

Name	Prakriti Sharma
Subject	CS550 Advance Operating Systems
CWID	A20575259
Topic	PA1

INDEX

SR. No	Contents	Page No.
1	Requirement	2
2	Tools Used	2
3	What exactly is a Publisher subscriber system? (Diagram Illustration)	2
4	Structure of the code	2
5	The Process Description	4
6	Screenshots of dry run for 7 given APIs (Postman and curls)	4-8
7	Testing Results – Single Pub Sub Connection Testing	9
8	Testing Results – Multiple Pub Sub Connection Testing	9-13
9	Testing Results – Benchmarking the 7 APIs	13-16
10	Testing Results – Plotting 7 Benchmark testing graphs for each API	17-20
11	What is Ping Pong Testing?	21
12	Testing Results – Single Ping Pong Testing	21
13	Testing Results – Multiple Ping Pong Testing	22
14	Testing Results – Graph of Ping Pong Testing	22
15	Note	23
16	Conclusion	23-24

Requirement:

The assignment is a Publisher Subscriber System in JAVA language.

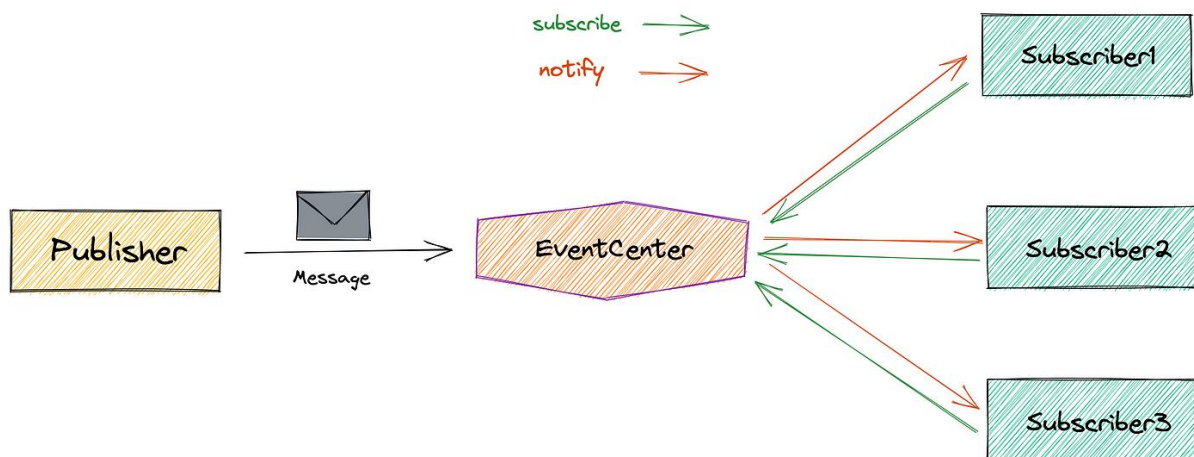
Tools Used:

Maven, Gradle, XChart, JavaFX, SpringBoot

What exactly is a Publisher Subscriber System?

A publisher-subscriber system is a messaging pattern where publishers send messages to a topic without knowing who subscribes to it. Subscribers express interest in specific topics and receive messages published to those topics. This decouples the sender and receiver, allowing for scalable and flexible communication. It's commonly used in event-driven architectures, messaging systems, and real-time applications.

DIAGRAM



Structure of the code:

The requirement according to the PA1 document states that we can divide the structure of the code into three different parts:

1. ClientAPILibrary

Client API Library has three files in total, one which declares an interface for the publisher and its functionalities, the client API implementation implements that interface and the client API controller renders that implementation into https REST APIs.

2. ClientProgram

The client program pulls the functions together for publisher and the subscriber individually, verifying its very basic functionality that is:

- Registering a publisher on the server. It has been assigned a specific format that is **PUB-#num**. Each time someone hits that endpoint, a new publisher with a counter++ equals to the #num is created.
- It later pulls the function to **register a topic under the publisher**.
- The publisher can **push a message to the pool of messages** using the push message to pool functionality.

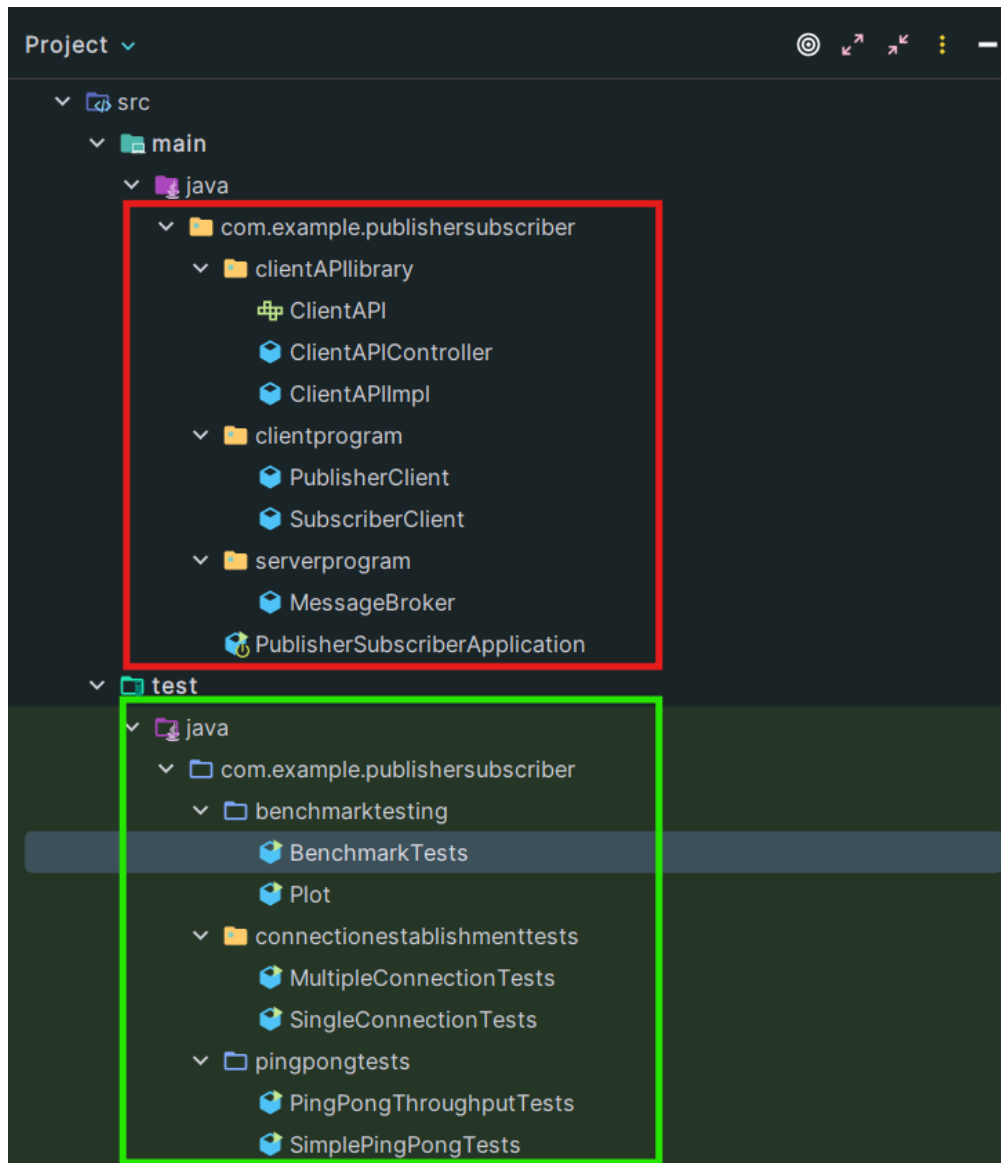
- Registering the subscriber on the server. It has been assigned a specific format that is **SUB-#num**. Each time someone hits that endpoint, a new subscriber with a counter++ equals to the #num is created.
- The subscriber then is able to **subscribe to a topic**.
- A **subscriber can pull the messages from the pool**, by providing its subscriber Id and the topic it wants to pull the freshly pushed messages from.
- **Basically, if a message is read by SUB-1 and not read by SUB-2, it will always stay in the memory. We use flagging concept to log if the message is read by all the subscriber or not, or what subscriber Ids have already read the message.**
- Once, a set of pushed messages are read by all the publishers, it is referred to as junk and then the junk is **dumped from the buffer memory**.

3. ServerProgram

The server program **maintains the anonymity of the communication**. It hides the details of message transportation from the users that is the client programs do not have to take care of the connection establishments. The **message broker** file initialises a server like communication platform **to hide the publishing details from the subscriber and vice versa**. It also maintains the buffer for different topic and handles request from the clients.

This covers the Overview section requirements.

The structure of the project files are as follows:



The Process Description:

The functionality concludes by deploying the APIs in the below order:

1. *registerPublisher()*
2. *createTopic(PID, String topic)*
3. *deleteTopic(Pid, String topic, String message)*
4. *send(PID, String topic, String message)*
5. *registerSubscriber()*
6. *subscribeTopic(SID, String topic)*
7. *List<String> pullMessages(SID, String topic)*

Below is the screenshot of the working APIs and the fulfilment of basic functionalities of the assignment:

This covers the Client API implementation requirements.

Register Publisher API

cURL : curl --location --request POST 'http://localhost:8080/api/publisher/register'

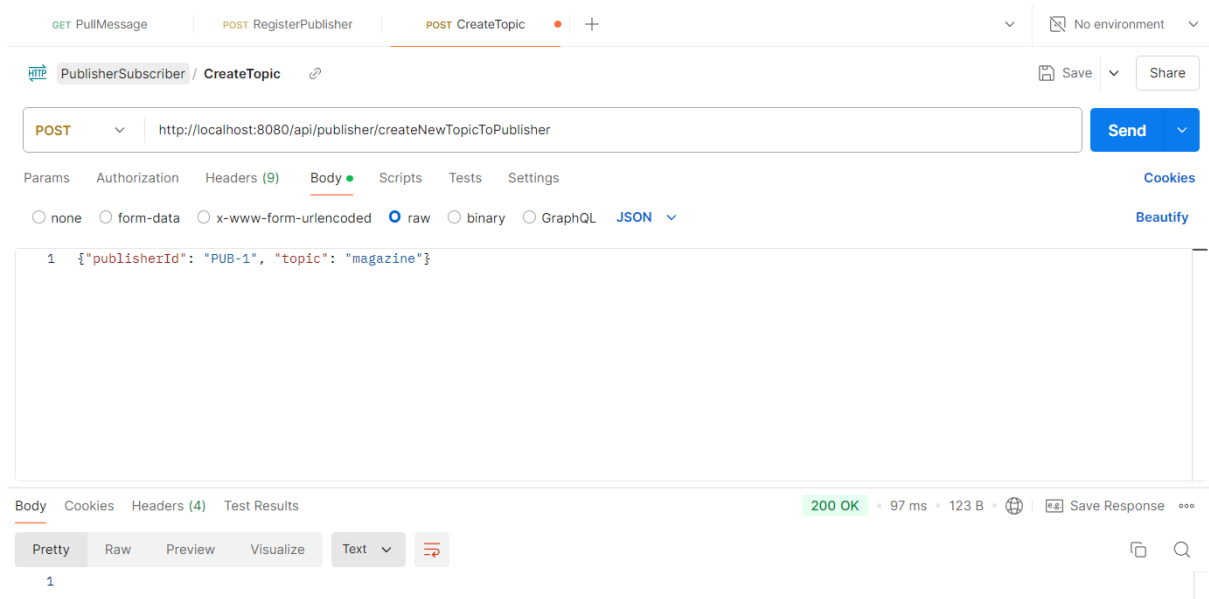
The register API return a unique ID which allows the client to act as a publisher or subscriber.

The screenshot shows a REST client interface for the 'POST RegisterPublisher' API. The URL is 'http://localhost:8080/api/publisher/register'. The response is '200 OK' with a status of 'OK', a response time of '46 ms', and a body size of '168 B'. The response body is 'PUB-1'.

Key	Value	Description
Key	Value	Description

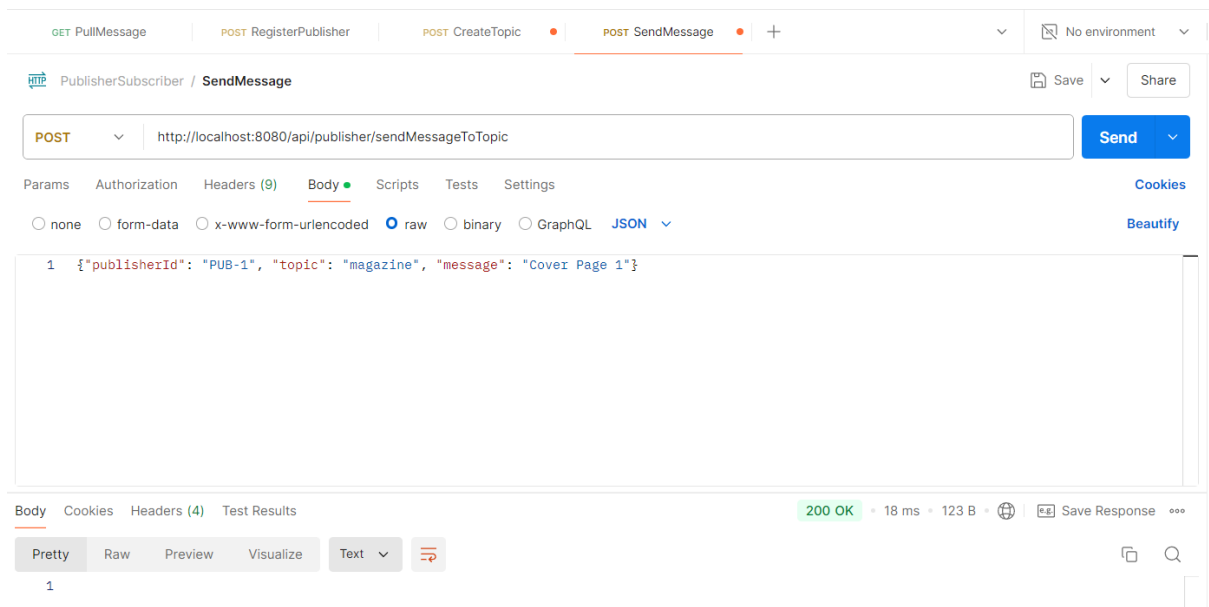
Create Topic API :

**cURL: curl --location 'http://localhost:8080/api/publisher/createNewTopicToPublisher' **
**--header 'Content-Type: application/json' **
--data '{"publisherId": "PUB-1", "topic": "magazine"}'



Send Message API :

```
cURL: curl --location 'http://localhost:8080/api/publisher/sendMessageToTopic' \
--header 'Content-Type: application/json' \
--data '{"publisherId": "PUB-1", "topic": "magazine", "message": "Cover Page 1"}'
```



Register Subscriber API :

cURL: `curl --location --request POST 'http://localhost:8080/api/subscriber/register'`

The screenshot shows the Postman interface for the **RegisterSubscriber** API endpoint. The URL is `http://localhost:8080/api/subscriber/register` and the method is **POST**. The Headers tab is selected, showing the following headers:

Key	Value	Description
<input checked="" type="checkbox"/> Postman-Token	<calculated when request is sent>	
<input checked="" type="checkbox"/> Content-Length	0	
<input checked="" type="checkbox"/> Host	<calculated when request is sent>	
<input checked="" type="checkbox"/> User-Agent	PostmanRuntime/7.42.0	
<input checked="" type="checkbox"/> Accept	*/*	
<input checked="" type="checkbox"/> Accept-Encoding	gzip, deflate, br	
<input checked="" type="checkbox"/> Connection	keep-alive	

The response is **200 OK** with a status of 12 ms and 168 B. The response body is displayed in the **Body** tab, showing the text `SUB-1`.

Subscribe to topic API

cURL: `curl --location 'http://localhost:8080/api/subscriber/subscribeToTopic' \`
`--header 'Content-Type: application/json' \`
`--data '{"subscriberId": "SUB-1", "topic": "magazine"}'`

The screenshot shows the Postman interface for the **SubscribeToTopic** API endpoint. The URL is `http://localhost:8080/api/subscriber/subscribeToTopic` and the method is **POST**. The Headers tab is selected, showing 9 headers. The Body tab is also selected, showing the JSON body:

```
1 {"subscriberId": "SUB-1", "topic": "magazine"}
```

The response is **200 OK** with a status of 9 ms and 123 B. The response body is displayed in the **Body** tab, showing the text `SUB-1`.

Pull messages from topic API

cURL: curl --location
 'http://localhost:8080/api/subscriber/pullMessagesFromPool?subscriberId=SUB-1&topic=magazine'

The screenshot shows a Postman interface for a GET request to `http://localhost:8080/api/subscriber/pullMessagesFromPool?subscriberId=SUB-1&topic=magazine`. The request is successful, returning a 200 OK status with a response time of 25 ms and a body size of 180 B. The response body is displayed in JSON format, showing an array with one element: `["Cover Page 1"]`.

Key	Value	Description
subscriberId	SUB-1	
topic	magazine	

Delete Topic API

cURL: curl --location --request DELETE
 'http://localhost:8080/api/publisher/deleteTopicFromPublisher' \
 --header 'Content-Type: application/json' \
 --data '{"publisherId": "PUB-1", "topic": "magazine"}'

The screenshot shows a Postman interface for a DELETE request to `http://localhost:8080/api/publisher/deleteTopicFromPublisher`. The request is successful, returning a 200 OK status with a response time of 11 ms and a body size of 123 B. The request body is displayed in JSON format: `{"publisherId": "PUB-1", "topic": "magazine"}`.

Testing Results:

Single Connection Testing

```

Project > SingleConnectionTests.java x
src
  main
    java
      com.example.publishersubscribe
        SingleConnectionTests.java
Run
  PublisherSubscriber [com.example.publishersubscribe.c... x

> 12:37:18 PM: Executing 'com.example.publishersubscribe.connectionestablishmenttests.SingleConnectionTests.main()'...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE

> Task :com.example.publishersubscribe.connectionestablishmenttests.SingleConnectionTests.main()
Newly registered Publisher ID: PUB-1
Created a topic successfully for PUB-1
Messages published by publisher: PUB-1
***** The messages lie in the pool *****
Subscriber SUB-1 subscribed to topic : education
Subscriber SUB-1 received: Publisher 1 Message 1
Subscriber SUB-1 received: Publisher 1 Message 2
The single connection functionality has been verified.

BUILD SUCCESSFUL in 5s
3 actionable tasks: 1 executed, 2 up-to-date
> 12:37:25 PM: Execution finished 'com.example.publishersubscribe.connectionestablishmenttests. ...
  
```

This tests the grounded basic functionality of the code that is one publisher-one subscriber system and just verifies the message receipt on the subscriber's end.

Multiple Connection Testing

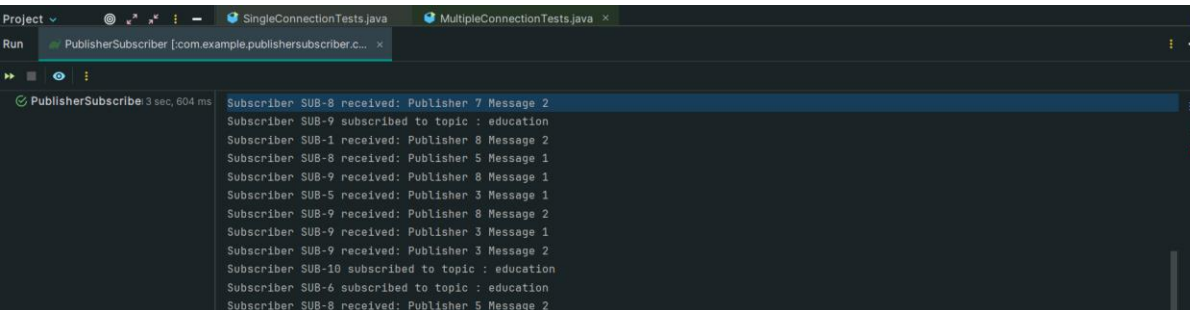
```

Project > SingleConnectionTests.java x MultipleConnectionTests.java x
Run
  PublisherSubscriber [com.example.publishersubscribe.c... x

> 12:41:16 PM: Executing 'com.example.publishersubscribe.connectionestablishmenttests.MultipleC ...

> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE

> Task :com.example.publishersubscribe.connectionestablishmenttests.MultipleConnectionTests.main()
Newly registered Publisher ID: PUB-1
Newly registered Publisher ID: PUB-5
Created a topic successfully for PUB-1
Newly registered Publisher ID: PUB-5
Created a topic successfully for PUB-5
Messages published by publisher: PUB-1
Newly registered Publisher ID: PUB-4
Created a topic successfully for PUB-4
Messages published by publisher: PUB-4
***** The messages lie in the pool *****
Newly registered Publisher ID: PUB-3
Created a topic successfully for PUB-3
Newly registered Publisher ID: PUB-2
Created a topic successfully for PUB-2
Messages published by publisher: PUB-2
***** The messages lie in the pool *****
Messages published by publisher: PUB-3
***** The messages lie in the pool *****
***** The messages lie in the pool *****
Messages published by publisher: PUB-5
  
```



Project ▾ SingleConnectionTests.java MultipleConnectionTests.java

Run PublisherSubscriber [com.example.publishersubscriber.c... x]

✓ PublisherSubscriber 3 sec, 604 ms

```
Subscriber SUB-8 received: Publisher 7 Message 2
Subscriber SUB-9 subscribed to topic : education
Subscriber SUB-1 received: Publisher 8 Message 2
Subscriber SUB-8 received: Publisher 5 Message 1
Subscriber SUB-9 received: Publisher 8 Message 1
Subscriber SUB-5 received: Publisher 3 Message 1
Subscriber SUB-9 received: Publisher 8 Message 2
Subscriber SUB-9 received: Publisher 3 Message 1
Subscriber SUB-9 received: Publisher 3 Message 2
Subscriber SUB-10 subscribed to topic : education
Subscriber SUB-6 subscribed to topic : education
Subscriber SUB-8 received: Publisher 5 Message 2
Subscriber SUB-8 received: Publisher 6 Message 1
Subscriber SUB-8 received: Publisher 6 Message 2
Subscriber SUB-8 received: Publisher 4 Message 1
Subscriber SUB-2 received: Publisher 8 Message 1
Subscriber SUB-1 received: Publisher 3 Message 1
Subscriber SUB-3 subscribed to topic : education
Subscriber SUB-3 received: Publisher 8 Message 1
Subscriber SUB-3 received: Publisher 8 Message 2
Subscriber SUB-3 received: Publisher 3 Message 1
Subscriber SUB-3 received: Publisher 3 Message 2
Subscriber SUB-3 received: Publisher 1 Message 1
Subscriber SUB-3 received: Publisher 1 Message 2
Subscriber SUB-3 received: Publisher 7 Message 1
Subscriber SUB-3 received: Publisher 7 Message 2
Subscriber SUB-3 received: Publisher 5 Message 1
Subscriber SUB-3 received: Publisher 5 Message 2
Subscriber SUB-3 received: Publisher 6 Message 1
```

Subscriber SUB-3 received: Publisher 6 Message 1

SubscriberSubscriber > src > . > test > java > com > example > publishersubscriber > connectionestablishmenttests > MultipleConnectionTests

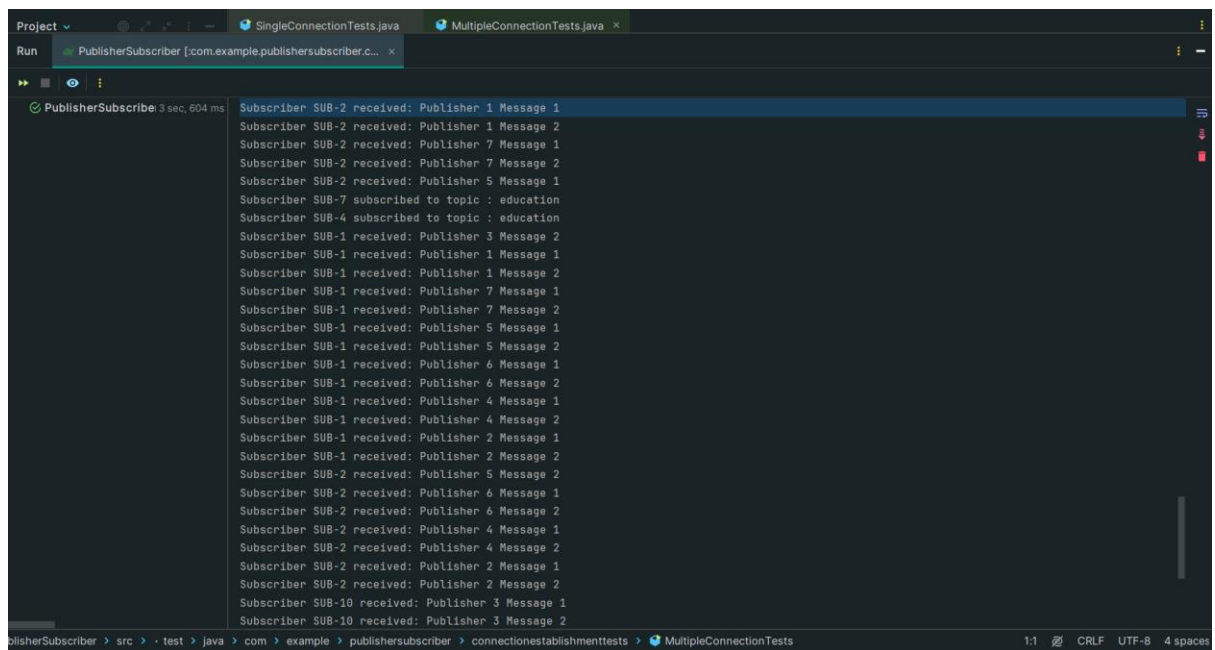
1:1 CRLF UTF-8 4 spaces

The screenshot shows an IDE with two tabs: `SingleConnectionTests.java` and `MultipleConnectionTests.java`. The `Run` button is highlighted, and the output console shows a successful test run for `PublisherSubscriber` with a duration of 3 seconds and 604 milliseconds. The output log contains 30 lines of messages, each indicating a subscriber (SUB-3, SUB-2, SUB-8, SUB-6, SUB-10, SUB-9) receiving a message from a publisher (Publisher 6, 4, 2, 8, 3, 1, 7, 5) with a specific message number (1 or 2). The messages are distributed among the subscribers, with SUB-3 receiving 6 messages, SUB-2 receiving 2, SUB-8 receiving 2, SUB-6 receiving 6, SUB-10 receiving 1, and SUB-9 receiving 11.

```
Project ▾ SingleConnectionTests.java MultipleConnectionTests.java
Run PublisherSubscriber [com.example.publishersubscriber.c...
PublisherSubscriber 3 sec, 604 ms
Subscriber SUB-3 received: Publisher 6 Message 1
Subscriber SUB-3 received: Publisher 6 Message 2
Subscriber SUB-3 received: Publisher 4 Message 1
Subscriber SUB-3 received: Publisher 4 Message 2
Subscriber SUB-3 received: Publisher 2 Message 1
Subscriber SUB-3 received: Publisher 2 Message 2
Subscriber SUB-2 received: Publisher 8 Message 2
Subscriber SUB-8 received: Publisher 4 Message 2
Subscriber SUB-8 received: Publisher 2 Message 1
Subscriber SUB-8 received: Publisher 2 Message 2
Subscriber SUB-6 received: Publisher 8 Message 1
Subscriber SUB-6 received: Publisher 8 Message 2
Subscriber SUB-6 received: Publisher 3 Message 1
Subscriber SUB-6 received: Publisher 3 Message 2
Subscriber SUB-6 received: Publisher 1 Message 1
Subscriber SUB-6 received: Publisher 1 Message 2
Subscriber SUB-10 received: Publisher 8 Message 1
Subscriber SUB-9 received: Publisher 1 Message 1
Subscriber SUB-9 received: Publisher 1 Message 2
Subscriber SUB-9 received: Publisher 7 Message 1
Subscriber SUB-9 received: Publisher 7 Message 2
Subscriber SUB-9 received: Publisher 5 Message 1
Subscriber SUB-9 received: Publisher 5 Message 2
Subscriber SUB-9 received: Publisher 6 Message 1
Subscriber SUB-9 received: Publisher 6 Message 2
Subscriber SUB-9 received: Publisher 4 Message 1
Subscriber SUB-9 received: Publisher 4 Message 2
Subscriber SUB-9 received: Publisher 2 Message 1
Subscriber SUB-9 received: Publisher 2 Message 2
```

The screenshot shows the same IDE as above, but with a different set of messages in the output console. The test run for `PublisherSubscriber` is still successful, with a duration of 3 seconds and 604 milliseconds. The output log contains 30 lines of messages, each indicating a subscriber (SUB-9, SUB-5, SUB-10, SUB-6, SUB-2) receiving a message from a publisher (Publisher 2, 3, 1, 7, 5, 4, 6) with a specific message number (1 or 2). The messages are distributed among the subscribers, with SUB-9 receiving 11 messages, SUB-5 receiving 11, SUB-10 receiving 1, SUB-6 receiving 10, and SUB-2 receiving 7.

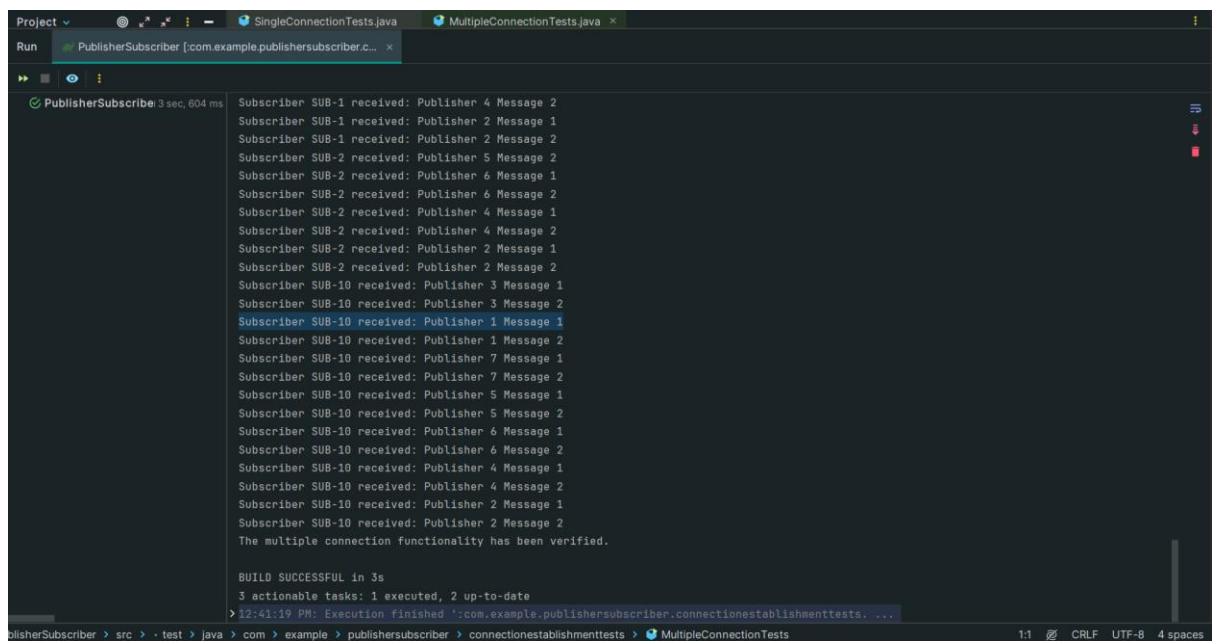
```
Project ▾ SingleConnectionTests.java MultipleConnectionTests.java
Run PublisherSubscriber [com.example.publishersubscriber.c...
PublisherSubscriber 3 sec, 604 ms
Subscriber SUB-9 received: Publisher 2 Message 2
Subscriber SUB-5 received: Publisher 3 Message 2
Subscriber SUB-5 received: Publisher 1 Message 1
Subscriber SUB-5 received: Publisher 1 Message 2
Subscriber SUB-5 received: Publisher 7 Message 1
Subscriber SUB-5 received: Publisher 7 Message 2
Subscriber SUB-5 received: Publisher 5 Message 1
Subscriber SUB-5 received: Publisher 5 Message 2
Subscriber SUB-5 received: Publisher 6 Message 1
Subscriber SUB-5 received: Publisher 6 Message 2
Subscriber SUB-5 received: Publisher 4 Message 1
Subscriber SUB-5 received: Publisher 4 Message 2
Subscriber SUB-5 received: Publisher 2 Message 1
Subscriber SUB-5 received: Publisher 2 Message 2
Subscriber SUB-10 received: Publisher 8 Message 2
Subscriber SUB-6 received: Publisher 7 Message 1
Subscriber SUB-6 received: Publisher 7 Message 2
Subscriber SUB-6 received: Publisher 5 Message 1
Subscriber SUB-6 received: Publisher 5 Message 2
Subscriber SUB-6 received: Publisher 6 Message 1
Subscriber SUB-6 received: Publisher 6 Message 2
Subscriber SUB-6 received: Publisher 4 Message 1
Subscriber SUB-6 received: Publisher 4 Message 2
Subscriber SUB-6 received: Publisher 2 Message 1
Subscriber SUB-6 received: Publisher 2 Message 2
Subscriber SUB-2 received: Publisher 3 Message 1
Subscriber SUB-2 received: Publisher 3 Message 2
Subscriber SUB-2 received: Publisher 1 Message 1
Subscriber SUB-2 received: Publisher 1 Message 2
```



The screenshot shows an IDE window with a project named 'PublisherSubscriber'. The 'Run' tab is active, displaying a log of messages received by subscribers. The log includes the following entries:

```
Subscriber SUB-2 received: Publisher 1 Message 1
Subscriber SUB-2 received: Publisher 1 Message 2
Subscriber SUB-2 received: Publisher 7 Message 1
Subscriber SUB-2 received: Publisher 7 Message 2
Subscriber SUB-2 received: Publisher 5 Message 1
Subscriber SUB-7 subscribed to topic : education
Subscriber SUB-4 subscribed to topic : education
Subscriber SUB-1 received: Publisher 3 Message 2
Subscriber SUB-1 received: Publisher 1 Message 1
Subscriber SUB-1 received: Publisher 1 Message 2
Subscriber SUB-1 received: Publisher 7 Message 1
Subscriber SUB-1 received: Publisher 7 Message 2
Subscriber SUB-1 received: Publisher 5 Message 1
Subscriber SUB-1 received: Publisher 5 Message 2
Subscriber SUB-1 received: Publisher 6 Message 1
Subscriber SUB-1 received: Publisher 6 Message 2
Subscriber SUB-1 received: Publisher 4 Message 1
Subscriber SUB-1 received: Publisher 4 Message 2
Subscriber SUB-1 received: Publisher 2 Message 1
Subscriber SUB-1 received: Publisher 2 Message 2
Subscriber SUB-2 received: Publisher 5 Message 2
Subscriber SUB-2 received: Publisher 6 Message 1
Subscriber SUB-2 received: Publisher 6 Message 2
Subscriber SUB-2 received: Publisher 4 Message 1
Subscriber SUB-2 received: Publisher 4 Message 2
Subscriber SUB-2 received: Publisher 2 Message 1
Subscriber SUB-2 received: Publisher 2 Message 2
Subscriber SUB-10 received: Publisher 3 Message 1
Subscriber SUB-10 received: Publisher 3 Message 2
```

The IDE interface shows the project structure on the left, with the 'Run' tab selected. The status bar at the bottom indicates the file encoding is UTF-8 and the line ending is CRLF.



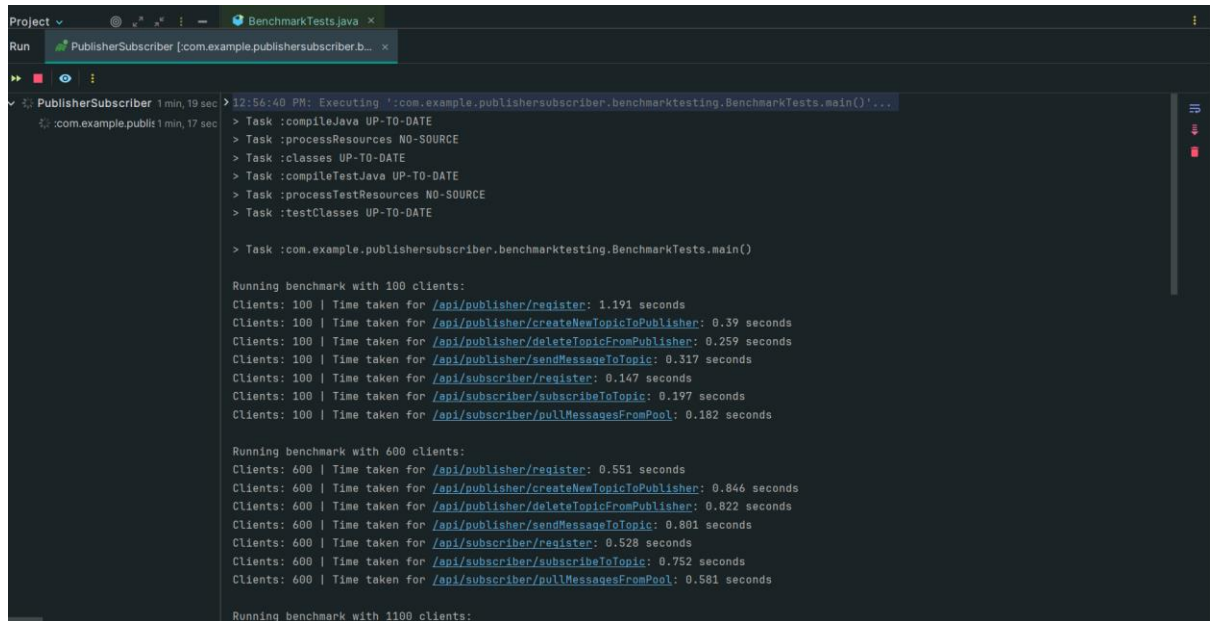
The screenshot shows the same IDE window, but the 'Run' tab now displays a log of messages received by subscribers, followed by a successful build and execution completion message. The log includes the following entries:

```
Subscriber SUB-1 received: Publisher 4 Message 2
Subscriber SUB-1 received: Publisher 2 Message 1
Subscriber SUB-1 received: Publisher 2 Message 2
Subscriber SUB-2 received: Publisher 5 Message 2
Subscriber SUB-2 received: Publisher 6 Message 1
Subscriber SUB-2 received: Publisher 6 Message 2
Subscriber SUB-2 received: Publisher 4 Message 1
Subscriber SUB-2 received: Publisher 4 Message 2
Subscriber SUB-2 received: Publisher 2 Message 1
Subscriber SUB-2 received: Publisher 2 Message 2
Subscriber SUB-10 received: Publisher 3 Message 1
Subscriber SUB-10 received: Publisher 3 Message 2
Subscriber SUB-10 received: Publisher 1 Message 1
Subscriber SUB-10 received: Publisher 1 Message 2
Subscriber SUB-10 received: Publisher 7 Message 1
Subscriber SUB-10 received: Publisher 7 Message 2
Subscriber SUB-10 received: Publisher 5 Message 1
Subscriber SUB-10 received: Publisher 5 Message 2
Subscriber SUB-10 received: Publisher 6 Message 1
Subscriber SUB-10 received: Publisher 6 Message 2
Subscriber SUB-10 received: Publisher 4 Message 1
Subscriber SUB-10 received: Publisher 4 Message 2
Subscriber SUB-10 received: Publisher 2 Message 1
Subscriber SUB-10 received: Publisher 2 Message 2
The multiple connection functionality has been verified.

BUILD SUCCESSFUL in 3s
3 actionable tasks: 1 executed, 2 up-to-date
12:41:19 PM: Execution finished 'com.example.publishersubscriber.connectionestablishmenttests. ...'
```

The IDE interface shows the project structure on the left, with the 'Run' tab selected. The status bar at the bottom indicates the file encoding is UTF-8 and the line ending is CRLF.

Benchmark Testing Results



The screenshot shows an IDE window with a project named 'PublisherSubscriber'. The 'Run' tab is active, displaying the output of a benchmark test. The test is titled 'BenchmarkTests.main()' and is executed at 12:56:40 PM. The output shows the execution of various tasks, including compiling Java, processing resources, and testing classes. The benchmark results are presented in three sections, one for each client count (100, 600, and 1100). Each section lists the time taken for various API endpoints, such as /api/publisher/register, /api/publisher/createNewTopicToPublisher, /api/publisher/deleteTopicFromPublisher, /api/publisher/sendMessageToTopic, /api/subscriber/register, /api/subscriber/subscribeToTopic, and /api/subscriber/pullMessagesFromPool. The times are measured in seconds and are generally higher for 600 and 1100 clients compared to 100 clients.

```
Project v BenchmarkTests.java x
Run PublisherSubscriber [com.example.publishersubscriber.b... x

12:56:40 PM: Executing ".com.example.publishersubscriber.benchmarktesting.BenchmarkTests.main()"
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE

> Task :com.example.publishersubscriber.benchmarktesting.BenchmarkTests.main()

Running benchmark with 100 clients:
Clients: 100 | Time taken for /api/publisher/register: 1.191 seconds
Clients: 100 | Time taken for /api/publisher/createNewTopicToPublisher: 0.39 seconds
Clients: 100 | Time taken for /api/publisher/deleteTopicFromPublisher: 0.259 seconds
Clients: 100 | Time taken for /api/publisher/sendMessageToTopic: 0.317 seconds
Clients: 100 | Time taken for /api/subscriber/register: 0.147 seconds
Clients: 100 | Time taken for /api/subscriber/subscribeToTopic: 0.197 seconds
Clients: 100 | Time taken for /api/subscriber/pullMessagesFromPool: 0.182 seconds

Running benchmark with 600 clients:
Clients: 600 | Time taken for /api/publisher/register: 0.551 seconds
Clients: 600 | Time taken for /api/publisher/createNewTopicToPublisher: 0.846 seconds
Clients: 600 | Time taken for /api/publisher/deleteTopicFromPublisher: 0.822 seconds
Clients: 600 | Time taken for /api/publisher/sendMessageToTopic: 0.801 seconds
Clients: 600 | Time taken for /api/subscriber/register: 0.528 seconds
Clients: 600 | Time taken for /api/subscriber/subscribeToTopic: 0.752 seconds
Clients: 600 | Time taken for /api/subscriber/pullMessagesFromPool: 0.581 seconds

Running benchmark with 1100 clients:
```

```

Project: Alt+1
BenchmarkTests.java x
Run: PublisherSubscriber [com.example.publishersubscriber.b... x

Running benchmark with 1100 clients:
Clients: 1100 | Time taken for /api/publisher/register: 0.87 seconds
Clients: 1100 | Time taken for /api/publisher/createNewTopicToPublisher: 0.882 seconds
Clients: 1100 | Time taken for /api/publisher/deleteTopicFromPublisher: 0.971 seconds
Clients: 1100 | Time taken for /api/publisher/sendMessageToTopic: 0.961 seconds
Clients: 1100 | Time taken for /api/subscriber/register: 0.973 seconds
Clients: 1100 | Time taken for /api/subscriber/subscribeToTopic: 0.973 seconds
Clients: 1100 | Time taken for /api/subscriber/pullMessagesFromPool: 1.402 seconds

Running benchmark with 1600 clients:
Clients: 1600 | Time taken for /api/publisher/register: 1.329 seconds
Clients: 1600 | Time taken for /api/publisher/createNewTopicToPublisher: 1.348 seconds
Clients: 1600 | Time taken for /api/publisher/deleteTopicFromPublisher: 1.188 seconds
Clients: 1600 | Time taken for /api/publisher/sendMessageToTopic: 1.017 seconds
Clients: 1600 | Time taken for /api/subscriber/register: 1.159 seconds
Clients: 1600 | Time taken for /api/subscriber/subscribeToTopic: 1.156 seconds
Clients: 1600 | Time taken for /api/subscriber/pullMessagesFromPool: 1.059 seconds

Running benchmark with 2100 clients:
Clients: 2100 | Time taken for /api/publisher/register: 1.12 seconds
Clients: 2100 | Time taken for /api/publisher/createNewTopicToPublisher: 1.826 seconds
Clients: 2100 | Time taken for /api/publisher/deleteTopicFromPublisher: 1.427 seconds
Clients: 2100 | Time taken for /api/publisher/sendMessageToTopic: 1.699 seconds
Clients: 2100 | Time taken for /api/subscriber/register: 1.434 seconds
Clients: 2100 | Time taken for /api/subscriber/subscribeToTopic: 1.895 seconds
Clients: 2100 | Time taken for /api/subscriber/pullMessagesFromPool: 1.856 seconds

Running benchmark with 2600 clients:

```

```

Project: BenchmarkTests.java x
Run: PublisherSubscriber [com.example.publishersubscriber.b... x

Running benchmark with 2600 clients:
Clients: 2600 | Time taken for /api/publisher/register: 2.661 seconds
Clients: 2600 | Time taken for /api/publisher/createNewTopicToPublisher: 2.114 seconds
Clients: 2600 | Time taken for /api/publisher/deleteTopicFromPublisher: 2.007 seconds
Clients: 2600 | Time taken for /api/publisher/sendMessageToTopic: 1.982 seconds
Clients: 2600 | Time taken for /api/subscriber/register: 2.138 seconds
Clients: 2600 | Time taken for /api/subscriber/subscribeToTopic: 2.136 seconds
Clients: 2600 | Time taken for /api/subscriber/pullMessagesFromPool: 1.896 seconds

Running benchmark with 3100 clients:
Clients: 3100 | Time taken for /api/publisher/register: 2.291 seconds
Clients: 3100 | Time taken for /api/publisher/createNewTopicToPublisher: 2.844 seconds
Clients: 3100 | Time taken for /api/publisher/deleteTopicFromPublisher: 2.802 seconds
Clients: 3100 | Time taken for /api/publisher/sendMessageToTopic: 3.068 seconds
Clients: 3100 | Time taken for /api/subscriber/register: 2.753 seconds
Clients: 3100 | Time taken for /api/subscriber/subscribeToTopic: 2.63 seconds
Clients: 3100 | Time taken for /api/subscriber/pullMessagesFromPool: 3.783 seconds

Running benchmark with 3600 clients:
Clients: 3600 | Time taken for /api/publisher/register: 3.757 seconds
Clients: 3600 | Time taken for /api/publisher/createNewTopicToPublisher: 4.08 seconds
Clients: 3600 | Time taken for /api/publisher/deleteTopicFromPublisher: 3.05 seconds
Clients: 3600 | Time taken for /api/publisher/sendMessageToTopic: 3.595 seconds
Clients: 3600 | Time taken for /api/subscriber/register: 6.988 seconds
Clients: 3600 | Time taken for /api/subscriber/subscribeToTopic: 5.28 seconds
Clients: 3600 | Time taken for /api/subscriber/pullMessagesFromPool: 3.835 seconds

Running benchmark with 4100 clients:
Clients: 4100 | Time taken for /api/publisher/register: 3.427 seconds

```



```

Project ▾  BenchmarkTests.java x
Run PublisherSubscriber [com.example.publishersubscriber.b... x

PublisherSub 3 min, 55 sec, 604 ms
Running benchmark with 4100 clients:
Clients: 4100 | Time taken for /api/publisher/register: 3.427 seconds
Clients: 4100 | Time taken for /api/publisher/createNewTopicToPublisher: 4.166 seconds
Clients: 4100 | Time taken for /api/publisher/deleteTopicFromPublisher: 4.561 seconds
Clients: 4100 | Time taken for /api/publisher/sendMessageToTopic: 5.249 seconds
Clients: 4100 | Time taken for /api/subscriber/register: 3.641 seconds
Clients: 4100 | Time taken for /api/subscriber/subscribeToTopic: 3.935 seconds
Clients: 4100 | Time taken for /api/subscriber/pullMessagesFromPool: 4.159 seconds

Running benchmark with 4600 clients:
Clients: 4600 | Time taken for /api/publisher/register: 3.364 seconds
Clients: 4600 | Time taken for /api/publisher/createNewTopicToPublisher: 4.229 seconds
Clients: 4600 | Time taken for /api/publisher/deleteTopicFromPublisher: 4.103 seconds
Clients: 4600 | Time taken for /api/publisher/sendMessageToTopic: 4.755 seconds
Clients: 4600 | Time taken for /api/subscriber/register: 2.868 seconds
Clients: 4600 | Time taken for /api/subscriber/subscribeToTopic: 3.893 seconds
Clients: 4600 | Time taken for /api/subscriber/pullMessagesFromPool: 4.475 seconds

Running benchmark with 5100 clients:
Clients: 5100 | Time taken for /api/publisher/register: 4.891 seconds
Clients: 5100 | Time taken for /api/publisher/createNewTopicToPublisher: 6.505 seconds
Clients: 5100 | Time taken for /api/publisher/deleteTopicFromPublisher: 5.798 seconds
Clients: 5100 | Time taken for /api/publisher/sendMessageToTopic: 4.99 seconds
Clients: 5100 | Time taken for /api/subscriber/register: 4.531 seconds
Clients: 5100 | Time taken for /api/subscriber/subscribeToTopic: 5.171 seconds
Clients: 5100 | Time taken for /api/subscriber/pullMessagesFromPool: 5.365 seconds

Running benchmark with 5600 clients:
Clients: 5600 | Time taken for /api/publisher/register: 5.212 seconds

```

PublisherSubscriber > src > .test > java > com > example > publishersubscriber > benchmarktesting > BenchmarkTests 15:14 CRLF UTF-8 4 spaces

```

Project ▾  BenchmarkTests.java x
Run PublisherSubscriber [com.example.publishersubscriber.b... x

PublisherSub 3 min, 55 sec, 604 ms
Running benchmark with 5600 clients:
Clients: 5600 | Time taken for /api/publisher/register: 5.212 seconds
Clients: 5600 | Time taken for /api/publisher/createNewTopicToPublisher: 5.328 seconds
Clients: 5600 | Time taken for /api/publisher/deleteTopicFromPublisher: 5.565 seconds
Clients: 5600 | Time taken for /api/publisher/sendMessageToTopic: 5.212 seconds
Clients: 5600 | Time taken for /api/subscriber/register: 5.14 seconds
Clients: 5600 | Time taken for /api/subscriber/subscribeToTopic: 5.054 seconds
Clients: 5600 | Time taken for /api/subscriber/pullMessagesFromPool: 4.864 seconds

Running benchmark with 6100 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

Running benchmark with 6600 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

Running benchmark with 7100 clients:

```

```

Project ▾ | BenchmarkTests.java x
Run | PublisherSubscriber [com.example.publishersubscriber.b... x

PublisherSub 3 min, 55 sec, 604 ms
Running benchmark with 7100 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

Running benchmark with 7600 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

Running benchmark with 8100 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

Running benchmark with 8600 clients:

```

```

Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

Running benchmark with 9100 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

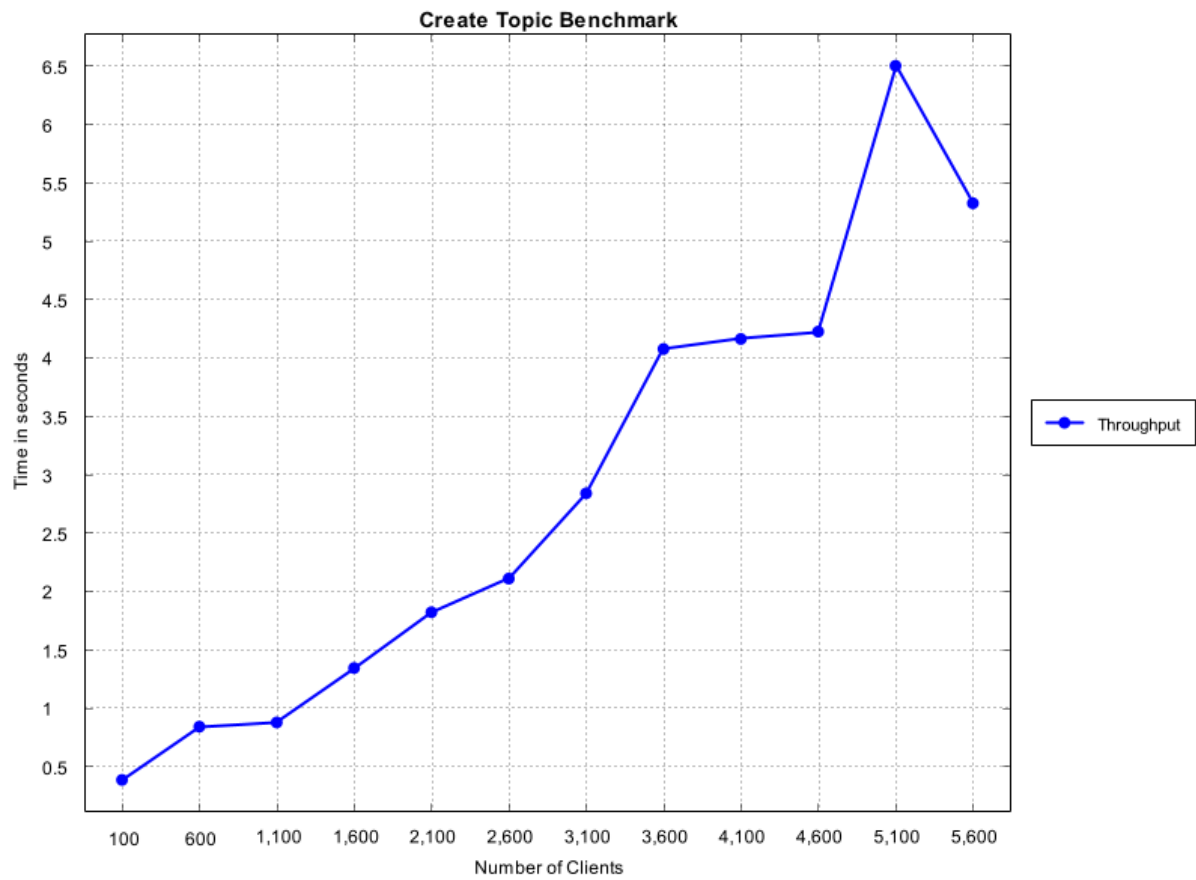
Running benchmark with 9600 clients:
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.
Server refused to connect after 6000 clients.

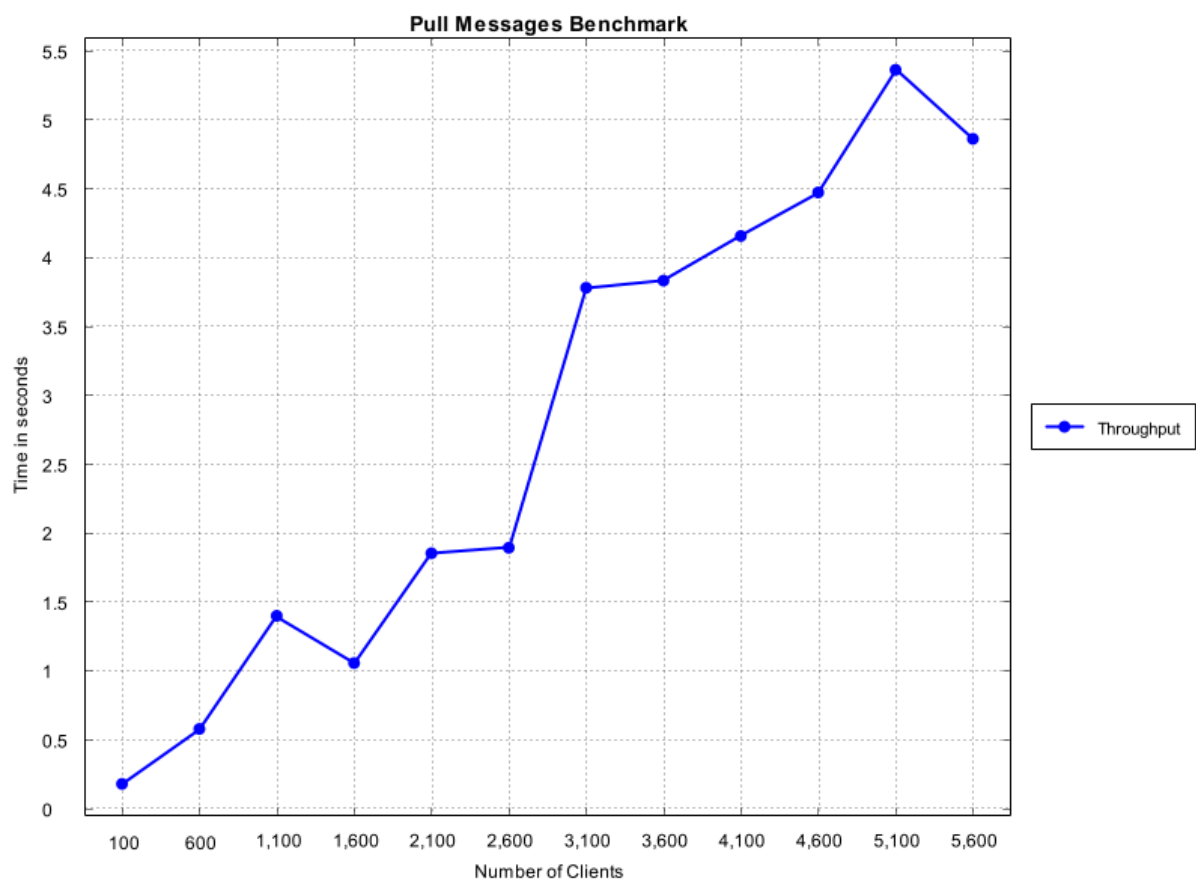
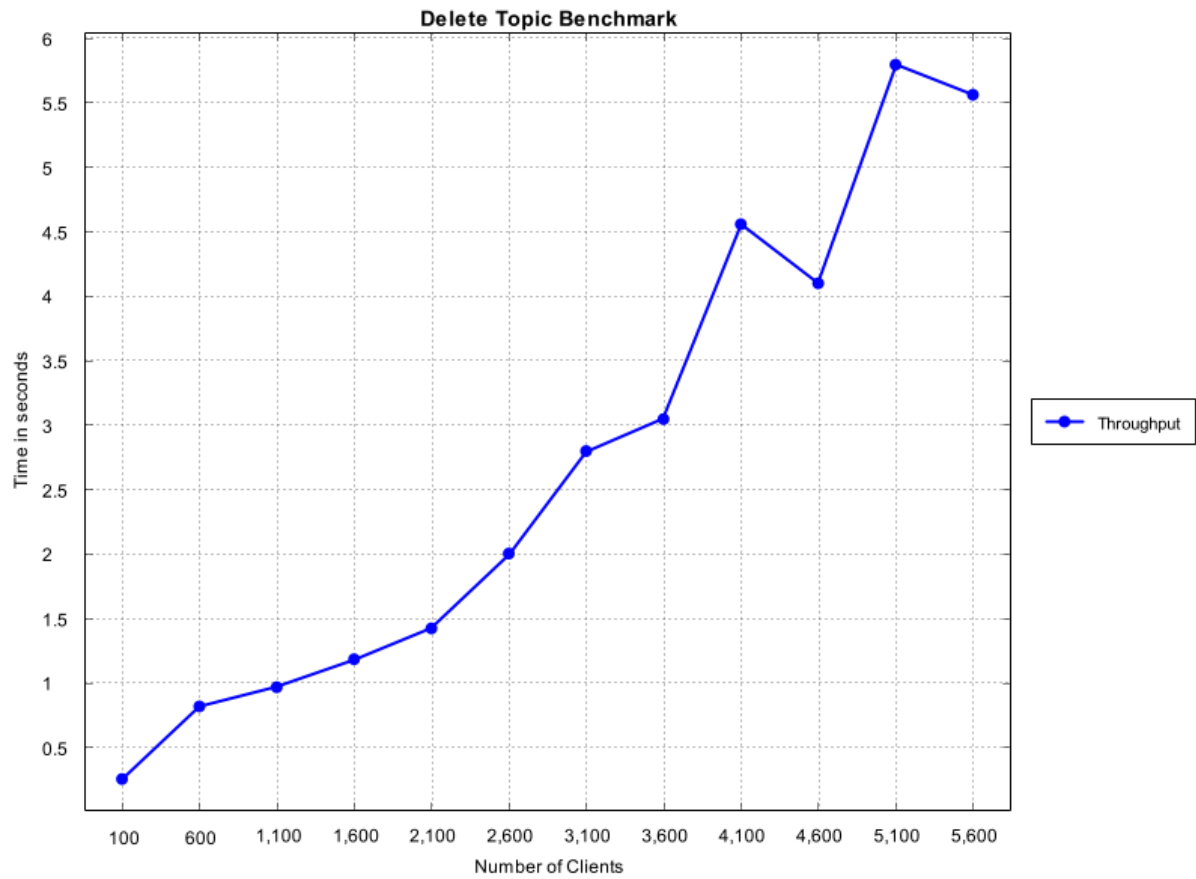
BUILD SUCCESSFUL in 3m 54s
3 actionable tasks: 1 executed, 2 up-to-date
> 1:00:36 PM: Execution finished 'com.example.publishersubscriber.benchmarktesting.BenchmarkTest ...

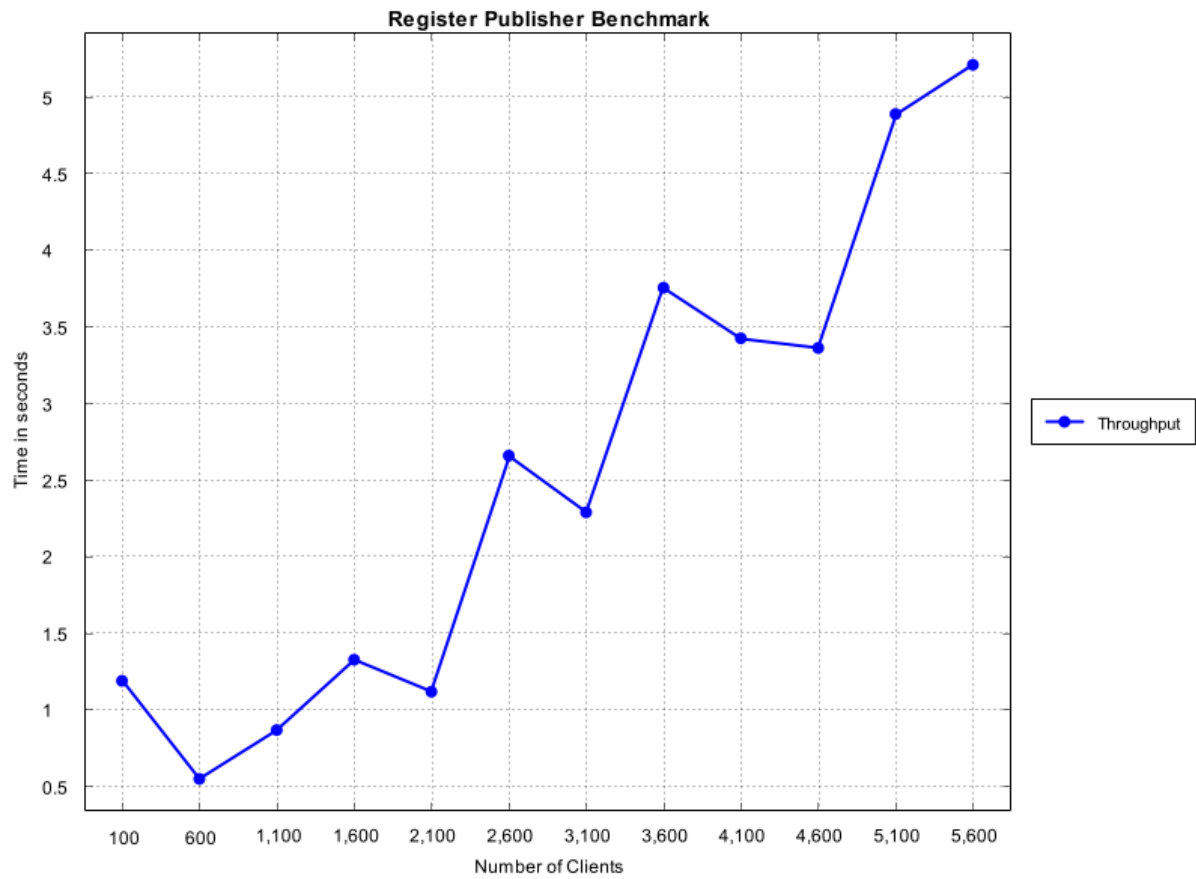
```

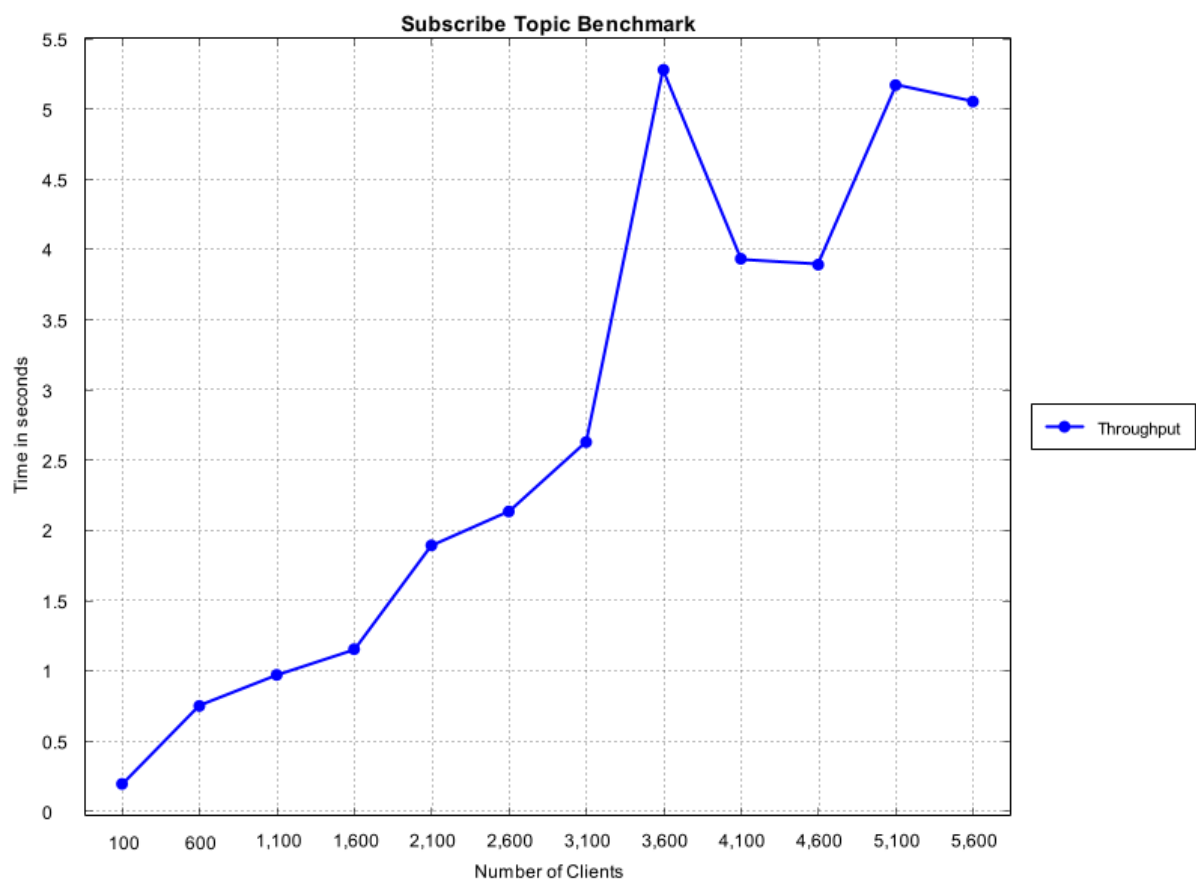
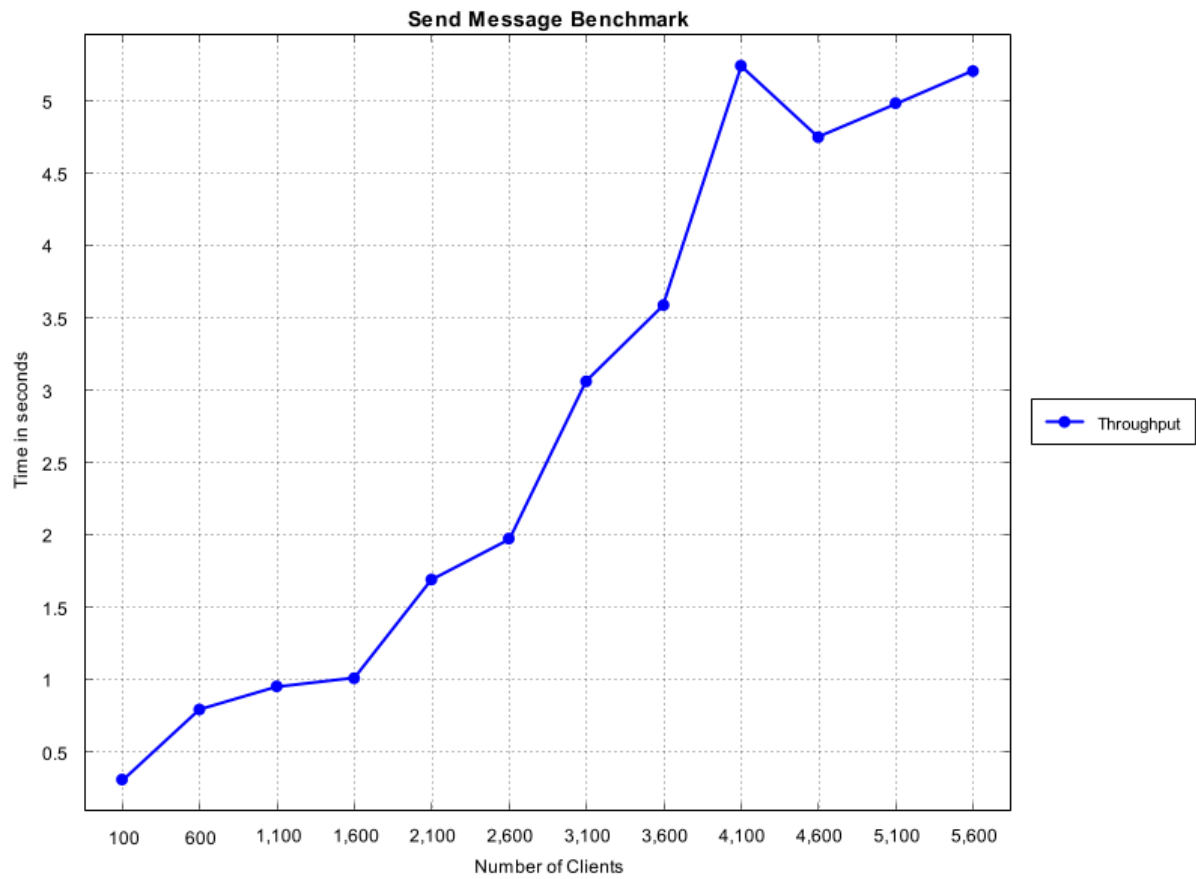
This saves the values to csv files in root folder and the Plot.java file, when run, plots a graph for each API plotting its benchmark results between time in seconds and the number of clients.

Screenshot of each benchmarked graph is as follows:







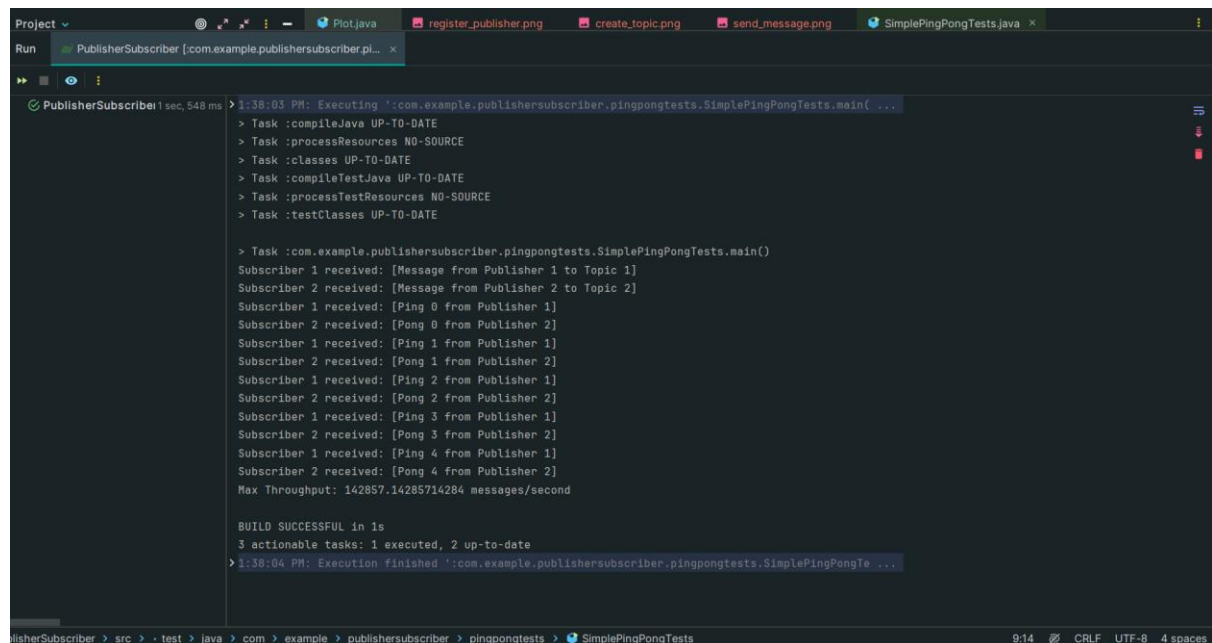


Ping Pong Testing

Ping Pong testing in a publisher-subscriber system is used to evaluate the round-trip communication efficiency between the publisher and subscriber. It involves sending a message from the publisher to the subscriber and having the subscriber respond back to the publisher, simulating a ping-pong-like exchange of messages. The purpose of this test is to measure latency, throughput, and the overall performance of the message broker, ensuring that messages are delivered and acknowledged in a timely manner. This helps assess how well the system can handle continuous back-and-forth communication under various loads, ensuring low-latency message exchange, which is critical in real-time applications.

Ping Pong testing in this context is applicable to all 7 APIs of the publisher-subscriber system. It assesses the round-trip communication between the client (either publisher or subscriber) and the server for each API operation, such as registering a publisher, creating or deleting a topic, sending messages, registering a subscriber, subscribing to topics, and pulling messages. The goal is to measure the time taken for a request to travel from the client to the server and back, ensuring that all components, including the message broker and RESTful endpoints, function efficiently under load. This helps determine the overall latency and throughput of the system for different operations.

Single Ping Pong Testing:



```

Project ▾
Run PublisherSubscriber [com.example.publishersubscriber.pi... x
> 1:38:03 PM: Executing 'com.example.publishersubscriber.pingpongtests.SimplePingPongTests.main( ...
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava UP-TO-DATE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE

> Task :com.example.publishersubscriber.pingpongtests.SimplePingPongTests.main()
Subscriber 1 received: [Message from Publisher 1 to Topic 1]
Subscriber 2 received: [Message from Publisher 2 to Topic 2]
Subscriber 1 received: [Ping 0 from Publisher 1]
Subscriber 2 received: [Pong 0 from Publisher 2]
Subscriber 1 received: [Ping 1 from Publisher 1]
Subscriber 2 received: [Pong 1 from Publisher 2]
Subscriber 1 received: [Ping 2 from Publisher 1]
Subscriber 2 received: [Pong 2 from Publisher 2]
Subscriber 1 received: [Ping 3 from Publisher 1]
Subscriber 2 received: [Pong 3 from Publisher 2]
Subscriber 1 received: [Ping 4 from Publisher 1]
Subscriber 2 received: [Pong 4 from Publisher 2]
Max Throughput: 142857.14285714284 messages/second

BUILD SUCCESSFUL in 1s
3 actionable tasks: 1 executed, 2 up-to-date
> 1:38:04 PM: Execution finished 'com.example.publishersubscriber.pingpongtests.SimplePingPongTe ...
PublisherSubscriber > src > . > test > java > com > example > publishersubscriber > pingpongtests > SimplePingPongTests 9:14 CRLF UTF-8 4 spaces

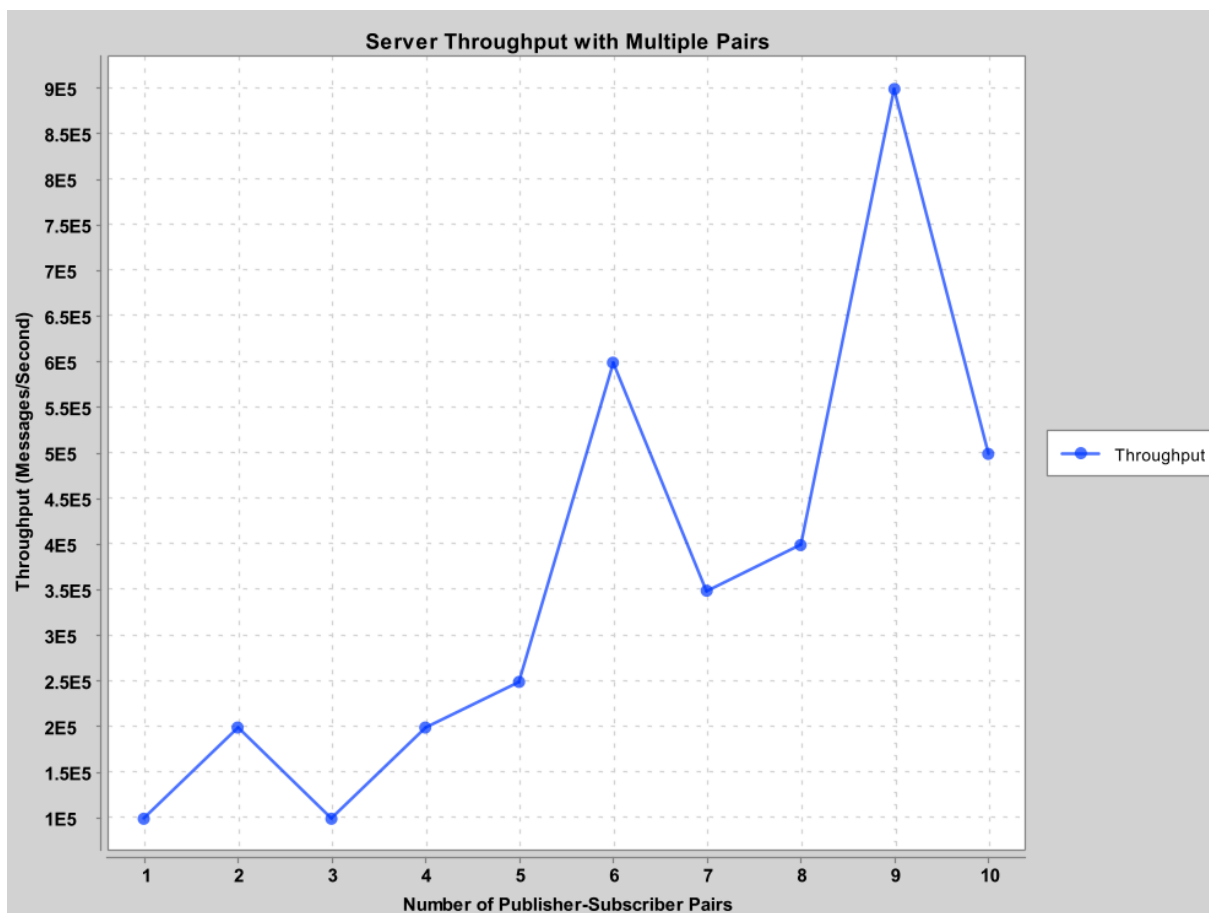
```

Multiple Ping Pong Test interaction:

```
Run PublisherSubscriber [com.example.publishersubscriber.pl... x
> Task :compileJava UP-TO-DATE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :compileTestJava
> Task :processTestResources NO-SOURCE
> Task :testClasses

> Task :com.example.publishersubscriber.pingpongtests.PingPongThroughputTests.main()
Starting ping-pong test with 1 pairs...
Throughput with 1 pairs: 100000.0 messages/second
Starting ping-pong test with 2 pairs...
Throughput with 2 pairs: 200000.0 messages/second
Starting ping-pong test with 3 pairs...
Throughput with 3 pairs: 100000.0 messages/second
Starting ping-pong test with 4 pairs...
Throughput with 4 pairs: 200000.0 messages/second
Starting ping-pong test with 5 pairs...
Throughput with 5 pairs: 250000.0 messages/second
Starting ping-pong test with 6 pairs...
Throughput with 6 pairs: 600000.0 messages/second
Starting ping-pong test with 7 pairs...
Throughput with 7 pairs: 350000.0 messages/second
Starting ping-pong test with 8 pairs...
Throughput with 8 pairs: 400000.0 messages/second
Starting ping-pong test with 9 pairs...
Throughput with 9 pairs: 900000.0 messages/second
Starting ping-pong test with 10 pairs...
Throughput with 10 pairs: 500000.0 messages/second
```

Plotting the observation:



NOTE:

For all the throughput testing, since I have made use of the SpringBoot server, the load it is able to take maxes out even 10000 clients which takes around 15 minutes to run on my system. So, I am just plotting a graph and not choking the connection.

CONCLUSION:

The graph plots for each API in the publisher-subscriber system are likely to show the performance of the system under increasing client load. The key metric being plotted is the time taken (in seconds) on the Y-axis against the number of clients on the X-axis.

Expected Graph Patterns for Each API:

1. Register Publisher API:

- Graph Title: "Register Publisher Benchmark"
- Expected Behaviour: Initially, as the number of clients increases, the time taken should increase slightly but stay relatively low since registering a publisher is a lightweight operation. However, as the number of clients grows, the time taken could rise exponentially due to bottlenecks or resource contention.
- Shape: Slight upward curve, with potential exponential rise after a large number of clients.

2. Create Topic API:

- Graph Title: "Create Topic Benchmark"
- Expected Behaviour: Similar to registering publishers, creating a topic is a lightweight task. The time taken should remain low for a smaller number of clients but will increase as concurrency rises.
- Shape: Gradually increasing curve, with sharper rise at high client counts.

3. Delete Topic API:

- Graph Title: "Delete Topic Benchmark"
- Expected Behaviour: Deleting a topic might take slightly longer than creating one, as it may require updating multiple internal structures. The time will increase with the number of clients, especially at higher loads.
- Shape: Similar to the "Create Topic" graph, but may rise more steeply at higher client counts.

4. Send Message API:

- Graph Title: "Send Message Benchmark"
- Expected Behaviour: Sending messages is expected to show a clear upward trend as the number of clients increases. This API requires interacting with the

message broker, which could cause bottlenecks when handling a large number of concurrent messages.

- Shape: Steeper upward curve compared to registration or topic operations, especially at higher loads.

5. Register Subscriber API:

- Graph Title: "Register Subscriber Benchmark"
- Expected Behaviour: Similar to the "Register Publisher" benchmark, registering a subscriber is relatively fast but could slow down under heavy client loads.
- Shape: Slightly increasing curve, with a possible steep rise after hitting a client threshold.

6. Subscribe Topic API:

- Graph Title: "Subscribe Topic Benchmark"
- Expected Behaviour: Subscribing to a topic involves updating subscription lists, which can take time as the number of topics and subscribers grows. The graph might show a moderate upward curve, especially as the system handles more clients.
- Shape: Gradual increase, potentially sharper at higher client counts.

7. Pull Messages API:

- Graph Title: "Pull Messages Benchmark"
- Expected Behaviour: Pulling messages may show the most significant increase in time taken, as it depends on the number of messages in the pool, the size of the messages, and the number of clients trying to access the pool. Under high loads, the time taken could spike dramatically.
- Shape: A steeper rise compared to the other APIs, with potential spikes under heavy load.

Overall Expected Behaviour:

- ✓ Initial Phase (lower number of clients): All APIs should exhibit relatively low and stable response times.
- ✓ Mid-Range Phase (medium client load): The curves will begin to show a gradual rise in time taken as more clients interact with the system concurrently.
- ✓ High Load Phase (large number of clients): At some point, the response time will increase exponentially as system bottlenecks or server resource limits are hit.

The send message and pull messages operations are likely to show more dramatic increases compared to registration or topic-related operations.