Part 1: Infrastructure Setup with Terraform: -

1. AWS Setup and Terraform Initialization:-

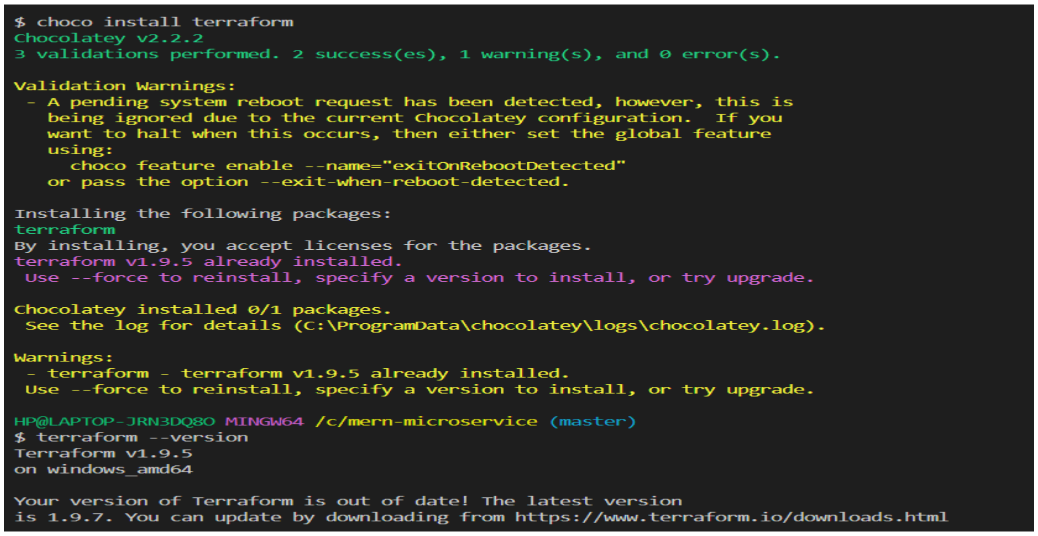
Install AWS CLI: Follow AWS CLI installation guide.

Once installed, configure it with:

```
bash
Copy code
aws configure
```

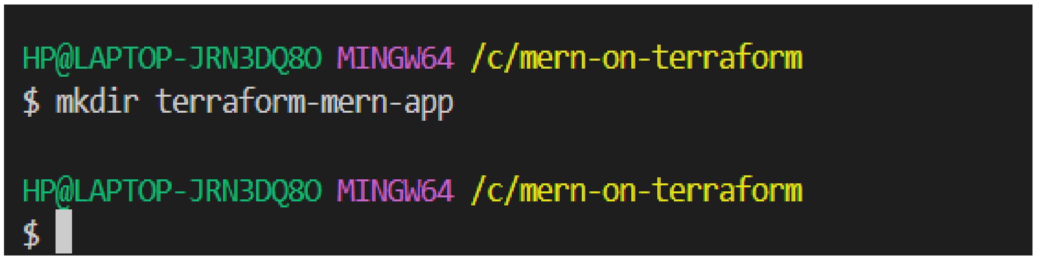
Enter your AWS access key, secret key, region, and output format when prompted.

- **Install Terraform**: Download and install Terraform from here.



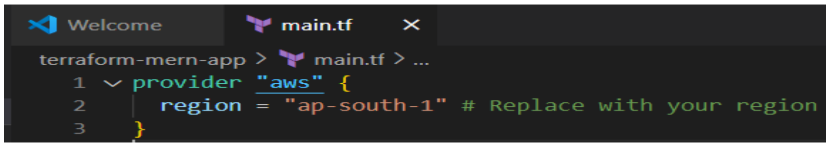
```
$ choco install terraform
Chocolatey v2.2.2
3 validations performed. 2 success(es), 1 warning(s), and 0 error(s).

Validation Warnings:
 - A pending system reboot request has been detected, however, this is
   being ignored due to the current Chocolatey configuration.  If you
   want to halt when this occurs, then either set the global feature
   using:
     choco feature enable --name="exitOnRebootDetected"
   or pass the option --exit-when-reboot-detected.

Installing the following packages:
terraform
By installing, you accept licenses for the packages.
terraform v1.9.5 already installed.
 Use --force to reinstall, specify a version to install, or try upgrade.

Chocolatey installed 0/1 packages.
 See the log for details (C:\ProgramData\chocolatey\logs\chocolatey.log).

Warnings:
 - terraform - terraform v1.9.5 already installed.
 Use --force to reinstall, specify a version to install, or try upgrade.

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-microservice (master)
$ terraform --version
Terraform v1.9.5
on windows_amd64

Your version of Terraform is out of date! The latest version
is 1.9.7. You can update by downloading from https://www.terraform.io/downloads.html
```

- **Initialize Terraform Project**: Create a new directory for your Terraform project:



```
HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform
$ mkdir terraform-mern-app

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform
$
```

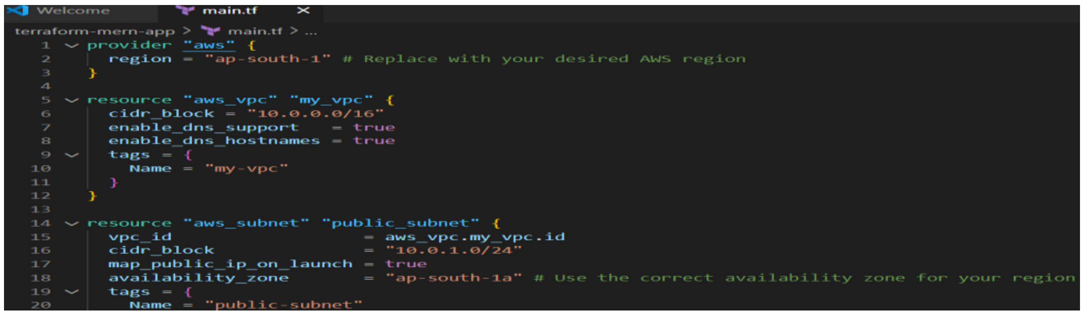
Create a file `main.tf` and start by configuring the provider: -



```
provider "aws" {
  region = "ap-south-1" # Replace with your region
}
```

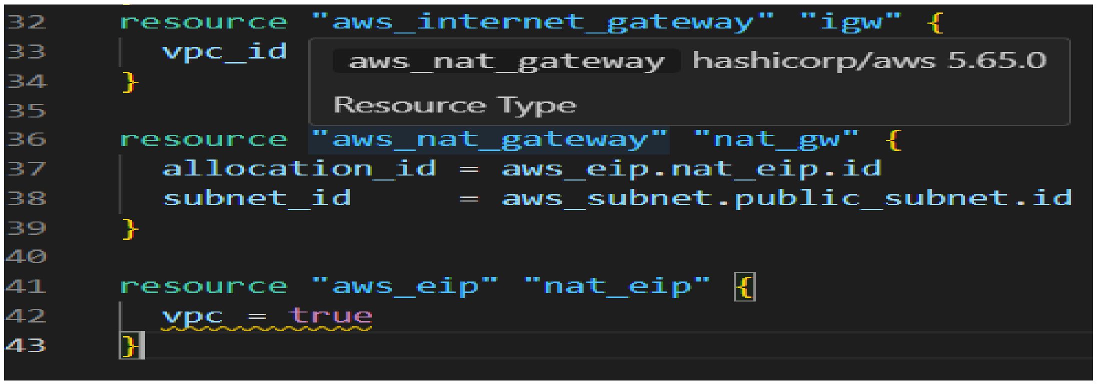
2. **VPC and Network Configuration: -**

- Create a VPC with subnets: Define the VPC and subnets (public and private) in main.tf: -



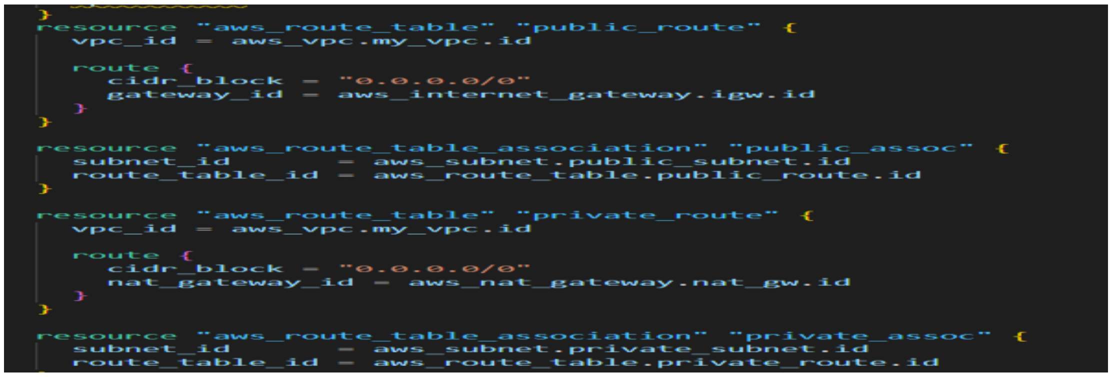
```
provider "aws" {
  region = "ap-south-1" # Replace with your desired AWS region
}

resource "aws_vpc" "my_vpc" {
  cidr_block = "10.0.0.0/16"
  enable_dns_support    = true
  enable_dns_hostnames = true
  tags = {
    Name = "my-vpc"
  }
}

resource "aws_subnet" "public_subnet" {
  vpc_id                  = aws_vpc.my_vpc.id
  cidr_block              = "10.0.1.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "ap-south-1a" # Use the correct availability zone for your region
  tags = {
    Name = "public-subnet"
```

- **Set up Internet and NAT Gateways**: Add this to `main.tf`:



```
resource "aws_internet_gateway" "igw" {
  vpc_id
}

resource "aws_nat_gateway" "nat_gw" {
  allocation_id = aws_eip.nat_eip.id
  subnet_id     = aws_subnet.public_subnet.id
}

resource "aws_eip" "nat_eip" {
  vpc = true
}
```

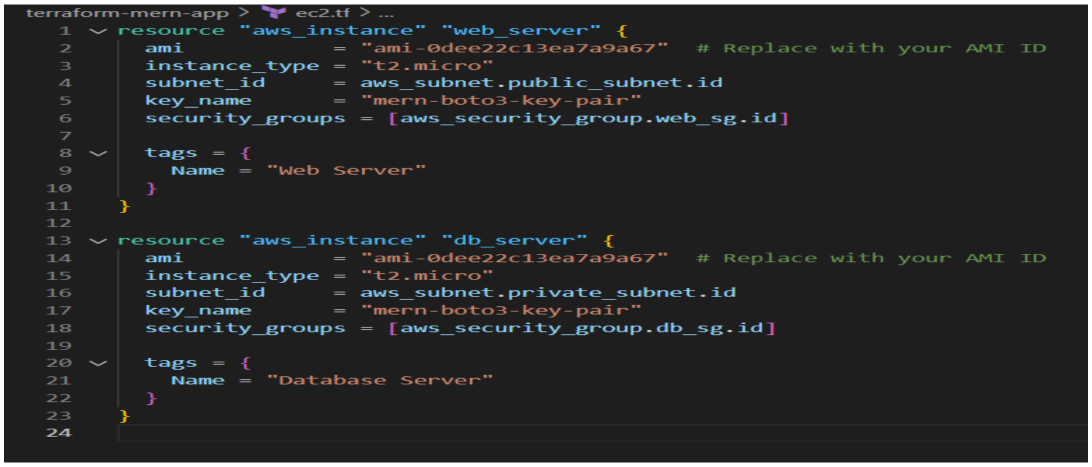
- **Route Tables Configuration: -**



```
}
resource "aws_route_table" "public_route" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.igw.id
  }
}

resource "aws_route_table_association" "public_assoc" {
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.public_route.id
}

resource "aws_route_table" "private_route" {
  vpc_id = aws_vpc.my_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.nat_gw.id
  }
}

resource "aws_route_table_association" "private_assoc" {
  subnet_id      = aws_subnet.private_subnet.id
  route_table_id = aws_route_table.private_route.id
```

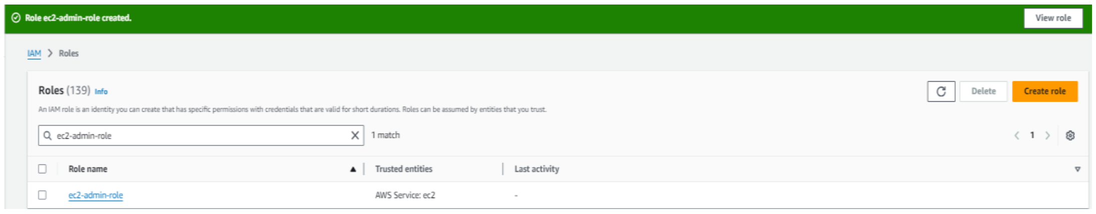
3. **EC2 Instance Provisioning: -**

- **Provision EC2 Instances**: Create one instance in the public subnet for the web server and one in the private subnet for MongoDB:



```
terraform-mern-app > ec2.tf > ...
resource "aws_instance" "web_server" {
  ami             = "ami-0dee22c13ea7a9a67"  # Replace with your AMI ID
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.public_subnet.id
  key_name        = "mern-boto3-key-pair"
  security_groups = [aws_security_group.web_sg.id]

  tags = {
    Name = "Web Server"
  }
}

resource "aws_instance" "db_server" {
  ami             = "ami-0dee22c13ea7a9a67"  # Replace with your AMI ID
  instance_type   = "t2.micro"
  subnet_id       = aws_subnet.private_subnet.id
  key_name        = "mern-boto3-key-pair"
  security_groups = [aws_security_group.db_sg.id]

  tags = {
    Name = "Database Server"
  }
}

```

- **IAM Role**: Create an IAM role for EC2 instances with necessary permissions:-



Role ec2-admin-role created.

IAM > Roles

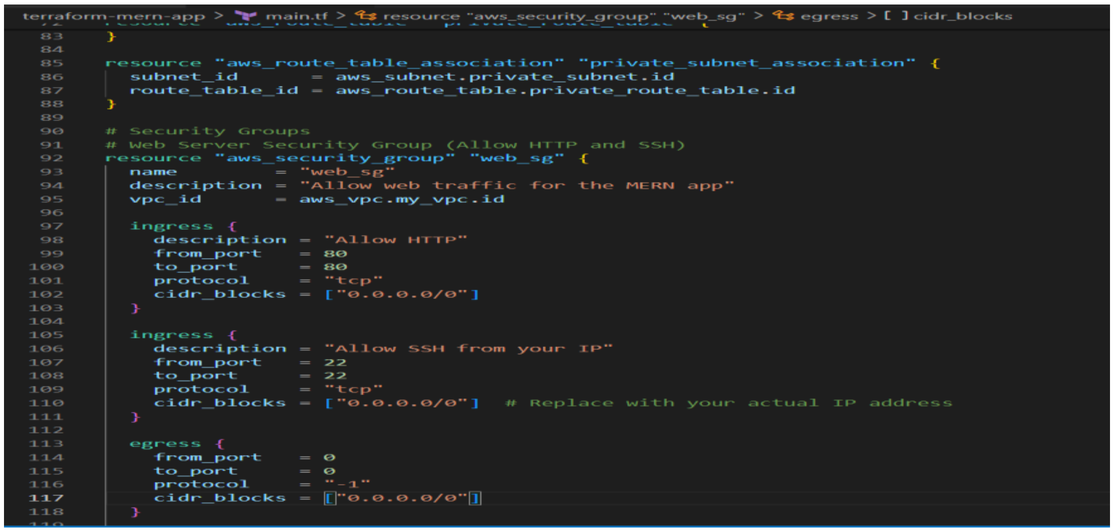
Roles (139)

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

ec2-admin-role 1 match

Role name	Trusted entities	Last activity
ec2-admin-role	AWS Service: ec2	-

- Infra creation on aws using terraform below is the main.tf file which is used to provision the total infra which is required to deploy our mern web-app on the aws console:-



```
terraform-mern-app > main.tf > resource "aws_security_group" "web_sg" > egress > [ ] cidr_blocks
}

resource "aws_route_table_association" "private_subnet_association" {
  subnet_id      = aws_subnet.private_subnet.id
  route_table_id = aws_route_table.private_route_table.id
}

# Security Groups
# Web Server Security Group (Allow HTTP and SSH)
resource "aws_security_group" "web_sg" {
  name        = "web_sg"
  description = "Allow web traffic for the MERN app"
  vpc_id      = aws_vpc.my_vpc.id

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

  ingress {
    description = "Allow SSH from your IP"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]  # Replace with your actual IP address
  }

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

Note: -you need to follow the below steps for deploying the mern-app on aws using the above script: -

a.terraform init (for the initializing the terraform in the aws environment) b.terraform plan (which will display what exactly will be dpeloyed after hitting the terraform apply.

c.terraform apply (which will actually provision the infra for which you wrote the main.tf amd the one infra which was showned to you after hitting the terraform plan.

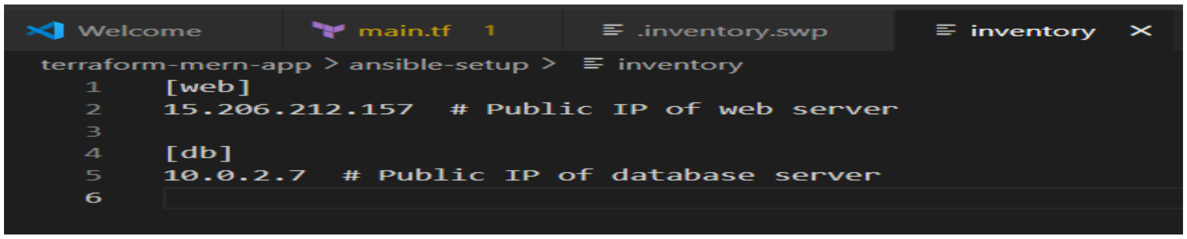
Part 2: Configuration and Deployment with Ansible

1. **Ansible Configuration: -**

A) Install Ansible: Follow the Ansible installation guide.

B) **Configure Ansible Inventory**: Create an inventory file (`inventory`) with EC2 instance IPs:-

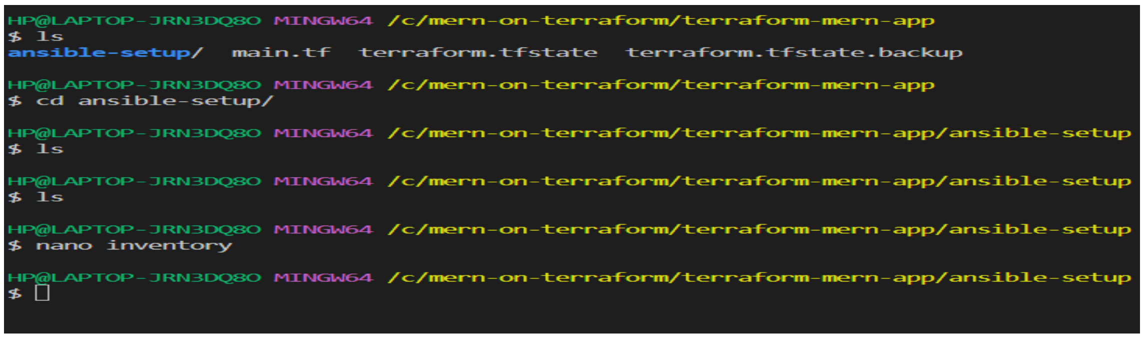
Step 2: Create the Ansible Inventory File



```
[web]
15.206.212.157   # Public IP of web server

[db]
10.0.2.7   # Public IP of database server
```

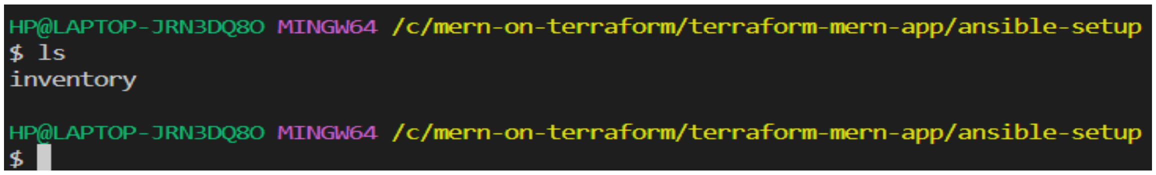
The inventory file contains the list of your EC2 instances that Ansible will manage. It should be created on your local machine.

- **Create a new directory** for your Ansible files:
  mkdir ansible-setup (creates the directory inside our project directory), cd ansible-setup(change the directory)



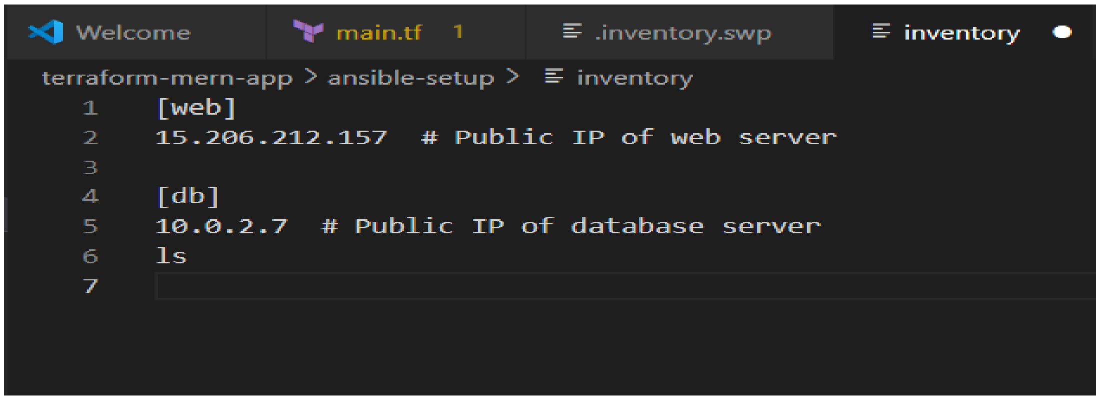
```
HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform/terraform-mern-app
$ ls
ansible-setup/  main.tf  terraform.tfstate  terraform.tfstate.backup

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform/terraform-mern-app
$ cd ansible-setup/

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform/terraform-mern-app/ansible-setup
$ ls

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform/terraform-mern-app/ansible-setup
$ ls

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform/terraform-mern-app/ansible-setup
$ nano inventory

HP@LAPTOP-JRN3DQ80 MINGW64 /c/mern-on-terraform/terraform-mern-app/ansible-setup
$ 
```

- **Create an inventory file** inside this directory:



```
HP@LAPTOP-JRN3DQ8O MINGW64 /c/mern-on-terraform/terraform-mern-app/ansible-setup
$ ls
inventory

HP@LAPTOP-JRN3DQ8O MINGW64 /c/mern-on-terraform/terraform-mern-app/ansible-setup
$ 
```

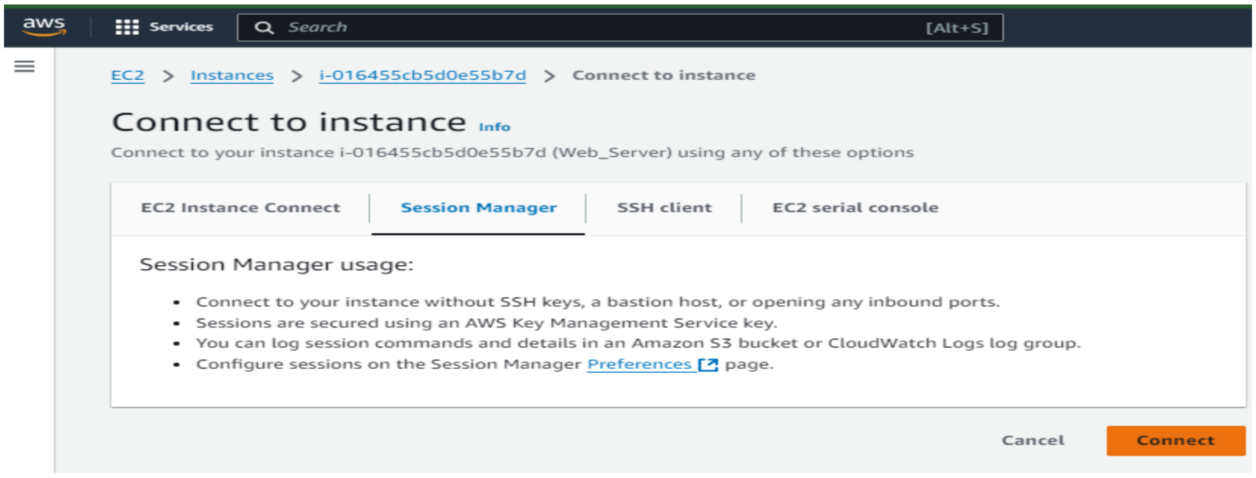
- Add the public IP addresses of your EC2 instances in the inventory file, organized by groups. Here's an example:



```
[web]
15.206.212.157   # Public IP of web server

[db]
10.0.2.7   # Public IP of database server
ls
```

Step 3: Set Up SSH Access to EC2 Instances

Note:-as the key pair was not created during the launch time of ec2 hence i was not able to take ssh into the same so in that case you need to

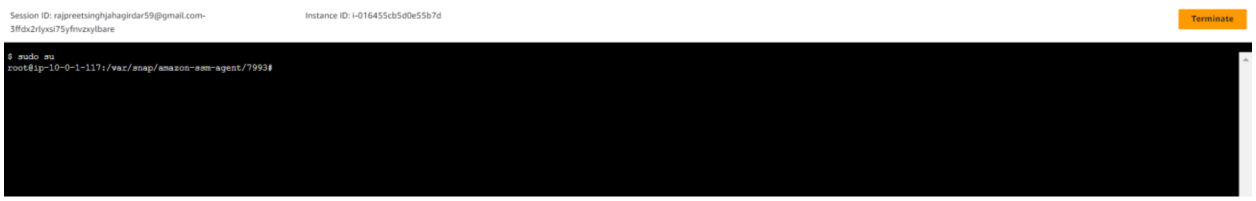


Connect to instance

Connect to your instance i-016455cb5d0e55b7d (Web_Server) using any of these options

Session Manager usage:

- Connect to your instance without SSH keys, a bastion host, or opening any inbound ports.
- Sessions are secured using an AWS Key Management Service key.
- You can log session commands and details in an Amazon S3 bucket or CloudWatch Logs log group.
- Configure sessions on the Session Manager Preferences page.



Session ID: rajpreetsinghjahagirdar59@gmail.com-3ffdx2rlyxsi75yfnvzxylbare

Instance ID: i-016455cb5d0e55b7d

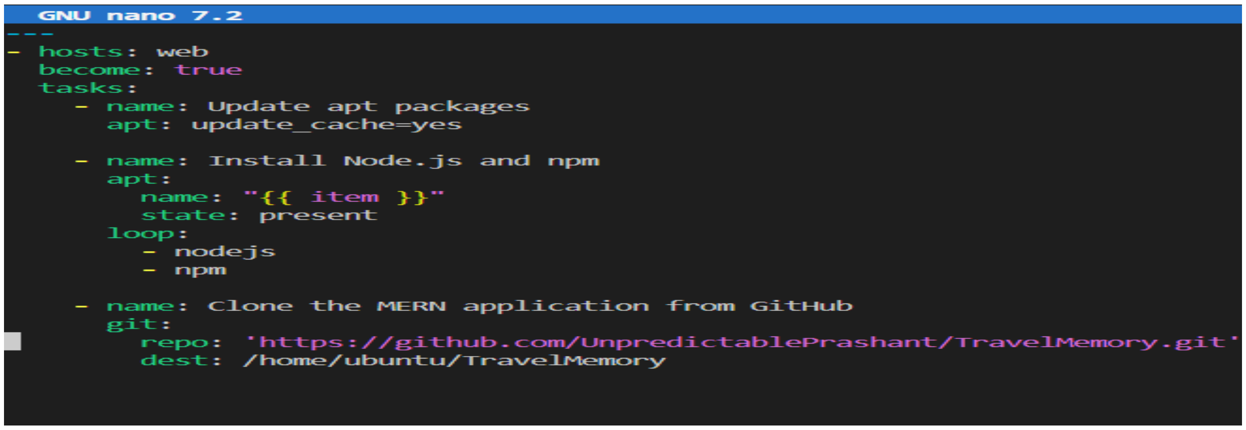
```
$ sudo su
root@ip-10-0-1-117:/var/snap/amazon-ssm-agent/7993#
```

Step 4: Write Your Ansible Playbooks:-

You can now create playbooks for configuring your servers.

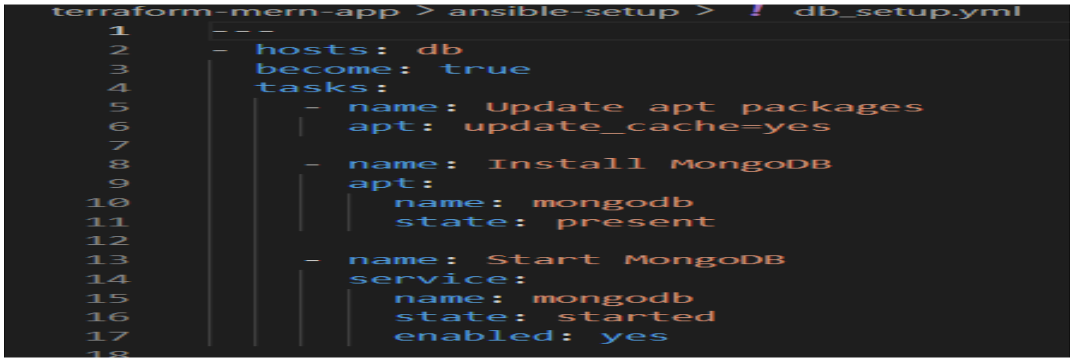
Create a playbook for the web server to install Node.js and clone the MERN application:

By hittting “nano web_setup.yml”



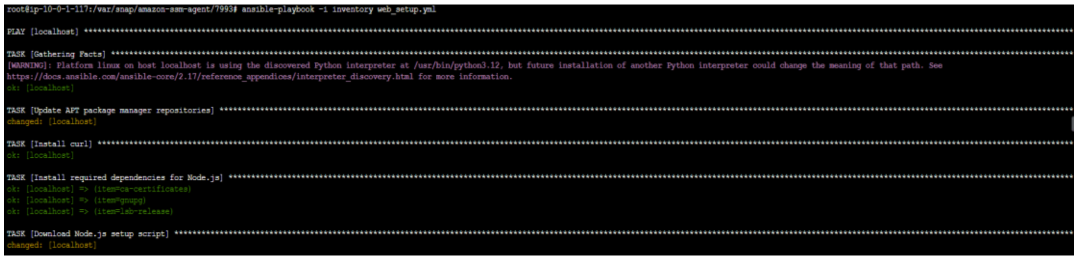
```
 GNU nano 7.2
---
- hosts: web
  become: true
  tasks:
    - name: Update apt packages
      apt: update_cache=yes

    - name: Install Node.js and npm
      apt:
        name: "{{ item }}"
        state: present
      loop:
        - nodejs
        - npm

    - name: Clone the MERN application from GitHub
      git:
        repo: 'https://github.com/UnpredictablePrashant/TravelMemory.git'
        dest: /home/ubuntu/TravelMemory
```

2.Create a playbook for the database server to install MongoDB:



```
terraform-mern-app > ansible-setup > ! db_setup.yml
---
- hosts: db
  become: true
  tasks:
    - name: Update apt packages
      apt: update_cache=yes

    - name: Install MongoDB
      apt:
        name: mongodb
        state: present

    - name: Start MongoDB
      service:
        name: mongodb
        state: started
        enabled: yes
```

3.Run the playbook for the web server: -



```
root@ip-10-0-1-117:/var/snap/amazon-ssm-agent/7993# ansible-playbook -i inventory web_setup.yml

PLAY [localhost] ***********************************************************************************************************************************

TASK [Gathering Facts] *****************************************************************************************************************************
[WARNING]: Platform linux on host localhost is using the discovered Python interpreter at /usr/bin/python3.12, but future installation of another Python interpreter could change the meaning of that path. See
https://docs.ansible.com/ansible-core/2.17/reference_appendices/interpreter_discovery.html for more information.
ok: [localhost]

TASK [Update APT package manager repositories] *****************************************************************************************************
changed: [localhost]

TASK [Install curl] ********************************************************************************************************************************
ok: [localhost]

TASK [Install required dependencies for Node.js] ***************************************************************************************************
ok: [localhost] => (item=ca-certificates)
ok: [localhost] => (item=gnupg)
ok: [localhost] => (item=lsb-release)

TASK [Download Node.js setup script] ***************************************************************************************************************
changed: [localhost]
```

5. Security Hardening:

 - Harden the security by configuring firewalls and security groups.

 - Implement additional security measures as needed (e.g., SSH key pairs, disabling root login).