Cloud Computing Project Report

 # Milestone 4

Group A Members	
ZHANG Yang	19441789
WU Peicong	19434405
LI Jinhui	19439822

Contents

1. Architecture of LINE Chatbot

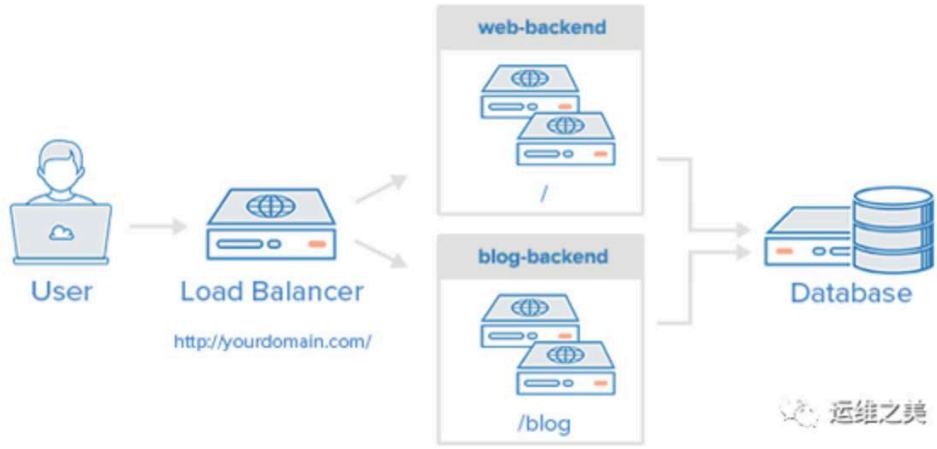
In the lectures of this course, there are many different concepts of architectures. It introduced two popular architectural styles, software and hardware service layers and Tiered Architecture. For our LINE Chatbot service, it actually related to three-tier architecture pattern and Client-Server architecture style.

The LINE Mobile APP is considered as the first tier of the three-tier architecture, which manages user interfaces and controls. The second tier is application server that focus on application logic. We implement Heroku cloud server[1] on this layer. Our Chatbot can run on this server efficient and also invoke other services such as Google Map, News collection API, etc. Heroku Cloud Server actually provides higher capability, reliability and lower running cost than local machines. On the third tier, we use Redis Server[2], which is a cloud database management system with easy accessibility and high reliability. In this layer, Redis Storage Server is mainly developed for storing News data, geography information and published records, etc.

The architecture of our LINE Chatbot actually belongs to Client-Server architectural model. LINE Mobile App is considered as the Client role. It sends requests to our own servers and gets responses from the servers. In addition, there are many servers, and each server is able to communicate with others through http protocol.

2. The methods to increase capacity

Load balancer is a traffic distribution control service that distributes access traffic to multiple back-end cloud servers according to the forwarding strategy. The specific process of load balancing refers to the traffic of user access through the load balancer server, such as nginx[3], etc., evenly distributed to multiple back-end servers by obeying a certain forwarding strategy. The back-end server can independently process request and respond to achieve the effect of spreading the load.



Load balancing[4] expands application service capabilities and enhances application availability. We can build high-performance clusters to cope with greater access traffic by using load balancing.

There are two main ways to implement load balancing on LINE Chatbot

1. Heroku's built-in load balancing

Since we are using a free version server, we just followed the Heroku developer tutorials. The main idea is that we can improve the capability of Chatbot by adding more dynos. The details are showed below.

Both horizontal and vertical scale are features of the professional dynos, and are not available to `free` or `hobby` dynos.

Scaling from the CLI

Scaling the number of dynos

You scale your dyno formation from the Heroku CLI with the `ps:scale` command:

```
$ heroku ps:scale web=2
Scaling dynos... done, now running web at 2:Standard-1X
```

The command above scales an app's `web` process type to 2 dynos.

You can scale multiple process types with a single command, like so:

```
$ heroku ps:scale web=2 worker=1
Scaling dynos... done, now running web at 2:Standard-1X, worker at 1:Standard-1X
```

You can specify a dyno quantity as an absolute number (like the examples above), or as an amount to add or subtract from the current number of dynos, like so:

```
$ heroku ps:scale web+2
Scaling dynos... done, now running web at 4:Standard-1X.
```

If you want to stop running a particular process type entirely, simply scale it to `0`:

```
$ heroku ps:scale worker=0
Scaling dynos... done, now running web at 0:Standard-1X.
```

Changing dyno types

To move a process type from `standard-1x` dynos up to `standard-2x` dynos for increased memory and CPU share:

```
$ heroku ps:scale web=2:standard-2x
Scaling dynos... done, now running web at 2:Standard-2X.
```

Note that when changing dyno types, you must still specify a dyno quantity (such as `2` in the example above).

Moving back down to `standard-1x` dynos works the same way:

```
$ heroku ps:scale web=2:standard-1x
Scaling dynos... done, now running web at 2:Standard-1X
```

See the documentation on dyno types for more information on dyno types and their characteristics.

HTTP routing

Depending on your dyno formation, some of your dynos will be running the command associated with the `web` process type, and some will be running other commands associated with other process types.

The dynos that run process types named `web` are different in one way from all other dynos - they will receive HTTP traffic. Heroku's HTTP routers distribute incoming requests for your application across your running web dynos.

So scaling an app's capacity to handle web traffic involves scaling the number of web dynos:

```
$ heroku ps:scale web+5
```

A random selection algorithm is used for HTTP request load balancing across web dynos - and this routing handles both HTTP and HTTPS traffic. It also supports multiple simultaneous connections, as well as timeout handling.

2. Load balancer service based on Python

Python is a powerful programming language. Using Python to develop this Chatbot is one of the requirements. Picture of sample codes is showed below.

```python
from __future__ import unicode_literals

import os
from argparse import ArgumentParser
from flask import Flask, redirect
import random

app = Flask(__name__)

heroku_port = os.getenv('PORT', None)

@app.route("/", methods=['GET'])
def index():
    host = ["https://couldcomputing.herokuapp.com/test","https://afternoon-hamlet-06392.herokuapp.com/test"]
    rand = random.randint(0, 1)
    return redirect(host[rand])

if __name__ == "__main__":
    arg_parser = ArgumentParser(usage='Usage: python ' + __file__ +
                                ' [--port <port>] [--help]')
    arg_parser.add_argument('-d', '--debug', default=False, help='debug')
    options = arg_parser.parse_args()

    app.run(host='0.0.0.0', debug=options.debug, port=heroku_port)
```

We need to do stress testing to evaluate the performance of the system when we implement load balancer for the servers. We generally use the number of requests processed by the server per unit time to describe its concurrent processing capabilities. Called Throughput, the unit is "req / s". Higher throughput means greater performance.

In this project, we select Apache AB as our stress testing tool. As sample picture showed below, we did some general stress testing for our Chatbot. Because of the servers is free version, the improvement is not significant. A better server may be a good solution.

```
Benchmarking couldcomputing1.herokuapp.com (be patient)
Completed 100 requests
Completed 200 requests
Completed 300 requests
Completed 400 requests
Completed 500 requests
Completed 600 requests
Completed 700 requests
Completed 800 requests
Completed 900 requests
Completed 1000 requests
Finished 1000 requests


Server Software:        Werkzeug/1.0.1
Server Hostname:        couldcomputing1.herokuapp.com
Server Port:            443
SSL/TLS Protocol:       TLSv1.2,ECDHE-RSA-AES128-GCM-SHA256,2048,128
Server Temp Key:        ECDH P-256 256 bits
TLS Server Name:        couldcomputing1.herokuapp.com

Document Path:          /
Document Length:        289 bytes

Concurrency Level:      10
Time taken for tests:   87.921 seconds
Complete requests:      1000
Failed requests:        507
   (Connect: 0, Receive: 0, Length: 507, Exceptions: 0)
Non-2xx responses:      1000
Total transferred:      547168 bytes
HTML transferred:       297112 bytes
Requests per second:    11.37 [#/sec] (mean)
Time per request:       879.211 [ms] (mean)
Time per request:       87.921 [ms] (mean, across all concurrent requests)
Transfer rate:          6.08 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      618  645  35.8    634    1453
Processing:   208  223  36.4    217     977
Waiting:      208  223  36.0    217     972
Total:        827  868  55.0    851    1676

Percentage of the requests served within a certain time (ms)
  50%    851
  66%    894
  75%    897
  80%    899
  90%    909
  95%    917
  98%    924
  99%    994
 100%   1676 (longest request)
```

3. PaaS, IaaS or SaaS?

LINE Chatbot is an application based on SaaS[5] (Software-as-a-service) mode. It is an application running on the cloud computing infrastructure. Users can access the application on the LINE Mobile APP. The end users don't need to care about technical issues. The development, management and maintenance of LINE Chatbot are all in the charge of the developers.

LINE Chatbot runs on the Heroku server. Heroku is a PAAS (Platform-as-a-Service) based service, which can deploy the required python libraries and codes to the cloud computing infrastructure of the supplier. Heroku is a software deployment platform, which abstracts the details of hardware and operating system, and can be expanded seamlessly. Developers only need to pay attention to their own business logic, not the bottom layer infrastructure.

In terms of system architecture, LINE Chatbot based on SaaS mode runs on Heroku based on PAAS mode, LINE Chatbot itself does not provide hardware infrastructure services, so it does not belong to IaaS mode. At the same time, using LINE Chatbot does not require user to develop and maintain the applications, so it does not belong to PAAS mode, LINE Chatbot is a complete application that can be used directly, so it is a SaaS mode service. From the perspective of target users, IaaS and PaaS are mostly aimed at software developers, while applications in SaaS mode are directly aimed at users, just like LINE Chatbot is a service for users who care about public health care, so it is a SaaS mode service.

References

[1] Heroku, 'Heroku Documentation - Python Version', 2020. [Online]. Available: https://devcenter.heroku.com/categories/python-support **. [Accessed: 15- March - 2020]**

[2] Redis, 'Redis-py Documentation', 2020. [Online]. Available: https://redis-py.readthedocs.io/en/latest/ **. [Accessed: 15- March - 2020]**

[3] 王志利, '利用 NGINX 最大化 Python 性能，第二部分：负载均衡和监控', 2016. [Online]. Available: http://blog.oneapm.com/apm-tech/703.html **. [Accessed: 1- April - 2020]**

[4] xybaby, '关于负载均衡的一切：总结与思考', 2017. [Online]. Available: https://www.cnblogs.com/xybaby/p/7867735.html **. [Accessed: 9- April - 2020]**

[5] Beanmoon, '云计算的三种服务模式：IaaS，PaaS 和 SaaS**', 2018. [Online]. Available:** https://www.cnblogs.com/beanmoon/archive/2012/12/10/2811547.html**. [Accessed: 1- April - 2020]**