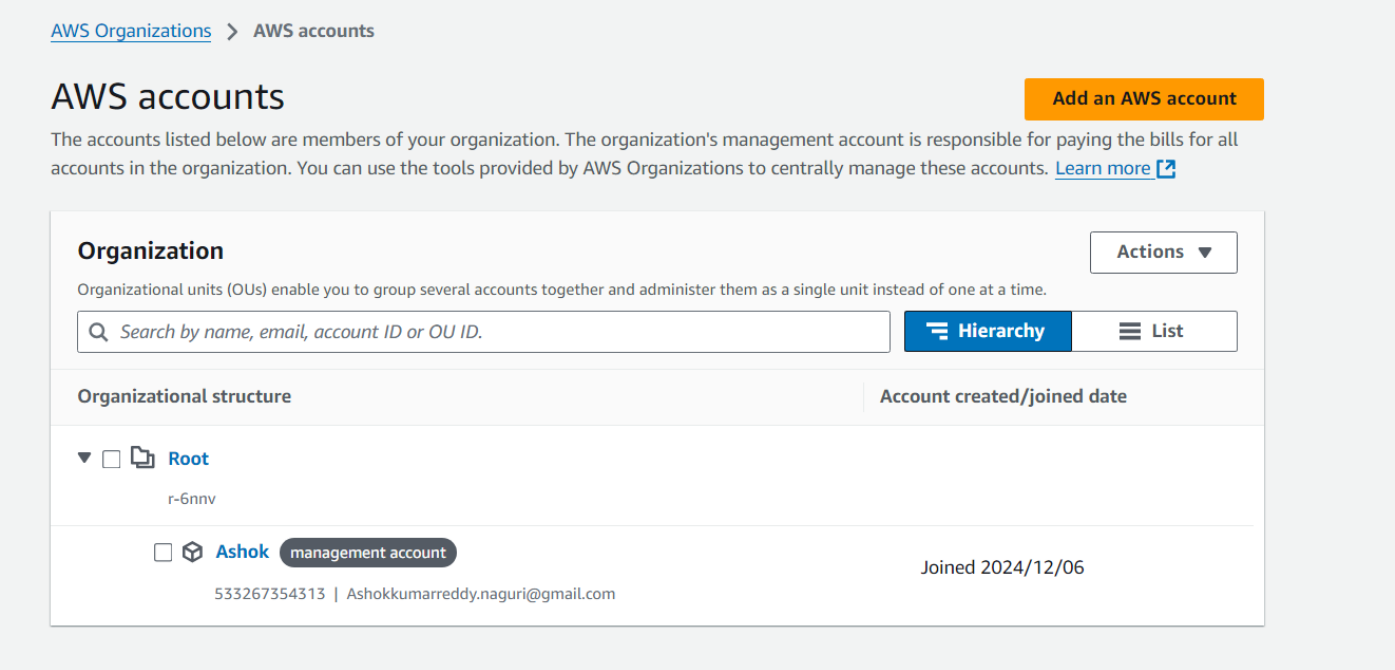
Final Project

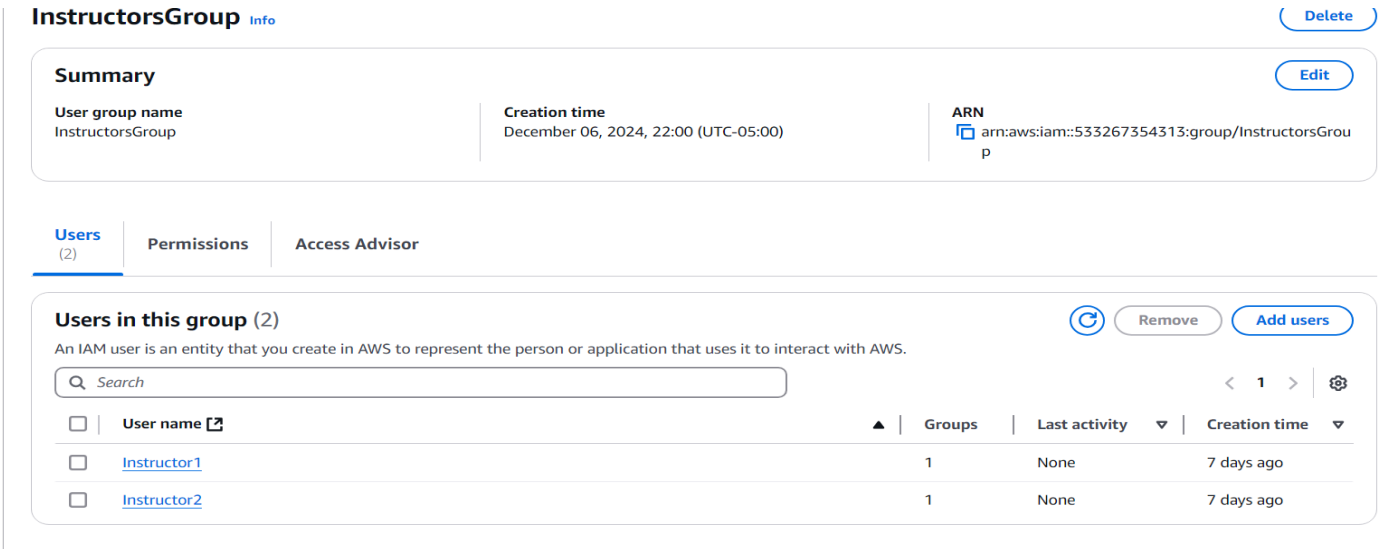
Part - 01

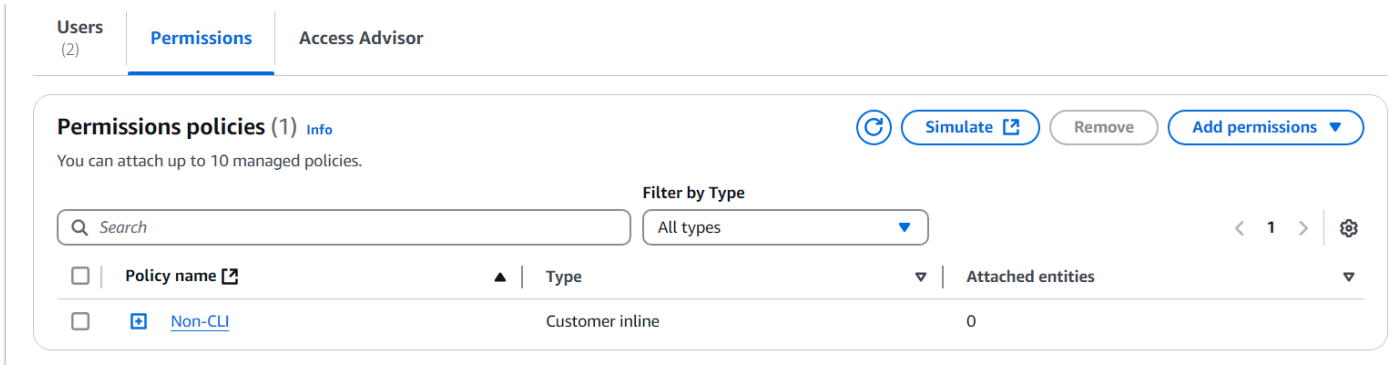
Setting up the environment:

- Creating an AWS org named **root.**



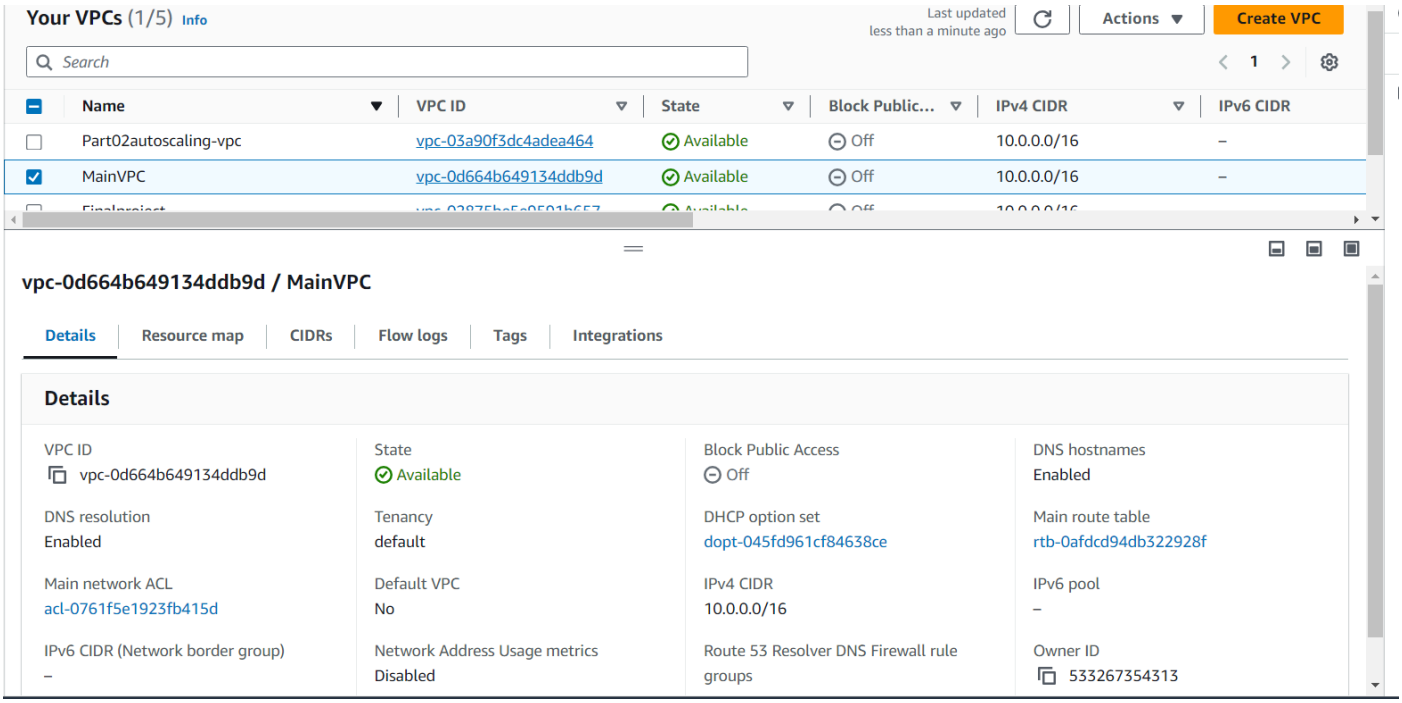
- Now, we create a IAM group for instructors with non-CLI access which means we can give the ReadOnlyAccess to them.





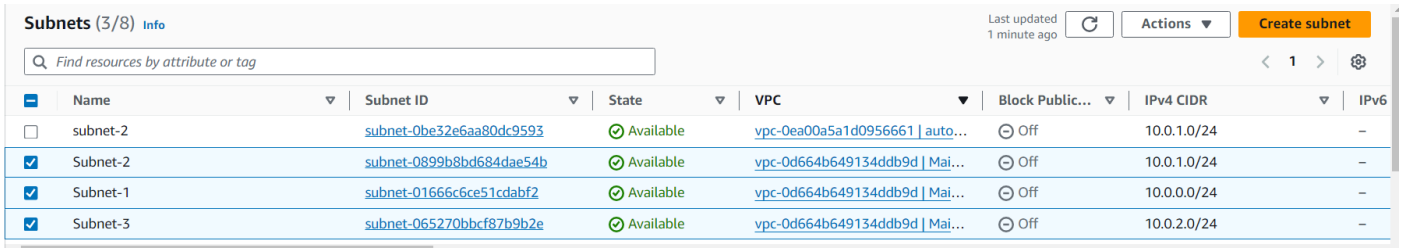
- Now create a VPC with DNS support and DNS hostnames enabled with suitable CIDR with the help of Terraform script.

```
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = {
    Name = "MainVPC"
  }
}
# Data Source for Availability Zones
data "aws_availability_zones" "available" {
  state = "available"
}
```



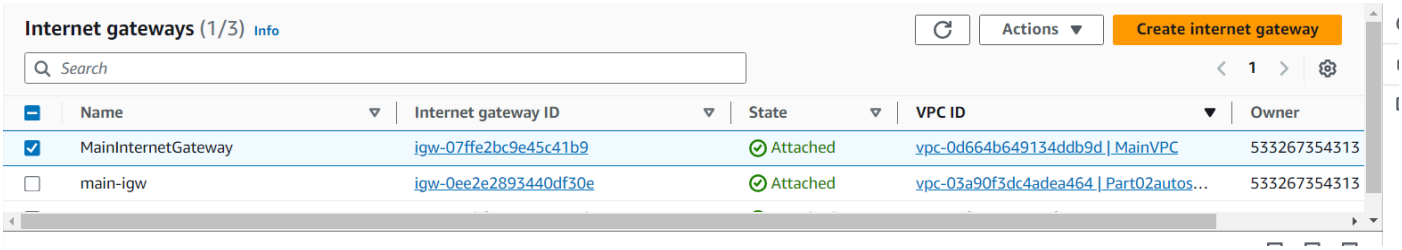
- Creating 3 Subnets of equal sizes which can talk each other

```
# Subnets
resource "aws_subnet" "subnets" {
  count = 3  # Create 3 subnets

  vpc_id                  = aws_vpc.main.id
  cidr_block              = cidrsubnet(aws_vpc.main.cidr_block, 8, count.index)  # Dynamic CIDR blocks
  availability_zone       = element(data.aws_availability_zones.available.names, count.index)
  map_public_ip_on_launch = true

  tags = {
    Name = "Subnet-${count.index + 1}"
  }
}
```



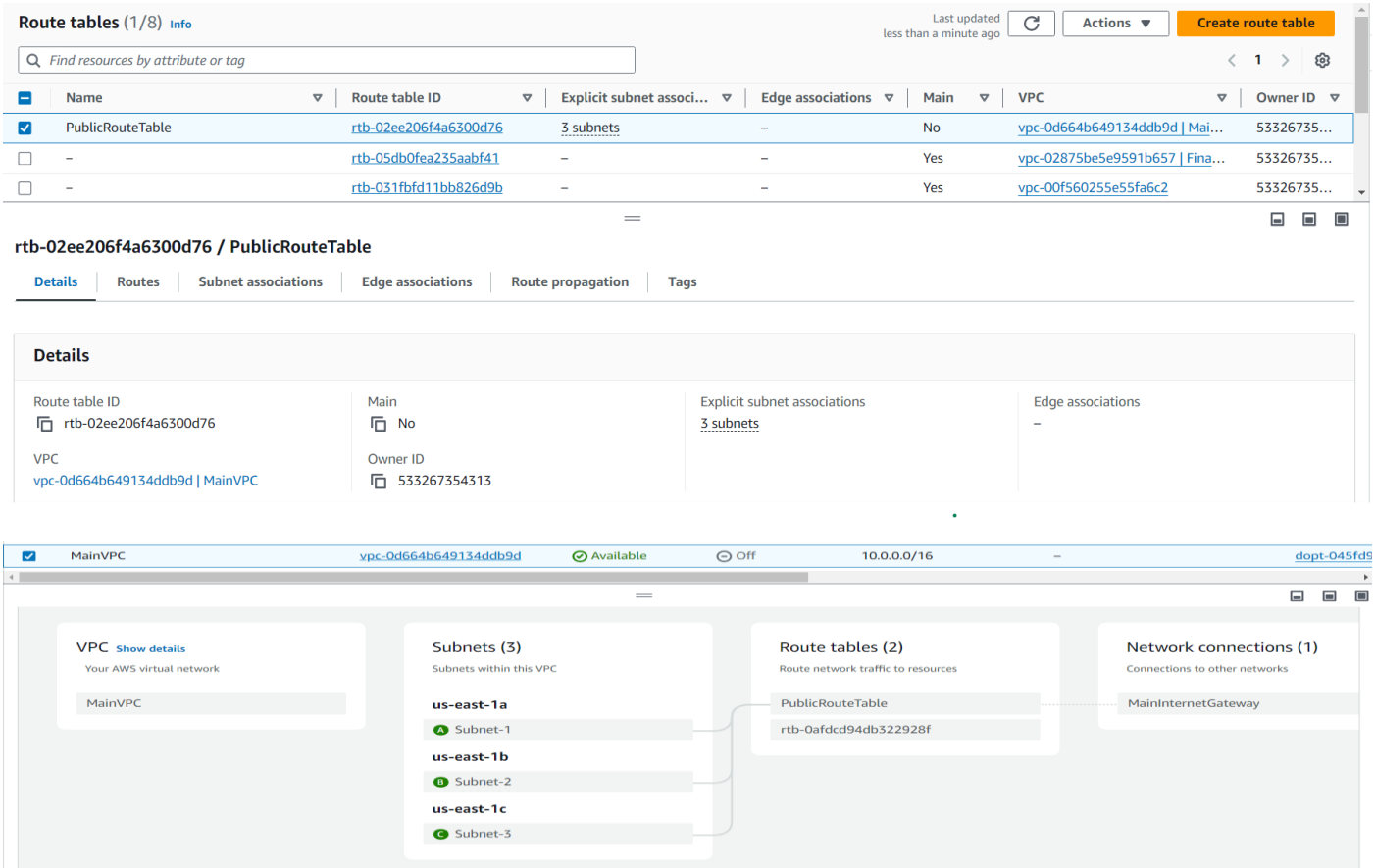
- Now, We create a single Gateway and attach it to our VPC

```
# Internet Gateway
resource "aws_internet_gateway" "igw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "MainInternetGateway"
  }
}
```



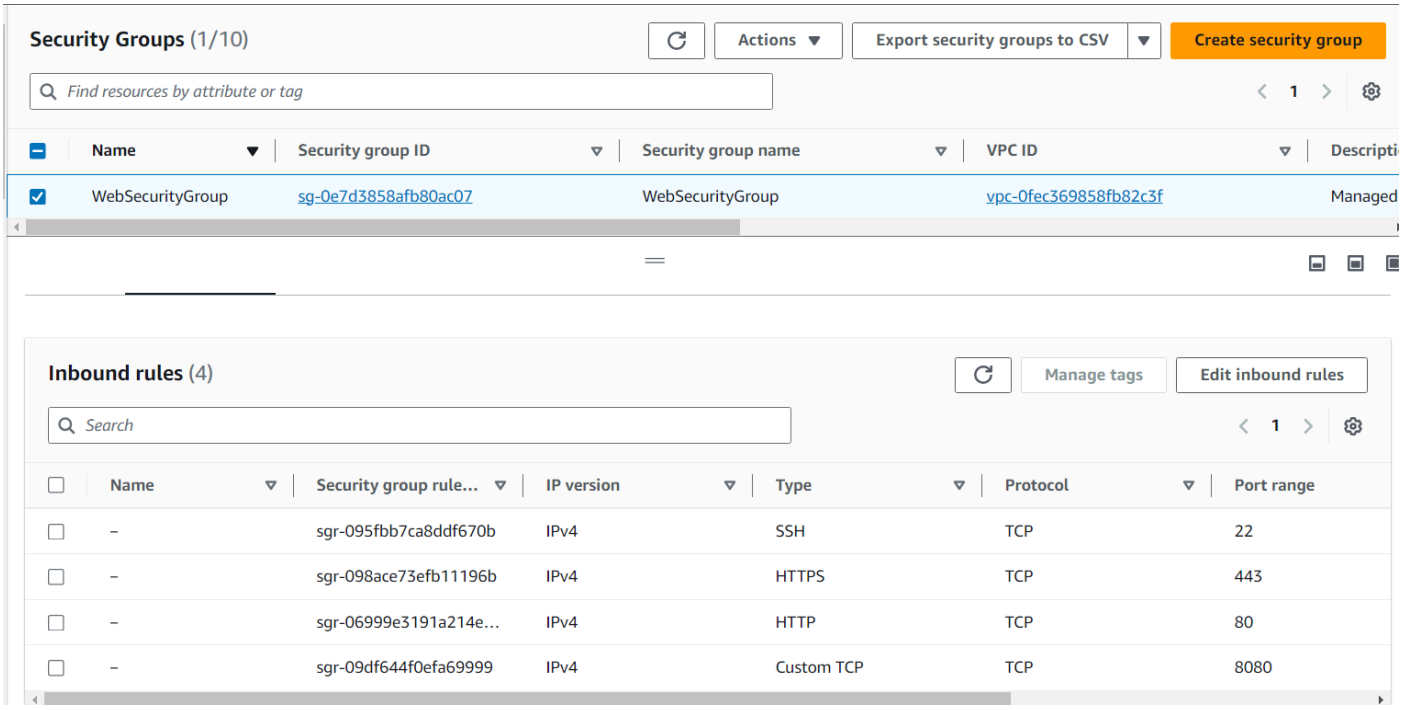
- Create a Routing Table and giving a public route 0.0.0.0 to the internet and also associating this public route to each subnet

```
# Route Table
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"  # Route for all internet traffic
    gateway_id = aws_internet_gateway.igw.id
  }

  tags = {
    Name = "PublicRouteTable"
  }
}

# Associate Route Table with Subnets
resource "aws_route_table_association" "subnet1" {
  subnet_id      = aws_subnet.subnets[0].id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "subnet2" {
  subnet_id      = aws_subnet.subnets[1].id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "subnet3" {
  subnet_id      = aws_subnet.subnets[2].id
  route_table_id = aws_route_table.public.id
}
```



- Creating a web-security group

```
# Security Group for Web Servers
resource "aws_security_group" "web_sg" {
  vpc_id = aws_vpc.main.id
  name   = "WebSecurityGroup"

  # Ingress Rules (Inbound Traffic)
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow SSH from anywhere
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow HTTP from anywhere
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow HTTPS from anywhere
  }

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Allow custom app traffic on port 8080
  }

  # Egress Rules (Outbound Traffic)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]  # Allow all outbound traffic
  }

  tags = {
    Name = "WebSecurityGroup"
  }
```



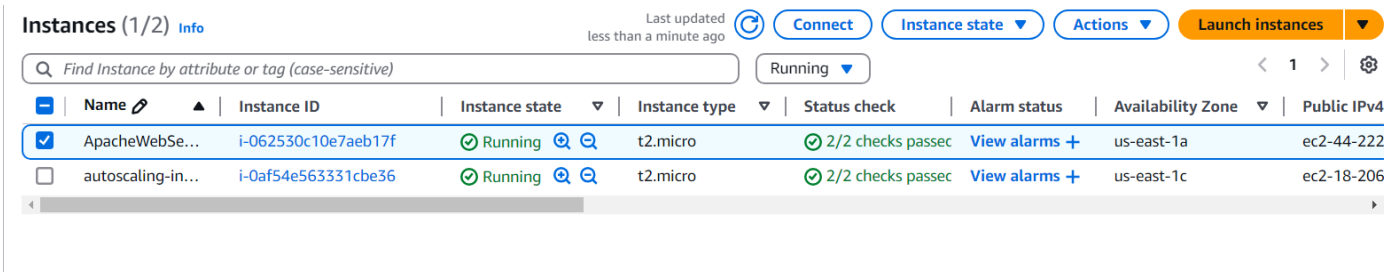
- Creating an EC2 instance with a standard AMI

```
# Amazon Linux 2 AMI data source
data "aws_ami" "amazon_linux" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
}

# EC2 Instance with Apache
resource "aws_instance" "web_server" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"
  subnet_id     = aws_subnet.subnets[0].id

  vpc_security_group_ids = [aws_security_group.web_sg.id]

  # User data script to install and start Apache
  user_data = <<-EOF
              #!/bin/bash
              yum update -y
              yum install -y httpd
              systemctl start httpd
              systemctl enable httpd
              echo "<h1>Hello from Group3 of Final Project</h1>" > /var/www/html/index.html

              echo "RDS_USERNAME=${var.rds_username}" >> /etc/environment
              echo "RDS_PASSWORD=${var.rds_password}" >> /etc/environment
              echo "RDS_DBNAME=${var.rds_dbname}" >> /etc/environment
              echo "RDSHOST_NAME=${var.rdshost_name}" >> /etc/environment

              # Reload environment variables
              source /etc/environment
              EOF

  # You might want to add your key pair name here
  # key_name = "your-key-pair-name"

  tags = {
    Name = "ApacheWebServer"
  }
}
```

```
# Output the public IP of the instance
output "web_server_public_ip" {
  value = aws_instance.web_server.public_ip
}

# Output the public DNS of the instance
output "web_server_public_dns" {
  value = aws_instance.web_server.public_dns
}
```

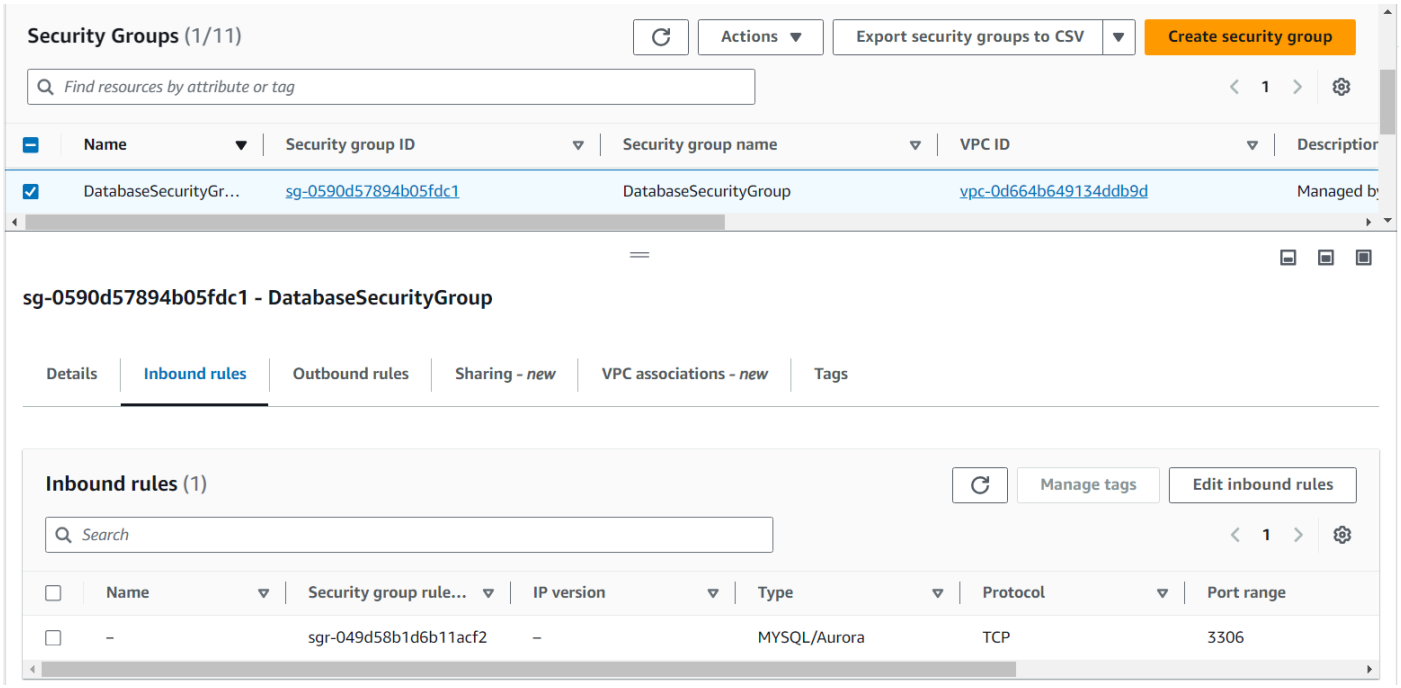
For the above variables we have created a terraform.tfvars file and store in out local system and not hardcoding the values into the script.

```
rds_username = "admin"
rds_password = "finalproject"
rds_dbname   = "mydb"
rdshost_name = "localhost"
```



- Creating appropriate dbsecurity groups

```
# Security Group for Database - only allowing traffic from web servers
resource "aws_security_group" "db_sg" {
  vpc_id = aws_vpc.main.id
  name   = "DatabaseSecurityGroup"

  # Ingress rule - allow traffic from web servers only
  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.web_sg.id]  # Only allow traffic from web security group
  }

  # Egress Rules
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "DatabaseSecurityGroup"
  }
}
```



Part-02

- Baking a new AMI ID using Ansible

```
---
- name: Create Custom AMI with Apache, Ansible, and Flask
  hosts: localhost
  gather_facts: false
  vars:
    instance_type: t2.micro
    region: us-east-1
    vpc_id: "{{ lookup('env', 'TF_VAR_vpc_id') }}"
    subnet_id: "{{ lookup('env', 'TF_VAR_subnet_id') }}"
    security_group_id: "{{ lookup('env', 'TF_VAR_security_group_id') }}"
    ami_name: "final-Project-{{ lookup('pipe', 'date +%Y%m%d%H%M%S') }}"
    base_ami: "ami-0453ec754f44f9a4a"

  tasks:
    - name: Launch EC2 instance
      amazon.aws.ec2_instance:
        name: "temp-instance-for-ami"
        instance_type: "{{ instance_type }}"
        image_id: "{{ base_ami }}"
        region: "{{ region }}"
        vpc_subnet_id: "{{ subnet_id }}"
        security_group: "{{ security_group_id }}"
        network:
          assign_public_ip: true
        wait: yes
        state: present
        user_data: |
          #!/bin/bash
          sudo yum update -y
          sudo yum install -y python3 python3-pip
          sudo yum groupinstall -y "Development Tools"
          sudo yum install -y python3-devel
          sudo pip3 install ansible flask flask-sqlalchemy flask-migrate mysql-connector-python pymysql psutil
          sudo yum install -y httpd
          sudo systemctl start httpd
          sudo systemctl enable httpd
          echo "<h1>Hello from Group4 of Final Project</h1>" | sudo tee /var/www/html/index.html
```

- Installing all necessary packages and also included the If the CPU usage increases over 90% then the script should stop.

```
---
- name: Create Custom AMI with Apache, Ansible, and Flask
  hosts: localhost
  gather_facts: false
  vars:
    instance_type: t2.micro
    region: us-east-1
    vpc_id: "{{ lookup('env', 'TF_VAR_vpc_id') }}"
    subnet_id: "{{ lookup('env', 'TF_VAR_subnet_id') }}"
    security_group_id: "{{ lookup('env', 'TF_VAR_security_group_id') }}"
    ami_name: "final-Project-{{ lookup('pipe', 'date +%Y%m%d%H%M%S') }}"
    base_ami: "ami-0453ec754f44f9a4a"

  tasks:
    - name: Launch EC2 instance
      amazon.aws.ec2_instance:
        name: "temp-instance-for-ami"
        instance_type: "{{ instance_type }}"
        image_id: "{{ base_ami }}"
        region: "{{ region }}"
        vpc_subnet_id: "{{ subnet_id }}"
        security_group: "{{ security_group_id }}"
        network:
          assign_public_ip: true
        wait: yes
        state: present
        user_data: |
          #!/bin/bash
          sudo yum update -y
          sudo yum install -y python3 python3-pip
          sudo yum groupinstall -y "Development Tools"
          sudo yum install -y python3-devel
          sudo pip3 install ansible flask flask-sqlalchemy flask-migrate mysql-connector-python pymysql psutil
          sudo yum install -y httpd
          sudo systemctl start httpd
          sudo systemctl enable httpd
          echo "<h1>Hello from Group4 of Final Project</h1>" | sudo tee /var/www/html/index.html
```

```
      # Start Python's built-in HTTP server
      nohup python3 -m http.server 80 --directory /var/www/html &
      # CPU Load Simulation Script
      nohup bash -c '
      while true; do
        # Start stress to load CPU for 30 seconds
        stress --cpu 2 --timeout 30 &
        PID=$!
        sleep 30
        kill $PID

        # Get current CPU utilization
        CURRENT_CPU=$(top -bn1 | grep "Cpu(s)" | \
        sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | \
        awk '{print 100 - $1}')

        # Check if CPU utilization exceeds 90%
        CPU_EXCEEDS=$(echo "$CURRENT_CPU > 90" | bc -l)

        if [ "$CPU_EXCEEDS" -eq 1 ]; then
          # Stop the loop if CPU usage is over 90%
          break
        fi

        # Maintain load for 30 seconds
        stress --cpu 2 --timeout 30 &
        sleep 30
        kill $!
      done' &
      # Set environment variables for RDS
      sudo echo "RDS_USERNAME=${var.rds_username}" >> /etc/environment
      sudo echo "RDS_PASSWORD=${var.rds_password}" >> /etc/environment
      sudo echo "RDS_DBNAME=${var.rds_dbname}" >> /etc/environment
      sudo echo "RDSHOST_NAME=${var.rdshost_name}" >> /etc/environment

  register: ec2

- name: Wait for user data script to complete
  pause:
    minutes: 2

- name: Create AMI
  amazon.aws.ec2_ami:
    instance_id: "{{ ec2.instances[0].instance_id }}"
    name: "{{ ami_name }}"
```

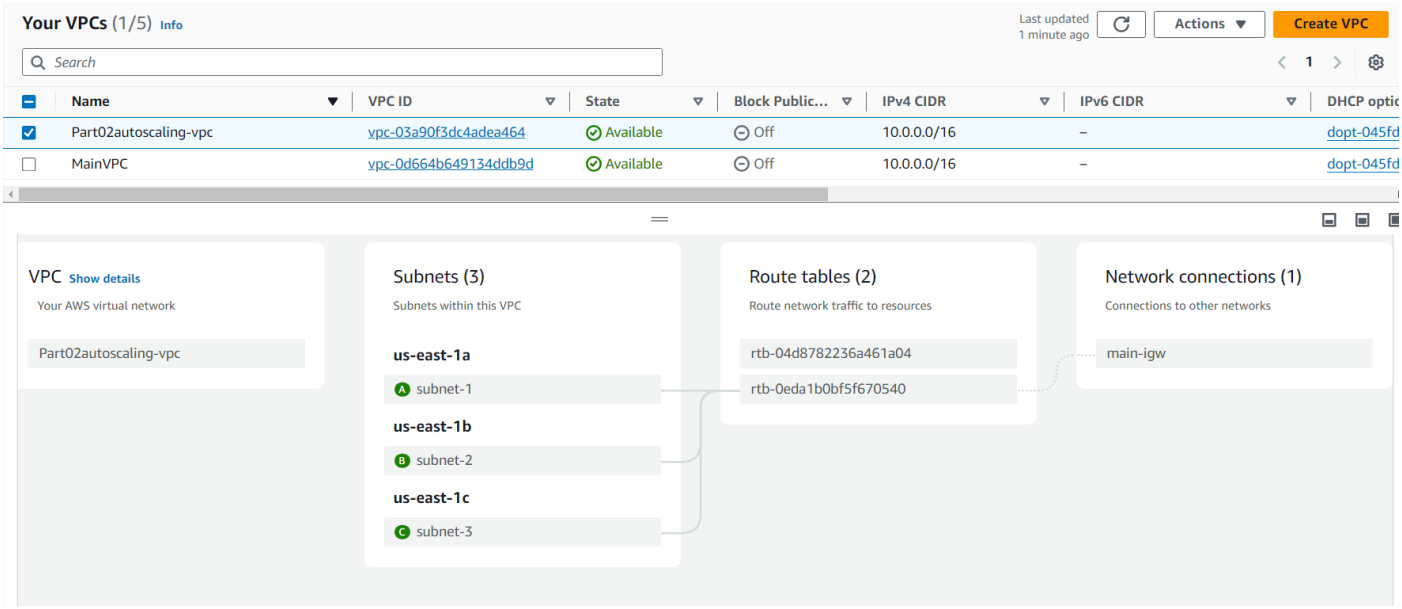
```
      region: "{{ region }}"
      wait: yes
      tags:
        Name: "{{ ami_name }}"
        Description: "AMI with Ansible, Apache, and Flask pre-installed"
    register: ami

  - name: Terminate temporary instance
    amazon.aws.ec2_instance:
      instance_ids: "{{ ec2.instances[0].instance_id }}"
      region: "{{ region }}"
      state: absent
      wait: yes
  - name: Save AMI ID to a file
    copy:
      content: "{{ ami.image_id }}"
      dest: "/mnt/d/ami_id.txt"

  - name: Output AMI ID
    debug:
      msg: "Created AMI with ID: {{ ami.image_id }}"
```

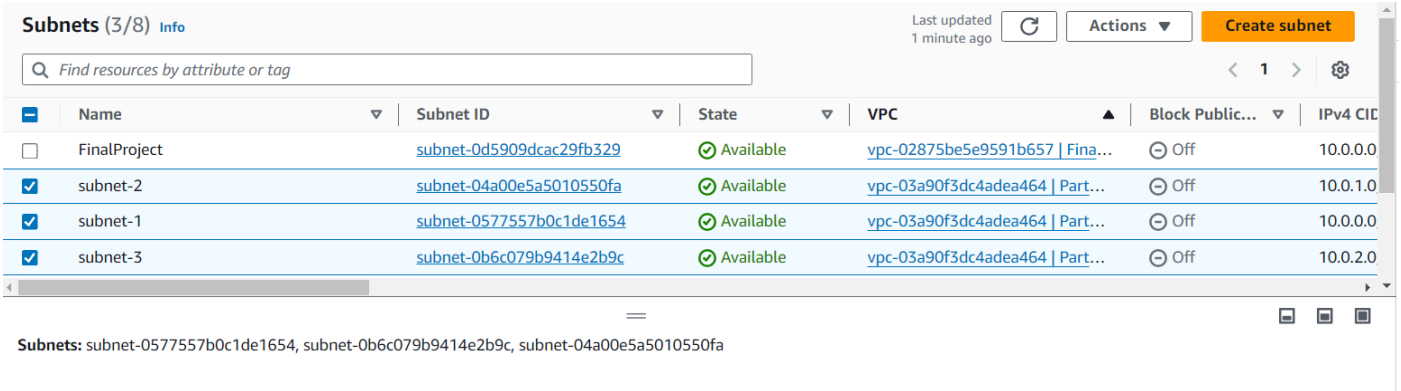
```
root@LAPTOP-2CSMSCDE:/mnt/d/Test folder for Infrastructure Automation and Tools/Project/part02# ansible-playbook playbook-ami.yaml
[WARNING]: No inventory was parsed, only implicit localhost is available
[WARNING]: provided hosts list is empty, only localhost is available. Note that the implicit localhost does not match 'all'

PLAY [Create Custom AMI with Apache, Ansible, and Flask] ***********************************************************

TASK [Launch EC2 instance] *****************************************************************************************
[DEPRECATION WARNING]: The network parameter has been deprecated, please use network_interfaces and/or network_interfaces_ids instead. This
 Deprecation warnings can be disabled by setting deprecation_warnings=False in ansible.cfg.
changed: [localhost]

TASK [Wait for user data script to complete] ***********************************************************************
Pausing for 60 seconds
(ctrl+C then 'C' = continue early, ctrl+C then 'A' = abort)
ok: [localhost]

TASK [Create AMI] **************************************************************************************************

changed: [localhost]

TASK [Terminate temporary instance] ********************************************************************************
changed: [localhost]

TASK [Save AMI ID to a file] ***************************************************************************************
changed: [localhost]

TASK [Output AMI ID] ***********************************************************************************************
ok: [localhost] => {
    "msg": "Created AMI with ID: ami-09f85dc11d55053aa"
}

PLAY RECAP *********************************************************************************************************
localhost                  : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

- Using Terraform to create a VPC with 3 subnets

```
# AWS Provider Configuration
provider "aws" {
  region = "us-east-1"  # Update with your desired region
}
data "aws_availability_zones" "available" {
  state = "available"
}

variable "instance_type" {
  description = "Type of EC2 instance"
}

variable "key_name" {
  description = "Name of the SSH key pair"
}



data "aws_ami" "latest_ami" {
  most_recent = true
  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*-x86_64-gp2"]
  }
  owners = ["amazon"]
}


# VPC Configuration
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_support   = true
  enable_dns_hostnames = true
  tags = { Name = "Part02autoscaling-vpc" }
}

# Subnet Configuration
resource "aws_subnet" "subnets" {
  count                   = 3  # Create three subnets in different Availability Zones
  vpc_id                  = aws_vpc.main.id
  cidr_block              = cidrsubnet(aws_vpc.main.cidr_block, 8, count.index)
  availability_zone       = data.aws_availability_zones.available.names[count.index]
  map_public_ip_on_launch = true

  tags = {
```

```
    Name = "subnet-${count.index + 1}"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id
  tags   = { Name = "main-igw" }
}

# Route Table
resource "aws_route_table" "public_rt" {
  vpc_id = aws_vpc.main.id
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main.id
  }
}

# Associate Route Table with Subnet
resource "aws_route_table_association" "subnet_association" {
  count          = length(aws_subnet.subnets)
  subnet_id      = aws_subnet.subnets[count.index].id
  route_table_id = aws_route_table.public_rt.id
}
```

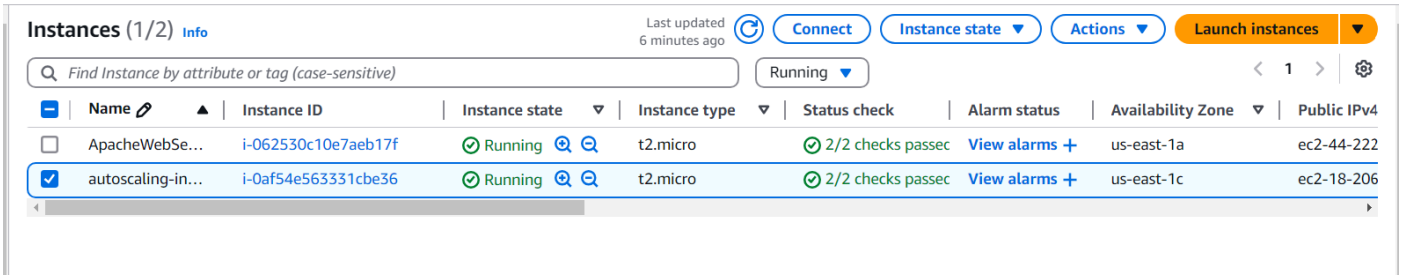




Subnets: subnet-0577557b0c1de1654, subnet-0b6c079b9414e2b9c, subnet-04a00e5a5010550fa

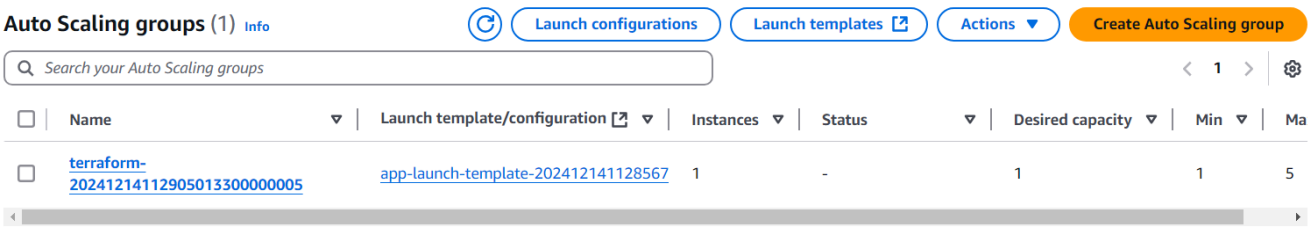
```hcl
# Security Group
resource "aws_security_group" "web_sg" {
  vpc_id = aws_vpc.main.id
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  # Allow HTTPS (port 443) traffic
  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Allow custom HTTP (port 8080) traffic
  ingress {
    from_port   = 8080
    to_port     = 8080
    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"]
  }
  tags = { Name = "web-sg" }
}
```

```hcl
resource "aws_launch_template" "app_lt" {
  name_prefix   = "app-launch-template-"
  image_id      = data.aws_ami.latest_ami.id
  instance_type = var.instance_type
  key_name      = var.key_name

  network_interfaces {
    security_groups = [aws_security_group.web_sg.id]
    associate_public_ip_address = true
  }

  lifecycle {
    create_before_destroy = true
  }

  tags = {
    Name = "app-launch-template"
  }
}
```



- Creating Auto Scaling Group

```
# Auto Scaling Group
resource "aws_autoscaling_group" "app_asg" {
  launch_template {
    id      = aws_launch_template.app_lt.id
    version = "$Latest"
  }
  min_size             = 1
  max_size             = 5
  desired_capacity     = 1
  vpc_zone_identifier  = aws_subnet.subnets[*].id
  health_check_type    = "EC2"
  health_check_grace_period = 300

  tag {
    key                 = "Name"
    value               = "autoscaling-instance"
    propagate_at_launch = true
  }
}

# Scale Up Policy
resource "aws_autoscaling_policy" "scale_up" {
  name                   = "scale-up-policy"
  scaling_adjustment     = 1
  adjustment_type        = "ChangeInCapacity"
  autoscaling_group_name = aws_autoscaling_group.app_asg.name
}

# Scale Down Policy
resource "aws_autoscaling_policy" "scale_down" {
  name                   = "scale-down-policy"
  scaling_adjustment     = -1
  adjustment_type        = "ChangeInCapacity"
  autoscaling_group_name = aws_autoscaling_group.app_asg.name
}
```



- Similarly, created a Cloudwatch alarm for high and low cpu usage

```
# CloudWatch Alarm for Scale Up
resource "aws_cloudwatch_metric_alarm" "scale_up_alarm" {
  alarm_name          = "scale-up-cpu-utilization"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = 60
  statistic           = "Average"
  threshold           = 70
  alarm_actions       = [aws_autoscaling_policy.scale_up.arn]
  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.app_asg.name
  }
}

# CloudWatch Alarm for Scale Down
resource "aws_cloudwatch_metric_alarm" "scale_down_alarm" {
  alarm_name          = "scale-down-cpu-utilization"
  comparison_operator = "LessThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = 60
  statistic           = "Average"
  threshold           = 40
  alarm_actions       = [aws_autoscaling_policy.scale_down.arn]
  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.app_asg.name
  }
}
```

```
[ec2-user@ip-10-0-0-177 ~]$ ls
cpu_load.sh
[ec2-user@ip-10-0-0-177 ~]$ sudo ./cpu_load.sh
Starting CPU Load Test
stress: info: [4321] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
Current CPU Usage: 0%
./cpu_load.sh: line 17:  4321 Terminated              stress --cpu 2 --timeout 30
stress: info: [4333] dispatching hogs: 2 cpu, 0 io, 0 vm, 0 hdd
```

```
Running transaction check
Running transaction test
top - 11:40:37 up 11 min,  1 user,  load average: 1.13, 0.42, 0.15
Tasks:  95 total,   3 running,  55 sleeping,   0 stopped,   0 zombie
top - 11:41:43 up 12 min,  2 users,  load average: 0.75, 0.43, 0.17
Tasks: 101 total,   3 running,  61 sleeping,   0 stopped,   0 zombie
%Cpu(s): 99.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  1.0 st
KiB Mem :   993492 total,   438160 free,    87716 used,   467616 buff/cache
KiB Swap:        0 total,        0 free,        0 used.   762156 avail Mem

 3896 root      20   0    7580     96      0 R 49.7  0.0   0:13.95 stress
 4323 root      20   0    7580    100      0 R 49.8  0.0   0:06.40 stress
 4324 root      20   0    7580    100      0 R 49.8  0.0   0:06.40 stress
    1 root      20   0  123600   5544   3920 S  0.0  0.6   0:00.74 systemd
    2 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 I  0.0  0.0   0:00.04 kworker/0:0
    4 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 kworker/0:0H
    6 root       0 -20       0      0      0 I  0.0  0.0   0:00.00 mm_percpu_wq
    7 root      20   0       0      0      0 S  0.0  0.0   0:00.15 ksoftirqd/0
    8 root      20   0       0      0      0 I  0.0  0.0   0:00.04 rcu_sched
    9 root      20   0       0      0      0 I  0.0  0.0   0:00.00 rcu_bh
   10 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 migration/0
   11 root      rt   0       0      0      0 S  0.0  0.0   0:00.00 watchdog/0
   12 root      20   0       0      0      0 S  0.0  0.0   0:00.00 cpuhp/0
   14 root      20   0       0      0      0 S  0.0  0.0   0:00.00 kdevtmpfs
```