Publisher Subscriber System

Centralised pub/sub implementation for phase 2

Abhishek Kumar

UB Person no: 50419133

akumar58@buffalo.edu

Abstract

Publish/subscribe is a style of messaging application in which the providers of information (publishers) have no direct link to specific consumers of that information (subscribers), but the interactions between publishers and subscribers are controlled by pub/sub brokers.

In a publish/subscribe system, a publisher does not need to know who uses the information (publication) that it provides, and a subscriber does not need to know who provides the information that it receives as the result of a subscription. Publications are sent from publishers to the pub/sub broker, subscriptions are sent from subscribers to the pub/sub broker, and the pub/sub broker forwards the publications to the subscribers.

Docker

Docker is an open-source project based on Linux containers. It uses Linux Kernel features like namespaces and control groups to create containers on top of an operating system.

Dockerfile

A Dockerfile is where you write the instructions to build a Docker image.

Docker Image

Images are read-only templates that you build from a set of instructions written in your Dockerfile. Images define both what you want your packaged application and its dependencies to look like *and* what processes to run when it's launched.

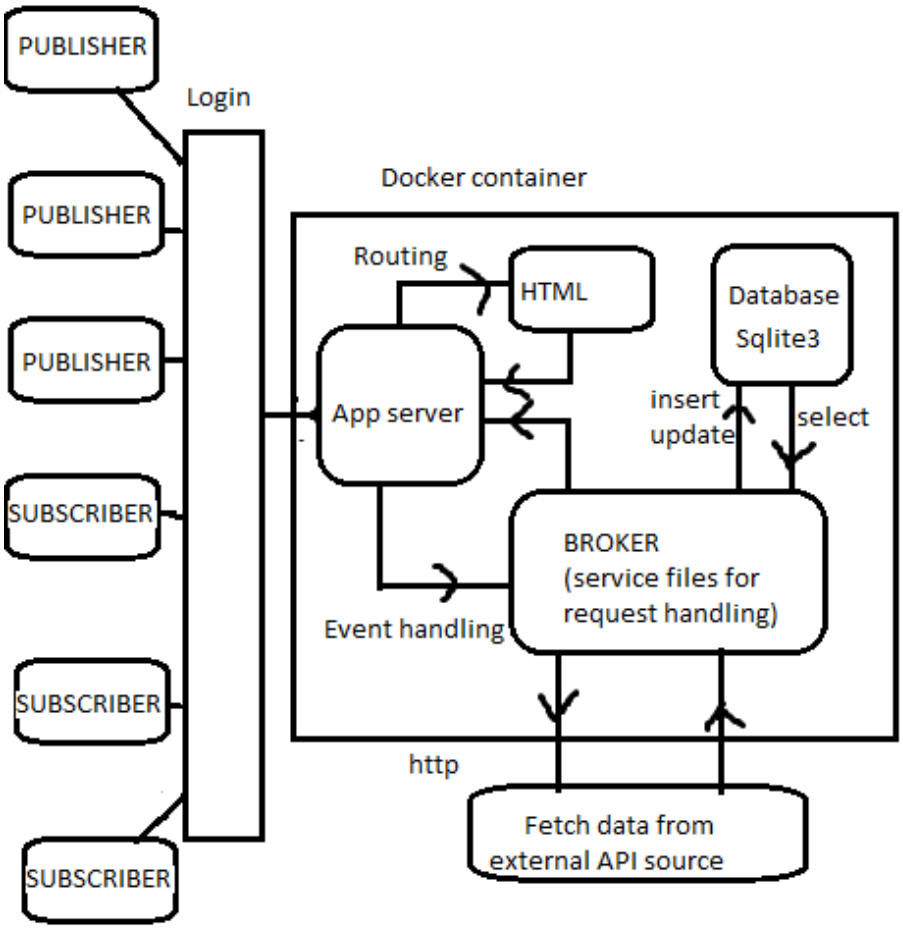
Introduction

In phase 2 on the project we have implemented a centralized pub/sub system using Flask, sqlite3 and docker. The application processes the inputs from publishers and subscribers for 3 topics and provides output on the basis of user identity and functionality.

High Level Design

The application is built with python and Flask using sqlite3 as a database for data persistence. Docker is used to run the application inside a container. The architecture model below shows the interaction between different components of the system.

Architecture model



HLD for centralized Pub/Sub system in Docker

Low Level Design

ER DIAGRAM

1. eventBroker tabe

Column name	Type
eid	INTEGER, PRIMARY KEY
eventData	TEXT

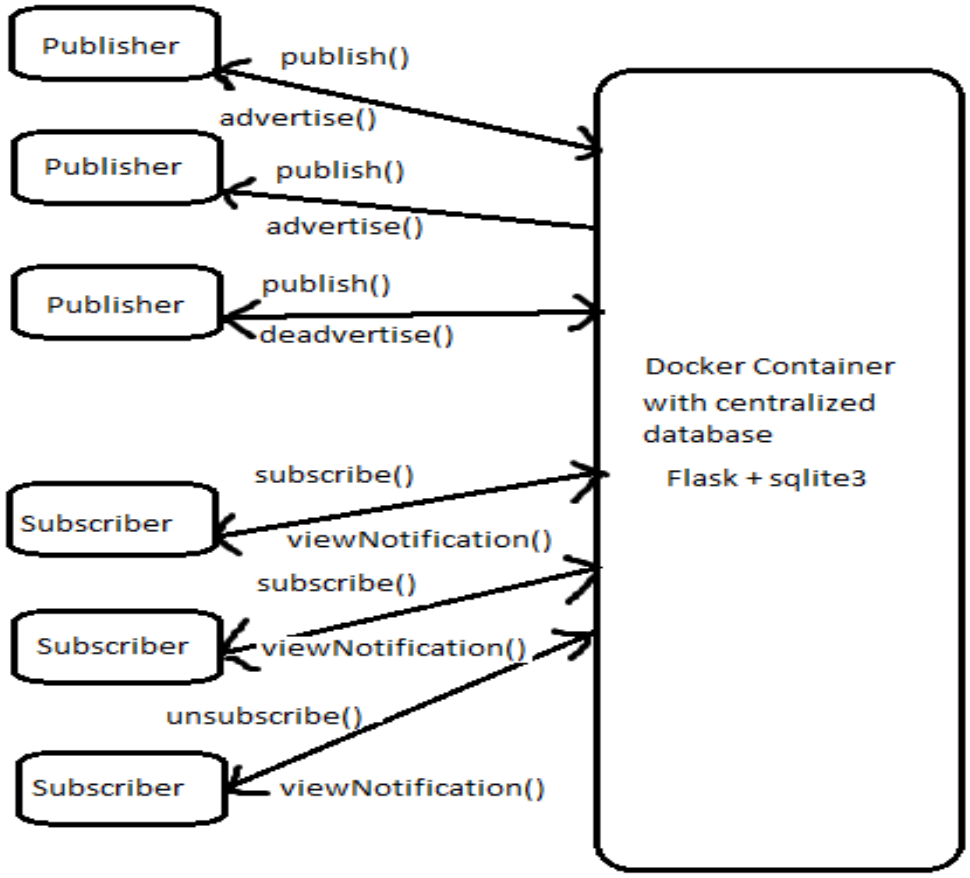
2. publisher table

Column name	Type
pid	TEXT
eid	INTEGER
advertisement	INTEGER

3. Subscriber table

Column name	Type
sid	TEXT
eid	INTEGER
subscription	INTEGER
notification	INTEGER

Interaction Diagram



LLD for pub/sub system

Project structure

1. phase2_sqlite_docker
    a. Templates
        - Index.html
        - Pub.html
        - sub.html
    b. app.py
    c. eventService.py
    d. pubService.py

e. subService.py
f. create_db.py
g. pubsub.db
h. Dockerfile
i. requirements.txt

Docker container contains:

1. Application server
2. Database
3. Template files

Service file

1. **eventBroker** - handles publish and fetch api requests
2. **pubService** - handles publisher requests
3. **subService** - handles subscriber requests

Functions

1. **publish(pid,eid)**
    - checks if (pid, eid) in publisher table
    - If not present , then stores the pid, eid in publisher table and set advertisement to 0 (default)
    - Then calls eventService to fetch data through api call
    - Then update eventData in eventBroker table at eid
    - Then prints advertisement as per publisher specification
2. **advertise(pid,eid)**
    - Checks if (pid, eid) in publisher table
    - If not present, then stores the pid , eid in publisher table
    - Then set advertisement to 1 for the (pid,eid)
3. **deadvertise(pid,eid)**
    - Checks if (pid, eid) in publisher table
    - If not present, then stores the pid , eid in publisher table
    - Then set advertisement to 0 for the (pid,eid)
4. **subscribe(sid,eid)**
    - check if (sid+eid) is present in subscriber table

- if not the add and set subscription to 1 for (sid,eid)
- if present , then just set subscription to 1 for (sid,eid)

5. **unsubscribe(sid,eid)**
    - check if (sid+eid) is present in subscriber table
    - if not the add and set subscription to 0 for (sid,eid)
    - if present , then just set subscription to 0 for (sid,eid)
6. **notify(pid,eid)**
    - After publisher publishes,check if publisher(pid) has advertisement == 1 for that eid
    - if yes then set notification for all sid subscribed to that eid to 1
    - subscriber will hit viewnotification() to check if they have any new updates in that topic and print message accordingly
    - After publish , login page will also display advertisement by publisher for the particular event
7. **view(sid,eid)**
    - checks if (sid + eid) present in subscriber table
    - if not then print(not subscribed)
    - else fetch data from eventBroker table based on eid and display message
8. **viewnotification(sid,eid)**
    - checks if (sid+eid) and notification is 1
    - if not then print "No new messages"
    - else if present print then "you have new updates"
    - Then set notification for that (sid, eid) back to 0

RESULTS

Integration Testing

Steps to run application:

1. Inside the project folder
    a. Run " **docker image build -t phase2-docker .** "
    b. Run " **docker image ls**" to see if image is created
    c. Then run " **docker run -p 5001:5000 -d phase2-docker**"
    d. Open browser and hit " **http://localhost:5001**"
    e. Enter password "**pppp**" for publisher
    f. Enter password "**ssss**" for subscriber

Input from publisher:

Publisher	Topic	Advertisement	Output
p1	1 : Dad Jokes	yes	Published and ad in login page
p2	2: Random advice	yes	Published and ad in login page
p3	3: Inspirational quotes	no	Published but no ad in login page

Output to subscriber:

Subscriber	Topic	Subscription	Output
s1	1 : Dad Jokes	yes	Notification with new joke
s2	2: Random advice	no	No updates
s3	3: Inspirational quotes	yes	No notification (since p3 is not advertising) and new inspirational quotes

REFERENCES

1. Pub/Sub, https://www.ibm.com/docs/en/app-connect/11.0.0?topic=applications-publishsubscribe-overview
2. Docker , https://docs.docker.com/.
3. Flask docs, http://flask.pocoo.org/docs