Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Big Data Engineering (DJ19DSL604)

AY: 2022-23

Experiment 3

Bhuvi Ghosh
60009210191

(Messaging Services)

Aim: Implement messaging system using AMPS

Theory:

Communication, in general, is the process of transferring data from a source to a destination by using any of the available modes such as audio, video, text or even signals, etc. This communication may be straightforward between one sender and one receiver or it may include multiple senders and receivers. On the basis of the number of senders and receivers involved, a communication can either be "Point-to-Point" or "Multi-point".

Point-to-Point Communication:

In telecommunications, a point-to-point connection is a communications link between two communication endpoints or nodes. A telephone call is an example of this, in which two phones are linked, and what one caller says can only be heard by the other.

A "point-to-multipoint" or broadcast link, on the other hand, allows multiple nodes to receive information sent by a single node. Leased lines and microwave radio relays are also examples of point-to-point connections.

In a point-to-point communication, there will be a transmitter and a receiver connected together with a suitable connection. The capacity of the connecting channel remains unchanged throughout the communication.

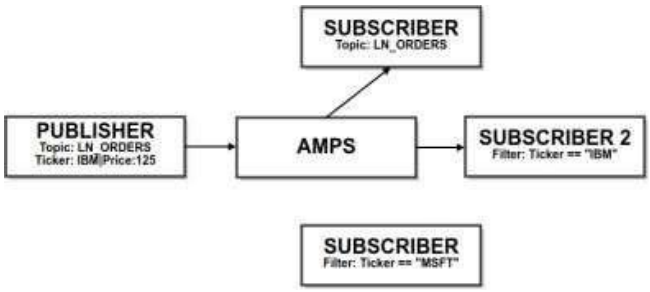
Multi-point Communication

In telecommunications, point-to-multipoint communication (P2MP, PTMP, or PMP) is a form of one-to-many communication that allows numerous routes to be established from a single site to several locations.

The usage of gigahertz radio frequencies for point-to-multipoint telecommunications is common in wireless Internet and IP telephony. P2MP systems have been created with and without a return channel from the numerous receivers. The system employs a kind of time-division multiplexing to enable the return channel traffic, which is transmitted from a central antenna to numerous receiving antennas.

AMPS from 60East Technologies is a fast messaging engine that supports both publish-subscribe messaging and queuing. It is not just a simple messaging engine it is also packed with some powerful feature like high availability, historical replay, aggregation and analytics, content filtering and continuous query, last value caching, focus tracking, and more.

AMPS PUB-SUB Model

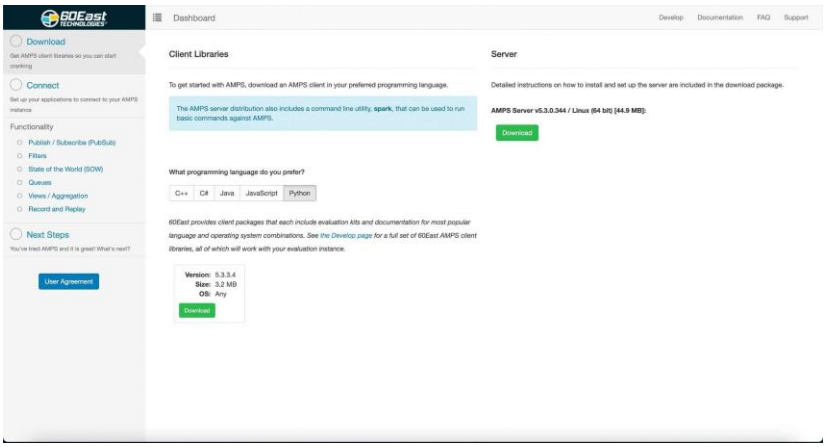


This is same as any other pub-sub model wherein a publisher publishes the message to a topic and subscriber subscribes to a topic and reads the message from it. But amps also provide some powerfully features like we can do **content filtering** where we can subscribe to a topic with a filter and with the help of that we can get only the filtered messages from AMPS server.

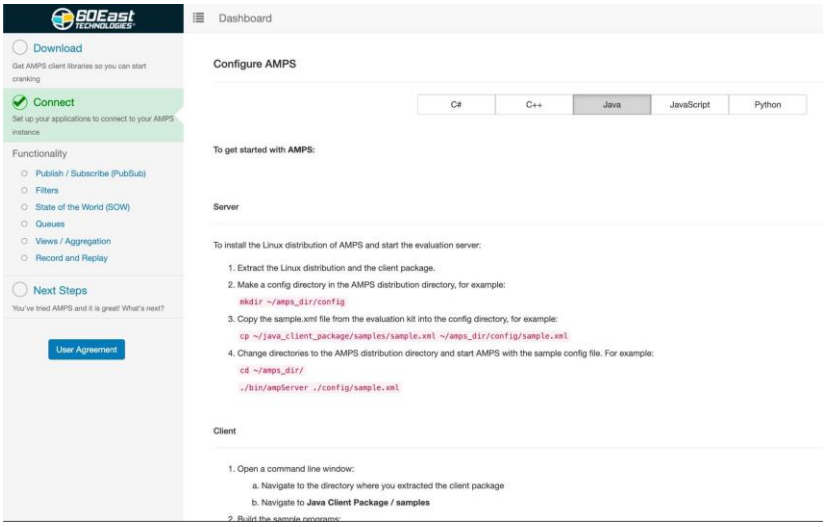
Setting up AMPS:

- Visit https://www.crankuptheamps.com/evaluate/

- Sign up with any email ID
- You'll receive an evaluation kit of AMPS in your email.
- Download the AMPS Server
- Download the Client Libraries (as per programming language preference)

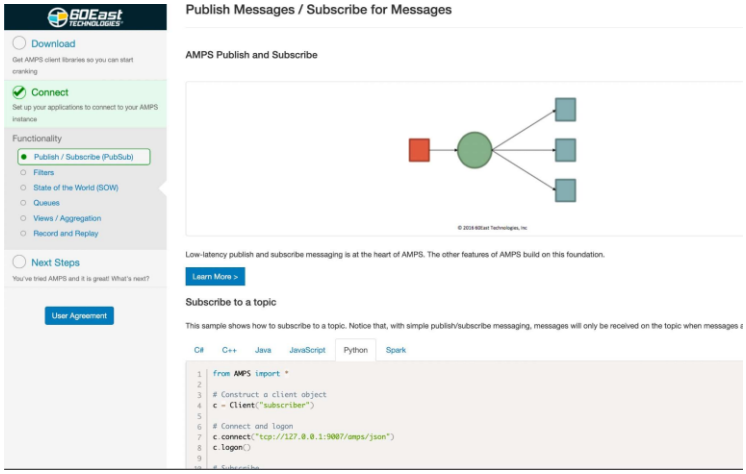


- After downloading the server binaries and client, click on Connect



- On the right side pane, you'll see the steps to configure AMPS Server and Client.
- Once configured, head over to the Functionality tab (below Connect)

- Test your AMPS configuration by publishing / subscribing to a topic.



AMPS is now ready to be explored.

Lab Assignment:

1. Setup AMPS server in the lab and all students will connect to it.
2. Write basic JSON messages (having flat structures), publish, and consume using AMPS.
3. Write nested JSON messages (having 5-6 levels of hierarchy), publish and subscribe using AMPS.
4. Apply Content Filtering, subscribing to relevant messages only.

Subscriber.py:

client.py

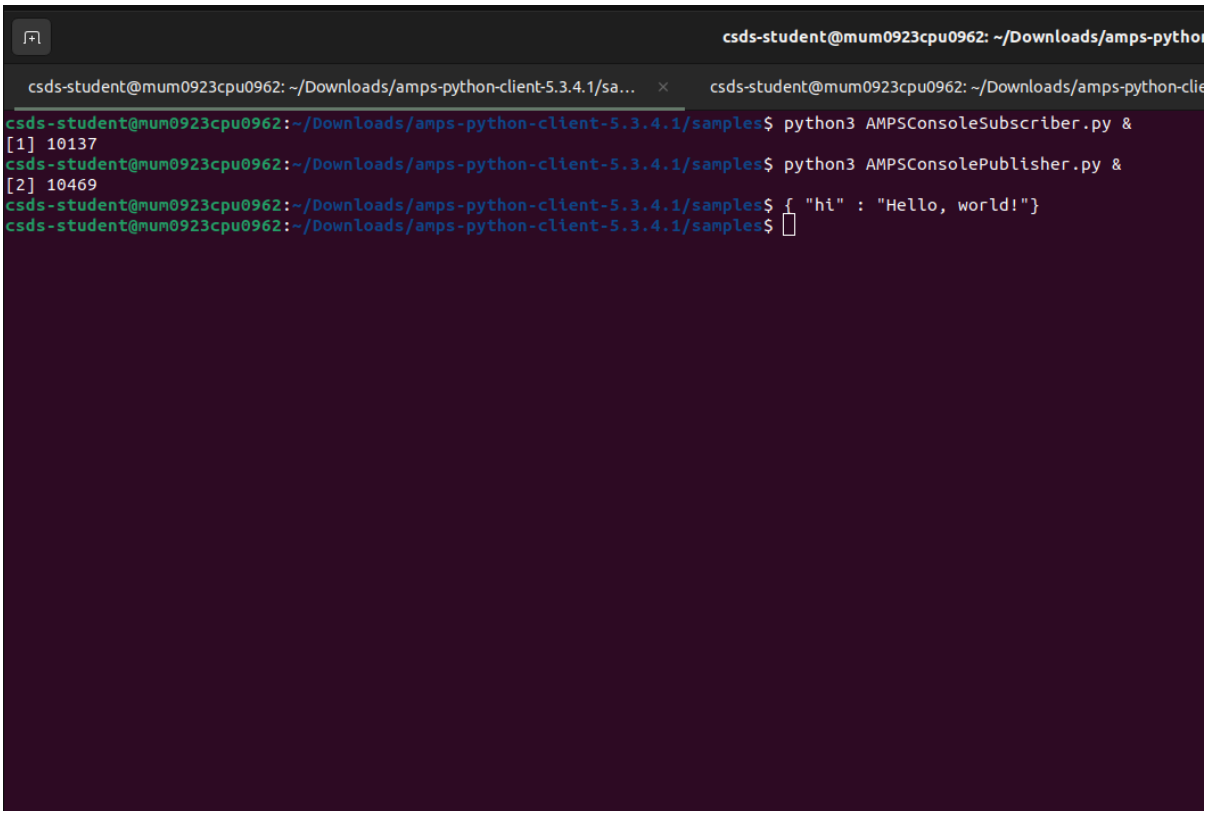
```python
from AMPS import Client

# Construct a client object
c = Client("subscriber")

# Connect and logon
c.connect("tcp://10.120.107.156:9092")
c.logon()

# Subscribe
for m in c.subscribe("test"):
    print ("Received message: %s" % (m.get_data()))
```

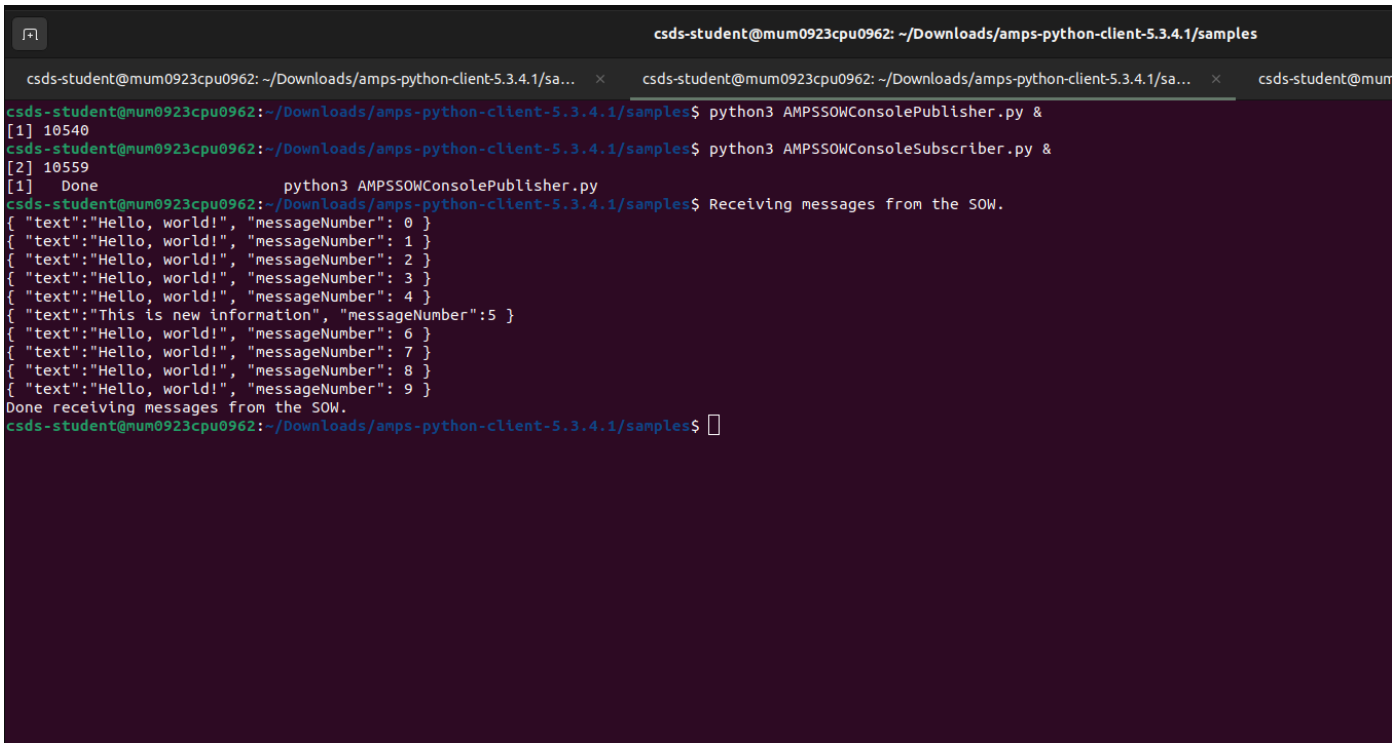
Publisher.py

publish.py

```python
from AMPS import Client

# Construct a client object
c = Client("publisher")

# Connect and logon
c.connect("tcp://10.120.107.156:9092/amps/json")
c.logon()

# Publish
c.publish("test", '{ "message" : "Hello, world!" }')
```

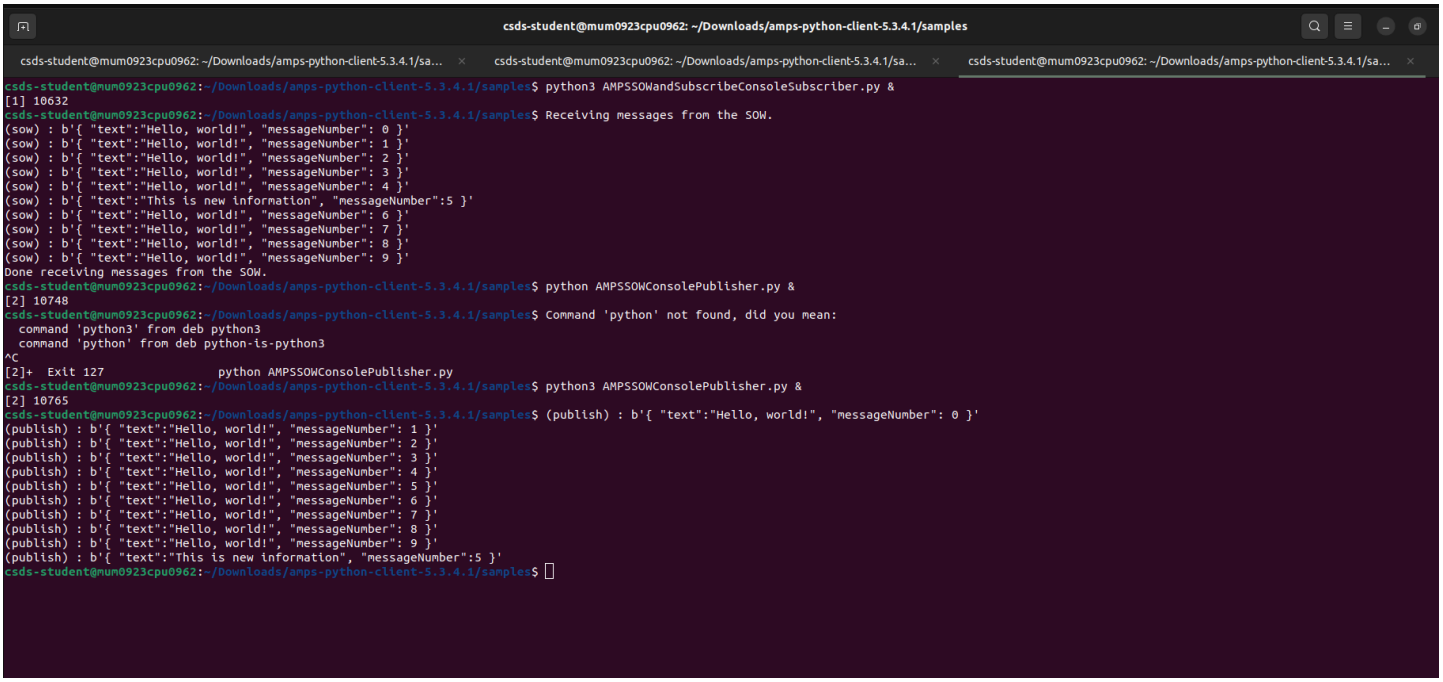
AMPSConsoleSubscriber.py & Publisher.py

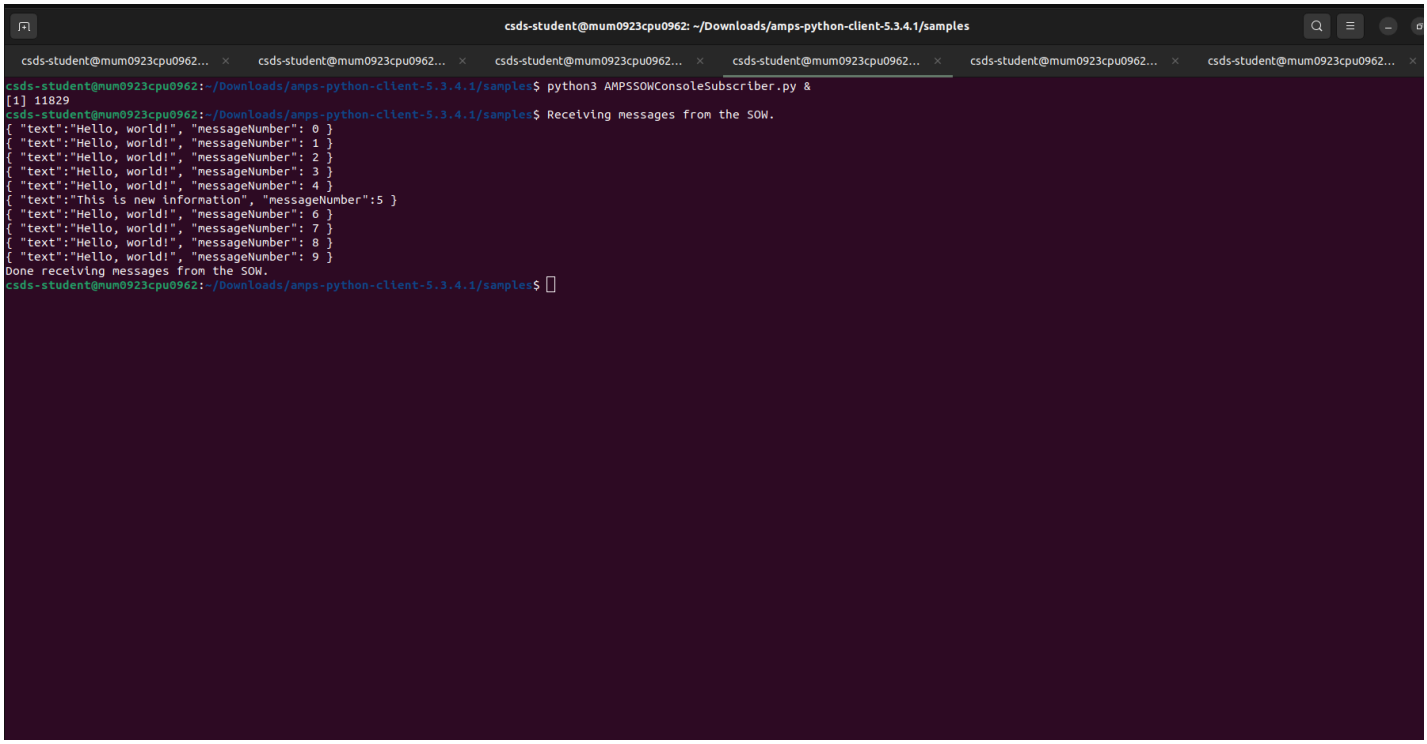




AMPSSOWConsoleSubsciber.py & Publisher.py

AMPSSOWandSubscribeConsoleSubscriber.py & Publisher.py:





Filter.py:

Open

Filter.py
~/Downloads/amps-python-client-5.3.4.1/samples

```python
import socket
import json

class Client:
    def __init__(self, name):
        self.name = name
        self.socket = None

    def connect(self, host, port):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((host, port))

    def logon(self):
        # Implement logon process if needed
        pass

    def subscribe(self, topic, filter):
        # Subscribe command
        subscribe_command = {
            "command": "subscribe",
            "topic": topic,
            "filter": filter
        }
        self.send_message(subscribe_command)

        # Receive messages
        while True:
            data = self.socket.recv(4096)
            if not data:
                break
            messages = json.loads(data.decode())
            for message in messages:
                yield message

    def send_message(self, message):
        serialized_message = json.dumps(message)
        self.socket.sendall(serialized_message.encode())

if __name__ == "__main__":
    # Construct a client object
    c = Client("client-name")

    # Connect to server
    c.connect("127.0.0.1", 9007)

    # Logon (if required)
    c.logon()

    # Subscribe with a filter
    for m in c.subscribe("test", "/details/items/description LIKE 'kitten' "):
        print("Received message for topic: %s : %s" % (m.get("topic"), m.get("data")))
```

```
csds-student@mum0923cpu0962:~/Downloads/amps-python-client-5.3.4.1/samples$ python3 Filter.py
csds-student@mum0923cpu0962:~/Downloads/amps-python-client-5.3.4.1/samples$
```

Apply Content Filtering, subscribing to relevant messages only.