**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.

**Solution**:

# Create resource group

resource "azurerm\_resource\_group" "rg" {

name = "my-resource-group"

location = "eastus"

}

# Create virtual network

resource "azurerm\_virtual\_network" "vnet" {

name = "my-vnet"

address\_space = ["10.0.0.0/16"]

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

}

# Create subnet for web tier

resource "azurerm\_subnet" "web" {

name = "web-subnet"

address\_prefix = "10.0.1.0/24"

virtual\_network\_name = azurerm\_virtual\_network.vnet.name

resource\_group\_name = azurerm\_resource\_group.rg.name

}

# Create subnet for app tier

resource "azurerm\_subnet" "app" {

name = "app-subnet"

address\_prefix = "10.0.2.0/24"

virtual\_network\_name = azurerm\_virtual\_network.vnet.name

resource\_group\_name = azurerm\_resource\_group.rg.name

}

# Create subnet for database tier

resource "azurerm\_subnet" "db" {

name = "db-subnet"

address\_prefix = "10.0.3.0/24"

virtual\_network\_name = azurerm\_virtual\_network.vnet.name

resource\_group\_name = azurerm\_resource\_group.rg.name

}

# Create network security group for web tier

resource "azurerm\_network\_security\_group" "web\_nsg" {

name = "web-nsg"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

security\_rule {

name = "http"

priority = 100

direction = "Inbound"

access = "Allow"

protocol = "Tcp"

source\_port\_range = "\*"

destination\_port\_range = "80"

source\_address\_prefix = "\*"

destination\_address\_prefix = "\*"

}

}

# Create network security group for app tier

resource "azurerm\_network\_security\_group" "app\_nsg" {

name = "app-nsg"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

}

# Create network security group for database tier

resource "azurerm\_network\_security\_group" "db\_nsg" {

name = "db-nsg"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

}

# Create public IP for web tier

resource "azurerm\_public\_ip" "web\_ip" {

name = "web-ip"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

allocation\_method = "Static"

}

# Create network interface for web tier

resource "azurerm\_network\_interface" "web\_nic" {

name = "web-nic"

location = azurerm\_resource\_group.rg.location

resource\_group\_name = azurerm\_resource\_group.rg.name

ip\_configuration {

name = "web-ip-config"

subnet\_id = azurerm\_subnet.web.id

private\_ip\_address\_allocation = "Dynamic

}

}

**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.

**Solution: In Azure**

from azure.identity import DefaultAzureCredential from azure.mgmt.compute import ComputeManagementClient

credential = DefaultAzureCredential()

subscription\_id = "" resource\_group\_name = ""

vm\_name = ""

compute\_client = ComputeManagementClient(credential, subscription\_id)

instance\_metadata = compute\_client.virtual\_machines.get(resource\_group\_name, vm\_name).instance\_view.instance\_metadata

metadata\_dict = {}

for item in instance\_metadata: metadata\_dict[item.name] = item.value

import json metadata\_json = json.dumps(metadata\_dict)

print(metadata\_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. The choice of language and implementation is up to you

**Solution**:

Python Function as below-

def get\_value(nested\_object, key): """ Given a nested object and a key, returns the value for that key. If the key is not found, returns None. """ # Split the key into a list of nested keys keys = key.split(".")

# Loop through each key in the list

for k in keys:

# If the key is not found, return None

if nested\_object.get(k) is None:

return None

# If the key is found, update the nested object to the value associated with the key

nested\_object = nested\_object[k]

# Return the final value

return nested\_object