

Name: Jeremy Bui  
Course: DS 561

## Homework 8

### Setup

- Create two new VMs along with their startup scripts and permissions
  - gcloud compute instances set-service-account webserver-vm-a  
--zone=us-central1-a  
--service-account=[844994497823-compute@developer.gserviceaccount.com](#) --scopes=https://www.googleapis.com/auth/cloud-platform
  - gcloud compute instances set-service-account webserver-vm-b  
--zone=us-central1-b  
--service-account=[844994497823-compute@developer.gserviceaccount.com](#) --scopes=https://www.googleapis.com/auth/cloud-platform
  - `#!/bin/bash`  
`sudo apt-get update`  
`sudo apt-get install -y python3`  
`cd /home/jbui/`  
`source myenv/bin/activate`  
`pip install google-cloud-storage google-cloud-logging`  
`google-cloud-pubsub requests`  
`chmod +x webserver.py`  
`nohup python3 webserver.py &`

#### VM instances

Filter Enter property name or value						
<input type="checkbox"/> Status	Name ↑	Zone	Recommendations	In use by	Internal IP	External IP
<input type="checkbox"/>	<a href="#">httpclient-vm</a>	us-central1-b			10.128.0.60 ( <a href="#">nic0</a> )	<a href="#">34.16.56.39</a> ( <a href="#">nic0</a> )
<input type="checkbox"/>	<a href="#">webserver-vm</a>	us-central1-a			10.128.0.59 ( <a href="#">nic0</a> )	<a href="#">34.71.30.141</a> ( <a href="#">nic0</a> )

#### Related actions

- Create healthchecker

**Name**

hw8-healthcheck

**Description**

hw8 health checker

**Region**

us-central1 (Iowa)

**Protocol**

TCP

**Port \***

8080

**Proxy protocol**

NONE

Request

Response

**Logs**

☒ On

Turning on Health check logs can increase costs in Logging.

☐ Off

**Health criteria**

Define how health is determined: how often to check, how long to wait for a response

SAVE

CANCEL

[EQUIVALENT REST](#)

- Create instance group for each VM

Google Cloud

ds561-bucs

Search (/) for resources, docs, products, and

←

Create Instance Group

New managed instance group (stateless)

Automatically manage groups of VMs that do stateless serving and batch processing.

New managed instance group (stateful)

Automatically manage groups of VMs that have persistent data or configurations (such as databases or legacy applications).

New unmanaged instance group

Manually manage groups of load balancing VMs.

Set up a group of load balancing VMs. [Learn more](#)

Name \*

webserver-group-a

Name is permanent

Description

Location

Region \*

us-central1 (Iowa)

Zone \*

us-central1-a

Network and instances

Select instances that reside in a single zone, VPC network, and subnet.

Network \*

default

Subnetwork \*

default

VM instances

webserver-vm-a

Select VMs

webserver-vm-a

Port mapping

To send traffic to instance group through a named port, create a named port to map the incoming traffic to a specific port number, then go to "Load Balancing" to create a load balancer using this instance group.

CREATE

CANCEL

EQUIVALENT CODE

Google Cloud

ds561-bucs

Search (/) for resources, docs, products, and more

←

Create Instance Group

New managed instance group (stateless)

Automatically manage groups of VMs that do stateless serving and batch processing.

New managed instance group (stateful)

Automatically manage groups of VMs that have persistent data or configurations (such as databases or legacy applications).

New unmanaged instance group

Manually manage groups of load balancing VMs.

Set up a group of load balancing VMs. [Learn more](#)

Name \*

webserver-group-b

Name is permanent

Description

Location

Region \*

us-central1 (Iowa)

Zone \*

us-central1-b

Network and instances

Select instances that reside in a single zone, VPC network, and subnet.

Network \*

default

Subnetwork \*

default

VM instances

webserver-vm-b

Select VMs

webserver-vm-b

Port mapping

To send traffic to instance group through a named port, create a named port to map the incoming traffic to a specific port number, then go to "Load Balancing" to create a load balancer using this instance group.

CREATE

CANCEL

EQUIVALENT CODE

- Create a network load balancer backend with both instance groups each

Load Balancer name \*  
hw8  
Lowercase, no spaces.  
Name is permanent

Region \*  
us-central1 (Iowa)

✓ Backend configuration

• Frontend configuration

ⓘ Review and finalize (optional)

Name  
hw8

Description

Backend type  
Instance group

Protocol  
TCP

Backends

✓ web-server-group us-central1-a

✓ http-client-group us-central1-b (Not saved)

ADD A BACKEND

CREATE CANCEL EQUIVALENT CODE

← Edit external passthrough Network Load Balancer

Load Balancer name  
hw8

Region  
us-central1

✓ Backend configuration

✓ Frontend configuration

ⓘ Review and finalize (optional)

webserver-group-b us-central1-b

ADD A BACKEND

Health check \*  
hw8-healthcheck  
region: us-central1, port: 8080, timeout: 5s, check interval: 5s, unhealthy threshold: 2 attempts

ⓘ The health check probes to your load balancer backends come from health check probe IP address ranges. Ensure you have configured ingress firewall rules that permit traffic from these ranges. [Learn more](#)

Session affinity  
None

Logging  
☒ Enable logging

Sample rate \*  
1

Optional fields  
☒ Exclude all optional fields  
☐ Include all optional fields  
☐ Custom

UPDATE CANCEL EQUIVALENT CODE

- Create a network load balancer front end to go to port 80

← Create external passthrough Network Load Balancer

! Make sure all fields are correct to continue

Load Balancer name \*  
hw8  
Lowercase, no spaces.  
Name is permanent

Region \*  
us-central1 (Iowa)  
?

✓ Backend configuration

✓ Frontend configuration

i Review and finalize (optional)

Frontend configuration

Specify an IP address, port and protocol. This IP address is the frontend IP for your clients requests.

^ New Frontend IP and port

Name (Optional)  
hw8-frontend  
Lowercase, no spaces.  
Name is permanent

Description

Protocol  
TCP

IP version  
IPv4

Network Service Tier ?  
☒ Premium ?

CREATE

CANCEL

EQUIVALENT CODE

← Edit external passthrough Network Load Balancer

← previous page

Load Balancer name  
hw8

Region  
us-central1

✓ Backend configuration

✓ Frontend configuration

i Review and finalize (optional)

^ Edit Frontend IP and port

Name  
webserver-frontend

Description  
-

Protocol  
TCP

IP version  
IPv4

IP address  
34.172.19.216

Ports  
8080

DONE

UPDATE

CANCEL

EQUIVALENT CODE

## Code Explanation:

### webserver.py

```
from http.server import BaseHTTPRequestHandler, HTTPServer
from google.cloud import storage, logging as cloudLogging, pubsub_v1
import json
import requests

# Google Cloud Logging setup
cloudLoggingClient = cloudLogging.Client()
cloudLogger = cloudLoggingClient.logger("httpServerLog")

# Configuration
bannedCountries = ['North Korea', 'Iran', 'Cuba', 'Myanmar', 'Iraq',
                   'Libya', 'Sudan', 'Zimbabwe', 'Syria']
projectId = 'ds561-bucs'
topicId = 'banned-country-topic'

def notifyTrackerApp(country):
    publisher = pubsub_v1.PublisherClient()
    topicPath = publisher.topic_path(projectId, topicId)
    messageData = json.dumps({'country': country}).encode('utf-8')
    publisher.publish(topicPath, messageData)

def fetchFileFromBucket(bucketPath):
    bucketName, filePath = bucketPath.split("/", 1)
    bucket = storage.Client().get_bucket(bucketName)
    blob = bucket.blob(filePath)
    if blob.exists():
        return blob.download_as_text()
    return None

def get_zone():
    metadata_url =
"http://metadata.google.internal/computeMetadata/v1/instance/zone"
    headers = {"Metadata-Flavor": "Google"}
    response = requests.get(metadata_url, headers=headers)
    if response.status_code == 200:
        return response.text.split('/')[ -1]
    return "unknown"
```

```

class RequestHandler(BaseHTTPRequestHandler):
    def log_message(self, format, *args):
        super().log_message(format, *args)

    def do_GET(self):
        country = self.headers.get('X-Country', 'unknown')
        if country in bannedCountries:
            cloudLogger.log_text("Access denied from banned country: " +
country, severity="ERROR")
            notifyTrackerApp(country)
            self.response_code = 400
            self.send_response(400)
            self.end_headers()
            self.wfile.write(b"Access denied: Request from banned
country")
            return

        try:
            fileContent = fetchFileFromBucket(self.path.strip("/"))
            if fileContent is None:
                cloudLogger.log_text("File " + self.path + " not found",
severity="ERROR")
                self.response_code = 404
                self.send_response(404)
                self.end_headers()
                self.wfile.write(b"File not found")
            else:
                zone = get_zone()
                self.response_code = 200
                self.send_response(200)
                self.send_header("X-Instance-Zone", zone)
                self.end_headers()
                self.wfile.write(fileContent.encode("utf-8"))
        except BrokenPipeError:
            cloudLogger.log_text("The server has timed out!",
severity="ERROR")
            print("The server has timed out!")
        except Exception as e:
            cloudLogger.log_text("Error occurred: " + str(e),
severity="ERROR")

```

```

        self.response_code = 500
        self.send_response(500)
        self.end_headers()

    def handle_unsupported_methods(self):
        cloudLogger.log_text("Unsupported method " + self.command + " was
used for " + self.path, severity="ERROR")
        self.response_code = 501
        self.send_response(501)
        self.end_headers()
        self.wfile.write((self.command + " not
implemented").encode("utf-8"))

    def do_POST(self):
        self.handle_unsupported_methods()

    def do_PUT(self):
        self.handle_unsupported_methods()

    def do_DELETE(self):
        self.handle_unsupported_methods()

    def do_HEAD(self):
        self.handle_unsupported_methods()

    def do_CONNECT(self):
        self.handle_unsupported_methods()

    def do_OPTIONS(self):
        self.handle_unsupported_methods()

    def do_TRACE(self):
        self.handle_unsupported_methods()

    def do_PATCH(self):
        self.handle_unsupported_methods()

def runServer(serverAddress, port):
    server = HTTPServer((serverAddress, port), RequestHandler)
    print(f"Server running on {serverAddress}:{port}...")

```



```

try:
    server.serve_forever()
except KeyboardInterrupt:
    print("Server stopped by user.")
finally:
    server.server_close()
    print("Server has been cleaned up.")

if __name__ == "__main__":
    runServer("0.0.0.0", 8080)

```

- Import google cloud dependencies (storage, logging, pubsub)
- Import request to allow for http requests and retrieval of zone metadata
- Import datetime for logging to keep track of when each request is handled
- Set up logging with cloudLogger and cloudLoggingClient to log events to google cloud
- Keep function fetchFileFromBucket to retrieve a file from google cloud storage
- Implement function get\_zone to get the google cloud zone by calling the google metadata server
  - The metadata url is "http://metadata.google.internal/computeMetadata/v1/instance/zone"
  - Each request will include a metadata-flavor header (needed by metadata server)
  - We can then parse the data we get back to indicate which instance zone is handling the request
- Keep do\_Get and log\_message to continue handling requests based on X-Country header and other requirements from past homework
- Update the try section inside do\_Get to get the zone and add it as a response
- Update runserver to initialize the web-server to run until it is closed/ended

### Modified section of http-client.py:

```

def make_request(domain, port, country, ip, filename, use_ssl,
ssl_context, follow, verbose, timeout):
    try:
        if verbose:
            print("Requesting ", filename, " from ", domain, port)

```

```

        conn = http.client.HTTPSConnection(domain, port,
context=ssl_context, timeout=timeout) if use_ssl else
http.client.HTTPConnection(domain, port, timeout=timeout)
        headers = build_headers(country, ip)
        conn.request("GET", filename, headers=headers)

    res = conn.getresponse()
    data = res.read()

    if verbose:
        print(res.status, res.reason)
        print(data)

    # Print the zone header if it exists
    zone_header = res.getheader("X-Instance-Zone")
    if zone_header:
        current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(f"[{current_time}] X-Instance-Zone:", zone_header)

    if follow:
        location_header = res.getheader('location')
        if location_header is not None:
            filename = urljoin(filename, location_header)
            make_request(domain, port, country, ip, filename, use_ssl,
ssl_context, follow, verbose, timeout)

    # It will take some times to reroute to VM
    except RemoteDisconnected:
        current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(f"[{current_time}] Warning: Remote server disconnected.
Retrying...")
        time.sleep(1)

    except (ConnectionResetError, ConnectionRefusedError, socket.timeout)
as e:
        current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        print(f"[{current_time}] Connection error: {e}. Retrying...")
        time.sleep(1)

```

```

finally:
    conn.close()
def print_zone_ratios():
    global zone_counter
    total_requests = sum(zone_counter.values())
    if total_requests > 0:
        print("\nZone Request Ratios:")
        for zone, count in zone_counter.items():
            ratio = (count / total_requests) * 100
            print(f"{zone}: {ratio:.2f}% ({count} requests)")
    else:
        print("\nNo requests have been processed yet.")

```

- Retrieve the X-Instance-Zone header from the response and print it with the timestamp to allow us to verify which zone the request is from
- Add a handling where the server may disconnect due to the load balancer being interrupted with a VM shutting down and needing to reroute requests
- Add counter to get ratio of requests by just adding to counter each time a request header is received

## **Metrics (step 5 and 6)**

- Need to run http-client.py with ip from load balancer: `python3 http-client.py --domain 34.172.19.216 --port 8080 --bucket jeremybui_ps2 --webdir files --num_requests 1000 --index 10000`
- Kill one VM (killed us-central1-b VM) and notice how long it takes to reroute.
- **Step 5 answer: 33 seconds (12:36:56 - 12:36:23)**
  - The VM was killed at 12:36:23 and http-client was modified to buffer while requests could not be served
  - There was a response back from us-central1-a again at 12:36:56 and all of the following requests were from a, showing that requests were rerouted

```

[2024-11-18 12:36:20] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:21] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:21] X-Instance-Zone: us-central1-b
[2024-11-18 12:36:22] X-Instance-Zone: us-central1-b
[2024-11-18 12:36:22] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:22] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:23] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:23] X-Instance-Zone: us-central1-b
[2024-11-18 12:36:23] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:56] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:57] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:57] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:57] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:58] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:58] X-Instance-Zone: us-central1-a
[2024-11-18 12:36:58] X-Instance-Zone: us-central1-a

```

- Restart the killed VM (us-central1-b) and report how quickly the load balancer notices its presence and starts routing requests to it
- **Step 6 Answer: 27 seconds (13:38:42 - 12:38:15)**
  - Script inside webserver-vm-b successfully started at around 12:38:15
  - Start serving first request inside webserver at 12:38:41
  - Http-client saw response at 12:38:42

```

> [i] 2024-11-18 12:38:15.281 EST startup-script: 0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
> [i] 2024-11-18 12:38:41.830 EST startup-script: 128.197.28.175 - - [18/Nov/2024 17:38:41] "GET /jeremybui_ps2/files/7538.html HTTP/1.1" 200 128.197.28.175

```

```

[2024-11-18 12:38:40] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:41] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:41] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:41] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:42] X-Instance-Zone: us-central1-b
[2024-11-18 12:38:42] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:42] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:43] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:43] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:43] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:44] X-Instance-Zone: us-central1-a
[2024-11-18 12:38:44] X-Instance-Zone: us-central1-a

```

- Ratio of requests between zones

```
[2024-11-18 19:41:14] X-Instance-Zone: us-central1-a
[2024-11-18 19:41:14] X-Instance-Zone: us-central1-b
[2024-11-18 19:41:15] X-Instance-Zone: us-central1-b

Zone Request Ratios:
us-central1-b: 46.95% (446 requests)
us-central1-a: 53.05% (504 requests)
```

Cost

Your total cost (November 12 – 18, 2024) ?

Last 7 days

Current month

Cost

\$4.21

—

Credits used

\$4.21

=

Total cost

\$0.00

[View details](#)

Forecasted total cost

\$0.00

0% vs. prev. 7d

