAME> --resource-group <RESOURCE GROUP NAME> --query id --output tsv)

# build the resource id for the queue
queueid="$storageid/queueservices/default/queues/<QUEUE NAME>

sa_esname = "<SPECIFY EVENT SUBSCRIPTION NAME>"
```

### Create an event subscription by using a managed identity for delivery

```
az eventgrid event-subscription create
--source-resource-id
/subscriptions/$subid/resourceGroups/$rg/providers/Microsoft.EventGrid/topics/$topicname
--delivery-identity-endpoint-type storagequeue
--delivery-identity systemassigned
--delivery-identity-endpoint $queueid
-n $sa_esname
```

### Create an event subscription by using a managed identity for delivery + deadletter

```
storageid=$(az storage account show --name demoStorage --resource-group gridResourceGroup --query id --output tsv)
deadletterendpoint="$storageid/blobServices/default/containers/<blob container name>"

az eventgrid event-subscription create
--source-resource-id
/subscriptions/$subid/resourceGroups/$rg/providers/Microsoft.EventGrid/topics/$topicname
--delivery-identity-endpoint-type storagequeue
--delivery-identity systemassigned
--delivery-identity-endpoint $queueid
--deadletter-identity-endpoint $deadletterendpoint
--deadletter-identity systemassigned
-n $sa_esname
```

## Private endpoints

Currently, it's not possible to deliver events using [private endpoints](#). That is, there is no support if you have strict network isolation requirements where your delivered events traffic must not leave the private IP space.

However, if your requirements call for a secure way to send events using an encrypted channel and a known identity of the sender (in this case, Event Grid) using public IP space, you could deliver events to Event Hubs, Service Bus, or Azure Storage service using an Azure event grid custom topic or a domain with system-managed identity configured as shown in this article. Then, you can use a private link configured in Azure Functions or your webhook deployed on your virtual network to pull events. See the sample: [Connect to private endpoints with Azure Functions](#).

Under this configuration, the traffic goes over the public IP/internet from Event Grid to Event Hubs, Service Bus,

or Azure Storage, but the channel can be encrypted and a managed identity of Event Grid is used. If you configure your Azure Functions or webhook deployed to your virtual network to use an Event Hubs, Service Bus, or Azure Storage via private link, that section of the traffic will evidently stay within Azure.

## Next steps

To learn about managed identities, see [What are managed identities for Azure resources](#).

# Deliver events using private link service

4/12/2021 • 2 minutes to read • [Edit Online](#)

Currently, it's not possible to deliver events using [private endpoints](#). That is, there is no support if you have strict network isolation requirements where your delivered events traffic must not leave the private IP space.

## Use managed identity

However, if your requirements call for a secure way to send events using an encrypted channel and a known identity of the sender (in this case, Event Grid) using public IP space, you could deliver events to Event Hubs, Service Bus, or Azure Storage service using an Azure event grid custom topic or a domain with system-managed identity. For details about delivering events using managed identity, see [Event delivery using a managed identity](#).

Then, you can use a private link configured in Azure Functions or your webhook deployed on your virtual network to pull events. See the sample: [Connect to private endpoints with Azure Functions](#).

Under this configuration, the traffic goes over the public IP/internet from Event Grid to Event Hubs, Service Bus, or Azure Storage, but the channel can be encrypted and a managed identity of Event Grid is used. If you configure your Azure Functions or webhook deployed to your virtual network to use an Event Hubs, Service Bus, or Azure Storage via private link, that section of the traffic will evidently stay within Azure.

## Deliver events to Event Hubs using managed identity

To deliver events to event hubs in your Event Hubs namespace using managed identity, follow these steps:

1. Enable system-assigned identity: [system topics, custom topics, and domains](#).
2. [Add the identity to the Azure Event Hubs Data Sender role on the Event Hubs namespace](#).
3. [Enable the Allow trusted Microsoft services to bypass this firewall setting on your Event Hubs namespace](#).
4. [Configure the event subscription](#) that uses an event hub as an endpoint to use the system-assigned identity.

## Deliver events to Service Bus using managed identity

To deliver events to Service Bus queues or topics in your Service Bus namespace using managed identity, follow these steps:

1. Enable system-assigned identity: [system topics, custom topics, and domains](#).
2. [Add the identity to the Azure Service Bus Data Sender role on the Service Bus namespace](#).
3. [Enable the Allow trusted Microsoft services to bypass this firewall setting on your Service Bus namespace](#).
4. [Configure the event subscription](#) that uses a Service Bus queue or topic as an endpoint to use the system-assigned identity.

## Deliver events to Storage

To deliver events to Storage queues using managed identity, follow these steps:

1. Enable system-assigned identity: [system topics, custom topics, and domains](#).

2. [Add the identity to the Storage Queue Data Message Sender role](#) on Azure Storage queue.
3. [Configure the event subscription](#) that uses a Service Bus queue or topic as an endpoint to use the system-assigned identity.

## Next steps

For more information about delivering events using a managed identity, see [Event delivery using a managed identity](#).

# Configure IP firewall for Azure Event Grid topics or domains

3/5/2021 • 6 minutes to read • [Edit Online](#)

By default, topic and domain are accessible from internet as long as the request comes with valid authentication and authorization. With IP firewall, you can restrict it further to only a set of IPv4 addresses or IPv4 address ranges in [CIDR \(Classless Inter-Domain Routing\)](#) notation. Publishers originating from any other IP address will be rejected and will receive a 403 (Forbidden) response. For more information about network security features supported by Event Grid, see [Network security for Event Grid](#).

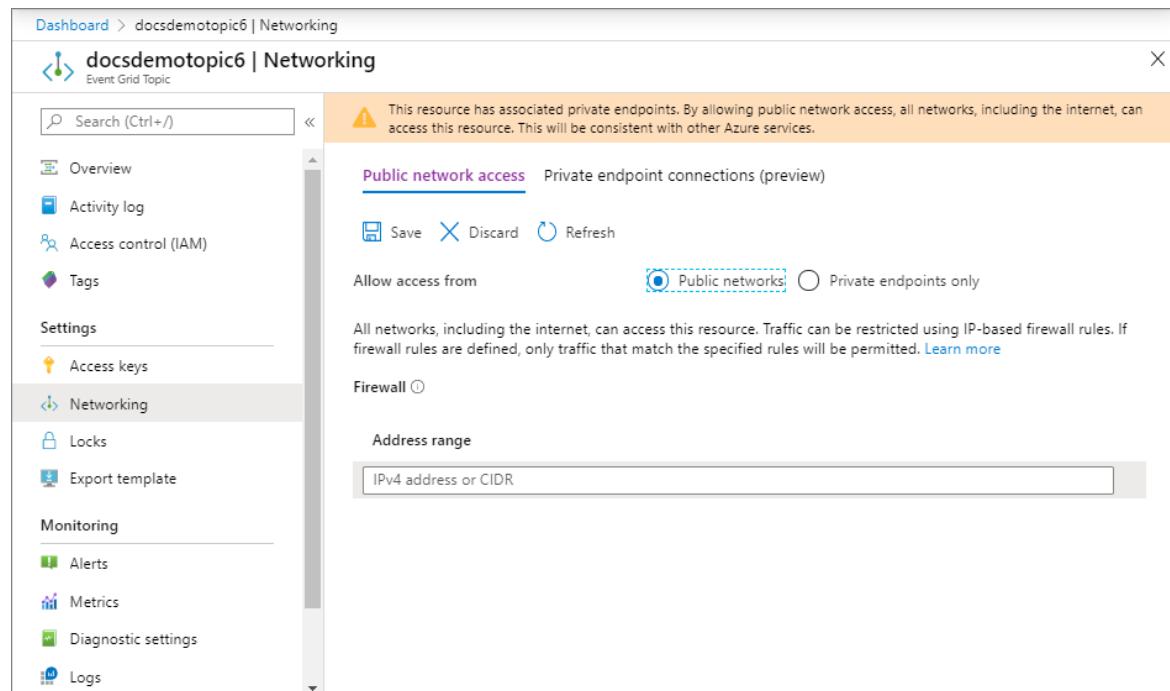
This article describes how to configure IP firewall settings for Azure Event Grid topics or domains.

## Use Azure portal

This section shows you how to use the Azure portal to create inbound IP firewall rules. The steps shown in this section are for topics. You can use similar steps to create inbound IP rules for domains.

1. In the [Azure portal](#), Navigate to your event grid topic or domain, and switch to the **Networking** tab.
2. Select **Public networks** to allow all network, including the internet, to access the resource.

You can restrict the traffic using IP-based firewall rules. Specify a single IPv4 address or a range of IP addresses in Classless inter-domain routing (CIDR) notation.



3. Select **Private endpoints only** to allow only private endpoint connections to access this resource. Use the **Private endpoint connections** tab on this page to manage connections.

The screenshot shows the Azure portal interface for managing an Event Grid Topic. The left sidebar lists various settings like Overview, Activity log, Access control (IAM), Tags, Settings, Access keys, Networking (which is selected), Locks, and Export template. The main content area is titled 'Public network access' and discusses private endpoint connections (preview). It includes buttons for Save, Discard, and Refresh. A note states: 'Only private endpoint connections can access this resource. Use the private endpoint connections tab above to manage connections.' A 'Learn more' link is provided.

4. Select **Save** on the toolbar.

## Use Azure CLI

This section shows you how to use Azure CLI commands to create topics with inbound IP rules. The steps shown in this section are for topics. You can use similar steps to create inbound IP rules for **domains**.

### Enable or disable public network access

By default, the public network access is enabled for topics and domains. You can also enable it explicitly or disable it. You can restrict traffic by configuring inbound IP firewall rules.

#### Enable public network access while creating a topic

```
az eventgrid topic create \
--resource-group $resourceGroupName \
--name $topicName \
--location $location \
--public-network-access enabled
```

#### Disable public network access while creating a topic

```
az eventgrid topic create \
--resource-group $resourceGroupName \
--name $topicName \
--location $location \
--public-network-access disabled
```

#### NOTE

When public network access is disabled for a topic or domain, traffic over public internet isn't allowed. Only private endpoint connections will be allowed to access these resources.

#### Enable public network access for an existing topic

```
az eventgrid topic update \
--resource-group $resourceGroupName \
--name $topicName \
--public-network-access enabled
```

#### Disable public network access for an existing topic

```
az eventgrid topic update \
--resource-group $resourceGroupName \
--name $topicName \
--public-network-access disabled
```

## Create a topic with single inbound ip rule

The following sample CLI command creates an event grid topic with inbound IP rules.

```
az eventgrid topic create \
--resource-group $resourceGroupName \
--name $topicName \
--location $location \
--public-network-access enabled \
--inbound-ip-rules <IP ADDR or CIDR MASK> allow
```

## Create a topic with multiple inbound ip rules

The following sample CLI command creates an event grid topic two inbound IP rules in one step:

```
az eventgrid topic create \
--resource-group $resourceGroupName \
--name $topicName \
--location $location \
--public-network-access enabled \
--inbound-ip-rules <IP ADDR 1 or CIDR MASK 1> allow \
--inbound-ip-rules <IP ADDR 2 or CIDR MASK 2> allow
```

## Update an existing topic to add inbound IP rules

This example creates an event grid topic first and then adds inbound IP rules for the topic in a separate command. It also updates the inbound IP rules that were set in the second command.

```
# create the event grid topic first
az eventgrid topic create \
--resource-group $resourceGroupName \
--name $topicName \
--location $location

# add inbound IP rules to an existing topic
az eventgrid topic update \
--resource-group $resourceGroupName \
--name $topicName \
--public-network-access enabled \
--inbound-ip-rules <IP ADDR or CIDR MASK> allow

# later, update topic with additional ip rules
az eventgrid topic update \
--resource-group $resourceGroupName \
--name $topicName \
--public-network-access enabled \
--inbound-ip-rules <IP ADDR 1 or CIDR MASK 1> allow \
--inbound-ip-rules <IP ADDR 2 or CIDR MASK 2> allow
```

## Remove an inbound IP rule

The following command removes the second rule you created in the previous step by specifying only the first rule while updating the setting.

```
az eventgrid topic update \
--resource-group $resourceGroupName \
--name $topicName \
--public-network-access enabled \
--inbound-ip-rules <IP ADDR 1 or CIDR MASK 1> allow
```

## Use PowerShell

This section shows you how to use Azure PowerShell commands to create Azure Event Grid topics with inbound IP firewall rules. The steps shown in this section are for topics. You can use similar steps to create inbound IP rules for **domains**.

### Prerequisites

Follow instructions from [How to: Use the portal to create an Azure AD application and service principal that can access resources](#) to create an Azure Active Directory application and note down the following values:

- Directory (tenant) ID
- Application (Client) ID
- Application (client) secret

### Prepare token and headers for REST API calls

Run the following prerequisite commands to get an authentication token to use with REST API calls, and authorization and other header information.

```
# replace <CLIENT ID> and <CLIENT SECRET>
$body = "grant_type=client_credentials&client_id=<CLIENT ID>&client_secret=<CLIENT
SECRET>&resource=https://management.core.windows.net"

# get the authentication token. Replace <TENANT ID>
$Token = Invoke-RestMethod -Method Post `

-Uri https://login.microsoftonline.com/<TENANT ID>/oauth2/token `

-Body $body `

-ContentType 'application/x-www-form-urlencoded'

# set authorization and content-type headers
$headers = @{}
$headers.Add("Authorization","$(($Token.token_type) " + " " + $($Token.access_token))")
$headers.Add("Content-Type","application/json")
```

### Enable public network access for an existing topic

By default, the public network access is enabled for topics and domains. You can restrict traffic by configuring inbound IP firewall rules.

```
$body = @{"properties"=@{"publicNetworkAccess"="enabled"} } | ConvertTo-Json -Depth 5

Invoke-RestMethod -Method 'Patch' `

-Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP
NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" `

-Headers $headers `

-Body $body `

| ConvertTo-Json -Depth 5
```

### Disable public network access for an existing topic

When public network access is disabled for a topic or domain, traffic over public internet isn't allowed. Only private endpoint connections will be allowed to access these resources.

```

$body = @{"properties"=@{"publicNetworkAccess"="disabled"}} | ConvertTo-Json -Depth 5

Invoke-RestMethod -Method 'Patch' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
    -Headers $Headers ` 
    -Body $body ` 
    | ConvertTo-Json -Depth 5

```

### Create an event grid topic with inbound rules in one step

```

# prepare the body for the REST PUT method. Notice that inbound IP rules are included.
$body = @{"location"=<LOCATION>; "sku"= @{"name"="basic"}; "properties"=@{"publicNetworkAccess"="enabled";
"inboundIpRules"=@(@{"ipmask"=<IP ADDR or CIDR MASK>";"action"="allow"})}} | ConvertTo-Json -Depth 5

# create the event grid topic with inbound IP rules
Invoke-RestMethod -Method 'Put' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
    -Headers $Headers ` 
    -Body $body

# verify that the topic was created
Invoke-RestMethod -Method 'Get' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
    -Headers $Headers ` 
    | ConvertTo-Json -Depth 5

```

### Create event grid topic first and then add inbound ip rules

```

# prepare the body for the REST PUT method. Notice that no inbound IP rules are specified.
$body = @{"location"=""; "sku"= @{"name"="basic"}; "properties"=@{"publicNetworkAccess"="enabled";}} | ConvertTo-Json -Depth 5

# create the Event Grid topic
Invoke-RestMethod -Method 'Put' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
        -Headers $Headers ` 
        -Body $body` 

# verify that the topic was created
Invoke-RestMethod -Method 'Get' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
        -Headers $Headers ` 
        | ConvertTo-Json -Depth 5

# prepare the body for REST PUT method. Notice that it includes inbound IP rules now. This feature is available in both basic and premium tiers.
$body = @{"location"=""; "sku"= @ {"name"="basic"}; "properties"=@ {"publicNetworkAccess"="enabled"; "inboundIpRules"=@(@ {"ipmask"="";"action"="allow"}, @ {"ipmask"="";"action"="allow"})}} | ConvertTo-Json -Depth 5

# update the topic with inbound IP rules
Invoke-RestMethod -Method 'Put' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
        -Headers $Headers ` 
        -Body $body` 

# verify that the topic was updated
Invoke-RestMethod -Method 'Get' ` 
    -Uri "https://management.azure.com/subscriptions/<AZURE SUBSCRIPTION ID>/resourceGroups/<RESOURCE GROUP NAME>/providers/Microsoft.EventGrid/topics/<EVENT GRID TOPIC NAME>?api-version=2020-06-01" ` 
        -Headers $Headers ` 
        | ConvertTo-Json -Depth 5

```

## Next steps

- For information about monitoring event deliveries, see [Monitor Event Grid message delivery](#).
- For more information about the authentication key, see [Event Grid security and authentication](#).
- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).
- To troubleshoot network connectivity issues, see [Troubleshoot network connectivity issues](#)

# Publish events to Azure Active Directory protected endpoints

4/27/2021 • 5 minutes to read • [Edit Online](#)

This article describes how to use Azure Active Directory (Azure AD) to secure the connection between your **event subscription** and your **webhook endpoint**. For an overview of Azure AD applications and service principals, see [Microsoft identity platform \(v2.0\) overview](#).

This article uses the Azure portal for demonstration, however the feature can also be enabled using CLI, PowerShell, or the SDKs.

## IMPORTANT

Additional access check has been introduced as part of create or update of event subscription on March 30, 2021 to address a security vulnerability. The subscriber client's service principal needs to be either an owner or have a role assigned on the destination application service principal. Please reconfigure your AAD Application following the new instructions below.

## Create an Azure AD Application

Register your Webhook with Azure AD by creating an Azure AD application for your protected endpoint. See [Scenario: Protected web API](#). Configure your protected API to be called by a daemon app.

## Enable Event Grid to use your Azure AD Application

This section shows you how to enable Event Grid to use your Azure AD application.

## NOTE

You must be a member of the [Azure AD Application Administrator role](#) to execute this script.

### Connect to your Azure tenant

First, connect to your Azure tenant using the `connect-AzureAD` command.

```
$myWebhookAadTenantId = "<Your Webhook's Azure AD tenant id>"  
Connect-AzureAD -TenantId $myWebhookAadTenantId
```

### Create Microsoft.EventGrid service principal

Run the following script to create the service principal for **Microsoft.EventGrid** if it doesn't already exist.

```
# This is the "Azure Event Grid" Azure Active Directory (AAD) AppId
$eventGridAppId = "4962773b-9cdb-44cf-a8bf-237846a00ab7"

# Create the "Azure Event Grid" AAD Application service principal if it doesn't exist
$eventGridSP = Get-AzureADServicePrincipal -Filter ("appId eq '" + $eventGridAppId + "')"
if ($eventGridSP -match "Microsoft.EventGrid")
{
    Write-Host "The Service principal is already defined.`n"
} else {
    # Create a service principal for the "Azure Event Grid" AAD Application and add it to the role
    Write-Host "Creating the Azure Event Grid service principal"
    $eventGridSP = New-AzureADServicePrincipal -AppId $eventGridAppId
}
```

### Create a role for your application

Run the following script to create a role for your Azure AD application. In this example, the role name is: **AzureEventGridSecureWebhookSubscriber**. Modify the PowerShell script's `$myTenantId` to use your Azure AD Tenant ID, and `$myAzureADApplicationObjectId` with the Object ID of your Azure AD Application

```

# This is your Webhook's Azure AD Application's ObjectId.
$myWebhookAadApplicationObjectId = "<Your webhook's aad application object id>"

# This is the name of the new role we will add to your Azure AD Application
$eventGridRoleName = "AzureEventGridSecureWebhookSubscriber"

# Create an application role of given name and description
Function CreateAppRole([string] $Name, [string] $Description)
{
    $appRole = New-Object Microsoft.Open.AzureAD.Model.AppRole
    $appRole.AllowedMemberTypes = New-Object System.Collections.Generic.List[string]
    $appRole.AllowedMemberTypes.Add("Application");
    $appRole.AllowedMemberTypes.Add("User");
    $appRole.DisplayName = $Name
    $appRole.Id = New-Guid
    $appRole.IsEnabled = $true
    $appRole.Description = $Description
    $appRole.Value = $Name;
    return $appRole
}

# Get my Azure AD Application, it's roles and service principal
$myApp = Get-AzureADApplication -ObjectId $myWebhookAadApplicationObjectId
$myAppRoles = $myApp.AppRoles

Write-Host "App Roles before addition of new role.."
Write-Host $myAppRoles

# Create the role if it doesn't exist
if ($myAppRoles -match $eventGridRoleName)
{
    Write-Host "The Azure Event Grid role is already defined.\n"
} else {
    # Add our new role to the Azure AD Application
    Write-Host "Creating the Azure Event Grid role in Azure Ad Application: "
    $myWebhookAadApplicationObjectId
    $newRole = CreateAppRole -Name $eventGridRoleName -Description "Azure Event Grid Role"
    $myAppRoles.Add($newRole)
    Set-AzureADApplication -ObjectId $myApp.ObjectId -AppRoles $myAppRoles
}

# print application's roles
Write-Host "My Azure AD Application's Roles: "
Write-Host $myAppRoles

```

## Create role assignment for the client creating event subscription

The role assignment should be created in the Webhook Azure AD App for the AAD app or AAD user creating the event subscription. Use one of the scripts below depending on whether an AAD app or AAD user is creating the event subscription.

### IMPORTANT

Additional access check has been introduced as part of create or update of event subscription on March 30, 2021 to address a security vulnerability. The subscriber client's service principal needs to be either an owner or have a role assigned on the destination application service principal. Please reconfigure your AAD Application following the new instructions below.

### Create role assignment for an event subscription AAD app

```

# This is the app id of the application which will create event subscription. Set to $null if you are not
# assigning the role to app.
$eventSubscriptionWriterAppId = "<the app id of the application which will create event subscription>"

$myServicePrincipal = Get-AzureADServicePrincipal -Filter ("appId eq '" + $myApp.AppId + "'")

$eventSubscriptionWriterSP = Get-AzureADServicePrincipal -Filter ("appId eq '" +
$eventSubscriptionWriterAppId + "'")
if ($eventSubscriptionWriterSP -eq $null)
{
    $eventSubscriptionWriterSP = New-AzureADServicePrincipal -AppId $eventSubscriptionWriterAppId
}

Write-Host "Creating the Azure Ad App Role assignment for application: " $eventSubscriptionWriterAppId
$eventGridAppRole = $myApp.AppRoles | Where-Object -Property "DisplayName" -eq -Value $eventGridRoleName
New-AzureADServiceAppRoleAssignment -Id $eventGridAppRole.Id -ResourceId $myServicePrincipal.ObjectId -
ObjectId $eventSubscriptionWriterSP.ObjectId -PrincipalId $eventSubscriptionWriterSP.ObjectId

```

#### Create role assignment for an event subscription AAD user

```

# This is the user principal name of the user who will create event subscription. Set to $null if you are
# not assigning the role to user.
$eventSubscriptionWriterUserPrincipalName = "<the user principal name of the user who will create event
subscription>"

$myServicePrincipal = Get-AzureADServicePrincipal -Filter ("appId eq '" + $myApp.AppId + "'")

Write-Host "Creating the Azure Ad App Role assignment for user: " $eventSubscriptionWriterUserPrincipalName
$eventSubscriptionWriterUser = Get-AzureADUser -ObjectId $eventSubscriptionWriterUserPrincipalName
$eventGridAppRole = $myApp.AppRoles | Where-Object -Property "DisplayName" -eq -Value $eventGridRoleName
New-AzureADUserAppRoleAssignment -Id $eventGridAppRole.Id -ResourceId $myServicePrincipal.ObjectId -ObjectId
$eventSubscriptionWriterUser.ObjectId -PrincipalId $eventSubscriptionWriterUser.ObjectId

```

#### Create role assignment for Event Grid Service principal

Run the New-AzureADServiceAppRoleAssignment command to assign Event Grid service principal to the role you created in the previous step.

```

$eventGridAppRole = $myApp.AppRoles | Where-Object -Property "DisplayName" -eq -Value $eventGridRoleName
New-AzureADServiceAppRoleAssignment -Id $eventGridAppRole.Id -ResourceId $myServicePrincipal.ObjectId -
ObjectId $eventGridSP.ObjectId -PrincipalId $eventGridSP.ObjectId

```

Run the following commands to output information that you'll use later.

```

Write-Host "My Webhook's Azure AD Tenant Id: $myWebhookAadTenantId"
Write-Host "My Webhook's Azure AD Application Id: $($myApp.AppId)"
Write-Host "My Webhook's Azure AD Application ObjectId Id $($myApp.ObjectId)"

```

## Configure the event subscription

When creating an event subscription, follow these steps:

1. Select the endpoint type as **Web Hook**.
2. Specify the endpoint **URI**.

## Create Event Subscription



Basic    Filters    Additional Features

Event Subscriptions listen for events emitted by the topic resource and send them to the endpoint resource. [Learn more](#)

### EVENT SUBSCRIPTION DETAILS

Name  ✓

Event Schema  ▼

### TOPIC DETAILS

Pick a topic resource for which events should be pushed to your destination. [Learn more](#)

Topic Type  ✓

Topic Resource  ▼

### EVENT TYPES

Pick which event types get pushed to your destination. [Learn more](#)

Filter to Event Types  ▼

### ENDPOINT DETAILS

Pick an event handler to receive your events. [Learn more](#)

Endpoint Type  ✓

Endpoint  ✓

3. Select the **Additional features** tab at the top of the **Create Event Subscriptions** page.

4. On the **Additional features** tab, do these steps:

- Select **Use AAD authentication**, and configure the tenant ID and application ID:
- Copy the Azure AD tenant ID from the output of the script and enter it in the **AAD Tenant ID** field.
- Copy the Azure AD application ID from the output of the script and enter it in the **AAD Application ID or URI** field. Alternatively, you can use the AAD Application ID URI. For more information about application ID URI, see [this article](#).

#### AAD AUTHENTICATION

By default Event Grid uses HTTPS query string parameters for WebHook authentication. If AAD authentication is enabled instead, Event Grid will request tokens at runtime from your AAD Application and use them to authenticate with your endpoints. [Learn more](#)

Use AAD authentication

AAD Tenant ID \*

<your-tenant-id>



AAD Application ID or URI \*

<your-application-id>



## Next steps

- For information about monitoring event deliveries, see [Monitor Event Grid message delivery](#).
- For more information about the authentication key, see [Event Grid security and authentication](#).

- For more information about creating an Azure Event Grid subscription, see [Event Grid subscription schema](#).

# Troubleshoot Azure Event Grid issues

3/20/2021 • 2 minutes to read • [Edit Online](#)

This article provides information that helps you with troubleshooting Azure Event Grid issues.

## Diagnostic logs

Enable diagnostic settings for Event Grid topics or domains to capture and view publish and delivery failure logs. For more information, see [Diagnostic logs](#).

## Metrics

You can view metrics for Event Grid topics and subscriptions, and create alerts on them. For more information, see [Event Grid metrics](#).

## Alerts

Create alerts on Azure Event Grid metrics and activity log operations. For more information, see [Alerts on Event Grid metrics and activity logs](#).

## Subscription validation issues

During event subscription creation, you may receive an error message that says the validation of the provided endpoint failed. For troubleshooting subscription validation issues, see [Troubleshoot Event Grid subscription validations](#).

## Network connectivity issues

There are various reasons for client applications not able to connect to an Event Grid topic/domain. The connectivity issues that you experience may be permanent or transient. To learn how to resolve these issues, see [Troubleshoot connectivity issues](#).

## Error codes

If you receive error messages with error codes like 400, 409, and 403, see [Troubleshoot Event Grid errors](#).

## Distributed tracing (.NET)

The Event Grid .NET library supports distributing tracing. To adhere to the [CloudEvents specification's guidance](#) on distributing tracing, the library sets the `traceparent` and `tracestate` on the `ExtensionAttributes` of a `CloudEvent` when distributed tracing is enabled. To learn more about how to enable distributed tracing in your application, take a look at the Azure SDK [distributed tracing documentation](#).

### Sample

See the [Line Counter sample](#). This sample app illustrates using Storage, Event Hubs, and Event Grid clients along with ASP.NET Core integration, distributed tracing, and hosted services. It allows users to upload a file to a blob, which triggers an Event Hubs event containing the file name. The Event Hubs Processor receives the event, and then the app downloads the blob and counts the number of lines in the file. The app displays a link to a page containing the line count. When the link is clicked, a CloudEvent containing the name of the file is published using Event Grid.

## Next steps

If you need more help, post your issue in the [Stack Overflow forum](#) or open a [support ticket](#).

# Enable Diagnostic logs for Azure event grid topics or domains

4/22/2021 • 3 minutes to read • [Edit Online](#)

This article provides step-by-step instructions to enable diagnostic settings for Event Grid topics or domains. These settings allow you to capture and view publish and delivery failure logs.

## IMPORTANT

For the schema for diagnostic logs, see [Diagnostic logs](#).

## Prerequisites

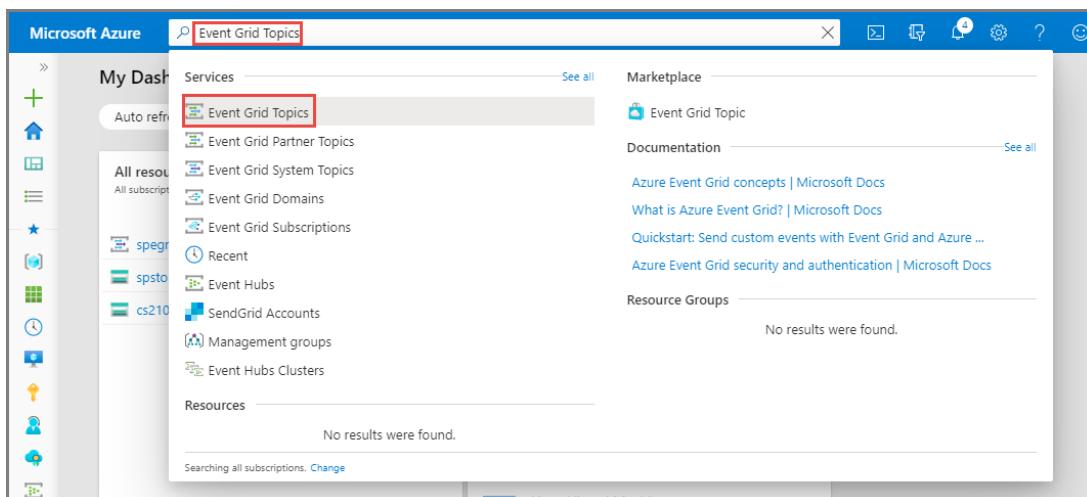
- A provisioned event grid topic
- A provisioned destination for capturing diagnostic logs. It can be one of the following destinations in the same location as the event grid topic:
  - Azure storage account
  - Event hub
  - Log Analytics workspace

## Enable diagnostic logs for a custom topic

### NOTE

The following procedure provides step-by-step instructions for enabling diagnostic logs for a topic. Steps for enabling diagnostic logs for a domain are very similar. In step 2, navigate to the event grid **domain** in the Azure portal.

1. Sign in to the [Azure portal](#).
2. Navigate to the event grid topic for which you want to enable diagnostic log settings.
  - a. In the search bar at the top, search for **Event Grid Topics**.



- a. Select the topic from the list for which you want to configure diagnostic settings.

3. Select **Diagnostic settings** under **Monitoring** in the left menu.

4. On the **Diagnostic settings** page, select **Add New Diagnostic Setting**.

The screenshot shows the 'Diagnostic settings' page for an Event Grid Topic named 'spegridtopic'. The left sidebar has a red box around the 'Diagnostic settings' link under the 'Monitoring' section. The main area shows a table with one row labeled 'No diagnostic settings defined'. Below the table is a button labeled '+ Add diagnostic setting' with a red box around it. To its right, text says 'Click 'Add Diagnostic setting' above to configure the collection of the following data:' followed by a bulleted list: '• DeliveryFailures', '• PublishFailures', and '• AllMetrics'.

5. Specify a **name** for the diagnostic setting.

6. Select the **DeliveryFailures** and **PublishFailures** options in the **Log** section.

The screenshot shows the 'Diagnostics settings' configuration page. The 'Diagnostic settings name' field contains 'spegridtopic-diagnostics'. In the 'Category details' section, under 'log', there are two checked checkboxes: 'DeliveryFailures' and 'PublishFailures', both with red boxes around them. In the 'Destination details' section, there are three unchecked checkboxes: 'Send to Log Analytics', 'Archive to a storage account', and 'Stream to an event hub'.

7. Enable one or more of the capture destinations for the logs, and then configure them by selecting a previous created capture resource.

- If you select **Archive to a storage account**, select **Storage account - Configure**, and then select the storage account in your Azure subscription.

**Diagnostics settings**

Save Discard Delete Provide feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic settings name \* spegridtopic-diagnostics

Category details		Destination details	
<b>log</b>		<input type="checkbox"/> Send to Log Analytics <input checked="" type="checkbox"/> Archive to a storage account	
<input checked="" type="checkbox"/> DeliveryFailures <input checked="" type="checkbox"/> PublishFailures		Retention (days)	0
		Retention (days)	0
<b>metric</b>		<input type="checkbox"/> AllMetrics <small>RetentionPolicy only applies to storage account. Retention policy ranges from 1 to 365 days. If you do not want to apply any retention policy and retain data forever, set retention (days) to 0.</small>	
		Retention (days)	0
<small>Information: You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.</small>			
		<input type="checkbox"/> Stream to an event hub	
		Location	East US
		Subscription	Visual Studio Ultimate with MSDN
		Storage account *	spegriddiagstorage

- If you select **Stream to an event hub**, select **Event hub - Configure**, and then select the Event Hubs namespace, event hub, and the access policy.

**Diagnostics settings**

Save Discard Delete Provide feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic settings name \* spegridtopic-diagnostics

Category details		Destination details	
<b>log</b>		<input type="checkbox"/> Send to Log Analytics <input type="checkbox"/> Archive to a storage account <input checked="" type="checkbox"/> Stream to an event hub	
<input checked="" type="checkbox"/> DeliveryFailures <input checked="" type="checkbox"/> PublishFailures		For potential partner integrations, see documentation <a href="#">here</a>	
<b>metric</b>		<input type="checkbox"/> AllMetrics	
		Subscription	Visual Studio Ultimate with MSDN
		Event hub namespace *	spegriddiagehubns
		Event hub name (optional)	myehub
		Event hub policy name	RootManageSharedAccessKey

- If you select **Send to Log Analytics**, select the Log Analytics workspace.

**Diagnostics settings**

Save Discard Delete Provide feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic settings name \* spegridtopic-diagnostics

Category details		Destination details	
<b>log</b>		<input checked="" type="checkbox"/> Send to Log Analytics	
<input checked="" type="checkbox"/> DeliveryFailures <input checked="" type="checkbox"/> PublishFailures		Subscription	Visual Studio Ultimate with MSDN
<b>metric</b>		<input type="checkbox"/> Archive to a storage account <input type="checkbox"/> Stream to an event hub	
		Log Analytics workspace	spegridlogworkspace ( eastus )
		AllMetrics	

8. Select **Save**. Then, select **X** in the right-corner to close the page.

9. Now, back on the **Diagnostic settings** page, confirm that you see a new entry in the **Diagnostics Settings** table.

The screenshot shows the Azure portal interface for an Event Grid Topic named 'spegridtopic'. The left sidebar has sections for Overview, Activity log, Access control (IAM), Tags, Settings (selected), Monitoring (Alerts, Metrics, Diagnostic settings selected), Logs, Support + troubleshooting, and New support request. The main content area is titled 'Diagnostic settings' and shows a table with one row for 'spegridtopic-diagnostics'. The row includes columns for Name ('spegridtopic-diagnostics'), Storage account ('spegriddiagstorage'), Event hub ('-'), Log Analytics workspace ('-'), and 'Edit setting' (button). Below the table, there's a note about adding diagnostic settings and a list of metrics: DeliveryFailures, PublishFailures, and AllMetrics.

You can also enable collection of all metrics for the topic.

## Enable diagnostic logs for a system topic

1. Sign in to the [Azure portal](#).
2. Navigate to the event grid topic for which you want to enable diagnostic log settings.
  - a. In the search bar at the top, search for **Event Grid System Topics**.

The screenshot shows the Microsoft Azure search results page. The search bar at the top contains the query 'Event Grid System Topics'. The results section is titled 'Services' and lists 'Event Grid System Topics' (which is highlighted with a red box) and 'Event Grid Partner Topics'. Below this, there are sections for 'Marketplace', 'Documentation' (with links to 'Azure Event Grid concepts | Microsoft Docs', 'System topics in Azure Event Grid | Microsoft Docs', 'What is Azure Event Grid? | Microsoft Docs', and 'Quickstart: Send custom events with Event Grid and Azure ...'), and 'Resource Groups' (with a note that 'No results were found'). On the left, there's a sidebar with 'My Dash' and a list of resources including 'spstor' and 'cs210'. At the bottom, there's a note that says 'Searching all subscriptions. Change'.

- b. Select the **system topic** for which you want to configure diagnostic settings.

The screenshot shows the 'Event Grid System Topics' page in the Azure portal. A single topic, 'spegridstoragesystopic1', is listed. The topic is of type 'MICROSOFT.STORAGE.STORAGEACCOUNTS', located in 'East US', associated with the resource group 'spegridrg', and part of the subscription 'Visual Studio Ultimate with MSDN'. The 'Name' column is sorted in ascending order.

3. Select **Diagnostic settings** under **Monitoring** on the left menu, and then select **Add diagnostic setting**.

The screenshot shows the 'Diagnostic settings' blade for the topic 'spegridstoragesystopic1'. The 'Diagnostic settings' section is empty, showing 'No diagnostic settings defined'. A yellow callout '1' points to the 'Diagnostic settings' link in the left sidebar. A yellow callout '2' points to the '+ Add diagnostic setting' button.

4. Specify a name for the diagnostic setting.

5. Select the **DeliveryFailures** in the Log section.

The screenshot shows the 'Diagnostics settings' configuration page. The 'Diagnostic settings name' field is filled with 'spstoragesystopicdiagsettings'. In the 'Category details' section, 'log' is selected, and 'DeliveryFailures' is checked. In the 'Destination details' section, 'Send to Log Analytics' is checked. Other destination options like 'Archive to a storage account' and 'Stream to an event hub' are also listed.

6. Enable one or more of the capture destinations for the logs, and then configure them by selecting a previous created capture resource.

- If you select **Send to Log Analytics**, select the Log Analytics workspace.

Dashboard > Event Grid System Topics > spegridstoragesystopic1 | Diagnostic settings >

## Diagnostics settings

Save  Discard  Delete  Provide feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic settings name *	mysystopicdiagsettings							
Category details								
<table border="1"> <tr> <td>log</td> <td>DeliveryFailures</td> </tr> <tr> <td>metric</td> <td>AllMetrics</td> </tr> </table>		log	DeliveryFailures	metric	AllMetrics			
log	DeliveryFailures							
metric	AllMetrics							
Destination details								
<table border="1"> <tr> <td><input checked="" type="checkbox"/> Send to Log Analytics</td> </tr> <tr> <td>Subscription</td> </tr> <tr> <td>Visual Studio Ultimate with MSDN</td> </tr> <tr> <td>Log Analytics workspace</td> </tr> <tr> <td>spegridlogworkspace (eastus)</td> </tr> <tr> <td><input type="checkbox"/> Archive to a storage account</td> </tr> <tr> <td><input type="checkbox"/> Stream to an event hub</td> </tr> </table>		<input checked="" type="checkbox"/> Send to Log Analytics	Subscription	Visual Studio Ultimate with MSDN	Log Analytics workspace	spegridlogworkspace (eastus)	<input type="checkbox"/> Archive to a storage account	<input type="checkbox"/> Stream to an event hub
<input checked="" type="checkbox"/> Send to Log Analytics								
Subscription								
Visual Studio Ultimate with MSDN								
Log Analytics workspace								
spegridlogworkspace (eastus)								
<input type="checkbox"/> Archive to a storage account								
<input type="checkbox"/> Stream to an event hub								

- If you select **Archive to a storage account**, select **Storage account - Configure**, and then select the storage account in your Azure subscription.

Dashboard > Event Grid System Topics > spegridstoragesystopic1 | Diagnostic settings >

## Diagnostics settings

Save  Discard  Delete  Provide feedback

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic settings name *	mysystopicdiagsettings											
Category details												
<table border="1"> <tr> <td>log</td> <td><input type="checkbox"/> DeliveryFailures</td> <td>Retention (days)</td> <td>0</td> </tr> <tr> <td>metric</td> <td><input type="checkbox"/> AllMetrics</td> <td>Retention (days)</td> <td>0</td> </tr> </table>		log	<input type="checkbox"/> DeliveryFailures	Retention (days)	0	metric	<input type="checkbox"/> AllMetrics	Retention (days)	0			
log	<input type="checkbox"/> DeliveryFailures	Retention (days)	0									
metric	<input type="checkbox"/> AllMetrics	Retention (days)	0									
<p><b>!</b> Retention only applies to storage account. Retention policy ranges from 1 to 365 days. If you do not want to apply any retention policy and retain data forever, set retention (days) to 0.</p>												
Destination details												
<table border="1"> <tr> <td><input type="checkbox"/> Send to Log Analytics</td> </tr> <tr> <td><input checked="" type="checkbox"/> Archive to a storage account</td> </tr> <tr> <td><b>!</b> You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.</td> </tr> <tr> <td><b>!</b> Showing all storage accounts including classic storage accounts</td> </tr> <tr> <td>Location</td> </tr> <tr> <td>East US</td> </tr> <tr> <td>Subscription</td> </tr> <tr> <td>Visual Studio Ultimate with MSDN</td> </tr> <tr> <td>Storage account *</td> </tr> <tr> <td>spegriddiagstorage</td> </tr> <tr> <td><input type="checkbox"/> Stream to an event hub</td> </tr> </table>		<input type="checkbox"/> Send to Log Analytics	<input checked="" type="checkbox"/> Archive to a storage account	<b>!</b> You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.	<b>!</b> Showing all storage accounts including classic storage accounts	Location	East US	Subscription	Visual Studio Ultimate with MSDN	Storage account *	spegriddiagstorage	<input type="checkbox"/> Stream to an event hub
<input type="checkbox"/> Send to Log Analytics												
<input checked="" type="checkbox"/> Archive to a storage account												
<b>!</b> You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.												
<b>!</b> Showing all storage accounts including classic storage accounts												
Location												
East US												
Subscription												
Visual Studio Ultimate with MSDN												
Storage account *												
spegriddiagstorage												
<input type="checkbox"/> Stream to an event hub												

- If you select **Stream to an event hub**, select **Event hub - Configure**, and then select the Event Hubs namespace, event hub, and the access policy.

## Diagnostics settings

[Save](#) [Discard](#) [Delete](#) [Provide feedback](#)

A diagnostic setting specifies a list of categories of platform logs and/or metrics that you want to collect from a resource, and one or more destinations that you would stream them to. Normal usage charges for the destination will occur. [Learn more about the different log categories and contents of those logs](#)

Diagnostic settings name \*

mysystopicdiagsettings ✓

Category details

log

DeliveryFailures

metric

AllMetrics

Destination details

Send to Log Analytics

Archive to a storage account

Stream to an event hub

For potential partner integrations, see documentation [here](#)

Subscription

Visual Studio Ultimate with MSDN

Event hub namespace \*

spegriddiagehubns

Event hub name (optional) ⓘ

logehub

Event hub policy name

RootManageSharedAccessKey

7. Select **Save**. Then, select X in the right-corner to close the page.

8. Now, back on the **Diagnostic settings** page, confirm that you see a new entry in the **Diagnostics Settings** table.

Dashboard > Event Grid System Topics > spegridstoragesystopic1 | Diagnostic settings ✎

Event Grid System Topic

Search (Ctrl+ /) Refresh Provide feedback

Overview Activity log Access control (IAM) Tags

Properties Locks Export template

Monitoring Metrics Diagnostic settings

New support request

Diagnostic settings are used to configure streaming export of platform logs and metrics for a resource to the destination of your choice. You may create up to five different diagnostic settings to send different logs and metrics to independent destinations. [Learn more about diagnostics settings](#)

Name	Storage account	Event hub	Log Analytics workspace	Edit setting
mysystopicdiagsettings	sptstorage0610	spegriddiagehubns/logehub	spegridlogworkspace	<a href="#">Edit setting</a>

+ Add diagnostic setting

Click 'Add Diagnostic setting' above to configure the collection of the following data:

- DeliveryFailures
- AllMetrics

You can also enable collection of all **metrics** for the system topic.

## Diagnostics settings

Diagnostic settings name \* mysystopicmetrics

Category details

log

- DeliveryFailures Retention (days) 0

metric

- AllMetrics Retention (days) 0

Destination details

- Send to Log Analytics
- Archive to a storage account

You'll be charged normal data rates for storage and transactions when you send diagnostics to a storage account.

Showing all storage accounts including classic storage accounts

Location: East US

Subscription: Visual Studio Ultimate with MSDN

Storage account \* spegriddiagstorage

Stream to an event hub

## View diagnostic logs in Azure Storage

- Once you enable a storage account as a capture destination, Event Grid starts emitting diagnostic logs. You should see new containers named **insights-logs-deliveryfailures** and **insights-logs-publishfailures** in the storage account.

Name	Last modified	Public access level	Lease state
insights-logs-deliveryfailures	10/31/2019, 5:18:23 PM	Private	Available
insights-logs-publishfailures	10/31/2019, 5:06:03 PM	Private	Available
Insights-metrics-pt1m	10/31/2019, 4:25:11 PM	Private	Available

- As you navigate through one of the containers, you'll end up at a blob in JSON format. The file contains log entries for either a delivery failure or a publish failure. The navigation path represents the **ResourceId** of the event grid topic and the timestamp (minute level) as to when the log entries were emitted. The blob/JSON file, which is downloadable, in the end adheres to the schema described in the next section.

Name	Modified	Access tier	Blob type	Size	Lease state
PT1H.json	11/1/2019, 3:29:38 PM		Append blob	3.45 KB	Available

3. You should see content in the JSON file similar to the following example:

```
{  
    "time": "2019-11-01T00:17:13.4389048Z",  
    "resourceId": "/SUBSCRIPTIONS/SAMPLE-SUBSCRIPTION-ID /RESOURCEGROUPS/SAMPLE-RESOURCEGROUP-  
NAME/PROVIDERS/MICROSOFT.EVENTGRID/TOPICS/SAMPLE-TOPIC-NAME ",  
    "eventSubscriptionName": "SAMPLEDESTINATION",  
    "category": "DeliveryFailures",  
    "operationName": "Deliver",  
    "message": "Message:outcome=NotFound, latencyInMs=2635, id=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx,  
systemId=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx, state=FilteredFailingDelivery, deliveryTime=11/1/2019  
12:17:10 AM, deliveryCount=0, probationCount=0, deliverySchema=EventGridEvent,  
eventSubscriptionDeliverySchema=EventGridEvent, fields=InputEvent, EventSubscriptionId, DeliveryTime,  
State, Id, DeliverySchema, LastDeliveryAttemptTime, SystemId, fieldCount=, requestExpiration=1/1/0001  
12:00:00 AM, delivered=False publishTime=11/1/2019 12:17:10 AM, eventTime=11/1/2019 12:17:09 AM,  
eventType=Type, deliveryTime=11/1/2019 12:17:10 AM, filteringState=FilteredWithRpc,  
inputSchema=EventGridEvent, publisher=DIAGNOSTICLOGSTEST-EASTUS.EASTUS-1.EVENTGRID.AZURE.NET,  
size=363, fields=Id, PublishTime, SerializedBody, EventType, Topic, Subject, FilteringHashCode,  
SystemId, Publisher, FilteringTopic, TopicCategory, DataVersion, MetadataVersion, InputSchema,  
EventTime, fieldCount=15, url=sb://diagnosticlogstesting-eastus.servicebus.windows.net/,  
deliveryResponse=NotFound: The messaging entity 'sb://diagnosticlogstesting-  
eastus.servicebus.windows.net/eh-diagnosticlogstest' could not be found. TrackingId:c98c5af6-11f0-  
400b-8f56-c605662fb849_G14, SystemTracker:diagnosticlogstesting-eastus.servicebus.windows.net:eh-  
diagnosticlogstest, Timestamp:2019-11-01T00:17:13, referenceId:  
ac141738a9a54451b12b4cc31a10dedc_G14:"  
}
```

## Next steps

For the log schema and other conceptual information about diagnostic logs for topics or domains, see [Diagnostic logs](#).

# Troubleshoot Azure Event Grid errors

4/10/2021 • 3 minutes to read • [Edit Online](#)

This troubleshooting guide provides you the following information:

- Azure Event Grid error codes
- Error messages
- Descriptions for the errors
- Recommended actions that you should take when you receive these errors.

## Error code: 400

ERROR CODE	ERROR MESSAGE	DESCRIPTION	RECOMMENDATION
HttpStatusCode.BadRequest 400	Topic name must be between 3 and 50 characters in length.	The custom topic name length should be between 3 and 50 characters in length. Only alphanumeric letters, digits and the '-' character are allowed in the topic name. Also, the name shouldn't start with the following reserved words: <ul style="list-style-type: none"><li>• Microsoft-</li><li>• EventGrid-</li><li>• System-</li></ul>	Choose a different topic name that adheres to the topic name requirements.
HttpStatusCode.BadRequest 400	Domain name must be between 3 and 50 characters in length.	The domain name length should be between 3 and 50 characters in length. Only alphanumeric letters, digits and the '-' character are allowed in the domain name. Also, the name shouldn't start with the following reserved words: <ul style="list-style-type: none"><li>• Microsoft-</li><li>• EventGrid-</li><li>• System-</li></ul>	Choose a different domain name that adheres to the domain name requirements.
HttpStatusCode.BadRequest 400	Invalid expiration time.	The expiration time for the event subscription determines when the event subscription will retire. This value should be a valid DateTime value in the future.	Make sure the Event Subscription expiration time in a valid DateTime format and it's set to be in the future.

## Error code: 409

Error code	Error message	Description	Recommended action
HttpStatusCode.Conflict 409	Topic with the specified name already exists. Choose a different topic name.	The custom topic name should be unique in a single Azure region to ensure a correct publishing operation. The same name can be used in different Azure regions.	Choose a different name for the topic.
HttpStatusCode.Conflict 409	Domain with the specified already exists. Choose a different domain name.	The domain name should be unique in a single Azure region to ensure a correct publishing operation. The same name can be used in different Azure regions.	Choose a different name for the domain.
HttpStatusCode.Conflict 409	Quota limit reached. For more information on these limits, see <a href="#">Azure Event Grid limits</a> .	Each Azure subscription has a limit on the number of Azure Event Grid resources that it can use. Some or all of this quota had been exceeded and no more resources could be created.	Check your current resources usage and delete any that aren't needed. If you can't delete any resources, create another Azure subscription and create Event Grid resources in that subscription.

## Error code: 403

Error code	Error message	Description	Recommended action
HttpStatusCode.Forbidden 403	Publishing to {Topic/Domain} by client {IpAddress} is rejected because of IpAddress filtering rules.	The topic or domain has IP firewall rules configured and access is restricted only to configured IP addresses.	Add the IP address to the IP firewall rules, see <a href="#">Configure IP firewall</a>
HttpStatusCode.Forbidden 403	Publishing to {Topic/Domain} by client is rejected as request came from Private Endpoint and no matching private endpoint connection found for the resource.	The topic or domain has private endpoints and publish request came from a private endpoint that's not configured or approved.	Configure a private endpoint for the topic/domain. <a href="#">Configure private endpoints</a>

Also, check if your webhook is behind an Azure Application Gateway or Web Application Firewall. If it's, disable the following firewall rules and do an HTTP POST again:

- 920300 (Request missing an accept header)
- 942430 (Restricted SQL character anomaly detection (args): # of special characters exceeded (12))
- 920230 (Multiple URL encoding detected)
- 942130 (SQL injection attack: SQL tautology detected.)
- 931130 (Possible remote file inclusion (RFI) attack = Off-domain reference/link)

## Next steps

If you need more help, post your issue in the [Stack Overflow forum](#) or open a [support ticket](#).

# Troubleshoot connectivity issues - Azure Event Grid

3/5/2021 • 5 minutes to read • [Edit Online](#)

There are various reasons for client applications not able to connect to an Event Grid topic/domain. The connectivity issues that you experience may be permanent or transient. If the issue happens all the time (permanent), you may want to check your organization's firewall settings, IP firewall settings, service tags, private endpoints, and more. For transient issues, running commands to check dropped packets, and obtaining network traces may help with troubleshooting the issues.

This article provides tips for troubleshooting connectivity issues with Azure Event Grid.

## Troubleshoot permanent connectivity issues

If the application isn't able to connect to the event grid at all, follow steps from this section to troubleshoot the issue.

### Check if there's a service outage

Check for the Azure Event Grid service outage on the [Azure service status site](#).

### Check if the ports required to communicate with Event Grid aren't blocked by organization's firewall

Verify that ports used in communicating with Azure Event Grid aren't blocked on your organization's firewall. See the following table for the outbound ports you need to open to communicate with Azure Event Grid.

PROTOCOL	POR
HTTPS	443

Here is a sample command that checks whether the 443 port is blocked.

```
.\psping.exe -n 25 -i 1 -q {sampletopicname}.{region}-{suffix}.eventgrid.azure.net:443 -nobanner
```

On Linux:

```
telnet {sampletopicname}.{region}-{suffix}.eventgrid.azure.net 443
```

### Verify that IP addresses are allowed in your corporate firewall

When you're working with Azure, sometimes you have to allow specific IP address ranges or URLs in your corporate firewall or proxy to access all Azure services you're using or trying to use. Verify that the traffic is allowed on IP addresses used by Event Grid. For IP addresses used by Azure Event Grid: see [Azure IP Ranges and Service Tags - Public Cloud](#) and [Service tag - AzureEventGrid](#).

The [Azure IP Ranges and Service Tags - Public Cloud](#) document also lists IP addresses by region. You can allow address ranges for the topic's region and the paired region in your corporate firewall or proxy. For a paired region for a region, see [Business continuity and disaster recovery \(BCDR\): Azure Paired Regions](#).

#### NOTE

New IP addresses could be added to AzureEventGrid service tag, though it's not usual. So it's good to do a weekly check on the service tags.

## Verify that AzureEventGrid service tag is allowed in your network security groups

If your application is running inside a subnet and if there's an associated network security group, confirm if either internet outbound is allowed or AzureEventGrid service tag is allowed. See [Service Tags](#)

## Check the IP Firewall settings for your Topic/Domain

Check that the public IP address of the machine on which the application is running isn't blocked by the EventGrid topic/domain IP firewall.

By default, Event Grid topics/domains are accessible from internet as long as the request comes with valid authentication and authorization. With IP firewall, you can restrict it further to only a set of IPv4 addresses or IPv4 address ranges in [CIDR \(Classless Inter-Domain Routing\)](#) notation.

The IP firewall rules are applied at the Event Grid topic/domain level. Therefore, the rules apply to all connections from clients using any supported protocol. Any connection attempt from an IP address that doesn't match an allowed IP rule on the Event Grid topic/domain is rejected as forbidden. The response doesn't mention the IP rule.

For more information, see [Configure IP firewall rules for an Azure Event Grid topic/domain](#).

### Find the IP addresses blocked by IP Firewall

Enable diagnostic logs for Event Grid topic/domain [Enable diagnostic logs](#). You'll see the IP address for the connection that's denied.

```
{  
  "time": "2019-11-01T00:17:13.4389048Z",  
  "resourceId": "/SUBSCRIPTIONS/SAMPLE-SUBSCRIPTION-ID/RESOURCEGROUPS/SAMPLE-RESOURCEGROUP-  
NAME/PROVIDERS/MICROSOFT.EVENTGRID/TOPICS/SAMPLE-TOPIC-NAME",  
  "category": "PublishFailures",  
  "operationName": "Post",  
  "message": "inputEventsCount=null, requestUri=https://SAMPLE-TOPIC-NAME.region-  
suffix.eventgrid.azure.net/api/events, publisherInfo=PublisherInfo(category=User,  
inputSchema=EventGridEvent, armResourceId=/SUBSCRIPTIONS/SAMPLE-SUBSCRIPTION-ID/RESOURCEGROUPS/SAMPLE-  
RESOURCEGROUP-NAME/PROVIDERS/MICROSOFT.EVENTGRID/TOPICS/SAMPLE-TOPIC-NAME), httpStatusCode=Forbidden,  
errorType=ClientIPRejected, errorMessage=Publishing to SAMPLE-TOPIC-NAME.{region}-  
{suffix}.EVENTGRID.AZURE.NET by client {clientIp} is rejected due to IpAddress filtering rules."  
}
```

## Check if the EventGrid topic/domain can be accessed using only a private endpoint

If the Event Grid topic/domain is configured to be accessible only via private endpoint, confirm that the client application is accessing the topic/domain over the private endpoint. To confirm it, check if the client application is running inside a subnet and there's a private endpoint for Event Grid topic/domain in that subnet.

[Azure Private Link service](#) enables you to access Azure Event Grid over a **private endpoint** in your virtual network. A private endpoint is a network interface that connects you privately and securely to a service powered by Azure Private Link. The private endpoint uses a private IP address from your VNet, effectively bringing the service into your VNet. All traffic to the service can be routed through the private endpoint, so no gateways, NAT devices, ExpressRoute or VPN connections, or public IP addresses are needed. Traffic between your virtual network and the service traverses over the Microsoft backbone network, eliminating exposure from the public Internet. You can connect to an instance of an Azure resource, giving you the highest level of granularity in access control.

For more information, see [Configure private endpoints](#).

## Troubleshoot transient connectivity issues

If you're experiencing intermittent connectivity issues, go through the following sections for troubleshooting tips.

## Run the command to check dropped packets

When there are intermittent connectivity issues, run the following command to check if there are any dropped packets. This command will try to establish 25 different TCP connections every 1 second with the service. Then, you can check how many of them succeeded/failed and also see TCP connection latency. You can download the `psping` tool from [here](#).

```
.\psping.exe -n 25 -i 1 -q {sampletopicname}.{region}-{suffix}.eventgrid.azure.net:443 -nobanner
```

You can use equivalent commands if you're using other tools such as `tcpping` [tcpping.exe](#).

Obtain a network trace if the previous steps don't help and analyze it using tools such as [Wireshark](#). Contact [Microsoft Support](#) if needed.

## Service upgrades/restarts

Transient connectivity issues may occur because of backend service upgrades and restarts. When they occur, you may see the following symptoms:

- There may be a drop in incoming messages/requests.
- The log file may contain error messages.
- The applications may be disconnected from the service for a few seconds.
- Requests may be momentarily throttled.

Catching these transient errors, backing off and then retrying the call will ensure that your code is resilient to these transient issues.

## Next steps

If you need more help, post your issue in the [Stack Overflow forum](#) or open a [support ticket](#).

# Troubleshoot Azure Event Grid subscription validations

3/5/2021 • 2 minutes to read • [Edit Online](#)

During event subscription creation, if you're seeing an error message such as

The attempt to validate the provided endpoint https://your-endpoint-here failed. For more details, visit <https://aka.ms/esvalidation>

, it indicates that there's a failure in the validation handshake. To resolve this error, verify the following aspects:

- Do an HTTP POST to your webhook url with a [sample SubscriptionValidationEvent](#) request body using Postman or curl or similar tool.
- If your webhook is implementing synchronous validation handshake mechanism, verify that the ValidationCode is returned as part of the response.
- If your webhook is implementing asynchronous validation handshake mechanism, verify that you're the HTTP POST is returning 200 OK.
- If your webhook is returning `403 (Forbidden)` in the response, check if your webhook is behind an Azure Application Gateway or Web Application Firewall. If it is, then you need to disable these firewall rules and do an HTTP POST again:
  - 920300 (Request missing an accept header)
  - 942430 (Restricted SQL character anomaly detection (args): # of special characters exceeded (12))
  - 920230 (Multiple URL encoding detected)
  - 942130 (SQL injection attack: SQL tautology detected.)
  - 931130 (Possible remote file inclusion (RFI) attack = Off-domain reference/link)

## IMPORTANT

For detailed information on endpoint validation for webhooks, see [Webhook event delivery](#).

The following sections show you how to validate an event subscriptions using Postman and Curl.

## Validate Event Grid event subscription using Postman

Here's an example of using Postman for validating a webhook subscription of an Event Grid event:

The screenshot shows the Postman interface with a POST request named "POST - Send Validate Subscription Event". The URL is set to `https://your-webhook-url.com`. The "Body" tab is selected, showing a JSON payload:

```

1 [
2   {
3     "id": "2d1781af-3a4c-4d7c-bd0c-e34b19da4e66",
4     "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
5     "subject": "",
6     "data": {
7       "validationCode": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"
8     },
9     "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
10    "eventTime": "2018-01-25T22:12:19.4556811Z",
11    "metadataVersion": "1",
12    "dataVersion": "1"
13  }
14]

```

The response section shows a status of 200 OK with a response body containing a validation code:

```

1 [
2   {
3     "ValidationResponse": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"
4   }
5 ]

```

Here is a sample SubscriptionValidationEvent JSON:

```
[
  {
    "id": "2d1781af-3a4c-4d7c-bd0c-e34b19da4e66",
    "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx",
    "subject": "",
    "data": {
      "validationCode": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6",
    },
    "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent",
    "eventTime": "2018-01-25T22:12:19.4556811Z",
    "metadataVersion": "1",
    "dataVersion": "1"
  }
]
```

Here is the sample successful response:

```
{
  "validationResponse": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"
}
```

To learn more about Event Grid event validation for webhooks, see [Endpoint validation with event grid events](#).

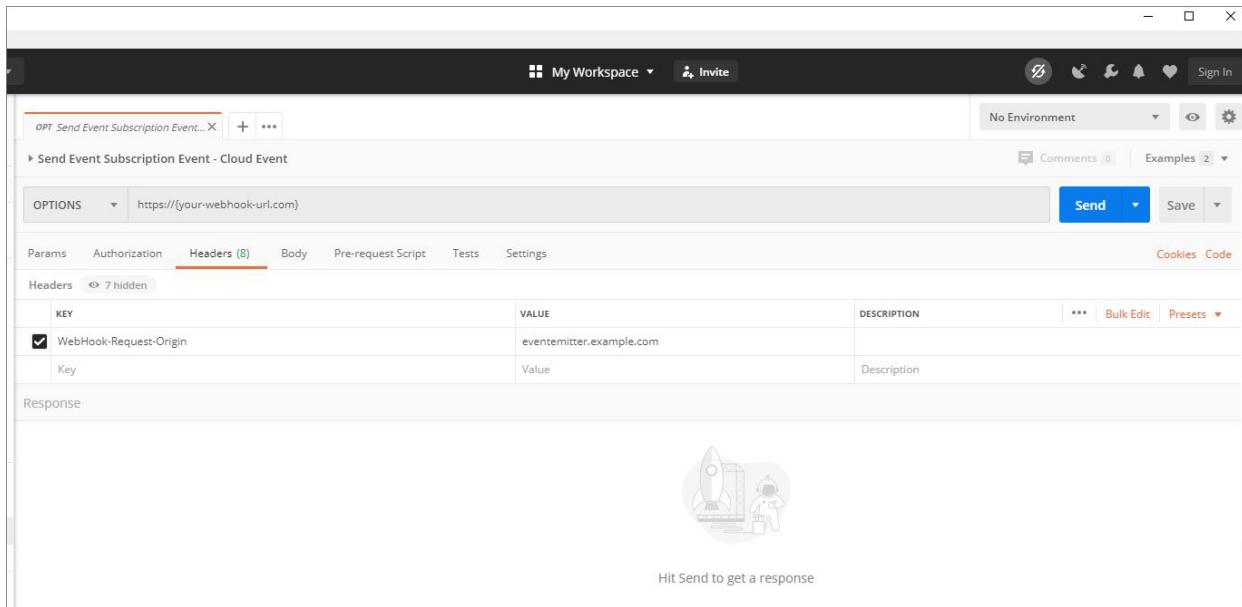
## Validate Event Grid event subscription using Curl

Here's the sample Curl command for validating a webhook subscription of an Event Grid event:

```
curl -X POST -d '[{"id": "2d1781af-3a4c-4d7c-bd0c-e34b19da4e66", "topic": "/subscriptions/xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx", "subject": "", "data": {"validationCode": "512d38b6-c7b8-40c8-89fe-f46f9e9622b6"}, "eventType": "Microsoft.EventGrid.SubscriptionValidationEvent", "eventTime": "2018-01-25T22:12:19.4556811Z", "metadataVersion": "1", "dataVersion": "1"}]' -H 'Content-Type: application/json' https://your-webhook-url.com
```

# Validate cloud event subscription using Postman

Here's an example of using Postman for validating a webhook subscription of a cloud event:



The screenshot shows the Postman application interface. At the top, there's a header bar with 'My Workspace' and 'Invite' buttons. To the right are icons for search, refresh, notifications, and sign-in. Below the header, a tab bar shows 'OPT\_Send Event Subscription Event...' and a 'Send Event Subscription Event - Cloud Event' card. The main workspace shows an 'OPTIONS' request to 'https://(your-webhook-url.com)'. The 'Headers' tab is selected, displaying a table with one row: 'WebHook-Request-Origin' set to 'eventemitter.example.com'. Other tabs like 'Params', 'Authorization', 'Body', 'Pre-request Script', 'Tests', and 'Settings' are visible. A 'Send' button is at the top right of the request area. Below the request, there's a 'Response' section with a placeholder message 'Hit Send to get a response' and a small rocket icon.

Use the **HTTP OPTIONS** method for validation with cloud events. To learn more about cloud event validation for webhooks, see [Endpoint validation with cloud events](#).

## Troubleshoot event subscription validation

### Next steps

If you need more help, post your issue in the [Stack Overflow forum](#) or open a [support ticket](#).

# Monitor Event Grid message delivery

3/18/2021 • 2 minutes to read • [Edit Online](#)

This article describes how to use the portal to see metrics for Event Grid topics and subscriptions, and create alerts on them.

## IMPORTANT

For a list of metrics supported Azure Event Grid, see [Metrics](#).

## View custom topic metrics

If you've published a custom topic, you can view the metrics for it.

1. Sign in to [Azure portal](#).
2. In the search bar at the topic, type **Event Grid Topics**, and then select **Event Grid Topics** from the drop-down list.

3. Select your custom topic from the list of topics.

4. View the metrics for the custom event topic on the **Event Grid Topic** page. In the following image, the **Essentials** section that shows the resource group, subscription etc. is minimized.

Dashboard > Resource groups > contosoegridrg >

### contosocustomtopic

Event Grid Topic

Search (Ctrl+ /) Event Subscription Delete Refresh

Overview Activity log Access control (IAM) Tags

Settings Access keys Networking Identity Locks Export template

Monitoring Alerts Metrics Diagnostic settings Logs

Support + troubleshooting New support request

**Essentials**

Show metrics: General Errors Latency Dead-Letter For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

Published Events (Sum) contosocustomtopic 2 Publish Failed Events contosocustomtopic -- Matched Events (Sum) contosocustomtopic 2 Delivery Failed Events contosocustomtopic -- Dropped Events (Sum) contosocustomtopic --

10:15 PM 10:30 PM 10:45 PM UTC-04:00

Search to filter items... Name Endpoint Prefix Filter Suffix Filter Event Types

contosotopicsubscription WebHook

You can create charts with supported metrics by using the **Metrics** tab of the Event Grid Topic page.

Dashboard > Event Grid Topics >

### contosocustomtopic | Metrics

Event Grid Topic

Search (Ctrl+ /) New chart Refresh Share Feedback Documentation X

Local Time: Last 24 hours (Automatic)

Overview Activity log Access control (IAM) Tags

Settings Access keys Networking Identity Locks Export template

**Metrics**

Monitoring Alerts Diagnostic settings Logs

Support + troubleshooting New support request

Chart Title

Add metric Add filter Apply splitting Line chart Drill into Logs New alert rule Pin to dashboard ...

Scope	Metric Namespace	Metric	Aggregation
contosocustomtopic	Event Grid Topics stand...	Select metric	Select aggregation
		Delivery Failed Events	
		Destination Processing Duration	
		Dropped Events	
		Matched Events	
		Publish Failed Events	
		Published Events	
		Unmatched Events	

Scope: contosocustomtopic Metric Namespace: Event Grid Topics stand... Metric: Select metric Aggregation: Select aggregation

Delivery Failed Events  
Destination Processing Duration  
Dropped Events  
Matched Events  
Publish Failed Events  
Published Events  
Unmatched Events

Filter + Split Apply filters and splits to identify outlying segments Plot multiple metrics Create charts with multiple metrics and resources Build custom dashboards Pin charts to your dashboards

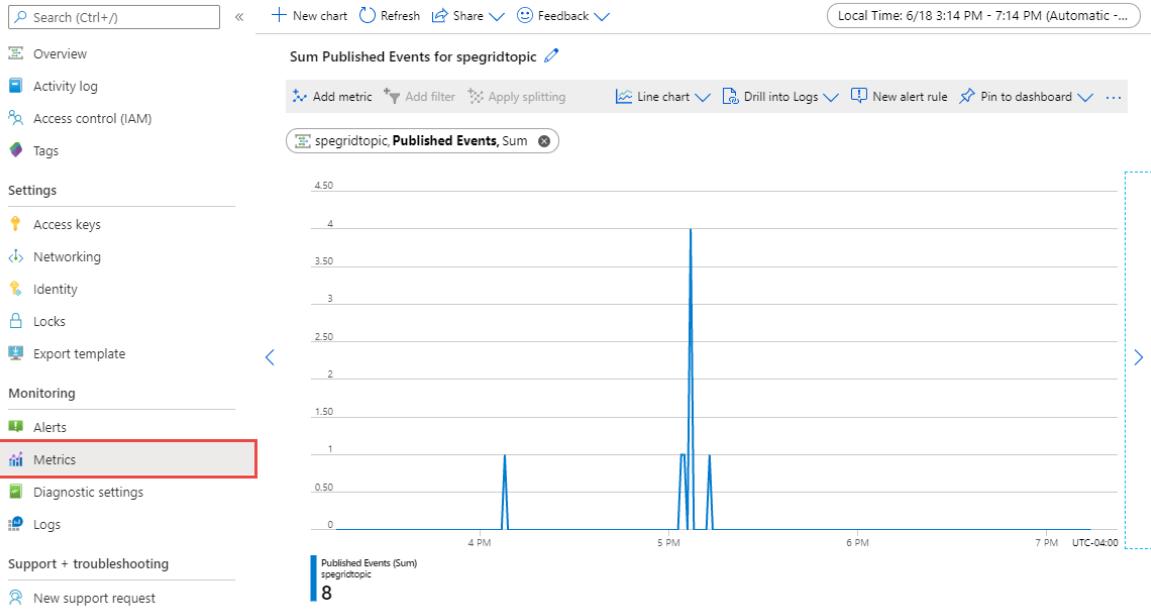
Tue 16 6 AM 12 PM 6 PM UTC-0400

For example, see the metrics chart for the **Published Events** metric.

## spegridr9 | Metrics

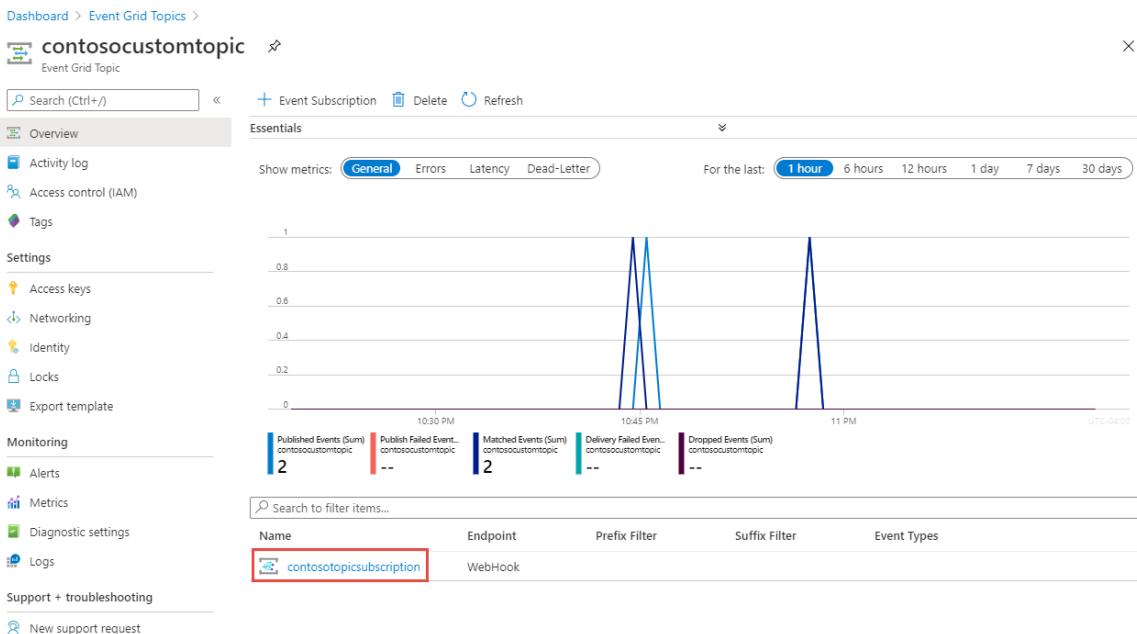
Event Grid Topic

Documentation

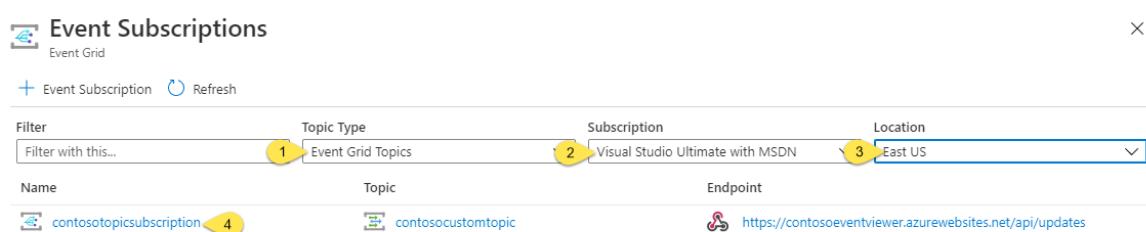


## View subscription metrics

1. Navigate to the **Event Grid Topic** page by following steps from the previous section.
2. Select the subscription from the bottom pane as shown in the following example.

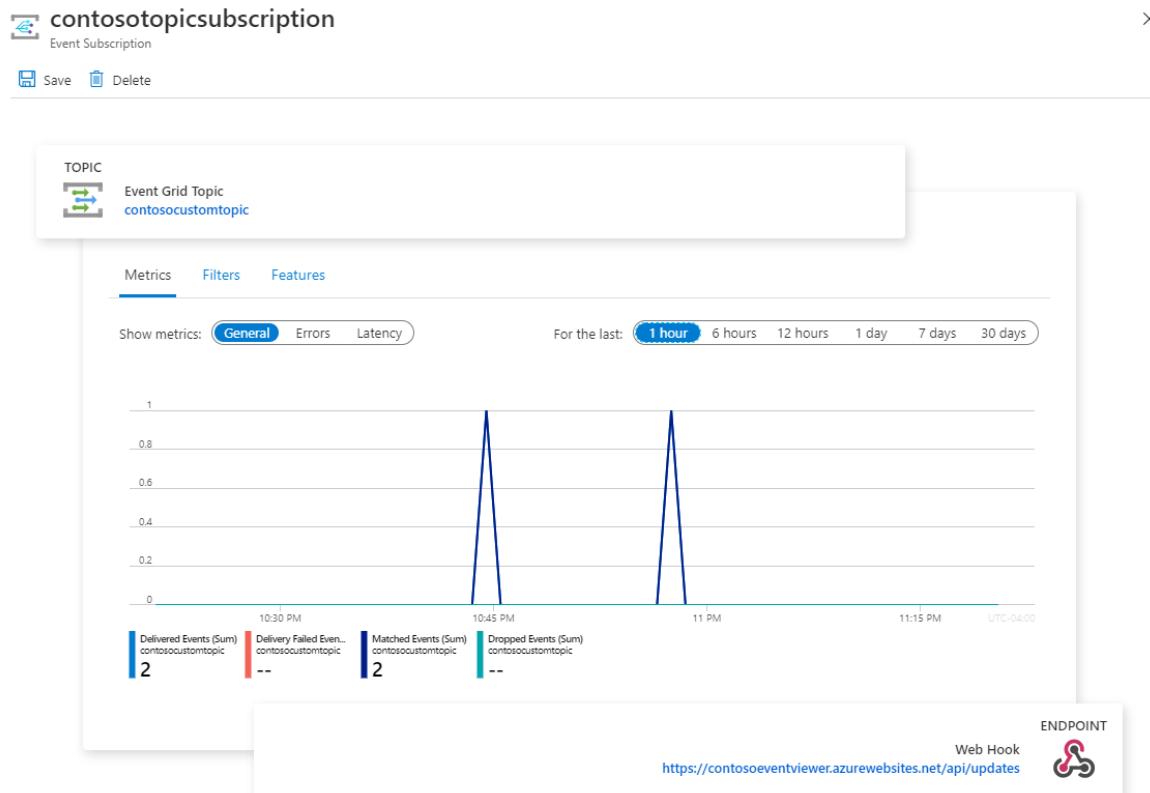


You can also search for **Event Grid Subscriptions** in the search bar in the Azure portal, select **Topic Type**, **Subscription**, and **Location** to see an event subscription.



For custom topics, select **Event Grid Topics** as **Topic Type**. For system topics, select the type of the Azure resource, for example, **Storage Accounts (Blob, GPv2)**.

3. See the metrics for the subscription on the home page for the subscription in a chart. You can see **General**, **Error**, and **Latency** metrics for past 1 hour, 6 hours, 12 hours, 1 day, 7 days, or 30 days.



## View system topic metrics

1. Sign in to [Azure portal](#).
2. In the search bar at the topic, type **Event Grid System Topics**, and then select **Event Grid System Topics** from the drop-down list.

Microsoft Azure

Event Grid System Topics

My Dashboard

Auto refresh : C

All resources All subscriptions

spegridssyst... spcontoso... cs21003bfc... spcontoso... spcontoso...

Event Grid System Topics

Event Grid Partner Topics

Event Grid Topics

Recent

Event Grid Domains

Event Grid Subscriptions

Event Hubs

SendGrid Accounts

Management groups

Event Hubs Clusters

No results were found.

Marketplace

Event Grid System Topic

Searching all subscriptions. Change

App Service Create Web Apps using .NET, Java, Node.js, Python, F...

https://portal.azure.com/?feature.customportal=false#dashboard

3. Select your system topic from the list of topics.

Dashboard >

Event Grid System Topics

Default Directory

Add Manage view Refresh Export to CSV Assign tags Feedback Leave preview

Filter by name... Subscription == all Resource group == all Location == all Add filter

No grouping

Name ↑	Topic Type ↑	Location ↑	Resource group ↑	Subscription ↑
<a href="#">spegridssystopic</a>	MICROSOFT.STORAGE.STORAGEACCOUNTS	East US	contosoegridrg	Visual Studio Ultimate with MSDN

< Previous Page 1 of 1 Next >

4. View the metrics for the system topic on the **Event Grid System Topic** page. In the following image, the **Essentials** section that shows the resource group, subscription etc. is minimized.

Dashboard > Event Grid System Topics >

**spegridssytopic** Event Grid System Topic

Search (Ctrl+ /) Event Subscription Delete Refresh

Overview Activity log Access control (IAM) Tags

Settings Properties Locks Export template

Monitoring Metrics Diagnostic settings

Support + troubleshooting New support request

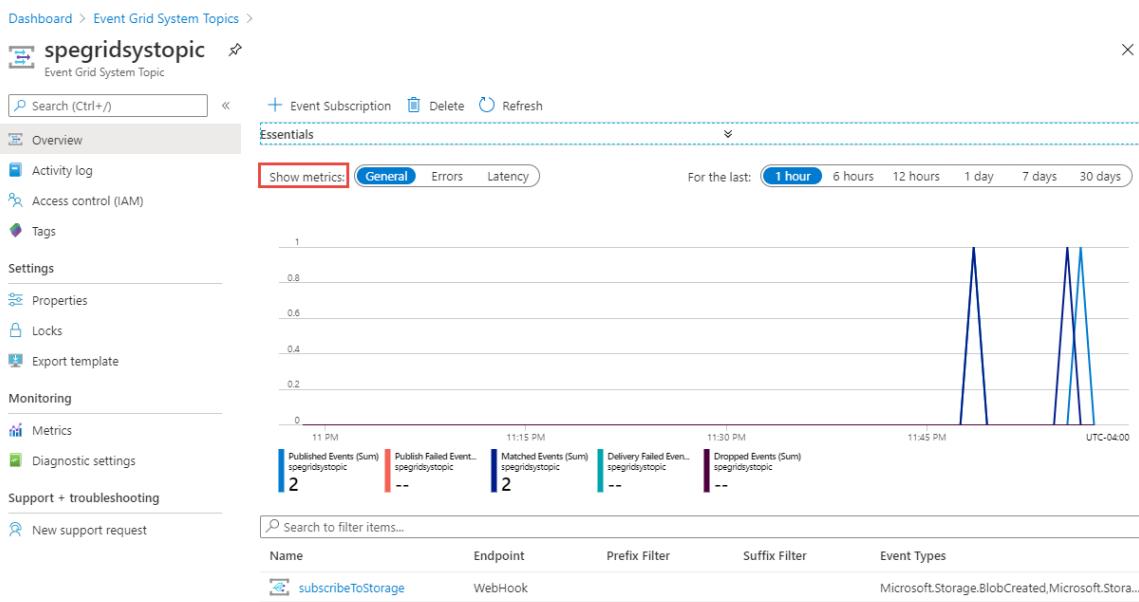
Event Subscription Errors Latency For the last: 1 hour 6 hours 12 hours 1 day 7 days 30 days

Metrics

Published Events (Sum) spegridssytopic 2 Publish Failed Event... spegridssytopic -- Matched Events (Sum) spegridssytopic 2 Delivery Failed Even... spegridssytopic -- Dropped Events (Sum) spegridssytopic --

Search to filter items...

Name	Endpoint	Prefix Filter	Suffix Filter	Event Types
subscribeToStorage	WebHook			Microsoft.Storage.BlobCreated,Microsoft.Stora...



You can create charts with supported metrics by using the **Metrics** tab of the Event Grid Topic page.

Dashboard > Event Grid System Topics >

**spegridssytopic** Event Grid System Topic

Search (Ctrl+ /) New chart Refresh Share Feedback Local Time: Last 24 hours (Automatic)

Overview Activity log Access control (IAM) Tags

Settings Properties Locks Export template

Monitoring Metrics

Diagnostic settings

Support + troubleshooting New support request

Chart Title

Add metric Add filter Apply splitting Line chart Drill into Logs New alert rule Pin to dashboard ...

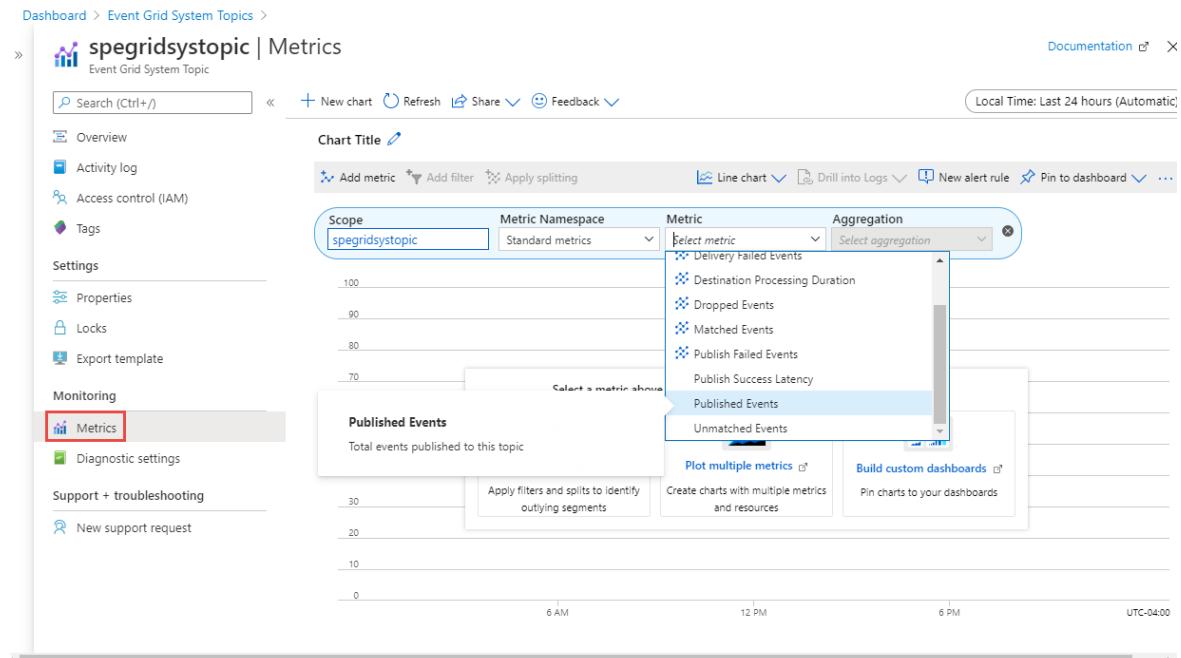
Scope: spegridssytopic Metric Namespace: Standard metrics Metric: Select metric Aggregation: Select aggregation

Metric Namespace: Standard metrics Metric: Published Events Aggregation: Sum

Published Events Total events published to this topic

Unmatched Events

Plot multiple metrics Build custom dashboards



### IMPORTANT

For a list of metrics supported Azure Event Grid, see [Metrics](#).

## Next steps

See the following articles:

- To learn how to create alerts on metrics and activity log operations, see [Set alerts](#).
- For information about event delivery and retries, [Event Grid message delivery and retry](#).

# Set alerts on Azure Event Grid metrics and activity logs

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article describes how to create alerts on Azure Event Grid metrics and activity log operations. You can create alerts on both publish and delivery metrics for Azure Event Grid resources (topics and domains). For system topics, [create alerts using the Metrics page](#).

## Create alerts on dead-lettered events

The following procedure shows you how to create an alert on **dead-lettered events** metric for a custom topic. In this example, an email is sent to the Azure resource group owner when the dead-lettered event count for a topic goes above 10.

1. On the **Event Grid Topic** page for your topic, select **Alerts** on the left menu, and then select **+ New alert rule**.

The screenshot shows the Azure portal interface for managing an Event Grid Topic named "mytopic0130". The left sidebar includes sections for Overview, Activity log, Access control (IAM), Tags, Settings (Access keys, Networking, Identity, Locks, Export template), Monitoring (Alerts, Metrics, Diagnostic settings, Logs), and Support + troubleshooting (New support request). The main content area is titled "Pay attention to what matters." It displays a message stating "You have not configured any alert rules." Below this is a callout: "Configure alert rules and attend to fired alerts to efficiently monitor your Azure resources. [Learn more](#)". To the right is a 3D chart graphic showing a pie chart and several bars. At the bottom right is a prominent red button labeled "+ New Alert Rule". The top navigation bar includes links for Search, Manage alert rules, Manage actions, View classic alerts, Refresh, and Provide feedback. The top left shows the topic name "mytopic0130 | Alerts" and the subscription/resource group information: <Contoso Subscription> / myegridrg / mytopic0130. The time range is set to "Past 24 hours".

The classic alerts can be accessed from [here](#).

2. On the **Create alert rule** page, verify that your topic is selected for the resource. Then, click **Select condition**.

## Create alert rule

Rules management

X

Create an alert rule to identify and address issues when important conditions are found in your monitoring data. [Learn more](#)  
When defining the alert rule, check that your inputs do not contain any sensitive content.

### Scope

Select the target resource you wish to monitor.

Resource	Hierarchy
 mytopic0130	 Contoso Subscription >  myegridrg

[Edit resource](#)

### Condition

Configure when the alert rule should trigger by selecting a signal and defining its logic.

#### Condition name

*No condition selected yet*[Select condition](#)

### Action group

Send notifications or invoke actions when the alert rule triggers, by selecting or creating a new action group. [Learn more](#)

Action group name	Contains actions
<i>No action group selected yet</i>	

[Select action group](#)

3. On the **Configure signal logic** page, follow these steps:

- Select a metric or an activity log entry. In this example, **Dead Lettered Events** is selected.

## Configure signal logic

X

Choose a signal below and configure the logic on the next screen to define the alert condition.

Signal type ⓘ

All

Monitor service ⓘ

All

Displaying 1 - 15 signals out of total 15 signals

Signal name	↑↓ Signal type	↑↓ Monitor service ↑↓
Published Events	↗ Metric	Platform
Publish Failed Events	↗ Metric	Platform
Unmatched Events	↗ Metric	Platform
Publish Success Latency	↗ Metric	Platform
Matched Events	↗ Metric	Platform
Delivery Failed Events	↗ Metric	Platform
Delivered Events	↗ Metric	Platform
Destination Processing Duration	↗ Metric	Platform
Dropped Events	↗ Metric	Platform
Dead Lettered Events	↗ Metric	Platform
All Administrative operations	Activity Log	Administrative
Write Topic (Microsoft.EventGrid/topics)	Activity Log	Administrative
Delete Topic (Microsoft.EventGrid/topics)	Activity Log	Administrative
List Topic Keys (Microsoft.EventGrid/topics)	Activity Log	Administrative
Regenerate Topic Key (Microsoft.EventGrid/topics)	Activity Log	Administrative

Done

- b. Select dimensions (optional).

## Configure signal logic

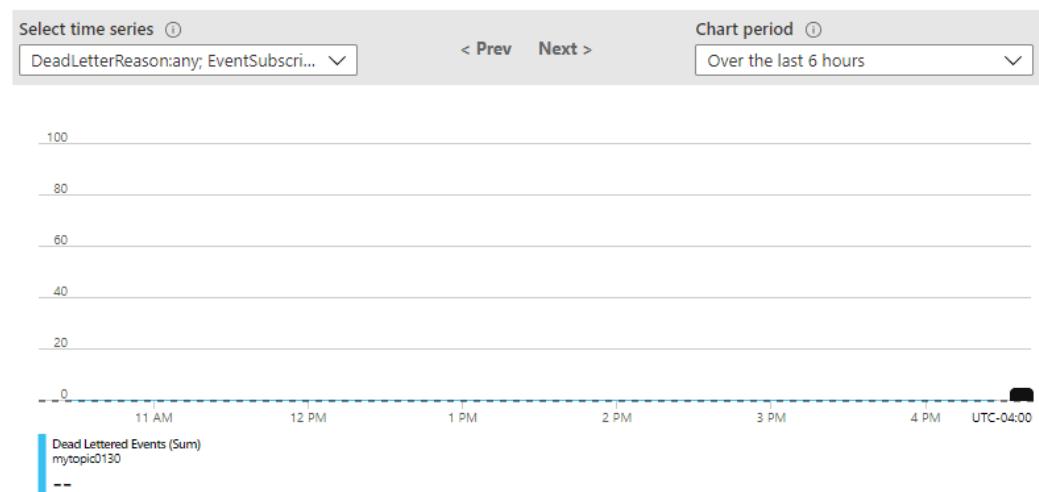
X

Define the logic for triggering an alert. Use the chart to view trends in the data.

[← Back to signal selection](#)

Dead Lettered Events (Platform)

Total dead lettered events matching to this event subscription



This metric supports dimensions. Selecting the dimension values will help you filter to the right time series. If you do not select any value for a dimension, that dimension will be ignored. ⓘ

Dimension name	Dimension values	Select *
DeadLetterReason	0 selected	<input checked="" type="checkbox"/>
EventSubscriptionName	0 selected	<input checked="" type="checkbox"/>

ⓘ Checking "Select \*" will dynamically select all current and future dimension values for the dimension.

**Done**

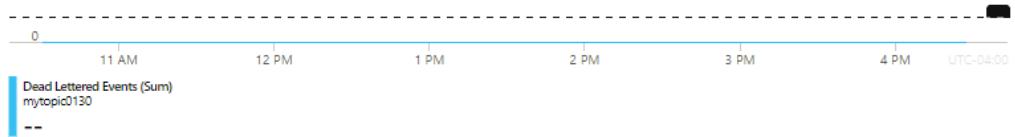
### NOTE

You can select + button for **EventSubscriptionName** to specify an event subscription name to filter events.

- c. Scroll down. In the Alert logic section, select an **Operator**, **Aggregation type**, and enter a **Threshold value**, and select **Done**. In this example, an alert is triggered when the total dead lettered events count is greater than 10.

## Configure signal logic

X



This metric supports dimensions. Selecting the dimension values will help you filter to the right time series. If you do not select any value for a dimension, that dimension will be ignored. ⓘ

Dimension name	Dimension values	Select *
DeadLetterReason	0 selected	<input checked="" type="checkbox"/>
EventSubscriptionName	0 selected	<input checked="" type="checkbox"/>

ⓘ Checking "Select \*" will dynamically select all current and future dimension values for the dimension.

### Alert logic

Threshold ⓘ

Static     Dynamic

Operator ⓘ

Greater than

Aggregation type \* ⓘ

Total

Threshold value \* ⓘ

10

count

### Condition preview

Whenever the total dead lettered events is greater than 10 count

Evaluated based on

Aggregation granularity (Period) \* ⓘ

5 minutes

Frequency of evaluation ⓘ

Every 1 Minute

**Done**

4. Back on the **Create alert rule** page, click **Select action group**.

## Create alert rule

X

Rules management

### Condition

Configure when the alert rule should trigger by selecting a signal and defining its logic.

Condition name	Estimated monthly cost (USD) ⓘ
<input checked="" type="checkbox"/> Whenever the total dead lettered events is greater than 10 count	\$ 0.00

Select condition Total \$ 0.00

ⓘ In an alert rule with multiple conditions, you can only select one value per dimension within each condition.

### Action group

Send notifications or invoke actions when the alert rule triggers, by selecting or creating a new action group. [Learn more](#)

Action group name	Contains actions
No action group selected yet	

[Select action group](#)

### Alert rule details

Provide details on your alert rule so that you can identify and manage it later.

Alert rule name * ⓘ	<input type="text" value="Specify the alert rule name"/>
Description	<input type="text" value="Specify the alert rule description"/>

5. Select **Create action group** on the toolbar to create a new action group. You can also select an existing action group.
6. On the **Add action group** page, follow these steps:
  - a. Enter a **name for the action group**.
  - b. Enter a **short name** for the action group.
  - c. Select an **Azure subscription** in which you want to create the action group.
  - d. Select the **Azure resource group** in which you want to create the action group.
  - e. Enter a **name for the action**.
  - f. Select the **action type**. In this example, **Email Azure Resource Manager Role** is selected, specifically the **Owners** role.
  - g. Select **OK** to close the page.

## Add action group

X

**Action group name \*** ⓘ ✓**Short name \*** ⓘ ✓**Subscription \*** ⓘ ↴**Resource group \*** ⓘ ↴**Actions**

Action name *	Action Type *	Status	Configure	Actions
Send email	Email Azure Resource Manager	<span>On</span>	Edit details	X
Unique name for the action	Select an action type			

[Azure Privacy Statement](#)[Azure Alerts Pricing](#)

 Have a consistent format in emails, notifications and other endpoints irrespective of monitoring source. You can enable per action by editing details. Click on the banner to learn more [↗](#)

**OK**

7. Back on the **Create alert rule** page, enter a name for the alert rule, and then select **Create alert rule**.

## Create alert rule

Rules management

Whenever the total dead lettered events is greater than 10 count

\$ 0.00

Select condition

Total \$ 0.00

i In an alert rule with multiple conditions, you can only select one value per dimension within each condition.

### Action group

Send notifications or invoke actions when the alert rule triggers, by selecting or creating a new action group. [Learn more](#)

Action group name

Contains actions

Email when deadletter count is greater than 10

1 Email Azure Resource Manager Role ⓘ

[Select action group](#)

### Alert rule details

Provide details on your alert rule so that you can identify and manage it later.

Alert rule name \* ⓘ

Alert when deadletter counter goes above 10 ✓

Description

Specify the alert rule description

Severity \* ⓘ

Sev 3

Enable alert rule upon creation



8. Now, on the **Alerts** page of the topic, you see a link to manage alert rules if there are no alerts yet. If there are alerts, select **Manager alert rules** on the toolbar.

Dashboard >

**mytopic0130 | Alerts** Event Grid Topic

[Search \(Ctrl+ /\)](#) [New alert rule](#) [Manage alert rules](#) [Manage actions](#) [View classic alerts](#) [Refresh](#) [Provide feedback](#)

Don't see a subscription? [Open Directory + Subscription settings](#)

Subscription *	Resource group	Resource	Time range
MDP_492270	myegridrg	mytopic0130	Past 24 hours

Selected subscriptions > myegridrg > mytopic0130

**Alerts**

All is good! You have no alerts.

[Manage alert rules \(1\)](#)

The classic alerts can be accessed from [here](#).

## Create alerts on other metrics or activity log operations

The previous section showed you how to create alerts on dead-lettered events. The steps for creating alerts on other metrics or activity log operations are similar.

For example, to create an alert on a delivery failure event, select **Delivery failed events** on the **Configure signal logic** page.

### Configure signal logic

X

Choose a signal below and configure the logic on the next screen to define the alert condition.

Signal type ⓘ

All

Monitor service ⓘ

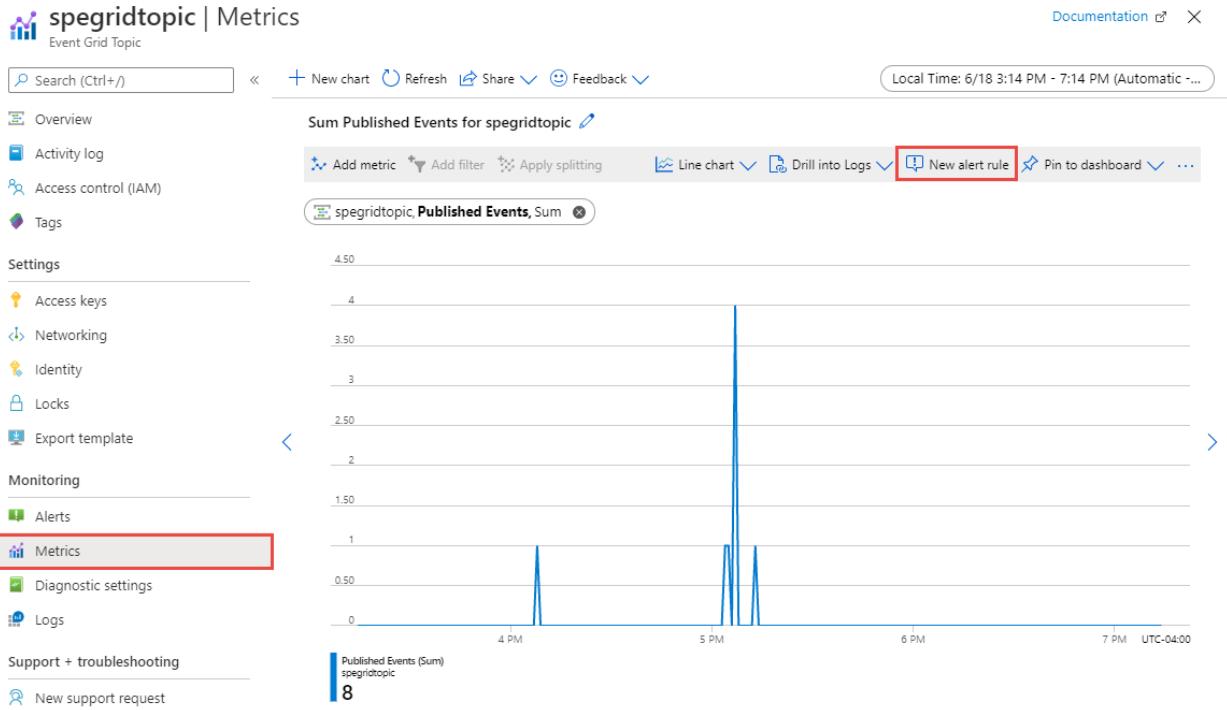
All

Displaying 1 - 15 signals out of total 15 signals

Signal name	Signal type	Monitor service
Published Events	Metric	Platform
Publish Failed Events	Metric	Platform
Unmatched Events	Metric	Platform
Publish Success Latency	Metric	Platform
Matched Events	Metric	Platform
Delivery Failed Events	Metric	Platform
Delivered Events	Metric	Platform
Destination Processing Duration	Metric	Platform
Dropped Events	Metric	Platform
Dead Lettered Events	Metric	Platform
All Administrative operations	Activity Log	Administrative
Write Topic (Microsoft.EventGrid/topics)	Activity Log	Administrative
Delete Topic (Microsoft.EventGrid/topics)	Activity Log	Administrative
List Topic Keys (Microsoft.EventGrid/topics)	Activity Log	Administrative
Regenerate Topic Key (Microsoft.EventGrid/topics)	Activity Log	Administrative

## Create alerts using the Metrics page

You can also create alerts by using the **Metrics** page. The steps are similar. For system topics, you can only use the **Metrics** page to create alerts, as the **Alerts** page isn't available.



#### NOTE

This article doesn't cover all the different steps and combinations that you can use to create an alert. For an overview of alerts, see [Alerts overview](#).

## Next steps

- For information about event delivery and retries, [Event Grid message delivery and retry](#).
- For an introduction to Event Grid, see [About Event Grid](#).
- To quickly get started using Event Grid, see [Create and route custom events with Azure Event Grid](#).

# Azure Event Grid quotas and limits

3/5/2021 • 2 minutes to read • [Edit Online](#)

This article lists quotas and limits in Azure Event Grid.

## Quotas and limits

The following limits apply to Azure Event Grid **topics** (system, custom, and partner topics).

RESOURCE	LIMIT
Custom topics per Azure subscription	100
Event subscriptions per topic	500
Publish rate for a custom or a partner topic (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)
Event size	1 MB
Private endpoint connections per topic	64
IP Firewall rules per topic	16

The following limits apply to Azure Event Grid **domains**.

RESOURCE	LIMIT
Topics per event domain	100,000
Event subscriptions per topic within a domain	500
Domain scope event subscriptions	50
Publish rate for an event domain (ingress)	5,000 events/sec or 5 MB/sec (whichever is met first)
Event Domains per Azure Subscription	100
Private endpoint connections per domain	64
IP Firewall rules per domain	16

## Next steps

See [Azure subscription and service limits, quotas, and constraints](#) for quotas and limits of all Azure services.

# Event Grid subscription schema

3/5/2021 • 2 minutes to read • [Edit Online](#)

To create an Event Grid subscription, you send a request to the Create Event subscription operation. Use the following format:

```
PUT /subscriptions/{subscription-id}/resourceGroups/{group-name}/providers/{resource-provider}/{resource-type}/{resource-name}/Microsoft.EventGrid/eventSubscriptions/{event-type-definitions}?api-version=2018-01-01
```

For example, to create an event subscription for a storage account named `examplestorage` in a resource group named `examplegroup`, use the following format:

```
PUT /subscriptions/{subscription-id}/resourceGroups/examplegroup/providers/Microsoft.Storage/storageaccounts/examplestorage/Microsoft.EventGrid/eventSubscriptions/{event-type-definitions}?api-version=2018-01-01
```

The Event Subscription name must be 3-64 characters in length and can only contain a-z, A-Z, 0-9, and "-". The article describes the properties and schema for the body of the request.

## Event subscription properties

PROPERTY	TYPE	DESCRIPTION
destination	object	The object that defines the endpoint.
filter	object	An optional field for filtering the types of events.

### destination object

PROPERTY	TYPE	DESCRIPTION
endpointType	string	The type of endpoint for the subscription (webhook/HTTP, Event Hub, or queue).
endpointUrl	string	The destination URL for events in this event subscription.

### filter object

PROPERTY	TYPE	DESCRIPTION
includedEventTypes	array	Match when the event type in the event message is an exact match to one of these event type names. Raises an error when event name does not match the registered event type names for the event source. Default matches all event types.

PROPERTY	TYPE	DESCRIPTION
subjectBeginsWith	string	A prefix-match filter to the subject field in the event message. The default or empty string matches all.
subjectEndsWith	string	A suffix-match filter to the subject field in the event message. The default or empty string matches all.
isSubjectCaseSensitive	string	Controls case-sensitive matching for filters.
enableAdvancedFilteringOnArrays	boolean	Enables using arrays for keys in advanced filtering. For more information, see <a href="#">Advanced filtering</a> .

## Example subscription schema

```
{
  "properties": {
    "destination": {
      "endpointType": "webhook",
      "properties": {
        "endpointUrl": "https://example.azurewebsites.net/api/HttpTriggerCSharp1?
code=VXbGWce53148Mt8wuotr0GPmyJ/nDT4hgdFj9DpBiRt38qqnnm5OFg=="
      }
    },
    "filter": {
      "includedEventTypes": [ "Microsoft.Storage.BlobCreated", "Microsoft.Storage.BlobDeleted" ],
      "subjectBeginsWith": "/blobServices/default/containers/mycontainer/log",
      "subjectEndsWith": ".jpg",
      "isSubjectCaseSensitive": "true"
    }
  }
}
```

## Next steps

- For an introduction to Event Grid, see [What is Event Grid?](#)

# Event Grid SDKs for management and publishing

5/18/2021 • 2 minutes to read • [Edit Online](#)

Event Grid provides SDKs that enable you to programmatically manage your resources and post events.

## Management SDKs

The management SDKs enable you to create, update, and delete event grid topics and subscriptions. Currently, the following SDKs are available:

- [.NET](#)
- [Go](#)
- [Java](#)
- [Node](#)
- [Python](#)
- [Ruby](#)

## Data plane SDKs

The data plane SDKs enable you to post events to topics by taking care of authenticating, forming the event, and asynchronously posting to the specified endpoint. They also enable you to consume first party events. Currently, the following SDKs are available:

PROGRAMMING LANGUAGE	SDK
.NET	Latest stable SDK: <a href="#">Azure.Messaging.EventGrid</a> Legacy SDK: <a href="#">Microsoft.Azure.EventGrid</a>
Java	Latest stable SDK: <a href="#">azure-messaging-eventgrid</a> Legacy SDK: <a href="#">azure-eventgrid</a>
Python	<a href="#">azure-eventgrid</a>
JavaScript	<a href="#">@azure/eventgrid</a>
Go	<a href="#">Azure SDK for Go</a>
Ruby	<a href="#">azure_event_grid</a>

## Next steps

- For example applications, see [Event Grid code samples](#).
- For an introduction to Event Grid, see [What is Event Grid?](#)
- For Event Grid commands in Azure CLI, see [Azure CLI](#).
- For Event Grid commands in PowerShell, see [PowerShell](#).

# Azure Policy built-in definitions for Azure Event Grid

5/14/2021 • 2 minutes to read • [Edit Online](#)

This page is an index of [Azure Policy](#) built-in policy definitions for Azure Event Grid. For additional Azure Policy built-ins for other services, see [Azure Policy built-in definitions](#).

The name of each built-in policy definition links to the policy definition in the Azure portal. Use the link in the **Version** column to view the source on the [Azure Policy GitHub repo](#).

## Azure Event Grid

NAME (AZURE PORTAL)	DESCRIPTION	EFFECT(S)	VERSION (GITHUB)
<a href="#">Azure Event Grid domains should disable public network access</a>	Disabling public network access improves security by ensuring that the resource isn't exposed on the public internet. You can limit exposure of your resources by creating private endpoints instead. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a> .	Audit, Deny, Disabled	1.0.0
<a href="#">Azure Event Grid domains should use private link</a>	Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to your Event Grid domain instead of the entire service, you'll also be protected against data leakage risks. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a> .	Audit, Disabled	1.0.2
<a href="#">Azure Event Grid topics should disable public network access</a>	Disabling public network access improves security by ensuring that the resource isn't exposed on the public internet. You can limit exposure of your resources by creating private endpoints instead. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a> .	Audit, Deny, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Azure Event Grid topics should use private link</a>	<p>Azure Private Link lets you connect your virtual network to Azure services without a public IP address at the source or destination. The Private Link platform handles the connectivity between the consumer and services over the Azure backbone network. By mapping private endpoints to your Event Grid topic instead of the entire service, you'll also be protected against data leakage risks. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a>.</p>	Audit, Disabled	1.0.2
<a href="#">Deploy - Configure Azure Event Grid domains with private endpoints</a>	<p>Private endpoints lets you connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to your resources, they'll be protected against data leakage risks. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a>.</p>	DeployIfNotExists, Disabled	1.0.0
<a href="#">Deploy - Configure Azure Event Grid topics with private endpoints</a>	<p>Private endpoints lets you connect your virtual network to Azure services without a public IP address at the source or destination. By mapping private endpoints to your resources, they'll be protected against data leakage risks. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a>.</p>	DeployIfNotExists, Disabled	1.0.0

NAME	DESCRIPTION	EFFECT(S)	VERSION
<a href="#">Modify - Configure Azure Event Grid domains to disable public network access</a>	Disable public network access for Azure Event Grid resource so that it isn't accessible over the public internet. This will help protect them against data leakage risks. You can limit exposure of the your resources by creating private endpoints instead. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a> .	Modify, Disabled	1.0.0
<a href="#">Modify - Configure Azure Event Grid topics to disable public network access</a>	Disable public network access for Azure Event Grid resource so that it isn't accessible over the public internet. This will help protect them against data leakage risks. You can limit exposure of the your resources by creating private endpoints instead. Learn more at: <a href="https://aka.ms/privateendpoints">https://aka.ms/privateendpoints</a> .	Modify, Disabled	1.0.0

## Next steps

- See the built-ins on the [Azure Policy GitHub repo](#).
- Review the [Azure Policy definition structure](#).
- Review [Understanding policy effects](#).