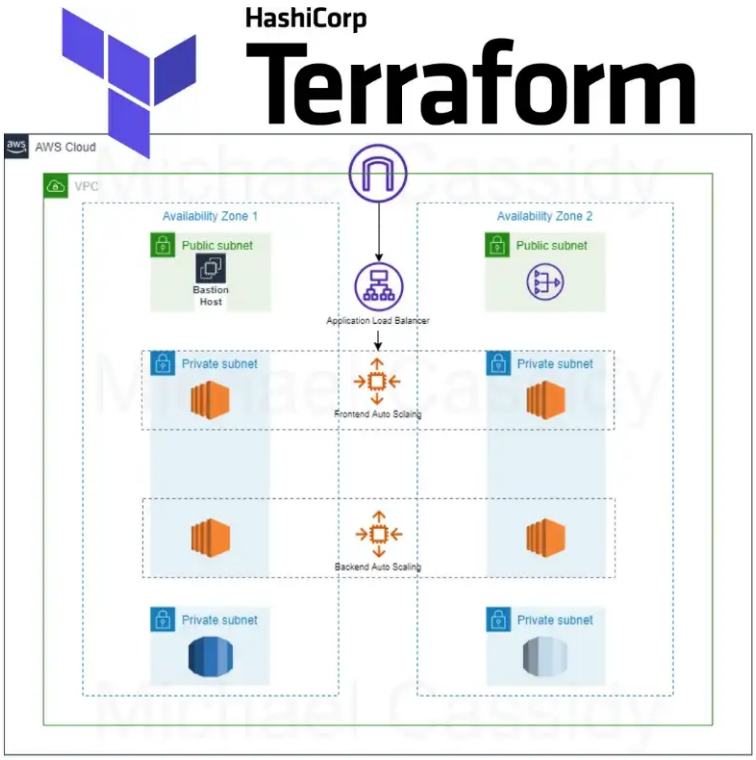
Michael Cassidy Follow

Jul 17 · 9 min read · · Listen

Terraform: AWS Three-Tier Architecture Design



In this project, we will create a three-tier architecture leveraging Terraform modules to make the process easily repeatable and reusable. Our architecture will reside in a

custom VPC.

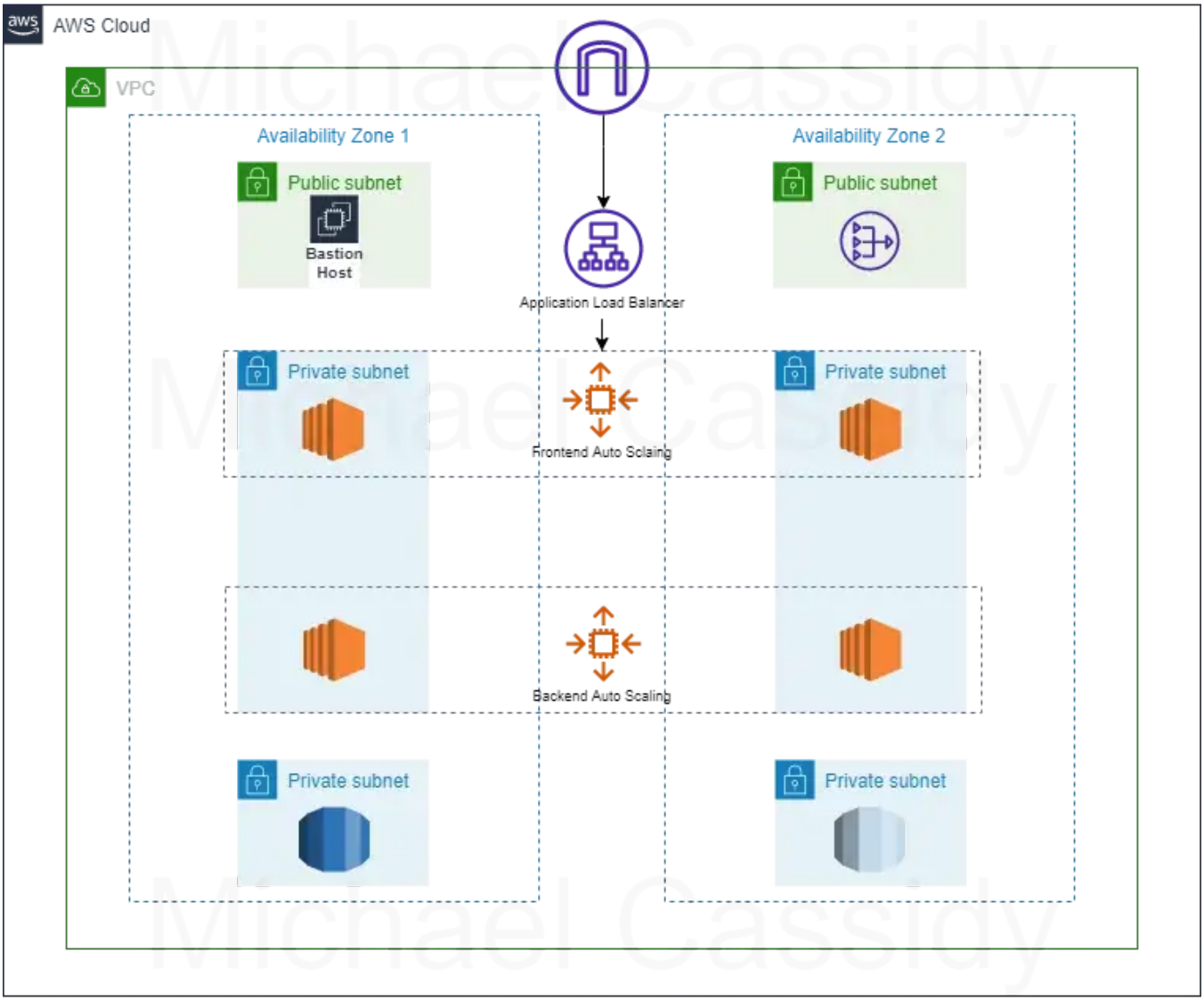
The web tier will have a bastion host and NAT gateway in the public subnets. The bastion host will serve as our access point to the underlying infrastructure. The NAT Gateway will allow our private subnets to access updates from the internet.

In the application tier, we will create an internet facing load balancer to direct internet traffic to an autoscaling group in the private subnets, along with a backend autoscaling group for our backend application. We will create a script to install the apache webserver in the frontend, and a script to install Node.js in the backend.

We will have another layer of private subnets in the database tier hosting a MySQL database which will will eventually access using Node.js. Keep in mind, this is simply an example of infrastructure one could use for a web application.

I will be using Visual Studio Code as my IDE, but feel free to use Cloud9 or another IDE of your choice. The steps in this tutorial will differ slightly if you choose to use a different IDE.

Here's a close up of our architecture:



Prerequisites

- AWS Account
- Proper Permissions for your user
- Terraform installed on your IDE
- AWS CLI installed and configured on your IDE
- SSH Agent (For Windows PowerShell)

Set Up

I will briefly go over the filesystem I will be using for this project. To learn more about why I use it for Terraform modules, and specifically AWS infrastructure,

please check out my article on Reusable EC2 Instances:

https://betterprogramming.pub/reusable-ec2-instances-using-terraform-modules-59aac51f1fb

Filesystem

In this filesystem, I have created folders that will represent the root (***terraform***), development environment (***dev***), in the ***us-east-1*** region, and utilizing the ***t2.micro*** instance type. The final folder will be the folder where all the root files reside. The next set of folders will represent the modules (***modules***). The modules folder consists of the 4 modules (***compute, database, loadbalancing, network***).

CODE

Let's start out in the root ***main.tf*** file. Here we will set up the locals block to refer to the folder system. We will also reference each module that we will be working with. Within each module block, different variables are defined. These variables are each introduced in the ***variables.tf*** files for each module. In the ***main.tf*** files of each module, we will point to these specified variables.

```
# --- root/main.tf ---

provider "aws" {
  region = local.location
}

locals {
  cwd           = reverse(split("/", path.cwd))
  instance_type = local.cwd[1]
  location      = local.cwd[2]
  environment   = local.cwd[3]
  vpc_cidr      = "10.123.0.0/16"
}

module "networking" {
  source            = "../../../../../modules/three-tier-deployment/networking"
  vpc_cidr          = local.vpc_cidr
  access_ip         = var.access_ip
  public_sn_count   = 2
  private_sn_count  = 2
  db_subnet_group   = true
  availabilityzone  = "us-east-1a"
  azs               = 2
}

module "compute" {
  source                  = "../../../../../modules/three-tier-deployment/compute"
  frontend_app_sg         = module.networking.frontend_app_sg
  backend_app_sg          = module.networking.backend_app_sg
  bastion_sg              = module.networking.bastion_sg
  public_subnets          = module.networking.public_subnets
  private_subnets         = module.networking.private_subnets
  bastion_instance_count  = 1
  instance_type           = local.instance_type
  key_name                = "Three-Tier-Terraform"
  lb_tg_name              = module.loadbalancing.lb_tg_name
  lb_tg                   = module.loadbalancing.lb_tg

}

module "database" {
  source               = "../../../../../modules/three-tier-deployment/database"
  db_storage           = 10
  db_engine_version    = "5.7.22"
  db_instance_class    = "db.t2.micro"
  db_name              = var.db_name
  dbuser               = var.dbuser
  dbpassword           = var.dbpassword
```

```
    dbpassword             = var.dbpassword
    db_identifier          = "three-tier-db"
    skip_db_snapshot       = true
    rds_sg                 = module.networking.rds_sg
    db_subnet_group_name = module.networking.db_subnet_group_name[0]
  }

  module "loadbalancing" {
    source                   = "../../../../../modules/three-tier-
  deployment/loadbalancing"
    lb_sg                    = module.networking.lb_sg
    public_subnets           = module.networking.public_subnets
    tg_port                  = 80
    tg_protocol              = "HTTP"
    vpc_id                   = module.networking.vpc_id
    app_asg                  = module.compute.app_asg
    listener_port            = 80
    listener_protocol        = "HTTP"
    azs                      = 2
  }
```

root main.tf hosted with ❤ by GitHub view raw

```
# --- networking/main.tf ---


### CUSTOM VPC CONFIGURATION

resource "random_integer" "random" {
  min = 1
  max = 100
}

resource "aws_vpc" "three_tier_vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "three_tier_vpc-${random_integer.random.id}"
  }
  lifecycle {
    create_before_destroy = true
  }
}

data "aws_availability_zones" "available" {
}

### INTERNET GATEWAY

resource "aws_internet_gateway" "three_tier_internet_gateway" {
  vpc_id = aws_vpc.three_tier_vpc.id

  tags = {
    Name = "three_tier_igw"
  }
  lifecycle {
    create_before_destroy = true
  }
}
```

networking-vpc-ig-main.tf hosted with ❤️ by GitHub view raw

Next up we will create the public subnets using a count variable to control the number we want. We will set up the cidr_block so that the subnets can exist within the specified VPC cidr range. The public subnet route table will route to the internet gateway. We will also create the NAT gateway that will connect with our private instances.

```
### PUBLIC SUBNETS (WEB TIER) AND ASSOCIATED ROUTE TABLES

resource "aws_subnet" "three_tier_public_subnets" {
  count                   = var.public_sn_count
  vpc_id                  = aws_vpc.three_tier_vpc.id
  cidr_block              = "10.123.${10 + count.index}.0/24"
  map_public_ip_on_launch = true
  availability_zone       = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "three_tier_public_${count.index + 1}"
  }
}

resource "aws_route_table" "three_tier_public_rt" {
  vpc_id = aws_vpc.three_tier_vpc.id

  tags = {
    Name = "three_tier_public"
  }
}

resource "aws_route" "default_public_route" {
  route_table_id         = aws_route_table.three_tier_public_rt.id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id = aws_internet_gateway.three_tier_internet_gateway.id
}

resource "aws_route_table_association" "three_tier_public_assoc" {
  count          = var.public_sn_count
  subnet_id      = aws_subnet.three_tier_public_subnets.*.id[count.index]
  route_table_id = aws_route_table.three_tier_public_rt.id
}


### EIP AND NAT GATEWAY

resource "aws_eip" "three_tier_nat_eip" {
  vpc = true
}

resource "aws_nat_gateway" "three_tier_ngw" {
  allocation_id     = aws_eip.three_tier_nat_eip.id
  subnet_id         = aws_subnet.three_tier_public_subnets[1].id
}
```

networking-public-subnet-main.tf hosted with ❤️ by GitHub view raw

We will create the private subnets next that will reside in the application tier and database tier. Here will will associate a private route table with the NAT Gateway.

```
### PRIVATE SUBNETS (APP TIER & DATABASE TIER) AND ASSOCIATED ROUTE TABLES

resource "aws_subnet" "three_tier_private_subnets" {
  count                   = var.private_sn_count
  vpc_id                  = aws_vpc.three_tier_vpc.id
  cidr_block              = "10.123.${20 + count.index}.0/24"
  map_public_ip_on_launch = false
  availability_zone       = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "three_tier_private_${count.index + 1}"
  }
}

resource "aws_route_table"
  vpc_id = aws_vpc.three_tier_vpc.id

  tags = {
    Name = "three_tier_private"
  }
}

resource "aws_route" "default_private_route" {
  route_table_id         = aws_route_table.three_tier_private_rt.id
  destination_cidr_block = "0.0.0.0/0"
  nat_gateway_id = aws_nat_gateway.three_tier_ngw.id
}


resource "aws_route_table_association" "three_tier_private_assoc" {
  count          = var.private_sn_count
  route_table_id = aws_route_table.three_tier_private_rt.id
  subnet_id      = aws_subnet.three_tier_private_subnets.*.id[count.index]
}


resource "aws_subnet" "three_tier_private_subnets_db" {
  count                   = var.private_sn_count
  vpc_id                  = aws_vpc.three_tier_vpc.id
  cidr_block              = "10.123.${40 + count.index}.0/24"
  map_public_ip_on_launch = false
  availability_zone       = data.aws_availability_zones.available.names[count.index]

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

networking-private-subnet-main.tf hosted with ❤️ by GitHub view raw

For the last part of the *main.tf* networking file, we will make security groups (sg) to allow proper permissions for each level. The bastion sg will allow you to connect to the bastion EC2 instance. Ideally, you would set the ***access_ip*** variable to your IP address. I have set it to allow any IP address, but that is not most secure. We have the load balancer sg, frontend app sg, backend app sg, and database sg. The description in each block tells us their purpose. I have also thrown in a database subnet group so that we can add it to our database subnets we created earlier.

```
### SECURITY GROUPS

resource "aws_security_group" "three_tier_bastion_sg" {
  name        = "three_tier_bastion_sg"
  description = "Allow SSH Inbound Traffic From Set IP"
  vpc_id      = aws_vpc.three_tier_vpc.id

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [var.access_ip]
  }

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


resource "aws_security_group" "three_tier_lb_sg" {
  name        = "three_tier_lb_sg"
  description = "Allow Inbound HTTP Traffic"
  vpc_id      = aws_vpc.three_tier_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    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"]
  }
}

resource "aws_security_group" "three_tier_frontend_app_sg" {
  name        = "three_tier_frontend_app_sg"
  description = "Allow SSH inbound traffic from Bastion, and HTTP inbound traffic
from loadbalancer"
  vpc_id      = aws_vpc.three_tier_vpc.id
```

```
48
49    ingress {
50      from_port       = 22
51      to_port         = 22
52      protocol        = "tcp"
```

```
1   # --- networking/variables.tf ---
2
3   variable "vpc_cidr" {
4     type = string
5   }
6
7   variable "public_sn_count" {
8     type = number
9   }
10
11  variable "private_sn_count" {
12    type = number
13  }
14
15  variable "access_ip" {
16    type = string
17  }
18
19  variable "db_subnet_group" {
20    type = bool
21  }
22
23  variable "availabilityzone" {}
24
25  variable "azs" {}
```

networking-variables.tf hosted with ❤️ by GitHub view raw

```
79      protocol        = "tcp"
80      security_groups = [aws_security_group.three_tier_frontend_app_sg.id]
81    }
82
83    ingress {
84      from_port       = 22
85      to_port         = 22
86      protocol        = "tcp"
87      security_groups = [aws_security_group.three_tier_bastion_sg.id]
88    }
89
90    egress {
91      from_port   = 0
92      to_port     = 0
93      protocol    = "-1"
```

```
# --- networking/outputs.tf ---

output "vpc_id" {
  value = aws_vpc.three_tier_vpc.id
}

output "db_subnet_group_name" {
  value = aws_db_subnet_group.three_tier_rds_subnetgroup.*.name
}

output "rds_db_subnet_group" {
  value = aws_db_subnet_group.three_tier_rds_subnetgroup.*.id
}

output "rds_sg" {
  value = aws_security_group.three_tier_rds_sg.id
}

output "frontend_app_sg" {
  value = aws_security_group.three_tier_frontend_app_sg.id
}

output "backend_app_sg" {
  value = aws_security_group.three_tier_backend_app_sg.id
}

output "bastion_sg" {
  value = aws_security_group.three_tier_bastion_sg.id
}

output "lb_sg" {
  value = aws_security_group.three_tier_lb_sg.id
}

output "public_subnets" {
  value = aws_subnet.three_tier_public_subnets.*.id
}

output "private_subnets" {
  value = aws_subnet.three_tier_private_subnets.*.id
}

output "private_subnets_db" {
  value = aws_subnet.three_tier_private_subnets_db.*.id
}
```

networking-outputs.tf hosted with ❤️ by GitHub view raw

Compute

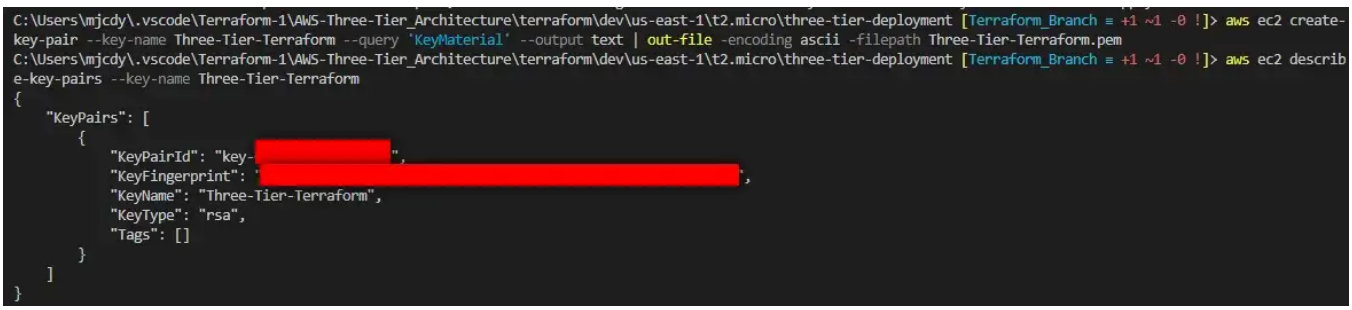
We will now head over to the compute module. For our ***main.tf*** file, we will obtain the latest AMI using the AWS SSM parameter store. Next, to be able to access our Bastion host, we will need a key pair. Let's generate one from the CLI. Enter the following command. Replace the ***italicized*** characters with your own:

```
aws ec2 create-key-pair --key-name MyKeyPair --query 'KeyMaterial' -
-output text | out-file -encoding ascii -filepath MyKeyPair.pem
```

To display the keypair you created, enter the following command:

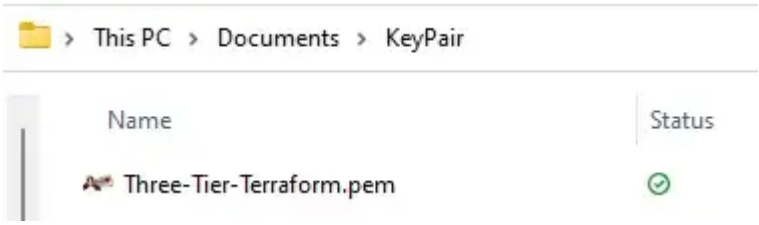
```
aws ec2 describe-key-pairs --key-name MyKeyPair
```

Your AWS account will hold the keypair. Here is what the process looks like from my command line:



For Windows

I use X Certificate and Key Manager to store my *keypair.pem* files. Make sure to locate the PEM file you made in your Windows filesystem. Create a keypair folder and move the PEM file to that location. Copy the path in your Windows file system by right clicking on the keypair PEM file:



We will come back to the keypair later.

Now we will create our auto scaling groups in the private subnets. We will attach our scripts to the appropriate group. The bastion auto scaling group could be set as a single instance, but we will create an autoscaling group because a failed host will be replaced automatically if EC2 health checks are failed. Finally we will attach the frontend group to the internet facing load balancer.

```
# --- compute/main.tf ---


# LATEST AMI FROM PARAMETER STORE

data "aws_ssm_parameter" "three-tier-ami" {
  name = "/aws/service/ami-amazon-linux-latest/amzn2-ami-hvm-x86_64-gp2"
}


# LAUNCH TEMPLATES AND AUTOSCALING GROUPS FOR BASTION

resource "aws_launch_template" "three_tier_bastion" {
  name_prefix            = "three_tier_bastion"
  instance_type          = var.instance_type
  image_id               = data.aws_ssm_parameter.three-tier-ami.value
  vpc_security_group_ids = [var.bastion_sg]
  key_name               = var.key_name

  tags = {
    Name = "three_tier_bastion"
  }
}

resource "aws_autoscaling_group" "three_tier_bastion" {
  name                = "three_tier_bastion"
  vpc_zone_identifier = var.public_subnets
  min_size            = 1
  max_size            = 1
  desired_capacity    = 1

  launch_template {
    id      = aws_launch_template.three_tier_bastion.id
    version = "$Latest"
  }
}


# LAUNCH TEMPLATES AND AUTOSCALING GROUPS FOR FRONTEND APP TIER

resource "aws_launch_template" "three_tier_app" {
  name_prefix            = "three_tier_app"
  instance_type          = var.instance_type
  image_id               = data.aws_ssm_parameter.three-tier-ami.value
  vpc_security_group_ids = [var.frontend_app_sg]
  user_data              = filebase64("install_apache.sh")
  key_name               = var.key_name

```

```
    tags = {
      Name = "three_tier_app"
```

```
# --- compute/variables.tf ---

variable "bastion_sg" {}
variable "frontend_app_sg" {}
variable "backend_app_sg" {}
variable "private_subnets" {}
variable "public_subnets" {}
variable "key_name" {}
variable "lb_tg_name" {}
variable "lb_tg" {}

variable "bastion_instance_count" {
  type = number
}

variable "instance_type" {
  type = string
}
```

compute-variables.tf hosted with ❤️ by GitHub view raw

```
  }

```

```
# --- compute/outputs.tf ---

output "app_asg" {
  value = aws_autoscaling_group.three_tier_app
}

output "app_backend_asg" {
  value = aws_autoscaling_group.three_tier_backend
}
```

compute-outputs.tf hosted with ❤️ by GitHub view raw

```

    tags = {
      Name = "three_tier_backend"
    }
  }

  resource "aws_autoscaling_group" "three_tier_backend" {
    name                = "three_tier_backend"
    vpc_zone_identifier = var.private_subnets
    min_size            = 2
    max_size            = 3
    desired_capacity    = 2

```

```
# --- loadbalancing/main.tf ---


# INTERNET FACING LOAD BALANCER

resource "aws_lb" "three_tier_lb" {
  name            = "three-tier-loadbalancer"
  security_groups = [var.lb_sg]
  subnets         = var.public_subnets
  idle_timeout    = 400

  depends_on = [
    var.app_asg
  ]
}

resource "aws_lb_target_group" "three_tier_tg" {
  name     = "three-tier-lb-tg-${substr(uuid(), 0, 3)}"
  port     = var.tg_port
  protocol = var.tg_protocol
  vpc_id   = var.vpc_id

  lifecycle {
    ignore_changes        = [name]
    create_before_destroy = true
  }
}

resource "aws_lb_listener" "three_tier_lb_listener" {
  load_balancer_arn = aws_lb.three_tier_lb.arn
  port              = var.listener_port
  protocol          = var.listener_protocol
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.three_tier_tg.arn
  }
}
```

loadbalancing-main.tf hosted with ❤️ by GitHub view raw

Below are the *variables.tf* and *outputs.tf* files for the loadbalancing module:

Database

Onto the last module. For the ***main.tf*** file here, we have a block that builds the database. One thing you could add for additional security is a a KMS key to encrypt the database.

Below are the ***variables.tf*** and ***outputs.tf*** files for the database:

Finally, back at the root folder, we will add the ***variables.tf*** file, ***outputs.tf*** file, and the ***tfvars*** file:

The ***outputs.tf*** root file will tell Terraform to output the code values in our terminal. We will return the endpoints for our load balancer and database:

Here are the scripts I used to install Apache and Node.js. The Apache script will allow you to test the internet facing load balancer:

Terraform Commands

Now that the infrastructure as code is set up, we can apply it to our AWS account. From the root directory, run a `terraform init`, then `terraform validate` if you want to see the validity of your code, `terraform plan` to map out the resources you will create, and `terraform apply` to execute the plan!

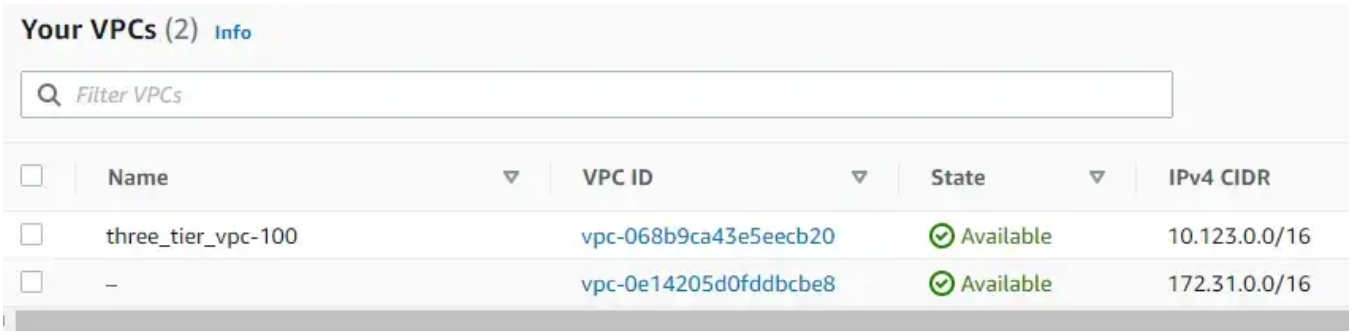
After your plan, you will see the number of resources to be created, and the outputs you will be shown. After the apply is complete, you should see that all of your resources have been created, along with the load balancer endpoints.

```
Apply complete! Resources: 37 added, 0 changed, 0 destroyed.

Outputs:

database_endpoint = "three-tier-db.clmzzmv9z7zf.us-east-1.rds.amazonaws.com:3306"
load_balancer_endpoint = "three-tier-loadbalancer-1977327318.us-east-1.elb.amazonaws.com"
```

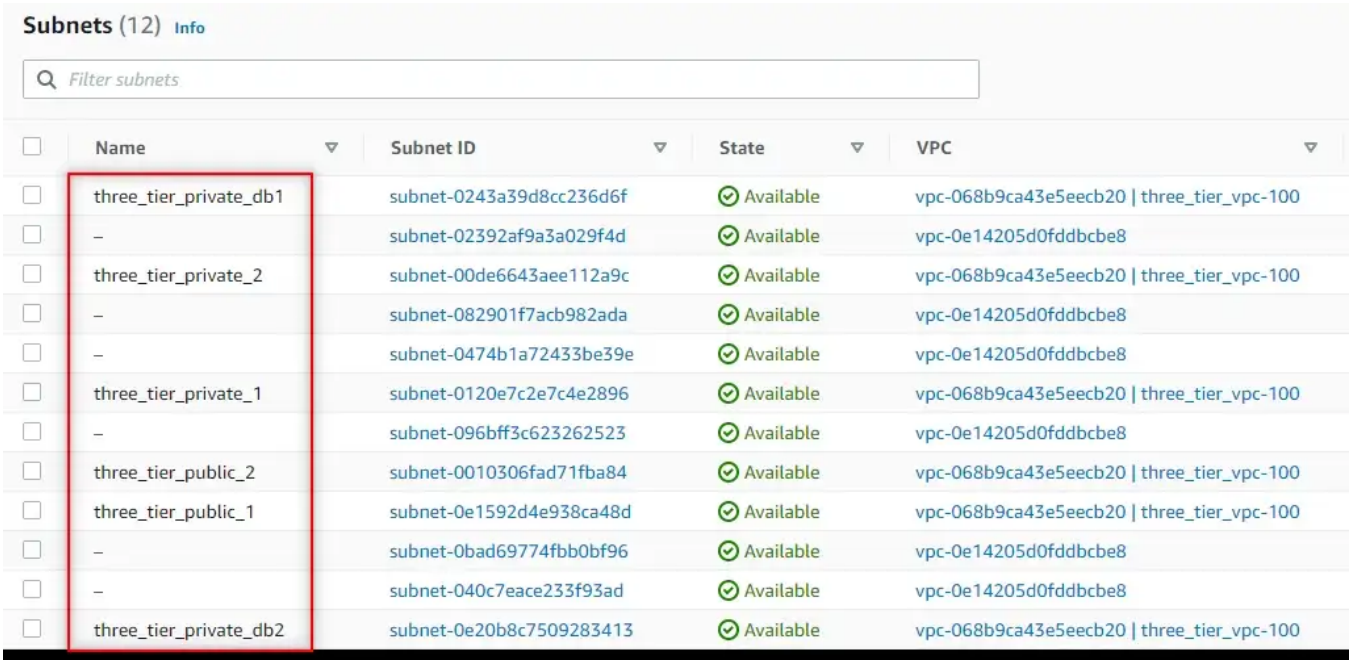
Testing

If you go into your AWS console, you should be able to see the VPC and subnets, internet gateway, route tables and associations, EC2 instances running in the proper locations, load balancers, and RDS database.

VPC



Subnets



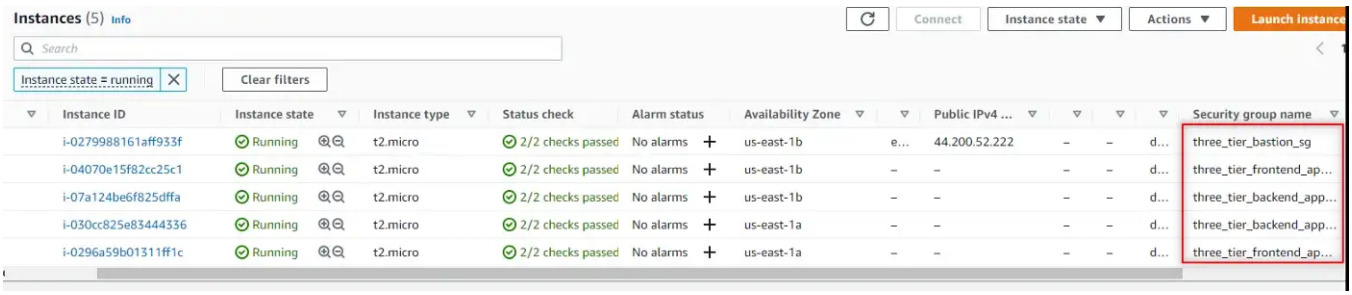
Subnets (12) Info

Filter subnets

Name	Subnet ID	State	VPC
three_tier_private_db1	subnet-0243a39d8cc236d6f	Available	vpc-068b9ca43e5eecb20 | three_tier_vpc-100
–	subnet-02392af9a3a029f4d	Available	vpc-0e14205d0fddbcbe8
three_tier_private_2	subnet-00de6643aee112a9c	Available	vpc-068b9ca43e5eecb20 | three_tier_vpc-100
–	subnet-082901f7acb982ada	Available	vpc-0e14205d0fddbcbe8
–	subnet-0474b1a72433be39e	Available	vpc-0e14205d0fddbcbe8
three_tier_private_1	subnet-0120e7c2e7c4e2896	Available	vpc-068b9ca43e5eecb20 | three_tier_vpc-100
–	subnet-096bff3c623262523	Available	vpc-0e14205d0fddbcbe8
three_tier_public_2	subnet-0010306fad71fba84	Available	vpc-068b9ca43e5eecb20 | three_tier_vpc-100
three_tier_public_1	subnet-0e1592d4e938ca48d	Available	vpc-068b9ca43e5eecb20 | three_tier_vpc-100
–	subnet-0bad69774fbb0bf96	Available	vpc-0e14205d0fddbcbe8
–	subnet-040c7eace233f93ad	Available	vpc-0e14205d0fddbcbe8
three_tier_private_db2	subnet-0e20b8c7509283413	Available	vpc-068b9ca43e5eecb20 | three_tier_vpc-100

EC2 instances

Only the Bastion will have a public IP address:



Instances (5) Info

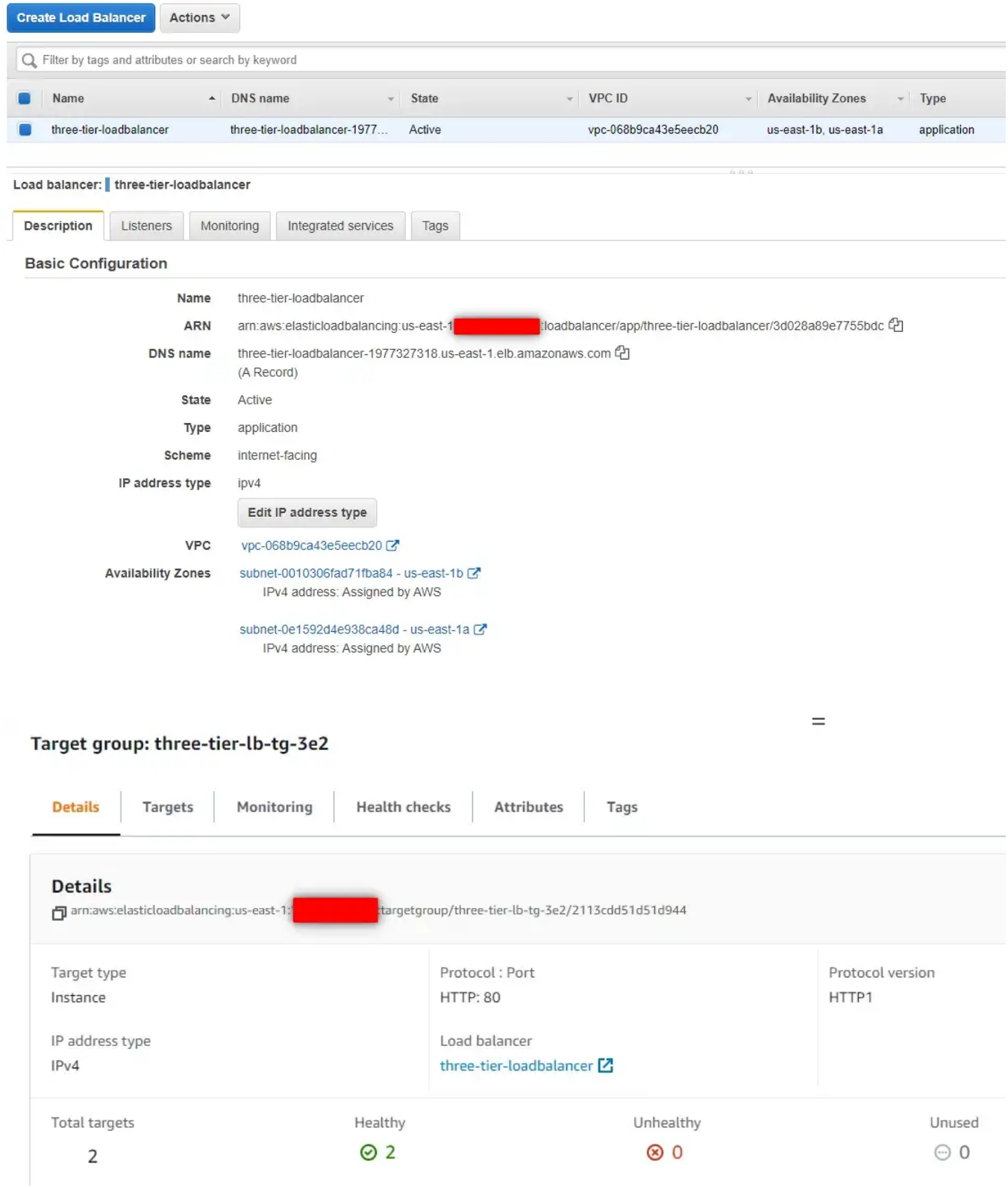
Connect | Instance state | Actions | Launch instance

Search

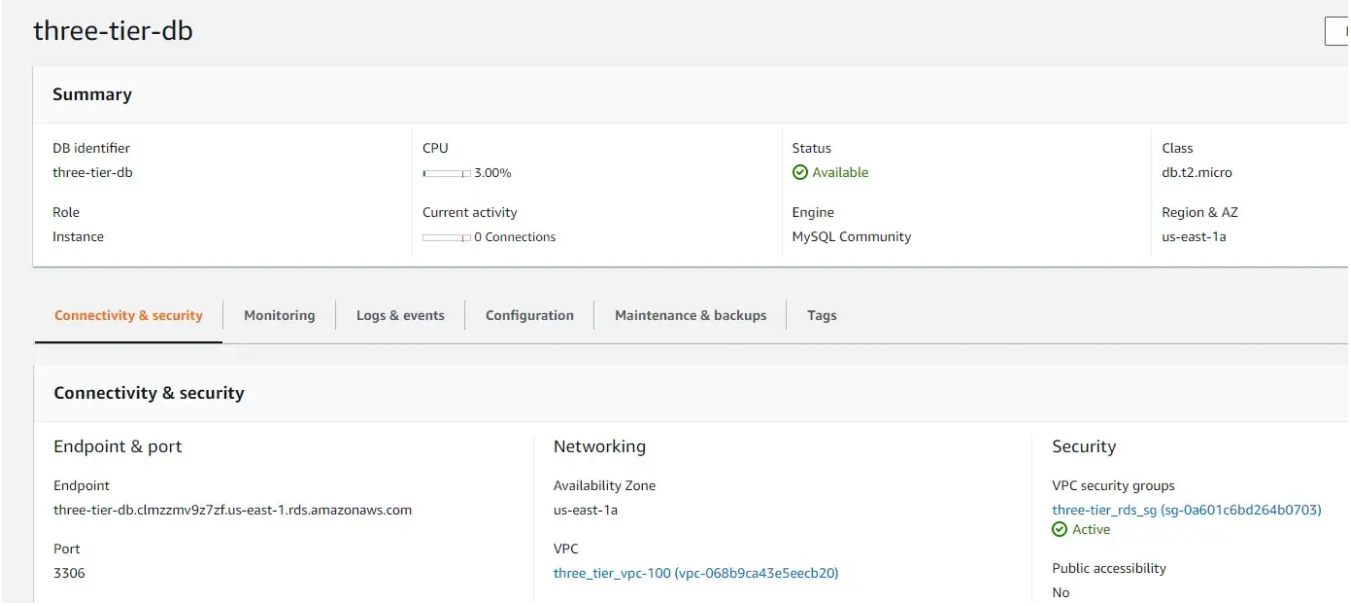
Instance state = running | Clear filters

Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 ...	Security group name
i-0279988161aff933f	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	44.200.52.222	three_tier_bastion_sg
i-04070e15f82cc25c1	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	–	three_tier_frontend_ap...
i-07a124be6f825dffa	Running	t2.micro	2/2 checks passed	No alarms	us-east-1b	–	three_tier_backend_app...
i-030cc825e83444336	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	–	three_tier_backend_app...
i-0296a59b01311ff1c	Running	t2.micro	2/2 checks passed	No alarms	us-east-1a	–	three_tier_frontend_ap...

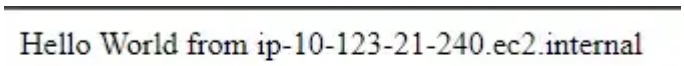
Load Balancer and Target Group


Create Load Balancer
Actions
Filter by tags and attributes or search by keyword
Name
DNS name
State
VPC ID
Availability Zones
Type
three-tier-loadbalancer
three-tier-loadbalancer-1977...
Active
vpc-068b9ca43e5eecb20
us-east-1b, us-east-1a
application
Load balancer: three-tier-loadbalancer
Description
Listeners
Monitoring
Integrated services
Tags
Basic Configuration
Name
three-tier-loadbalancer
ARN
arn:aws:elasticloadbalancing:us-east-1
:loadbalancer/app/three-tier-loadbalancer/3d028a89e7755bdc
DNS name
three-tier-loadbalancer-1977327318.us-east-1.elb.amazonaws.com
(A Record)
State
Active
Type
application
Scheme
internet-facing
IP address type
ipv4
Edit IP address type
VPC
vpc-068b9ca43e5eecb20
Availability Zones
subnet-0010306fad71fba84 - us-east-1b
IPv4 address: Assigned by AWS
subnet-0e1592d4e938ca48d - us-east-1a
IPv4 address: Assigned by AWS
Target group: three-tier-lb-tg-3e2
Details
Targets
Monitoring
Health checks
Attributes
Tags
Details
arn:aws:elasticloadbalancing:us-east-1:
:targetgroup/three-tier-lb-tg-3e2/2113cdd51d51d944
Target type
Instance
Protocol : Port
HTTP: 80
Protocol version
HTTP1
IP address type
IPv4
Load balancer
three-tier-loadbalancer
Total targets
2
Healthy
2
Unhealthy
0
Unused
0

Database



If we copy the load balancer endpoint we got from our Terraform output, and place it in the search bar, we will see the message we specified in our script for the Apache webserver.



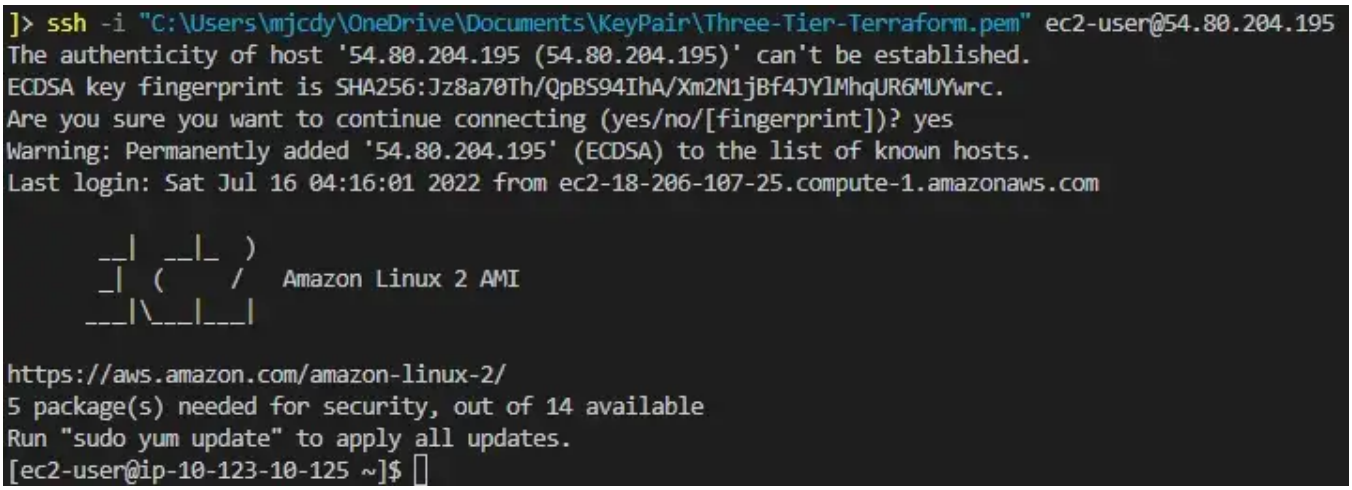
If we refresh the page, we should see the IP address from the other instance in our frontend autoscaling group.

Make sure to test out the infrastructure. You will need to use the keypair to SSH into the bastion host. Locate the public IP address of your Bastion Instance in the console:

```
ssh -i <Keypair_Path> ec2-user@<Public_IP_Address>
```

For example, my file path included looks like this:

```
ssh -i "C:\Users\mjcdy\OneDrive\Documents\KeyPair\Three-Tier-Terraform.pem" ec2-user@54.80.204.195
```

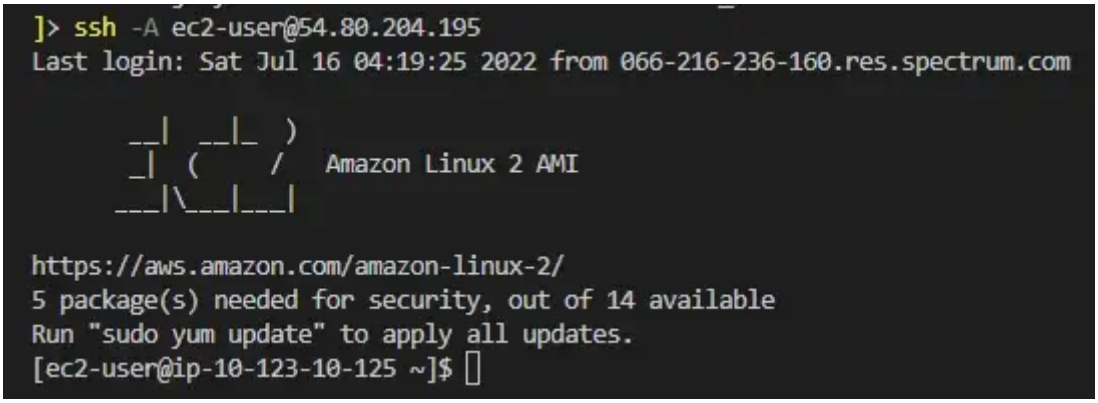


Important

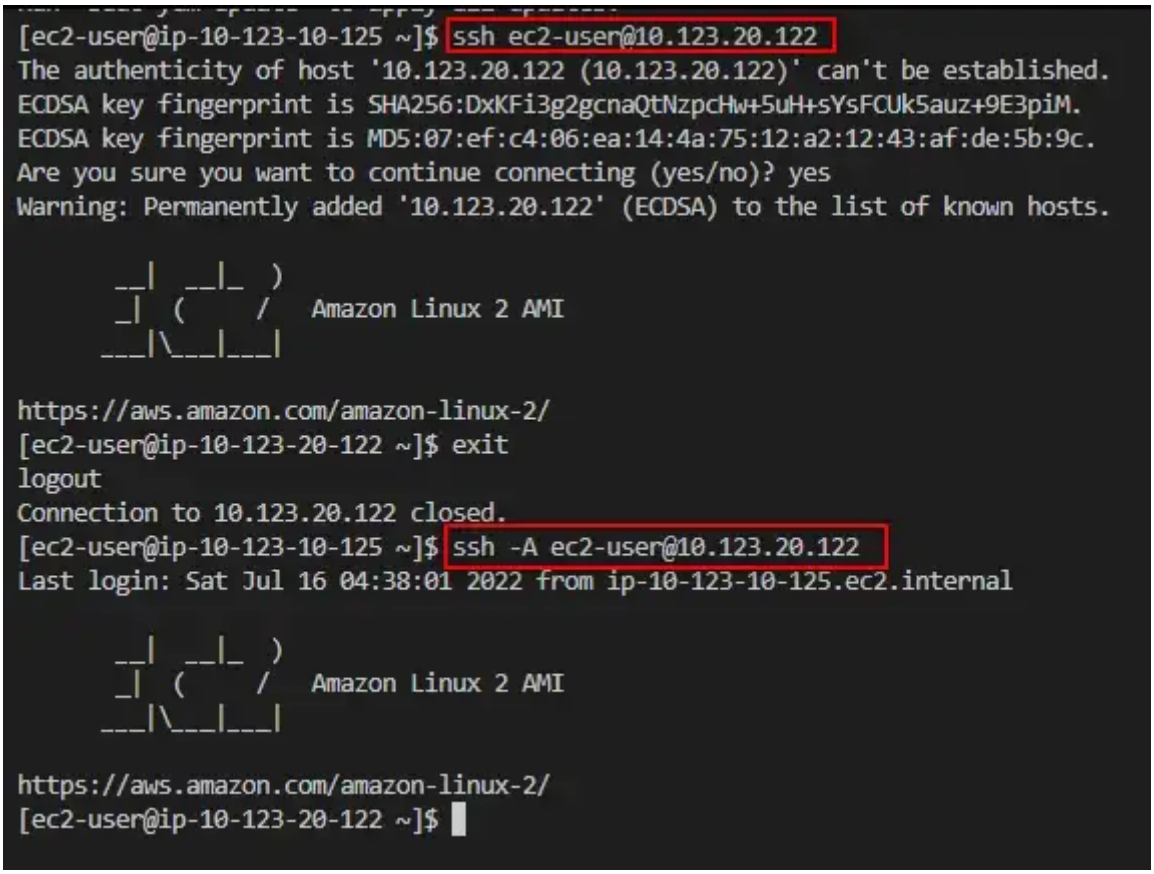
If you want to SSH into the backend application instances, you will need the SSH agent if you are using Windows PowerShell. You should be able to SSH into the frontend, then backend private subnets as well. Simply use the private IP when going into the private subnets. You will need to bring your keypair with you to each instance. Here is the command you would use to do so when entering the Bastion host:

```
ssh -A ec2-user@<Public_IP_Address>
```

This assumes your SSH agent is set up and the keypair path has been added to the agent.

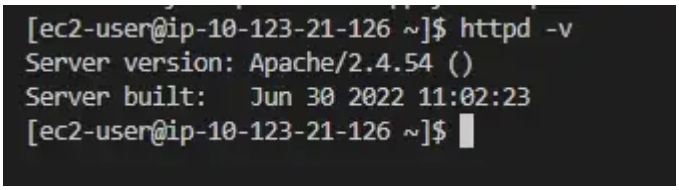


Now that we are in the Bastion Host instance, let's move onto the frontend application instances where the Apache webserver is installed:



As you can see, I forgot to bring the keys with me, so I had to go back to the Bastion host to get them!

I will switch over to the other frontend instance in my autoscaling group and check to see if Apache is installed:



In order to get into one of the backend application instances, we will go back to the Bastion, and then go to the backend:

```
[ec2-user@ip-10-123-21-126 ~]$ exit
logout
Connection to 10.123.21.126 closed.
[ec2-user@ip-10-123-10-125 ~]$ ssh -A ec2-user@10.123.21.191
The authenticity of host '10.123.21.191 (10.123.21.191)' can't be established.
ECDSA key fingerprint is SHA256:SFnPO5ZpEhR5I7ApPeByC75OG+4vIWmf9fqvs1NUuz0.
ECDSA key fingerprint is MD5:df:9a:c3:b0:99:7b:71:bf:cc:62:a9:89:5b:58:63:30.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.123.21.191' (ECDSA) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
4 package(s) needed for security, out of 7 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-123-21-191 ~]$
```

Now we can check to see if Node.js was installed:

```
Run "sudo yum update" to apply all updates.
[ec2-user@ip-10-123-21-191 ~]$ node --version
v16.16.0
[ec2-user@ip-10-123-21-191 ~]$
```

Now we will install the MySQL driver to access our AWS database using Node.js. Use the command `npm install mysql`. Then we will make a JavaScript file to connect with the database:

```
[ec2-user@ip-10-123-21-191 ~]$ npm install mysql

added 11 packages, and audited 12 packages in 978ms

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 8.11.0 -> 8.14.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.14.0
npm notice Run npm install -g npm@8.14.0 to update!
npm notice
[ec2-user@ip-10-123-21-191 ~]$ touch mysql_connection.js
[ec2-user@ip-10-123-21-191 ~]$ ls
mysql_connection.js  node_modules  package.json  package-lock.json
[ec2-user@ip-10-123-21-191 ~]$ vim
[ec2-user@ip-10-123-21-191 ~]$ vim mysql_connection.js
```

Go in to edit the file using vim, or your preferred text editor:

```
var mysql = require('mysql');
var con = mysql.createConnection({
host: "database_endpoint from Terraform Output",
user: "dbuser from tfvars file",
password: "dbpassword from tfvars file"
});
con.connect(function(err) {
if (err) throw err;
console.log("Connected!");
});
```

```
var con = mysql.createConnection({
  host: "three-tier-db.clmzzmv9z7zf.us-east-1.rds.amazonaws.com",
  user: "admin",
  password: "[redacted]
});

con.connect(function(err) {
  if (err) throw err;
  console.log("Connected!");
});
~
~
```

Make sure to *:wq* then we will use the command `node yourfile.js`

```
[ec2-user@ip-10-123-21-191 ~]$ node mysql_connection.js
Connected!
```

Exit out of the instances. We are done here.

Make sure to run a `terraform destroy` to avoid incurring extra charges for your resources. Thank you for reading!

The directory where the repository for this code is located at:

https://github.com/Michael-Cassidy-88/Terraform/tree/main/AWS-Three-Tier_Architecture

*More content at **PlainEnglish.io**. Sign up for our **free weekly newsletter**. Follow us on **Twitter** and **LinkedIn**. Check out our **Community Discord** and join our **Talent Collective**.*

Terraform AWS Nodejs Cloud Computing Technology