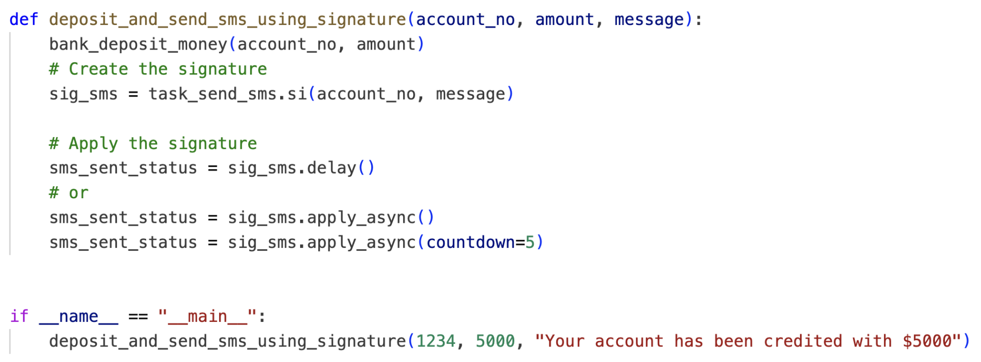
Chapter - 5 : Signatures & Chaining of the functionality

A `signature()` wraps the arguments, keyword arguments, and execution options of a single task invocation



```python
def deposit_and_send_sms_using_signature(account_no, amount, message):
    bank_deposit_money(account_no, amount)
    # Create the signature
    sig_sms = task_send_sms.si(account_no, message)

    # Apply the signature
    sms_sent_status = sig_sms.delay()
    # or
    sms_sent_status = sig_sms.apply_async()
    sms_sent_status = sig_sms.apply_async(countdown=5)


if __name__ == "__main__":
    deposit_and_send_sms_using_signature(1234, 5000, "Your account has been credited with $5000")
```

creating Signatures

Example:1

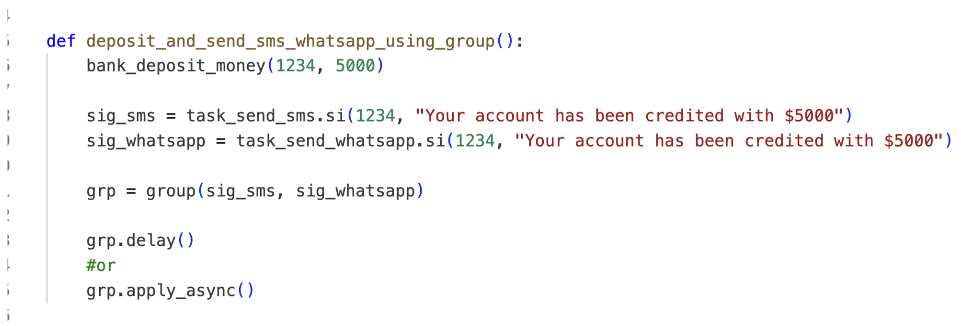
With signature, we can directly invoke the tasks.

Signatues not only makes it easy to invoke the tasks but also help us in grouping the sequence of tasks

Grouping Tasks using Signatures

In the banking example, we don't need to put the `sms_send` and `whatsapp_send` sequentially to the queue, we can club these two call together and send them at once

Also, since sending SMS an Whatsapp is not dependent on each other, they can run in parallel also in any order. A group allows the items in the group to be called concurrently

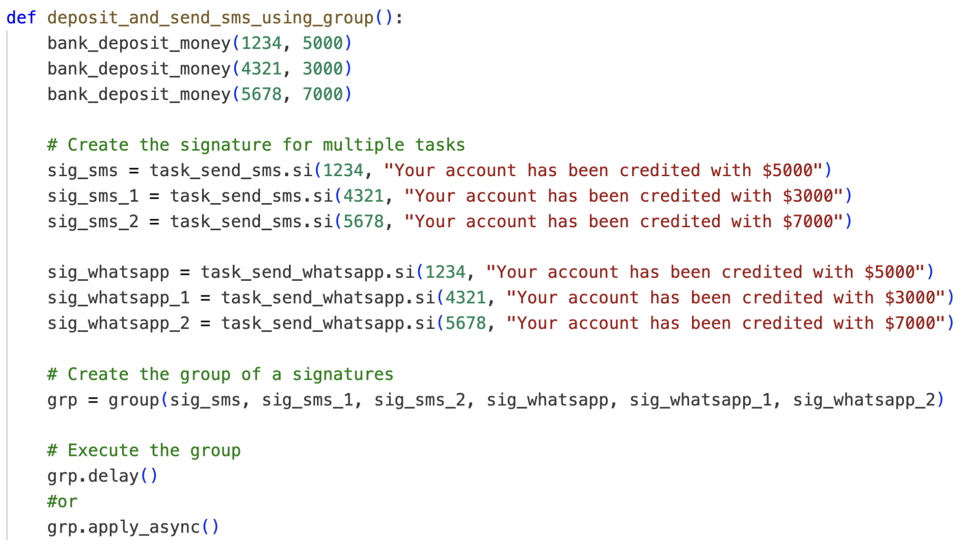


```python
def deposit_and_send_sms_whatsapp_using_group():
    bank_deposit_money(1234, 5000)

    sig_sms = task_send_sms.si(1234, "Your account has been credited with $5000")
    sig_whatsapp = task_send_whatsapp.si(1234, "Your account has been credited with $5000")

    grp = group(sig_sms, sig_whatsapp)

    grp.delay()
    #or
    grp.apply_async()
```

Group Signatures

Example:2

It’s possible to group as many signatures as possible



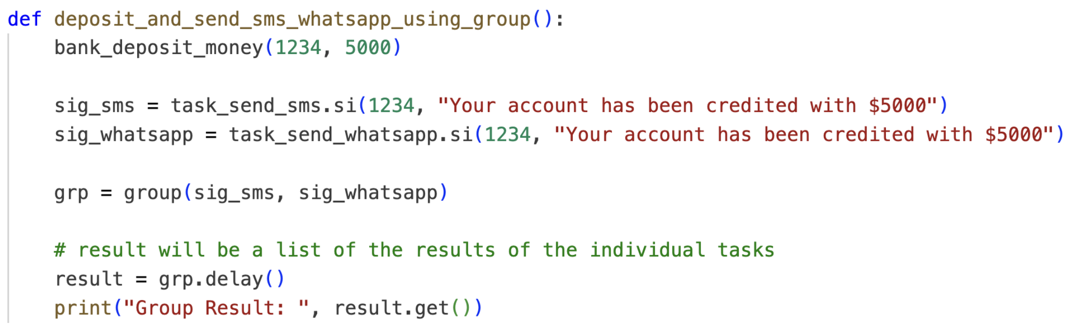
```python
def deposit_and_send_sms_using_group():
    bank_deposit_money(1234, 5000)
    bank_deposit_money(4321, 3000)
    bank_deposit_money(5678, 7000)

    # Create the signature for multiple tasks
    sig_sms = task_send_sms.si(1234, "Your account has been credited with $5000")
    sig_sms_1 = task_send_sms.si(4321, "Your account has been credited with $3000")
    sig_sms_2 = task_send_sms.si(5678, "Your account has been credited with $7000")

    sig_whatsapp = task_send_whatsapp.si(1234, "Your account has been credited with $5000")
    sig_whatsapp_1 = task_send_whatsapp.si(4321, "Your account has been credited with $3000")
    sig_whatsapp_2 = task_send_whatsapp.si(5678, "Your account has been credited with $7000")

    # Create the group of a signatures
    grp = group(sig_sms, sig_sms_1, sig_sms_2, sig_whatsapp, sig_whatsapp_1, sig_whatsapp_2)

    # Execute the group
    grp.delay()
    #or
    grp.apply_async()
```

Multiple Group Signatures

Example:3

Taking Group Task Results

It’s possible that tasks being executed are returning back the results, we can get the results of the multiple tasks using a single `.get()` and the results will be returned as a list



```python
def deposit_and_send_sms_whatsapp_using_group():
    bank_deposit_money(1234, 5000)

    sig_sms = task_send_sms.si(1234, "Your account has been credited with $5000")
    sig_whatsapp = task_send_whatsapp.si(1234, "Your account has been credited with $5000")

    grp = group(sig_sms, sig_whatsapp)

    # result will be a list of the results of the individual tasks
    result = grp.delay()
    print("Group Result: ", result.get())
```

In a group, all tasks will be running concurrently

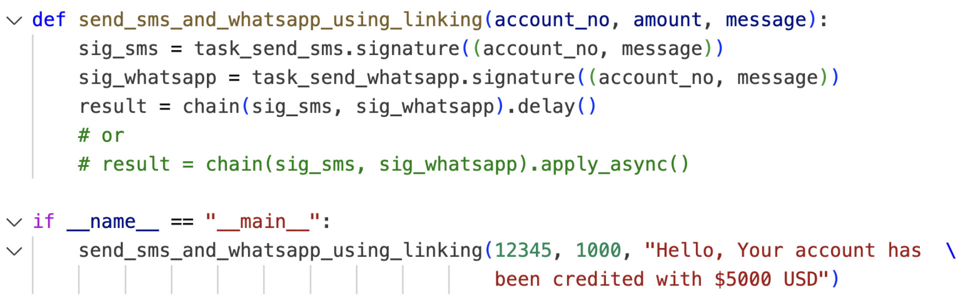
Example:4

Chaining of Tasks.

Sometimes we need to run things sequentially, which takes into consideration the results of the previous activity. For example, in this case, we want to send WhatsApp message and after sending the SMS and in the WhatsApp message we want to add that an SMS is also sent.

In these scenarios, we can chain the tasks where it not only executes sequentially, but can also chain the output of previous task as input to the next one

Here is how we can chain the calls using signatures



```python
def send_sms_and_whatsapp_using_linking(account_no, amount, message):
    sig_sms = task_send_sms.signature((account_no, message))
    sig_whatsapp = task_send_whatsapp.signature((account_no, message))
    result = chain(sig_sms, sig_whatsapp).delay()
    # or
    # result = chain(sig_sms, sig_whatsapp).apply_async()

if __name__ == "__main__":
    send_sms_and_whatsapp_using_linking(12345, 1000, "Hello, Your account has  \
                                    been credited with $5000 USD")
```

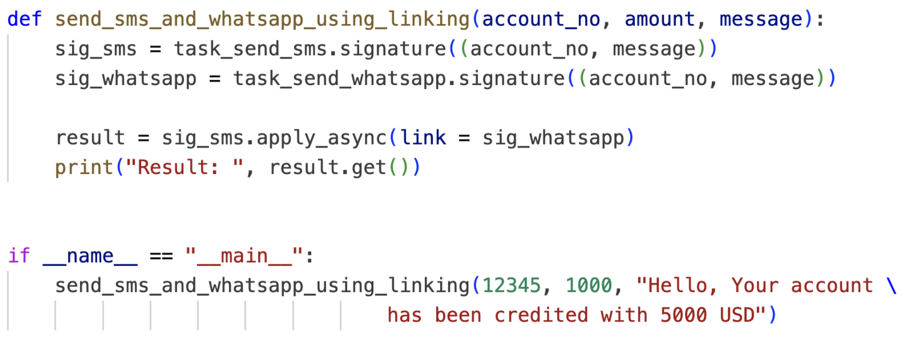
Chaining

Example:5

We can also return an array as a result of the last task being executed. Here is how we can do the same

```
return [sms_sent_status, "WhatsApp Sent Successfully!!!!!!!"]
```

Similar chaining can also be observed by linking the signatures together, but there is a difference, the difference will be visible in the return value



```python
def send_sms_and_whatsapp_using_linking(account_no, amount, message):
    sig_sms = task_send_sms.signature((account_no, message))
    sig_whatsapp = task_send_whatsapp.signature((account_no, message))

    result = sig_sms.apply_async(link = sig_whatsapp)
    print("Result: ", result.get())


if __name__ == "__main__":
    send_sms_and_whatsapp_using_linking(12345, 1000, "Hello, Your account \
                             has been credited with 5000 USD")
```

Linking

Example:6

__By looking at the output of both `chain` and `link`, we now know which one we should use under what circumstances._

Chords - Group Callback execution

A chord is a task that only executes after all of the tasks in a group have finished executing.
_chords are not supported by `RabbitMQ` , however, it will work with `redis and others`,

Grouping the tasks to take the run in parellel

```
from celery import chord, group

def send_sms_and_whatsapp_using_chain(account_no, amount, message):
    # group 1 to sent message
    sig_sms = task_send_sms.signature((account_no, message))
    sig_whatsapp = task_send_whatsapp.signature((account_no, message))

    group_msg_send = group(sig_sms, sig_whatsapp)

    # group 2 to count the number of messages sent
    sig_sms_count = total_sms_sent.signature()
    sig_whatsapp_count = total_whatsapp_sent.signature()

    group_msg_status = group(sig_sms_count, sig_whatsapp_count)

    # create a chord with the two groups
    chord_result = chord(group_msg_send, group_msg_status)

    result = chord_result.delay()

    print("Result: ", result.get())
```

Example:7