GCP Architecture :

A 3 tier environment is a common setup. Use a tool of your choosing/familiarity to create these resources. Please remember we will not be judging on the outcome but more on the approach, style and reproducibility.

The following is the diagram of a 3 tier environment . This infrastructure is shown below .

Presentation Tier :

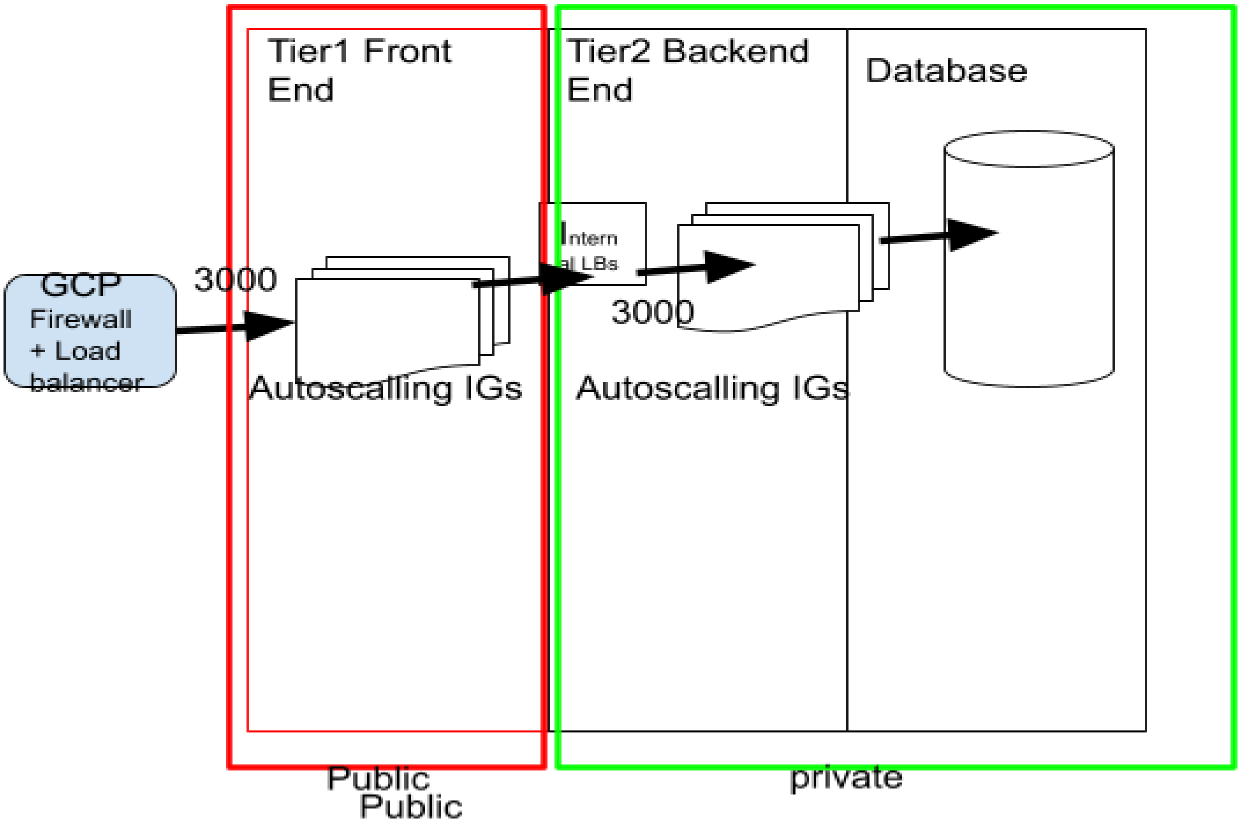
The Front End tier is made scalable by using Instance groups for running the front end. It has a Load balancer which will distribute the external load based on the health of the instances provided

Application Backend tier:

The Backend is made scalable by using Instance groups for running the front end. It has a Load balancer which will distribute the internal load based on the health of the instances , it can scale up and down .

Data Tier:

The data tier will usually have a database which will be encrypted with a vendor managed key. It stores the application and provides the output . I have provided GCP PAAS (cloud sql for database)



There are several other pieces to make this system robust and complete like

High Availability :

For High availability the architecture just needs to be **replicated across 2 zones** and global load balancer in the front . This makes a primary and a secondary backup . The database on the secondary is kept updated and can be used for read only .

Backup and recovery :

The DR and Back for compute resources that need to be created with persistent by creating snapshots of persistent disks to protect against data loss due to user error.

User management :

Onboarding users on the cloud and setting their roles and permissions based on the organization .

Security and caching :

A cloud CDN can be used to further applied users a better feel if using static files . For security we can google KMS store passwords , certificates and security token.

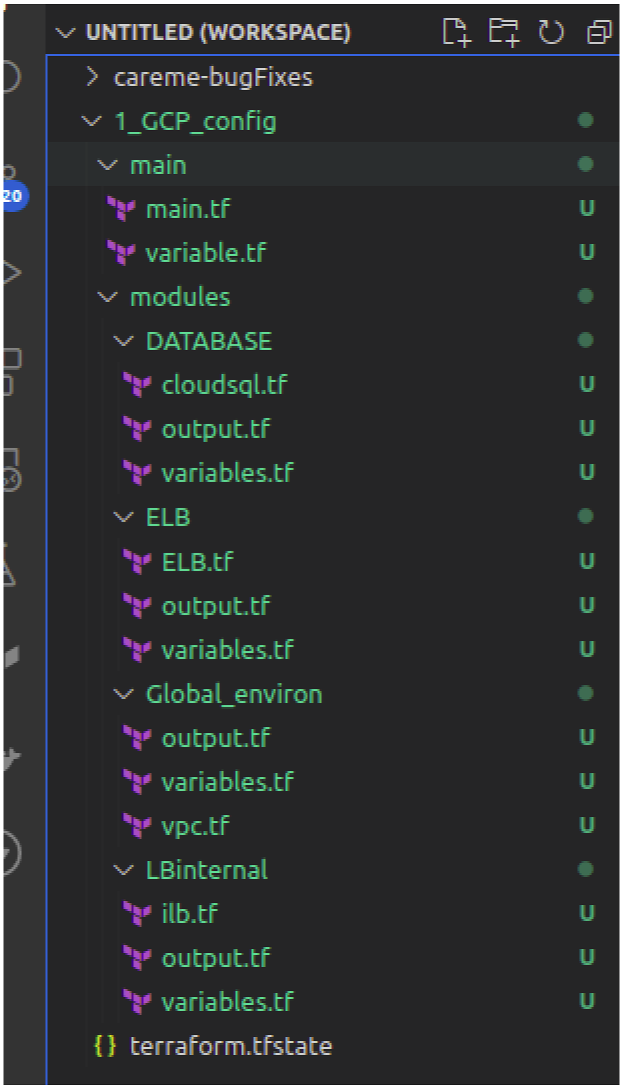
Solution : This above architecture is implemented using gcloud commands in

Gloud solution

Filename : **Sol1_gcloud_architectute.txt**
Stage 1 : Create the Gcloud Environment :
Stage 2 : Create the network and Firewalls :
Stage 3: Creation of Scalable Instance groups for Front end and Backend :
Stage 4: Creation of External and Internal Load balancers:
Stage 5: Creating a Cloud PAAS DB (Cloud SQL to be used by the LBs)

Terraform Solution

Folder Name : Sol1_terraform_architecture
***** Prerequisite of service Account is needed**
The folder Architecture



```
UNTITLED (WORKSPACE)
  > careme-bugFixes
  v 1_GCP_config
    v main
        main.tf              U
        variable.tf          U
    v modules
      v DATABASE
          cloudsql.tf        U
          output.tf          U
          variables.tf       U
      v ELB
          ELB.tf             U
          output.tf          U
          variables.tf       U
      v Global_environ
          output.tf          U
          variables.tf       U
          vpc.tf             U
      v LBinternal
          ilb.tf             U
          output.tf          U
          variables.tf       U
    {} terraform.tfstate
```

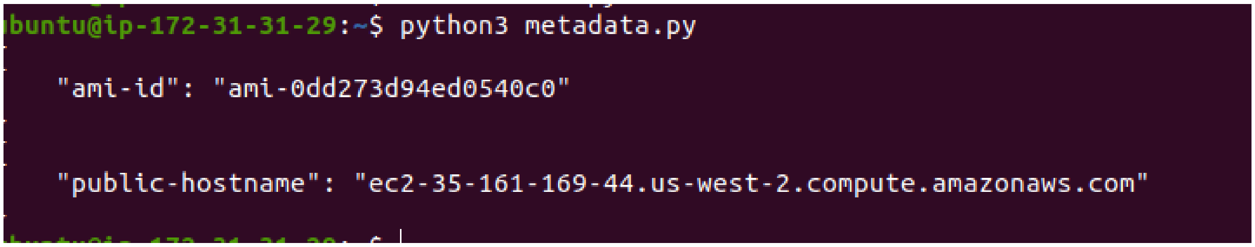
We have to run
> terraform init
> terraform validate

> Terraform plan –output my3tier.plan
>terraform apply –auto-approve main/

++
Q2:   METADATA FOR A EC2 INSTAVE IN JSON

The instance metadata needs to be fetched from the instance and a the metadataservice is running ('http://169.254.169.254/latest/meta-data/') which provides the all metadata and we use it in json format . The code provided just use the above uri and gets the instance metadata .

The code is run
> python3 metadata.py



```
buntu@ip-172-31-31-29:~$ python3 metadata.py

    "ami-id": "ami-0dd273d94ed0540c0"


    "public-hostname": "ec2-35-161-169-44.us-west-2.compute.amazonaws.com"
```

Here we have shown the meta data for
> ami-id
> public-hostname

We can get the metadata for all the values

++

Q3: STRING FIND VALUE FOR COMBINED KEYS

For {"a":{"b":{"c":{"d":"e"}}}} for a

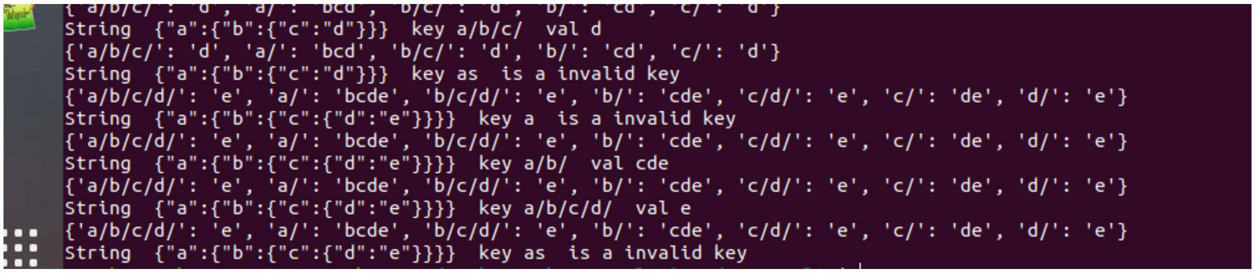
key of a/b/c/d/ Val = e
Key b/c/d/   Val = e

Solution:
We can consider this structure as tree
A →b →c →d –e

Wherefor a key we can get the last node/nodes as val
We use DFS
As we have DFS key = A–B then DFS will spill values from B to end node that is E
So we get C D E as the value for key .

Implementation:

This is implemented in a clever way with string modification and keeping the string in a map of the key
a



```
{'a/b/c/': 'd', 'a/': 'bcd', 'b/c/': 'd', 'b/': 'cd', 'c/': 'd'}
String  {"a":{"b":{"c":"d"}}}  key a/b/c/  val d
{'a/b/c/': 'd', 'a/': 'bcd', 'b/c/': 'd', 'b/': 'cd', 'c/': 'd'}
String  {"a":{"b":{"c":"d"}}}  key as  is a invalid key
{'a/b/c/d/': 'e', 'a/': 'bcde', 'b/c/d/': 'e', 'b/': 'cde', 'c/d/': 'e', 'c/': 'de', 'd/': 'e'}
String  {"a":{"b":{"c":{"d":"e"}}}}  key a  is a invalid key
{'a/b/c/d/': 'e', 'a/': 'bcde', 'b/c/d/': 'e', 'b/': 'cde', 'c/d/': 'e', 'c/': 'de', 'd/': 'e'}
String  {"a":{"b":{"c":{"d":"e"}}}}  key a/b/  val cde
{'a/b/c/d/': 'e', 'a/': 'bcde', 'b/c/d/': 'e', 'b/': 'cde', 'c/d/': 'e', 'c/': 'de', 'd/': 'e'}
String  {"a":{"b":{"c":{"d":"e"}}}}  key a/b/c/d/  val e
{'a/b/c/d/': 'e', 'a/': 'bcde', 'b/c/d/': 'e', 'b/': 'cde', 'c/d/': 'e', 'c/': 'de', 'd/': 'e'}
String  {"a":{"b":{"c":{"d":"e"}}}}  key as  is a invalid key
```