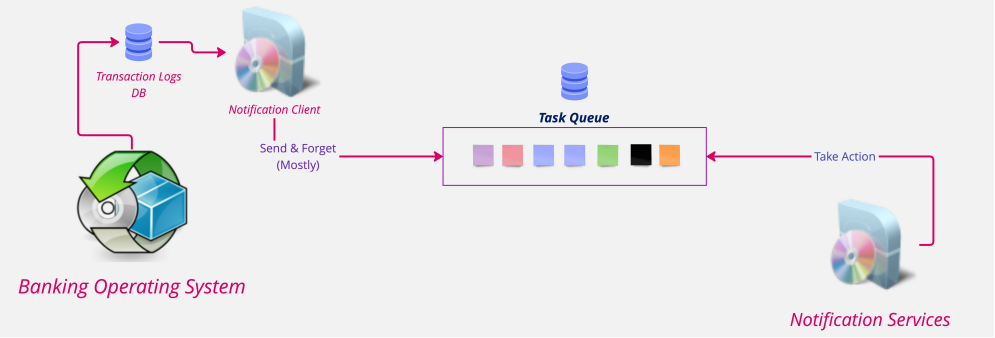
Chapter - 4 : Two way Communication

Moving beyond fire and forget



Return_Value

There are genuine reasons to get the results of the actions being done via celery tasks. For e.g. A confirmation of the SMS being actually sent by the celery Task

_ Please Note:- The producers are generally not expected to wait for the results to make the call `Asynchronous`, we are doing it here only for the example use cases_

There are two ways to do it

1. Inline Waiting for the notification

This is equivalent to having a blocking call

2. Look for notification later

Once we complete the work we're doing (banking transaction here), then we can get the information about the notifications being sent

*In the case of RabbitMQ, celery uses **RPC*** to return back the results. Celery does it automatically without any manual change in RabbitMQ. We just need to make sure that we provide the place for the same_

Here is how the RPC backend can be used in the celery.

```
app = Celery('cel_main', backend='rpc://', broker='pyamqp://')
```

Here is how the changed files will look like

Example : 1

task_one.py

```python
app = Celery('cel_main', backend='rpc://', broker='pyamqp://')

# senf SMS notifications to the account holder
@app.task
def task_send_sms(account_no, message):
    time.sleep(5)  # sleep for 5 seconds
    print("Sending SMS to account holder of {0} with message \n\n {1}".format(account_no, message))
    return "SMS Sent Successfully!!!!!!!"

@app.task
def task_send_whatsapp(account_no, message):
    time.sleep(5)  # sleep for 5 seconds
    print("Sending WhatsApp to account holder of {0} with message \n\n {1}".format(account_no, message))
    return "WhatsApp Message Sent Successfully!!!!!!!"
```

entry_point.py

```python
# Deposit money into the bank account and send the sms & whatsapp
def deposit_and_send_sms(account_no, amount, message):
    bank_deposit_money(account_no, amount)
    # send SMS and WhatsApp

    sms_sent_status = task_send_sms.delay(account_no, message)
    whatsapp_sent_status = task_send_whatsapp.delay(account_no, message)

    print("SMS Status: ", sms_sent_status.get())
    print("WhatsApp Status: ", whatsapp_sent_status.get())
```

Return_Value

Also, we don't need to consume the return value immediately, we can do it later also

-

Example : 2

```python
sms_sent_status = None
whatsapp_sent_status = None


# Deposit money into the bank account and send the sms & whatsapp
def deposit_and_send_sms(account_no, amount, message):
    bank_deposit_money(account_no, amount)
    # send SMS and WhatsApp

    global sms_sent_status
    sms_sent_status = task_send_sms.delay(account_no, message)
    global whatsapp_sent_status
    whatsapp_sent_status = task_send_whatsapp.delay(account_no, message)


def check_sms_whatsapp_status():
    time.sleep(10)
    print("SMS Status: ", sms_sent_status.get())
    print("WhatsApp Status: ", whatsapp_sent_status.get())


if __name__ == "__main__":
    deposit_and_send_sms(1234, 5000, "Your account has been credited with $5000")
    # check the status of the SMS and WhatsApp
    check_sms_whatsapp_status()
```

Return_Late

you can also check the status of tasks whether its completed or not

Classroom Exercise

Instead of .delay(), use apply_async to run the same code as(1) and emulate the sequential behaviour without turning off the concurrency