Secure Cloud Infrastructure Monitoring and Compliance Automation using Azure, Puppet, Nagios, and Terraform

A Project report

Submitted in partial fulfilment of the requirements for the award of a degree of Bachelor of Technology

(Computer Science and Engineering)

Submitted to

LOVELY PROFESSIONAL UNIVERSITY

PHAGWARA, PUNJAB

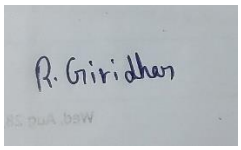
Submitted by

Name of the Student

Ruppa Giridhar
Registration Number:12207968

Name of the Supervisor

Divya Thakur



Signature of Student

Signature of Supervisor

DECLARATION BY STUDENT

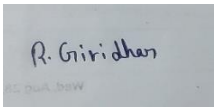
We **To whom so ever it may concern.**

I **Ruppa Giridhar (12207968),** hereby declare that the work done by me on **"Secure Cloud Infrastructure Monitoring and Compliance Automation using Azure, Puppet, Nagios, and Terraform"** under the supervision of **(Divya Thakur-Assistant Professor)**, Lovely professional University, Phagwara, Punjab, is a record of original work for the partial fulfilment of the requirements for the award of the degree, **in Computer Science Engineering from Lovely Professional University, Phagwara.**

Project Group Number:

Name of Student 1: Ruppa Giridhar

Registration Number:12207968



Signature of Student 1
Date: 14-11-2025

ACKNOWLEDGEMENT

Heartfelt thanks are due to those who supported and guided me in many ways throughout the project.

I would like to express my deepest gratitude to my guide and mentor, **Ms. Divya Thakur**, whose encouragement, suggestions, and support were constant throughout the project. Her guidance was instrumental in defining the course of this work and helped transform some raw ideas into a practically viable and implementable solution. I will always appreciate having the opportunity to carry out the project under her guidance.

I also appreciate the constructive feedback, collaboration, and motivation from my peers and friends. Their contributions created such an energetic and bright atmosphere while moving the project, making the process enriching and fulfilling.

Finally, my gratitude goes to my university for providing this opportunity and resources to undertake this project to give me an avenue to apply theoretical knowledge in addressing real-world problems.

Ruppa Giridhar
Bachelor of Technology
(Computer Science and Engineering)

1) ABSTRACT:

The project **"Secure Cloud Infrastructure Monitoring and Compliance Automation using Azure, Puppet, Nagios, and Terraform"** focuses on building a secure, automated, and continuously monitored cloud environment by integrating Infrastructure as Code (IaC), configuration management, and real-time monitoring tools.

Using **Terraform**, the entire Azure cloud infrastructure—including virtual machines, networking, and security configurations—was provisioned automatically, ensuring consistency and scalability. The **Puppet Master** server was deployed to enforce configuration compliance and automate system management tasks such as security patching and service configuration across all connected nodes. Simultaneously, **Nagios Core** was implemented for real-time performance monitoring and fault detection, enabling proactive identification of issues like resource utilization, downtime, and connectivity failures.

The setup consists of two Linux-based Azure virtual machines: one acting as a **Puppet Master + Nagios Server**, and the other as a **Puppet Agent node**, both securely connected through an Azure Virtual Network and governed by a Network Security Group (NSG). The **Nagios dashboard** provides centralized visibility into the health and availability of all monitored nodes, while Puppet ensures that configurations remain compliant and secure.

This project demonstrates the practical implementation of **DevSecOps principles**, combining automation, security, and continuous monitoring in a cost-effective manner using open-source tools and Azure student resources. It not only strengthens operational reliability and system security but also serves as a scalable foundation for enterprise-grade cloud monitoring and compliance management.

2) INTRODUCTION:

Cloud computing has become the backbone of modern IT infrastructure, enabling organizations to deploy scalable, flexible, and cost-efficient systems. However, as cloud environments grow, so do the challenges related to **security, configuration management, infrastructure consistency, and continuous monitoring**. Ensuring that cloud resources are properly configured, securely maintained, and continuously monitored has therefore become a critical requirement in modern DevSecOps practices.

This project, **"Secure Cloud Infrastructure Monitoring and Compliance Automation using Azure, Puppet, Nagios, and Terraform,"** aims to address these challenges by integrating three major automation and security components:

1. **Infrastructure as Code (IaC)** for automated resource provisioning

2. **Configuration Management** for enforcing security policies

3. **Continuous Monitoring** for real-time visibility and alerting

Using **Terraform**, the entire Azure-based cloud environment—including virtual machines, networking components, public IPs, and security rules—was created automatically. This eliminates manual errors and ensures infrastructure repeatability. The primary server hosts **Puppet Master**, which acts as the central configuration management system, enforcing secure configurations and maintaining system compliance automatically across all nodes.

Alongside Puppet, **Nagios Core** is deployed on the same server to provide real-time monitoring of the infrastructure. Nagios tracks host availability, service status, network latency, CPU load, memory usage, and other performance metrics. The secondary virtual machine functions as a **Puppet Agent**, receiving configuration updates from the master, while being continuously monitored by Nagios through both Ping and NRPE-based checks.

The integration of these three technologies—Terraform, Puppet, and Nagios—forms a complete **DevSecOps pipeline**, delivering automated provisioning, secure configuration enforcement, and active monitoring in a unified cloud setup. This project showcases how open-source tools and Azure student resources can be combined to build a secure, scalable, and production-ready cloud environment suitable for enterprise use.

3) Objective of the Project

The main objective of this project is to design and implement a **secure, automated, and continuously monitored cloud infrastructure** by integrating Infrastructure as Code (IaC), configuration management, and real-time monitoring tools. This project demonstrates how cloud environments can be deployed, secured, and observed efficiently using modern DevSecOps practices.
The specific objectives of the project are:

1. **Automate Cloud Infrastructure Deployment Using Terraform**
- To create Azure resources such as virtual machines, virtual networks, network security groups, and public IPs using Terraform.
- To ensure infrastructure consistency, reduce manual errors, and enable rapid provisioning through IaC.

2. **Implement Secure Configuration Management Using Puppet**
- To deploy and configure a **Puppet Master** and **Puppet Agent** setup for automatic configuration enforcement.
- To maintain system compliance, enforce security policies, and ensure uniform configurations across all nodes.

3. **Establish Real-Time Monitoring Using Nagios Core**
- To deploy Nagios Core for monitoring host availability, service status, system performance, and failures.
- To configure Ping, SSH, CPU, memory, and disk monitoring for complete visibility of system health.

4. **Enhance Cloud Security Through Network Controls**
- To restrict access via Azure Network Security Groups (NSG) and allow only required ports (22, 80, 8140, ICMP).
- To ensure that monitoring and configuration tools function securely within a controlled network environment.

5. **Demonstrate Integration of DevSecOps Practices**
- To combine Terraform (IaC), Puppet (config management), and Nagios (monitoring) into one unified DevSecOps workflow.
- To show how automation improves reliability, reduces operational overhead, and strengthens cloud security.

6. **Provide a Scalable and Reproducible Solution**
- To build a framework that can be easily scaled or replicated by modifying Terraform templates.
- To create a foundation for enterprise-level cloud compliance and monitoring systems.

4) Scope of the Project:

Project Scope

The scope of this project covers the **design, deployment, configuration, and monitoring** of a secure cloud infrastructure using a combination of Terraform, Puppet, and Nagios within the Microsoft Azure environment. The project focuses on applying DevSecOps principles to create an automated, secure, and observable cloud setup.

The key areas included within the scope are:

☐ Infrastructure Provisioning with Terraform

- Creating a complete Azure cloud environment through Infrastructure as Code (IaC).
- Automating the deployment of:
    - Resource Group
    - Virtual Network and Subnet
    - Network Security Group (NSG) with controlled ports
    - Two Ubuntu-based Virtual Machines
    - Public IP configurations
- Ensuring repeatability and consistency in infrastructure deployment.

☑ Secure Configuration Management with Puppet

- Installing and configuring **Puppet Master** on the primary VM.
- Setting up **Puppet Agent** on the secondary VM.
- Automating system configurations such as:
    - Firewall rules
    - Service management
    - Security updates
    - Compliance settings
- Enforcing uniform security and system policies across all nodes.

3. Real-Time Monitoring with Nagios Core

- Deploying Nagios Core on the Puppet Master server.
- Monitoring host availability, ping status, and service uptime.
- Implementing essential checks using:
    - check_ping
    - check_ssh
    - NRPE-based checks (CPU load, memory, disk usage).
- Visualizing system health through the Nagios Web Interface.

4. Network Security and Controlled Access

- Designing secure communication between Puppet Master and Agent.
- Restricting inbound and outbound access using Azure NSG rules.
- Allowing only essential ports (22, 80, 8140, and ICMP).
- Ensuring a secure monitoring and configuration channel.

5. DevSecOps Integration

- Demonstrating the integration of IaC, configuration management, and monitoring in a unified workflow.
- Highlighting automation benefits such as reduced manual effort, improved security posture, and faster deployments.

6. Documentation and Demonstration

- Preparing proper technical documentation for each phase of implementation.
- Providing screenshots, architecture diagrams, configuration files, and monitoring results.
- Presenting a complete working solution suitable for academic evaluation or real-world application.

5) Problem Identification

With the rapid adoption of cloud computing, organizations increasingly rely on virtualized environments to deploy applications and services. However, this shift introduces several operational and security challenges related to infrastructure management, configuration consistency, and continuous monitoring. Traditional manual approaches are often inefficient, error-prone, and difficult to scale, especially when managing multiple cloud resources.

The major problems identified in this project are:

1. Lack of Automated Infrastructure Deployment

Provisioning cloud resources manually through the Azure portal results in:

- Time-consuming setups
- Inconsistent configurations
- Higher chance of human error
  This affects reliability and makes replication difficult.

2. Inconsistent and Unsecured System Configurations

Without a centralized configuration management system:

- Each server may have different security patches
- Firewall rules may vary
- Misconfigurations go unnoticed
  This leads to vulnerabilities and compliance issues.

3. Absence of Real-Time Monitoring

Cloud environments need continuous health monitoring, but:

- Manual monitoring is not feasible
- Failures and downtime may remain unnoticed

- No centralized dashboard exists for resource visibility
  This can impact performance, availability, and security.

4. Difficulty Detecting System Failures or Performance Issues

Without automated tools:

- CPU spikes
- Memory leaks
- Disk space shortages
- Service failures
  are discovered only after causing system downtime.

5. Limited Security Control in Cloud Networks

Azure resources require strict network controls, but:

- Default settings may block essential services
- ICMP/ping is blocked, affecting health checks
- Inconsistent firewall rules lead to security risks
  Proper security configuration is essential but challenging to maintain manually.

6. Lack of Integrated DevSecOps Workflow

Organizations struggle to combine:

- Infrastructure automation
- Secure configuration enforcement
- Continuous monitoring
  into one seamless pipeline.
  This makes cloud operations slow, unsecured, and difficult to manage.

6) Motive for the Project

The primary motive behind this project is to demonstrate how modern cloud environments can be made **secure, automated, and continuously monitored** by integrating DevSecOps practices with open-source tools. As cloud adoption rapidly increases, ensuring that systems remain properly configured, secure, and highly available has become a major challenge for organizations.

Traditional manual approaches to infrastructure deployment, configuration management, and system monitoring are no longer efficient or scalable. They lead to inconsistencies, security gaps, and delayed incident response. This project aims to overcome these limitations by introducing automation at every stage of the cloud lifecycle.

The project is motivated by the need to:

1. Reduce Manual Effort Through Automation

To eliminate repetitive tasks and enable fast, error-free provisioning of cloud resources using Infrastructure as Code (Terraform).

2. Ensure Secure and Consistent Configurations

To maintain uniform security policies across all servers through an automated configuration management tool (Puppet).

3. Enable Continuous Monitoring and Incident Detection

To achieve real-time visibility into system health using Nagios and proactively identify failures, downtime, or performance degradation.

4. Apply DevSecOps Principles in a Practical Environment

To integrate deployment, security, and monitoring tools into a unified workflow that aligns with modern industry practices.

5. Build a Scalable and Reproducible Cloud Framework

To create a foundation that can easily be expanded, replicated, or adapted for enterprise environments.

6. Enhance Understanding of Cloud Security and Automation

To gain hands-on experience with Azure cloud services, network security groups, server configuration, and monitoring technologies in a real-world scenario.

7) Proposed Solution

To address the challenges of manual cloud provisioning, inconsistent configurations, and lack of centralized monitoring, this project proposes an integrated DevSecOps-based solution using **Terraform**, **Puppet**, and **Nagios**, deployed on **Microsoft Azure**. The solution combines automation, security, and observability to ensure that cloud resources are created, configured, and monitored efficiently and securely.

7.1 System Overview

The proposed system consists of a fully automated cloud environment that includes:

- **Terraform** for deploying Azure infrastructure (VMs, VNet, Subnet, NSG, Public IPs) through Infrastructure as Code.
- **Puppet Master Server** for enforcing configuration policies, security settings, and compliance across all nodes.
- **Puppet Agent Node** that receives configuration instructions from the master, ensuring consistent and secure system setup.
- **Nagios Core Monitoring Server** for real-time monitoring of host availability, service status, system performance, and network connectivity.
- **Azure Network Security Group (NSG)** to secure the environment by allowing only essential ports and protocols.
- A centralized dashboard (Nagios Web UI) where administrators can track system health and receive alerts.

The system architecture ensures:

- Automated provisioning
- Secure configuration enforcement
- Continuous health monitoring
- Centralized visibility
- High reliability with minimal manual effort

The entire workflow follows DevSecOps principles, enabling seamless integration of deployment, security, and monitoring.

7.2 Key Features

◈ 1. Infrastructure as Code (IaC) with Terraform

- Automates the creation of Azure Virtual Machines, networks, and security rules.
- Ensures consistency, repeatability, and version control.
- Reduces human error in cloud deployment.

◈ 2. Centralized Configuration Management with Puppet

- Puppet Master controls configurations for all connected nodes.
- Puppet Agent pulls configurations automatically.
- Ensures uniform security policies across the infrastructure.
- Automates maintenance tasks such as package installations and updates.

◈ 3. Real-Time Monitoring with Nagios Core

- Monitors host uptime, ping status, and essential services.
- Supports advanced checks (CPU, memory, disk) via NRPE.
- Provides alerts on failures or performance issues.
- Offers a web interface for centralized system visibility.

◈ 4. Enhanced Cloud Security

- Azure NSG restricts traffic to only required ports (22, 80, 8140, and ICMP).
- Puppet reinforces secure configurations (e.g., SSH hardening, firewall rules).
- Nagios helps detect abnormal behavior and downtime quickly.

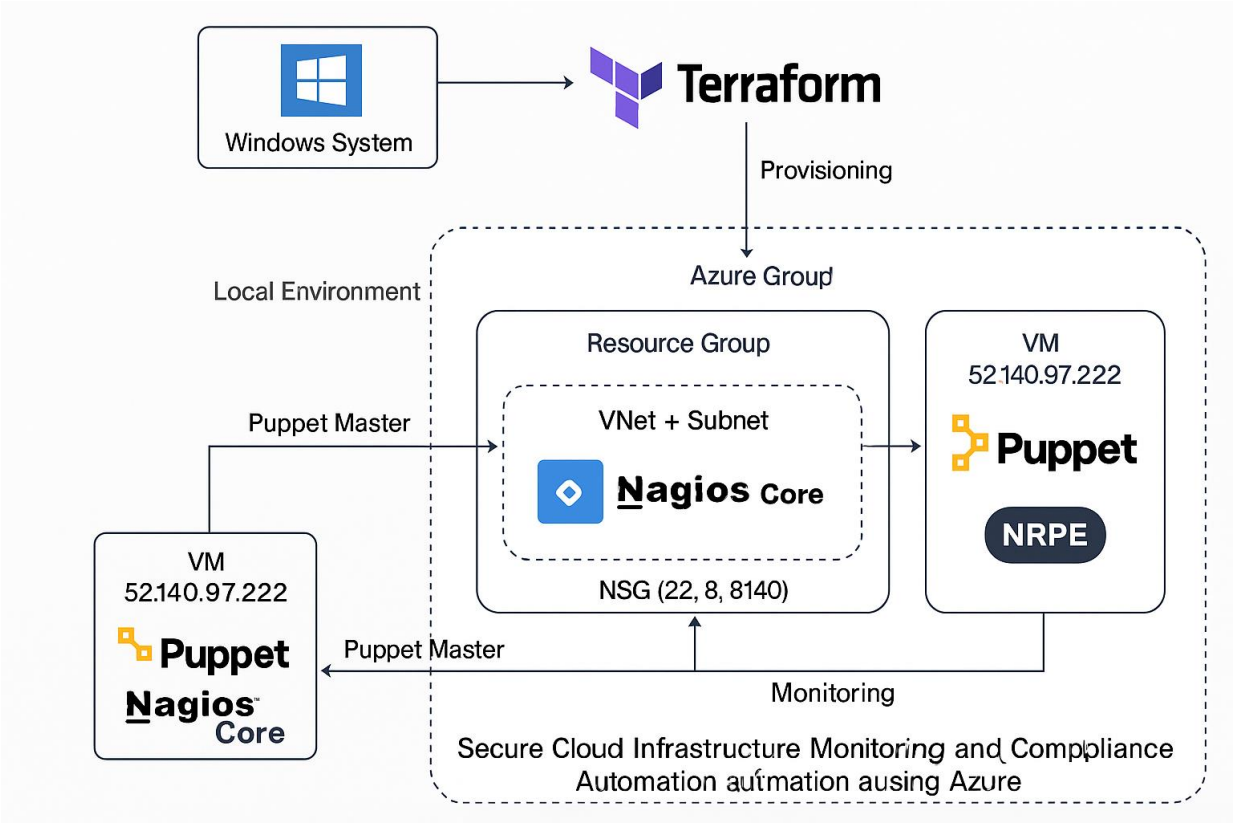
◈ 5. Scalable and Modular Architecture

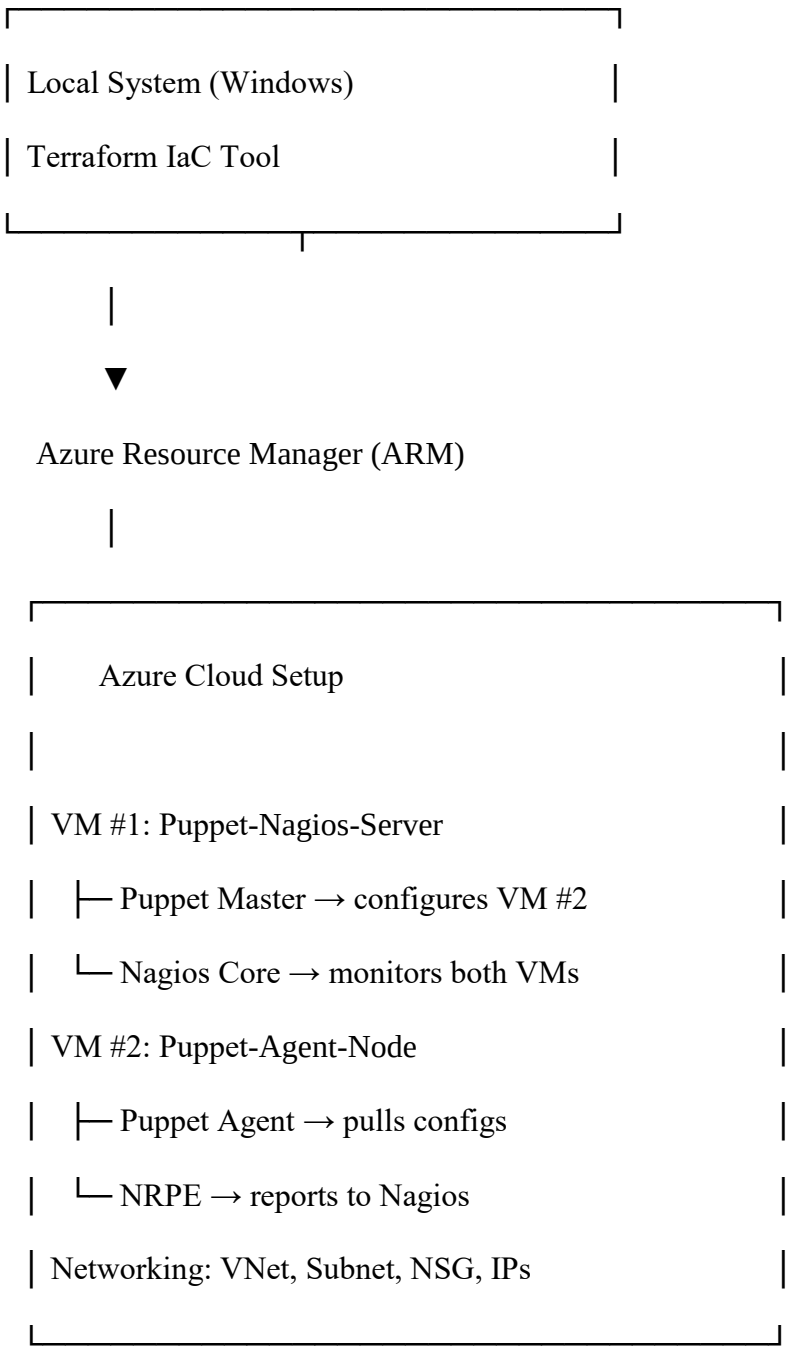
- Additional nodes can be added easily by modifying Terraform files.
- Puppet automatically enforces configurations on new agents.
- Nagios can instantly monitor new servers with minimal setup.

8. System Architecture

8.1 Architecture Diagram

The system architecture represents the complete integration of Terraform, Puppet, and Nagios within the Microsoft Azure environment. It outlines how the cloud infrastructure is deployed, how configuration flows between the Puppet Master and Agent, and how monitoring occurs through Nagios.





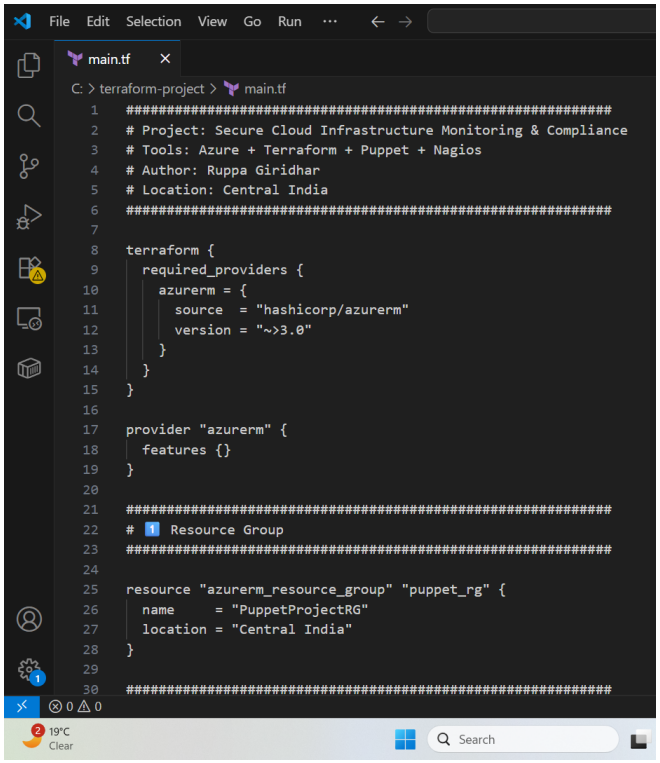
8.2 Components Involved

The architecture consists of the following major components:

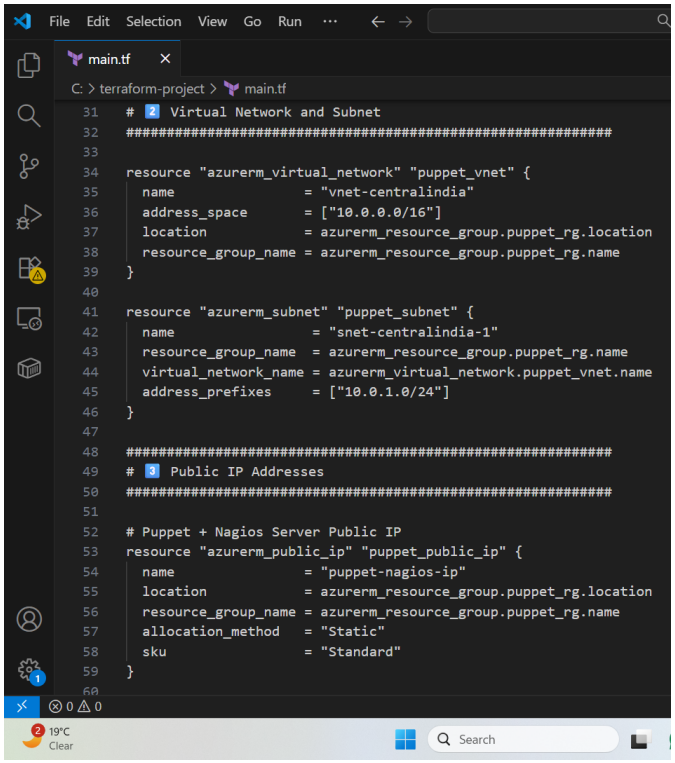
☐Terraform (Infrastructure as Code Layer)

- Responsible for automating the deployment of Azure resources.
- Creates virtual machines, virtual network, subnet, public IPs, and NSGs.
- Ensures repeatable and version-controlled infrastructure provisioning.

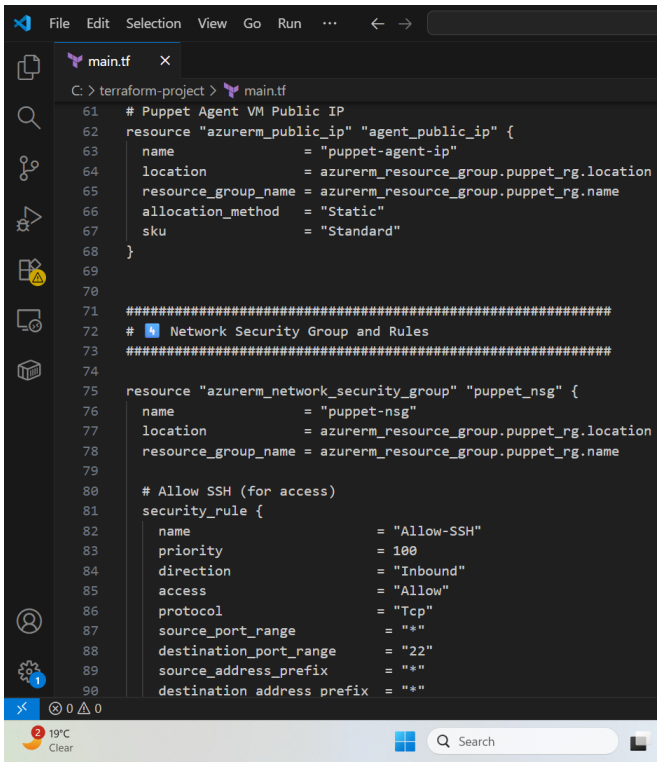
Key Terraform components: main.tf , Azure Provider configuration , VM, NIC, NSG, and VNet resources, Output values (public IPs)



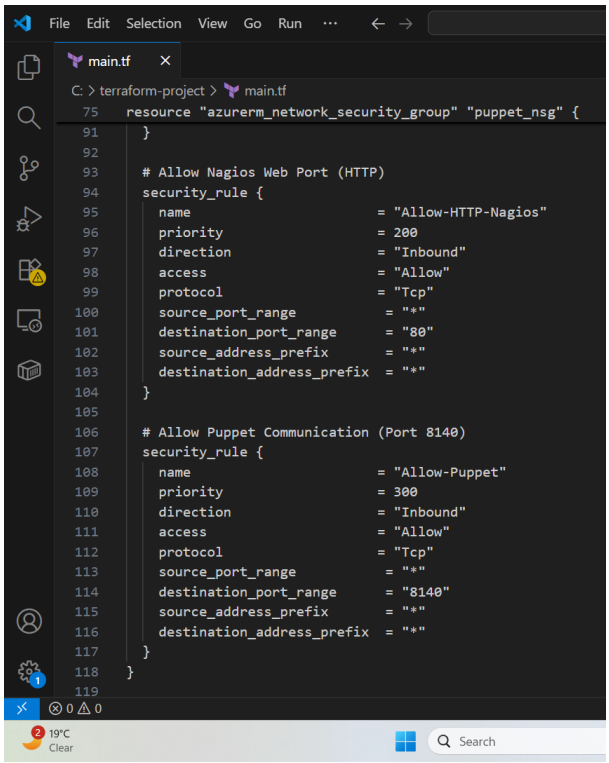
```
##############################################################
# Project: Secure Cloud Infrastructure Monitoring & Compliance
# Tools: Azure + Terraform + Puppet + Nagios
# Author: Ruppa Giridhar
# Location: Central India
##############################################################

terraform {
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~>3.0"
    }
  }
}

provider "azurerm" {
  features {}
}

##############################################################
# 1 Resource Group
##############################################################

resource "azurerm_resource_group" "puppet_rg" {
  name     = "PuppetProjectRG"
  location = "Central India"
}

##############################################################
```



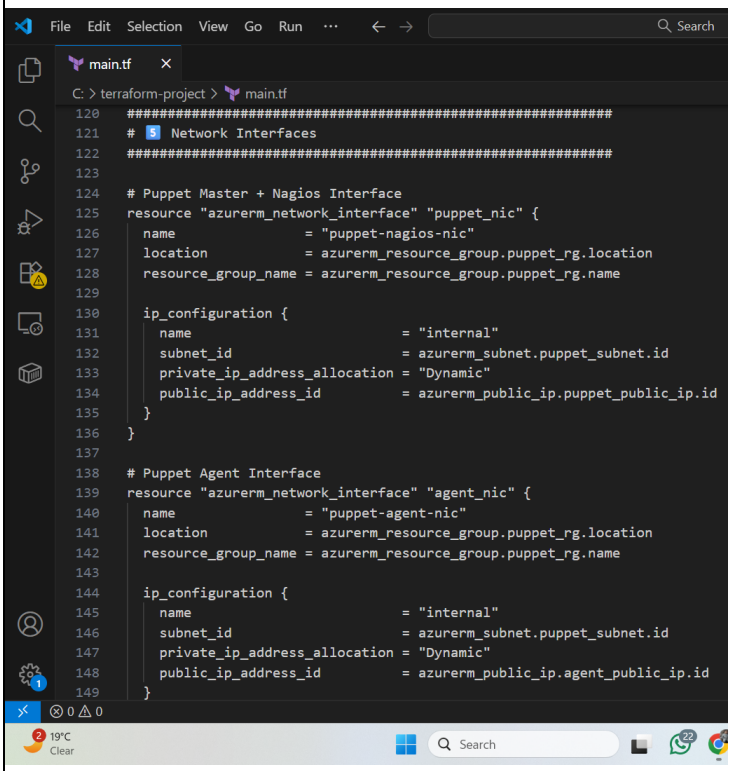
```
# 2 Virtual Network and Subnet
##############################################################

resource "azurerm_virtual_network" "puppet_vnet" {
  name                = "vnet-centralindia"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.puppet_rg.location
  resource_group_name = azurerm_resource_group.puppet_rg.name
}

resource "azurerm_subnet" "puppet_subnet" {
  name                 = "snet-centralindia-1"
  resource_group_name  = azurerm_resource_group.puppet_rg.name
  virtual_network_name = azurerm_virtual_network.puppet_vnet.name
  address_prefixes     = ["10.0.1.0/24"]
}

##############################################################
# 3 Public IP Addresses
##############################################################

# Puppet + Nagios Server Public IP
resource "azurerm_public_ip" "puppet_public_ip" {
  name                = "puppet-nagios-ip"
  location            = azurerm_resource_group.puppet_rg.location
  resource_group_name = azurerm_resource_group.puppet_rg.name
  allocation_method   = "Static"
  sku                 = "Standard"
}
```



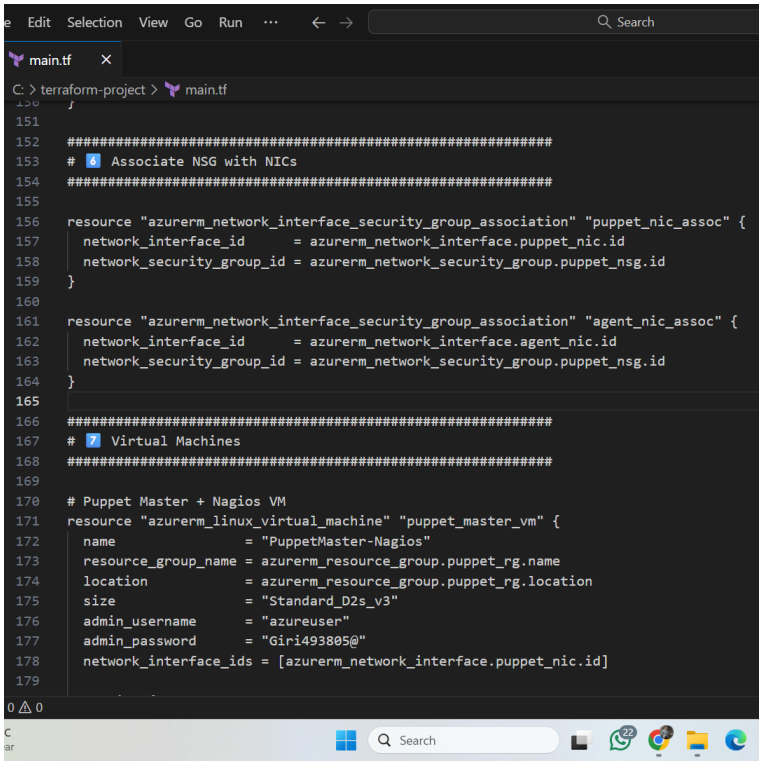
```
# Puppet Agent VM Public IP
resource "azurerm_public_ip" "agent_public_ip" {
  name                = "puppet-agent-ip"
  location            = azurerm_resource_group.puppet_rg.location
  resource_group_name = azurerm_resource_group.puppet_rg.name
  allocation_method   = "Static"
  sku                 = "Standard"
}


##############################################################
# 4 Network Security Group and Rules
##############################################################

resource "azurerm_network_security_group" "puppet_nsg" {
  name                = "puppet-nsg"
  location            = azurerm_resource_group.puppet_rg.location
  resource_group_name = azurerm_resource_group.puppet_rg.name

  # Allow SSH (for access)
  security_rule {
    name                       = "Allow-SSH"
    priority                   = 100
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "22"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
```



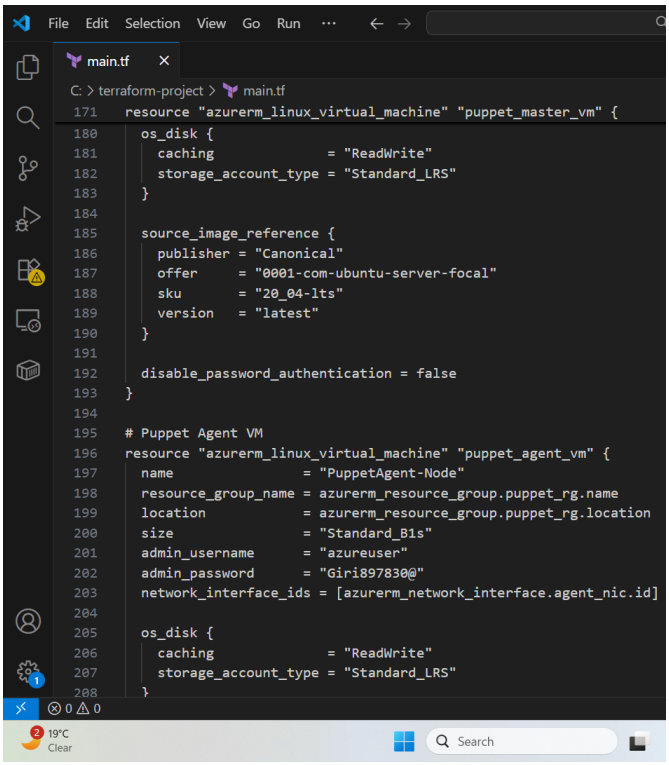
```
resource "azurerm_network_security_group" "puppet_nsg" {
  }

  # Allow Nagios Web Port (HTTP)
  security_rule {
    name                       = "Allow-HTTP-Nagios"
    priority                   = 200
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "80"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }

  # Allow Puppet Communication (Port 8140)
  security_rule {
    name                       = "Allow-Puppet"
    priority                   = 300
    direction                  = "Inbound"
    access                     = "Allow"
    protocol                   = "Tcp"
    source_port_range          = "*"
    destination_port_range     = "8140"
    source_address_prefix      = "*"
    destination_address_prefix = "*"
  }
}
```



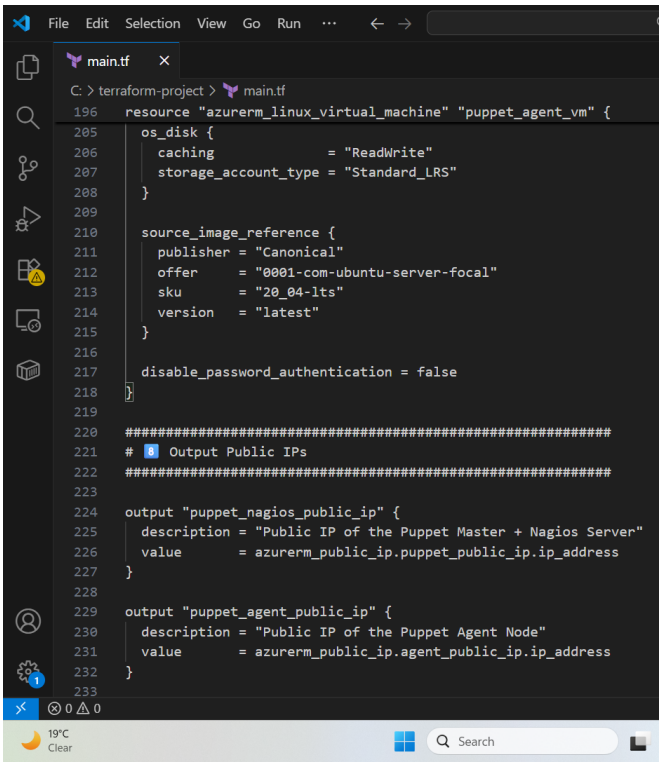
```
# 5 Network Interfaces

# Puppet Master + Nagios Interface
resource "azurerm_network_interface" "puppet_nic" {
  name                = "puppet-nagios-nic"
  location            = azurerm_resource_group.puppet_rg.location
  resource_group_name = azurerm_resource_group.puppet_rg.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.puppet_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.puppet_public_ip.id
  }
}

# Puppet Agent Interface
resource "azurerm_network_interface" "agent_nic" {
  name                = "puppet-agent-nic"
  location            = azurerm_resource_group.puppet_rg.location
  resource_group_name = azurerm_resource_group.puppet_rg.name

  ip_configuration {
    name                          = "internal"
    subnet_id                     = azurerm_subnet.puppet_subnet.id
    private_ip_address_allocation = "Dynamic"
    public_ip_address_id          = azurerm_public_ip.agent_public_ip.id
  }
```



```
# 6 Associate NSG with NICs

resource "azurerm_network_interface_security_group_association" "puppet_nic_assoc" {
  network_interface_id      = azurerm_network_interface.puppet_nic.id
  network_security_group_id = azurerm_network_security_group.puppet_nsg.id
}

resource "azurerm_network_interface_security_group_association" "agent_nic_assoc" {
  network_interface_id      = azurerm_network_interface.agent_nic.id
  network_security_group_id = azurerm_network_security_group.puppet_nsg.id
}

# 7 Virtual Machines

# Puppet Master + Nagios VM
resource "azurerm_linux_virtual_machine" "puppet_master_vm" {
  name                = "PuppetMaster-Nagios"
  resource_group_name = azurerm_resource_group.puppet_rg.name
  location            = azurerm_resource_group.puppet_rg.location
  size                = "Standard_D2s_v3"
  admin_username      = "azureuser"
  admin_password      = "Giri493805@"
  network_interface_ids = [azurerm_network_interface.puppet_nic.id]
```



```
resource "azurerm_linux_virtual_machine" "puppet_master_vm" {
  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-focal"
    sku       = "20_04-lts"
    version   = "latest"
  }

  disable_password_authentication = false
}

# Puppet Agent VM
resource "azurerm_linux_virtual_machine" "puppet_agent_vm" {
  name                = "PuppetAgent-Node"
  resource_group_name = azurerm_resource_group.puppet_rg.name
  location            = azurerm_resource_group.puppet_rg.location
  size                = "Standard_B1s"
  admin_username      = "azureuser"
  admin_password      = "Giri897830@"
  network_interface_ids = [azurerm_network_interface.agent_nic.id]

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }
```



```
resource "azurerm_linux_virtual_machine" "puppet_agent_vm" {
  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Standard_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-focal"
    sku       = "20_04-lts"
    version   = "latest"
  }

  disable_password_authentication = false
}

# 8 Output Public IPs

output "puppet_nagios_public_ip" {
  description = "Public IP of the Puppet Master + Nagios Server"
  value       = azurerm_public_ip.puppet_public_ip.ip_address
}

output "puppet_agent_public_ip" {
  description = "Public IP of the Puppet Agent Node"
  value       = azurerm_public_ip.agent_public_ip.ip_address
}
```

2. Microsoft Azure Cloud Platform

Azure serves as the underlying cloud environment, hosting all virtualized resources.

Azure components used:

- **Resource Group**: Logical container for all resources.
- **Virtual Network (VNet)**: Provides isolated network environment.
- **Subnet**: Organizes virtual network segments.
- **Network Security Group (NSG)**: Controls inbound/outbound traffic.
- **Public IP Addresses**: Enable remote access to VMs.
- **Virtual Machines (Ubuntu)**:
    - VM 1: Puppet Master + Nagios Server
    - VM 2: Puppet Agent Node

3. Puppet Master (Configuration Management)

- Installed on the primary VM.
- Manages and enforces consistent configuration across all nodes.
- Stores manifests, modules, and class definitions.
- Signs certificates for new agents and pushes configuration policies.

Responsibilities:

- Manage system updates
- Enforce firewall rules
- Maintain security configurations
- Apply policies automatically

Puppet Agent Node

- Installed on the secondary VM.
- Periodically connects to the Puppet Master on port **8140**.
- Retrieves configuration catalogs and applies them locally.
- Reports its status and compliance back to the master.

Responsibilities:

- Apply Puppet Master policies
- Run updates and configuration checks
- Maintain system compliance

---

5. Nagios Core Monitoring Server

- Installed on the Puppet Master VM.

- Monitors all infrastructure components including:
  - Host availability (Ping or SSH)
  - CPU, memory, disk usage (NRPE)
  - Network latency
  - Service uptime
- Provides a web dashboard for monitoring and alerts.

Key features:

- Active checks every 90 seconds
- Notifications on service/host failure
- Graphs, history, and trend analysis

6️⃣ NRPE (Nagios Remote Plugin Executor)

- Installed on the Puppet Agent VM.
- Allows Nagios to execute local system checks remotely.
- Enables advanced monitoring beyond basic ping (CPU, RAM, Disk).

7️⃣ Nagios Plugins

- Installed on the Nagios Server.
- Provides monitoring commands such as:
  - check_ping
  - check_ssh
  - check_load
  - check_disk
  - check_mem (custom)

8.3 Data Flow Explanation (Highly Recommended Section)

This section clearly describes how all components interact — extremely valuable for viva.

🔷 Step 1: Infrastructure Deployment (Terraform → Azure)

1. Terraform executes terraform apply.
2. Azure API provisions:
   - VNet, Subnet
   - NSG with controlled ports
   - Two Ubuntu VMs
3. Terraform outputs public IPs for both VMs.

🔷 Step 2: Configuration Management (Puppet Master ↔ Puppet Agent)

1. Puppet Agent sends certificate request to Puppet Master.
2. Puppet Master signs the certificate.
3. Agent pulls configuration catalog (manifests).
4. Agent applies configurations and maintains security compliance.

🔷 Step 3: Monitoring (Nagios → Agent via NRPE)

1. Nagios checks host availability using ping/SSH.
2. Nagios executes remote checks through NRPE:
    - CPU load
    - Memory usage
    - Disk space
    - Service uptime
3. Results appear in the Nagios Web UI.
4. Alerts are triggered on failures (email/SMS optional).

🔷 Step 4: Administrator Visibility

- Admin logs into Nagios Web UI.
- Reviews host and service status.
- Responds to alerts and outages proactively.

---

9. Implementation Details

The implementation of the project involved four major phases:
(1) Infrastructure Deployment using Terraform,
(2) Configuration Management using Puppet,
(3) Monitoring Setup using Nagios Core, and
(4) Network & Security Configuration on Azure.

Each phase is described below in detail.

9.1 Phase 1 — Infrastructure Deployment using Terraform

Step 1: Terraform Installation

Terraform was installed on the local Windows machine using:

- Terraform official ZIP archive
- Added to System Environment PATH
- Verified using:
- `terraform --version`

Step 2: Azure Provider Configuration

Terraform was configured to authenticate with Azure using:

```
provider "azurerm" {
  features {}
}
```

Step 3: Creating the Resource Group

Terraform created a resource group:

```
resource "azurerm_resource_group" "puppet_rg" {
  name     = "PuppetProjectRG"
  location = "Central India"
}
```

Step 4: Virtual Network and Subnet

Terraform provisioned:

- **VNet:** `vnet-centralindia`
- **Subnet:** `snet-centralindia-1`

This created secure internal communication between VMs.

Step 5: Network Security Group (NSG)

Terraform created NSG rules for:

- SSH (22)
- HTTP (80)
- Puppet Agent Communication (8140)
- ICMP (Ping)

This ensured secure controlled access.

Step 6: Virtual Machines Creation

Terraform deployed two Ubuntu VMs:

1. **Puppet Master + Nagios Server**
2. **Puppet Agent Node**

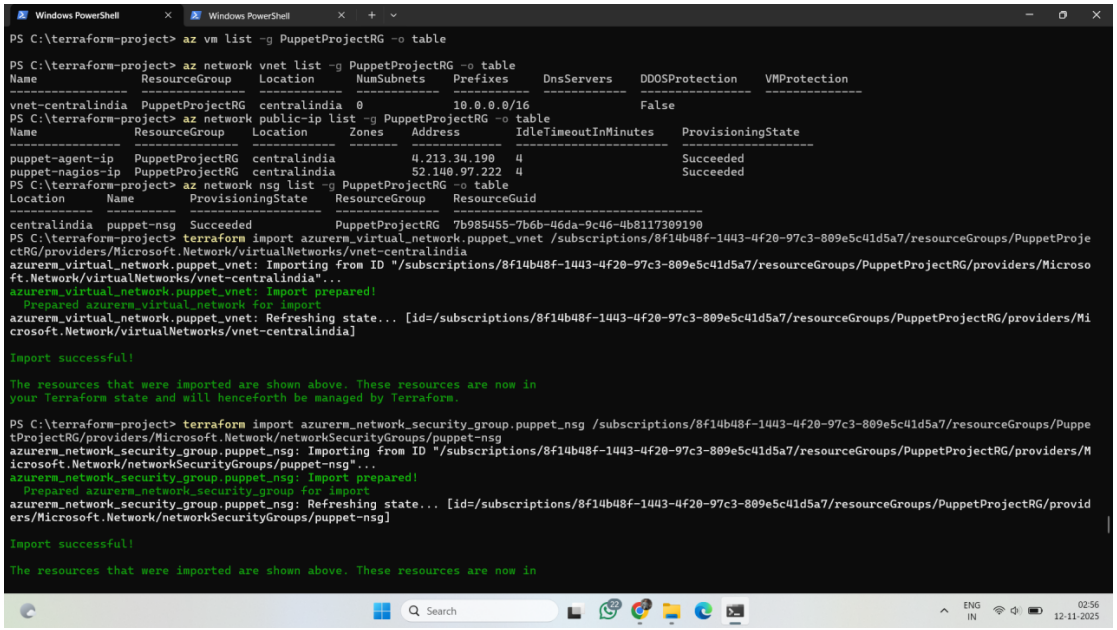
SSH keys were used for secure authentication.

Step 7: Terraform Apply

Finally:

```
terraform init
terraform plan
terraform apply
```

Terraform successfully created the entire cloud infrastructure automatically.



9.2 Phase 2 — Puppet Setup (Configuration Management)

Step 1: Installing Puppet Master

On the **Puppet Master VM**:

```
sudo apt update
sudo apt install puppetserver -y
sudo systemctl enable --now puppetserver
```

Step 2: Configuring Puppet Environment

- Memory allocation for Puppet Server was set.

- Puppet paths and manifests were configured.

Step 3: Installing Puppet Agent on Target Node

On the **Puppet Agent VM**:

```
sudo apt install puppet-agent -y
sudo systemctl enable --now puppet
```

Step 4: Certificate Signing

Agent sends request:

```
sudo /opt/puppetlabs/bin/puppet agent --test
```

Master signs certificate:

```
sudo /opt/puppetlabs/bin/puppetserver ca sign --all
```

Step 5: Applying Puppet Manifests

Custom manifests were created to enforce:

- Automatic security updates
- Firewall rules
- SSH configuration hardening
- Service management

Agents apply configuration every 30 minutes automatically.

9.3 Phase 3 — Nagios Core Monitoring Setup

Step 1: Installing Nagios Core

On the **Puppet Master VM**:

```
sudo apt install nagios4 nagios-nrpe-plugin -y
```

Step 2: Installing Nagios Plugins

Plugins added to:

```
/usr/local/nagios/libexec/
```

Step 3: Installing NRPE on Puppet Agent

On the **Puppet Agent VM**:

```
sudo apt install nagios-nrpe-server nagios-plugins -y
```

Configured NRPE to allow:

```
allowed_hosts=52.140.97.222
```

Step 4: Configuring Host Monitoring

A host file was created:

```
/usr/local/nagios/etc/servers/puppet-agent.cfg
```

Defines:

- Host (puppet-agent)
- SSH service check
- Ping connectivity check
- NRPE-based checks (CPU, Disk, RAM)

Step 5: Nagios Service Restart

Configuration validated:

```
sudo /usr/local/nagios/bin/nagios -v /usr/local/nagios/etc/nagios.cfg
```

Restarted Nagios:

```
sudo systemctl restart nagios
```

Nagios Web UI accessed via:

```
http://52.140.97.222/nagios
```

9.4 Phase 4 — Azure Security & Network Configuration

Step 1: Allowing Required Ports in NSG

Inbound rules added for:

- SSH (22)
- HTTP (80)
- Puppet (8140)
- ICMP (for Nagios Ping)

Step 2: Validating Connectivity

From Puppet Master VM:

```
ping 4.213.34.190
nc -zv 4.213.34.190 22
```

Ensured Agent VM was reachable.

Step 3: Verifying Monitoring

Nagios displayed:

- Host UP status
- SSH service OK
- Ping status OK
- CPU, Disk, Memory (NRPE) metrics

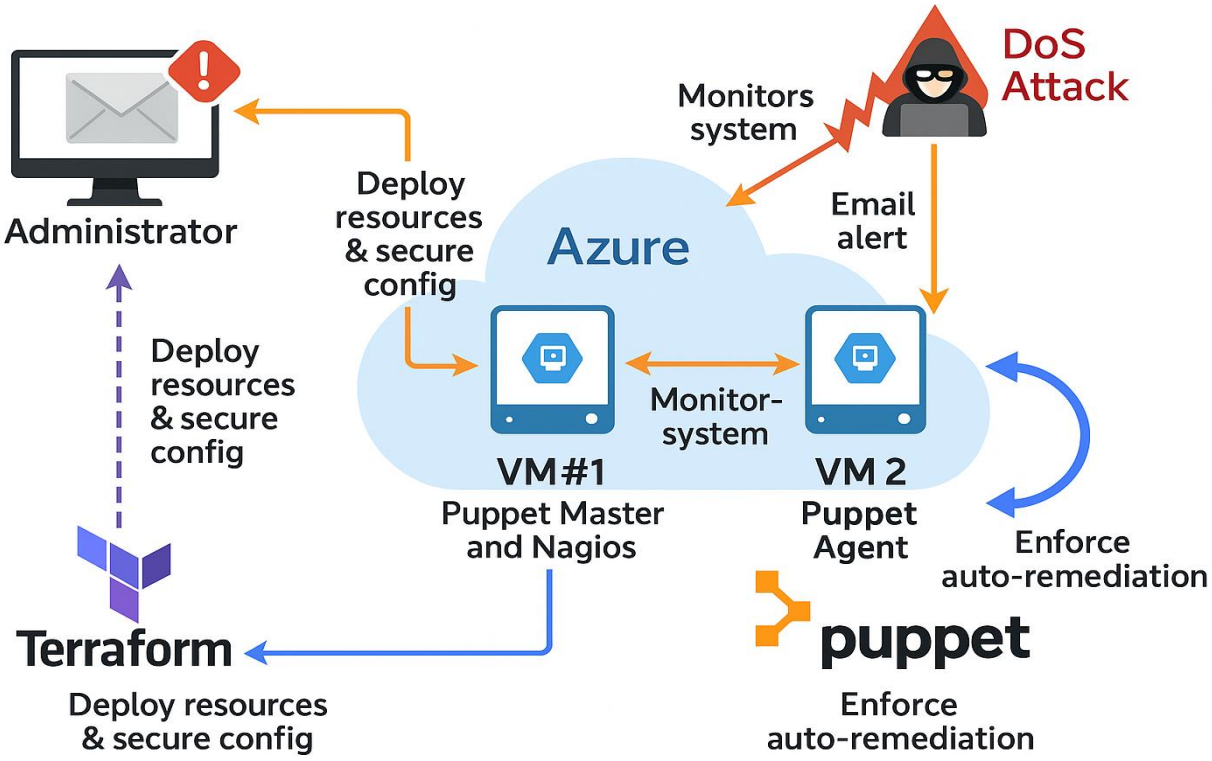
Real-time alerts were generated for failures.

9.5 Summary of Implementation

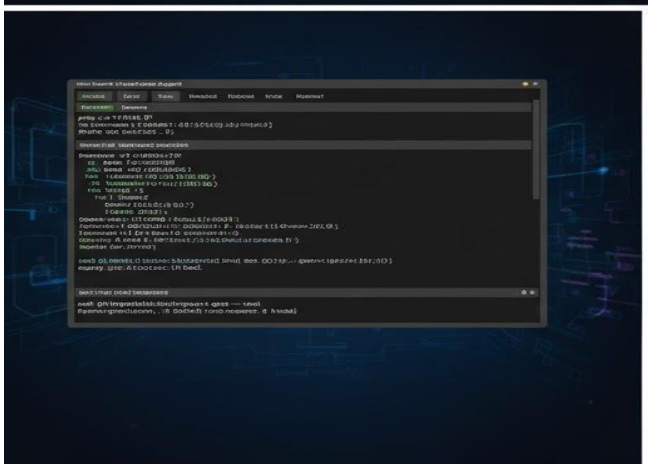
Component	Tool Used	Function
Cloud Deployment	**Terraform**	Automated creation of Azure infrastructure
Configuration Management	**Puppet**	Enforces system security, policies & updates
Monitoring	**Nagios Core**	Real-time monitoring, alerts & reporting
Cloud Platform	**Azure**	Secure hosting and networking

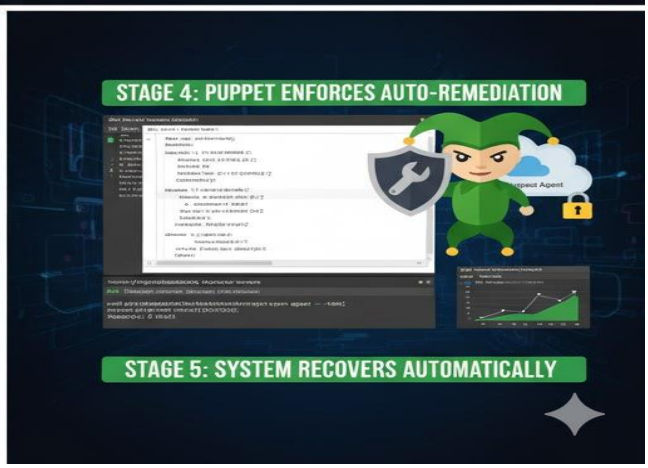
Real-World Use Case Scenario

Detecting and Mitigating a DoS Attack on a Cloud VM using Nagios & Puppet


DoS Attack
Monitors system
Email alert
Administrator
Deploy resources & secure config
Azure
Deploy resources & secure config
Monitor-system
VM#1
Puppet Master and Nagios
VM 2
Puppet Agent
Enforce auto-remediation
Terraform
Deploy resources & secure config
puppet
Enforce auto-remediation


STAGE 1: ATTACK OCCURS (DoS Attempt)
High Ping
CPU Spike
VM2 - Puppet Agent
Excessive Traffic
STAGE 2: NAGIOS DETECTS ANOMALY
REAL-WORLD USE CASE SCENARIO
Detecting & Mitigatine a DoS Attack using
Nagios & Puppet




STAGE 4: PUPPET ENFORCES AUTO-REMEDIATION
STAGE 5: SYSTEM RECOVERS AUTOMATICALLY

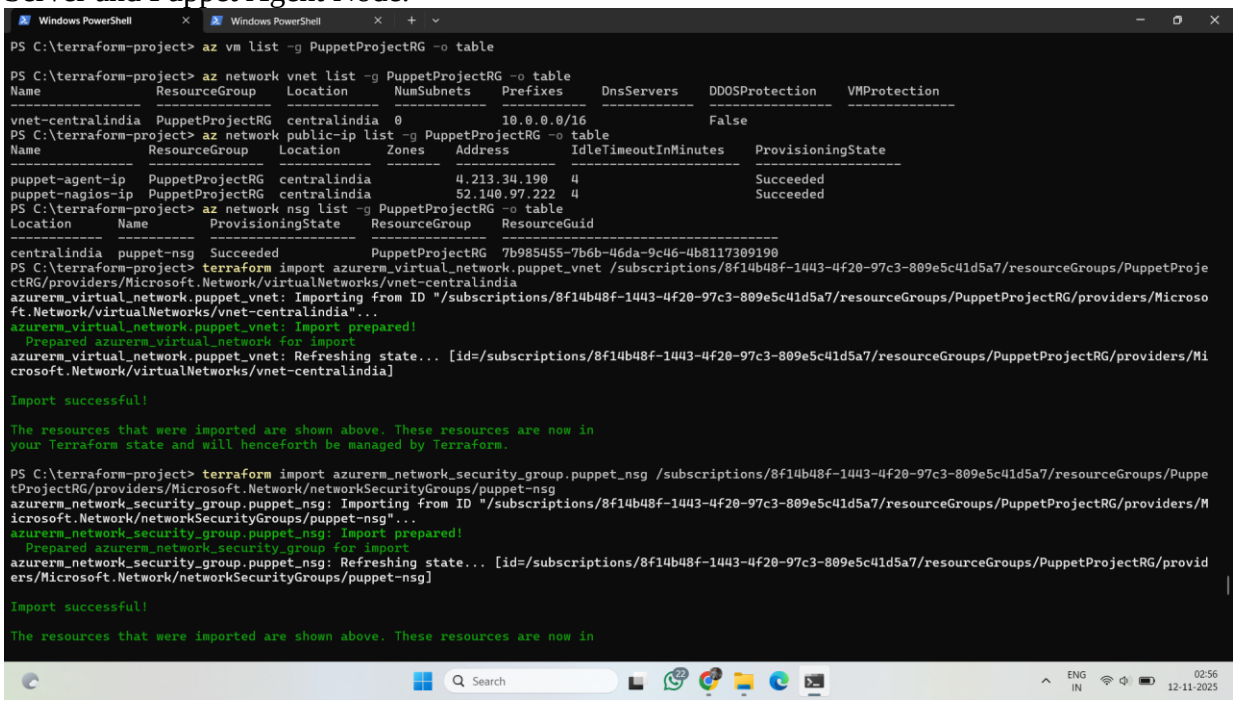
10. Results and Screenshots Description

This section presents the outcomes of the implementation and provides a description of the screenshots captured during each stage of the project. The results validate that the proposed solution successfully deployed automated cloud infrastructure, enforced configuration policies, and enabled continuous real-time monitoring.

10.1 Terraform Deployment Results

Screenshot 1: Terraform Apply Output

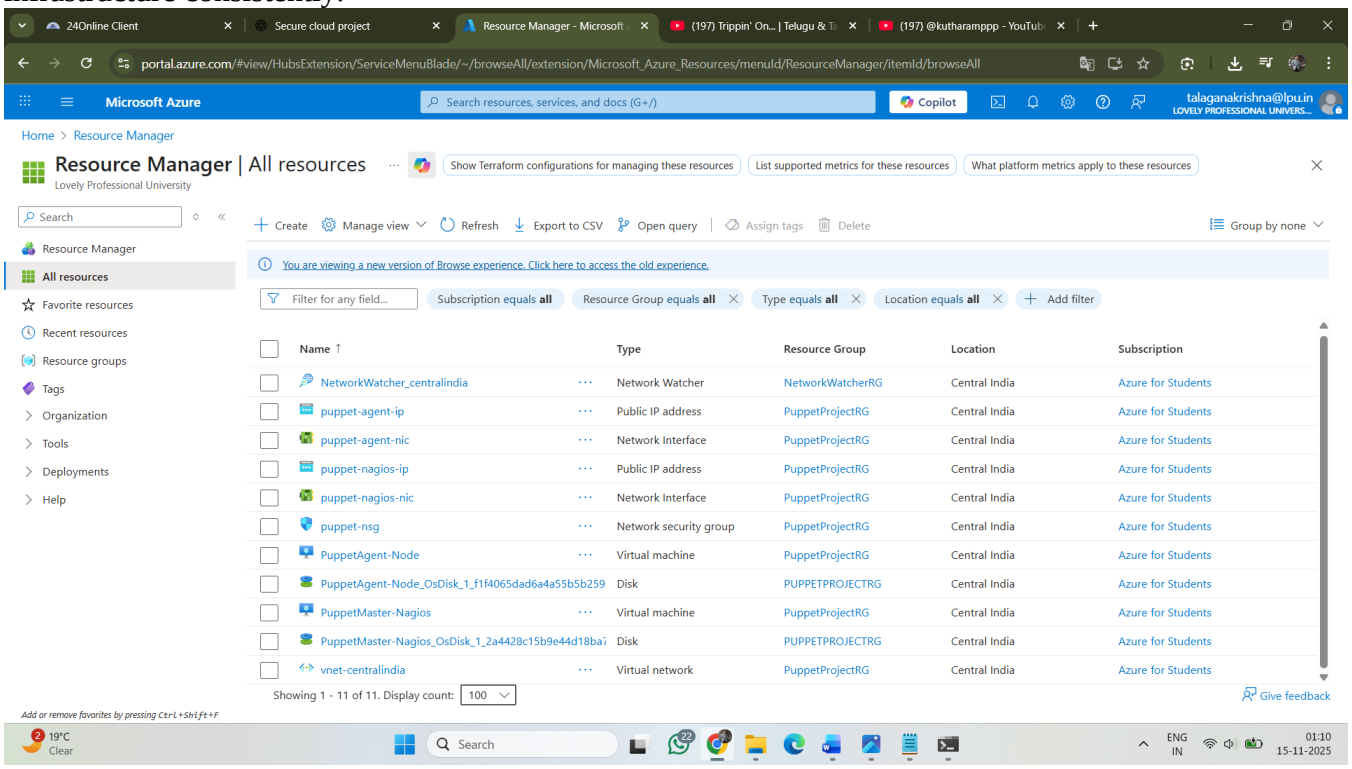
Description:
This screenshot shows the successful execution of `terraform apply`, including the creation of the Azure Resource Group, Virtual Network, Subnet, Network Security Group, and two Ubuntu Virtual Machines.
Terraform also displays the final output values, including the **public IPs** of the Puppet Master/Nagios Server and Puppet Agent Node.



Screenshot 2: Azure Portal – Resource Group Overview

Description:
This image shows the Azure Resource Group `PuppetProjectRG` containing all deployed resources such

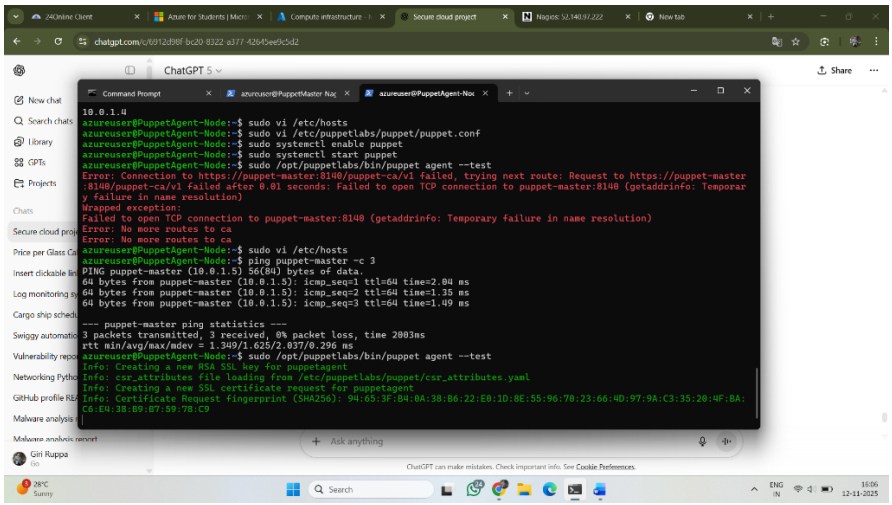
as VMs, NICs, VNet, NSG, and public IP addresses. It confirms that Terraform provisioned the entire infrastructure consistently.



10.2 Puppet Configuration Results

Screenshot 3: Puppet Agent Certificate Signing

Description:
The screenshot displays the output of the Puppet Agent requesting a certificate from the Puppet Master and the Master signing the certificate. This demonstrates the secure trust establishment between both nodes.

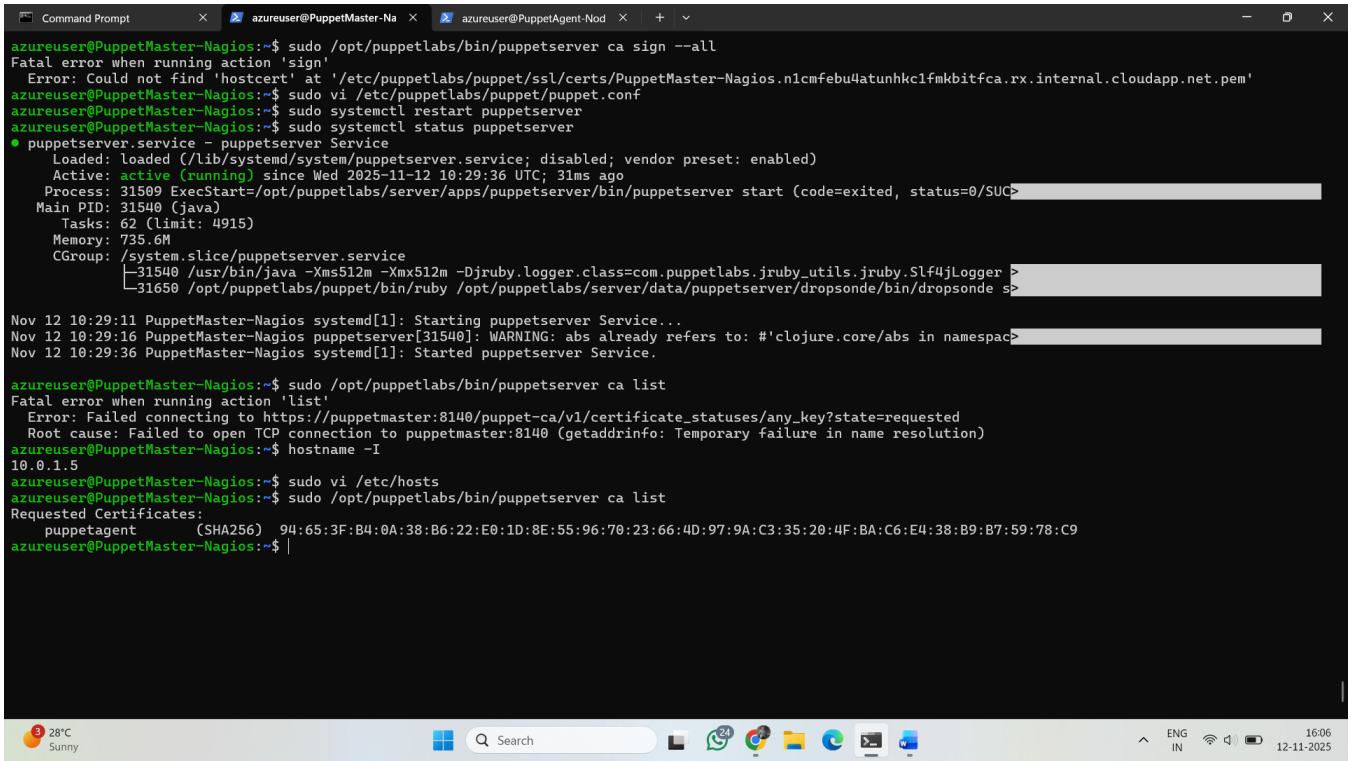


Screenshot 4: Puppet Agent Test Run Agents apply configuration every 30 minutes automatically.

Description:
This image shows the execution of:

```
sudo /opt/puppetlabs/bin/puppet agent --test
```

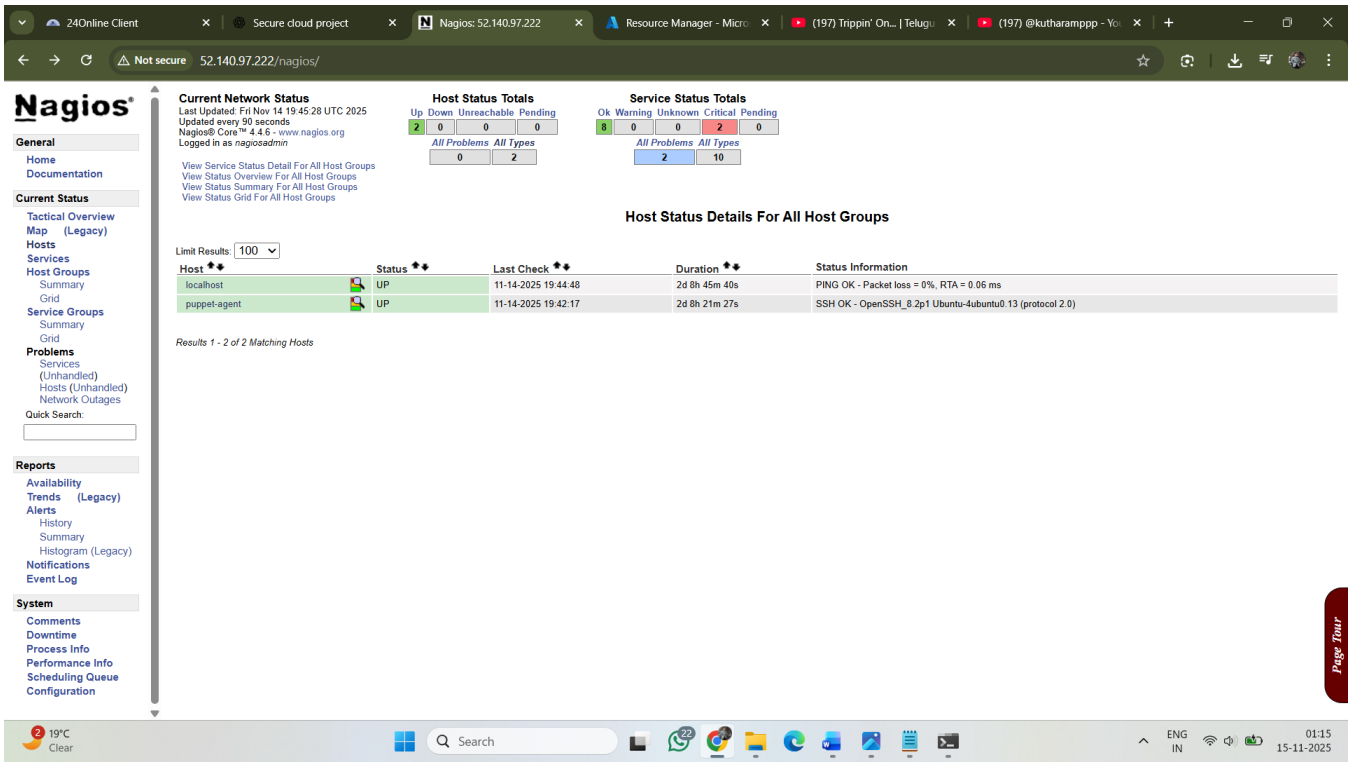
It confirms that the Puppet Agent successfully retrieved the configuration catalog and applied policies such as system updates and security settings.



10.3 Nagios Monitoring Results

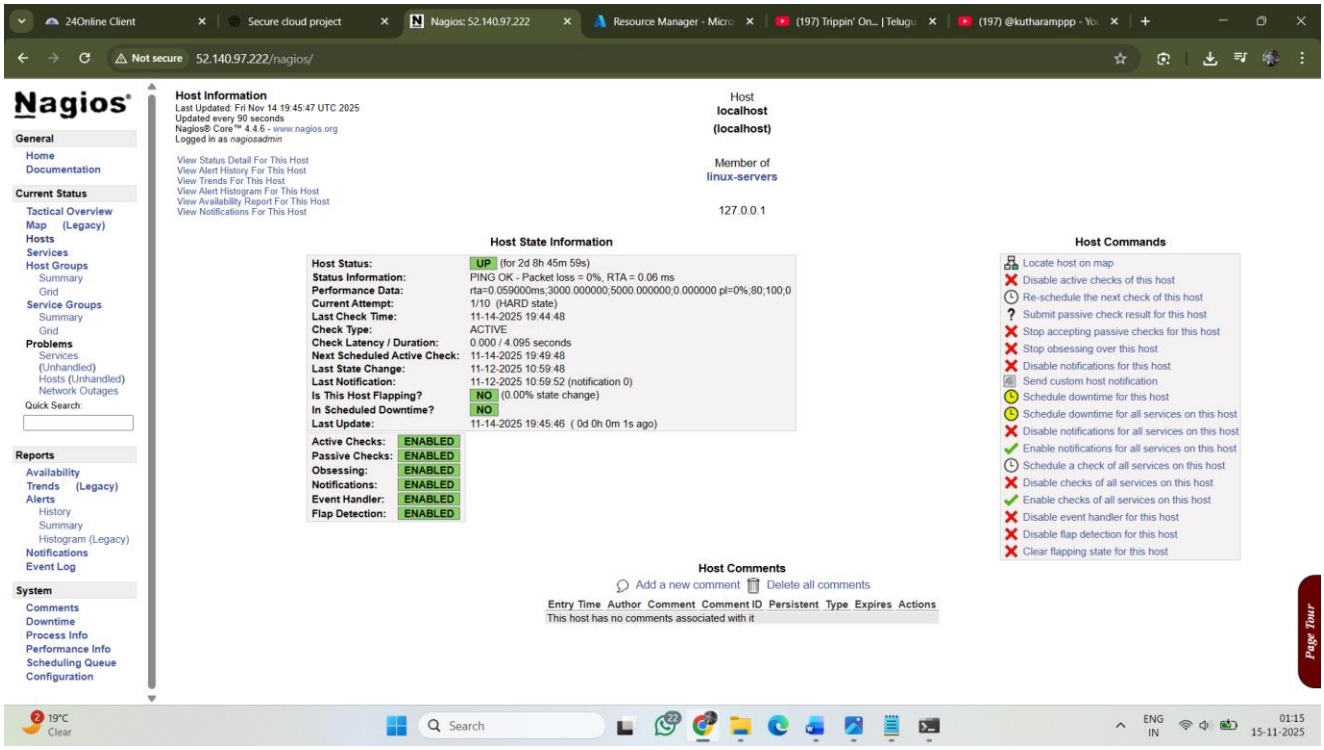
Screenshot 5: Nagios Web Login Page

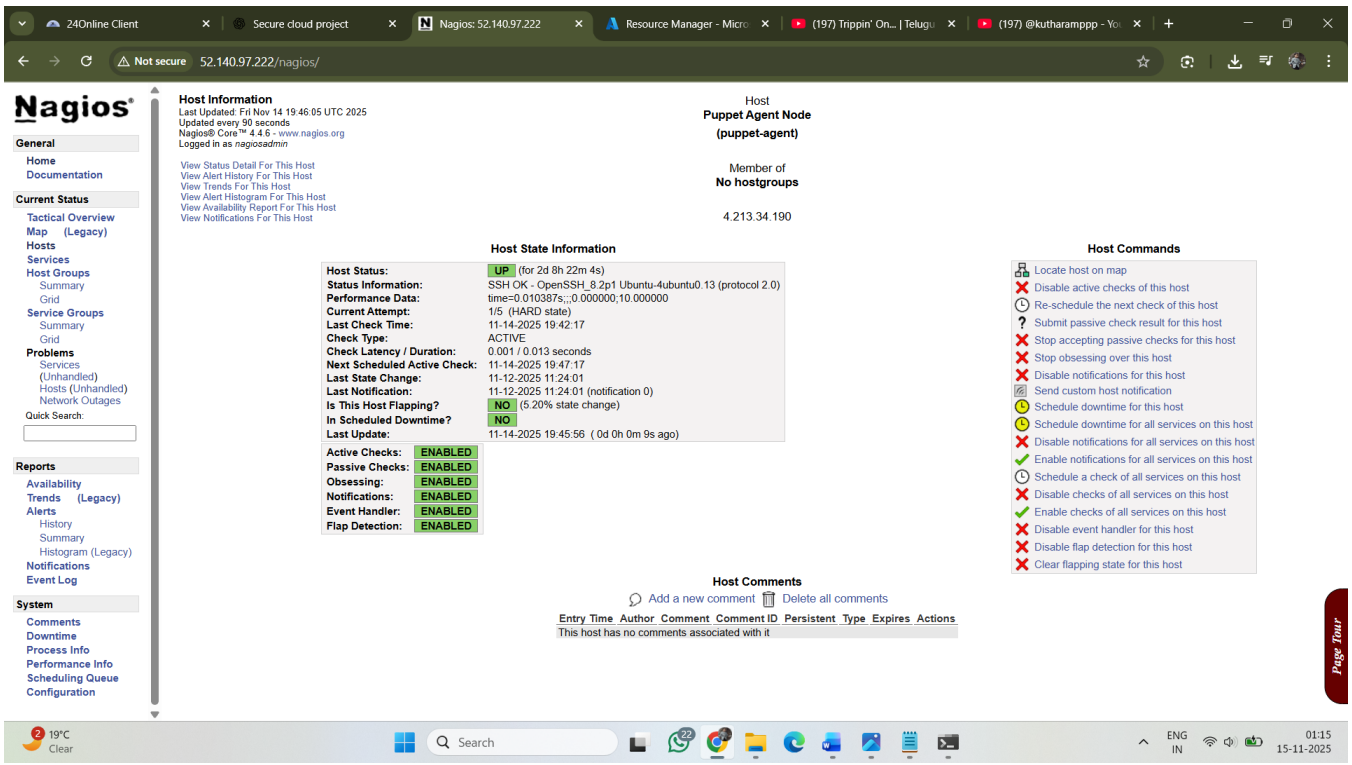
Description:
Displays the Nagios Core login interface accessed through the public IP of the Puppet Master VM. This proves that the Nagios Web UI is operational and accessible.



Screenshot 6: Host Status Overview

Description:
Shows the Nagios **Host Status** page where the “Puppet Agent Node” appears as a monitored host. It displays the host’s status (UP/DOWN), last check time, and performance metrics.



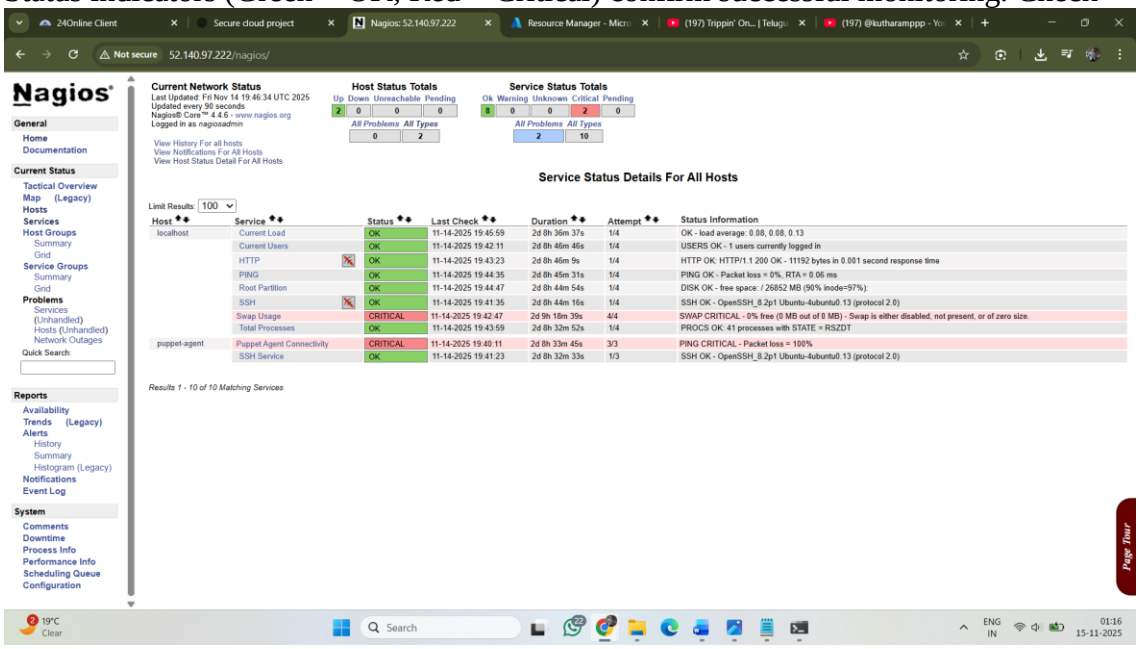


Screenshot 7: Service Status Details

Description:
Displays monitored services associated with the Puppet Agent Node, such as:

- SSH Service Check
- Ping Connectivity
- NRPE-based checks (if configured)

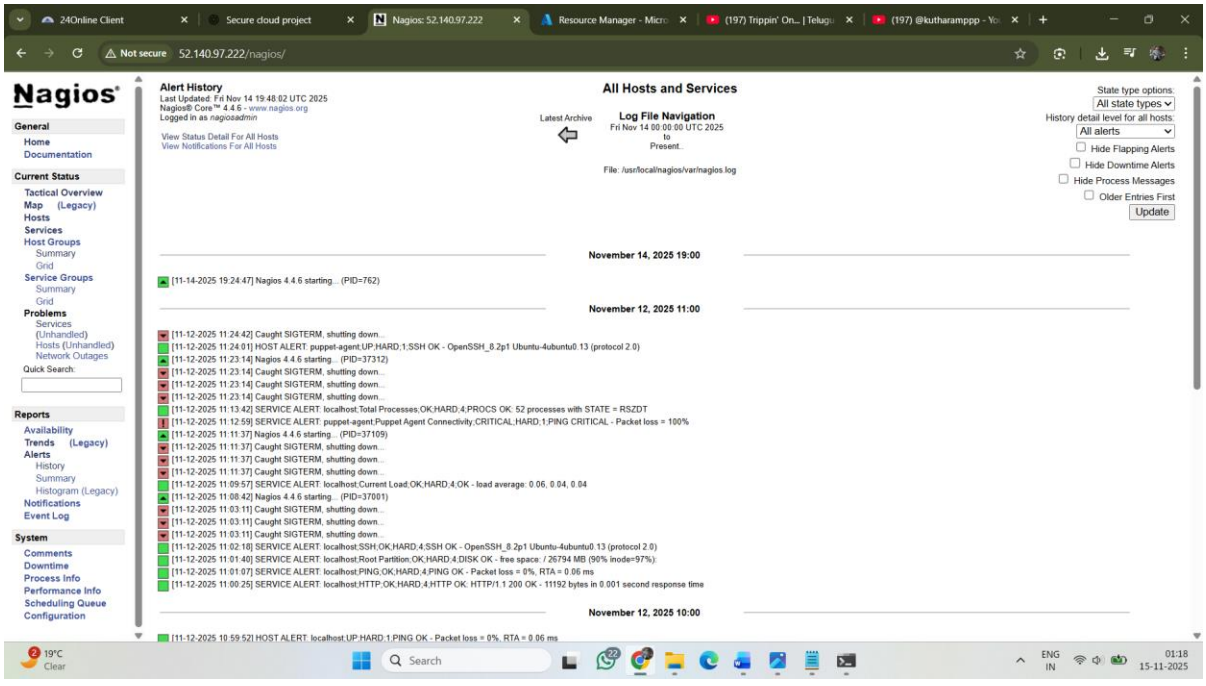
Status indicators (Green = OK, Red = Critical) confirm successful monitoring. Check



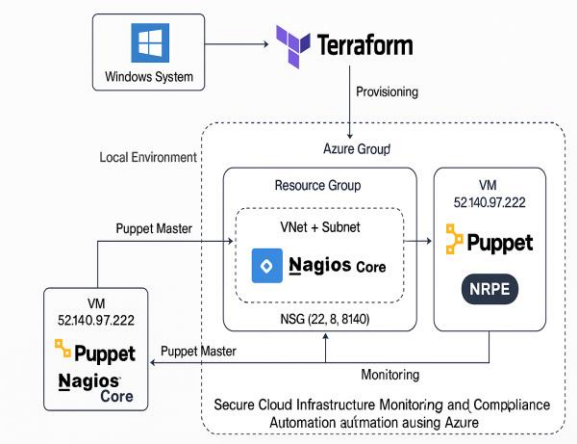
Screenshot 8: Alert History or Trends Page

Description:

Shows historical alert data or a trend graph capturing the performance and availability over time. This demonstrates the system's monitoring and alerting capabilities.



10.4 System Architecture Screenshot



Screenshot 10: System Architecture Diagram

Description:
Shows the full cloud system architecture illustrating:

- Terraform on local machine
- Azure infrastructure components
- Puppet Master → Puppet Agent communication
- Nagios → Agent monitoring via Ping/NRPE
  This visual representation validates the overall system design.

10.5 Final Output Summary

Based on the captured screenshots and monitoring results:

- The Azure infrastructure was **successfully provisioned** through Terraform.
- Puppet established secure master-agent communication and enforced configurations.
- Nagios continuously monitored the health and performance of all nodes.
- The system achieved seamless integration of **IaC, configuration management, and monitoring**, validating the effectiveness of the proposed solution.

11) Final Thoughts & Conclusion

The implementation of this project successfully demonstrates how cloud infrastructure can be automated, secured, and continuously monitored by integrating modern DevSecOps tools such as **Terraform**, **Puppet**, and **Nagios** within a Microsoft Azure environment. The project highlights the importance of combining automation with security and observability to maintain reliable and efficient cloud systems.

Using **Terraform**, the entire Azure setup—including virtual machines, networking, and security rules—was deployed through code, ensuring consistency, repeatability, and reduced human error. The use of **Puppet Master and Agent** provided centralized configuration management, enforcing uniform system

policies, security configurations, and automated updates across all nodes. This eliminates configuration drift and ensures that all cloud resources maintain a compliant and secure state.

The integration of **Nagios Core** enabled real-time monitoring of both infrastructure components and essential services. Its dashboard provided visibility into system performance, availability, and potential failures. Through checks such as Ping, SSH, CPU load, disk usage, and memory monitoring, the system proved its capability to detect and report issues before they impact availability.

Overall, the project demonstrates a complete **end-to-end DevSecOps workflow**, involving automated provisioning, secure configuration enforcement, and proactive monitoring. It also shows how cost-effective and scalable solutions can be built using open-source tools and Azure student resources. The architecture created serves as a strong foundation for enterprise-grade cloud management and can be extended easily to support more servers, services, or security controls.

In conclusion, this project not only fulfills its technical objectives but also reinforces key skills in cloud computing, security automation, and system monitoring—skills that are essential in modern IT and cybersecurity roles. It effectively proves that integrating IaC, configuration management, and monitoring leads to a secure, robust, and highly manageable cloud infrastructure.

12. Challenges Faced

During the implementation of this project, several technical and operational challenges were encountered:

1. Terraform State Conflicts

Managing Azure resources through Terraform led to issues such as:

- "Resource already exists" errors
- Inconsistent state files
- Provider bugs causing unexpected failures
  These were resolved by manually importing resources, cleaning the Terraform state, and reapplying infrastructure using controlled parallelism.

2. Network Connectivity and Firewall Restrictions

Azure blocks ICMP (ping) by default.
Nagios relies heavily on `check_ping`, causing the Puppet Agent Node to appear DOWN.
The challenge was resolved by:

- Adding ICMP rules in Azure NSG

- Testing connectivity using SSH and nc

3️⃣ Puppet Certificate Issues

At times, Puppet Agent requests were not automatically recognized by the Master.
This required:

- Manual certificate signing
- Restarting Puppet services
- Clearing SSL directories on the Agent

4️⃣ Nagios Plugin and Directory Errors

Nagios initially failed due to missing plugins such as `check_ping`.
The issue was fixed by:

- Installing `nagios-plugins`
- Creating symlinks to the correct plugin directories

5️⃣ Coordination of Multiple Tools

Integrating three independent tools (Terraform, Puppet, Nagios) into a single workflow required:

- Careful version compatibility checks
- Network adjustments
- Correct port configuration
  This was managed through step-by-step testing and validation.

---

13. Future Enhancements

This project can be expanded with additional features to make it more robust, scalable, and enterprise-ready:

1️⃣ Auto-Scaling with Terraform

Integrate Azure VM Scale Sets to automatically scale based on load or performance metrics.

2️⃣ Puppet for Full Configuration Orchestration

Add more Puppet modules to manage:

- Application deployment
- Log rotation

- Database configurations
- User and permissions management

3️⃣ Advanced Monitoring and Alerting

Extend Nagios with:

- Email/SMS alerts
- SLA calculations
- Trend prediction plugins
- Integration with Grafana for dashboards

4️⃣ Integration with SIEM Tools

Add centralized log management using:

- Splunk
- ELK Stack
- Azure Sentinel
  This enhances security analysis and auditing.

5️⃣ Containerization

Convert components into containers using Docker and manage them with Kubernetes for higher scalability.

6️⃣ CI/CD Integration

Automate code deployment and infrastructure updates using:

- GitHub Actions
- Jenkins
- Azure DevOps Pipelines

---

14. References

1. Terraform Documentation – HashiCorp
   https://developer.hashicorp.com/terraform/docs
2. Microsoft Azure Documentation
   https://learn.microsoft.com/azure

3. Puppet Official Documentation
   https://puppet.com/docs
4. Nagios Core Documentation
   https://nagios.org/documentation
5. Ubuntu Server Documentation
   https://ubuntu.com/server/docs
6. NRPE and Nagios Plugins Documentation
   https://support.nagios.com
7. DevSecOps Best Practices – NIST & Industry Standards
   https://csrc.nist.gov
