Cloud Computing 2022/23 spring

Practice 6 - Load balancing

In this lab, we will take a look at load-balancing web applications running on top of IaaS cloud instances. The Nginx is used as a load balancer and configured with certain (such as round-robin, weighted round-robin, IP-hash, etc.) load balancing algorithms. You will set up 3 instances: Two with a simple web server application and the Third one with a load balancer. The goal is to learn how load balancing works in the cloud, how to set up load generation using Locust for simple performance testing, and take a look at different load balancing strategies. The simple application is based on a Python Flask webserver with Redis (in-memory database).

Exercise 6.1. Setting up an Application Server instances

We will create the VM instances for hosting the Python-based Web Server. We will download and set up an already implemented Flask application that keeps track of how many users have visited the page and from which IP-s.

- Create two VM instances for two application servers with VM Image: ubuntu22.04 and choose the VM flavors as below:
  - Application Server 1 (Flavour: m3.nano)
  - Application Server 2 (Flavour: m1.xsmall)
    - Make sure to enable the `Delete Volume on Instance Delete` option

- Log into the instances through SSH
- Install the required software packages on the two instances:
  - Refresh the list of software packages: `sudo apt update`
  - Install the following ubuntu packages: `sudo apt install python3-venv && sudo apt install redis-server`

About the Python Application and Redis

This web app checks the IP address of who is making the request and stores/increments counter values in a hashmap, tracking how many times each IP address has visited (key: ip address, value: no. of requests). The HashMap is stored in the Redis in-memory data structure store. In this lab we are simply using Redis as a database, but Redis is actually designed for distributed use, meaning it can be used as a fast cache/database/message broker shared by multiple machines in a cluster.

- Prepare & run the application on both of the servers
  - Clone the sample code of the application `git clone https://shivupoojar87@bitbucket.org/shivupoojar87/2023lab6app.git`
  - Create and run the flask application as mentioned in Practice Session 1
  - Don't forget to install pip packages using the *pip install* command

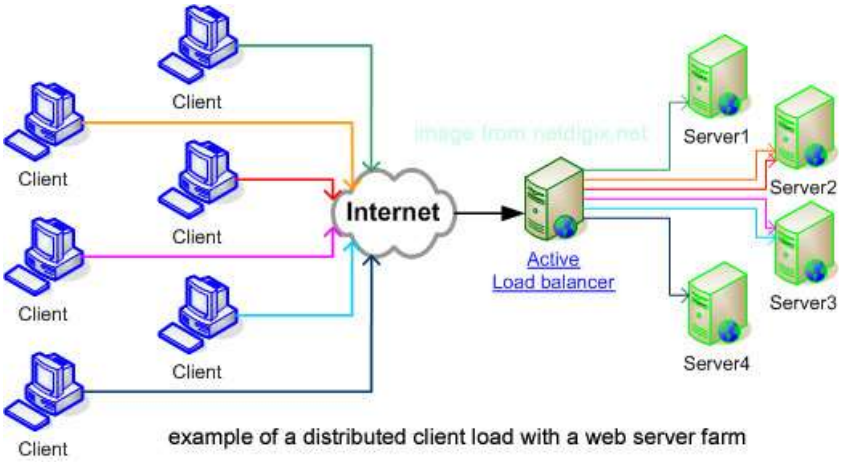
- Access your instance through a web browser using its Public IP to check that the website is working as required in both of the VMs
- Modify `./templates/index.html` to make the web page more personal to you.
  - How you decide to modify it is up to you, but there should at least be your Full Name present.
  - Background color could be added to make difference between two application servers so that you can recognize them when accessing the load balancer.

- If at any point you need to clear your entire Redis storage, you can run this command:
  - `redis-cli FLUSHDB`

Exercise 6.2. Setting up load balancer instance and balancing multiple application servers

In this exercise, we set up another instance and install a NginX load balancer on it.

A Load balancer (LB) distributes all user requests to a web page across multiple application servers. Load balancers are used to increase capacity (concurrent users) and reliability of applications and are a required component for auto-scaling: meaning changing the number of application servers dynamically based on number of active users.



(image taken from http://www.netdigix.com/linux-loadbalancing.php)

- We are using Nginx (http://nginx.org/en/) as a load balancer.
- **Create a new instance** (Should also be Ubuntu 22.04) for the load balancer
  - Make sure to select the "Delete Volume on Instance Delete" option in the Source tab
  - Use `m3.nano` flavor
- Install Nginx on the instance
  - Refresh the list of software packages: `sudo apt update`
  - Install the following ubuntu package: `nginx-full`
- Modify Nginx configuration to add your application server IP into the list of managed services.
  - Download the example nginx load balancer configuration: `wget https://courses.cs.ut.ee/LTAT.06.008/2023_spring/uploads/Main/default`
  - Overwrite the default nginx site configuration on the instance with the modified configuration
    - `sudo cp default /etc/nginx/sites-enabled/default`
  - Modify the `default` configuration file `sudo nano /etc/nginx/sites-enabled/default`
    - Find the **upstream** _____ block
    - Modify for example `server 172.17.64.109:5000;` lines in the configuration file with IP of your application servers.
- Reload Nginx service
  - `sudo service nginx reload`
  - Remember to run this command again every time you modify the local default file
- Visit the IP address of the **load balancer** using a web browser and verify that it displays your any of the application server
- You can also check current incoming HTTP connections to your **Application Server** using the following command: `netstat | grep http` (You can run your application in the background using nohup)

Exercise 6.3. Generating additional user traffic for benchmarking

Let us check whether the load balancer can now handle a higher number of simultaneous user requests.

- Your task is to generate a large number of user requests to the load balancer and verify how many of those requests are sent to your application server by the load balancer.
- To generate a large number of the user requests, we will use Locust, which is a Python-based framework for simulating users and generating web traffic.

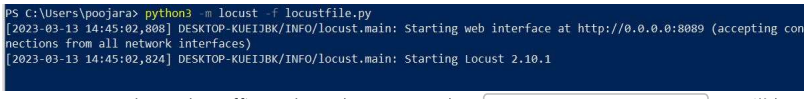
PS!! Locust installation can be done on your local machine as pip package.

- Let us, install Locust and generate web user requests.
  - Install Locust as a pip package `pip3 install locust`

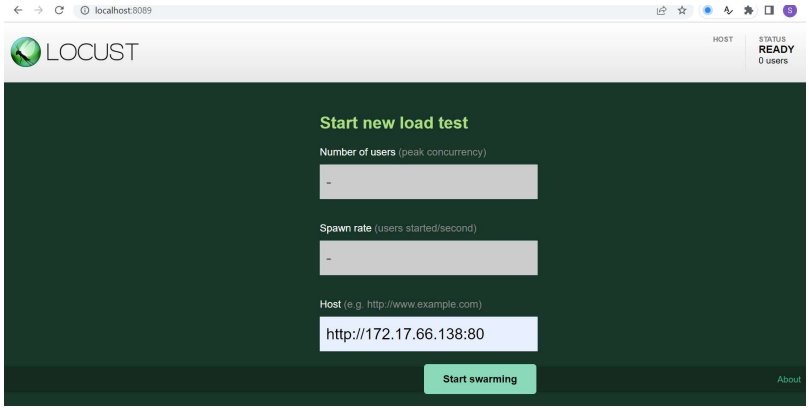
- Creating web traffic using the locust tool requires writing **locustfile.py**. A nice tutorial for creating locustfile.py can be found here: http://docs.locust.io/en/stable/quickstart.html
  - Let us write `locustfile.py` for our web application testing as shown below. Here, *@task* is a web task that is executed by the locust tool. In this code, it invokes the endpoint **/** . The **stages** contains the load specification with elements:
    - duration -- Time to run the test. For example 60 seconds here.
    - users -- Total user count
    - spawn_rate -- Number of users to start/stop per second

```python
from locust import User,HttpUser, TaskSet, events, task, constant
from locust import LoadTestShape

class HelloWorldUser(HttpUser):
    @task
    def hello_world(self):
        self.client.get("/")
    wait_time = constant(1)

class StagesShape(LoadTestShape):
    stages = [{'users': 100, 'duration': 60, 'spawn_rate': 10}]

    def tick(self):
        run_time = self.get_run_time()
        for stage in self.stages:
            if run_time < stage["duration"]:
                tick_data = (stage["users"], stage["spawn_rate"])
                return tick_data
        return None
```

- Run the locust in the terminal/Windows PowerShell of your machine using `python3 -m locust -f locustfile.py` and you should see the output as below

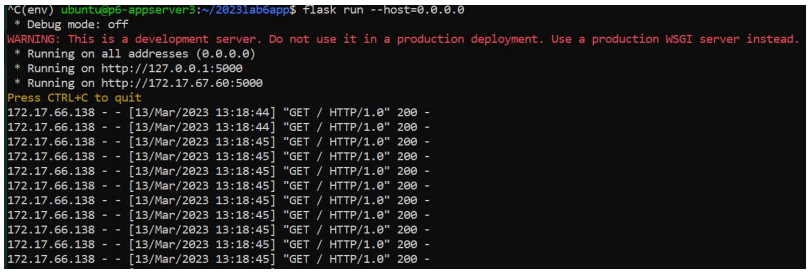


- Locust provides a web interface to generate the web traffic and can be accessed at `http://localhost:8089`. It will be shown as below:

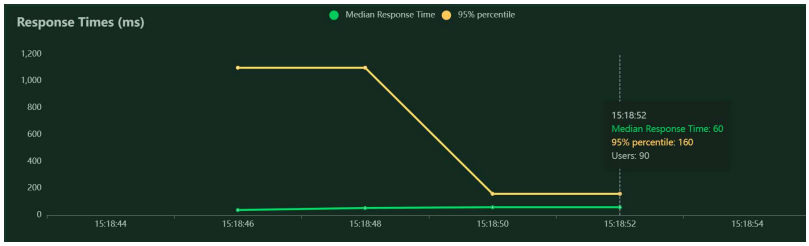


- Change the Host (e.g. http://172.17.66.138:80) to http://VM_IP:80 (VM_IP is the IP of nginx load balancer)
- Click on Start Swarming

- Now the user traffic is initiated and meanwhile open your VM terminals and it should display the invocations of the application service where the load balancer served the request.



- Move on to the locust web user interface and Wait for the completion of the load test.
  - Note down how many requests ended up in the load test. You can get from # **Requests** from the dashboard and answer the question below.
    - Question 1: How much percent of your generated user requests ended up visiting each of your servers? (Total count of user requests sent to each application server can be seen by opening the application server on the browser or by opening the load balancer on the web and then calculating the percent based on total requests sent from locust)
  - Check the response time graph by click on **Chart-->Response Time (ms)**, that shows two metrics:
    - 95% percentile response time: It indicates a response time of 95% of the user requests.
    - Mean response time: It indicates the mean response time of all user requests in a given time.



- Download the test report from locust UI and name it as "Task6_3_report.html". (To download the report, go to Locust UI-->Download Data-->Download Report).

Exercise 6.4. Working with load-balancing techniques using Nginx service

In this exercise, we are going to learn different load-balancing techniques that are supported in nginx load balancer. Load balancing is a technique to distribute incoming user requests across multiple application servers. By default, the round-robin technique is used in the Nginx load balancer. The list of different strategies are listed below and you can also have look here

1. Round Robin and weighted Round Robin: Distributes requests across the upstream servers in order.
2. Least Connections and weighted Least Connections: Forwards requests to the server with the lowest number of active connections.
3. Least Time and weighted Least Time: Forwards requests to the least loaded server, based on a calculation that combines response time and the number of active connections. (Mostly used in commercial version NGINX PLus)
4. IP Hash (based on client IP address) and weighted IP Hash: Distributes requests based on the first three octets of the client IP address.
5. Hash (on specified request characteristics): Distributes requests based on a specified key, for example, client IP address or request URL.
6. Random with Two Choices: Picks two servers at random and forwards the request to the one with the lower number of active connections.

By default, nginx uses the Round robin approach, which we have experienced in the previous tasks.

Now let us work with the weighted load balancing technique:
Weighted round-robin configuration is accomplished by appending a weight value to the end of each entry in the server group section of the NGINX site configuration file. Set the weight of your slowest server to 1, and then set the weight of other servers relative to that setting. One server receives twice the amount of traffic, while the other server receives four times the amount.

- Now let us modify `sudo nano /etc/nginx/sites-enabled/default` by assigning weights to each application server. Assign weights to each server in the **upstream**. For example `server server1.example.com weight=1;` and `server server2.example.com weight=2;`
- Reload the nginx server.
- Move on to locust web UI, Click on **New Test** and start swarming.
- Wait for the load test to complete and answer the question
    - Question 2: How much percent of your generated user requests ended up visiting each of your servers and why?
    - Take a screenshot of the load balancer with running application server 1
    - Take a screenshot of the load balancer with running application server 2

Exercise 6.5. Comparing the response time w.r.t different load-balancing techniques.

The flask web application used in previous tasks does not perform any compute or memory-intensive operations. We will modify the application by adding a compute-intensive task and checking the response time of user requests w.r.t various load-balancing techniques.

Task 6.5.1: Modify the web application

We will add code to calculate the value Pi for 1000 decimal points.

- Add some Pi calculation code to the Python application on BOTH application servers
- You can find a Python implementation for computing PI from here:
    - https://levelup.gitconnected.com/generating-the-value-of-pi-to-a-known-number-of-decimals-places-in-python-e93986bb474d (Scroll to Writing Chudnovsky algorithm in python codes)
    - Add the Pi calculation code as the method `def compute_pi(n):` in app.py and call it in the request handling part (the *home()* function) with compute_pi(1000). Also update the index.html template file so that it outputs the calculated value of Pi as well (you need to pass the value to the template as an argument to the *render_template()* at the end of *home()*) .
    - As a result, your application should now also print out a very long number which represents the first X digits of Pi.

Task 6.5.2: Configure load balancer with least_conn technique

- Configure Nginx to use the `least_conn;` load balancing strategy.
- Restart the locust from your machine terminal.
- Start the users swarming in locust Web UI and track how many user requests end up in **application server 1** vs **application server 2**

- Check the CPU utilization using `top` command on both application server instances while the requests are being sent to the instances and answer the following question.
    - Question 3: What is the CPU load value on both instances? What does this indicate in regard to the performance of these cloud instance types?

Question 4: When using the `least_conn;` load balancing strategy, what percent of user requests are sent to the first and second application servers?

Question 5: Which of these instances is more capable in terms of responding to user requests? Download the test report from locust UI and name it as "Task6_5_least_conn".

Task 6.5.3: Try with Round robin and weight-based load balancing techniques.

- Configure Round robin technique (basically remove least_conn from the default configuration file and reload nginx)
- Generate the load by click on **New Test**--> Start Swarming
- Wait until the completion of the load test.
- Download the test report from locust UI and name it as Task6_5_round_robin.html for the deliverable.
- Repeat the same procedure for the weight-based technique.
- Download the test report from locust UI and name it as Task6_5_weight_based.html for the deliverable.

Finally, you should have three reports (least_conn(from task 6.5.2), round-robin, and weight based) and answer the following question

Question 6: Which technique has the overall minimum, maximum, and moderate response time? why is it so?

Deliverables

- DO NOT leave your instances running after submitting the solution. This lab requires quite a few VM CPU cores per student and may mean some students can not start lab exercises until previous students shut down their instances.
- Submit deliverables from the exercise:
    1. **Exercise 6.3**:- Answer to Q1, Task6_3_report.html
    2. **Exercise 6.4**:- Answer to Q2, 2 screenshots
    3. **Exercise 6.5** (Task 6.5.2):- Answer to Q3,Q4,Q5 and report Task6_5_least_conn.html
    4. **Exercise 6.5** (Task 6.5.3):- 2 reports, Answer to Q6.

Task: Lab 6 - nginx

Current submission: ZIP

(16:51 17.03.2023)

If your homework consists of multiple files (for example HTML + CSS + JS) it should be archived before submitting.

File: Choose File | No file chosen

Comment:

Submit

In case of issues, check these potential solutions to common issues:

1. If you get errors about permissions when running command line commands inside the instance:
    - In several exercises you are asked to modify, delete or edit files outside your home folder.
    - You will have to use sudo command to elevate file manipulation command permissions to be able to complete such operations.
    - For example: `sudo nano /etc/nginx/sites-enabled/default` will run `nano` command under root user with elevated permissions.
    - NB! But be careful, not everything should be run through sudo!

The courses of the Institute of Computer Science are supported by following programs:

European Union
European Social Fund

Investing
in your future