

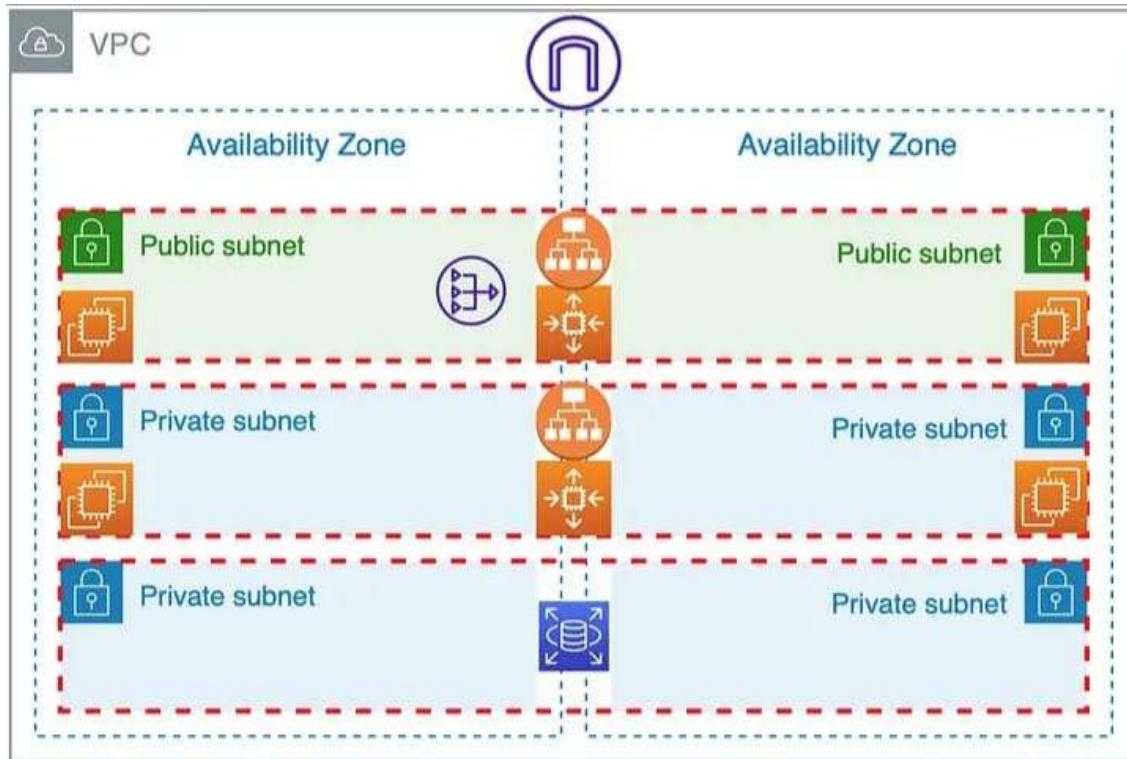
P. CHARAN KUMAR REDDY

Mail Id: charankumarreddyperam@gmail.com

Source code GitHub Link:

<https://github.com/charanreddy7958/3-Tier-Architecture-using-Terraform>

3-Tier Architecture using Terraform



What is a Three-Tier Architecture?

A **three-tier architecture** separates your application into:

1. **Presentation Layer** (Frontend)
2. **Application Layer** (Backend)
3. **Data Layer** (Database)

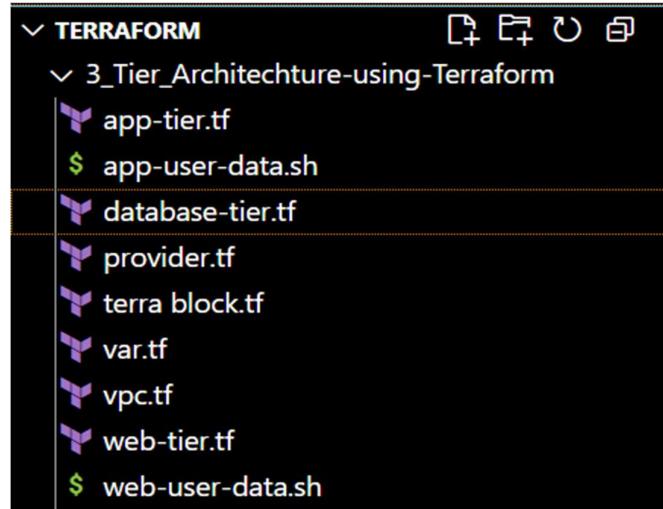
This improves scalability, security, and maintainability.

Steps to Deploy Three-Tier Architecture Using Terraform

❖ Step 1: Define Project Structure

three-tier-aws/

```
|── terraform-block.tf  
|── provider.tf  
|── variables.tf  
|── vpc.tf  
|── web-tier.tf  
|── app-tier.tf  
|── database-tier.tf  
|── user-data1.tf  
|── user-data2.tf
```



```
LENOVO T480@22BF5A1207 MINGW64 ~/Desktop/Terraform/3_Tier_Archite  
cture-using-Terraform  
$ $ terraform init  
Initializing the backend...  
Initializing provider plugins...  
- Finding hashicorp/aws versions matching "6.14.1"...  
- Installing hashicorp/aws v6.14.1...
```

Terraform initialization

```

terraform init
terraform validate
terraform plan
terraform apply

```

Terraform commands

by using the above commands run the terraform

◊ Step 2: Create VPC, Subnets and security group (vpc.tf)

- Create a custom VPC

The screenshot shows the AWS VPC dashboard with the title 'Your VPCs (1/2)'. It lists two VPCs: 'Project-3' (selected) and another unnamed one. The 'Project-3' row includes columns for Name, VPC ID, State, Block Public..., and IPv4 CIDR.

Name	VPC ID	State	Block Public...	IPv4 CIDR
-	vpc-0b3440b999cc49e8d	Available	Off	172.31.0.0/16
Project-3	vpc-04487230b9d7ec9a0	Available	Off	13.0.0.0/16

- Define public and private subnets

The screenshot shows the AWS Subnets page with the title 'Subnets (12)'. It lists 12 subnets across various VPCs. The subnets are grouped by VPC, with each VPC having a header row.

Name	Subnet ID	State	VPC
public-2	subnet-0ec093be66dbae1e9	Available	vpc-04487230b9d7ec9a0 Proj...
private-1	subnet-091025316f46a9a70	Available	vpc-04487230b9d7ec9a0 Proj...
public-1	subnet-015f7235479bc4a7e	Available	vpc-04487230b9d7ec9a0 Proj...
private-2	subnet-01741667f2f443bf7	Available	vpc-04487230b9d7ec9a0 Proj...
private-3	subnet-084e8ced66b4de52d	Available	vpc-04487230b9d7ec9a0 Proj...

- Add Internet Gateway and NAT Gateway

The screenshot shows the AWS Internet Gateways page with the title 'Internet gateways (2)'. It lists two Internet Gateways: 'IGW' and 'igw-0c2326fc9bd07e16'. The second gateway is attached to the 'Project-3' VPC.

Name	Internet gateway ID	State	VPC ID
IGW	igw-0c2326fc9bd07e16	Attached	vpc-04487230b9d7ec9a0 Proj...

NAT gateways (1) [Info](#)

Name	NAT gateway ID	Connectivity...	State	State message	Primary
gw NAT	nat-098231ecd74427189	Public	Available	-	34.231.128.1

NAT GATEWAY

- Route tables for public/private traffic

<input type="checkbox"/>	public-RT	rtb-0257fc9ea919b3e9a	2 subnets	-	No
<input type="checkbox"/>	private-RT	rtb-08676f2923d450131	4 subnets	-	No

ROUTE TABLES

- Create Security Group: Allow their protocols

Security Groups (18) [Info](#)

Name	Security group ID	Security group name	VPC ID
-	sg-0e5ce8886bdb3cf8	cloudfrontlb	vpc-0b3440b999cc49e8

SECURITY GROUPS

```

resource "aws_vpc" "main" { ... }

resource "aws_subnet" "public" { ... }

resource "aws_subnet" "private" { ... }

resource "aws_internet_gateway" "igw" { ... }

resource "aws_nat_gateway" "nat" { ... }

resource "aws_security_group" "frontend_sg" { ... }

resource "aws_security_group" "backend_sg" { ... }

resource "aws_security_group" "db_sg" { ... }

```

❖ Step 2: Launch EC2 Instances

- In public subnet
- Use Apache
- Assign public IP
- In private subnet
- Host Java/Tomcat

Instances (4) Info

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
webtier-1	i-0467474a3dd4504bf	Running	t3.micro	Initializing	View alarms +
apptier-1	i-0d105270207070e85	Running	t3.micro	Initializing	View alarms +
webtier-2	i-0bf2c262160da55ef	Running	t3.micro	3/3 checks passed	View alarms +
apptier-2	i-0d5b2e6dc2e2992d2	Running	t3.micro	3/3 checks passed	View alarms +

INSTANCES

- Add Auto Scaling Groups and Load Balancers

Target groups (2/3) Info

Name	ARN	Port	Protocol	Target type
apptier-tg	arn:aws:elasticloadbalancing:us-east-1:420053132464:targetgroup/apptier-tg/53f3433a2a2a4a20	80	HTTP	Instance
cloudfront	arn:aws:elasticloadbalancing:us-east-1:420053132464:targetgroup/cloudfront-544215232/53f3433a2a2a4a20	80	HTTP	Instance
webtier-tg	arn:aws:elasticloadbalancing:us-east-1:420053132464:targetgroup/webtier-tg/53f3433a2a2a4a20	80	HTTP	Instance

2 target groups selected

Amazon Machine Images (AMIs) (2) Info

Name	AMI name	AMI ID	Source	Own
terraform-ami	ami-0f54b1ad49a200674	420053132464/terraform-ami	420C	
terraform-ami1	ami-0511dd00488e8d7b	420053132464/terraform-ami1	420C	

TARGET GROUPS & AMI

The screenshot shows the AWS Elastic Load Balancing (ELB) console. The left sidebar has 'EC2' selected under 'Load balancers'. The main area is titled 'Load balancers (2/3)' and contains a table with two rows:

Name	Status	Type	Scheme	IP address type	VPC ID
apptier-lb	Active	application	Internet-facing	IPv4	vpc-04487
webtier-lb	Active	application	Internet-facing	IPv4	vpc-04487

LOAD BALANCERS

The screenshot shows the AWS Auto Scaling Groups console. The left sidebar has 'Auto Scaling Groups' selected under 'Auto Scaling'. The main area is titled 'Auto Scaling groups (2)' and contains a table with two rows:

Name	Launch template/configuration	Instances	Status	Desired
terraform-20251001153925281900000013	webtier-launch-template-202510011539	2	-	2
terraform-20251001153924810100000012	apptier-launch-template-202510011539	2	-	2

AUTO SCALING

The screenshot shows the AWS EC2 Instances console. The left sidebar has 'Instances' selected under 'EC2'. The main area is titled 'Instances (8/8)' and contains a table with eight rows:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status
apptier-asg-in...	i-0b2bd714de55763a3	Running	t3.micro	3/3 checks passed	View alarms +
webtier-1	i-046747a3dd45048f	Running	t3.micro	3/3 checks passed	View alarms +
apptier-1	i-0d105270207070e85	Running	t3.micro	3/3 checks passed	View alarms +
webtier-asg-in...	i-08eefdf87f4a3994c4	Running	t3.micro	Initializing	View alarms +
webtier-asg-in...	i-02405fd072663c1ea	Running	t3.micro	3/3 checks passed	View alarms +
webtier-2	i-0d8f2c262160da55ef	Running	t3.micro	3/3 checks passed	View alarms +
apptier-2	i-0d5b2e6dc2e2992d2	Running	t3.micro	3/3 checks passed	View alarms +

INSTANCES AFTER AUTO-SCALING

```
resource "aws_instance" "web-tier" { ... }
resource "aws_instance" "app-tier" { ... }
```

◊ Step 3: Create RDS Instance (rds.tf)

- In private subnet
- MySQL engine
- Enable multi-AZ (optional)
- Connect only from backend SG

The screenshot shows the AWS Aurora and RDS interface. On the left, there's a sidebar with options like Dashboard, Databases, Performance insights, Snapshots, Exports in Amazon S3, Automated backups, and Reserved instances. The main area is titled 'Subnet groups (1)'. It has a table with columns: Name, Description, Status, and VPC. One row is listed: 'my-db-subnet-group' (Managed by Terraform), Status 'Complete', and VPC 'vpc-04487230b9d7ec9a0'. There are buttons for 'Edit', 'Delete', and 'Create DB subnet group'.

SUBNET GROUP

- Connect to DB internally

The screenshot shows the AWS Aurora and RDS interface. On the left, there's a sidebar with options like Dashboard, Databases, Performance insights, Snapshots, Exports in Amazon S3, Automated backups, and Reserved instances. The main area is titled 'terrafo...'. It has a summary card with details: DB identifier 'terraform-20251001155822918700000001', Status 'Available', Class 'db.t3.micro', Role 'Instance', Current activity '0 Connections', Engine 'MySQL Community', and Region & AZ 'us-east-1b'. There are buttons for 'Modify' and 'Actions'.

DATABASE

```
resource "aws_db_instance" "rds" { ... }
```

◊ Step 4: Add Variables(var.tf)

- Use variables.tf for AMIs, instance types, subnet IDs, DB passwords.

Variable “instance types” {...}

Variable “password” {...}

◊ Step 5: Initialize and Deploy

terraform init

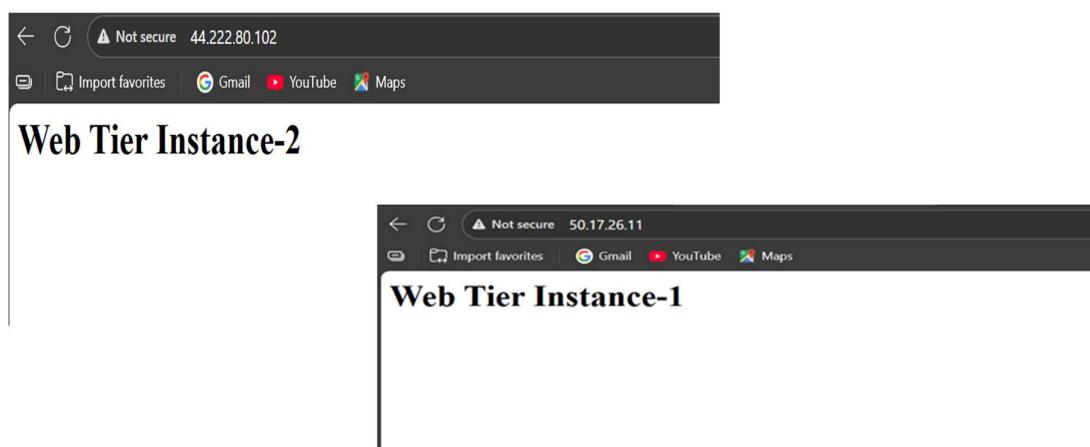
terraform validate

terraform plan

terraform apply

◊ Step 6: Test and Validate

- Access frontend via public IP



ACCESS WEB-SERVERS-1 & 2

- Ensure backend connects to DB using APP servers

```
root@ip-110-0-23-71:~# mysql -h database1.cxkeaqaaq645.eu-west-3.rds.amazonaws.com -u admin -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 50
Server version: 8.0.41 Source distribution

Copyright (c) 2000, 2025, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

- Create Database in it

```
mysql> CREATE DATABASE charan;
Query OK, 1 row affected (0.02 sec)
```

- By using above database create table and insert some values

```
mysql> use charan;
Database changed
mysql> CREATE TABLE Books (
    ->     BookID INT PRIMARY KEY,
    ->     Title VARCHAR(100),
    ->     Author VARCHAR(100),
    ->     PublishedYear INT,
    ->     Genre VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO Books (BookID, Title, Author, PublishedYear, Genre)
    -> VALUES
    -> (1, 'To Kill a Mockingbird', 'Harper Lee', 1960, 'Fiction'),
    -> (2, '1984', 'George Orwell', 1949, 'Dystopian'),
    -> (3, 'The Great Gatsby', 'F. Scott Fitzgerald', 1925, 'Classic'),
    -> (4, 'Sapiens', 'Yuval Noah Harari', 2011, 'Non-fiction');
Query OK, 4 rows affected (0.01 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> show tables;
+-----+
| Tables_in_charan |
+-----+
| Books           |
+-----+
1 row in set (0.00 sec)
```

- Connect other server, install MySQL server, and check the table is located or not which we created in app-server-1

```
A newer release of "Amazon Linux" is available.
Version 2023.9.20250929:
Run "/usr/bin/dnf check-release-update" for full release and version update info
      #
      #####
      #### \
      \###
      \#/ __-> https://aws.amazon.com/linux/amazon-linux-2023
      ~~. / /
      _/m/ ~
[ec2-user@ip-13-0-2-204 ~] $ █
```

```
[ec2-user@ip-13-0-2-204 ~]$ sudo dnf install mysql-community-client -y
Last metadata expiration check: 0:00:12 ago on Wed Oct  1 16:38:03 2025.
Package mysql-community-client-8.0.43-1.el9.x86_64 is already installed.
Dependencies resolved.
Nothing to do.
Complete!
[ec2-user@ip-13-0-2-204 ~]$ sudo systemctl start mysqld
sudo systemctl enable mysqld
[ec2-user@ip-13-0-2-204 ~]$ mysql -h terraform-2025100115582291870000001.ce5kw22wkcjd.us-east-1.rds.amazonaws.com -u charan -p
mysql> show databases;
+-----+
| Database |
+-----+
| charan   |
| information_schema |
| mysql    |
| performance_schema |
| sys      |
+-----+
5 rows in set (0.00 sec)

mysql> use charan;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-----+
| Tables_in_charan |
+-----+
| Books            |
+-----+
1 row in set (0.00 sec)

mysql> select* from Bokks;
ERROR 1146 (42S02): Table 'charan.Bokks' doesn't exist
mysql> select* from Books;
+-----+-----+-----+-----+
| BookID | Title        | Author       | Publishedyear | Genre      |
+-----+-----+-----+-----+
|      1 | To Kill a Mockingbird | Harper Lee   |          1960 | Fiction   |
|      2 | 1984           | George Orwell |          1949 | Dystopian |
+-----+-----+-----+-----+
```

- Successfully showed the table in app-server-2 which we created in the app-server-1.

Source code: <https://github.com/charanreddy7958/3-Tier-Architecture-using-Terraform>

CONCLUSION

The implementation of a **Three-Tier Architecture in AWS using Terraform** successfully demonstrates the power of Infrastructure as Code (IaC) in building scalable, secure, and modular cloud environments. By separating the application into **Web**, **Application**, and **Database** tiers, we achieved:

- **Improved fault isolation** and resource control
- **Automated provisioning** of EC2, VPC, and RDS components
- **Reusable Terraform modules** for consistent deployments
- **High availability** through Multi-AZ configurations and security best practices

This project highlights how DevOps and cloud engineering principles can be applied to design production-ready architectures with minimal manual intervention. It also reinforces the importance of modular design, version control, and declarative infrastructure in modern cloud workflows.