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

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:

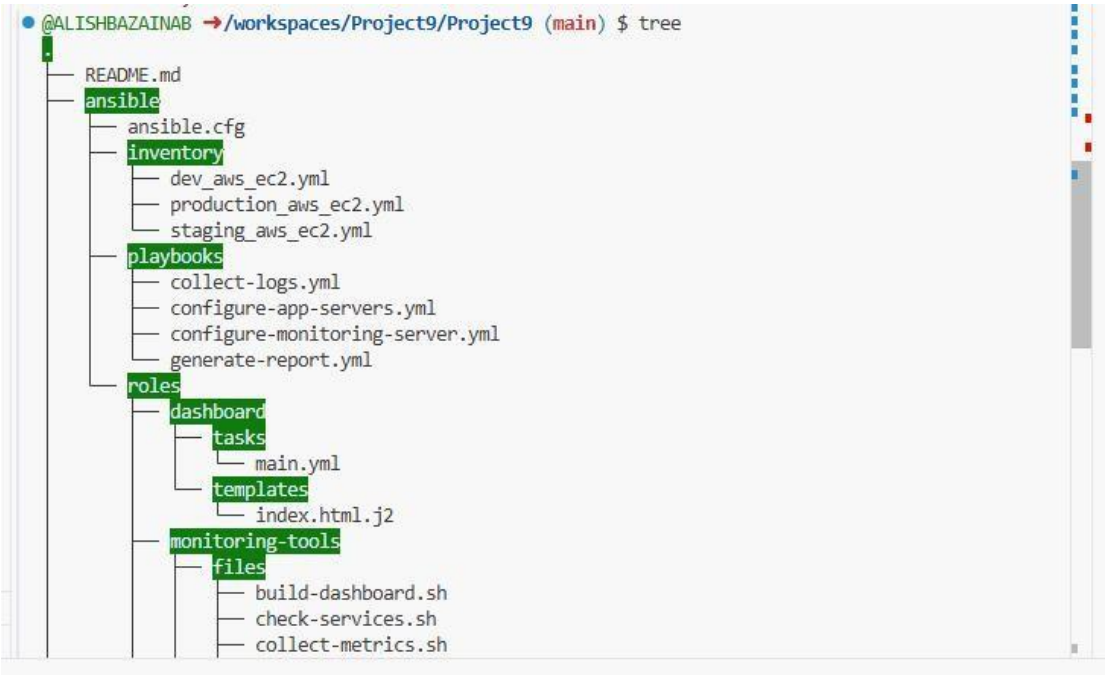


Figure 1.1: Project Repository Structure - Root Level

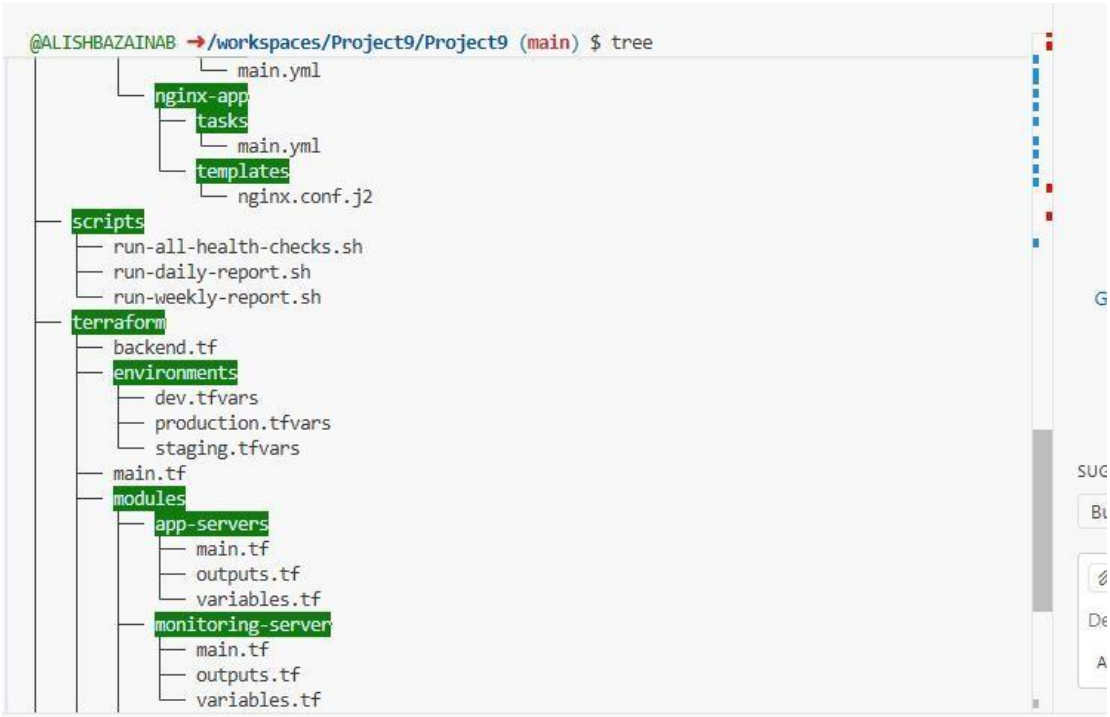


Figure 1.2: Project Repository Structure - Terraform Modules

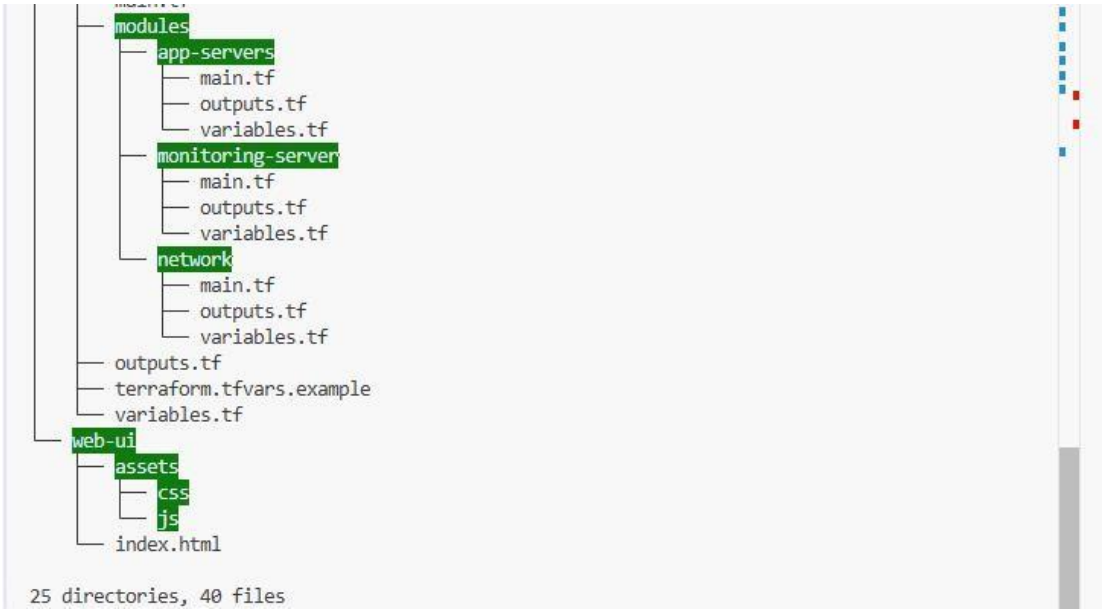


Figure 1.3: Project Repository Structure - Ansible Roles

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.

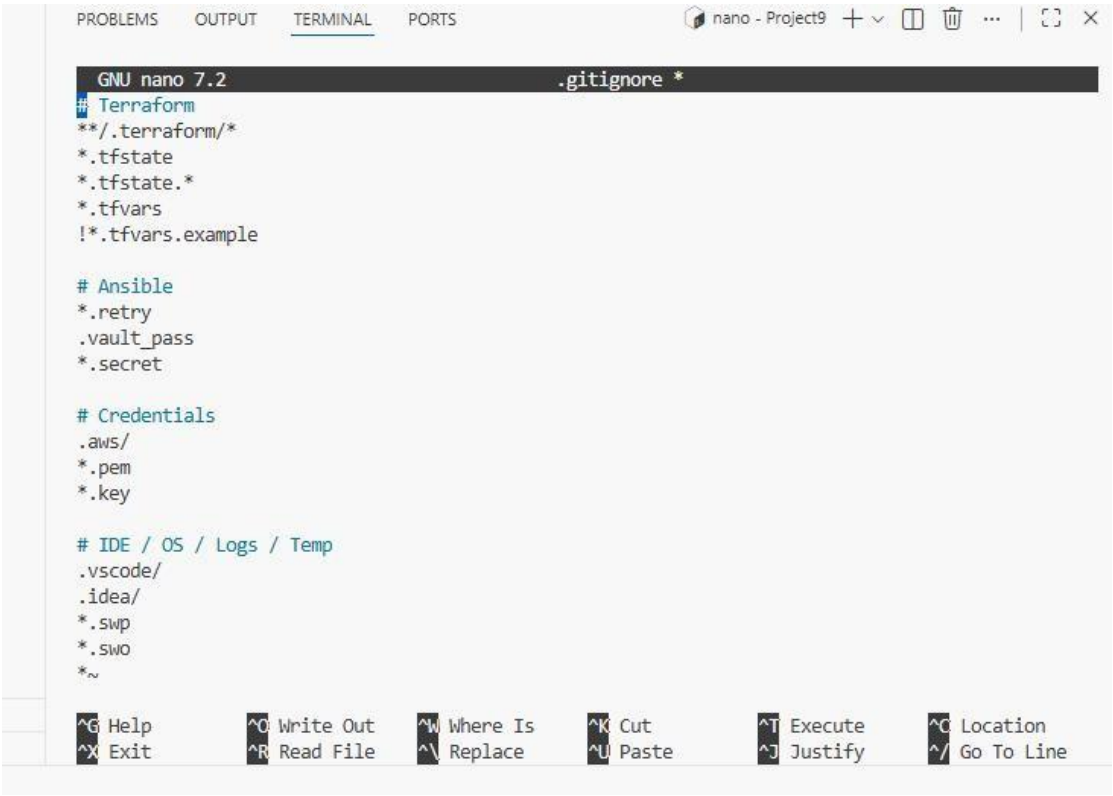


Figure 1.4: .gitignore File Configuration

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

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.

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.

Figure 2.1: Network Module

Figure 2.2: Network Module - VPC and Subnet Configuration

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.

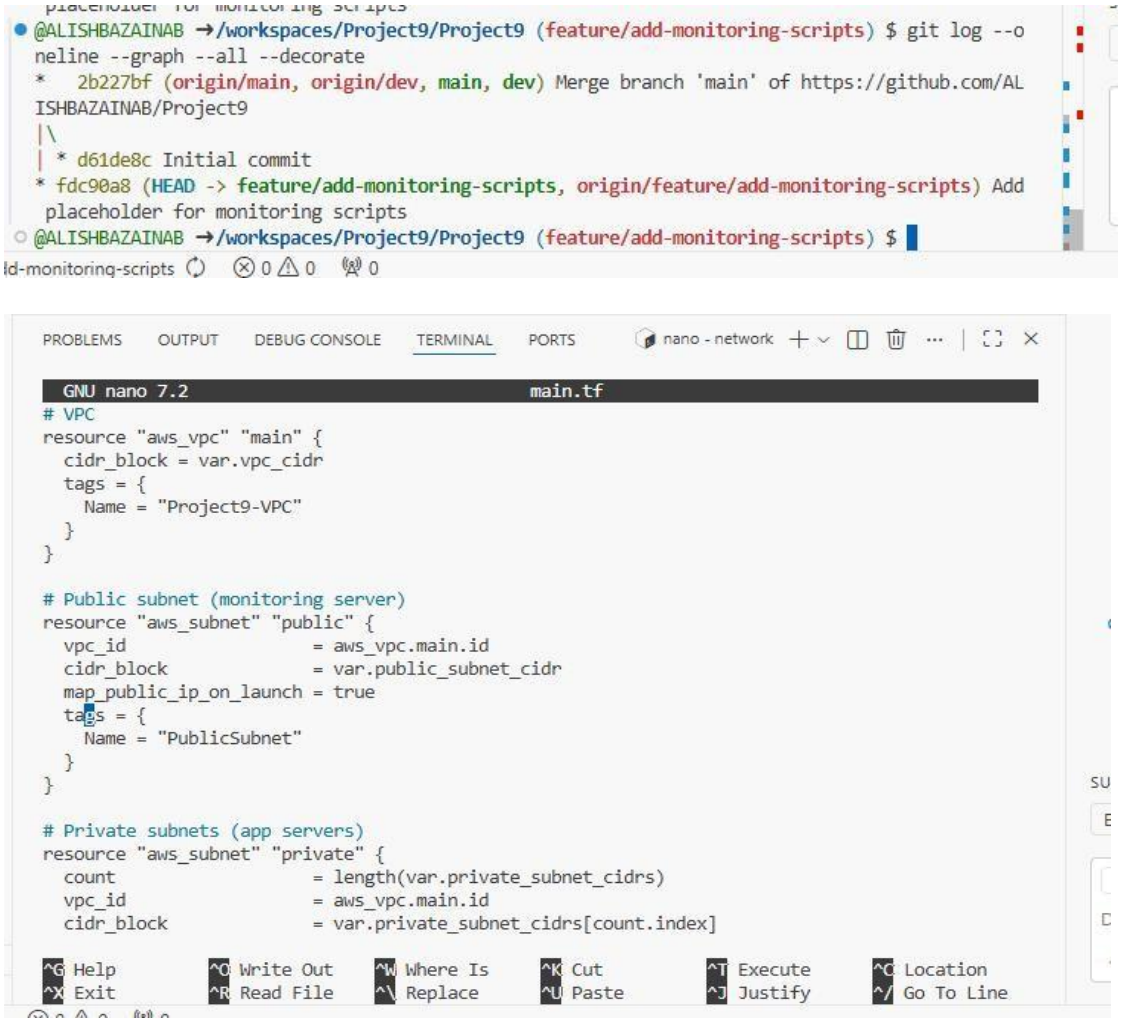


Figure 2.4: Monitoring Server Module - EC2 Instance Configuration

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

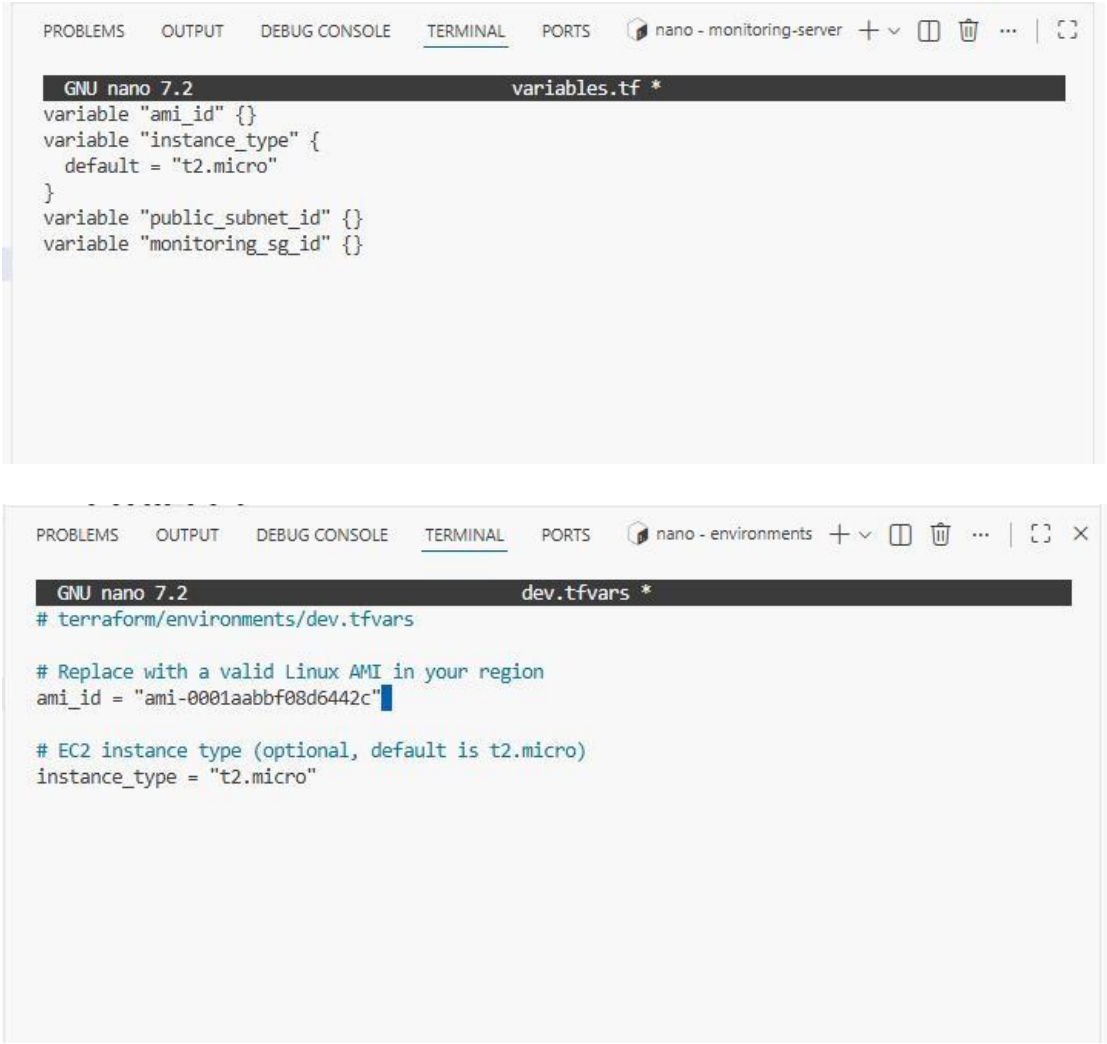


Figure 2.6: Root Terraform Configuration - Provider and Module Calls

Figure 2.7: Root Terraform Configuration - Network Module

Figure 2.8: Root Terraform Configuration - Variables

Figure 2.9: Root Terraform Configuration - Outputs

Terraform Initialization and Application

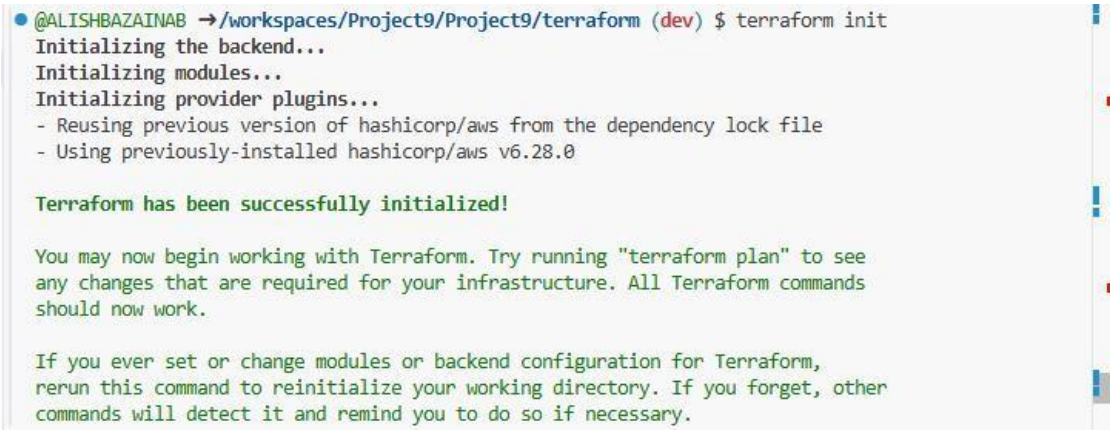


Figure 2.10: Terraform Initialization Success

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=environm
ents/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-0464d1c7fcb44686f]
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-0d4ddc3e
da75887a3]

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=environ
ments/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-0464d1c7fcb44686f]
module.network.aws_route_table.public: Refreshing state... [id=rtb-08987b62a48aea602]
module.network.aws_route_table_association.public_assoc: Refreshing state... [id=rtbassoc-0d4ddc3e
da75887a3]
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.

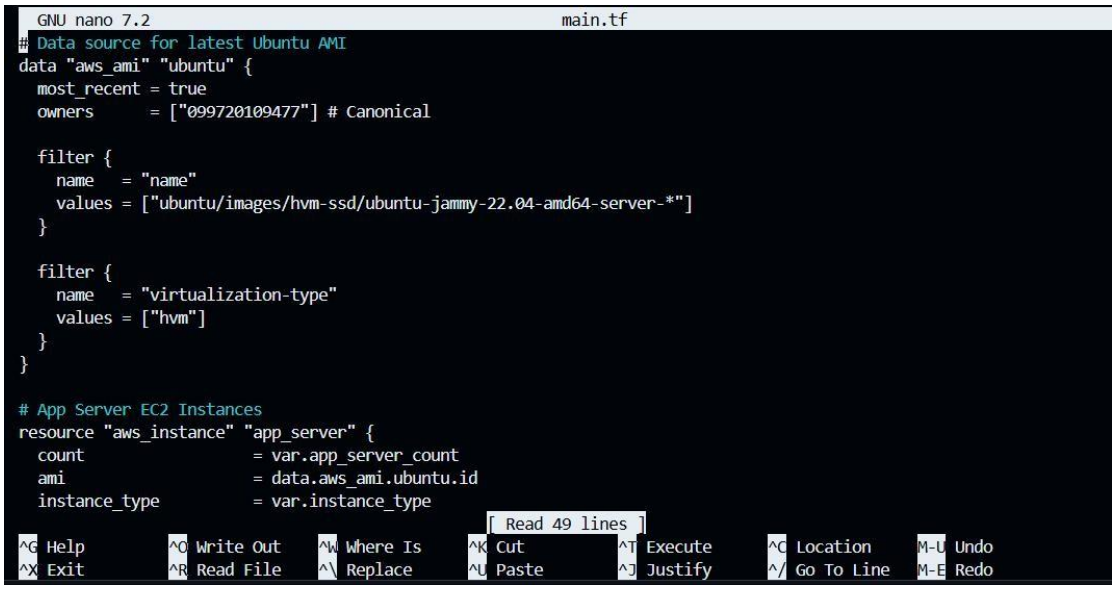


Figure 2.14: Application Servers Module - EC2 Configuration

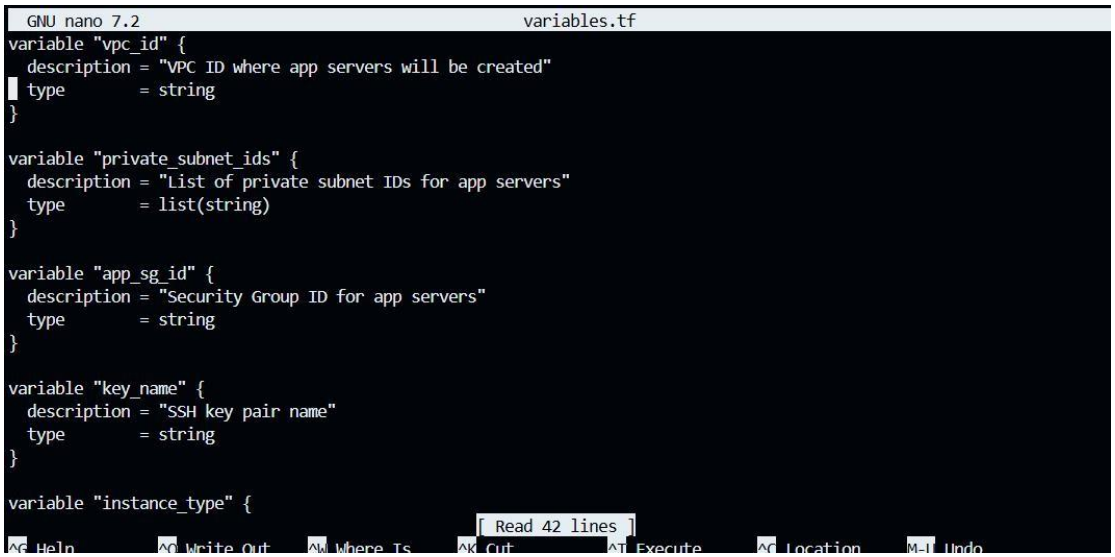


Figure 2.15: Application Servers Module - Variables

```
 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

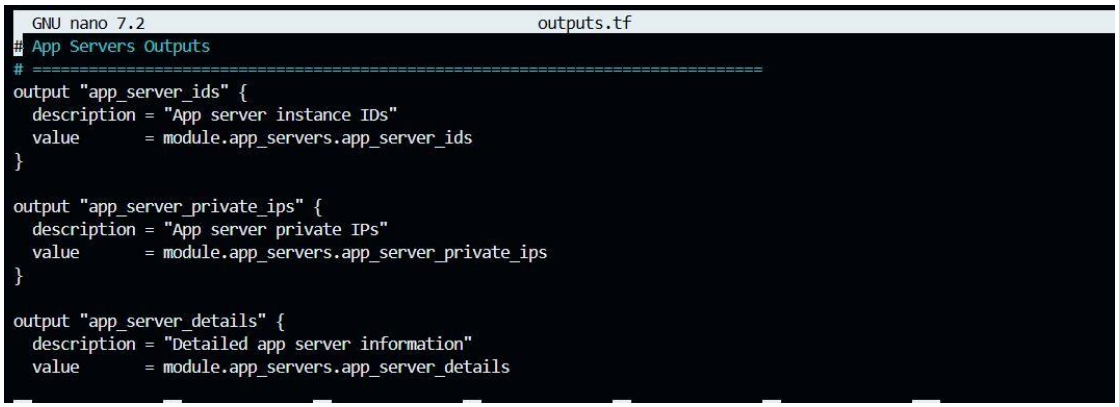


Figure 2.19: Application Servers Additional Configuration

Backend Configuration

Figure 2.20: Terraform Backend Configuration

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"
```

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

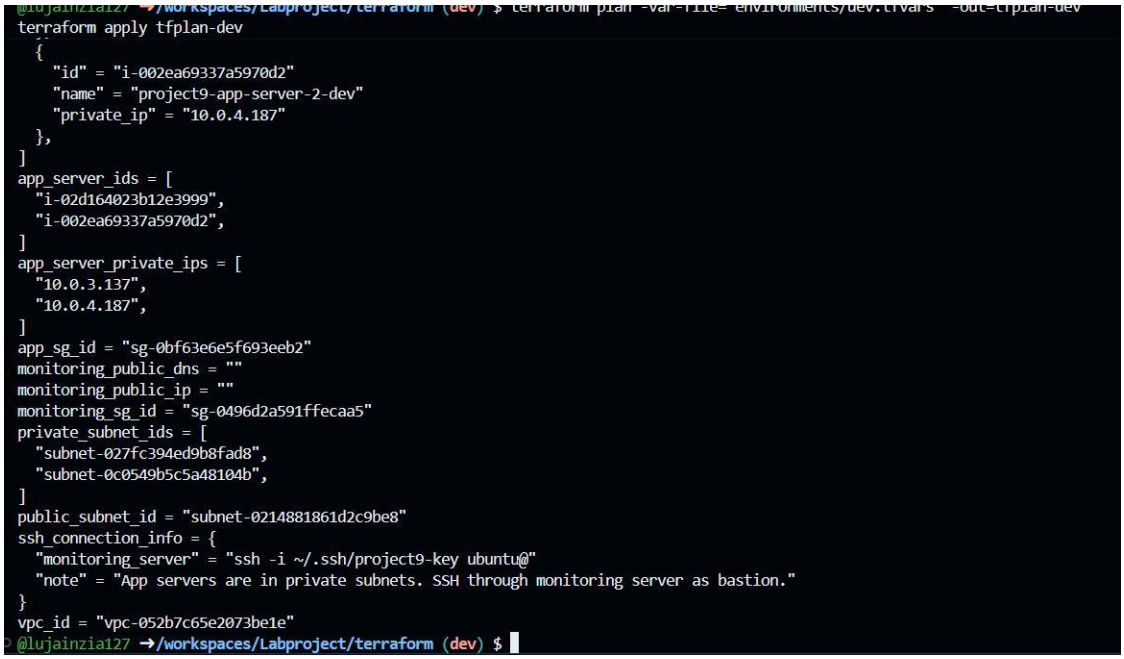


Figure 2.25: Application Servers - Provisioning Success

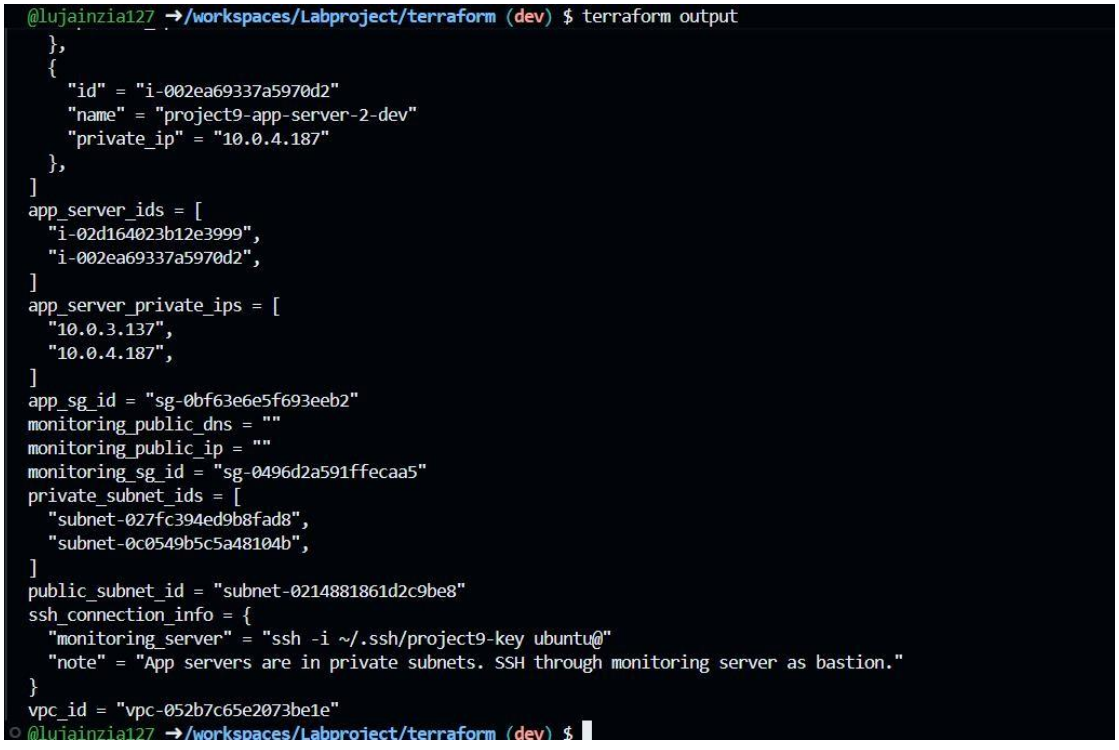


Figure 2.26: Application Servers - Output Showing Private IPs

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

```
[defaults]
# Inventory
inventory = ./inventory/dev_aws_ec2.yml
host_key_checking = False
remote_user = ec2-user
private_key_file = ~/.ssh/project9-key

# Performance
forks = 10
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible_facts
fact_caching_timeout = 3600

# 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

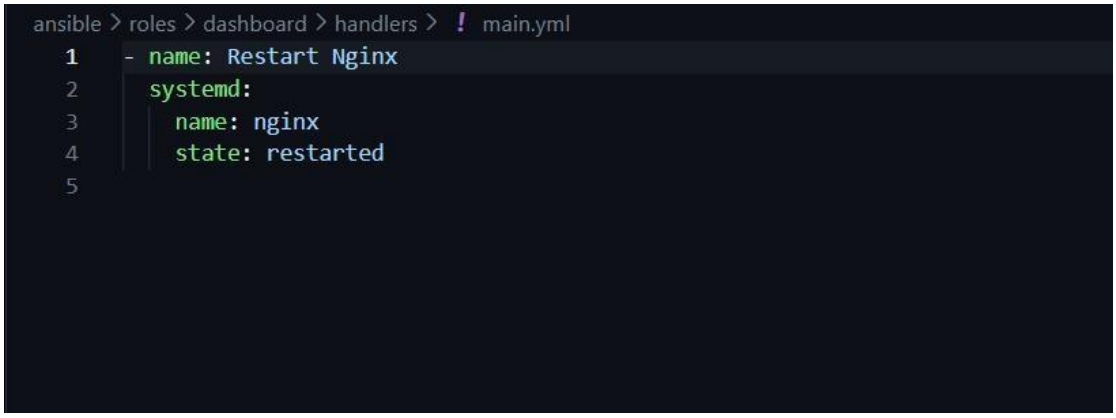


Figure 3.4: Nginx App Role - Handlers for Service Restart

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

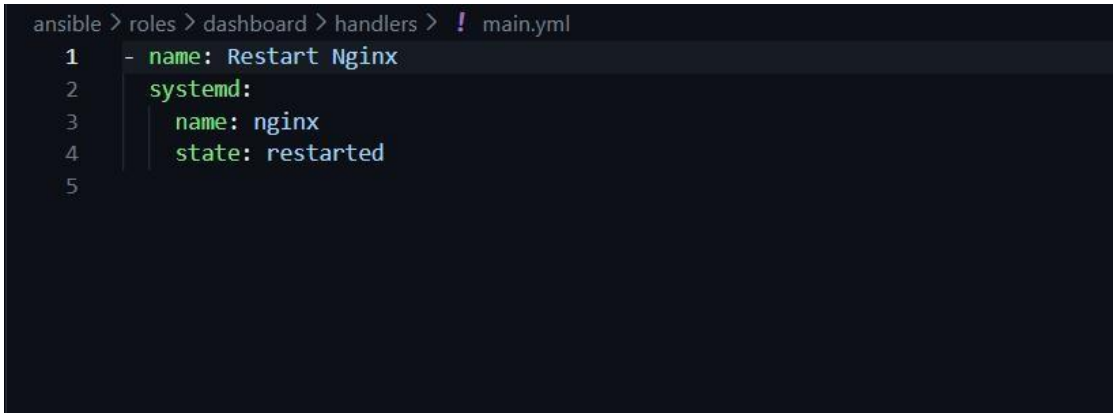


Figure 3.5: Dashboard Role - Nginx Handlers

```
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 "$HEALTH_DATA" | jq -r '.servers[] | @json' | while read -r server; do
264     if [ "$STATUS" = "UP" ]; then
265         CLASS="up"
266         BADGE_CLASS="up"
267     else
268         CLASS="down"
269         BADGE_CLASS="down"
270     fi
271
272     SERVERS_HTML+="<div class=\"server-item $CLASS\">$SERVER_IP<span class=\"status-badge $BADGE_CLASS
273 done
274
275 sed -i "s|SERVERS_PLACEHOLDER|$SERVERS_HTML|g" "$DASHBOARD_FILE"
276
277 echo "Dashboard updated at $TIMESTAMP"
```

Figure 3.8: Build Dashboard Script - Generates HTML Dashboard

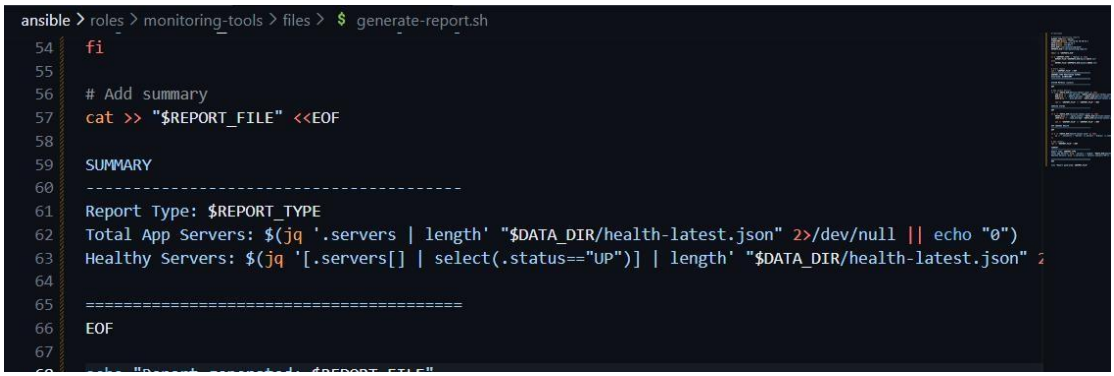


Figure 3.9: Generate Report Script - Creates Daily/Weekly Reports

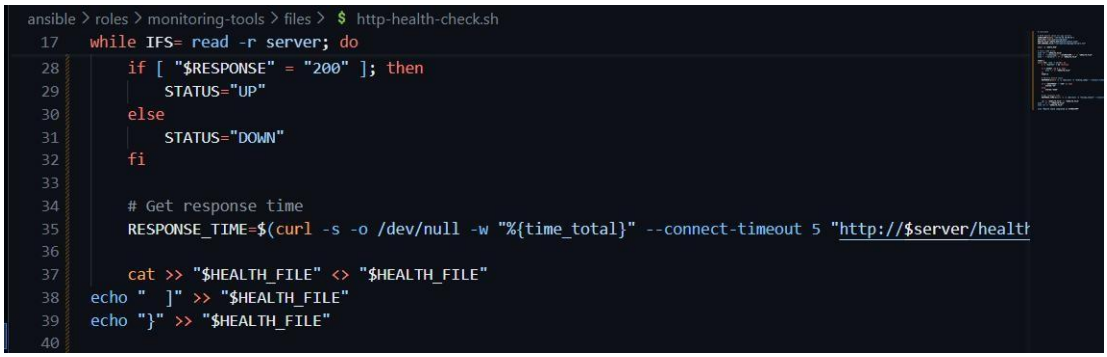


Figure 3.10: HTTP Health Check Script - Checks /health Endpoints

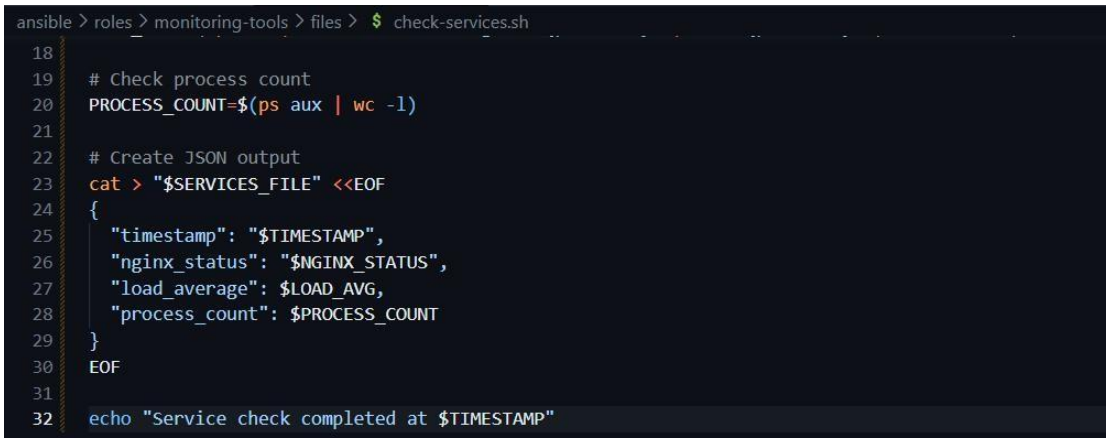


Figure 3.11: Check Services Script - Monitors Service Status

Figure 3.12: Collect Metrics Script - Gathers System Metrics

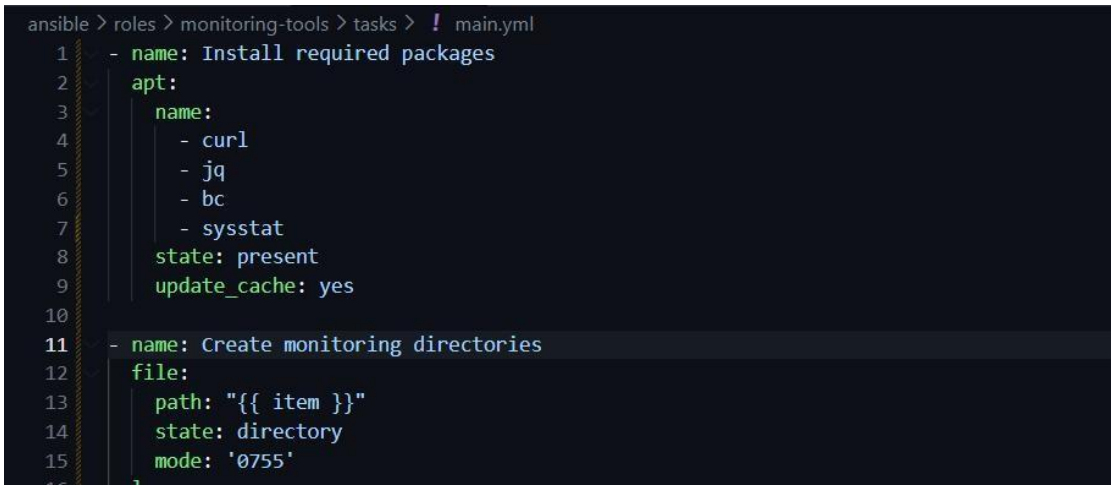


Figure 3.13: Monitoring Tools Role - Installation and Cron Setup

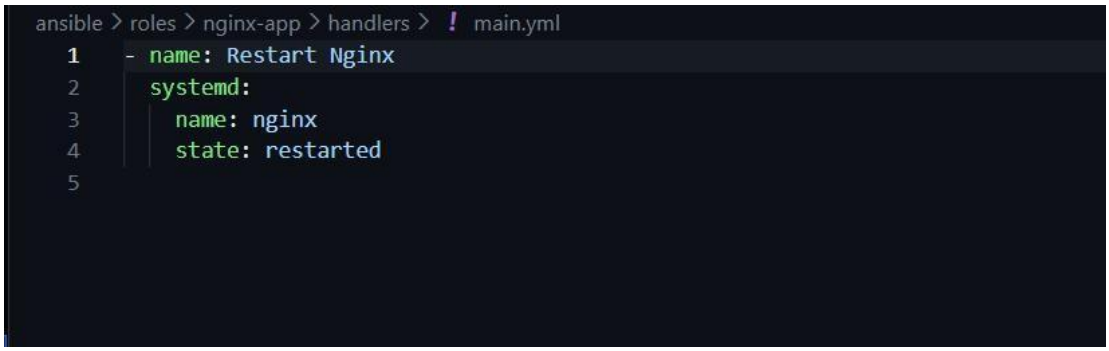


Figure 3.14: Nginx App Role - Handler Configuration

```
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

```yaml
ansible > playbooks > ! collect-logs.yml
- name: Collect Logs from App Servers
  tasks:
    - name: Create local logs directory
      delegate_to: localhost
      become: no

    - name: Fetch Nginx access logs
      fetch:
        src: /var/log/nginx/access.log
        dest: "./collected-logs/{{ inventory_hostname }}/{{ ansible_date_time.date }}/access.log"
        flat: yes

    - name: Fetch Nginx error logs
      fetch:
        src: /var/log/nginx/error.log
        dest: "./collected-logs/{{ inventory_hostname }}/{{ ansible_date_time.date }}/error.log"
```

Figure 3.18: Collect Logs Playbook - Uses Ansible Fetch Module

```yaml
ansible > playbooks > ! generate-report.yml
  hosts: role_monitoring
  become: yes

  tasks:
    - name: Generate daily report
      command: /usr/local/bin/monitoring/generate-report.sh daily
      when: report_type == "daily" or report_type is not defined

    - name: Generate weekly report
      command: /usr/local/bin/monitoring/generate-report.sh weekly
      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

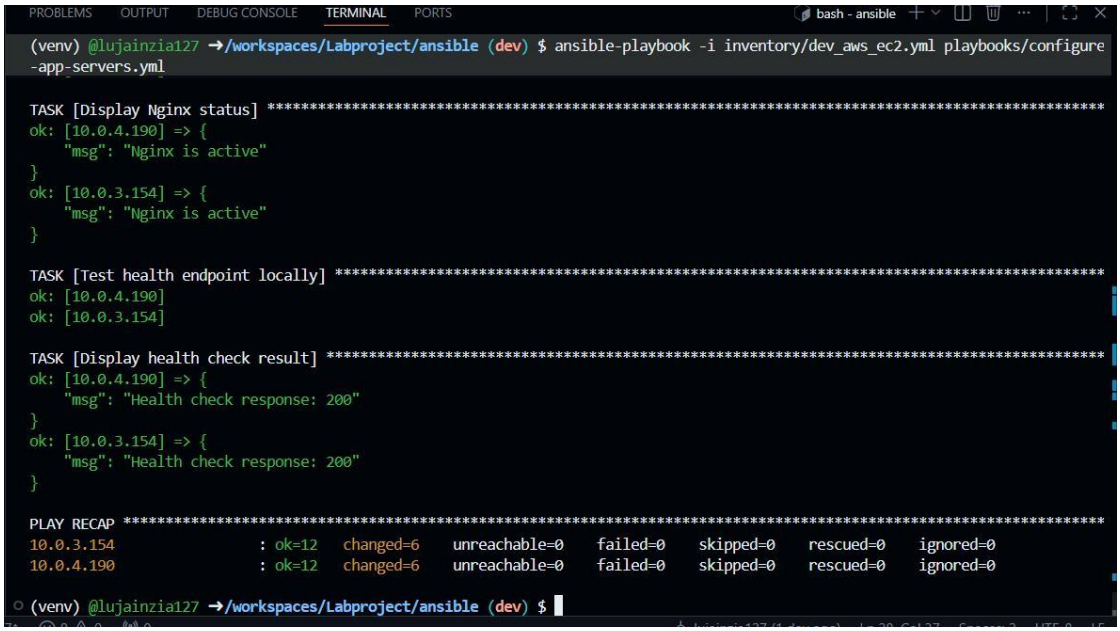


Figure 3.21: Configure App Servers Playbook Execution - Success

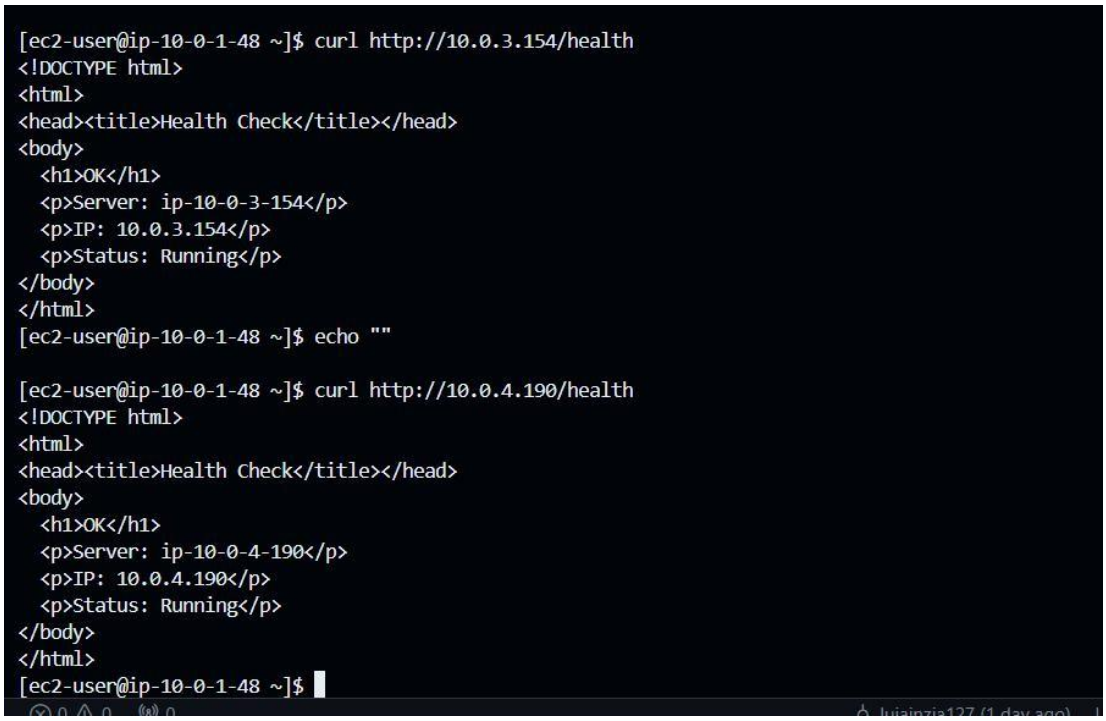


Figure 3.22: Curl Test Showing /health Endpoint Working on All Servers

3.3 Log Collection Using Ansible

Figure 3.23: Monitoring Server Configuration Playbook Execution

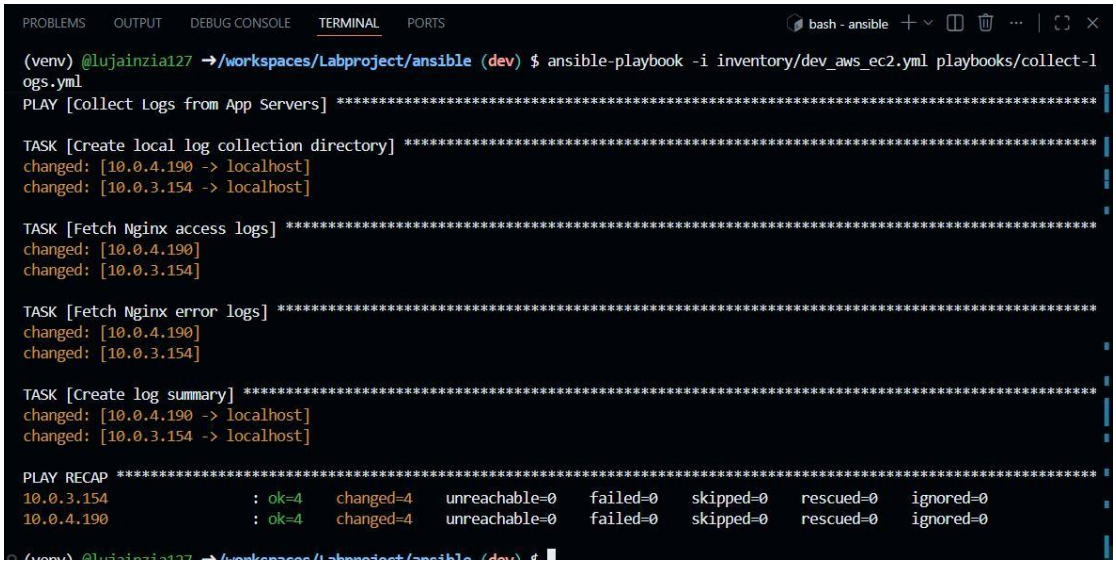


Figure 3.24: Log Collection Playbook - Fetching Logs from App Servers

```
(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

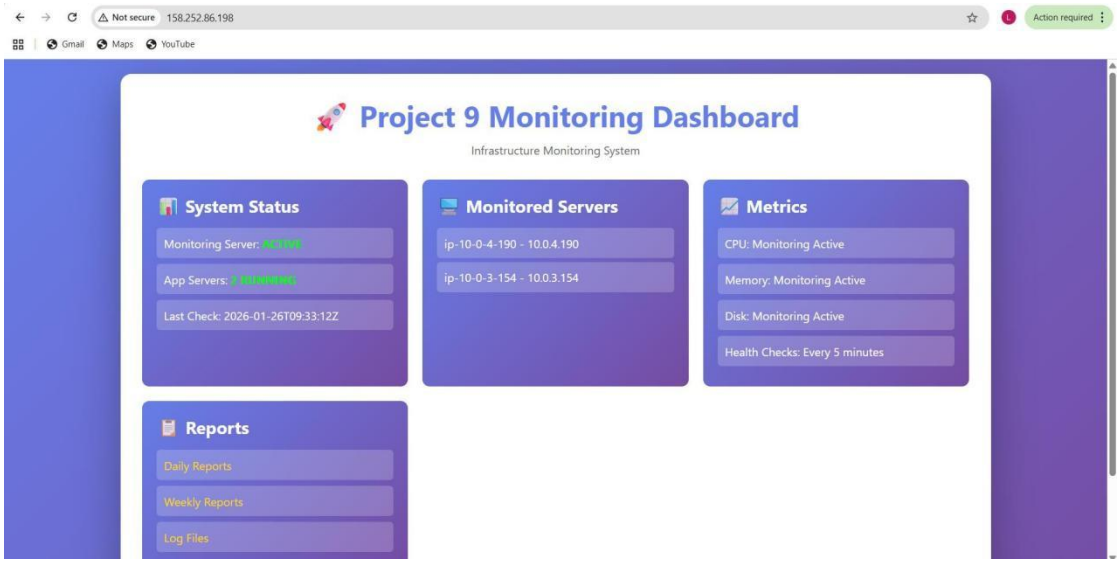


Figure 4.2: Monitoring Dashboard - Header and Server Status

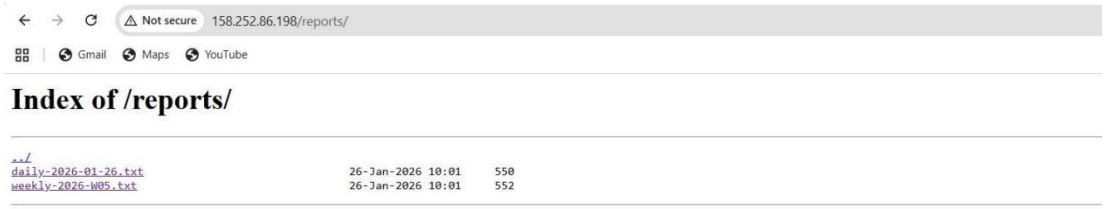


Figure 4.3: Monitoring Dashboard - Links to Weekly and Daily Reports

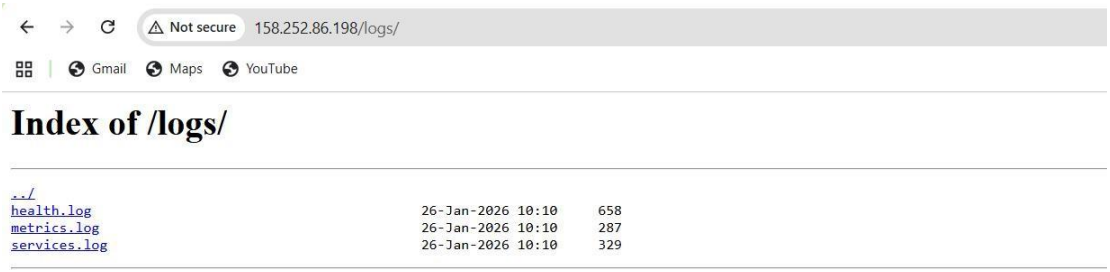


Figure 4.4: Dashboard - Links to log Files

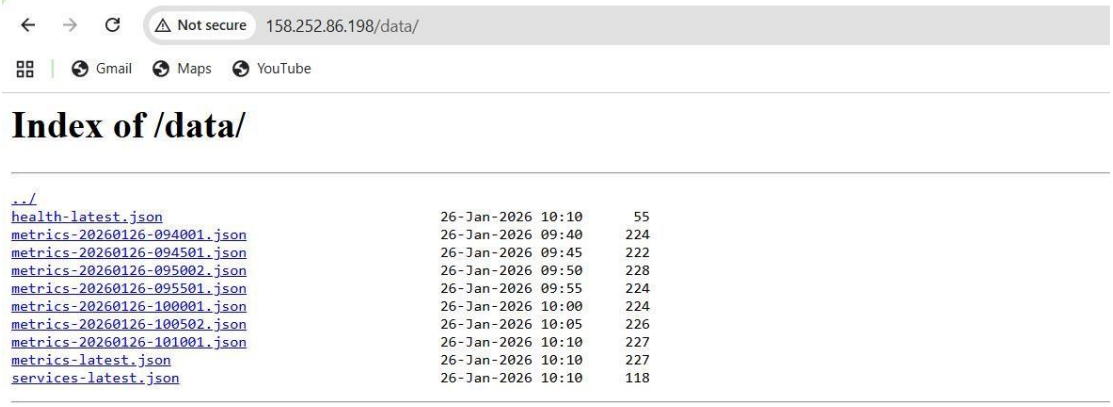


Figure 4.5: Dashboard - Links to Reports and Collected Logs

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.

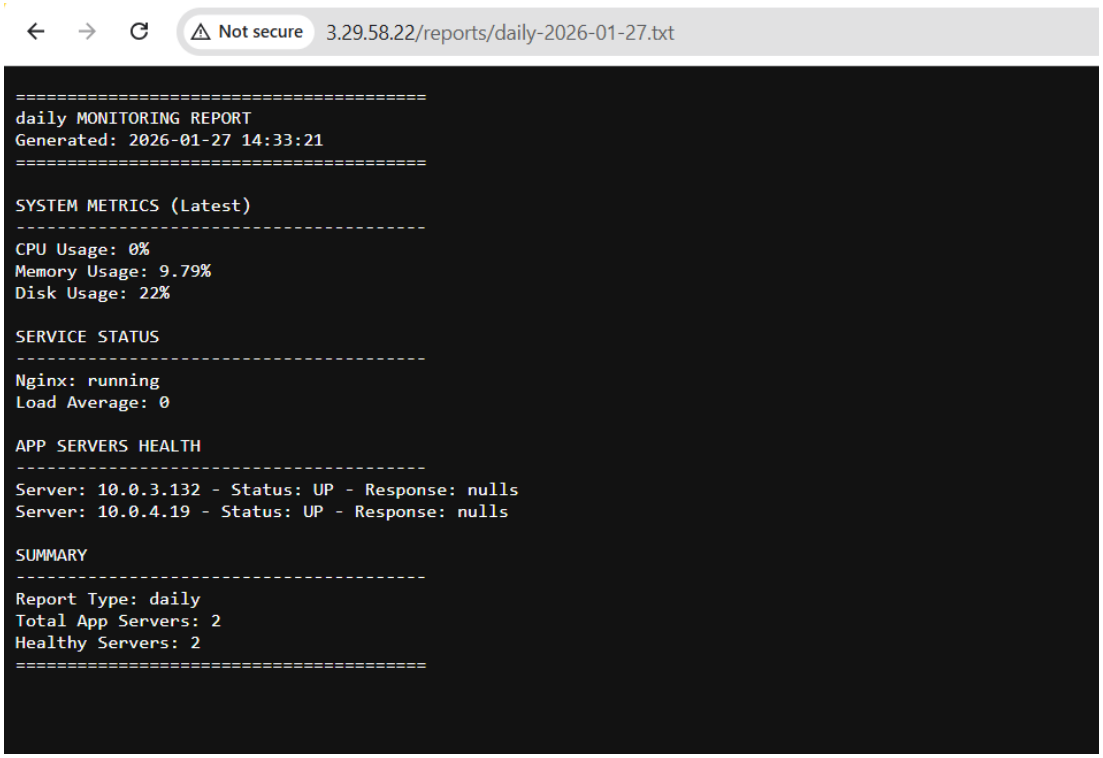


Figure 4.6 :Report -Daily Monitoring

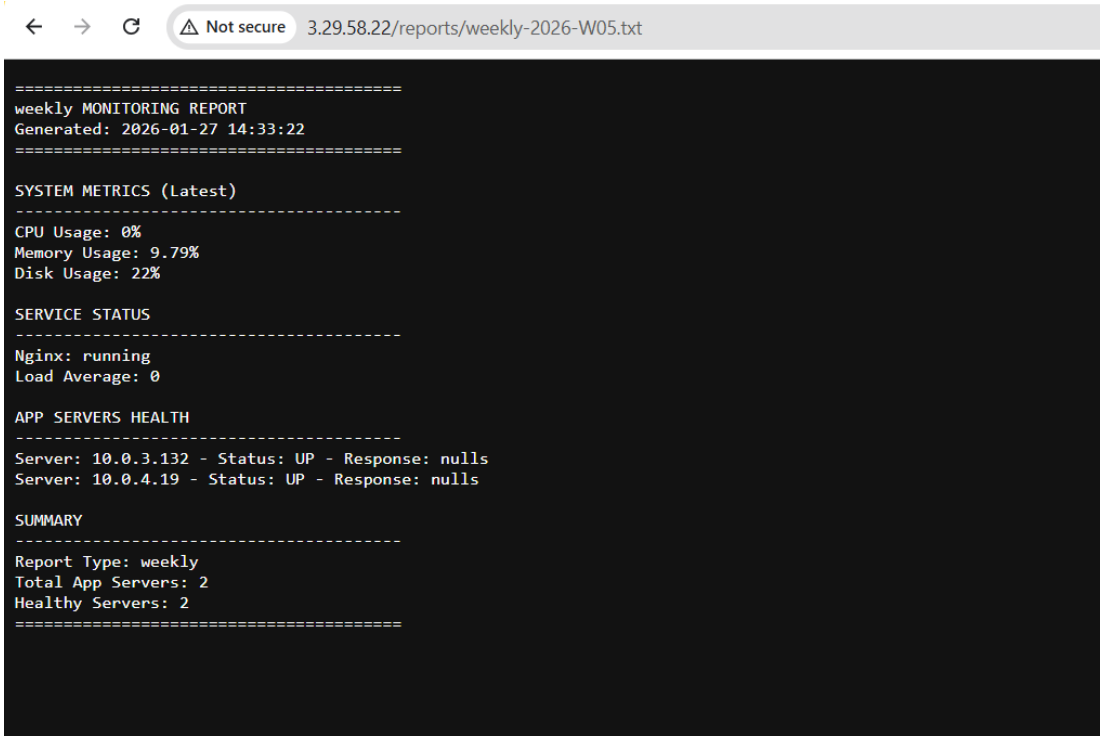


Figure 4.7 : Report -Weekly Monitoring

Merging Branch

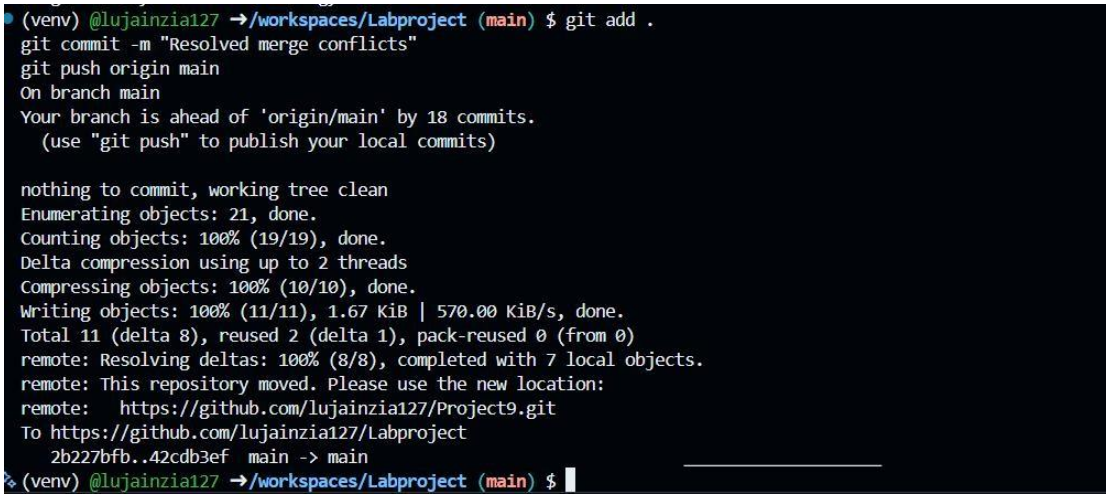


Figure 5 : Git Branch Merge History

Part 5 : Documentation

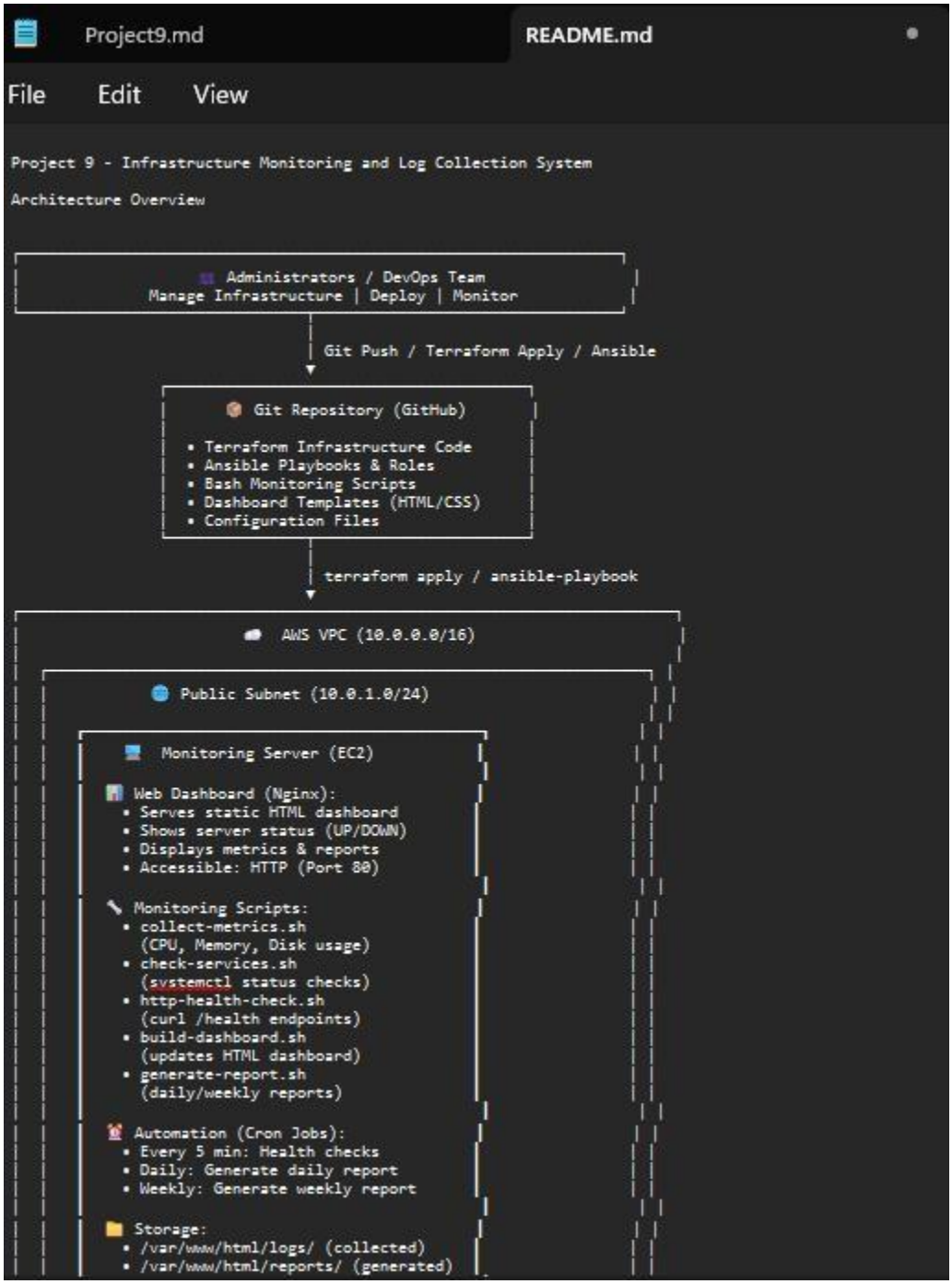


Figure 5.1: README.md - Project Overview and Architecture

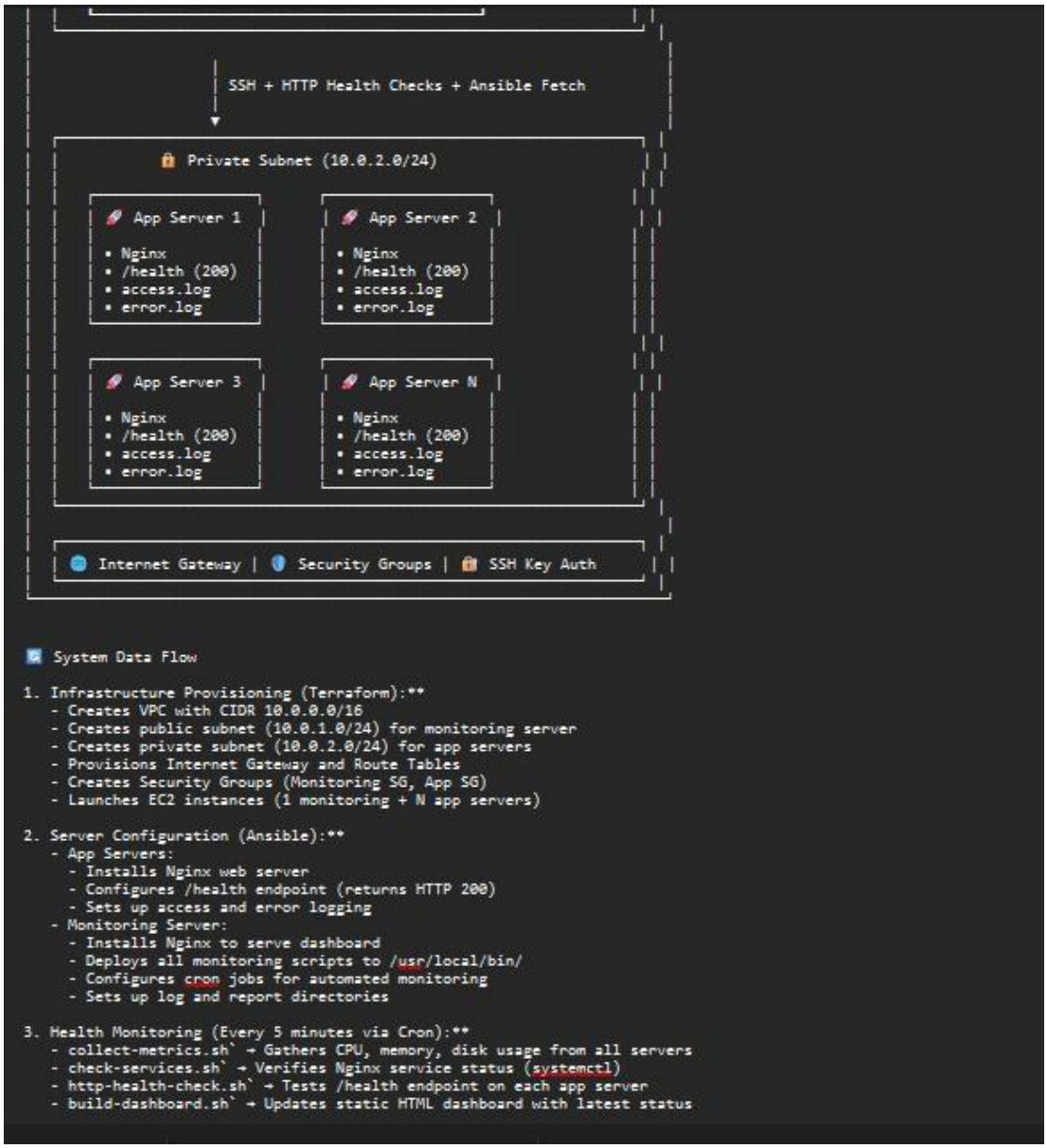
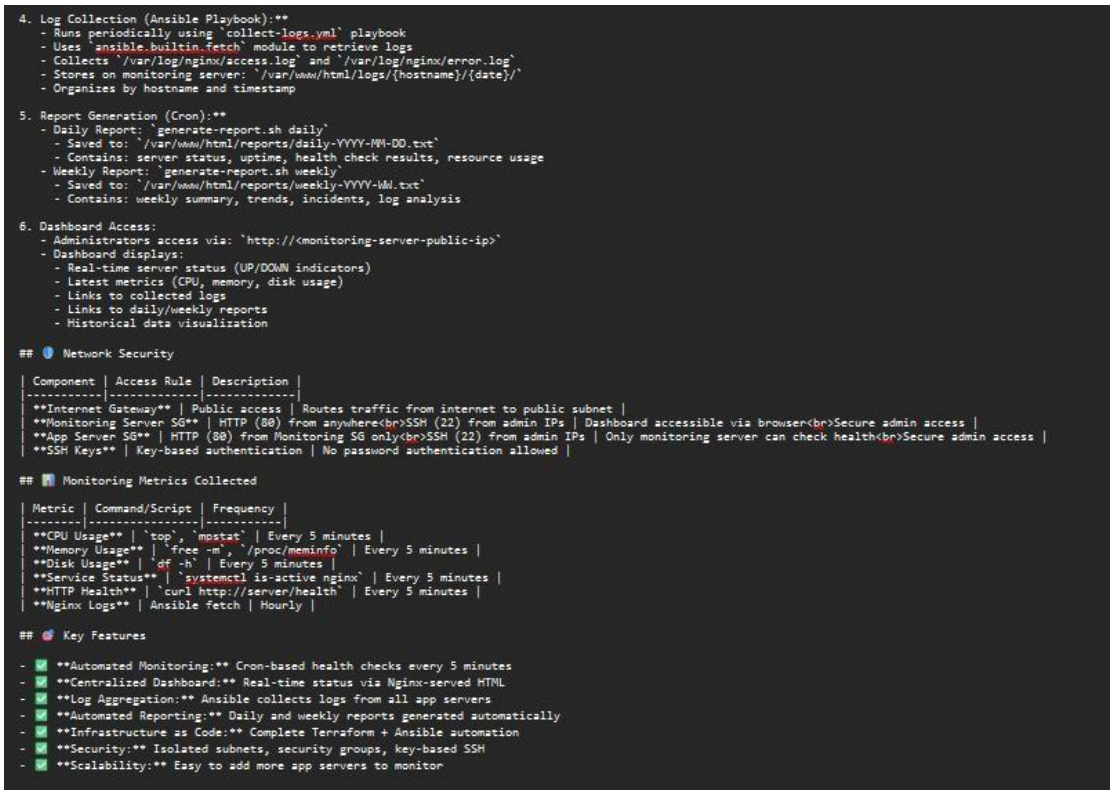


Figure 5.2: README.md - Setup Instructions and Configuration Guide



```
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-WW.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

- ✅ **Automated Monitoring:** Cron-based health checks every 5 minutes
- ✅ **Centralized Dashboard:** Real-time status via Nginx-served HTML
- ✅ **Log Aggregation:** Ansible collects logs from all app servers
- ✅ **Automated Reporting:** Daily and weekly reports generated automatically
- ✅ **Infrastructure as Code:** Complete Terraform + Ansible automation
- ✅ **Security:** Isolated subnets, security groups, key-based SSH
- ✅ **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