MASTER IMPLEMENTATION MANUAL

Autonomous DevSecOps Architecture
with Active Threat Mitigation

Rahul Datta

January 26, 2026

Strategic Vision: Self-Defending Zero-Trust Private Cloud

This project constructs an **Enterprise-Grade Isolated Network** operating on **Defense-in-Depth** principles. By physically segmenting networks into distinct security zones (DMZ and Internal Vault), implementing automated threat detection/response, and maintaining comprehensive forensic logging, we create a system where:

- Code deploys automatically without human intervention (DevOps)
- Attacks are detected and neutralized in milliseconds (Active Defense)
- All security events are preserved in a tamper-proof vault (Forensics)
- Management infrastructure remains invisible to attackers (Zero-Trust)

This eliminates the "flat network" vulnerability where compromising one service grants access to entire infrastructure.

Platform: Windows 11 Host + Oracle VirtualBox 7.0 + Ubuntu Server 22.04 LTS

System Credentials:

- **DMZ:** User: `dmz-bastion-admin` / Pass: `admin` / Host: `dmz-bastion`
- **Internal:** User: `internal-vault-admin` / Pass: `admin` / Host: `internal-vault`

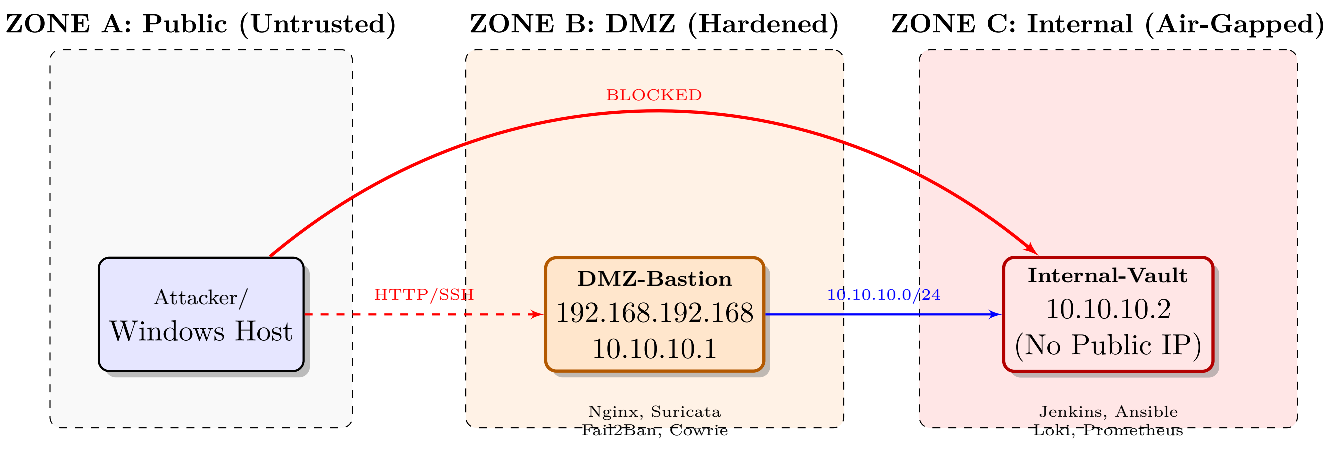
Critical: Tactical Note: "Install First, Isolate Later"

This manual uses a practical workaround: both VMs start with NAT internet access to download dependencies. Once installation completes, we permanently disable Internal VM's internet connection.
Why? Configuring package managers to use a proxy *before the proxy exists* creates circular dependencies. This is an installation sequence optimization, not the core security strategy.

Contents

1 Architecture Overview

1.1 The Three-Zone Security Model


ZONE A: Public (Untrusted)
ZONE B: DMZ (Hardened)
ZONE C: Internal (Air-Gapped)
BLOCKED
Attacker/ Windows Host
HTTP/SSH
DMZ-Bastion 192.168.192.168 10.10.10.1
10.10.10.0/24
Internal-Vault 10.10.10.2 (No Public IP)
Nginx, Suricata Fail2Ban, Cowrie
Jenkins, Ansible Loki, Prometheus

Foundational Concept: Network Segmentation Philosophy

Flat Network Problem: Typical setups place web servers and management tools (Jenkins, databases) on the same network. One compromised web service = immediate access to management infrastructure.
DMZ Solution: A "Demilitarized Zone" sits between untrusted (internet) and trusted (internal) networks. Only public-facing services live in the DMZ. Even if compromised, attackers cannot reach internal management without breaching a second defense layer.
Air Gap: The Internal Vault has *zero direct internet connectivity.* It communicates with the outside world only by routing through the DMZ's proxy, making it invisible to external attackers.

1.2 VirtualBox Network Adapter Types

Adapter Type	**Behavior**	**Use Case**
NAT	VM gets internet via host. VM receives dynamic IP (10.0.2.x). Host cannot directly access VM.	Temporary package installation
Host-Only	Virtual network connecting host and VMs. Static IPs (192.168.192.x). No internet.	Management access from Windows
Internal Network	Isolated network between VMs only. Host cannot access it.	DMZ↔Internal communication (10.10.10.x)

2 Phase 1: The Foundation

Phase Objective: Build and Provision Infrastructure

Create two virtual machines, establish VirtualBox networking, install base OS, and download all required software packages *while internet access is unrestricted.*

2.1 Step 1.1: VirtualBox Host-Only Network

Why This Matters: Manual IP Assignment

DHCP assigns IPs dynamically (can change on reboot). Servers need **predictable addresses**. Configuration files will hardcode IPs like `10.10.10.1`, so these must never change. Disabling DHCP forces static assignment.

Procedure:

1. Open VirtualBox → **Tools** → **Network Manager**
2. Click **Create**
3. Configure:
   - **IPv4 Address:** `192.168.192.1` (Windows host's IP on this network)
   - **Network Mask:** `255.255.255.0` (Allows 192.168.192.1–254)
   - **DHCP Server: Disabled** (Critical)

Technical Deep-Dive: CIDR Notation Explained

`192.168.192.0/24` means:
- **Network:** 192.168.192.0 (identifies the network itself)
- **/24:** First 24 bits = network (192.168.192), last 8 bits = hosts (0–255)
- **Subnet Mask:** 255.255.255.0 (binary: 11111111.11111111.11111111.00000000)
- **Usable IPs:** 192.168.192.1 through 192.168.192.254 (256 total minus network/broadcast)

2.2 Step 1.2: Virtual Machine Creation

VM 1: DMZ-Bastion (The Gateway)

- **Name:** `dmz-bastion`
- **Resources:** 2 vCPUs, 4096 MB RAM, 30 GB disk
- **OS:** Ubuntu Server 22.04 LTS
- **Network Adapters:**
  1. Adapter 1 (NAT): Temporary internet
  2. Adapter 2 (Host-Only): Management (192.168.192.168)
  3. Adapter 3 (Internal "intnet"): Gateway to Vault (10.10.10.1)

VM 2: Internal-Vault (The Secure Core)

- **Name:** `internal-vault`
- **Resources:** 2 vCPUs, 4096 MB RAM, 30 GB disk
- **OS:** Ubuntu Server 22.04 LTS
- **Network Adapters:**
  1. Adapter 1 (NAT): Temporary internet (disabled in Phase 2)
  2. Adapter 2 (Internal "intnet"): Connection to DMZ (10.10.10.2)

Critical: Installation Credentials

Use these **exact credentials** during Ubuntu installation:

- **DMZ:** Username: `dmz-bastion-admin`, Password: `admin`, Hostname: `dmz-bastion`
- **Internal:** Username: `internal-vault-admin`, Password: `admin`, Hostname: `internal-vault`

These must match exactly for SSH key copying and Ansible inventory to work.

2.3 Step 1.3: Base System Configuration

Run the following commands on BOTH VMs after initial login.

Foundational Concept: Linux Swap Space

What is Swap? Dedicated disk space used as "overflow RAM." When physical RAM is exhausted, the kernel moves inactive memory pages to swap.
Why 4GB? Jenkins + Docker + Monitoring can peak at 3-4GB RAM. Without swap, Linux OOM (Out-Of-Memory) killer terminates processes randomly. Swap prevents catastrophic failure.
Performance Note: Disk is 100× slower than RAM. Swap is emergency insurance, not a RAM replacement.

Listing 1: Switch to Root User

```
# Execute all subsequent commands as root (superuser)
sudo -i
```

Explanation:

- `sudo`: Execute command as root
- `-i`: Start interactive root shell (avoids typing sudo repeatedly)

Listing 2: Create 4GB Swap File

```
# Allocate 4GB file instantly (faster than dd)
fallocate -l 4G /swapfile

# Secure the file (only root read/write)
chmod 600 /swapfile

# Format as swap space
mkswap /swapfile

# Activate swap immediately
swapon /swapfile

# Make permanent across reboots
echo '/swapfile none swap sw 0 0' >> /etc/fstab
```

Line-by-Line:

- `fallocate -l 4G`: Creates 4GB file by allocating disk blocks (no actual writing)
- `chmod 600`: Sets permissions to `rw----` (prevents non-root from reading swap data)
- `mkswap`: Writes swap signature and metadata
- `swapon`: Activates swap immediately
- `echo ... » /etc/fstab`: Appends mount instruction. Format: `<device> <mount> <type> <opts> <dump> <pass>`

Listing 3: Install Essential Tools

```
# Update package index
apt update

# Upgrade all packages
apt upgrade -y

# Install critical utilities
apt install -y curl wget git unzip htop net-tools \
               software-properties-common jq tree vim python3-pip
```

Package Purposes:

- `curl/wget`: Download files from URLs
- `git`: Version control system
- `htop`: Interactive process viewer
- `net-tools`: Legacy networking commands (`ifconfig`, `netstat`)
- `software-properties-common`: Provides `add-apt-repository`
- `jq`: JSON parser for command-line
- `python3-pip`: Python package installer

2.4 Step 1.4: Zone-Specific Software

2.4.1 DMZ-Bastion Installation

Why This Matters: DMZ Tool Roles

Docker: Containerized web application (isolated from host)
Nginx: Reverse proxy (routes `/jenkins` to Internal)
Squid: HTTP/HTTPS proxy (enables Internal updates)
Suricata: Network IDS with deep packet inspection
Fail2Ban: IPS that reads Suricata logs, updates firewall
Cowrie: SSH honeypot (traps attackers)

Listing 4: DMZ Software Stack (Run ONLY on dmz-bastion)

```
# Install Docker (container runtime)
apt install -y docker.io

# Install web server and proxy
apt install -y nginx squid

# Install intrusion prevention
apt install -y fail2ban

# Add Suricata PPA and install
add-apt-repository ppa:oisf/suricata-stable -y
apt update
apt install -y suricata

# Create Cowrie log directory
mkdir -p /var/log/cowrie
chmod 777 /var/log/cowrie

# Add user to docker group (enables docker without sudo)
usermod -aG docker dmz-bastion-admin
```

Technical Deep-Dive: User Groups and Docker Socket

Problem: Docker daemon runs as root. By default, only root can access `/var/run/docker.sock`.
Solution: The `docker` group has socket access. Adding user to this group grants Docker privileges without sudo.
Command Breakdown:

- `usermod`: Modify user account
- `-aG`: `-a` = append, `-G` = supplementary groups
- `docker`: Group name
- `dmz-bastion-admin`: Username

Security Note: This grants near-root privileges (container escape via volume mounts possible). Only for trusted users.

2.4.2 Internal-Vault Installation

Why This Matters: Internal Tool Roles

Jenkins: CI/CD automation (runs pipelines, triggers Ansible)
Ansible: Configuration management (deploys to DMZ via SSH)
Docker: Runs monitoring stack (Prometheus, Loki, Grafana)
Python Requests: HTTP library for querying Loki (forensics)

Listing 5: Internal Vault Stack (Run ONLY on internal-vault)

```
# Install Java Runtime (Jenkins dependency)
apt install -y openjdk-17-jre

# Install Docker, Ansible, sshpass, and ACL (Required for permissions)
apt install -y docker.io ansible sshpass acl

# Add Jenkins GPG key
curl -fsSL https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key | \
  tee /usr/share/keyrings/jenkins-keyring.asc > /dev/null

# Add Jenkins repository
echo "deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/" | \
  tee /etc/apt/sources.list.d/jenkins.list > /dev/null

# Update and install Jenkins
apt update && apt install -y jenkins

# Install Python HTTP library
pip3 install requests

# Add user to docker group
usermod -aG docker internal-vault-admin
```

Technical Deep-Dive: APT Repository Management

Why Manual Addition? Ubuntu's default repos don't include Jenkins. We add the official Jenkins repository.
GPG Key Verification: `jenkins.io-2023.key` is Jenkins's public GPG key. APT uses this to verify package signatures (prevents malicious injection).

Signed-By Directive: `[signed-by=...]` tells APT to trust packages from this repo only if signed by this specific key (not global keyring).
Why sshpass? Ansible prefers SSH keys. `sshpass` enables password-based SSH for initial setup before key deployment.

⚠ Critical: Mandatory Reboot

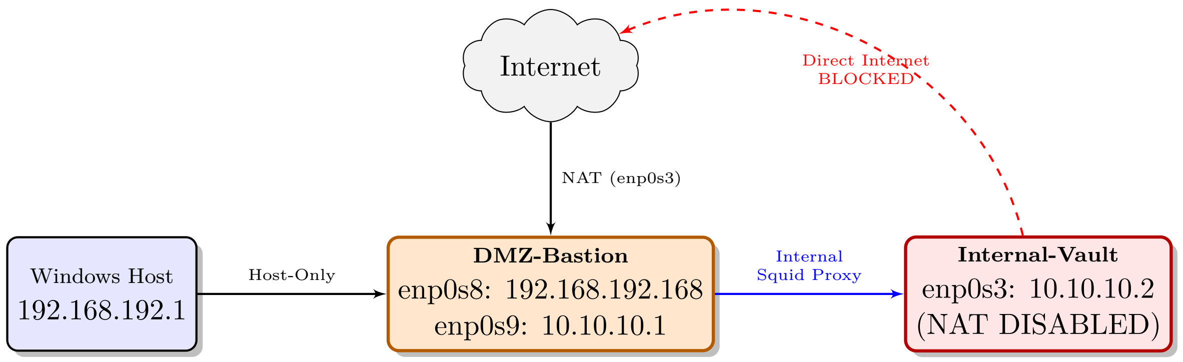
Group membership changes (`usermod -aG docker`) require logout/login. Reboot both VMs:
`reboot`
After reboot, verify: `docker ps` (should work without sudo)

3 Phase 2: Networking & Isolation

Phase Objective: Establish the Air Gap

Configure static IPs, disable Internal VM's direct internet, configure DMZ as HTTP proxy gateway, and verify Internal can only reach internet through DMZ.

3.1 Network Architecture Post-Phase 2



Foundational Concept: Foundational Concept: The Two Proxies

In this architecture, we use **two different proxies** for opposite purposes. It is crucial not to confuse them.

1. **Squid (Forward Proxy):**
   - **Direction:** Outbound (Internal VM → Internet)
   - **Purpose:** Allows the air-gapped Internal VM to download updates (apt, pip, docker) without having a direct internet connection.
   - **Analogy:** A "Gate Pass" office. Employees (Internal VM) must ask the office (Squid) to fetch items from outside.
2. **Nginx (Reverse Proxy):**
   - **Direction:** Inbound (User/Admin → Internal Jenkins)
   - **Purpose:** Allows you to access the Jenkins dashboard running on the private Internal VM by connecting to the public-facing DMZ IP.
   - **Analogy:** A "Receptionist". You talk to the receptionist (Nginx on DMZ), and they forward your request to the back-office staff (Jenkins), who you cannot see directly.

3.2 Step 2.1: DMZ Static IP Configuration

Foundational Concept: Netplan Configuration System

What is Netplan? Modern Ubuntu uses Netplan (YAML-based) for network config, replacing legacy `/etc/network/interfaces`. Netplan generates config for underlying renderers (NetworkManager or systemd-networkd).
Declarative Model: Describe desired state; Netplan makes it happen. Idempotent (safe to apply multiple times).
YAML Rule: Use **spaces for indentation**, never tabs. Indentation defines structure.

On DMZ-Bastion:

Listing 6: Edit Netplan Configuration

```
sudo nano /etc/netplan/00-installer-config.yaml
```

Replace entire contents with:

Listing 7: DMZ Netplan Configuration

```
network:
  ethernets:
    enp0s3:  # Adapter 1: NAT (Internet)
      dhcp4: true  # Keep DHCP for auto internet

    enp0s8:  # Adapter 2: Host-Only (Management)
      dhcp4: no
      addresses: [192.168.192.168/24]
      # No gateway - not our default route

    enp0s9:  # Adapter 3: Internal (Gateway to Vault)
      dhcp4: no
      addresses: [10.10.10.1/24]
      # Makes DMZ the gateway for 10.10.10.0/24

  version: 2
```

Interface Breakdown:

- `enp0s3` (NAT): DHCP enabled, gets automatic IP (10.0.2.15), becomes default route for internet
- `enp0s8` (Host-Only): Static `192.168.192.168`, Windows host accesses DMZ services here
- `enp0s9` (Internal): Static `10.10.10.1`, DMZ becomes gateway router for Internal Vault

Listing 8: Apply Configuration

```
sudo netplan apply
```

Technical Deep-Dive: Netplan Apply Process

When running `netplan apply`:
1. Validates YAML syntax
2. Generates backend config (systemd-networkd files in `/run/systemd/network/`)
3. Restarts network services
4. Applies new IPs and routes

Verify: `ip addr show` (should see all three interfaces with correct IPs)
Test Routing: `ip route show` (default route should be via enp0s3)

3.3 Step 2.2: Internal Vault Isolation

Critical: Point of No Return

We are about to permanently disable Internal VM's direct internet. Ensure ALL packages from Step 1.4 are installed before proceeding. After this step, packages can only be installed via proxy.

Step-by-Step:

1. **Shutdown** the Internal-Vault VM completely
2. In VirtualBox, select Internal-Vault VM
3. Go to **Settings** → **Network** → **Adapter 1**
4. **Uncheck** "Enable Network Adapter" (disables NAT permanently)
5. Click OK and **Start** the VM

On Internal-Vault, edit Netplan:

Listing 9: Edit Internal Netplan

```
sudo nano /etc/netplan/00-installer-config.yaml
```

Replace entire contents with:

Listing 10: Internal Vault Network Configuration

```
network:
  ethernets:
    enp0s3:  # Only interface (Internal Network)
      dhcp4: no
      addresses: [10.10.10.2/24]
      routes:
        - to: default
          via: 10.10.10.1  # DMZ is our gateway
      nameservers:
        addresses: [8.8.8.8, 8.8.4.4]  # Google DNS

  version: 2
```

Configuration Explained:

- `addresses: [10.10.10.2/24]`: Static IP on internal network
- `routes: - to: default via: 10.10.10.1`: All traffic routes through DMZ gateway
- `nameservers`: DNS servers for domain resolution (routed via DMZ proxy)

Listing 11: Apply Configuration

```
sudo netplan apply
```

Technical Deep-Dive: Routing and Default Gateway

What is a Gateway? A router that forwards packets to destinations outside the local network. When the Internal VM wants to reach the internet, it sends packets to `10.10.10.1` (the DMZ), which then forwards them.
Default Route: The route used when no specific route matches. Format: "send all traffic (`default`) to gateway (`via 10.10.10.1`)".
Verify Routing:

- `ip route show` should show: `default via 10.10.10.1 dev enp0s3`
- `ping 10.10.10.1` should succeed (can reach DMZ)
- `ping 8.8.8.8` will FAIL (no proxy configured yet)

3.4 Step 2.3: Squid Proxy Configuration

Why This Matters: Why a Proxy?

The Internal VM needs to download packages and Docker images, but has no direct internet. The DMZ acts as a "proxy" - it receives HTTP/HTTPS requests from Internal, fetches content from the internet, and forwards responses back.
This maintains the air gap (Internal never directly connects to internet) while enabling necessary updates.

3.4.1 Configure Squid on DMZ

Listing 12: Edit Squid Configuration (On DMZ-Bastion)

```
sudo nano /etc/squid/squid.conf
```

Find the line: `http_access allow localhost`

Add these TWO lines ABOVE it:

```
acl internal_net src 10.10.10.0/24
http_access allow internal_net
```

Explanation:

- `acl internal_net src 10.10.10.0/24`: Defines an Access Control List named "internal_net" matching source IPs from 10.10.10.0/24
- `http_access allow internal_net`: Grants HTTP access to IPs matching this ACL
- Must be ABOVE `localhost` line because Squid processes rules top-to-bottom

Listing 13: Restart Squid

```
sudo systemctl restart squid
```

Technical Deep-Dive: Squid Default Port

Squid listens on port **3128** by default. Verify with: `netstat -tulpn | grep 3128`
You should see: `0.0.0.0:3128` (listening on all interfaces)

3.4.2 Configure Proxy on Internal Vault

Listing 14: Set Proxy Environment Variables (On Internal-Vault)

```
# Add proxy settings to shell profile
echo 'export http_proxy="http://10.10.10.1:3128"' >> ~/.bashrc
echo 'export https_proxy="http://10.10.10.1:3128"' >> ~/.bashrc
echo 'export HTTP_PROXY="http://10.10.10.1:3128"' >> ~/.bashrc
echo 'export HTTPS_PROXY="http://10.10.10.1:3128"' >> ~/.bashrc

# Apply immediately
source ~/.bashrc
```

Why Both Cases? Some tools check lowercase (`http_proxy`), others check uppercase (`HTTP_PROXY`). Setting both ensures compatibility.

Listing 15: Configure APT to Use Proxy

```
# Create APT proxy configuration
echo 'Acquire::http::Proxy "http://10.10.10.1:3128";' | \
  sudo tee /etc/apt/apt.conf.d/proxy.conf
```

Critical: Verification Test

Run on Internal-Vault:
`curl -I http://example.com`
Expected Output:
`HTTP/1.1 200 OK`

```
...
Via: 1.1 dmz-bastion (squid/4.13)
```

The `Via:` header confirms traffic is routed through Squid. If this hangs or times out, check:

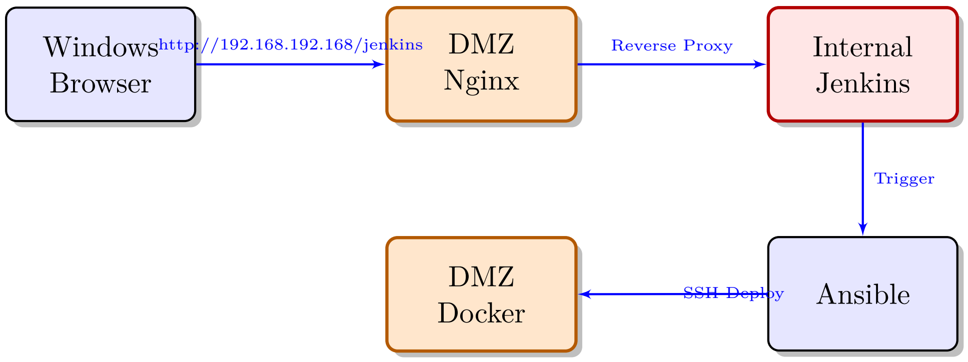
- Squid ACL configuration (`10.10.10.0/24` allowed?)
- Squid service running (`systemctl status squid`)
- Netplan routing (`ip route show`)

4 Phase 3: The CI/CD Pipeline

Phase Objective: Automate Deployment

Establish SSH trust between Internal and DMZ, create Ansible playbook for container deployment, configure Nginx reverse proxy for Jenkins access, and integrate Jenkins with Ansible for zero-touch deployment.

4.1 The Pipeline Architecture



Foundational Concept: CI/CD Pipeline Philosophy

Continuous Integration (CI): Automatically test and build code on every commit.
Continuous Deployment (CD): Automatically deploy tested code to production.
Our Flow:

1. User triggers Jenkins job via web UI
2. Jenkins executes Ansible playbook
3. Ansible SSHs to DMZ and deploys Docker container
4. Web application updates with zero downtime

Why This Matters: Manual deployment is error-prone and slow. Automation ensures consistency, enables rapid iteration, and eliminates "works on my machine" issues.

4.2 Step 3.1: SSH Key-Based Authentication

Why This Matters: Password vs Key Authentication

Password: User types password each time. Vulnerable to brute force, cannot be automated easily.
SSH Keys: Cryptographic key pair (private key on client, public key on server). Private key never transmitted. Enables passwordless automation.
Trust Model: By copying the public key to DMZ's `authorized_keys`, we're saying "anyone with the matching private key can log in as this user."

On Internal-Vault (logged in as internal-vault-admin):

Listing 16: Generate SSH Key Pair

```
# Generate RSA key pair (press Enter for all prompts)
ssh-keygen -t rsa -b 4096
```

Parameters:

- `-t rsa`: Key type (RSA algorithm)

- `-b 4096`: Key size (4096 bits for strong security)
- Default location: `~/.ssh/id_rsa` (private key) and `~/.ssh/id_rsa.pub` (public key)

Listing 17: Copy Public Key to DMZ

```
# Copy public key to DMZ's authorized_keys
ssh-copy-id dmz-bastion-admin@10.10.10.1
```

Critical: Configure Jenkins SSH Access

Jenkins runs as a separate user (`jenkins`). We must copy the generated keys so Jenkins can use them:

```
# Create directory for jenkins SSH keys
sudo mkdir -p /var/lib/jenkins/.ssh

# Copy the admin key
sudo cp ~/.ssh/id_rsa /var/lib/jenkins/.ssh/

# Set ownership to jenkins user
sudo chown -R jenkins:jenkins /var/lib/jenkins/.ssh

# Secure the key permissions
sudo chmod 600 /var/lib/jenkins/.ssh/id_rsa
```

Listing 18: Verify Passwordless Login

```
# Should connect without password prompt
ssh dmz-bastion-admin@10.10.10.1 "echo SSH Trust Verified"
```

Expected Output: `SSH Trust Verified`

Technical Deep-Dive: SSH Key Authentication Flow

1. Client initiates connection
2. Server sends challenge encrypted with client's public key
3. Only the matching private key can decrypt the challenge
4. Client decrypts and sends response
5. Server verifies response and grants access

Security Note: Never share private key (`id_rsa`). Always keep it on the source machine with `chmod 600` permissions.

4.3 Step 3.2: Ansible Configuration

Why This Matters: Why Ansible?

Problem: Manually SSHing to servers, running commands, and deploying containers is tedious and error-prone.
Ansible Solution: Declare desired state (YAML playbooks), Ansible makes it happen. Idempotent (running twice produces same result). Agentless (uses SSH, no daemon needed on targets).
Our Use Case: Deploy/update Docker containers on DMZ from Internal (triggered by Jenkins).

On Internal-Vault:

Listing 19: Create Project Directory

```
mkdir -p ~/ops && cd ~/ops
```

Listing 20: Create Ansible Inventory File

```
nano inventory.ini
```

Paste this content:

```
[dmz]
10.10.10.1 ansible_port=22 ansible_user=dmz-bastion-admin \
           ansible_password=admin \
           ansible_become_password=admin \
           ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

Explanation:

- `[dmz]`: Group name (can reference in playbooks as `hosts: dmz`)
- `10.10.10.1`: Target host IP
- `ansible_port`: SSH port (Starts at 22; change to 2222 after Phase 6)
- `ansible_user`: Username for SSH connection
- `ansible_password`: SSH login password
- `ansible_become_password`: Password required for sudo (become: yes) privileges
- `ansible_ssh_common_args`: Disables SSH host key verification for automation

⚠ Critical: Security Consideration

`StrictHostKeyChecking=no` disables MITM (Man-In-The-Middle) protection. In a production environment, you should:

1. Pre-populate `known_hosts` with legitimate server fingerprints
2. Use `StrictHostKeyChecking=yes` (default)

For this isolated lab environment, it's acceptable for convenience.

Listing 21: Create Ansible Playbook

```
nano deploy.yml
```

Paste this content:

```
---
- hosts: dmz
  become: yes
  tasks:
    - name: Ensure Docker container is running
      community.docker.docker_container:
        name: webapp
        image: nginx:latest
        state: started
        restart_policy: always
        ports:
          - "8082:80"
```

Playbook Breakdown:

- `hosts: dmz`: Target the `[dmz]` group from inventory
- `become: yes`: Execute tasks with sudo (required for Docker)

- `community.docker.docker_container`: Ansible module for managing Docker containers
- `name: webapp`: Container name (used for reference)
- `image: nginx:latest`: Docker image to run
- `state: started`: Ensure container is running (create if doesn't exist)
- `restart_policy: always`: Auto-restart on failure or reboot
- `ports: "8082:80"`: Map host port 8082 to container port 80

Technical Deep-Dive: Port Mapping Explained

Why Port 8082? Nginx is already using port 80 on the DMZ host. Docker cannot bind to the same port twice.
Format: `"HOST_PORT:CONTAINER_PORT"`

- `8082`: Port on DMZ host machine
- `80`: Port inside the container (Nginx default)

Traffic flow: `User` → `DMZ:8082` → `Container:80`
Later, Nginx will reverse-proxy `DMZ:80` → `localhost:8082` to unify access.

Listing 22: Test Ansible Playbook

```
ansible-playbook -i inventory.ini deploy.yml
```

Expected Output:

```
PLAY [dmz] ******************************************

TASK [Ensure Docker container is running] *********
changed: [10.10.10.1]

PLAY RECAP ******************************************
10.10.10.1     : ok=1    changed=1    unreachable=0    failed=0
```

Verification: From Windows browser, visit `http://192.168.192.168:8082`

You should see the Nginx welcome page.

4.4 Step 3.3: Nginx Reverse Proxy Configuration

Foundational Concept: Reverse Proxy Concept

Forward Proxy: Client knows about proxy, explicitly sends traffic through it (like Squid).
Reverse Proxy: Client thinks it's talking directly to origin server. Proxy intercepts and forwards traffic transparently.
Benefits:

- **URL Unification:** Single entry point (`192.168.192.168`) for multiple services
- **Security:** Backend servers never exposed directly
- **Load Balancing:** Distribute traffic across multiple backends
- **SSL Termination:** Nginx handles HTTPS, backends use plain HTTP

On DMZ-Bastion:

Listing 23: Edit Nginx Default Site

```
sudo nano /etc/nginx/sites-available/default
```

Replace entire server block with:

```
server {
    listen 80;
    server_name _;  # Match any hostname

    # Route 1: Web Application (Docker container)
    location / {
        proxy_pass http://localhost:8082;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }

    # Route 2: Jenkins (Internal Vault)
    location /jenkins {
        proxy_pass http://10.10.10.2:8080/jenkins;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Required for Jenkins WebSocket connections
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }
}
```

Configuration Explained:

- `listen 80`: Accept HTTP connections on port 80
- `location /`: Default route (matches all paths not caught by other locations)
- `proxy_pass http://localhost:8082`: Forward requests to Docker container
- `location /jenkins`: Matches paths starting with `/jenkins`
- `proxy_pass http://10.10.10.2:8080/jenkins`: Forward to Internal Jenkins
- `proxy_set_header Host`: Preserve original host header
- `X-Real-IP / X-Forwarded-For`: Pass client's real IP to backend
- `X-Forwarded-Proto`: Indicate original protocol (HTTP/HTTPS)

Technical Deep-Dive: Nginx Location Matching

Nginx processes location blocks in specific order:

1. Exact matches (`location = /jenkins`)
2. Prefix matches (`location /jenkins`)
3. Regular expressions (`location ~/jenkins`)
4. Default (`location /`)

Our config: `/jenkins/*` routes to Jenkins, everything else routes to Docker webapp.

Listing 24: Test and Restart Nginx

```
# Test configuration syntax
sudo nginx -t

# If OK, restart Nginx
sudo systemctl restart nginx
```

Verification:

- `http://192.168.192.168` → Should show webapp (Docker Nginx)
- `http://192.168.192.168/jenkins` → Should reach Jenkins (after next step)

4.5 Step 3.4: Jenkins Prefix Configuration

Why This Matters: The Jenkins Prefix Problem

Problem: Jenkins assumes it owns the root URL (`/`). When Nginx sends `/jenkins/*` requests, Jenkins generates URLs without the `/jenkins` prefix, breaking links and CSS.
Solution: Configure Jenkins to expect a URL prefix using the `-prefix` flag.
Why Systemd Override? Jenkins is managed by systemd. We use a drop-in override file to modify startup parameters without editing the main service file (which could be overwritten by updates).

On Internal-Vault:

Listing 25: Create Systemd Override Directory

```
sudo mkdir -p /etc/systemd/system/jenkins.service.d/
```

Listing 26: Create Override Configuration File

```
sudo nano /etc/systemd/system/jenkins.service.d/override.conf
```

Paste this content:

```
[Service]
# Configure proxy for Jenkins plugin downloads (Required for internet access)
Environment="JAVA_OPTS=-Dhttp.proxyHost=10.10.10.1 \
  -Dhttp.proxyPort=3128 \
  -Dhttps.proxyHost=10.10.10.1 \
  -Dhttps.proxyPort=3128 \
  -Dhttp.nonProxyHosts=localhost"

# Reset ExecStart (required before redefining)
ExecStart=

# Start Jenkins with /jenkins prefix
ExecStart=/usr/bin/jenkins --prefix=/jenkins
```

Configuration Breakdown:

- `[Service]`: Section for service-specific settings
- `Environment="JAVA_OPTS=..."`: Set Java system properties
- `-Dhttp.proxyHost`: Java HTTP proxy host
- `-Dhttp.nonProxyHosts=localhost`: Don't proxy localhost traffic
- `ExecStart=`: Empty ExecStart (resets inherited value)
- `ExecStart=/usr/bin/jenkins -prefix=/jenkins`: New start command with prefix

Technical Deep-Dive: Systemd Drop-In Files

Main Service File: `/lib/systemd/system/jenkins.service` (managed by package, don't edit)
Override Directory: `/etc/systemd/system/jenkins.service.d/` - Files here merge with main service file - Takes precedence over main file - Survives package updates
Why Reset ExecStart? Systemd appends by default. For `ExecStart`, we must first clear

(`ExecStart=`) then redefine, otherwise both old and new commands run.

Listing 27: Apply Changes

```
# Reload systemd configuration
sudo systemctl daemon-reload

# Restart Jenkins
sudo systemctl restart jenkins

# Check status
sudo systemctl status jenkins
```

Wait 30-60 seconds for Jenkins to fully start.

⚠ Critical: Initial Jenkins Setup

Access Jenkins: Open Windows browser to `http://192.168.192.168/jenkins`
Unlock Jenkins: You'll need the initial admin password.
On Internal-Vault, retrieve password:

```
sudo cat /var/lib/jenkins/secrets/initialAdminPassword
```

Copy this password and paste into the browser.
Setup Steps:

1. Select "Install suggested plugins"
2. Create admin user (or skip and continue as admin)
3. Confirm Jenkins URL: `http://192.168.192.168/jenkins/`

4.6 Step 3.5: Jenkins Pipeline Integration

⚠ Critical: Set Jenkins User Permissions

Jenkins runs as a restricted user (`jenkins`) and cannot access the admin's home directory by default. We must grant access via ACL (Access Control Lists) to allow the pipeline to run:

```
# Allow Jenkins to traverse home directory
sudo setfacl -m u:jenkins:x /home/internal-vault-admin

# Allow Jenkins to read/write in ops directory
sudo setfacl -R -m u:jenkins:rwx /home/internal-vault-admin/ops
```

In Jenkins Web UI:

1. Click "New Item"
2. Enter name: `DeployWebApp`
3. Select "Freestyle project"
4. Click OK
5. Scroll to "Build" section
6. Click "Add build step" → "Execute shell"

7. Paste this script:

Listing 28: Jenkins Build Script

```
# Set proxy for Ansible (if needed for pulling roles)
export http_proxy=http://10.10.10.1:3128
export https_proxy=http://10.10.10.1:3128

# Navigate to ops directory
cd /home/internal-vault-admin/ops

# Run Ansible playbook
ansible-playbook -i inventory.ini deploy.yml
```

Save the job and click "Build Now".

Expected Result: Build should succeed (blue ball). Check console output for Ansible success message.

Technical Deep-Dive: Jenkins Build Process

When you click "Build Now":

1. Jenkins creates workspace directory
2. Executes shell script as `jenkins` user
3. Script runs Ansible playbook
4. Ansible SSHs to DMZ and ensures webapp container is running
5. Jenkins records build result (success/failure)

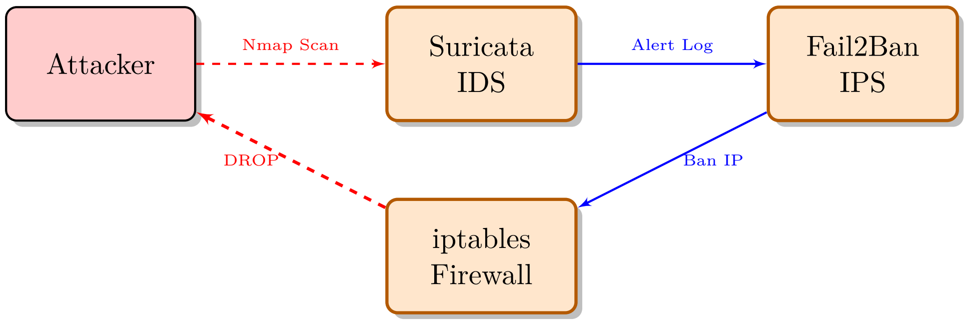
Continuous Deployment: In production, you'd trigger this automatically on Git commits using Jenkins Git plugin and webhooks.

5 Phase 4: Active Defense System

Phase Objective: Detect and Block Threats Automatically

Configure Suricata IDS to monitor network traffic, configure Fail2Ban to parse Suricata alerts, and establish automatic firewall rule injection to block attackers in real-time.

5.1 Active Defense Architecture



Foundational Concept: IDS vs IPS

IDS (Intrusion Detection System): Monitors network traffic, identifies suspicious patterns, logs alerts. *Passive - observes but doesn't block.*
IPS (Intrusion Prevention System): Actively blocks detected threats by updating firewall rules. *Active - stops attacks in real-time.*
Our Implementation:

- **Suricata**: IDS (detects port scans, writes alerts)
- **Fail2Ban**: IPS (reads Suricata alerts, bans IPs)
- **iptables**: Firewall (drops packets from banned IPs)

5.2 Step 4.1: Suricata Configuration

Why This Matters: Why Suricata?

Alternatives: Snort (older, text-based logs), Zeek (powerful but complex).
Suricata Advantages:

- Multi-threaded (better performance)
- Native JSON output (`eve.json`) for modern log tools
- Built-in protocol analyzers (HTTP, TLS, DNS)
- Active community and rule updates

On DMZ-Bastion:

Listing 29: Configure Custom Rules

```
# 1. Disable the default noisy rule set to prevent false-positives
sudo mv /etc/suricata/suricata.rules /etc/suricata/suricata.rules.disabled

# 2. Create a high-fidelity rule file
sudo nano /var/lib/suricata/rules/local.rules
```

Paste this content (Tuned for Browser vs Attack distinction):

```
# Detect Nmap SYN Scan (Threshold 100 to allow legitimate browsers)
```

```
alert tcp any any -> $HOME_NET any (msg:"ET SCAN Nmap SYN Scan"; flags:S,12;
    threshold: type both, track by_src, count 100, seconds 5; sid:1000001; rev:1;)
```

Listing 30: Edit Suricata Configuration

```
sudo nano /etc/suricata/suricata.yaml
```

Find and modify these settings:

```
# Define your network (around line 16)
vars:
  address-groups:
    HOME_NET: "[192.168.192.168]"
    EXTERNAL_NET: "!$HOME_NET"

# Configure rule loading (around line 1900)
default-rule-path: /var/lib/suricata/rules
rule-files:
  - local.rules
# - suricata.rules (Ensure this is commented out)

# Set interface to monitor (around line 580)
af-packet:
  - interface: enp0s8  # Host-Only adapter
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes

# Ensure logging is enabled (around line 400)
outputs:
  - fast:
      enabled: yes
      filename: fast.log
  - eve-log:
      enabled: yes
      filetype: regular
      filename: eve.json
```

Listing 31: Restart Suricata and Verify

```
sudo systemctl restart suricata

# Verify the configuration is valid
sudo suricata -T -c /etc/suricata/suricata.yaml
```

Technical Deep-Dive: Suricata Detection Rules & Thresholds

Suricata uses signature-based detection. Rules define patterns to match in network traffic. In this configuration, we set `count 100` for scan detection.
Reasoning: Modern browsers loading Jenkins dashboards open dozens of parallel connections. A low count (e.g., 10) causes false-positive bans for legitimate users. An Nmap scan hits 1,000 ports almost instantly; a threshold of 100 successfully distinguishes an attacker from a user.

Critical: Troubleshooting: If Suricata Fails

If the service fails to start or detects no traffic:
1. Validate Config: Run test mode to check for syntax errors: `sudo suricata -T -c /etc/suricata/suricata.yaml`
2. Check Interface: Verify packets are reaching the VM: `sudo tcpdump -i enp0s8`
3. Watch Logs: Real-time alert monitoring: `sudo tail -f /var/log/suricata/fast.log`

5.3 Step 4.2: Fail2Ban Configuration

Why This Matters: How Fail2Ban Works

Process Flow:

1. Fail2Ban tails log files in real-time
2. Uses regex patterns (filters) to extract attacker IPs
3. Counts failures per IP
4. After threshold reached, executes action (iptables ban)
5. After ban time expires, removes iptables rule

Why Not Just Suricata? Suricata detects and logs. Fail2Ban enforces consequences (blocks traffic). Separation of concerns: detection vs prevention.

5.3.1 Create Suricata Filter

On DMZ-Bastion:

Listing 32: Create Fail2Ban Filter for Suricata

```
sudo nano /etc/fail2ban/filter.d/suricata.conf
```

Paste this content:

```
[Definition]
# Regex to extract IP from Suricata fast.log format
failregex = ^.*\{TCP\} <HOST>:\d+ -> .*$
ignoreregex =
```

Technical Deep-Dive: Regular Expression Testing

Test your regex before deploying:

```
# Create test log entry
echo "12/23/2024-10:15:30 [**] [1:2100498:7] GPL SCAN nmap XMAS [**] {TCP}
    192.168.192.1:54321 -> 192.168.192.168:80" > /tmp/test.log

# Test filter
sudo fail2ban-regex /tmp/test.log /etc/fail2ban/filter.d/suricata.conf
```

Should show: `Failregex: 1 total, 1 matched`

5.3.2 Create Fail2Ban Jail

Listing 33: Create Jail Configuration

```
sudo nano /etc/fail2ban/jail.local
```

Paste this content:

```
[suricata]
enabled = true
filter = suricata
logpath = /var/log/suricata/fast.log
maxretry = 1
findtime = 600
bantime = 600
# CRITICAL: Prevent banning the Gateway, Internal Network, and Localhost
ignoreip = 127.0.0.1/8 10.10.10.0/24 192.168.192.168
action = iptables-allports[name=suricata, protocol=all]
```

Configuration Explained:

- `enabled = true`: Activate this jail
- `filter = suricata`: Use `/etc/fail2ban/filter.d/suricata.conf` filter
- `logpath`: File to monitor
- `maxretry = 1`: Ban after 1 detected alert (aggressive for demos)
- `findtime = 600`: Count alerts within 600 seconds (10 minutes)
- `bantime = 600`: Ban for 600 seconds (10 minutes)
- `action = iptables-allports`: Block all ports (not just the attacked one)

Foundational Concept: Tuning Ban Parameters

maxretry: Balance between false positives and security.
- `1`: Ultra-aggressive (any single alert = ban). Good for demos, risky for production.
- `3-5`: Moderate (reduces false positives from legitimate scanners).
- `10+`: Conservative (only bans persistent attackers).

Production Recommendation: Start with `maxretry = 5`, adjust based on false positive rate.
findtime: Time window for counting retries. Shorter = stricter (less tolerance for repeated attempts).
bantime: Duration of ban. Can use `-1` for permanent ban, or `86400` for 24 hours.

Listing 34: Restart Fail2Ban

```
sudo systemctl restart fail2ban

# Check status
sudo systemctl status fail2ban

# Verify jail is active
sudo fail2ban-client status
```

Expected output should include: `Jail list:  suricata`

Listing 35: Check Suricata Jail Details

```
sudo fail2ban-client status suricata
```

Should show:

```
Status for the jail: suricata
|- Filter
|  |- Currently failed: 0
|  ‘- Total failed:     0
```

```
`- Actions
   |- Currently banned: 0
   `- Total banned:     0
```

5.4 Step 4.3: Testing Active Defense

Critical: Attack Simulation

We will now simulate an attack to verify the defense system works. This will temporarily ban your Windows host IP.
Important: Wait 10 minutes (bantime) or manually unban yourself: `sudo fail2ban-client set suricata unbanip 192.168.192.1`

On Windows Host, open PowerShell or Command Prompt:

Listing 36: Run Nmap Port Scan

```
nmap -sS 192.168.192.168
```

Parameter Explanation:

- `-sS`: SYN scan (half-open scan, stealthier than full TCP connect)
- `192.168.192.168`: Target IP (DMZ)

Expected Behavior:

1. Scan starts, probes multiple ports
2. After 2-3 seconds, scan hangs or times out
3. No ports reported as open (even though port 80 is open)

On DMZ-Bastion, verify the ban:

Listing 37: Check Fail2Ban Status

```
sudo fail2ban-client status suricata
```

Should show:

```
Currently banned: 1
Total banned:     1
Banned IP list:   192.168.192.1
```

Listing 38: Verify iptables Rule

```
sudo iptables -L -n | grep 192.168.192.1
```

Should show a DROP rule:

```
DROP       all  --  192.168.192.1          0.0.0.0/0
```

Listing 39: Check Suricata Log

```
sudo tail -20 /var/log/suricata/fast.log
```

Should show alerts like:

```
12/23/2024-15:42:11 [**] [1:2100498:7] GPL SCAN nmap XMAS [**]
{TCP} 192.168.192.1:54321 -> 192.168.192.168:80
```

Technical Deep-Dive: What Just Happened?

1. Nmap sent SYN packets to multiple ports on DMZ
2. Suricata detected the port scan pattern
3. Wrote alert to `fast.log`
4. Fail2Ban (tailing the log) matched the regex
5. After 1 match (maxretry=1), executed iptables action
6. iptables inserted DROP rule for source IP
7. All subsequent packets from 192.168.192.1 are silently dropped
8. Scan hangs because no responses received (packets dropped at kernel level)

Response Time: Typically 1-3 seconds from attack to ban. Fast enough to prevent most automated attacks from completing.

Listing 40: Manually Unban Your IP (if needed)

```
sudo fail2ban-client set suricata unbanip 192.168.192.1
```

Verify access restored:

```
# From Windows, test connection
curl http://192.168.192.168
```

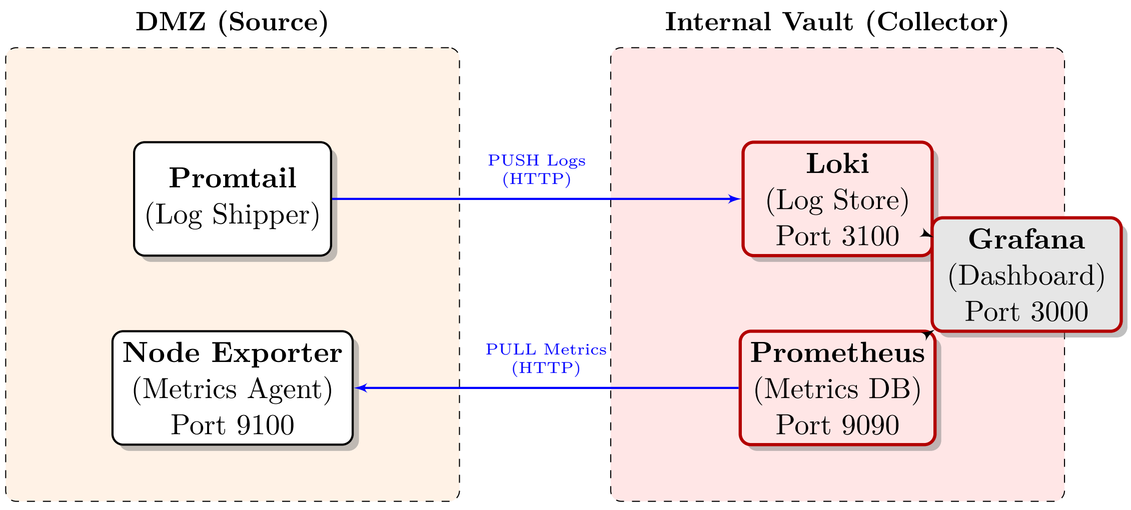
Should successfully connect to webapp.

6 Phase 5: Observability & Logging

Phase Objective: Centralized Security Monitoring

We will construct a "Single Pane of Glass" to monitor the health and security of the infrastructure. Unlike traditional logging where you check individual servers, we will aggregate all metrics and security logs into a central dashboard inside the Internal Vault.

6.1 Observability Architecture


DMZ (Source)
Internal Vault (Collector)
Promtail (Log Shipper)
PUSH Logs (HTTP)
Loki (Log Store) Port 3100
Grafana (Dashboard) Port 3000
Node Exporter (Metrics Agent) Port 9100
PULL Metrics (HTTP)
Prometheus (Metrics DB) Port 9090

Foundational Concept: Push vs. Pull Architecture

This architecture uses two distinct methods for data collection:
1. The Pull Model (Prometheus): The central server (Prometheus) actively connects to the DMZ (Node Exporter) every 15 seconds to "scrape" or download CPU/RAM stats.
Advantage: If the DMZ goes down, Prometheus knows immediately because the scrape fails.
2. The Push Model (Promtail/Loki): The agent on the DMZ (Promtail) watches log files. When a new line is written, it immediately "pushes" it to the Internal Vault (Loki).
Advantage: Ideal for event-based data like intrusion alerts.

6.2 Step 5.1: Deploying the Collector Stack

We will deploy the "Brain" of our monitoring system on the Internal Vault. This consists of three containers:

- **Prometheus:** Time-series database for metrics.
- **Loki:** Index-free log aggregation system (like Splunk, but lightweight).
- **Grafana:** The visual web interface.

Critical: Docker Proxy Requirement

The Docker Daemon runs as a background service. Even though you configured the proxy for your user shell in Phase 2, the Docker Daemon **cannot** reach the internet to pull images unless explicitly configured.

On Internal-Vault:

Listing 41: Configure Docker Daemon Proxy

```
# Create systemd directory for Docker overrides
```

```
sudo mkdir -p /etc/systemd/system/docker.service.d

# Create proxy configuration
sudo nano /etc/systemd/system/docker.service.d/http-proxy.conf
```

Paste this content:

```
[Service]
Environment="HTTP_PROXY=http://10.10.10.1:3128"
Environment="HTTPS_PROXY=http://10.10.10.1:3128"
Environment="NO_PROXY=localhost,127.0.0.1,10.10.10.1,10.10.10.2"
```

Listing 42: Apply Proxy and Install Compose

```
# Reload systemd and restart Docker
sudo systemctl daemon-reload
sudo systemctl restart docker

# Install Docker Compose
sudo apt install -y docker-compose
```

Create Project Files:

Listing 43: Create Monitoring Workspace

```
mkdir -p ~/monitor && cd ~/monitor
nano docker-compose.yml
```

Paste this content (Using Host Networking):

```
version: '3'
services:
  prometheus:
    image: prom/prometheus:latest
    container_name: prometheus
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    network_mode: "host" # Shares IP with VM (Port 9090)
    restart: always

  loki:
    image: grafana/loki:latest
    container_name: loki
    network_mode: "host" # Shares IP with VM (Port 3100)
    restart: always

  grafana:
    image: grafana/grafana:latest
    container_name: grafana
    network_mode: "host" # Shares IP with VM (Port 3000)
    environment:
      - GF_SECURITY_ADMIN_PASSWORD=admin
    restart: always
```

Technical Deep-Dive: Why "network_mode: host"?

Usually, Docker containers have their own private IPs. However, we need Prometheus to scrape targets and Loki to receive logs on the VM's specific IP address. Using `network_mode: "host"` removes network isolation for these containers, allowing them to bind directly to the VM's network interface (10.10.10.2).

Listing 44: Configure Prometheus Scraper

```
nano prometheus.yml
```

Paste this content:

```
global:
  scrape_interval: 15s

scrape_configs:
  - job_name: 'dmz_bastion'
    static_configs:
      # Target the DMZ IP on port 9100 (Node Exporter)
      - targets: ['10.10.10.1:9100']
        labels:
          instance: 'dmz-bastion'
```

Listing 45: Start the Stack

```
sudo docker-compose up -d
```

6.3 Step 5.2: Configuring DMZ Agents

Now we must install the "Agents" on the DMZ that will send data to our collectors.

6.3.1 Part A: Promtail (Log Shipper)

Promtail reads log files (like Suricata's `eve.json` and Cowrie's `cowrie.json`) and pushes them to Loki.

On DMZ-Bastion:

Listing 46: Create Promtail Configuration

```
nano ~/promtail-config.yaml
```

Paste this content:

```
server:
  http_listen_port: 9080
  grpc_listen_port: 0

positions:
  filename: /tmp/positions.yaml

clients:
  # Push logs to Internal Vault IP on Loki port
  - url: http://10.10.10.2:3100/loki/api/v1/push

scrape_configs:
  - job_name: security_logs
  static_configs:
   - targets:
     - localhost
    labels:
     job: suricata
     host: dmz-bastion
     __path__: /var/log/suricata/eve.json

 - job_name: honeypot_logs
  static_configs:
   - targets:
     - localhost
    labels:
     job: honeypot
     host: dmz-bastion
     __path__: /var/log/cowrie/cowrie.json
```

Listing 47: Run Persistent Promtail Container

```
# This command is robust: it restarts on boot, bypasses the proxy for local
# traffic, and mounts both suricata and cowrie log directories.
sudo docker run -d \
  --name promtail \
  --restart unless-stopped \
  --network host \
  -e NO_PROXY="localhost,127.0.0.1,10.10.10.1,10.10.10.2" \
  -v /var/log/suricata:/var/log/suricata:ro \
  -v /var/log/cowrie:/var/log/cowrie:ro \
  -v /home/dmz-bastion-admin/promtail-config.yaml:/etc/promtail/config.yml:ro \
  grafana/promtail:latest \
  -config.file=/etc/promtail/config.yml
```

6.3.2 Part B: Node Exporter (Metrics Agent)

Node Exporter is a lightweight binary that exposes system metrics (CPU, Disk, RAM) as an HTTP endpoint.

Listing 48: Install Node Exporter

```
cd ~
wget
    https://github.com/prometheus/node_exporter/releases/download/v1.6.1/node_exporter-1.6.1.linux-

tar xvfz node_exporter-1.6.1.linux-amd64.tar.gz
cd node_exporter-1.6.1.linux-amd64
./node_exporter &
```

Verify: `curl localhost:9100/metrics` should scroll a large list of data.

6.4 Step 5.3: Accessing the Dashboard

The Grafana dashboard runs on `10.10.10.2:3000`. Since the Internal Vault has no public IP, your Windows browser cannot reach it directly. We must use an SSH Tunnel.

Technical Deep-Dive: Deep Dive: SSH Local Port Forwarding

The Concept: We can trick your Windows computer into thinking Grafana is running locally. **The Command:** `-L 3000:10.10.10.2:3000`

1. You connect to the DMZ (192.168.192.168).
2. SSH listens on your Windows **localhost:3000**.
3. Any traffic sent to Windows:3000 is encrypted, sent to DMZ, and forwarded to **10.10.10.2:3000**.

1. Establish Tunnel (Run on Windows Host):

```
ssh -L 3000:10.10.10.2:3000 dmz-bastion-admin@192.168.192.168
```

Keep this terminal window open. Note: After Phase 6, you must use port 2222 for this command.

2. Access Grafana: Open Chrome/Edge on Windows and go to **http://localhost:3000**.

- **Login:** admin / admin
- **Skip** password change if desired.

3. Connect Data Sources:

- Go to **Connections** → **Data Sources** → **Add data source**.
- **Prometheus:** URL = `http://localhost:9090`. (Use localhost because Grafana and Prometheus are on the same VM). Click "Save & Test".
- **Loki:** URL = `http://localhost:3100`. Click "Save & Test".

You now have a live view of the entire secure infrastructure.

7 Phase 6: Forensics & Honeypot

◎ Phase Objective: Trap and Analyze Attackers

We will swap the default SSH ports. The real administrative SSH will move to a non-standard port (2222), and we will place the Cowrie Honeypot on standard port 22. This ensures automated bots attacking the default port hit the trap immediately.

7.1 Step 6.1: Move Real SSH to Port 2222

⚠ Critical: Risk of Lockout

We are changing the SSH port. If the configuration fails, you could lose remote access. Ensure you verify the new port works before closing your current session.

On DMZ-Bastion:

Listing 49: Modify SSH Daemon Config

```
# Use sed to replace Port 22 with Port 2222
sudo sed -i 's/#Port 22/Port 2222/' /etc/ssh/sshd_config
# Fallback: if line is just "Port 22", replace that too
sudo sed -i 's/Port 22/Port 2222/' /etc/ssh/sshd_config
```

Listing 50: Restart SSH Service

```
sudo systemctl restart ssh
```

Verify you can still connect (Open a NEW terminal window):

```
# Replace with your DMZ-Bastion IP if different
ssh -p 2222 dmz-bastion-admin@192.168.192.168
```

7.2 Step 6.2: Deploy Cowrie on Port 22

Now that Port 22 is free, we will bind the Cowrie Honeypot to it. The official `cowrie/cowrie` image requires explicit configuration via environment variables to enable JSON logging and disable its default SSH proxying behavior.

On DMZ-Bastion:

Listing 51: Run Persistent Cowrie Container

```
# Stop and remove any old Cowrie containers first
sudo docker stop cowrie >/dev/null 2>&1 || true
sudo docker rm cowrie >/dev/null 2>&1 || true

# Run Cowrie using environment variables and a restart policy
docker run -d -p 22:2222 \
  --restart unless-stopped \
  -v /var/log/cowrie:/cowrie/var/log/cowrie \
  -e COWRIE_OUTPUT_JSONLOG_ENABLED=true \
  -e COWRIE_OUTPUT_JSONLOG_LOGFILE=/cowrie/var/log/cowrie/cowrie.json \
  -e COWRIE_SSH_PROXY_ENABLED=false \
  --name cowrie \
  cowrie/cowrie
```

Configuration Explained:

- `-p 22:2222`: Exposes the honeypot on host port 22.
- `-restart unless-stopped`: Ensures the honeypot starts on boot.
- `-v ...`: Mounts the host log directory into the container.
- `-e COWRIE_..._ENABLED=true`: Enables the JSON log output module.
- `-e COWRIE_..._LOGFILE=...`: Tells Cowrie where to write the JSON log file inside the container.
- `-e COWRIE_..._PROXY_ENABLED=false`: **Crucial.** Disables the feature that forwards known usernames to the real SSH server, ensuring all attempts are trapped.

Why This Matters: Strategic Port Swapping

The Trap: 99% of automated malware scans port 22. By placing Cowrie here, we guarantee high-fidelity alerts. **The Defense:** Moving real SSH to 2222 reduces log noise and prevents log-filling DoS attacks against the real daemon.

7.3 Step 6.3: Python Forensic Analysis

This script queries the Loki database on the Internal Vault to find and parse alerts from the honeypot.

On Internal-Vault:

Listing 52: Create Python Script

```
nano ~/forensics.py
```

Paste this content:

Listing 53: forensics.py

```
import requests
import json
import time
import os

# Automatically bypass proxy for localhost to prevent connection errors
os.environ["NO_PROXY"] = "localhost,127.0.0.1"

LOKI_URL = "http://localhost:3100/loki/api/v1/query_range"

def analyze_honeypot_logs():
    print("--- QUERYING SECURE LOG VAULT FOR HONEYPOT ACTIVITY ---")

    query = '{job="honeypot"}'
    params = {
        'query': query,
        'limit': 50,
        'start': str(int((time.time() - 3600) * 1e9)), # Last 1 hour
        'direction': 'forward'
    }

    try:
        response = requests.get(LOKI_URL, params=params)
        response.raise_for_status()
        data = response.json()
        found_attack = False
```

```python
        for stream in data.get('data', {}).get('result', []):
            for value_pair in stream['values']:
                log_line = value_pair[1]

                try:
                    # Safely parse JSON, skipping any non-JSON lines
                    log_entry = json.loads(log_line)
                except json.JSONDecodeError:
                    continue

                if log_entry.get('eventid') == 'cowrie.login.failed':
                    found_attack = True
                    ip = log_entry.get('src_ip', 'Unknown')
                    user = log_entry.get('username', 'Unknown')
                    passwd = log_entry.get('password', 'Unknown')

                    print(f"\n[!!!] ALERT: FAILED LOGIN DETECTED")
                    print(f"      - Attacker IP: {ip}")
                    print(f"      - Username:    {user}")
                    print(f"      - Password:    {passwd}")

        if not found_attack:
            print("\n[-] No failed login attempts found in the last hour.")

    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    analyze_honeypot_logs()
```

8 Demonstration Scenario: "From Code Commit to Active Threat Defense"

Strategic Vision: Demonstration Overview

This scenario demonstrates a complete end-to-end lifecycle: verifying network isolation, deploying code via CI/CD, repelling a live attack, and analyzing forensic data.

8.1 Phase 1: Verifying the Air Gap

Step 1: Demonstrate Network Isolation from the Windows Host

- **Action:** Open a Command Prompt on your Windows 11 host.
- **Command (Success):** Ping the DMZ-Bastion's management IP.

```
ping 192.168.192.168
```

- **Command (Failure):** Attempt to ping the Internal-Vault's IP.

```
ping 10.10.10.2
```

- **Result:** "Request timed out." This proves the Internal-Vault is invisible to the outside world.

8.2 Phase 2: Automated Deployment (CI/CD)

Step 2: Show Initial State

- **Action:** Open browser to `http://192.168.192.168`.
- **Result:** View the current web application.

Step 3: Trigger Deployment

- **Action:** Navigate to `http://192.168.192.168/jenkins` (via Reverse Proxy).
- **Instruction:** Click "Build Now" on the **DeployWebApp** job.
- **Observation:** Jenkins executes the Ansible playbook on the Internal-Vault, which connects to the DMZ to update the container.

Step 4: Verify Success

- **Action:** Refresh `http://192.168.192.168`.
- **Result:** The application is running, demonstrating a zero-touch deployment.

8.3 Phase 3: Real-Time Active Threat Mitigation

Step 5: Launch Attack

- **Action:** From Windows Command Prompt, launch a stealth scan.

```
nmap -sS 192.168.192.168
```

- **Result:** The scan will hang indefinitely.

Step 6: Witness Countermeasure

- **Action:** On DMZ-Bastion, check the IPS status.

```
sudo fail2ban-client status suricata
```

- **Result:** Your Windows IP (`192.168.192.1`) appears in the "Banned IP list".

Step 7: Verify Firewall Block

- **Action:** Check the kernel firewall rules.

```
sudo iptables -L -n | grep '192.168.192.1'
```

- **Result:** A `DROP` rule exists for your IP.

Step 8: Unban to Proceed

- **Action:** Manually remove the ban.

```
sudo fail2ban-client set suricata unbanip 192.168.192.1
```

8.4 Phase 4: Honeypot & Forensic Analysis

Step 9: Simulate Brute-Force Attack

- **Action:** From Windows, SSH into the **default port 22**.

```
ssh attacker@192.168.192.168
```

- **Explanation:** Real SSH is on port 2222. Port 22 is the Cowrie honeypot.

⚠ Critical: Re-Establishing SSH Tunnel for Grafana

After moving the real SSH service to port 2222 in Phase 6, the original SSH tunnel command will no longer work. You must close the old terminal and run this new command from Windows to access Grafana.

```
ssh -p 2222 -L 3000:10.10.10.2:3000 dmz-bastion-admin@192.168.192.168
```

Step 10: Show Centralized Logs

- **Action:** Access Grafana at `http://localhost:3000` (via the new SSH tunnel).
- **Query:** Use Loki to search for `{job="honeypot"}`. The attack logs appear instantly.

Step 11: Run Automated Forensics

- **Action:** On Internal-Vault, run the analysis script. (The script now handles proxy settings automatically).

```
python3 ~/forensics.py
```

- **Result:** The script alerts on the attack, extracting the attacker's IP and credentials.

⚠ Critical: Infrastructure Maintenance: SSH Ports

After moving the SSH port to 2222 in Phase 6, the initial Ansible setup will require an update. You must edit `~/ops/inventory.ini` on the Internal-Vault and change `ansible_port=22` to `ansible_port=2222` to maintain management connectivity.

9 Conclusion

Strategic Vision: Architecture Realized

This project successfully constructed a miniature, cloud-agnostic model of a secure, modern data center. By meticulously integrating infrastructure-as-code, automated CI/CD pipelines, and an active defense system, it demonstrates the core principles of DevSecOps in a tangible, operational environment.