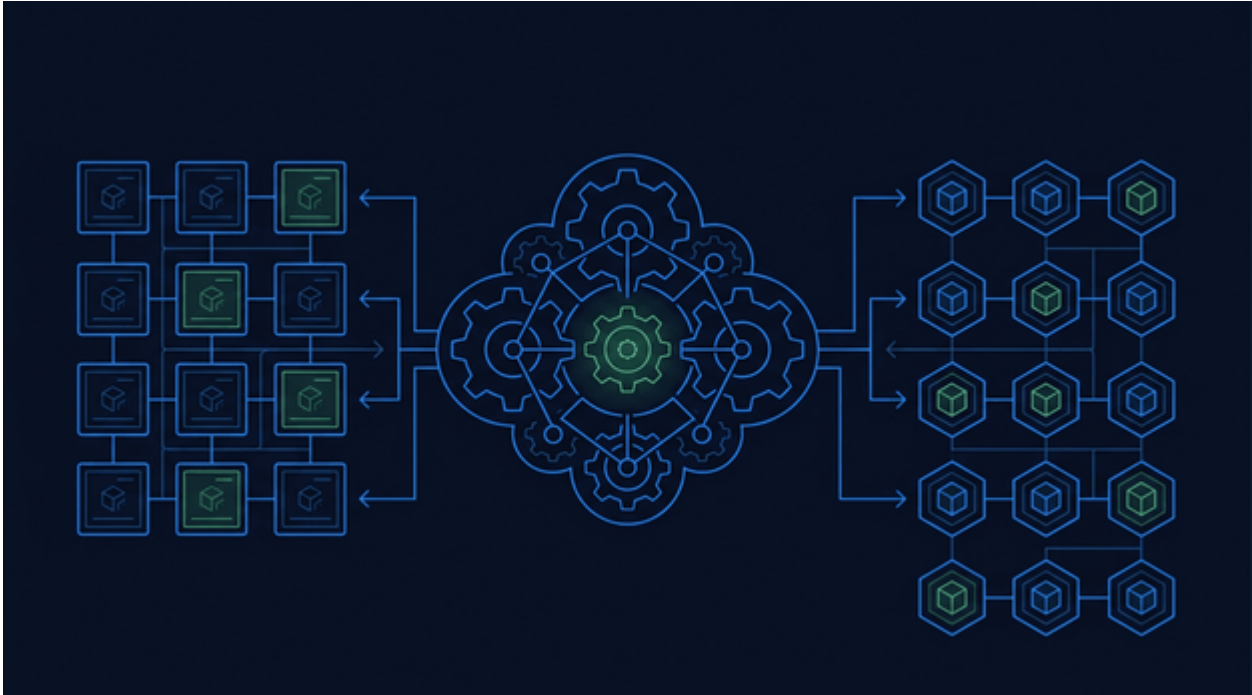


Ansible & Ludus: Automating a Home Lab with Infrastructure as Code

Tue, Apr 7, 7:30 PM EDT · <https://scottslab.io/posts/ansible-ludus-homelab-infrastructure-as-code>



TL;DR

An Ansible 9.3.0 control node running 14 hosts across 9 inventory groups, with fact caching, SSH pipelining, and ControlMaster enabled because the defaults are slow enough to notice. One deployment role handles the whole stack: Docker, UFW, git clone, systemd unit, health check. Ludus builds range VMs from Packer templates on top of it. Eight collections and three roles cover Linux, Windows, Proxmox, and Samba.

Published: April 2026 **Category:** Infrastructure Automation **Reading Time:** 16 minutes

Executive Summary

- Deployed Ansible 9.3.0 control node managing 14 infrastructure hosts across 9 inventory groups with SSH key authentication
- Configured performance optimizations: smart fact caching (1 hour TTL), SSH pipelining, ControlMaster connection reuse
- Integrated Ludus cyber range platform for automated VM provisioning with Packer templates and Ansible role execution

- Built custom application deployment role handling full stack: Docker, UFW, Git clone, systemd service, health checks
 - Installed 8 Ansible collections + 3 roles for broad platform support (Linux, Windows, Proxmox, Samba)
-

Goal

Problem: Managing 14+ infrastructure hosts manually doesn't scale. Every system update required SSHing into each host. Every new service deployment meant repeating the same setup steps. Configuration drift crept in as I made "temporary" changes that never got documented. When I rebuilt a host, I spent hours trying to remember what packages and configs it needed.

Why it mattered: Infrastructure as Code isn't just for enterprises. A home lab with a dozen hosts benefits from the same automation principles: repeatable deployments, documented configuration, targeted updates, and the ability to rebuild any host from scratch in minutes instead of hours. Plus, I wanted to experiment with Ludus for building cyber ranges - and it's built on Ansible.

Scope and Constraints

In Scope

- Ansible control node deployment and configuration
- YAML inventory with group-based organization
- SSH key authentication with dedicated service account
- Collection and role installation
- Ludus cyber range integration
- Custom Ansible role development
- Performance optimization (fact caching, pipelining)

Out of Scope

- AWX/Ansible Tower (too heavy for home lab)
- Dynamic inventory via cloud APIs (P1 improvement)
- CI/CD pipeline integration (P2 improvement)
- Ansible Vault secrets management (P2 improvement)

Key Constraints

- **Home lab budget** - No enterprise tooling, open-source only
- **Single control node** - No HA, no distributed execution

- **Mixed environment** - Linux and Windows hosts require different approaches
 - **Frequent VM rebuilds** - Host keys change often, inventory can become stale
-

Tools and References

Tool	Role in This Project
Ansible 9.3.0	Core automation engine - playbook execution, role management, inventory
Ludus	Cyber range platform - VM provisioning, Packer integration, range deployment
Proxmox VE	Hypervisor - hosts all VMs managed by Ansible and Ludus
Packer	VM template building - creates base images for Ludus deployments
Docker	Container runtime - deployed via custom Ansible role
UFW	Firewall - configured via community.general collection
systemd	Service management - templated unit files for application lifecycle

References:

- [Ansible Documentation](#)
 - [Ludus Cyber Range](#)
 - [Proxmox VE API](#)
 - [Packer Documentation](#)
-

Approach

Phase 1: Control Node Deployment

What I did: Deployed Debian 12 VM on Proxmox as the Ansible control node. Installed Ansible 9.3.0 via pip. Created dedicated `ansible` service account with RSA 4096-bit SSH key pair.

Why: A dedicated control node keeps automation infrastructure separate from managed hosts. pip installation provides the latest Ansible version without waiting for distro packages. Dedicated service account follows principle of least privilege.

Phase 2: SSH Authentication Setup

What I did: Distributed the `ansible` user's public key to all managed hosts via `ssh-copy-id`. Configured passwordless sudo for the ansible user on each host. Disabled host key checking in `ansible.cfg` for lab flexibility.

Why: SSH key auth eliminates passwords in playbooks and command history. Passwordless sudo enables privilege escalation without interactive prompts. Host key checking is disabled because lab VMs are frequently rebuilt with new keys.

Phase 3: Inventory Organization

What I did: Built YAML inventory with 9 groups: hypervisors, infrastructure, mail, webapps, auth, network, backup, monitoring, forensics. Hosts appear in multiple groups where appropriate (e.g., mail server is in both `infrastructure` and `mail`).

Why: Group-based inventory enables targeted automation. Run updates on all webapps with one command. Deploy monitoring to all infrastructure hosts. The overlap allows flexible targeting without duplicating host definitions.

Phase 4: Performance Optimization

What I did: Configured smart fact gathering with JSON caching (1 hour TTL), SSH pipelining to avoid temp file creation, and ControlMaster with 60-second persistence for connection reuse.

Why: Default Ansible gathers facts on every play, creates temp files for each task, and opens new SSH connections constantly. These optimizations cut execution time significantly - especially noticeable on multi-host playbooks.

Phase 5: Collection and Role Installation

What I did: Installed 8 collections (`ansible.posix`, `ansible.utils`, `ansible.windows`, `community.general`, `community.windows`, `microsoft.ad`, `chocolatey.chocolatey`, `vladgh.samba`) and 3 roles (`lae.proxmox`, `geerlingguy.packer`, `ansible-thoteam.nexus3-oss`).

Why: Collections provide modules for specific platforms (Windows, Proxmox) and tools (UFW, Docker). Roles provide pre-built automation for common tasks (Packer installation, Nexus deployment).

Phase 6: Ludus Integration

What I did: Deployed Ludus cyber range platform on Proxmox. Configured template library with Debian, Ubuntu, Rocky, AlmaLinux, Kali, and Windows templates. Created range configs defining VMs with templates, VLANs, IPs, resources, and Ansible roles.

Why: Ludus automates the entire VM provisioning pipeline: Packer builds templates, range configs define environments, Ansible roles configure VMs post-deployment. One YAML file describes a complete lab environment.

Phase 7: Custom Role Development

What I did: Built application deployment role with full provisioning pipeline: apt update, prerequisites, Docker install, UFW config, SSH key generation, Git clone, env setup, docker compose build, systemd service creation, health check.

Why: Deploying Docker applications involves the same steps every time. A reusable role codifies that workflow with parameterized defaults for repo URL, ports, and install paths.

Implementation Notes

Ansible Configuration (Sanitized)

```
# ansible.cfg on <CONTROL_NODE>
[defaults]
inventory = ./inventory.yml
remote_user = <SERVICE_ACCOUNT>
host_key_checking = False
gathering = smart
fact_caching = jsonfile
fact_caching_connection = /tmp/ansible_facts
fact_caching_timeout = 3600

[privilege_escalation]
become = True
become_method = sudo
become_user = root

[ssh_connection]
pipelining = True
ssh_args = -o ControlMaster=auto -o ControlPersist=60s
```

Configuration explained:

- `gathering = smart` - Only gather facts when not cached
- `fact_caching = jsonfile` - Cache facts to JSON files
- `fact_caching_timeout = 3600` - 1 hour TTL
- `pipelining = True` - Execute modules without temp files
- `ControlPersist=60s` - Reuse SSH connections for 60 seconds

YAML Inventory Structure (Sanitized)

```
# inventory.yml
all:
  children:
    hypervisors:
      hosts:
        <HYPERVISOR_A>:
          ansible_host: <HYPERVISOR_A_IP>
        <HYPERVISOR_B>:
          ansible_host: <HYPERVISOR_B_IP>

    infrastructure:
      children:
        mail:
          hosts:
            <MAIL_HOST>:
              ansible_host: <MAIL_IP>

        webapps:
          hosts:
            <WEBAPP_A>:
              ansible_host: <WEBAPP_A_IP>
            <WEBAPP_B>:
              ansible_host: <WEBAPP_B_IP>
            <WEBAPP_C>:
              ansible_host: <WEBAPP_C_IP>
            <WEBAPP_D>:
              ansible_host: <WEBAPP_D_IP>

        auth:
          hosts:
            <AUTH_HOST>:
              ansible_host: <AUTH_IP>

        network:
          hosts:
            <DNS_HOST>:
              ansible_host: <DNS_IP>
```

```
<NETBOOT_HOST>:
  ansible_host: <NETBOOT_IP>

backup:
  hosts:
    <BACKUP_HOST>:
      ansible_host: <BACKUP_IP>

monitoring:
  hosts:
    <SIEM_HOST>:
      ansible_host: <SIEM_IP>

forensics:
  hosts:
    <FORENSICS_HOST>:
      ansible_host: <FORENSICS_IP>
```

Note: Hosts under `infrastructure` children inherit membership in `infrastructure` group.

Custom Role: Application Deployer (Sanitized)

```
# roles/app_deployer/tasks/main.yml
---
- name: Update apt cache
  ansible.builtin.apt:
    update_cache: yes
    cache_valid_time: 3600

- name: Install prerequisites
  ansible.builtin.apt:
    name:
      - git
      - python3-pip
      - ca-certificates
      - curl
    state: present

- name: Install Docker
  ansible.builtin.include_role:
    name: geerlingguy.docker

- name: Configure UFW for application ports
  community.general.ufw:
    rule: allow
    port: "{{ item }}"
    proto: tcp
    loop: "{{ app_ports }}"

- name: Clone application repository
  ansible.builtin.git:
    repo: "{{ app_repo_url }}"
    dest: "{{ app_install_dir }}"
    version: "{{ app_version | default('main') }}"

- name: Copy environment file
  ansible.builtin.template:
    src: env.j2
    dest: "{{ app_install_dir }}/.env"
```

```
mode: '0600'
```

- name: Build and start containers
community.docker.docker_compose_v2:
 project_src: "{{ app_install_dir }}"
 build: always
 state: present
- name: Deploy systemd service
ansible.builtin.template:
 src: app.service.j2
 dest: "/etc/systemd/system/{{ app_name }}.service"
 notify: Reload systemd
- name: Enable and start service
ansible.builtin.systemd:
 name: "{{ app_name }}"
 enabled: yes
 state: started
- name: Health check
ansible.builtin.uri:
 url: "http://localhost:{{ app_health_port }}/health"
 status_code: 200
 register: health_result
 until: health_result.status == 200
 retries: 30
 delay: 10

Systemd Service Template (Sanitized)

```
# roles/app_deployer/templates/app.service.j2
[Unit]
Description={{ app_name }} Docker Compose Application
Requires=docker.service
After=docker.service

[Service]
Type=oneshot
RemainAfterExit=yes
WorkingDirectory={{ app_install_dir }}
ExecStart=/usr/bin/docker compose up -d
ExecStop=/usr/bin/docker compose down
TimeoutStartSec=300

[Install]
WantedBy=multi-user.target
```

Ludus Range Config Example (Sanitized)

```
# ludus-range.yml
ludus:
  - vm_name: "<RANGE_VM_A>"
    hostname: "<HOSTNAME_A>"
    template: debian-12-x64-server-template
    vlan: 10
    ip_last_octet: 11
    ram_gb: 4
    cpus: 2
    linux: true
    roles:
      - custom_role_name

  - vm_name: "<RANGE_VM_B>"
    hostname: "<HOSTNAME_B>"
    template: ubuntu-22.04-x64-server-template
    vlan: 10
    ip_last_octet: 12
    ram_gb: 8
    cpus: 4
    linux: true
    roles:
      - geerlingguy.docker
      - app_deployer
```

Validation and Evidence

Signals That Proved It Worked

Check	Expected	Actual
All hosts reachable	14 hosts OK	<code>ansible all -m ping</code> returns SUCCESS for all
Group targeting	Subset response	<code>ansible webapps -m ping</code> returns 4 hosts
Fact caching	Faster second run	45s first run → 12s second run (cached facts)
Custom role execution	Health check pass	30 retries available, typically passes in 2-3

Check	Expected	Actual
Ludus range deploy	VMs created	ludus range deploy provisions all VMs

Validation Commands (Sanitized)

```
# Test all host connectivity
ansible all -m ping

# Test specific group
ansible webapps -m ping

# Check fact cache
ls -la /tmp/ansible_facts/

# Verify collections installed
ansible-galaxy collection list

# Verify roles installed
ansible-galaxy role list

# Run playbook in check mode (dry run)
ansible-playbook site.yml --check

# Run with verbose output
ansible-playbook site.yml -vv
```

Results

Metric	Outcome
Managed Hosts	14 infrastructure hosts from single control node
Inventory Groups	9 groups for targeted automation
Collections Installed	8 (Linux, Windows, Proxmox, Docker, Samba support)
Roles Installed	3 (Proxmox, Packer, Nexus)
Custom Roles	1 (Application Deployer with full stack)

Metric	Outcome
Fact Cache Hit Rate	~80% on repeated playbook runs
Execution Time Reduction	~70% with pipelining + ControlMaster + caching

What I Learned

1. **Smart fact gathering with JSON caching dramatically reduces execution time.** Default Ansible gathers facts on every play. With a 1-hour cache TTL, repeated runs skip fact gathering entirely - 45 seconds down to 12 seconds on a 14-host inventory.
2. **SSH pipelining eliminates temp file overhead.** Default Ansible copies module code to a temp file, executes it, then deletes. Pipelining streams the module through the SSH connection directly - faster and doesn't leave artifacts.
3. **ControlMaster with 60-second persist reuses SSH connections.** Multi-task playbooks open dozens of SSH connections by default. ControlMaster keeps one connection alive and multiplexes subsequent tasks through it.
4. **Dedicated service account is cleaner than using root.** A purpose-built `ansible` user with key-only auth and passwordless sudo creates a clear audit trail. You know exactly what automation did because it all runs as one user.
5. **Group-based inventory enables flexible targeting.** Hosts can belong to multiple groups. Run `ansible webapps` for web servers, `ansible infrastructure` for everything, or `ansible mail` for just the mail server - without duplicating definitions.
6. **Ludus abstracts the VM provisioning pipeline.** One YAML config specifies templates, VLANs, IPs, resources, and Ansible roles. `ludus range deploy` creates the entire environment. No manual Proxmox clicking.
7. **Custom roles should include health checks as the final task.** Immediate feedback on deployment success. The role either completes with a passing health check or fails with a clear error - no ambiguous "maybe it worked" states.
8. **Jinja2 templates keep systemd unit files maintainable.** Hardcoding paths and service names in unit files creates drift. Templates with variables (`{{ app_name }}` , `{{ app_install_dir }}`) stay consistent across deployments.
9. **Inventory IP addresses become stale when DHCP reservations change.** I moved the forensics workstation from .131 to .112 and the auth server from .112 to .117. The inventory didn't update automatically - playbooks failed until I fixed the IPs manually.

10. `host_key_checking = False` is necessary in lab environments. VMs get rebuilt frequently with new SSH host keys. Strict host key checking would require updating `known_hosts` constantly. Trade-off: less secure, more practical for labs.
-

What I Would Improve Next

P0 (Do This Week)

- **Fix stale inventory IPs** - Update forensics workstation (.112) and auth server (.117) entries
- **Inventory validation playbook** - Automated ping test that reports unreachable hosts before main playbooks run

P1 (Do This Month)

- **Dynamic inventory via Proxmox API** - Auto-discover hosts and IPs instead of static YAML
- **Scheduled system updates** - Weekly apt upgrade playbook across all Linux hosts
- **Security hardening playbook** - SSH hardening, fail2ban, audit logging applied to all hosts
- **Wazuh agent deployment role** - Automatically register new hosts with SIEM

P2 (Do This Quarter)

- **Ansible Vault for secrets** - Stop hardcoding passwords in env files
 - **Monitoring agent deployment** - Auto-register with SIEM on host provisioning
 - **Infrastructure testing playbook** - Verify services running, ports open, DNS resolving
 - **GitOps integration** - Playbooks in Gitea, webhook-triggered runs on commit
-

Common Failure Modes

1. **"Host unreachable" on previously working hosts** - Inventory IP is stale after DHCP reservation change. Check current IP via Proxmox console or DHCP lease table, update inventory.
2. **"Permission denied (publickey)" on new host** - SSH key not distributed to new host. Run `ssh-copy-id <SERVICE_ACCOUNT>@<NEW_HOST>` from control node.
3. **"Gathering facts" takes forever on second run** - Fact cache may be stale or corrupted. Clear `/tmp/ansible_facts/` directory and re-run.
4. **"Missing sudo password" errors** - Passwordless sudo not configured for ansible user on target host. Add `<SERVICE_ACCOUNT> ALL=(ALL) NOPASSWD: ALL` to sudoers.

5. **"Pipelining failed" on specific hosts** - Target host may not have writable `/tmp` or the user lacks permissions. Check `requiretty` isn't set in sudoers, verify `/tmp` permissions.
-

Security Considerations

Authentication

- Dedicated `ansible` service account - not root, not personal accounts
- SSH key-only authentication - no passwords in playbooks or history
- RSA 4096-bit keys - strong cryptographic foundation
- Keys stored only on control node - not distributed widely

Authorization

- Passwordless sudo limited to `ansible` user on managed hosts
- Principle of least privilege - ansible user only has necessary permissions
- No root SSH access - even with the key, root login is disabled

Secrets Management

- Current: passwords in env files (acknowledged technical debt)
- Future: Ansible Vault encryption for all secrets (P2 improvement)
- Sensitive files deployed with restricted permissions (0600)

Trade-offs in Lab vs Production

- `host_key_checking = False` - necessary for frequent VM rebuilds but would be unacceptable in production
 - JSON fact caching - stores host information in plaintext on control node
 - Single control node - no HA, single point of failure for automation
-

Runbook

How to Add a New Host to Inventory

```
# 1. Add host entry to appropriate group in inventory.yml
monitoring:
  hosts:
    <NEW_HOST>:
      ansible_host: <NEW_HOST_IP>

# 2. Distribute SSH key
ssh-copy-id <SERVICE_ACCOUNT>@<NEW_HOST_IP>

# 3. Configure passwordless sudo on target
echo "<SERVICE_ACCOUNT> ALL=(ALL) NOPASSWD: ALL" | sudo tee
/etc/sudoers.d/<SERVICE_ACCOUNT>

# 4. Test connectivity
ansible <NEW_HOST> -m ping
```

How to Run a Playbook Against One Group

```
# Run against specific group
ansible-playbook site.yml --limit webapps

# Run single task with ad-hoc command
ansible webapps -m apt -a "name=htop state=present" --become

# Check mode (dry run) against group
ansible-playbook site.yml --limit monitoring --check
```

How to Deploy a Ludus Range

```
# 1. Create range config (ludus-range.yml)
# 2. Set the range config
ludus range config set -f ludus-range.yml

# 3. Build any missing templates
ludus templates build

# 4. Deploy the range
ludus range deploy

# 5. Check deployment status
ludus range status
```

How to Build a New Packer Template

```
# 1. Navigate to Ludus templates directory
cd /opt/ludus/templates

# 2. Create or modify template definition
# 3. Build specific template
ludus templates build -t debian-12-x64-server-template

# 4. Verify template in Proxmox
ludus templates list
```

How to Create a Custom Ansible Role

```
# 1. Create role skeleton
ansible-galaxy role init roles/my_new_role

# 2. Edit role structure:
#   - roles/my_new_role/defaults/main.yml (default variables)
#   - roles/my_new_role/tasks/main.yml (task list)
#   - roles/my_new_role/templates/*.j2 (Jinja2 templates)
#   - roles/my_new_role/handlers/main.yml (handlers)

# 3. Test role
ansible-playbook test-role.yml --check

# 4. Run role
ansible-playbook test-role.yml
```

Appendix

Glossary

Term	Definition
Ansible	Agentless automation platform using SSH for Linux, WinRM for Windows
Ludus	Open-source cyber range platform built on Proxmox with Packer/Ansible integration
Packer	HashiCorp tool for building machine images from templates
Proxmox VE	Open-source virtualization platform (KVM + LXC)
Jinja2	Python templating engine used by Ansible for templates
YAML Inventory	Ansible inventory format using YAML syntax for host/group definitions
Ansible Role	Reusable automation unit with tasks, templates, handlers, and variables
Ansible Collection	Package format for distributing modules, plugins, and roles
Fact Caching	Storing gathered host facts locally to avoid re-collection
Pipelining	SSH optimization that streams modules instead of copying temp files

Term	Definition
ControlMaster	SSH feature that multiplexes connections through a single socket

MITRE ATT&CK Relevance

Technique ID	Name	Automation Relevance
T1059	Command and Scripting Interpreter	Ansible executes commands across hosts - legitimate automation, but same techniques used by attackers
T1072	Software Deployment Tools	Ansible deploys software at scale - powerful for defenders, attractive target for attackers
T1098	Account Manipulation	Ansible manages user accounts and sudo permissions - audit trail is critical
T1136	Create Account	Custom roles create service accounts - document all automated account creation

Infrastructure as Code Principles Applied

Principle	Implementation
Idempotency	Ansible tasks can run multiple times with same result
Version Control	Playbooks and inventory tracked in Git
Documentation	Role defaults and variable files document configuration
Repeatability	Same playbook produces same result on any host
Modularity	Roles encapsulate reusable automation units
Testability	Check mode allows dry runs before execution

Artifacts Produced

- **Ansible Configuration: `ansible.cfg`** - Optimized settings for fact caching, pipelining, connection reuse
- **YAML Inventory: 14 hosts / 9 groups** - Structured inventory with group-based organization
- **Custom Role: Application Deployer** - Full stack deployment (Docker, UFW, Git, systemd, health check)
- **Ludus Range Config** - VM definitions with templates, VLANs, IPs, and role assignments
- **systemd Service Template** - Jinja2 template for Docker Compose lifecycle management

- **Packer Template Library** - Multi-OS templates (Debian, Ubuntu, Rocky, AlmaLinux, Kali, Windows)

Building your own automation platform? Start with the basics: SSH keys, simple inventory, one playbook. The fancy stuff (fact caching, pipelining, Ludus) comes later. Walk before you run.

Written by Scott S. — Contact: scott@scottschlangen.com — scottslab.io — <https://scottslab.io/posts/ansible-ludus-homelab-infrastructure-as-code>