IEMS 5722

Mobile Network Programming and Distributed Server Architecture

2015-2016 Semester 2

Assignment 3: Building an Application Server

Due Date: 18th March, 2016

Notes:

i.) Read carefully the instructions and requirements to understand what you have to do

ii.) Follow the instructions in Section 4 to submit your files for marking

iii.) Late submissions will receive 30% mark penalty

1. Objectives

- To learn how to develop a server-side application using Python and Flask
- To learn how to develop APIs for mobile apps
- To learn how to use HTTP for communication between server and client
- To learn how to deploy an HTTP server-side application using Nginx, Gunicorn and Supervisor

2. Instructions

In this assignment, you will try to develop the **server-side application** for the mobile app developed in Assignment 2. You should setup a server using **Nginx** as the Web server, develop the server application using **Python** and **Flask**, and deploy the application using **Gunicorn** and **Supervisor**. The API you develop in this assignment should be the same as the ones provided in Assignment 2. Hence, the app developed in Assignment 2 should work with the server application developed in this assignment without any modification except the URL of the APIs.

2.1. Setup Server

You are strongly advised to develop your server application on a cloud platform such as **Amazon AWS** or **Google Cloud Compute Engine**. Both of these services offer some forms of free trials. Whenever possible, choose **Ubuntu Linux 14.04** as the operating system. All the following instruction assumes that your server is running Ubuntu 14.04.

In this assignment, you will have to use the following software applications: Nginx Web server, Gunicorn WSGI server, Supervisor, MySQL Database and various Python modules. Use the following commands to install these packages first.

```
$ sudo apt-get update
$ sudo apt-get install nginx gunicorn supervisor
```

```
$ sudo apt-get install mysql-server
$ sudo apt-get install python-pip
$ sudo apt-get install python-mysqldb
```

2.2. Develop Server Application

First of all, develop your server application and implement the APIs for the mobile app. Python is by default installed in Ubuntu. To check whether you have Python installed:

```
$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you see the interactive shell of Python coming up, that means Python is available. In addition, you will have to install the **Flask** framework and some dependencies for developing the server application.

```
$ sudo pip install flask
```

To check whether Flask has been successfully installed, try to import the package in the Python interactive shell:

```
ubuntu@ip-xxx-xxx-xxx-xxx:~$ python
Python 2.7.6 (default, Mar 22 2014, 22:59:56)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import Flask
>>>
```

If no error is reported, then Flask has already been successfully installed.

2.2.1. Database

To support the functions to be developed, you will need to have a database that can help you store the data (e.g. messages from the users) and let you retrieve the data in the future. In this assignment, you should use the **MySQL** database. During installation, you should have set up an administration account. You can check whether you have successfully installed MySQL by invoking the command line interface:

```
$ mysql -uroot -p
```

```
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5001
Server version: 5.5.47-0ubuntu0.14.04.1 (Ubuntu)
Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

If you can enter the command line client successfully, this means that MySQL has successfully installed.

To support the server application, you should first create a database for your application. Use the following command to create a database called "**iems5722**":

```
mysql> CREATE DATABASE iems5722;
```

You will then have to create some tables to store the data. You can create your own database schema if you like, otherwise you can use the following queries to create two tables for use in this assignment.

```
CREATE TABLE `chatrooms` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(100) NOT NULL,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET=utf8;

CREATE TABLE IF NOT EXISTS `messages` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `chatroom_id` int(11) NOT NULL,
  `user_id` int(11) NOT NULL,
  `name` varchar(100) NOT NULL,
  `message` varchar(300) NOT NULL,
  `timestamp` datetime NOT NULL,
  PRIMARY KEY (`id`)
) DEFAULT CHARSET=utf8;
```

Go on to **populate** the ‘chatrooms’ table by using the **INSERT statement**. Once you are done with creating the database and the tables, you can now focus on the development of the APIs.

2.2.2. APIs

You will have to implement the following three APIs in your server application.

1. **`GET: /iems5722/get_chatrooms`**
   For retrieving a list of chat rooms from the servers
   No input parameters required
   Sample output:


```
{
  "data": [
    {
      "id": 3,
      "name": "Chatroom 002"
    },
    {
      "id": 2,
      "name": "General Chatroom"
    }
  ],
  "status": "OK"
}
```


2. **`GET: /iems5722/get_messages`**
   For retrieving a list of messages in a specific chat room
   Input parameters: chatroom_id, page
   Sample output:


```
{
  "data": {
    "current_page": 1,
    "messages": [
      {
        "message": "18",
        "name": "Albert",
        "timestamp": "2016-01-23 19:36",
        "user_id": 1
      },
      {
        "message": "17",
        "name": "Albert",
        "timestamp": "2016-01-23 19:36",
        "user_id": 1
      },
      {
        "message": "16",
        "name": "Albert",
        "timestamp": "2016-01-23 19:36",
        "user_id": 1
      },
```

```
      {
        "message": "15",
        "name": "Albert",
        "timestamp": "2016-01-23 19:36",
        "user_id": 1
      },
      {
        "message": "14",
        "name": "Albert",
        "timestamp": "2016-01-23 19:36",
        "user_id": 1
      }
    ],
    "total_pages": 4
  },
  "status": "OK"
}
```

3. **POST: /iems5722/send_message**

   For submitting a message to the server when the user has input some text and clicked the send button

   Input parameters: chatroom_id, user_id, name, message

   Sample output:

```
{
  "status": "OK"
}
```

Note: All responses of your API must follow exactly the specified format. Otherwise the app you developed in Assignment 2 will not work. In addition, your application should check if the client side has supplied the correct parameters, if any parameter is missing, you should return the following response instead:

```
{
  "message": "<error message>"
  "status": "ERROR"
}
```

2.3. Test Your Application

Once you are done with the development of the server application, you can first test your application by directly executing the application (make sure that you have enabled debug mode in your Flask application. For example:

```
$ python app.py
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger pin code: 482-038-008
```

Then you can access port 5000 (or the port you have set) from your browser to test your application. For example, if the server's IP address is 123.45.67.89, the URL will be http://123.45.67.89:5000/. Make sure you have set the firewall of your server such that it allows traffic through the port you specified. (Refer to the pages below if you are using Amazon AWS or Google Compute Engine)

- Amazon AWS
  http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/authorizing-access-to-an-instance.html
- Google Computer Engine
  https://cloud.google.com/compute/docs/networking

2.4. Deploy Your Application

If you have tested your application and see that it is ready to be deployed, then you should follow the steps below to deploy your application. When deploying an application, we would like to make sure that it is monitored, and when it crashes or stopped for some reasons, it will be automatically restarted, so that the client will not see an interruption of services for a long time.

As discussed in lecture, we will use Gunicorn to deploy your Flask application, and then use Supervisor to monitor the Gunicorn process. Finally, we have Nginx as the Web server to redirect requests to Gunicorn if the requests are for your app. The figure below illustrates what you will achieve by the end of this section.

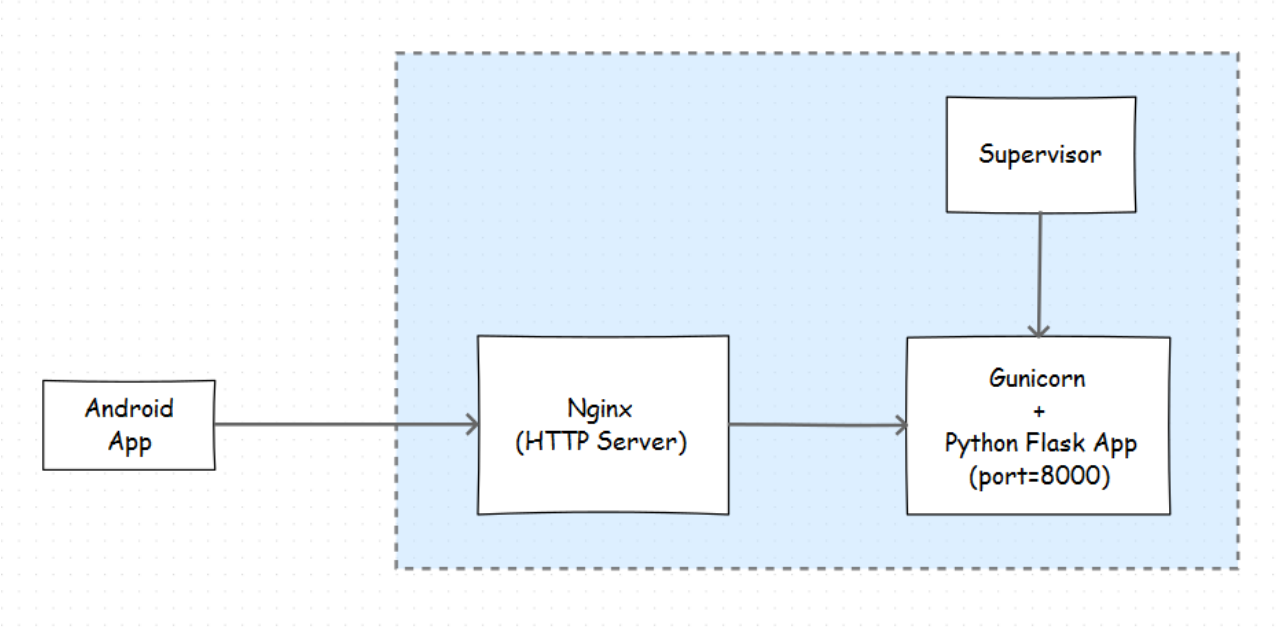

Figure 1: System Architecture

2.4.1. Gunicorn and Supervisor

You will use Gunicorn to host your Flask application. In the directory in which you have created the

server application, you can use the following command to ask Gunicorn to run your application on port 8000 (Gunicorn listens at this port for incoming requests, and will route these requests to the functions in your application depending on the path requested).

```
$ gunicorn app:app –b localhost:8000
2016-02-23 21:44:16 [23407] [INFO] Starting gunicorn 17.5
2016-02-23 21:44:16 [23407] [INFO] Listening at: http://127.0.0.1:8000 (23407)
2016-02-23 21:44:16 [23407] [INFO] Using worker: sync
2016-02-23 21:44:16 [23412] [INFO] Booting worker with pid: 23412
...
```

If you see something like above, then you have successfully used Gunicorn to host your application.

When you system is in production, you would like it to be monitored by some other processes, such that it can be restarted automatically after it has crashed, or has been killed by the operating system for some reasons. This is where Supervisor comes into the scene. To put it in a simple way, instead of directly running Gunicorn, we ask Supervisor to help us execute and monitor Gunicorn to host our Flask application.

To use Supervisor, we have to first create a configuration file. Below is an example:

```
[program:iems5722]
command = gunicorn app:app -b localhost:8000
directory = /home/albert/iems5722
user = albert
autostart = true
autorestart = true
stdout_logfile = /home/albert/iems5722/app.log
redirect_stderr = true
```

For detailed descriptions of the commands and parameters, refer to the documentation of Supervisor at http://supervisord.org/. We briefly go through the lines here:

- Line 1 specifies a program called ‘iems5722’, you can use any other name in place of ‘iems5722’
- Line 2 specifies the command to be executed by Supervisor. In this case we want to execute Gunicorn and ask it to host our application
- Line 3 specifies the directory in which the command should be executed
- Line 4 specifies the user who will execute the command
- Line 5 and 6 specify that Supervisor should start the app automatically on startup, and restart the app automatically when it has been killed

- Line 6 and 7 specify where to redirect the standard output and standard error

You should create a similar configuration file, and then follow the steps below to deploy your application. Firstly, copy your configuration file to the following path **`/etc/supervisor/conf.d`**, assuming that your configuration file is named **`app.conf`**:

```
$ sudo cp app.conf /etc/supervisor/conf.d/
```

Next, we need to ask Supervisor to read your configuration file and update its status (executing the command specified in the configuration file:

```
$ sudo supervisorctl reread
$ sudo supervisorctl update
```

By now, if everything is configured correctly, your application should now be running. You can check by using the following command:

```
$ sudo supervisorctl
Iems5722                  RUNNING    pid 24000, uptime 0:00:20
supervisor>
```

Your next step is to set up the Nginx HTTP server such that it will redirect any request related to your application to the Gunicorn server.

2.4.2. Nginx

To configure Nginx, you will need to create a configuration file under the directory **`/etc/nginx/sites-enabled/`**. The following is an example configuration that would probably work in your case:

```
server {
    listen 80;
    listen [::]:80;

    root /home/albert/iems5722;
    index index.php index.html index.htm;

    server_name localhost;

    location /iems5722 {
        proxy_pass http://localhost:8000;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header Host $http_host;
        proxy_http_version 1.1;
        proxy_redirect off;
```

```
        proxy_buffering off;
    }

}
```

The configuration asks Nginx to forward any request to the path /iems5722 (and its sub-paths) to port 8000, which is where Gunicorn is listening for incoming requests. The other parameters simply set the headers when Nginx is redirecting the HTTP requests. For details of these parameters, you can refer to the documentation of Nginx: http://nginx.org/en/docs/.

Save the above configuration in file named iems5722.conf, and restart Nginx using the following command.

```
$ sudo service nginx restart
```

By this time, everything should be ready, and you should be able to access the APIs by accessing URLs such as **`http://123.45.67.89/iems5722/get_chatrooms`**.

3. Requirements

The server application you developed should have the following features:

- A server application written in Python using the Flask framework
- The server application should implement the three APIs to support the instant messaging app
- API 1: GET: /iems5722/get_chatrooms – returns a list of chatrooms available (you should at least create one chatroom in your database)
- API 2: GET: /iems5722/get_messages – returns the list of messages in a particular chat room, sorted in reverse chronological order (the latest message comes first), and implements the paging mechanism as shown in the example
- API 3: POST: /iems5722/send_message – allow the app to submit a new message to a chat room, it should insert the data into the database, and automatically fill in the timestamp fill using the time at which the message is received

4. Submission

To facilitate marking of the assignments, you should strictly follow the instructions below. To submit your assignment, create a folder name <your_student_id>_assgn3. In the folder, you should include the following items:

- All source codes of your **Python server application**
- A text file containing the **full URLs** of the three APIs

Compress this folder using ZIP, you should now have a file named **<your_student_id>_assgn3.zip**.

Submit it in the CUHK eLearning System online: https://elearn.cuhk.edu.hk/

5. Marking Criteria (Total 100 Marks)

- **(30 marks)** The /iems5722/get_chatrooms API correctly queries the database, retrieves the list of chat rooms, and returns a JSON response in the specific format.
- **(30 marks)** The /iems5722/get_messages API correctly queries the database, retrieves the list of messages (of a particular page), and returns a JSON response in the specific format.
- **(40 marks)** The /iems5722/send_message API correctly receives the parameters from the client, inserts a new record into the database, and returns a JSON response in the specific format.