

## EXERCISE 1: Correction System Using RabbitMQ

**Task 1:** Using Pika's syntax, declare and bind the necessary queues/exchanges and their connections as shown in Figure 1.

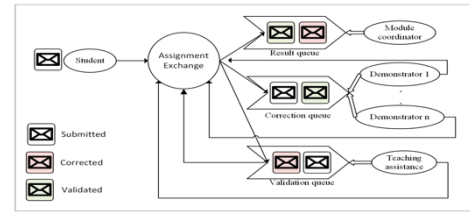
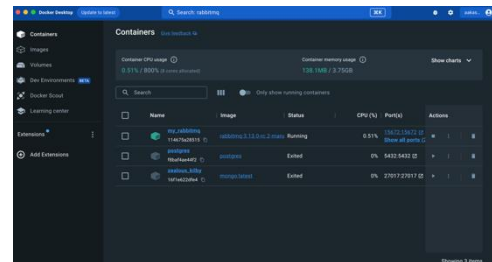


Figure 1: Practical correction through assignment broadcasting

Install RabbitMQ image in docker

Here we created docker container with RabbitMQ image for the exercises.



```
ex1 > task1_2.py > ...
1 import pika
2 import json
3
4 # Connect to RabbitMQ Software through this command
5 connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
6 channel = connection.channel()
7
8 # Task 1
9 # Using Pika's syntax, declare the necessary queues/exchanges and their connections as shown in Figure 1
10
11 channel.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
12
13 channel.queue_declare(queue='correction_queue')
14 channel.queue_declare(queue='validation_queue')
15 channel.queue_declare(queue='results_queue')
16
17 channel.queue_bind(exchange='assignment_exchange', queue='correction_queue', routing_key='correction')
18 channel.queue_bind(exchange='assignment_exchange', queue='validation_queue', routing_key='validation')
19 channel.queue_bind(exchange='assignment_exchange', queue='results_queue', routing_key='results')
20
```

### A. Declaration of Exchange:

`channel.exchange_declare(exchange='assignment_exchange', exchange_type='direct')` creates a 'direct' exchange with the name 'assignment\_exchange'.

### B. Queue Declarations:

Using the following methods, three queues are established for results validation, correction.

The expressions `channel.queue_declare(queue='correction_queue')`, `channel.queue_declare(queue='validation_queue')`, and `channel.queue_declare(queue='results_queue')` represent the queue operators.

### C. Binding Queues:

Using `channel.queue_bind(exchange='assignment_exchange', queue='correction_queue', routing_key='correction')`, each queue is bound to the exchange with a unique routing key ('correction', 'validation', 'results').

**Task 2:** Using Pika's syntax, define where each actor (demonstrator, teaching assistant, module coordinator) should listen/consume.

```
# Task 2
# Using Pika's syntax, declare where a demonstrator should listen/subscribe

def demonstrator_callback(ch, method, properties, body):
    assignment = json.loads(body.decode('utf-8'))
    print(f"Received message for correction: {assignment}")

    assignment['status'] = 'corrected'

    channel.basic_publish(
        exchange='assignment_exchange',
        routing_key='correction',
        body=json.dumps(assignment)
    )
    print("Assignment corrected")

channel.basic_consume(queue='correction_queue', on_message_callback=demonstrator_callback, auto_ack=True)

# Task 4
# Using Pika's syntax, declare where the module coordinator should listen/subscribe

def coordinator_callback(ch, method, properties, body):
    assignment = json.loads(body.decode('utf-8'))
    print(f"Received message for confirmation: {assignment}")

    if assignment['status'] == 'validated':
        assignment['status'] = 'confirmed'

    print(f"Assignment confirmed: {assignment}")

channel.basic_consume(queue='results_queue', on_message_callback=coordinator_callback, auto_ack=True)

print('Waiting for messages')
channel.start_consuming()
```

```
# Task 3
# Using Pika's syntax, declare where a TA should listen/subscribe

def ta_callback(ch, method, properties, body):
    assignment = json.loads(body.decode('utf-8'))
    print(f"Received message for validation: {assignment}")

    if assignment['status'] == 'corrected':
        assignment['status'] = 'validated'

        channel.basic_publish(
            exchange='assignment_exchange',
            routing_key='validation',
            body=json.dumps(assignment)
        )
        print("Assignment validated")
    else:
        print("Ignoring the assignment as it's not in the corrected state.")

channel.basic_consume(queue='validation_queue', on_message_callback=ta_callback, auto_ack=True)
```

**Demonstrator Subscriber:** Create a callback method (demonstrator\_callback) to handle messages that the demonstrator gets from the "correction\_queue."

Following message processing and correction logic, the function communicates the updated assignment back to the exchange using the routing key "correction."

Subscribes the demonstrator via channel.basic\_consume to the 'correction\_queue'.

**TA Subscriber:** Names a callback function (ta\_callback) to handle messages from the 'validation\_queue' that the TA receives.

Uses the routing key "validation" to publish the validated assignment back to the exchange after verifying that it is in the "corrected" state.

TA is subscribed to the 'validation\_queue' through channel.basic\_consume.

**Module Coordinator Subscriber:** Creates a callback function (coordinator\_callback) to handle messages from the 'results\_queue' that are received by the module coordinator.

Confirms whether the assignment is in the "validated" state, executes logic for confirmation, and generates a message for confirmation.

using channel.basic\_consume, subscribes the coordinator to the 'results\_queue'.

## Summary:

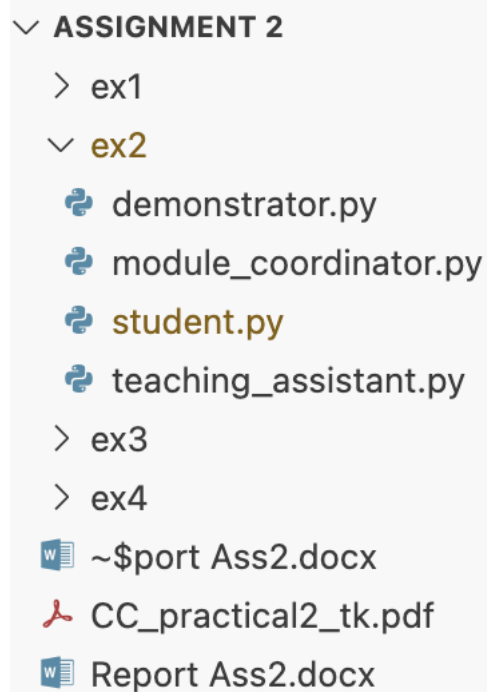
- **Message Queues and Exchanges:** Knowing how to use RabbitMQ to set up queues and exchanges for communication in a distributed system.
- **Publish-Subscribe Pattern:** This pattern allows for flexible communication by allowing entities to both publish and subscribe to messages.
- **Asynchronous Communication:** Understanding the advantages of asynchronous communication can lead to increased system responsiveness and efficiency.
- **Routing messages:** Ensure correct processing by assigning messages to designated queues according to exchanges and routing keys.
- **RabbitMQ with Pika:** Getting familiar with RabbitMQ and the Python Pika module for RabbitMQ interaction.
- **Understanding Message Flow:** Gain knowledge about how messages move across the system by interacting with producer and consumer scripts.
- **Developing abilities to recognize and fix problems that arise during the activity is known as debugging and troubleshooting.**

## EXERCISE 2 :

### Steps:

1. Create 4 files and add those data in that files. In ex2 folder.

Student, demonstrator,  
module\_cordinator ,  
teaching\_assistance



### 1. student.py -

The `submit_assignment` method of the `Student` class, defined in this script, submits assignments via RabbitMQ.

It connects, starts an exchange labeled "assignment\_exchange," publishes an assignment message to the exchange's "correction" queue, and so on.

The script then prints a confirmation message and terminates the RabbitMQ connection.

```

student.py 1 X
ex2 > student.py > Student
1 import pika
2 import json
3
4 class Student:
5     def __init__(self, student_id, answer):
6         self.student_id = student_id
7         self.answer = answer
8
9     def submit_assignment(self):
10         try:
11             connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
12             channel = connection.channel()
13             channel.exchange_declare(exchange='assignment_exchange', exchange_type='fanout')
14             assignment_message = {
15                 'StudentID': self.student_id,
16                 'Answer': self.answer,
17                 'status': 'submitted'
18             }
19
20             channel.basic_publish(
21                 exchange='assignment_exchange',
22                 routing_key='correction',
23                 body=json.dumps(assignment_message)
24             )
25             print("Assignment submitted for correction")
26             connection.close()
27         except Exception as e:
28             print(f"Error submitting assignment: {e}")
29
30 if __name__ == "__main__":
31     student_id = 23200858
32     answer = "Solution"
33     student = Student(student_id, answer)
34     student.submit_assignment()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

- (base) aakashukr@Akashs-MacBook-Air ex2 % python student.py  
Assignment submitted for correction
- (base) aakashukr@Akashs-MacBook-Air ex2 %

## 2. Demonstrator.py -

This script defines a Demonstrator class that listens for assignment messages in RabbitMQ.

It processes incoming messages by marking them as 'corrected' and sends them back to the 'assignment exchange'.

The class sets up a RabbitMQ connection, declares an exchange and a queue, and starts listening for messages until interrupted.

```
35 # demonstrator.py 1 X
ex2) demonstrator.py > ...
1 import pika
2 import json
3
4 class Demonstrator:
5     def __init__(self):
6         self.connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
7
8         self.channel = self.connection.channel()
9         self.channel.exchange_declare('assignment_exchange', exchange_type='fanout')
10        self.channel.queue_declare('assignment_queue')
11        self.channel.queue_declare('correction_queue')
12        self.channel.queue_bind(exchange='assignment_exchange', queue='correction_queue')
13        self.channel.basic_consume(queue='correction_queue', on_message_callback=self.demonstrator_callback, auto_ack=True)
14
15    def demonstrator_callback(self, ch, method, properties, body):
16        assignment = json.loads(body.decode('utf-8'))
17        print(f'Received message for correction: {assignment}')
18        assignment['status'] = 'corrected'
19        self.channel.basic_publish(
20            exchange='assignment_exchange',
21            routing_key='correction',
22            body=json.dumps(assignment))
23        print("Assignment corrected")
24
25    def listen_and_correct(self):
26        print('Demonstrator is waiting for messages.....')
27        self.channel.start_consuming()
28
29 if __name__ == "__main__":
30     demonstrator = Demonstrator()
31     demonstrator.listen_and_correct()
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
```

### 3. teaching\_assitance.py -

The script creates a class called `TeachingAssistant`, which serves as a receiver for assignment messages in the `'validation_queue'` of RabbitMQ.

As messages are received, it determines if the assignment has been "corrected." If so, it returns it to the "assignment\_exchange" after marking it as "validated."

The script establishes a RabbitMQ connection, creates a queue (named "validation\_queue") and an exchange (named "assignment\_exchange"), binds the queue to the exchange, and configures a callback function (named "ta\_callback") to process incoming messages.

The message-consuming process is started via the `start_listening` method, and the teaching assistant waits for messages until the user interrupts it (CTRL+C).

#### 4. module\_cordinator.py -

A `ModuleCoordinator` class is defined in this script, and it is designed to listen for assignment messages in the RabbitMQ's `results_queue`.

It examines incoming messages, prints a confirmation message, and verifies assignments if they are in the "validated" condition.

After establishing a RabbitMQ connection, the class creates a queue and an exchange and begins to wait for messages until they are interrupted.

```
> teaching_assistant.py X
ex2 > ▶ teaching_assistant.py ...
1 import pika
2 import json
3
4 class TeachingAssistant:
5     def __init__(self):
6         connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
7         self.channel = connection.channel()
8         self.channel.exchange_declare(exchange='assignment_exchange', exchange_type='fanout')
9         self.channel.queue_bind(queue='validation_queue')
10        self.channel.queue_bind(exchange='assignment_exchange', queue='validation_queue')
11        self.channel.basic_consume(queue='validation_queue', on_message_callback=self.ta_callback, auto_ack=True)
12
13    def ta_callback(self, ch, method, properties, body):
14        assignment = json.loads(body.decode('utf-8'))
15        print(f'Received message for validation: {assignment}')
16
17        if assignment['status'] == 'corrected':
18            assignment['status'] = 'validated'
19            self.channel.basic_publish(
20                exchange='assignment_exchange',
21                routing_key='validation',
22                body=json.dumps(assignment))
23        else:
24            print("Ignoring the assignment as it's not in the 'corrected' state.")
25
26    def start_listening(self):
27        print("Teaching Assistant is waiting for messages.....")
28        self.channel.start_consuming()
29
30 if __name__ == "__main__":
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

(base) aakashshukre@Aakashs-MacBook-Air ex2 % python teaching\_assistant.py  
Teaching Assistant is waiting for messages....  
  
|

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

Received message for validation: {'StudentID': 23280858, 'Answer': 'Solution', 'status': 'corrected'} Assignment validated  
Received message for validation: {'StudentID': 23280858, 'Answer': 'Solution', 'status': 'validated'} Ignoring the assignment as it's not in the 'corrected' state.  
Received message for validation: {'StudentID': 23280858, 'Answer': 'Solution', 'status': 'validated'} Ignoring the assignment as it's not in the 'corrected' state.  
Received message for validation: {'StudentID': 23280858, 'Answer': 'Solution', 'status': 'corrected'} Assignment validated  
Received message for validation: {'StudentID': 23280858, 'Answer': 'Solution', 'status': 'validated'} Ignoring the assignment as it's not in the 'corrected' state.  
Received

```
module coordinator.py' >
nx2: 1 import pika
2 import json
3
4 class ModuleCoordinator:
5     def __init__(self):
6         connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
7         self.channel = connection.channel()
8         self.channel.exchange_declare(exchange='assignment_exchange', exchange_type='fanout')
9         self.channel.queue_declare(queue='results_queue')
10        self.channel.queue_bind(exchange='assignment_exchange', queue='results_queue')
11        self.channel.basic_consume(queue='results_queue', on_message_callback=self.coordinator_callback, auto_ack=True)
12
13    def coordinator_callback(self, ch, method, properties, body):
14        assignment = json.loads(body.decode('utf-8'))
15        print(f"Received message for confirmation: {assignment}")
16
17        if assignment['status'] == 'validated':
18            assignment['status'] = 'confirmed'
19            print(f"Assignment confirmed: {assignment}")
20        else:
21            print(f"Ignoring the assignment as it's not in the 'validated' state.")
22
23    def start_listening(self):
24        print(f"Module Coordinator is waiting for messages.....")
25        self.channel.start_consuming()
26
27 if __name__ == '__main__':
28     demonstrator = ModuleCoordinator()
29     demonstrator.start_listening()
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

base) akash@akash-MacBook-Air nx2 % python module\_coordinator.py

Module Coordinator is waiting for messages.....

## message, and verifies

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

Received message for confirmation: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'validated'}

Assignment confirmed: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'confirmed'}

Received message for confirmation: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'validated'}

Assignment confirmed: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'confirmed'}

Received message for confirmation: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'validated'}

Assignment confirmed: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'confirmed'}

Received message for confirmation: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'validated'}

Assignment confirmed: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'confirmed'}

Received message for confirmation: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'validated'}

Assignment confirmed: {'StudentID': '23200858', 'Answer': 'Solution', 'status': 'confirmed'}

## Summary:

- Configured RabbitMQ as a message queue system.
  - examined distributed system communication between the coordinator of the module, the student, the demonstrator, and the teaching assistant.
  - Used the asynchronous publish-subscribe technique for communication.
  - Exhibited the division of responsibilities in system design.
  - Learned about RabbitMQ routing, queue binding, and debugging.
  - Effective communication within decentralized systems is essential.
  - Asynchronous processing for scalability and fault tolerance.
  - Clear system design with clearly defined roles.
- 
- In conclusion, combining message queues and containerization, Exercise Two offers useful guidance for creating scalable, fault-tolerant, and effective distributed systems.

## Output Observation:

1. Open Docker and run RabbitMQ and open <http://localhost:15672/#/queues>
2. Check output in the RabbitMQ localhost in queues where you can see 3 section. Here we can see the outputs in the RabbitMQ software to check the response.

**RabbitMQ**™ RabbitMQ 3.13.0-rc.2 Erlang 26.1.2

Overview Connections Channels **Exchanges** Queues and Streams Admin

### Exchanges

▼ All exchanges (8)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Virtual host	Name	Type	Features	Message rate in	Message rate out	+/-
/	(AMQP default)	direct	D			
/	amq.direct	direct	D			
/	amq.fanout	fanout	D			
/	amq.headers	headers	D			
/	amq.match	headers	D			
/	amq.rabbitmq.trace	topic	D I			
/	amq.topic	topic	D			
/	assignment_exchange	direct		4,699/s	4,699/s	

**RabbitMQ**™ RabbitMQ 3.13.0-rc.2 Erlang 26.1.2

Overview Connections Channels Exchanges **Queues and Streams** Admin

### Overview

▼ Totals

Queued messages (last minute) ?

Message rates (last minute) ?

Global counts ?

Nodes

Name	File descriptors	Socket descriptors	Erlang processes	Memory	Disk space	Uptime	Info	Reset state	+
node@114.76.28.113	39	1	440	144 MB	51 GB	15m 44s	Basic OK 2 OK	This node	All nodes

Chore statistics

Ports and contents

Export definitions

Import definitions

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Google Group Discord Slack Plugins GitHub

**RabbitMQ**™ RabbitMQ 3.13.0-rc.2 Erlang 26.1.2

Overview Connections **Channels** Exchanges Queues and Streams Admin

### Channel: 192.168.65.1:46384 -> 172.17.0.2:5672 (1)

▼ Overview

Message rates (last minute) ?

Details

Connection	State	Messages unacknowledged	Pending Raft commands
192.168.65.1:46384	running	0	0

Consumer tag	Queue	Ack required	Exclusive	Prefetch count	Active	Activity status	Consumer Timeout	Arguments
ctag1-fsdef318595463f8bfbee2374c378	correction_queue	0	0	0	0	up	1800000	
ctag1-94254955d3e14fc5b370f97a7f01eb5	validation_queue	0	0	0	0	up	1800000	
ctag1-8971cb17d9784959525351735f819b3	results_queue	0	0	0	0	up	1800000	

Runtime Metrics (Advanced)

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Google Group Discord Slack Plugins GitHub

▼ Bindings

This exchange

↓

To	Routing key	Arguments
correction_queue	correction_queue	Unbind
results_queue	results_queue	Unbind
validation_queue	validation_queue	Unbind

**RabbitMQ**™ RabbitMQ 3.13.0-rc.2 Erlang 26.1.2

Overview **Connections** Channels Exchanges Queues and Streams Admin

### Connections

▼ All connections (1)

Pagination

Page 1 of 1 - Filter:  ☐ Regex ?

Overview	Details	Network	+/-
<b>Name</b> 192.168.65.1:46384	<b>User name</b> guest	<b>State</b> running	
	<b>SSL / TLS</b> 0	<b>Protocol</b> AMQP 0-9-1	
	<b>Channels</b> 1	<b>From client</b> 698 KIB/s	<b>To client</b> 906 KIB/s

HTTP API Documentation Tutorials New releases Commercial edition Commercial support Google Group Discord Slack Plugins GitHub



## EXERCISE 3 : optimizes the assignment correction system from Exercise 1

### 1. Student.py –

```

student.py 1 X
ex3 > student.py > ...
1 import pika, json
2
3 creds, conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
4 ch = conn.channel()
5
6 ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
7
8 students = [
9     {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'submitted'},
10    {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'submitted'}
11 ]
12 for student in students:
13     message = json.dumps(student)
14     print(f"{message}")
15     routing_key = 'demonstrator_dm' if student['Module'] == 'data_mining' else 'demonstrator_cc'
16     ch.basic_publish(exchange='assignment_exchange', routing_key=routing_key, body=message)
17     print(f"{routing_key}")
18     print(f"Student {student['StudentID']} sent {message}")
19
20 conn.close()
21
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
(base) aakashsukre@Akashs-MacBook-Air ex3 % python student.py
{"StudentID": 23200858, "Solution": "Task 3 - Cloud Computing", "Module": "cloud_computing", "status": "submitted"}
demonstrator_cc
Student 23200858 sent {"StudentID": 23200858, "Solution": "Task 3 - Cloud Computing", "Module": "cloud_computing", "status": "submitted"}
{"StudentID": 12345678, "Solution": "Task 3 - Data Mining", "Module": "data_mining", "status": "submitted"}
demonstrator_dm
Student 12345678 sent {"StudentID": 12345678, "Solution": "Task 3 - Data Mining", "Module": "data_mining", "status": "submitted"}
(base) aakashsukre@Akashs-MacBook-Air ex3 %

```

- The module ('data\_mining' or 'cloud\_computing') is initialized in the Student file.
- The class constructor creates a RabbitMQ channel and connection.
- The definition declares a queue unique to the module and creates a direct exchange called "assignment\_exchange."
- The submit\_assignment function creates an assignment dictionary and uses RabbitMQ to publish it to the relevant module queue.
- Additionally, a message confirming the submission's success is printed by the procedure.
- After the assignment is turned in, the RabbitMQ connection is terminated.
- It creates an instance of the Student class for the Cloud Computing and Data Mining modules.
- The submit\_assignment method is used to turn in assignments for both modules.

## 2. demonstrator\_dm.py -

```

demonstrator_dm.py 1 X
ex3 > demonstrator_dm.py > main
1 import pika, json, sys, os
2 from time import sleep
3
4 def main():
5     creds = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
6     ch = conn.channel()
7     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
8     q_name = ch.queue_declare(queue='correction_queue_dm', method.queue)
9     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_dm')
10
11     def callback(ch, method, properties, body):
12         message = json.loads(body)
13         print(f"Data Mining Demonstrator received {message}")
14         message['status'] = 'corrected'
15         ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(message))
16         print(f"Data Mining Demonstrator corrected {json.dumps(message)}")
17
18     ch.basic_consume(queue='correction_queue_dm', on_message_callback=callback, auto_ack=True)
19     print('Data Mining Demonstrator is waiting for assignments')
20     ch.start_consuming()
21
22 if __name__ == '__main__':
23     try:
24         main()
25     except KeyboardInterrupt:
26         print('Interrupted')
27         try:
28             sys.exit(0)
29         except SystemExit:
30             os._exit(0)
31

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

(base) aakashsukre@Akashs-MacBook-Air ex3 % python demonstrator\_dm.py  
Data Mining Demonstrator is waiting for assignments

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
(base) aakashsukre@Akashs-MacBook-Air ex3 % python demonstrator_dm.py
Data Mining Demonstrator received {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'submitted'}
Data Mining Demonstrator corrected {"StudentID": 12345678, "Solution": "Task 3 - Data Mining", "Module": "data_mining", "status": "corrected"}

```

## 3. demonstrator\_cc.py

```

demonstrator_cc.py 1 X
ex3 > demonstrator_cc.py > main
1 import pika, json, sys, os
2
3 def main():
4     creds = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
5     ch = conn.channel()
6     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
7     q_name = ch.queue_declare(queue='correction_queue_cc', method.queue)
8     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_cc')
9
10     def callback(ch, method, properties, body):
11         msg = json.loads(body)
12         print(f"Cloud Computing Demonstrator received {msg}")
13         msg['status'] = 'corrected'
14         ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(msg))
15         print(f"Cloud Computing Demonstrator corrected {json.dumps(msg)}")
16
17     ch.basic_consume(queue='correction_queue_cc', on_message_callback=callback, auto_ack=True)
18     print('Cloud Computing Demonstrator is waiting for assignments')
19     ch.start_consuming()
20
21 if __name__ == '__main__':
22     try:
23         main()
24     except KeyboardInterrupt:
25         print('Interrupted')
26         try:
27             sys.exit(0)
28         except SystemExit:
29             os._exit(0)
30

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

(base) aakashsukre@Akashs-MacBook-Air ex3 % python demonstrator\_cc.py  
Cloud Computing Demonstrator is waiting for assignments

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE
(base) aakashsukre@Akashs-MacBook-Air ex3 % python demonstrator_cc.py
Cloud Computing Demonstrator received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'submitted'}
Cloud Computing Demonstrator corrected {"StudentID": 23200858, "Solution": "Task 3 - Cloud Computing", "Module": "cloud_computing", "status": "corrected"}

```

- In the above code which explains a system, demonstrators are represented by the CloudComputingDemonstrator and DataMiningDemonstrator definition.
- They connect to RabbitMQ, define dedicated queues for their modules, start with a 'corrected' state, and wait endlessly for assignments to be sent in.
- When an assignment is received, it is published to the 'validation' queue via the 'assignment\_exchange' and its status is changed to 'corrected'.
- Within the entire system, these definition help with assignment correction for the Data Mining and Cloud Computing modules.

### 3. teaching\_assistance.py -

```
teaching_assistant.py 1 X
ex3 > teaching_assistant.py > ...
1 import pika, json, sys, os
2 from time import sleep
3
4 def main():
5     conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
6     ch = conn.channel()
7     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
8
9     result = ch.queue_declare(queue='validation_queue')
10    q_name = result.method.queue
11    ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='validation')
12
13    def callback(_, __, body):
14        message = json.loads(body)
15        print(f'Teaching Assistant received {message}')
16        message['status'] = 'validated'
17        ch.basic_publish(exchange='assignment_exchange', routing_key='confirmation', body=json.dumps(message))
18        print(f'Teaching Assistant Validated {json.dumps(message)}')
19
20    ch.basic_consume(queue='validation_queue', on_message_callback=callback, auto_ack=True)
21    print('Teaching Assistant is waiting for assignments')
22    ch.start_consuming()
23
24 if __name__ == '__main__':
25     try:
26         main()
27     except KeyboardInterrupt:
28         print('Interrupted')
29         try:
30             sys.exit(0)
31         except SystemExit:
32             os._exit(0)
33
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

(base) akashsukre@Akashi-MacBook-Air ex3 % python teaching\_assistant.py  
Teaching Assistant is waiting for assignments

### 4. module\_cordinator.py -

```
module_cordinator.py 1 X
ex3 > module_cordinator.py > ...
1 import pika, json, sys, os
2
3 def main():
4     conn = pika.PlainCredentials("guest", "guest"), pika.BlockingConnection(pika.ConnectionParameters('localhost'))
5     ch = conn.channel()
6     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
7
8     q_name = ch.queue_declare(queue='confirmation_queue').method.queue
9     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='confirmation')
10
11    def callback(ch, method, properties, body):
12        message = json.loads(body)
13        print(f'Module Coordinator received {message}')
14        message['status'] = 'corrected'
15        ch.basic_publish(exchange='assignment_exchange', routing_key='test', body=json.dumps(message))
16        print(f'Module Coordinator corrected {json.dumps(message)}')
17
18    ch.basic_consume(queue='confirmation_queue', on_message_callback=callback, auto_ack=True)
19    print('Module Coordinator is waiting for assignments')
20    ch.start_consuming()
21
22 if __name__ == '__main__':
23     try:
24         main()
25     except KeyboardInterrupt:
26         print('Interrupted')
27         try:
28             sys.exit(0)
29         except SystemExit:
30             os._exit(0)
31
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

(base) akashsukre@Akashi-MacBook-Air ex3 % python module\_cordinator.py  
Module Coordinator is waiting for assignments

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
(base) akashsukre@Akashi-MacBook-Air ex3 % python module_cordinator.py
Module Coordinator is waiting for assignments
Module Coordinator received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'validated'}
Module Coordinator corrected {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'corrected'}
Teaching Assistant Validated {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'validated'}
Module Coordinator received {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'validated'}
Module Coordinator corrected {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'corrected'}
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS POSTMAN CONSOLE

```
(base) akashsukre@Akashi-MacBook-Air ex3 % python teaching_assistant.py
Teaching Assistant is waiting for assignments
Teaching Assistant received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'validated'}
Teaching Assistant Validated {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'corrected'}
Module Coordinator received {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'corrected'}
Teaching Assistant Validated {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'validated'}
Teaching Assistant Validated {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'corrected'}
```

- The TeachingAssistant and ModuleCoordinator python scripts creates a RabbitMQ connection and channel for communication.
- The def establishes a validation-specific queue (called "validation\_queue") and ('confirmation\_queue') and creates a direct exchange called "assignment\_exchange".
- The 'validation\_queue' and 'confirmation\_queue' are the place to continuously listen for incoming assignments.
- The callback method is initiated upon receiving a new assignment with the status 'corrected' and 'validated'. JSON-formatted messages.
- Next, using the 'assignment\_exchange', the assignment is republished to the 'confirmation' routing key with its status updated to 'validated'.
- The above both script stays active and prepared to validate new assignments by starting the message consumption process with the start\_consuming method.
- In the event of an interruption, exception handling guarantees a proper exit using os.\_exit(0) or sys.exit(0).

**Teaching Assistance :**

Function: Provides as the system's representation of a teaching assistant.  
Responsibility: The checks corrected assignments and listens for assignments in the validation\_queue.

Status Update: 'assignment\_exchange' modifies the assignment status to 'verified' and passes it to the 'confirmation' routing key.

**Module Coordinator :**

Function: In the system, this entity represents a Module Coordinator.

Responsibility: Gives attention to assignments in the confirmation\_queue and verifies assignments that have been validated.

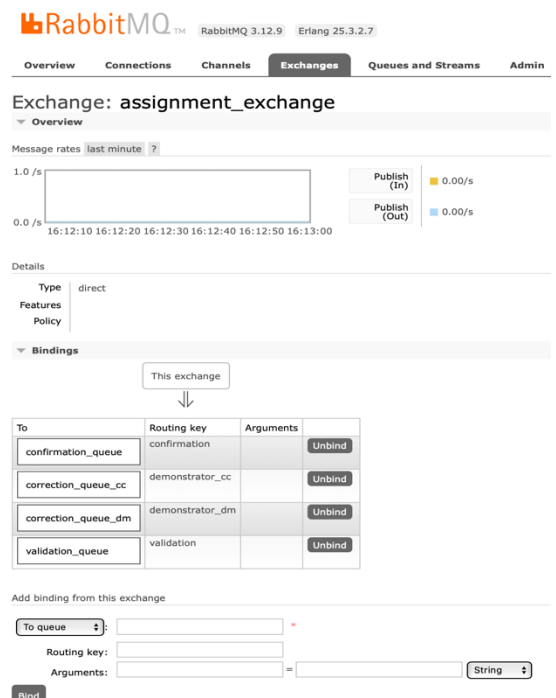
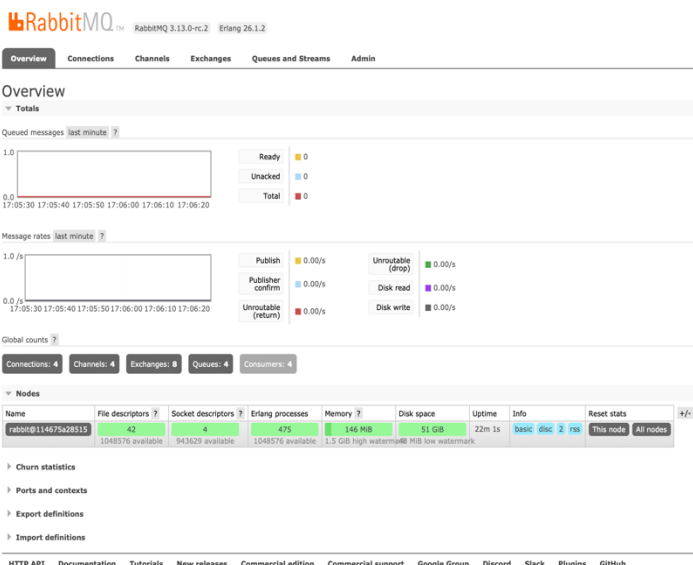
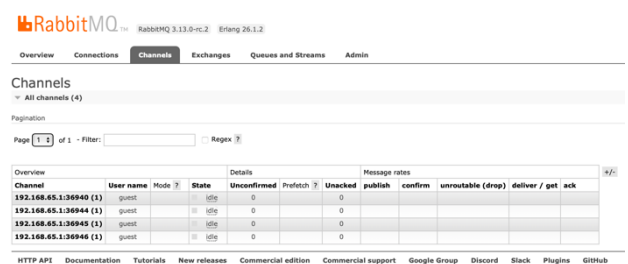
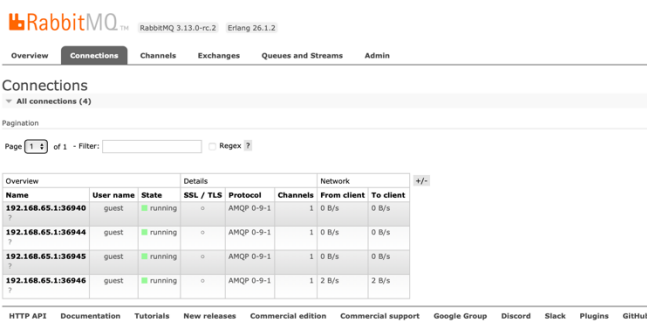
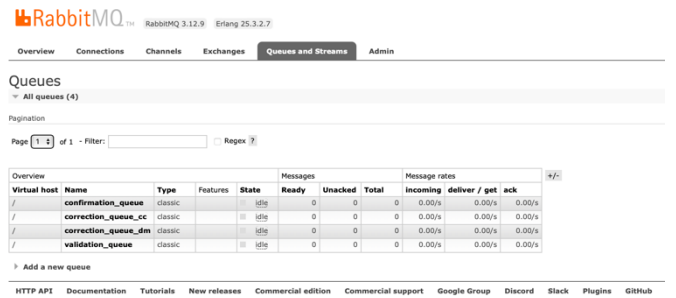
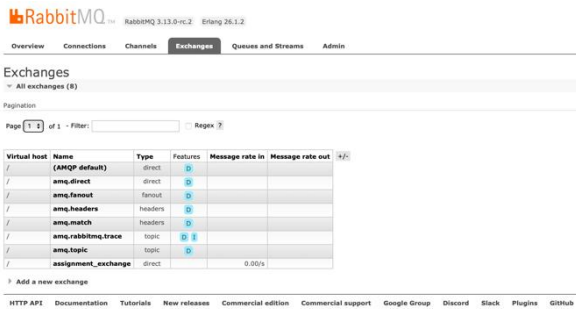
Status Update: 'confirmed' is the new assignment status, which is forwarded to the 'confirmation' routing key via the 'assignment\_exchange'.

**Summary:**

- Its main objective is to distribute tasks to demonstrators in an effective manner according to their modules.
- Students, teaching assistants, coordinators of the modules, and demonstrators of cloud computing and data mining are the entities participating. Assignments submitted by students are forwarded to demonstrators and queues that are appropriate.
- In order to minimize resource waste, the system makes sure assignments are only handled by appropriate groups.
- The aim is to construct this optimized scenario in Python using Pika and RabbitMQ. The goal is to present a simplified and more effective assignment correction procedure.

## Output Observation:

Here, we can see every tab from the RabbitMQ program, providing us with information about how the exercise 3 processes link with each other conveniently and efficiently optimize the system.



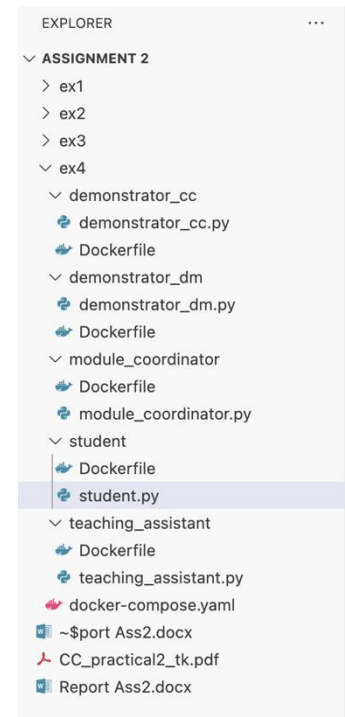
## EXERCISE 4 :

### Steps:

1. Create 5 folders and add those 5 files in that files. In ex4 folder.

Student,  
demonstrator\_cc,  
demonstrator\_dm,  
module\_cordinator ,  
teaching\_assistance,

2. add 5 Docker files in each folder respectively Dockerfile
3. Create docker-compose.yml file



Here we can see the below files and their code respectively.

### 1. demonstrator\_dm.py Docker file

```
demonstrator_dm.py 1 X
ex4 > demonstrator_dm.py > demonstrator_dm.py > establish_rabbitmq_connection
1 import pika
2 import json
3 import time
4
5 def establish_rabbitmq_connection():
6     attempts = 0
7     max_attempts = 5
8
9     while attempts < max_attempts:
10         try:
11             conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials('guest', 'guest')))
12             return conn
13         except pika.exceptions.AMQConnectionError:
14             print(f'Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})')
15             attempts += 1
16             time.sleep(2) # Wait for 2 seconds before retrying
17     print("Exceeded maximum connection attempts. Exiting.")
18     exit(1)
19
20 def main():
21     conn = establish_rabbitmq_connection()
22     ch = conn.channel()
23     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
24     q_name = ch.queue_declare(queue='correction_queue_dm', method=queue)
25     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_dm')
26
27     def callback(ch, method, properties, body):
28         message = json.loads(body)
29         print(f"Data Mining Demonstrator received {message}")
30         message['status'] = 'corrected'
31         ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(message))
32         print(f"Data Mining Demonstrator corrected {json.dumps(message)}")
33
34     ch.basic_consume(queue='correction_queue_dm', on_message_callback=callback, auto_ack=True)
35     print("Data Mining Demonstrator is waiting for assignments")
36     ch.start_consuming()
37
38 if __name__ == '__main__':
39     try:
40         main()
41     except KeyboardInterrupt:
42         print('Interrupted')
43         exit(0)
```

### Dockerfile

ex4 > demonstrator\_cc > Dockerfile > ...

```
1 FROM python:3.8
2 FROM python:slim-buster
3 WORKDIR /app
4 RUN pip install pika
5 COPY demonstrator_cc.py .
6 CMD ["python", "demonstrator_cc.py"]
7 ENV PYTHONUNBUFFERED=1
8
```

## 2. demonstrator\_cc.py Docker file

```

demonstrator_cc.py 1 X
ex4 > demonstrator_cc.py > establish_rabbitmq_connection
1 import pika
2 import json
3 import time
4
5 def establish_rabbitmq_connection():
6     attempts = 0
7     max_attempts = 5
8
9     while attempts < max_attempts:
10         try:
11             conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials('guest', 'guest')))
12             return conn
13         except pika.exceptions.AMQPConnectionError:
14             print(f'Failed to connect to RabbitMQ. Retrying... ((attempts)/(max_attempts))')
15             attempts += 1
16             time.sleep(2)
17     print('Exceeded maximum connection attempts. Exiting.')
18     exit(1)
19
20 def main():
21     conn = establish_rabbitmq_connection()
22     ch = conn.channel()
23     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
24     q_name = ch.queue_declare(queue='correction_queue_cc', method=queue)
25     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='demonstrator_cc')
26
27     def callback(ch, method, properties, body):
28         msg = json.loads(body)
29         print(f'Cloud Computing Demonstrator received {msg}')
30         msg['status'] = 'corrected'
31         ch.basic_publish(exchange='assignment_exchange', routing_key='validation', body=json.dumps(msg))
32         print(f'Cloud Computing Demonstrator corrected {json.dumps(msg)}')
33
34     ch.basic_consume(queue='correction_queue_cc', on_message_callback=callback, auto_ack=True)
35     print('Cloud Computing Demonstrator is waiting for assignments')
36     ch.start_consuming()
37
38 if __name__ == '__main__':
39     try:
40         main()
41     except KeyboardInterrupt:
42         print('Interrupted')
43         exit(0)
44     ..

```

Dockerfile X

ex4 > demonstrator\_cc > Dockerfile > ...

```

1 FROM python:3.8
2 FROM python:slim-buster
3 WORKDIR /app
4 RUN pip install pika
5 COPY demonstrator_cc.py .
6 CMD ["python", "demonstrator_cc.py"]
7 ENV PYTHONUNBUFFERED=1
8

```

## 3. module\_coordinator.py and Docker file

```

module_coordinator.py 1 X
ex4 > module_coordinator.py > establish_rabbitmq_connection
1 import pika
2 import json
3 import time
4
5 def establish_rabbitmq_connection():
6     attempts = 0
7     max_attempts = 5
8
9     while attempts < max_attempts:
10         try:
11             conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials('guest', 'guest')))
12             return conn
13         except pika.exceptions.AMQPConnectionError:
14             print(f'Failed to connect to RabbitMQ. Retrying... ((attempts)/(max_attempts))')
15             attempts += 1
16             time.sleep(2)
17     print('Exceeded maximum connection attempts. Exiting.')
18     exit(1)
19
20 def main():
21     conn = establish_rabbitmq_connection()
22     ch = conn.channel()
23     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
24     q_name = ch.queue_declare(queue='confirmation_queue', method=queue)
25     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='confirmation')
26
27     def callback(ch, method, properties, body):
28         message = json.loads(body)
29         print(f'Module Coordinator received {message}')
30         message['status'] = 'confirmed'
31         ch.basic_publish(exchange='assignment_exchange', routing_key='test', body=json.dumps(message))
32         print(f'Module Coordinator corrected {json.dumps(message)}')
33
34     ch.basic_consume(queue='confirmation_queue', on_message_callback=callback, auto_ack=True)
35     print('Module Coordinator is waiting for assignments')
36     ch.start_consuming()
37
38 if __name__ == '__main__':
39     try:
40         main()
41     except KeyboardInterrupt:
42         print('Interrupted')
43         exit(0)
44     ..

```

Dockerfile X

ex4 > module\_coordinator > Dockerfile > ...

```

1 FROM python:3.8
2 FROM python:slim-buster
3 WORKDIR /app
4 RUN pip install pika
5 COPY module_coordinator.py .
6 CMD ["python", "module_coordinator.py"]
7 ENV PYTHONUNBUFFERED=1

```

## 4. teaching\_assistant.py and Docker file

```

teaching_assistant.py 1 X
ex4 > teaching_assistant.py > establish_rabbitmq_connection
1 import pika
2 import json
3 import time
4
5 def establish_rabbitmq_connection():
6     attempts = 0
7     max_attempts = 5
8
9     while attempts < max_attempts:
10         try:
11             conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials('guest', 'guest')))
12             return conn
13         except pika.exceptions.AMQPConnectionError:
14             print(f'Failed to connect to RabbitMQ. Retrying... ((attempts)/(max_attempts))')
15             attempts += 1
16             time.sleep(2)
17     print('Exceeded maximum connection attempts. Exiting.')
18     exit(1)
19
20 def main():
21     conn = establish_rabbitmq_connection()
22     ch = conn.channel()
23     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
24     result = ch.queue_declare(queue='validation_queue')
25     q_name = result.method.queue
26     ch.queue_bind(exchange='assignment_exchange', queue=q_name, routing_key='validation')
27
28     def callback(ch, method, properties, body):
29         message = json.loads(body)
30         print(f'Teaching Assistant received {message}')
31         message['status'] = 'validated'
32         ch.basic_publish(exchange='assignment_exchange', routing_key='confirmation', body=json.dumps(message))
33         print(f'Teaching Assistant Validated {json.dumps(message)}')
34
35     ch.basic_consume(queue='validation_queue', on_message_callback=callback, auto_ack=True)
36     print('Teaching Assistant is waiting for assignments')
37     ch.start_consuming()
38
39 if __name__ == '__main__':
40     try:
41         main()
42     except KeyboardInterrupt:
43         print('Interrupted')
44         exit(0)
45     ..

```

Dockerfile X

ex4 > teaching\_assistant > Dockerfile > ...

```

1 FROM python:3.8
2 FROM python:slim-buster
3 WORKDIR /app
4 RUN pip install pika
5 COPY teaching_assistant.py .
6 CMD ["python", "teaching_assistant.py"]
7 ENV PYTHONUNBUFFERED=1

```

## 5. student.py and Docker file

```

student.py 1 X
ex4 > student > student.py > ...
1 import pika
2 import json
3 import time
4
5 def establish_rabbitmq_connection():
6     attempts = 0
7     max_attempts = 5
8
9     while attempts < max_attempts:
10         try:
11             conn = pika.BlockingConnection(pika.ConnectionParameters('rabbitmq', 5672, '/', pika.PlainCredentials('guest', 'guest')))
12             return conn
13         except pika.exceptions.AMQPConnectionError:
14             print(f"Failed to connect to RabbitMQ. Retrying... ({attempts}/{max_attempts})")
15             attempts += 1
16             time.sleep(2)
17     print("Exceeded maximum connection attempts. Exiting.")
18     exit(1)
19
20 def main():
21     conn = establish_rabbitmq_connection()
22     ch = conn.channel()
23     ch.exchange_declare(exchange='assignment_exchange', exchange_type='direct')
24
25     students = [
26         {'StudentID': 23200858, 'Solution': 'Task 3 - Cloud Computing', 'Module': 'cloud_computing', 'status': 'submitted'},
27         {'StudentID': 12345678, 'Solution': 'Task 3 - Data Mining', 'Module': 'data_mining', 'status': 'submitted'}
28     ]
29     for student in students:
30         message = json.dumps(student)
31         print(f"({message})")
32         routing_key = 'demonstrator_dm' if student['Module'] == 'data_mining' else 'demonstrator_cc'
33         ch.basic_publish(exchange='assignment_exchange', routing_key=routing_key, body=message)
34         print(f"({routing_key})")
35     print(f"Student {student['StudentID']} sent {message}")
36     conn.close()
37
38 if __name__ == '__main__':
39     try:
40         main()
41     except KeyboardInterrupt:
42         print('Interrupted')
43         exit(0)

```

```

Dockerfile X
ex4 > student > Dockerfile > ...
1 FROM python:3.8
2 FROM python:slim-buster
3 WORKDIR /app
4 RUN pip install pika
5 COPY student.py .
6 CMD ["python", "student.py"]
7 ENV PYTHONUNBUFFERED=1
8
9

```

### Python Script file:

The above all python script files contains below description of the code :

#### 1. establish\_connection function:

- This attempts to establish a connection to RabbitMQ.
- This retries up to five times, pausing every two seconds in between.
- Prints a message if the connection fails after each attempt.

#### 2. main function:

- Calls establish\_connection to get a RabbitMQ connection.
- Creates a channel, declares an exchange, and sets up a queue and binding.
- Establishes a queue and binding, declares an exchange, and creates a channel.
- Callback is invoked by consuming messages from the 'correction\_queue\_cc' queue.
- Prints messages when assignments are received and corrected.
- Starts reading messages and biding wait for assignments.

#### 3. if \_\_name\_\_ == '\_\_main\_\_':

- When the script is run, it carries out the main function.

### Docker File:

The above all Docker files contains below description of the code :

By using Docker's default bridge network, communication between containers in a Dockerized environment can be easily established by substituting 'rabbitmq' for 'localhost' in the code. Especially when containers are connected within the same network, it offers a reliable and portable method of referencing the RabbitMQ server.

1. Base Image (FROM): Indicates where the image begins.
2. The Working Directory (WORKDIR): establishes the directory for all commands that follow.
3. Dependencies Installation (RUN): Uses commands to set up dependencies or install programs.
4. File Copy (COPY): Transfers data from the local computer to the virtual machine.
5. Command (CMD): Specifies the initial command that a container will execute upon start-up.



## 1. Docker-compose.yml

```

docker-compose.yml X
ex4 > docker-compose.yml
1  version: '3'
2  networks:
3    rabbitmq_go_net:
4      driver: bridge
5
6  services:
7    rabbitmq:
8      image: "rabbitmq:3.13-rc-management"
9      container_name: rabbitmq_container
10     ports:
11       - "5672:5672"
12       - "15672:15672"
13     environment:
14       RABBITMQ_DEFAULT_USER: guest
15       RABBITMQ_DEFAULT_PASS: guest
16     networks:
17       - rabbitmq_go_net
18
19     demonstrator_cc:
20       build: demonstrator_cc
21       command: python ./demonstrator_cc.py
22       restart: unless-stopped
23       depends_on:
24         - rabbitmq
25       networks:
26         - rabbitmq_go_net
27
28     demonstrator_dm:
29       build: demonstrator_dm
30       command: python ./demonstrator_dm.py
31       restart: unless-stopped
32       depends_on:
33         - rabbitmq
34       networks:
35         - rabbitmq_go_net
36
37     teaching_assistant:
38       build: teaching_assistant
39       restart: unless-stopped
40       command: python ./teaching_assistant.py
41       depends_on:
42         - rabbitmq
43       networks:
44         - rabbitmq_go_net
45
46     module_coordinator:
47       build: module_coordinator
48       restart: unless-stopped
49       command: python ./module_coordinator.py
50       depends_on:
51         - rabbitmq
52       networks:
53         - rabbitmq_go_net
54
55     student:
56       build: student
57       depends_on:
58         - rabbitmq
59         - demonstrator_cc
60         - demonstrator_dm
61         - teaching_assistant
62         - module_coordinator
63       stdin_open: true
64       tty: true
65       networks:
66         - rabbitmq_go_net
67
  
```

Here is the explanation of the above compose file:

The official RabbitMQ image is used to create a RabbitMQ service with default credentials and open messaging ports.

The above file involves five additional services that represent different entities: teaching assistant, demonstrator\_dm, demonstrator\_cc, module\_coordinator, and student. Every service depends on the RabbitMQ service, builds from its own directory, and gives an instruction to run a Python script at startup.

To guarantee appropriate management and service availability, dependencies and restart policies are established on each service, and they are all connected to a unique bridge network called rabbitmq\_go\_net.

### Summary:

This above method makes it possible to create a reproducible and controlled environment for teaching purposes, showcasing the concepts of automation, containerization, and microservices communication via Docker and RabbitMQ.

## Output Observation:

Here, we can see information about how the exercise 4 processes link with each other conveniently and efficiently optimize the system via Docker and RabbitMQ

### 1. Build and compose UP

```
⇒ [student stage-1 4/4] COPY student.py .
⇒ [student] exporting to image
⇒ exporting layers
⇒ writing image sha256:1f4b1742f93de20bcfe299dd6c52006088e186c1a39e5776602855708a22bb1d
⇒ naming to docker.io/library/ex4-student
[+] Running 6/6
✔ Container rabbitmq_container Started
✔ Container ex4-module_coordinator-1 Started
✔ Container ex4-demonstrator_dm-1 Started
✔ Container ex4-demonstrator_cc-1 Started
✔ Container ex4-teaching_assistant-1 Started
✔ Container ex4-student-1 Started
* Terminal will be reused by tasks, press any key to close it.
```

### 2. Docker Execution Picture

