SNO Disconnected Deployment Guide

102 – The Acme Evolution Path: Automation & Zero Trust

Introduction: Logic Before the Platform

Acme utilizes the **Ansible Automation Platform (AAP)** to implement and enforce **Zero Trust (ZT)** security policies. This evolution path is designed to move newcomers from basic local automation to enterprise-scale security orchestration.

We start on the **Command Line (CLI)** to build a fundamental understanding of Ansible logic. This ensures that when you move into AAP, you can effectively troubleshoot playbooks, interpret execution logs, and design resilient ZT workflows.

The Four Stages of Technical Adoption

Stage 1: The Foundation (Static Automation)

- **Technical Focus:** YAML syntax, local inventory files, and basic fact gathering.
- **ZT Objective: Pillar 2 (Device Identity).** Use Ansible to query system details and verify that an asset matches its expected identity profile.
- **Goal:** Run your first "Discovery" playbook locally against a known target.

Stage 2: The Toolbox (Certified Content Collections)

- **Technical Focus:** Moving from "Built-in" modules to **Red Hat Certified Collections**.
- **ZT Objective: Pillar 7 (Visibility).** Use vendor-supported toolsets (for Windows, Cisco, Cloud) to gather deep telemetry and verify security settings without writing custom code.
- **Goal:** Install and use a Certified Collection to query a specialized platform.

Stage 3: The Source of Truth (Dynamic Inventory)

- **Technical Focus:** Configuring **Inventory Plugins** for VMware, AWS, or Azure.
- **ZT Objective: Continuous Monitoring.** ZT requires real-time knowledge of network assets. Dynamic inventories ensure Ansible always targets the "live" environment rather than an outdated text file.
- **Goal:** Connect Ansible to a cloud provider to discover hosts automatically.

Stage 4: The Enterprise Hub (Ansible Automation Platform)

- **Technical Focus:** Git-based workflows (IaC), Role-Based Access Control (RBAC), and centralized logging.
- **ZT Objective: Pillar 1 (User Identity) & Pillar 6 (Orchestration).** Ensuring only authorized admins trigger remediations and that every change is logged and auditable.
- **Goal:** Sync a Git repository to AAP and execute a Job Template with specific user permissions.

Zero Trust & Technical Resources

The Frameworks

At Acme, our automation goals are guided by these federal standards:

- **DoD Zero Trust Strategy:** DoD CIO Library > Focuses on the 7 Pillars: User, Device, App/Workload, Data, Network, Automation, and Visibility.
- **CISA Zero Trust Maturity Model:** CISA ZTMM

Training & Documentation

- **Interactive Labs:** Red Hat Ansible Labs
- **Best Practices:** Ansible Community Best Practices
- **Zero Trust Strategy:** Automating Zero Trust with Red Hat

---

Summary of the Path

Phase	Tooling	ZT Alignment
Stage 1	CLI & Local Files	Device Identity
Stage 2	Certified Collections	Visibility & Telemetry
Stage 3	Inventory Plugins	Continuous Monitoring
Stage 4	AAP & Git	Orchestration & RBAC

Stage 1: The Foundation (Static Automation)

Introduction

This is the first stage of the Acme Evolution Path. Before scaling to the Ansible Automation Platform (AAP), we must master the core mechanics of "Infrastructure as Code." In this stage, you will perform a **Discovery Scan** to verify system identity and security posture.

This aligns with the **Zero Trust Maturity Model**, specifically **Pillar 2 (Device Identification)** and **Pillar 7 (Visibility)**.

The Six Key Parts of Stage 1

1. **Project Structure:** Organizing files into a professional, scalable layout.
2. **Inventory:** Defining the manual "Source of Truth" for your targets.
3. **Group Variables:** Securely managing connection settings for different platforms.
4. **Facts:** The discovery engine that identifies system details automatically.
5. **Modules:** The specialized tools used to interact with Operating Systems.
6. **Conditionals (`when`):** Logic that allows one playbook to handle both Linux and Windows.

Part 1: Project Structure

Ansible requires a specific directory structure to apply settings correctly. This organization is the first step in moving away from manual scripts toward managed code.

Ensure your folder looks like this:

```
acme-automation/
├── inventory.ini           # (Part 2) The list of targets
├── discovery_lab.yml       # (Parts 4-6) The automation logic
(Playbook)
└── group_vars/             # (Part 3) Folder for group-specific
```

```
settings
    └── windows.yml          # Connection details for Windows
```

Part 2: Inventory

The inventory file is your map. In Stage 1, we use a **Static Inventory** to manually define the IP addresses or hostnames of our assets.

`inventory.ini`

```
[linux]
192.168.1.10

[windows]
192.168.1.20

[all_servers:children]
linux
windows
```

Part 3: Group Variables

We use `group_vars` to separate **how** we connect from **what** we do. This ensures our playbooks remain generic and reusable.

`group_vars/windows.yml`

```yaml
---
# Connection settings applied only to the [windows] group
ansible_user: "Administrator"
ansible_password: "SecurePassword123"
ansible_connection: "winrm"
ansible_winrm_server_cert_validation: ignore
```

Parts 4, 5, & 6: Facts, Modules, and Conditionals

The **Playbook** combines these three parts to perform the work. This script is "Read-Only," meaning it gathers intelligence without altering the system.

`discovery_lab.yml`

```yaml
---
# FOCUS: Pillar 2 (Device) and Pillar 7 (Visibility)
- name: Zero Trust Resource Discovery
  hosts: all
  become: no
  gather_facts: yes # PART 4: Automatically gathers system
        intelligence.

  tasks:
    - name: Discovery Scan of Device and OS
      # PART 5: The 'debug' module displays findings to the
        console.
      debug:
        msg:
          - "Hostname: {{ ansible_hostname }}"
          - "Family: {{ ansible_os_family }}"
          - "Distribution: {{ ansible_distribution }}
        {{ ansible_distribution_version }}"

    # DATA COLLECTOR: Feeds the Zero Trust reporting engine.
    - name: Save Discovery Data to Local Report
      # PART 5: The 'lineinfile' module records findings for
        auditing.
      lineinfile:
        path: "./discovery_report.txt"
        line: "Host: {{ ansible_hostname }} | OS:
        {{ ansible_distribution }} | IP:
        {{ ansible_default_ipv4.address }}"
        create: yes
      delegate_to: localhost

    # --- LINUX SECTION ---
    - name: Linux Only - Query Firewall Status
      ansible.builtin.service_facts:
      # PART 6: Logic to ensure Linux tools only run on Linux.
      when: ansible_os_family == "RedHat"
```

```yaml
# --- WINDOWS SECTION ---
- name: Windows Only - Query Firewall Status
  # PART 5: Win_service is the tool for Windows Service
    Control.
  ansible.windows.win_service:
    name: MpsSvc
  register: win_fw_status
  # PART 6: Logic to ensure Windows tools only run on Windows.
  when: ansible_os_family == "Windows"

- name: Report Windows Firewall
  debug:
    msg: "Windows Firewall is {{ win_fw_status.state }}"
  when: ansible_os_family == "Windows"
```

Stage 1 Review & Resources

Before moving to **Stage 2**, ensure you can explain why we use `delegate_to: localhost` and how `ansible_facts` supports device identification.

- **Documentation:** Variables and Facts
- **Best Practices:** Ansible Tips and Tricks

Stage 2: The Toolbox (Certified Content Collections)

Introduction

As an Acme administrator, you shouldn't have to "reinvent the wheel" for common tasks like managing Windows Registries or Cisco Switches. **Collections** allow you to package and distribute playbooks, roles, and modules.

This stage aligns with **Pillar 7 (Visibility)** and **Pillar 6 (Automation/ Orchestration)** by standardizing the toolsets used to gather telemetry and enforce security policies.

The Three Key Parts of Stage 2

1. **Ansible Galaxy:** The community hub for discovering automation content.
2. **Certified Content:** High-assurance collections maintained by Red Hat and partners (Microsoft, Cisco, etc.).
3. **Namespace & Collection Name:** Understanding the "Fully Qualified Collection Name" (FQCN) to ensure you are calling the right tool.

Part 1: Understanding the "Certified" Difference

At Acme, we prioritize **Certified Content** over community-made scripts whenever possible.

- **Community (Galaxy):** Open-source, created by individuals. Great for inspiration, but use with caution in high-security environments.
- **Certified (Automation Hub):** Tested, hardened, and supported by Red Hat and the vendor. These are required for Acme's production **Zero Trust** workflows to ensure supply-chain security.

Part 2: Namespaces and FQCN

To use a collection, you must refer to it by its **Fully Qualified Collection Name (FQCN)**. This prevents conflicts between different modules with the same name.

The Anatomy of a Tool: `microsoft.ad.user`

- **Namespace:** `microsoft` (The owner)
- **Collection:** `ad` (The toolbox)
- **Module:** `user` (The specific tool)

Part 3: Using Collections in a Playbook

In this example, we will use the `microsoft.windows` and `community.general` collections. These provide much deeper "Visibility" into the system than the basic modules used in Stage 1.

advanced_discovery.yml

```yaml
---
- name: Advanced Security Discovery
  hosts: windows
  gather_facts: yes

  # COLLECTIONS: This keyword tells Ansible which toolboxes to
        open.
  collections:
    - microsoft.windows
    - community.general

  tasks:
    - name: Query Windows Security Updates
      # We are using a specialized module from the
        microsoft.windows collection
      microsoft.windows.win_updates:
        state: searched
      register: update_results

    - name: Report Update Compliance (Pillar 7)
      debug:
        msg: "Found {{ update_results.found_update_count }}
        missing security updates."

    - name: Check for Unauthorized Local Users
      microsoft.windows.win_user:
        name: "Guest"
        state: query
      register: guest_status

    - name: Visibility Report
      debug:
        msg: "Security Alert: Guest account is
        {{ guest_status.state }}"
      when: guest_status.state == "present"
```

Part 4: Managing Collections via Requirements

As you scale at Acme, you won't install collections manually. You will use a `requirements.yml` file. This file allows **AAP** to automatically download the correct tools when your job runs.

`collections/requirements.yml`

```yaml
---
collections:
  - name: microsoft.windows
    version: 2.1.0
  - name: cisco.ios
  - name: redhat.rhel_system_roles
```

Stage 2 Review & Resources

By the end of Stage 2, you should understand that you are an **assembler** of trusted tools. You use Certified Collections to ensure that Acme's automation is stable, secure, and vendor-supported.

- **Search for Tools:** Ansible Galaxy
- **Acme's Official Hub:** Red Hat Automation Hub
- **Documentation:** Using Collections

Stage 3: The Source of Truth (Dynamic Inventory)

Introduction

In Stage 1, we used a static `inventory.ini` file. In Stage 3, we replace that file with an **Inventory Plugin**. This plugin acts as a translator between Ansible and your infrastructure providers (like AWS, Azure, or VMware).

This aligns with **Pillar 7 (Visibility and Analytics)** and **Continuous Monitoring**. You cannot secure what you cannot see; Dynamic Inventory ensures that 100% of your active assets are accounted for in every automation run.

The Three Key Parts of Stage 3

1. **Inventory Plugins:** The code that communicates with your cloud or virtualization API.
2. **Plugin Configuration (`.yml`):** The file that defines *where* to look and *how* to group discovered hosts.
3. **Keyed Groups:** Automatically organizing hosts based on their metadata (e.g., "all servers in the 'Production' VPC").

Part 1: Static vs. Dynamic Thinking

At Acme, manual inventories are considered a security risk. If an admin spins up a new server and forgets to add it to the `.ini` file, that server remains unpatched and unmonitored.

- **Static:** "I think these 10 servers exist."
- **Dynamic:** "I am asking the Cloud Provider to tell me exactly what is running right now."

Part 2: Configuring the Plugin

Instead of a list of IPs, we create a configuration file that ends in `aws_ec2.yml`, `azure_rm.yml`, or `vmware_vm_inventory.yml`. This tells Ansible to use the corresponding plugin from the **Collections** we learned about in Stage 2.

Example: `acme_aws_inventory.yml`

```yaml
---
# This tells Ansible to use the AWS EC2 plugin
plugin: amazon.aws.aws_ec2

# Define the regions to scan for assets
regions:
  - us-east-1
```

```
  - us-west-2

# KEYED GROUPS: This is the "Magic" of Stage 3.
# It automatically creates Ansible groups based on the Tags in
        AWS.
keyed_groups:
  - key: tags.Environment
    prefix: env
  - key: tags.Role
    prefix: role
  - key: instance_type
    prefix: type

# Only include hosts that are actually running (Pillar 2: Device
        Identity)
filters:
  instance-state-name: : running
```

Part 3: Testing the Real-Time Feed

Once the configuration is set, you don't run a playbook yet. First, you verify that Ansible can "see" the environment.

Command: `ansible-inventory -i acme_aws_inventory.yml --graph`

Output Example:

```
@all:
  |--@env_prod:
  |  |--ec2-3-80-1-10.compute-1.amazonaws.com
  |  |--ec2-3-80-1-11.compute-1.amazonaws.com
  |--@role_webserver:
  |  |--ec2-3-80-1-10.compute-1.amazonaws.com
  |--@type_t3_medium:
  |  |--ec2-3-80-1-10.compute-1.amazonaws.com
```

How Stage 3 Supports Zero Trust

- **Eliminates Shadow IT:** If an unauthorized user spins up a VM, the Dynamic Inventory will find it on the next scan, allowing Acme's security playbooks to immediately apply "deny-all" firewall rules.

- **Metadata-Based Security:** We can write playbooks that target `hosts: env_prod`. Ansible will only run against servers that the Cloud Provider has cryptographically verified as belonging to the Production environment.

Stage 3 Review & Resources

Before moving to **Stage 4**, practice running the `ansible-inventory` command against a lab environment. Notice how you no longer need to type IP addresses.

- **Guide:** How to use Inventory Plugins
- **VMware Example:** Community VMware Inventory Plugin
- **AWS Example:** Amazon AWS EC2 Inventory Plugin

Stage 4: The Enterprise Hub (Ansible Automation Platform)*

In a **Zero Trust** environment, the CLI is risky because it lacks centralized auditing and access control. Stage 4 solves this by providing a governed "Control Plane" for all your Stage 1, 2, and 3 work.

Introduction

Up until now, you have been running automation from your own workstation. In Stage 4, you will move your code into **Git** and use **AAP** to execute it. This transition allows Acme to enforce security policies at scale, with full visibility into who ran what, when, and where.

This aligns with **Pillar 1 (User Identity)** and **Pillar 6 (Automation & Orchestration)**.

The Three Key Parts of Stage 4

1. **Version Control (Git):** The "Single Source of Truth" for your code.
2. **Role-Based Access Control (RBAC):** Defining exactly which admins can run which playbooks.

3. **Job Templates:** Creating “Push-Button” automation that anyone (with permission) can use.

Part 1: The Git Workflow (Infrastructure as Code)

At Acme, we no longer keep playbooks on our local hard drives. They live in a Git repository.

- **Review:** Changes are proposed via “Pull Requests” and peer-reviewed.
- **Sync:** AAP automatically pulls the latest code from Git.
- **Audit:** We have a permanent record of every line of code ever changed.

Part 2: Governance with RBAC**

In Stage 1, if you had the SSH key, you could do anything. In Stage 4, AAP acts as the **Policy Enforcement Point**.

- **Least Privilege:** You can give a junior admin permission to run a “Security Scan” playbook without giving them the actual root passwords or SSH keys.
- **Credential Management:** AAP stores the sensitive passwords (for Windows or Linux) in an encrypted vault. The admin never sees the password; they only see the “Run” button.

Part 3: Creating a Job Template

A Job Template is the “App-ification” of your playbook. It combines your **Playbook (from Git)**, your **Inventory (from Stage 3)**, and your **Credentials**.

The Job Template View in AAP:

- **Inventory:** “Acme-AWS-Dynamic-Inventory”
- **Project:** “Security-Audit-Playbooks”
- **Playbook:** `discovery_lab.yml`
- **Credentials:** “Domain-Admin-Vault”

How Stage 4 Completes the Zero Trust Mission

- **Pillar 1 (User Identity):** AAP integrates with Acme's Active Directory/SSO. We know exactly which human triggered an automation task.
- **Pillar 6 (Orchestration):** AAP can coordinate complex workflows (e.g., "Scan the server, if it fails, quarantine it in the firewall, then notify the security team").
- **Centralized Logging:** Every task performed by Ansible is logged in AAP. This provides the "Audit Trail" required for federal compliance and incident response.

Stage 4 Review & Resources

Stage 4 is the "Final State" for an Acme administrator. Your goal is to move all the discovery and enforcement logic you learned in the previous stages into this governed platform.

- **Interactive Lab:** Writing Your First Controller Job Template
- **Video:** Understanding Ansible Controller RBAC
- **Documentation:** AAP Controller User Guide

Congratulations!

You have completed the **Acme Evolution Path**. You started with a single line of YAML and ended with an Enterprise-grade security orchestration platform.

Final Review: The Acme Automation Journey

Executive Summary

The goal of this path was to transition from manual, high-risk administrative tasks to a governed, automated **Infrastructure as Code (IaC)** model. By following these four stages, you have built the foundation required to support Acme's **Zero Trust Architecture**.

```
graph LR
    S1[Stage 1: Logic] --> S2[Stage 2: Tools]
    S2 --> S3[Stage 3: Visibility]
    S3 --> S4[Stage 4: Governance]
    style S1 fill:#f9f,stroke:#333,stroke-width:2px
    style S4 fill:#00f,stroke:#fff,stroke-width:4px,color:#fff
```

Stage 1: The Foundation (Logic)

- **Key Concept:** Writing YAML playbooks and understanding the "Ansible Way."
- **Zero Trust Focus: Pillar 2 (Device Identity).** Verifying that a device is what it says it is.
- **Takeaway:** You learned that **Facts** provide the telemetry needed to make security decisions.
- **Artifacts:** `inventory.ini`, `discovery_lab.yml`.

Stage 2: The Toolbox (Certified Content)

- **Key Concept:** Leveraging **Red Hat Certified Collections** to use vendor-supported code.
- **Zero Trust Focus: Pillar 7 (Visibility).** Using trusted tools (Microsoft, Cisco, Red Hat) to audit deep system settings.
- **Takeaway:** Don't reinvent the wheel. Use **Fully Qualified Collection Names (FQCN)** to call hardened, professional modules.
- **Artifacts:** `requirements.yml`, `collections/`.

Stage 3: The Source of Truth (Dynamic Inventory)

- **Key Concept:** Replacing static text files with **Inventory Plugins** that talk to APIs.
- **Zero Trust Focus: Continuous Monitoring.** Automating the discovery of "Shadow IT" and ensuring no asset is left unmanaged.
- **Takeaway:** The infrastructure (AWS, Azure, VMware) is the only "Source of Truth." Use **Keyed Groups** to organize hosts by real-time metadata.
- **Artifacts:** `acme_aws_inventory.yml` (or similar plugin config).

Stage 4: The Enterprise Hub (AAP & Git)

- **Key Concept:** Centralizing execution in the **Ansible Automation Platform**.
- **Zero Trust Focus: Pillar 1 (User Identity) & Pillar 6 (Orchestration).** Enforcing **Least Privilege** through Role-Based Access Control (RBAC).
- **Takeaway:** The CLI is for development; AAP is for production. Store code in **Git**, encrypt secrets in **AAP Credentials**, and scale via **Job Templates**.
- **Artifacts:** Git Repository, AAP Job Templates, Surveys.

The Zero Trust Compliance Checklist

As you move into production at Acme, every automation you build should answer these three questions:

1. **Visibility:** Does this playbook provide telemetry for **Pillar 7**?
2. **Least Privilege:** Does the AAP Job Template use the minimum credential required for the task (**Pillar 1**)?
3. **Policy Enforcement:** Does this automation ensure the device stays in its "Target State" (**Pillar 2**)?

Continuing Your Education

- **Acme Internal Git:** Review existing playbooks to see Stage 2 and 3 in practice.

- **Red Hat Training:** Use the Ansible Interactive Labs for deep dives into specific collections.
- **Zero Trust Frameworks:** Keep the DoD Zero Trust Strategy as your north star for all security automation.

Closing Note

Automation is not a “one and done” task—it is a continuous evolution. As Acme’s infrastructure changes, your playbooks and inventories must adapt. You now have the skills to build, scale, and secure that environment.

Ansible Zero Trust Conversation