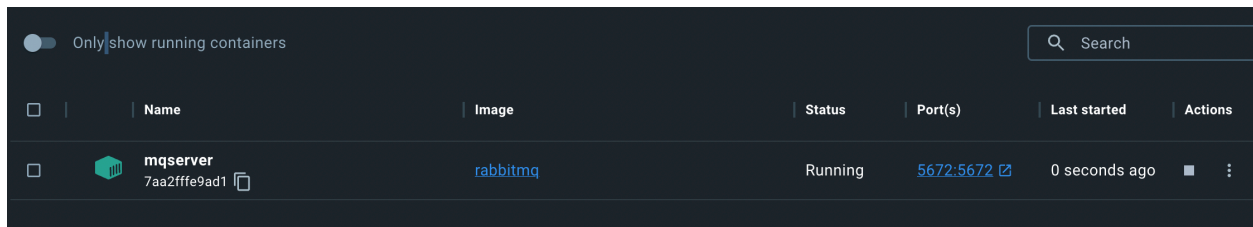


Introduction

In this lab, I was able to successfully write three programs in Python using gRPC and RabbitMQ to simulate the communication of a few rovers, the deminers and ground control together, as a continuation of Lab 2. I created a new deminer file within my previous project, to be able to demine and defuse the mines through a RabbitMQ channel rather than allowing the dig command to execute. This involved creating a RabbitMQ channel through a docker extension, but I was able to successfully get everything running and executing as intended.



The screenshot shows the Docker Desktop interface. At the top, there is a toggle switch labeled 'Only show running containers' which is turned on. To the right is a search bar. Below this is a table with columns: Name, Image, Status, Port(s), Last started, and Actions. One container is listed: 'mqserver' with ID '7aa2fffe9ad1', image 'rabbitmq', status 'Running', port '5672:5672', and '0 seconds ago' last started. There are checkboxes and icons in the Actions column.

	Name	Image	Status	Port(s)	Last started	Actions
<input type="checkbox"/>	mqserver 7aa2fffe9ad1	rabbitmq	Running	5672:5672	0 seconds ago	■ ⋮

Deminer

My deminer.py file is an implementation of a message queue using RabbitMQ. It defines two functions, demine_mine() and main(), that work together to consume messages from a demine-queue, generate a random PIN for each message, and publish a new message to a defused-queue with the corresponding PIN and serial number. The main() function sets up a consumer on the demine-queue, and the demine_mine() function handles incoming messages by generating a random PIN, computing a hash value, and publishing a new message to the defused-queue. The hash function is the same one I used in the previous lab. I have attached screenshots below of the RabbitMQ channel GUI, as well as the output of the mines and pins gathered from the client, preparing to demine them.

```
pin:40478887077
Beginning to demine MN0123
Serial Number: MN0123
Temporary Key: 52858262314MN0123
Hash Value: 000a16f6b93436c4be99cca04679b9bfa7351898f10f15c3e1f1be73601acc5e
pin:52858262314
Beginning to demine HIJ234
Serial Number: HIJ234
Temporary Key: 62854814756HIJ234
Hash Value: 000a185059396198b250b44dbd79a24f8e9e78b752e71855b07f340edf2af2b6
pin:62854814756
Beginning to demine STU901
Serial Number: STU901
Temporary Key: 72564417615STU901
Hash Value: 000523abafaaa84f274fa6f8988fd7b23ff017d96a6c0500c994cc7a75e33ccc
pin:72564417615
```

Server

In my server, I define a function called defused_mines() that is responsible for consuming messages from a RabbitMQ queue named defused-queue. It establishes a connection to the RabbitMQ broker and creates a channel for communication, then declares the defused-queue on the channel. The callback()

function is defined to handle incoming messages by extracting the serial number and PIN from the message payload and printing a message to the console indicating that the mine has been defused. The `channel.basic_consume()` method is called to start consuming messages from the defused-queue, using the `callback()` function as the message handler. Once the consumer is set up, the function enters a loop to continuously consume messages from the queue until the program is interrupted (e.g., with Ctrl+C). Finally, the code creates a new thread to execute the `defused_mines()` function and calls the `serve()` function to start the main server. I have attached screenshots below highlighting the RabbitMQ channel receiving message and data, as can be seen by the uptick in the charts at the time of execution of the code. These upticks are the messages and PINS being sent through the different queues. I have also attached screenshots of the output of the server, showing the mines defused and with what pin, matching the screenshot above.

Message 6

The server reported 2 messages remaining.

Exchange	(AMQP default)
Routing Key	defused-queue
Redelivered	•
Properties	
Payload	{ "serial_num": "MN0123", "pin": 52858262314 }
44 bytes	
Encoding: string	

Message 7

The server reported 1 messages remaining.

Exchange	(AMQP default)
Routing Key	defused-queue
Redelivered	•
Properties	
Payload	{ "serial_num": "HIJ234", "pin": 62854814756 }
44 bytes	
Encoding: string	

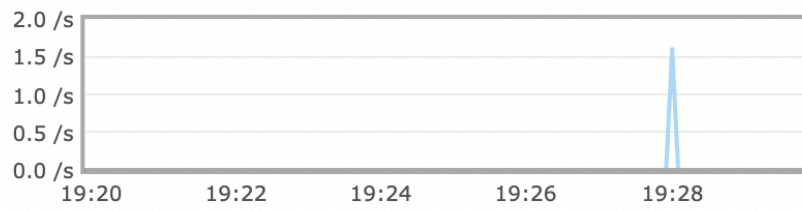
Message 8

The server reported 0 messages remaining.

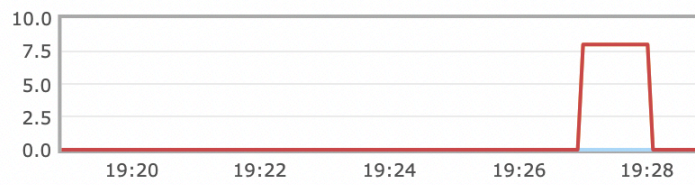
Exchange	(AMQP default)
Routing Key	defused-queue
Redelivered	•
Properties	
Payload	{ "serial_num": "STU901", "pin": 72564417615 }
44 bytes	
Encoding: string	

```
Print defuse mine with MN0123and pin52858262314
Print defuse mine with HIJ234and pin62854814756
Print defuse mine with STU901and pin72564417615
```

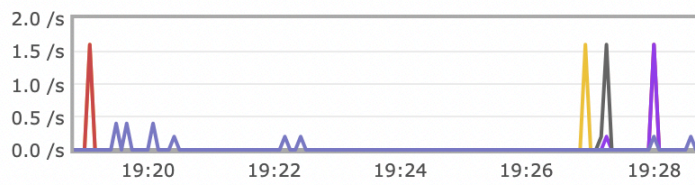
Message rates last ten minutes ?



Queued messages last ten minutes ?



Message rates last ten minutes ?



Queues

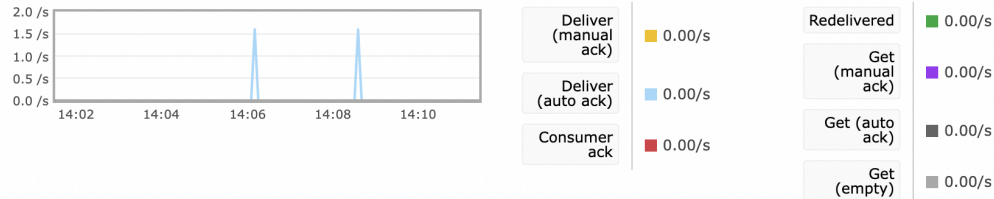
► All queues (2)

Overview				Messages			Message rates				+/-
Name	Type	Features	State	Ready	Unacked	Total	incoming	deliver / get	ack		
defused-queue	classic		idle	0	0	0	0.00/s	0.00/s	0.00/s		
demine-queue	classic		idle	0	0	0	0.00/s	0.00/s	0.00/s		

Channel 127.0.0.1:15552 > 127.0.0.1:15572 (1)

▼ Overview

Message rates last ten minutes ?



Details

Connection	127.0.0.1:64552	State	idle	Messages unacknowledged	0	Pending Raft commands	0
Username	guest	Prefetch count	0	Messages unconfirmed	0		
Mode ?		Global prefetch count	0	Messages uncommitted	0		
				Acks uncommitted	0		

▼ Consumers (1)

Consumer tag	Queue	Ack required	Exclusive	Prefetch count	Active ?	Activity status	Arguments
ctag1.50f3d6b1e89b44be8318f3fe16fd1c6e	defused-queue	o	o	0	•	up	

► Runtime Metrics (Advanced)

Conclusion

In conclusion, this lab provided an opportunity to further develop my skills in Python programming using gRPC and RabbitMQ. Three programs were written to simulate the communication between rovers, deminers, and ground control as a continuation of Lab 2. A new deminer file was created within the existing project to demine and defuse mines through a RabbitMQ channel, rather than allowing the dig command to execute. Despite the need to create a RabbitMQ channel through a docker extension, the implementation was successful, and all programs were able to run and execute as intended. Overall, this lab provided valuable experience in using messaging systems for asynchronous communication in distributed systems.