##Challenge 1

To create 3 tier architecture in AWS:-

1. Create Virtual private cloud
2. Create an internet gateway
3. Attach VPC to internet gateway
4. Create public and private subnets
5. Create public and private route tables→ public and private subnets need to be associated with public and private route tables respectively.
6. Create NAT gateway
7. Create an autoscaling group

##Challenge 2

Write a code that will query the metadata of an instance within AWS and provide json formatted output.

```
import requests
import json

metadata_url= 'http://169.254.169.254/latest/'

def expand_tree(url, arr):
    output = {}
    for item in arr:
        new_url = url + item
        r = requests.get(new_url)
        text = r.text
        if item[-1] == "/":
            list_of_values = r.text.splitlines()
            output[item[:-1]] = expand_tree(new_url, list_of_values)
        elif is_json(text):
            output[item] = json.loads(text)
        else:
            output[item] = text
    return output

def get_metadata():
    initial = ["meta-data/"]
    result = expand_tree(metadata_url, initial)
    return result

def get_metadata_json():
```

```
    metadata = get_metadata()
    metadata_json = json.dumps(metadata, indent=4, sort_keys=True)
    return metadata_json

def is_json(myjson):
    try:
        json.loads(myjson)
    except ValueError:
        return False
    return True

if __name__ == '__main__':
    print(get_metadata_json())
```

##Challenge 3

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

```
def getKey(obj: dict):
    keys = list(obj)
    if len(keys) != 1:
        raise Exception('either multiple keys or empty dict found')
    else:
        return keys[0]


def getNestedValue(obj: dict, key: str, isFound = False):
    if type(obj) is not dict and not isFound:
        return None
    if (isFound or (key in obj.keys())) :
        if type(obj[key]) is dict:
            return getNestedValue(obj[key], getKey(obj[key]), True)
        else:
            return obj[getKey(obj)]
    else:
        nestedKey = getKey(obj)
        return getNestedValue(obj[nestedKey], key, False)

if __name__ == '__main__':
    obj = {'a': {'b': {'c': 'd'}}}
    value = getNestedValue(obj, 'c')
    print(value)
```