Internal Load Balancer

I ran the following script for the backend services and used Gemini AI on the cloud shell editor to help me understand it more:

Script Analysis

Backend.sh

The script executes three main commands in sequence to set up and run a Python web server.

1. Setting Directory Permissions

bash

```
sudo chmod -R 777 /usr/local/sbin/
```

- **sudo**: This runs the command with "superuser" or root privileges, which is necessary for modifying system directories.
- **chmod**: This is the "change mode" command, used to alter file and directory permissions.
- **-R 777**: This is where the permission change happens.
    - The -R flag makes the change "recursive," applying it to the /usr/local/sbin directory and every file and subdirectory within it.
    - 777 is an octal representation of permissions. It grants read (4), write (2), and execute (1) permissions to all three user classes: the **owner**, the **group**, and **others**. This is a highly permissive setting.
- **/usr/local/sbin/**: This is the target directory. It's a standard location on Linux systems for locally installed scripts and binaries for system administration.

2. Creating the Python Web Server Script

bash

Show full code block

```
sudo cat << EOF > /usr/local/sbin/serveprimes.py

import http.server


def is_prime(a): return a!=1 and all(a % i for i in range(2,int(a**0.5)+1))


class myHandler(http.server.BaseHTTPRequestHandler):

  def do_GET(s):

    s.send_response(200)
    s.send_header("Content-type", "text/plain")
    s.end_headers()
    s.wfile.write(bytes(str(is_prime(int(s.path[1:]))).encode('utf-8')))


http.server.HTTPServer(("",80),myHandler).serve_forever()
EOF
```

- This command uses a "here document" (<< EOF...EOF) to write a multi-line Python script directly into the file /usr/local/sbin/serveprimes.py. The sudo cat > ... part takes the text block and, as root, writes it to the specified file.
- **The Python Script (serveprimes.py)**:
    - It uses Python's built-in http.server module to create a simple web server. This module is great for basic tasks but isn't recommended for production use.
    - The is_prime(a) function implements a basic algorithm to check if a number is prime.
    - The myHandler class defines how the server responds to incoming GET requests. It extracts the number from the URL path (e.g., the 13 from http://your-ip/13), checks if it's prime, and sends back a plain text response of either "True" or "False".
    - Finally, HTTPServer(("",80),myHandler).serve_forever() starts the server. It listens on port 80 for requests from any IP address.

3. Running the Server in the Background

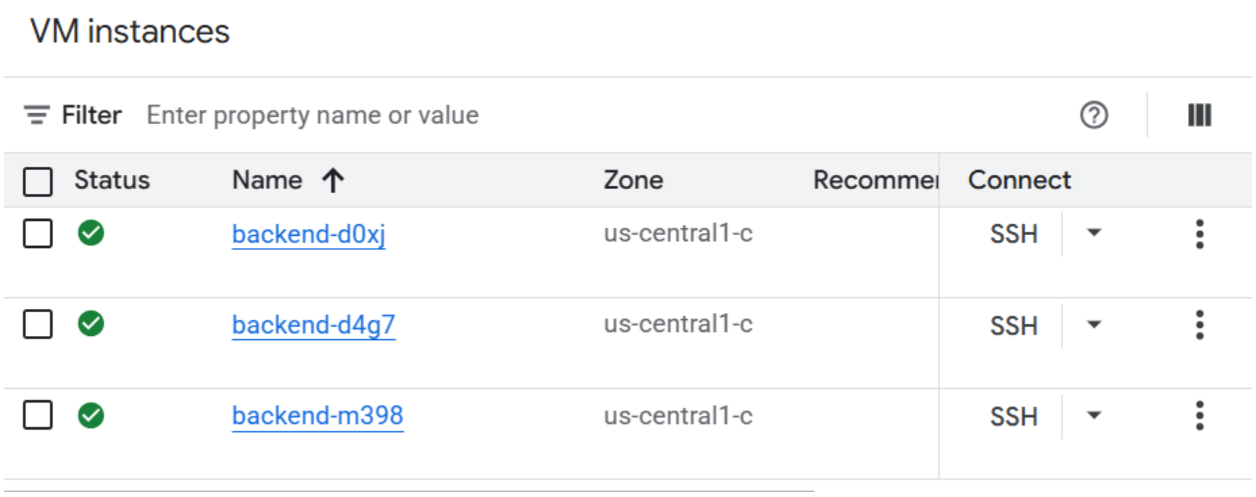
bash

```
nohup python3 /usr/local/sbin/serveprimes.py >/dev/null 2>&1 &
```

This command executes the Python script and ensures it keeps running as a background service.

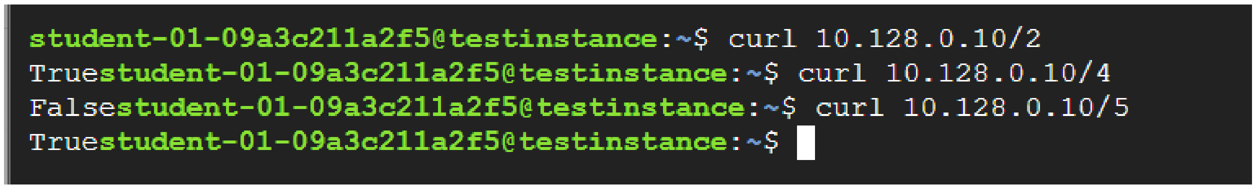
- **nohup**: Short for "no hangup," this command prevents the script from being terminated when the user logs out or closes the terminal. It intercepts the "hangup" signal (SIGHUP) that the shell sends to its child processes upon exit.
- **python3 ...**: This is the command that actually runs our web server script.
- **>/dev/null 2>&1**: This part handles the script's output.
    - >/dev/null redirects the standard output (stdout) to /dev/null, a special file that discards anything written to it.
    - 2>&1 redirects the standard error (stderr) to the same location as the standard output, effectively silencing all logs and error messages from the script.
- **&**: The ampersand at the end tells the shell to run the entire command in the background, freeing up your terminal for other commands immediately.

I created 3 VM instances using the Managed Instance Group feature.(create and maintain identical copies of your script)


VM instances

Status	Name	Zone	Recommendations	Connect
✓	backend-d0xj	us-central1-c		SSH
✓	backend-d4g7	us-central1-c		SSH
✓	backend-m398	us-central1-c		SSH

Backend


```
student-01-09a3c211a2f5@testinstance:~$ curl 10.128.0.10/2
Truestudent-01-09a3c211a2f5@testinstance:~$ curl 10.128.0.10/4
Falsestudent-01-09a3c211a2f5@testinstance:~$ curl 10.128.0.10/5
Truestudent-01-09a3c211a2f5@testinstance:~$
```

Frontend.sh
The startup script is a shell script executed by the Compute Engine VM when it boots up. It consists of three main shell commands to set up and run the Python application.

Shell Command	**Function**
sudo chmod -R 777 /usr/local/sbin/	**Permission Setting**
sudo cat << EOF > /usr/local/sbin/getprimes.py ... EOF	**File Creation**
nohup python3 /usr/local/sbin/getprimes.py >/dev/null 2>&1 &	**Execution & Backgrounding**

🐍 Python Application (getprimes.py) Explained

This Python file creates a simple HTTP server using the built-in http.server library and is designed for high concurrency using threading to interact with our internal services.

1. Imports and Configuration

- **import urllib.request**: Used for making HTTP requests to external/internal URLs—specifically, connecting to our Internal Load Balancer.
- **from multiprocessing.dummy import Pool as ThreadPool**: **Crucial for performance.** This imports the ThreadPool class, which allows the script to make **multiple concurrent (simultaneous) requests** to the internal service, vastly speeding up the process of checking 100 numbers.
- **PREFIX="http://IP/"**: This constant defines the base URL for the **Internal Application Load Balancer**. The web server will send all prime number calculation requests to this address.
- **def get_url(number):**: This function constructs the full URL (e.g., http://IP/17) and uses urllib.request.urlopen() to fetch the result (which is either the string "True" or "False") from the internal service.

2. Request Handler (myHandler Class)

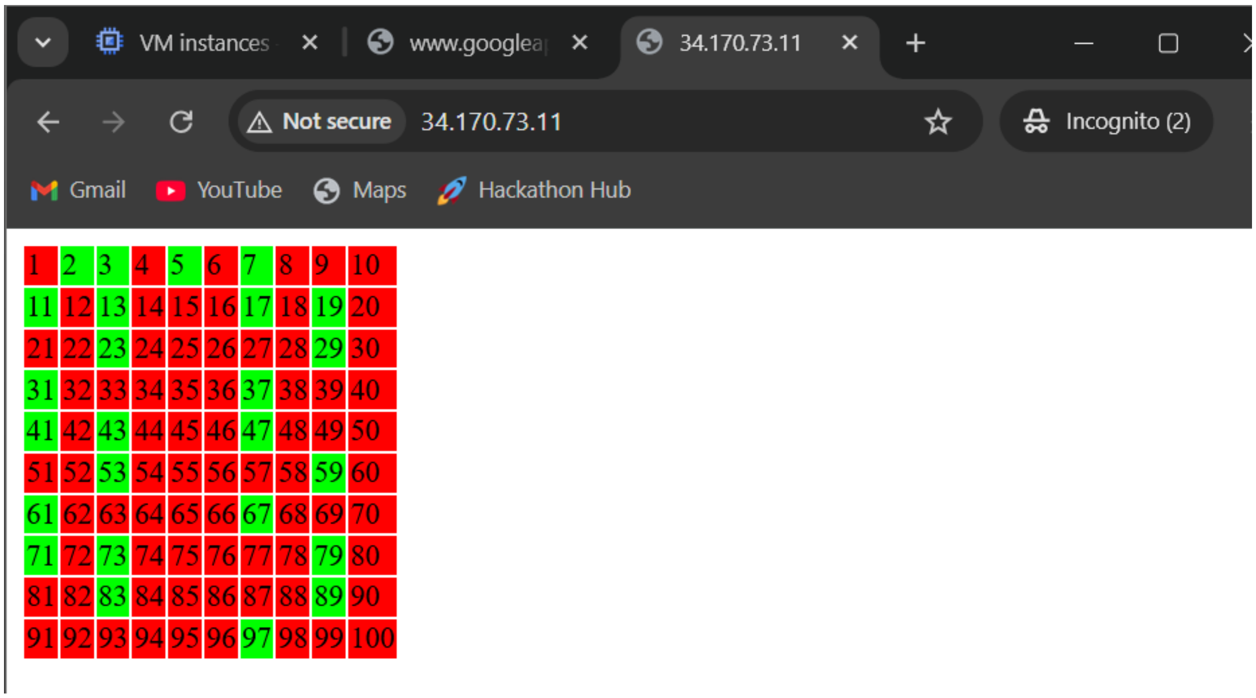
The do_GET(s) method is the core logic that runs every time a public user accesses the web server.

Code Block	**Purpose**
Header Setup	Sets the HTTP response code to 200 (OK) and the content type to text/html, as the output will be a web page.
i = int(s.path[1:]) ... else 1	**Input Parsing:** Extracts the starting number from the URL path (e.g., if the user visits /100, it starts the prime check at 100). Defaults to 1 if no number is provided.
pool = ThreadPool(10)	**Concurrency Setup:** Initializes a thread pool with **10 workers**, meaning it can manage 10 requests to the internal service at the same time.
results = pool.map(get_url,range(i,i+100))	**Parallel Service Call:** This is the most important step. It makes **100 parallel requests** to the Internal Load Balancer, starting from i and going up to $i+99$. The pool.map function collects the results ("True" or "False") efficiently.
HTML/Table Generation	The subsequent for loop iterates through the 100 results and dynamically generates an HTML table: - It inserts <tr> (table row) every 10 iterations to create a 10x10 matrix.

Code Block	Purpose
	- If the result is "True", it sets the cell background to **green** (#00ff00). - If the result is not "True" (i.e., "False"), it sets the cell background to **red** (#ff0000).

I also created a VM instance for the frontend that collects results and formats them into a coloured html table and sends the final page back to the Public User.

Frontend



1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Summary of the Architecture Flow

1. A Public User sends a request to the Frontend VM's External IP on port 80.
2. The Frontend VM receives the request and runs the getprimes.py script.
3. The script initiates 100 concurrent HTTP requests internally.
4. These 100 requests hit the Internal Load Balancer IP (http://10.128.0.10/).
5. The Internal Load Balancer distributes these 100 requests across the Internal Service Tier VMs (the prime number calculators).
6. Each calculator VM returns either "True" or "False".

7. The Frontend VM collects all 100 results, formats them into a colored HTML table, and sends the final page back to the Public User.