**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). Please remember we will not be judged on the outcome but more focusing on the approach, style and reproducibility.

**Approach #1:**

To create a 3-tier environment on AWS, we will set up a web application with a front-end, an application server, and a database. Here is the 1st approach that I can think:

1. Front-end (Presentation Tier):

• Choose a service to host front-end. AWS offers various options, such as Amazon S3 (Simple Storage Service) for static websites or Amazon EC2 (Elastic Compute Cloud) for more flexibility.

• If we choose Amazon S3:

• Create an S3 bucket and enable static website hosting.

• Upload front-end files to the bucket.

• If we choose Amazon EC2:

• Launch an EC2 instance and select an appropriate instance type and operating system.

• Connect to the instance using SSH or RDP, depending on the operating system.

• Install and configure a web server like Apache or Nginx.

• Upload front-end files to the web server's document root.

2. Application Server (Application Tier):

• Choose a service to host application server. AWS provides options like Amazon EC2, AWS Lambda, or AWS Elastic Beanstalk.

• If we choose Amazon EC2:

• Launch another EC2 instance with an appropriate instance type and operating system.

• Connect to the instance using SSH or RDP.

• Install and configure the necessary software and dependencies for application server, such as Node.js, Java, or Python.

• Deploy application code on the instance.

3. Database (Data Tier):

• Choose a database service. AWS offers Amazon RDS (Relational Database Service) for relational databases like MySQL, PostgreSQL, or Amazon Aurora, or Amazon DynamoDB for NoSQL databases. We will prefer to choose AWS managed database service.

**Approach #2:**

To create a 3-tier environment on AWS we can use infrastructure-as-code (IaaC) tools like AWS CloudFormation. These tools allow us to define and provision cloud resources in a reproducible and automated manner.

Here is the 2nd approach to create a 3-tier environment on AWS using CloudFormation:

1. Define the network infrastructure:

* Create a Virtual Private Cloud (VPC) to isolate the environment.
* Define subnets for each tier (public, private, database).
* Create NACLs for those subnets to allow/deny the traffic.
* Configure route tables and internet gateway for public access.

1. Provision the compute resources:

* Create EC2 instances for each tier (web, application, database).
* Configure security groups to control inbound/outbound traffic.
* Attach the instances to the appropriate subnets.

1. Set up load balancing and auto-scaling:

* Create an Elastic Load Balancer (ELB) or Application Load Balancer (ALB) for the web tier.
* Configure the load balancer to distribute traffic across the web instances.
* Implement auto-scaling policies to automatically add/remove instances based on demand.

1. Configure the database tier:

* Set up a managed database service like Amazon RDS for the database tier.
* Define the database instance with appropriate configuration and security settings.

1. Deploy and configure application software:

* Use a configuration management tool (e.g., Ansible) to deploy and configure the required software on the instances.
* Customize the application settings based on your requirements.

1. Testing and validation:

* Implement monitoring and logging solutions to track the performance and health of the environment.
* Conduct thorough testing to ensure proper functionality and scalability.
* Validate the deployment against defined success criteria.

**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.

import json

import requests

import boto3

def get\_aws\_instance\_metadata():

response = requests.get('http://169.254.169.254/latest/dynamic/instance-identity/document')

data = response.json()

return data

def main():

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

json\_data = json.dumps(aws\_metadata, indent=4)

print(json\_data)

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

    main()

**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

**Solution:**

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 usage

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

key1 = "a/b/c"

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

print(result1) # Output: d

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

key2 = "x/y/z"

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

print(result2)  # Output: a