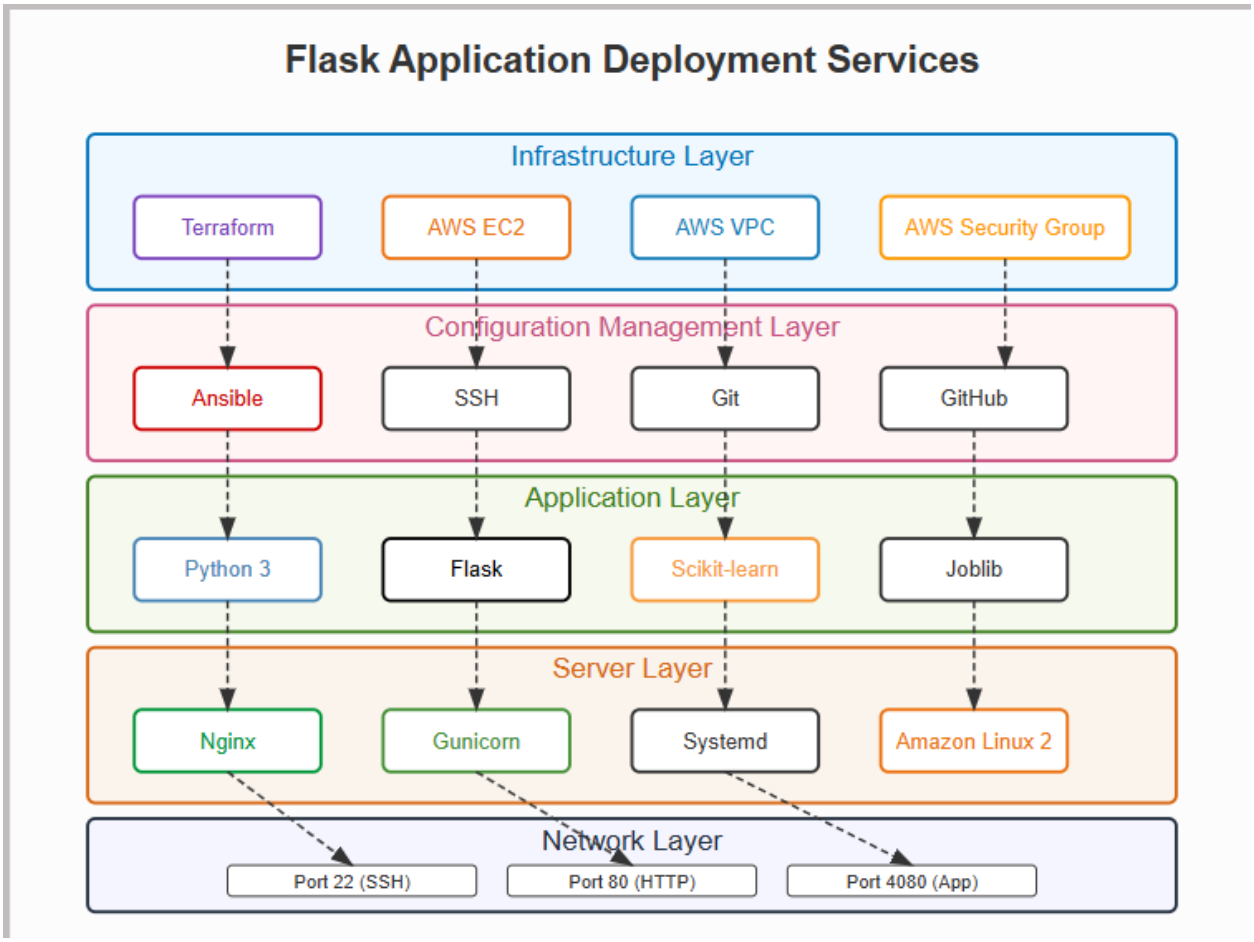
cprime

Ansible & Terraform for Seamless Flask App Deployment

NAME : T MANOJ

EMPID : LYAKE2KHS

AWS Services Used



EC2 (Elastic Compute Cloud)

- **Purpose**: Virtual servers in the cloud
- **How it's used here**: Two EC2 instances are created - one for the Flask application and another for Ansible
- **Configuration**: Both instances use the Amazon Linux 2 AMI (ami-0c02fb55956c7d316) and t2.micro instance type
- **Role**: Provides the computing infrastructure for both the application and automation

VPC (Virtual Private Cloud)

- **Purpose**: Isolated network environment in AWS
- **How it's used here**: The deployment uses the default VPC
- **Role**: Provides networking capabilities for the EC2 instances

Security Group

- **Purpose**: Virtual firewall for EC2 instances
- **How it's used here**: A security group named "app-security-group" is created with specific inbound and outbound rules
- **Configuration**:
  - Allows SSH (port 22) from anywhere
  - Allows HTTP (port 80) from anywhere
  - Allows application traffic (port 4080) from anywhere
  - Allows all outbound traffic
- **Role**: Controls traffic to and from the EC2 instances

Application Services

Flask

- **Purpose**: Python web framework for building web applications
- **How it's used here**: Runs the cardio heart detection application
- **Configuration**: Deployed from a GitHub repository (https://github.com/manojmanu9441/cardio_heart_detection.git)
- **Role**: Serves as the application framework

Gunicorn

- **Purpose**: WSGI HTTP Server for Python applications
- **How it's used here**: Runs as a systemd service, binding to port 4080
- **Configuration**: Configured with 3 worker processes
- **Role**: Manages multiple worker processes and handles HTTP requests to the Flask application

Nginx

- **Purpose**: Web server and reverse proxy
- **How it's used here**: Installed and configured to proxy requests to Gunicorn
- **Configuration**: Listens on port 80 and forwards requests to localhost:4080
- **Role**: Acts as a reverse proxy, handling client connections and forwarding requests to the application server

Automation Services

Terraform

- **Purpose**: Infrastructure as Code (IaC) tool
- **How it's used here**: Defines and creates all AWS resources
- **Configuration**: Contained in the main.tf file
- **Role**: Automates the provisioning of cloud infrastructure

Ansible

- **Purpose**: Configuration management and application deployment tool
- **How it's used here**: Installed on the Ansible server to configure the Flask server
- **Configuration**: Uses a playbook (deploy_flask_app.yml) to automate application deployment
- **Role**: Automates the software installation and configuration on the Flask server

Additional Components

Scikit-learn

- **Purpose**: Machine learning library for Python
- **How it's used here**: Required dependency for the cardio heart detection application
- **Role**: Provides machine learning functionality for the heart disease prediction model

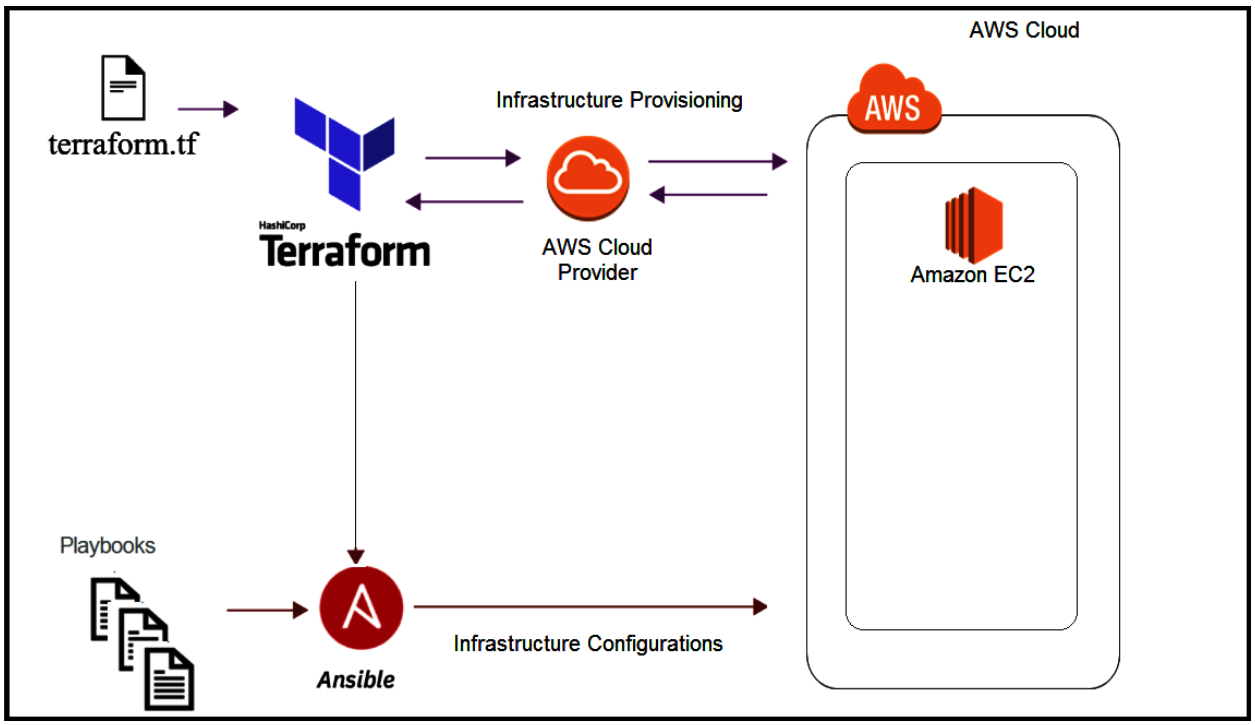
Systemd

- **Purpose**: System and service manager for Linux
- **How it's used here**: Manages the Gunicorn service
- **Configuration**: Defined in /etc/systemd/system/gunicorn.service
- **Role**: Ensures the application starts automatically and restarts if it fails

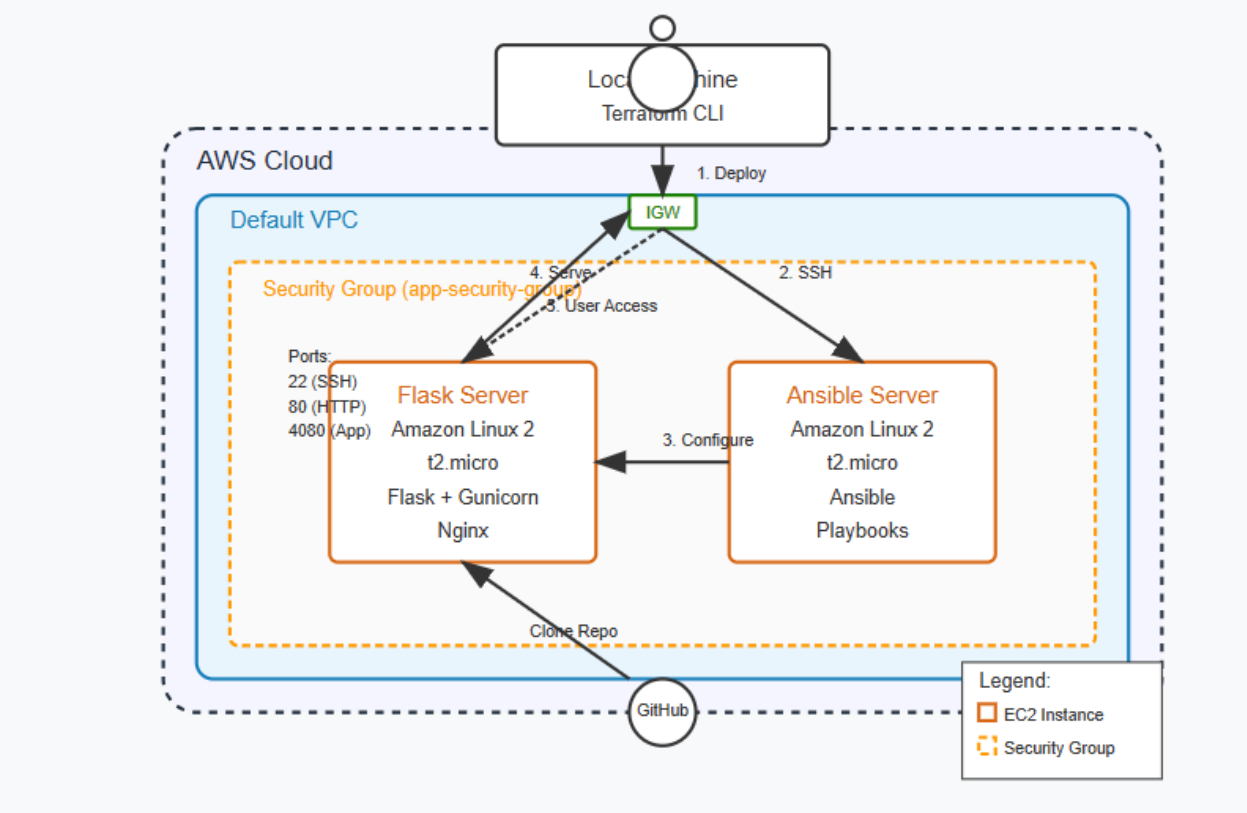
GitHub

- **Purpose**: Source code repository
- **How it's used here**: Hosts the Flask application code
- **Role**: Provides version control and code storage for the application

System Architecture



WorkFlow



Steps to Deploy Flask Web Application Using Terraform and Ansible

Prerequisites

1. Install Terraform on your local machine

```
manoj@IND-140:~/ansible$ terraform --version
Terraform v1.11.2
on linux_amd64
+ provider registry.terraform.io/hashicorp/aws v5.92.0
```

2. Generate an AWS key pair named "ansible" and download the `.pem` file

```
-rw-r--r--  1 manoj manoj  1675 Mar 21 10:45 ansible.pem
```

3. Configure AWS CLI with your credentials

```
manoj@IND-140:~/ansible$ aws configure
AWS Access Key ID [****************WMMA]:
AWS Secret Access Key [****************TjOf]:
Default region name [us-east-2]:
Default output format [None]:
```

Step 1: Prepare Your Environment

1. Create a new directory for your project

```
drwxr-xr-x  3 manoj manoj 4096 Mar 21 14:19 ansible/
```

2. Copy the Terraform configuration code into a file named `main.tf`

```
manoj@IND-140: ~/ansible
manoj@IND-140:~/ansible$ cat main.tf
provider "aws" {
  region = "us-east-1"
}

# Fetch the default VPC
data "aws_vpc" "default" {
  default = true
}

# Create a security group for our instances
resource "aws_security_group" "app_security_group" {
  name        = "app-security-group"
  description = "Security group for Flask and Ansible servers"
  vpc_id      = data.aws_vpc.default.id

  # SSH access from anywhere
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # HTTP access for Flask application
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Application port access
```

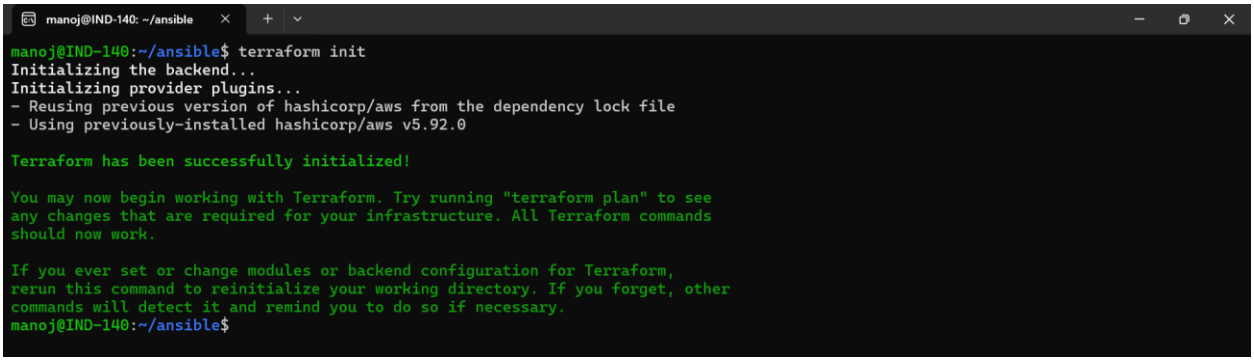
3. Place your `ansible.pem` key file in the same directory
4. Ensure the key file has the correct permissions: `chmod 400 ansible.pem`

```
manoj@IND-140:~/ansible$ chmod 400 ansible.pem
manoj@IND-140:~/ansible$
```

Step 2: Initialize Terraform

1. Open a terminal in your project directory

2. Run the initialization command: `terraform init`
   This will download the necessary AWS provider plugins



Step 3: Validate the Configuration

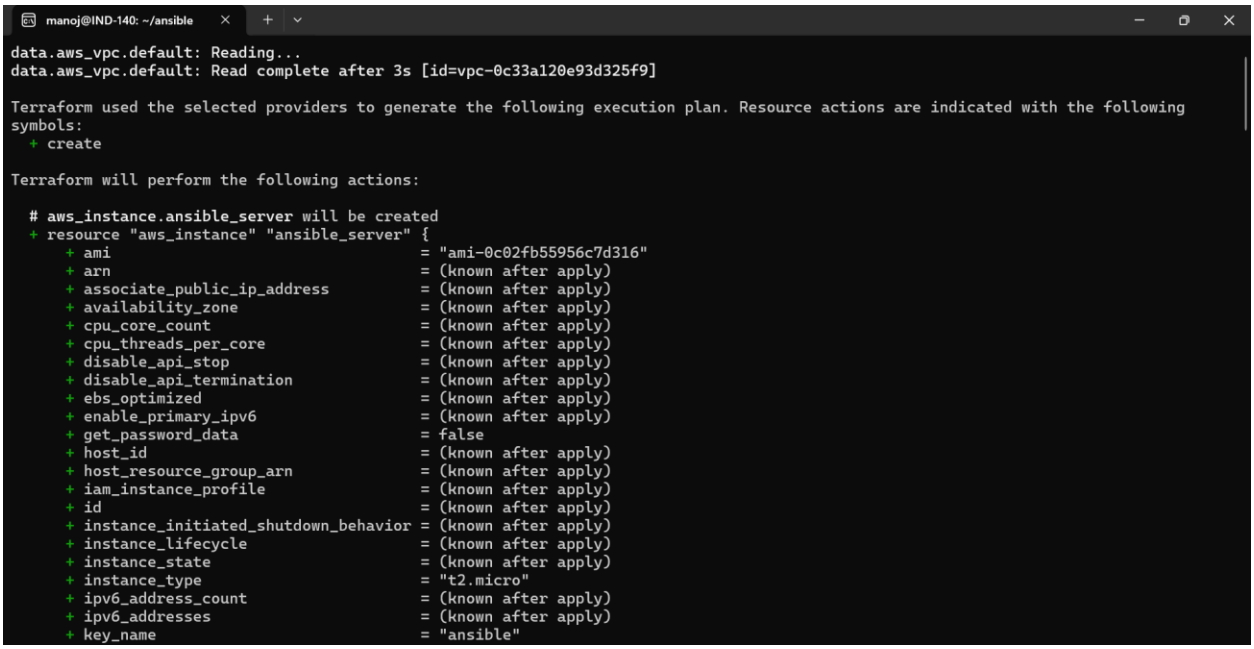
1. Run the validation command to check for syntax errors: `terraform validate`

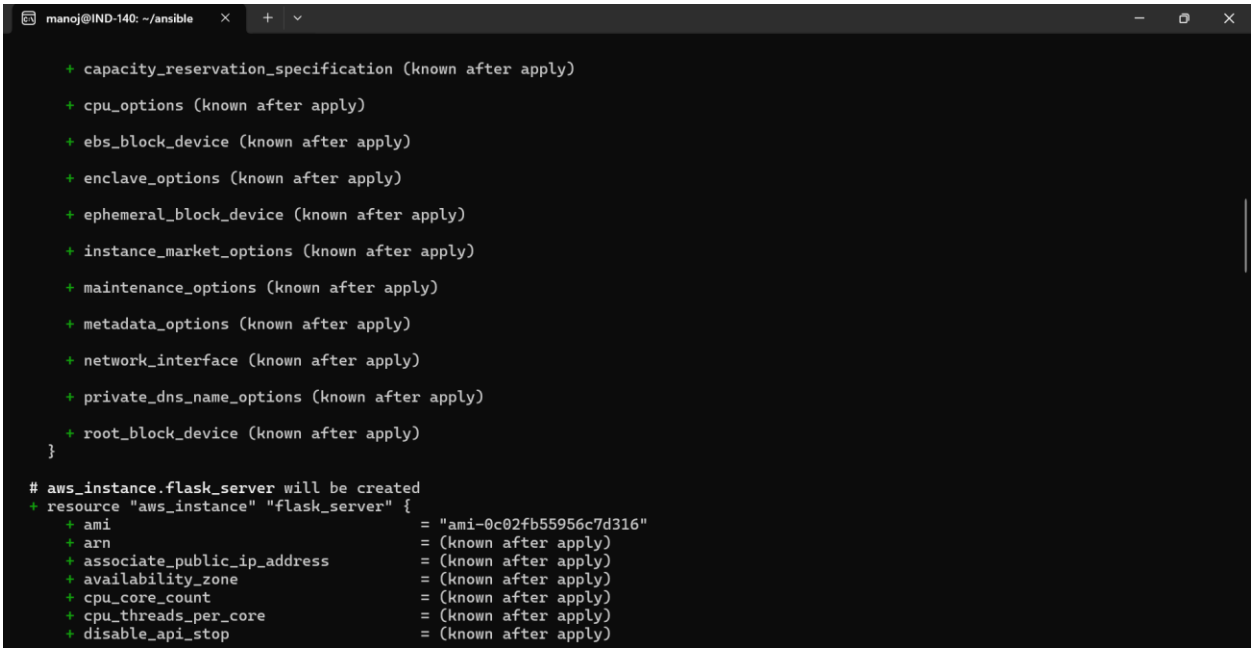
2. If successful, proceed to the next step

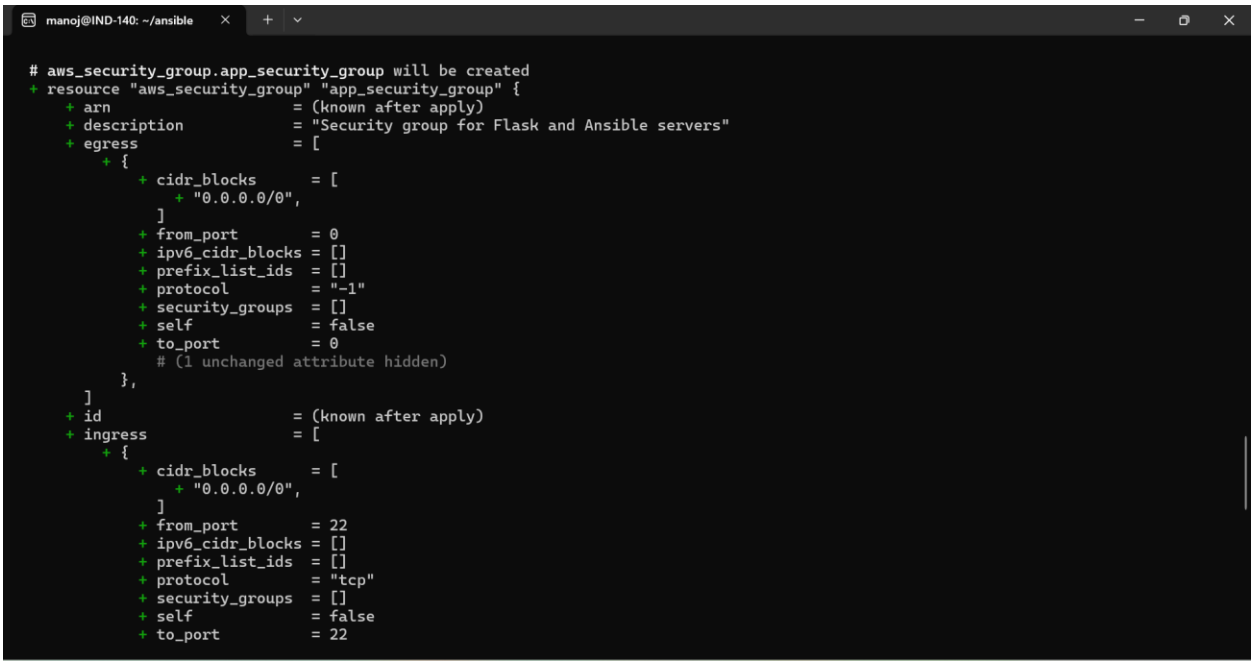
Step 4: Review the Execution Plan

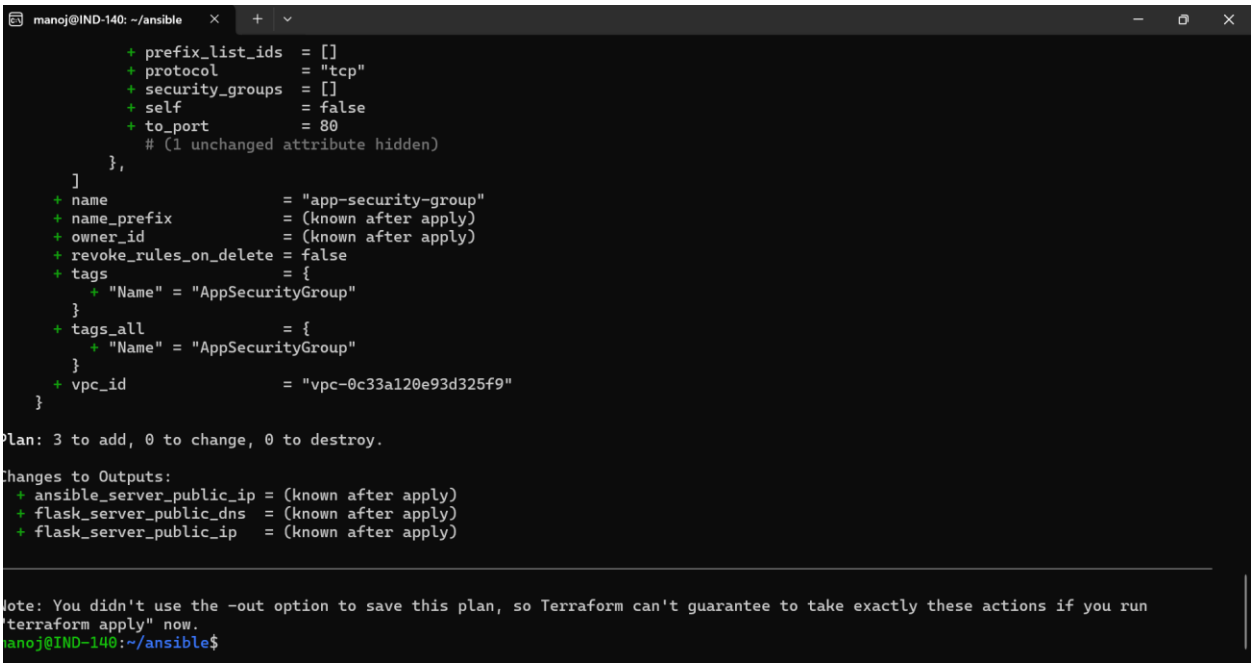
1. Generate and review the execution plan: `terraform plan`

2. Verify that Terraform will create:
   a. An AWS security group
   b. A Flask server instance
   c. An Ansible server instance


manoj@IND-140: ~/ansible
data.aws_vpc.default: Reading...
data.aws_vpc.default: Read complete after 3s [id=vpc-0c33a120e93d325f9]
Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:
+ create
Terraform will perform the following actions:
# aws_instance.ansible_server will be created
+ resource "aws_instance" "ansible_server" {
+ ami = "ami-0c02fb55956c7d316"
+ arn = (known after apply)
+ associate_public_ip_address = (known after apply)
+ availability_zone = (known after apply)
+ cpu_core_count = (known after apply)
+ cpu_threads_per_core = (known after apply)
+ disable_api_stop = (known after apply)
+ disable_api_termination = (known after apply)
+ ebs_optimized = (known after apply)
+ enable_primary_ipv6 = (known after apply)
+ get_password_data = false
+ host_id = (known after apply)
+ host_resource_group_arn = (known after apply)
+ iam_instance_profile = (known after apply)
+ id = (known after apply)
+ instance_initiated_shutdown_behavior = (known after apply)
+ instance_lifecycle = (known after apply)
+ instance_state = (known after apply)
+ instance_type = "t2.micro"
+ ipv6_address_count = (known after apply)
+ ipv6_addresses = (known after apply)
+ key_name = "ansible"

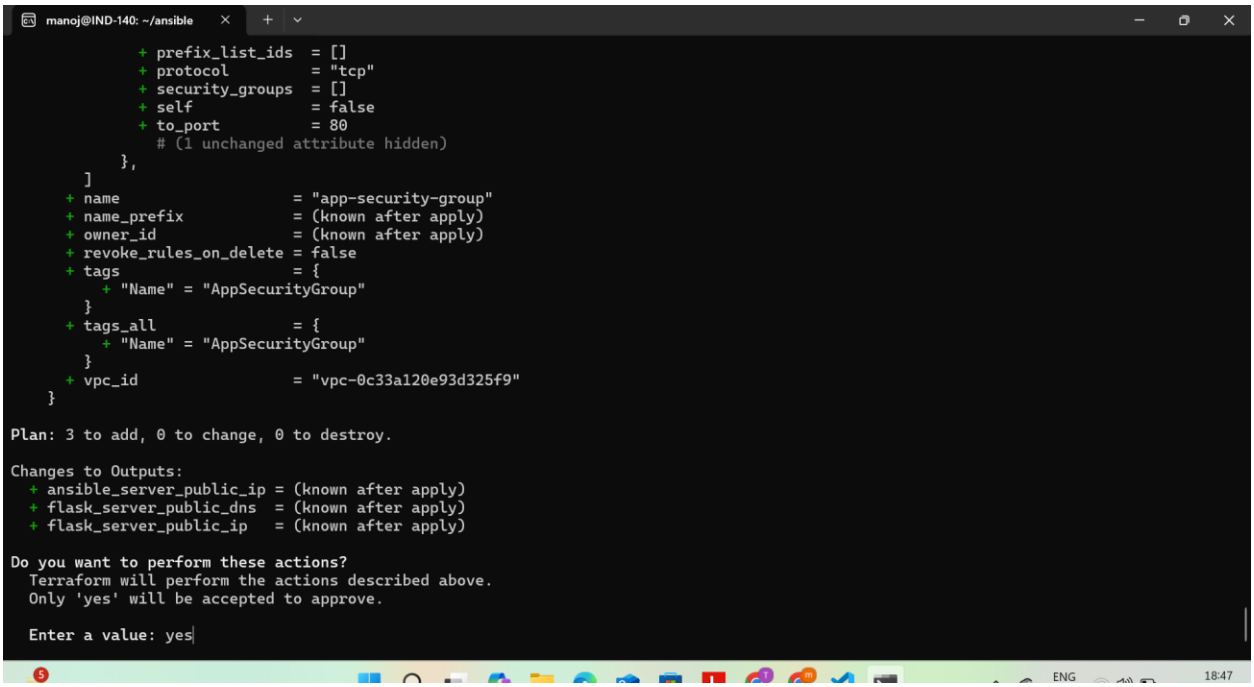

manoj@IND-140: ~/ansible
+ capacity_reservation_specification (known after apply)
+ cpu_options (known after apply)
+ ebs_block_device (known after apply)
+ enclave_options (known after apply)
+ ephemeral_block_device (known after apply)
+ instance_market_options (known after apply)
+ maintenance_options (known after apply)
+ metadata_options (known after apply)
+ network_interface (known after apply)
+ private_dns_name_options (known after apply)
+ root_block_device (known after apply)
}
# aws_instance.flask_server will be created
+ resource "aws_instance" "flask_server" {
+ ami = "ami-0c02fb55956c7d316"
+ arn = (known after apply)
+ associate_public_ip_address = (known after apply)
+ availability_zone = (known after apply)
+ cpu_core_count = (known after apply)
+ cpu_threads_per_core = (known after apply)
+ disable_api_stop = (known after apply)



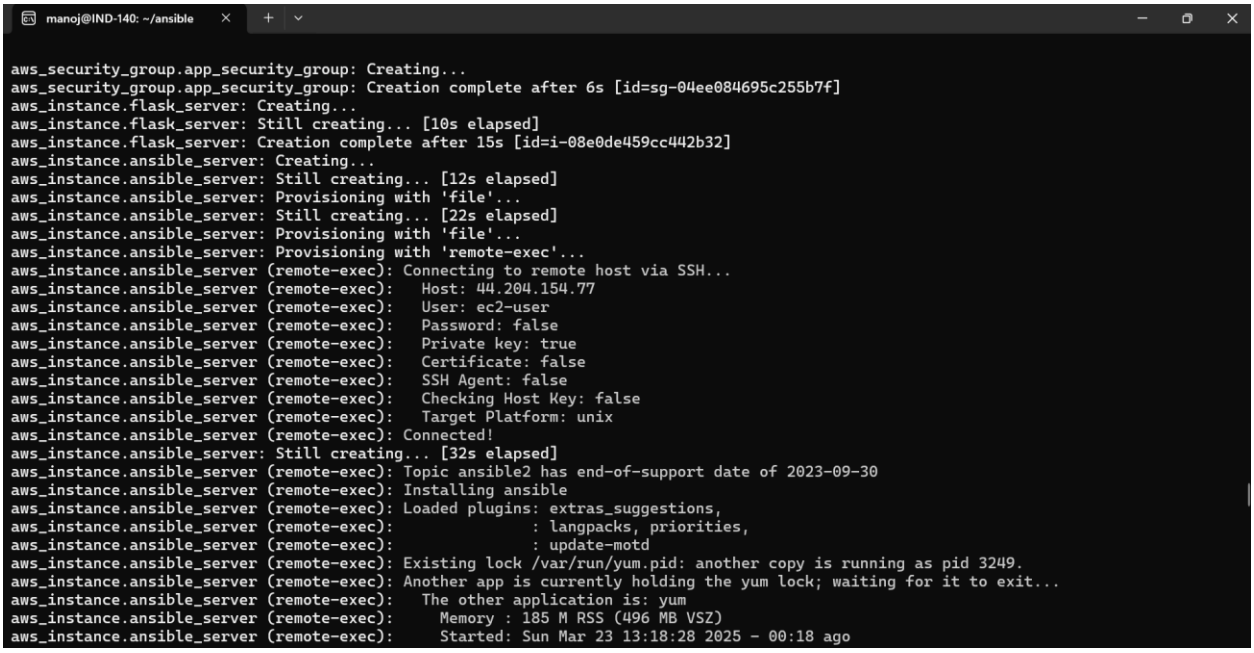


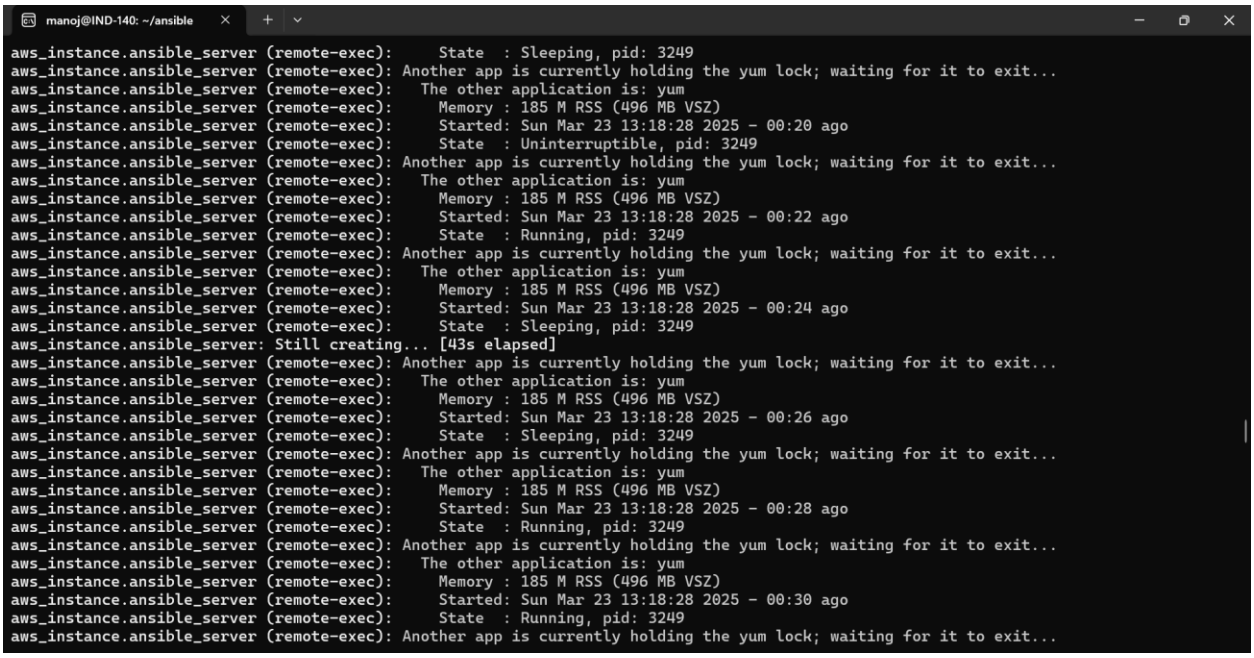
Step 5: Apply the Configuration

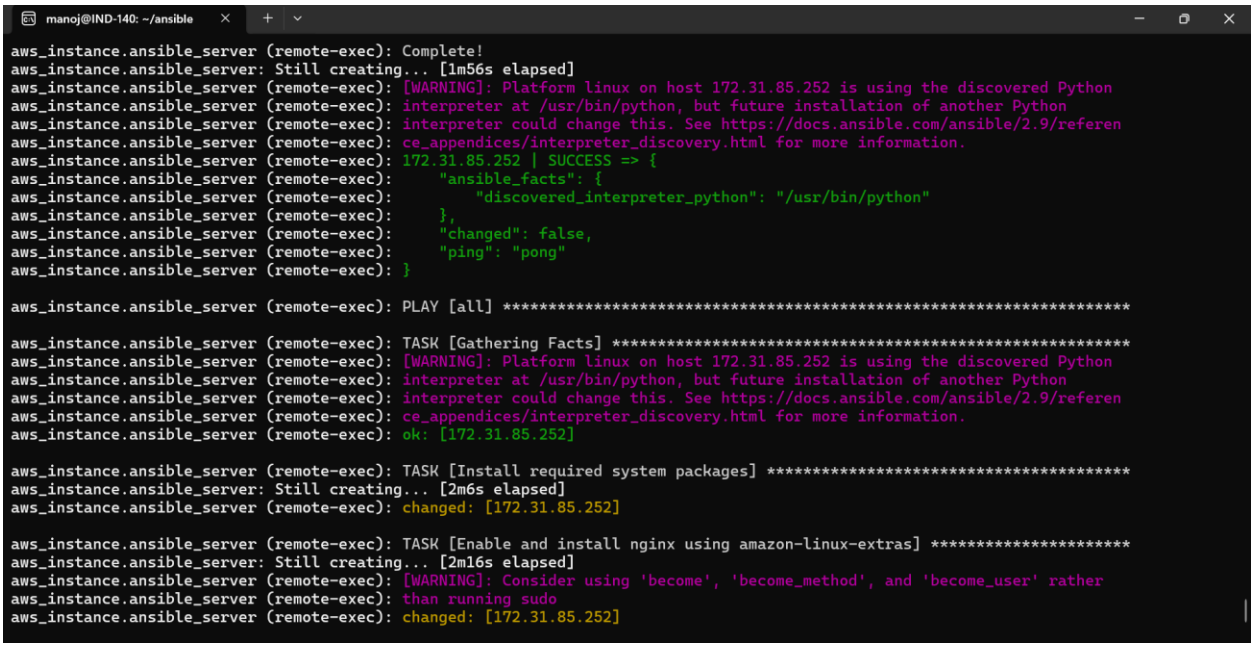
1. Apply the Terraform configuration: `terraform apply`
2. Type `yes` when prompted to confirm



3. Wait for the infrastructure deployment (this may take several minutes)




manoj@IND-140: ~/ansible
aws_instance.ansible_server (remote-exec):   State : Sleeping, pid: 3249
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...
aws_instance.ansible_server (remote-exec):   The other application is: yum
aws_instance.ansible_server (remote-exec):   Memory : 185 M RSS (496 MB VSZ)
aws_instance.ansible_server (remote-exec):   Started: Sun Mar 23 13:18:28 2025 - 00:20 ago
aws_instance.ansible_server (remote-exec):   State : Uninterruptible, pid: 3249
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...
aws_instance.ansible_server (remote-exec):   The other application is: yum
aws_instance.ansible_server (remote-exec):   Memory : 185 M RSS (496 MB VSZ)
aws_instance.ansible_server (remote-exec):   Started: Sun Mar 23 13:18:28 2025 - 00:22 ago
aws_instance.ansible_server (remote-exec):   State : Running, pid: 3249
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...
aws_instance.ansible_server (remote-exec):   The other application is: yum
aws_instance.ansible_server (remote-exec):   Memory : 185 M RSS (496 MB VSZ)
aws_instance.ansible_server (remote-exec):   Started: Sun Mar 23 13:18:28 2025 - 00:24 ago
aws_instance.ansible_server (remote-exec):   State : Sleeping, pid: 3249
aws_instance.ansible_server: Still creating... [43s elapsed]
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...
aws_instance.ansible_server (remote-exec):   The other application is: yum
aws_instance.ansible_server (remote-exec):   Memory : 185 M RSS (496 MB VSZ)
aws_instance.ansible_server (remote-exec):   Started: Sun Mar 23 13:18:28 2025 - 00:26 ago
aws_instance.ansible_server (remote-exec):   State : Sleeping, pid: 3249
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...
aws_instance.ansible_server (remote-exec):   The other application is: yum
aws_instance.ansible_server (remote-exec):   Memory : 185 M RSS (496 MB VSZ)
aws_instance.ansible_server (remote-exec):   Started: Sun Mar 23 13:18:28 2025 - 00:28 ago
aws_instance.ansible_server (remote-exec):   State : Running, pid: 3249
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...
aws_instance.ansible_server (remote-exec):   The other application is: yum
aws_instance.ansible_server (remote-exec):   Memory : 185 M RSS (496 MB VSZ)
aws_instance.ansible_server (remote-exec):   Started: Sun Mar 23 13:18:28 2025 - 00:30 ago
aws_instance.ansible_server (remote-exec):   State : Running, pid: 3249
aws_instance.ansible_server (remote-exec): Another app is currently holding the yum lock; waiting for it to exit...

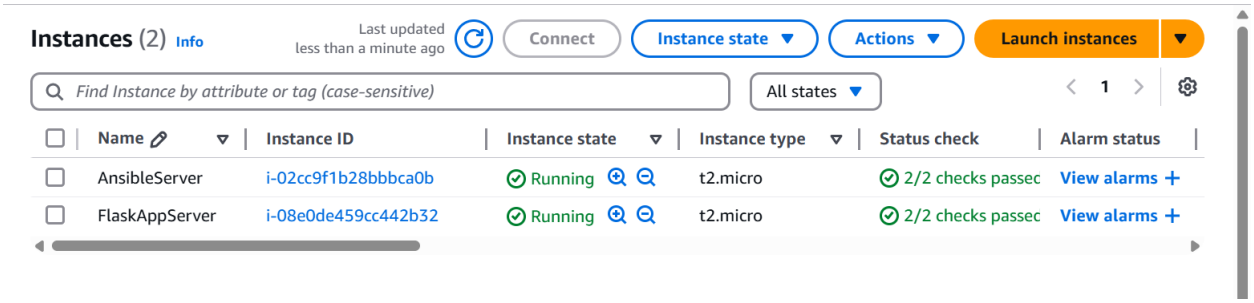

manoj@IND-140: ~/ansible
aws_instance.ansible_server (remote-exec): Complete!
aws_instance.ansible_server: Still creating... [1m56s elapsed]
aws_instance.ansible_server (remote-exec): [WARNING]: Platform linux on host 172.31.85.252 is using the discovered Python
aws_instance.ansible_server (remote-exec): interpreter at /usr/bin/python, but future installation of another Python
aws_instance.ansible_server (remote-exec): interpreter could change this. See https://docs.ansible.com/ansible/2.9/referen
aws_instance.ansible_server (remote-exec): ce_appendices/interpreter_discovery.html for more information.
aws_instance.ansible_server (remote-exec): 172.31.85.252 | SUCCESS => {
aws_instance.ansible_server (remote-exec):   "ansible_facts": {
aws_instance.ansible_server (remote-exec):   "discovered_interpreter_python": "/usr/bin/python"
aws_instance.ansible_server (remote-exec):   },
aws_instance.ansible_server (remote-exec):   "changed": false,
aws_instance.ansible_server (remote-exec):   "ping": "pong"
aws_instance.ansible_server (remote-exec): }
aws_instance.ansible_server (remote-exec): PLAY [all] ***
aws_instance.ansible_server (remote-exec): TASK [Gathering Facts] ***
aws_instance.ansible_server (remote-exec): [WARNING]: Platform linux on host 172.31.85.252 is using the discovered Python
aws_instance.ansible_server (remote-exec): interpreter at /usr/bin/python, but future installation of another Python
aws_instance.ansible_server (remote-exec): interpreter could change this. See https://docs.ansible.com/ansible/2.9/referen
aws_instance.ansible_server (remote-exec): ce_appendices/interpreter_discovery.html for more information.
aws_instance.ansible_server (remote-exec): ok: [172.31.85.252]
aws_instance.ansible_server (remote-exec): TASK [Install required system packages] ************************************
aws_instance.ansible_server: Still creating... [2m6s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
aws_instance.ansible_server (remote-exec): TASK [Enable and install nginx using amazon-linux-extras] *********************
aws_instance.ansible_server: Still creating... [2m16s elapsed]
aws_instance.ansible_server (remote-exec): [WARNING]: Consider using 'become', 'become_method', and 'become_user' rather
aws_instance.ansible_server (remote-exec): than running sudo
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

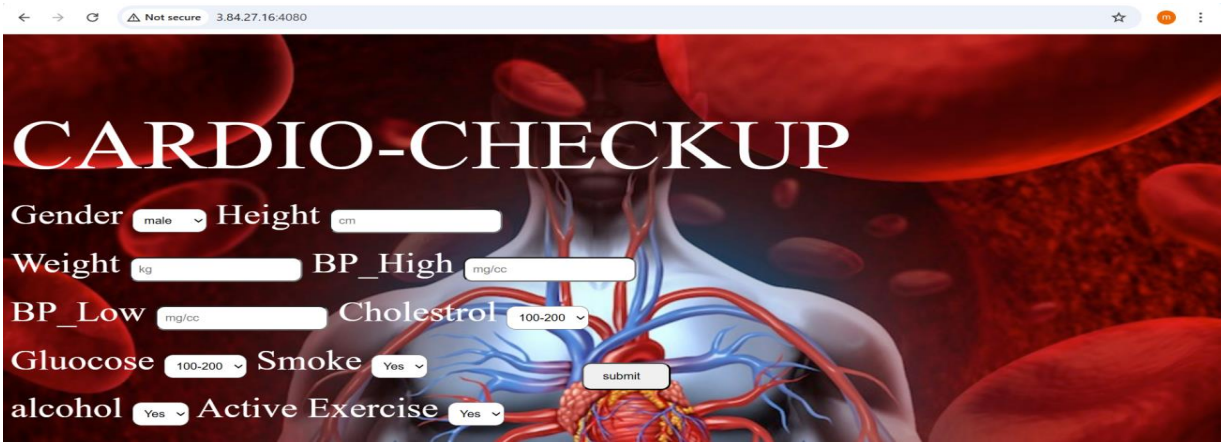
```
manoj@IND-140: ~/ansible
aws_instance.ansible_server (remote-exec): TASK [Install required system packages] ************************************
aws_instance.ansible_server: Still creating... [2m6s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Enable and install nginx using amazon-linux-extras] *********************
aws_instance.ansible_server: Still creating... [2m16s elapsed]
aws_instance.ansible_server (remote-exec): [WARNING]: Consider using 'become', 'become_method', and 'become_user' rather
aws_instance.ansible_server (remote-exec): than running sudo
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Ensure pip is installed] ***********************************************
aws_instance.ansible_server: Still creating... [2m27s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Install Python dependencies] ******************************************
aws_instance.ansible_server: Still creating... [2m37s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Clone Flask application from GitHub] ********************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Configure Gunicorn systemd service] ********************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Reload systemd and enable Gunicorn] ********************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Configure Nginx for Flask] *****************************************
aws_instance.ansible_server: Still creating... [2m47s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]

aws_instance.ansible_server (remote-exec): TASK [Restart Nginx] *****************************************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

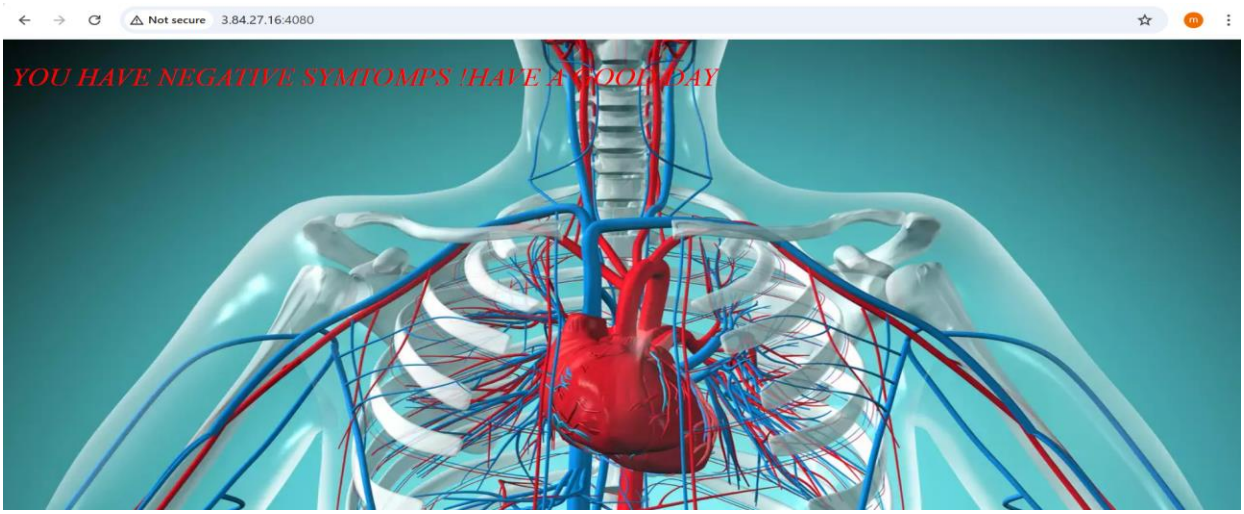
```
aws_instance.ansible_server (remote-exec): PLAY RECAP *********************************************************************
aws_instance.ansible_server (remote-exec): 172.31.85.252             : ok=10   changed=9    unreachable=0    failed=0    skipped=0
  rescued=0    ignored=0

aws_instance.ansible_server: Creation complete after 2m49s [id=i-02cc9f1b28bbbca0b]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

ansible_server_public_ip = "44.204.154.77"
flask_server_public_dns = "ec2-3-84-27-16.compute-1.amazonaws.com"
flask_server_public_ip = "3.84.27.16"
manoj@IND-140:~/ansible$
```

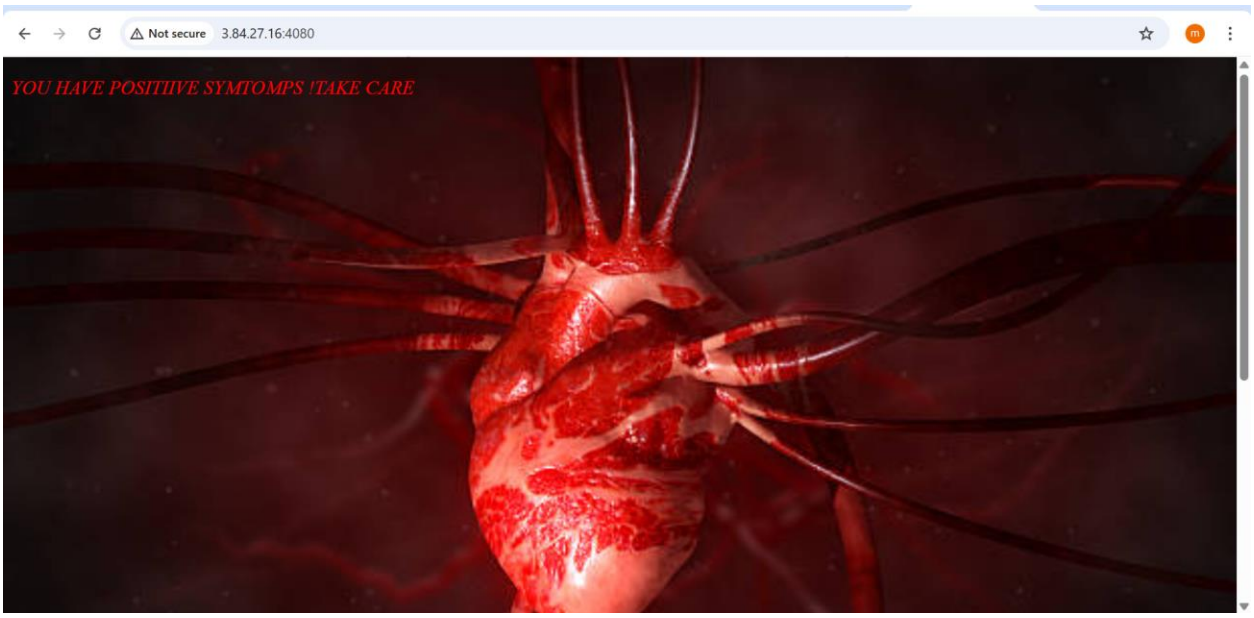
```
manoj@IND-140:~/ansible$ curl 3.84.27.16:4080
<!doctype html>
<html>
    <head>
        <title>
            CARDIO-CHECKUP
        </title>
        <link href= '/static/style.css' rel='stylesheet'>
    </head>
    <body>
        <p >CARDIO-CHECKUP</p>

        <form actions='#' method='POST'>
            <div class='first'>
                <label for='gender'>Gender</label>
                <select name='gender' required>
                    <option value='2'>male</option>
                    <option value='1'>female</option>
                </select>
                <label for='height'>Height</label>
                <input type='text' name='height' placeholder='cm' required/>
            </div>
            <div class='second'>
                <label for='weight'>Weight</label>
                <input type='text' name='weight' placeholder='kg' required/>
                <label for='bp_high'>BP_High</label>
                <input type='text' name='bp_high' required placeholder='mg/cc'/>

            </div>
            <div class='third'>
                <label for='bp_low'>BP_Low</label>
                <input type='text' name='bp_low' required placeholder='mg/cc' />
```

Output :


Instances (2) Info
Last updated less than a minute ago
Connect
Instance state
Actions
Launch instances
Find Instance by attribute or tag (case-sensitive)
All states
1
Name
Instance ID
Instance state
Instance type
Status check
Alarm status
AnsibleServer
i-02cc9f1b28bbbca0b
Running
t2.micro
2/2 checks passed
View alarms
FlaskAppServer
i-08e0de459cc442b32
Running
t2.micro
2/2 checks passed
View alarms


Not secure
3.84.27.16:4080
CARDIO-CHECKUP
Gender
male
Height
cm
Weight
kg
BP_High
mg/cc
BP_Low
mg/cc
Cholestrol
100-200
Gluocose
100-200
Smoke
Yes
submit
alcohol
Yes
Active Exercise
Yes


Not secure
3.84.27.16:4080
YOU HAVE NEGATIVE SYMTOMPS !HAVE A GOOD DAY



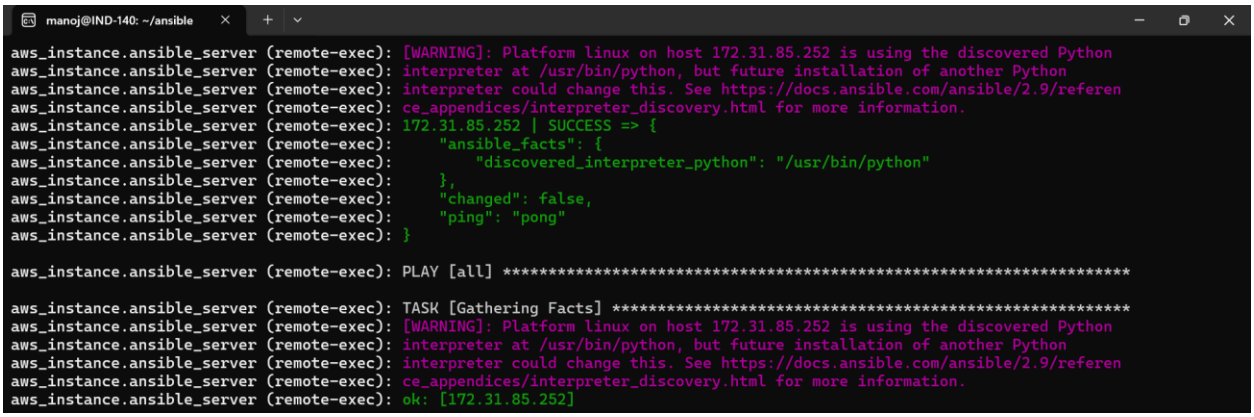
Ansible Playbook Explanation

The playbook (`deploy_flask_app.yml`) is created on the Ansible server and contains instructions for configuring the Flask server. Here's a detailed breakdown:

Playbook Header and Variables

```
- hosts: all
  user: ec2-user
  become: yes
  vars:
    app_port: 4080
```

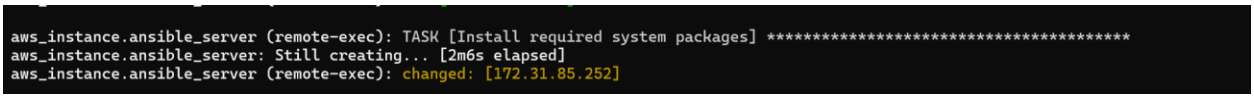
- `hosts: all`: Targets all servers in the inventory (in this case, just the Flask server)
- `user: ec2-user`: Connects as the ec2-user (default Amazon Linux user)
- `become: yes`: Executes tasks with elevated (sudo) privileges
- `vars`: Defines variables - here setting the application port to 4080



System Package Installation

```
- name: Install required system packages
  yum:
    name:
      - python3
      - python3-pip
      - git
    state: present
```

- Installs Python 3, pip package manager, and git using the yum package manager
- These are the base requirements for running a Python Flask application



Nginx Installation

```
- name: Enable and install nginx using amazon-linux-extras
  shell: |
    sudo amazon-linux-extras enable nginx1
    sudo yum clean metadata
    sudo yum install -y nginx
```

- Uses Amazon Linux Extras to install Nginx
- This module provides access to newer versions of software on the stable Amazon Linux platform
- Cleans metadata and installs Nginx web server

```
aws_instance.ansible_server (remote-exec): TASK [Enable and install nginx using amazon-linux-extras] *********************
aws_instance.ansible_server: Still creating... [2m16s elapsed]
aws_instance.ansible_server (remote-exec): [WARNING]: Consider using 'become', 'become_method', and 'become_user' rather
aws_instance.ansible_server (remote-exec): than running sudo
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Python Dependencies Setup

```
- name: Ensure pip is installed
  pip:
    name: pip
    state: latest
    executable: pip3

- name: Install Python dependencies
  pip:
    name:
      - flask
      - gunicorn
      - joblib
      - scikit-learn
    executable: pip3
```

- Updates pip to the latest version
- Installs the Python packages required for the application:
    - Flask: Web framework
    - Gunicorn: WSGI HTTP server
    - joblib: Library for saving/loading scikit-learn models
    - scikit-learn: Machine learning library for the heart disease prediction model

```
aws_instance.ansible_server (remote-exec): TASK [Ensure pip is installed] ********************************************
aws_instance.ansible_server: Still creating... [2m27s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

```
aws_instance.ansible_server (remote-exec): TASK [Install Python dependencies] ****************************************
aws_instance.ansible_server: Still creating... [2m37s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Application Deployment

```
- name: Clone Flask application from GitHub
  git:
    repo: https://github.com/manojmanu9441/cardio_heart_detection.git
    dest: /home/ec2-user/cardio_heart_detection
```

```
    version: main
```

- Uses git to clone the Flask application repository
- Places it in the ec2-user's home directory under "cardio_heart_detection"
- Specifically uses the main branch of the repository

```
aws_instance.ansible_server (remote-exec): TASK [Clone Flask application from GitHub] ***********************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Gunicorn Service Configuration

```
- name: Configure Gunicorn systemd service
  copy:
    dest: /etc/systemd/system/gunicorn.service
    content: |
      [Unit]
      Description=Gunicorn instance to serve Flask app
      After=network.target

      [Service]
      User=ec2-user
      Group=ec2-user
      WorkingDirectory=/home/ec2-user/cardio_heart_detection
      Environment="FLASK_RUN_PORT={{ app_port }}"
      ExecStart=/usr/local/bin/gunicorn --workers 3 --bind
0.0.0.0:{{ app_port }} main:app

      [Install]
      WantedBy=multi-user.target
```

- Creates a systemd service file for Gunicorn
- Configures Gunicorn to:
    - Run as the ec2-user
    - Use the application directory as its working directory
    - Start after the network is available
    - Run with 3 worker processes
    - Bind to the configured port (4080)
    - Serve the Flask app from main.py (main:app)

```
aws_instance.ansible_server (remote-exec): TASK [Configure Gunicorn systemd service] ************************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Gunicorn Service Activation

```
- name: Reload systemd and enable Gunicorn
  systemd:
    name: gunicorn
    enabled: yes
    state: restarted
    daemon_reload: yes
```

- Reloads systemd to recognize the new service
- Enables the Gunicorn service to start on boot
- Ensures the service is running (restarted)

```
aws_instance.ansible_server (remote-exec): TASK [Reload systemd and enable Gunicorn] ************************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Nginx Configuration

```
- name: Configure Nginx for Flask
  copy:
    dest: /etc/nginx/conf.d/flaskapp.conf
    content: |
      server {
          listen 80;
          server_name _;

          location / {
              proxy_pass http://127.0.0.1:{{ app_port }};
              proxy_set_header Host $host;
              proxy_set_header X-Real-IP $remote_addr;
              proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
              proxy_set_header X-Forwarded-Proto $scheme;
          }
      }
```

- Creates an Nginx server configuration file
- Configures Nginx to:
    - Listen on port 80 (standard HTTP port)
    - Accept connections for any server name
    - Proxy all requests to the Gunicorn server running on localhost:4080
    - Pass HTTP headers to preserve client information

```
aws_instance.ansible_server (remote-exec): TASK [Configure Nginx for Flask] ********************************************
aws_instance.ansible_server: Still creating... [2m47s elapsed]
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Nginx Activation

```
- name: Restart Nginx
  systemd:
    name: nginx
    state: restarted
    enabled: yes
```

- Ensures Nginx is enabled to start on boot
- Restarts Nginx to apply the new configuration

```
aws_instance.ansible_server (remote-exec): TASK [Restart Nginx] ****************************************************
aws_instance.ansible_server (remote-exec): changed: [172.31.85.252]
```

Playbook Execution Flow

The playbook follows a logical progression:

1. Installs system prerequisites
2. Sets up web server (Nginx)
3. Installs Python dependencies
4. Deploys the application code
5. Configures the application server (Gunicorn)
6. Sets up the reverse proxy (Nginx)
7. Starts all services

```
aws_instance.ansible_server (remote-exec): PLAY RECAP *********************************************************************
aws_instance.ansible_server (remote-exec): 172.31.85.252             : ok=10   changed=9    unreachable=0    failed=0    skipped=0
  rescued=0    ignored=0

aws_instance.ansible_server: Creation complete after 2m49s [id=i-02cc9f1b28bbbca0b]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

ansible_server_public_ip = "44.204.154.77"
flask_server_public_dns = "ec2-3-84-27-16.compute-1.amazonaws.com"
flask_server_public_ip = "3.84.27.16"
```

Step 6: Verify the Deployment

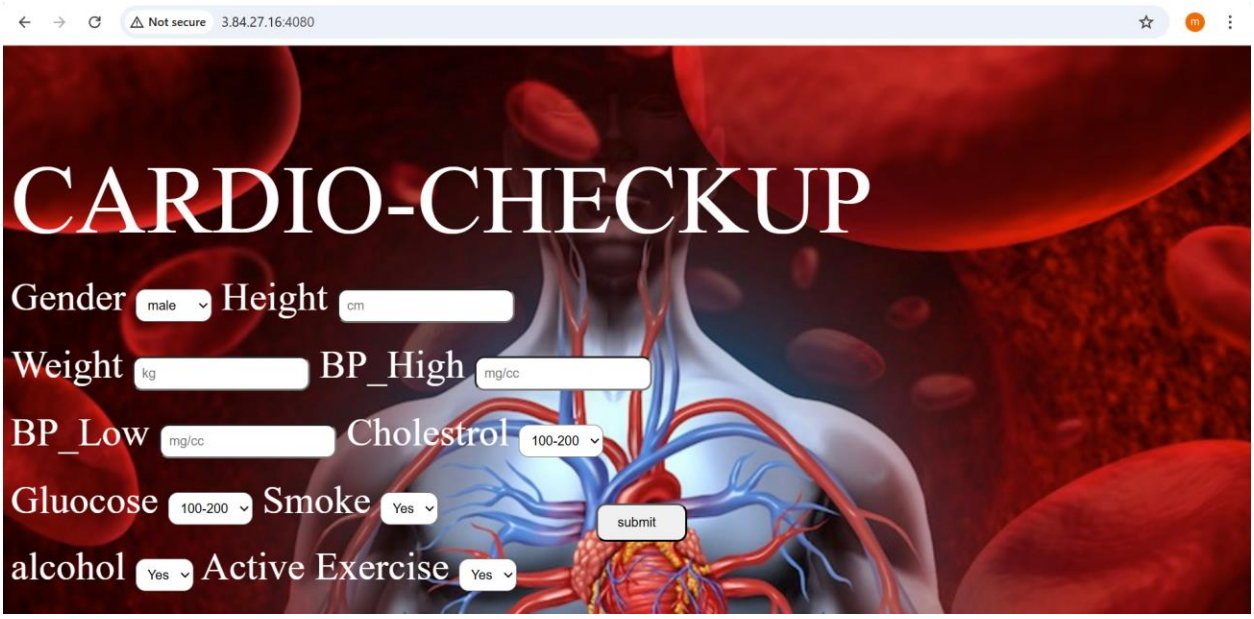
1. Once the deployment is complete, Terraform will output:
    a. The Flask server's public IP address
    b. The Flask server's public DNS name
    c. The Ansible server's public IP address
2. You can access these values later with: `terraform output`

```
</html>manoj@IND-140:~/ansible$ terraform output
ansible_server_public_ip = "44.204.154.77"
flask_server_public_dns = "ec2-3-84-27-16.compute-1.amazonaws.com"
flask_server_public_ip = "3.84.27.16"
manoj@IND-140:~/ansible$
```

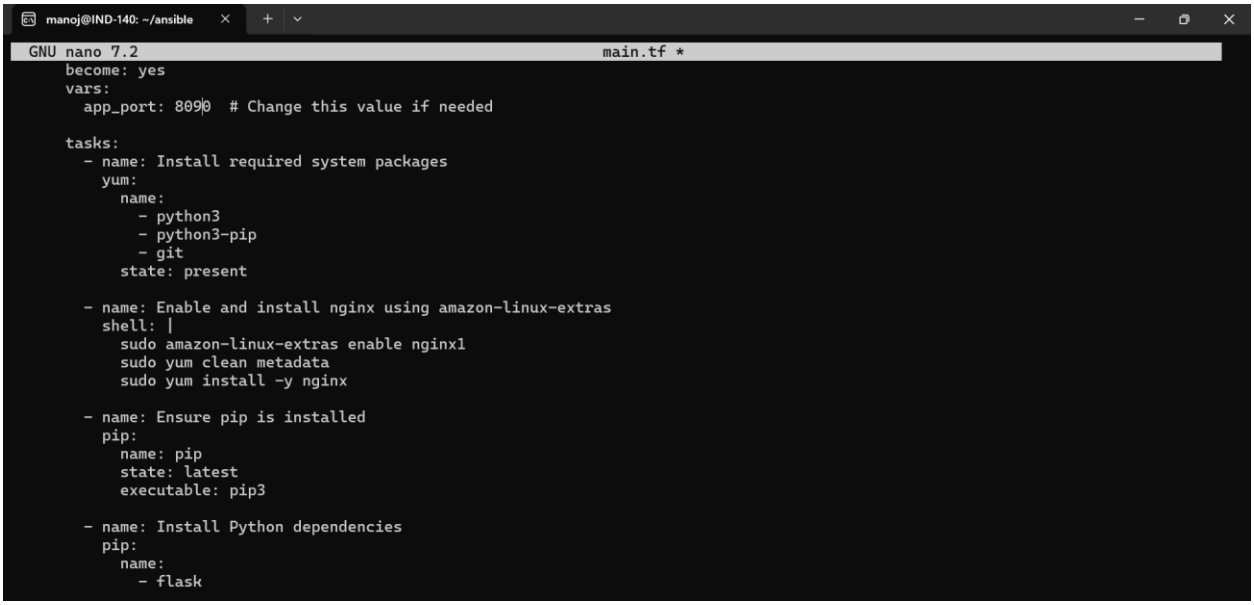
Step 7: Access Your Flask Application

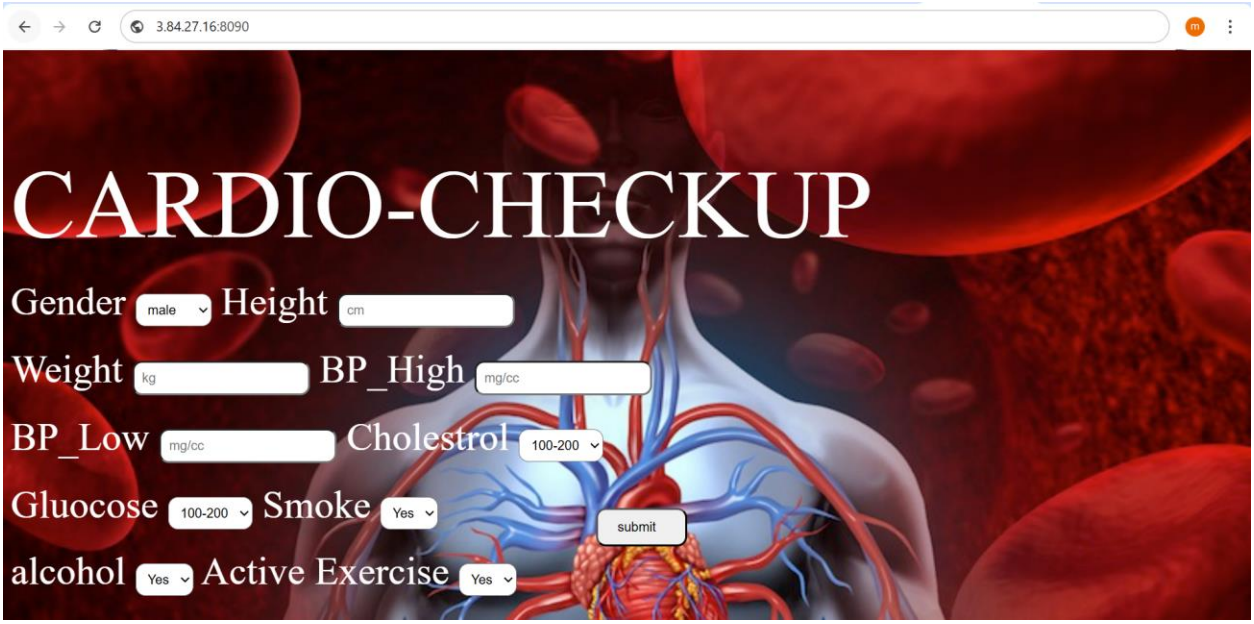
1. Open a web browser
2. Navigate to the Flask server's public DNS name or IP address
   http://<flask_server_public_dns>

3. The cardio heart detection application should be available



Port Change Verfication





CHANGED PORT TO 8090

Step 9: Clean Up Resources (when done)

1. To destroy all resources created by Terraform: `terraform destroy`

2. Type `yes` when prompted to confirm

```
Do you really want to destroy all resources?
  Terraform will destroy all your managed infrastructure, as shown above.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes
```

```
aws_instance.ansible_server: Destroying... [id=i-02cc9f1b28bbbca0b]
aws_instance.ansible_server: Still destroying... [id=i-02cc9f1b28bbbca0b, 11s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-02cc9f1b28bbbca0b, 21s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-02cc9f1b28bbbca0b, 31s elapsed]
aws_instance.ansible_server: Still destroying... [id=i-02cc9f1b28bbbca0b, 42s elapsed]
aws_instance.ansible_server: Destruction complete after 45s
aws_instance.flask_server: Destroying... [id=i-08e0de459cc442b32]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 10s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 22s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 32s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 42s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 52s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 1m4s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 1m14s elapsed]
aws_instance.flask_server: Still destroying... [id=i-08e0de459cc442b32, 1m24s elapsed]
aws_instance.flask_server: Destruction complete after 1m27s
aws_security_group.app_security_group: Destroying... [id=sg-04ee084695c255b7f]
aws_security_group.app_security_group: Destruction complete after 3s

Destroy complete! Resources: 3 destroyed.
```

Terraform Provider Configuration

```
provider "aws" {
  region = "us-east-1"
}
```

- **Purpose**: Configures Terraform to use the AWS provider
- **Details**: Sets the AWS region to US East (N. Virginia)
- **Significance**: All resources will be created in this region

VPC Data Source

```
data "aws_vpc" "default" {
  default = true
}
```

- **Purpose**: Retrieves information about the default VPC
- **Details**: Uses a data source to fetch existing infrastructure instead of creating new
- **Significance**: Allows placing resources in the existing default VPC without creating a new one

Security Group Resource

```
resource "aws_security_group" "app_security_group" {
  name        = "app-security-group"
  description = "Security group for Flask and Ansible servers"
  vpc_id      = data.aws_vpc.default.id

  # Ingress rules (inbound traffic)
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
```

```
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 4080
    to_port     = 4080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Egress rule (outbound traffic)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "AppSecurityGroup"
  }
}
```

- **Purpose**: Creates a firewall for the EC2 instances
- **Details**:
    - Creates a security group in the default VPC
    - Opens three inbound ports: 22 (SSH), 80 (HTTP), and 4080 (Application)
    - Allows all outbound traffic
    - Tags the security group for identification
- **Significance**: Controls what traffic can reach and leave the servers

Flask Server EC2 Instance

```
resource "aws_instance" "flask_server" {
  ami                    = "ami-0c02fb55956c7d316"  # Amazon Linux 2
AMI
  instance_type          = "t2.micro"
  key_name               = "ansible"  # Replace with your key pair
  vpc_security_group_ids = [aws_security_group.app_security_group.id]

  tags = {
    Name = "FlaskAppServer"
  }
}
```

- **Purpose**: Creates the EC2 instance for the Flask application
- **Details**:
  - Uses Amazon Linux 2 AMI
  - Uses t2.micro instance type (eligible for free tier)
  - References the "ansible" key pair for SSH access
  - Associates the previously created security group
  - Tags the instance as "FlaskAppServer"
- **Significance**: This is the server that will run the Flask application

Ansible Server EC2 Instance

```
resource "aws_instance" "ansible_server" {
  ami                    = "ami-0c02fb55956c7d316"  # Amazon Linux 2
AMI
  instance_type          = "t2.micro"
  key_name               = "ansible"
  vpc_security_group_ids = [aws_security_group.app_security_group.id]

  # Ensure Ansible instance is created after the Flask server
  depends_on = [aws_instance.flask_server]
```

- **Purpose**: Creates the EC2 instance for Ansible control
- **Details**:

    - Similar configuration to the Flask server
    - Added `depends_on` to ensure the Flask server is created first
- **Significance**: This server will run Ansible to configure the Flask server

File Provisioner for SSH Key

```
# Upload the private key file to Ansible server
 provisioner "file" {
   source      = "ansible.pem"  # Your local key file
   destination = "/home/ec2-user/ansible.pem"

   connection {
     type        = "ssh"
     user        = "ec2-user"
     private_key = file("ansible.pem")
     host        = self.public_ip
   }
 }
```

- **Purpose**: Uploads the SSH key to the Ansible server
- **Details**:
    - Copies the local ansible.pem file to the Ansible server
    - Establishes an SSH connection to do this
- **Significance**: Allows the Ansible server to SSH into the Flask server

File Provisioner for Ansible Playbook

```
# Create the playbook file for Flask deployment
 provisioner "file" {
   content     = <<-EOF
   # Ansible playbook content...
   EOF
   destination = "/home/ec2-user/deploy_flask_app.yml"

   connection {
     type        = "ssh"
     user        = "ec2-user"
```

```
      private_key = file("ansible.pem")
      host        = self.public_ip
    }
  }
```

- **Purpose**: Creates the Ansible playbook on the Ansible server
- **Details**:
  - Uses a heredoc (<<-EOF) to define the content directly in the Terraform file
  - Places the playbook in the ec2-user's home directory
- **Significance**: Defines all the steps to configure the Flask server

Remote-Exec Provisioner

```
 # Configure the Ansible server
  provisioner "remote-exec" {
    inline = [
      "sudo amazon-linux-extras install ansible2 -y",
      "sudo yum install -y git",
      "chmod 400 /home/ec2-user/ansible.pem",

      # Create inventory file with proper SSH key configuration
      "echo '[flask_servers]' > /home/ec2-user/hosts",
      "echo '${aws_instance.flask_server.private_ip} ansible_user=ec2-
user ansible_ssh_private_key_file=/home/ec2-user/ansible.pem
ansible_ssh_common_args=\"-o StrictHostKeyChecking=no\"' >> /home/ec2-
user/hosts",

      # Test connection
      "ansible -i /home/ec2-user/hosts all -m ping",

      # Run the playbook to deploy the Flask application
      "ansible-playbook -i /home/ec2-user/hosts /home/ec2-
user/deploy_flask_app.yml"
    ]

    connection {
      type        = "ssh"
      user        = "ec2-user"
```

```
      private_key = file("ansible.pem")
      host        = self.public_ip
    }
  }

  tags = {
    Name = "AnsibleServer"
  }
}
```

- **Purpose**: Executes commands on the Ansible server after creation
- **Details**:
  - Installs Ansible and Git
  - Sets proper permissions on the SSH key
  - Creates an inventory file listing the Flask server
  - Tests the connection with a ping
  - Runs the Ansible playbook
- **Significance**: Performs the actual configuration and deployment

Output Configuration

```
# Output the public IPs and DNS for easy access
output "flask_server_public_ip" {
  value = aws_instance.flask_server.public_ip
}

output "flask_server_public_dns" {
  value = aws_instance.flask_server.public_dns
}

output "ansible_server_public_ip" {
  value = aws_instance.ansible_server.public_ip
}
```

- **Purpose**: Displays important information after deployment
- **Details**:
  - Outputs the public IP and DNS of the Flask server

    - Outputs the public IP of the Ansible server
- **Significance**: Provides easy access to server addresses without having to look them up in the AWS console

APPENDIX :

MAIN.tf

```
provider "aws" {
  region = "us-east-1"
}

# Fetch the default VPC
data "aws_vpc" "default" {
  default = true
}

# Create a security group for our instances
resource "aws_security_group" "app_security_group" {
  name        = "app-security-group"
  description = "Security group for Flask and Ansible servers"
  vpc_id      = data.aws_vpc.default.id

  # SSH access from anywhere
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # HTTP access for Flask application
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Application port access
  ingress {
    from_port   = 4080
```

```
    to_port     = 4080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow all outbound traffic
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "AppSecurityGroup"
  }
}

# EC2 instance for Flask App Server (Amazon Linux)
resource "aws_instance" "flask_server" {
  ami                    = "ami-0c02fb55956c7d316"  # Amazon Linux 2 AMI
  instance_type          = "t2.micro"
  key_name               = "ansible"  # Replace with your key pair
  vpc_security_group_ids = [aws_security_group.app_security_group.id]

  tags = {
    Name = "FlaskAppServer"
  }
}

# EC2 instance for Ansible Server (Amazon Linux)
resource "aws_instance" "ansible_server" {
  ami                    = "ami-0c02fb55956c7d316"  # Amazon Linux 2 AMI
  instance_type          = "t2.micro"
  key_name               = "ansible"
  vpc_security_group_ids = [aws_security_group.app_security_group.id]

  # Ensure Ansible instance is created after the Flask server
  depends_on = [aws_instance.flask_server]

  # Upload the private key file to Ansible server
  provisioner "file" {
    source      = "ansible.pem"  # Your local key file
```

```
    destination = "/home/ec2-user/ansible.pem"

    connection {
      type        = "ssh"
      user        = "ec2-user"
      private_key = file("ansible.pem")
      host        = self.public_ip
    }
  }

  # Create the playbook file for Flask deployment
  provisioner "file" {
    content     = <<-EOF
    - hosts: all
      user: ec2-user
      become: yes
      vars:
        app_port: 4080  # Change this value if needed

      tasks:
        - name: Install required system packages
          yum:
            name:
              - python3
              - python3-pip
              - git
            state: present

        - name: Enable and install nginx using amazon-linux-extras
          shell: |
            sudo amazon-linux-extras enable nginx1
            sudo yum clean metadata
            sudo yum install -y nginx

        - name: Ensure pip is installed
          pip:
            name: pip
            state: latest
            executable: pip3

        - name: Install Python dependencies
          pip:
            name:
```

```
            - flask
            - gunicorn
            - joblib
            - scikit-learn
          executable: pip3

      - name: Clone Flask application from GitHub
        git:
          repo: https://github.com/manojmanu9441/cardio_heart_detection.git
          dest: /home/ec2-user/cardio_heart_detection
          version: main

      - name: Configure Gunicorn systemd service
        copy:
          dest: /etc/systemd/system/gunicorn.service
          content: |
            [Unit]
            Description=Gunicorn instance to serve Flask app
            After=network.target

            [Service]
            User=ec2-user
            Group=ec2-user
            WorkingDirectory=/home/ec2-user/cardio_heart_detection
            Environment="FLASK_RUN_PORT={{ app_port }}"
            ExecStart=/usr/local/bin/gunicorn --workers 3 --bind
0.0.0.0:{{ app_port }} main:app

            [Install]
            WantedBy=multi-user.target

      - name: Reload systemd and enable Gunicorn
        systemd:
          name: gunicorn
          enabled: yes
          state: restarted
          daemon_reload: yes

      - name: Configure Nginx for Flask
        copy:
          dest: /etc/nginx/conf.d/flaskapp.conf
          content: |
            server {
```

```
                listen 80;
                server_name _;

                location / {
                    proxy_pass http://127.0.0.1:{{ app_port }};
                    proxy_set_header Host $host;
                    proxy_set_header X-Real-IP $remote_addr;
                    proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
                    proxy_set_header X-Forwarded-Proto $scheme;
                }
            }

        - name: Restart Nginx
          systemd:
            name: nginx
            state: restarted
            enabled: yes
    EOF
    destination = "/home/ec2-user/deploy_flask_app.yml"

    connection {
      type        = "ssh"
      user        = "ec2-user"
      private_key = file("ansible.pem")
      host        = self.public_ip
    }
  }

  # Configure the Ansible server
  provisioner "remote-exec" {
    inline = [
      "sudo amazon-linux-extras install ansible2 -y",
      "sudo yum install -y git",
      "chmod 400 /home/ec2-user/ansible.pem",

      # Create inventory file with proper SSH key configuration
      "echo '[flask_servers]' > /home/ec2-user/hosts",
      "echo '${aws_instance.flask_server.private_ip} ansible_user=ec2-user
ansible_ssh_private_key_file=/home/ec2-user/ansible.pem
ansible_ssh_common_args=\"-o StrictHostKeyChecking=no\"' >> /home/ec2-
user/hosts",
```

```
      # Test connection
      "ansible -i /home/ec2-user/hosts all -m ping",

      # Run the playbook to deploy the Flask application
      "ansible-playbook -i /home/ec2-user/hosts /home/ec2-
user/deploy_flask_app.yml"
    ]

    connection {
      type        = "ssh"
      user        = "ec2-user"
      private_key = file("ansible.pem")
      host        = self.public_ip
    }
  }

  tags = {
    Name = "AnsibleServer"
  }
}

# Output the public IPs and DNS for easy access
output "flask_server_public_ip" {
  value = aws_instance.flask_server.public_ip
}

output "flask_server_public_dns" {
  value = aws_instance.flask_server.public_dns
}

output "ansible_server_public_ip" {
  value = aws_instance.ansible_server.public_ip
}
```

ANSIBLE.yml

```
- hosts: all
      user: ec2-user
      become: yes
      vars:
        app_port: 4080  # Change this value if needed
```

```
  tasks:
    - name: Install required system packages
      yum:
        name:
          - python3
          - python3-pip
          - git
        state: present

    - name: Enable and install nginx using amazon-linux-extras
      shell: |
        sudo amazon-linux-extras enable nginx1
        sudo yum clean metadata
        sudo yum install -y nginx

    - name: Ensure pip is installed
      pip:
        name: pip
        state: latest
        executable: pip3

    - name: Install Python dependencies
      pip:
        name:
          - flask
          - gunicorn
          - joblib
          - scikit-learn
        executable: pip3

    - name: Clone Flask application from GitHub
      git:
        repo: https://github.com/manojmanu9441/cardio_heart_detection.git
        dest: /home/ec2-user/cardio_heart_detection
        version: main

    - name: Configure Gunicorn systemd service
      copy:
        dest: /etc/systemd/system/gunicorn.service
        content: |
          [Unit]
          Description=Gunicorn instance to serve Flask app
          After=network.target
```

```
            [Service]
            User=ec2-user
            Group=ec2-user
            WorkingDirectory=/home/ec2-user/cardio_heart_detection
            Environment="FLASK_RUN_PORT={{ app_port }}"
            ExecStart=/usr/local/bin/gunicorn --workers 3 --bind
0.0.0.0:{{ app_port }} main:app

            [Install]
            WantedBy=multi-user.target

        - name: Reload systemd and enable Gunicorn
          systemd:
            name: gunicorn
            enabled: yes
            state: restarted
            daemon_reload: yes

        - name: Configure Nginx for Flask
          copy:
            dest: /etc/nginx/conf.d/flaskapp.conf
            content: |
              server {
                  listen 80;
                  server_name _;

                  location / {
                      proxy_pass http://127.0.0.1:{{ app_port }};
                      proxy_set_header Host $host;
                      proxy_set_header X-Real-IP $remote_addr;
                      proxy_set_header X-Forwarded-For
$proxy_add_x_forwarded_for;
                      proxy_set_header X-Forwarded-Proto $scheme;
                  }
              }

        - name: Restart Nginx
          systemd:
            name: nginx
            state: restarted
            enabled: yes
```