iranda Domeneghetti

y obvious that our technology exceeds our humanity.

RabbitMQ + .Net Core 3.1 + Docker

SEPTEMBER 2, 2020SEPTEMBER 2, 2020 / FERNANDODOMENEGHETTI
Neste exemplo você precisa já ter instalado em seu computador:

- .Net Core 3.1
- Docker instalado e configurado
- Editor de código, neste exemplo utilizei o VsCode durante o desenvolvimento

Primeiro passo é subir um ambiente RabbitMQ utilizando o Docker, para isso rode o comando abaixo em seu cmd\terminal:

docker run -d –hostname my-rabbit –name some-rabbit -p 15672:15672 -p 5672:5672 rabbitmq:3-management

Este comando ira baixar a imagem do RabbitMQ3 Management, e ao criar o container ira mapear a porta default 15672 para acessar o Rabbit via browser, a porta 5672 para o trafego das mensagens via protocolo amqp e definimos o nome default do host para my-rabbit

Para validar se o container está em execução, basta executar o comando docker container ls

```
CONTAINER ID        IMAGE                  COMMAND                  CREATED             STATUS              PORTS
                    NAMES
91d370d14ae0        rabbitmq:3-management   "docker-entrypoint.s…"   About a minute ago   Up About a minute   4369/tcp, 5671/tcp, 0.0.0.0:5672->5672/tcp, 15671/tcp, 25672/tcp
, 0.0.0.0:15672->15672/tcp   some-rabbit
```

(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/dockerps.png)

Agora basta acessar o RabbitMQ no navegar através da url http://localhost:15672/ (http://localhost:15672/), o usuário default para o login é:

- Usuario: guest
- Senha: guest

Configurando a Queue e o Exchange:

Embora seja explicado nos próximos tópicos o que\para que serve a Queue e a Exchange, acho importante ja configurarmos uma e ver funcionando, assim quando vier a explicação, tudo fará mais sentido.

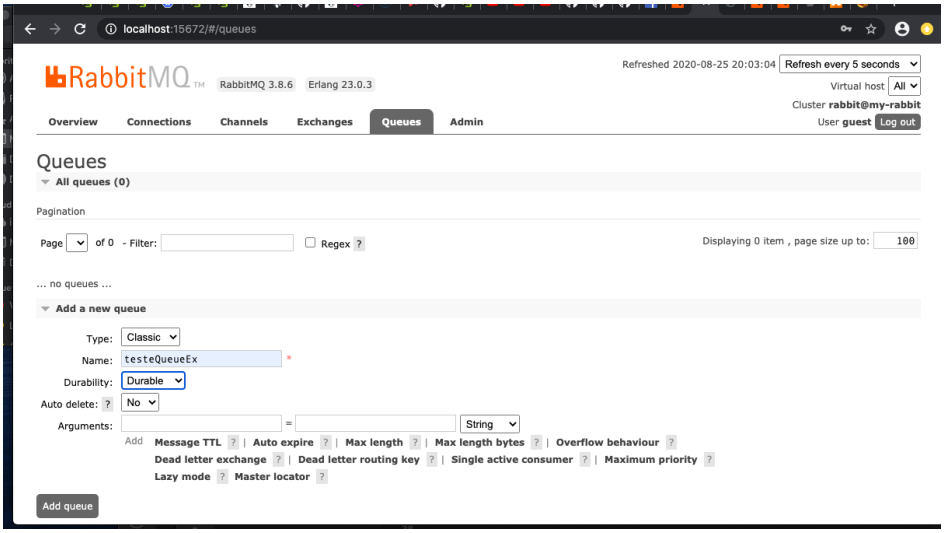
No projeto de Read da fila você vera que embora seja um projeto de leitura, existe uma declaração de uma Queue, isto não tem problema, desde que todos os projetos que utilizem a Queue estajam parametrizando elas com os mesmos valores:

```
channel.QueueDeclare(queue: "testeQueueEx",
                     durable: true,
                     exclusive: false,
                     autoDelete: false,
                     arguments: null);
```

(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/DeclaracaoQueue.png)

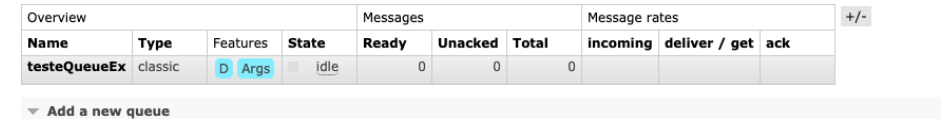
Neste exemplo vamos configurar a Queue via browser e já realizar o vinculo com a Exchange, para criar a Queue basta acessar o menu Queues você verá na página que foi aberta no lado esquerdo um botão com o texto “Add queue”, basta clicar sobre ele e preencher os dados da seguinte maneira:

- Type: Classic
- Name: testeQueueEx
- Durability: Durable
- Auto Delete: No
- Arguments: [Não preencher]



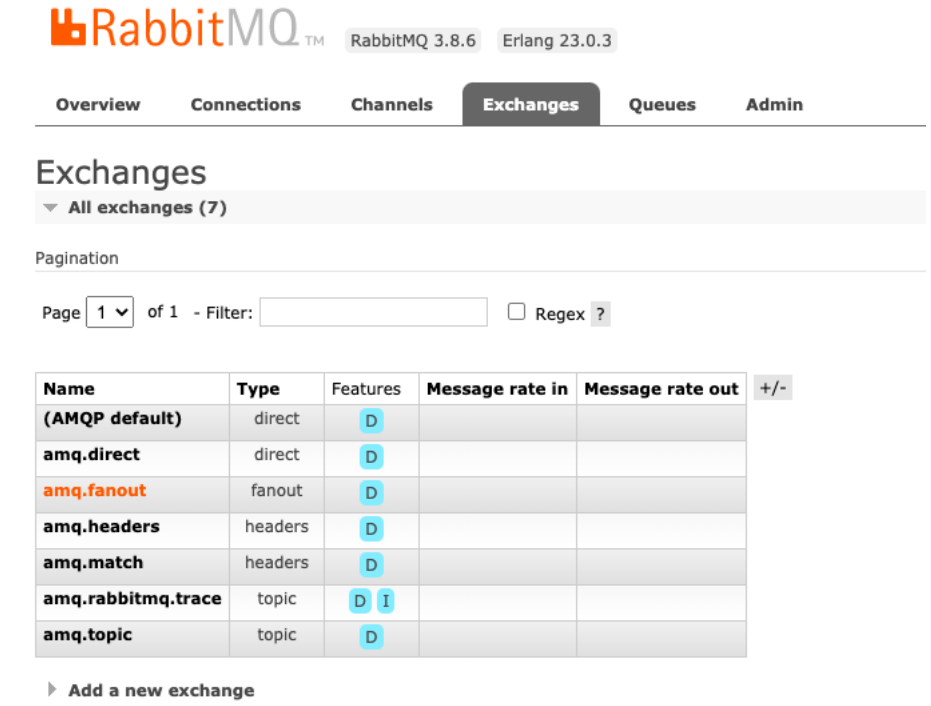
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/CriacaoQueue.png)

Após basta clicar novamente sobre o botão “Add queue”, que a mesma será salva e você será redirecionado para as queues existentes, contendo agora a sua:



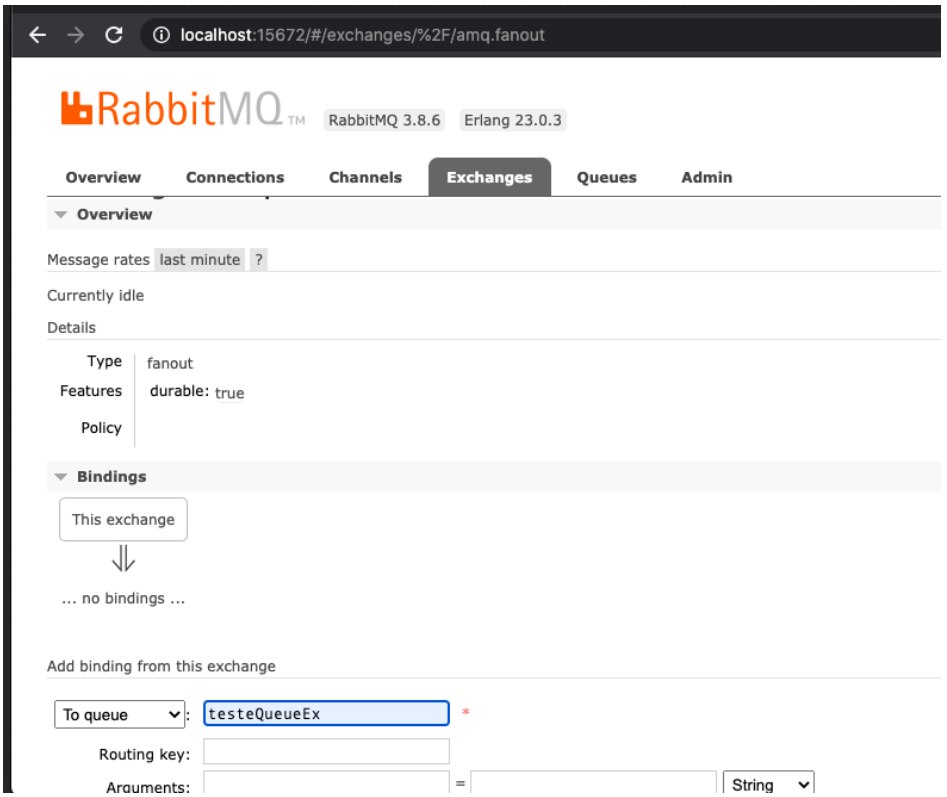
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/QueueCriada.png)

Ao lado do menu da Queues, você tem acesso ao menu Exchanges, clique sobre ele para ver as Exchanges existentes, em seguida clique na opção “amq.fanout”



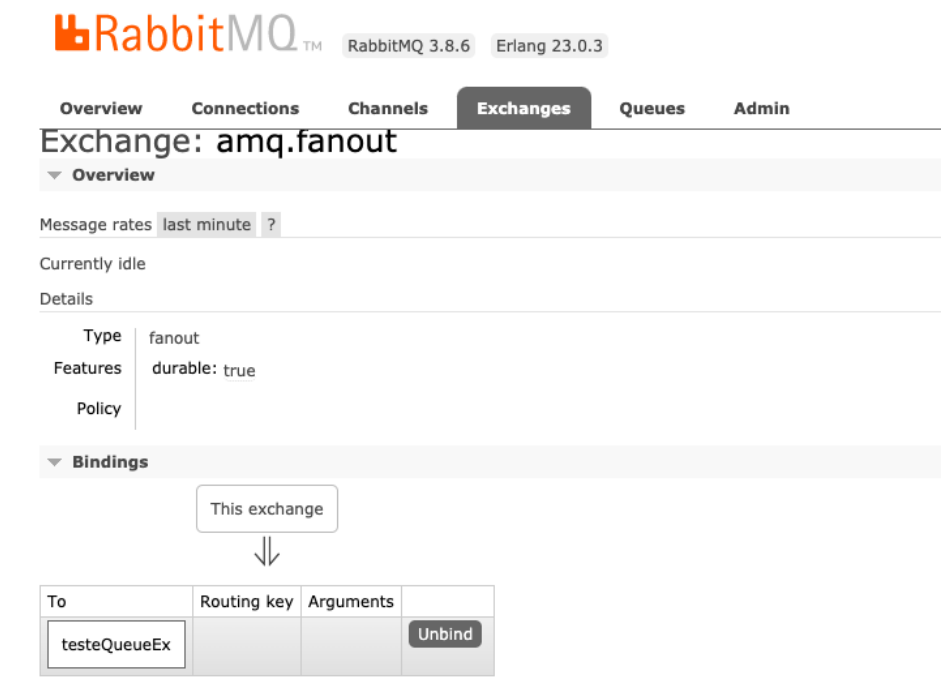
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/Exchanges.png)

Ao clicar na opção “amq.fanout” você será redirecionado a uma outra página, nesta página você verá uma opção chamada “Bindings”, clique sobre ela para realizar um vinculo com a Queue que criamos no passo anterior, basta informar o nome da Queue e clicar em “bind”:



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/ExchangeRealizandoBindingQueue.png)

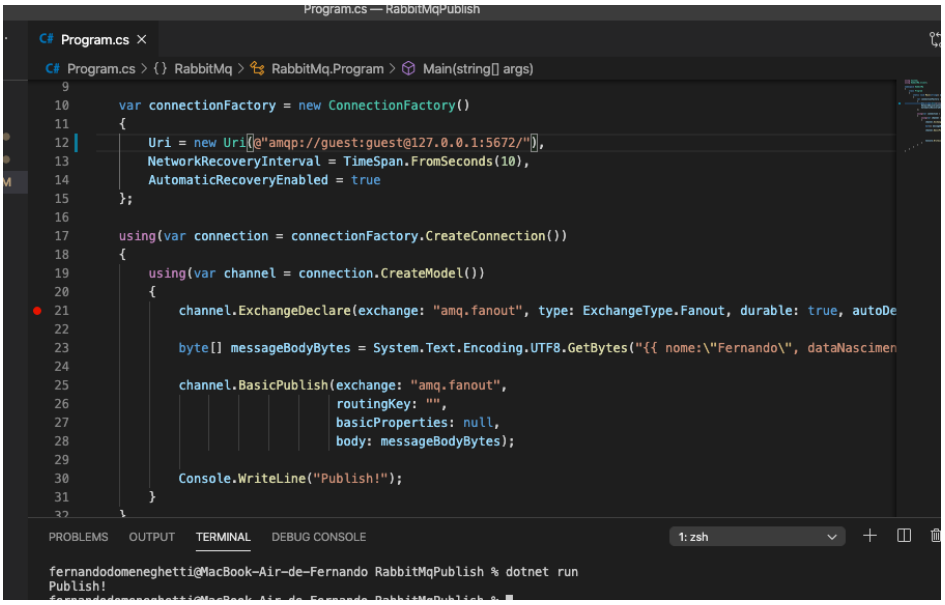
Ao realizar o bind você verá uma imagem mostrando o vinculo criado:



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/ExchangeAposBinding.png)

Pronto! Seu ambiente está configurado e já podemos testar, exato é somente isso!

abra o projeto RabbitMqPublish e no terminal basta digitar “dotnet run”, assim que for executado você no console um texto “Publish!”



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/Publish.png)

Entre as linhas 10 e 15 está sendo criado a conexão com o Rabbit, mas precisamente na lina 12 onde informo na Uri

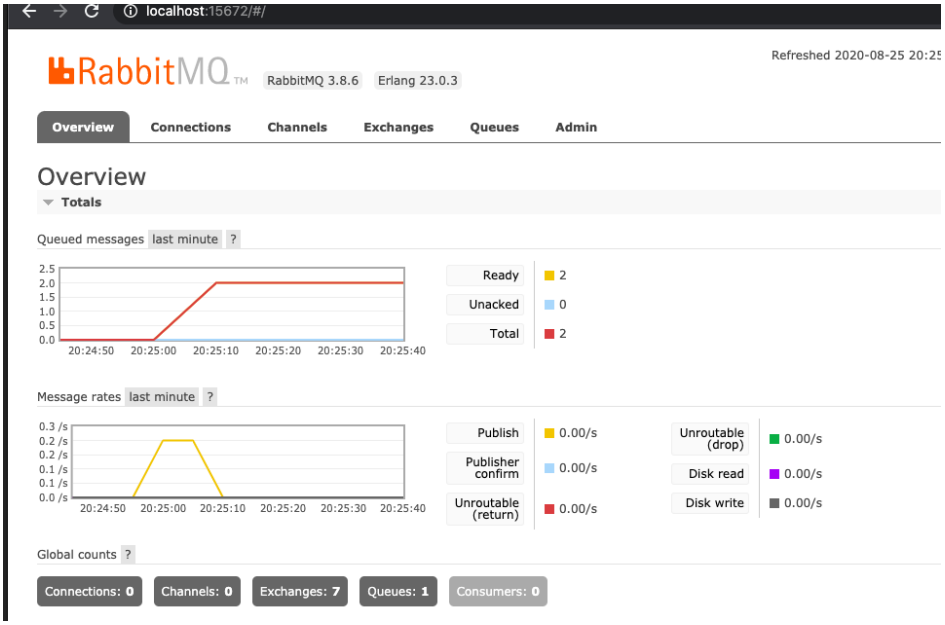
- Amqp é o protocolo de comunicação
- guest:guest são os dados de login no Rabbit
- 127.0.0.0:5672 é referente ao servidor do Rabbit, como estamos rodando o Rabbit via Docker no próprio pc, informo apenas o ip seguido da porta mapeada no Docker
- / a última barra serve para informar o host, por estar utilizando o host padrão, não será informado nenhum valor após a /, caso você crie um novo host pode então passar qual o host estaria sendo utilizado neste processo

Na linha 21 declaro o Exchange utilizado

Na linha 23 é a mensagem que será enviada a fila, o ponto de atenção aqui é que o Rabbit necessita que o conteúdo esteja em uma base64

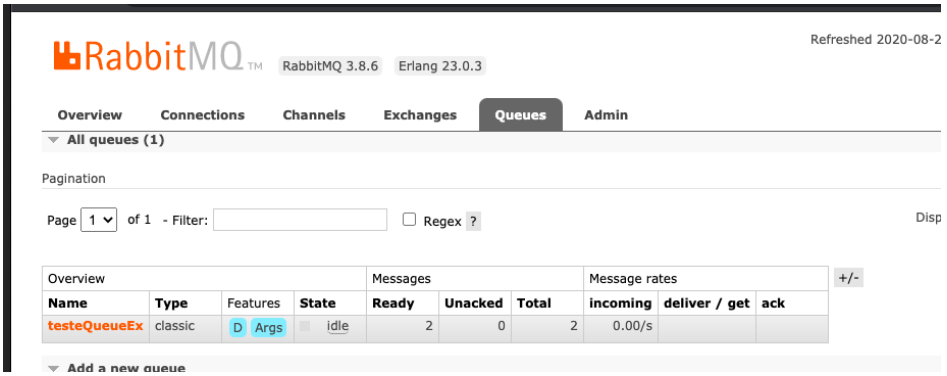
Na linha 25 é a publicação da mensagem em si informando o Exchange e o body.

Volte ao RabbitMQ, note na página overview que as mensagens já foram identificadas:



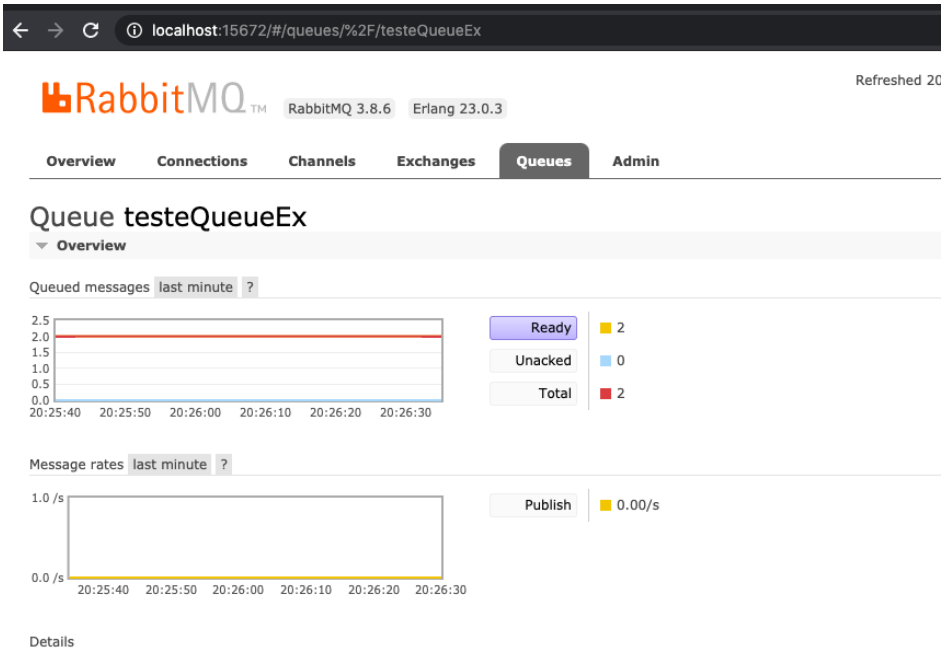
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/RabbitOverview.png)

Ao acessar o menu Queues, você verá na queue que você criou que na área Messages a coluna Ready e Total possuem agora um valor referente a quantidade de publish realizados e que ainda não foram lidos



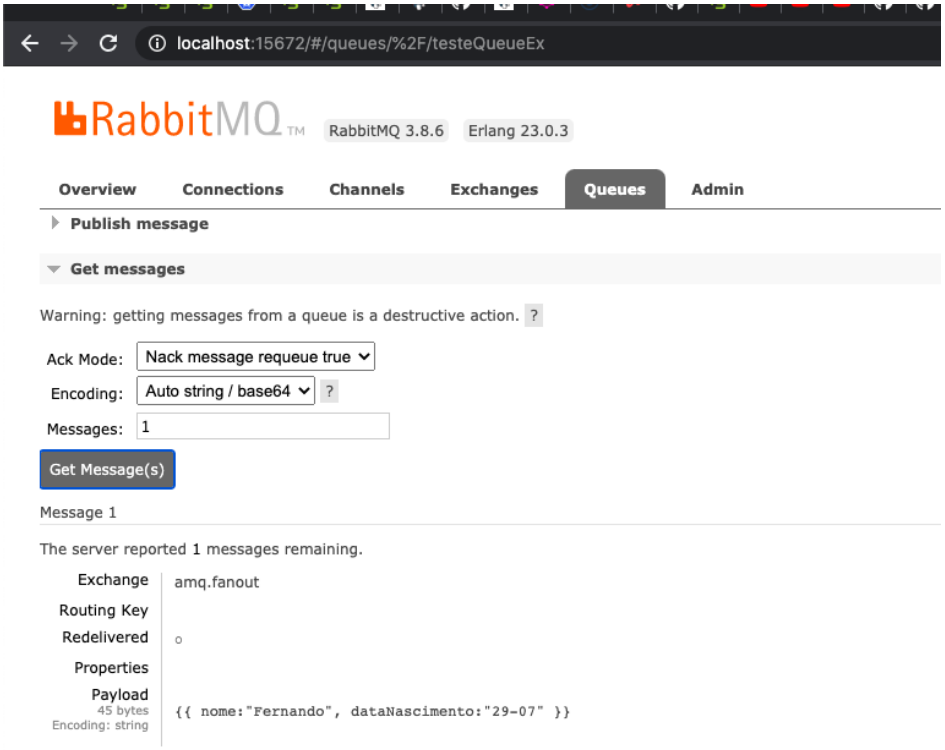
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/QueueWithMessage.png)

Clique novamente sobre a queue, nesta página além de conseguir ver a quantidade de mensagens que estão na fila para serem lidos como mostra o print abaixo:



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/QueueCountMessage.png)

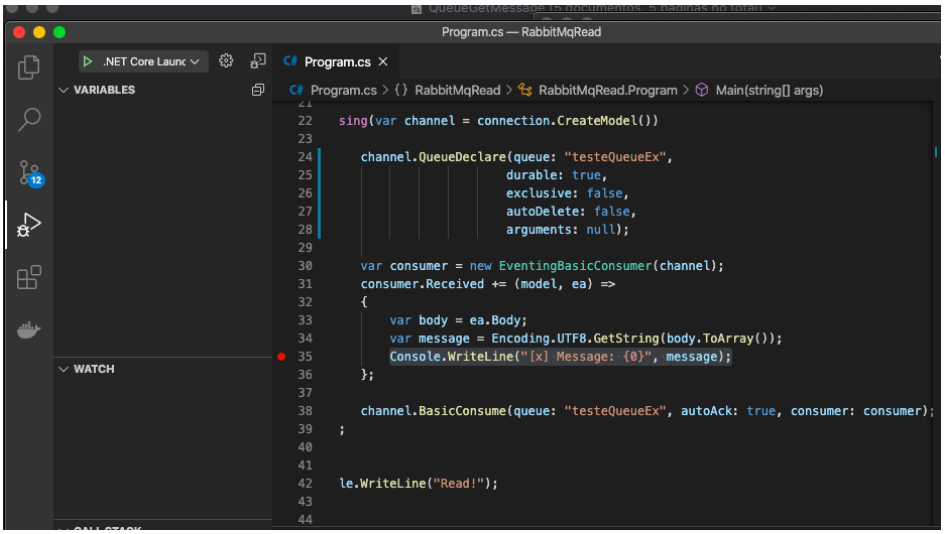
Você tem mais abaixo um menu na página como nome de “Get messages” para visualizar o conteúdo das mensagens, mantenhas as configurações da tela como o print abaixo e clique em seguida no botão “Get Message(s)”



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/QueueGetMessage.png)

Este passo apenas te ajuda a visualizar a sua mensagem, mas não remove ela da fila e também não existe nenhum processamento sobre a mesma ainda, isto será feito neste momento.

No VsCode coloque um breakpoint sobre Console.Write na linha 35 e em sseguida inicialize o modo de depuração do vscode

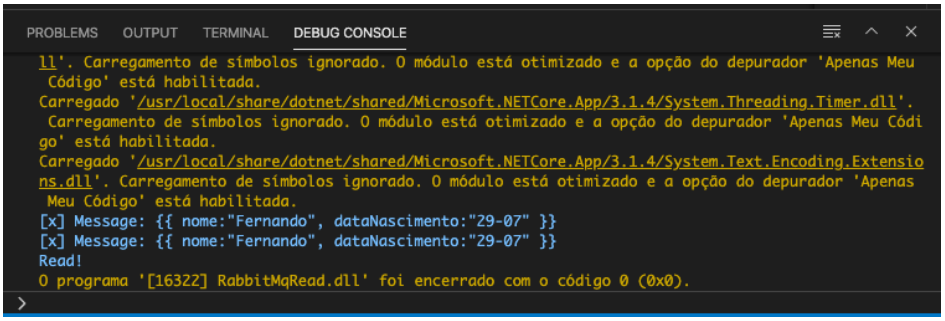


(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/AreaDebugVsCode.png)

Assim que a mensagem for lida, ao passar o mouse sobre a propriedade message note que você já consegue ver o conteúdo

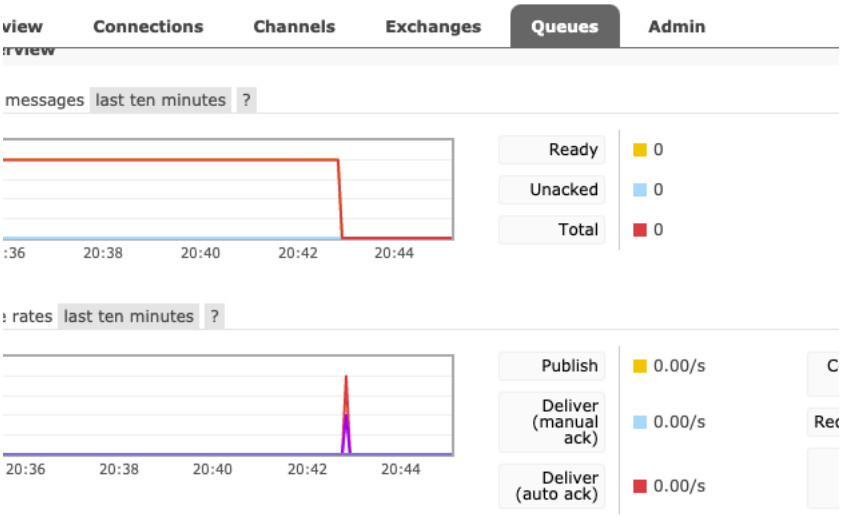
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/DebugAnalisandoMensagemLida.png)

Assim que concluir o debug, você verá que as mensagens lidas, tem seu conteúdo escrito no console junto com a palavra “Read!”:



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/DebugConsole.png)

Importante voltar ao navegador do RabbitMQ e ver como ficou a Queue após a leitura, acesse o menu Queue, você verá que a quantidade de mensagens agora da fila é 0 novamente, ao acessar sua Queue veja no gráfico que após as mensagens serem lidas elas foram removidas e o gráfico ajuda inclusive a ver este momento:



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/MensagemSaiuQueue.png)

Agora que você já configurou e já viu uma mensagem sendo envianda e também já fez a leitura do conteúdo das mensagens na fila, vamos entender melhor o que são cada coisa e como elas se relacionam…

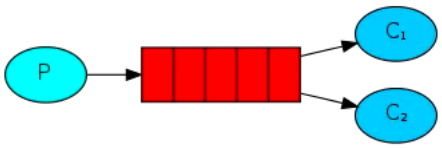
(https://github.com/domeneghetti/RabbitMQ-Exemplo01#vis%C3%A3o-geral-do-publish--consumer--subscribe)Visão Geral do Publish \ Consumer \ Subscribe

- Publish: São os responsáveis pelo publicação da mensagem na Queue\Exchange
- Consumer: São os ouvintes das Queues, os mesmos capturam a mensagem na Queue gerada pelo Publish e processam ela.
- Subscribe: Em um modelo mais simples um Publish publicaria a mensagem diretamente em uma Queue que teria seu(s) consumidor(es), já no modelo Publish\Subscribe o Publish manda a mensagem em uma Exchange do tipo Fanout e a mensagem será clonada e entregue a todas as Queues ligados a ela.

Com base no que foi falado acima, o caminho mais simple seria, um Publish simplesmente envia a mensagem em uma queue que terá um Consumer ouvindo ela e processará a mesma, como no desenho abaixo:

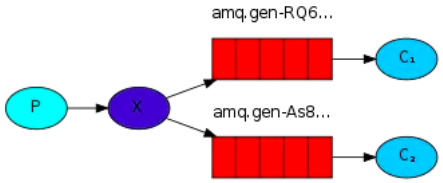
(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/Simples.png)

Porém a quantidade de mensagens a serem processadas pode em muitos casos ser muito alta, neste caso você poderá colocar mais Consumer ouvindo a mesma fila, isso não deverá ser um problema, pois o RabbitMQ trabalha de forma atômica, entregando uma mensagem por vez aos consumers, além de garantir que a mensagem seja entregue somente a um Consumer.



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/Worker.png)

Já no modelo Publish\Subscribe, você entregará sua mensagem não a uma Queue em especifico mas sim a uma Exchange que distribuirá a mesma entre as Queues (exemplo do fonte deste artigo) e que será processada por um ou mais Consumers.



(https://github.com/domeneghetti/RabbitMQ-Exemplo01/blob/master/Imagens/PublishSubscribe.png)

(https://github.com/domeneghetti/RabbitMQ-Exemplo01#documenta%C3%A7%C3%A3o-rabbitmq)Documentação RabbitMQ

Por fim, o site do próprio RabbitMQ tem muita informação, muitos exemplos em diversas linguagens e que com certeza vai te ajudar.

Link: https://www.rabbitmq.com/getstarted.html (https://www.rabbitmq.com/getstarted.html)

Todo conteúdo e fonte estão no meu Github https://github.com/domeneghetti/RabbitMQ-Exemplo01 (https://github.com/domeneghetti/RabbitMQ-Exemplo01)

Valeu!

Uncategorized
.NET CORE 3.1 **RABBITMQ**