ACADEMY OF ATHENS

HELLENIC REPUBLIC
National and Kapodistrian
University of Athens

Postgraduate Program: «Data Science and Information Technologies»

# Big Data Management

Programming Assignment

**Postgraduate Student:** Gavrielatos Marios

**Registration Number**: 7115152100023

## Introduction

In this following project I implemented a simple Publisher-Subscriber (Pub/Sub) system. The Pub/Sub system consists of three main parts. The Publisher creates messages that correspond to different topics, feed of messages, and publishes these messages. The Broker will receive these messages and it will forward them to the Subscribers that are subscribed to that specific topic. By the term messages we refer to the data that moves through the service. The Subscriber can receive these messages and send messages in order to subscribe or unsubscribe from different topics. The architecture of the Pub/Sub system can be is in **Figure 1**.

Pub/Sub
Broker

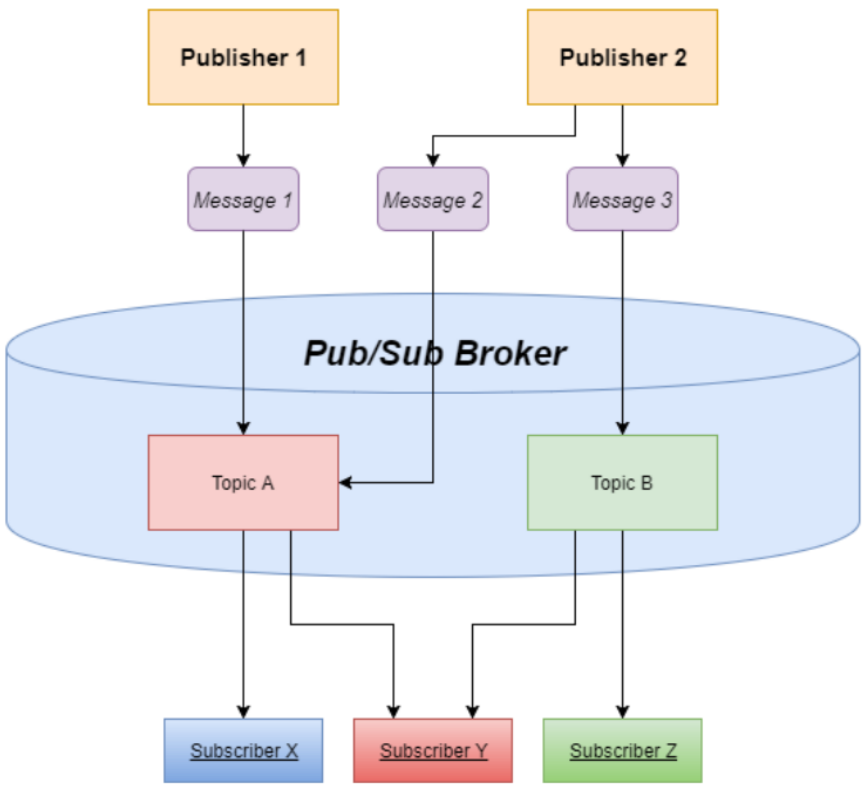

***Figure 1**: Simple architecture of Pub/Sub system. Two Publishers send messages that correspond to 2 different topics. The Broker receives these messages and sends them to the three Subscribers, according to their subscription status.*

For the implementation of various functions of this Pub/Sub system we implemented threading. Threads help us create separate flow of execution in our code. In Python the different threads do not execute at the same time, however for tasks that spend most of their time waiting for external commands (e.g., input from a user for a Subscriber/Publisher, a Broker accepting new connections, etc.) threading is a very convenient solution.

The project was written in Python. In the following sections I will describe how the three different parts of the Pub/Sub system work and how they interact in this implementation, as well as I will suggest a way of run the python scripts.

## The Subscriber

*Python script*: `subscriber`

The Subscriber, as we have already discussed, will interact with the Broker in order to receive messages on a specified subscription and subscribe or unsubscribe from topics. For this purpose, when the Subscriber is activated, it will have to connect to the Broker and subscribe to one or more topics. The specific Subscriber expects to receive a `command_file` where there are commands that it will execute once started and connected to the Broker, before giving control to the user from the keyboard. If no `command_file` is provided then the subscriber will ask the user for an input.

The execution of the script starts by checking if the necessary arguments were provided. These arguments are the following:

1. Subscriber ID: When an instance of a Subscriber is initiated from terminal, it expects to receive the same ID for the following commands. This ID will be sent to the broker so that it can keep track of the subscriber and print messages accordingly.
2. Subscriber Port: The IP address of the Subscriber is implied to be that of the current machine the subscriber is running on.
3. Broker IP
4. Broker Port

If the optional `command_file` was provided during the Subscriber's initialization the file will be stored and read line by line. Before reading the file, a connection to the Broker is established using the socket module. A socket is the endpoints in communication between programs on some network. The address family that is used to designate the type of addresses that our socket can communicate with is AF_INET (IPV4) and the connection protocol that was used is SOCK_STREAM (TCP socket) which means that the connection of the Subscriber and the Broker will be valid until it is terminated by one of the parties.

While reading the `command_file`, the format of each line will be checked and if there is a wrongly formatted line a message will appear and the execution of the code will continue ignoring that line. The context and the format of an example `command_file` must be the following:

```
2 sub #hello
10 sub #world
300 unsub #hello
25 sub #great
```

The first column represents the time in seconds the Subscriber should wait before sending the message. The second line is the command. It will have either of the two values, `sub` or `unsub`. The `sub` command will tell the Broker to subscribe the Subscriber's ID to the topic that is stated on the third column. The `unsub` command will tell the Broker to unsubscribe the Subscriber's ID from the topic. While reading the `command_file` the Subscriber will send the corresponding message to the Broker and wait for an OK message in order to continue.

Upon the end of the execution of the commands from the `command_file`, a thread which will be constantly active until the connection is terminated, will be initiated. This thread will ask the Subscriber for more commands from the keyboard. The messages that the subscriber will send to the broker over the connected socket, either through the `command_file` or through the keyboard, must have the following format: `SUB_ID COMMAND TOPIC`. If the format is incorrect a message will appear, prompting the user to enter the command in the correct format. At the same time, the Subscriber will wait for messages that correspond to the topics that the corresponding Subscriber's ID is subscribed to. When the connection is terminated the execution of the Subscriber script will end.

## The Publisher

*Python script*: `publisher`

The initialization of the Publisher will follow the same steps as the Subscriber. The connection to the Broker is the same as the Subscriber. The execution of the script starts by checking if the necessary arguments were provided. These arguments are the following:

1. Publisher ID: When an instance of a subscriber is initiated from terminal, it expects to receive the same ID for the following commands. This ID will be sent to the Broker so that it can keep track of the publisher and print messages accordingly.
2. Publisher Port: The IP address of the publisher is implied to be that of the current machine the subscriber is running on.
3. Broker IP
4. Broker Port

The Publisher is able to accept a `command_file` the same way as the subscriber and the command will be executed when the connection to the Broker is established before giving control to the user from the keyboard. The Publisher is connected to the Broker using a socket, the address family that is used is AF_INET (IPV4) and the connection protocol is SOCK_STREAM (TCP socket). The `command_file` must have the following format:

```
3 pub #hello This is the first message
5 pub #world This is another message
```

The first column represents the time in seconds the Publisher should wait before sending the message. The second line is the command which will always be `pub`. The third is the topic which the message, that come after the topic in the `command_file` line, will be send to. After each command is sent to the broker, the publisher will wait for an OK message before continuing the iteration. Once the publisher receives an OK message from the Broker it prints out in the console a similar message to the following:

```
Published msg for topic #hello: This is the first message
```

After the commands from the `command_file` have been executed, a thread which will be constantly active, until the connection is terminated, will be initiated. This thread will ask the Publisher for more commands from the keyboard. The messages that the publisher will send to the broker over the connected socket, either through the `command_file` or through the keyboard, have the following format: `PUB_ID COMMAND TOPIC`. In contrast to the Subscriber, the Publisher is not able to accept messages from the broker (except the confirmation messages).

## The Broker

*Python script:* `broker`

The Broker will have to connect the Publishers with the Subscribers. It will store the topics and the corresponding Subscribers' IDs that have subscribed to the different topics. When a Publisher sends a new message the Broker should forward the message to all subscribers that have subscribed to that topic. When the Broker is activated, it will accept from the command line the port of this specific Broker where Subscribers will connect and the port where Publishers will connect. The command is the following:

```
broker -s s_port -p p_port
```

Upon reading the command from the command line, a TXT file is created. The TXT will be filled according to the messages sent by the Subscriber. It will consist of two columns. In the first column, the topics are stored and on the second column the subscriber ID which is subscribed to that topic is stored (Topic-Sub ID pair). An example of such file is the following:

```
#hello s1
#world s1
#hello s2
```

The IP address of the broker is also assigned at this point. A global dictionary is also initialized in which we will store the Subscriber ID (key), and the connection information of that subscriber (value). We will call this dictionary `active_subs`. A Pub socket and a Sub socket is initiated.

The main function creates two threads. The first thread calls a function which will continuously accept new connections from Publishers and when it accepts a new connection a Publisher thread for this connection will be created. The second thread calls a function which will continuously accept new connections from Subscribers and create a new Subscriber thread. The function of this Pub and Sub thread swill be discussed in the following paragraphs.

The Pub thread accepts the information of the connection and receives the messages sent by the specific publisher. Other publisher threads are also able to be created, connect, send and receive messages at the same time. When a message is received, the TXT file where the pairs Topic-Sub ID are stored is being read. Furthermore, the connection information is retrieved from the global dictionary. The broker forwards the message to the appropriate subscribers using their connection information.

The Sub thread accepts the information of the connection and receives the messages sent by the specific Subscriber and sends back the appropriate messages. Other Subscriber threads are also able to be created, connect, send and receive messages at the same time. When a message is received, the connection information of the subscriber is store in the global dictionary. Moreover, the pair Topic Sub ID is stored in the TXT file if the command is `sub`. The pair is deleted from the TXT file if the command in `unsub`.

## How to run the scripts

Make the files executable

```
chmod +x path/to/broker path/to/publisher path/to/subscriber
```

The files will be able to run in the following way:

```
path/to/broker -s 9000 -p 9090
path/to/subscriber -i s1 -r 8000 -h 127.0.0.1 -p 9000 -f subscriber1.cmd
path/to/publisher -i p1 -r 8200 -h 127.0.0.1 -p 9090 -f publisher1.cmd
```

If we do not want to write the path to the file then every time, we must add the executable file to $PATH.

Arguments:

- `-s`: the port of this specific Broker where Subscribers will connect. The -s and -p values must be different.
- `-p`: the port of this specific Broker where Publishers will connect. The -s and -p values must be different.
- `-i`: Subscriber/Publisher ID.
- `-r`: Subscriber/Publisher Port.
- `-h`: Broker IP.
- `-p`: Broker Port for the Subscriber/Publisher.
- `-f`: Command file.