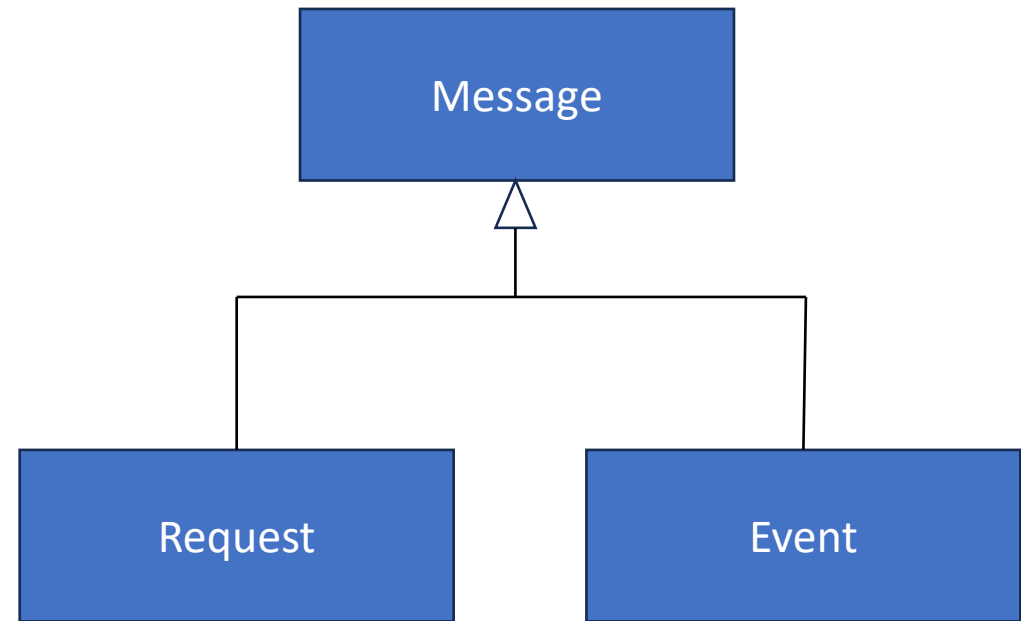


# Agenda

- Some Theory
- PoC description
- PoC demo
- Code used in the PoC
- Constructing the PoC (your turn 😊)
- Debugging tips

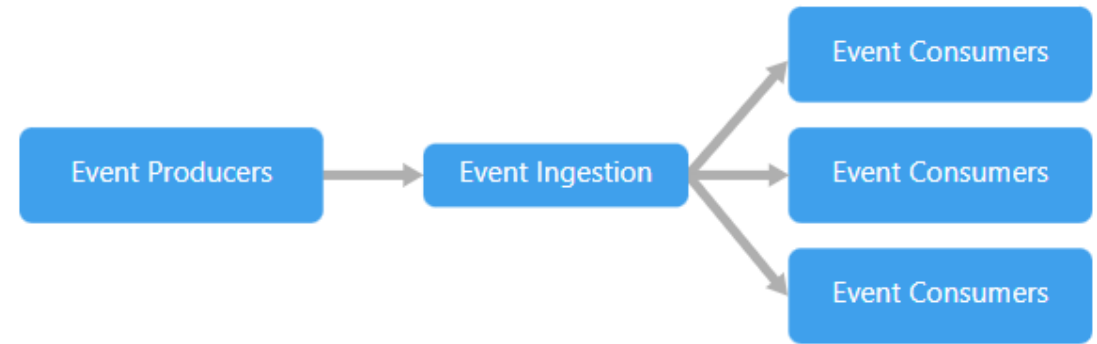
# What is an Event?

- A *Message* is a payload of data sent from one system to another.
- A *Request* is a type of message that asks for an action to be performed (in the future).
  - E.g., “Please process the data”
- An *Event* is a type of message that notifies something has happened (in the past).
  - E.g., “The processing is complete”



# Event-Driven Architecture

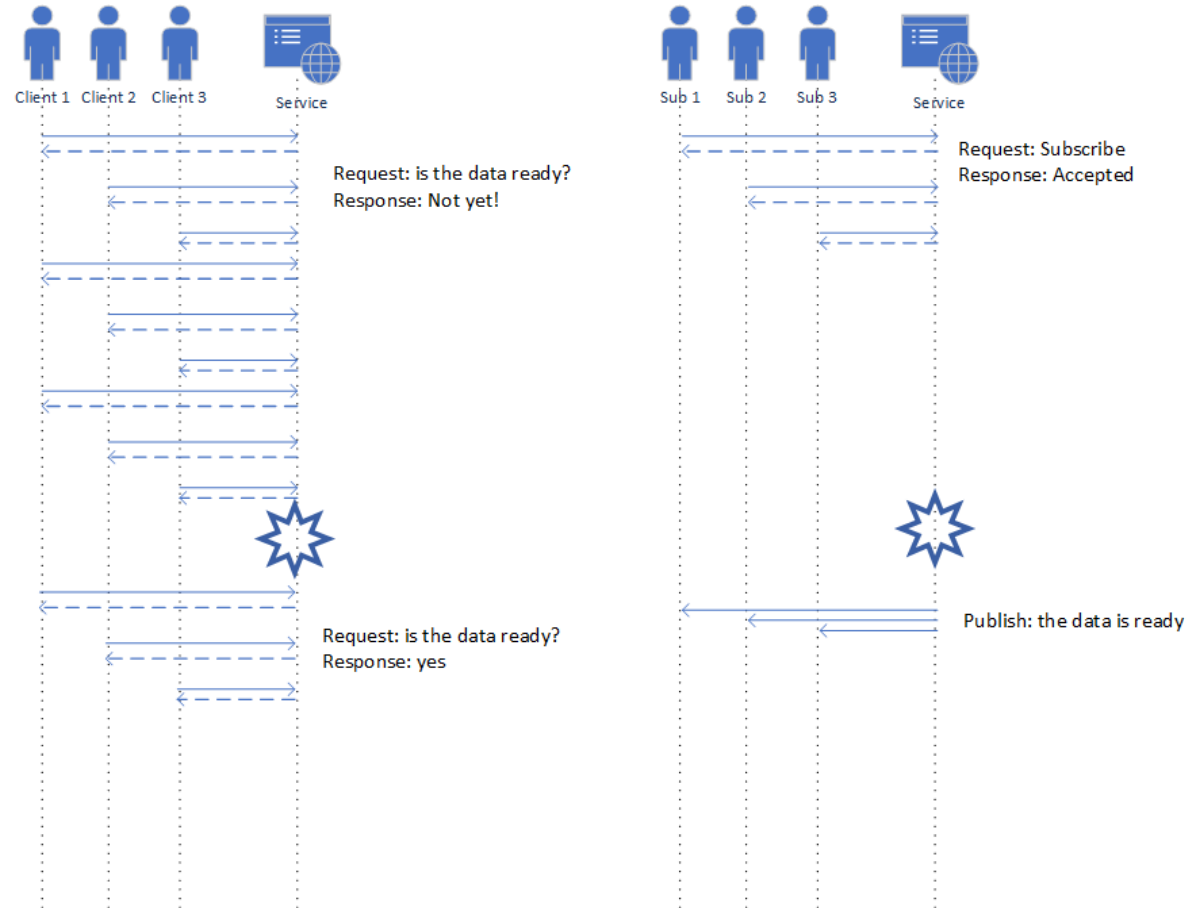
- An event-driven architecture consists of **event producers** that generate a stream of events, and **event consumers** that listen for the events.
- Events are delivered in near real time, so consumers *can* respond immediately to events as they occur.
- Producers are decoupled from consumers — a producer doesn't know which consumers are listening.
- Consumers are also decoupled from each other, and every consumer sees all of the events.



# Publish-Subscribe vs. Event Streaming

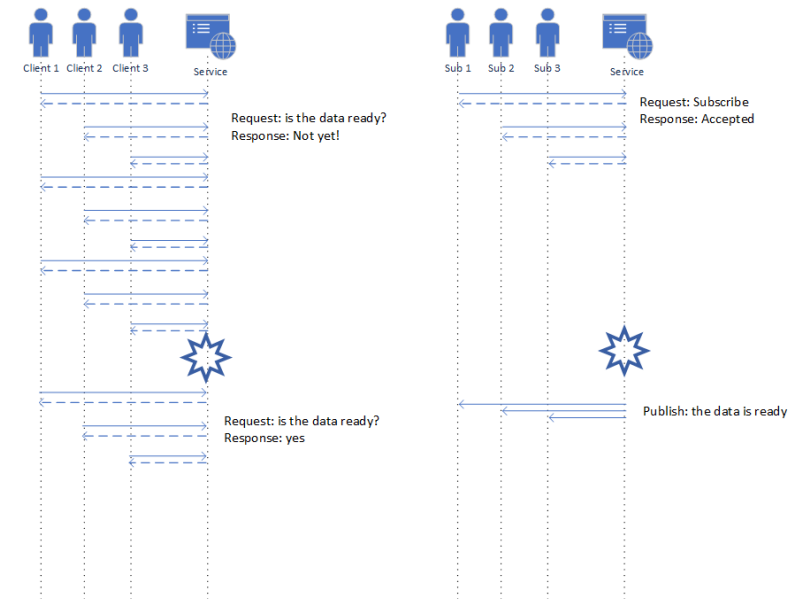
Pub-Sub	Event Streaming
The messaging infrastructure ( <i>Event Ingestion</i> ) keeps track of subscriptions.	Events are written to a log. Events are strictly ordered (within a partition) and durable.
When an event is published, it sends the event to each subscriber ( <i>Consumer</i> ).	Clients ( <i>Consumers</i> ) don't subscribe to the stream, instead a client can read from any part of the stream.
After an event is received, it can't be replayed, and new subscribers don't see the event	The client is responsible for advancing its position in the stream. That means a client can join at any time, and can replay events.

# Scenario: Request-Response vs. Pub-Sub



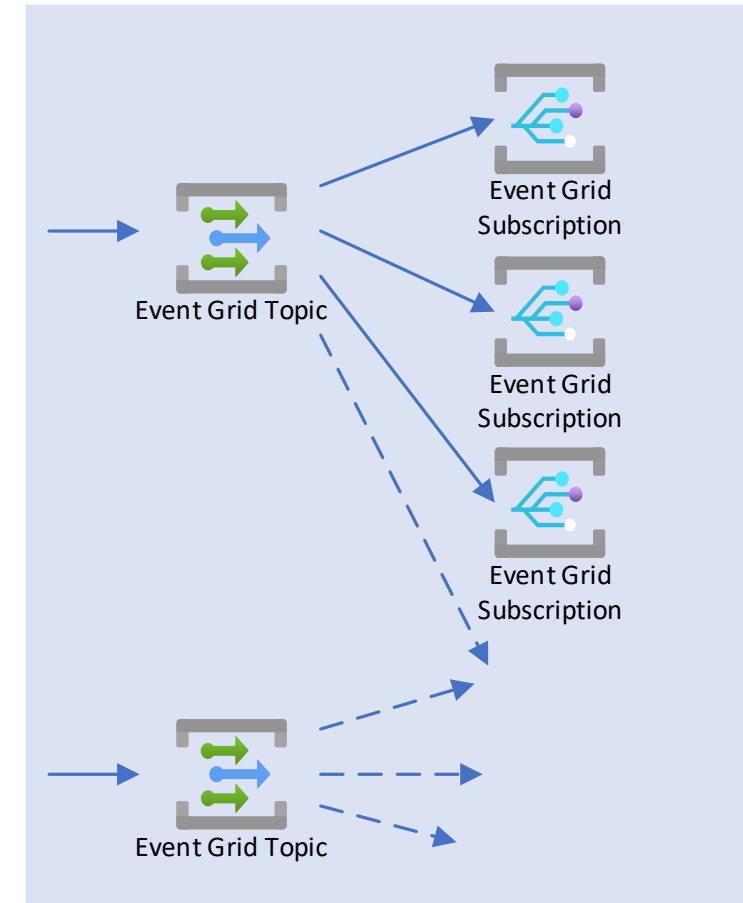
# Proof of Concept

- Simple example of event-driven, pub-sub operation.
- External client to an event-driven system hosted in Azure.
  - Client issues request to subscribe for notification of an event, then fetches results when they are available.
  - Client interactions:
    - POST request to subscribe
    - Receive notification
    - GET the associated data
  - Internally, publish to Event Grid, Subscribers to Topics are notified.



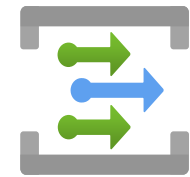
# Event Grid (1/2)

- Event Publishers raise events in Event Grid TOPICS
- Subscribers listen on a SUBSCRIPTION to a TOPIC
  - Event Grid can PUSH events to subscribers
  - (or Subscribers can PULL, June 2023)
- Event Grid filters and routes events
  - Built-in delivery retry mechanism
- Highly-scalable, high throughput (MQTT, HTTP )
- Security: Access Control, Certificates, TLS support



# Event Grid (2/2)

- Many use cases; e.g.
  - Ingest multiple data sources (many-to-one, fan-in)
  - Broadcast alerts (one-to-many, fan-out)
- Many event handlers (subscriber types)
  - Webhook, Azure Function, Event Hub, Service Bus queues and topics, Relay hybrid connections, Storage queues
- “*Pay-per-event*” pricing model
  - First 100,000 operations per month are free
  - Basic tier: £0.467 per million operations
  - “Operations in Event Grid include all ingress events, advanced match, delivery attempt, and management calls. You’re charged per million operations with the first 100,000 operations free each month.”



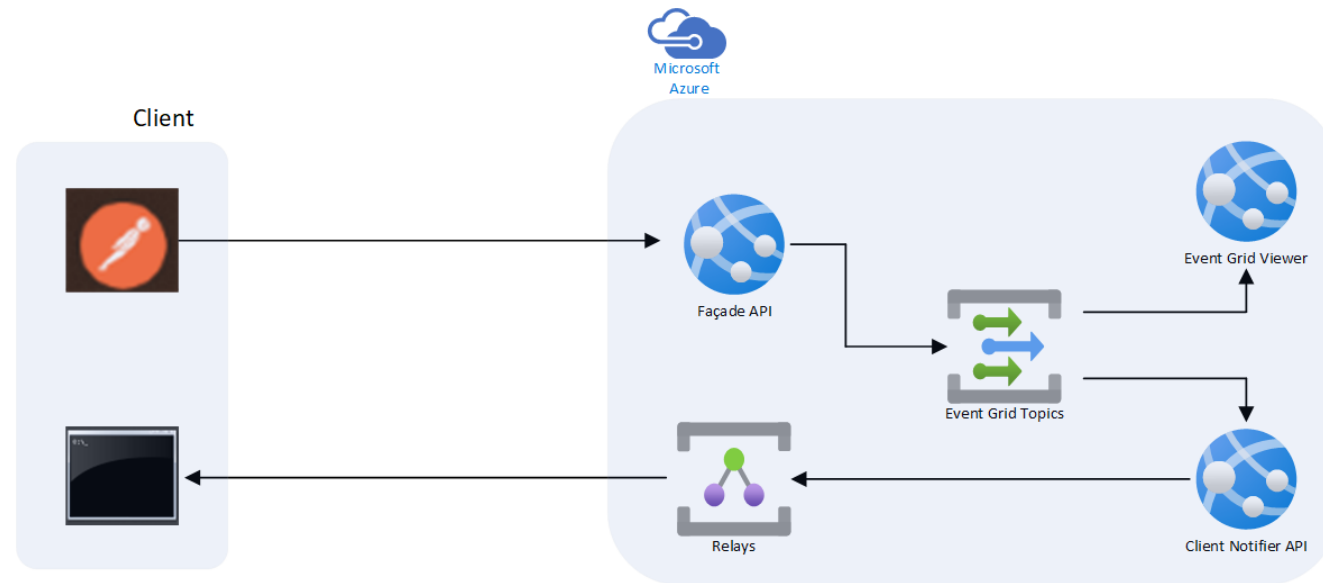
Event Grid Topics



Event Grid  
Subscriptions



# Diagram of PoC components



## Local representation of a Client:

- PostMan (Client-Server request for asynchronous operation)
- AzureRelayListener (Client endpoint subscribing to the operation result)

## Azure:

- Facadeapi (“event source”)
- EventGrid Topic (“publisher”)
- EventGridViewer (“subscriber”)
- ClientNotifierApi (“subscriber”)
- Azure Relay

# User: Request and Notification

Event Driven Example / **FacadePost**

POST https://facadeapiXXXXXXXXXX.azurewebsites.net/api/Facade

Send

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies

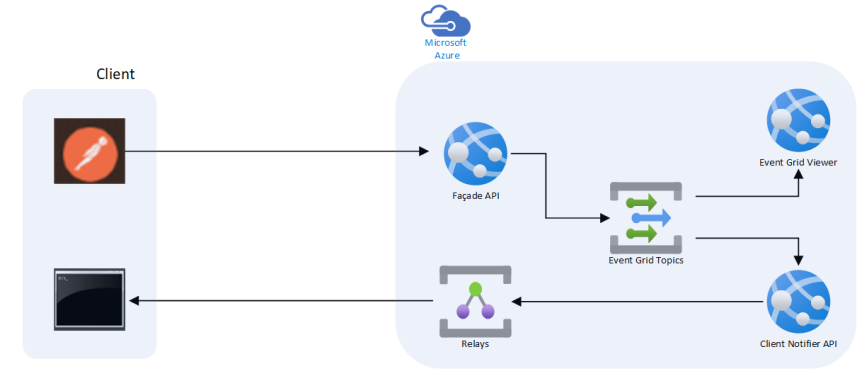
none form-data x-www-form-urlencoded **raw** binary GraphQL JSON Beautify

```
1 "dummyString"
2
```

Body Cookies (2) **Headers (4)** Test Results


Key	Value
Content-Length	0
Date	Fri, 12 Apr 2024 16:33:35 GMT
Server	Kestrel
Location	api/Facade/123456789




Status: 202 Accepted Time: 27 ms Size: 130 B Save as example



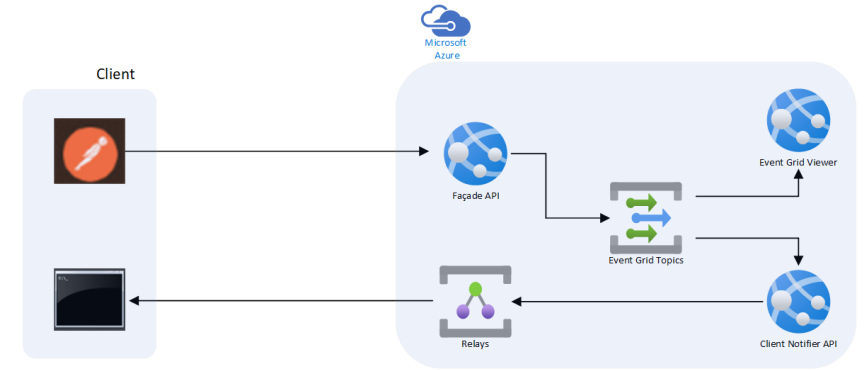
```
Microsoft Visual Studio Debug Console
Online
Server listening
New session
{
  "id": "03a476cb-785d-4a8e-805c-a1567ffef1f",
  "subject": "recordId",
  "data": {
    "RecordId": 123456789,
    "UserString": "dummyString",
    "Hour": 16,
    "Minute": 33,
    "Second": 24
  },
  "eventType": "NewRequest",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2024-04-12T16:33:24.22833Z",
  "topic": "/subscriptions/{resourceGroup}"
}
```

# Internal: Monitoring

 **Azure Event Grid Viewer**

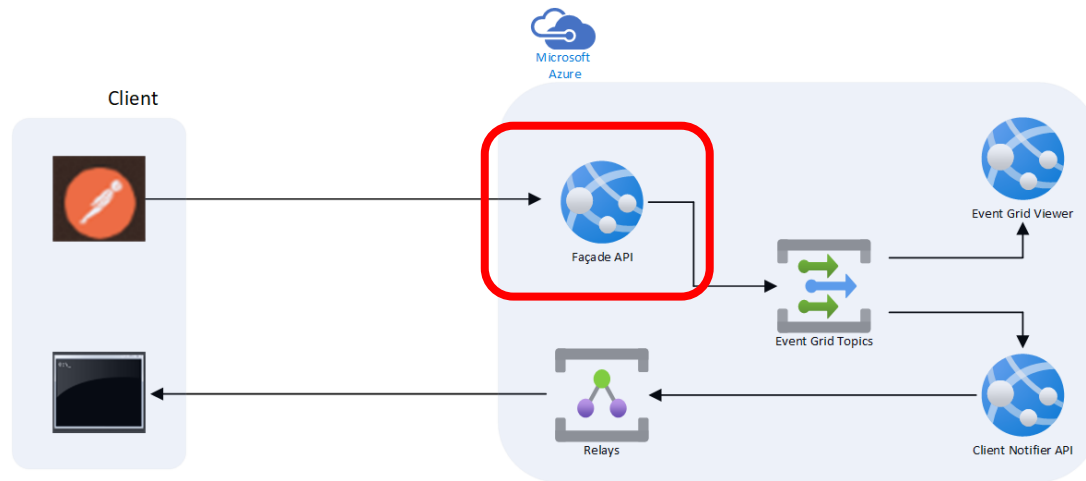
	Event Type	Subject
	NewRequest	recordId
	NewRequest	recordId
	NewRequest	recordId

```
{
  "id": "03a476cb-785d-4a8e-805c-a1567ffef1f",
  "subject": "recordId",
  "data": {
    "RecordId": 123456789,
    "UserString": "dummyString",
    "Hour": 16,
    "Minute": 33,
    "Second": 24
  },
  "eventType": "NewRequest",
  "dataVersion": "1.0",
  "metadataVersion": "1",
  "eventTime": "2024-04-12T16:33:24.22833Z",
  "topic": "/subscriptions/ resourceGroups/adh_eventgrid/providers/Microsoft.EventGrid/topics/adhEventGridTopic"
}
```



(App Insights  
and / or  
live Log Stream)

# Code: Façade API



- Façade API: controller has POST (request) and GET (result) endpoints.
  - POST:
    - Calls Service *ProcessRequest* method to create a payload and post it to an Event Grid Topic
    - Controller returns 202 with the URL to fetch the result (including the recordId) in the header
    - OR, 400 for an empty request body, 500 if the Service reported an error
  - GET:
    - Calls Service *GetRecord* method (hard-coded to return 200 and a Record including the requested ID and current time)
    - Would return 404 if recordId is not found, or 202 if the recordId is found but the results are not yet ready
- Event Grid client functionality provided by EventGridPublisher
  - Class *EventGridClient* constructed with Topic endpoint URI and Topic access key
  - Single method, *PublishEventGridEvent* uses SDK class and method *EventGridPublisherClient.SendEventsAsync*

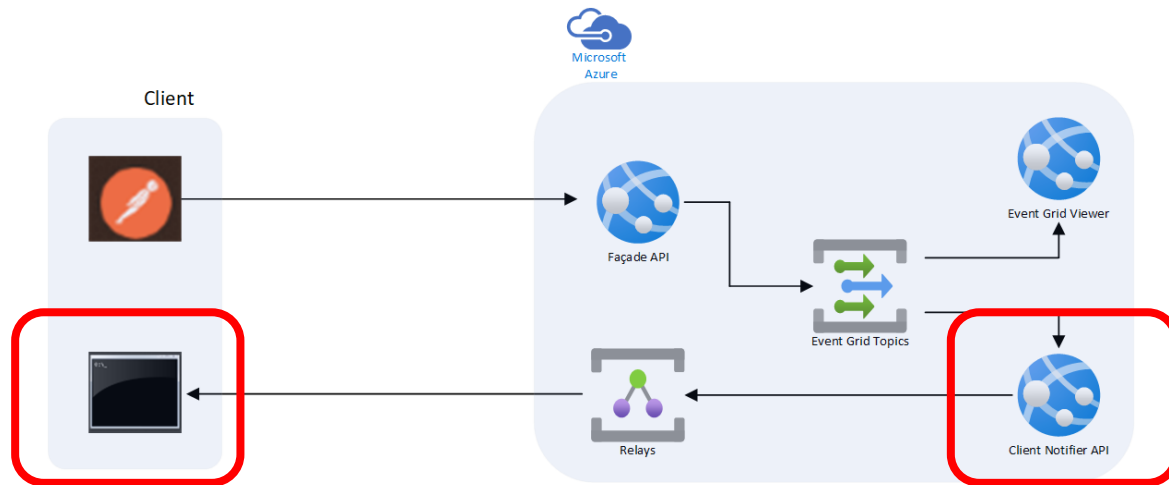
# Code: Azure Relay Listener, Client Notifier API

- `Azure_relay_listener`: console application that registers with, then listens for messages on an Azure Relay Hybrid Connection

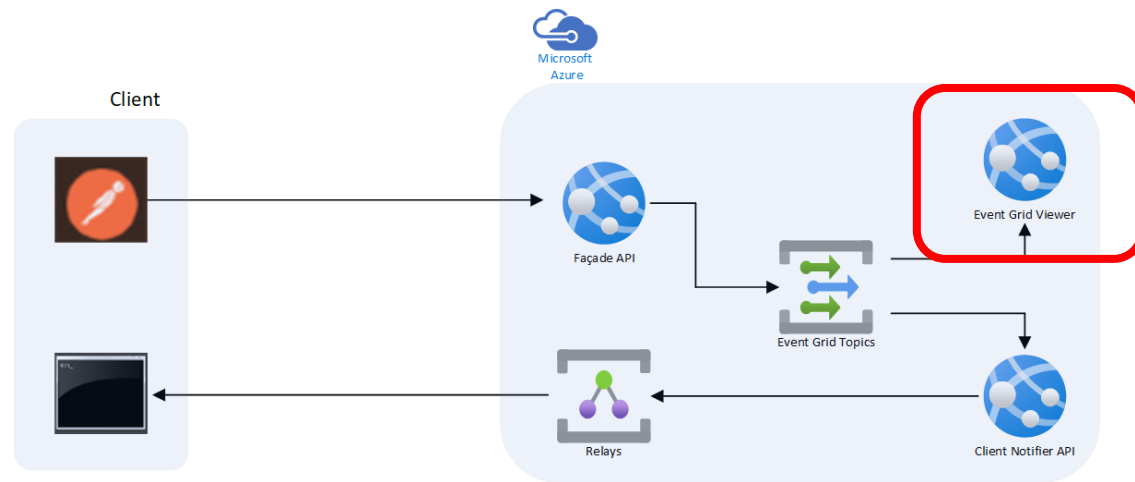
- Echoes back what is received, prefixed with "Echo: "
- Code must be provided with values for *RelayNamespace*, *HybridConnectionName*, *KeyName*, *KeyValue*

- ClientNotifier API: controller written as an Event Grid Subscriber.

- Method *HandleValidation* for confirming connection with Event Grid.
- Method *HandleGridEvents* for events from Event Grid. Parses the payload and sends it Azure Relay (*Namespace*, *Connection Name*, *KeyName*, *Key* to be provided by user)
- Method *HandleCloudEvents* present but not implemented
- 400 returned in any other case.



# Event Grid Viewer (3<sup>rd</sup> party code)



- [Azure-Samples/azure-event-grid-viewer: Live view of events from Azure Event Grid with ASP.NET Core and SignalR \(github.com\)](https://github.com/Azure-Samples/azure-event-grid-viewer)
- App that displays events from Azure Event Grid in near-real time
- *UpdatesController* controller written as an Event Grid Subscriber

Constructing the PoC

# Publish the App Services

- Event Grid Viewer:
  - [Azure-Samples/azure-event-grid-viewer: Live view of events from Azure Event Grid with ASP.NET Core and SignalR \(github.com\)](#)
  - “Deploy to Azure” (use your Azure subscription)
- Façade, Client Notifier APIs
  - Clone this repo
  - Open the ClientNotifierApi and FacadeApi, solutions in Visual Studio
  - Right-click the respective project and publish to your Azure subscription
- Ensure all are running.



# Azure Relay, AzureRelayListener

- Azure Portal: Create a Relay
  - Create a Hybrid Connection
    - Requires Client Auth
  - If not present, create Shared access policy “*RootManageSharedAccessKey*” with all privileges
- Source code:
  - Open *AzureRelayListener* in Visual Studio
  - *RelayNamespace* = Relay name
  - *ConnectionName* = Hybrid Connection name
  - *SASKeyName* = *RootManageSharedAccessKey*
  - *Key* = Primary Key Value of the above
  - (\*\*Note these values\*\*)
  - Build and Run

```
7  public class Program
8  {
9      private const string RelayNamespace = "{RelayNamespace}.servicebus.windows.net";
10     private const string ConnectionName = "{HybridConnectionName}";
11     private const string KeyName = "{SASKeyName}";
12     private const string Key = "{SASKey}";
13
14     public static void Main(string[] args)
15     {
16         RunAsync().GetAwaiter().GetResult();
17     }
```

# Event Grid Topic – part 1

- From your Resource Group, search Azure Portal for “Event Grid Topic”
  - Specify Name, Region only (leave other settings with default values)
- Create a Subscription for Event Grid Viewer (default values, except the following)
  - Enter a Name
  - Event Schema = Event Grid Schema
  - Endpoint Type = Web Hook
    - “Configure an endpoint” – enter https:// followed by the FQDN for the Event Grid Viewer App Service, followed by /api/updates
- *(Creation will fail if the above settings are incorrect or the app has not started)*

# Set the App Service environment variables

- Façade API, App Settings (to write to Event Grid Topic)
  - *EventGridTopicEndpoint* = <Your Topic Endpoint>
  - *EventGridTopicKey* = value for Key 1 or Key 2 (at Topic, Access keys)
- ClientNotifier API, App Settings (to write to Relay)
  - *RelayNamespace* = <Relay name>.servicebus.windows.net
  - *ConnectionName* = Hybrid Connection name
  - *KeyName* = *RootManageSharedAccessKey*
  - *Key* = Primary Key Value of the above

# Event Grid Topic – part 2

- Create a Subscription for Client Notifier API (default values, except the following)
  - Enter a Name
  - Event Schema = Event Grid Schema
  - Endpoint Type = Web Hook
    - “Configure an endpoint” – enter https:// followed by the FQDN for the Client Notifier API App Service, followed by /ClientNotifier
- *(Creation will fail if the above settings are incorrect or the API has not started)*

# PostMan

- Download if not already installed
  - <https://www.postman.com/downloads/>
- Get the FQDN of the Facade Api
- Create a POST query
  - For the URL, specify <https://>, Facade Api FQDN, and append /api/Facade
  - For “Body”, select “raw” and “JSON”
  - Enter your chosen string as the payload
  - Send ... and expect
    - A 202 Response, with Location api/Facade/123456789 specified in the Response header
    - A row to appear on the Event Grid Viewer
    - Output on the Azure Relay Listener console

# Debug:

## Azure Relay Listener configuration

- Azure Relay Listener errors in configuration
  - Asserts at initialisation
- Error messages:
  - RelayNamespace => Unable to connect to remote server
  - ConnectionName => Endpoint does not exist
  - KeyName => Invalid signature
  - Key => Invalid signature

# Debug:

## Event Grid Topic Subscription configuration

- Errors in specifying the Web Hook
  - Asserts at initialisation / configuration validation
- Two types of error
  - Failure to put <https://> before the FQDN
    - Immediately, error indicated on the UI “Only HTTPS web hooks are supported as endpoints.”
  - Incomplete or erroneous path to the endpoint
    - Failed validation error indicated on the UI, “Webhook validation handshake failed”

# Debug:

## Facade API configuration for Event Grid Topic

- For erroneous values of either, or both of the below no activity will be evident on Event Grid Viewer
  - EventGridTopicEndpoint
  - EventGridTopicKey
- A full implementation of the code would include logging (skipped for this PoC).



# Debug:

## Client Notifier API configuration for Azure Relay

- For erroneous values of any of the below no activity will be evident on Azure Relay Listener.
- Error messages on the “Log Stream” output:
  - RelayNamespace – “Endpoint does not exist”
  - ConnectionName “No such host is known”
  - KeyName “The token has an invalid signature”
  - Key “The token has an invalid signature”

# Event Grid Topic Retries

- “Event Grid provides durable delivery. It tries to deliver each message **at least once** for each matching subscription immediately. If a subscriber's endpoint doesn't acknowledge receipt of an event or if there's a failure, Event Grid retries delivery based on a fixed [retry schedule](#) and [retry policy](#).”
- ([Azure Event Grid delivery and retry - Azure Event Grid | Microsoft Learn](#))
- This PoC used HTTP for sending events: the Client Notifier API returned a status code of 200 for success, and 400 for a bad request
  - Event Grid Topic will not retry on receipt of 400, but will for a 5XX error
  - Logs can exhibit the same event being sent multiple times if the subscriber returns a 5XX error message.