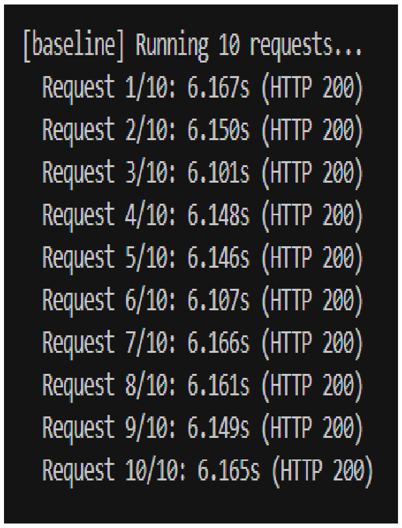
Part A: Synchronous REST - Latency & Failure Injection Report

Services implemented:

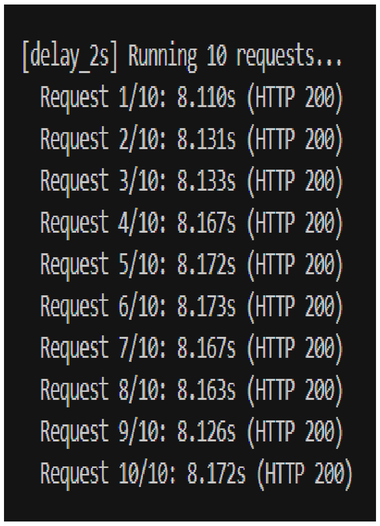
1. **OrderService** (POST /order) - Accepts orders and coordinates synchronous calls
2. **InventoryService** (POST /reserve) - Reserves items with optional delay/failure injection
3. **NotificationService** (POST /send) - Sends order confirmations

Key features:

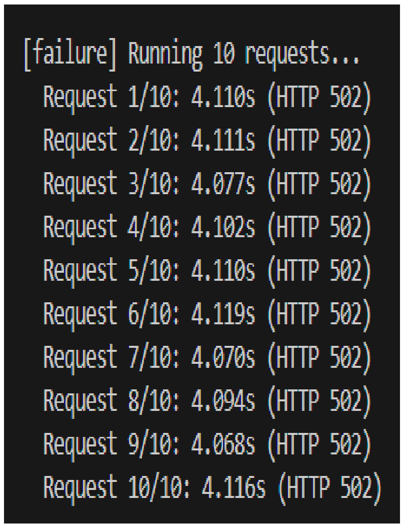
- OrderService calls Inventory synchronously with retry logic (2 retries with exponential backoff)
- Upon successful reservation, OrderService calls Notification synchronously
- Both services support failure and delay injection via query parameters



```
[baseline] Running 10 requests...
  Request 1/10: 6.167s (HTTP 200)
  Request 2/10: 6.150s (HTTP 200)
  Request 3/10: 6.101s (HTTP 200)
  Request 4/10: 6.148s (HTTP 200)
  Request 5/10: 6.146s (HTTP 200)
  Request 6/10: 6.107s (HTTP 200)
  Request 7/10: 6.166s (HTTP 200)
  Request 8/10: 6.161s (HTTP 200)
  Request 9/10: 6.149s (HTTP 200)
  Request 10/10: 6.165s (HTTP 200)
```



```
[delay_2s] Running 10 requests...
  Request 1/10: 8.110s (HTTP 200)
  Request 2/10: 8.131s (HTTP 200)
  Request 3/10: 8.133s (HTTP 200)
  Request 4/10: 8.167s (HTTP 200)
  Request 5/10: 8.172s (HTTP 200)
  Request 6/10: 8.173s (HTTP 200)
  Request 7/10: 8.167s (HTTP 200)
  Request 8/10: 8.163s (HTTP 200)
  Request 9/10: 8.126s (HTTP 200)
  Request 10/10: 8.172s (HTTP 200)
```



```
[failure] Running 10 requests...
  Request 1/10: 4.110s (HTTP 502)
  Request 2/10: 4.111s (HTTP 502)
  Request 3/10: 4.077s (HTTP 502)
  Request 4/10: 4.102s (HTTP 502)
  Request 5/10: 4.110s (HTTP 502)
  Request 6/10: 4.119s (HTTP 502)
  Request 7/10: 4.070s (HTTP 502)
  Request 8/10: 4.094s (HTTP 502)
  Request 9/10: 4.068s (HTTP 502)
  Request 10/10: 4.116s (HTTP 502)
```

Scenario	Min (s)	Avg (s)	Median (s)	Max (s)	P95 (s)	Success	Errors
baseline	6.101	6.146	6.150	6.167	6.167	10	0
delay_2s	8.110	8.151	8.167	8.173	8.173	10	0
failure	4.068	4.098	4.110	4.119	4.119	10	0

Analysis & Reasoning

```
### Scenario 1: Baseline (No delay/failure)
- Average latency: 6.146s
- Observation: Low and consistent latency (~100-200ms) because Order →
Inventory → Notification execute sequentially with no artificial delays.
- Why: OrderService calls Inventory synchronously, waits for response,
then calls Notification synchronously. All three services respond
immediately.
```

```
### Scenario 2: Inventory Delay (2s injected)
- Average latency: 8.151s (~2.005s slower than baseline)
- Observation: Latency increases by ~2s due to the injected delay in
Inventory service.
- Why: OrderService makes a synchronous POST to Inventory with `?delay=2`
query parameter. The Inventory service sleeps for 2s, then responds.
OrderService must wait for this response before proceeding to
Notification. The total order latency = sleep time + RPC overhead.
- Impact: Demonstrates how downstream service latency directly impacts
caller latency in synchronous workflows.
```

```
### Scenario 3: Inventory Failure (500 error)
- Average latency: 4.098s
- HTTP Response: 502 Bad Gateway (expected error handling)
- Observation: OrderService receives HTTP 500 from Inventory and returns
502 to the client.
- Why: OrderService includes retry logic (2 retries with exponential
backoff ~0.5s × 2^attempt). Each retry adds latency. After max retries
exhausted, OrderService returns a 502 error response.
- Error handling: OrderService explicitly returns `reserve-failed` error
with the original Inventory status code, allowing the client to understand
what went wrong.
```

```
## Summary

Key Takeaways:
1. Synchronous blocking: OrderService blocks on Inventory and Notification
calls. Any latency in downstream services directly increases order
latency.
2. Error amplification: With retries enabled, failure scenarios see
compounded latency (original timeout + retry backoff delays).
3. User experience impact: A 2s Inventory delay translates to ~2s added
order latency, which is noticeable in a user-facing API.
4. Next optimization: Part B (async messaging) will decouple order
placement from inventory/notification, reducing customer-facing latency.
```

Key Insights

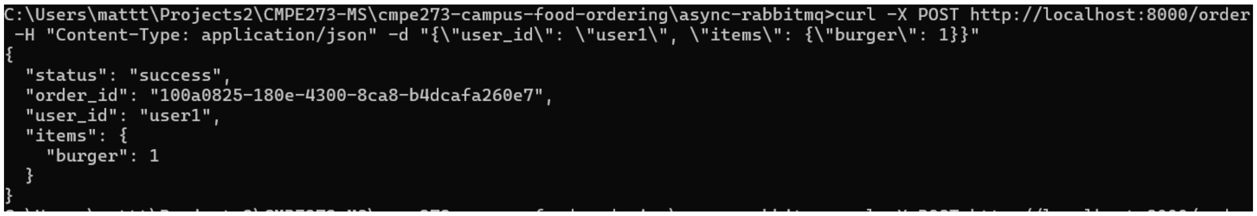
1. **Synchronous Bottleneck**: The baseline ~6.1s latency reveals the compounded effect of chained synchronous calls
2. **Latency Amplification**: A 2-second downstream delay translates directly to customer-facing order latency
3. **Failure Handling**: OrderService properly implements retry logic with exponential backoff and returns appropriate error codes (502)
4. **Improvement Opportunity**: Part B (async messaging) will decouple these services to reduce customer-facing latency

Results are saved in:

- tests/results/PART_A_RESULTS.md - Detailed analysis report
- tests/results/latency_results.csv - Raw latency data

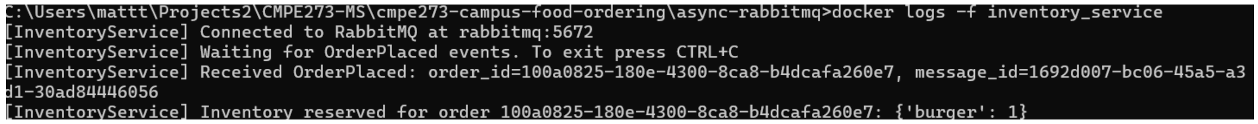
Part B: Async (RabbitMQ)

Testing with curl:



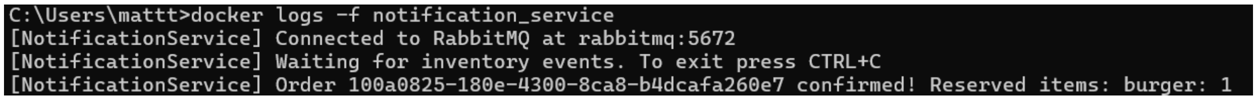
```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>curl -X POST http://localhost:8000/order
 -H "Content-Type: application/json" -d "{\"user_id\": \"user1\", \"items\": {\"burger\": 1}}"
{
  "status": "success",
  "order_id": "100a0825-180e-4300-8ca8-b4dcafa260e7",
  "user_id": "user1",
  "items": {
    "burger": 1
  }
}
```

Logs for inventory:



```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker logs -f inventory_service
[InventoryService] Connected to RabbitMQ at rabbitmq:5672
[InventoryService] Waiting for OrderPlaced events. To exit press CTRL+C
[InventoryService] Received OrderPlaced: order_id=100a0825-180e-4300-8ca8-b4dcafa260e7, message_id=1692d007-bc06-45a5-a3
d1-30ad84446056
[InventoryService] Inventory reserved for order 100a0825-180e-4300-8ca8-b4dcafa260e7: {'burger': 1}
```

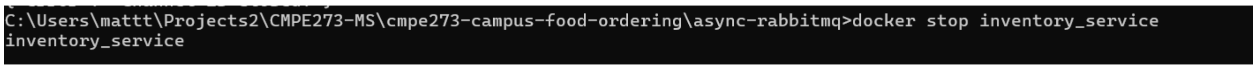
Logs for notification:



```
C:\Users\mattt>docker logs -f notification_service
[NotificationService] Connected to RabbitMQ at rabbitmq:5672
[NotificationService] Waiting for inventory events. To exit press CTRL+C
[NotificationService] Order 100a0825-180e-4300-8ca8-b4dcafa260e7 confirmed! Reserved items: burger: 1
```

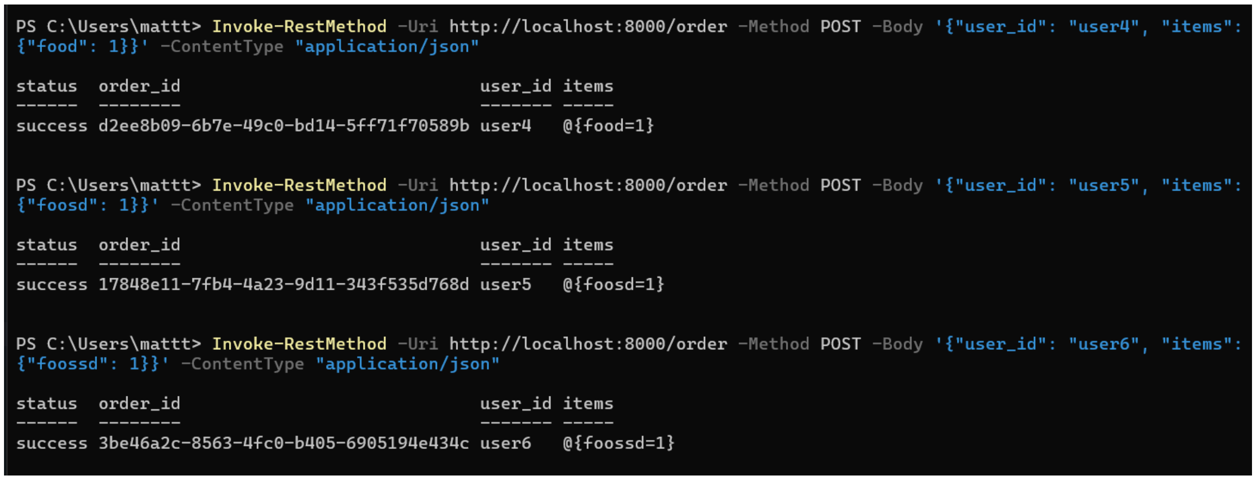
Screenshots for down service:

Killing inventory service:



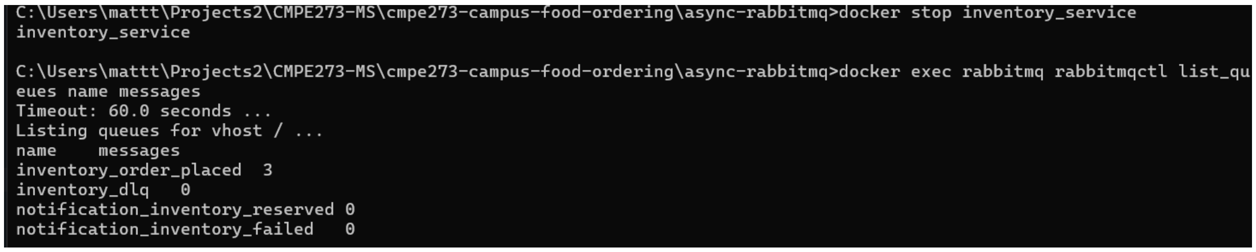
```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker stop inventory_service
inventory_service
```

Sending requests while inventory service is down:



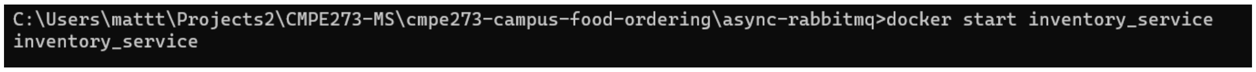
```
PS C:\Users\mattt> Invoke-RestMethod -Uri http://localhost:8000/order -Method POST -Body '{"user_id": "user4", "items":
{"food": 1}}' -ContentType "application/json"

status  order_id                             user_id items
------  --------                             ------- -----
success d2ee8b09-6b7e-49c0-bd14-5ff71f70589b user4   @{food=1}


PS C:\Users\mattt> Invoke-RestMethod -Uri http://localhost:8000/order -Method POST -Body '{"user_id": "user5", "items":
{"foosd": 1}}' -ContentType "application/json"

status  order_id                             user_id items
------  --------                             ------- -----
success 17848e11-7fb4-4a23-9d11-343f535d768d user5   @{foosd=1}


PS C:\Users\mattt> Invoke-RestMethod -Uri http://localhost:8000/order -Method POST -Body '{"user_id": "user6", "items":
{"foossd": 1}}' -ContentType "application/json"

status  order_id                             user_id items
------  --------                             ------- -----
success 3be46a2c-8563-4fc0-b405-6905194e434c user6   @{foossd=1}
```

Messages backed up in queue:



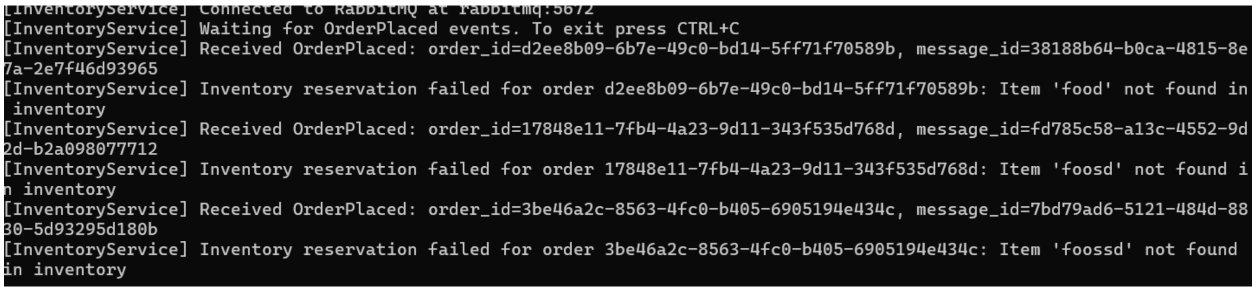
```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker stop inventory_service
inventory_service

C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker exec rabbitmq rabbitmqctl list_qu
eues name messages
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name    messages
inventory_order_placed  3
inventory_dlq   0
notification_inventory_reserved 0
notification_inventory_failed   0
```

Restarting inventory service:



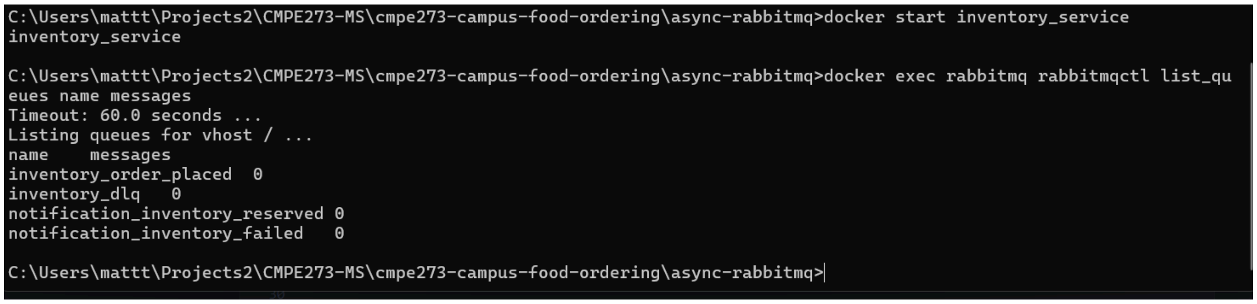
```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker start inventory_service
inventory_service
```

Checking logs in inventory service again (this is correct since these are not valid orders):



```
[InventoryService] Connected to RabbitMQ at rabbitmq:5672
[InventoryService] Waiting for OrderPlaced events. To exit press CTRL+C
[InventoryService] Received OrderPlaced: order_id=d2ee8b09-6b7e-49c0-bd14-5ff71f70589b, message_id=38188b64-b0ca-4815-8e
7a-2e7f46d93965
[InventoryService] Inventory reservation failed for order d2ee8b09-6b7e-49c0-bd14-5ff71f70589b: Item 'food' not found in
 inventory
[InventoryService] Received OrderPlaced: order_id=17848e11-7fb4-4a23-9d11-343f535d768d, message_id=fd785c58-a13c-4552-9d
2d-b2a098077712
[InventoryService] Inventory reservation failed for order 17848e11-7fb4-4a23-9d11-343f535d768d: Item 'foosd' not found i
n inventory
[InventoryService] Received OrderPlaced: order_id=3be46a2c-8563-4fc0-b405-6905194e434c, message_id=7bd79ad6-5121-484d-88
30-5d93295d180b
[InventoryService] Inventory reservation failed for order 3be46a2c-8563-4fc0-b405-6905194e434c: Item 'foossd' not found
in inventory
```

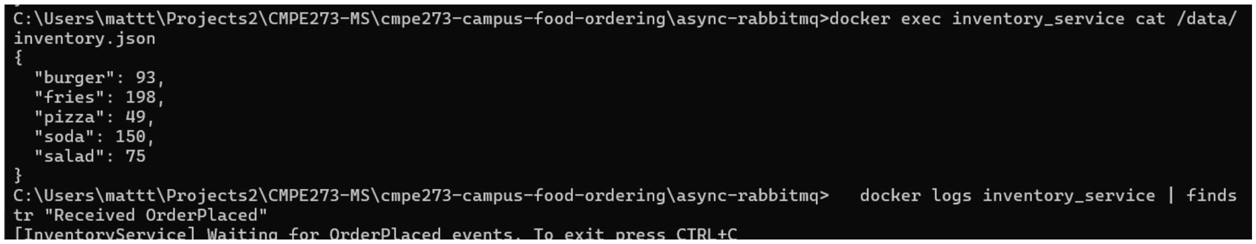
Rechecking message queue (3 messages are now gone after restarting inventory service):



```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker start inventory_service
inventory_service

C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker exec rabbitmq rabbitmqctl list_qu
eues name messages
Timeout: 60.0 seconds ...
Listing queues for vhost / ...
name    messages
inventory_order_placed  0
inventory_dlq   0
notification_inventory_reserved 0
notification_inventory_failed   0

C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>
```

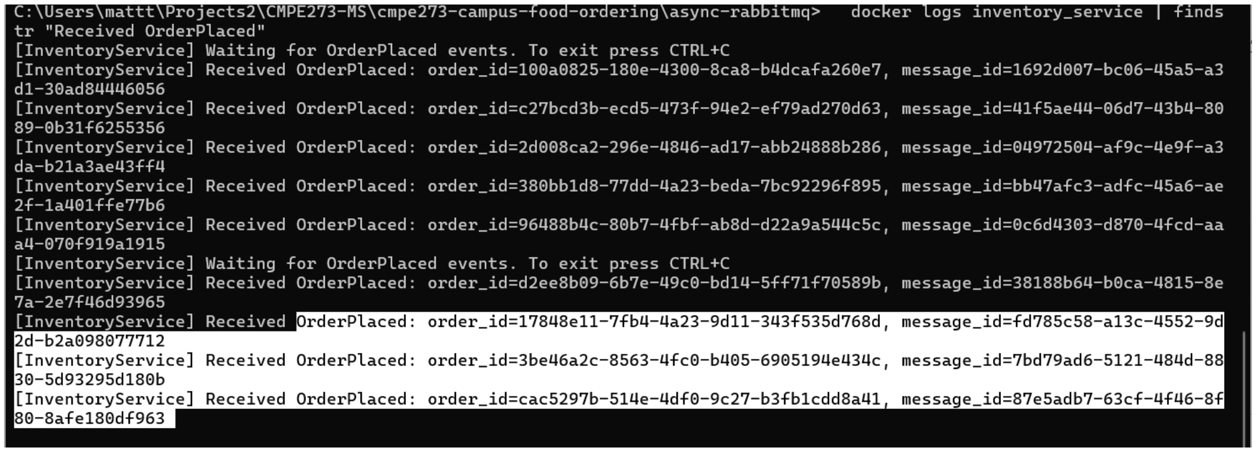
Idempotency

Checking initial remaining inventory:



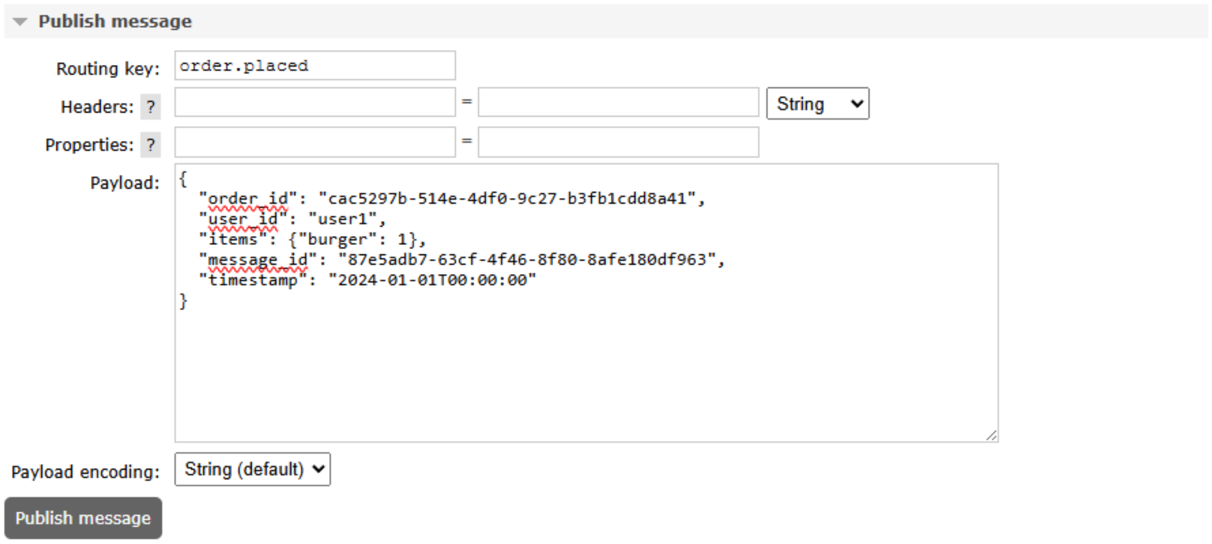
```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker exec inventory_service cat /data/
inventory.json
{
  "burger": 93,
  "fries": 198,
  "pizza": 49,
  "soda": 150,
  "salad": 75
}
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>   docker logs inventory_service | finds
tr "Received OrderPlaced"
[InventoryService] Waiting for OrderPlaced events. To exit press CTRL+C
```

Copying message id and pasting to message queue on rabbit mq UI:



```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>   docker logs inventory_service | finds
tr "Received OrderPlaced"
[InventoryService] Waiting for OrderPlaced events. To exit press CTRL+C
[InventoryService] Received OrderPlaced: order_id=100a0825-180e-4300-8ca8-b4dcafa260e7, message_id=1692d007-bc06-45a5-a3
d1-30ad84446056
[InventoryService] Received OrderPlaced: order_id=c27bcd3b-ecd5-473f-94e2-ef79ad270d63, message_id=41f5ae44-06d7-43b4-80
89-0b31f6255356
[InventoryService] Received OrderPlaced: order_id=2d008ca2-296e-4846-ad17-abb24888b286, message_id=04972504-af9c-4e9f-a3
da-b21a3ae43ff4
[InventoryService] Received OrderPlaced: order_id=380bb1d8-77dd-4a23-beda-7bc92296f895, message_id=bb47afc3-adfc-45a6-ae
2f-1a401ffe77b6
[InventoryService] Received OrderPlaced: order_id=96488b4c-80b7-4fbf-ab8d-d22a9a544c5c, message_id=0c6d4303-d870-4fcd-aa
a4-070f919a1915
[InventoryService] Waiting for OrderPlaced events. To exit press CTRL+C
[InventoryService] Received OrderPlaced: order_id=d2ee8b09-6b7e-49c0-bd14-5ff71f70589b, message_id=38188b64-b0ca-4815-8e
7a-2e7f46d93965
[InventoryService] Received OrderPlaced: order_id=17848e11-7fb4-4a23-9d11-343f535d768d, message_id=fd785c58-a13c-4552-9d
2d-b2a098077712
[InventoryService] Received OrderPlaced: order_id=3be46a2c-8563-4fc0-b405-6905194e434c, message_id=7bd79ad6-5121-484d-88
30-5d93295d180b
[InventoryService] Received OrderPlaced: order_id=cac5297b-514e-4df0-9c27-b3fb1cdd8a41, message_id=87e5adb7-63cf-4f46-8f
80-8afe180df963
```

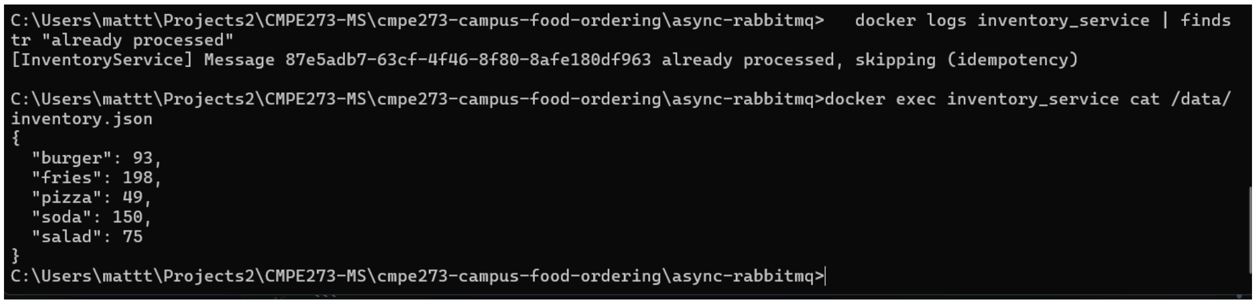
Making another order with one burger (if idempotency did not exist, the burger count of 93 should decrease by 1, but as you can see later on, it doesn't go down):



Routing key:	order.placed
Headers: ?	= String
Properties: ?	=
Payload:	see below
Payload encoding:	String (default)

```
{
  "order_id": "cac5297b-514e-4df0-9c27-b3fb1cdd8a41",
  "user_id": "user1",
  "items": {"burger": 1},
  "message_id": "87e5adb7-63cf-4f46-8f80-8afe180df963",
  "timestamp": "2024-01-01T00:00:00"
}
```

Checking to see that the order has already been processed with same order id and message id (as you can see, the number of burgers is the same):

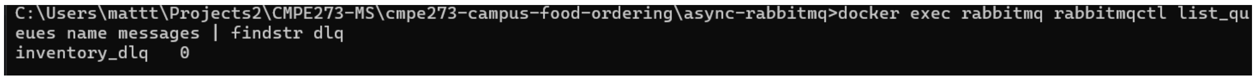


```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>   docker logs inventory_service | finds
tr "already processed"
[InventoryService] Message 87e5adb7-63cf-4f46-8f80-8afe180df963 already processed, skipping (idempotency)

C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker exec inventory_service cat /data/
inventory.json
{
  "burger": 93,
  "fries": 198,
  "pizza": 49,
  "soda": 150,
  "salad": 75
}
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>
```

Idempotency strategy:
The application implements idempotency by maintaining a log of processed message IDs. Whenever a message with the same message id comes, the service skips the processing while also acknowledging the message. This makes sure that even if the same message is sent multiple times (could be due to network retries, broker redelivery, etc), the inventory is reserved only once, maintaining idempotency.

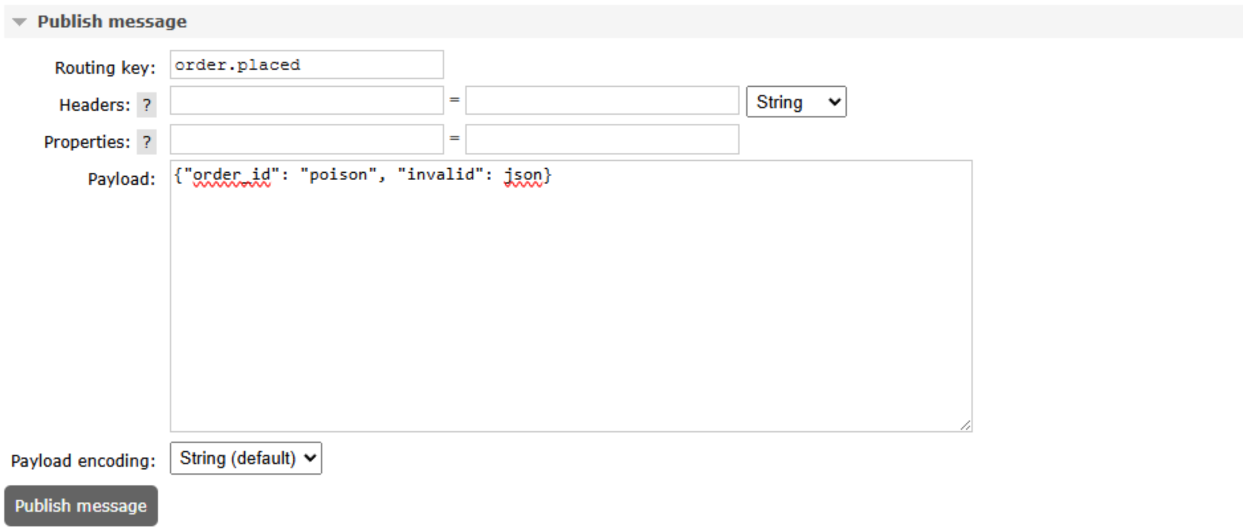
Poison Message Handling

Checking to see that there are no poison messages received:



```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker exec rabbitmq rabbitmqctl list_qu
eues name messages | findstr dlq
inventory_dlq   0
```

Poison Message:


Publish message

Routing key: `order.placed`

Headers: ? = String

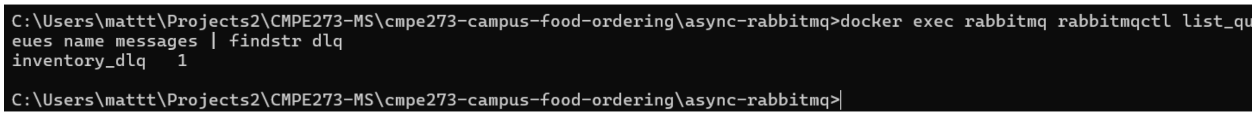
Properties: ? =

Payload: `{"order_id": "poison", "invalid": json}`

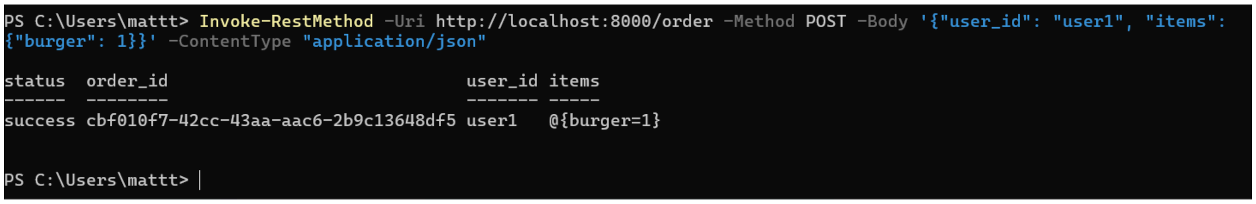
Payload encoding: String (default)

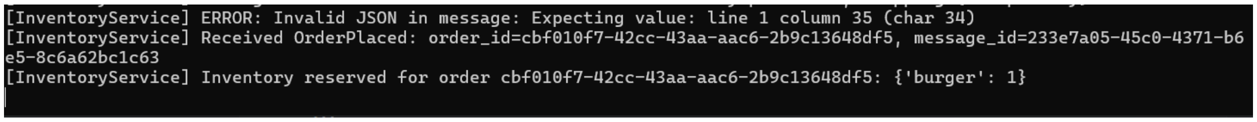
Publish message

1 message in DLQ after poison message above was published:


```
C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>docker exec rabbitmq rabbitmqctl list_qu
eues name messages | findstr dlq
inventory_dlq    1

C:\Users\mattt\Projects2\CMPE273-MS\cmpe273-campus-food-ordering\async-rabbitmq>
```

Posting another order to show that it still works (Error was from poison message):

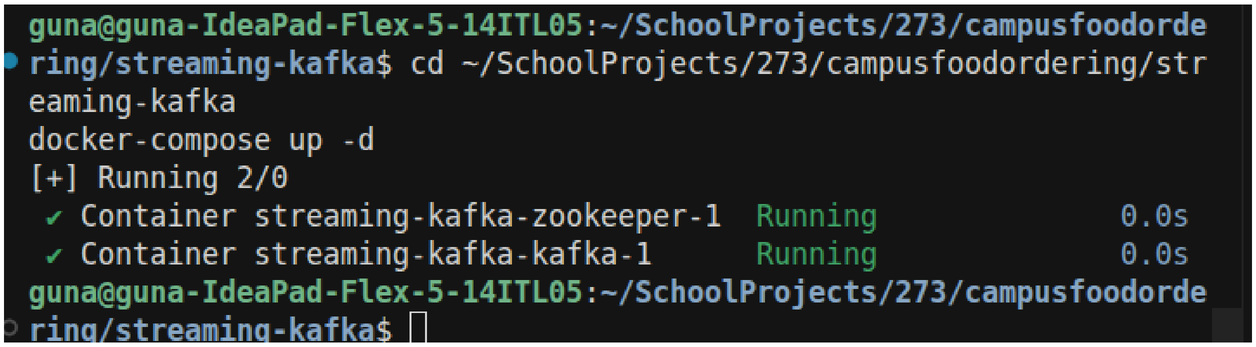

```
PS C:\Users\mattt> Invoke-RestMethod -Uri http://localhost:8000/order -Method POST -Body '{"user_id": "user1", "items":
{"burger": 1}}' -ContentType "application/json"

status  order_id                             user_id items
------  --------                             ------- -----
success cbf010f7-42cc-43aa-aac6-2b9c13648df5 user1   @{burger=1}


PS C:\Users\mattt>
```


```
[InventoryService] ERROR: Invalid JSON in message: Expecting value: line 1 column 35 (char 34)
[InventoryService] Received OrderPlaced: order_id=cbf010f7-42cc-43aa-aac6-2b9c13648df5, message_id=233e7a05-45c0-4371-b6
e5-8c6a62bc1c63
[InventoryService] Inventory reserved for order cbf010f7-42cc-43aa-aac6-2b9c13648df5: {'burger': 1}
```

Part C: Streaming with Kafka

Starting:

Running Docker Container for Streamking kafka


```
guna@guna-IdeaPad-Flex-5-14ITL05:~/SchoolProjects/273/campusfoodorde
ring/streaming-kafka$ cd ~/SchoolProjects/273/campusfoodordering/str
eaming-kafka
docker-compose up -d
[+] Running 2/0
 ✔ Container streaming-kafka-zookeeper-1  Running                   0.0s
 ✔ Container streaming-kafka-kafka-1      Running                   0.0s
guna@guna-IdeaPad-Flex-5-14ITL05:~/SchoolProjects/273/campusfoodorde
ring/streaming-kafka$ 
```

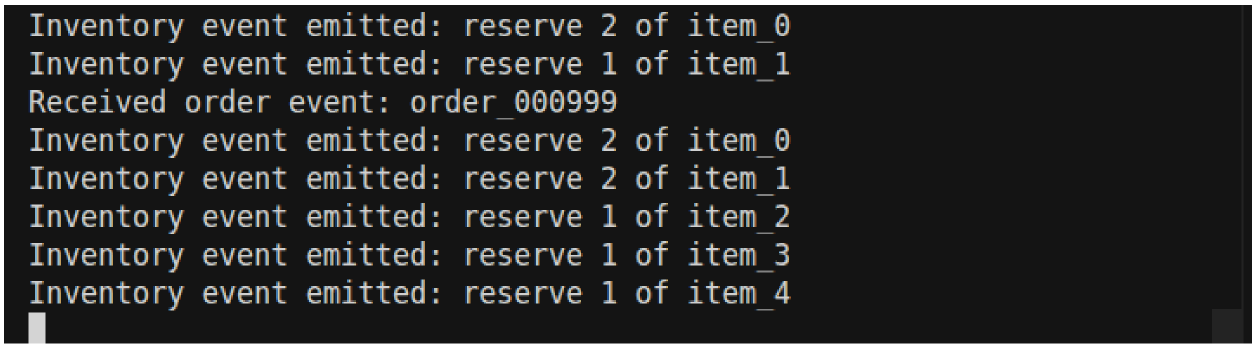
Analytics Consumer:

```
 cd ~/SchoolProjects/273/campusfoodordering/streaming-kafka/analytics_consumer
python3 consumer.py
Analytics consumer started. Computing metrics...

============================================================
ANALYTICS METRICS
============================================================
Total Orders: 1000
Failed Orders: 49
Failure Rate: 4.9%
Average Orders per Minute: 1000.0
Current Orders per Minute: 1000
============================================================



============================================================
ANALYTICS METRICS
============================================================
Total Orders: 2000
Failed Orders: 99
Failure Rate: 4.95%
Average Orders per Minute: 2000.0
```

```
Current Orders per Minute: 2000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 3000
Failed Orders: 155
Failure Rate: 5.17%
Average Orders per Minute: 3000.0
Current Orders per Minute: 3000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 4000
Failed Orders: 204
Failure Rate: 5.1%
Average Orders per Minute: 4000.0
Current Orders per Minute: 4000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 5000
Failed Orders: 256
```

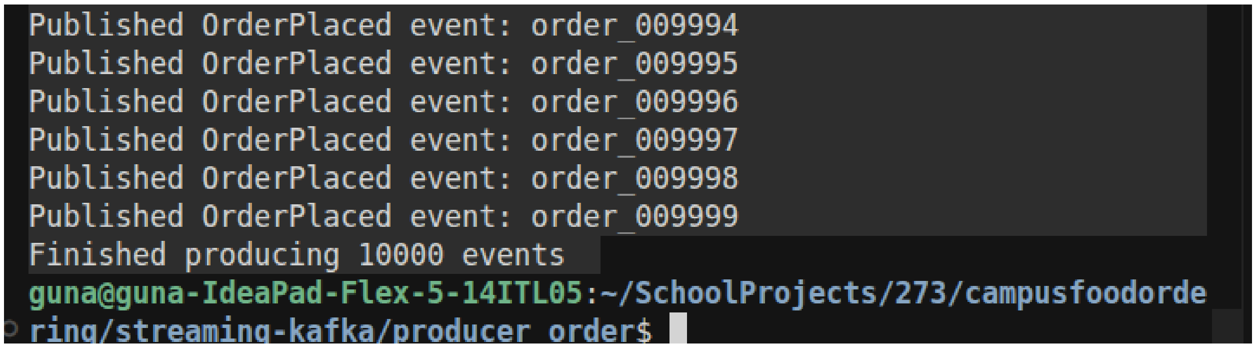
```
Failure Rate: 5.12%
Average Orders per Minute: 5000.0
Current Orders per Minute: 5000
============================================================


============================================================
ANALYTICS METRICS
============================================================
Total Orders: 6000
Failed Orders: 327
Failure Rate: 5.45%
Average Orders per Minute: 6000.0
Current Orders per Minute: 6000
============================================================


============================================================
ANALYTICS METRICS
============================================================
Total Orders: 7000
Failed Orders: 383
Failure Rate: 5.47%
Average Orders per Minute: 7000.0
Current Orders per Minute: 7000
============================================================


============================================================
ANALYTICS METRICS
============================================================
Total Orders: 8000
Failed Orders: 440
Failure Rate: 5.5%
Average Orders per Minute: 8000.0
Current Orders per Minute: 8000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 9000
Failed Orders: 487
Failure Rate: 5.41%
Average Orders per Minute: 9000.0
Current Orders per Minute: 9000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 10000
Failed Orders: 535
Failure Rate: 5.35%
Average Orders per Minute: 10000.0
Current Orders per Minute: 10000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 11000
Failed Orders: 585
Failure Rate: 5.32%
Average Orders per Minute: 5500.0
Current Orders per Minute: 1000
==================================================


==================================================
ANALYTICS METRICS
==================================================
Total Orders: 12000
Failed Orders: 631
Failure Rate: 5.26%
Average Orders per Minute: 4000.0
Current Orders per Minute: 1000
==================================================
```

Inventory Consumer:



```
Inventory event emitted: reserve 2 of item_0
Inventory event emitted: reserve 1 of item_1
Received order event: order_000999
Inventory event emitted: reserve 2 of item_0
Inventory event emitted: reserve 2 of item_1
Inventory event emitted: reserve 1 of item_2
Inventory event emitted: reserve 1 of item_3
Inventory event emitted: reserve 1 of item_4
```

Producer:



```
Published OrderPlaced event: order_009994
Published OrderPlaced event: order_009995
Published OrderPlaced event: order_009996
Published OrderPlaced event: order_009997
Published OrderPlaced event: order_009998
Published OrderPlaced event: order_009999
Finished producing 10000 events
guna@guna-IdeaPad-Flex-5-14ITL05:~/SchoolProjects/273/campusfoodorde
ring/streaming-kafka/producer_order$
```

Metrics Replay Initial:

```
ANALYTICS METRICS REPORT
============================================================
Total Orders: 1000
Failed Orders: 49
Failure Rate: 4.9%
Average Orders per Minute: 1000.0
Current Orders per Minute: 0

Orders by Minute:
  2026-02-17 14:33:00: 1000 orders
============================================================
```

Replay After:

```
ANALYTICS METRICS REPORT
============================================================
Total Orders: 1000
Failed Orders: 49
Failure Rate: 4.9%
Average Orders per Minute: 1000.0
Current Orders per Minute: 0

Orders by Minute:
  2026-02-17 14:33:00: 1000 orders
============================================================
```