Setting Up a Development Environment in AWS Using Terraform and VS Code

This guide walks you through setting up a development environment in AWS using Terraform and Visual Studio Code (VS Code). It covers everything from creating resources in AWS to configuring SSH and Docker.

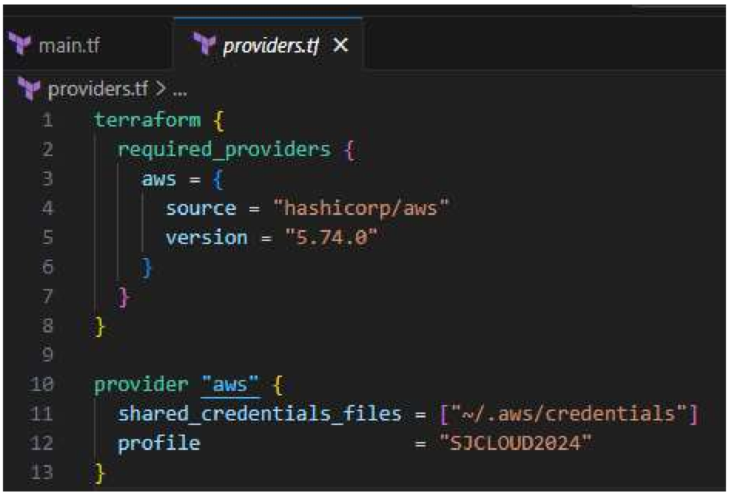
---

1. Setting Up the provider.tf File

The **provider.tf** file is essential for Terraform to interact with the AWS API and manage infrastructure.

Steps:

1. Add the AWS provider block in your **provider.tf** file. Reference the Terraform documentation for details: AWS Provider Documentation.
2. Include your AWS credentials. Use the shared credentials format to avoid hardcoding sensitive information.
   - As this blog is aimed to show you how to set up a Dev environment, I will not be going into how to set up IAM users and credentials. For details on setting up IAM users and credentials, refer to YouTube tutorials or Medium articles.
3. Run **terraform init** to:
   - Initialize the working directory.
   - Download necessary provider plugins and modules.
   - Set up the backend for storing infrastructure state.



```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
      version = "5.74.0"
    }
  }
}

provider "aws" {
  shared_credentials_files = ["~/.aws/credentials"]
  profile                  = "SJCLOUD2024"
}
```

---

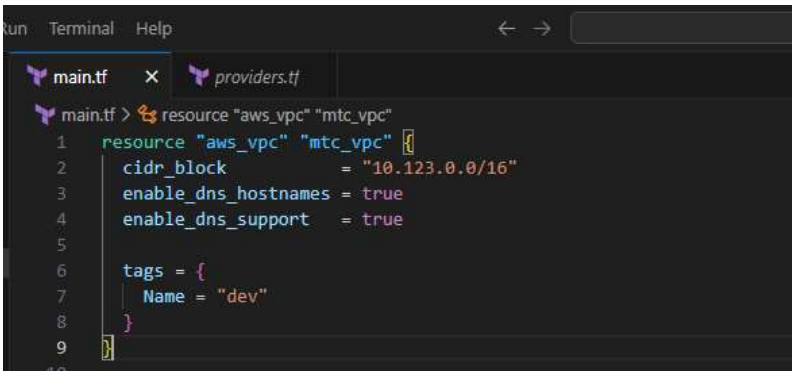
2. Setting Up the main.tf File

The **main.tf** file defines the resources you want to create in AWS.

Steps:

2.1 Create an AWS VPC (Virtual Private Cloud)

- Define the VPC resource in **main.tf**.
- Assign a name (e.g., "Dev") to your VPC.

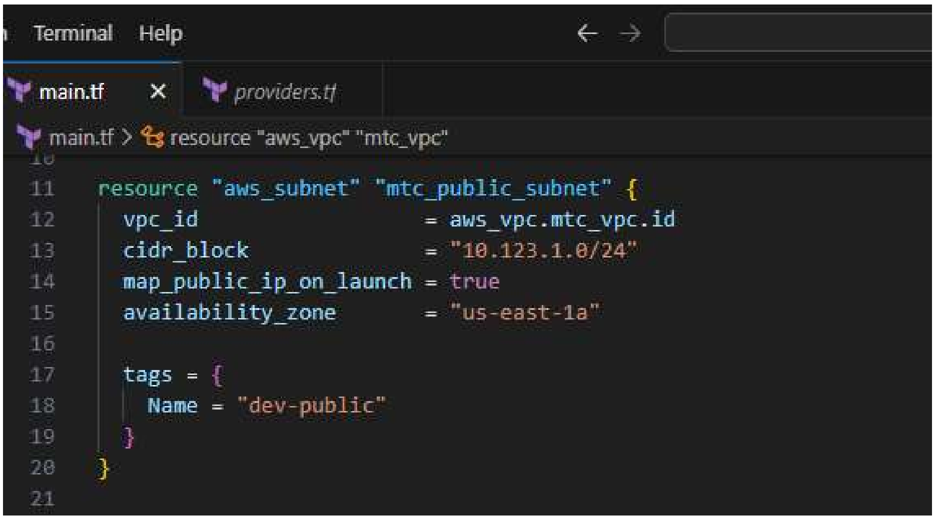


```
resource "aws_vpc" "mtc_vpc" {
  cidr_block           = "10.123.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "dev"
  }
}
```

Reference: VPC Resource Documentation.

2.2 Add a Public Subnet

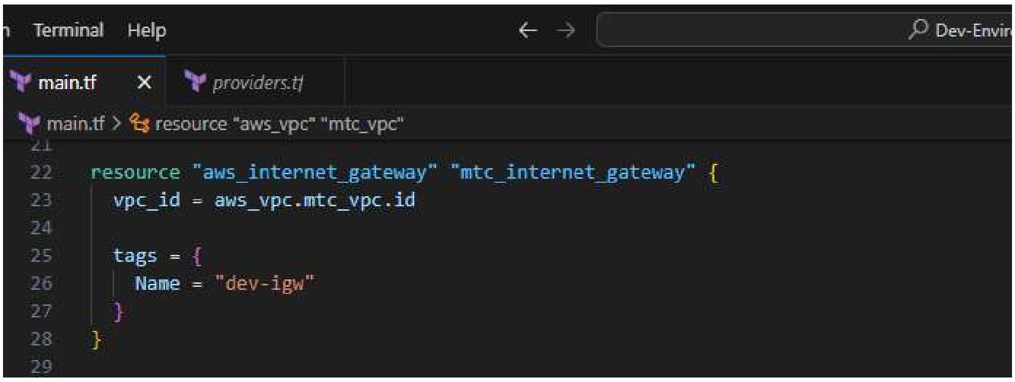
- Define a public subnet within the VPC.
- Include the following arguments:
  - map_public_ip_on_launch: Ensures instances launched in this subnet get a public IP address.
  - vpc_id: Links the subnet to your VPC.
  - cidr_block and availability_zone: Configure IP ranges and availability zones.



```
resource "aws_subnet" "mtc_public_subnet" {
  vpc_id                  = aws_vpc.mtc_vpc.id
  cidr_block              = "10.123.1.0/24"
  map_public_ip_on_launch = true
  availability_zone       = "us-east-1a"

  tags = {
    Name = "dev-public"
  }
}
```

2.3 Create an Internet Gateway

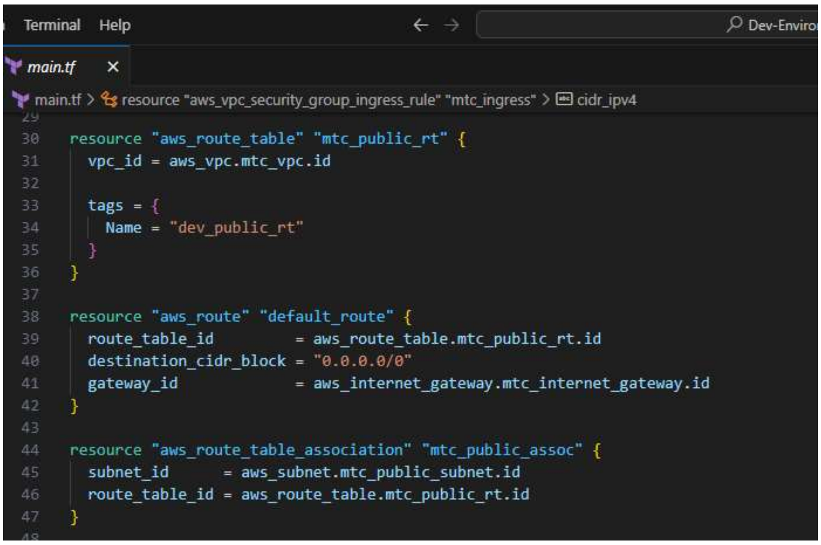
- Add an internet gateway to provide internet access to resources in the VPC.



```
resource "aws_internet_gateway" "mtc_internet_gateway" {
  vpc_id = aws_vpc.mtc_vpc.id

  tags = {
    Name = "dev-igw"
  }
}
```

2.4 Configure a Route Table

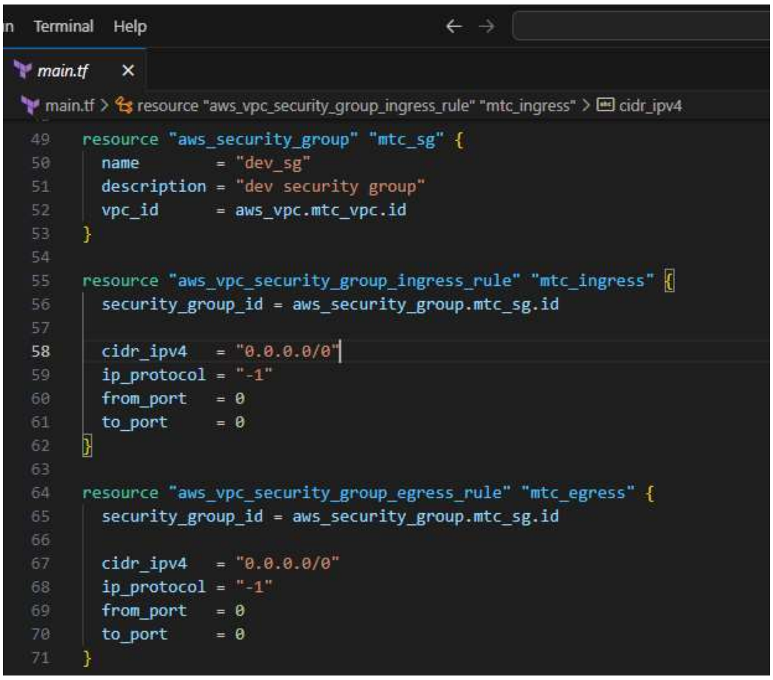
- Create a route table to route traffic from the subnet to the internet gateway.
- Create a route association table to link the route table to the subnet.


Terminal
Help
Dev-Enviro
main.tf
main.tf > resource "aws_vpc_security_group_ingress_rule" "mtc_ingress" > cidr_ipv4

```
resource "aws_route_table" "mtc_public_rt" {
  vpc_id = aws_vpc.mtc_vpc.id

  tags = {
    Name = "dev_public_rt"
  }
}

resource "aws_route" "default_route" {
  route_table_id         = aws_route_table.mtc_public_rt.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.mtc_internet_gateway.id
}

resource "aws_route_table_association" "mtc_public_assoc" {
  subnet_id      = aws_subnet.mtc_public_subnet.id
  route_table_id = aws_route_table.mtc_public_rt.id
}
```

2.5 Add Security Groups

- Define security groups to control inbound and outbound traffic for your EC2 instance. These act as a virtual firewall to protect resources.


n Terminal Help
main.tf
main.tf > resource "aws_vpc_security_group_ingress_rule" "mtc_ingress" > cidr_ipv4

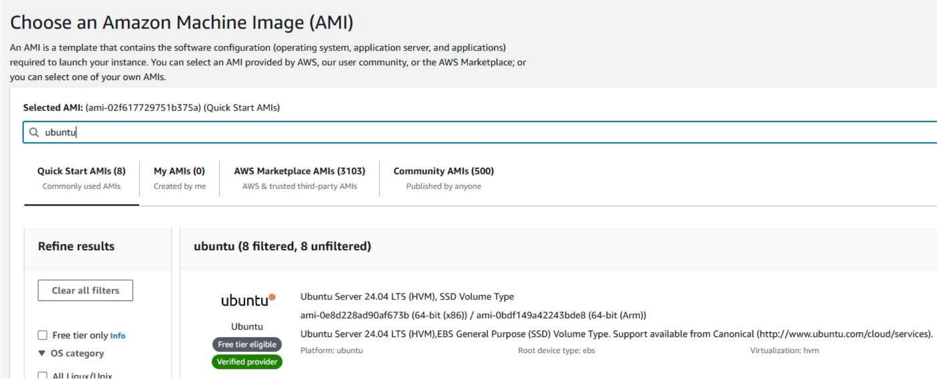
```
resource "aws_security_group" "mtc_sg" {
  name        = "dev_sg"
  description = "dev security group"
  vpc_id      = aws_vpc.mtc_vpc.id
}

resource "aws_vpc_security_group_ingress_rule" "mtc_ingress" {
  security_group_id = aws_security_group.mtc_sg.id

  cidr_ipv4   = "0.0.0.0/0"
  ip_protocol = "-1"
  from_port   = 0
  to_port     = 0
}

resource "aws_vpc_security_group_egress_rule" "mtc_egress" {
  security_group_id = aws_security_group.mtc_sg.id

  cidr_ipv4   = "0.0.0.0/0"
  ip_protocol = "-1"
  from_port   = 0
  to_port     = 0
}
```

3. Working with AMIs (Amazon Machine Images)

To deploy an EC2 instance, you need to specify an AMI.

Steps:

1. Use the EC2 AMI browser to find a suitable AMI (e.g., Ubuntu).



Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Selected AMI: (ami-02f617729751b375a) (Quick Start AMIs)

ubuntu

ubuntu (8 filtered, 8 unfiltered)

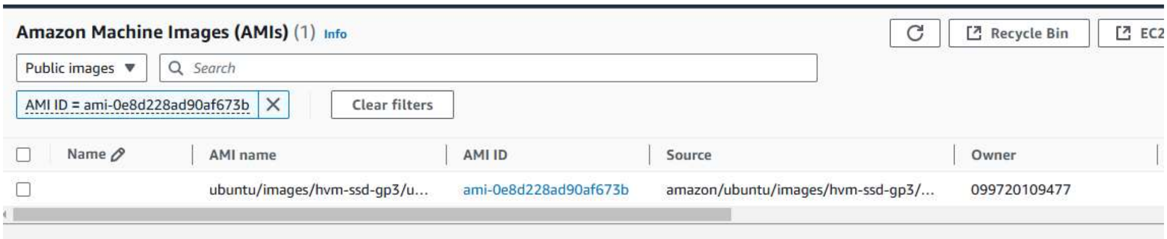
Ubuntu Server 24.04 LTS (HVM), SSD Volume Type

ami-0e8d228ad90af673b (64-bit (x86)) / ami-0bdf149a42243bde8 (64-bit (Arm))

Ubuntu Server 24.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).

Platform: ubuntu   Root device type: ebs   Virtualization: hvm

2. Copy the AMI ID and owner code.

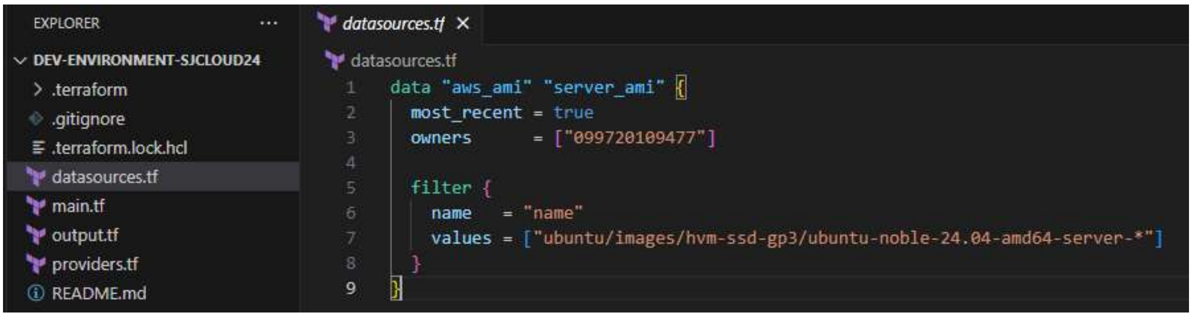


Amazon Machine Images (AMIs) (1) Info

AMI ID = ami-0e8d228ad90af673b

□	Name	AMI name	AMI ID	Source	Owner
□		ubuntu/images/hvm-ssd-gp3/u...	ami-0e8d228ad90af673b	amazon/ubuntu/images/hvm-ssd-gp3/...	099720109477

3. Create a datasources.tf file to define a data source for the AMI.
   - Use filters to ensure the latest AMI is selected dynamically.



```
data "aws_ami" "server_ami" {
  most_recent = true
  owners      = ["099720109477"]

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd-gp3/ubuntu-noble-24.04-amd64-server-*"]
  }
}
```

4. Generating an SSH Key Pair

An SSH key pair is required to securely access your EC2 instance

Steps:

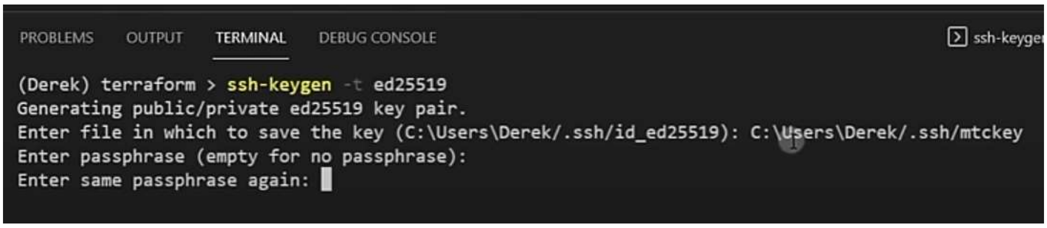
1. Run the following command to generate a key pair:

ssh-keygen -t ed25519

2. Save the key pair (e.g., mtckey).

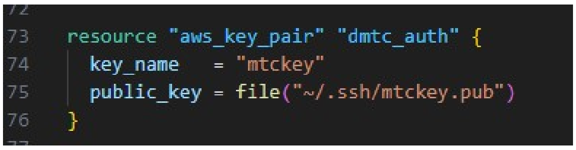
3. Verify the key pair in the .ssh folder using:

ls ~/.ssh



```
(Derek) terraform > ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (C:\Users\Derek/.ssh/id_ed25519): C:\Users\Derek/.ssh/mtckey
Enter passphrase (empty for no passphrase):
Enter same passphrase again: 
```

4. Add the public key to your main.tf file:

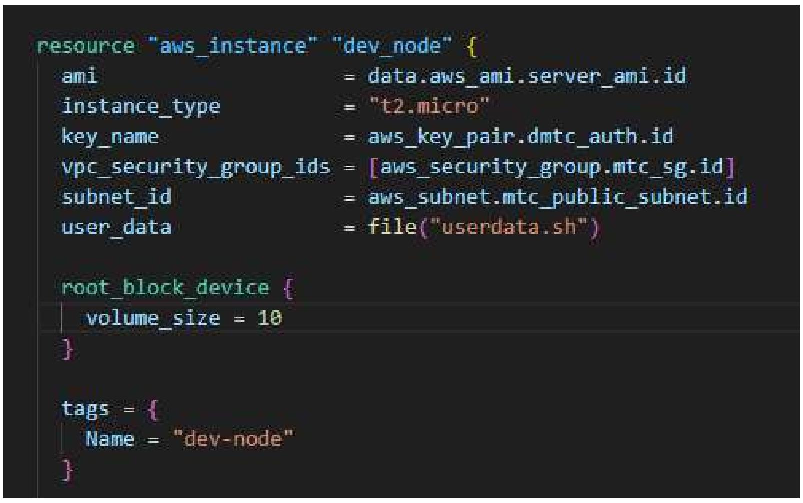


```
resource "aws_key_pair" "dmtc_auth" {
  key_name   = "mtckey"
  public_key = file("~/.ssh/mtckey.pub")
}
```

5. Creating an EC2 Instance

Define an EC2 instance in your main.tf file. Reference resources like:

- AMI
- Instance type
- Key pair
- Security group
- Subnet
- User data (see below)



```
resource "aws_instance" "dev_node" {
  ami                    = data.aws_ami.server_ami.id
  instance_type          = "t2.micro"
  key_name               = aws_key_pair.dmtc_auth.id
  vpc_security_group_ids = [aws_security_group.mtc_sg.id]
  subnet_id              = aws_subnet.mtc_public_subnet.id
  user_data              = file("userdata.sh")

  root_block_device {
    volume_size = 10
  }

  tags = {
    Name = "dev-node"
  }
```

User Data Script

Use a user data script to bootstrap the instance with Docker:

Create a new file called “userdata.sh” and insert the below script.

#!/bin/bash

sudo apt-get update -y &&

**sudo apt-get install -y **

**apt-transport-https **

```
 ca-certificates \

 curl \

 gnupg-agent \

 software-properties-common &&

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add - &&

sudo add-apt-repository -y "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable" &&

sudo apt-get update -y &&

sudo apt-get install -y docker-ce docker-ce-cli containerd.io &&

sudo usermod -aG docker ubuntu
```

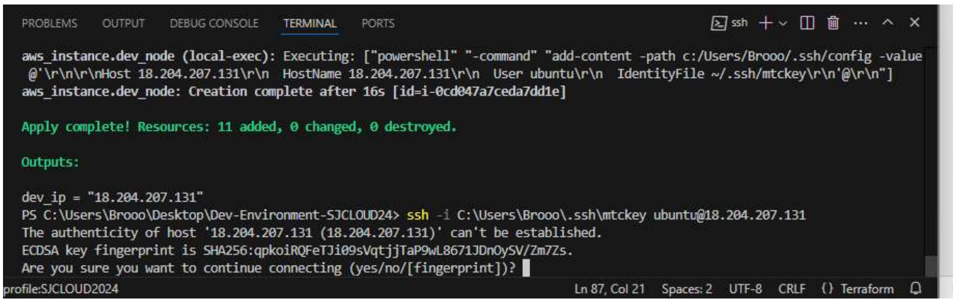
---

5. Testing SSH connectivity and Docker

Steps:

1. Run the following Terraform commands in VS Code:
2. terraform init
3. terraform plan
4. terraform apply
5. After deployment, get the public IP address of the EC2 instance from the AWS console.
6. SSH into the instance:

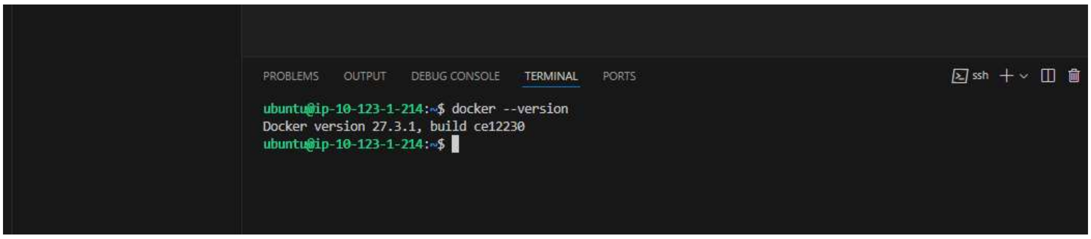
ssh -i ~/.ssh/mtckey ubuntu@<public-ip-address>



```
aws_instance.dev_node (local-exec): Executing: ["powershell" "-command" "add-content -path c:/Users/Brooo/.ssh/config -value
 @'\r\n\r\nHost 18.204.207.131\r\n  HostName 18.204.207.131\r\n  User ubuntu\r\n  IdentityFile ~/.ssh/mtckey\r\n'@\r\n"]
aws_instance.dev_node: Creation complete after 16s [id=i-0cd047a7ceda7dd1e]

Apply complete! Resources: 11 added, 0 changed, 0 destroyed.

Outputs:

dev_ip = "18.204.207.131"
PS C:\Users\Brooo\Desktop\Dev-Environment-SJCLOUD24> ssh -i C:\Users\Brooo\.ssh\mtckey ubuntu@18.204.207.131
The authenticity of host '18.204.207.131 (18.204.207.131)' can't be established.
ECDSA key fingerprint is SHA256:qpkoiRQFeTJi09sVqtjjTaP9wL8671JDnOySV/Zm7Zs.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

7. Verify Docker installation:

docker –version

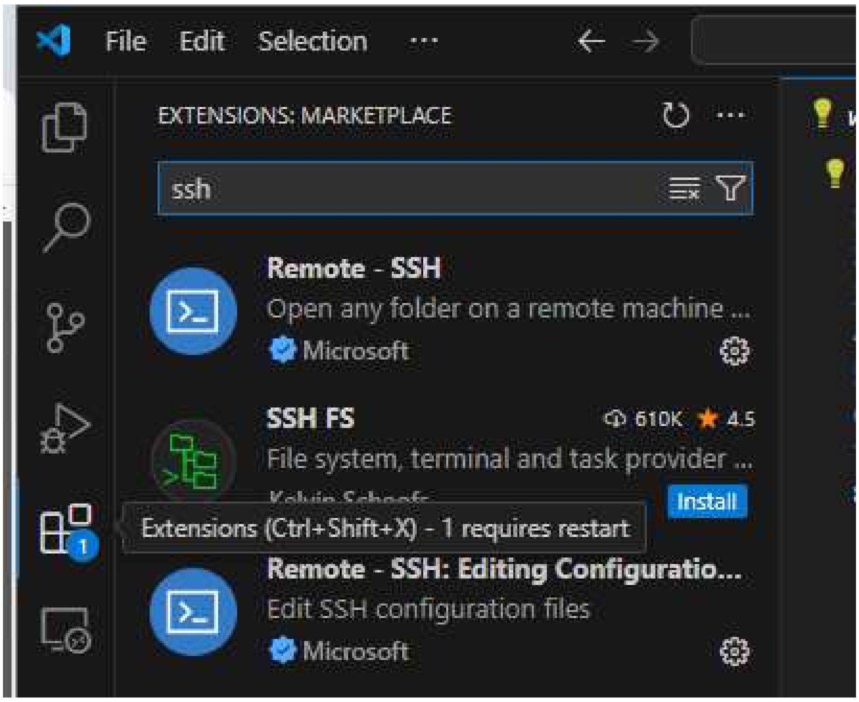


```
ubuntu@ip-10-123-1-214:~$ docker --version
Docker version 27.3.1, build ce12230
ubuntu@ip-10-123-1-214:~$ 
```

7. Configuring VS Code for SSH

Steps:

1. Install the "Remote - SSH" extension in VS Code.



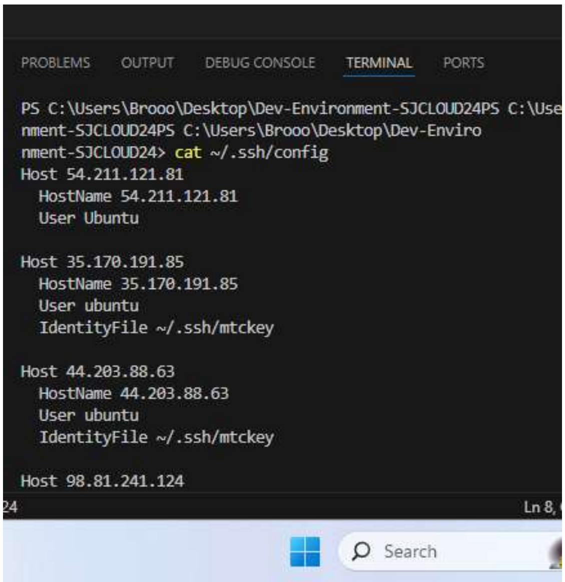
Remote - SSH
Open any folder on a remote machine ...
Microsoft

SSH FS 610K 4.5
File system, terminal and task provider ...
Kelvin Schoofs

Remote - SSH: Editing Configuratio...
Edit SSH configuration files
Microsoft

2. Check config file in .ssh folder:

Once installed, you should have a config file saved in your ssh folder. In order to check this, you can type “cat ~/.ssh/config” in the VSCOSE terminal. You should see the below displayed. The below shows the host, hostname and user of which we will use to login.



```
PS C:\Users\Brooo\Desktop\Dev-Environment-SJCLOUD24PS C:\Use
nment-SJCLOUD24PS C:\Users\Brooo\Desktop\Dev-Enviro
nment-SJCLOUD24> cat ~/.ssh/config
Host 54.211.121.81
  HostName 54.211.121.81
  User Ubuntu

Host 35.170.191.85
  HostName 35.170.191.85
  User ubuntu
  IdentityFile ~/.ssh/mtckey

Host 44.203.88.63
  HostName 44.203.88.63
  User ubuntu
  IdentityFile ~/.ssh/mtckey

Host 98.81.241.124
```

3. Create a new temaplte file for ssh config script:

To write the script we will need to use a template file. You will need to create a file based on your operating system whether you're on windows, linux or Mac. Linux and Mac can use the same script. However, windows is going to be different. I'm going to call the new template file "windows-ssh-config.tpl".

The script for this windows file is:

add-content -path c:/Users/*Brooo*/.ssh/config -value @'

Host ${hostname}

HostName ${hostname}

User ${user}

IdentityFile ${identityfile}

'@

Within this script you will need to change the first line which contains the full path to the config file stored in on your PC (underlined). Lines 2-5 will add the hostname, user and identity file to our config. What we are doing in this script is specifying the hostname, user and identity file from our provisioner section in the **main.ft** file. However, before we create the provisioner section you will need to save your tpl file.

---

8. Using Terraform Variables

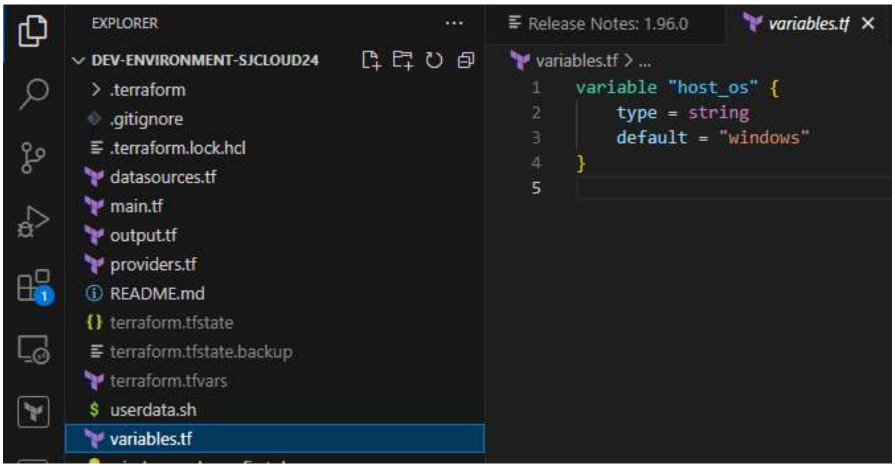
We are also going to use variable to start optimizing our script and making it more dynamic.

Steps:

1. Create a **variables.tf** file:
2. variable "host_os" {

3. default = "windows"

}

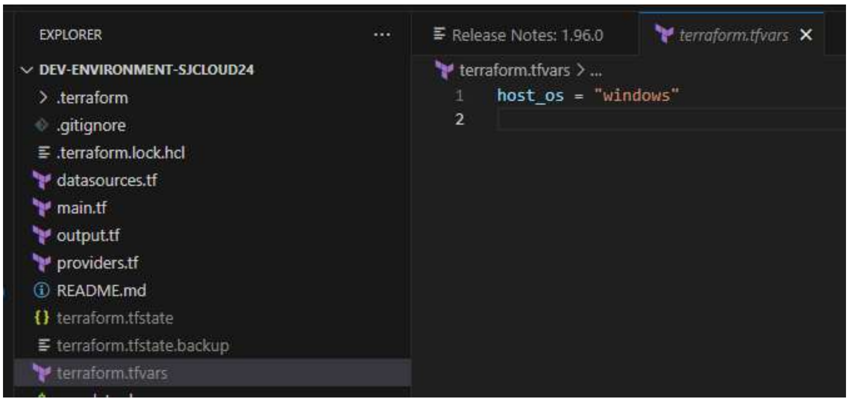


```
variable "host_os" {
    type = string
    default = "windows"
}
```

4. Create a **terraform.tfvars** file to define variable values:

host_os = "windows"

We are also going to create another file called "terraform.tfvars" as per below which will be our default name for our variable definition file. What we'll need to do is define our host_os which will be "windows".



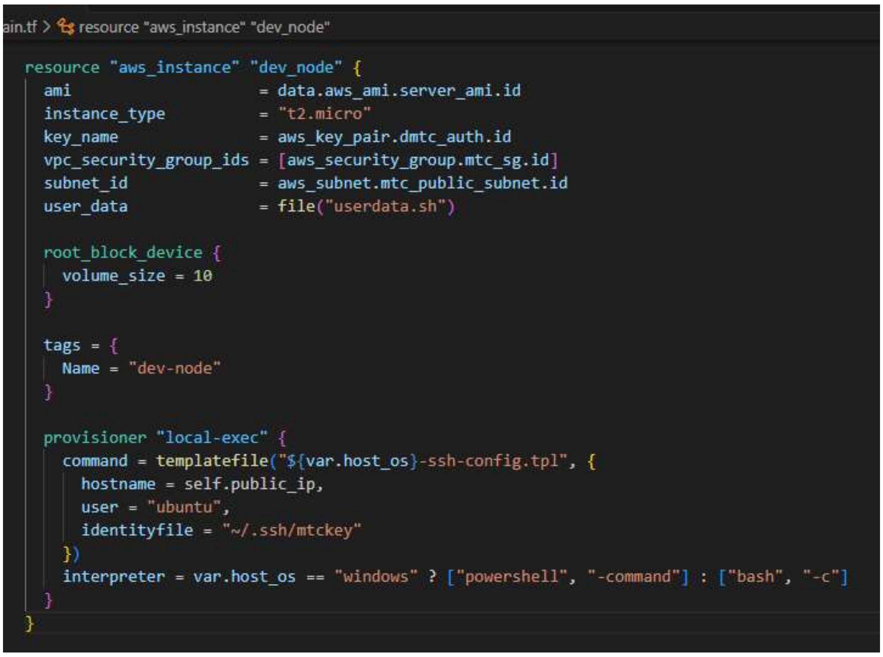
```
host_os = "windows"
```

9.Provisioning with Terraform

In this section, we will utilize a provisioner in our **main.tf** file to configure VSCode on our local terminal, enabling us to SSH into our EC2 instance. Specifically, we will use the local-exec provisioner, which allows us to run commands and configure our local machine.

It is important to note that **provisioners** are always executed within the context of another resource. In our case, the local-exec provisioner will run commands that help set up SSH access seamlessly.

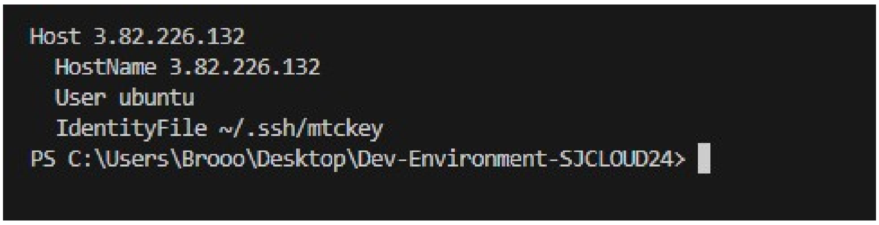
Additionally, we will use the self.public_ip syntax to access the public IP address of the EC2 instance. The self attribute allows us to retrieve resource-specific information such as public and private IP addresses, making it flexible and useful when dynamically referencing resource properties.



```
ain.tf > resource "aws_instance" "dev_node"

resource "aws_instance" "dev_node" {
  ami                    = data.aws_ami.server_ami.id
  instance_type          = "t2.micro"
  key_name               = aws_key_pair.dmtc_auth.id
  vpc_security_group_ids = [aws_security_group.mtc_sg.id]
  subnet_id              = aws_subnet.mtc_public_subnet.id
  user_data              = file("userdata.sh")

  root_block_device {
    volume_size = 10
  }

  tags = {
    Name = "dev-node"
  }

  provisioner "local-exec" {
    command = templatefile("${var.host_os}-ssh-config.tpl", {
      hostname = self.public_ip,
      user = "ubuntu",
      identityfile = "~/.ssh/mtckey"
    })
    interpreter = var.host_os == "windows" ? ["powershell", "-command"] : ["bash", "-c"]
  }
}
```

Key Note: the provisioner resource needs to be within the instance resources as per above snapshot.

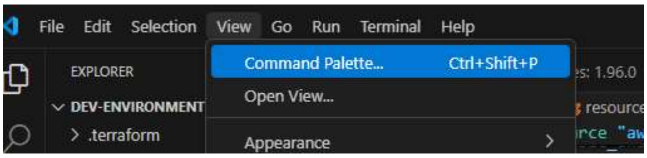
At this point you can write terraform init, plan and apply in the VSCODE terminal. Once your instances has been created and the 2 status checks confirmed, go back to VSCODE and type "cat ~/.ssh/config". This should show the hostname, user and identity-file as per below. The IP address for the host and hostname should match your public IP address within your instance.

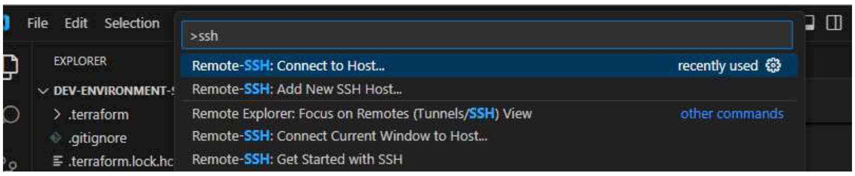


```
Host 3.82.226.132
  HostName 3.82.226.132
  User ubuntu
  IdentityFile ~/.ssh/mtckey
PS C:\Users\Brooo\Desktop\Dev-Environment-SJCLOUD24>
```

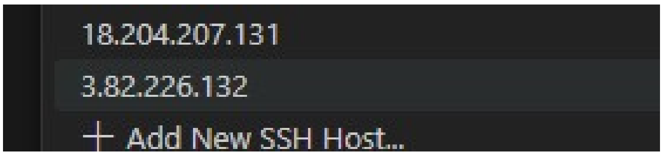
Now we can check if we can access our dev environment via the SHH host in VSCODE.

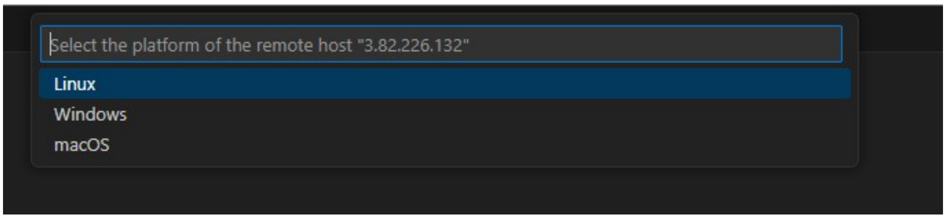
Step1: go to 'view' in the top ribbon and click on 'command palette', then search "ssh connect to host".


File Edit Selection View Go Run Terminal Help
EXPLORER
DEV-ENVIRONMENT
.terraform
Command Palette... Ctrl+Shift+P
Open View...
Appearance

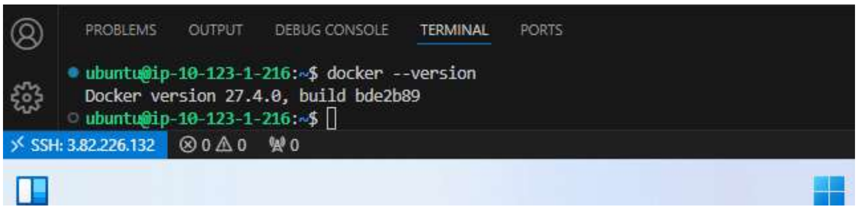

File Edit Selection
EXPLORER
DEV-ENVIRONMENT-
.terraform
.gitignore
.terraform.lock.hc
>ssh
Remote-SSH: Connect to Host... recently used
Remote-SSH: Add New SSH Host...
Remote Explorer: Focus on Remotes (Tunnels/SSH) View other commands
Remote-SSH: Connect Current Window to Host...
Remote-SSH: Get Started with SSH

Step2: you should be able to see your public IP address from your EC2 instance within the search bar. Once located click on your public IP address, then click on Linux as we our instance is a ubuntu Linux machine. Press 'continue' and at this point a new VSCODE window should pop up. Open a new terminal in VSCODE and you should now in active within your dev ubuntu instance. Your IP address should match your private IP address of your instance in AWS console.





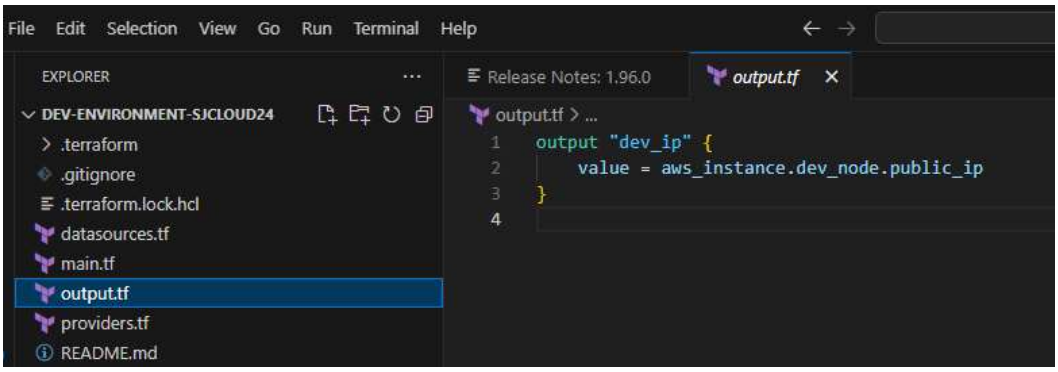
Step 3: you can check if docker is installed on your machine by writing "**docker --version**" in the terminal.



```
ubuntu@ip-10-123-1-216:~$ docker --version
Docker version 27.4.0, build bde2b89
ubuntu@ip-10-123-1-216:~$ 
```

10. Adding Outputs

To avoid manually finding the public IP, add an output file (output.tf):



```
output "dev_ip" {
    value = aws_instance.dev_node.public_ip
}

```

11. Summary: Setting Up a Development Environment in AWS Using Terraform and VS Code

Setting up a development environment in AWS using Terraform and VS Code offers a modern, efficient, and secure way to manage infrastructure and coding workflows. This approach combines Infrastructure as Code (IaC) practices with a flexible development environment, enabling automation, scalability, and consistency.

Benefits

1. **Infrastructure as Code (IaC) with Terraform:**
    - Automation: Terraform automates the provisioning and management of AWS resources, eliminating manual setup and reducing errors.
    - Consistency: The same Terraform scripts can be reused across environments (e.g., dev, test, production), ensuring predictable infrastructure.
    - Version Control: Terraform files can be stored in a Git repository, enabling version tracking and collaboration.
    - Scalability: Terraform allows for quick scaling of resources as your project grows.
    - Cost Efficiency: You can automate resource cleanup to avoid unnecessary expenses.
2. **Integrated Development Environment (IDE) with VS Code:**
    - Enhanced Productivity: VS Code extensions for Terraform and AWS simplify coding, linting, and debugging.
    - Seamless Integration: Extensions like Remote Development allow you to work directly in AWS-hosted instances.
    - Customizability: VS Code can be tailored to your specific needs with themes, extensions, and settings.
3. **Security Benefits:**
    - Centralized infrastructure management in AWS allows for better security monitoring.
    - Terraform ensures consistent security configurations across all deployed resources.
    - Role-based access control (RBAC) can be enforced using AWS Identity and Access Management (IAM).