MASTER IMPLEMENTATION MANUAL

Autonomous DevSecOps Architecture
with Active Threat Mitigation

(PART IV: Automated System Maintenance)

Ashish Kumar Yadav

January 27, 2026

♟ Strategic Vision: Why Ansible? The "Internal Mechanic"

In Part 3, we built a Python "Watchdog." That watchdog checks **Uptime** (Network Layer) — it tells you if the door is locked.
In Part 4, we re-introduce Ansible for **Maintenance** (System Layer) — it is the mechanic that goes inside to fix the engine. Python scripts cannot easily run ‘apt-get upgrade‘ or check disk usage percentage across remote servers. Ansible handles this natively.
We will reuse the **SMTP Tunnel** created in Part 3 to allow Ansible to email us a "Daily Health Report" covering:

1. **Disk Usage:** Ensuring logs haven’t filled the drive.
2. **Patch Status:** Checking for security updates.
3. **Service Integrity:** Verifying configuration files.

Contents

1 Phase 1: Re-establishing Connectivity

Phase Objective: Target the Management Port

In Part 3, we moved the real SSH to port **2222**. We must point Ansible to this new port so it can bypass the Honeypot.

1.1 Step 1.1: Update Inventory

On Internal-Vault:

Listing 1: Edit Inventory File

```
nano ~/ops/inventory.ini
```

Update content (Note the port change):

```
[dmz]
# Using IP to ensure connectivity even if DNS fails
10.10.10.1 ansible_port=2222 \
           ansible_user=dmz-bastion-admin \
           ansible_password=admin \
           ansible_become_password=admin \
           ansible_ssh_common_args='-o StrictHostKeyChecking=no'
```

1.2 Step 1.2: Verify Real Access

Listing 2: Test Connection

```
ansible -i ~/ops/inventory.ini dmz -m ping
```

Expected Result: `SUCCESS`. If this fails, check your DMZ Firewall settings from Part 3.

2 Phase 2: The Maintenance Playbook

Phase Objective: System Health & Reporting

This playbook performs tasks that an external Python script cannot do. It checks internal disk usage and package versions, then uses the SMTP Tunnel to email a report.

2.1 Step 2.1: Create the Playbook

Listing 3: Create Maintenance Playbook

```
mkdir -p ~/ops/maintenance
nano ~/ops/maintenance/daily_health.yml
```

Listing 4: Daily Health Playbook

```
---
- name: Daily System Health Check
  hosts: dmz
  become: yes
  vars:
    # Email Credentials (Reusing from Part 3)
    smtp_host: "smtp.gmail.com"
    smtp_port: 587
    sender_email: "your.email@gmail.com"
    sender_pass: "xxxx xxxx xxxx xxxx" # App Password
    recipient_email: "your.email@gmail.com"

  tasks:
    # --- TASK 1: CHECK DISK USAGE ---
    # Python Watchdog cannot see inside the disk. Ansible can.
    - name: Get Disk Usage
      shell: "df -h / | tail -1 | awk '{print $5}'"
      register: disk_usage

    # --- TASK 2: CHECK FOR SECURITY UPDATES ---
    - name: Update Apt Cache
      apt:
        update_cache: yes

    - name: Check for Upgradable Packages
      shell: "apt list --upgradable | grep -v 'Listing...' | wc -l"
      register: updates_count

    # --- TASK 3: SEND EMAIL REPORT ---
    # Uses the Part 3 NAT Tunnel to reach Gmail
    - name: Email Health Report
      mail:
        host: "{{ smtp_host }}"
        port: "{{ smtp_port }}"
        username: "{{ sender_email }}"
        password: "{{ sender_pass }}"
        to: "{{ recipient_email }}"
        subject: "[Ansible] Daily Health Report: {{ inventory_hostname }}"
        body: |
          System Health Report for {{ inventory_hostname }}
          ---------------------------------------------
          Disk Usage (Root): {{ disk_usage.stdout }}
          Security Updates Pending: {{ updates_count.stdout }}
```

```
        Status:
        - SSH Port: 2222 (Secured)
        - Honeypot Port: 22 (Active)

        This is an automated message from the Internal Vault.
      secure: starttls
    delegate_to: localhost # Send from the Vault, about the DMZ
    become: no
```

Technical Deep-Dive: Why "delegate_to: localhost"?

The ‘mail‘ module runs on the machine sending the email. We want the **Internal Vault** to send the email because we configured its DNS and Routing in Part 3. The data (`disk_usage`) comes from the DMZ, but the email is sent by the Vault.

3 Phase 3: Automation via Jenkins

Phase Objective: Scheduled Maintenance

We don't want to run this manually. We will schedule Jenkins to run this Ansible playbook every day, ensuring we are notified of disk issues or patch requirements immediately.

3.1 Step 3.1: Configure Jenkins Job

1. Open Jenkins: `https://jenkins.corp.local`
2. **New Item** -> `Daily-System-Maintenance` -> **Freestyle Project**.
3. **Build Triggers:** Select **Build periodically**.
4. **Schedule:** `H 8 * * *` (Runs every day at 8:xx AM).
5. **Build Steps:** Add "Execute shell".

Paste the Command:

```
cd /home/internal-vault-admin/ops
ansible-playbook -i inventory.ini maintenance/daily_health.yml
```

3.2 Step 3.2: Verify

1. Click **Build Now** to test immediately.
2. Check the **Console Output**.
3. Check your **Gmail Inbox**.

Expected Email Content:

```
Subject: [Ansible] Daily Health Report: 10.10.10.1

System Health Report for 10.10.10.1
--------------------------------------------
Disk Usage (Root): 14%
Security Updates Pending: 3
...
```

4 Conclusion

This completes the architecture. We now have a clear separation of duties:

- **Python Watchdog (Part 3):** Checks external availability (Is the website up?).
- **Ansible Engine (Part 4):** Checks internal health (Is the disk full? Are we patched?).

By reusing the secure SMTP tunnel for Ansible reporting, we have fully integrated our Operations toolset with our Security infrastructure.