DevOps Home Task

Version 1.0

Produced by:

Bhavesh Raj

Munich

Revision History

Date	Version	Description	Author
30th SEP 2020	1.0	Initial Draft	Bhavesh Raj

adnymics	Version: 1.0
Task Perform	Version Date: 30th Sep 2020

Task 1a:

Using one-liner bash command, create a histogram of relative volume per hour with any 24 span from:
https://poloniex.com/public?ommand=returnTradeHistory&currencyPair=BTC_ETH&start=1580515200&end=1583020800

Solution:

The given URL is not correct there was **"C"** letter is missing after public?command
https://poloniex.com/public?command=returnTradeHistory&currencyPair=BTC_ETH&start=1580515200&end=1583020800

one-liner bash command to create Histogram

Observation: From given url unable to create the histogram of relative volume but I am sharing one liner command to display histogram connection.

```
netstat -an | grep ESTABLISHED | awk '\''{print $5}'\'' | awk -F: '\''{print $1}'\'' | sort | uniq -c | awk '\''{ printf("%s\t%s\t",$2,$1); for (i = 0; i < $1; i++) {printf("*")}; print ""}'\''
```

Other Solution:

Using `maphimbu` from the *Debian* stda package:

```
# use 'jot' to generate 100 random numbers between 1 and 5
# and 'maphimbu' to print sorted "histogram":
jot -r 100 1 5 | maphimbu -s 1
```

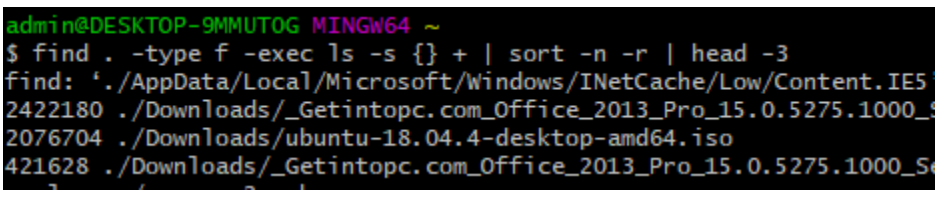
Output:

1	20
2	21
3	20
4	21
5	18

Task 1b:

One-liner command to find top 3 biggest files in the file system

$ find . -type f -exec ls -s {} + | sort -n -r | head -3



Task 2:

Docker ubuntu-nginx-keepalived Demo

GitHub repo:
https://github.com/rajbhavesh7/Nginx-Active-Passive-Keepalived

A small demo to run two ubuntu containers with nginx webserver service in active-passive mode by using keepalived. This demo is made to simulate a scenario with two running nginx servers, one as a master and another as a backup, in order to achieve high availability.

````
## Run command

```
docker-compose up -d
```

Now, visit `localhost:8000` and you would see `hello I'm master!`.

```
$ docker ps -a
CONTAINER ID        IMAGE                                          COMMAND
              CREATED             STATUS                                 PORTS
                  NAMES
3c3cf32e89a5        ubuntu-nginx-keepalived_keepalived_backup   "sh /
entrypoint.sh"      7 seconds ago       Up 5 seconds
0.0.0.0:8881->80/tcp     ubuntu-nginx-keepalived_keepalived_backup_1
b3eb005c3ebd        haproxy:1.7-alpine                             "/docker-
entrypoint.…"   7 seconds ago       Up 5 seconds                           0.0.
0.0:8000->6301/tcp   ubuntu-nginx-keepalived_proxy_1
23c5f89e6a82        ubuntu-nginx-keepalived_keepalived_master   "sh /
entrypoint.sh"      7 seconds ago       Up 5 seconds
0.0.0.0:8880->80/tcp     ubuntu-nginx-keepalived_keepalived_master_1
```

Run pause command to pause master:
```
$ docker pause ubuntu-nginx-keepalived_keepalived_master_1
```

Now, visit `localhost:8000` and you should see `hello I'm backup!`.
````

```
You can unpause master again to bring it back,
```
```
$ docker unpause ubuntu-nginx-keepalived_keepalived_master_1
```

Visit `localhost:8000` and you should see `hello I'm master!` again.

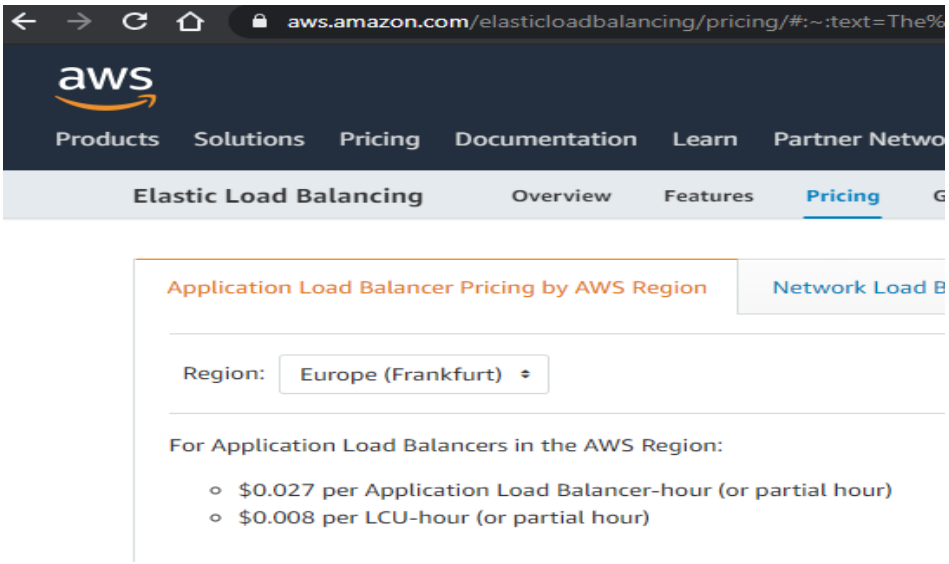
Task 3:

LCU capacity and the monthly cost for operating an AWS Application Load balancer in Frankfurt region.

- 26.6 new connections / s
- 27 active connections / s (each connection lasting 50 ms)
- processed 15000000 Bytes per minute

Step 1:

First of all I have visited amazon website to find the LCU price calculation for Frankfurt region.



Step 2:

An LCU measures the dimensions on which the Application Load Balancer processes your traffic (averaged over an hour). The four dimensions measured are:

- **New connections:** Number of newly established connections per second. Typically, many requests are sent per connection.
- **Active connections:** Number of active connections per minute.
- **Processed bytes:** The number of bytes processed by the load balancer in Gigabytes (GB) for HTTP(S) requests and responses.

- **Rule evaluations:** It is the product of number of rules processed by your load balancer and the request rate. The first 10 processed rules are free (Rule evaluations = Request rate * (Number of rules processed - 10 free rules)

An LCU Contains:

- 25 new connections per second.
- 3,000 active connections per minute.
- 1 GB per hour for EC2 instances, containers and IP addresses as targets and 0.4 GB per hour for Lambda functions as targets.
- 1,000 rule evaluations per second.

Step 3:

Calculation the cost optimization price from given data points.

1. New Connection = (26.6 connection per second / 25 connections per second) = **1.064**
2. Active connection = (27 active connections ms / 3,000 active connections per minute) = **0.0225** (Note converted the given value from mili second to per minutes)
3. Processed Bytes (GBs per hour) = **0.9** GB/hr (converted given value from bytes per min to per hour)

Step 3:

Using these values, the hourly bill is calculated by taking the maximum LCUs consumed across the four dimensions. In this example, the processed bytes dimension (1.064 LCUs) is greater than new connections (0.0225 LCUs), active connections (0.0225 LCUs), and rule evaluations (0.25 LCU) resulting in a total charge of $0.00864 per hour (1.064 LCUs * $0.008 per LCU) or $6.12 per month ($0.008512 * 24 hours * 30 days).

Step 4:

Adding the hourly charge of $0.027, the total Application Load Balancer costs are:

$0.35512 per hour ($0.027 hourly charge + $0.008512 LCU charge);

Or

$25.56 per month ($0.35512 * 24 hours * 30 days).

Task 4:

Launch 3 ec2, t2.small instances with a tag: “env=task”, all of the instances should also have security group “ssh” assigned to them where access on port 2222 is allows to all ip’s.

Step 1:

GitHub repo:

https://github.com/rajbhavesh7/ec2-t2.small-Instances

Step 2:

Created terraform reusable module to spin 3 t2.small ec2 instance on aws provider.

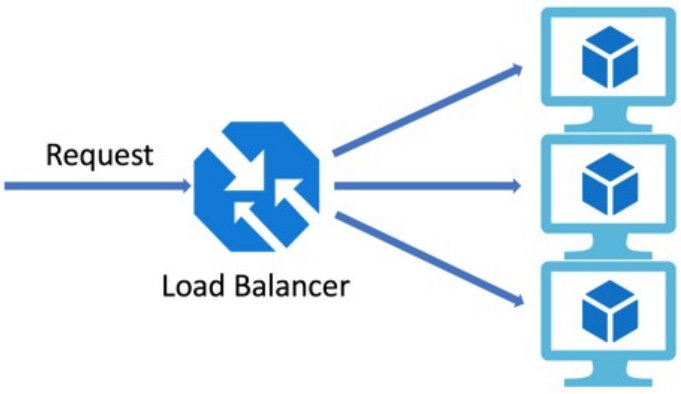
Task 5: Scale Monolithic Application

Step 1:

There are multiple way to scale Monolithic Application which is running in the cloud. `Load Balancer` and `Traffic Manger` are used to distribute network traffic load between multiple instances so that no one will fail.

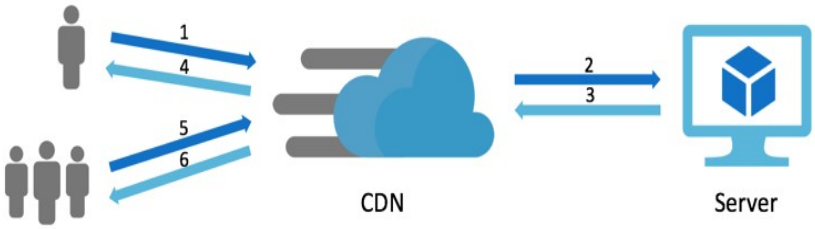
We can use multiple algorithms, these are some of them:

- **Round Robin**: rotating sequential manner.
- **Least Connection**: the current request goes to the server that is servicing the least number of active sessions at the current time.
- **Chained Failover**: redirects to the next server only if the previous one cannot accept any further requests.
- **Weighted Response Time**: redirects to the server with the current fastest response.



Content Delivery Network (CDN)

CDN is used to reduce the latency for getting static content from the server to the user's location. This latency is mainly caused by 2 reasons. The first reason is the physical distance between the user and the server. CDNs are located in multiple locations called Point-of-Presence (POP) around the world. So that it is possible to get one that is closer to the user location than your servers are. The second is accessing the file on the disk, so CDN might use a combination of HDD, SSD or even RAM to cache these data, depending on the frequency of data access frequency. A time-to-live (TTL) can be applied to the cache to say it should expire at a certain time. CDN caches static content files. But, what if you need to cache some dynamic data? This can be solved by using Cache.



Cache

When lots of SQL requests to the database gives the same result, then it is better to cache this data in memory to ensure faster data access and reduce the load on the database. The typical case is the top 10 products displayed on the home page for all users. Because it uses RAM memory and not disks, it can save as much data as the RAM do. Data is stored as key-value pairs. Cache can be distributed across multiple regions.

Elastic search

Elasticsearch can be used to store a predefined search queries and their results. Because it saves data in-memory, it is so much faster than querying the data from database. It also can provide a near real-time search-as-you-type suggestions. This reduces the load on the database, which itself reduces the load on the server as the request will be processed in less time.
ElasticSearch is accessible via REST API and requires changing the app's source code to use it.
Elasticsearch could also be used for log analytics and real-time application monitoring.

When creating multiple copies of the server hosting the whole monolith web application is still not enough or not efficient to handle all users, then let's think about splitting the application into two parts: web app and web API.

Separate Web App/frontend and Web API/backend

Monolith web apps are typically composed of 2 tiers: Web API and frontend web app. This is the case for ASP.NET MVC apps where the views and business logic live together in one server.

In this case, the server will not only process the request to get data to the user but it also renders web pages to generate HTML content. The second role could be done by the client using SPA (Single Page Application) approach. So, the Views part of the app will be moved on a separate server. Consequently, two servers instead of one are now serving users.

Task 6:

Python script that processes the files from here (you can manually extract everything in advance) and count how many times the term “bitcoin” was searched in the English Wikipedia version.

Step 1:

GitHub repo:

https://github.com/rajbhavesh7/Bitcoin

Step 2:

Created python script to count how many times the term bitcoin was searched.