CSE 484 Assignment 4

Message Broker - https://tsh.io/blog/message-broker/

Task 1: Write two small programs in Python; a producer (sender) that sends a single message, and a consumer (receiver) that receives messages and prints them out. It's a "Hello World" of messaging.

Steps:

Checking if our Cloud VM Instance has Python installed or not

```
root@159.138.121.53's password:
Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.4.0-100-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Sun 27 Nov 2022 01:17:52 PM CST

  System load:  0.83              Processes:               148
  Usage of /:   3.9% of 196.61GB  Users logged in:         0
  Memory usage: 4%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for eth0:    192.168.1.150

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

176 updates can be applied immediately.
119 of these updates are standard security updates.
To see these additional updates run: apt list --upgradable


The list of available updates is more than a week old.
To check for new updates run: sudo apt update


        Welcome to Huawei Cloud Service

Last login: Thu Nov 17 14:59:14 2022 from 202.134.10.137
root@ecs-docker-muballigh:~# docker images
REPOSITORY                                TAG        IMAGE ID       CREATED         SIZE
img-static-site-example                   latest     4fe50929e639   10 days ago     149MB
muballighhossain/sample-website-muballigh 0.1        4fe50929e639   10 days ago     149MB
muballigh-test-site                       latest     ffc73a938767   10 days ago     149MB
<none>                                    <none>     dd4825ca80f1   10 days ago     149MB
registry                                  2          81c944c2288b   2 weeks ago     24.1MB
<none>                                    <none>     f6b9880e1b96   2 weeks ago     163MB
ubuntu                                    latest     a8780b506fa4   3 weeks ago     77.8MB
nginx                                     latest     76c69feac34e   4 weeks ago     142MB
httpd                                     latest     fe8735c23ec5   4 weeks ago     145MB
muballighhossain/mysql-test-muba          firsttry   3f3946d5572f   6 weeks ago     517MB
mysql/mysql-server                        latest     3f3946d5572f   6 weeks ago     517MB
localhost:5000/mysql/mysql-server         latest     3f3946d5572f   6 weeks ago     517MB
hello-world                               latest     feb5d9fea6a5   14 months ago   13.3kB
centos                                    latest     5d0da3dc9764   14 months ago   231MB
root@ecs-docker-muballigh:~# python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

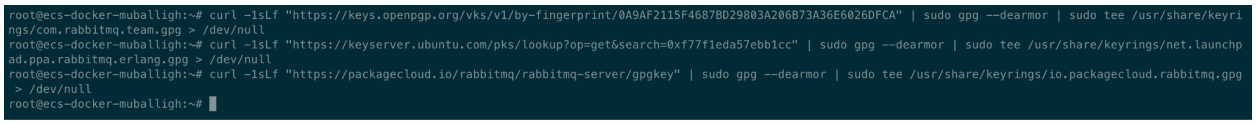
Installing rabbitmq

```
apt-get install curl gnupg apt-transport-https -y
```

```
root@ecs-docker-muballigh:~# apt-get install curl gnupg apt-transport-https -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
curl is already the newest version (7.68.0-1ubuntu2.14).
gnupg is already the newest version (2.2.19-3ubuntu2.2).
The following NEW packages will be installed:
  apt-transport-https
0 upgraded, 1 newly installed, 0 to remove and 170 not upgraded.
Need to get 1,704 B of archives.
After this operation, 162 kB of additional disk space will be used.
Get:1 http://repo.huaweicloud.com/ubuntu focal-updates/universe amd64 apt-transport-https all 2.0.9 [1,704 B]
Fetched 1,704 B in 0s (94.6 kB/s)
Selecting previously unselected package apt-transport-https.
(Reading database ... 124793 files and directories currently installed.)
Preparing to unpack .../apt-transport-https_2.0.9_all.deb ...
Unpacking apt-transport-https (2.0.9) ...
Setting up apt-transport-https (2.0.9) ...
root@ecs-docker-muballigh:~#
```

Add repository signing keys for RabbiMQ main, ErLang, and RabbitMQ PackageCloud repositories respectively:

```
curl -1sLf
"https://keys.openpgp.org/vks/v1/by-fingerprint/0A9AF2115F4687BD29803A206
B73A36E6026DFCA" | sudo gpg --dearmor | sudo tee
/usr/share/keyrings/com.rabbitmq.team.gpg > /dev/null
curl -1sLf
"https://keyserver.ubuntu.com/pks/lookup?op=get&search=0xf77f1eda57ebb1cc
" | sudo gpg --dearmor | sudo tee
/usr/share/keyrings/net.launchpad.ppa.rabbitmq.erlang.gpg > /dev/null

curl -1sLf "https://packagecloud.io/rabbitmq/rabbitmq-server/gpgkey" |

sudo gpg --dearmor | sudo tee

/usr/share/keyrings/io.packagecloud.rabbitmq.gpg > /dev/null
```

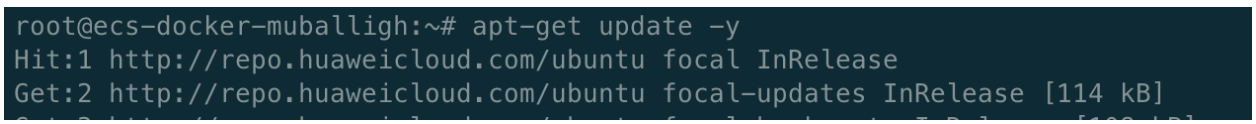


Create a new file at /etc/apt/sources.list.d/rabbitmq.list and add the following repositories for ErLang and RabbitMQ respectively that are suited for Ubuntu 22.04 jammy release:

```
deb [signed-by=/usr/share/keyrings/net.launchpad.ppa.rabbitmq.erlang.gpg]
http://ppa.launchpad.net/rabbitmq/rabbitmq-erlang/ubuntu jammy main
deb-src
[signed-by=/usr/share/keyrings/net.launchpad.ppa.rabbitmq.erlang.gpg]
http://ppa.launchpad.net/rabbitmq/rabbitmq-erlang/ubuntu jammy main
deb [signed-by=/usr/share/keyrings/io.packagecloud.rabbitmq.gpg]
https://packagecloud.io/rabbitmq/rabbitmq-server/ubuntu/ jammy main

deb-src [signed-by=/usr/share/keyrings/io.packagecloud.rabbitmq.gpg]

https://packagecloud.io/rabbitmq/rabbitmq-server/ubuntu/ jammy main
```

Save the file and you are ready to update your repository listings:

```
apt-get update -y
```



After your repository listings are updated, continue with installing required ErLang packages:

```
apt-get install -y erlang-base \
    erlang-asn1 erlang-crypto erlang-eldap erlang-ftp erlang-inets \
    erlang-mnesia erlang-os-mon erlang-parsetools erlang-public-key \
    erlang-runtime-tools erlang-snmp erlang-ssl \
    erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl
```

```
root@ecs-docker-muballigh:~# apt-get install -y erlang-base \
>     erlang-asn1 erlang-crypto erlang-eldap erlang-ftp erlang-inets \
>     erlang-mnesia erlang-os-mon erlang-parsetools erlang-public-key \
>     erlang-runtime-tools erlang-snmp erlang-ssl \
>     erlang-syntax-tools erlang-tftp erlang-tools erlang-xmerl
Reading package lists... Done
Building dependency tree
Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:

The following packages have unmet dependencies:
 erlang-base : Depends: libc6 (>= 2.34) but 2.31-0ubuntu9.2 is to be installed
               Depends: libstdc++6 (>= 11) but 10.3.0-1ubuntu1~20.04 is to be installed
               Recommends: libsctp1 (>= 1.0.19+dfsg) but 1.0.18+dfsg-1 is to be installed
 erlang-crypto : Depends: libssl3 (>= 3.0.0~~alpha1) but it is not installable
 erlang-os-mon : Depends: libc6 (>= 2.34) but 2.31-0ubuntu9.2 is to be installed
E: Unable to correct problems, you have held broken packages.
root@ecs-docker-muballigh:~#
```

Finally, we can install RabbitMQ server and its dependencies:

```
apt-get install rabbitmq-server -y --fix-missing
```

```
root@ecs-docker-muballigh:~# apt-get install rabbitmq-server -y --fix-missing
Reading package lists... Done
Building dependency tree
Reading state information... Done
Some packages could not be installed. This may mean that you have
requested an impossible situation or if you are using the unstable
distribution that some required packages have not yet been created
or been moved out of Incoming.
The following information may help to resolve the situation:
```

If all went well, you should see a rabbitmq-server process up and running:

```
systemctl status rabbitmq-server
```

```
root@ecs-docker-muballigh:~# systemctl status rabbitmq-server
● rabbitmq-server.service - RabbitMQ Messaging Server
     Loaded: loaded (/lib/systemd/system/rabbitmq-server.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2022-11-28 13:17:41 CST; 19min ago
   Main PID: 2872 (beam.smp)
     Status: "Initialized"
      Tasks: 91 (limit: 9442)
     Memory: 88.5M
     CGroup: /system.slice/rabbitmq-server.service
             ├─2853 /bin/sh /usr/sbin/rabbitmq-server
             ├─2872 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 64 -MBas ageffcbf -MHas ageffcbf -MBlmbcs 512 -MHlmbcs 512 -MMmcs 30 -P 1048576 -t 5000000 -stbt db -zdbbl 1
             ├─3155 erl_child_setup 65536
             ├─3199 inet_gethost 4
             └─3200 inet_gethost 4
```

To check the list of all available RabbitMQ plugins, run the following command:

```
rabbitmq-plugins list
```

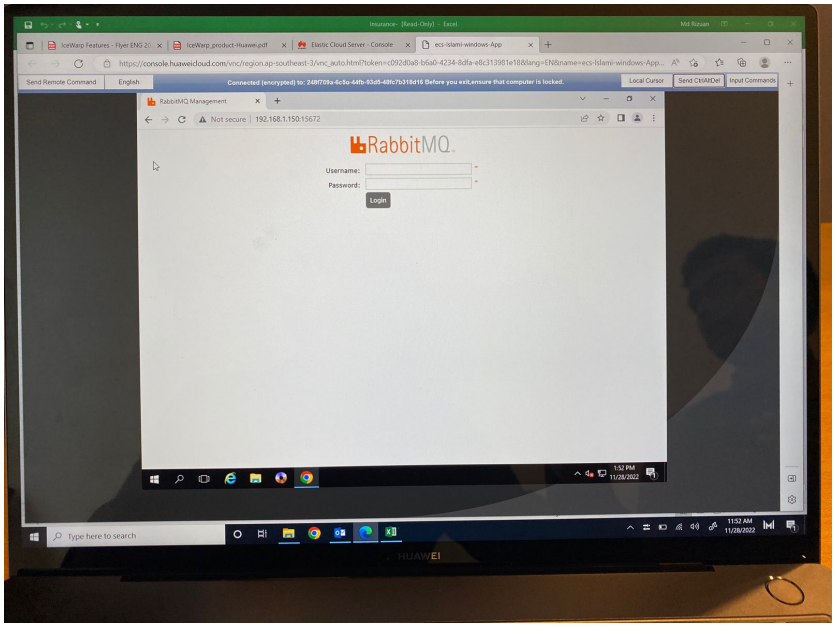
```
root@ecs-docker-muballigh:~# rabbitmq-plugins li
Listing plugins with pattern ".*" ...
 Configured: E = explicitly enabled; e = implici
  | Status: * = running on rabbit@ecs-docker-muba
  |/
[  ] rabbitmq_amqp1_0                  3.8.2
[  ] rabbitmq_auth_backend_cache       3.8.2
[  ] rabbitmq_auth_backend_http        3.8.2
[  ] rabbitmq_auth_backend_ldap        3.8.2
[  ] rabbitmq_auth_backend_oauth2      3.8.2
[  ] rabbitmq_auth_mechanism_ssl       3.8.2
[  ] rabbitmq_consistent_hash_exchange 3.8.2
[  ] rabbitmq_event_exchange           3.8.2
[  ] rabbitmq_federation               3.8.2
[  ] rabbitmq_federation_management    3.8.2
[  ] rabbitmq_jms_topic_exchange       3.8.2
[E*] rabbitmq_management               3.8.2
[e*] rabbitmq_management_agent         3.8.2
[  ] rabbitmq_mqtt                     3.8.2
[  ] rabbitmq_peer_discovery_aws       3.8.2
[  ] rabbitmq_peer_discovery_common    3.8.2
[  ] rabbitmq_peer_discovery_consul    3.8.2
[  ] rabbitmq_peer_discovery_etcd      3.8.2
[  ] rabbitmq_peer_discovery_k8s       3.8.2
[  ] rabbitmq_prometheus               3.8.2
[  ] rabbitmq_random_exchange          3.8.2
[  ] rabbitmq_recent_history_exchange  3.8.2
[  ] rabbitmq_sharding                 3.8.2
[  ] rabbitmq_shovel                   3.8.2
[  ] rabbitmq_shovel_management        3.8.2
[  ] rabbitmq_stomp                    3.8.2
[  ] rabbitmq_top                      3.8.2
[  ] rabbitmq_tracing                  3.8.2
[  ] rabbitmq_trust_store              3.8.2
[e*] rabbitmq_web_dispatch             3.8.2
[  ] rabbitmq_web_mqtt                 3.8.2
[  ] rabbitmq_web_mqtt_examples        3.8.2
[  ] rabbitmq_web_stomp                3.8.2
[  ] rabbitmq_web_stomp_examples       3.8.2
root@ecs-docker-muballigh:~#
```

You can enable the RabbitMQ management plugin by using the following command:

```
rabbitmq-plugins enable rabbitmq_management
```

```
root@ecs-docker-muballigh:~# rabbitmq-plugins enable rabbitmq_management
Enabling plugins on node rabbit@ecs-docker-muballigh:
rabbitmq_management
The following plugins have been configured:
  rabbitmq_management
  rabbitmq_management_agent
  rabbitmq_web_dispatch
Applying plugin configuration to rabbit@ecs-docker-muballigh...
Plugin configuration unchanged.
root@ecs-docker-muballigh:~#
```

You can now connect to RabbitMQ web interface. To gain access, open your web browser and type the URL http://your-server-ip:15672



Now we will create the files:
We're going to use Pika 1.0.0, which is the Python client recommended by the RabbitMQ team. To install it you can use the pip package management tool:

```
python3 -m pip install pika --upgrade
```

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 -m pip install pika --upgrade
Requirement already up-to-date: pika in /usr/local/lib/python3.8/dist-packages (1.3.1)
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

Our first program send.py will send a single message to the queue. The first thing we need to do is to establish a connection with RabbitMQ server.

It will contain the following commands:

```
#!/usr/bin/env python
import pika

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='hello')

channel.basic_publish(exchange='', routing_key='hello', body='Hello
World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

```
root@ecs-docker-muballigh:/# cd root
root@ecs-docker-muballigh:~# ls
dockerfiles
root@ecs-docker-muballigh:~# cd dockerfiles
root@ecs-docker-muballigh:~/dockerfiles# ls
dockerfile  htmlapplication  Modern-Landing-Page-UI  testsite
root@ecs-docker-muballigh:~/dockerfiles# mkdir rabbitmq
root@ecs-docker-muballigh:~/dockerfiles# cd rabbitmq
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# touch send.py
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

```
  GNU nano 4.8
#!/usr/bin/env python
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='',
                      routing_key='hello',
                      body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# nano send.py
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# cat send.py
#!/usr/bin/env python
import pika

connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='',
                      routing_key='hello',
                      body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

Our second program receive.py will receive messages from the queue and print them on the screen.
It will contain the following commands:

```
#!/usr/bin/env python
import pika, sys, os

def main():
    connection =
pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='hello')

    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body)

    channel.basic_consume(queue='hello', on_message_callback=callback,
auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
```

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# nano receive.py
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# cat receive.py
#!/usr/bin/env python
import pika, sys, os

def main():
    connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
    channel = connection.channel()

    channel.queue_declare(queue='hello')

    def callback(ch, method, properties, body):
        print(" [x] Received %r" % body)

    channel.basic_consume(queue='hello', on_message_callback=callback, auto_ack=True)

    print(' [*] Waiting for messages. To exit press CTRL+C')
    channel.start_consuming()

if __name__ == '__main__':
    try:
        main()
    except KeyboardInterrupt:
        print('Interrupted')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

Now we can try out our programs in a terminal. First, let's start a consumer, which will run continuously waiting for deliveries:

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 receive.py
 [*] Waiting for messages. To exit press CTRL+C
```

Now start the producer in a new terminal. The producer program will stop after every run:

```
❯ ssh root@114.119.172.119
root@114.119.172.119's password:
Welcome to Ubuntu 20.04.5 LTS (GNU/Linux 5.4.0-100-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Mon 28 Nov 2022 01:58:17 PM CST

  System load:  0.0               Processes:               151
  Usage of /:   4.5% of 196.61GB  Users logged in:         1
  Memory usage: 8%                IPv4 address for docker0: 172.17.0.1
  Swap usage:   0%                IPv4 address for eth0:    192.168.1.150

 * Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
   just raised the bar for easy, resilient and secure K8s cluster deployment.

   https://ubuntu.com/engage/secure-kubernetes-at-the-edge

0 updates can be applied immediately.


*** System restart required ***

        Welcome to Huawei Cloud Service

Last login: Mon Nov 28 13:00:21 2022 from 119.8.179.23
root@ecs-docker-muballigh:~# cd dockerfiles/rabbitmq
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 send.py
 [x] Sent 'Hello World!'
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 send.py
 [x] Sent 'Hello World!'
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

The consumer will print the message:

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# nano receive.py
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 receive.py
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received b'Hello World!'
 [x] Received b'Hello World!'
```

Task 2 : Create a *Work Queue* that will be used to distribute time-consuming tasks among multiple workers.

Steps:

We will slightly modify the *send.py* code from our previous example, to allow arbitrary messages to be sent from the command line. This program will schedule tasks to our work queue, so let's name it new_task.py:

The commands:

```
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)

message = ' '.join(sys.argv[1:]) or "Hello World!"
channel.basic_publish(
    exchange='',
    routing_key='task_queue',
    body=message,
    properties=pika.BasicProperties(
        delivery_mode=pika.spec.PERSISTENT_DELIVERY_MODE
    ))
print(" [x] Sent %r" % message)
connection.close()
```

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# cat new_task.py
#!/usr/bin/env python
import pika
import sys

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)

message = ' '.join(sys.argv[1:]) or "Hello World!"
channel.basic_publish(
    exchange='',
    routing_key='task_queue',
    body=message,
    properties=pika.BasicProperties(
        delivery_mode=pika.spec.PERSISTENT_DELIVERY_MODE
    ))
print(" [x] Sent %r" % message)
connection.close()
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

Our old *receive.py* script also requires some changes: it needs to fake a second of work for every dot in the message body. It will pop messages from the queue and perform the task, so let's call it worker.py:

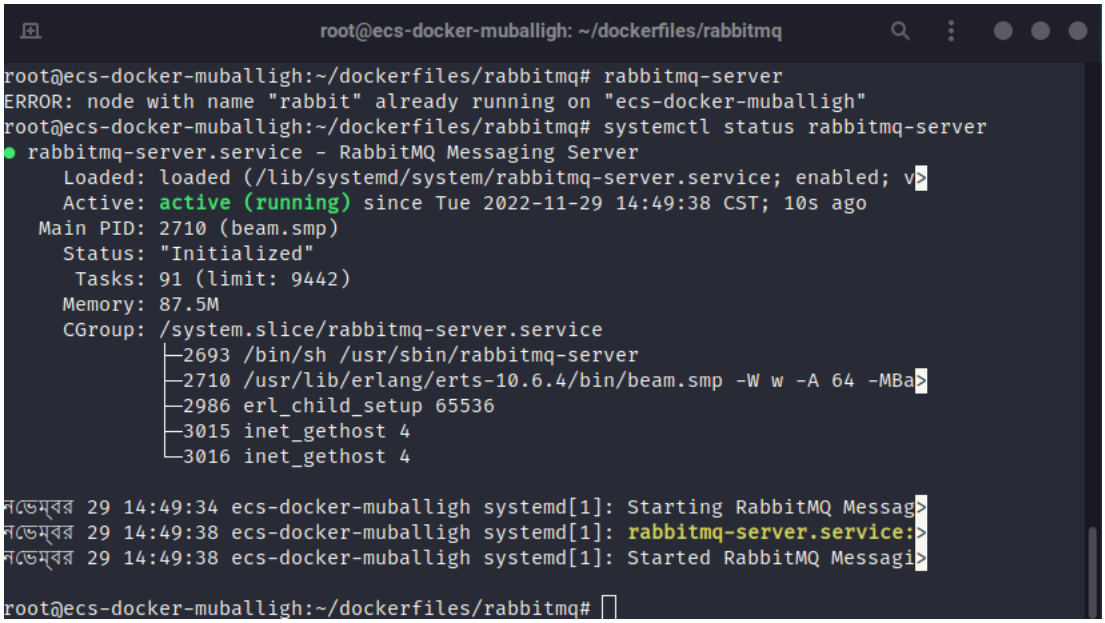
The Commands:

```
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode())
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()
```

```
  GNU nano 4.8                                                    worker.py
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode())
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()
```

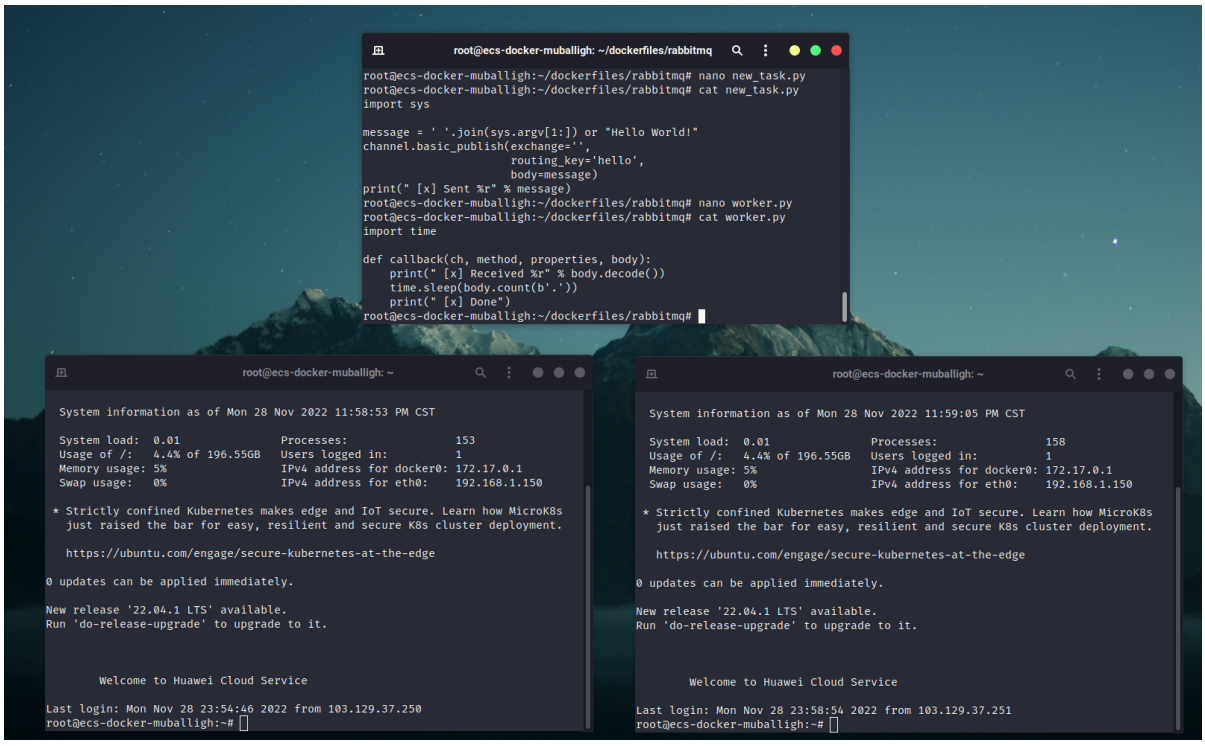
```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# nano worker.py
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# cat worker.py
#!/usr/bin/env python
import pika
import time

connection = pika.BlockingConnection(
    pika.ConnectionParameters(host='localhost'))
channel = connection.channel()

channel.queue_declare(queue='task_queue', durable=True)
print(' [*] Waiting for messages. To exit press CTRL+C')


def callback(ch, method, properties, body):
    print(" [x] Received %r" % body.decode())
    time.sleep(body.count(b'.'))
    print(" [x] Done")
    ch.basic_ack(delivery_tag=method.delivery_tag)


channel.basic_qos(prefetch_count=1)
channel.basic_consume(queue='task_queue', on_message_callback=callback)

channel.start_consuming()
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq#
```

Checking the Status of our rabbit-mq server

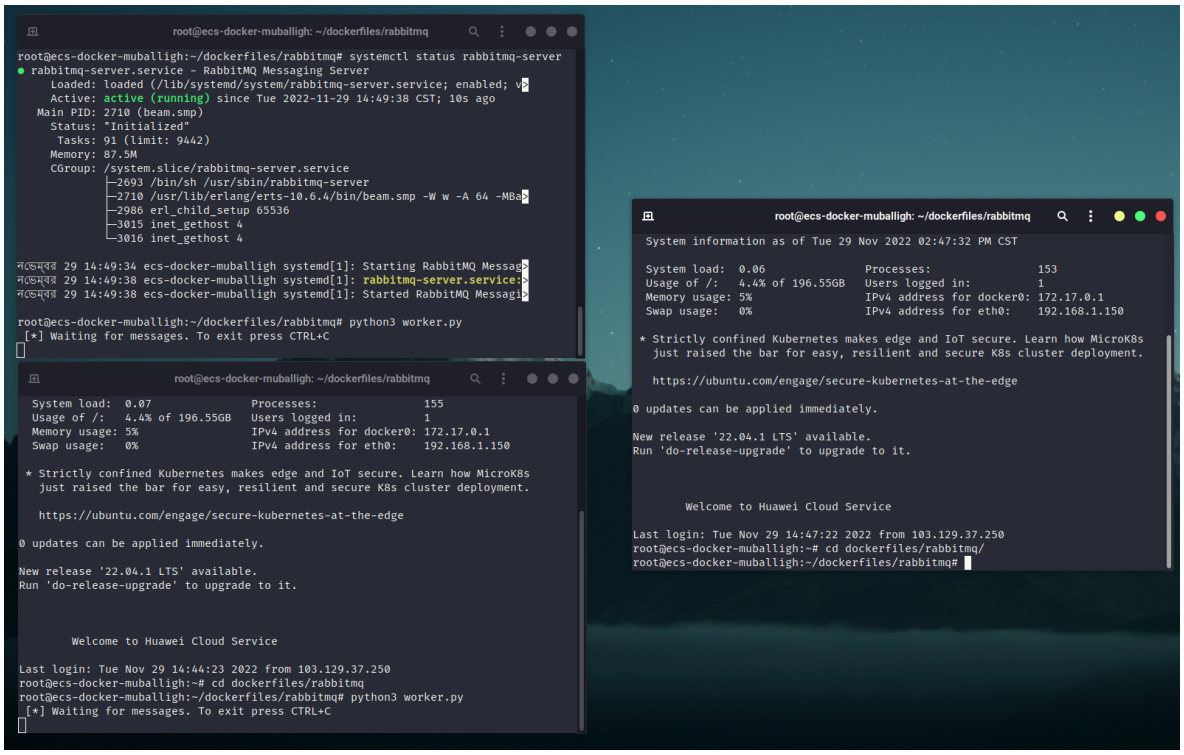


One of the advantages of using a Task Queue is the ability to easily parallelise work. If we are building up a backlog of work, we can just add more workers and that way, scale easily.

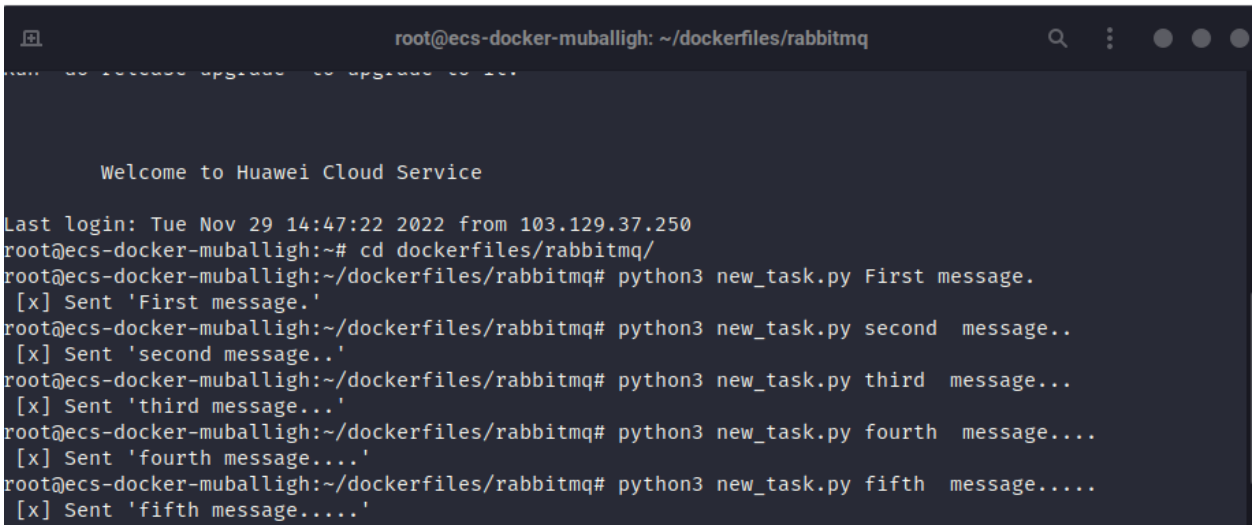
First, let's try to run two `worker.py` scripts at the same time. They will both get messages from the queue, but how exactly? Let's see.



You need three consoles open. Two will run the worker.py script. These consoles will be our two consumers - C1 and C2.



In the third one we'll publish new tasks. Once you've started the consumers you can publish a few messages:



By default, RabbitMQ will send each message to the next consumer, in sequence. On average every consumer will get the same number of messages. This way of distributing messages is called round-robin. Try this out with three or more workers.

```
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 worker.py
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received 'First message.'
 [x] Done
 [x] Received 'third message...'
 [x] Done
 [x] Received 'fifth message.....'
 [x] Done
```

```
Last login: Tue Nov 29 14:44:23 2022 from 103.129.37.250
root@ecs-docker-muballigh:~# cd dockerfiles/rabbitmq
root@ecs-docker-muballigh:~/dockerfiles/rabbitmq# python3 worker.py
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received 'second message..'
 [x] Done
 [x] Received 'fourth message....'
 [x] Done
```