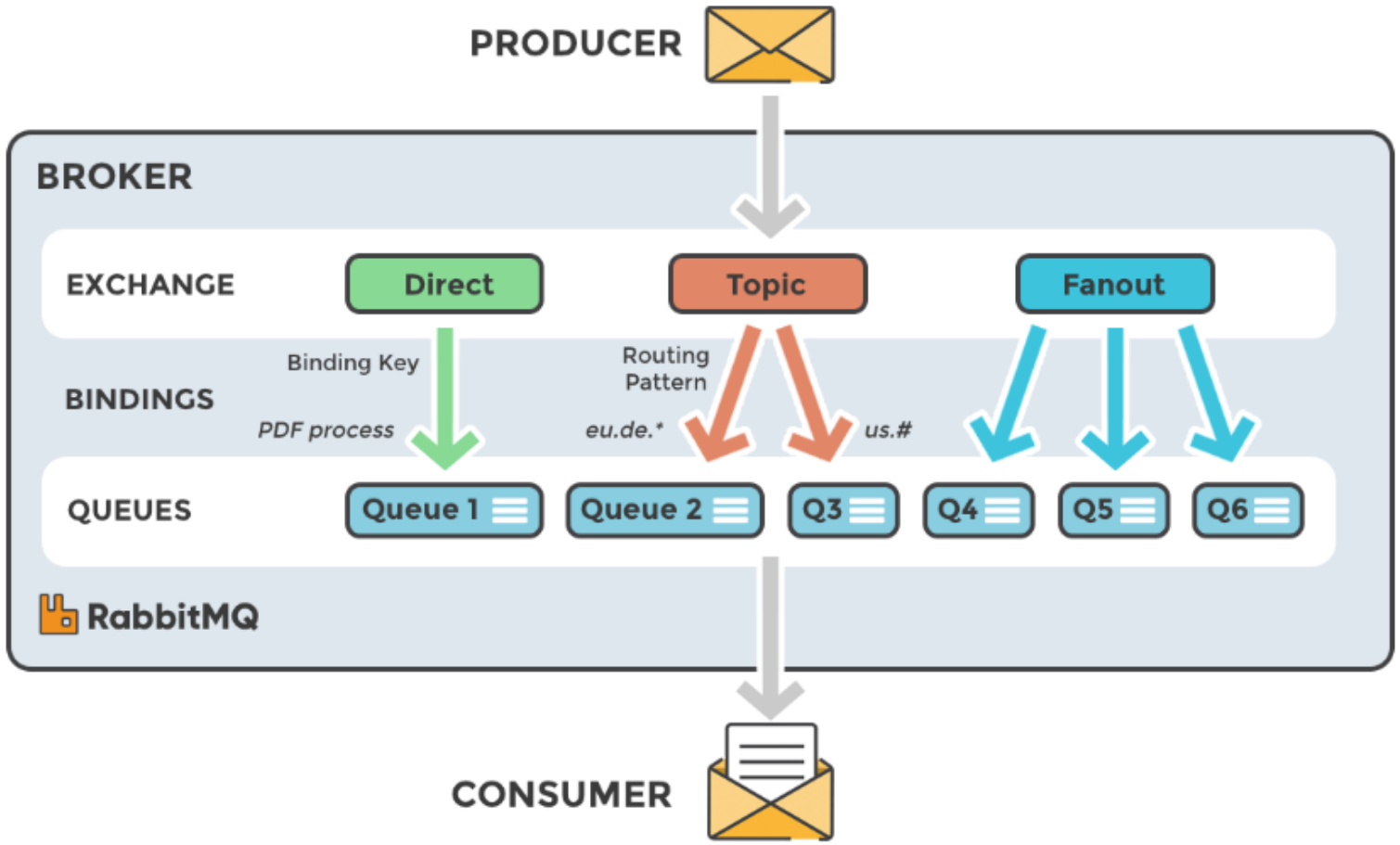
COMUNICAÇÃO EM SISTEMAS DISTRIBUÍDOS COM RABBITMQ

ALUNOS: Henrique Fazollo e João Pedro Diniz

RabbitMQ

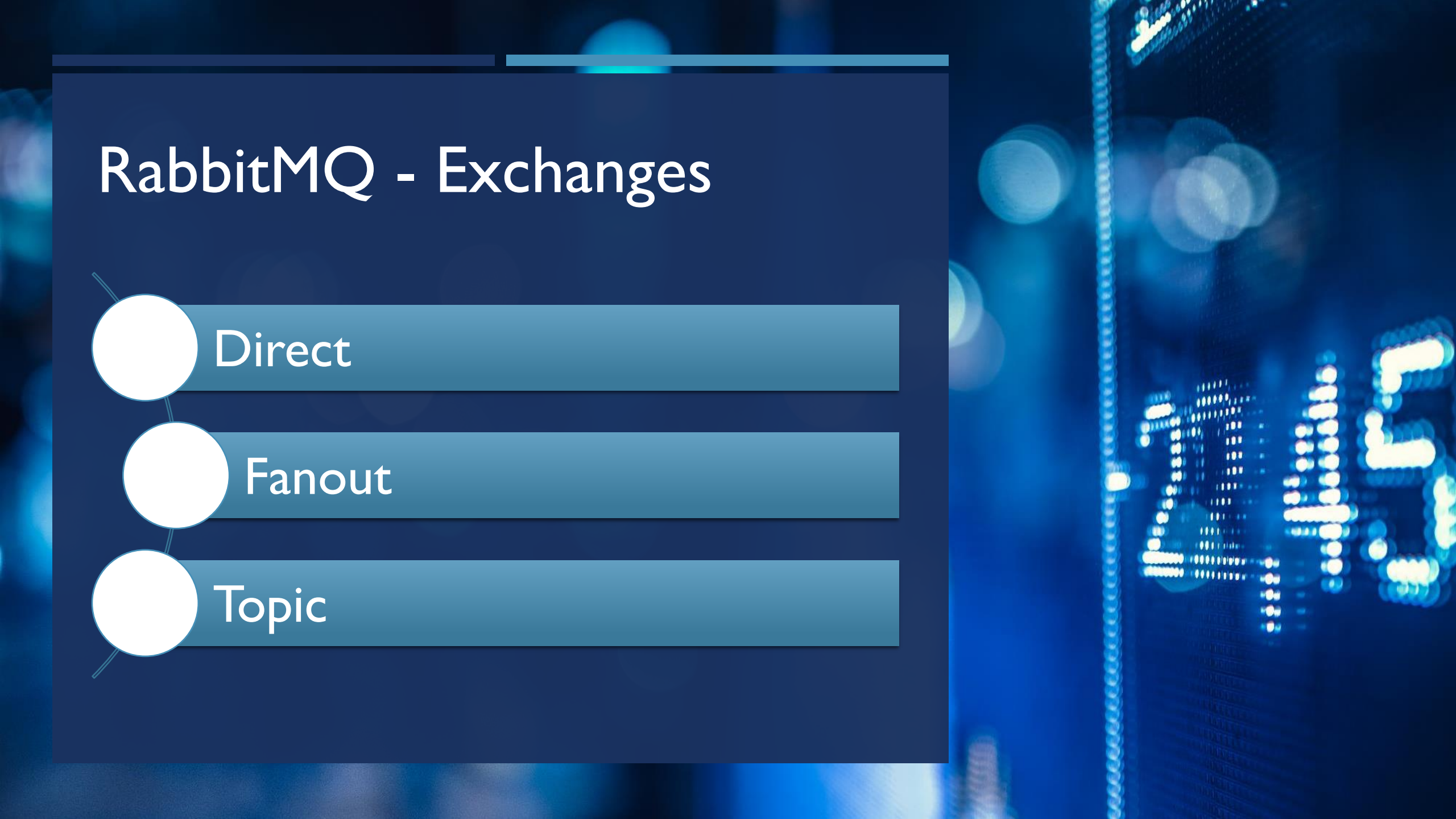
- É o serviço de mensageria mais utilizado no mercado.
- O servidor RabbitMQ é uma das implementações mais populares do AMQP.
- Em vez de manipular diretamente filas, os produtores entram em contato com uma exchange, que, por sua vez, coloca mensagens em uma ou várias filas.
- Em essência, uma exchange permite que um produtor use uma RPC síncrona ou assíncrona para um gerenciador de filas.

RabbitMQ - Arquitetura do Sistema



RabbitMQ - Exchanges

- Direct
- Fanout
- Topic



RabbitMQ - Exchanges

- **Direct Exchange (Troca Direta)**
  - Roteia mensagens para filas com base em uma chave de roteamento.
  - A mensagem vai para a fila cuja chave de roteamento corresponde exatamente à chave especificada na mensagem.
  - É útil quando você tem várias filas e deseja direcionar mensagens específicas para filas específicas.

- **Fanout Exchange (Troca de Emissão)**
  - Roteia mensagens para todas as filas vinculadas a ela.
  - Ignora completamente a chave de roteamento na mensagem.
  - É útil quando você deseja transmitir a mesma mensagem para várias filas.

- **Topic Exchange (Troca de Tópico)**
  - Roteia mensagens para filas com base em padrões de chave de roteamento (palavras-chave).
  - As filas são vinculadas à troca com uma chave de roteamento que inclui padrões.
  - É útil quando você deseja rotear mensagens com base em critérios específicos.

RabbitMQ - Vantagens

- Suporte a Múltiplos Protocolos de Linguagens
- Distribuição e Escalabilidade
- Confiabilidade e Persistência

RabbitMQ - Suporte a Múltiplos Protocolos de Linguagens

• **Protocolos:** RabbitMQ suporta AMQP (Advanced Message Queuing Protocol), MQTT, STOMP, entre outros, permitindo uma comunicação flexível e robusta.

• **Linguagens:** Há bibliotecas disponíveis para diversas linguagens, incluindo Python, Java, JavaScript, Ruby, C#, PHP, Go, entre outras. Isso facilita a integração de diferentes partes de um sistema desenvolvido em várias linguagens.

RabbitMQ - Distribuição e Escalabilidade

• **Clustering:** RabbitMQ suporta clustering, permitindo que várias instâncias do RabbitMQ trabalhem juntas para fornecer alta disponibilidade e escalabilidade.

• **Federation:** Permite conectar servidores RabbitMQ em diferentes data centers ou nuvens, permitindo a distribuição geográfica de mensagens.

• **Shovel:** Move mensagens de uma instância de RabbitMQ para outra de forma automatizada e confiável.

RabbitMQ - Confiabilidade e Persistência

- **Mensagens Persistentes:** Suporte para mensagens persistentes que são salvas em disco, garantindo que não sejam perdidas mesmo em caso de falha do servidor.

- **Acknowledgements:** Confirmação de entrega de mensagens, garantindo que as mensagens só sejam removidas da fila após confirmação de recebimento pelo consumidor.

- **Dead Letter Exchanges:** Mensagens que não podem ser entregues podem ser redirecionadas para uma dead letter exchange para análise posterior.

Producer - Conexão via Docker

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

vm1@vm1:~/Documents/producer$ sudo docker run -d --rm --name rabbitmq -p 5672:5672 -p 15672:15672 rabbitmq:3.13-management
[sudo] password for vm1:
352768198c45977a69b0c495a028e68b69a350101b04cb386334ecc39532957e
vm1@vm1:~/Documents/producer$
```

Producer - Conexão com Servidor e Criação de Canal

```
amqp.connect('amqp://localhost', (err, conn) => {
    if (err) {
        throw err;
    }

    conn.createChannel((err, ch) => {
        if (err) {
            throw err;
        }
```

Producer - Interação com Usuário

```
const promptUser = () => {
    rl.question('Escolha o tipo de exchange (direct, fanout, topic): ', (exchangeType) => {
        if (!['direct', 'fanout', 'topic'].includes(exchangeType)) {
            console.log('Tipo de exchange invalido');
            return promptUser();
        }

        rl.question('Digite a mensagem: ', (msg) => {
            if (exchangeType === 'direct' || exchangeType === 'topic') {
                rl.question('Digite as chaves de roteamento (separadas por vírgula): ', (routingKeysInput) => {
                    const routingKeys = routingKeysInput.split(',').map(key => key.trim());
                    sendMessage(ch, exchangeType, msg, routingKeys);
                    promptUser();
                });
            } else {
                sendMessage(ch, exchangeType, msg);
                promptUser();
            }
        });
    });
};
```

Producer - Interação com Usuário

```
const sendMessage = (channel, exchangeType, msg, routingKeys = ['']) => {
    const exchange = exchangeType === 'direct' ? 'direct_logs' :
                     exchangeType === 'fanout' ? 'logs' : 'topic_logs';

    channel.assertExchange(exchange, exchangeType, { durable: false });
    routingKeys.forEach((routingKey) => {
        channel.publish(exchange, routingKey, Buffer.from(msg));
        console.log(` [x] Enviou ${exchangeType} com a mensagem: '${msg}' e chave de roteamento: '${routingKey}'`);
    });
};
```

Consumer - Log de Mensagens

```python
def log_message(self, exchange_type, body, routing_key=None):
    log_entry = {
        "exchange_type": exchange_type,
        "message": body.decode(),
        "timestamp": datetime.now().isoformat()
    }
    if routing_key is not None:
        log_entry["routing_key"] = routing_key

    with open('message_log.json', 'a') as log_file:
        log_file.write(json.dumps(log_entry) + '\n')

    self.add_message(exchange_type, body, routing_key)
```

Consumer - Filtro de Mensagens

```python
def filter_direct_messages(self):
    routing_key = self.direct_filter_input.text()
    self.direct_list.clear()
    with open('message_log.json', 'r') as log_file:
        for line in log_file:
            log_entry = json.loads(line)
            if log_entry['exchange_type'] == 'direct' and (routing_key == '' or log_entry.get('routing_key') == routing_key):
                self.add_message('direct', log_entry['message'].encode(), log_entry.get('routing_key'))
    self.update_message_counts()

def filter_topic_messages(self):
    routing_pattern = self.topic_filter_input.text()
    self.topic_list.clear()
    with open('message_log.json', 'r') as log_file:
        for line in log_file:
            log_entry = json.loads(line)
            if log_entry['exchange_type'] == 'topic' and (routing_pattern == ''
                                                          or self.match_routing_pattern(log_entry.get('routing_key', ''), routing_pattern)):
                self.add_message('topic', log_entry['message'].encode(), log_entry.get('routing_key'))
    self.update_message_counts()
```

Consumer - Callback para Mensagens

```
def callback_direct(self, ch, method, properties, body):
    self.log_message('direct', body, method.routing_key)

def callback_fanout(self, ch, method, properties, body):
    self.log_message('fanout', body)

def callback_topic(self, ch, method, properties, body):
    self.log_message('topic', body, method.routing_key)
```

Consumer - Correspondência para Padrões

```
def match_routing_pattern(self, routing_key, pattern):
    parts = routing_key.split('.')
    pattern_parts = pattern.split('.')
    if len(parts) != len(pattern_parts):
        return False
    for i in range(len(parts)):
        if pattern_parts[i] != '#' and parts[i] != pattern_parts[i]:
            return False
    return True
```

Consumer - Consumo de Mensagens

```python
def consume_direct(self):
    connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.1.10'))
    channel = connection.channel()
    channel.exchange_declare(exchange='direct_logs', exchange_type='direct')
    result = channel.queue_declare(queue='', exclusive=True)
    queue_name = result.method.queue
    routing_keys = ['projeto', 'sistemas', 'distribuidos']
    for routing_key in routing_keys:
        channel.queue_bind(exchange='direct_logs', queue=queue_name, routing_key=routing_key)
    channel.basic_consume(queue=queue_name, on_message_callback=self.callback_direct, auto_ack=True)
    channel.start_consuming()

def consume_fanout(self):
    connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.1.10'))
    channel = connection.channel()
    channel.exchange_declare(exchange='logs', exchange_type='fanout')
    result = channel.queue_declare(queue='', exclusive=True)
    queue_name = result.method.queue
    channel.queue_bind(exchange='logs', queue=queue_name)
    channel.basic_consume(queue=queue_name, on_message_callback=self.callback_fanout, auto_ack=True)
    channel.start_consuming()

def consume_topic(self):
    connection = pika.BlockingConnection(pika.ConnectionParameters('192.168.1.10'))
    channel = connection.channel()
    channel.exchange_declare(exchange='topic_logs', exchange_type='topic')
    result = channel.queue_declare(queue='', exclusive=True)
    queue_name = result.method.queue
    routing_keys = ['sistemas.distribuidos', 'projeto.sistemas', 'projeto.distribuidos']
    for routing_key in routing_keys:
        channel.queue_bind(exchange='topic_logs', queue=queue_name, routing_key=routing_key)
    channel.basic_consume(queue=queue_name, on_message_callback=self.callback_topic, auto_ack=True)
    channel.start_consuming()
```

MUITO OBRIGADO A TODOS!
BOA NOITE!