Azure Event Grid overview

Agenda

01 Azure Event Grid view

02 Concepts in Azure Event Grid

03 Azure Event Grid Schema

04 Event Grid Security and Authentication

05 Types of Authentication

06 Management Access Control

07 Event Filtering for Event Grid Schemas

08 Types of Filtering

Agenda

09 Capture Events through Event Hubs

10 Azure Event Hub from Apache Kafka Applications

11 Event Hub Dedicated overview

12 Analyzing and Troubleshooting Apps

13 Types of Availability Sets

14 Application Map

15 Transient Fault

16 Azure Cache for Redis Overview

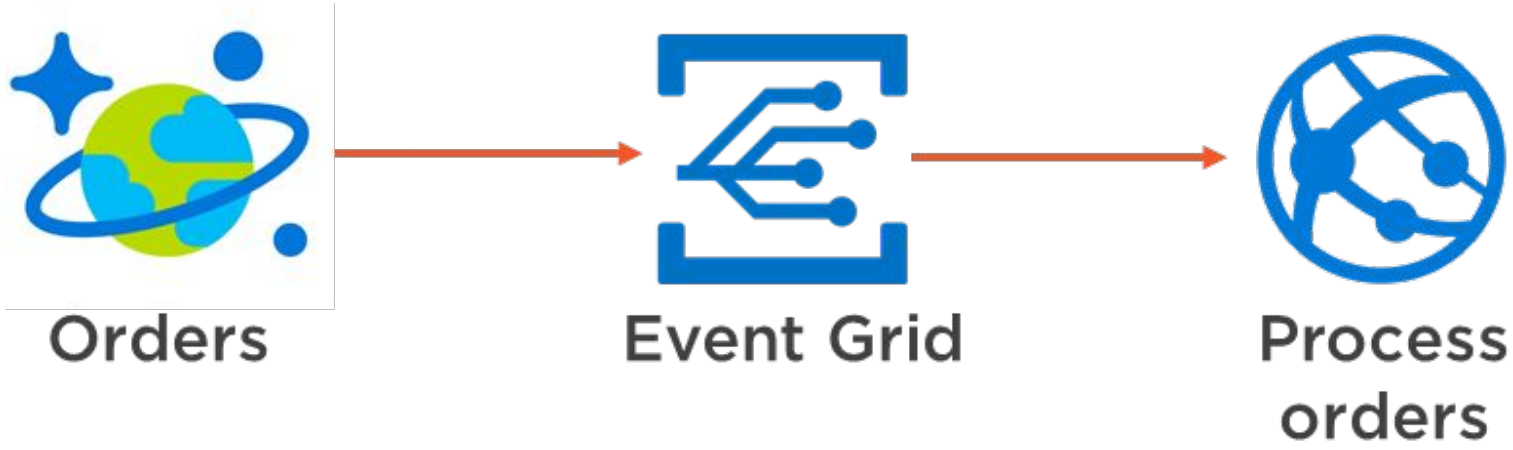
17 Configuring Azure Cache

18 Connecting to Redis cache using Stach.exchange Redis

Azure Event Grid

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.

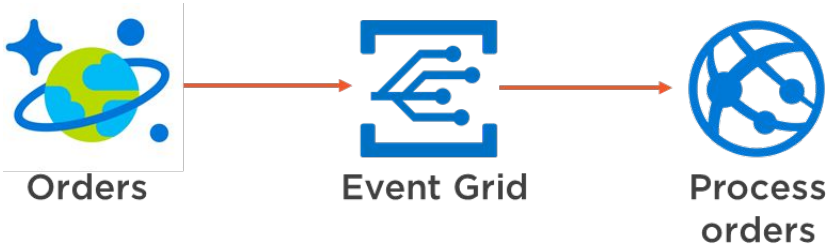


Azure Event Grid

Concepts in Azure Event Grid

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

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

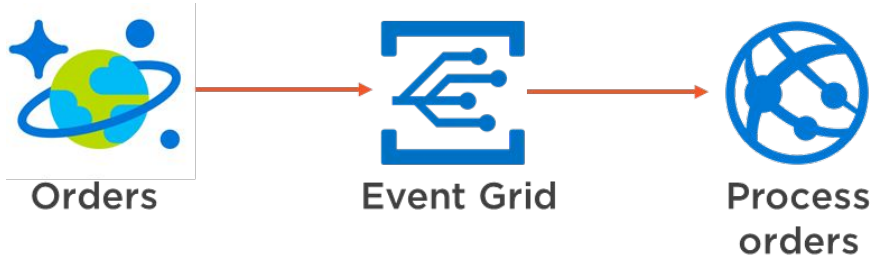


What is an Event?

EVENTS

An **event** is the smallest amount of information that fully describes something that happened in the system. Every event has common information like: source of the event, time the event took place, and unique identifier. Every event also has specific information that is only relevant to the specific type of event.

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

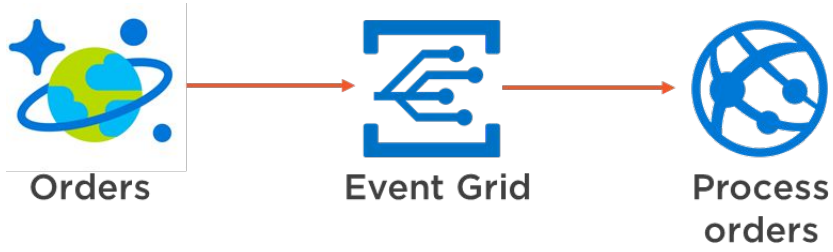


What is an Event?

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.



What is an Event Grid Topic?

The Event Grid Topic provides an endpoint where the source sends events. The publisher creates the event grid topic, and decides whether an event source needs one topic or more than one topic. A topic is used for a collection of related events. To respond to certain types of events, subscribers decide which topics to subscribe to.

System topics are built-in topics provided by Azure services. You don't see system topics in your Azure subscription because the publisher owns the topics, but you can subscribe to them. To subscribe, you provide information about the resource you want to receive events from. As long as you have access to the resource, you can subscribe to its events.

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

What is an Event Subscription?

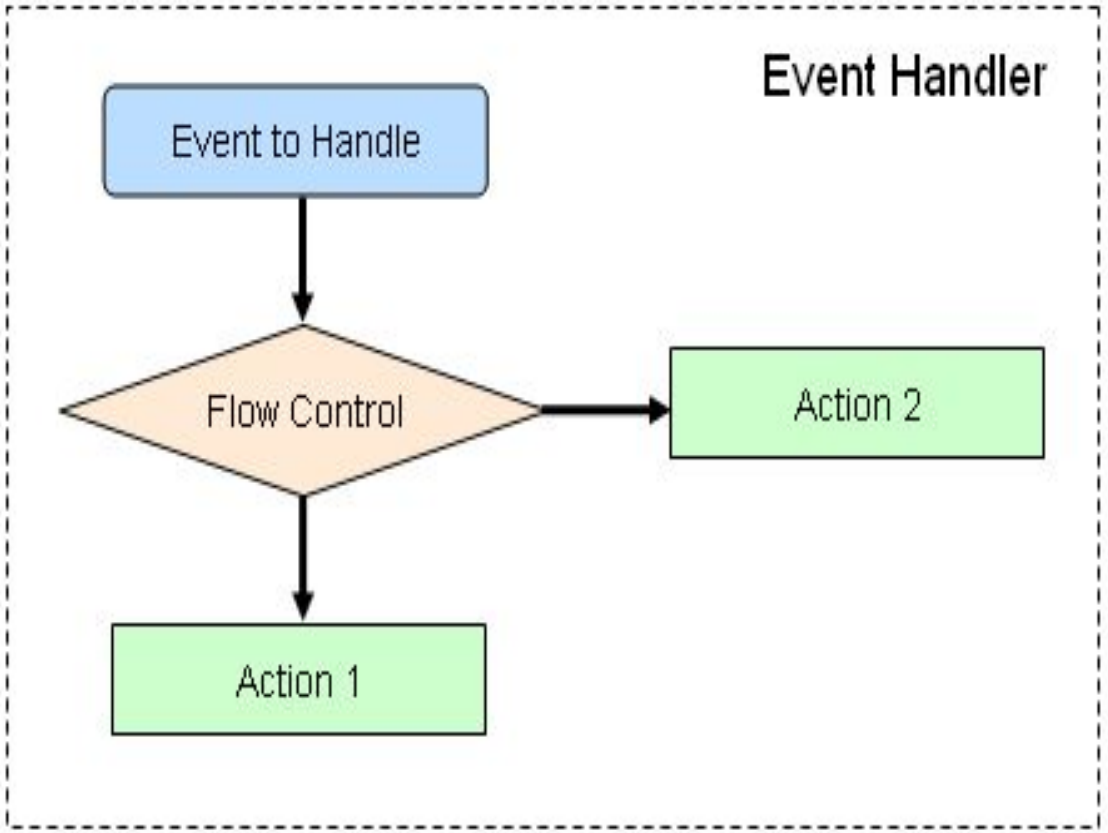
Event subscriptions

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

What is an Event Subscription?

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.



Azure Event Grid Schema

What is an Event Grid Event Schema?

Event Grid Schema

Events consist of a set of five required string properties and a required data object. 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.

What is an Event Grid Event Schema?

Event Grid Schema

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 64 KB (General Availability) or 1 MB (preview). If an event or the array is greater than the size limits, you receive the response **413 Payload** Too Large.

Event Grid sends the events to subscribers in an array that has a single event.

What is an Event Grid Event Schema?

Example :

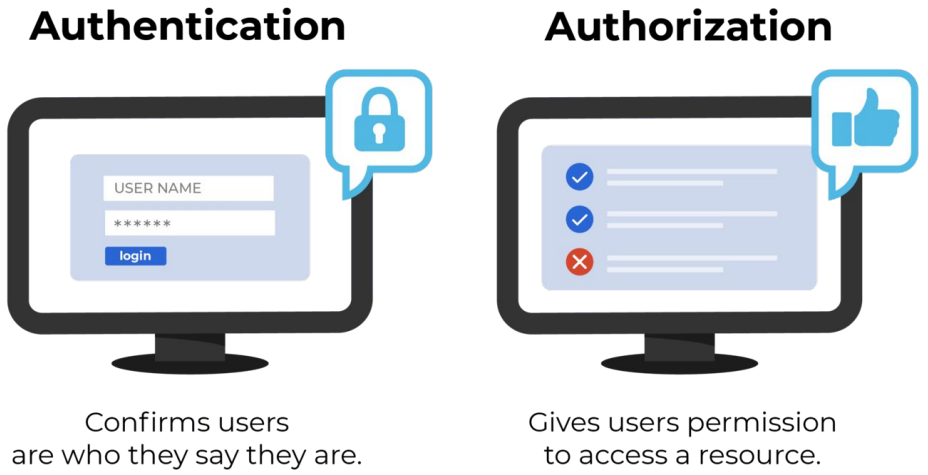
```
[
  {
    "topic": string,
    "subject": string,
    "id": string,
    "eventType": string,
    "eventTime": string,
    "data":{
      object-unique-to-each-publisher
    },
    "dataVersion": string,
    "metadataVersion": string
  }
]
```

Event Grid Security and Authentication

Event Grid Security and Authentication

Azure Event Grid has three types of authentication:

- WebHook event delivery
- Event subscriptions
- Custom topic publishing

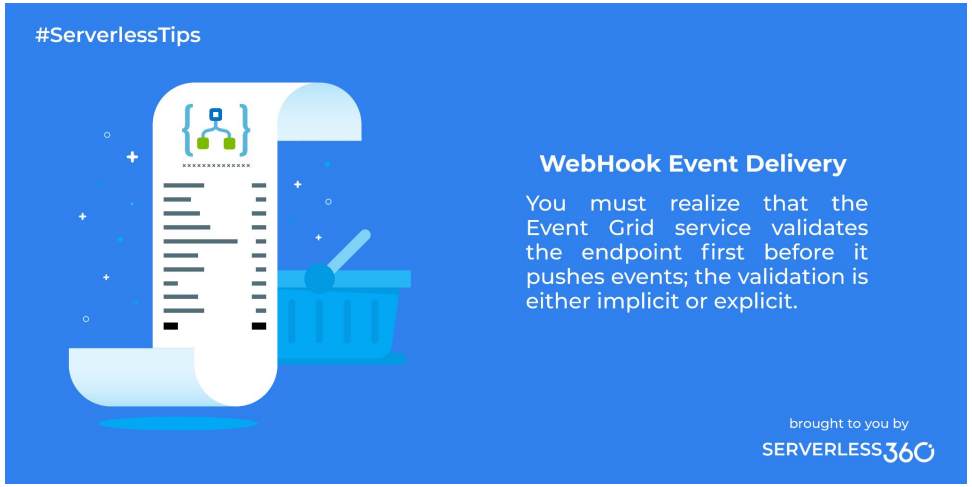


Event Grid Security and Authentication

WebHook Event delivery

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.



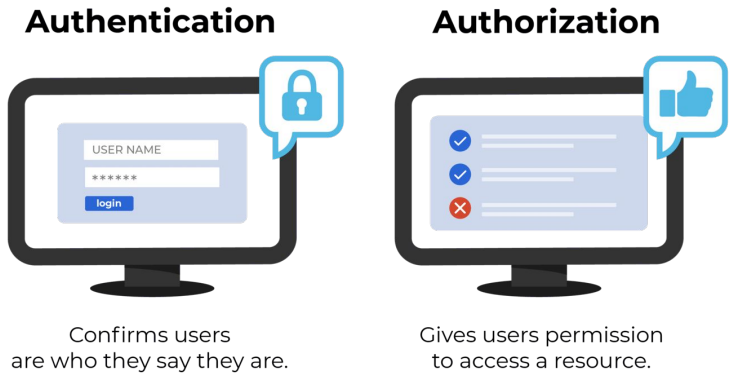
Event Grid Security and Authentication

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

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:



Event Grid Security and Authentication

ValidationCode handshake (programmatic): If you control the source code for your endpoint, this method is recommended. 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.

ValidationURL handshake (manual): In certain cases, you can't access the source code of the endpoint to implement the ValidationCode handshake. For example, if you use a third-party service (like Zapier or IFTTT), you can't programmatically respond with the validation code.

Event Grid Security and Authentication

Event subscription

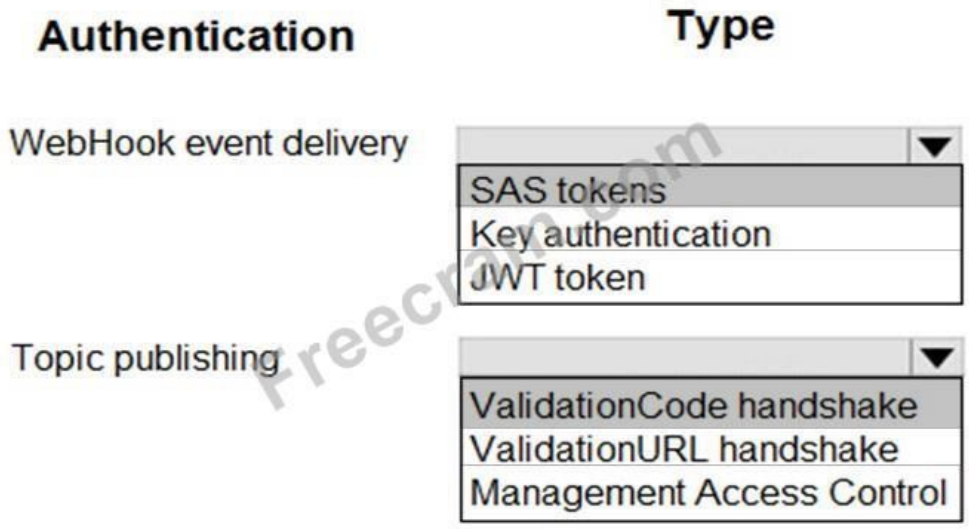
To subscribe to an event, you must prove that you have access to the event source and handler. Proving that you own a WebHook was covered in the preceding section.You must have the Microsoft.EventGrid/EventSubscriptions/Write permission on the resource that is the event source. You need this permission 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.

Event Grid Security and Authentication

Custom topic publishing

Custom topics use either Shared Access Signature (SAS) or key authentication. SAS is recommended, but key authentication provides simple programming and is compatible with many existing webhook publishers.

You include the authentication value in the HTTP header. For SAS, use `aeg-sas-token` for the header value. For key authentication, use `aeg-sas-key` for the header value.

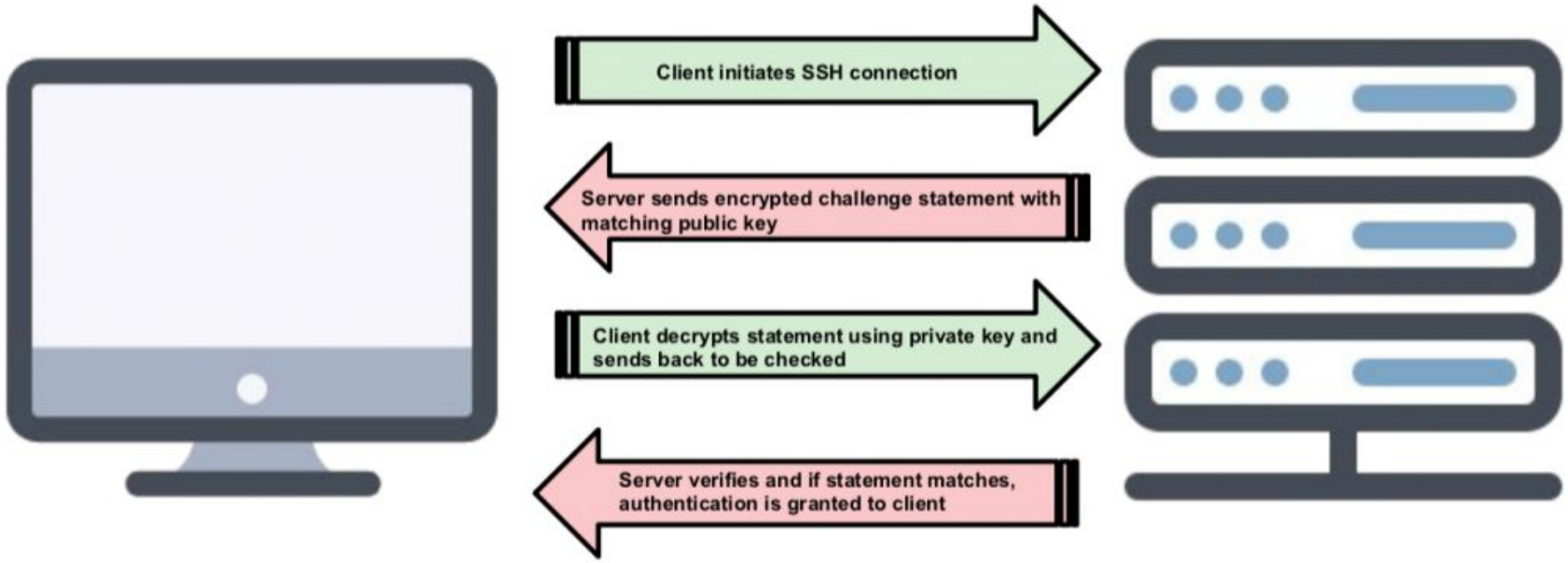


Event Grid Security and Authentication

Key authentication

Key authentication is the simplest form of authentication. Use the format:

```
aeg-sas-key: <your key>
```

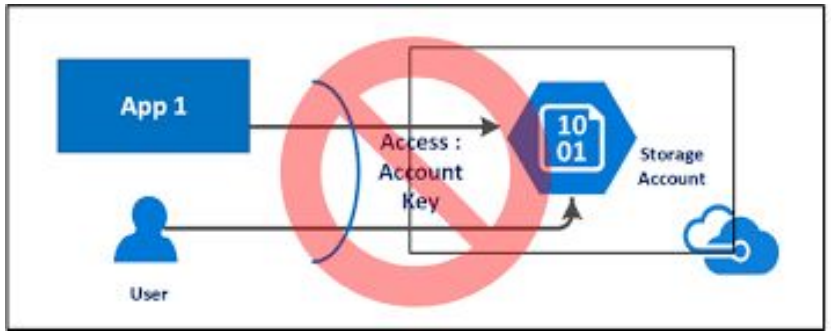


Event Grid Security and Authentication

SAS tokens

SAS tokens for Event Grid include the resource, an expiration time, and a signature. The format of the SAS token is: r={resource}&e={expiration}&s={signature}.

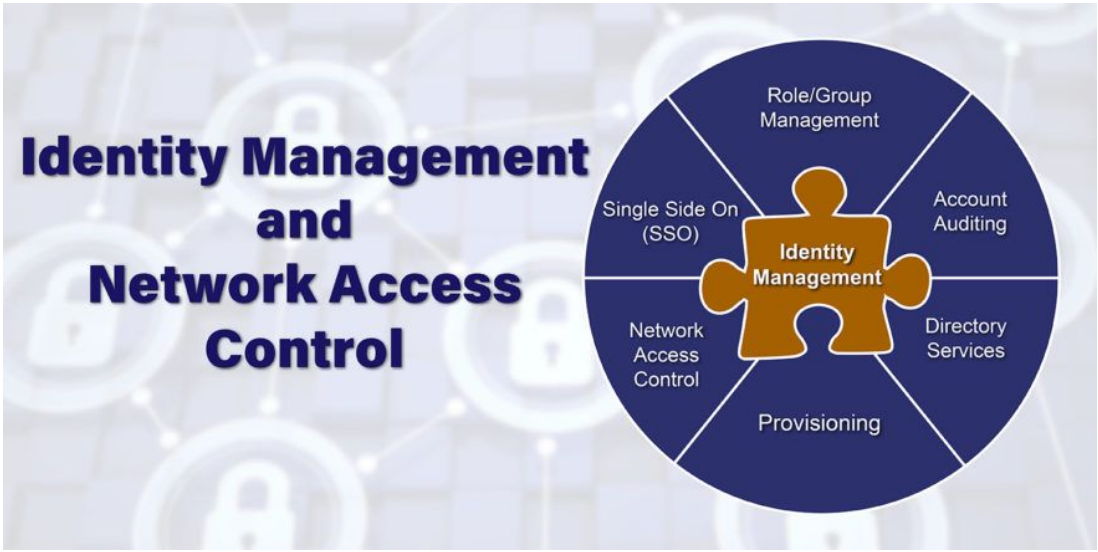
The resource is the path for the event grid topic to which you're sending events. For example, a valid resource path is: **https://<yourtopic>.<region>.eventgrid.azure.net/eventGrid/api/events**



Event Grid Security and Authentication

Management Access Control

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's role-based access control (RBAC).



Event Grid Security and Authentication

Operation types

Event Grid supports the following actions:

- Microsoft.EventGrid/*/read
- Microsoft.EventGrid/*/write
- Microsoft.EventGrid/*/delete
- Microsoft.EventGrid/eventSubscriptions/getFullUrl/action
- Microsoft.EventGrid/topics/listKeys/action
- Microsoft.EventGrid/topics/regenerateKey/action-The last three operations return potentially secret information, which gets filtered out of normal read operations. It's recommended that you restrict access to these operations.

Event Grid Security and Authentication

Built-in roles

Event Grid provides two built-in roles for managing event subscriptions. They are 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.

- EventGrid EventSubscription Contributor: manage Event Grid subscription operations
- EventGrid EventSubscription Reader: read Event Grid subscriptions

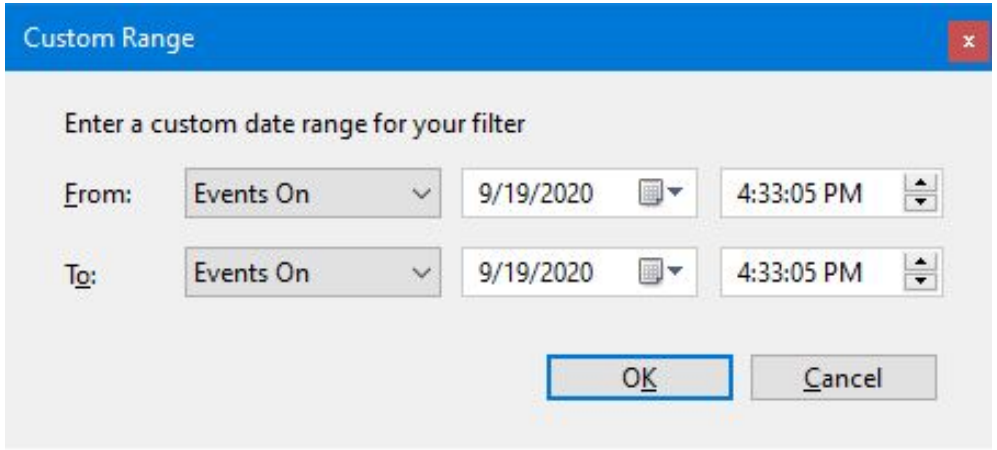
Event filtering for Event Grid subscriptions

Event Grid Security and Authentication

Event filtering for Event Grid subscriptions

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

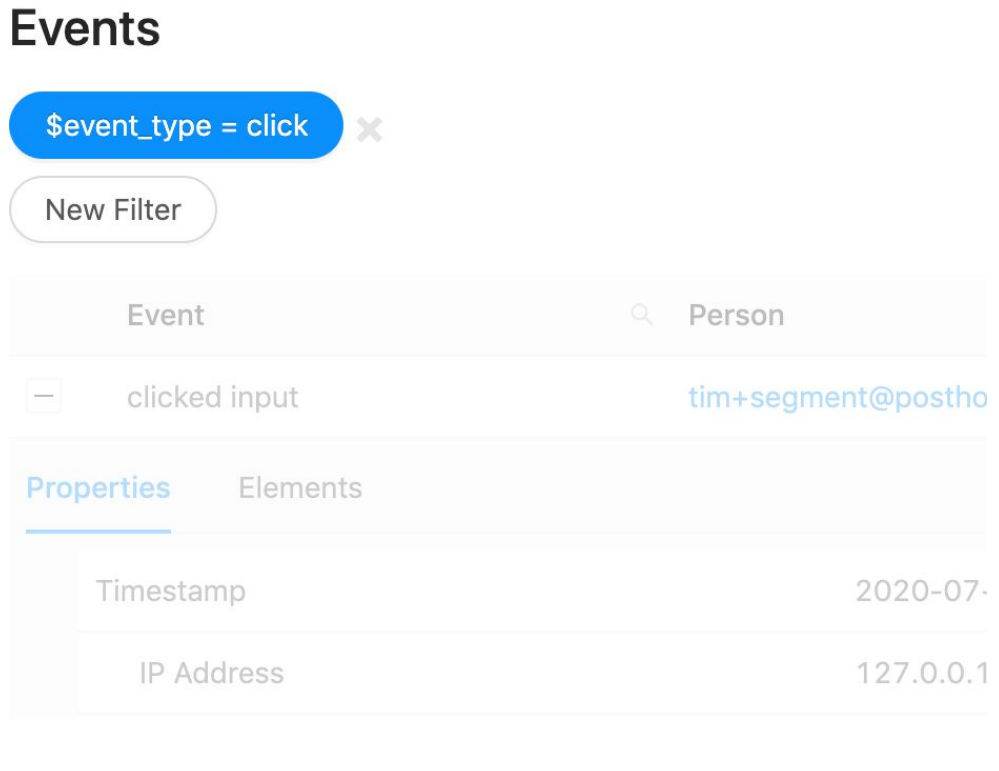
- Event types
- Subject begins with or ends with
- Advanced fields and operators



Event Grid Security and Authentication

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.



Event Grid Security and Authentication

The JSON syntax for filtering by event type is:

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

Event Grid Security and Authentication

Subject filtering

For simple filtering by subject, specify a starting or ending value for the subject. For example, you can specify the subject ends with `.txt` to only get events related to uploading a text file to storage account. Or, you can filter the subject begins with `/blobServices/default/containers/testcontainer` to get all events for that container but not other containers in the storage account.

The JSON syntax for filtering by subject is:

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

Event Grid Security and Authentication

Advanced filtering

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

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

Event Grid Security and Authentication

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

Hands-on: Route custom events to web endpoint by using Portal and Event Grid

Azure Event Hubs overview

Azure Event HUB

Azure Event HUB

Azure Event Hubs is a big data streaming platform and event ingestion service. It can receive and process millions of events per second. Data sent to an event hub can be transformed and stored by using any real-time analytics provider or batching/storage adapters.

Azure Event HUB

Azure Event HUB

Event Hubs represents the “front door” for an event pipeline, often called an *event ingestor* in solution architectures. An event ingestor is a component or service that sits between event publishers and event consumers to decouple the production of an event stream from the consumption of those events. Event Hubs provides a unified streaming platform with time retention buffer, decoupling event producers from event consumers.

Azure Event HUB

Azure Event HUB

Event Hubs represents the “front door” for an event pipeline, often called an *event ingestor* in solution architectures. An event ingestor is a component or service that sits between event publishers and event consumers to decouple the production of an event stream from the consumption of those events. Event Hubs provides a unified streaming platform with time retention buffer, decoupling event producers from event consumers.

Key architecture components

Azure Event HUB

Key architecture components

Event Hubs contains the following key components:

- **Event producers**: Any entity that sends data to an event hub. Event publishers can publish events using HTTPS or AMQP 1.0 or Apache Kafka (1.0 and above)
- **Partitions**: Each consumer only reads a specific subset, or partition, of the message stream.
- **Consumer groups**: A view (state, position, or offset) of an entire event hub. Consumer groups enable consuming applications to each have a separate view of the event stream. They read the stream independently at their own pace and with their own offsets.

Azure Event HUB

Key architecture components

Event Hubs contains the following key components:

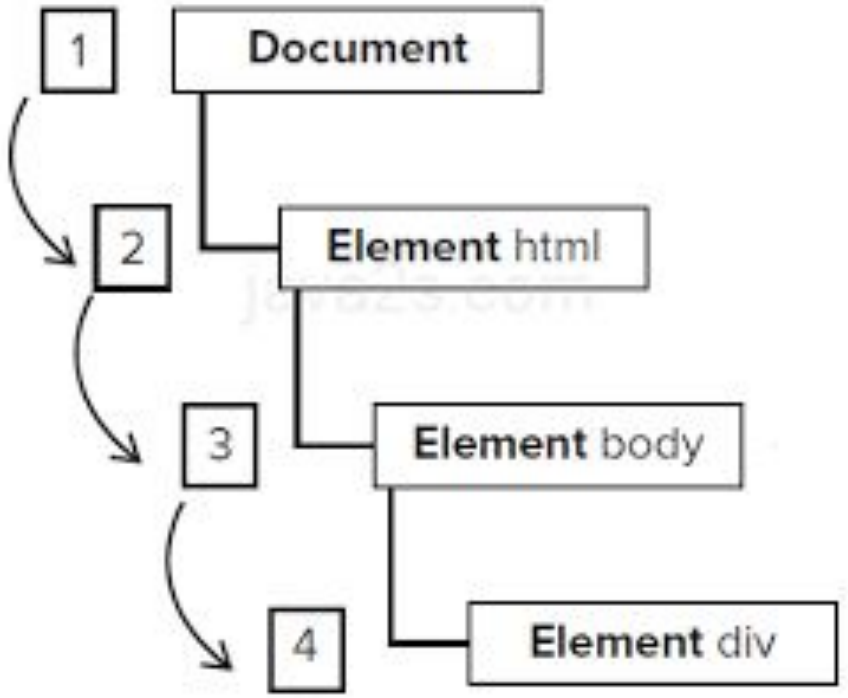
- **Throughput units**: Pre-purchased units of capacity that control the throughput capacity of Event Hubs.
- **Event receivers**: Any entity that reads event data from an event hub. All Event Hubs consumers connect via the AMQP 1.0 session. The Event Hubs service delivers events through a session as they become available. All Kafka consumers connect via the Kafka protocol 1.0 and later.

Capture events through Azure Event Hubs

Azure Event HUB

Capture events through Azure Event Hubs

Azure Event Hubs enables you to automatically capture the streaming data in Event Hubs in an Azure Blob storage or Azure Data Lake Storage account of your choice, with the added flexibility of specifying a time or size interval. Setting up Capture is fast, there are no administrative costs to run it, and it scales automatically with Event Hubs throughput units.



Azure Event HUB

How Event Hubs Capture works

Event Hubs is a time-retention durable buffer for telemetry ingress, similar to a distributed log. The key to scaling in Event Hubs is the partitioned consumer model. Each partition is an independent segment of data and is consumed independently. Over time this data ages off, based on the configurable retention period. As a result, a given event hub never gets “too full.”

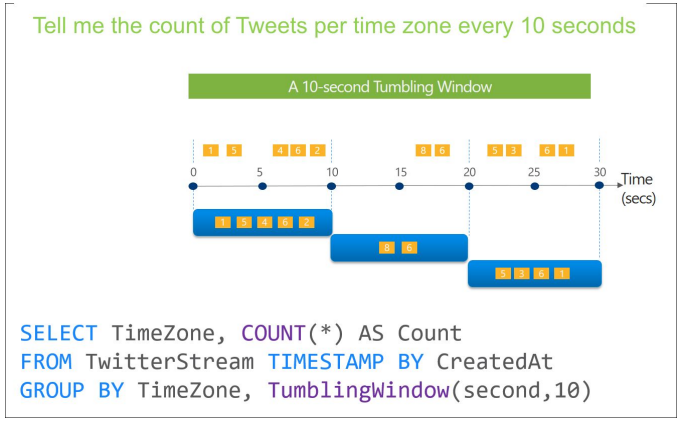
Event Hubs Capture enables you to specify your own Azure Blob storage account and container, or Azure Data Lake Store account, which are used to store the captured data. Captured data is written in Apache Avro format: a compact, fast, binary format that provides rich data structures with inline schema. This format is widely used in the Hadoop ecosystem, Stream Analytics, and Azure Data Factory.

Azure Event HUB

Capture windowing

Event Hubs Capture enables you to set up a window to control capturing. This window is a minimum size and time configuration with a “first wins policy,” meaning that the first trigger encountered causes a capture operation. Each partition captures independently and writes a completed block blob at the time of capture, named for the time at which the capture interval was encountered. The storage naming convention is as follows:

{Namespace}/{EventHub}/{PartitionId}/{Year}/{Month}/{Day}/{Hour}/{Minute}/{Second}

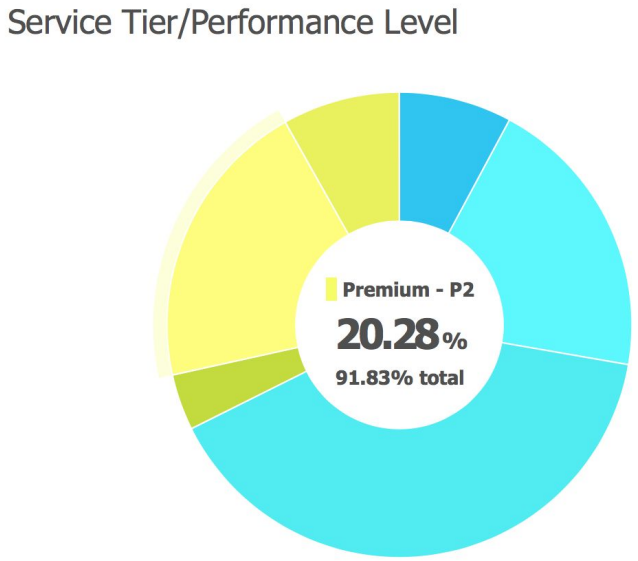


Azure Event HUB

Scaling to throughput units

Event Hubs traffic is controlled by throughput units. A single throughput unit allows 1 MB per second or 1000 events per second of ingress and twice that amount of egress. Event Hubs Capture copies data directly from the internal Event Hubs storage, bypassing throughput unit egress quotas and saving your egress for other processing readers, such as Stream Analytics or Spark.

Once configured, Event Hubs Capture runs automatically when you send your first event, and continues running. To make it easier for your downstream processing to know that the process is working, Event Hubs writes empty files when there is no data. This process provides a predictable cadence and marker that can feed your batch processors.

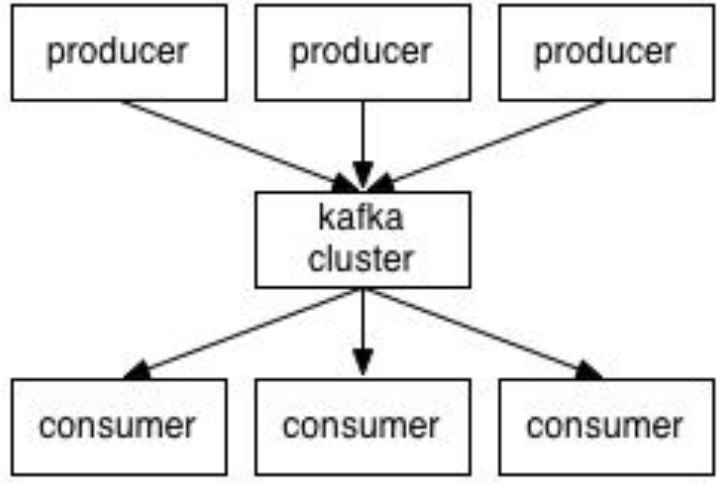


Use Azure Event Hubs from Apache Kafka applications

Azure Event HUB

Event Hubs provides a Kafka endpoint that can be used by your existing Kafka based applications as an alternative to running your own Kafka cluster. Event Hubs supports Apache Kafka protocol 1.0 and later, and works with your existing Kafka applications, including MirrorMaker.

You may start using the Kafka endpoint from your applications with no code change but a minimal configuration change. You update the connection string in configurations to point to the Kafka endpoint exposed by your event hub instead of pointing to your Kafka cluster. Then, you can start streaming events from your applications that use the Kafka protocol into Event Hubs.



Azure Event HUB

Kafka and Event Hub conceptual mapping

Kafka Concept	Event Hubs Concept
Cluster	Namespace
Topic	Event Hub
Partition	Partition
Consumer Group	Consumer Group
Offset	Offset

Azure Event HUB

Key differences between Kafka and Event Hubs

While Apache Kafka is software, which you can run wherever you choose, Event Hubs is a cloud service similar to Azure Blob Storage. There are no servers or networks to manage and no brokers to configure. You create a namespace, which is an FQDN in which your topics live, and then create Event Hubs or topics within that namespace. For more information about Event Hubs and namespaces, see Event Hubs features. As a cloud service, Event Hubs uses a single stable virtual IP address as the endpoint, so clients do not need to know about the brokers or machines within a cluster.

Event Hubs Dedicated overview

Azure Event HUB

Event Hub Dedicated -

Event Hubs clusters offer single-tenant deployments for customers with the most demanding streaming needs. This single-tenant offering has a guaranteed 99.99% SLA and is available only on our Dedicated pricing tier. An Event Hubs cluster can ingress millions of events per second with guaranteed capacity and sub-second latency.

Namespaces and event hubs created within the Dedicated cluster include all features of the Standard offering and more, but without any ingress limits. It also includes the popular Event Hubs Capture feature at no additional cost, allowing you to automatically batch and log data streams to Azure Storage or Azure Data Lake.

Azure Event HUB

Dedicated Event Hubs benefits

- Single-tenancy: A Dedicated cluster guarantees capacity at full scale, and can ingress up to gigabytes of streaming data with fully durable storage and sub-second latency to accommodate any burst in traffic.
- Access to features: The Dedicated offering includes features like Capture at no additional cost, as well as exclusive access to upcoming features like Bring Your Own Key (BYOK).
- Cost Savings: At high ingress volumes (>100 TUs), a cluster costs significantly less per hour than purchasing a comparable quantity of throughput units in the Standard offering.

Azure Event Hubs authentication and security model

Azure Event HUB

Azure Event Hubs authentication and security model

The Azure Event Hubs security model meets the following requirements:

- Only clients that present valid credentials can send data to an event hub.
- A client cannot impersonate another client.
- A rogue client can be blocked from sending data to an event hub.

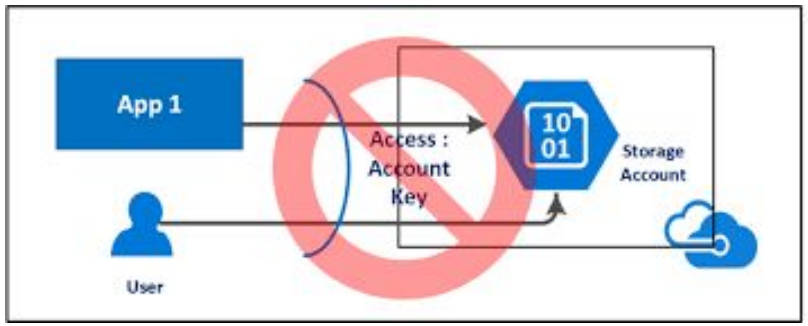
Azure Event HUB

Client authentication

The Event Hubs security model is based on a combination of Shared Access Signature (SAS) tokens and *event publishers*. An event publisher defines a virtual endpoint for an event hub. The publisher can only be used to send messages to an event hub. It is not possible to receive messages from a publisher.

Each Event Hubs client is assigned a unique token, which is uploaded to the client. The tokens are produced such that each unique token grants access to a different unique publisher. A client that possesses a token can only send to one publisher, but no other publisher. All tokens are signed with a SAS key. Typically, all tokens are signed with the same key.

Azure Event HUB



Create the SAS key

When creating an Event Hubs namespace, the service automatically generates a 256-bit SAS key named RootManageSharedAccessKey. This rule has an associated pair of primary and secondary keys that grant send, listen, and manage rights to the namespace. You can also create additional keys. It is recommended that you produce a key that grants send permissions to the specific event hub. For the remainder of this topic, it is assumed that you named this key EventHubSendKey.

Generate tokens

You can generate tokens using the SAS key. You must produce only one token per client. Tokens can then be produced using the following method. All tokens are generated using the `EventHubSendKey` key. The 'resource' parameter corresponds to the URI endpoint of the service(event hub in this case).

When calling this method, the URI should be specified as `//<NAMESPACE>.servicebus.windows.net/<EVENT_HUB_NAME>/publishers/<PUBLISHER_NAME>`. For all tokens, the URI is identical, with the exception of `PUBLISHER_NAME`, which should be different for each token. Ideally, `PUBLISHER_NAME` represents the ID of the client that receives that token.

Azure Event HUB

Sending data

Once the tokens have been created, each client is provisioned with its own unique token.

When the client sends data into an event hub, it tags its send request with the token. To prevent an attacker from eavesdropping and stealing the token, the communication between the client and the event hub must occur over an encrypted channel.

Denying client access

If a token is stolen by an attacker, the attacker can impersonate the client whose token has been stolen. Denying access to a client renders that client unusable until it receives a new token that uses a different publisher.

Azure Event HUB

Authentication of back-end applications

A client can create a consumer group if the request to create the consumer group is accompanied by a token that grants manage privileges for the event hub, or for the namespace to which the event hub belongs. A client is allowed to consume data from a consumer group if the receive request is accompanied by a token that grants receive rights on that consumer group, the event hub, or the namespace to which the event hub belongs.

In the absence of SAS authentication for individual consumer groups, you can use SAS keys to secure all consumer groups with a common key. This approach enables an application to consume data from any of the consumer groups of an event hub.

Analyzing and troubleshooting apps

Troubleshooting Apps

View activity logs to audit actions on resources

The Azure Activity Log provides insight into subscription-level events that have occurred in Azure. This includes a range of data, from Azure Resource Manager operational data to updates on Service Health events. The Activity Log was previously known as *Audit Logs* or *Operational Logs*, since the Administrative category reports control-plane events for your subscriptions.

Use the Activity Log, to determine the *what*, *who*, and *when* for any write operations (PUT, POST, DELETE) taken on the resources in your subscription. You can also understand the status of the operation and other relevant properties.

The Activity Log does not include read (GET) operations or operations for resources that use the Classic/RDFE model.

Troubleshooting Apps

View the Activity Log

View the Activity Log for all resources from the Monitor menu in the Azure portal. View the Activity Log for a particular resource from the Activity Log option in that resource's menu. You can also retrieve Activity Log records with PowerShell, CLI, or REST API.

Export Activity Log

Export the Activity Log to Azure Storage for archiving or stream it to an Event Hub for ingestion by a third-party service or custom analytics solution.

Alert on Activity Log

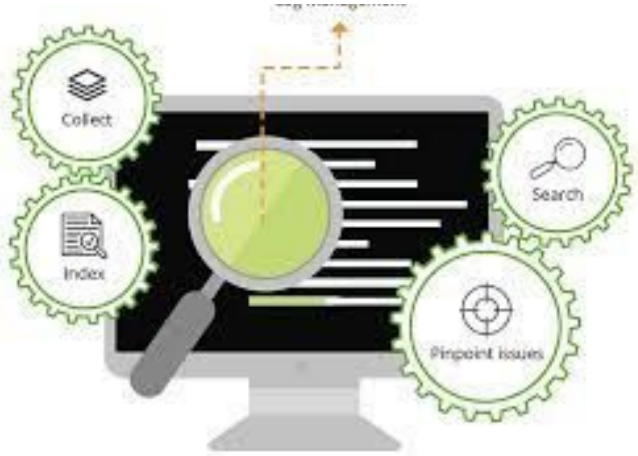
You can create an alert when particular events are created in the Activity Log with an Activity Log alert. You can also create an alert using a log query when your Activity Log is connected to a Log Analytics workspace, but there is a cost to log query alerts. There is no cost for Activity Log alerts.

Troubleshooting Apps

Monitor availability and responsiveness of any website

After you've deployed your web app/website, you can set up recurring tests to monitor availability and responsiveness. Azure Application Insights sends web requests to your application at regular intervals from points around the world. It can alert you if your application isn't responding, or if it responds too slowly.

You can set up availability tests for any HTTP or HTTPS endpoint that is accessible from the public internet. You don't have to make any changes to the website you're testing. In fact, it doesn't even have to be a site you own. You can test the availability of a REST API that your service depends on.

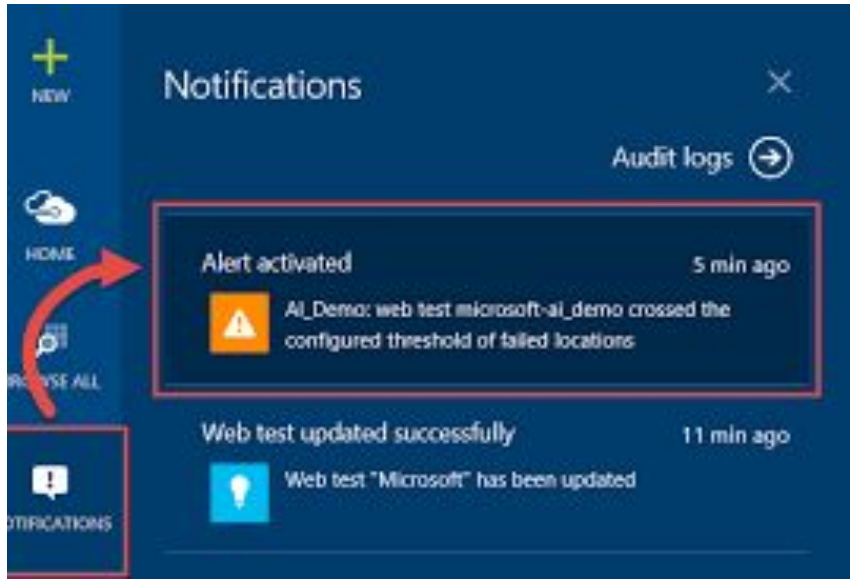


Types of availability tests

Troubleshooting Apps

There are three types of availability tests:

- **URL ping test**: a simple test that you can create in the Azure portal.
- **Multi-step web test:** A recording of a sequence of web requests, which can be played back to test more complex scenarios. Multi-step web tests are created in Visual Studio Enterprise and uploaded to the portal for execution.

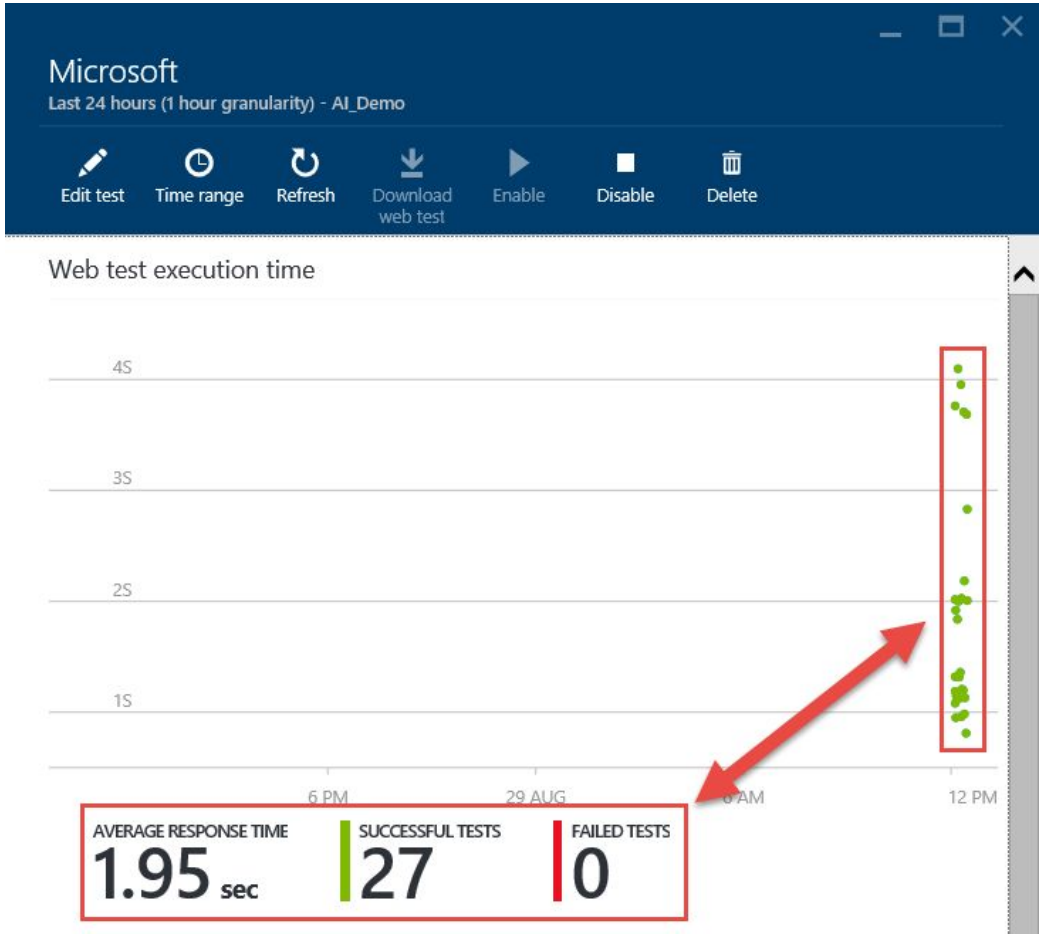


- **Custom Track Availability Tests:** If you decide to create a custom application to run availability tests, the TrackAvailability() method can be used to send the results to Application Insights.You can create up to 100 availability tests per Application Insights resource.

Troubleshooting Apps

URL ping test

The name “URL ping test” is a bit of a misnomer. To be clear, this test is not making any use of ICMP (Internet Control Message Protocol) to check your site's availability. Instead it uses more advanced HTTP request functionality to validate whether an endpoint is responding. It also measures the performance associated with that response, and adds the ability to set custom success criteria coupled with more advanced features like parsing dependent requests, and allowing for retries.



Hands-on: Monitor availability with URL ping tests

Application Map: Triage Distributed Applications in Application Insights

Troubleshooting Apps

Application Map -

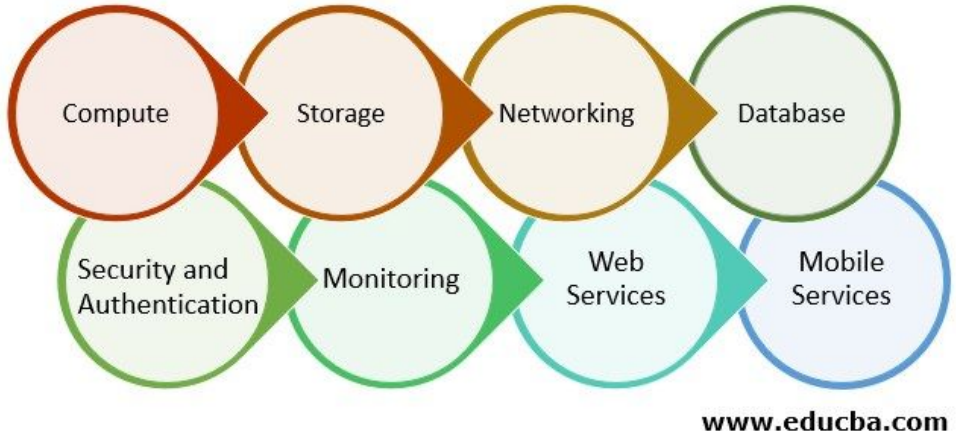
Application Map helps you spot performance bottlenecks or failure hotspots across all components of your distributed application. Each node on the map represents an application component or its dependencies; and has health KPI and alerts status. You can click through from any component to more detailed diagnostics, such as Application Insights events. If your app uses Azure services, you can also click through to Azure diagnostics, such as SQL Database Advisor recommendations.

Troubleshooting Apps

What is a Component?

Components are independently deployable parts of your distributed/microservices application. Developers and operations teams have code-level visibility or access to telemetry generated by these application components.

- Components are different from “observed” external dependencies such as SQL, EventHub etc. which your team/organization may not have access to (code or telemetry).
- Components run on any number of server/role/container instances.



Troubleshooting Apps

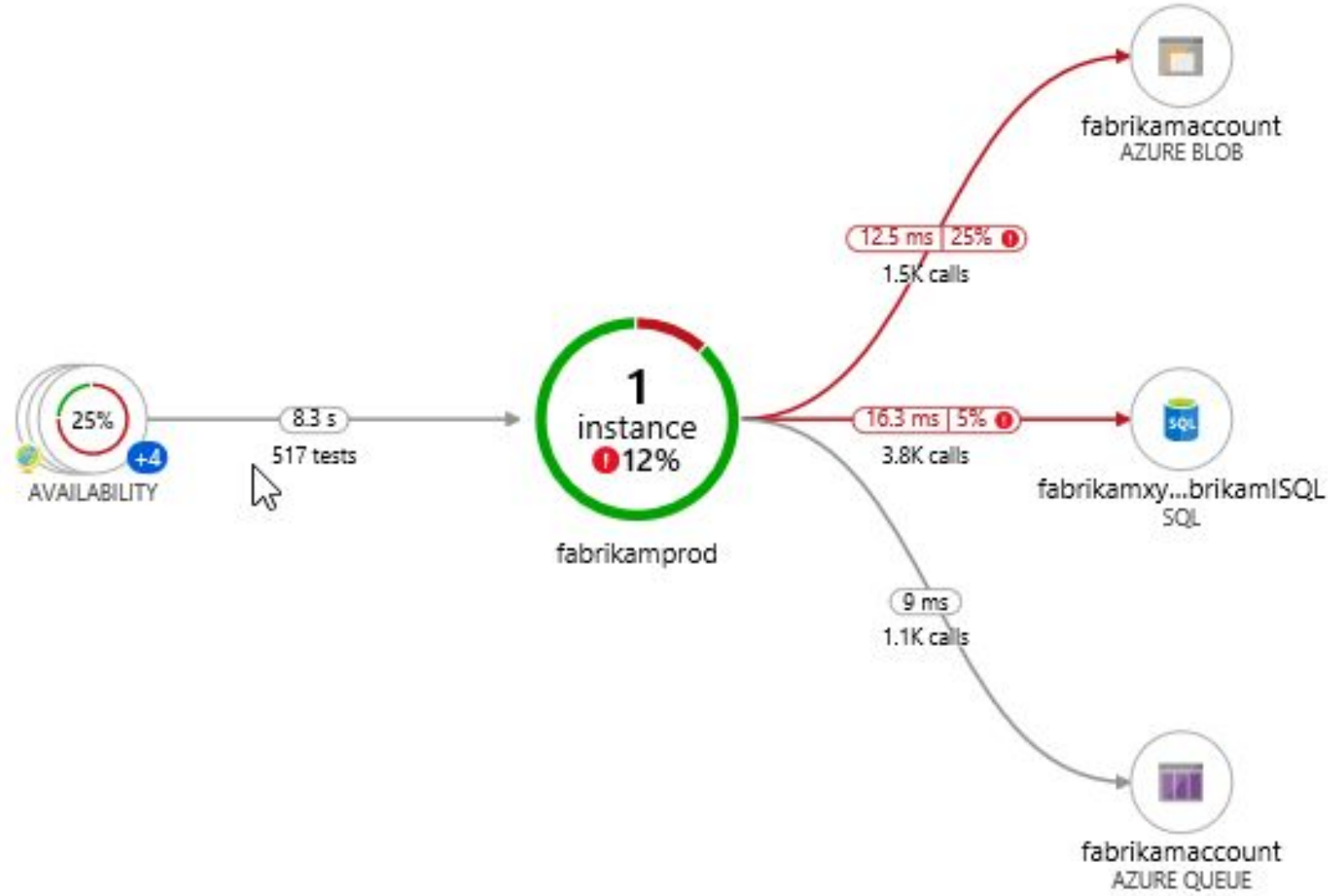
Composite Application Map

You can see the full application topology across multiple levels of related application components. Components could be different Application Insights resources, or different roles in a single resource. The app map finds components by following HTTP dependency calls made between servers with the Application Insights SDK installed.

This experience starts with progressive discovery of the components. When you first load the application map, a set of queries are triggered to discover the components related to this component. A button at the top-left corner will update with the number of components in your application as they are discovered.

On selecting Update map components, the map is refreshed with all components discovered until that point. Depending on the complexity of your application, this may take a minute to load.

Troubleshooting Apps



Troubleshooting Apps

One of the key objectives with this experience is to be able to visualize complex topologies with hundreds of components.

Click on any component to see related insights and go to the performance and failure triage experience for that component.

Set cloud role name

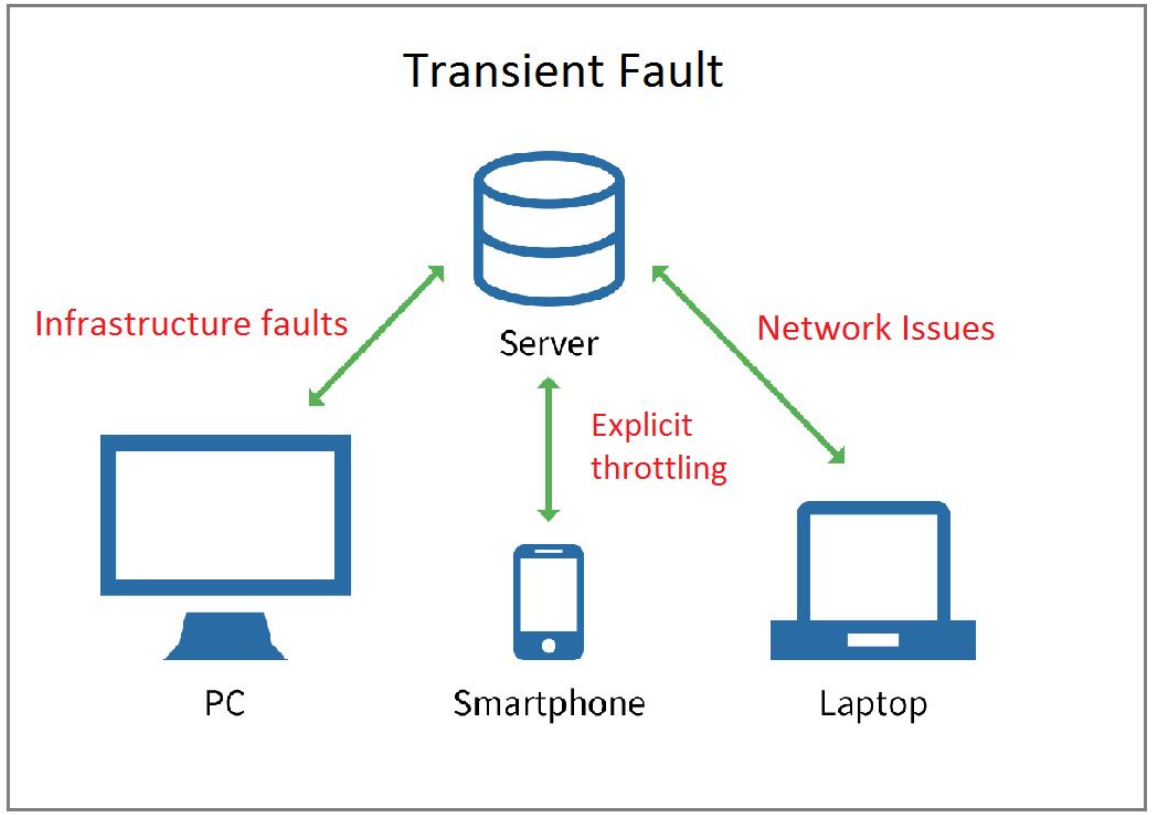
Application Map uses the cloud role name property to identify the components on the map. The Application Insights SDK automatically adds the cloud role name property to the telemetry emitted by components. For example, the SDK will add a website name or service role name to the cloud role name property. However, there are cases where you may want to override the default value.

Transient Faults

Transient Fault

An application that communicates with elements running in the cloud has to be sensitive to the transient faults that can occur in this environment. Faults include the momentary loss of network connectivity to components and services, the temporary unavailability of a service, or timeouts that occur when a service is busy.

These faults are typically self-correcting, and if the action that triggered a fault is repeated after a suitable delay, it's likely to be successful.



Transient Faults

Handling transient errors

In the cloud, transient faults aren't uncommon, and an application should be designed to handle them elegantly and transparently. This minimizes the effects faults can have on the business tasks the application is performing.

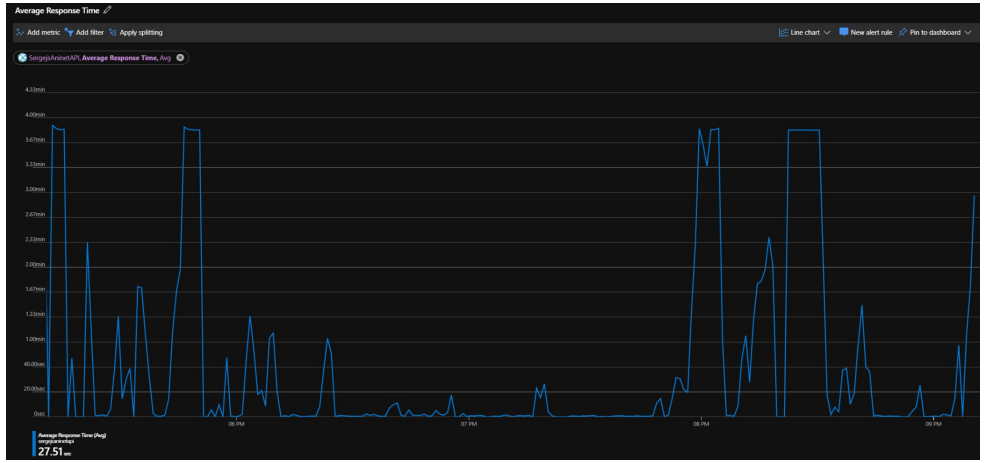
If an application detects a failure when it tries to send a request to a remote service, it can handle the failure using the following strategies:

- Cancel: If the fault indicates that the failure isn't transient or is unlikely to be successful if repeated, the application should cancel the operation and report an exception.

	Asynchronous	Synchronous
Poisoned Message	Dead Letter Queue	Return Error
Transient Error	Retry Process Dead Letter Queue (if needed)	Retry Process Return error (if needed)

Transient Faults

- **Retry:** If the specific fault reported is unusual or rare, it might have been caused by unusual circumstances, such as a network packet becoming corrupted while it was being transmitted. In this case, the application could retry the failing request again immediately, because the same failure is unlikely to be repeated, and the request will probably be successful.
- **Retry after a delay**: If the fault is caused by one of the more commonplace connectivity or busy failures, the network or service might need a short period of time while the connectivity issues are corrected or the backlog of work is cleared. The application should wait for a suitable amount of time before retrying the request.



Transient Faults

If the request still fails, the application can wait and make another attempt. If necessary, this process can be repeated with increasing delays between retry attempts, until some maximum number of requests have been attempted. The delay can be increased incrementally or exponentially depending on the type of failure and the probability that it'll be corrected during this time.

Retrying after a transient error

The following diagram illustrates invoking an operation in a hosted service using this pattern. If the request is unsuccessful after a predefined number of attempts, the application should treat the fault as an exception and handle it accordingly.

1. The application invokes an operation on a hosted service. The request fails, and the service host responds with HTTP response code 500 (internal server error).

Transient Faults

2. The application waits for a short interval and tries again. The request still fails with HTTP response code 500.

3. The application waits for a longer interval and tries again. The request succeeds with HTTP response code 200 (OK).The application should wrap all attempts to access a remote service in code that implements a retry policy matching one of the strategies listed above. Requests sent to different services can be subject to different policies

Transient Faults

Why do transient faults occur in the cloud?

There are several reasons for this:

- Many resources in a cloud environment are shared, and access to these resources is subject to throttling in order to protect the resource. Some services will refuse connections when the load rises to a specific level, or a maximum throughput rate is reached, in order to allow processing of existing requests and to maintain performance of the service for all users. Throttling helps to maintain the quality of service for neighbors and other tenants using the shared resource.
- There are often more hardware components, including network infrastructure such as routers and load balancers, between the application and the resources and services it uses. This additional infrastructure can occasionally introduce additional connection latency and transient connection faults.
- Network conditions between the client and the server may be variable, especially when communication crosses the Internet. Even in on-premises locations, heavy traffic loads may slow communication and cause intermittent connection failures.

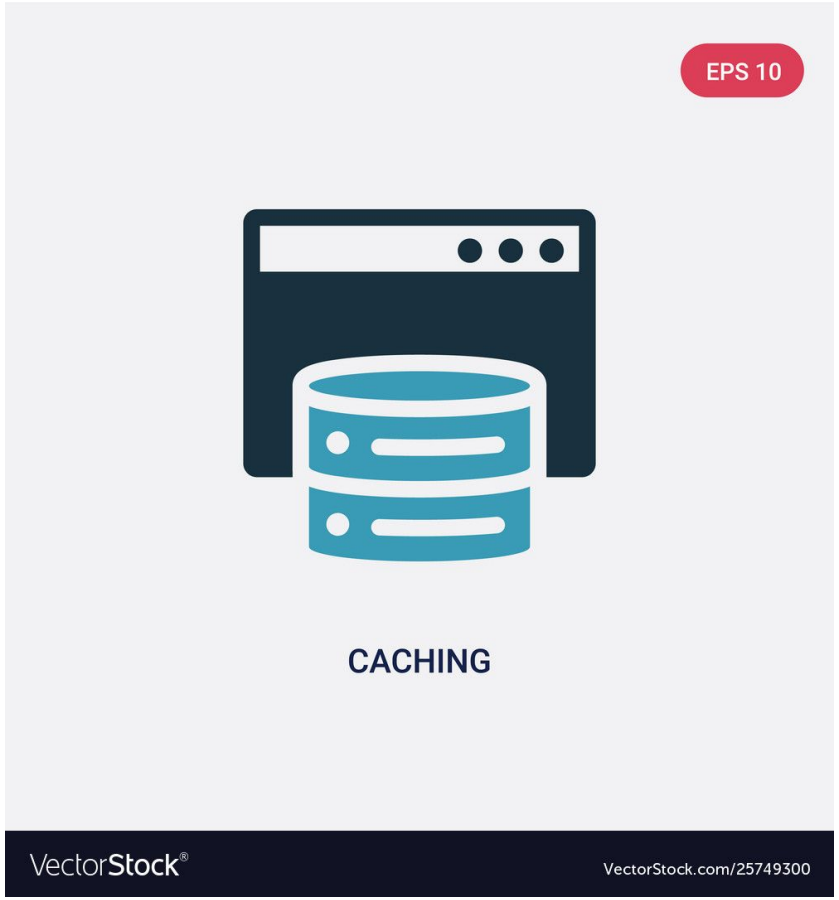
Azure Cache for Redis overview

Azure Cache for Redis

Caching is the act of storing frequently accessed data in memory that is very close to the application that consumes the data. Caching is used to increase performance and reduce the load on your servers. Azure Cache for Redis can be used to create an in-memory cache that can provide excellent latency and potentially improve performance. Azure Cache for Redis is based on the popular software Redis.

It gives you access to a secure, dedicated Redis cache, managed by Microsoft. A cache created using Azure Cache for Redis is accessible from any application within Azure. Azure Cache for Redis is typically used to improve the performance of systems that rely heavily on back-end data stores.

.

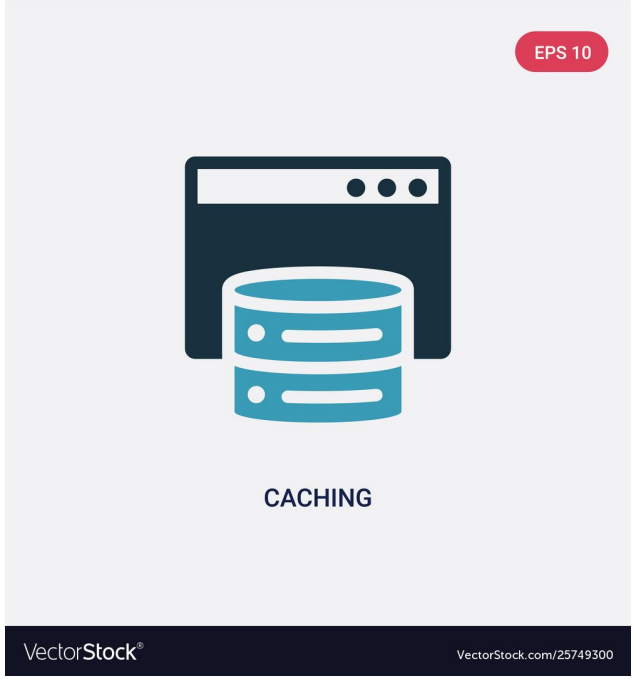


Azure Cache for Redis

What type of data can be stored in the cache?

Redis supports a variety of data types all oriented around binary safe strings. This means that you can use any binary sequence for a value, from a string like “i-love-rocky-road” to the contents of an image file. An empty string is also a valid value.

- Binary-safe strings (most common)
- Lists of strings
- Unordered sets of strings
- Hashes
- Sorted sets of strings
- Maps of strings



Azure Cache for Redis

What is a Redis key?

Redis keys are also binary safe strings. Here are some guidelines for choosing keys:

- Avoid long keys. They take up more memory and require longer lookup times because they have to be compared byte-by-byte. If you want to use a binary blob as the key, generate a unique hash and use that as the key instead. The maximum size of a key is 512 MB, but you should never use a key that size.
- Use keys which can identify the data. For example, “sport:football;date:2008-02-02” would be a better key than "fb:8-2-2". The former is more readable and the extra size is negligible. Find the balance between size and readability.
- Use a convention. A good one is “object:id”, as in "sport:football".

Azure Cache for Redis

What is a Redis key?

Redis keys are also binary safe strings. Here are some guidelines for choosing keys:

- Avoid long keys. They take up more memory and require longer lookup times because they have to be compared byte-by-byte. If you want to use a binary blob as the key, generate a unique hash and use that as the key instead. The maximum size of a key is 512 MB, but you should never use a key that size.
- Use keys which can identify the data. For example, “sport:football;date:2008-02-02” would be a better key than "fb:8-2-2". The former is more readable and the extra size is negligible. Find the balance between size and readability.
- Use a convention. A good one is “object:id”, as in "sport:football".

Azure Cache for Redis

How is data stored in a Redis cache?

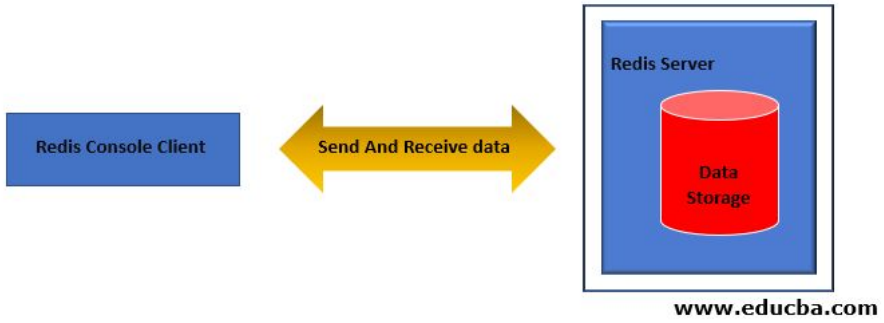
Data in Redis is stored in *nodes* and *clusters*.Nodes are a space in Redis where your data is stored.

Clusters are sets of three or more nodes your dataset is split across. Clusters are useful because your operations will continue if a node fails or is unable to communicate to the rest of the cluster.

What are Redis caching architectures?

Redis caching architecture is how we distribute our data in the cache. Redis distributed data in three major ways:

1. Single node
2. Multiple node
3. Clustered



Azure Cache for Redis

Redis caching architectures are split across Azure by tiers:

Basic cache: A basic cache provides you with a single node Redis cache. The complete dataset will be stored in a single node. This tier is ideal for development, testing, and non-critical workloads.

Standard cache: The standard cache creates multiple node architectures. Redis replicates a cache in a two-node primary/secondary configuration. Azure manages the replication between the two nodes.

Premium tier: The premium tier includes the features of the standard tier but adds the ability to persist data, take snapshots, and back up data. With this tier, you can create a Redis cluster that shards data across multiple Redis nodes to increase available memory.

Configure Azure Cache for Redis

Azure Cache for Redis

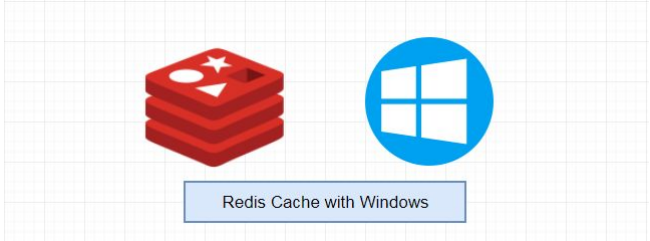
Create and configure the Azure Cache for Redis instance

You can create a Redis cache using the Azure portal, the Azure CLI, or Azure PowerShell. There are several parameters you will need to decide in order to configure the cache properly for your purposes.

Name

The Redis cache will need a globally unique name. The name has to be unique within Azure because it is used to generate a public-facing URL to connect and communicate with the service.

The name must be between 1 and 63 characters, composed of numbers, letters, and the ‘-’ character. The cache name can't start or end with the '-' character, and consecutive ‘-’ characters aren't valid.



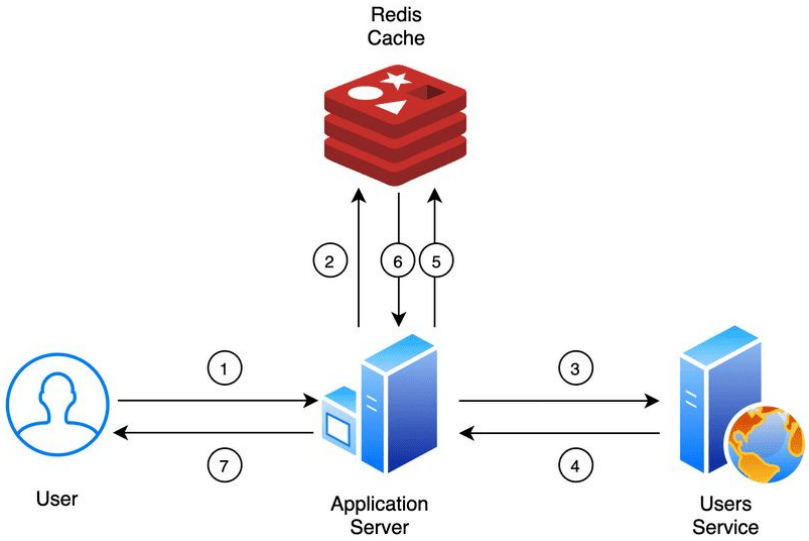
Azure Cache for Redis

Resource Group

The Azure Cache for Redis is a managed resource and needs a resource group owner. You can either create a new resource group, or use an existing one in a subscription you are part of.

Location

You will need to decide where the Redis cache will be physically located by selecting an Azure region. You should always place your cache instance and your application in the same region. Connecting to a cache in a different region can significantly increase latency and reduce reliability. If you are connecting to the cache outside of Azure, then select a location close to where the application consuming the data is running.

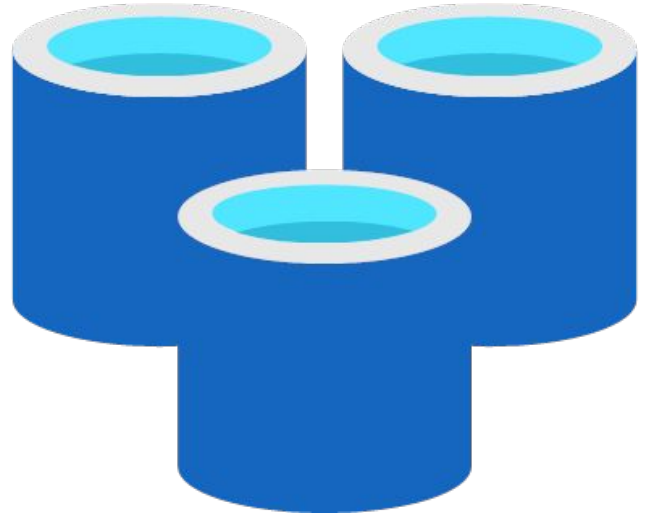


Azure Cache for Redis

Pricing tier

As mentioned in the last unit, there are three pricing tiers available for an Azure Cache for Redis.

- **Basic:** Basic cache ideal for development/testing. Is limited to a single server, 53 GB of memory, and 20,000 connections. There is no SLA for this service tier.
- **Standard**: Production cache which supports replication and includes an 99.99% SLA. It supports two servers, and has the same memory/connection limits as the Basic tier.
- **Premium:** Enterprise tier which builds on the Standard tier and includes persistence, clustering, and scale-out cache support. This is the highest performing tier with up to 530 GB of memory and 40,000 simultaneous connections.



Azure Cache for Redis

Virtual Network support

If you create a premium tier Redis cache, you can deploy it to a virtual network in the cloud. Your cache will be available to only other virtual machines and applications in the same virtual network. This provides a higher level of security when your service and cache are both hosted in Azure, or are connected through an Azure virtual network VPN.

Clustering support

With a premium tier Redis cache, you can implement clustering to automatically split your dataset among multiple nodes. To implement clustering, you specify the number of shards to a maximum of 10. The cost incurred is the cost of the original node, multiplied by the number of shards.

Azure Cache for Redis

Redis supports a set of known commands. A command is typically issued as `COMMAND parameter1 parameter2 parameter3`.

Here are some common commands you can use:

COMMAND	DESCRIPTION
PING	Ping the server.
set [key][value]	Set a key/value pair.
get [key]	Get a value key pair.
exists [key]	Return ‘1’ if the key exists in the cache.
type [key]	Returns the type associated to the value for the given key.
incr [key]	Increment the given value associated with key by '1'.
`del[key]`	Deletes the value associated with the key.
flushdb	Delete *all* keys and values in the database.

Azure Cache for Redis

Adding an expiration time to values

Caching is important because it allows us to store commonly used values in memory. However, we also need a way to expire values when they are stale. In Redis this is done by applying a time to live (TTL) to a key.

When the TTL elapses, the key is automatically deleted, exactly as if the DEL command were issued. Here are some notes on TTL expirations.

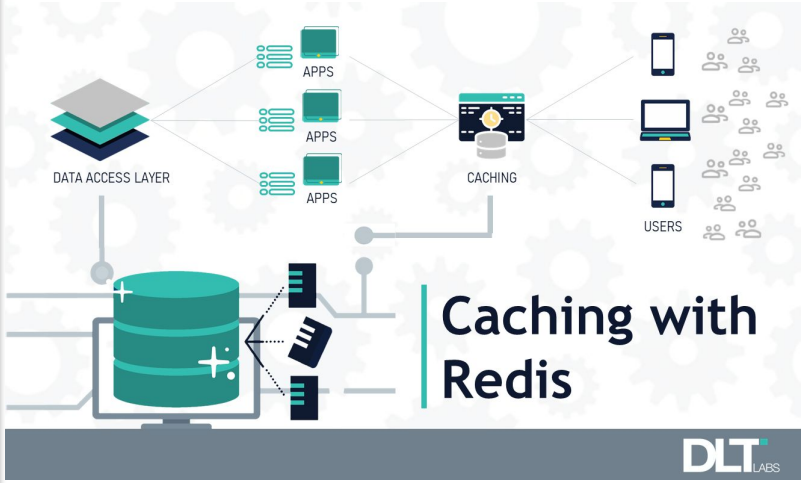
- Expirations can be set using seconds or milliseconds precision.
- The expire time resolution is always 1 millisecond.
- Information about expires are replicated and persisted on disk, the time virtually passes when your Redis server remains stopped.

Azure Cache for Redis

Accessing a Redis cache from a client

To connect to an Azure Cache for Redis instance, you'll need several pieces of information. Clients need the host name, port, and an access key for the cache. You can retrieve this information in the Azure portal through the **Settings > Access Keys page.**

- The host name is the public Internet address of your cache, which was created using the name of the cache. For example `sports results.redis.cache.windows.net`.
- The access key acts as a password for your cache. There are two keys created: primary and secondary. You can use either key, two are provided in case you need to change the primary key. You can switch all of your clients to the secondary key, and regenerate the primary key.



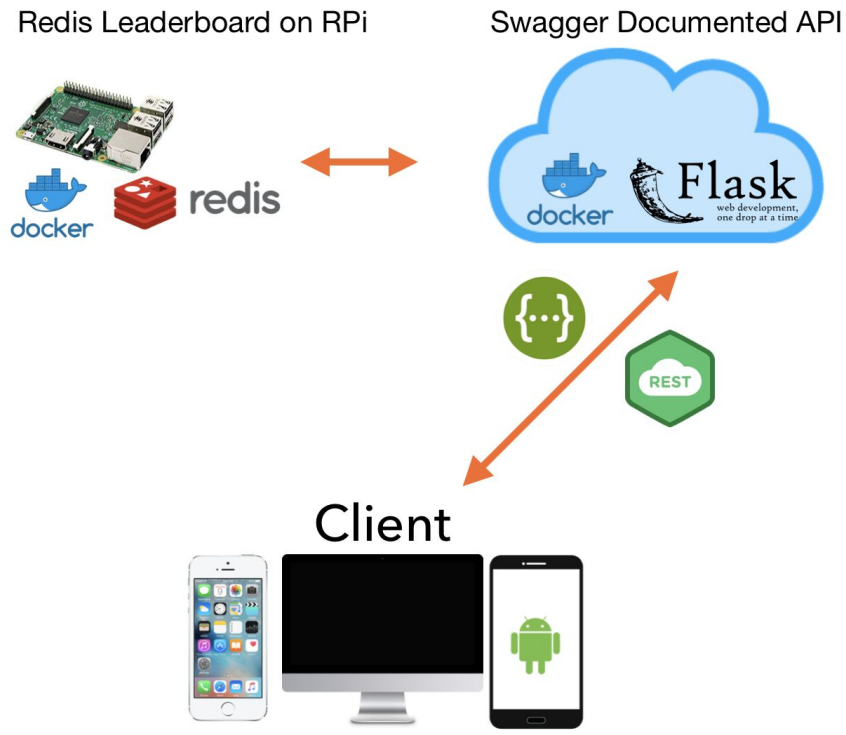
Azure Cache for Redis

Use the client API to interact with Redis

As mentioned earlier, Redis is an in-memory NoSQL database which can be replicated across multiple servers. It is often used as a cache, but can be used as a formal database or even message-broker.

It can store a variety of data types and structures and supports a variety of commands you can issue to retrieve cached data or query information about the cache itself. **Executing commands on the Redis cache**

Typically, a client application will use a client library to form requests and execute commands on a Redis cache. You can get a list of client libraries directly from the Redis clients page.



Azure Cache for Redis

Connecting to your Redis cache with StackExchange.Redis

Recall that we use the host address, port number, and an access key to connect to a Redis server. Azure also offers a connection string for some Redis clients which bundles this data together into a single string.

You can pass this string to StackExchange.Redis to create a connection the server.

Notice that there are two additional parameters at the end:

- **ssl** - ensures that communication is encrypted.
- **abortConnection** - allows a connection to be created even if the server is unavailable at that moment.

Hands-on : Create an open-source Redis Cache

How can we help you?

IntelliPaat Blog

YouTube

India: +91-7022374614

US: 1-800-216-8930 (TOLL FREE)

sales@intellipaat.com

24/7 Chat with Our Course Advisor