**About this library:**

This is a reusable component for EventBus architecture, developed in dot net core (V6.0) and dot net standard (V2.1) and targeted for the Dot Net developer’s community.

Some of the widely used messaging broker (**RabbitMQ, MQTT, Azure Service Bus, Azure Storage Queue, Redis DB, Kafka)** implementations are provided and developers can follow the example and documentation to implement additional brokers if required.

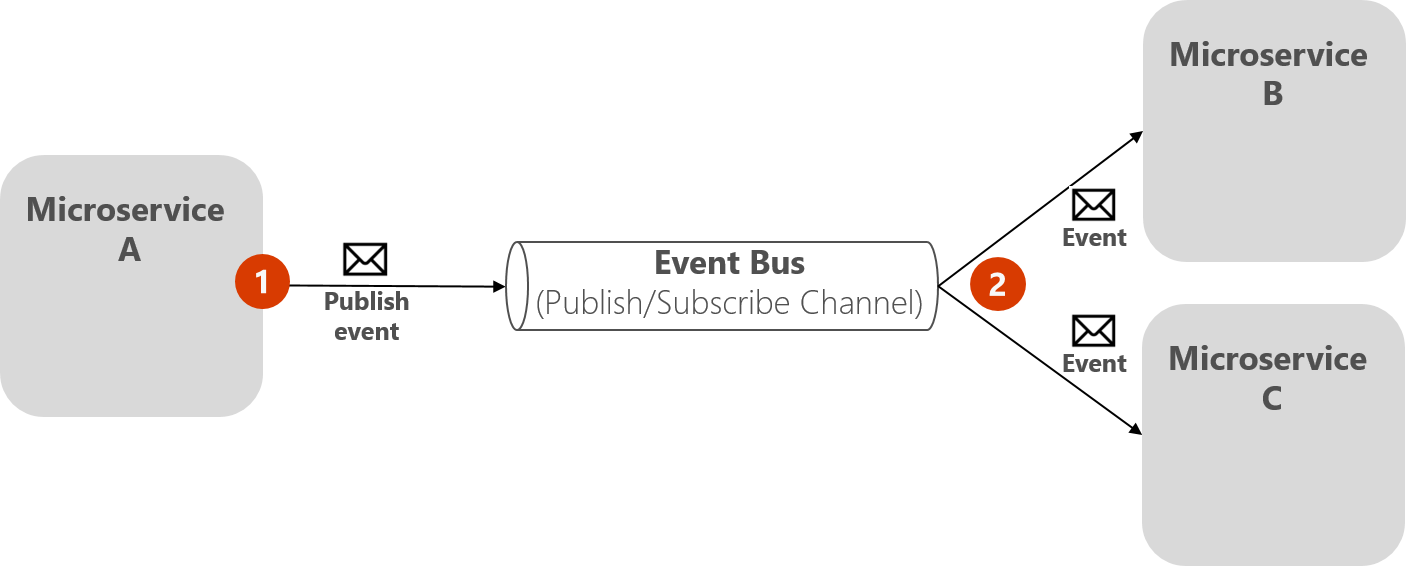
The technical documentation and example samples are provided for the developers to quick start with it with minimal time to be spent on using and learning this library.

All technical documentation can be found in the Doc folder and can be accessed by clicking the Index.html page.

**Why this library?**

* Out of the box support to ingest and consume data from popular event brokers like (RabbitMQ, Kafka, Mqtt etc.)
* Easy to integrate and implement EventBus pattern in the microservice projects. Publish-subscribe based mechanism.
* Developers can publish and handle the events through dedicated event handlers defined by them.
* Resiliency built-in, i.e., all possible error conditions are handled using Polly Dot Net library.
* Examples and demo projects given for easy integration and understanding.

**Architecture:**



Mqtt

Kafka

RabbitMQ

**How to use this library (with 3 simple steps)?**

* **Step 1**

**Add the service configuration in the dotnet core app as mentioned below, It may vary based on the broker we are using, below is an example of the Mqtt Broker implementation**

//Mqtt

services.AddSingleton<IMqttConnection>(serviceProvider =>

{

return new MqttConnection(Log.Logger, “127.0.0.1”);

});

services.AddSingleton<IEventBus, MqttEventBus>(serviceProvider =>

{

var mqttPersistentConnection = serviceProvider.GetRequiredService<IMqttConnection>();

var scopeFactory = serviceProvider.GetRequiredService<IServiceScopeFactory>();

var eventBusSubcriptionsManager = serviceProvider.GetRequiredService<IEventBusSubscriptionManager>();

return new MqttEventBus(mqttPersistentConnection, eventBusSubcriptionsManager, scopeFactory, Log.Logger);

});

* **Step 2**

Subscribe to the event if required.

//Subscribe

var eventBus = services.BuildServiceProvider().GetRequiredService<IEventBus>();

eventBus.Subscribe<EventOne, EventHandlerOne>();

* **Step 3**

Handle the event, for that add an event handler.

//handler

services.AddScoped<EventHandlerOne>();

**Sample Code below (Also included in the project):**

using DASS.AI.EventBus.Abstraction.Bus;

using DASS.AI.EventBus.Abstraction.SubscriptionManager;

using DASS.AI.EventBus.Mqtt;

using DemoEventsAndHandlers;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.Extensions.Hosting;

using Serilog;

namespace DemoPublisher

{

sealed class Program

{

private Program()

{ }

public static void Main(string[] args)

{

CreateHostBuilder(args).Build().Run();

}

public static IHostBuilder CreateHostBuilder(string[] args)

{

return Host.CreateDefaultBuilder(args)

.UseWindowsService().

ConfigureServices((hostContext, services) =>

{

Log.Logger = new LoggerConfiguration()

.Enrich.FromLogContext()

.WriteTo.Console()

.CreateLogger();

//services.AddSingleton<IRabbitMQConnection>(serviceProvider =>

//{

// return new RabbitMQConnection(Log.Logger);

//});

//services.AddSingleton<IEventBus, RabbitMQEventBus>(serviceProvider =>

//{

// var rabbitMQPersistentConnection = serviceProvider.GetRequiredService<IRabbitMQConnection>();

// var scopeFactory = serviceProvider.GetRequiredService<IServiceScopeFactory>();

// var eventBusSubcriptionsManager = serviceProvider.GetRequiredService<IEventBusSubscriptionManager>();

// return new RabbitMQEventBus(rabbitMQPersistentConnection, "PublisherQueue", eventBusSubcriptionsManager, scopeFactory, Log.Logger);

//});

//Mqtt

services.AddSingleton<IMqttConnection>(serviceProvider =>

{

return new MqttConnection(Log.Logger);

});

services.AddSingleton<IEventBus, MqttEventBus>(serviceProvider =>

{

var mqttPersistentConnection = serviceProvider.GetRequiredService<IMqttConnection>();

var scopeFactory = serviceProvider.GetRequiredService<IServiceScopeFactory>();

var eventBusSubcriptionsManager = serviceProvider.GetRequiredService<IEventBusSubscriptionManager>();

return new MqttEventBus(mqttPersistentConnection, eventBusSubcriptionsManager, scopeFactory, Log.Logger);

});

//Redis

//services.AddSingleton<IRedisConnection>(serviceProvider =>

//{

// return new RedisConnection("localhost:6379");

//});

//services.AddSingleton<IEventBus, RedisEventBus>(serviceProvider =>

//{

// var redisPersistentConnection = serviceProvider.GetRequiredService<IRedisConnection>();

// var scopeFactory = serviceProvider.GetRequiredService<IServiceScopeFactory>();

// var eventBusSubcriptionsManager = serviceProvider.GetRequiredService<IEventBusSubscriptionManager>();

// return new RedisEventBus(redisPersistentConnection, eventBusSubcriptionsManager, scopeFactory, Log.Logger);

//});

//services.AddSingleton<IAzureStorageQueueConnection>(serviceProvider =>

//{

// string storageConnectionString = "DefaultEndpointsProtocol=https;AccountName=sukantatest2566474934;AccountKey=J3UhQ/CMOn4

//});

services.AddSingleton<IEventBusSubscriptionManager, EventBusSubscriptionManager>();

services.AddHostedService<PublisherService>();

//handler

services.AddScoped<EventHandlerOne>();

//Subscribe

var eventBus = services.BuildServiceProvider().GetRequiredService<IEventBus>();

eventBus.Subscribe<EventOne, EventHandlerOne>();

});

}

}

}