To Build a production ready distributed task queueing system with celery.

Hello!

I am Vishrut Kohli

Software Engineer at **Grofers**.

You can find me at

https://www.linkedin.com/in/lazycoder07/

https://www.reddit.com/user/ramenandcode

http://vishrutkohli.github.io/

Production Ready?

- Highly Efficient
- Scalable
- Transparent
- Resilient.

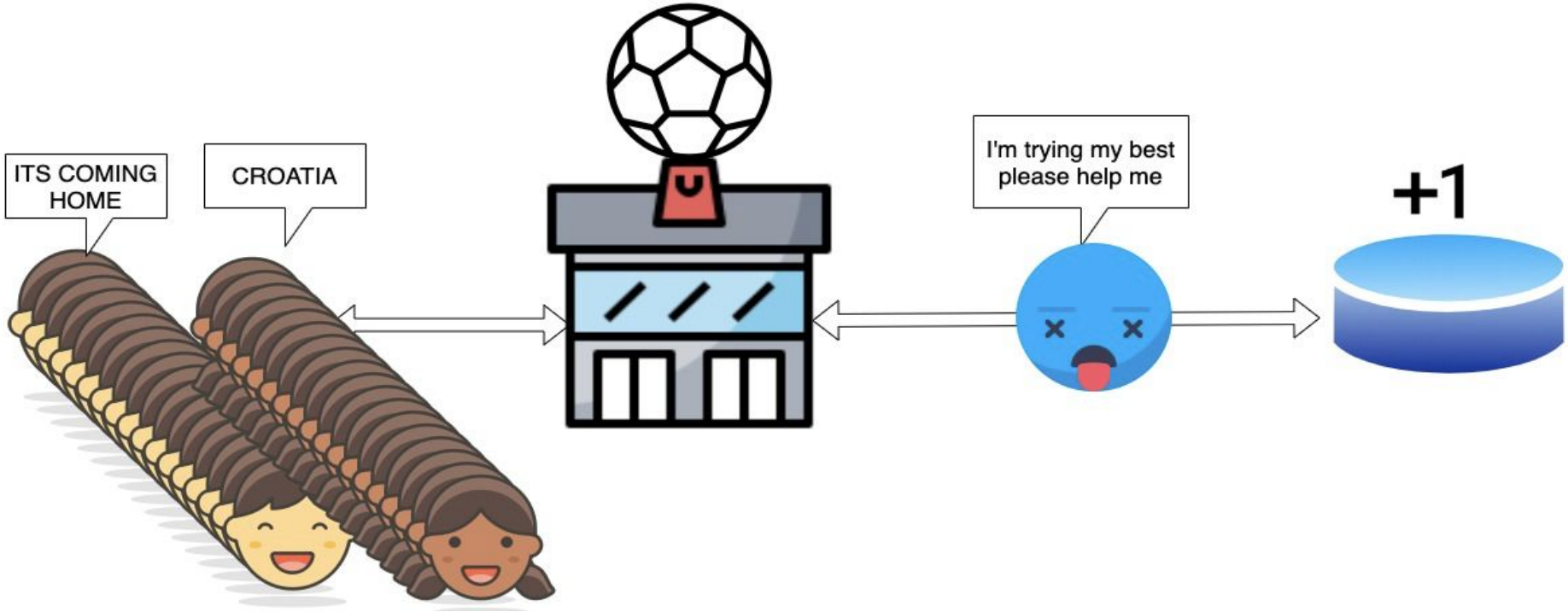
Talk content

- What are task queues and why we need them?
- What is and why Celery?
- **Building** a distributed task queueing system.
- **Tuning** a distributed task queueing system for better efficiency.
- Adding **resiliency** to the system
- What to do in times of **SOS**?
- **Monitoring** the system we built.
- Most importantly **bad jokes**.

Prerequisites

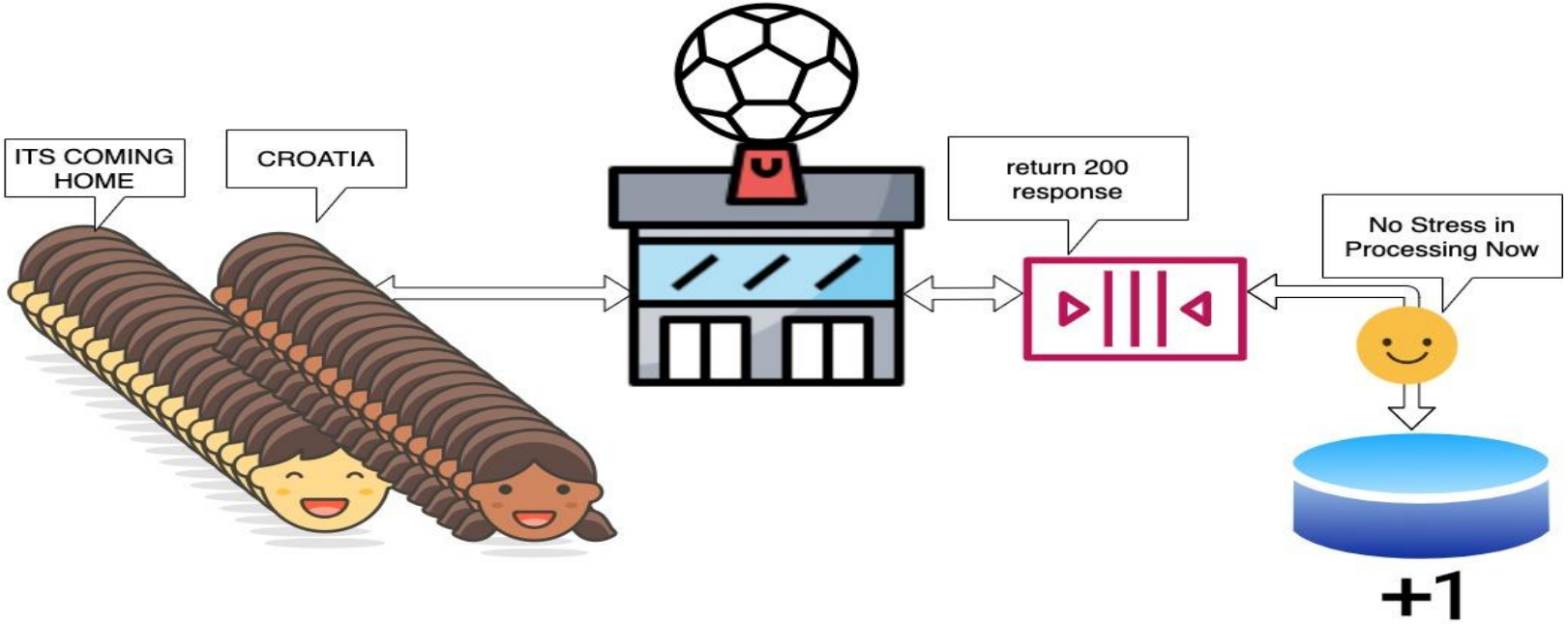
- Basic knowledge of python.
- Basic knowledge of web development.
- Worked or heard about celery before.
- A sense of Humour and love for Gifs.

Why Task Queues?




THERE'S GOTTA BE A BETTER WAY!

Task Queues come to rescue?



What is and why celery?

Keywords we will use in this talk

- Task Queues
- Task
- Worker
- Broker
- result_backend

Which broker to choose?

- RabbitMQ
- Redis
- etc.

Each one is great for their specific use case .

23:00

Step 1: **Build**

Let's think of an **Ecommerce(Grofers) warehouse** to build.
There are going to be 3 things which happen there.

1. **Picking** of the products
2. **packing** of the products
3. **delivery** of the order.

One boy doing all the work

PICKING

PACKING

DELIVERY

One boy and one girl doing all the work

PICKING

PACKING

DELIVERY

Specialised people doing their work

PICKING

PACKING

DELIVERY

Why Pipelines?

1. It gives us the ability to see bottlenecks and scale smaller components. Instead of the whole system.
2. This will give the ability to give different kind of machines to different fragments.
3. It helps us keep track of the status of the tasks and will add some kind of resiliency(Self-healing capability) to the system by enabling retries at every step.

Running two pipelines in ||

Code for our application

```
import logging
from MyWebframework import success_response
from tasks import picking

logger = logging.getLogger(__name__)

def OrderRecieverApi(order):
  logger.info("I have recieved the order lets delegate it to celery")
  picking.delay(order)
  return success_response()
```

Code for our pipeline

```
@app.task(queue="picking_queue")
def Picking(order):
    stuff = lets_pick_stuff_from_the_aisle(order)
    Packing.delay(stuff)

@app.task(queue="packing_queue")
def Packing(stuff):
    packed_stuff = lets_pack_stuff_we_got(stuff)
    Delivery.delay(packed_stuff)

@app.task(queue="delivery_queue")
def Delivery(packed_stuff):
    lets_deliver_the_Stuff_on_time_and_make_customer_happy(packed_stuff)
```

Code for our pipeline

```
@app.task(queue="picking_queue")
def Picking(order):
    stuff = lets_pick_stuff_from_the_aisle(order)
    Packing.delay(stuff)

@app.task(queue="packing_queue")
def Packing(stuff):
    packed_stuff = lets_pack_stuff_we_got(stuff)
    Delivery.delay(packed_stuff)

@app.task(queue="delivery_queue")
def Delivery(packed_stuff):
    lets_deliver_the_Stuff_on_time_and_make_customer_happy(packed_stuff)
```

Code for our pipeline

```
@app.task(queue="picking_queue")
def Picking(order):
    stuff = lets_pick_stuff_from_the_aisle(order)
    Packing.delay(stuff)

@app.task(queue="packing_queue")
def Packing(stuff):
    packed_stuff = lets_pack_stuff_we_got(stuff)
    Delivery.delay(packed_stuff)

@app.task(queue="delivery_queue")
def Delivery(packed_stuff):
    lets_deliver_the_Stuff_on_time_and_make_customer_happy(packed_stuff)
```

18:00

Step 2: **Tune**

Some tips , tricks and configuration settings to get most out of celery.

Always benchmark before moving to further optimization

RabbitMQ™ RabbitMQ 3.8.5 Erlang 23.0.3

Refreshed 2020-07-31 16:56:28 Refresh every 5 seconds

Virtual host All

Cluster **rabbit@pricing-rabbitmq-rc**

User **pricing_admin** Log out

Overview **Connections** **Channels** **Exchanges** **Queues** **Admin**

Queues

All queues (73)

Pagination

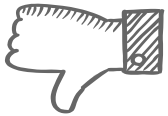
Page 1 of 1 - Filter: Regex ?

Displaying 73 items , page size up to: 100

Overview					Messages			Message rates			+/-
Virtual host	**Name**	**Type**	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**	
/v3	Picking Queue	classic	AD TTL Exp	idle	2550	50	2600				
/v3	Packing Queue	classic	AD TTL Exp	idle	375	5	380				
/v3	Delivery Queue	classic	AD TTL Exp	idle	336	100	436				

Can we use batching ?

- Takes stress off your Database by enabling batch CRUD operations.



- Retries and failures will also happen on batches even when one part of batch fails.

New code for our application

```
import logging
from MyWebFramework import success_response
from tasks import OrderAggregator

logger = logging.getLogger(__name__)

def OrderRecieverApi(order):
    logger.info("lets delegate the order to celery")
    OrderAggregator.delay(order)
    return success_response()
```

New code for our pipeline

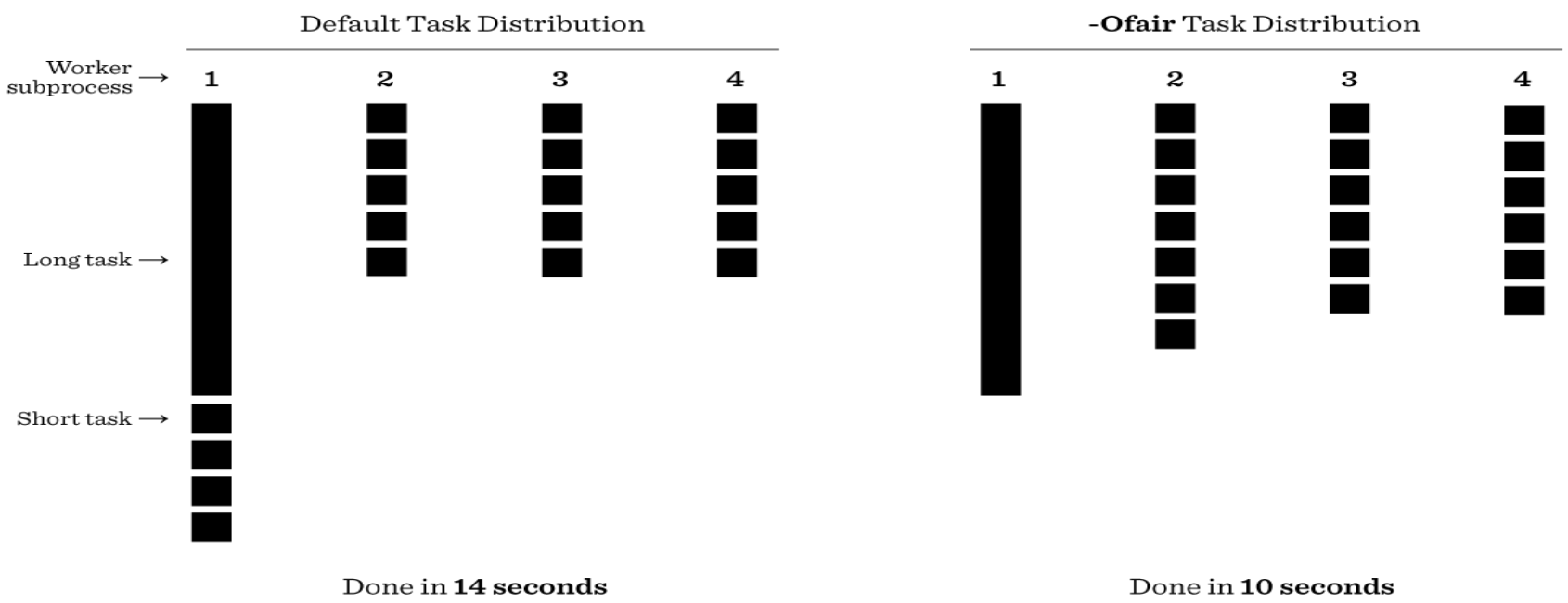
```
@app.task(queue="aggregator_queue")
def OrderAggregator(order):
order_chunk = order_chunking_logic(order)
Picking.delay(order_chunk)

@app.task(queue="picking_queue")
def Picking(order_chunk):
stuff = lets_pick_stuff_from_the_aisle(order_chunk)
Packing.delay(stuff)

@app.task(queue="packing_queue")
def Packing(stuff):
packed_stuff = lets_pack_stuff_we_got(stuff)
Delivery.delay(packed_stuff)

@app.task(queue="delivery_queue")
def Delivery(packed_stuff):
lets_deliver_the_Stuff_on_time_and_make_customer_happy(packed_stuff)
```

Always split tasks into IO-bound and CPU bound tasks

- Gvent
- Eventlet
- Prefork

```
celery -A proj worker -P <pool> -c <concurrency_needed>
```

Use -Ofair optimization when possible.



Keep track of results only if you need them

```
CELERY_IGNORE_RESULT = True
```

13:00

Step 3: **Resiliency**

"Software errors are inevitable. Chaos is not."

-***sentry.io***

```python
@app.task(queue="picking_queue",
          autoretry_for=(Exception,),
          retry_kwargs={'max_retries': 5})
def Picking(order_chunk):
    stuff = lets_pick_stuff_from_the_aisle(order_chunk)
    Packing.delay(stuff)
```

```
@app.task(queue="picking_queue",
          autoretry_for=(Exception,),
          retry_kwargs={'max_retries': 5}
          exponential_backoff=10)
def Picking(order_chunk):
    stuff = lets_pick_stuff_from_the_aisle(order_chunk)
    Packing.delay(stuff)
```

```
@app.task(queue="picking_queue",
          autoretry_for=(Exception,),
          retry_kwargs={'max_retries': 5}
          exponential_backoff=10,
          acks_late=True)
def Picking(order_chunk):
    stuff = lets_pick_stuff_from_the_aisle(order_chunk)
    Packing.delay(stuff)
```

```
@app.task(queue="picking_queue",
          autoretry_for=(Exception,),
          retry_kwargs={'max_retries': 5}
          exponential_backoff=10,
          acks_late=True,
          retry_jitter=True)
def Picking(order_chunk):
    stuff = lets_pick_stuff_from_the_aisle(order_chunk)
    Packing.delay(stuff)
```

Use a DLQ to capture Circuit breaked failures.

09:00

Step 4: **SOS**

Always use Max tasks per child/max memory per child when you suspect a memory leak in your task

```
celery -A proj worker -P <pool> -c <concurrency_needed>
--max-tasks-per-child=<Number_of_tasks>
```

```
celery -A proj worker -P <pool> -c <concurrency_needed>
--max-memory-per-child=<Memory_in_Kib>
```

07:00

Step 5: **Monitor**

Celery Flower

localhost:5555/workers

Celery Flower | Workers | Tasks | Monitor | Docs | About

Workers

Shut Down

	Name	Status	Concurrency	Completed Tasks	Running Tasks	Queues
	celery1.pi.local	Online	4	13902	0	images, data, video
	celery2.pi.local	Online	4	13900	0	images, data, video
	celery3.pi.local	Online	4	13826	0	images, data, video
	celery4.pi.local	Online	1	1989	0	data
	celery5.pi.local	Online	1	1983	0	data
	celery6.pi.local	Offline	3	2245	3	
	celery7.pi.local	Online	3	2283	3	celery, data
	celery8.pi.local	Online	3	2279	3	celery
	celery9.pi.local	Online	3	2287	3	celery

```
flower -A PyjamasLive --port=5555
```

RabbitMQ™ RabbitMQ 3.8.5 Erlang 23.0.3

Refr

Overview | Connections | Channels | Exchanges | **Queues** | Admin

/v3	celery_rule_processing@celery-rule-processing-57b7864cd8-9944v.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	celery_rule_processing@celery-rule-processing-sbc-56d67db6f6-kpnf4.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	celery_rule_processing@celery-rule-processing-sbc-9d56b6ffd-j5kk7.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	celery_rule_processing@celery-rule-processing-sbc-9d56b6ffd-wl6ml.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	celery_run_id_processing@celery-run-id-processing-sbc-786499876c-2mjx8.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	celery_run_id_processing@celery-run-id-processing-sbc-786499876c-vfg6c.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	evaluation_queue	classic	D	idle	0	0	0	0.00/s	11/s	0.00/s
/v3	event_manager_queue	classic	D	idle	0	0	0	0.20/s	0.20/s	0.00/s
/v3	fallback_queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/v3	fallback_queue@celery-fallback-7df65476d-rzxpc.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	inventory_update_scheduler_queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/v3	inventory_update_scheduler_queue@celery-inventory-update-76bf959c48-87x44.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	price_query_queue	classic	D	running	0	10	10	20/s	14/s	7.6/s
/v3	pricing-celery@product-pricing-celery-7f54cb8d5d-5vl6d.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	pricing-celery@product-pricing-celery-7f54cb8d5d-znt8b.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	pricing_celery	classic	D	running	0	0	0	0.00/s	0.40/s	0.00/s
/v3	redis_key_queue	classic	D	idle	0	0	0			
/v3	resolution_queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/v3	rule_processing_queue	classic	D	running	498	80	578	0.00/s	0.20/s	0.20/s
/v3	run_id_queue	classic	D	idle	0	0	0			
/v3	sbc_evaluation_queue	classic	D	idle	0	0	0			
/v3	sbc_pricing_celery	classic	D	running	0	0	0	0.00/s	0.20/s	0.00/s
/v3	sbc_pricing_celery@celery-sbc-pricing-68686fc449-5vsv6.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			
/v3	sbc_redis_key_queue	classic	D	idle	0	0	0			
/v3	sbc_rule_processing_queue	classic	D	idle	0	0	0			
/v3	sbc_run_id_queue	classic	D	idle	0	0	0			
/v3	stock-in-queue	classic	D	idle	0	0	0	0.00/s	0.00/s	0.00/s
/v3	stock-in-queue@celery-stock-in-6964bb6496-9srmm.celery.pidbox	classic	AD TTL Exp	idle	0	0	0			

Step 6: **SLEEP SOUNDLY**

questions? tomatoes

05:00

00:00

Thank you