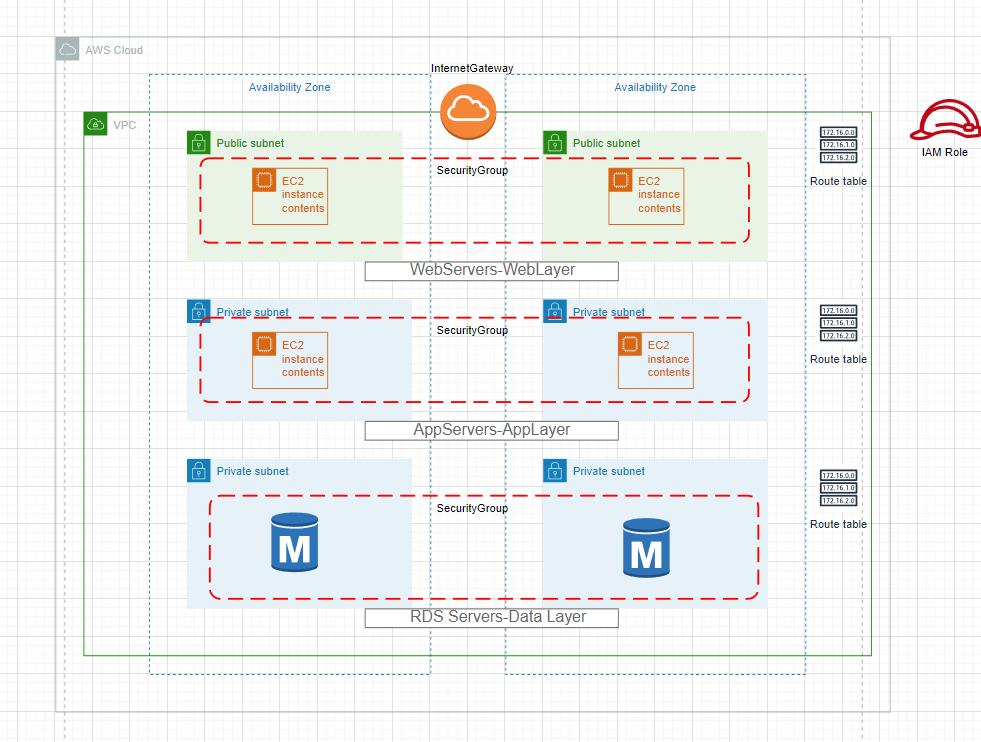
Challenge #1:

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

The first tier of architecture is a Web tier. It will consist of 2 public subnets in separate Availability zones, and EC2 instances launching a webpage with access to the internet. The second tier is an Application tier. This tier will consist of 2 private subnets, EC2 instances that have inbound access from the web tier. The third tier is a Database tier. This tier will have an RDS database in 2 private subnets with inbound access from the application tier above it.

To Automate the infrastructure and resources creation we can either use CloudFormation or terraform. Attached is the YAML code to create the below all resources.

Architectural View:



Challenge #2:

We need to write code that will query the meta data of an instance within AWS or Azure or GCP and provide a json formatted output. The choice of language and implementation is up to you.

Python Code to query metadata of an instance within AWS:

import boto3

import json

def get\_aws\_instance\_metadata():

    ec2\_client = boto3.client('ec2', region\_name='your\_region')  # Replace 'your\_region' with the region of your instance.

    instance\_id = 'your\_instance\_id'  # Replace 'your\_instance\_id' with the ID of your instance.

    response = ec2\_client.describe\_instances(InstanceIds=[instance\_id])

    instance\_metadata = response['Reservations'][0]['Instances'][0]

    return instance\_metadata

if \_\_name\_\_ == '\_\_main\_\_':

    aws\_instance\_metadata = get\_aws\_instance\_metadata()

    # Access specific data keys individually

    instance\_id = aws\_instance\_metadata['InstanceId']

    instance\_type = aws\_instance\_metadata['InstanceType']

    public\_ip = aws\_instance\_metadata['PublicIpAddress']

    # Create a dictionary containing the desired keys

    desired\_data = {

        'InstanceId': instance\_id,

        'InstanceType': instance\_type,

        'PublicIpAddress': public\_ip

    }

    json\_output = json.dumps(desired\_data, indent=2)

    print(json\_output)

In the above code, we can access specific data keys (e.g., 'InstanceId', 'InstanceType', 'PublicIpAddress') individually and create a dictionary (desired\_data) containing these keys. We can then convert the desired\_data dictionary into JSON format and print it.

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. The choice of language and implementation is up to you.

Example Inputs

object = {“a”:{“b”:{“c”:”d”}}}

key = a/b/c

object = {“x”:{“y”:{“z”:”a”}}}

key = x/y/z

value = a

Python Function:

def get\_value\_from\_nested\_object(obj, key):

    keys = key.split('/')

    value = obj

    try:

        for k in keys:

            value = value[k]

        return value

    except (KeyError, TypeError):

        return None

# Example inputs

object1 = {"a": {"b": {"c": "d"}}}

key1 = "a/b/c"

object2 = {"x": {"y": {"z": "a"}}}

key2 = "x/y/z"

# Testing the function with the example inputs

result1 = get\_value\_from\_nested\_object(object1, key1)

result2 = get\_value\_from\_nested\_object(object2, key2)

print(result1)  # Output: "d"

print(result2)  # Output: "a"

How it will work:

The get\_value\_from\_nested\_object function takes two arguments:

obj (the nested object) and key (the key to find the value in the nested object).

It splits the key using '/' as a delimiter, and then iterates through the keys in the nested object until it reaches the final value.