Task 1

Terraform configuration file (main.tf) that uses Ansible as the configuration management tool to create and deploy a running instance of a web server, secure it with appropriate ports and HTTPS redirection, and apply automated tests to validate the server configuration.

# Define the provider and its configuration

provider "aws" {

region = "us-west-2" # Replace with your desired region

}

# Create a security group to control inbound and outbound traffic

resource "aws\_security\_group" "web\_sg" {

name = "web\_sg"

description = "Security group for the web server"

ingress {

from\_port = 443

to\_port = 443

protocol = "tcp"

cidr\_blocks = ["0.0.0.0/0"]

}

egress {

from\_port = 0

to\_port = 0

protocol = "-1"

cidr\_blocks = ["0.0.0.0/0"]

}

}

# Create an EC2 instance

resource "aws\_instance" "web\_server" {

ami = "ami-0c94855ba95c71c99" # Replace with your desired AMI

instance\_type = "t2.micro" # Replace with your desired instance type

key\_name = "my\_keypair" # Replace with your key pair name

security\_group\_ids = [aws\_security\_group.web\_sg.id]

provisioner "local-exec" {

command = "ansible-playbook -i '${self.public\_ip},' playbook.yml"

}

connection {

type = "ssh"

user = "ubuntu"

private\_key = file("~/.ssh/my\_keypair.pem")

host = self.public\_ip

}

}

In the above Terraform configuration, we use the aws\_security\_group resource to create a security group allowing inbound traffic only on port 443 (HTTPS). Then, we use the aws\_instance resource to create an EC2 instance and associate it with the security group.

Inside the aws\_instance resource, we use the local-exec provisioner to execute an Ansible playbook (playbook.yml) on the instance. The command argument runs the ansible-playbook command with the inventory containing the public IP address of the instance. You'll need to create the Ansible playbook (playbook.yml) separately.

Remember to replace the placeholder values (AMI ID, instance type, key pair name) with your specific values.

After defining the Terraform configuration, you can run the usual Terraform commands (terraform init, terraform plan, terraform apply) to create and deploy the web server instance.

To complete the configuration with Ansible and apply the desired setup, you'll need to write the playbook.yml file separately. Here's an example of an Ansible playbook (playbook.yml) that installs and configures Nginx to serve the desired HTML content, enables HTTPS, and redirects HTTP to HTTPS:

---

- name: Configure web server

hosts: all

become: yes

tasks:

- name: Install Nginx

apt:

name: nginx

state: present

- name: Copy HTML file

copy:

src: hello\_world.html

dest: /var/www/html/index.html

- name: Generate self-signed SSL certificate

shell: |

openssl req -x509 -nodes -days 365 -newkey rsa:2048 \

-keyout /etc/nginx/cert.key -out /etc/nginx/cert.crt \

**Task 2 :**

You and Fredrick are good friends. Yesterday, Fredrick received  credit cards from **ABCD Bank**. He wants to verify whether his credit card numbers are valid or not. You happen to be great at regex so he is asking for your help!

A valid credit card from **ABCD Bank** has the following characteristics:  
  
► It must start with a ,  or .  
► It must contain exactly  digits.  
► It must only consist of digits (-).  
► It may have digits in groups of , separated by one hyphen **"-"**.  
► It must NOT use any other separator like ' ' , '\_', etc.  
► It must NOT have  or more consecutive repeated digits.

Python code to validate credit card numbers based on the provided characteristics:

import re

def validate\_credit\_card(number):

# Check if the number starts with 4, 5, 6, or has 16 digits and consists only of digits and hyphens

if not re.match(r'^[456]\d{3}(-?\d{4}){3}$', number):

return False

# Remove hyphens and check for consecutive repeated digits

number = number.replace('-', '')

if re.search(r'(\d)\1{3}', number):

return False

return True

# Test the function with some examples

credit\_cards = [

'4253625879615786', # Valid

'4424424424442444', # Invalid (consecutive repeated digits)

'5122-2368-7954-3214', # Valid

'5123 - 3567 - 8912 - 3456', # Invalid (spaces used as separators)

'61234-567-8912-3456', # Invalid (too many digits in a group)

'4123356789123456', # Invalid (does not start with 4, 5, 6)

'41234567891234567' # Invalid (more than 16 digits)

]

for card in credit\_cards:

if validate\_credit\_card(card):

print(f'{card} is a valid credit card number.')

else:

print(f'{card} is an invalid credit card number.')

This code defines a function validate\_credit\_card that takes a credit card number as input and returns True if the number is valid according to the provided characteristics, or False otherwise.

The regular expression ^[456]\d{3}(-?\d{4}){3}$ is used to match the credit card number pattern. It ensures that the number starts with 4, 5, or 6, followed by three digits, and then has three groups of four digits (with an optional hyphen in between).

After that, the function removes the hyphens from the number and checks for consecutive repeated digits using the regular expression (\d)\1{3}. If any four consecutive digits are found, it means the number is invalid.

Finally, the code tests the function with some example credit card numbers and prints whether each number is valid or invalid.