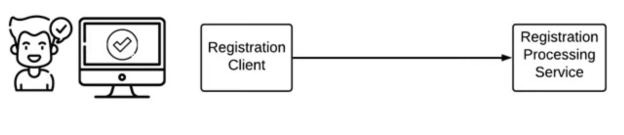
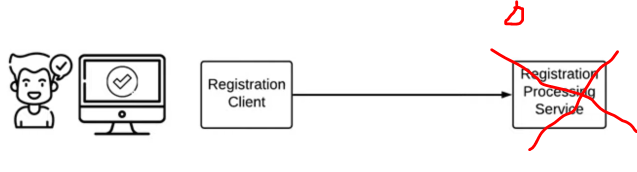
**Event Bus Or Event Hub :: Event driven Architecture**

**What is Request based Architecture ?**

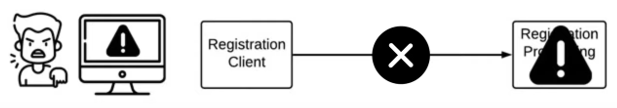
1. Client send request to sever and get response like in any asp.net /JSP/PHP application



1. Service 1 send request and get response from Service 2 ( sync /Async )
2. Now lets Consider below scenario , Registration processing service is down



Then will not get response & our “ Request will be lost”



1. But if our application is not critical for sequence of information / vigilant then there is

Nothing wrong in this approach.

Because at same time we can response to client that “Something wrong “ ,Please process

Again .

1. Now think our application is Healthcare / Ecom / Payment gateway application , then sequence of process should be capture in such a way that we can get information (when /what /why) happened. Which is not possible in Request based architecture.

Lets understand with Health care sector

1. Health care system is hosted on cloud in distributed environment.
2. There are different application which consume this system data like Patient App etc .
3. There is one system IOT which is installed in Hospital
4. Now suppose there is a patient “A” ‘s Blood pressure critically low then IOT system send

Request to Heath care system or Follow-up & Monitoring system( Handle by Nurse/doctor).

1. But If Follow-up & Monitoring system( Handle by Nurse/doctor) is not sending any request to Health care system then they will never know the Status of “A”
2. Suppose there is some machine installed in hospital and its stopped working , till the time consumer Application will not send request then will never know the machine status to replace or make a backup plan for that machine.

**What is Event driven architecture ?**

It is complementary solution for Microsrevices /SOA . Event Process is captured in event driven architectural pattern.

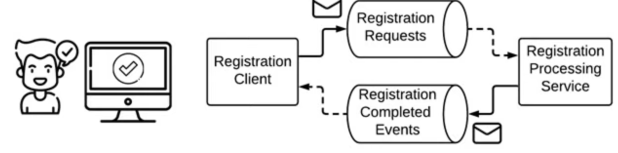


Stream 1

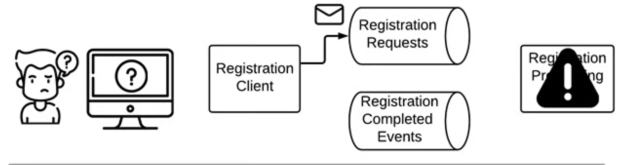
S

**Event Bus**

Stream 2



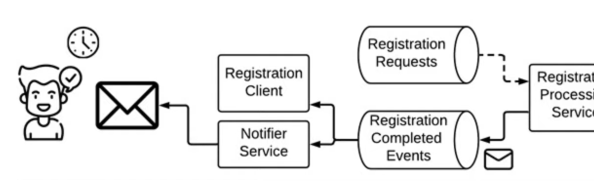
1. Now suppose Registration Processing service is now



1. But when the service will UP , then will get Request from Event Bus

Which Service has already subscribed will process and sent to Event Bus

And then Subscriber of Response event will get notify



**Event Bus Architecture works on Pub/Sub design pattern.**

There are various design pattern available like Rabbit MQ /Kafka etc.

Rabbit MQ :

Rabbit MQ is message Broker used to support Event Based Architecture.

(2) (8)

**Exchange**

 (7)

Q2

(3) Q1

Rabbit MQ

(6)

Bindings (4)

1. Application (Producer) generate Message and send to Rabbit MQ Exchange
2. Rabbit MQ exchange send to Queue which are bind with Exchange
3. The Application uses those messages is called Consumer.

There are various ways that Massages are routed in Rabbit MQ

1. Direct
2. Topic
3. Fanout
4. Header

RabbitMQ UI Server Port for RQ

**How to Implement Rabbit MQ in .net Core**

1. **Set UP Rabbit MQ at Local machine**
2. Install Docker
3. > Docker pull rabbitmq:3-management
4. > docker run -p 15672:15672 -p 5672:5672 –-name rabbitMQTest rabbitMq:3-management
5. Check Rabbit MQ container is running or not
6. 
7. **Guest/guest credential**
8. Producer Creation in .net

Install below Nuget packages in application

1. MassTransit
2. MassTransit.RabbitMQ
3. MassTransit.Extensions.DependencyInjection
4. Create a Background service at Web application (MVC Web client ) act as Producer

public class BusService : IHostedService

{

private readonly IBusControl \_busControl;

public BusService(IBusControl busControl)

{

\_busControl = busControl;

}

public Task StartAsync(CancellationToken cancellationToken)

{

return \_busControl.StartAsync(cancellationToken);

}

public Task StopAsync(CancellationToken cancellationToken)

{

return \_busControl.StopAsync(cancellationToken);

}

}

1. Configure service at Startup.cs

services.AddMassTransit();

services.AddSingleton(provider => Bus.Factory.CreateUsingRabbitMq(

cfg =>

{

cfg.Host("localhost", "/", h => { });

services.AddSingleton(provider=>provider.GetRequiredService<IBusControl>());

services.AddSingleton<IHostedService, BusService>();

}));

1. Sdf

[HttpPost]

public async Task<IActionResult> RegisterOrder(OrderViewModel model)

{

using(MemoryStream ms =new MemoryStream())

{

using(var uploadedfile= model.File.OpenReadStream())

{

await uploadedfile.CopyToAsync(ms);

}

model.ImageData = ms.ToArray();

model.ImageUrl = model.File.FileName;

model.OrderId = Guid.NewGuid();

var targetUri = new Uri(RabbitMqMassTransitConstants.MQUrl + RabbitMqMassTransitConstants.RegisterOrderServiceQueue);

var endpoint = await \_busControl.GetSendEndpoint(targetUri);

await endpoint.Send<IRegisterOrderCommand>(

new

{ Contract to transfer data to RMQ

model.OrderId,

model.Email,

model.ImageData,

model.ImageUrl

}

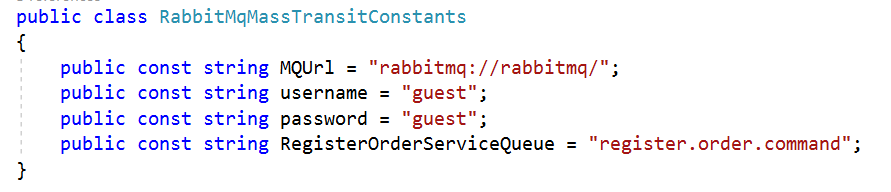
);

}

ViewData["OrderId"] = model.OrderId;

return View("Thanks");

}



1. This will publish message on Exchange named “registered.Order.command”
2. Now On Rabbit MQ we have to Add Queue “Order” under this exchange using binding
3. Then if we publish message from publisher it will appear in “Order” Queue.
4. Consumer Creation in .net Core application
5. Will create a consumer class

public class RegisterOrderCommandConsumer : IConsumer<IRegisterOrderCommand>

{

private readonly IHttpClientFactory \_clientFactory;

private readonly IOrderRepository \_orderRepo;

public RegisterOrderCommandConsumer(IOrderRepository orderRepo, IHttpClientFactory clientFactory)

{

\_orderRepo = orderRepo;

\_clientFactory = clientFactory;

}

public async Task Consume(ConsumeContext<IRegisterOrderCommand> context)

{

var result = context.Message;

if (result.OrderId != null && result.ImageUrl != null

&& result.Email != null && result.ImageData != null)

{

SaveOrder(result);

}

}

private void SaveOrder(IRegisterOrderCommand result)

{

Order order = new Order

{

OrderId = result.OrderId,

Email = result.Email,

Status = Status.Registered,

ImageUrl = result.ImageUrl,

ImageData = result.ImageData

};

\_orderRepo.RegisterOrder(order);

}

}

ii.

services.AddMassTransit(c=> {

c.AddConsumer<RegisterOrderCommandConsumer>();

});

services.AddSingleton(provider => Bus.Factory.CreateUsingRabbitMq(cfg =>

{

cfg.Host("localhost", "/", h => { });

cfg.ReceiveEndpoint(RabbitMqMassTransitConstants.RegisterOrderServiceQueue, e =>

{ // for polling

e.PrefetchCount = 16;

e.UseMessageRetry(x => x.Interval(2, TimeSpan.FromSeconds(10)));

e.Consumer<RegisterOrderCommandConsumer>(provider); // configure Consumer class

});

// cfg.ConfigureEndpoints(provider);

}));