



---

Project 09  
**INFRASTRUCTURE MONITORING AND LOG COLLECTION SYSTEM**

*Course Title : Cloud Computing*

**Submitted by :**  
Lujain Zia (2023-BSE-034)  
Alishba Zainab (2023-BSE-006)  
Fatima Kashif (2023-BSE-018)

**Submitted to :**  
Engr. Waqas Saleem

**Date : January 27, 2026**

---

**Department of Software Engineering**

Fatima Jinnah Women University

# Table of Contents

1. Executive Summary	3
2. Part 1: Git Repository & Structure	3
1.1 Repository Structure	3
1.2 .gitignore Configuration	5
1.3 Branching Strategy	6
3. Part 2: Terraform Infrastructure	7
2.1 Network Module - VPC & Subnets	7
2.2 Monitoring Server Module	9
2.3 Application Servers Module	11
4. Part 3: Ansible Configuration	20
3.1 Configure Application Servers	20
3.2 Configure Monitoring Server	21
3.3 Log Collection Using Ansible	28
5. Part 4: Dashboard & Reporting	29
4.1 Simple Web Dashboard	29
4.2 Automated Reporting	31
6. Part 5: Documentation & Basic Runbooks	34
7. GitHub Repository	37

# Executive Summary

This document presents the complete implementation of **Project 9: Infrastructure Monitoring and Log Collection System**. The project demonstrates the practical application of cloud computing concepts, infrastructure automation, and system monitoring using industry-standard tools including **Terraform**, **Ansible**, and **AWS services**.

The system architecture consists of a centralized monitoring server that continuously monitors multiple application servers, collects logs, generates reports, and displays real-time status through a web-based dashboard. All infrastructure is provisioned using Terraform (Infrastructure as Code), configured using Ansible (Configuration Management), and monitored using custom Bash scripts.

## Part 1: Git Repository & Structure

### 1.1 Repository Structure

The project follows a well-organized directory structure that separates concerns and facilitates maintainability. The repository is structured as follows:

```
● @ALISHBAZAINAB ➔ /workspaces/Project9/Project9 (main) $ tree
.
├── README.md
└── ansible
    ├── ansible.cfg
    ├── inventory
    │   ├── dev_aws_ec2.yml
    │   ├── production_aws_ec2.yml
    │   └── staging_aws_ec2.yml
    ├── playbooks
    │   ├── collect-logs.yml
    │   ├── configure-app-servers.yml
    │   ├── configure-monitoring-server.yml
    │   └── generate-report.yml
    └── roles
        ├── dashboard
        │   ├── tasks
        │   │   └── main.yml
        │   └── templates
        │       └── index.html.j2
        └── monitoring-tools
            └── files
                ├── build-dashboard.sh
                ├── check-services.sh
                └── collect-metrics.sh
```

Figure 1.1: Project Repository Structure - Root Level

```
@ALISHBAZAINAB ➔ /workspaces/Project9/Project9 (main) $ tree
.
├── main.yml
├── nginx-app
│   ├── tasks
│   │   └── main.yml
│   └── templates
│       └── nginx.conf.j2
├── scripts
│   ├── run-all-health-checks.sh
│   ├── run-daily-report.sh
│   └── run-weekly-report.sh
└── terraform
    ├── backend.tf
    ├── environments
    │   ├── dev.tfvars
    │   ├── production.tfvars
    │   └── staging.tfvars
    ├── main.tf
    └── modules
        ├── app-servers
        │   ├── main.tf
        │   ├── outputs.tf
        │   └── variables.tf
        └── monitoring-server
            ├── main.tf
            ├── outputs.tf
            └── variables.tf
```

*Figure 1.2: Project Repository Structure - Terraform Modules*

```
.
├── modules
│   ├── app-servers
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   ├── monitoring-server
│   │   ├── main.tf
│   │   ├── outputs.tf
│   │   └── variables.tf
│   └── network
│       ├── main.tf
│       ├── outputs.tf
│       └── variables.tf
└── outputs.tf
└── terraform.tfvars.example
└── variables.tf
└── web-ui
    ├── assets
    │   ├── css
    │   └── js
    └── index.html
```

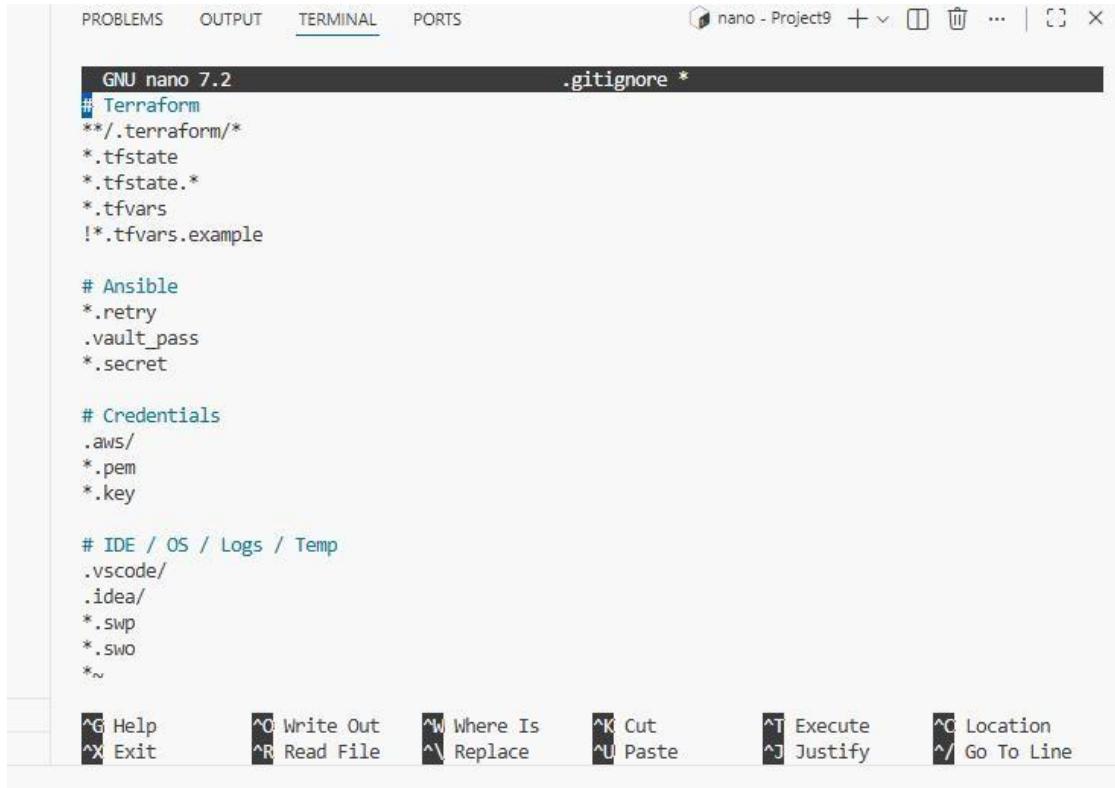
25 directories, 40 files

*Figure 1.3: Project Repository Structure - Ansible Roles*

The repository was initialized with Git and includes all necessary directories for Terraform modules, Ansible playbooks, roles, and monitoring scripts.

## 1.2 .gitignore Configuration

A comprehensive .gitignore file was created to exclude sensitive files, temporary files, and build artifacts from version control. This ensures that credentials, state files, and IDE-specific files are not accidentally committed to the repository.



The screenshot shows a terminal window with the nano text editor open. The title bar says "nano - Project9". The menu bar includes "PROBLEMS", "OUTPUT", "TERMINAL", and "PORTS". The status bar at the bottom shows keyboard shortcuts for various functions like Help, Exit, Write Out, Read File, Where Is, Replace, Cut, Paste, Execute, Justify, Location, and Go To Line.

```
GNU nano 7.2                               .gitignore *
# Terraform
**/.terraform/*
*.tfstate
*.tfstate.*
*.tfvars
!*.tfvars.example

# Ansible
*.retry
.vault_pass
*.secret

# Credentials
.aws/
*.pem
*.key

# IDE / OS / Logs / Temp
.vscode/
.idea/
*.swp
*.swo
*~
```

Figure 1.4: .gitignore File Configuration

The .gitignore includes exclusions for:

- Terraform state files and .terraform directories
- Ansible retry files and vault passwords
- AWS credentials and SSH keys
- IDE-specific files (.vscode, .idea)
- Temporary and log files

```

● @ALISHBAZAINAB →/workspaces/Project9/Project9 (main) $ git status
On branch main

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
  new file: .gitignore
  new file: README.md
  new file: ansible/ansible.cfg
  new file: ansible/inventory/dev_aws_ec2.yml
  new file: ansible/inventory/production_aws_ec2.yml
  new file: ansible/inventory/staging_aws_ec2.yml
  new file: ansible/playbooks/collect-logs.yml
  new file: ansible/playbooks/configure-app-servers.yml
  new file: ansible/playbooks/configure-monitoring-server.yml
  new file: ansible/playbooks/generate-report.yml
  new file: ansible/roles/dashboard/tasks/main.yml
  new file: ansible/roles/dashboard/templates/index.html.j2
  new file: ansible/roles/monitoring-tools/files/build-dashboard.sh
  new file: ansible/roles/monitoring-tools/files/check-services.sh
  new file: ansible/roles/monitoring-tools/files/collect-metrics.sh
  new file: ansible/roles/monitoring-tools/files/generate-report.sh
  new file: ansible/roles/monitoring-tools/files/http-health-check.sh
  new file: ansible/roles/monitoring-tools/tasks/main.yml
  new file: ansible/roles/nginx-app/tasks/main.yml

```

*Figure 1.5: Git Status Showing Clean Working Directory*

Git status was verified to ensure no sensitive files are being tracked in version control.

### 1.3 Branching Strategy

A simple yet effective branching strategy was implemented with main and dev branches. Feature branches are created from dev for experimental work, then merged back via pull requests.

```

● @ALISHBAZAINAB →/workspaces/Project9/Project9 (feature/add-monitoring-scripts) $ git log --oneline --graph --all --decorate
* 2b227bf (origin/main, origin/dev, main, dev) Merge branch 'main' of https://github.com/ALISHBAZAINAB/Project9
| \
| * d61de8c Initial commit
* fdc90a8 (HEAD → feature/add-monitoring-scripts, origin/feature/add-monitoring-scripts) Add placeholder for monitoring scripts
○ @ALISHBAZAINAB →/workspaces/Project9/Project9 (feature/add-monitoring-scripts) $ 

```

*Figure 1.6: Git Branching Strategy*

The branching model follows:

- main - Stable production-ready code
- dev - Active development branch
- feature/\* - Feature-specific branches merged into dev

# Part 2: Terraform Infrastructure

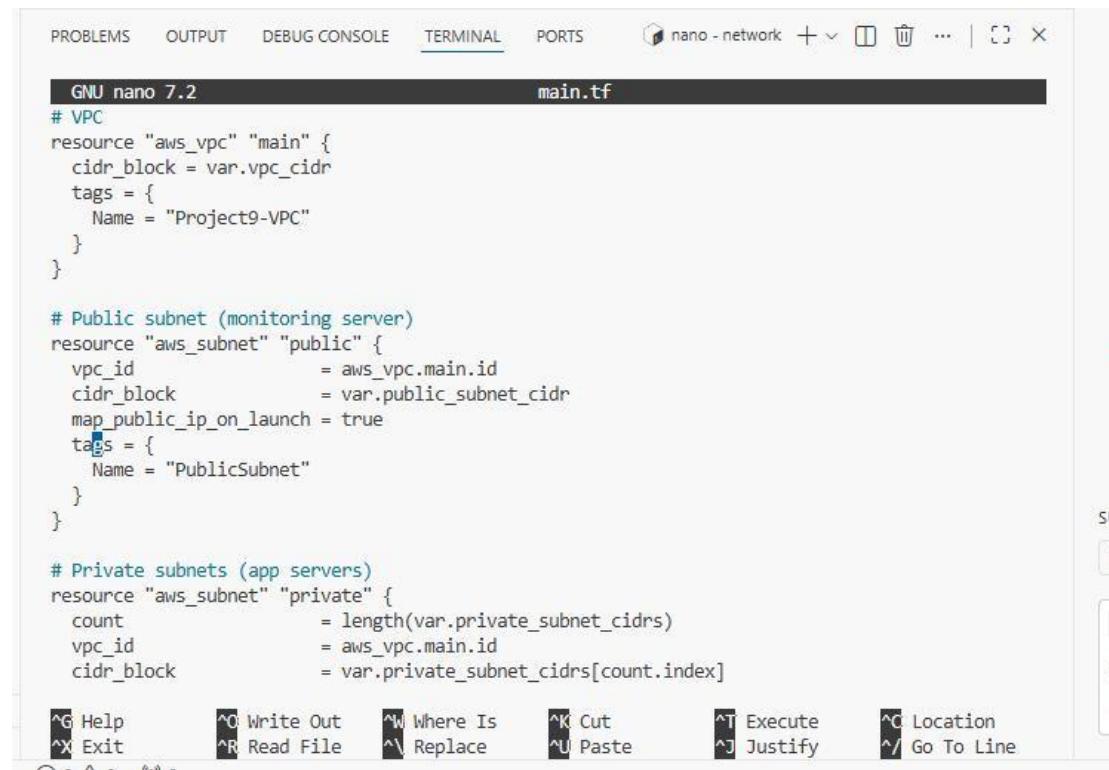
## 2.1 Network Module - VPC & Subnets

The network module creates the foundational AWS networking infrastructure including VPC, subnets etc. The architecture uses a public subnet for the monitoring server and private subnets for application servers.



```
git log --oneline --graph --all --decorate
@ALISHBAZAINAB →/workspaces/Project9/Project9 (feature/add-monitoring-scripts) $ git log --oneline --graph --all --decorate
* 2b227bf (origin/main, origin/dev, main, dev) Merge branch 'main' of https://github.com/ALISHBAZAINAB/Project9
| \
| * d61de8c Initial commit
* fd90a8 (HEAD → feature/add-monitoring-scripts, origin/feature/add-monitoring-scripts) Add placeholder for monitoring scripts
@ALISHBAZAINAB →/workspaces/Project9/Project9 (feature/add-monitoring-scripts) $
```

Figure 2.1: Network Module



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS nano - network + × ⌂ ... | ⌂ X
GNU nano 7.2 main.tf
# VPC
resource "aws_vpc" "main" {
  cidr_block = var.vpc_cidr
  tags = {
    Name = "Project9-VPC"
  }
}

# Public subnet (monitoring server)
resource "aws_subnet" "public" {
  vpc_id          = aws_vpc.main.id
  cidr_block     = var.public_subnet_cidr
  map_public_ip_on_launch = true
  tags = {
    Name = "PublicSubnet"
  }
}

# Private subnets (app servers)
resource "aws_subnet" "private" {
  count           = length(var.private_subnet_cidrs)
  vpc_id          = aws_vpc.main.id
  cidr_block     = var.private_subnet_cidrs[count.index]
}

^G Help      ^O Write Out   ^W Where Is   ^K Cut       ^T Execute   ^C Location
^X Exit      ^R Read File   ^V Replace    ^U Paste     ^J Justify   ^/ Go To Line
```

Figure 2.2: Network Module - VPC and Subnet Configuration

```

GNU nano 7.2                               variables.tf
variable "vpc_cidr" {
  description = "CIDR block for VPC"
  default     = "10.0.0.0/16"
}

variable "public_subnet_cidr" {
  description = "CIDR block for public subnet"
  default     = "10.0.1.0/24"
}

variable "private_subnet_cidrs" {
  description = "List of CIDRs for private subnets"
  type        = list(string)
  default     = ["10.0.2.0/24", "10.0.3.0/24"]
}

variable "allowed_ssh_ip" {
  description = "Your public IP for SSH access"
  default     = "4.240.18.230/32"
}

```

[ Read 20 lines ]

**GNU nano 7.2**

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location  
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

⊗ 0 △ 0 ↵ 0

```

GNU nano 7.2                               outputs.tf
output "vpc_id" {
  value = aws_vpc.main.id
}

output "public_subnet_id" {
  value = aws_subnet.public.id
}

output "private_subnet_ids" {
  value = aws_subnet.private[*].id
}

output "monitoring_sg_id" {
  value = aws_security_group.monitoring.id
}

output "app_sg_id" {
  value = aws_security_group.app.id
}

```

[ Read 19 lines ]

**GNU nano 7.2**

^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location  
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line

Figure 2.3: Network Module - Security Groups Configuration

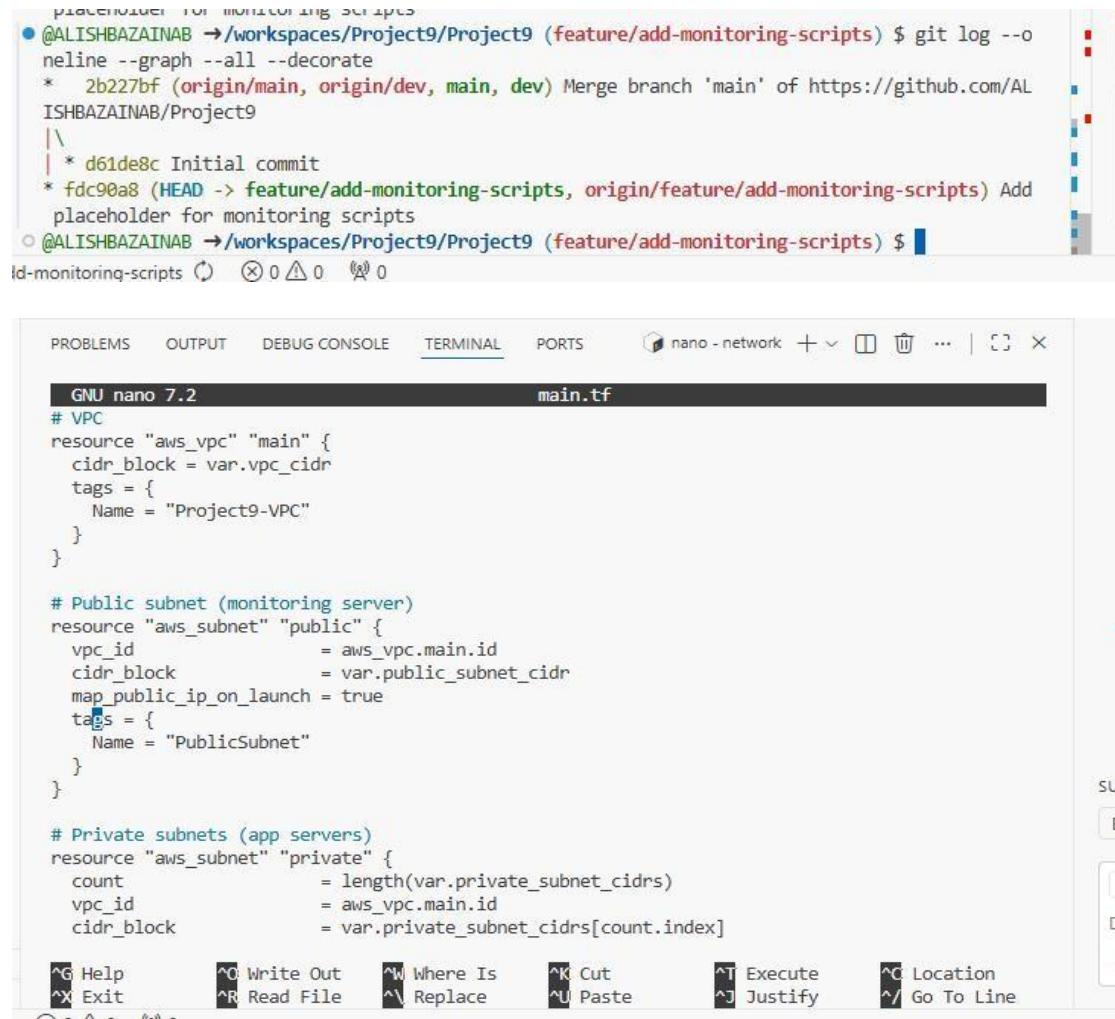
The network module provisions:

- VPC with CIDR block 10.0.0.0/16
- Public subnet for monitoring server
- Private subnet for application servers

- Security groups for monitoring and application servers

## 2.2 Monitoring Server Module

The monitoring server module provisions an EC2 instance in the public subnet with appropriate security group rules to allow HTTP access for the dashboard and SSH access for administration.



The terminal window shows the command `git log --oneline --graph --all --decorate` with the following output:

```

@ALISHBAZAINAB →/workspaces/Project9/Project9 (feature/add-monitoring-scripts) $ git log --oneline --graph --all --decorate
* 2b227bf (origin/main, origin/dev, main, dev) Merge branch 'main' of https://github.com/ALISHBAZAINAB/Project9
| \
| * d61de8c Initial commit
* fdc90a8 (HEAD → feature/add-monitoring-scripts, origin/feature/add-monitoring-scripts) Add placeholder for monitoring scripts
○ @ALISHBAZAINAB →/workspaces/Project9/Project9 (feature/add-monitoring-scripts) $

```

The code editor window shows the `main.tf` file with the following Terraform configuration:

```

# VPC
resource "aws_vpc" "main" {
  cidr_block = var.vpc_cidr
  tags = {
    Name = "Project9-VPC"
  }
}

# Public subnet (monitoring server)
resource "aws_subnet" "public" {
  vpc_id           = aws_vpc.main.id
  cidr_block       = var.public_subnet_cidr
  map_public_ip_on_launch = true
  tags = {
    Name = "PublicSubnet"
  }
}

# Private subnets (app servers)
resource "aws_subnet" "private" {
  count           = length(var.private_subnet_cidrs)
  vpc_id           = aws_vpc.main.id
  cidr_block       = var.private_subnet_cidrs[count.index]
}

```

*Figure 2.4: Monitoring Server Module - EC2 Instance Configuration*

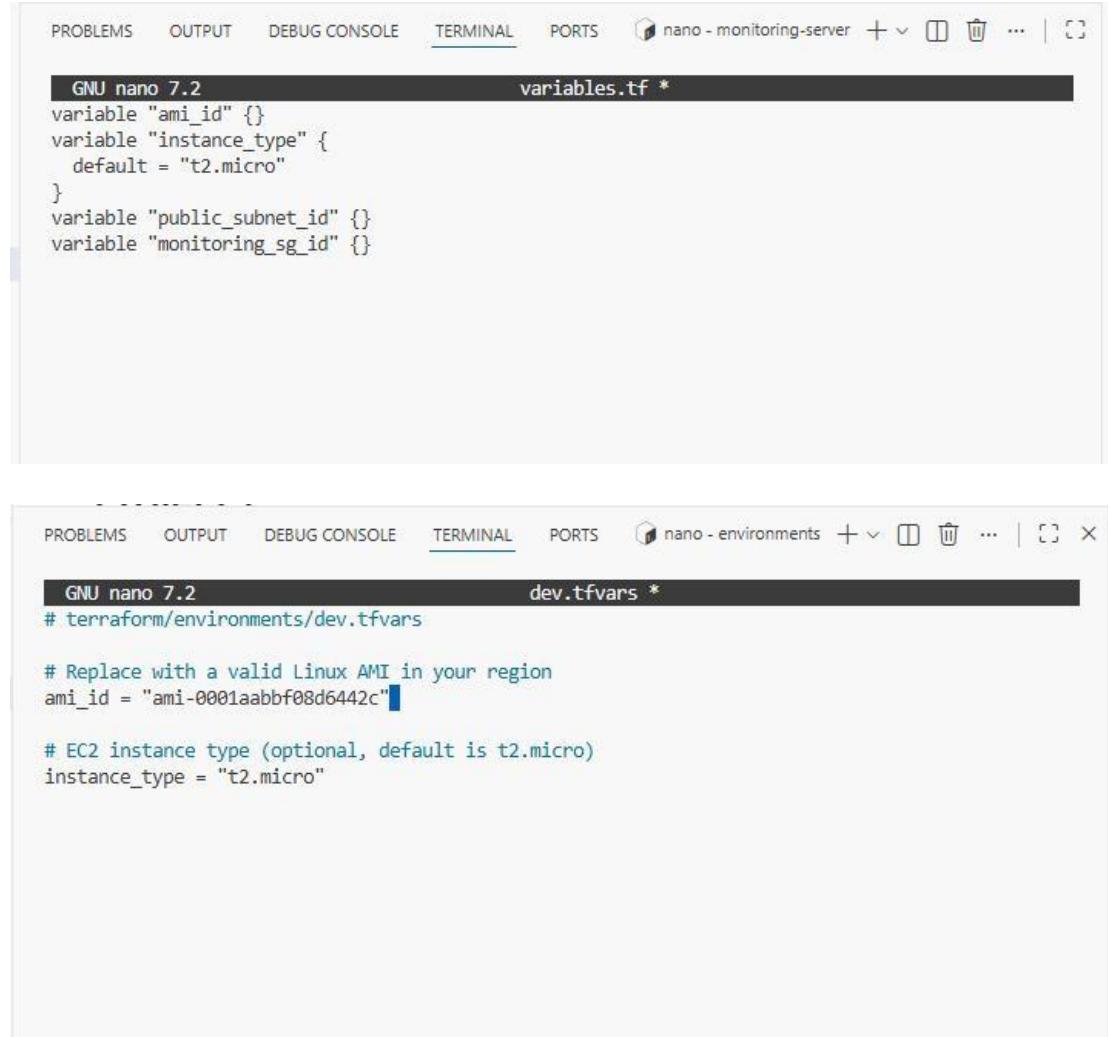
*Figure 2.5: Monitoring Server Module - Outputs and Variables*

The monitoring server is configured with:

- Public IP address for dashboard access

- Security group allowing HTTP (port 80) and SSH (port 22)
- Appropriate instance type and AMI
- Role tag: monitoring

## 2.3 Terraform Root Configuration



```
GNU nano 7.2                                     variables.tf *
variable "ami_id" {}
variable "instance_type" {
  default = "t2.micro"
}
variable "public_subnet_id" {}
variable "monitoring_sg_id" {}

GNU nano 7.2                                     dev.tfvars *
# terraform/environments/dev.tfvars

# Replace with a valid Linux AMI in your region
ami_id = "ami-0001aabbf08d6442c"

# EC2 instance type (optional, default is t2.micro)
instance_type = "t2.micro"
```

*Figure 2.6: Root Terraform Configuration - Provider and Module Calls*

The screenshot shows a code editor window with the title bar "nano - monitoring-server". The menu bar includes "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", "PORTS", and a "nano" icon. The tab bar shows "main.tf \*". The code in the editor is:

```
GNU nano 7.2                               main.tf *
resource "aws_instance" "monitoring" {
    ami           = var.ami_id
    instance_type = var.instance_type
    subnet_id     = var.public_subnet_id
    vpc_security_group_ids = [var.monitoring_sg_id]
    associate_public_ip_address = true
    tags = {
        Name = "Monitoring-Server"
        Role = "monitoring"
    }
}
```

Figure 2.7: Root Terraform Configuration - Network Module

The screenshot shows a code editor window with the title bar "nano - terraform". The menu bar includes "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", "PORTS", and a "nano" icon. The tab bar shows "variables.tf \*". The code in the editor is:

```
GNU nano 7.2                               variables.tf *
# terraform/variables.tf

# AMI for monitoring server
variable "ami_id" {}

# EC2 instance type
variable "instance_type" {
    default = "t2.micro"
}
```

Figure 2.8: Root Terraform Configuration - Variables



The screenshot shows a terminal window titled "GNU nano 7.2" with the file name "outputs.tf \*". The code in the file is as follows:

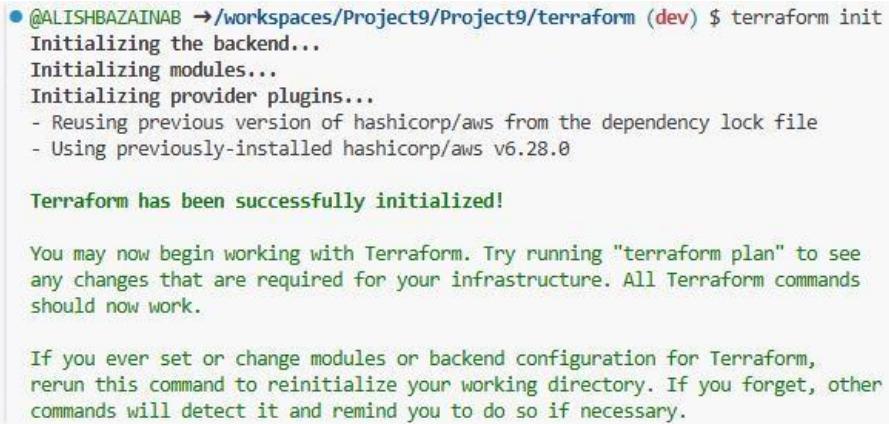
```
value = module.network.app_sg_id

# Monitoring server outputs
output "monitoring_public_ip" {
  value = module.monitoring_server.monitoring_public_ip
}

output "monitoring_public_dns" {
  value = module.monitoring_server.monitoring_public_dns
}
```

Figure 2.9: Root Terraform Configuration - Outputs

## Terraform Initialization and Application



```
@ALISHBAZAINAB →/workspaces/Project9/Project9/terraform (dev) $ terraform init
Initializing the backend...
Initializing modules...
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v6.28.0

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Figure 2.10: Terraform Initialization Success

The infrastructure was initialized using `terraform init`, which downloaded required providers and initialized the backend.

```

● @ALISHBAZAINAB →/workspaces/Project9/Project9/terraform (dev) $ terraform plan -var-file=environments/dev.tfvars
module.monitoring_server.data.aws_ami.ecs: Reading...
module.network.aws_vpc.main: Refreshing state... [id=vpc-003d09949e78b96e0]
module.monitoring_server.data.aws_ami.ecs: Read complete after 1s [id=ami-02db56e0f77a046f5]
module.network.aws_subnet.private[1]: Refreshing state... [id=subnet-0464d1c7fcba44686f]
module.network.aws_subnet.public: Refreshing state... [id=subnet-098f24858f27ad8a5]
module.network.aws_internet_gateway.igw: Refreshing state... [id=igw-0f5d3001914037ae9]
module.network.aws_subnet.private[0]: Refreshing state... [id=subnet-0c1b1fe46524616d0]
module.network.aws_security_group.monitoring: Refreshing state... [id=sg-044d9bc5c83c9220f]
module.network.aws_route_table.public: Refreshing state... [id=rtb-08987b62a48aea602]
module.network.aws_security_group.app: Refreshing state... [id=sg-07e0499391ee032f9]
module.network.aws_route_table_association.public_assoc: Refreshing state... [id=rtbassoc-0d4ddc3eda75887a3]

```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

```

# module.monitoring_server.aws_instance.monitoring will be created
+ resource "aws_instance" "monitoring" {
    ami
        = "ami-02db56e0f77a046f5"

```

*Figure 2.11: Terraform Plan Output Showing Resources to be Created*

```

● @ALISHBAZAINAB →/workspaces/Project9/Project9/terraform (dev) $ terraform apply -var-file=environments/dev.tfvars
module.monitoring_server.data.aws_ami.ecs: Reading...
module.network.aws_vpc.main: Refreshing state... [id=vpc-003d09949e78b96e0]
module.monitoring_server.data.aws_ami.ecs: Read complete after 0s [id=ami-02db56e0f77a046f5]
module.network.aws_subnet.public: Refreshing state... [id=subnet-098f24858f27ad8a5]
module.network.aws_internet_gateway.igw: Refreshing state... [id=igw-0f5d3001914037ae9]
module.network.aws_subnet.private[0]: Refreshing state... [id=subnet-0c1b1fe46524616d0]
module.network.aws_security_group.monitoring: Refreshing state... [id=sg-044d9bc5c83c9220f]
module.network.aws_subnet.private[1]: Refreshing state... [id=subnet-0464d1c7fcba44686f]
module.network.aws_route_table.public: Refreshing state... [id=rtb-08987b62a48aea602]
module.network.aws_route_table_association.public_assoc: Refreshing state... [id=rtbassoc-0d4ddc3eda75887a3]
module.network.aws_security_group.app: Refreshing state... [id=sg-07e0499391ee032f9]

```

Terraform used the selected providers to generate the following execution plan. Resource actions are indicated with the following symbols:

+ create

Terraform will perform the following actions:

*Figure 2.12: Terraform Apply - Infrastructure Provisioning Success*

```

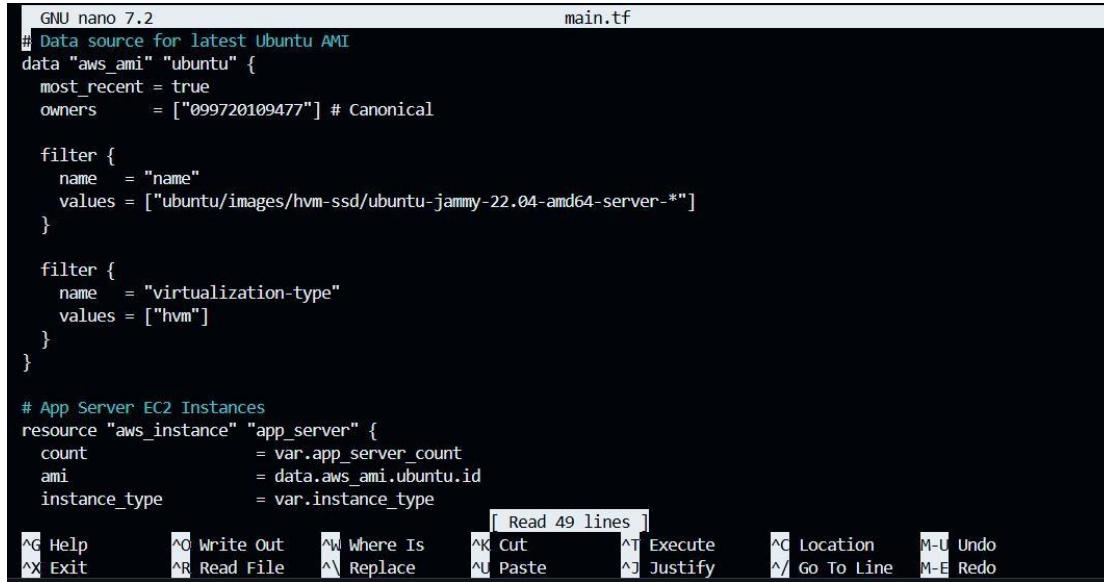
● @ALISHBAZAINAB →/workspaces/Project9/Project9/terraform (dev) $ terraform output
monitoring_public_dns = ""
monitoring_public_ip = "158.252.134.67"
monitoring_sg_id = "sg-044d9bc5c83c9220f"
public_subnet_id = "subnet-098f24858f27ad8a5"
○ @ALISHBAZAINAB →/workspaces/Project9/Project9/terraform (dev) $

```

*Figure 2.13: Terraform Output Values - Monitoring Server IP*

## 2.3 Application Servers Module

The application servers module provisions multiple EC2 instances in the private subnet. These servers are configured to only accept HTTP traffic from the monitoring server's security group.



```

GNU nano 7.2                               main.tf
# Data source for latest Ubuntu AMI
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical

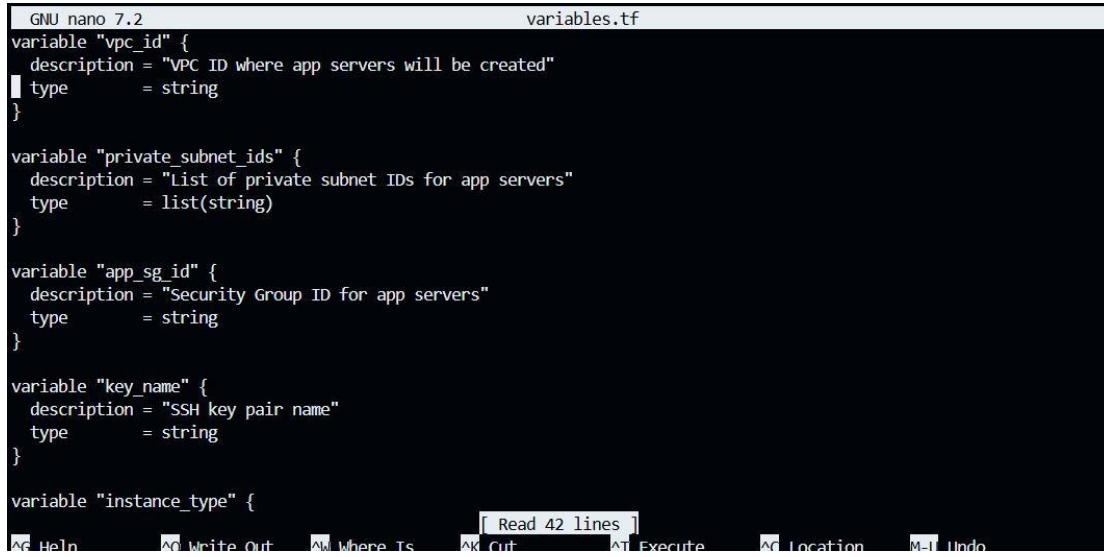
  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }

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

# App Server EC2 Instances
resource "aws_instance" "app_server" {
  count            = var.app_server_count
  ami              = data.aws_ami.ubuntu.id
  instance_type   = var.instance_type
}

```

Figure 2.14: Application Servers Module - EC2 Configuration



```

GNU nano 7.2                               variables.tf
variable "vpc_id" {
  description = "VPC ID where app servers will be created"
  type        = string
}

variable "private_subnet_ids" {
  description = "List of private subnet IDs for app servers"
  type        = list(string)
}

variable "app_sg_id" {
  description = "Security Group ID for app servers"
  type        = string
}

variable "key_name" {
  description = "SSH key pair name"
  type        = string
}

variable "instance_type" {
  description = "AWS instance type"
  type        = string
}

```

Figure 2.15: Application Servers Module - Variables

```

GNU nano 7.2                                         outputs.tf
output "app_server_ids" {
  description = "List of app server instance IDs"
  value       = aws_instance.app_server[*].id
}

output "app_server_private_ips" {
  description = "List of app server private IP addresses"
  value       = aws_instance.app_server[*].private_ip
}

output "app_server_details" {
  description = "Detailed information about app servers"
  value       = [
    for idx, instance in aws_instance.app_server : {
      id          = instance.id
      private_ip = instance.private_ip
      name        = instance.tags["Name"]
    }
  ]
}

```

*Figure 2.16: Application Servers Module - Outputs*

## Application Servers Root Configuration

```

# =====
# App Servers Module
# =====
module "app_servers" {
  source = "./modules/app-servers"

  vpc_id          = module.network.vpc_id
  private_subnet_ids = module.network.private_subnet_ids
  app_sg_id       = module.network.app_sg_id
  key_name        = var.key_name
  instance_type   = var.app_instance_type
  app_server_count = var.app_server_count
  environment     = var.environment
  project_name    = var.project_name
}

```

*Figure 2.17: Application Servers Root Configuration*

```

GNU nano 7.2                                         variables.tf
# =====
# App Servers Variables
# =====
variable "app_instance_type" {
  description = "Instance type for app servers"
  type       = string
  default    = "t2.micro"
}

variable "app_server_count" {
  description = "Number of app servers to create"
  type       = number
  default    = 2
}

```

*Figure 2.18: Application Servers Variables Configuration*

```

GNU nano 7.2                               outputs.tf
# App Servers Outputs
# =====
output "app_server_ids" {
  description = "App server instance IDs"
  value       = module.app_servers.app_server_ids
}

output "app_server_private_ips" {
  description = "App server private IPs"
  value       = module.app_servers.app_server_private_ips
}

output "app_server_details" {
  description = "Detailed app server information"
  value       = module.app_servers.app_server_details
}

```

*Figure 2.19: Application Servers Additional Configuration*

## Backend Configuration

```

# =====
# Terraform Backend Configuration
# =====
# This is a placeholder for remote state management
# For this project, we'll use local state

# Uncomment and configure for S3 backend:
# terraform {
#   backend "s3" {
#     bucket      = "your-terraform-state-bucket"
#     key         = "project9/terraform.tfstate"
#     region      = "me-central-1"
#     encrypt     = true
#     dynamodb_table = "terraform-state-lock"
# }

```

*Figure 2.20: Terraform Backend Configuration*

```

GNU nano 7.2                               dev.tfvars
# Development Environment Configuration
environment      = "dev"
aws_region       = "me-central-1"
project_name     = "project9"
key_name         = "project9-key"

# Network Configuration
vpc_cidr          = "10.0.0.0/16"
public_subnet_cidrs = ["10.0.1.0/24", "10.0.2.0/24"]
private_subnet_cidrs = ["10.0.3.0/24", "10.0.4.0/24"]
availability_zones = ["me-central-1a", "me-central-1b"]

# Security
allowed_ssh_cidr = "0.0.0.0/0"
allowed_http_cidr = "0.0.0.0/0"

```

*Figure 2.21: Backend Configuration-Development*

```

GNU nano 7.2                                     staging.tfvars
# Staging Environment Configuration
environment          = "staging"
aws_region           = "me-central-1"
project_name         = "project9"
key_name             = "project9-key"

# Network Configuration
vpc_cidr              = "10.1.0.0/16"
public_subnet_cidrs   = ["10.1.1.0/24", "10.1.2.0/24"]
private_subnet_cidrs  = ["10.1.3.0/24", "10.1.4.0/24"]
availability_zones    = ["me-central-1a", "me-central-1b"]

# Security
allowed_ssh_cidr     = "0.0.0.0/0"
allowed_http_cidr     = "0.0.0.0/0"

```

*Figure 2.22: Backend Configuration-Staging*

```

GNU nano 7.2                                     production.tfvars
# Production Environment Configuration
environment          = "production"
aws_region           = "me-central-1"
project_name         = "project9"
key_name             = "project9-key"

# Network Configuration
vpc_cidr              = "10.2.0.0/16"
public_subnet_cidrs   = ["10.2.1.0/24", "10.2.2.0/24"]
private_subnet_cidrs  = ["10.2.3.0/24", "10.2.4.0/24"]
availability_zones    = ["me-central-1a", "me-central-1b"]

# Security - CHANGE THIS IN PRODUCTION!
allowed_ssh_cidr     = "0.0.0.0/0"
allowed_http_cidr     = "0.0.0.0/0"

```

*Figure 2.23: Backend Configuration - Complete Setup*

## Application Servers Deployment

```

@lujainzia127 ~/workspaces/Labproject/terraform (dev) $ terraform init
- network in modules/network
Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v6.28.0...
- Installed hashicorp/aws v6.28.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.

```

*Figure 2.24: Application Servers - Terraform Initialization*

```

@lujainzia127 ➔ /workspaces/Labproject/terraform (dev) $ terraform plan -var-file=environments/dev.tvars -out=tfplan-dev
terraform apply tfplan-dev
{
  "id" = "i-002ea69337a5970d2"
  "name" = "project9-app-server-2-dev"
  "private_ip" = "10.0.4.187"
},
]
app_server_ids = [
  "i-02d164023b12e3999",
  "i-002ea69337a5970d2",
]
app_server_private_ips = [
  "10.0.3.137",
  "10.0.4.187",
]
app_sg_id = "sg-0bf63e6e5f693eeb2"
monitoring_public_dns = ""
monitoring_public_ip = ""
monitoring_sg_id = "sg-0496d2a591ffecaa5"
private_subnet_ids = [
  "subnet-027fc394ed9b8fad8",
  "subnet-0c0549b5c5a48104b",
]
public_subnet_id = "subnet-0214881861d2c9be8"
ssh_connection_info = {
  "monitoring_server" = "ssh -i ~/.ssh/project9-key ubuntu@"
  "note" = "App servers are in private subnets. SSH through monitoring server as bastion."
}
vpc_id = "vpc-052b7c65e2073be1e"
@lujainzia127 ➔ /workspaces/Labproject/terraform (dev) $ 

```

Figure 2.25: Application Servers - Provisioning Success

```

@lujainzia127 ➔ /workspaces/Labproject/terraform (dev) $ terraform output
{
  "id" = "i-002ea69337a5970d2"
  "name" = "project9-app-server-2-dev"
  "private_ip" = "10.0.4.187"
},
]
app_server_ids = [
  "i-02d164023b12e3999",
  "i-002ea69337a5970d2",
]
app_server_private_ips = [
  "10.0.3.137",
  "10.0.4.187",
]
app_sg_id = "sg-0bf63e6e5f693eeb2"
monitoring_public_dns = ""
monitoring_public_ip = ""
monitoring_sg_id = "sg-0496d2a591ffecaa5"
private_subnet_ids = [
  "subnet-027fc394ed9b8fad8",
  "subnet-0c0549b5c5a48104b",
]
public_subnet_id = "subnet-0214881861d2c9be8"
ssh_connection_info = {
  "monitoring_server" = "ssh -i ~/.ssh/project9-key ubuntu@"
  "note" = "App servers are in private subnets. SSH through monitoring server as bastion."
}
vpc_id = "vpc-052b7c65e2073be1e"
@lujainzia127 ➔ /workspaces/Labproject/terraform (dev) $ 

```

Figure 2.26: Application Servers - Output Showing Private IPs

# Part 3: Ansible Configuration

## 3.1 Configure Application Servers

Ansible roles were created to configure Nginx on all application servers. Each server is configured with a /health endpoint that returns HTTP 200, along with access and error logging.

### Nginx App Role Configuration

```
ansible > ansible.cfg
  1 [defaults]
  2   # Inventory
  3 inventory = ./inventory/dev_aws_ec2.yml
  4 host_key_checking = False
  5 remote_user = ec2-user
  6 private_key_file = ~/.ssh/project9-key
  7
  8   # Performance
  9 forks = 10
10 gathering = smart
11 fact_caching = jsonfile
12 fact_caching_connection = /tmp/ansible_facts
13 fact_caching_timeout = 3600
14
15   # Output
```

Figure 3.1: Nginx App Role - Installation and Configuration Tasks

```
GNU nano 7.2                                     dev_aws_ec2.yml
plugin: aws_ec2
regions:
- me-central-1
filters:
  tag:Environment: dev
  instance-state-name: running
keyed_groups:
- key: tags.Role
  prefix: role
- key: tags.Environment
  prefix: env
hostnames:
- private-ip-address
```

Figure 3.2: Nginx App Role - Service Management Tasks

```
GNU nano 7.2                                     main.yml
- name: Update apt cache
  apt:
    update_cache: yes
    cache_valid_time: 3600
- name: Install Nginx
  apt:
    name: nginx
    state: present
- name: Create health check page
  copy:
    dest: /var/www/html/health
    content: |
      OK
      mode: '0644'
- name: Create server identification page
  template:
```

*Figure 3.3: Nginx Configuration Template with Health Endpoint*

```
ansible > roles > dashboard > handlers > ! main.yml
1   - name: Restart Nginx
2     systemd:
3       name: nginx
4       state: restarted
5
```

*Figure 3.4: Nginx App Role - Handlers for Service Restart*

This role installs and configures Nginx on application servers with:

- /health endpoint returning HTTP 200
- Custom index page identifying the server
- Proper logging configuration
- Service management handlers

### 3.2 Configure Monitoring Server (10 marks)

The monitoring server was configured with multiple monitoring scripts and scheduled cron jobs to automatically collect metrics, check service health, and generate reports.

#### Dashboard Role Configuration

```
ansible > roles > dashboard > handlers > ! main.yml
1   - name: Restart Nginx
2     systemd:
3       name: nginx
4       state: restarted
5
```

*Figure 3.5: Dashboard Role - Nginx Handlers*

```

ansible > roles > dashboard > templates > nginx-dashboard.conf.j2
  1  nginx
  2    server {
  3      listen 80 default_server;
  4      listen [::]:80 default_server;
  5
  6      root /var/www/html;
  7      index index.html;
  8
  9      server_name _;
 10
 11      access_log /var/log/nginx/dashboard-access.log;
 12      error_log /var/log/nginx/dashboard-error.log;
 13
 14      location / {
 15        try_files $uri $uri/ =404;

```

Figure 3.6: Dashboard Nginx Configuration for Serving Dashboard

```

ansible > roles > dashboard > tasks > ! main.yml
  22  - name: Ensure Nginx is started
  23    systemd:
  24      name: nginx
  25      state: started
  26      enabled: yes
  27
  28  - name: Create symlinks for reports and logs
  29    file:
  30      src: "{{ item.src }}"
  31      dest: "{{ item.dest }}"
  32      state: link
  33      force: yes
  34  loop:
  35    - { src: '/var/monitoring/reports', dest: '/var/www/html/reports' }
  36    - { src: '/var/monitoring/logs', dest: '/var/www/html/logs' }

```

Figure 3.7: Dashboard Role - Setup Tasks

## Monitoring Tools Role

```

ansible > roles > monitoring-tools > files > build-dashboard.sh
  259 echo "$HEALTH_DATA" | jq -r '.servers[] | @json' | while read -r server; do
  260   if [ "$STATUS" = "UP" ]; then
  261     CLASS="up"
  262     BADGE_CLASS="up"
  263   else
  264     CLASS="down"
  265     BADGE_CLASS="down"
  266   fi
  267
  268   SERVERS_HTML+="

$SERVER_IP<span class=\"status-badge $BADGE_CLASS\">$STATUS</span>

"
  269 done
  270
  271 sed -i "s|${SERVERS_PLACEHOLDER}|$SERVERS_HTML|g" "$DASHBOARD_FILE"
  272
  273 echo "Dashboard updated at $TIMESTAMP"

```



Figure 3.8: Build Dashboard Script - Generates HTML Dashboard

```

ansible > roles > monitoring-tools > files > $ generate-report.sh
54   fi
55
56 # Add summary
57 cat >> "$REPORT_FILE" <<EOF
58
59 SUMMARY
60 -----
61 Report Type: $REPORT_TYPE
62 Total App Servers: $(jq '.servers | length' "$DATA_DIR/health-latest.json" 2>/dev/null || echo "0")
63 Healthy Servers: $(jq '[.servers[] | select(.status=="UP")] | length' "$DATA_DIR/health-latest.json" 2>/dev/null || echo "0")
64 -----
65 EOF
66
67 echo "Report generated to $REPORT_FILE"

```

*Figure 3.9: Generate Report Script - Creates Daily/Weekly Reports*

```

ansible > roles > monitoring-tools > files > $ http-health-check.sh
17 while IFS= read -r server; do
18   if [ "$RESPONSE" = "200" ]; then
19     STATUS="UP"
20   else
21     STATUS="DOWN"
22   fi
23
24   # Get response time
25   RESPONSE_TIME=$(curl -s -o /dev/null -w "%{time_total}" --connect-timeout 5 "http://$server/health")
26
27   cat >> "$HEALTH_FILE" <> "$HEALTH_FILE"
28   echo "]" >> "$HEALTH_FILE"
29   echo "}" >> "$HEALTH_FILE"
30
31 done
32
33 EOF

```

*Figure 3.10: HTTP Health Check Script - Checks /health Endpoints*

```

ansible > roles > monitoring-tools > files > $ check-services.sh
18
19 # Check process count
20 PROCESS_COUNT=$(ps aux | wc -l)
21
22 # Create JSON output
23 cat > "$SERVICES_FILE" <<EOF
24 {
25   "timestamp": "$TIMESTAMP",
26   "nginx_status": "$NGINX_STATUS",
27   "load_average": $LOAD_AVG,
28   "process_count": $PROCESS_COUNT
29 }
30 EOF
31
32 echo "Service check completed at $TIMESTAMP"

```

*Figure 3.11: Check Services Script - Monitors Service Status*

```

ansible > roles > monitoring-tools > files > $ collect-metrics.sh
1 #!/bin/bash
2
3 # Monitoring script to collect system metrics
4 TIMESTAMP=$(date '+%Y-%m-%d %H:%M:%S')
5 DATA_DIR="/var/monitoring/data"
6 METRICS_FILE="$DATA_DIR/metrics-latest.json"
7
8 # Create data directory if not exists
9 mkdir -p "$DATA_DIR"
10
11 # Collect CPU usage
12 CPU_USAGE=$(top -bn1 | grep "cpu(s)" | sed "s/.*/ *\\([0-9.]*)\\%* id.*\\1/\" | awk '{print 100 - $1}'")
13
14 # Collect memory usage
15 TOTAL_MEM=$(free -m | awk 'NR==2{print $2}')
16 USED_MEM=$(free -m | awk 'NR==2{print $3}')

```

Figure 3.12: Collect Metrics Script - Gathers System Metrics

```

ansible > roles > monitoring-tools > tasks > ! main.yml
1 - name: Install required packages
2   apt:
3     name:
4       - curl
5       - jq
6       - bc
7       - sysstat
8     state: present
9     update_cache: yes
10
11 - name: Create monitoring directories
12   file:
13     path: "{{ item }}"
14     state: directory
15     mode: '0755'

```

Figure 3.13: Monitoring Tools Role - Installation and Cron Setup

```

ansible > roles > nginx-app > handlers > ! main.yml
1 - name: Restart Nginx
2   systemd:
3     name: nginx
4     state: restarted
5

```

Figure 3.14: Nginx App Role - Handler Configuration

```
ansible > roles > nginx-app > templates > index.html.j2
 1 | App Server - {{ inventory_hostname }}
 2 |
 3 |     body {
 4 |         font-family: Arial, sans-serif;
 5 |         max-width: 800px;
 6 |         margin: 50px auto;
 7 |         padding: 20px;
 8 |         background: #f5f5f5;
 9 |     }
10 |     .container {
11 |         background: white;
12 |         padding: 30px;
13 |         border-radius: 8px;
14 |         box-shadow: 0 2px 4px rgba(0,0,0,0.1);
15 |     }
16 |     h1 { color: #333; }
```

Figure 3.15: Nginx App Role - Index Page Template

## Ansible Playbooks

```
ansible > playbooks > ! configure-app-servers.yml
 1 | - name: Configure Application Servers
 2 |   hosts: role_app
 3 |   become: yes
 4 |
 5 |   tasks:
 6 |     - name: Include nginx-app role
 7 |       include_role:
 8 |         name: nginx-app
```

Figure 3.16: Configure App Servers Playbook

```
ansible > playbooks > ! configure-monitoring-server.yml
 5 | vars:
 6 |   app_servers: "{{ groups['role_app'] | default([]) }}"
 7 |
 8 | tasks:
 9 |   - name: Include monitoring-tools role
10 |     include_role:
11 |       name: monitoring-tools
12 |
13 |   - name: Include dashboard role
14 |     include_role:
15 |       name: dashboard
```

Figure 3.17: Configure Monitoring Server Playbook

```

ansible > playbooks > ! collect-logs.yml
  1   - name: Collect Logs from App Servers
  5     tasks:
  6       - name: Create local logs directory
  7         delegate_to: localhost
  8         become: no
  9
 11      - name: Fetch Nginx access logs
 12        fetch:
 13          src: /var/log/nginx/access.log
 14          dest: "./collected-logs/{{ inventory_hostname }}/{{ ansible_date_time.date }}/access.log"
 15          flat: yes
 16
 17      - name: Fetch Nginx error logs
 18        fetch:
 19          src: /var/log/nginx/error.log
 20          dest: "./collected-logs/{{ inventory_hostname }}/{{ ansible_date_time.date }}/error.log"
 21
 22
 23

```

Figure 3.18: Collect Logs Playbook - Uses Ansible Fetch Module

```

ansible > playbooks > ! generate-report.yml
  2   hosts: role_monitoring
  3   become: yes
  4
  5   tasks:
  6     - name: Generate daily report
  7       command: /usr/local/bin/monitoring/generate-report.sh daily
  8       when: report_type == "daily" or report_type is not defined
  9
 10    - name: Generate weekly report
 11      command: /usr/local/bin/monitoring/generate-report.sh weekly
 12      when: report_type == "weekly"

```

Figure 3.19: Generate Report Playbook - Triggers Report Generation

## Playbook Execution and Verification

```

● (venv) @lujainzia127 →/workspaces/Labproject (dev) $ cd ansible
● (venv) @lujainzia127 →/workspaces/Labproject/ansible (dev) $ ansible all -i inventory/dev_aws_ec2.yml -m ping
[WARNING]: Collection amazon.aws does not support Ansible version 2.14.0
10.0.1.48 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.7"
    },
    "changed": false,
    "ping": "pong"
}
10.0.4.190 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.7"
    },
    "changed": false,
    "ping": "pong"
}
10.0.3.154 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3.7"
    },
    "changed": false,
    "ping": "pong"
}
● (venv) @lujainzia127 →/workspaces/Labproject/ansible (dev) $ █

```

Figure 3.20: Ansible Connectivity Test - All Hosts Reachable

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
(venv) @lujainzia127 →/workspaces/Labproject/ansible (dev) $ ansible-playbook -i inventory/dev_aws_ec2.yml playbooks/configure-app-servers.yml

TASK [Display Nginx status] *****
ok: [10.0.4.190] => {
    "msg": "Nginx is active"
}
ok: [10.0.3.154] => {
    "msg": "Nginx is active"
}

TASK [Test health endpoint locally] *****
ok: [10.0.4.190]
ok: [10.0.3.154]

TASK [Display health check result] *****
ok: [10.0.4.190] => {
    "msg": "Health check response: 200"
}
ok: [10.0.3.154] => {
    "msg": "Health check response: 200"
}

PLAY RECAP *****
10.0.3.154      : ok=12   changed=6    unreachable=0   failed=0    skipped=0   rescued=0   ignored=0
10.0.4.190      : ok=12   changed=6    unreachable=0   failed=0    skipped=0   rescued=0   ignored=0

```

Figure 3.21: Configure App Servers Playbook Execution - Success

```

[ec2-user@ip-10-0-1-48 ~]$ curl http://10.0.3.154/health
<!DOCTYPE html>
<html>
<head><title>Health Check</title></head>
<body>
    <h1>OK</h1>
    <p>Server: ip-10-0-3-154</p>
    <p>IP: 10.0.3.154</p>
    <p>Status: Running</p>
</body>
</html>
[ec2-user@ip-10-0-1-48 ~]$ echo ""

[ec2-user@ip-10-0-1-48 ~]$ curl http://10.0.4.190/health
<!DOCTYPE html>
<html>
<head><title>Health Check</title></head>
<body>
    <h1>OK</h1>
    <p>Server: ip-10-0-4-190</p>
    <p>IP: 10.0.4.190</p>
    <p>Status: Running</p>
</body>
</html>
[ec2-user@ip-10-0-1-48 ~]$ 

```

Figure 3.22: Curl Test Showing /health Endpoint Working on All Servers

### 3.3 Log Collection Using Ansible

```
(venv) @lujainzia127 ~/workspaces/Labproject/ansible (dev) $ ansible-playbook -i inventory/dev_aws_ec2.yml playbooks/configure-monitoring-server.yml
changed: [10.0.1.48] => (item={'name': 'Generate weekly report', 'minute': '0', 'hour': '0', 'job': '[ $(date +%%u) -eq 1 ] && /usr/local/bin/monitoring/generate-report.sh weekly >> /var/monitoring/logs/reports.log 2>&1'})
TASK [Display app server IPs] ****
ok: [10.0.1.48] => {
    "msg": "Monitoring these app servers: ['10.0.4.190', '10.0.3.154']"
}

TASK [Verify cron jobs] ****
ok: [10.0.1.48]

TASK [Display cron jobs] ****
ok: [10.0.1.48] => {
    "msg": [
        "#Ansible: Collect metrics and health checks",
        "* * * * * /usr/local/bin/monitoring/collect-metrics.sh >> /var/monitoring/logs/metrics.log 2>&1 && /usr/local/bin/monitoring/check-services.sh >> /var/monitoring/logs/services.log 2>&1 && /usr/local/bin/monitoring/http-health-check.sh >> /var/monitoring/logs/health.log 2>&1 && /usr/local/bin/monitoring/build-dashboard.sh >> /var/monitoring/logs/dashboard.log 2>&1",
        "#Ansible: Generate daily report",
        "#0 0 * * * /usr/local/bin/monitoring/generate-report.sh daily >> /var/monitoring/logs/reports.log 2>&1",
        "#Ansible: Generate weekly report",
        "#0 0 * * * [ $(date +%%u) -eq 1 ] && /usr/local/bin/monitoring/generate-report.sh weekly >> /var/monitoring/logs/reports.log 2>&1"
    ]
}

PLAY RECAP ****
10.0.1.48 : ok=10    changed=2     unreachable=0    failed=0     skipped=0    rescued=0    ignored=0
```

Figure 3.23: Monitoring Server Configuration Playbook Execution

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS bash - ansible + ×
(venv) @lujainzia127 ~/workspaces/Labproject/ansible (dev) $ ansible-playbook -i inventory/dev_aws_ec2.yml playbooks/collect-logs.yml
PLAY [Collect Logs from App Servers] ****
TASK [Create local log collection directory] ****
changed: [10.0.4.190 -> localhost]
changed: [10.0.3.154 -> localhost]

TASK [Fetch Nginx access logs] ****
changed: [10.0.4.190]
changed: [10.0.3.154]

TASK [Fetch Nginx error logs] ****
changed: [10.0.4.190]
changed: [10.0.3.154]

TASK [Create log summary] ****
changed: [10.0.4.190 -> localhost]
changed: [10.0.3.154 -> localhost]

PLAY RECAP ****
10.0.3.154      : ok=4    changed=4     unreachable=0    failed=0     skipped=0    rescued=0    ignored=0
10.0.4.190      : ok=4    changed=4     unreachable=0    failed=0     skipped=0    rescued=0    ignored=0
```

Figure 3.24: Log Collection Playbook - Fetching Logs from App Servers

```
(venv) @lujainzia127 →/workspaces/Labproject/ansible (dev) $ ansible-playbook -i inventory/dev_aws_ec2.yml playbooks/configure-monitoring-server.yml

# Generate reports
ansible-playbook -i inventory/dev_aws_ec2.yml playbooks/generate-report.yml
    "Daily report: ['Report generated: /var/monitoring/reports/daily-2026-01-26.txt']",
    "Weekly report: ['Report generated: /var/monitoring/reports/weekly-2026-W05.txt']"
}

TASK [List generated reports] *****
changed: [10.0.1.48]

TASK [Display report files] *****
ok: [10.0.1.48] => {
    "report_list.stdout_lines": [
        "total 8.0K",
        "-rw-r--r-- 1 root root 550 Jan 26 10:01 daily-2026-01-26.txt",
        "-rw-r--r-- 1 root root 552 Jan 26 10:01 weekly-2026-W05.txt"
    ]
}

PLAY RECAP *****
10.0.1.48 : ok=5    changed=3    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

*Figure 3.25: Report Generation Playbook - Creating Daily/Weekly Reports*

## Part 4: Dashboard & Reporting

### 4.1 Simple Web Dashboard

A web-based dashboard was created to display real-time system status, server health, collected metrics, and links to reports. The dashboard is served by Nginx and updated automatically by the build-dashboard.sh script.

```
(venv) @lujainzia127 →/workspaces/Labproject/ansible (dev) $ ansible-playbook -i inventory/dev_aws_ec2.yml playbooks/configure-monitoring-server.yml
TASK [dashboard : Create symbolic links to monitoring data] *****
changed: [10.0.1.48]

TASK [dashboard : Configure Nginx for monitoring server] *****
changed: [10.0.1.48]

TASK [dashboard : Remove default Nginx config] *****
ok: [10.0.1.48]

RUNNING HANDLER [dashboard : Restart Nginx] *****
changed: [10.0.1.48]

TASK [Verify cron jobs] *****
ok: [10.0.1.48]

TASK [Display monitoring server public IP] *****
ok: [10.0.1.48] => {
    "msg": "Dashboard URL: http://10.0.1.48"
}

PLAY RECAP *****
10.0.1.48 : ok=17    changed=7    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

*Figure 4.1: Cron Jobs for Automated Monitoring and Dashboard Updates*

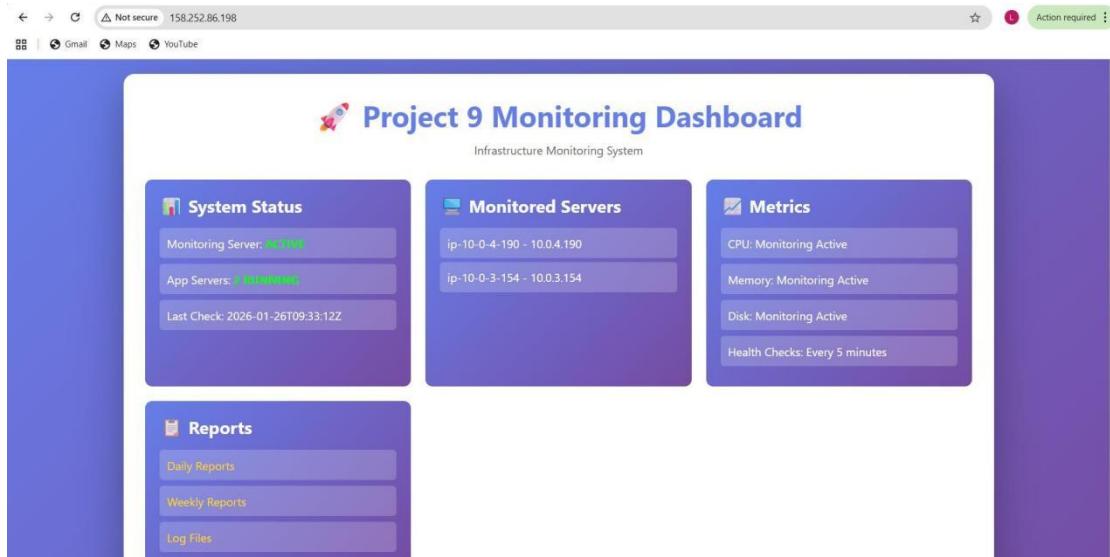


Figure 4.2: Monitoring Dashboard - Header and Server Status

The screenshot shows a list of reports on the dashboard. Two files are listed:

- [daily-2026-01-26.txt](#) (26-Jan-2026 10:01, 550 bytes)
- [weekly-2026-W05.txt](#) (26-Jan-2026 10:01, 552 bytes)

Figure 4.3: Monitoring Dashboard - Links to Weekly and Daily Reports

<a href="#">..</a>		
<a href="#">health.log</a>	26-Jan-2026 10:10	658
<a href="#">metrics.log</a>	26-Jan-2026 10:10	287
<a href="#">services.log</a>	26-Jan-2026 10:10	329

Figure 4.4: Dashboard - Links to Log Files

<a href="#">..</a>		
<a href="#">health-latest.json</a>	26-Jan-2026 10:10	55
<a href="#">metrics-20260126-094001.json</a>	26-Jan-2026 09:40	224
<a href="#">metrics-20260126-094501.json</a>	26-Jan-2026 09:45	222
<a href="#">metrics-20260126-095002.json</a>	26-Jan-2026 09:50	228
<a href="#">metrics-20260126-095501.json</a>	26-Jan-2026 09:55	224
<a href="#">metrics-20260126-100001.json</a>	26-Jan-2026 10:00	224
<a href="#">metrics-20260126-100502.json</a>	26-Jan-2026 10:05	226
<a href="#">metrics-20260126-101001.json</a>	26-Jan-2026 10:10	227
<a href="#">metrics-latest.json</a>	26-Jan-2026 10:10	227
<a href="#">services-latest.json</a>	26-Jan-2026 10:10	118

Figure 4.5: Dashboard - Links to Reports and Collected Logs

The dashboard displays:

- Real-time server status (UP/DOWN) for all application servers
- System metrics (CPU, Memory, Disk usage)
- Last health check timestamp

## 4.2 Automated Reporting (7 marks)

Daily and weekly reports are automatically generated via cron jobs. These reports contain system metrics, health check results, service status, and log summaries. Reports are stored in /var/www/html/reports/ and are accessible through the dashboard.

The screenshot shows a web browser window with the URL `3.29.58.22/reports/daily-2026-01-27.txt`. The page content is a text-based monitoring report:

```
=====
daily MONITORING REPORT
Generated: 2026-01-27 14:33:21
=====

SYSTEM METRICS (Latest)
-----
CPU Usage: 0%
Memory Usage: 9.79%
Disk Usage: 22%

SERVICE STATUS
-----
Nginx: running
Load Average: 0

APP SERVERS HEALTH
-----
Server: 10.0.3.132 - Status: UP - Response: nulls
Server: 10.0.4.19 - Status: UP - Response: nulls

SUMMARY
-----
Report Type: daily
Total App Servers: 2
Healthy Servers: 2
=====
```

Figure 4.6 :Report -Daily Monitoring

The screenshot shows a web browser window with the URL `3.29.58.22/reports/weekly-2026-W05.txt`. The page content is a text-based monitoring report:

```
=====
weekly MONITORING REPORT
Generated: 2026-01-27 14:33:22
=====

SYSTEM METRICS (Latest)
-----
CPU Usage: 0%
Memory Usage: 9.79%
Disk Usage: 22%

SERVICE STATUS
-----
Nginx: running
Load Average: 0

APP SERVERS HEALTH
-----
Server: 10.0.3.132 - Status: UP - Response: nulls
Server: 10.0.4.19 - Status: UP - Response: nulls

SUMMARY
-----
Report Type: weekly
Total App Servers: 2
Healthy Servers: 2
=====
```

Figure 4.7 : Report -Weekly Monitoring

## Merging Branch

```
● (venv) @lujainzia127 →/workspaces/Labproject (main) $ git add .
git commit -m "Resolved merge conflicts"
git push origin main
On branch main
Your branch is ahead of 'origin/main' by 18 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
Enumerating objects: 21, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 2 threads
Compressing objects: 100% (10/10), done.
Writing objects: 100% (11/11), 1.67 KiB | 570.00 KiB/s, done.
Total 11 (delta 8), reused 2 (delta 1), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (8/8), completed with 7 local objects.
remote: This repository moved. Please use the new location:
remote:   https://github.com/lujainzia127/Project9.git
To https://github.com/lujainzia127/Labproject
  2b227fb..42cdb3ef main -> main
* [new branch]      main        -> 2b227fb..42cdb3ef
(venv) @lujainzia127 →/workspaces/Labproject (main) $ █
```

Figure 5 : Git Branch Merge History

## Part 5 : Documentation

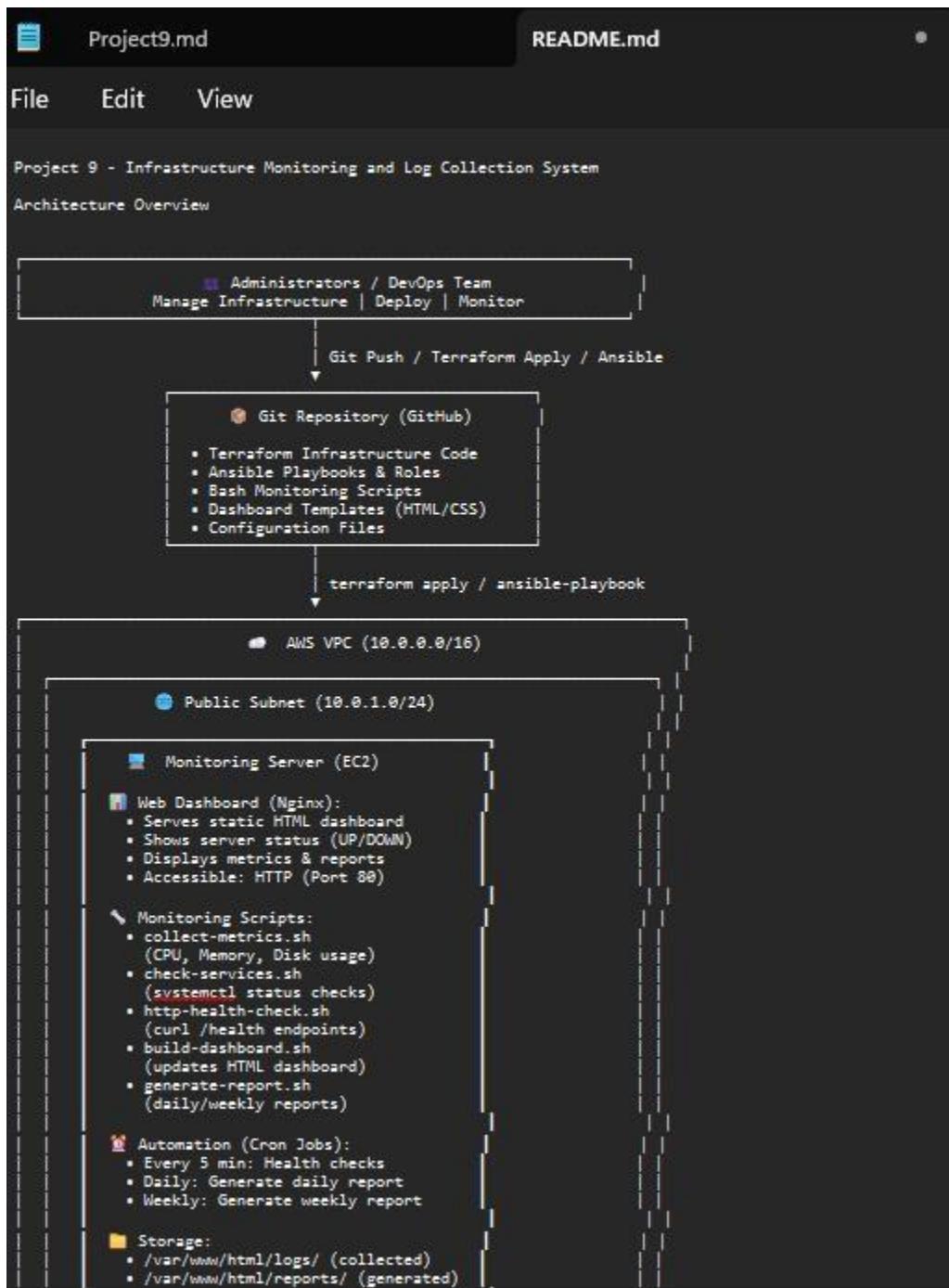


Figure 5.1: README.md - Project Overview and Architecture



Figure 5.2: README.md - Setup Instructions and Configuration Guide

```

4. Log Collection (Ansible Playbook):**
- Runs periodically using 'collect-logs.yml' playbook
- Uses 'ansible.builtin.fetch' module to retrieve logs
- Collects '/var/log/nginx/access.log' and '/var/log/nginx/error.log'
- Stores on monitoring server: '/var/www/html/logs/{hostname}/{date}/'
- Organizes by hostname and timestamp

5. Report Generation (Cron)**
- Daily Report: 'generate-report.sh daily'
- Saved to: '/var/www/html/reports/daily-YYYY-MM-DD.txt'
- Contains: server status, uptime, health check results, resource usage
- Weekly Report: 'generate-report.sh weekly'
- Saved to: '/var/www/html/reports/weekly-YYYY-MM.txt'
- Contains: weekly summary, trends, incidents, log analysis

6. Dashboard Access:
- Administrators access via: 'http://<monitoring-server-public-ip>'
- Dashboard displays:
  - Real-time server status (UP/DOWN indicators)
  - Latest metrics (CPU, memory, disk usage)
  - Links to collected logs
  - Links to daily/weekly reports
  - Historical data visualization

## 🔒 Network Security
| Component | Access Rule | Description |
|-----|-----|-----|
**Internet Gateway** | Public access | Routes traffic from internet to public subnet |
**Monitoring Server SG** | HTTP (80) from anywhere<br>SSH (22) from admin IPs | Dashboard accessible via browser<br>Secure admin access |
**App Server SG** | HTTP (80) from Monitoring SG only<br>SSH (22) from admin IPs | Only monitoring server can check health<br>Secure admin access |
**SSH Keys** | Key-based authentication | No password authentication allowed |

## 📈 Monitoring Metrics Collected
| Metric | Command/Script | Frequency |
|-----|-----|-----|
**CPU Usage** | `top` , `mpstat` | Every 5 minutes |
**Memory Usage** | `free -m` , `/proc/meminfo` | Every 5 minutes |
**Disk Usage** | `df -h` | Every 5 minutes |
**Service Status** | `systemctl is-active nginx` | Every 5 minutes |
**HTTP Health** | `curl http://server/health` | Every 5 minutes |
**Nginx Logs** | Ansible fetch | Hourly |

## 🌐 Key Features
- [x] **Automated Monitoring:** Cron-based health checks every 5 minutes
- [x] **Centralized Dashboard:** Real-time status via Nginx-served HTML
- [x] **Log Aggregation:** Ansible collects logs from all app servers
- [x] **Automated Reporting:** Daily and weekly reports generated automatically
- [x] **Infrastructure as Code:** Complete Terraform + Ansible automation
- [x] **Security:** Isolated subnets, security groups, key-based SSH
- [x] **Scalability:** Easy to add more app servers to monitor

```

*Figure 5.3: README.md - Usage Instructions and Troubleshooting*

The README.md includes:

- Project overview and objectives
- System architecture diagram
- Prerequisites and setup instructions
- Terraform provisioning steps
- Ansible configuration procedures
- Dashboard access instructions
- Troubleshooting guide

## System Architecture

The system architecture consists of:

- **Monitoring Server (Public Subnet)**
  - Nginx serving web dashboard
  - Monitoring scripts (bash)
  - Cron jobs for automated checks
  - Log storage from application servers
  - Report generation
- **Application Servers (Private Subnet)**

- Nginx web servers
- /health endpoint for monitoring
- Access and error logs
- Sample application

- **Network Infrastructure**

- VPC with public and private subnets
- Security groups for access control
- Internet Gateway for monitoring server

- **Monitoring Flow**

- Daily: Generate daily report
- Weekly: Generate weekly report

## GitHub Repository

Repository Link: <https://github.com/lujainzia127/Project9>