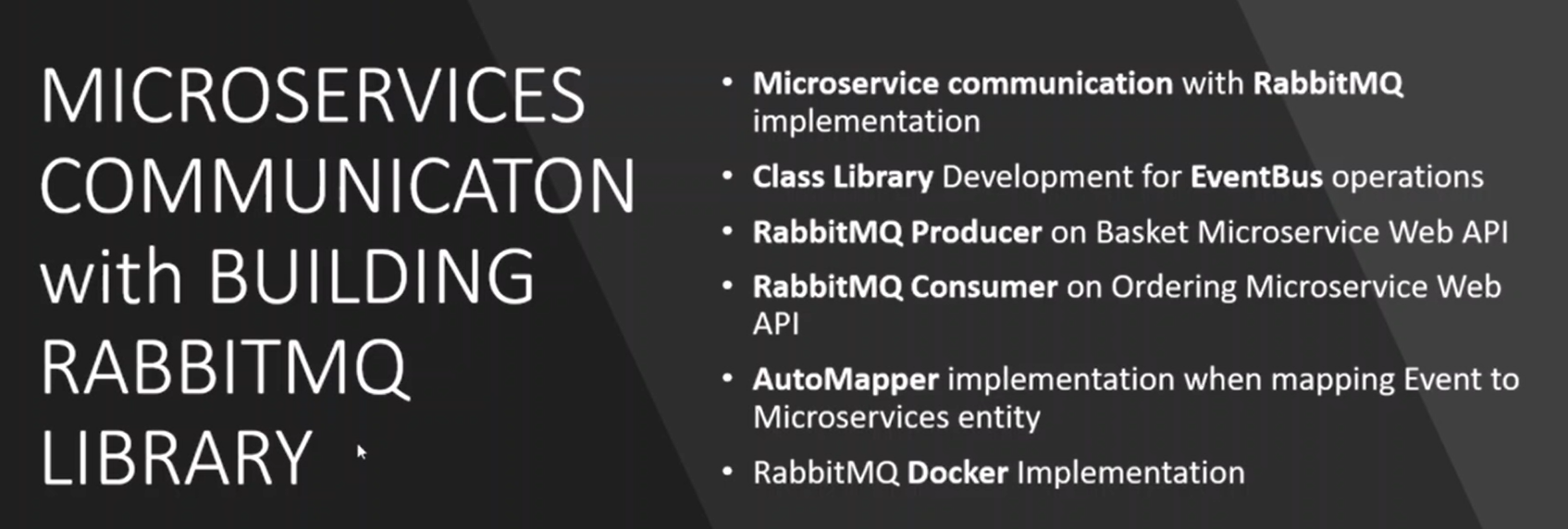
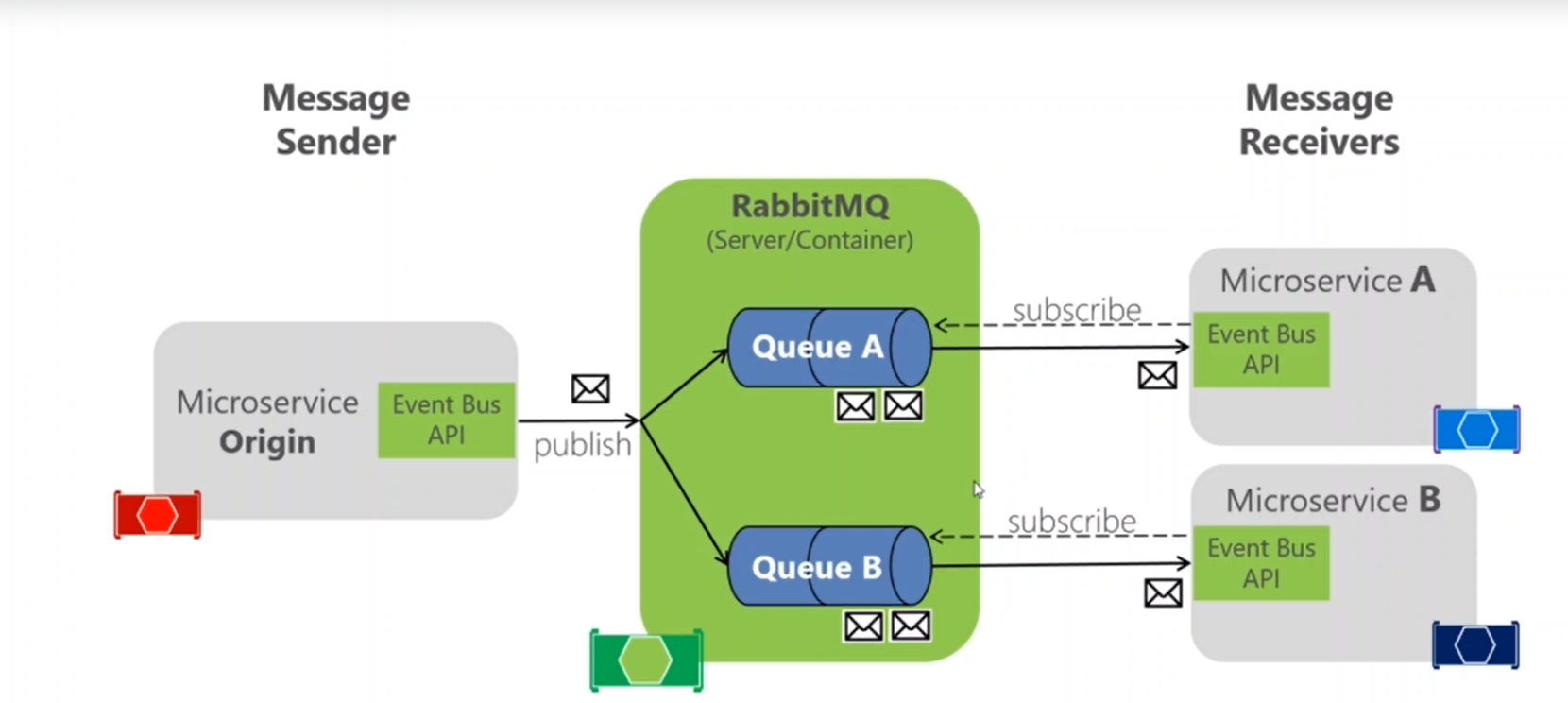
**Section04 Micro Service communication with building Rabbit MQ library**

**Notes:-**

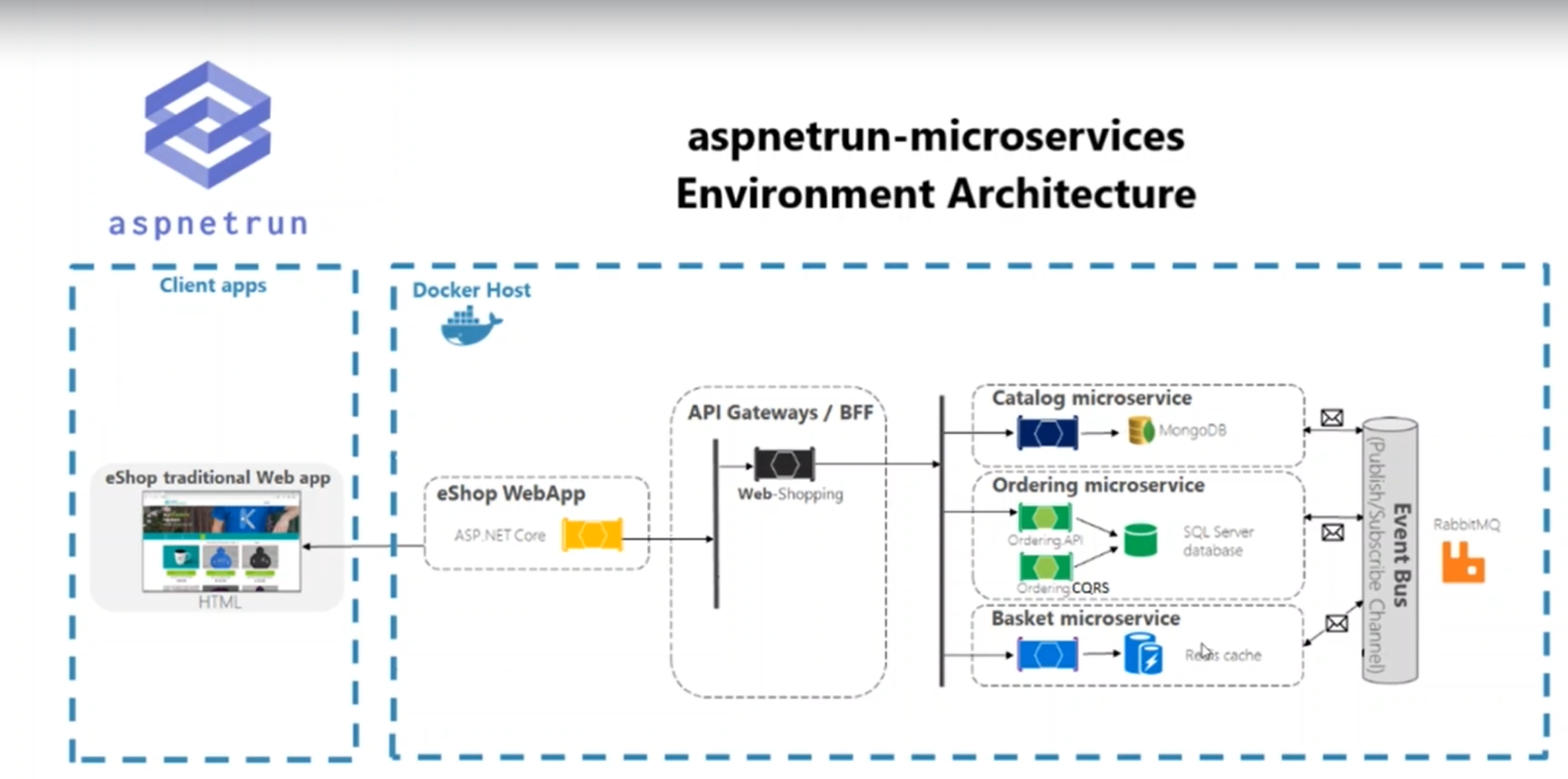


**Class library for event bus operation represent the producer and consumer for the service bus**

**Lesson01 Rabbit MQ architecture**

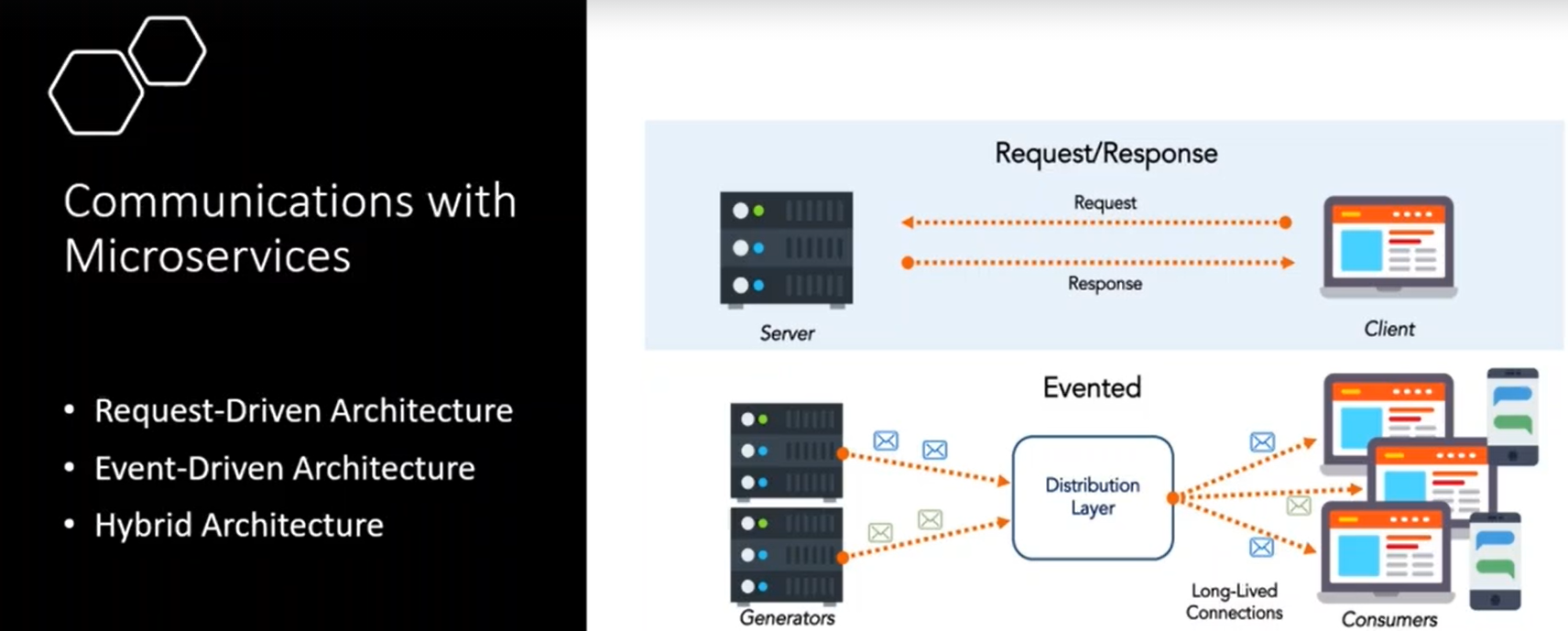
**Notes:-**

**1-Rabbit MQ is using event-bus class library for the consumer side**



**Lesson02 Communications with Micro services**

**Notes:-**



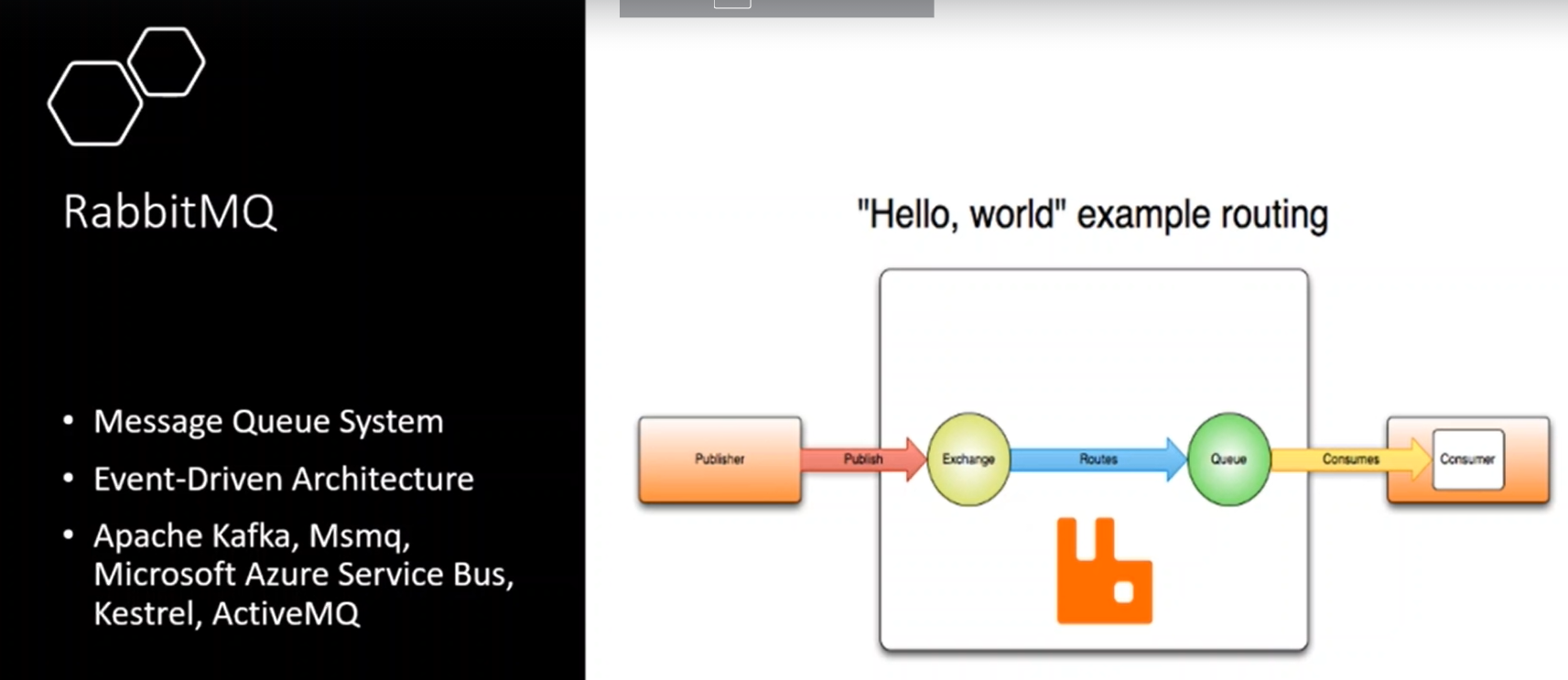
**1-The recommended way to communicate between micro services with using event-driven architecture which using Rabbit MQ with publisher and subscriber for the queue in the service bus**

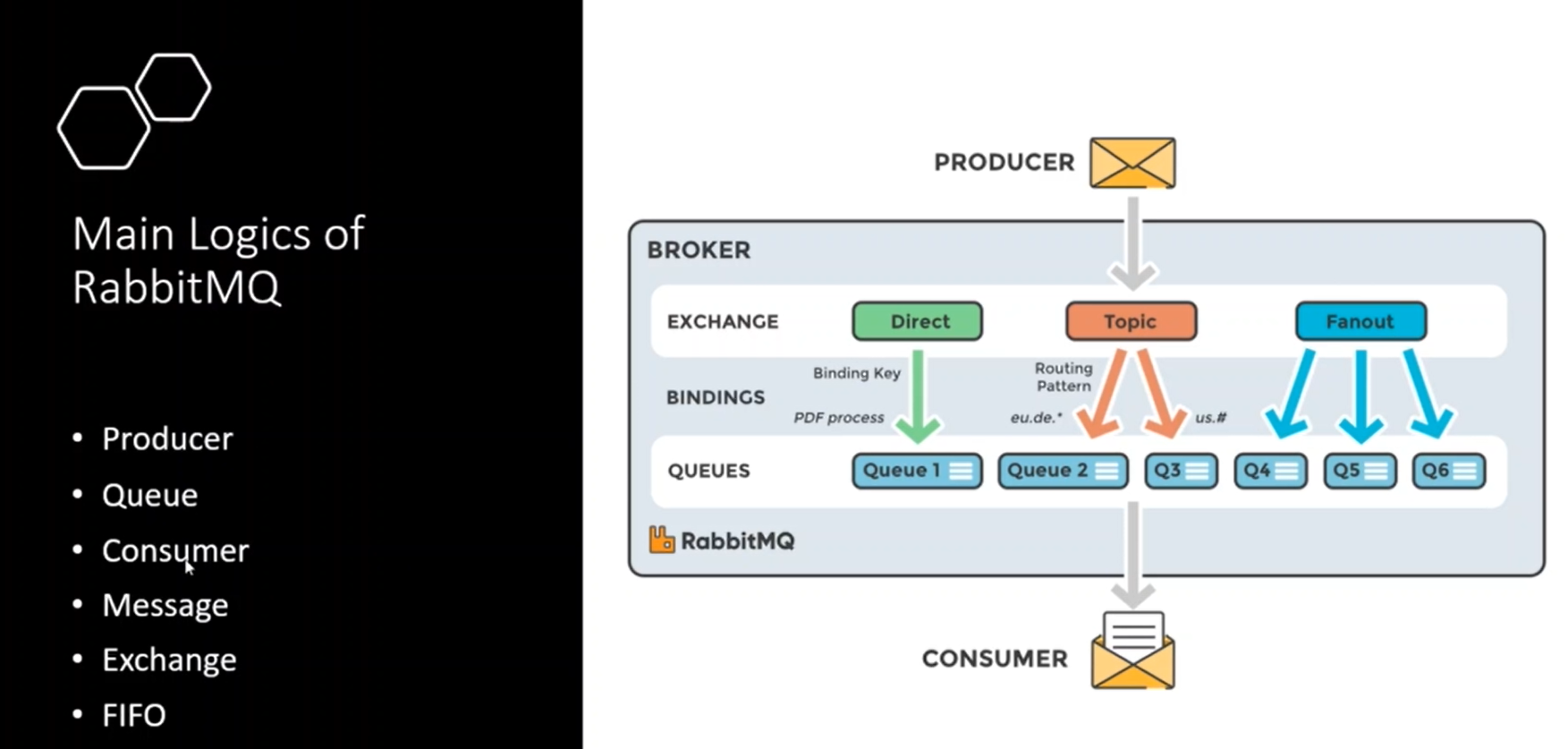
**2-It’s also ensure the data integrity and the data in meaningful state , so you can following the data states with looking for the event history.**

**3-Request-driven architecture between SPA and the microservices or between the micro services but its stateless which means you can keep track of the request sent to the micro service**

**Lesson03 what is the RabbitMQ**

**Notes:-**





**FIFO (first in first out which RabbitMQ working in this algorithim)**

**Producer: which produce the message like BasketAPI**

**Consumer: which consume the message like OrderingAPI**

**Queue: is the store for the message**

**Message: is the data we send the message**

**Exchange: is the structure which queue to send the message based on the routing keys**

**Exchange Types  
Types:-**

**1-Direct: which directly send the message to the queue without detect which subscriber consume the message**

**2-Topic: which depend on the public subscriber pattern, which you have to detect the subscriber before send message to the service bus, so this message is only consumed by this consumer only**

**3-Fanout: which using broadcasting pattern which send the same message to all queues**

**Steps:-**

**0-install the queue Client package library on the class library as below**

**1-create class library project called EventBusRabbitMQ with create the following**

**A-Common > EventBusConstants.cs**

**//which declare the constant value such as the queue name**

**namespace EventBusRabbitMQ.Common{**

**public static class EventBusConstants{**

**public const string BasketCheckoutQueue = "basketCheckoutQueue";}}**

**B-Events > BasketCheckoutEvent.cs**

**//which declare the class equalivant of the BasketCheckout.cs**

**public class BasketCheckoutEvent{**

**//RequestId represnet the request generated and send to the RabbitMQ**

**public Guid RequestId { get; set; }**

**public string UserName { get; set; }**

**public decimal TotalPrice { get; set; }**

**//BilingAddress**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string EmailAddress { get; set; }**

**public string AddressLine { get; set; }**

**public string Country { get; set; }**

**public string State { get; set; }**

**public string ZipCode { get; set; }**

**//Payment**

**public string CardName { get; set; }**

**public string CardNumber { get; set; }**

**public string Expiration { get; set; }**

**public string CVV { get; set; }**

**public int PaymentMethod { get; set; }}**

**2-create interface called IRabbitMQConnection.cs**

**//this interface contains the main functionality of the RabbitMQ Connection like //TryConnect() which create connection and CreateModel which create queue and apply //operation and so on**

**public interface IRabbitMQConnection : IDisposable{**

**bool IsConnected { get; }**

**bool TryConnect();**

**//its perform create queue and perform queue operations after apply connect**

**IModel CreateModel();}**

**3-create class RabbitMQConnection.cs which inherit from the interface**

**public class RabbitMQConnection : IRabbitMQConnection{**

**//provide the operation of create connection and close connection also**

**private readonly IConnectionFactory \_connectionFactory;**

**//provide the created channel that communicate with RabbitMQ**

**private IConnection \_connection;**

**//flag to check if channel disposed or not**

**private bool \_disposed;**

**//we initialzie the constructor to check if the Microservice connect to RabbitMQ or not**

**public RabbitMQConnection(IConnectionFactory connectoinFactory){**

**\_connectionFactory = connectoinFactory ?? throw new ArgumentNullException(nameof(connectoinFactory));**

**if (!IsConnected){TryConnect();}}**

**//define custom property to check if connection is not null and is opend and not dispoed**

**public bool IsConnected{**

**get{return \_connection != null && \_connection.IsOpen && !\_disposed;}}**

**//to make try create connection if failed it will retry reconnect**

**public bool TryConnect(){**

**try{\_connection = \_connectionFactory.CreateConnection();}**

**catch (BrokerUnreachableException){**

**Thread.Sleep(2000);**

**\_connection = \_connectionFactory.CreateConnection();}**

**if (IsConnected)**

**return true;**

**else**

**return false;}**

**//IModel refer to the RabbitMQ to perform create fresh channel under the connection created with session to apply operation**

**public IModel CreateModel(){**

**if (!IsConnected){throw new InvalidOperationException("No rabbit connection");}**

**return \_connection.CreateModel();}**

**//dispose connect to the created channel**

**public void Dispose(){**

**if (\_disposed)**

**return;**

**try{\_connection.Dispose();}**

**catch{throw;}}}**

**4-create Producer > EventBusRabbitMQProducer.cs**

**public class EventBusRabbitMQProducer{**

**private readonly IRabbitMQConnection \_connection;**

**public EventBusRabbitMQProducer(IRabbitMQConnection connection){**

**\_connection = connection ?? throw new ArgumentNullException(nameof(connection));}**

**//to publish the event to the queue**

**public void PublishBasketCheckout(string queueName, BasketCheckoutEvent publishModel){**

**using (var channel = \_connection.CreateModel()){**

**//configure the channel with the connection**

**//durable : determine store value in the memory or Physical Queue**

**//execlusive : give you permission to use this queue of its other connection**

**//auto Delete : if set to true it will delete the message automatically after period of time**

**channel.QueueDeclare(queue:queueName,durable:false,exclusive:false,autoDelete:false,arguments:null);**

**var message = JsonConvert.SerializeObject(publishModel);**

**var body = Encoding.UTF8.GetBytes(message);**

**//set basic property for the channel**

**IBasicProperties props = channel.CreateBasicProperties();**

**props.Persistent = true;**

**props.DeliveryMode = 2;**

**//sending part**

**channel.ConfirmSelect();**

**//send the message to the queue which using basic type of exchange which send the message directly to the queue**

**channel.BasicPublish(exchange: "", routingKey: queueName, mandatory: true, basicProperties: props, body: body);**

**channel.WaitForConfirmsOrDie();**

**//Act Part**

**channel.BasicAcks += (sender, eventArgs) =>{Console.WriteLine("Sent RabbitMQ");};**

**channel.ConfirmSelect();}}}**

**5-on the basketAPI > startup.cs we register the EventBusRabbitMQProducer as below**

**public void ConfigureServices(IServiceCollection services){**

**//we register the auto mapper**

**services.AddAutoMapper(typeof(Startup));**

**//we register the EventBusRabbitMQProducer**

**services.AddSingleton<EventBusRabbitMQProducer>();}**

**6-on the Basket.API install the package AutoMapper**

**7-on the Basket.API > Entities > create class BasketCheckout**

**//we duplicate the class as the BasketCheckoutEvent without RequestId**

**public class BasketCheckout{**

**public string UserName { get; set; }**

**public decimal TotalPrice { get; set; }**

**//BilingAddress**

**public string FirstName { get; set; }**

**public string LastName { get; set; }**

**public string EmailAddress { get; set; }**

**public string AddressLine { get; set; }**

**public string Country { get; set; }**

**public string State { get; set; }**

**public string ZipCode { get; set; }**

**//Payment**

**public string CardName { get; set; }**

**public string CardNumber { get; set; }**

**public string Expiration { get; set; }**

**public string CVV { get; set; }**

**public int PaymentMethod { get; set; }}**

**8-on the Basket.API > Mapping > BasketMapping to apply mapping**

**public class BasketMapping : Profile{**

**public BasketMapping(){CreateMap<BasketCheckout, BasketCheckoutEvent>().ReverseMap();}}**

**9-on the BasketController.cs we ineject the IMapper and EventBusRabbitMQProducer that inject into the startup.cs**

**public class BasketController : ControllerBase{**

**//apply DI of the IMapper on the BasketController**

**private readonly IBasketRepository \_repository;**

**private readonly IMapper \_mapper;**

**private readonly EventBusRabbitMQProducer \_eventBus;**

**public BasketController(IBasketRepository repository,IMapper mapper, EventBusRabbitMQProducer eventBus){**

**\_repository = repository ?? throw new ArgumentNullException(nameof(repository));**

**\_mapper = mapper ?? throw new ArgumentNullException(nameof(mapper));**

**\_eventBus = eventBus ?? throw new ArgumentNullException(nameof(eventBus));}**

**[Route("[action]")]**

**[HttpPost]**

**[ProducesResponseType((int)HttpStatusCode.Accepted)]**

**[ProducesResponseType((int)HttpStatusCode.BadRequest)]**

**public async Task<IActionResult> Checkout([FromBody] BasketCheckout basketCheckout){**

**//get total price of basket**

**//remove the basket**

**//send checkout event to rabbitmq**

**var basket = await \_repository.GetBasket(basketCheckout.UserName);**

**if (basket is null)**

**return BadRequest();**

**var basketRemoved = await \_repository.DeleteBasket(basket.UserName);**

**if (!basketRemoved){return BadRequest();}**

**//we will mapp the basketCheckout object to BasketCheckout Event by using Auto Mapper**

**var eventMessage = \_mapper.Map<BasketCheckoutEvent>(basketCheckout);**

**eventMessage.RequestId = Guid.NewGuid();**

**eventMessage.TotalPrice = basket.TotalPrice;**

**try{\_eventBus.PublishBasketCheckout(EventBusConstants.BasketCheckoutQueue,**

**eventMessage);}**

**catch{throw;}**

**return Accepted();}}**