SWE Assignment-3

Table of Contents

Team Members

1) Vineeth Bhat (2021101103)

2) Ishwar B Balappanawar (2021101023)

3) Swayam Agrawal (2021101068)

4) Mitansh Kayathwal (2021101026)

5) G.L.Vaishnavi (2023204009)

Architectural Patterns

1) Micro Services

Microservices Explanation

Microservices architecture breaks down the application into small, independent services, each responsible for specific functionalities.

- This allows for better scalability, flexibility, and maintainability.
- Each microservice can be developed, deployed, and scaled independently, which aligns well with the requirement of supporting up to 1000 requests/second.
- Additionally, microservices can help manage resources efficiently, enabling graceful degradation in case of sensor power outage or failures.

High level architectural diagrams

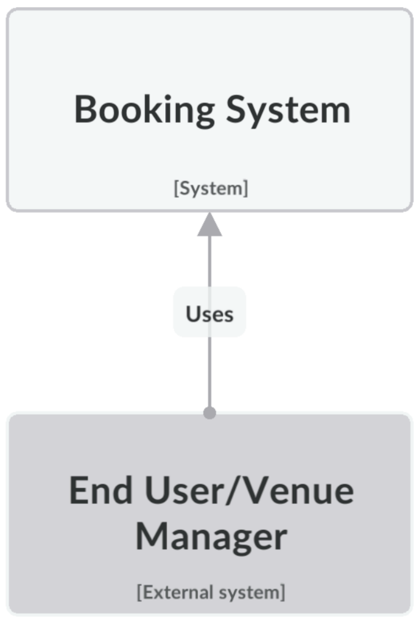
The diagrams can be viewed here as well with description for all the apps (containers):

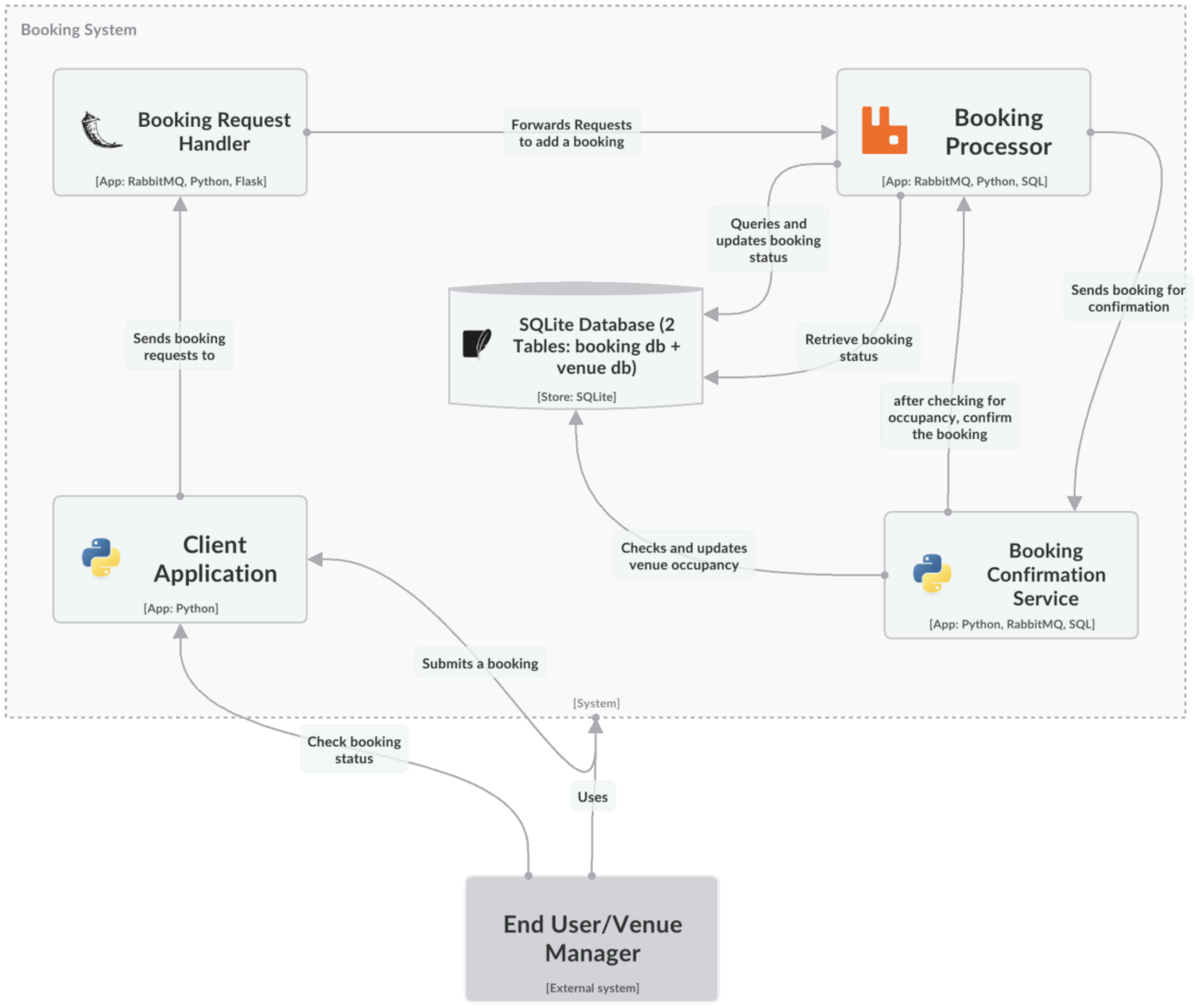
Explain complex software systems

IcePanel is a collaborative diagramming tool that helps software engineering and product teams align on technical decisions. Create an interactive map of your software systems and give your teams full context about how things work.

https://s.icepanel.io/aBtDn9Z0T0o6iY/MSqD

Architectural Diagrams


Booking System
[System]
Uses
End User/Venue Manager
[External system]



Booking System
Booking Request Handler
[App: RabbitMQ, Python, Flask]
Forwards Requests to add a booking
Booking Processor
[App: RabbitMQ, Python, SQL]
Queries and updates booking status
SQLite Database (2 Tables: booking db + venue db)
[Store: SQLite]
Retrieve booking status
Sends booking for confirmation
Sends booking requests to
after checking for occupancy, confirm the booking
Client Application
[App: Python]
Checks and updates venue occupancy
Booking Confirmation Service
[App: Python, RabbitMQ, SQL]
Submits a booking
[System]
Check booking status
Uses
End User/Venue Manager
[External system]

We can further decompose the containers to create a component diagram, but since the assignment mentions to create only high level diagrams, that hasn't been included.

An example: Consider "Booking Processor" container. We can use component blocks to represent internal components like "Request Receiver," "Booking Manager," "Confirmation Handler," and "Status Manager." We then connect these components with arrows to illustrate the flow of information, like how "Request Receiver" sends data to "Booking Manager."

Overview of System

Overview:
The seat booking system prototype is a microservices-based architecture built to handle booking requests for various venues. It consists of several components including a client interface, booking request handler, booking processor, and booking confirmation service. The system utilizes Flask for creating RESTful APIs and RabbitMQ for inter-service communication via message queues.

Client Code:
The client code provides a simple command-line interface for users to make bookings and check booking statuses. It interacts with the backend services via HTTP requests using the requests library.

Booking Request Handler (booking_request_handler.py):
This component receives booking requests from clients and forwards them to the booking processor via RabbitMQ. It exposes two endpoints '/book' for making bookings and '/status/<request_id>' for checking booking statuses.

Booking Processor (booking_processor.py):
The booking processor is responsible for processing booking requests and managing booking statuses. It listens for incoming requests from the booking request handler via RabbitMQ, processes them, updates the status, and sends back responses. It also handles status check requests and updates via separate queues.

Booking Confirmation Service (booking_confirmation.py):
Upon receiving confirmation messages from the booking processor, this service checks if the requested seats can be booked without exceeding the venue's maximum occupancy. It updates the venue's occupancy accordingly and sends status updates back to the booking processor.

Utilities (utilities.py):
This module contains utility functions for managing SQLite databases, including

creating tables, inserting data, updating statuses, and retrieving occupancy information. It is used by both the booking processor and the booking confirmation service.

Instructions for Running

You need RabbitMQ running for the code to function. Please use

```
sudo apt-get update
sudo apt-get install rabbitmq-server
sudo systemctl start rabbitmq-server
sudo systemctl enable rabbitmq-server
sudo systemctl status rabbitmq-server
```

After this, you may run the files as:

1. **Initial Setup:**
   Run
   `clean_init.py` to initialize the SQLite database and create necessary tables.
2. **Start Services:**
   - Run `booking_request_handler.py` to start the booking request handler service.
   - Run `booking_processor.py` to start the booking processor service.
   - Run `booking_confirmation.py` to start the booking confirmation service.
3. **Client Usage:**
   - Execute the client code (`client.py`) to interact with the system.
   - Choose the action:
     - `[1] Make Booking`: Enter venue ID and number of seats to make a booking.
     - `[2] Check Status`: Enter request ID to check booking status.
4. **Monitoring:**

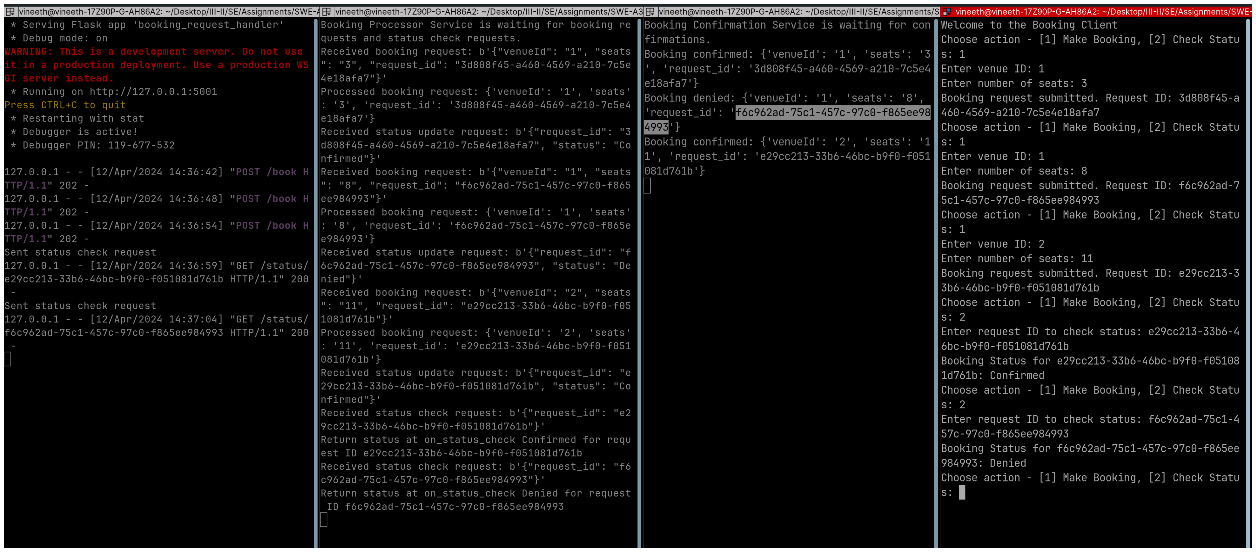
- Monitor the console output of each service for logs and messages.
- Check the SQLite database (`request_booking_db.sqlite`) for status updates and venue occupancy.

Quality

requirements

1. **Performance:**
    - **NOTE**: Added load testing results below this section
    - SQLite typically exhibits high performance for single-threaded, low to moderate concurrency workloads. It offers fast read and write operations, with transactional support ensuring data integrity. *For our operation, we understand that SQL gaurantees upto a million requests/second* satisfying the requirement of a 1000 requests/second (https://stackoverflow.com/a/57846740). This is subject to us having access to actual servers rather than just running it on our laptops.
    - RabbitMQ is known for its low-latency message delivery and high throughput, capable of handling thousands to millions of messages per second depending on hardware and configuration. *For our operation, we note that it can handle queue sizes of upto 1MiB*, showing that it is sufficient for maintaining enough messages in the queue to counteract if the SQL server ever slows down.
2. **Reliability:**
    - SQLite ensures reliability through ACID transactions and crash recovery mechanisms. It provides data durability by persisting changes to disk before committing transactions.
    - RabbitMQ ensures reliability by persisting messages to disk and employing message acknowledgments to prevent message loss. Clustering and mirrored queues enhance fault tolerance and data redundancy, further improving reliability.

Example



```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A
 * Serving Flask app 'booking_request_handler'
 * Debug mode: on
WARNING: This is a development server. Do not use
it in a production deployment. Use a production WS
GI server instead.
 * Running on http://127.0.0.1:5001
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 119-677-532

127.0.0.1 - - [12/Apr/2024 14:36:42] "POST /book H
TTP/1.1" 202 -
127.0.0.1 - - [12/Apr/2024 14:36:48] "POST /book H
TTP/1.1" 202 -
127.0.0.1 - - [12/Apr/2024 14:36:54] "POST /book H
TTP/1.1" 202 -
Sent status check request
127.0.0.1 - - [12/Apr/2024 14:36:59] "GET /status/
e29cc213-33b6-46bc-b9f0-f051081d761b HTTP/1.1" 200
 -
Sent status check request
127.0.0.1 - - [12/Apr/2024 14:37:04] "GET /status/
f6c962ad-75c1-457c-97c0-f865ee984993 HTTP/1.1" 200
 -
```

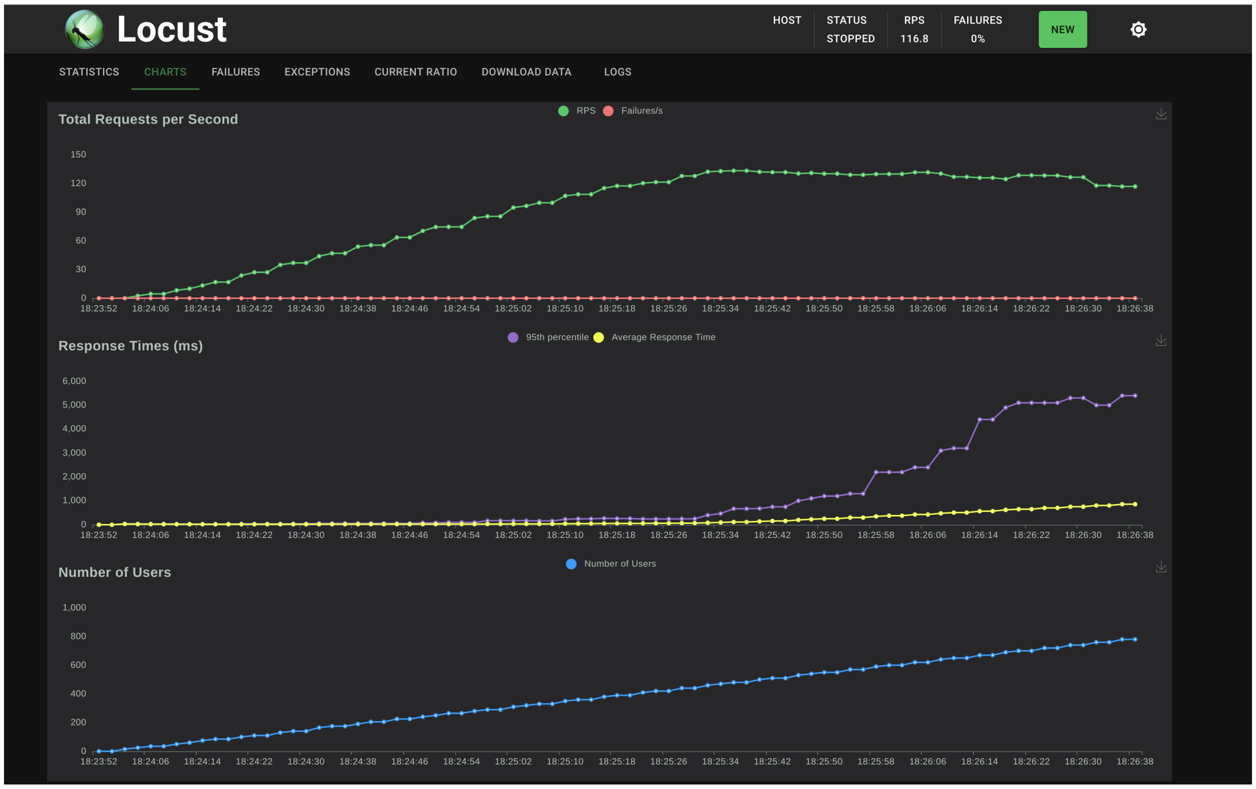
```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3
Booking Processor Service is waiting for booking re
quests and status check requests.
Received booking request: b'{"venueId": "1", "seats
": "3", "request_id": "3d808f45-a460-4569-a210-7c5e
4e18afa7"}'
Processed booking request: {'venueId': '1', 'seats'
: '3', 'request_id': '3d808f45-a460-4569-a210-7c5e4
e18afa7'}
Received status update request: b'{"request_id": "3
d808f45-a460-4569-a210-7c5e4e18afa7", "status": "Co
nfirmed"}'
Received booking request: b'{"venueId": "1", "seats
": "8", "request_id": "f6c962ad-75c1-457c-97c0-f865
ee984993"}'
Processed booking request: {'venueId': '1', 'seats'
: '8', 'request_id': 'f6c962ad-75c1-457c-97c0-f865e
e984993'}
Received status update request: b'{"request_id": "f
6c962ad-75c1-457c-97c0-f865ee984993", "status": "De
nied"}'
Received booking request: b'{"venueId": "2", "seats
": "11", "request_id": "e29cc213-33b6-46bc-b9f0-f05
1081d761b"}'
Processed booking request: {'venueId': '2', 'seats'
: '11', 'request_id': 'e29cc213-33b6-46bc-b9f0-f051
081d761b'}
Received status update request: b'{"request_id": "e
29cc213-33b6-46bc-b9f0-f051081d761b", "status": "Co
nfirmed"}'
Received status check request: b'{"request_id": "e2
9cc213-33b6-46bc-b9f0-f051081d761b"}'
Return status at on_status_check Confirmed for requ
est ID e29cc213-33b6-46bc-b9f0-f051081d761b
Received status check request: b'{"request_id": "f6
c962ad-75c1-457c-97c0-f865ee984993"}'
Return status at on_status_check Denied for request
 ID f6c962ad-75c1-457c-97c0-f865ee984993
```

```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/S
Booking Confirmation Service is waiting for con
firmations.
Booking confirmed: {'venueId': '1', 'seats': '3
', 'request_id': '3d808f45-a460-4569-a210-7c5e4
e18afa7'}
Booking denied: {'venueId': '1', 'seats': '8',
'request_id': 'f6c962ad-75c1-457c-97c0-f865ee98
4993'}
Booking confirmed: {'venueId': '2', 'seats': '1
1', 'request_id': 'e29cc213-33b6-46bc-b9f0-f051
081d761b'}
```

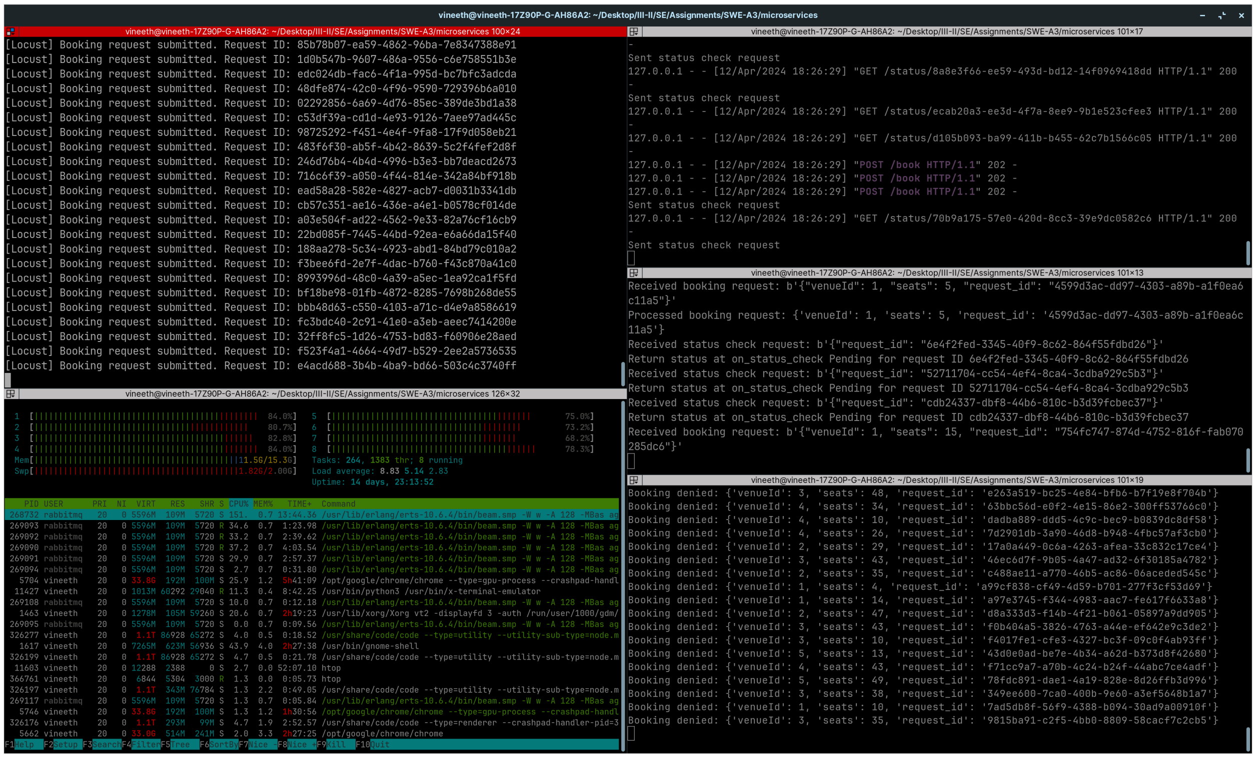
```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-
Welcome to the Booking Client
Choose action - [1] Make Booking, [2] Check Statu
s: 1
Enter venue ID: 1
Enter number of seats: 3
Booking request submitted. Request ID: 3d808f45-a
460-4569-a210-7c5e4e18afa7
Choose action - [1] Make Booking, [2] Check Statu
s: 1
Enter venue ID: 1
Enter number of seats: 8
Booking request submitted. Request ID: f6c962ad-7
5c1-457c-97c0-f865ee984993
Choose action - [1] Make Booking, [2] Check Statu
s: 1
Enter venue ID: 2
Enter number of seats: 11
Booking request submitted. Request ID: e29cc213-3
3b6-46bc-b9f0-f051081d761b
Choose action - [1] Make Booking, [2] Check Statu
s: 2
Enter request ID to check status: e29cc213-33b6-4
6bc-b9f0-f051081d761b
Booking Status for e29cc213-33b6-46bc-b9f0-f05108
1d761b: Confirmed
Choose action - [1] Make Booking, [2] Check Statu
s: 2
Enter request ID to check status: f6c962ad-75c1-4
57c-97c0-f865ee984993
Booking Status for f6c962ad-75c1-457c-97c0-f865ee
984993: Denied
Choose action - [1] Make Booking, [2] Check Statu
s: 
```

- In this demonstration, we run the handler, the processor, the confirmation and the client in separate terminals and run the following operations
  - Book 3 seats in venue 1 - this is accepted
  - Book 8 seats in venue 1 - this is denied since we define the max occupancy of the venue as 10
  - Book 11 seats in venue 2 - this is accepted as the max occupancy of venue 2 is 12
  - Check status of the operation of booking 11 seats in venue 2 - this returns Confirmed to the client
  - Check status of the operation of booking 8 seats in venue 1 - this returns Denied as explained earlier

Load Testing Results



Load Testing on Locust



```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3/microservices

vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3/microservices 100×24
[Locust] Booking request submitted. Request ID: 85b78b07-ea59-4862-96ba-7e8347388e91
[Locust] Booking request submitted. Request ID: 1d0b547b-9607-486a-9556-c6e758551b3e
[Locust] Booking request submitted. Request ID: edc024db-fac6-4f1a-995d-bc7bfc3adcda
[Locust] Booking request submitted. Request ID: 48dfe874-42c0-4f96-9590-729396b6a010
[Locust] Booking request submitted. Request ID: 02292856-6a69-4d76-85ec-389de3bd1a38
[Locust] Booking request submitted. Request ID: c53df39a-cd1d-4e93-9126-7aee97ad445c
[Locust] Booking request submitted. Request ID: 98725292-f451-4e4f-9fa8-17f9d058eb21
[Locust] Booking request submitted. Request ID: 483f6f30-ab5f-4b42-8639-5c2f4fef2d8f
[Locust] Booking request submitted. Request ID: 246d76b4-4b4d-4996-b3e3-bb7deacd2673
[Locust] Booking request submitted. Request ID: 716c6f39-a050-4f44-814e-342a84bf918b
[Locust] Booking request submitted. Request ID: ead58a28-582e-4827-acb7-d0031b3341db
[Locust] Booking request submitted. Request ID: cb57c351-ae16-436e-a4e1-b0578cf014de
[Locust] Booking request submitted. Request ID: a03e504f-ad22-4562-9e33-82a76cf16cb9
[Locust] Booking request submitted. Request ID: 22bd085f-7445-44bd-92ea-e6a66da15f40
[Locust] Booking request submitted. Request ID: 188aa278-5c34-4923-abd1-84bd79c010a2
[Locust] Booking request submitted. Request ID: f3bee6fd-2e7f-4dac-b760-f43c870a41c0
[Locust] Booking request submitted. Request ID: 8993996d-48c0-4a39-a5ec-1ea92ca1f5fd
[Locust] Booking request submitted. Request ID: bf18be98-01fb-4872-8285-7698b268de55
[Locust] Booking request submitted. Request ID: bbb48d63-c550-4103-a71c-d4e9a8586619
[Locust] Booking request submitted. Request ID: fc3bdc40-2c91-41e0-a3eb-aeec7414200e
[Locust] Booking request submitted. Request ID: 32ff8fc5-1d26-4753-bd83-f60906e28aed
[Locust] Booking request submitted. Request ID: f523f4a1-4664-49d7-b529-2ee2a5736535
[Locust] Booking request submitted. Request ID: e4acd688-3b4b-4ba9-bd66-503c4c3740ff
```

```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3/microservices 126×32

  1  [|||||||||||||||||||||||||||||||||||||||||||    84.0%]   5  [|||||||||||||||||||||||||||||||||||||||||||      75.0%]
  2  [||||||||||||||||||||||||||||||||||||||||       80.7%]   6  [|||||||||||||||||||||||||||||||||||||||          73.2%]
  3  [||||||||||||||||||||||||||||||||||||||||||     82.8%]   7  [||||||||||||||||||||||||||||||||||||||           68.2%]
  4  [|||||||||||||||||||||||||||||||||||||||||||    84.0%]   8  [|||||||||||||||||||||||||||||||||||||||||||||    78.3%]
  Mem[||||||||||||||||||||||||||||||||||||||||||11.5G/15.3G]   Tasks: 264, 1383 thr; 8 running
  Swp[||||||||||||||||||||||||||||||||||||||||||1.82G/2.00G]   Load average: 8.83 5.14 2.83
                                                               Uptime: 14 days, 23:13:52

    PID USER       PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 268732 rabbitmq    20   0 5596M  109M  5720 S 151.  0.7 13:44.36 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
 269093 rabbitmq    20   0 5596M  109M  5720 R 34.6  0.7  1:23.98 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
 269092 rabbitmq    20   0 5596M  109M  5720 R 33.2  0.7  2:39.62 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
 269090 rabbitmq    20   0 5596M  109M  5720 R 37.2  0.7  4:03.54 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
 269091 rabbitmq    20   0 5596M  109M  5720 S 29.9  0.7  2:57.37 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
 269094 rabbitmq    20   0 5596M  109M  5720 S  2.7  0.7  0:31.80 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
   5704 vineeth     20   0 33.8G  192M  100M S 25.9  1.2  5h41:09 /opt/google/chrome/chrome --type=gpu-process --crashpad-handl
  11427 vineeth     20   0 1013M 60292 29040 R 11.3  0.4  8:42.25 /usr/bin/python3 /usr/bin/x-terminal-emulator
 269108 rabbitmq    20   0 5596M  109M  5720 S 10.0  0.7  0:12.18 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
   1463 vineeth     20   0 1278M  105M 59260 S 20.6  0.7  2h19:23 /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/
 269095 rabbitmq    20   0 5596M  109M  5720 S  0.0  0.7  0:09.56 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
 326277 vineeth     20   0  1.1T 86928 65272 S  4.0  0.5  0:18.52 /usr/share/code/code --type=utility --utility-sub-type=node.m
   1617 vineeth     20   0 7265M  623M 56936 S 43.9  4.0  2h27:38 /usr/bin/gnome-shell
 326199 vineeth     20   0  1.1T 86928 65272 S  4.7  0.5  0:21.78 /usr/share/code/code --type=utility --utility-sub-type=node.m
  11603 vineeth     20   0 12288  2388     0 S  2.7  0.0 52:07.10 htop
 366761 vineeth     20   0  6844  5304  3000 R  1.3  0.0  0:05.73 htop
 326197 vineeth     20   0  1.1T  343M 76784 S  1.3  2.2  0:49.05 /usr/share/code/code --type=utility --utility-sub-type=node.m
 269117 rabbitmq    20   0 5596M  109M  5720 S  1.3  0.7  0:05.84 /usr/lib/erlang/erts-10.6.4/bin/beam.smp -W w -A 128 -MBas ag
   5746 vineeth     20   0 33.8G  192M  100M S  1.3  1.2  1h30:56 /opt/google/chrome/chrome --type=gpu-process --crashpad-handl
 326176 vineeth     20   0  1.1T  293M   99M S  4.7  1.9  2:52.57 /usr/share/code/code --type=renderer --crashpad-handler-pid=3
   5662 vineeth     20   0 33.0G  514M  241M S  2.0  3.3  2h27:25 /opt/google/chrome/chrome
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3/microservices 101×17
-
Sent status check request
127.0.0.1 - - [12/Apr/2024 18:26:29] "GET /status/8a8e3f66-ee59-493d-bd12-14f0969418dd HTTP/1.1" 200
-
Sent status check request
127.0.0.1 - - [12/Apr/2024 18:26:29] "GET /status/ecab20a3-ee3d-4f7a-8ee9-9b1e523cfee3 HTTP/1.1" 200
-
127.0.0.1 - - [12/Apr/2024 18:26:29] "GET /status/d105b093-ba99-411b-b455-62c7b1566c05 HTTP/1.1" 200
-
127.0.0.1 - - [12/Apr/2024 18:26:29] "POST /book HTTP/1.1" 202 -
127.0.0.1 - - [12/Apr/2024 18:26:29] "POST /book HTTP/1.1" 202 -
127.0.0.1 - - [12/Apr/2024 18:26:29] "POST /book HTTP/1.1" 202 -
Sent status check request
127.0.0.1 - - [12/Apr/2024 18:26:29] "GET /status/70b9a175-57e0-420d-8cc3-39e9dc0582c6 HTTP/1.1" 200
-
Sent status check request
```

```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3/microservices 101×13
Received booking request: b'{"venueId": 1, "seats": 5, "request_id": "4599d3ac-dd97-4303-a89b-a1f0ea6
c11a5"}'
Processed booking request: {'venueId': 1, 'seats': 5, 'request_id': '4599d3ac-dd97-4303-a89b-a1f0ea6c
11a5'}
Received status check request: b'{"request_id": "6e4f2fed-3345-40f9-8c62-864f55fdbd26"}'
Return status at on_status_check Pending for request ID 6e4f2fed-3345-40f9-8c62-864f55fdbd26
Received status check request: b'{"request_id": "52711704-cc54-4ef4-8ca4-3cdba929c5b3"}'
Return status at on_status_check Pending for request ID 52711704-cc54-4ef4-8ca4-3cdba929c5b3
Received status check request: b'{"request_id": "cdb24337-dbf8-44b6-810c-b3d39fcbec37"}'
Return status at on_status_check Pending for request ID cdb24337-dbf8-44b6-810c-b3d39fcbec37
Received booking request: b'{"venueId": 1, "seats": 15, "request_id": "754fc747-874d-4752-816f-fab070
285dc6"}'
```

```
vineeth@vineeth-17Z90P-G-AH86A2: ~/Desktop/III-II/SE/Assignments/SWE-A3/microservices 101×19
Booking denied: {'venueId': 3, 'seats': 48, 'request_id': 'e263a519-bc25-4e84-bfb6-b7f19e8f704b'}
Booking denied: {'venueId': 4, 'seats': 34, 'request_id': '63bbc56d-e0f2-4e15-86e2-300ff53766c0'}
Booking denied: {'venueId': 4, 'seats': 10, 'request_id': 'dadba889-ddd5-4c9c-bec9-b0839dc8df58'}
Booking denied: {'venueId': 4, 'seats': 26, 'request_id': '7d2901db-3a90-46d8-b948-4fbc57af3cb0'}
Booking denied: {'venueId': 2, 'seats': 29, 'request_id': '17a0a449-0c6a-4263-afea-33c832c17ce4'}
Booking denied: {'venueId': 3, 'seats': 43, 'request_id': '46ec6d7f-9b05-4a47-ad32-6f30185a4782'}
Booking denied: {'venueId': 2, 'seats': 35, 'request_id': 'c488ae11-a770-46b5-ac86-06aceded545c'}
Booking denied: {'venueId': 1, 'seats': 4, 'request_id': 'a99cf838-cf49-4d59-b701-277f3cf53d69'}
Booking denied: {'venueId': 1, 'seats': 14, 'request_id': 'a97e3745-f344-4983-aac7-fe617f6633a8'}
Booking denied: {'venueId': 2, 'seats': 47, 'request_id': 'd8a333d3-f14b-4f21-b061-05897a9dd905'}
Booking denied: {'venueId': 3, 'seats': 43, 'request_id': 'f0b404a5-3826-4763-a44e-ef642e9c3de2'}
Booking denied: {'venueId': 3, 'seats': 10, 'request_id': 'f4017fe1-cfe3-4327-bc3f-09c0f4ab93ff'}
Booking denied: {'venueId': 5, 'seats': 13, 'request_id': '43d0e0ad-be7e-4b34-a62d-b373d8f42680'}
Booking denied: {'venueId': 4, 'seats': 43, 'request_id': 'f71cc9a7-a70b-4c24-b24f-44abc7ce4adf'}
Booking denied: {'venueId': 5, 'seats': 49, 'request_id': '78fdc891-dae1-4a19-828e-8d26ffb3d996'}
Booking denied: {'venueId': 3, 'seats': 38, 'request_id': '349ee600-7ca0-400b-9e60-a3ef5648b1a7'}
Booking denied: {'venueId': 1, 'seats': 10, 'request_id': '7ad5db8f-56f9-4388-b094-30ad9a00910f'}
Booking denied: {'venueId': 3, 'seats': 35, 'request_id': '9815ba91-c2f5-4bb0-8809-58cacf7c2cb5'}
```

Results shown on local system

So, we successfully demonstrate that our system is scalable and can handle uptop 100 requests/seconds on our local machine. Increasing compute speeds will increase this as demonstrated by the image of system resources using `htop` .

2) Publisher Subscriber Pattern

Publisher-Subscriber Explanation

Pub-Sub architecture is a messaging pattern where senders of messages, called publishers, do not program the messages to be sent directly to specific receivers, called subscribers. Instead, messages are published to channels, without knowledge of what (if any) subscribers there may be. Subscribers express interest in one or more channels and receive messages only if they are subscribed to the relevant channel(s).

- Publishers and subscribers can be added or removed independently, allowing for **scalable** and **flexible** systems.
- loose coupling - components don't need to know about each other, promoting flexibility and easier modifications.
- Messages are sent asynchronously, improving system responsiveness.
- **Example Services**: Services like sensors publish data, while display services subscribe to receive and display relevant data.
- Subscribers control their own message consumption, enabling decentralized decision-making.
- **Load Balancing and Scalability-** Load balancers distribute messages among subscribers for scalability.

High level architectural diagrams

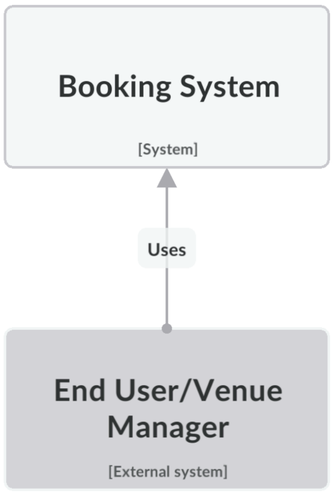
The diagrams can be viewed here as well with description for all the apps (containers):

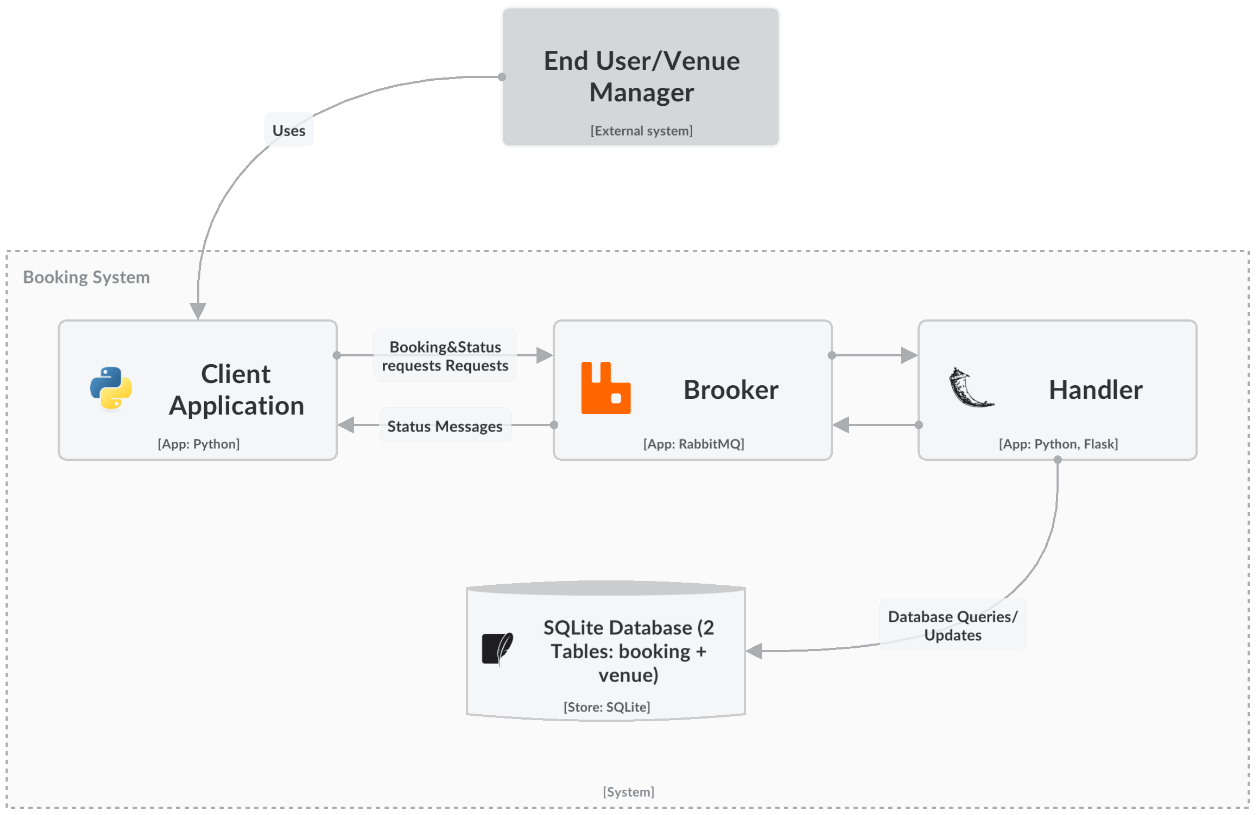
Explain complex software systems

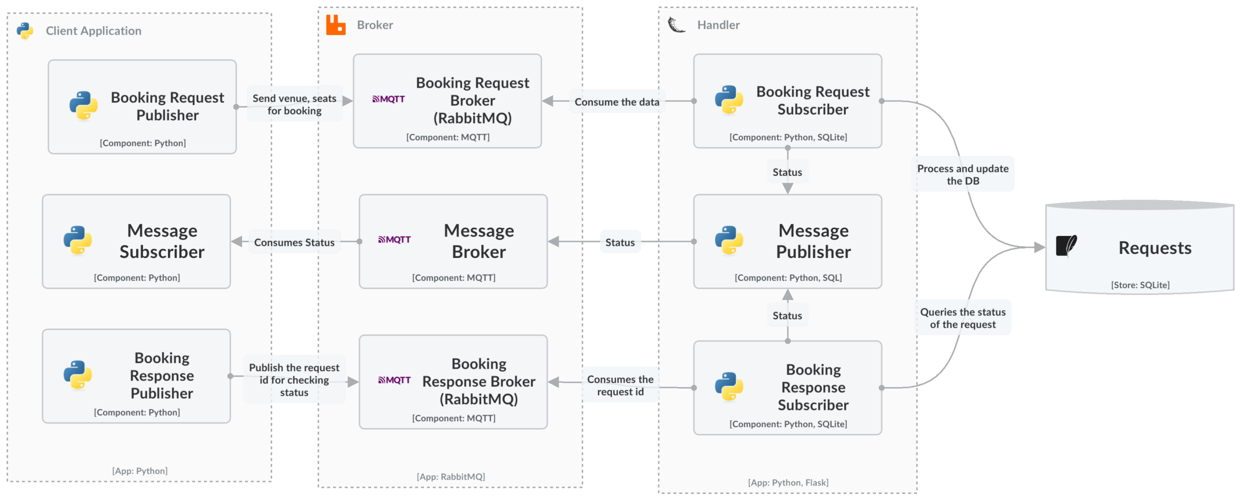
IcePanel is a collaborative diagramming tool that helps software engineering and product teams align on technical decisions. Create an interactive map of your software systems and give your teams full context about how things work.

https://s.icepanel.io/aBtDn9Z0T0o6iY/MSqD

Architectural Diagrams







Overview of the System

Overview:

This prototype employs a publisher-subscriber (Pub-Sub) architecture to distribute event information to subscribers interested in specific topics. It consists of distinct components, including publishers, subscribers, and a message broker for facilitating communication. The system utilizes RabbitMQ as the message broker for efficient message delivery.

Publishers:

`booking_request_publisher`: Responsible for publishing booking requests, including venue and seat details, to the message broker.

`booking_response_publisher`: Publishes request IDs to the message broker for querying booking request statuses.

`message_publisher`: Publishes status updates and other messages to the message broker for distribution to subscribers.

Subscribers:

`booking_request_subscriber`: Subscribes to booking requests, processes them, updates the database, and publishes status updates.

`booking_response_subscriber`: Subscribes to request IDs, queries the database for statuses, and publishes them.

`message_subscriber`: Subscribes to general messages and status updates published by the system.

Client:

Utilizes `booking_request_publisher` to submit booking requests and `booking_response_publisher` to inquire about the status of requests. Additionally, it subscribes to messages relevant to its requests via `message_subscriber`.

Handler:

Orchestrates the flow of data within the system. It includes logic for receiving requests from clients, directing them to the appropriate processing components (e.g., `booking_request_subscriber`, `booking_response_subscriber`), and managing the response flow back to clients.

Utilizes `message_publisher.py` for distributing status updates.

Message Broker:

`RabbitMQ`: Acts as the message broker for the system, facilitating communication between publishers and subscribers. It efficiently routes messages from publishers to subscribers based on topics or routing keys, ensuring reliable message delivery.

Utilities:
This module contains utility functions for managing SQLite databases, including creating tables, inserting data, updating statuses, and retrieving occupancy information. It is used by both the booking processor and the booking confirmation service.

Instructions for Running

You need RabbitMQ running for the code to function. Please use

```
sudo apt-get update
sudo apt-get install rabbitmq-server
sudo systemctl start rabbitmq-server
sudo systemctl enable rabbitmq-server
sudo systemctl status rabbitmq-server
```

After this, you may run the files as:

1. **Initial Setup:**
   Run
   `clean_init.py` to initialize the SQLite database and create necessary tables.
2. **Client Usage:**

- Execute the client code (`client.py`) to interact with the system.
- Choose the action:
    - `[1] Make Booking`: Enter venue ID and number of seats to make a booking.
    - `[2] Check Status`: Enter request ID to check booking status.

3. **Publishers/Subscribers :**
    - Run `request_subscriber.py` to start the listening to messages sent by booking_request_publisher (Client)
    - Run `response_subscriber.py` to start the listening to messages sent by booking_response_publisher (Client)
    - Run `message_` `subscriber.py` to start the listening to messages sent by message_publisher (Handler)
4. **Monitoring:**
    - Monitor the console output of each service for logs and messages.
    - Check the SQLite database (`request_booking_db.sqlite`) for status updates and venue occupancy using `testdb.py`

Quality

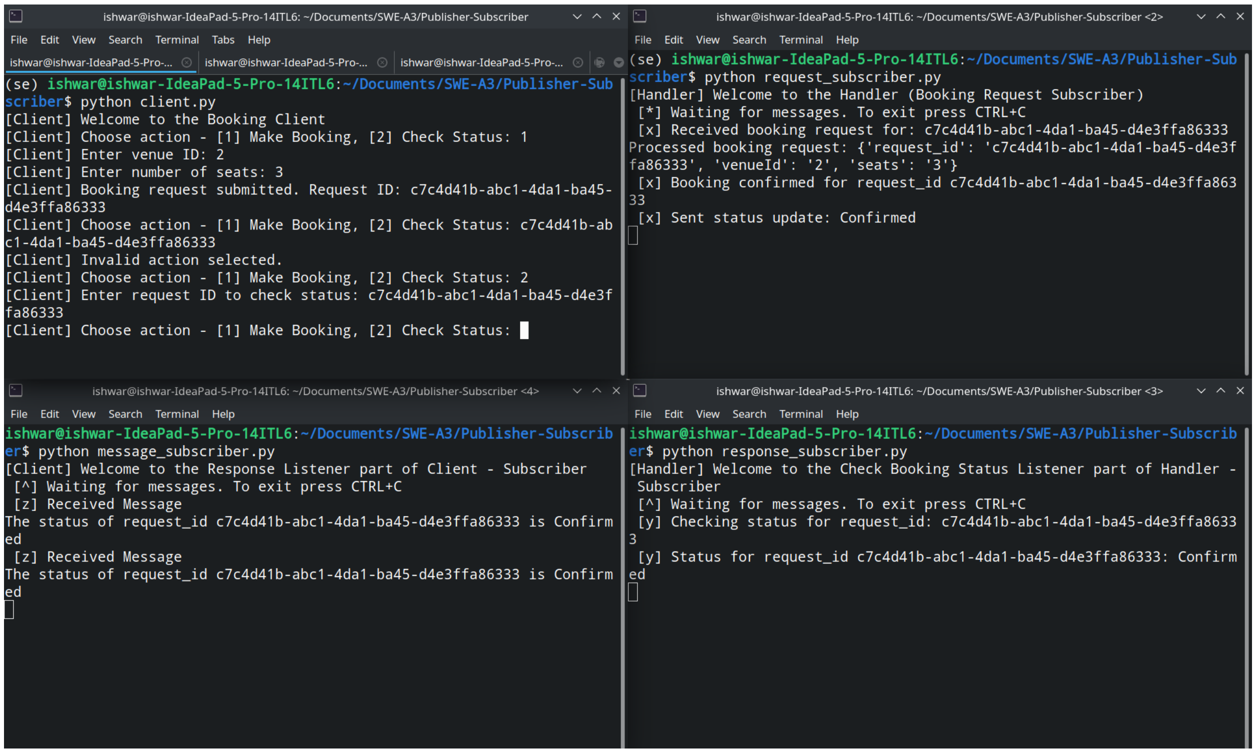
requirements

1. **Performance:**
    - **NOTE**: Added load testing results below this section
    - SQLite typically exhibits high performance for single-threaded, low to moderate concurrency workloads. It offers fast read and write operations, with transactional support ensuring data integrity. *For our operation, we understand that SQL gaurantees upto a million requests/second* satisfying the requirement of a 1000 requests/second (https://stackoverflow.com/a/57846740). This is subject to us having access to actual servers rather than just running it on our laptops.

- RabbitMQ is known for its low-latency message delivery and high throughput, capable of handling thousands to millions of messages per second depending on hardware and configuration. *For our operation, we note that it can handle queue sizes of upto 1MiB*, showing that it is sufficient for maintaining enough messages in the queue to counteract if the SQL server ever slows down.

2. **Reliability:**
   - SQLite ensures reliability through ACID transactions and crash recovery mechanisms. It provides data durability by persisting changes to disk before committing transactions.
   - RabbitMQ ensures reliability by persisting messages to disk and employing message acknowledgments to prevent message loss. Clustering and mirrored queues enhance fault tolerance and data redundancy, further improving reliability.

Example Demonstration:


ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber
File Edit View Search Terminal Tabs Help
ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber <2>
ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber <4>
ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber <3>

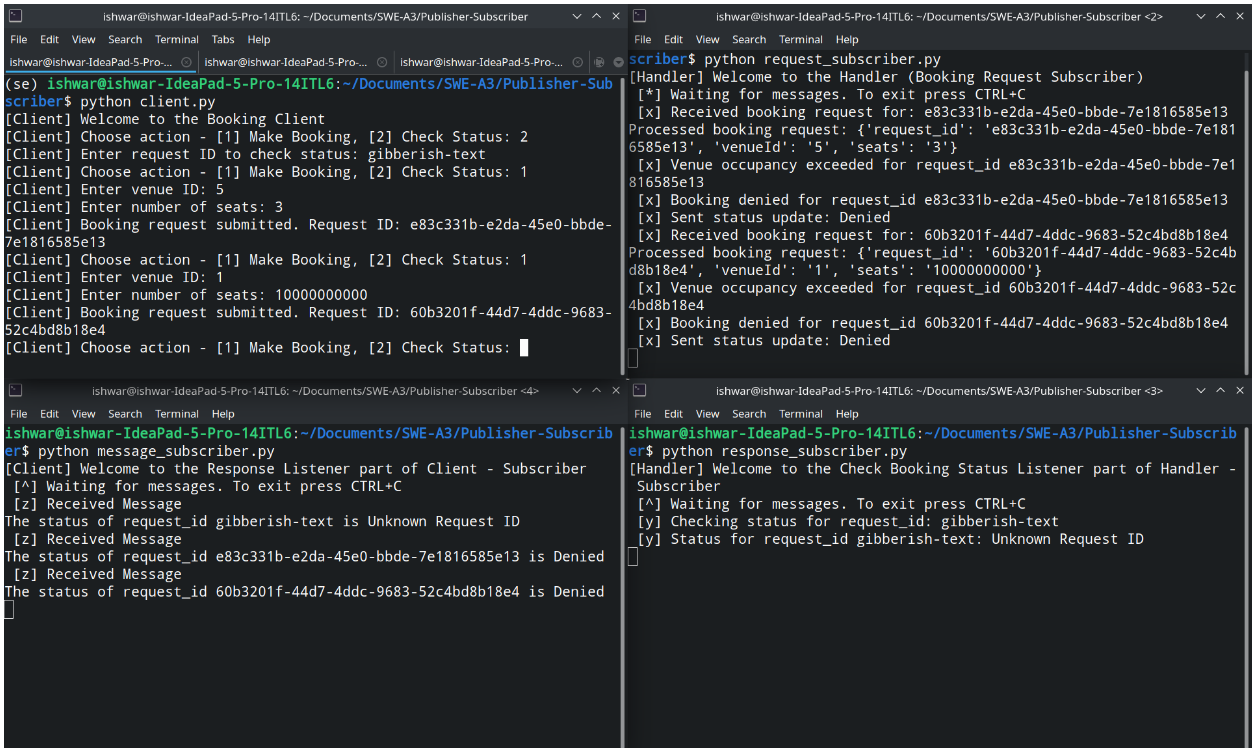
```
(se) ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Sub
scriber$ python client.py
[Client] Welcome to the Booking Client
[Client] Choose action - [1] Make Booking, [2] Check Status: 1
[Client] Enter venue ID: 2
[Client] Enter number of seats: 3
[Client] Booking request submitted. Request ID: c7c4d41b-abc1-4da1-ba45-
d4e3ffa86333
[Client] Choose action - [1] Make Booking, [2] Check Status: c7c4d41b-ab
c1-4da1-ba45-d4e3ffa86333
[Client] Invalid action selected.
[Client] Choose action - [1] Make Booking, [2] Check Status: 2
[Client] Enter request ID to check status: c7c4d41b-abc1-4da1-ba45-d4e3f
fa86333
[Client] Choose action - [1] Make Booking, [2] Check Status:
```

```
(se) ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Sub
scriber$ python request_subscriber.py
[Handler] Welcome to the Handler (Booking Request Subscriber)
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received booking request for: c7c4d41b-abc1-4da1-ba45-d4e3ffa86333
Processed booking request: {'request_id': 'c7c4d41b-abc1-4da1-ba45-d4e3f
fa86333', 'venueId': '2', 'seats': '3'}
 [x] Booking confirmed for request_id c7c4d41b-abc1-4da1-ba45-d4e3ffa863
33
 [x] Sent status update: Confirmed
```

```
ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Subscrib
er$ python message_subscriber.py
[Client] Welcome to the Response Listener part of Client - Subscriber
 [^] Waiting for messages. To exit press CTRL+C
 [z] Received Message
The status of request_id c7c4d41b-abc1-4da1-ba45-d4e3ffa86333 is Confirm
ed
 [z] Received Message
The status of request_id c7c4d41b-abc1-4da1-ba45-d4e3ffa86333 is Confirm
ed
```

```
ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Subscrib
er$ python response_subscriber.py
[Handler] Welcome to the Check Booking Status Listener part of Handler -
 Subscriber
 [^] Waiting for messages. To exit press CTRL+C
 [y] Checking status for request_id: c7c4d41b-abc1-4da1-ba45-d4e3ffa8633
3
 [y] Status for request_id c7c4d41b-abc1-4da1-ba45-d4e3ffa86333: Confirm
ed
```

- Book 2 seats in venue 2 - this is accepted, message sub receives the status.
- Check status of the above operation - response sub get the request and the status is sent to message sub



```
(se) ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Sub
scriber$ python client.py
[Client] Welcome to the Booking Client
[Client] Choose action - [1] Make Booking, [2] Check Status: 2
[Client] Enter request ID to check status: gibberish-text
[Client] Choose action - [1] Make Booking, [2] Check Status: 1
[Client] Enter venue ID: 5
[Client] Enter number of seats: 3
[Client] Booking request submitted. Request ID: e83c331b-e2da-45e0-bbde-
7e1816585e13
[Client] Choose action - [1] Make Booking, [2] Check Status: 1
[Client] Enter venue ID: 1
[Client] Enter number of seats: 10000000000
[Client] Booking request submitted. Request ID: 60b3201f-44d7-4ddc-9683-
52c4bd8b18e4
[Client] Choose action - [1] Make Booking, [2] Check Status:
```

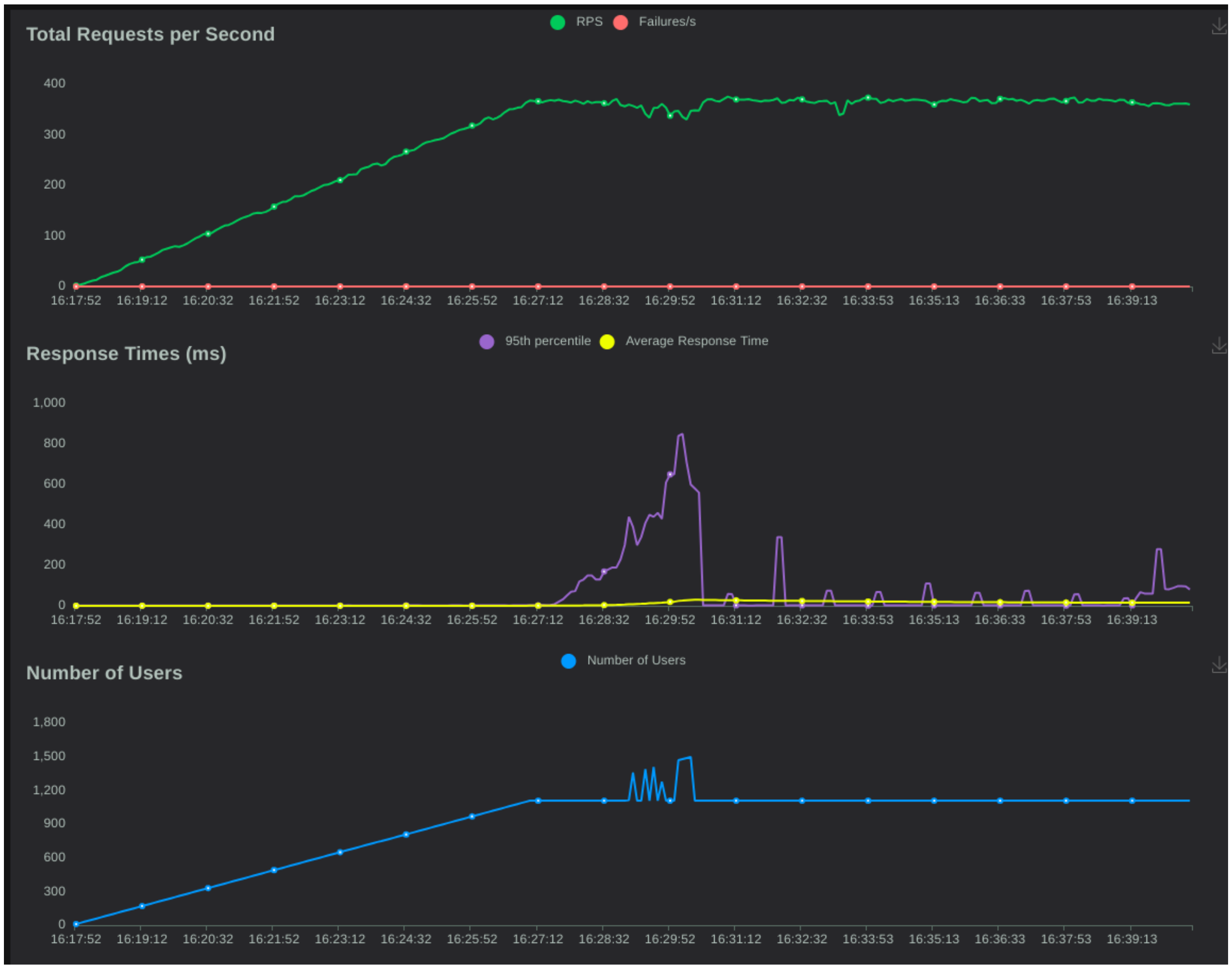
```
scriber$ python request_subscriber.py
[Handler] Welcome to the Handler (Booking Request Subscriber)
 [*] Waiting for messages. To exit press CTRL+C
 [x] Received booking request for: e83c331b-e2da-45e0-bbde-7e1816585e13
Processed booking request: {'request_id': 'e83c331b-e2da-45e0-bbde-7e181
6585e13', 'venueId': '5', 'seats': '3'}
 [x] Venue occupancy exceeded for request_id e83c331b-e2da-45e0-bbde-7e1
816585e13
 [x] Booking denied for request_id e83c331b-e2da-45e0-bbde-7e1816585e13
 [x] Sent status update: Denied
 [x] Received booking request for: 60b3201f-44d7-4ddc-9683-52c4bd8b18e4
Processed booking request: {'request_id': '60b3201f-44d7-4ddc-9683-52c4b
d8b18e4', 'venueId': '1', 'seats': '10000000000'}
 [x] Venue occupancy exceeded for request_id 60b3201f-44d7-4ddc-9683-52c
4bd8b18e4
 [x] Booking denied for request_id 60b3201f-44d7-4ddc-9683-52c4bd8b18e4
 [x] Sent status update: Denied
```

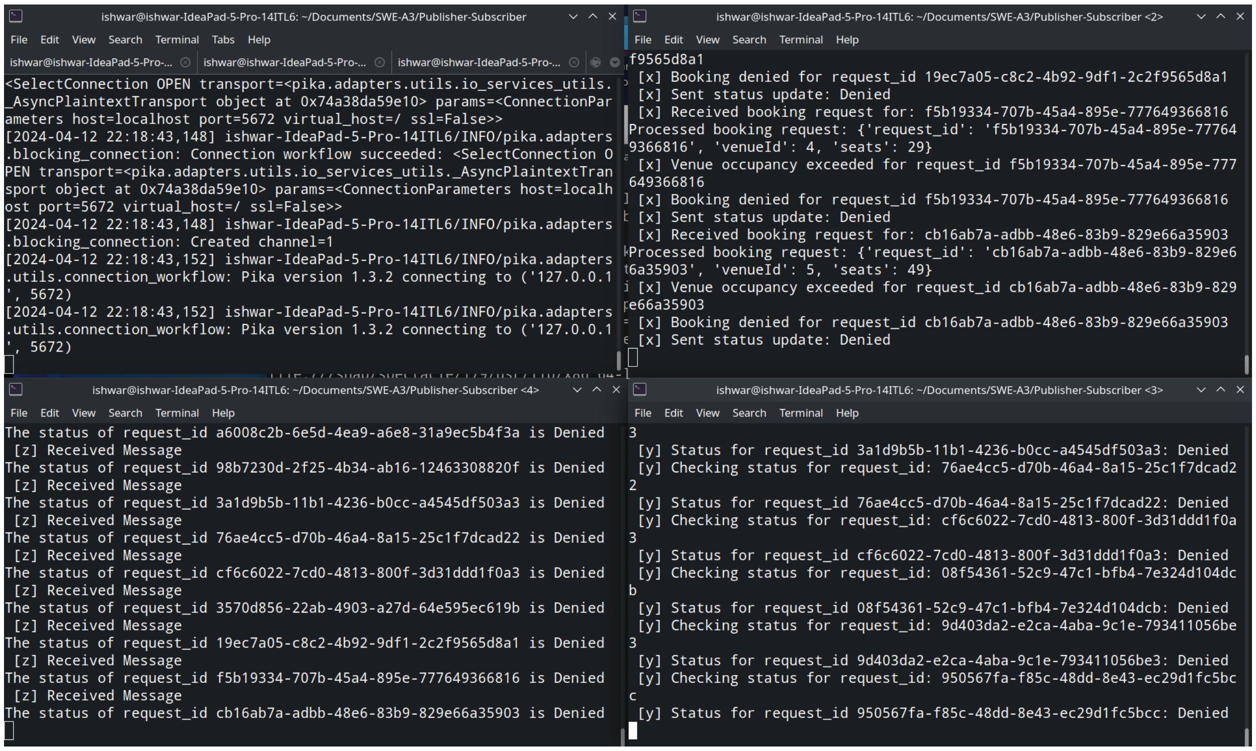
```
ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Subscrib
er$ python message_subscriber.py
[Client] Welcome to the Response Listener part of Client - Subscriber
 [^] Waiting for messages. To exit press CTRL+C
 [z] Received Message
The status of request_id gibberish-text is Unknown Request ID
 [z] Received Message
The status of request_id e83c331b-e2da-45e0-bbde-7e1816585e13 is Denied
 [z] Received Message
The status of request_id 60b3201f-44d7-4ddc-9683-52c4bd8b18e4 is Denied
```

```
ishwar@ishwar-IdeaPad-5-Pro-14ITL6:~/Documents/SWE-A3/Publisher-Subscrib
er$ python response_subscriber.py
[Handler] Welcome to the Check Booking Status Listener part of Handler -
 Subscriber
 [^] Waiting for messages. To exit press CTRL+C
 [y] Checking status for request_id: gibberish-text
 [y] Status for request_id gibberish-text: Unknown Request ID
```

- Checking status of a gibberish request id → Unknown Request id
- Similarly trying to book in unavailable venues/ booking too many seats → Denied

Load Testing Results


Total Requests per Second
RPS
Failures/s
400
300
200
100
0
16:17:52
16:19:12
16:20:32
16:21:52
16:23:12
16:24:32
16:25:52
16:27:12
16:28:32
16:29:52
16:31:12
16:32:32
16:33:53
16:35:13
16:36:33
16:37:53
16:39:13
Response Times (ms)
95th percentile
Average Response Time
1,000
800
600
400
200
0
Number of Users
Number of Users
1,800
1,500
1,200
900
600
300
0


ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber
File Edit View Search Terminal Tabs Help

```
<SelectConnection OPEN transport=<pika.adapters.utils.io_services_utils.
_AsyncPlaintextTransport object at 0x74a38da59e10> params=<ConnectionPar
ameters host=localhost port=5672 virtual_host=/ ssl=False>>
[2024-04-12 22:18:43,148] ishwar-IdeaPad-5-Pro-14ITL6/INFO/pika.adapters
.blocking_connection: Connection workflow succeeded: <SelectConnection O
PEN transport=<pika.adapters.utils.io_services_utils._AsyncPlaintextTran
sport object at 0x74a38da59e10> params=<ConnectionParameters host=localh
ost port=5672 virtual_host=/ ssl=False>>
[2024-04-12 22:18:43,148] ishwar-IdeaPad-5-Pro-14ITL6/INFO/pika.adapters
.blocking_connection: Created channel=1
[2024-04-12 22:18:43,152] ishwar-IdeaPad-5-Pro-14ITL6/INFO/pika.adapters
.utils.connection_workflow: Pika version 1.3.2 connecting to ('127.0.0.1
', 5672)
[2024-04-12 22:18:43,152] ishwar-IdeaPad-5-Pro-14ITL6/INFO/pika.adapters
.utils.connection_workflow: Pika version 1.3.2 connecting to ('127.0.0.1
', 5672)
```

ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber <2>
File Edit View Search Terminal Help

```
f9565d8a1
 [x] Booking denied for request_id 19ec7a05-c8c2-4b92-9df1-2c2f9565d8a1
 [x] Sent status update: Denied
 [x] Received booking request for: f5b19334-707b-45a4-895e-777649366816
Processed booking request: {'request_id': 'f5b19334-707b-45a4-895e-77764
9366816', 'venueId': 4, 'seats': 29}
 [x] Venue occupancy exceeded for request_id f5b19334-707b-45a4-895e-777
649366816
 [x] Booking denied for request_id f5b19334-707b-45a4-895e-777649366816
 [x] Sent status update: Denied
 [x] Received booking request for: cb16ab7a-adbb-48e6-83b9-829e66a35903
Processed booking request: {'request_id': 'cb16ab7a-adbb-48e6-83b9-829e6
6a35903', 'venueId': 5, 'seats': 49}
 [x] Venue occupancy exceeded for request_id cb16ab7a-adbb-48e6-83b9-829
e66a35903
 [x] Booking denied for request_id cb16ab7a-adbb-48e6-83b9-829e66a35903
 [x] Sent status update: Denied
```

ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber <4>
File Edit View Search Terminal Help

```
The status of request_id a6008c2b-6e5d-4ea9-a6e8-31a9ec5b4f3a is Denied
 [z] Received Message
The status of request_id 98b7230d-2f25-4b34-ab16-12463308820f is Denied
 [z] Received Message
The status of request_id 3a1d9b5b-11b1-4236-b0cc-a4545df503a3 is Denied
 [z] Received Message
The status of request_id 76ae4cc5-d70b-46a4-8a15-25c1f7dcad22 is Denied
 [z] Received Message
The status of request_id cf6c6022-7cd0-4813-800f-3d31ddd1f0a3 is Denied
 [z] Received Message
The status of request_id 3570d856-22ab-4903-a27d-64e595ec619b is Denied
 [z] Received Message
The status of request_id 19ec7a05-c8c2-4b92-9df1-2c2f9565d8a1 is Denied
 [z] Received Message
The status of request_id f5b19334-707b-45a4-895e-777649366816 is Denied
 [z] Received Message
The status of request_id cb16ab7a-adbb-48e6-83b9-829e66a35903 is Denied
```

ishwar@ishwar-IdeaPad-5-Pro-14ITL6: ~/Documents/SWE-A3/Publisher-Subscriber <3>
File Edit View Search Terminal Help

```
3
 [y] Status for request_id 3a1d9b5b-11b1-4236-b0cc-a4545df503a3: Denied
 [y] Checking status for request_id: 76ae4cc5-d70b-46a4-8a15-25c1f7dcad2
2
 [y] Status for request_id 76ae4cc5-d70b-46a4-8a15-25c1f7dcad22: Denied
 [y] Checking status for request_id: cf6c6022-7cd0-4813-800f-3d31ddd1f0a
3
 [y] Status for request_id cf6c6022-7cd0-4813-800f-3d31ddd1f0a3: Denied
 [y] Checking status for request_id: 08f54361-52c9-47c1-bfb4-7e324d104dc
b
 [y] Status for request_id 08f54361-52c9-47c1-bfb4-7e324d104dcb: Denied
 [y] Checking status for request_id: 9d403da2-e2ca-4aba-9c1e-793411056be
3
 [y] Status for request_id 9d403da2-e2ca-4aba-9c1e-793411056be3: Denied
 [y] Checking status for request_id: 950567fa-f85c-48dd-8e43-ec29d1fc5bc
c
 [y] Status for request_id 950567fa-f85c-48dd-8e43-ec29d1fc5bcc: Denied
```

So, we successfully demonstrate that our system is scalable and can handle uptop 350+ requests/seconds on our local machine.

Comparison

Average Response Time

Micro-services starts with **20 ms** response time, whereas Pub Sub starts with **0.3 ms**

Buy the time there are 500 users,

Response Time for Micro-services **- 619.89 ms,** Response Time for Pub Sub - **0.53 ms**

Buy the time there are 1000 users,

Response Time for Micro-services - **2014.42 ms,** Response Time for Pub Sub - **0.62 ms**

Requests per Second

Buy the time there are 500 users,

RPS for Micro-services - **89,** RPS for Pub Sub - **334**

Buy the time there are 1000 users,

RPS for Micro-services - **101,** RPS for Pub Sub - **367**

- 0 fails in both

Explanation

Possible reasons for the better performance of Pub Sub Architecture

1. **Decoupling in Pub Sub:**
   Pub Sub architecture inherently decouples the components involved in communication. In a micro-services architecture, each service typically communicates with others via synchronous HTTP requests, leading to tight coupling and potential bottlenecks as the system scales. In contrast, Pub Sub allows for asynchronous communication where publishers and subscribers are decoupled. This decoupling can lead to more efficient resource utilization and better scalability, especially under heavy loads.
2. **Message queues are faster than HTTP requests:**
   Pub Sub architectures often rely on message queues for communication between components. Message queues are optimized for fast, asynchronous message delivery and can handle high throughput with low latency. In contrast, micro-services communicating via HTTP requests may incur overhead due to protocol negotiation, connection setup, and teardown. As the number of users increases, the overhead associated with managing HTTP connections in micro-services architecture can lead to longer response times

and lower RPS compared to the more lightweight message queue approach in Pub Sub architecture.

3. **Scalability and Elasticity:** Pub Sub architectures are often more scalable and elastic compared to traditional micro-services architectures. With Pub Sub, it's easier to add or remove subscribers without impacting publishers or other subscribers. This scalability allows Pub Sub systems to handle spikes in traffic more gracefully by distributing the workload across multiple instances or nodes.
4. **Reduced Latency Through Parallel Processing:** In Pub Sub architectures, messages can be processed in parallel by multiple subscribers, leading to reduced overall latency. This parallel processing capability enables Pub Sub systems to handle high throughput without sacrificing responsiveness. On the other hand, in micro-services architectures, synchronous communication may lead to sequential processing of requests, which can introduce latency, especially under heavy loads.
5. **Fault Tolerance and Reliability:** Pub Sub architectures often provide built-in fault tolerance and reliability features. Messages published to a topic are typically persisted and can be redelivered to subscribers in case of failures. This ensures that messages are not lost even if some components in the system fail. In contrast, in micro-services architectures, ensuring fault tolerance and reliability often requires additional complexity and infrastructure, such as implementing retries, circuit breakers, and distributed tracing.
6. **Optimized Resource Utilization:** Pub Sub architectures can optimize resource utilization by dynamically adjusting the number of instances or nodes based on workload demands. For example, Pub Sub systems can automatically scale up or down the number of subscribers based on message volume or processing requirements. This dynamic resource allocation allows Pub Sub architectures to efficiently utilize computing resources and minimize costs, especially in cloud environments where resources are billed based on usage.