



# RabbitMQ

## How to make microservices talk

Ms.C. **João Daher**

Computer Science @ UFLA

Masters Artificial Intelligence @ UNIFEI

Backend Developer @ eduK

# What?



Message broker

Written in Erlang

Implements AMQP Protocol

 RabbitMQ

# Why?



Event-driven processing

Deliver later

Asynchronous processing

Loose coupling

Load balancing

Open source

 RabbitMQ

# Alternatives?



Redis

AWS **SNS+SQS**



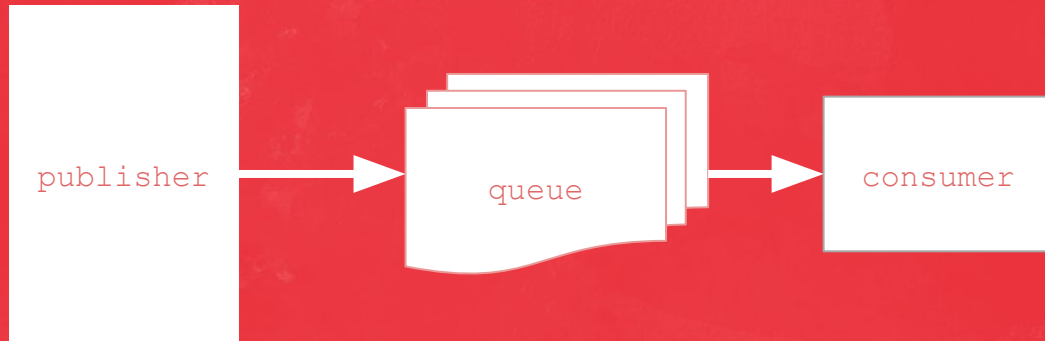
Apache **Kafka**



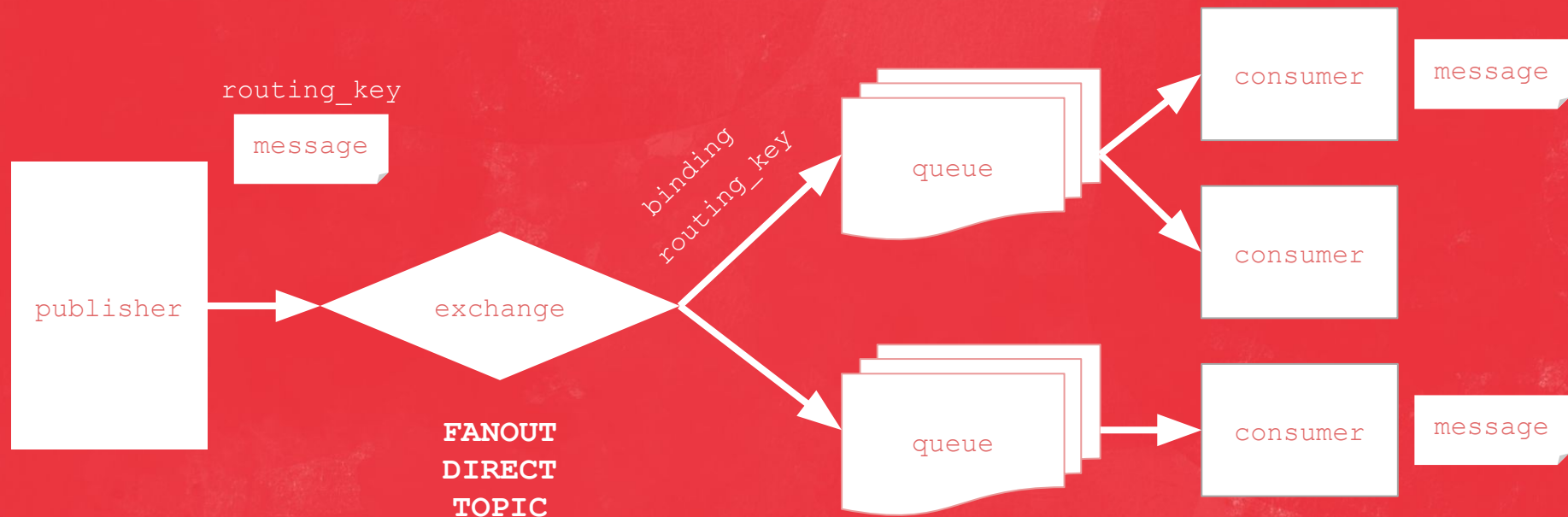
Google **Pubsub**



# PubSub?

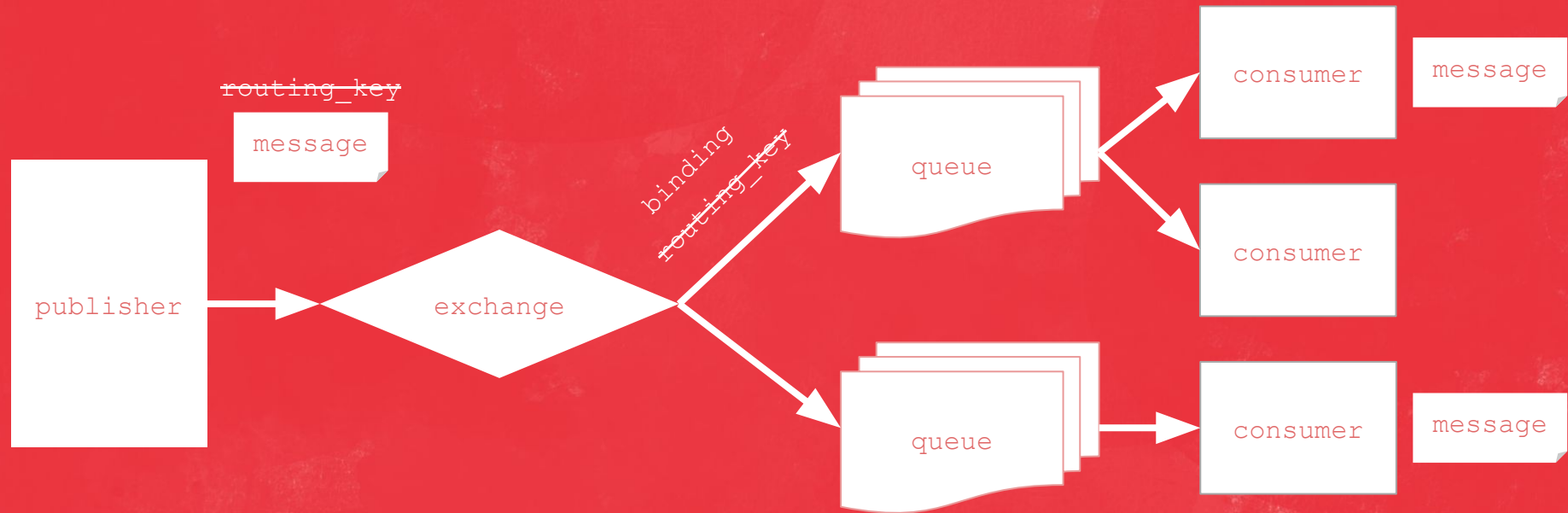


# AMQP?

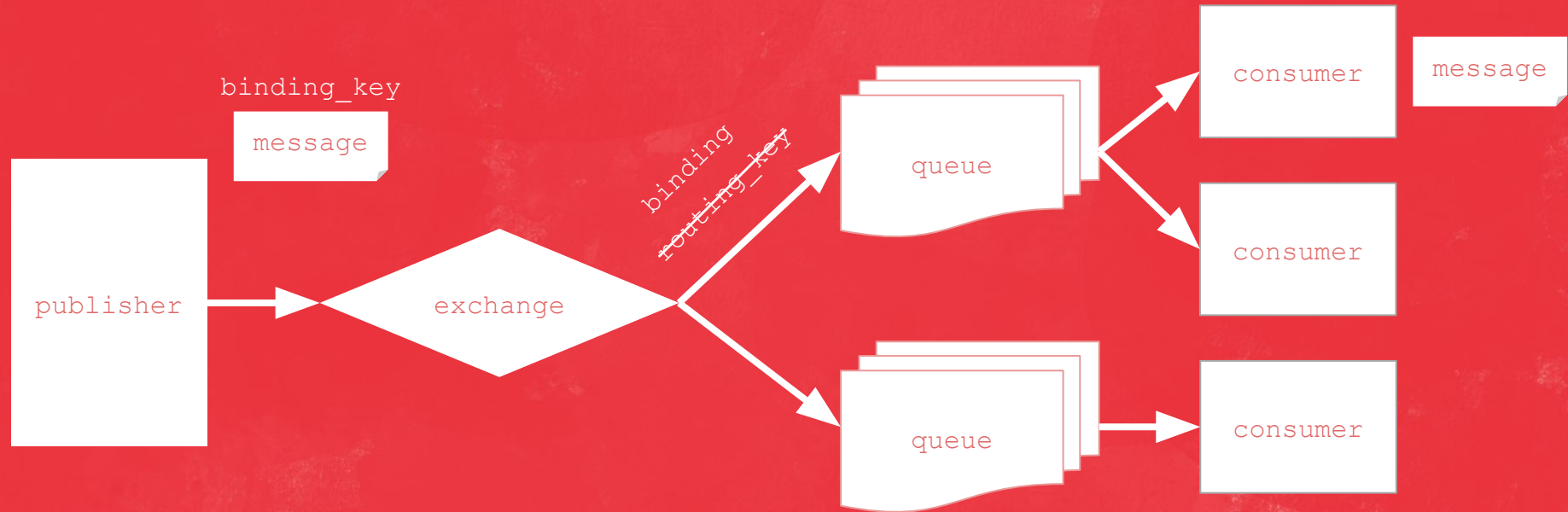




# Fanout?

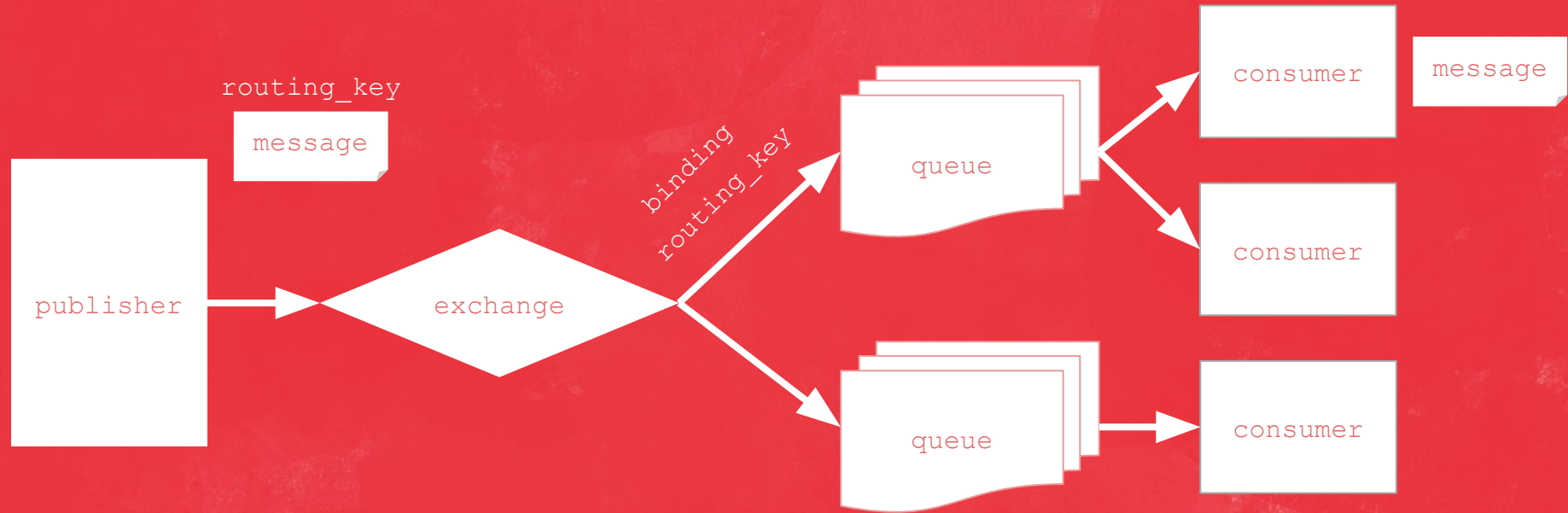


# Direct?



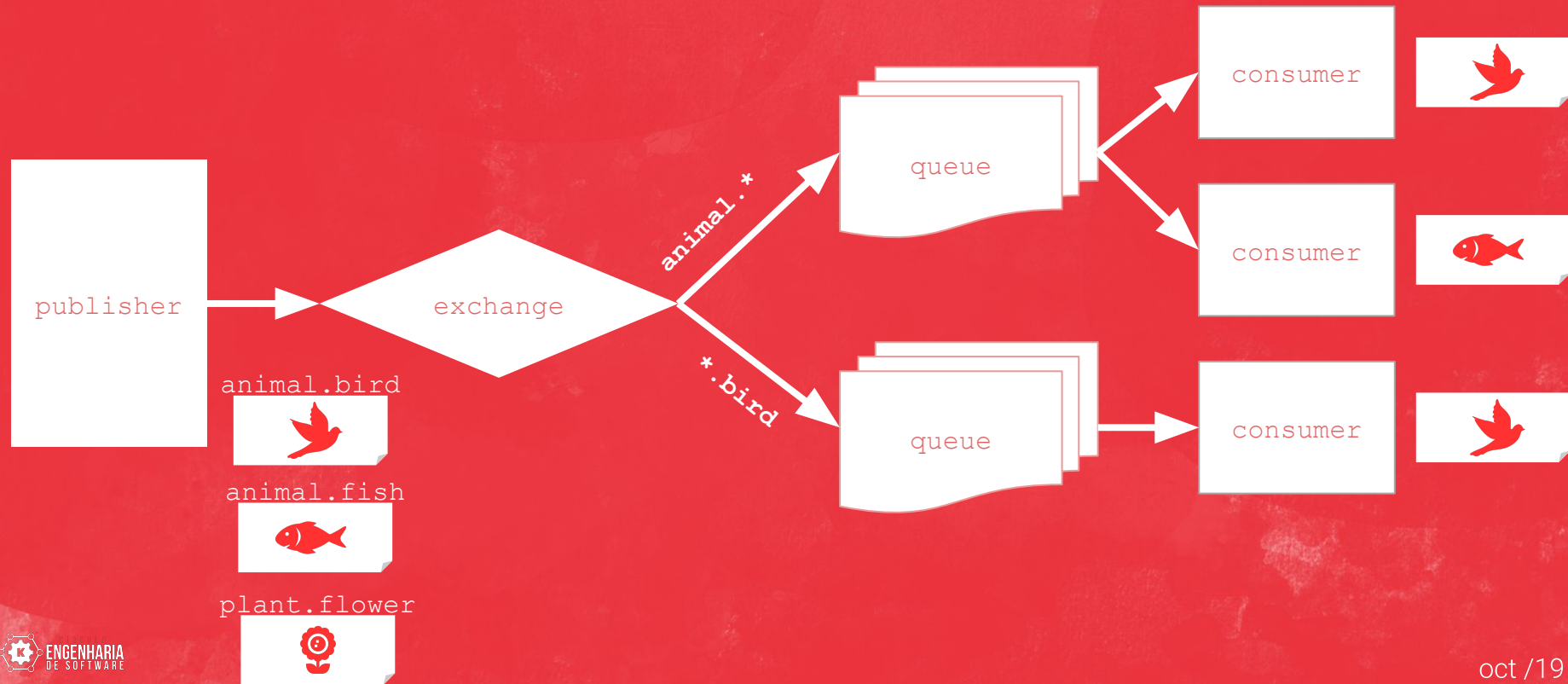


# Topic?



# How does it work?

K



# Python?



## Pika

```
pip install pika
```

Small & Compact

Low Level

## Kombu

```
pip install kombu
```

Retrying

Failover

Connection Pool

High Level

Celery Project

# Publishing?



## Kombu Producer

```
from kombu import Connection, Exchange, Producer

conn = Connection("amqp://localhost:5672/")
my_exchange = Exchange(name='nature', type='topic')
producer = Producer(
    exchange=my_exchange,
    routing_key="animal.bird",
    channel=conn.channel(),
)
producer.publish(message={'bird_name': 'dove'})
```

animal.bird



# Consuming?



## Kombu Consumer

```
from kombu import Connection, Exchange, Queue, Consumer

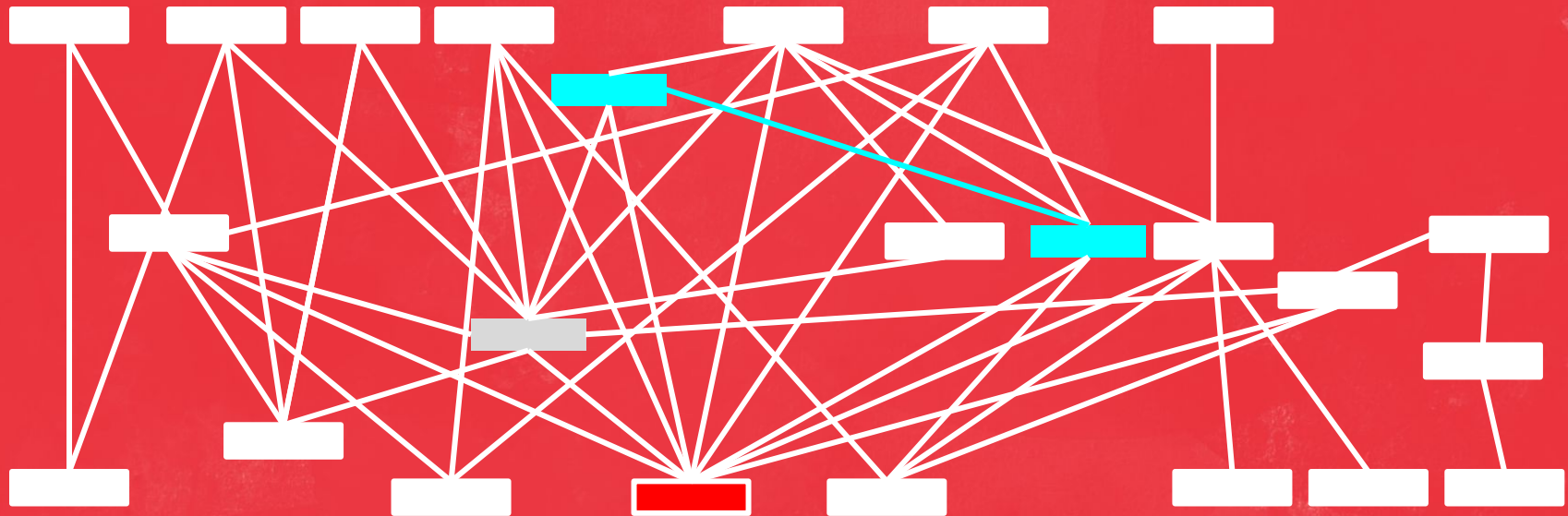
conn = Connection("amqp://localhost:5672/")
my_exchange = Exchange(name='nature', type='topic')
bird_queue = Queue(name="birds_only", exchange=exchange, routing_key="*.bird")
bird_queue.maybe_bind(conn)
bird_queue.declare()
with Consumer(conn, queues=bird_queue, callbacks=[process_message]):
    conn.drain_events()

def process_message(body, message):
    print(f"Do some magic with this: {body}")
    message.ack()
```



# The Problem IRL?

K



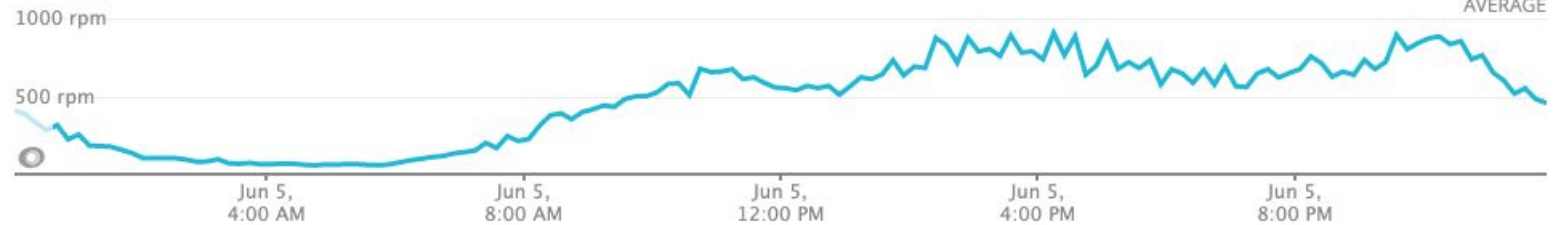


# The Problem IRL?

K



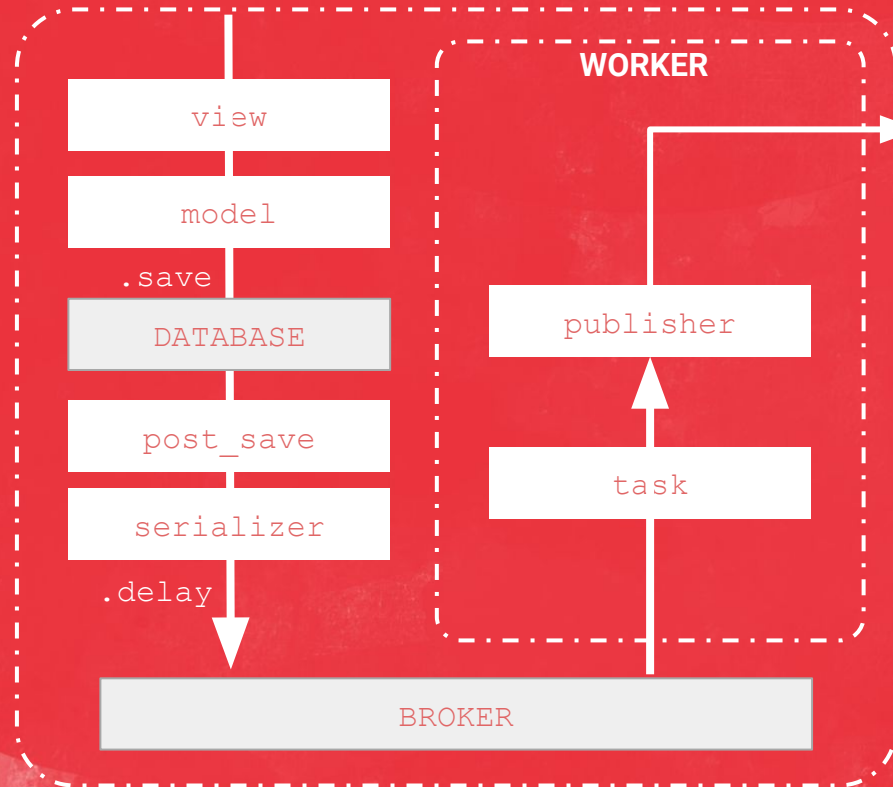
## Throughput



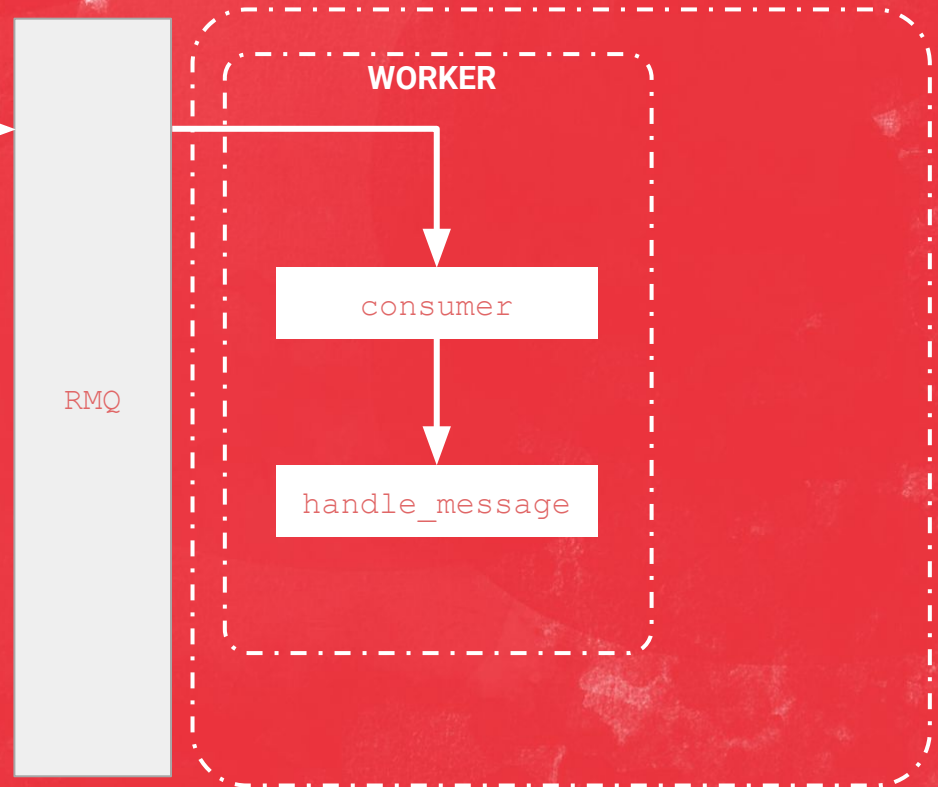
# Solution?



## CONSUMPTION



## RECOMMENDATION



# Pattern?



```
class Watch(Model):
    user_id = IntegerField()
    lesson_id = IntegerField()
    position = IntegerField()

class WatchSerializer(ModelSerializer):
    class Meta:
        model = Watch

    @receiver(post_save, sender=Watch)
    def publish_watch(sender, instance, created, **kwargs):
        PublishModelTask().delay(
            data=<body>,
            routing_key=<routing_key>,
            exchange=<exchange>
        )
```

## Message

body

```
{
    user_id: 42,
    lesson_id: 100,
    position: 314
}
```

routing\_key

created

exchange

watch

queue

recommendation.watch.\*

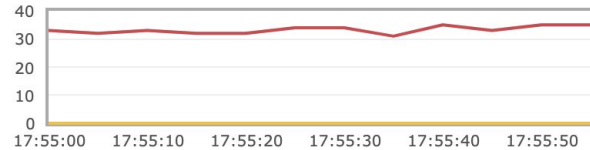
# Results?



## Overview

▼ Totals

Queued messages last minute ?



Ready

0

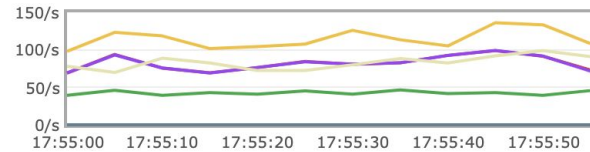
Unacked

32

Total

32

Message rates last minute ?



Publish

108/s

Publisher confirm

0.00/s

Deliver (manual ack)

73/s

Deliver (auto ack)

46/s

Consumer ack

72/s

Redelivered

0.00/s

Get (manual ack)

0.00/s

Get (auto ack)

0.00/s

Return

0.00/s

Disk read

0.00/s

Disk write

91/s



CÍRCULO  
**ENGENHARIA**  
DE SOFTWARE

VIVA  
DA SUA  
PAIXÃO

**CONTACT**  
joao@daher.dev

**WE ARE HIRING**  
<http://eduk.breezy.hr>