Terraform and Ansible to Create an Secure Self-Host Cloud Infrastructure

Felipe Bueno - a61491

João Guilherme Martins Silva - a61480

Leonardo Bazán - a61440

First practical work of SSI

Trabalho orientado por:

Prof. Thiago Pedrosa

Bragança

2024-2025

Contents

Chapter 1

Introduction

This work focuses on the development of security deployments for cloud environments and system hardening. Its primary objective is to configure a self-hosted cloud provider in a secure and efficient manner.

To achieve this, Infrastructure as Code (IaC) tools were employed, including Terraform for creating and provisioning virtual machines and Ansible playbooks for hardening and auditing these virtual machines. The implementation is designed to leverage Proxmox as the self-hosted cloud provider.

The workflow is divided into three main stages. The first stage involves configuring the local machine (self-hosted cloud) with Proxmox, a powerful and efficient open-source operating system based on Debian, designed to simplify server management. The second stage utilizes Terraform to create and manage the virtual machine infrastructure. Finally, the third stage focuses on hardening and auditing the virtual machines for secure and reliable delivery using Ansible.

Chapter 2

Development

2.1 Proxmox

The Proxmox Virtual Environment (PVE) is an open-source virtualization platform that enables the creation of VMs by combining container-based virtualization techniques (LXC) and full virtualization (KVM - Kernel-based Virtual Machine). The tool also provides an intuitive visual interface accessible through a web browser, which simplifies the management of virtualized environments [1].

The choice of Proxmox as the virtualization environment for VM management was based on factors such as the following: Integration, considering that Terraform has providers that enable seamless integration between both technologies; Simplified management, as previously mentioned, Proxmox offers a graphical interface that facilitates the management of virtual machines; and Robustness, as Proxmox is a robust platform with a wide range of functionalities.

2.1.1 Instalation

Proxmox is a virtualization tool that operates as a complete operating system based on Linux Debian, and its installation is typically performed on physical hardware [1]. However, due to the unavailability of a physical server, we opted for an alternative solution to simulate a physical environment. Thus, the tool was installed on a Virtual Machine, which allowed us to create a working environment for the project.

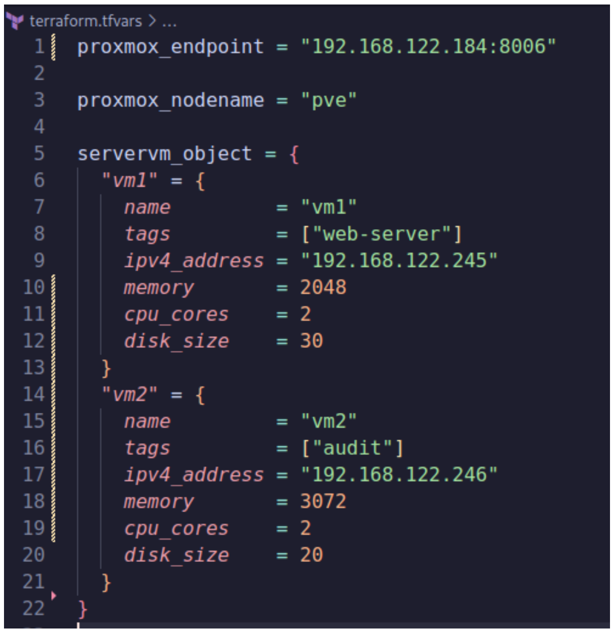
2.2 Terraform

Using Terraform to manage virtual machines on a Proxmox server is a great way to streamline and automate your infrastructure. By leveraging Terraform's Infrastructure as Code (IaC) capabilities, we can quickly deploy and manage VMs without the need for manual configuration, and integrating Ansible with Terraform code to harden self-hosted cloud VMs on a Proxmox server can significantly enhance our infrastructure's security.

Terraform Proxmox provider made by BPG was chosen to manage the infrastructure creation[2]. The main configuration file and provider declaration can be visualized at Appendix A.

To handle our infrastructure rising using Terraform, the approach used was using cloud-init configuration with operation system cloud images. For this work 3 OS was tested: Ubuntu Server Jammy, CentOS 8 and Debian 12. For final evaluation was used Debian 12 Cloud Image, because we had several errors applying the configuration at Ubuntu, resulting in a development lock, and CentOS 8 already initializes with some hardening configurations, making it unfeasible the hardening implementation study purposed for this project.

With the OS cloud-image defined, its necessary stablish the cloud-init configurations, these include network device configuration, user account setup, and the generation of an SSH key pair to enable secure access. This approach provides automated and dynamic configuration within the Proxmox environment. To enhance security, each `Terraform Destroy/Apply` cycle generates a new SSH key pair, ensuring that credentials are not reused. The generated credentials are exported and used in a file that enables Ansible to connect to each machine (see example file in Appendix C).



```
proxmox_endpoint = "192.168.122.184:8006"

proxmox_nodename = "pve"

servervm_object = {
  "vm1" = {
    name         = "vm1"
    tags         = ["web-server"]
    ipv4_address = "192.168.122.245"
    memory       = 2048
    cpu_cores    = 2
    disk_size    = 30
  }
  "vm2" = {
    name         = "vm2"
    tags         = ["audit"]
    ipv4_address = "192.168.122.246"
    memory       = 3072
    cpu_cores    = 2
    disk_size    = 20
  }
}
```

Figure 2.1: Terraform machine specification file example (`terraform.tfvars`)

The virtual machines and their configurations are defined in `terraform.tfvars` file, where an object containing machine-specific properties describes each virtual machine along with its individual configuration parameters. These configurations are used by Terraform to provision the infrastructure in Proxmox. Once the infrastructure is fully deployed, the IP addresses and group information of the virtual machines are exported to an `inventory.ini` file (Example at Appendix C), which is used by Ansible to execute the hardening playbook. This playbook ensures the continuous and secure management of virtual machines in the Proxmox environment.

2.3 Ansible

Manual hardening and virtual machine configuration processes are slow, error-prone, and can result in configuration inconsistencies. Ansible is an open-source and agent-less (don't requires any software in the guest system) IT automation tool that helps with system configuration, offering a repeatable and efficient solution for hardening tasks [3], [4].

For hardening configuration, configurations were selected from dev-sec hardening roles

[5] as reference to create the ansible playbook configuration, and roles for hardening and virtual machines configurations.

Ansbile directory configuration is strutucure to establish the working of the playbook, splitting the roles configuration from the inventory where all the machines are mapped to execute the playbook.

- `inventory.ini`: File that specify virtual machines address. Following the template example file at Appendix C, this file is auto-made during Terraform execution.

- `playbook.yml`: YML file that contain the tasks to be executed, and establish which role is assign to which virtual machine. The main used file can be seen in the Appendix C

- `roles/`: The directory containing all project roles.

- `roles/*role-name*/`: The Role configuration directory.

  – `defaults/main.yml`: Default variables files for Ansible execution.

  – `tasks/main.yml`: The main tasks file where you define what the role does step-by-step.

  – `handlers/main.yml`: Handlers triggered by task changes (e.g., restarting services).

  – `templates/`: Directory for template files.

  – `vars/`: Directory for variables files, such as OS vars for different operating systems (e.g., such as `Debian.yml`, `Ubuntu.yml`).

All the Ansible's roles configuration files are in the source code attached at this submission or can be found in the GitHub repository: `https://github.com/jgmsilva/SSI-cloud-project`.

2.3.1 Hardening Role

Creating a role for hardening is implicated only in organization and modularization of the playbook. The hardening will be applied at any machine intended to be a cloud server host, this way, those virtual machines needs to be secure for the user just be worried about host your application.

The role will follow the KISS (Keep it Simple) principle, so any package needed to run the application, such as docker, node etc, need to be defined at ansible configuration to be installed. Some tasks done by this role includes:

- Remove deprecated or insecure packages
- Disable CTRL-ALT-DEL, ensuring that its not possible to logout by brute force
- Limit access to home directories of regular (non-system, non-root) accounts
- Change Debian/Ubuntu systems so ssh starts traditionally instead of socket-activated
- Ensure privilege separation directory exists
- Activate SELinux
- Create sshd configuration
- Protect Linux Kernel /etc/sysctl.conf
- Modprobe configuration

Obs. Change the ssh start system its only in Debian like system because recent versions changed the traditional way to start to use socket-based, so changing to the previous and general way used by others Operating System shows up the best approach by Dev-Sec, general discussion for this implementation can be accessed here .

SSHD - OpenSSH Server

The main focus on creating the vms is to be used as cloud servers, its important to establish secure connections to the machines by ssh (Security Shell), so its necessary to hardening the sshd configuration, the OpenSSH server process.

The configuration file defines many configuration for the connection, here are some of the most important variables used to create the ssh configuration file:

`ssh_max_auth_retries:10` - Maximum number of authentication attempts permitted per connection. Once the number of failures reaches half this value, additional failures are logged.

`ssh_max_sessions:10` - Maximum number of open sessions permitted from a given connection.

`ssh_permit_root_login:"no"` - Disable the root login, limiting the system control access.

`sshd_authenticationmethods:publickey and ssh_server_password_login:false` - Specify that the only authentication method allowed is using publickey from a encrypted key paring, and disabling the password login.

SELinux

SELinux (Security-Enhance Linux) is a Linux kernel module to manage access control policies, this security extension have 3 policies configuration: disable; permissive, only give warns; and enforce, the chosen one for the virtual machines, this policy protects the system from malicious or flawed applications that can damage or destroy the system.

Memory need to be at minimum of 1024GB, if not the system falls in memory deadlock on boot.

Modprobe for kernel modules configuration

Installing modporbes enables the installation and disables of kernel modules, for this hardening configuration it will be main used to disable unused file systems and USB/firewire/thunderbolt devices, that users can't connect USB devices or install malware/viruses, even being virtual machines and the users will not have access to the physical server, it's good for hardening, disabling unused modules reduce possible attack vectors.

2.3.2 Wazuh Platform

The Wazuh Cybersecurity Platform[6] is an integrated Security Information and Event Management (SIEM) and Extended Detection and Response (XDR) platform, designed to be highly scalable. Wazuh began as a fork of the popular XDR program OSSEC, originating what is today the Wazuh server and agent. This base product was then incremented with the Wazuh Indexer and Dashboard forked from the OpenSearch project, a search and analytics tool that is itself a fusion of two forks from other open-source projects - ElasticSearch and Kibana.

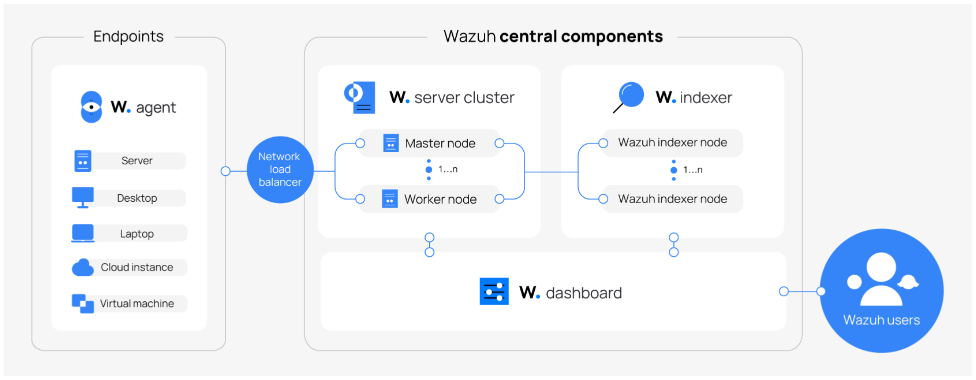


Figure 2.2: The Wazuh platform architecture

The wazuh agent, just like it's original ossec counterpart, is a daemon that runs on the endpoints one wish to monitor, providing functionalities such as log and file integrity monitoring, threat and vulnerability detection and incident response. It works based on a XML configuration file located in /var/ossec/etc/ossec.conf (configuration template is located in appendix B).

All incidents reported by the endpoints through the wazuh agents are sent to the wazuh server, which is responsible for anlysing the incoming data and generating alerts based its rules. The processed data is the securely sent throught the Filebeat shipper to the wazuh indexer. The wazuh server configuration is also stored at /var/ossec/etc/ossec.conf (configuration template is located in appendix B), but with different parameters from the agent.
All data analyzed by the wazuh server is stored and indexed by the wazuh indexer for quick lookup and access. It uses JSON documents to store data in a structured way, allowing for efficient query and analysis.
To visualize the gathered data and allow for an easier real-time monitoring we have the wazuh dashboard. It is responsible for querying the wazuh indexer and dysplaying the results in custom views, while providding details on overall system health parameters and allowing for ephemeral configuration changes.

Repository Setup

To ensure that we are installing the original binaries compiled for the source code, all components share a setup phase where the GPG key from the wazuh organization is downloaded and stored, while the signed packages are obtained directly from wazuh's own apt repository. The specific urls are defined in the `wazuh_repo` variable in both the `wazuh-agent` and `wazuh-manager` roles.

Certificate Generation and Deployment

Since the Wazuh architecture allows for its central components to be deployed in different machines, there is great concern in ensuring proper encryption and authentication between them. This is achieved in practice through the deployment of asymmetric keys to all components signed by a local Certificate Authority. When manually deploying to a cloud configuration, one must specify a `config.yml` file that matches exactly the deployed infrstructure. A combination of Ansible variables and Jinja2 template dinamically builds this configuration file, generates locally the certificates if they do not yet exist and later deploys the to their respective components.

Wazuh Manager

Due to our limited resources for running multiple VMs, we chose to combine all wazuh central components into a single `wazuh-manager` role. We found that there is a minimum requirement of 3Gb RAM memory for the correct execution of all components, as the machine would not boot with less.
The four central components are managed through individual configuration files, respectively:

- Wazuh Server: `ossec.conf`
- Filebeat: `filebeat.yml`
- Wazuh Indexer: `opensearch.yml`
- Wazuh Dashboard: `opensearch_dashboards.yml`

This highlights the utility of using central managements systems such as Ansible, as it eliminates many small error points over distributed configurations.

Wazuh Agent

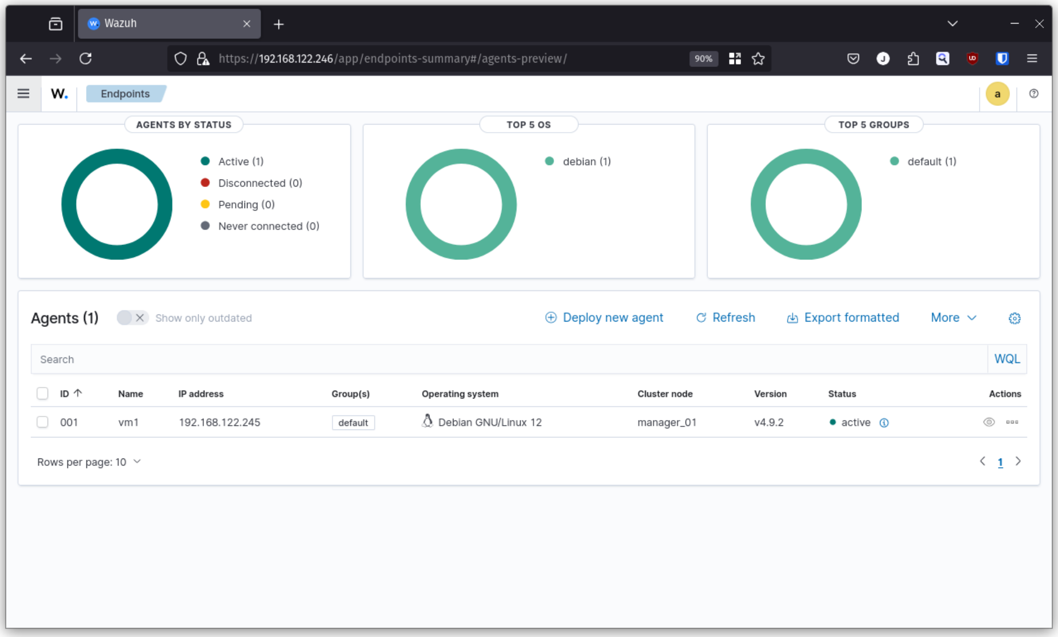
The monitored endpoints are configured through the addition of the `wazuh-agent` role. The playbook automatically populates the configuration file with the IP of all servers present in the inventory, allowing the agent to auto-enroll to all servers after installation.

Chapter 3

Results

3.1 Wazuh Enrollment

We can observe the correct setup of the Wazuh ecosystem if, after installation, we can correclty observe the wazuh dashboard at port 443 and all of its connected agents.

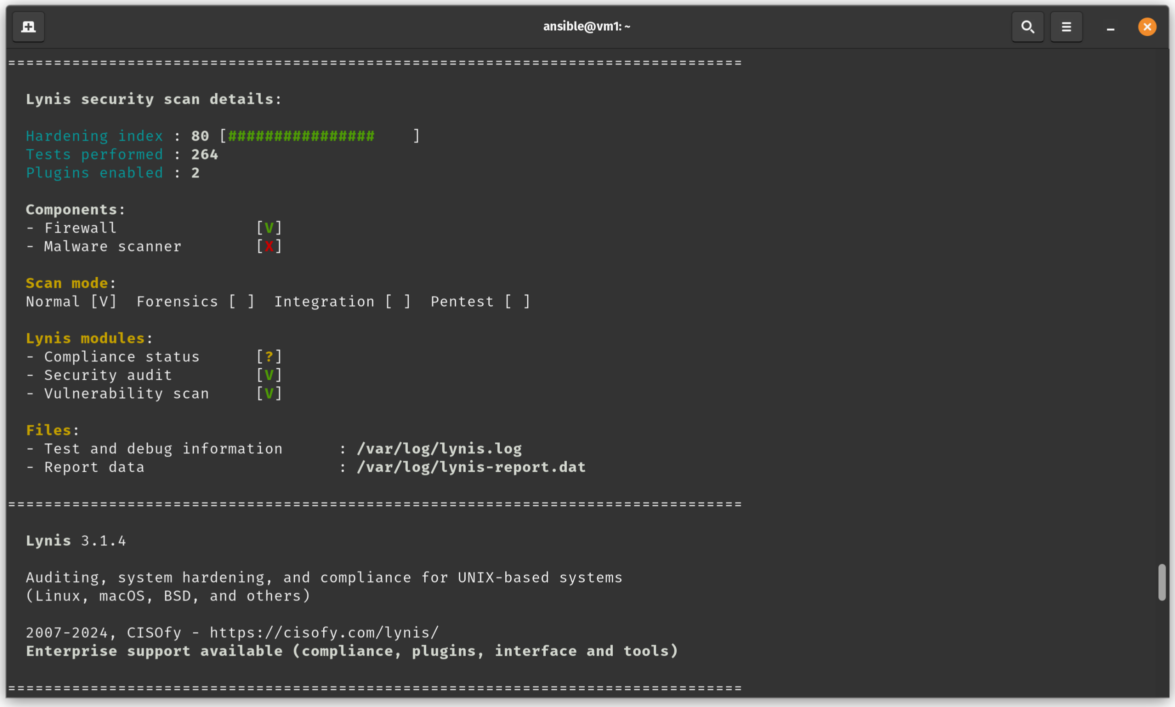


ID	Name	IP address	Group(s)	Operating system	Cluster node	Version	Status	Actions
001	vm1	192.168.122.245	default	Debian GNU/Linux 12	manager_01	v4.9.2	● active	

Figure 3.1: Wazuh displaying all connected agents

3.2 Lynis Evaluation

To evaluate the hardening process, we chose the Lynis Tool, an open-source security auditing tool. This approach allows us to assess the current state of the machine, identify processes and security topics being addressed, and follow the suggestions provided in the `audit system` method output. These recommendations help us further improve the hardening process. The final score output is shown in Image 3.2.



```
================================================================================

  Lynis security scan details:

  Hardening index : 80 [################    ]
  Tests performed : 264
  Plugins enabled : 2

  Components:
  - Firewall               [V]
  - Malware scanner        [X]

  Scan mode:
  Normal [V]  Forensics [ ]  Integration [ ]  Pentest [ ]

  Lynis modules:
  - Compliance status      [?]
  - Security audit         [V]
  - Vulnerability scan     [V]

  Files:
  - Test and debug information      : /var/log/lynis.log
  - Report data                     : /var/log/lynis-report.dat

================================================================================

  Lynis 3.1.4

  Auditing, system hardening, and compliance for UNIX-based systems
  (Linux, macOS, BSD, and others)

  2007-2024, CISOfy - https://cisofy.com/lynis/
  Enterprise support available (compliance, plugins, interface and tools)

================================================================================
```

Figure 3.2: Final Lynis Score after hardening using `audit system`

3.3 Final Results

To conclude this work, further details on the development process and the specific configurations applied can be found in the complete source code of the project, available in our GitHub repository: `https://github.com/jgmsilva/SSI-cloud-project`.

Chapter 4

Conclusion

Starting with the objective of building a secure Infrastructure-As-Code(IaC) configuration, we have leveled the main strengths of both Terraform and Ansible to provide a general baseline of security within a self-hosted Proxmox instance. We encountered many challenges in balancing convenience with security guarantees, especially when trying to minimize the attack surface for man-in-the-middle attacks. A key challenge came when trying to automate the implementation of the SELinux system on our base Debian virtual images, as the security solution of RedHat and Canonical are incompatible with each other. There is a great potential for future works to improve this current solution, both in extending current functionality and improving the existing structure. A promising extension would be to refactor the current terraform configuration into a provider-agnostic module, enabling it to be deployed to any cloud platform supported by terraform. The initial deployment could also be improved, as the first execution of the Ansible playbook must trust in the ssh fingerprint provided by the newly created machines, opening a possible vector of attack.

Bibliography

[1] Proxmox, *Proxmox - powerful open-source server solutions*, `https://www.proxmox.com/en/`, [Accessed 12-12-2024].

[2] P. Boldyrev, *GitHub - bpg/terraform-provider-proxmox: Terraform / OpenTofu Provider for Proxmox VE — github.com*, `https://github.com/bpg/terraform-provider-proxmox`, [Accessed 11-12-2024].

[3] Ansible, *How ansible works*, `https://www.ansible.com/how-ansible-works/`, [Accessed 12-12-2024].

[4] A. Critelli, *5 ways to harden a new system with ansible*, `https://www.redhat.com/en/blog/harden-new-system-ansible`, [Accessed 12-12-2024], 2020.

[5] dev-sec, *GitHub - dev-sec/ansible-collection-hardening: This Ansible collection provides battle tested hardening for Linux, SSH, nginx, MySQL — github.com*, `https://github.com/dev-sec/ansible-collection-hardening`, [Accessed 11-12-2024].

[6] Wazuh, *Wazuh: The open source security platform*, `https://www.wazuh.com`, [Accessed 12-12-2024].

Appendix A

Terraform Files

```
resource "proxmox_virtual_environment_vm" "client_vm" {
  for_each  = try(var.servervm_object, {})
  name      = each.value.name
  node_name = var.proxmox_nodename
  tags      = each.value.tags

  initialization {

    ip_config {
      ipv4 {
        address = "${each.value.ipv4_address}/24"
        gateway = "192.168.122.1"
      }
    }

    user_account {
      username = "ansible"
      keys     = [trimspace(tls_private_key.ubuntu_vm_key.
   public_key_openssh)]

    }
    # user_data_file_id = proxmox_virtual_environment_file.
   user_data_cloud_config.id
  }
  stop_on_destroy = true
  disk {
    interface    = "virtio0"
    size         = each.value.disk_size
    datastore_id = "local-lvm"
    iothread     = true
    discard      = "on"
    file_id      = proxmox_virtual_environment_download_file.
   ubuntu_cloud_image.id
  }

  cpu {
    cores = each.value.cpu_cores
    type  = "x86-64-v2-AES" # recommended for modern CPUs
  }

  memory {
    dedicated = each.value.memory
    floating  = each.value.memory # set equal to dedicated to enable
   ballooning
  }

  network_device {
    bridge = "vmbr0"
  }
}
resource "proxmox_virtual_environment_download_file" "ubuntu_cloud_image
   " {
  content_type = "iso"
  datastore_id = "local"
  node_name    = var.proxmox_nodename
  # url           = "https://cloud.centos.org/centos/8-stream/x86_64/images/CentOS-Stream-GenericCloud-8-latest.x86_64.qcow2"
  url       = "https://cloud.debian.org/images/cloud/bookworm/20230612-1409/debian-12-genericcloud-amd64-20230612-1409.qcow2"
  file_name = "debian12.img"
  # url = "https://cloud-images.ubuntu.com/jammy/current/jammy-server-cloudimg-amd64.img"
}

resource "tls_private_key" "ubuntu_vm_key" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

locals {
  inventory_groups = {
    audits  = [for vm in values(proxmox_virtual_environment_vm.client_vm) : var.servervm_object[vm.name].ipv4_address if contains(vm.tags, "audit")]
    servers = [for vm in values(proxmox_virtual_environment_vm.client_vm) : var.servervm_object[vm.name].ipv4_address if contains(vm.tags, "web-server")]
  }
}
resource "local_file" "ansible_inventory" {
  filename        = "./inventory.ini"
  file_permission = "0666"
  content         = templatefile("./templates/inventory.tpl", local.inventory_groups)
}
resource "local_file" "ansible_private_key" {
  filename        = "./keys/ansible_key.pem"
  content         = tls_private_key.ubuntu_vm_key.private_key_openssh
  file_permission = "0600"
}
```

```

resource "local_file" "ansible_public_key" {
  filename        = "./keys/ansible_key.pub"
  content         = trimspace(tls_private_key.ubuntu_vm_key.
   public_key_openssh)
  file_permission = "0666"
}

resource "local_file" "known_hosts" {
  filename        = "./terraform_known_hosts"
  content         = ""
  file_permission = "0666"
}

# resource "proxmox_virtual_environment_file" "user_data_cloud_config" {
#   content_type = "snippets"
#   datastore_id = "local"
#   node_name    = "pve"
#   source_raw {
#     data = <<-EOF
#     #cloud-config
#     hostname: vm1
#     users:
#       - default
#       - name: ansible
#         groups:
#           - sudo
#         shell: /bin/bash
#         ssh_authorized_keys:
#           - ${trimspace(tls_private_key.ubuntu_vm_key.
   public_key_openssh)}
#         sudo: ALL=(ALL) NOPASSWD:ALL
#     write_files:
#       - path: /etc/fstab
#         content: |
#           LABEL=cloudimg-rootfs   /    ext4   defaults    0 1
#     runcmd:
#       - "mkfs.ext4 /dev/vda1"
#       - "mkdir -p /mnt/data"
#       - "mount /dev/vda1 /mnt/data"
#       - "echo '/dev/vda1 /mnt/data ext4 defaults 0 1' >> /etc/fstab"
#       - apt update
#     EOF
#     file_name = "user-data-cloud-config.yaml"
#   }
# }
```

Listing A.1: Main Terraform file

```
terraform {
  required_providers {
    proxmox = {
      source  = "bpg/proxmox"
      version = "0.68.0"
    }
  }
}
provider "proxmox" {
  endpoint = "https://${var.proxmox_endpoint}/api2/json"
  username = "root@pam" #change for your proxmox access
  password = "proxmox"  #change for your proxmox access
  insecure = true
  ssh {
    agent = true
  }
}
```

Listing A.2: Terraform provider specification file

Appendix B

Wazuh Configuration Files

Template for the wazuh server and agent components as definedhttps://github.com/wazuh/wazuh/tree/mas
in `https://github.com/wazuh/wazuh/tree/master/etc/templates/config`

```
header-comments.template

<ossec_config>
    <client>
      <server>
        <address>192.168.10.100</address>
      </server>
      <config-profile>distribution, distributionVersion</config-profile>
    </client>
    <client_buffer>
      <!-- Agent buffer options -->
      <disabled>no</disabled>
      <queue_size>5000</queue_size>
      <events_per_second>500</events_per_second>
    </client_buffer>

    logging.template

    rootcheck.template

    wodle-openscap.template

    wodle-syscollector.template

    syscheck.template

    localfile-logs*

    localfile-commands.template

    localfile-extra.template

    <active-response>
      <disabled>no</disabled>
    </active-response>
</ossec_config>
```

Listing B.1: Wazuh Agent template configuration

```
header-comments.template

<ossec_config>
    global.template

    logging.template

    alerts.template

    remote-secure.template

    [remote-syslog.template]

    rootcheck.template

    wodle-openscap.template

    wodle-syscollector.template

    syscheck.template

    global-ar.template

    ar-commands.template

    ar-definitions.template

    localfile-logs*

    localfile-commands.template

    localfile-extra.template

    rules.template
</ossec_config>
```

Listing B.2: Wazuh Server template configuration file

Appendix C

Ansible Playbook Files

```yaml
- name: Init Machines
  hosts: all
  tasks:
    - name: Ping my hosts
      ansible.builtin.ping:

    - name: Wait for apt lock to be released
      shell: |
        while fuser /var/lib/dpkg/lock-frontend /var/lib/apt/lists/lock >/dev/null 2>&1; do
          echo "Waiting for other apt/dpkg processes to finish..."
          sleep 5
        done
      retries: 5
      delay: 10
      register: apt_lock_check
      until: apt_lock_check.rc == 0

    - name: Install basics packages
      become: true
      ansible.builtin.apt:
        name: ["acl"]
        state: present
        update_cache: true
      retries: 5
      delay: 10
      register: apt_result
      until: apt_result is succeeded

- hosts: audit
  become: true
  roles:
    - wazuh-manager
  vars:
    instances:
      node1:
        name: node-1 # Important: must be equal to indexer_node_name.
        ip: 127.0.0.1
        role: indexer
    ansible_shell_allow_world_readable_temp: true

- hosts: server
  become: true
  roles:
    - hardening
    - wazuh-agent
  vars:
    wazuh_managers: "{{ groups['audit'] }}"
```

Listing C.1: Ansible YAML Playbook File

```
[all:vars]
ansible_ssh_common_args = "-F ./ssh_config.local"

[audit]
192.168.122.246

[server]
```

```
192.168.122.245
```

Listing C.2: Auto Generated Ansible Inventory Example File

```
Host 192.168.*.*
    User ansible
    StrictHostKeyChecking accept-new
    UserKnownHostsFile ./terraform_known_hosts
    IdentityFile ./keys/ansible_key.pem
```

Listing C.3: Auto Generated SSH Access File