**Challenge #1**

A 3-tier environment is a common setup. Use a tool of your choosing/familiarity create these resources on a cloud environment (**GCP Cloud**).

Solution:-

This can be accomplished by using either GCP Console or Google cloud command line tool.

We can follow below steps to create common setup :

1. Create a Virtual Private Cloud (VPC) network for the resources in the 3-tier environment.
2. Create a firewall rule to allow incoming traffic to the web tier
3. Create a managed instance group for the web tier and configure it with a startup script to install a web server and configure it with a basic website. Similarly we need to do the above steps for Application tier and Database tier.
4. Create a load balancer to distribute incoming traffic to the web tier
5. Create an instance template for each tier and attach it to the corresponding managed instance group.
6. Create a managed disk and attach it to each instance in the database tier for persistence of data.

**Challenge #2**

We need to write code that will query the meta data of an instance within GCP and provide a json formatted output.

Solution:-

In Python:

import requests

import json

url = <http://metadata.google/computemetadata/v1/instances/> #Url to access the metadata of the instance

url += "id" # Add the specific metadata item,wants to retrieve

headers = {'Metadata-Flavor': 'Google'} # Add the header to specify that the request has came from a Google Compute Engine instance

response = requests.get(url, headers=headers) # Make the request and retrieve the metadata

if response.status\_code == 200:

metadata = response.text # If the request is successful, retrieve the metadata value

metadata\_dict = json.loads(metadata)

print(json.dumps(metadata\_dict, indent=4)) # Convert the metadata to a dictionary and print it as JSON

**Challenge #3**

We have a nested object. We would like a function where you pass in the object and a key and get back the value.

Solution :-

In Python-

def get\_nested\_value(data, keys):

if type(data) is not dict:

return None

if len(keys) == 1:

if keys[0] in data:

return data[keys[0]]

else:

return None

if keys[0] in data:

return get\_nested\_value(data[keys[0]], keys[1:])

else:

return None