

AI Application

Part A.: Creating Application

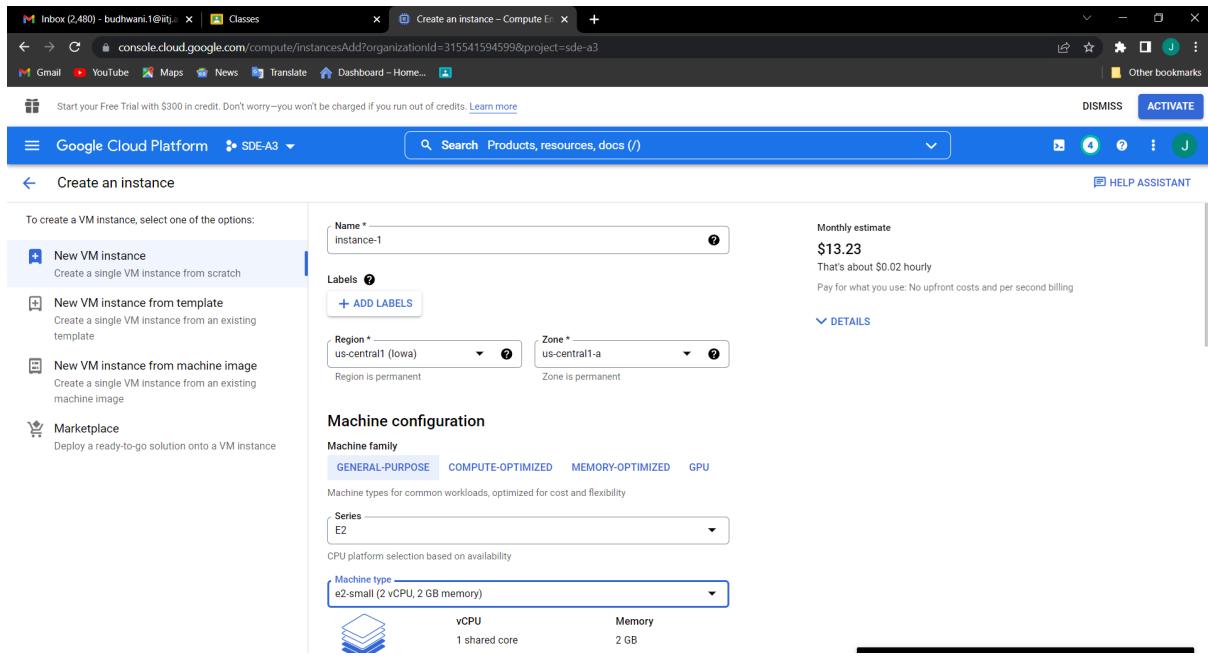
In this Assignment, we were required to create an application using three google cloud services which were **Cloud Compute service, Cloud Storage, and Cloud AI service.**

First, we need to **Create a Project SDE-A3** and then do all the below steps in created project.

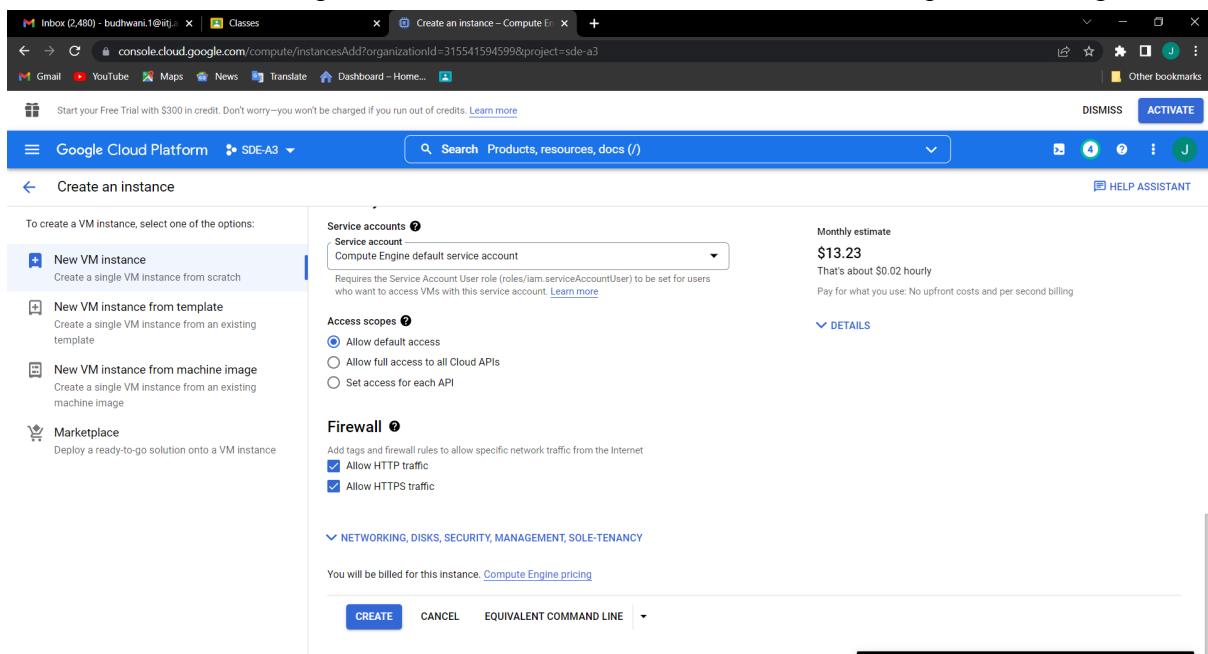
Creating Services:

First Create a **Cloud Compute service** and host our site:

1. To use the compute engine first create an instance of the engine.
2. Go to **compute engine** google cloud -> **create instance** -> give instance name and select appropriate CPUs and memory required for your application.



3. Under the firewall, the option **enables HTTP and HTTPS** to enable requests and responses.



4. Finally, creating the instance will take some time.
5. After the instance is created you will see a screen below.

The screenshot shows the Google Cloud Platform Compute Engine interface. On the left, there's a sidebar with 'Compute Engine' selected under 'Virtual machines'. The main area displays a table of VM instances. One instance, 'm21cs007', is listed with the following details:

Status	Name	Zone	Recommendations	In use by	Internal IP	External IP	Connect
Up	m21cs007	us-central1-a			10.128.0.3 (nic0)	34.135.107.217	SSH

6. After creating the instance connect to it using the ssh connection by any of the options provided below.

The screenshot shows the same Google Cloud Platform Compute Engine interface as above, but with a context menu open over the 'm21cs007' instance row. The menu options are:

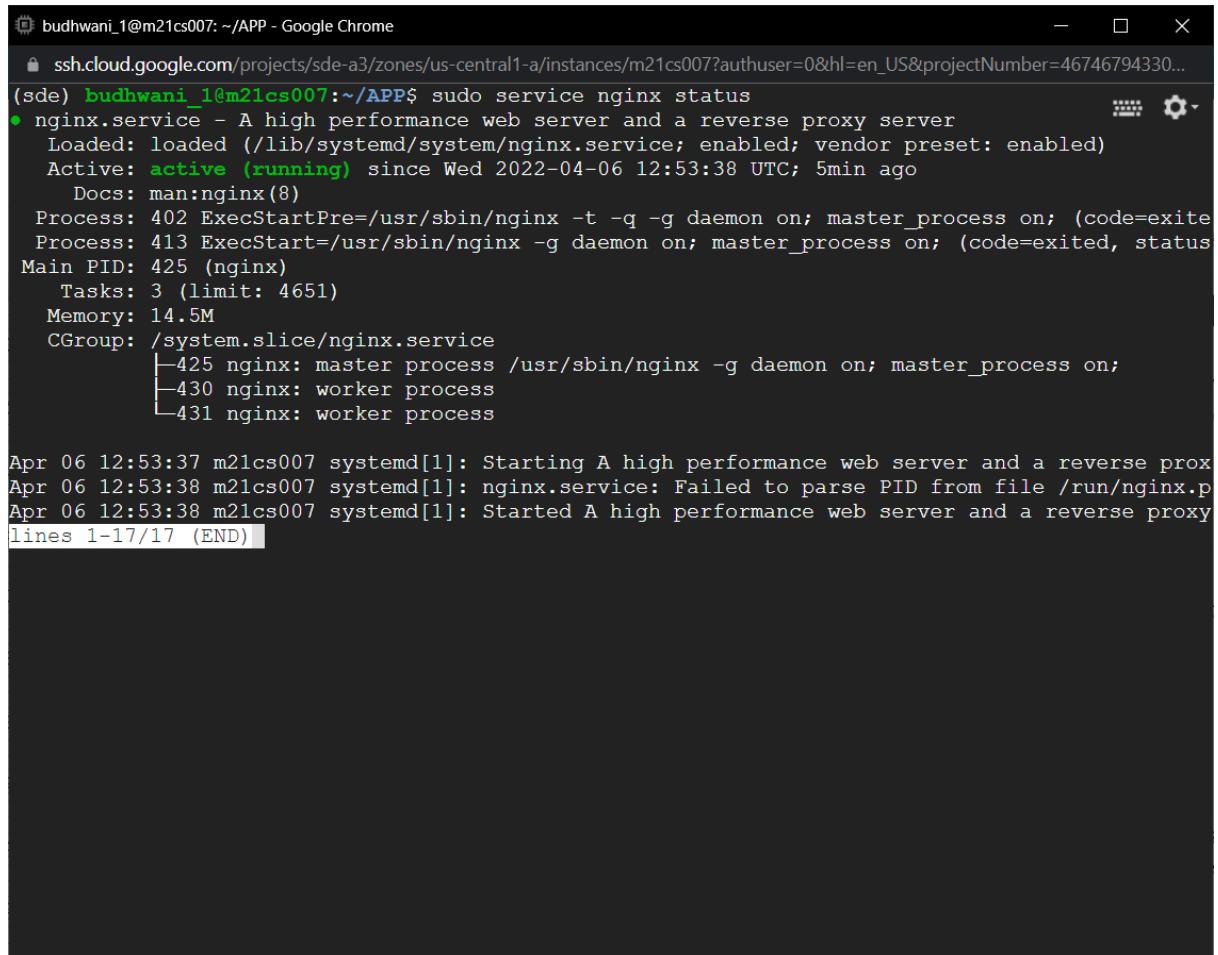
- Open in browser window
- Open in browser window on custom port
- Open in browser window using provided private SSH key
- View gcloud command
- Use another SSH client

7. For this assignment, the open in the browser window is selected After selecting you will see a screen as below.

```
https://ssh.cloud.google.com/v2/ssh/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=4... — □ ×  
ssh.cloud.google.com/v2/ssh/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=46746...  
SSH-IN-BROWSER Terminal  
Linux m21cs007 4.19.0-20-cloud-amd64 #1 SMP Debian 4.19.235-1 (2022-03-17) x86_64  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/*copyright.  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Sat Apr 2 07:19:26 2022 from 35.235.241.16  
budhwani_1@m21cs007:~$
```

8. Now create a Folder named **APP** that contains all of our required libraries and codes.
9. For coding, the **Flask framework** is used and for hosting the Flask application **gunicorn** with **Nginx server** is used.
10. For setting up the environment reference used the [link](#) or Commands used for setting up the environment are:
- Update the shell to recent libraries - **sudo apt-get upgrade && sudo apt-get update**
 - For Installing the latest version of python - **sudo apt-get install Python3**
 - For Installing the latest version of pip - **sudo apt-get install Python Python3-pip**
 - For Installing Nginx - **sudo apt-get install nginx**
 - For starting Nginx - **sudo service nginx start**
 - For checking the status of nginx - **sudo service nginx status**

After the previous steps status of nginx looks like this.

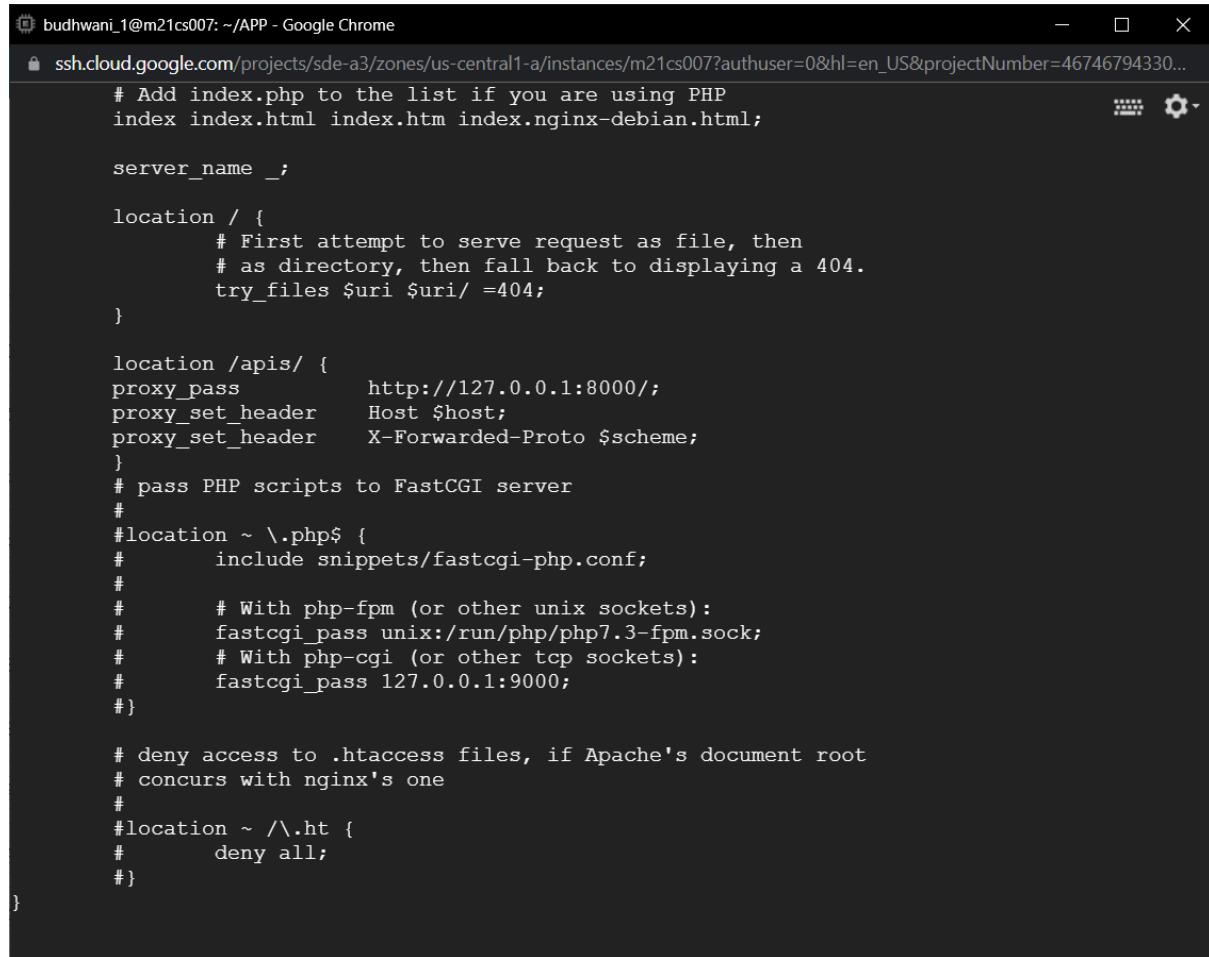


The screenshot shows a terminal window titled "Google Chrome" with the URL "ssh.cloud.google.com/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=46746794330...". The command "sudo service nginx status" is run, displaying the following output:

```
(sde) budhwani_1@m21cs007:~/APP$ sudo service nginx status
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
  Active: active (running) since Wed 2022-04-06 12:53:38 UTC; 5min ago
    Docs: man:nginx(8)
 Process: 402 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exite
 Process: 413 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status
Main PID: 425 (nginx)
  Tasks: 3 (limit: 4651)
 Memory: 14.5M
 CGroup: /system.slice/nginx.service
         └─425 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
           ├─430 nginx: worker process
           ├─431 nginx: worker process
           └─432 nginx: worker process

Apr 06 12:53:37 m21cs007 systemd[1]: Starting A high performance web server and a reverse prox
Apr 06 12:53:38 m21cs007 systemd[1]: nginx.service: Failed to parse PID from file /run/nginx.p
Apr 06 12:53:38 m21cs007 systemd[1]: Started A high performance web server and a reverse proxy
lines 1-17/17 (END)
```

11. Now that we have installed the nginx we need to provide the request points in the conf file which can be found at **cd /etc/nginx/sites-available/default** for the assignment **/apis/** is used and in **/apis/** further bifurcation is made for each operation after making changes we further need to again start the nginx server.



The screenshot shows a terminal window titled "budhwani_1@m21cs007: ~/APP - Google Chrome". The URL in the address bar is "ssh.cloud.google.com/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=46746794330...". The content of the terminal window is the nginx configuration file:

```
# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;

location / {
    # First attempt to serve request as file, then
    # as directory, then fall back to displaying a 404.
    try_files $uri $uri/ =404;
}

location /apis/ {
proxy_pass          http://127.0.0.1:8000/;
proxy_set_header    Host $host;
proxy_set_header    X-Forwarded-Proto $scheme;
}
# pass PHP scripts to FastCGI server
#
#location ~ \.php$ {
#    include snippets/fastcgi-php.conf;
#
#    # With php-fpm (or other unix sockets):
#    fastcgi_pass unix:/run/php/php7.3-fpm.sock;
#    # With php-cgi (or other tcp sockets):
#    fastcgi_pass 127.0.0.1:9000;
#}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny all;
#}

}
```

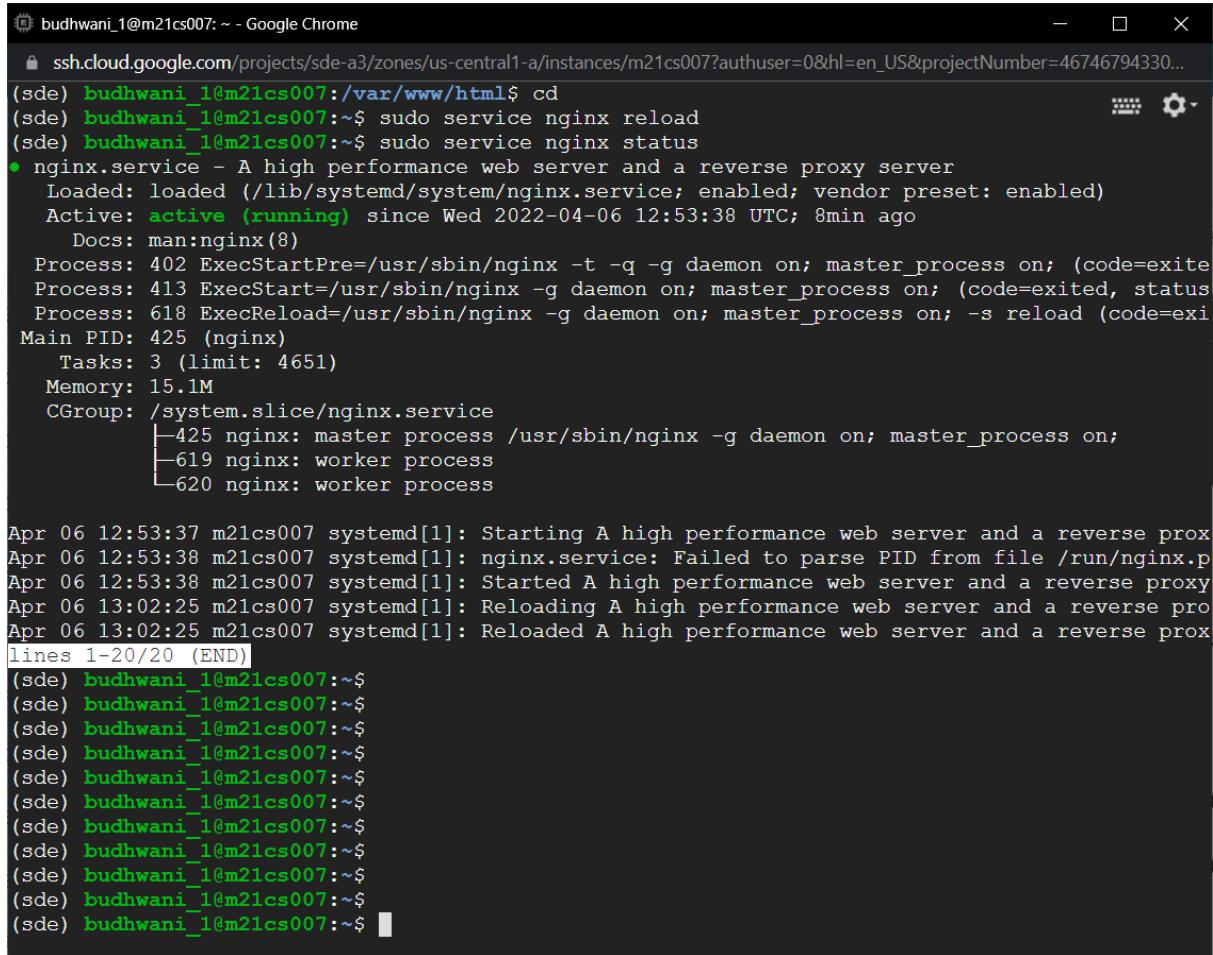
12. After making the changes in the config file we need to make changes in the index.html to create a webpage for our appropriate needs which can be found at `cd /var/www/html/index.html`.

The screenshot shows a terminal window titled "budhwani_1@m21cs007: /var/www/html - Google Chrome". The URL in the address bar is "ssh.cloud.google.com/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=46746794330...". The terminal session shows the following commands and output:

```
#          try_files $uri $uri/ =404;
#
#}
(sde) budhwani_1@m21cs007:~/APP$ cd /var/www/html
(sde) budhwani_1@m21cs007:/var/www/html$ ls
index.nginx-debian.html
(sde) budhwani_1@m21cs007:/var/www/html$ cat index.nginx-debian.html
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>

<body>
<h1><center>SDE 2022</center></h1>
<h2><center>ASSIGNMENT 3 GOOGLE CLOUD APPLICATION</center></h2>
<h3><center>Jayesh Budhwani M21CS007</center></h3>
<p align = "left">
This is a web application that is hosted on <b> Compute engine using Nginx, Flask, and Gunicorn </b>. The application does Sentiment Analysis using<b> Google Cloud AI service named as Natural Language API</b> on user provided data and then stores that data and its analysis on<b> Cloud Mysql server</b> and then shows the result to the user.<br>
The user can also see the complete database.
</p>
</body>
</html>
(sde) budhwani_1@m21cs007:/var/www/html$
```

13. After making the changes in the nginx server we can see our hosted site after again starting our nginx service using **sudo service nginx reload && sudo service nginx status**.

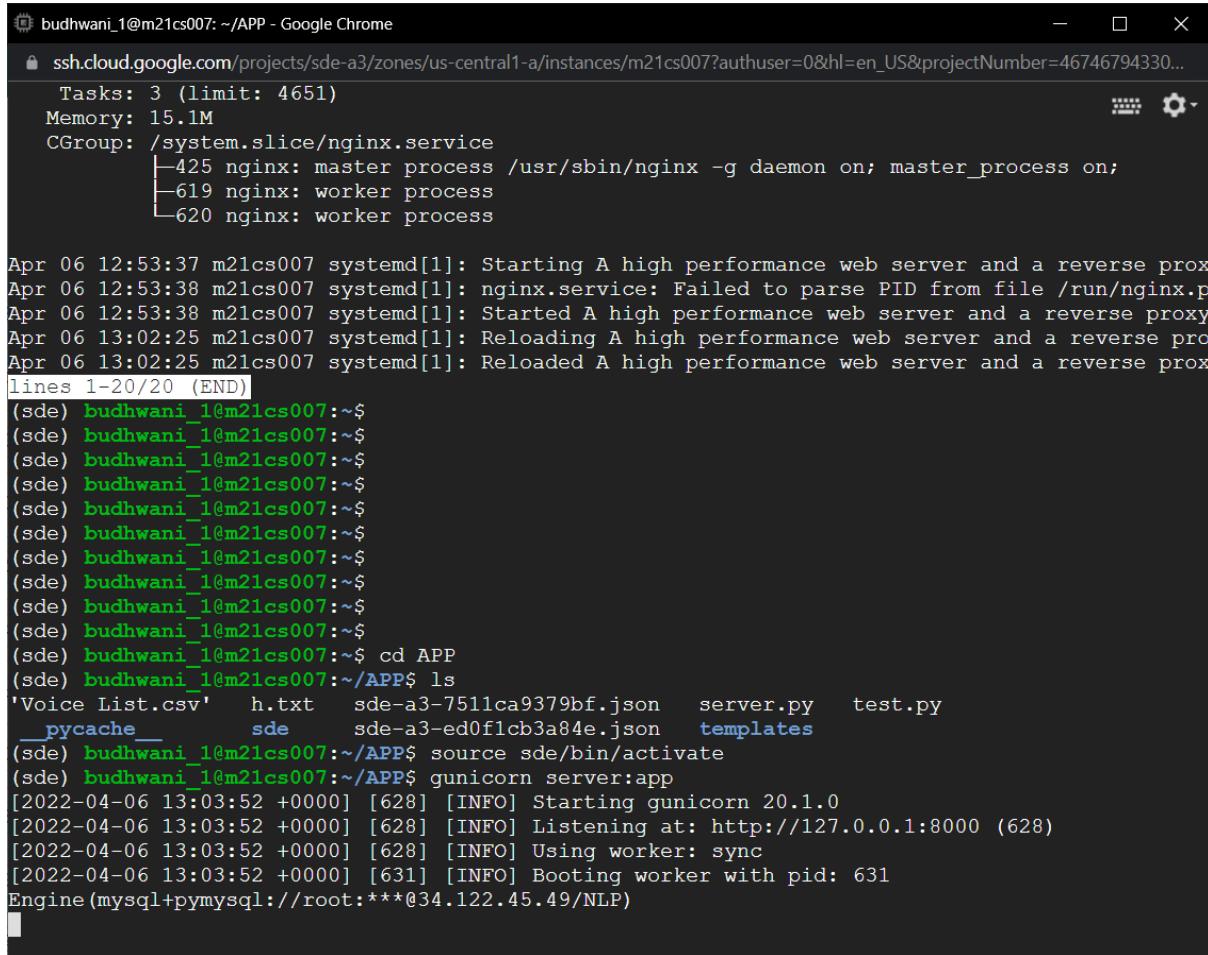


```
budhwani_1@m21cs007: ~ - Google Chrome
ssh.cloud.google.com/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=46746794330...
(sde) budhwani_1@m21cs007:/var/www/html$ cd
(sde) budhwani_1@m21cs007:~$ sudo service nginx reload
(sde) budhwani_1@m21cs007:~$ sudo service nginx status
● nginx.service - A high performance web server and a reverse proxy server
  Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
  Active: active (running) since Wed 2022-04-06 12:53:38 UTC; 8min ago
    Docs: man:nginx(8)
Process: 402 ExecStartPre=/usr/sbin/nginx -t -q -g daemon on; master_process on; (code=exited, status 0)
Process: 413 ExecStart=/usr/sbin/nginx -g daemon on; master_process on; (code=exited, status 0)
Process: 618 ExecReload=/usr/sbin/nginx -g daemon on; master_process on; -s reload (code=exited, status 0)
Main PID: 425 (nginx)
  Tasks: 3 (limit: 4651)
 Memory: 15.1M
 CGroup: /system.slice/nginx.service
        ├─425 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
        ├─619 nginx: worker process
        └─620 nginx: worker process

Apr 06 12:53:37 m21cs007 systemd[1]: Starting A high performance web server and a reverse proxy...
Apr 06 12:53:38 m21cs007 systemd[1]: nginx.service: Failed to parse PID from file /run/nginx.pid: Invalid argument
Apr 06 12:53:38 m21cs007 systemd[1]: Started A high performance web server and a reverse proxy...
Apr 06 13:02:25 m21cs007 systemd[1]: Reloading A high performance web server and a reverse proxy...
Apr 06 13:02:25 m21cs007 systemd[1]: Reloaded A high performance web server and a reverse proxy...
lines 1-20/20 (END)
(sde) budhwani_1@m21cs007:~$
```

14. Now to set up our flask project and environment use commands.

- First, create a virtual environment so that all of our libraries can be in a single place using the command - **python3 -m venv sde** (sde is name of our virtual environment).
- Now enable the virtual environment - **source sde/bin/activate**
- Now install Flask - **pip3 install flask**
- Now install gunicorn - **pip3 install gunicorn**
- Now create a folder named **templates** for our HTML templates and create **server.py** which contains the flask code.
- As we know the flask run locally so we use gunicorn to host this local flask on our compute instance.
- Run gunicorn using command **gunicorn server:app**



The screenshot shows a terminal window titled "budhwani_1@m21cs007: ~/APP - Google Chrome". The URL in the address bar is "ssh.cloud.google.com/projects/sde-a3/zones/us-central1-a/instances/m21cs007?authuser=0&hl=en_US&projectNumber=46746794330...". The terminal output shows the following:

```
Tasks: 3 (limit: 4651)
Memory: 15.1M
CGroup: /system.slice/nginx.service
    └─425 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
      ├─619 nginx: worker process
      └─620 nginx: worker process

Apr 06 12:53:37 m21cs007 systemd[1]: Starting A high performance web server and a reverse proxy...
Apr 06 12:53:38 m21cs007 systemd[1]: nginx.service: Failed to parse PID from file /run/nginx.pid: Invalid argument
Apr 06 12:53:38 m21cs007 systemd[1]: Started A high performance web server and a reverse proxy...
Apr 06 13:02:25 m21cs007 systemd[1]: Reloading A high performance web server and a reverse proxy...
Apr 06 13:02:25 m21cs007 systemd[1]: Reloaded A high performance web server and a reverse proxy...

(sde) budhwani_1@m21cs007:~$ cd APP
(sde) budhwani_1@m21cs007:~/APP$ ls
'Voice List.csv'  h.txt  sde-a3-7511ca9379bf.json  server.py  test.py
__pycache__        sde   sde-a3-ed0f1cb3a84e.json  templates
(sde) budhwani_1@m21cs007:~/APP$ source sde/bin/activate
(sde) budhwani_1@m21cs007:~/APP$ gunicorn server:app
[2022-04-06 13:03:52 +0000] [628] [INFO] Starting gunicorn 20.1.0
[2022-04-06 13:03:52 +0000] [628] [INFO] Listening at: http://127.0.0.1:8000 (628)
[2022-04-06 13:03:52 +0000] [628] [INFO] Using worker: sync
[2022-04-06 13:03:52 +0000] [631] [INFO] Booting worker with pid: 631
Engine(mysql+pymysql://root:***@34.122.45.49/NLP)
```

15. After setting up the gunicorn and flask add the functionalities of AI services and Cloud SQL.

Second, we need to create an AI service for this assignment the Natural Language AI service was used for Sentiment Analysis.

1. Go to Cloud API and Services, then search for Natural Language API.

The screenshot shows the Google Cloud Platform APIs & Services dashboard. The left sidebar lists 'Enabled APIs & services' including Library, Credentials, OAuth consent screen, Domain verification, and Page usage agreements. The main area has tabs for 'Traffic' and 'Median latency'. A search bar at the top right is set to 'Natural Language'. Below it, under 'DOCUMENTATION & TUTORIALS', there are links to various Cloud Natural Language API documentation pages. Under 'MARKETPLACE', there are links to the Cloud Natural Language API and the Natural Language Accelerator.

2. Enable the Cloud Natural Language API.

The screenshot shows the Google Cloud Marketplace page for the Cloud Natural Language API. At the top, there's a 'TRY THIS API' button and an 'API Enabled' status indicator. Below that, there are tabs for 'OVERVIEW' and 'DOCUMENTATION'. The 'OVERVIEW' tab is active, displaying a brief description: 'Provides natural language understanding technologies, such as sentiment analysis, entity...'. It also features a 'MANAGE' button. The 'DOCUMENTATION' tab is shown below. To the right, there's a section titled 'Additional details' with information like Type: SaaS & APIs, Last updated: 7/23/21, Category: Machine learning, Google Enterprise APIs, and Service name: language.googleapis.com.

3. Next, create a **Service Account** to Access the API in **IAM and Admin** panel by adding appropriate details

Create service account

Service account details

Service account name: assignment3
Display name for this service account

Service account ID: assignment3
Email address: assignment3@sde-a3.iam.gserviceaccount.com

Service account description: (Describe what this service account will do)

Grant this service account access to project (optional)

Grant users access to this service account (optional)

DONE **CANCEL**

4. After creating the service account **create an account key in JSON format** under the **Manages keys** option.

Email	Status	Name	Description	Key ID	Key creation date	OAuth 2	Actions
assignment3@sde-a3.iam.gserviceaccount.com	Green	assignment3		7511ca9379bf5106e2c1cdf44d7a35b395cf3e8	Mar 30, 2022	108160	⋮
assignment3@sde-a3.iam.gserviceaccount.com	Green	assignment3		ed0f1cb3a84ed8d753d051a16920552a57c9cdca	Mar 30, 2022	113821	⋮
compute@developer.gserviceaccount.com	Green	Compute Engine default service account	No keys				⋮

Manage details
Manage permissions
Manage keys
View metrics
View logs
Disable
Delete

5. Then under the add key option, there is an option of creating new key use that to create a new key in JSON format.

The screenshot shows the Google Cloud Platform IAM & Admin interface. On the left sidebar, under the 'Service Accounts' section, the 'Workload Identity Federation...' option is selected. In the main content area, a modal window titled 'Create private key for "assignment3"' is open. The modal contains the following information:

- A warning message: "Downloads a file that contains the private key. Store the file securely because this key can't be recovered if lost."
- A 'Key type' section with two options:
 - JSON (Recommended)
 - P12 (For backward compatibility with code using the P12 format)
- A table showing one key entry:

Type	Status	Key
●	Active	ed0f1cb...
- Buttons at the bottom: 'CANCEL' and 'CREATE'.

6. Then save this key and upload the same key to the previously created instance.

Now we also have a Cloud AI service with Compute Instance Finally we need Cloud storage to store the data.

For Cloud Storage, Cloud SQL was used, to create a Cloud SQL instance use the following steps.

1. Go to **SQL in Cloud Left side Pane**, Click on **Create Instance** then Select the Type of Instance you need like **Mysql**, Postgres, and SQL server. Mysql was chosen for this assignment.

The screenshot shows the Google Cloud Platform interface for creating a Cloud SQL instance. At the top, there are tabs for 'Inbox (2,482) - budhwani.t@iitj.', 'SDE Jan 2022', and 'Create an instance - SQL - SDE-a3'. Below the tabs, the URL is 'console.cloud.google.com/sql/choose-instance-engine?organizationId=315541594599&project=sde-a3'. The main content area has a blue header bar with 'Google Cloud Platform' and 'SDE-A3' on the left, and a search bar with 'Search Natural Language' on the right. Underneath, there's a breadcrumb trail: 'SQL' > 'Create an instance'. The main content is titled 'Choose your database engine' and lists three options:

- MySQL**: Versions: 8.0, 5.7, 5.6. A 'Choose MySQL' button is present.
- PostgreSQL**: Versions: 14, 13, 12, 11, 10, 9.6. A 'Choose PostgreSQL' button is present.
- SQL Server**: Versions: 2019, 2017. A 'Choose SQL Server' button is present.

At the bottom of the content area, there's a link 'Want more context on the Cloud SQL database engines? [Learn more](#)'.

2. Then after selecting Mysql Enter the instance name and appropriate details like **size, CPUs, Password, etc.**. Click on **create to create the instance will take around 5-6 minutes**.

The screenshot shows the 'Create a MySQL instance' configuration page. At the top, there are tabs for 'Inbox (2,482) - budhwani.t@iitj.', 'SDE Jan 2022', and 'Create a MySQL instance - SQL - SDE-a3'. Below the tabs, the URL is 'console.cloud.google.com/sql/instances/create;engine=MySQL?organizationId=315541594599&project=sde-a3'. The main content area has a blue header bar with 'Google Cloud Platform' and 'SDE-A3' on the left, and a search bar with 'Search Natural Language' on the right. Underneath, there's a breadcrumb trail: '← Create a MySQL instance'. The main content is divided into sections:

- Instance info**:
 - Instance ID *: m21cs007-sq
 - Password *: Set a password for the root user. [Learn more](#)
 - No password:
 - Database version *: MySQL 5.7
- Summary**:

Region	us-central1 (Iowa)
DB Version	MySQL 5.7
vCPUs	4 vCPU
Memory	26 GB
Storage	100 GB
Network throughput (MB/s)	1,000 of 2,000
Disk throughput (MB/s)	Read: 48.0 of 240.0 Write: 48.0 of 240.0
IOPS	Read: 3,000 of 15,000 Write: 3,000 of 15,000
Connections	Public IP
Backup	Automated
Availability	Multiple zones (Highly available)
Point-in-time recovery	Enabled
- Zonal availability**:
 - Single zone
 - In case of outage, no failover. Not recommended for production.

Choose region and zonal availability

For better performance, keep your data close to the services that need it. Region is permanent, while zone can be changed any time.

Region

us-central1 (Iowa)

Zonal availability

Single zone
In case of outage, no failover. Not recommended for production.

Multiple zones (Highly available)
Automatic failover to another zone within your selected region. Recommended for production instances. Increases cost.

SPECIFY ZONES

Customize your instance

You can also customize instance configurations later

SHOW CONFIGURATION OPTIONS

CREATE INSTANCE **CANCEL**

Summary	
Region	us-central1 (Iowa)
DB Version	MySQL 5.7
vCPUs	4 vCPU
Memory	26 GB
Storage	100 GB
Network throughput (MB/s)	1,000 of 2,000
Disk throughput (MB/s)	Read: 48.0 of 240.0 Write: 48.0 of 240.0
IOPS	Read: 3,000 of 15,000 Write: 3,000 of 15,000
Connections	Public IP
Backup	Automated
Availability	Multiple zones (Highly available)
Point-in-time recovery	Enabled

3. Then after the instance is created it looks like this.

Instances – SQL – SDE-A3 – Google Cloud Platform

SQL Instances

Filter Enter property name or value

Instance ID	Type	Public IP address	Private IP address	Instance connection name	High availability	Location	Storage used	Labels	Actions
m21cs007-sql	MySQL 5.7	34.122.45.49	10.128.0.3	sde-a3:us-central1:m21cs007-sql	ENABLED	us-central1-f	1 GB of 100 GB		

4. Then click on the instance and see all activity.

The screenshot shows the Google Cloud Platform SQL Overview page for the instance **m21cs007-sql**. The left sidebar includes links for Overview, Connections, Users, Databases, Backups, Replicas, and Operations. The main area displays a chart titled "CPU utilization" showing usage over the last 24 hours. Below the chart, there is a "Connect to this instance" section with a public IP address (34.122.45.49) and a "Configuration" section detailing 4 vCPUs, 26 GB Memory, and 100 GB SSD storage.

5. After this, we need to give access to users for accessing the Databases for this go-to **connection -> networking**.

The screenshot shows the Google Cloud Platform SQL Connections Networking page for the instance **m21cs007-sql**. The left sidebar includes links for Overview, Connections, Users, Databases, Backups, Replicas, and Operations. The main area is titled "Connections" and "NETWORKING". It shows the configuration for Instance IP assignment, with "Public IP" selected. A warning message at the bottom states: "You have added 0.0.0.0/0 as an allowed network. This prefix will allow any IPv4 client to pass the network firewall and make login attempts to your instance." Other tabs like SECURITY and CONNECTIVITY TESTS are also visible.

- Then under add network add a network like **0.0.0.0/0** with the name **ALL** to give access to all public networks.

The screenshot shows the Google Cloud Platform SQL Connections page. On the left, there's a sidebar with 'PRIMARY INSTANCE' options like Overview, Connections (which is selected), Users, Databases, Backups, Replicas, and Operations. The main area is titled 'Connections' and shows a dropdown menu with 'ALL (0.0.0.0/0)'. Below it, a 'New network' section is open, showing a 'Name' input field with 'ALL' and a 'Network' input field with '0.0.0.0/0'. At the bottom of this dialog are 'CANCEL' and 'DONE' buttons. Below the dialog, there's an 'ADD NETWORK' button and an 'App Engine authorization' note. At the very bottom are 'SAVE' and 'DISCARD CHANGES' buttons.

- Then Click on Save this will take some time to restart the instance.

Finally, we have created all three services. Now combining all three into an application.

For this application, we are using the computer engine to host a flask application that takes user input then perform the sentiment analysis, entity analysis, and classification analysis, and then provide the result and store that result in the Cloud SQL for future query and also it shows all the previous done experiments in the application.

For this we need to perform some steps:

- We need to create a file **server.py** that contains the required code of flask and also we need to create the folder templates that contain the HTML files which are used to perform operations and show results.
- In the server.py file, we need to give access to our Cloud Natural Language API which can be done using the os library provided by python

```
os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = "<key_file.json>"
```

and also we need to access the cloud Mysql which can be done using

```
engine = sqlalchemy.create_engine('mysql+pymysql://root:root@34.122.45.49/NLP')
```

where **34.122.45.49** is the public IP of Mysql and **NLP** is the name of the database and the User name and Password are both root respectively.
- For running this file without errors some libraries need to be installed using pip or pip3 command which are mysql.connector, sqlalchemy, pymysql, google.cloud, google-cloud-language, etc.
- Finally, after creating these files we need to run the flask project using the gunicorn so that our hosting on the nginx server can be done without any problem.
 - Use command `gunicorn server:app`
- Finally, go to the link `http://<IP_of_instance>` like to see the Main Home page.
- Go to `http://<IP_of_instance>/apis` like to see sub home page.
- Go to `http://<IP_of_instance>/apis/show` like to see the records present in Mysql Database.
- And finally go to `http://<IP_of_instance>/apis/data` like to perform new sentiment analysis.

For verification of the application, you can all see that the records are entered into the database.

Part B: Creating a Load Balancer.

Step 1: Create an Instance template.

- First, we need to go to the **engine instance template**, and then we need to **create the instance template**.

The screenshot shows the Google Cloud Platform Instance templates page. The left sidebar is collapsed, showing the Compute Engine section with 'Virtual machines' expanded, showing 'VM instances', 'Instance templates' (which is selected), 'Sole-tenant nodes', 'Machine images', 'TPUs', 'Committed use discounts', and 'Migrate for Compute Engine...'. Below that is the 'Storage' section with 'Disks', 'Snapshots', 'Marketplace', and 'Release Notes'. The main content area has a title 'Instance templates' with a sub-instruction: 'An instance template lets you describe a VM instance. You can then create groups of identical instances based on the template.' A 'CREATE INSTANCE TEMPLATE' button is at the bottom. The top navigation bar includes tabs for 'Compute Engine', 'Classes', and 'Instance templates', along with a search bar and various icons.

- After this enter the appropriate name for the template and click on create the template.

The screenshot shows the 'Create an instance template' dialog. The left sidebar is collapsed, showing the same Compute Engine sections as the previous screenshot. The main form starts with a description: 'Describe a VM instance once and then use that template to create groups of identical instances' with a 'Learn more' link. It has a 'Name' field containing 'm21cs007-sde'. Below it is a 'Label' section with a '+ ADD LABELS' button. To the right, there's a note about estimated costs: 'These are estimated costs for a VM instance created using this template: Monthly estimate \$13.23 That's about \$0.02 hourly. Pay for what you use: No upfront costs and per second billing'. Under 'Machine configuration', there's a 'Machine family' section with tabs for 'GENERAL-PURPOSE', 'COMPUTE-OPTIMIZED', 'MEMORY-OPTIMIZED', and 'GPU'. The 'GENERAL-PURPOSE' tab is selected. Below it is a 'Series' dropdown set to 'E2'. A 'CPU platform selection based on availability' section follows, with a 'Machine type' dropdown set to 'e2-small (2 vCPU, 2 GB memory)'. At the bottom are 'CREATE', 'CANCEL', and 'EQUIVALENT COMMAND LINE' buttons.

- After creating the template it will look as below.

The screenshot shows the Google Cloud Platform Compute Engine Instance templates page. The left sidebar has 'Compute Engine' selected under 'Virtual machines'. The main area displays a table of instance templates. One row is selected, showing the details: Name (m21cs007-sde), Machine type (e2-small), Image (debian-10-buster-v20220317), Disk type (Balanced persistent disk), Placement policy (No policy), In use by (None), Creation time (Apr 6, 2022, 12:16:46 PM UTC+05:30). There are 'CREATE INSTANCE TEMPLATE', 'REFRESH', 'CREATE INSTANCE GROUP', 'COPY', and 'DELETE' buttons at the top of the table.

Step 2: Create an Instance Group

- Go to Instance Template created in the previous step in compute engine and click on **create instance group**.

The screenshot shows the Google Cloud Platform Compute Engine Instance template details page for 'm21cs007-sde'. The left sidebar has 'Compute Engine' selected under 'Virtual machines'. The main area shows basic information (Name: m21cs007-sde, Type: Instance Template, Creation time: Apr 6, 2022, 12:16:46 PM UTC+05:30) and machine configuration (Machine type: e2-small, Minimum CPU platform: None, vCPUs to core ratio: -, Display device: Disabled, GPUs: None). At the top, there are buttons for 'CREATE VM', 'CREATE SIMILAR', 'CREATE INSTANCE GROUP', and 'DELETE'. The 'CREATE INSTANCE GROUP' button is highlighted.

- Provide the details like **Group name**, select **Multiple zones**, **number of instances** managed should be 2 under autoscaling, under autoscaling metric select **70% CPU utilization** for using another instance.

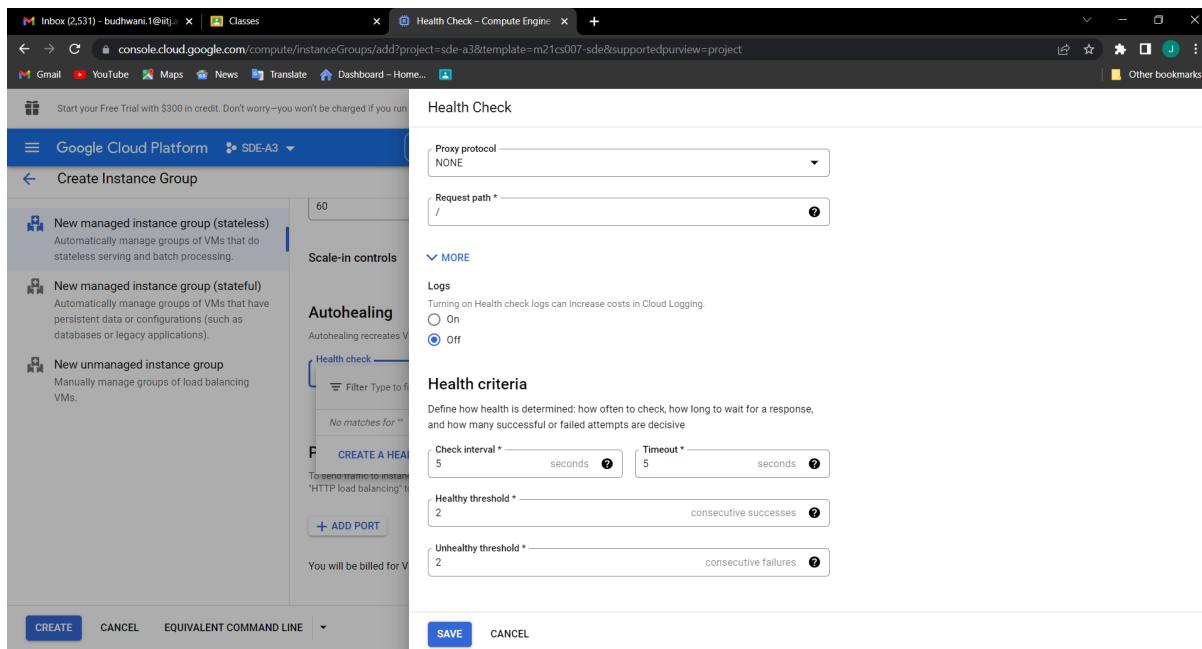
The screenshot shows the 'Create Instance Group' page in the Google Cloud Platform. The 'Name' field is set to 'm21cs007-group'. The 'Instance template' is 'm21cs007-sde'. Under 'Location', the 'Multiple zones' option is selected. At the bottom, there are 'CREATE', 'CANCEL', and 'EQUIVALENT COMMAND LINE' buttons.

The screenshot shows the 'Create Instance Group' page with more detailed configurations. Under 'Location', 'Multiple zones' is selected, with 'Region' set to 'us-central1 (Iowa)' and 'Zones' set to 'us-central1-c, us-central1-f, and us-central1-b'. The 'Target distribution shape' is set to 'Even'. Under 'Autoscaling', the mode is 'On: add and remove instances to the group', with 'Minimum number of instances' at 1 and 'Maximum number of instances' at 2. A note at the bottom states: 'To maximize availability, the minimum number of instances should be at least equal to the number of zones.' Additional information about autoscaling is provided below.

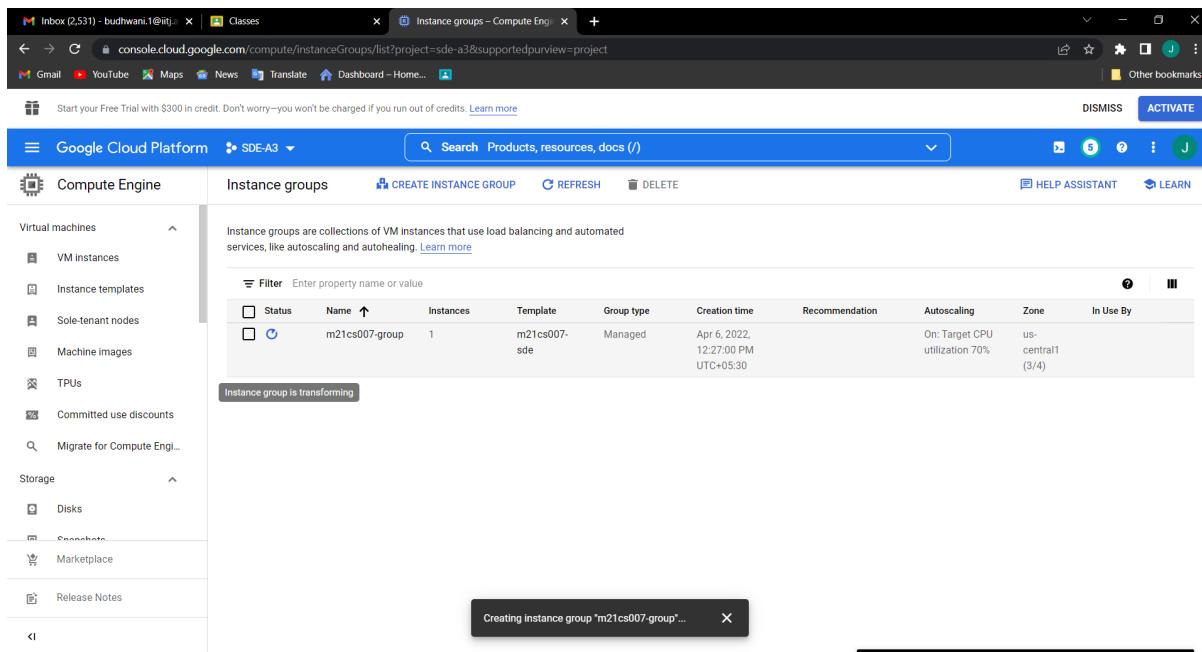
The screenshot shows the 'Create Instance Group' page in the Google Cloud Platform. On the left sidebar, there are three options: 'New managed instance group (stateless)', 'New managed instance group (stateful)', and 'New unmanaged instance group'. The 'New managed instance group (stateless)' option is selected. In the main pane, under the 'Autoscaling metrics' section, there is a sub-section titled 'Edit metric' where 'CPU utilization' is set as the metric type and the target is set to 70%. Below this, the 'Predictive autoscaling' section is shown, with the 'Optimize for availability' option selected. At the bottom of the page, there are 'CREATE', 'CANCEL', and 'EQUIVALENT COMMAND LINE' buttons.

- After setting the above there is one more thing that is most important which is creating a **health check** to check whether the instances are healthy and if not after how much time they need to be created again or deleted.

The screenshot shows the 'Create Instance Group' page with the 'Health Check' configuration section highlighted. The 'Name' field is filled with 'm21cs007-health-check'. The 'Protocol' is set to 'HTTP' and 'Port' to '80'. The 'Proxy protocol' is set to 'NONE'. The 'Request path' is set to '/'. Under the 'Logs' section, the 'On' option is selected. At the bottom of the page, there are 'SAVE' and 'CANCEL' buttons.



- After this, we need to create this instance group and after creation, it looks like the below.



- It will take some time till the instance group is running.

Step 3: Creating Load Balancer.

- Go to Network services to create a load balancer.

The screenshot shows the Google Cloud Platform Network services page. On the left, there's a sidebar with options like Network services, Load balancing, Cloud DNS, Cloud CDN, Cloud NAT, Traffic Director, Service Directory, Cloud Domains, Private Service Connect, Marketplace, and Release Notes. The main area has a header "Get real-time analytics with Network Intelligence Center" with a "GO TO NETWORK INTELLIGENCE CENTER" button and a "REMIND ME LATER" link. Below this, there are tabs for LOAD BALANCERS, BACKENDS, and FRONTENDS, with LOAD BALANCERS selected. A central box titled "Load balancing" contains the text "Load balancers distribute incoming network traffic across multiple VM instances to help your application scale." and a "CREATE LOAD BALANCER" button. At the bottom, a note says "To edit load balancing resources like forwarding rules and target proxies, go to the [link]".

- Under create, the load balancer option selects **HTTP Load balancer**.

The screenshot shows the "Create a load balancer" page. The sidebar is identical to the previous one. The main area has a header "Create a load balancer" with a "HELP ASSISTANT" link. It displays three options: "HTTP(S) Load Balancing", "TCP Load Balancing", and "UDP Load Balancing". The "HTTP(S) Load Balancing" section is active, showing "Layer 7 load balancing for HTTP and HTTPS applications" with a "Learn more" link, "Configure" (HTTP LB, HTTPS LB), "Options" (Internet-facing or internal, Single or multi-region), and a "START CONFIGURATION" button. The other sections are similar but inactive.

- Provide the below options.

Please answer a few questions to help us select the right load balancing type for your application

Internet facing or internal only

Do you want to load balance traffic from the Internet to your VMs or serverless services, or only between VMs in your network?

From Internet to my VMs or serverless services

Only between my VMs

Global or Regional

Do you want to deploy your application in global, regional, or classic mode?

Global HTTP(S) Load Balancer (classic)

Global HTTP(S) Load Balancer

Regional HTTP(S) Load Balancer

CONTINUE

- Now we need to set the **backend configuration**.

Name * m21cs007-backend

Description

Backend type Instance group

Protocol HTTP Named port * http

Timeout * 30 seconds

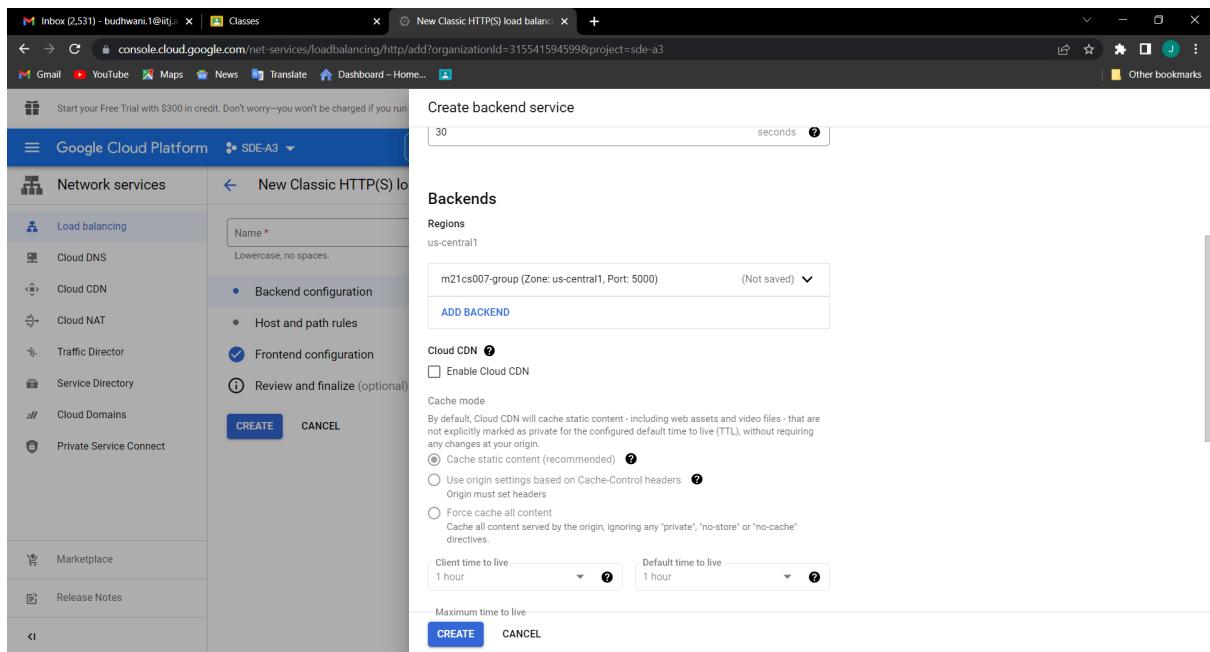
Backends

Regions us-central1

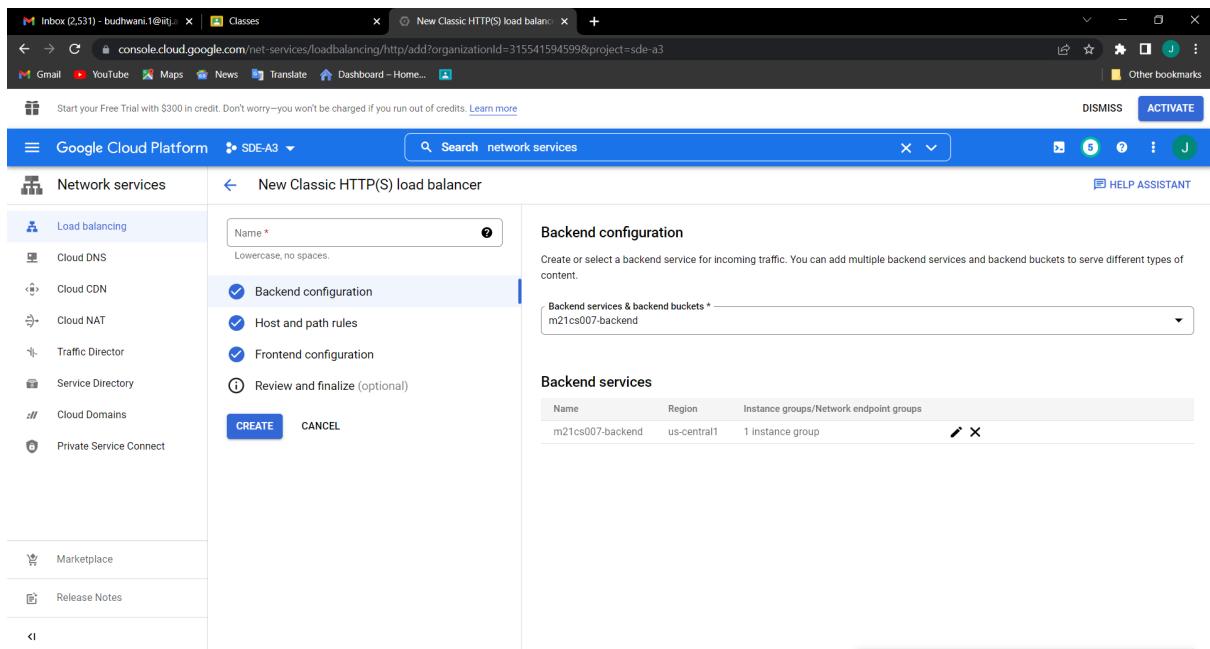
New backend

Instance group * m21cs007-group

CREATE **CANCEL**



- After creating the backend service it looks like the below.



- Finally, select create to create a load balancer. It will take some time to create.

The screenshot shows the Google Cloud Platform Network services Load balancing page. On the left, there's a sidebar with options like Cloud DNS, Cloud CDN, Cloud NAT, Traffic Director, Service Directory, Cloud Domains, and Private Service Connect. The main area has tabs for LOAD BALANCERS, BACKENDS, and FRONTENDS. Under LOAD BALANCERS, there's a table with one row for 'm21cs007-loadbalancer'. The table columns include Name, Load balancer type, Protocols, Region, and Backends. A note below the table says: "To view or delete load balancing resources like forwarding rules and target proxies, go to the [load balancing components view](#)".

The screenshot shows the Google Cloud Platform Network services Load balancer details page for 'm21cs007-loadbalancer'. The left sidebar is identical to the previous screenshot. The main content area is titled 'Load balancer details' and includes sections for 'Frontend', 'Host and path rules', 'Backend', and 'Backend services'. In the 'Frontend' section, it shows a single entry for 'HTTP' with IP:Port 34.149.43.252:80 and Network Tier Premium. The 'Host and path rules' section shows a single entry for 'All unmatched (default)' pointing to 'm21cs007-backend'. The 'Backend' section lists '1. m21cs007-backend' with details: Endpoint protocol HTTP, Named port http, Timeout 30 seconds, Health check [m21cs007-health-check](#), and Cloud CDN Disabled. The 'Backend services' section shows a table for 'm21cs007-group' with columns: Name, Type, Zone, Healthy, Autoscaling, Balancing mode, Selected ports, and Capacity. The table shows 2 of 2 healthy instances, a target CPU utilization of 70%, and a capacity of 100%.

- After Creating the load balancer we can see that there would be two new instances created in the compute engine instances as shown below. The First one is the base application.

The screenshot shows the Google Cloud Platform Compute Engine Instance Groups Overview page. The instance group 'm21cs007-group' is selected. Key details shown include:

- Instances by status:** 2 instances, both healthy (100% healthy).
- Autoscaling:** On (min 2, max 3). Predictive autoscaling is on.
- Status:** Ready.
- Creation Time:** Apr 6, 2022, 12:27:00 PM UTC+05:30.
- Description:** Number of instances: 2; Template: m21cs007-sde; Location: us-central1; In use by: m21cs007-backend.
- Instance Group Members:** Two instances listed:

Status	Name	Creation Time	Template	Per instance config	Internal IP	External IP	Health Check Status	Connect
Ready	m21cs007-group-ks7k	Apr 6, 2022, 1:20:58 PM UTC+05:30	m21cs007-sde		10.128.0.8 (nic0)	35.232.198.124 (nic0)	Healthy	SSH
Ready	m21cs007-group-khls	Apr 6, 2022, 12:27:29 PM UTC+05:30	m21cs007-sde		10.128.0.5 (nic0)	35.225.138.182 (nic0)	Healthy	SSH

- Now to show how the application would run using a load balancer is really simple just use the load balancer Ip address to access either of the two applications. We have differentiated the **two applications using different text styles**.

Original

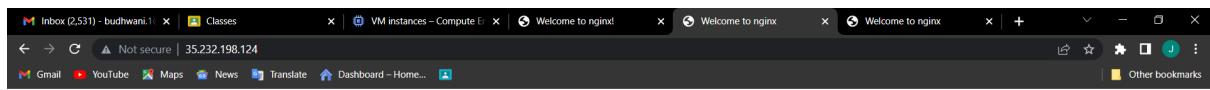
SDE 2022

ASSIGNMENT 3 GOOGLE CLOUD APPLICATION

Jayesh Budhwani M21CS007

This is a web application that is hosted on Compute engine using Nginx, Flask, and Gunicorn. The application does Sentiment Analysis using Google Cloud AI service named as Natural Language API on user provided data and then stores that data and its analysis on Cloud MySql server and then shows the result to the user. The user can also see the complete database.

First Load Balancer Instance



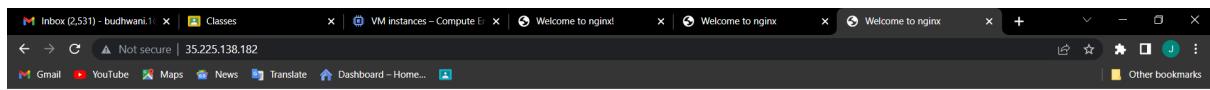
SDE 2022

ASSIGNMENT 3 GOOGLE CLOUD APPLICATION

Jayesh Budhwani M21CS007

This is a web application that is hosted on **Compute engine using Nginx, Flask, and Gunicorn**. The application does Sentiment Analysis using **Google Cloud AI service named as Natural Language API** on user provided data and then stores that data and its analysis on **Cloud MySQL server** and then shows the result to the user.
The user can also see the complete database.

- **Second Load Balancer Instance**



SDE 2022

ASSIGNMENT 3 GOOGLE CLOUD APPLICATION

Jayesh Budhwani M21CS007

This is a web application that is hosted on **Compute engine using Nginx, Flask, and Gunicorn**. The application does Sentiment Analysis using **Google Cloud AI service named as Natural Language API** on user provided data and then stores that data and its analysis on **Cloud MySQL server** and then shows the result to the user.
The user can also see the complete database.

As we can see all the three instances are working perfectly with different text styles to differentiate between them the first one is not a part of the load balancer the other two are a part of the load balancer.

References:

1. <https://cloud.google.com/natural-language/docs/reference/libraries>
2. <https://cloud.google.com/sql/docs/mysql/admin-api/libraries>