Terraform Guide

Naeem Dosh (Fiverr)

February 3, 2026

Contents

Terraform Infrastructure Guide

Project: TaxPlanner.app - HIPAA POC **Developer**: Naeem Dosh (Fiverr) **Infrastructure as Code**: Terraform **Version**: 1.0

Table of Contents

Overview

What is Terraform?

Terraform is an Infrastructure as Code (IaC) tool that allows you to define and provision cloud infrastructure using declarative configuration files.

Why Terraform for This Project?

- **Reproducible**: Deploy identical infrastructure anywhere
- **Version Controlled**: Track infrastructure changes in Git
- **Automated**: Reduce manual errors
- **Documented**: Code serves as documentation

- **Safe**: Preview changes before applying

Infrastructure Deployed

This Terraform configuration deploys: - VPC with public and private subnets - Application Load Balancer with SSL - EC2 instance with Docker - Security groups and IAM roles - S3 bucket for backups - Secrets Manager for credentials - CloudWatch logging

---

Prerequisites

Required Tools

1. **Terraform** (>= 1.0)

   ```
   # Install Terraform
   wget https://releases.hashicorp.com/terraform/1.6.0/terraform_1.6.0_linux_amd64.zip
   unzip terraform_1.6.0_linux_amd64.zip
   sudo mv terraform /usr/local/bin/
   terraform --version
   ```

2. **AWS CLI** (>= 2.0)

   ```
   # Install AWS CLI
   curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
   unzip awscliv2.zip
   sudo ./aws/install
   aws --version
   ```

3. **AWS Account**
   - Active AWS account
   - HIPAA BAA signed (if handling PHI)
   - Appropriate permissions

AWS Credentials

Configure AWS credentials:

```
aws configure

# You'll be prompted for:
AWS Access Key ID: YOUR_ACCESS_KEY
AWS Secret Access Key: YOUR_SECRET_KEY
Default region name: us-east-2
Default output format: json
```

Or use environment variables:

```
export AWS_ACCESS_KEY_ID="your-access-key"
export AWS_SECRET_ACCESS_KEY="your-secret-key"
export AWS_DEFAULT_REGION="us-east-2"
```

Directory Structure

```
terraform/
||| main.tf                  # Provider and backend configuration
||| variables.tf             # Input variables
||| outputs.tf               # Output values
||| vpc.tf                   # VPC, subnets, routing
||| security.tf              # Security groups
||| alb.tf                   # Load balancer, certificates
||| ec2.tf                   # EC2 instance
||| iam.tf                   # IAM roles and policies
||| s3.tf                    # S3 bucket for backups
||| secrets.tf               # AWS Secrets Manager
||| user_data.sh             # EC2 bootstrap script
||| terraform.tfvars         # Variable values (gitignored)
||| terraform.tfvars.example # Template
||| backend-setup/           # Initial S3/DynamoDB setup
    ||| main.tf
```

Terraform Files Explained

1. main.tf

Purpose: Provider configuration and remote state backend

```
terraform {
  required_version = ">= 1.0"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  backend "s3" {
    bucket         = "hipaa-poc-tfstate-730543776652"
    key            = "terraform.tfstate"
    region         = "us-east-2"
    encrypt        = true
    dynamodb_table = "hipaa-poc-tfstate-locks"
  }
}

provider "aws" {
  region = var.aws_region
  default_tags {
    tags = {
      Project     = var.project_name
      ManagedBy   = "Terraform"
      Environment = "Production"
    }
  }
}
```

Key Points: - Uses AWS provider version 5.x - Remote state in S3 (encrypted) - DynamoDB for state locking - Default tags applied to all resources

---

2. variables.tf

Purpose: Define input variables

```
variable "aws_region" {
  description = "AWS region"
  type        = string
  default     = "us-east-2"
}

variable "project_name" {
  description = "Project name prefix"
  type        = string
  default     = "hipaa-poc"
}

variable "domain_name" {
  description = "Domain name for SSL certificate"
  type        = string
}

variable "google_client_id" {
  description = "Google OAuth Client ID"
  type        = string
  sensitive   = true
}

variable "google_client_secret" {
  description = "Google OAuth Client Secret"
  type        = string
  sensitive   = true
}

variable "vpc_cidr" {
  description = "CIDR block for VPC"
  type        = string
  default     = "10.0.0.0/16"
}

variable "availability_zones" {
  description = "Availability zones"
  type        = list(string)
  default     = ["us-east-2a", "us-east-2b"]
}

variable "instance_type" {
  description = "EC2 instance type"
  type        = string
  default     = "t3.small"
}
```

Key Points: - Sensitive variables marked as sensitive - Default values provided where appropriate - Type validation for safety

---

3. vpc.tf

Purpose: Network infrastructure

Resources Created: - 1 VPC (10.0.0.0/16) - 2 Public subnets (10.0.0.0/24, 10.0.1.0/24) - 2 Private subnets (10.0.10.0/24, 10.0.11.0/24) - 1 Internet Gateway - Route tables and associations

```
# VPC
resource "aws_vpc" "main" {
  cidr_block           = var.vpc_cidr
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name = "${var.project_name}-vpc"
  }
}

# Public Subnets
resource "aws_subnet" "public" {
  count                   = 2
  vpc_id                  = aws_vpc.main.id
  cidr_block              = cidrsubnet(var.vpc_cidr, 8, count.index)
  availability_zone       = var.availability_zones[count.index]
  map_public_ip_on_launch = true

  tags = {
    Name = "${var.project_name}-public-${count.index + 1}"
  }
}

# Private Subnets
resource "aws_subnet" "private" {
  count             = 2
  vpc_id            = aws_vpc.main.id
  cidr_block        = cidrsubnet(var.vpc_cidr, 8, count.index + 10)
  availability_zone = var.availability_zones[count.index]

  tags = {
    Name = "${var.project_name}-private-${count.index + 1}"
  }
}
```

---

4. security.tf

Purpose: Security groups and network access control

Security Groups: 1. **ALB Security Group**: Allow HTTP/HTTPS from internet 2. **EC2 Security Group**: Allow traffic only from ALB

```
# ALB Security Group
resource "aws_security_group" "alb" {
  name_prefix = "${var.project_name}-alb-"
  vpc_id      = aws_vpc.main.id

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

  # HTTP from anywhere (redirects to HTTPS)
  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "HTTP from internet"
  }

  # Outbound to EC2
  egress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.ec2.id]
    description     = "To EC2 instances"
  }

  tags = {
    Name = "${var.project_name}-alb-sg"
  }
}

# EC2 Security Group
resource "aws_security_group" "ec2" {
  name_prefix = "${var.project_name}-ec2-"
  vpc_id      = aws_vpc.main.id

  # HTTP from ALB only
  ingress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.alb.id]
    description     = "HTTP from ALB"
  }

  # Outbound internet access
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
    description = "All outbound traffic"
  }

  tags = {
    Name = "${var.project_name}-ec2-sg"
  }
}
```

---

5. alb.tf

Purpose: Application Load Balancer and SSL certificate

Resources: - ACM Certificate (SSL/TLS) - Application Load Balancer - Target Group - HTTPS Listener - HTTP Listener (redirect)

```
# ACM Certificate
resource "aws_acm_certificate" "main" {
  domain_name       = var.domain_name
  validation_method = "DNS"

  lifecycle {
    create_before_destroy = true
  }

  tags = {
    Name = "${var.project_name}-cert"
  }
}

# Application Load Balancer
resource "aws_lb" "main" {
  name               = "${var.project_name}-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb.id]
  subnets            = aws_subnet.public[*].id

  enable_deletion_protection = false
  drop_invalid_header_fields = true

  tags = {
    Name = "${var.project_name}-alb"
  }
}

# Target Group
resource "aws_lb_target_group" "main" {
  name     = "${var.project_name}-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.main.id

  health_check {
    enabled             = true
    healthy_threshold   = 2
    interval            = 30
    matcher             = "200"
    path                = "/"
    port                = "traffic-port"
    protocol            = "HTTP"
    timeout             = 5
    unhealthy_threshold = 2
  }
  tags = {
    Name = "${var.project_name}-tg"
  }
}

# HTTPS Listener
resource "aws_lb_listener" "https" {
  load_balancer_arn = aws_lb.main.arn
  port              = 443
  protocol          = "HTTPS"
  ssl_policy        = "ELBSecurityPolicy-TLS13-1-2-2021-06"
  certificate_arn   = aws_acm_certificate.main.arn

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.main.arn
  }

  depends_on = [aws_acm_certificate.main]
}

# HTTP Listener (Redirect)
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.main.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
    type = "redirect"

    redirect {
      port        = "443"
      protocol    = "HTTPS"
      status_code = "HTTP_301"
    }
  }
}
```

---

6. ec2.tf

Purpose: EC2 instance configuration

Key Features: - Latest Amazon Linux 2023 AMI - User data script for bootstrap - Encrypted root and data volumes - IMDSv2 required - IAM instance profile attached

```
# Get latest Amazon Linux 2023 AMI
```

```
data "aws_ami" "amazon_linux_2023" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["al2023-ami-*-x86_64"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}

# EBS Volume for data
resource "aws_ebs_volume" "data" {
  availability_zone = var.availability_zones[0]
  size              = 30
  type              = "gp3"
  encrypted         = true

  tags = {
    Name = "${var.project_name}-data"
  }
}

# EC2 Instance
resource "aws_instance" "app" {
  ami           = data.aws_ami.amazon_linux_2023.id
  instance_type = var.instance_type
  subnet_id     = aws_subnet.private[0].id

  vpc_security_group_ids = [aws_security_group.ec2.id]
  iam_instance_profile   = aws_iam_instance_profile.ec2.name

  # User data for bootstrapping
  user_data = templatefile("${path.module}/user_data.sh", {
    secret_arn  = aws_secretsmanager_secret.app_secrets.arn
    aws_region  = var.aws_region
    domain_name = var.domain_name
    s3_bucket   = aws_s3_bucket.backups.id
  })

  # Root volume
  root_block_device {
    volume_size           = 30
    volume_type           = "gp3"
    encrypted             = true
    delete_on_termination = true
  }

  # Metadata options (IMDSv2)
  metadata_options {
    http_tokens                 = "required"
    http_put_response_hop_limit = 1
    http_endpoint               = "enabled"
  }

  tags = {
    Name = "${var.project_name}-app"
  }
}

# Attach data volume
resource "aws_volume_attachment" "data" {
  device_name = "/dev/xvdf"
  volume_id   = aws_ebs_volume.data.id
  instance_id = aws_instance.app.id
}

# Register with target group
resource "aws_lb_target_group_attachment" "main" {
  target_group_arn = aws_lb_target_group.main.arn
  target_id        = aws_instance.app.id
  port             = 80
}
```

---

7. iam.tf

Purpose: IAM roles and policies

Roles Created: 1. **EC2 Role**: For the application instance - SSM access (no SSH needed) - CloudWatch Logs - S3 backup access - Secrets Manager read

```
# IAM Role for EC2
resource "aws_iam_role" "ec2" {
  name = "${var.project_name}-ec2-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Action = "sts:AssumeRole"
      Effect = "Allow"
      Principal = {
        Service = "ec2.amazonaws.com"
      }
    }]
  })

  tags = {
    Name = "${var.project_name}-ec2-role"
  }
}

# Attach AWS managed policy for SSM
resource "aws_iam_role_policy_attachment" "ssm" {
  role       = aws_iam_role.ec2.name
  policy_arn = "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
}

# Attach AWS managed policy for CloudWatch
resource "aws_iam_role_policy_attachment" "cloudwatch" {
  role       = aws_iam_role.ec2.name
  policy_arn = "arn:aws:iam::aws:policy/CloudWatchAgentServerPolicy"
}

# Custom policy for secrets and S3
resource "aws_iam_role_policy" "app_permissions" {
  name = "${var.project_name}-app-permissions"
  role = aws_iam_role.ec2.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "secretsmanager:GetSecretValue"
        ]
        Resource = aws_secretsmanager_secret.app_secrets.arn
      },
      {
        Effect = "Allow"
        Action = [
          "s3:PutObject",
          "s3:GetObject",
          "s3:ListBucket"
        ]
        Resource = [
          aws_s3_bucket.backups.arn,
          "${aws_s3_bucket.backups.arn}/*"
        ]
      }
    ]
  })
}

# Instance Profile
resource "aws_iam_instance_profile" "ec2" {
  name = "${var.project_name}-ec2-profile"
  role = aws_iam_role.ec2.name
}
```

---

8. s3.tf

Purpose: S3 bucket for backups

```
# S3 Bucket for backups
resource "aws_s3_bucket" "backups" {
  bucket = "${var.project_name}-backups-${data.aws_caller_identity.current.account_id}"

  tags = {
    Name = "${var.project_name}-backups"
  }
}

# Enable versioning
resource "aws_s3_bucket_versioning" "backups" {
  bucket = aws_s3_bucket.backups.id

  versioning_configuration {
    status = "Enabled"
  }
}

# Enable encryption
resource "aws_s3_bucket_server_side_encryption_configuration" "backups" {
  bucket = aws_s3_bucket.backups.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}

# Block public access
resource "aws_s3_bucket_public_access_block" "backups" {
  bucket = aws_s3_bucket.backups.id
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

# Lifecycle rule for backups
resource "aws_s3_bucket_lifecycle_configuration" "backups" {
  bucket = aws_s3_bucket.backups.id

  rule {
    id     = "delete-old-backups"
    status = "Enabled"

    expiration {
      days = 30
    }
  }
}
```

---

9. secrets.tf

Purpose: Store sensitive credentials

```
# Generate random app secret
resource "random_password" "app_secret" {
  length  = 32
  special = true
}

# Generate random DB encryption key
resource "random_password" "db_key" {
  length  = 32
  special = false
}

# Secrets Manager Secret
resource "aws_secretsmanager_secret" "app_secrets" {
  name = "${var.project_name}/app-secrets"

  tags = {
    Name = "${var.project_name}-secrets"
  }
}

# Secret Version
```

```
resource "aws_secretsmanager_secret_version" "app_secrets" {
  secret_id = aws_secretsmanager_secret.app_secrets.id

  secret_string = jsonencode({
    GOOGLE_CLIENT_ID     = var.google_client_id
    GOOGLE_CLIENT_SECRET = var.google_client_secret
    APP_SECRET           = random_password.app_secret.result
    DB_ENCRYPTION_KEY    = random_password.db_key.result
  })
}
```

---

10. outputs.tf

Purpose: Display important values after deployment

```
output "alb_dns_name" {
  description = "DNS name of the load balancer"
  value       = aws_lb.main.dns_name
}

output "ec2_instance_id" {
  description = "ID of the EC2 instance"
  value       = aws_instance.app.id
}

output "s3_bucket_name" {
  description = "Name of the S3 backup bucket"
  value       = aws_s3_bucket.backups.id
}

output "acm_certificate_arn" {
  description = "ARN of the ACM certificate"
  value       = aws_acm_certificate.main.arn
}

output "acm_validation_records" {
  description = "DNS records for ACM validation"
  value = {
    for dvo in aws_acm_certificate.main.domain_validation_options : dvo.domain_name => {
      name  = dvo.resource_record_name
      type  = dvo.resource_record_type
      value = dvo.resource_record_value
    }
  }
}

output "ssm_connect_command" {
  description = "Command to connect to EC2 via SSM"
  value       = "aws ssm start-session --target ${aws_instance.app.id} --region ${var.aws_regi
}

output "next_steps" {
  description = "Next steps after deployment"
  value = <<EOT

NEXT STEPS:
===========
1. Add DNS CNAME record for ${var.domain_name}:
   - Name: ${var.domain_name}
   - Type: CNAME
   - Value: ${aws_lb.main.dns_name}

2. Validate ACM certificate by adding DNS record:
   (See acm_validation_records output above)

3. Update Google OAuth redirect URI to:
   https://${var.domain_name}/login.php

4. Connect to EC2 via SSM:
   aws ssm start-session --target ${aws_instance.app.id} --region ${var.aws_region}

EOT
}
```

Variables Configuration

Creating terraform.tfvars

1. Copy the example file:

   ```
   cd terraform
   cp terraform.tfvars.example terraform.tfvars
   ```

2. Edit with your values:

   ```
   # terraform.tfvars
   aws_region           = "us-east-2"
   project_name         = "hipaa-poc"
   domain_name          = "taxplanner.app"
   google_client_id     = "your-client-id.apps.googleusercontent.com"
   google_client_secret = "your-client-secret"

   # Optional overrides
   instance_type        = "t3.small"
   vpc_cidr             = "10.0.0.0/16"
   ```

3. **IMPORTANT**: Never commit `terraform.tfvars` to Git

   ```
   # Already in .gitignore
   *.tfvars
   !terraform.tfvars.example
   ```

Deployment Steps

Initial Deployment

Step 1: Setup Backend (One-time)

```
cd terraform/backend-setup
terraform init
terraform apply

# This creates:
# - S3 bucket for state
# - DynamoDB table for locking
```

Step 2: Configure Variables

```
cd ..
cp terraform.tfvars.example terraform.tfvars
nano terraform.tfvars  # Edit with your values
```

Step 3: Initialize Terraform

```
terraform init

# This will:
# - Download AWS provider
# - Configure S3 backend
# - Initialize modules
```

Step 4: Plan Deployment

```
terraform plan -out=tfplan

# Review the plan carefully
# Shows what will be created/modified/destroyed
```

Step 5: Apply Configuration

```
terraform apply tfplan

# Type 'yes' when prompted
# Takes ~10-15 minutes
```

Step 6: Note Outputs

```
terraform output

# Save important values:
# - ALB DNS name
# - ACM validation records
# - EC2 instance ID
```

Step 7: Configure DNS Add DNS records shown in output: 1. ACM validation CNAME 2. Domain CNAME | ALB

Step 8: Wait for Certificate

```
# Check certificate status
aws acm describe-certificate \
  --certificate-arn $(terraform output -raw acm_certificate_arn) \
  --region us-east-2 \
  --query 'Certificate.Status'

# Wait for "ISSUED" (5-30 minutes)
```

Step 9: Verify Application

```
# Test HTTP redirect
curl -I http://taxplanner.app

# Test HTTPS
curl -I https://taxplanner.app
```

Resource Details

Resources Created

Resource Type	Count	Purpose
VPC	1	Network isolation
Subnets	4	2 public + 2 private
Internet Gateway	1	Internet access
Route Tables	2	Traffic routing
Security Groups	2	ALB + EC2
ALB	1	Load balancing
Target Group	1	Health checks
ALB Listeners	2	HTTP + HTTPS
ACM Certificate	1	SSL/TLS
EC2 Instance	1	Application
EBS Volumes	2	Root + data
IAM Role	1	EC2 permissions

Resource Type	Count	Purpose
IAM Policies	3	SSM, CloudWatch, Custom
S3 Bucket	1	Backups
Secrets Manager	1	Credentials

Total: ~25 resources

Resource Dependencies

```
VPC
 ||| Subnets
 |    ||| Public Subnets
 |    |    ||| Internet Gateway
 |    |    ||| ALB
 |    ||| Private Subnets
 |         ||| EC2 Instance
 |              ||| EBS Volumes
 ||| Security Groups
 |    ||| ALB SG
 |    ||| EC2 SG
 ||| Route Tables

ACM Certificate | ALB Listener (HTTPS)
IAM Role | EC2 Instance
Secrets Manager | EC2 User Data
S3 Bucket | EC2 IAM Policy
```

State Management

Remote State Backend

Configuration: - **Storage**: S3 bucket (encrypted) - **Locking**: DynamoDB table - **Versioning**: Enabled - **Region**: us-east-2

State File Location:

```
s3://hipaa-poc-tfstate-730543776652/terraform.tfstate
```

State Commands

```
# View state
terraform state list

# Show specific resource
terraform state show aws_instance.app

# Pull remote state
```

```
terraform state pull > terraform.tfstate.backup

# Refresh state
terraform refresh
```

State Locking

Automatic locking prevents concurrent modifications:

```
DynamoDB Table: hipaa-poc-tfstate-locks
Key: LockID (S)
```

Troubleshooting

Common Issues

Issue 1: "Error creating VPC" **Cause**: Insufficient permissions or region limit

Solution:

```
# Check AWS credentials
aws sts get-caller-identity

# Verify permissions
aws iam get-user-policy --user-name your-user --policy-name your-policy

# Check VPC limits
aws ec2 describe-account-attributes --attribute-names max-vpcs
```

Issue 2: "Certificate stuck in PENDING_VALIDATION" **Cause**: DNS records not added correctly

Solution:

```
# Check DNS propagation
nslookup _xxx.taxplanner.app

# Verify CNAME record value matches exactly
terraform output acm_validation_records

# DNS can take 5-30 minutes to propagate
```

Issue 3: "Error: InvalidInstanceID" **Cause**: SSM agent not ready on EC2

Solution:

```
# Wait 2-3 minutes after instance launch
# Check instance status
aws ec2 describe-instance-status --instance-ids i-xxx

# Verify IAM role attached
```

```
aws ec2 describe-instances --instance-ids i-xxx \
  --query 'Reservations[0].Instances[0].IamInstanceProfile'
```

Issue 4: "Target unhealthy" **Cause**: Application not running or health check failing

Solution:

```
# Check target health
aws elbv2 describe-target-health \
  --target-group-arn $(terraform output -raw target_group_arn)

# Connect to instance
aws ssm start-session --target i-xxx

# Check application
docker ps
curl http://localhost
```

Terraform Commands for Debugging

```
# Enable debug logging
export TF_LOG=DEBUG
export TF_LOG_PATH=./terraform-debug.log

# Detailed plan
terraform plan -out=tfplan -var-file=terraform.tfvars

# Show plan in JSON
terraform show -json tfplan > plan.json

# Validate configuration
terraform validate

# Format code
terraform fmt -recursive

# Graph dependencies
terraform graph | dot -Tsvg > graph.svg
```

Maintenance

Updating Infrastructure

Minor Changes (variables, tags)

```
# Edit terraform.tfvars
nano terraform.tfvars

# Plan changes
```

```
terraform plan -out=tfplan

# Review carefully
terraform show tfplan

# Apply
terraform apply tfplan
```

Major Changes (instance type, AMI)

```
# Plan with target
terraform plan -target=aws_instance.app -out=tfplan

# Apply
terraform apply tfplan

# Verify
terraform output ec2_instance_id
```

Destroying Infrastructure

WARNING: This destroys all resources!

```
# Plan destruction
terraform plan -destroy -out=tfplan

# Review what will be destroyed
terraform show tfplan

# Destroy
terraform apply tfplan

# Or use destroy command
terraform destroy
```

Upgrading Terraform

```
# Check current version
terraform version

# Download new version
wget https://releases.hashicorp.com/terraform/1.7.0/terraform_1.7.0_linux_amd64.zip

# Replace binary
unzip terraform_1.7.0_linux_amd64.zip
sudo mv terraform /usr/local/bin/

# Re-initialize
terraform init -upgrade
```

Backup State File

```
# Download state
aws s3 cp s3://hipaa-poc-tfstate-730543776652/terraform.tfstate ./backup/

# Or use Terraform
terraform state pull > backup/terraform.tfstate.$(date +%Y%m%d)
```

Best Practices

Security

- Never commit `terraform.tfvars`
- Use remote state (S3) with encryption
- Enable state locking (DynamoDB)
- Use sensitive = true for secrets
- Review plans before applying
- Use specific provider versions

Code Organization

- One resource type per file
- Use meaningful variable names
- Add descriptions to variables
- Include outputs for important values
- Use consistent naming conventions
- Add comments for complex logic

Workflow

- Always run `terraform plan` first
- Review changes carefully
- Test in dev environment first
- Use workspaces for multiple environments
- Version control Terraform code
- Document changes in Git commits

Additional Resources

Official Documentation

- **Terraform**: https://www.terraform.io/docs
- **AWS Provider**: https://registry.terraform.io/providers/hashicorp/aws/latest/docs
- **Terraform Best Practices**: https://www.terraform-best-practices.com/

Useful Commands Reference

```
# Initialize
terraform init

# Validate
terraform validate

# Format
terraform fmt

# Plan
terraform plan

# Apply
terraform apply

# Destroy
terraform destroy

# Output
terraform output

# State
terraform state list
terraform state show RESOURCE

# Import existing resource
terraform import RESOURCE ID

# Taint (mark for recreation)
terraform taint RESOURCE

# Untaint
terraform untaint RESOURCE
```

Contact

Developer: Naeem Dosh **Platform**: Fiverr **Project**: TaxPlanner.app **Infrastructure**: Terraform v1.x **Provider**: AWS (us-east-2)

Date: February 3, 2026 **Version**: 1.0

End of Terraform Guide