**🔍 Ansible Automation Platform (AAP) vs. Ansible Core / AWX**

| **Feature/Capability** | **Ansible Core** | **AWX (Community Edition)** | **Ansible Automation Platform (AAP)** |
| --- | --- | --- | --- |
| **Publisher** | open source community) | Red Hat (community project) | Red Hat (enterprise product) |
| **Support** | Community only | Community only | Full Red Hat support (SLA-based) |
| **Updates** | Frequent, fast | Community-paced | Stable releases, tested, enterprise-grade |
| **Web UI (GUI)** | ❌ Not included | ✅ Yes (AWX UI) | ✅ Yes (Automation Controller) |
| **REST API** | ❌ No | ✅ Yes | ✅ Yes (fully supported + backward-compatible) |
| **RBAC (Role-Based Access Control)** | ❌ No | ✅ Limited | ✅ Advanced and customizable |
| **Job Scheduling** | ❌ No | ✅ Yes | ✅ Yes, with workflow chaining |
| **Workflow Automation** | ❌ No | ✅ Yes | ✅ Yes (GUI + YAML) |
| **Logging & Auditing** | Manual | Basic logs | Centralized, enterprise audit-ready |
| **Secrets Management** | Ansible Vault (CLI) | Basic | Integration with Vaults + RBAC |
| **Scalability** | Single node | Single/multi-node (manual) | Enterprise-scale, clustering, HA |
| **Execution Environments (EEs)** | Manual setup | Basic support | Full support with ansible-builder, registry integration |
| **Content Management** | CLI | Limited | ✅ Automation Hub + Private Repos |
| **CI/CD Integration** | Manual | Basic | ✅ Deep Git integration, webhooks, API |
| **Analytics** | ❌ No | ❌ No | ✅ Automation Analytics (via cloud.redhat.com) |
| **Offline Content Delivery** | ❌ No | ❌ No | ✅ Available via disconnected Automation Hub |
| **License** | GPL | Apache | Requires RHEL subscription and AAP license |
| **Best For** | Individuals, CLI users | Testing GUI workflows | Enterprises, regulated environments |

**📦 What is AWX?**

**AWX** is the **open-source upstream project** of Ansible Tower / Automation Controller. Think of it as the "development version" of AAP’s UI and API features.

* It provides a web interface, REST API, and task engine.
* It is **not supported by Red Hat**, intended for **test/dev environments**.
* Installable via containerized setup using awx-operator.

**💼 What is Ansible Automation Platform (AAP)?**

**AAP** is Red Hat’s **commercial enterprise solution** that:

* Combines Automation Controller (based on AWX),
* Automation Hub,
* Private Galaxy for content,
* Automation Mesh for scaling,
* And Execution Environments for containerized, portable runtimes.

It’s designed for **security, performance, support, and scalability** in production environments.

**🧠 Key Differences: AAP vs AWX/Core**

| **Aspect** | **AWX/Core** | **AAP** |
| --- | --- | --- |
| **Installation** | Manual (docker, operator) | Official installer, containerized or VM |
| **Execution Environments** | Supported, manual | First-class support, integrated |
| **Automation Hub** | Not available | ✅ Use private/custom collections |
| **Analytics** | ❌ | ✅ (Red Hat hosted or disconnected) |
| **Role-Based Access** | Limited | Granular RBAC and orgs/teams |
| **License** | Open Source | Paid subscription |
| **Offline Use** | ❌ | ✅ (disconnected deployments supported) |
| **Security & Compliance** | Manual | FIPS, audit logging, SCAP/CIS automation |

**✅ Enterprise Benefits of AAP**

| **Enterprise Requirement** | **AAP Feature** |
| --- | --- |
| **Security & Compliance** | FIPS-compliant cryptography, SSO integration, RBAC, auditing |
| **Governance** | Role delegation, job approvals, logs |
| **Reliability** | HA, clustering, automation mesh |
| **Scalability** | Job slicing, multi-node controller |
| **Content Curation** | Automation Hub, private Galaxy, signed collections |
| **Performance Optimization** | Execution Environments, custom base images |
| **Support** | 24x7 enterprise support by Red Hat |
| **DevOps Integration** | CI/CD pipelines, webhook triggers, GitOps ready |

**🛠️ Example Use Cases by Platform**

| **Use Case** | **Suitable Platform** |
| --- | --- |
| Test playbooks locally or via CLI | Ansible Core |
| Small team experimenting with GUI | AWX |
| Enterprise production automation | AAP |
| Multi-team RBAC and auditing | AAP |
| Centralized job scheduling | AWX / AAP |
| Disconnected / air-gapped automation | AAP only |

**✅ Visualized Architecture: AAP 2.5 Containerized on Single Node (All-in-One)**

+--------------------------------------------------------------------------------------+

| RHEL 9 Host (rhel9master.ansible.local) |

| [4 vCPU | 16 GB RAM | 60 GB Disk | Podman Installed] |

+--------------------------------------------------------------------------------------+

| |

| +------------------------+ +---------------------+ +----------------------+ |

| | Automation Controller| | Automation Hub | | Event Driven Ansible| |

| | (controller) | | (hub) | | (eda-server) | |

| | - Web UI/API | | - Collections repo | | - WebSocket events | |

| | - Job templates | | - EE signing keys | | - Rulebooks | |

| +----------+-------------+ +----------+----------+ +-----------+----------+ |

| | | | |

| | | | |

| v v v |

| +--------------+ +--------------------+ +------------------+ |

| | PostgreSQL DB |<------>| controller/hub/eda |<------->| Redis (Standalone)| |

| | (Single DB) | | use local database | | For real-time data| |

| +--------------+ +--------------------+ +------------------+ |

| |

| +-------------------------------------------------------------------------------+ |

| | Execution Environments (Podman containers) | |

| | - ansible-core | |

| | - collections | |

| | - python dependencies | |

| +-------------------------------------------------------------------------------+ |

| |

| +---------------------+ +----------------------+ +---------------------+ |

| | Automation Gateway | --> | Authentication Proxy | --> | Internal Routing | |

| | - Single entrypoint | | - SSO/AuthN | | - API coordination | |

| +---------------------+ +----------------------+ +---------------------+ |

| |

+--------------------------------------------------------------------------------------+

<< External Access >>

https://rhel9master.ansible.local

**🔍 Component Roles in Your Deployment**

| **Component** | **Deployed As** | **Function** |
| --- | --- | --- |
| **Automation Controller** | Podman container | Orchestration engine; manages playbooks, inventories, and jobs |
| **Automation Hub** | Podman container | Collection repository; stores certified/validated Ansible content |
| **Event-Driven Ansible** | Podman container | Handles rules + events for automation beyond scheduled triggers |
| **PostgreSQL DB** | Podman container | Backend DB for controller, hub, and EDA |
| **Redis** | Podman container | Event streaming, caching |
| **Execution Environments** | Podman containers | Runs ansible-core with custom dependencies |
| **Automation Gateway** | Podman container | Unified access layer for all platform components |

**Lab1 : Managing Job Templates in Ansible Automation Platform**

**Objective**

This lab demonstrates how to create and manage **Job Templates** in Ansible Automation Platform (AAP) by utilizing **Projects** and **Inventories**. By the end of this lab, you will be able to:

1. **Create a Project** to store playbooks.
2. **Define an Inventory** with managed hosts.
3. **Create a Job Template** linking the Project and Inventory.
4. **Execute the Job Template** and verify results.

**Lab Setup**

**Prerequisites**

✅ Ansible Automation Platform (or AWX) installed and accessible.  
✅ SSH access to at least **two managed nodes** (Linux machines).  
✅ A Git repository containing Ansible playbooks (or create one during the lab).

**Step 1: Create a Project**

A **Project** is a logical collection of Ansible playbooks stored in a source control repository (Git, SVN, etc.).

**Steps:**

1. Log in to **Ansible Automation Platform**.
2. Navigate to **Projects** → **Add Project**.
3. Fill in the details:
   * **Name:** Apache\_Deployment
   * **Organization:** Default
   * **Source Control Type:** Git
   * **Source Control URL:** https://github.com/your-repo/apache-setup.git
   * **Branch/Tag/Commit:** main
4. Click **Save**.
5. **Sync the Project** (to fetch playbooks from Git).

---

- name: Install and Start Apache

hosts: all

tasks:

# Update apt package cache (recommended before installing)

- name: Update apt package cache

ansible.builtin.apt:

update\_cache: yes

# Install Apache (apache2 package on Ubuntu/Debian)

- name: Install Apache2

ansible.builtin.apt:

name: apache2

state: present

# Ensure systemd is reloaded after installing new service

- name: Reload systemd to pick up Apache service

ansible.builtin.systemd:

daemon\_reload: yes

# Ensure Apache is running and enabled on boot

- name: Start and enable Apache service

ansible.builtin.service:

name: apache2

state: started

enabled: yes

# Create a basic index.html

- name: Add simple index.html

ansible.builtin.copy:

content: "Hello from {{ ansible\_hostname }}!"

dest: /var/www/html/index.html

mode: '0644'

**Verification:**

* Under **Projects**, check if Apache\_Deployment appears.
* Confirm the playbook (install\_apache.yml) is visible under **Files**.

**Step 2: Create an Inventory**

An **Inventory** defines the hosts and groups where Ansible will run jobs.

**Steps:**

1. Navigate to **Inventories** → **Add Inventory**.
2. Fill in:
   * **Name:** Web\_Servers
   * **Organization:** Default
3. Click **Save**.
4. **Add Hosts**:
   * Go to **Hosts** tab → **Add Host**.
   * Enter:
     + **Name:** web1.example.com
     + **Variables:** ansible\_user: ansible ansible\_ssh\_private\_key\_file: /path/to/key , , **Ansible\_host:** ip address
   * Repeat for web2.example.com

**OR USE DYNAMIC INVENTORY ( in next lab)**

1. **Create a instance Group** (optional):
   * Under **Groups** tab → **Add Group**.
   * Name: webservers
   * Add hosts (web1, web2) to this group.

**Verification:**

* Under **Inventories**, check if Web\_Servers has the correct hosts.

**Step 3: Create a Job Template**

A **Job Template** combines a **Project**, **Inventory**, and **Playbook** to define an executable job.

**Steps:**

1. Navigate to **Templates** → **Add Job Template**.
2. Fill in:
   * **Name:** Deploy\_Apache
   * **Job Type:** Run
   * **Inventory:** Web\_Servers
   * **Project:** Apache\_Deployment
   * **Playbook:** install\_apache.yml (from the project)
   * **Credentials:** Select an existing SSH credential (or create one).
   * **Verbosity:** 0 (Normal)
   * **Limit (Optional):** webservers (to restrict execution to a group)
3. Click **Save**.

**Verification:**

* Under **Templates**, confirm Deploy\_Apache is listed.

**Step 4: Execute the Job Template**

**Steps:**

1. From the **Templates** list, click **Launch** on Deploy\_Apache.
2. Monitor the **Job Output** in real-time.
3. Verify:
   * Successfully installed Apache on web1 and web2.
   * No failed tasks.

**Expected Output:**

plaintext

Copy

PLAY [Install Apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TASK [Gathering Facts] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [web1.example.com]

ok: [web2.example.com]

TASK [Install Apache] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

changed: [web1.example.com]

changed: [web2.example.com]

PLAY RECAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

web1.example.com : ok=2 changed=1 unreachable=0 failed=0

web2.example.com : ok=2 changed=1 unreachable=0 failed=0

**Step 5: Advanced Features (Optional)**

**A. Survey (Prompt for Variables)**

1. Edit Deploy\_Apache Job Template.
2. Under **Survey** tab → **Add Survey**.
3. Add a question:
   * **Question:** Enter Apache version
   * **Variable:** apache\_version
   * **Default:** 2.4
4. Modify the playbook to use {{ apache\_version }}.
5. **Re-launch** the job—now it will prompt for input.

**B. Scheduling Jobs**

1. Edit Deploy\_Apache.
2. Under **Schedule** → **Add Schedule**.
3. Set a **cron-like schedule** (e.g., daily at 3 AM).

**Conclusion**

✅ Created a **Project** linked to a Git repo.  
✅ Defined an **Inventory** with managed hosts.  
✅ Built a **Job Template** combining Project + Inventory + Playbook.  
✅ Executed and verified the job.

This workflow enables **repeatable, automated deployments** across your infrastructure!

**🧪 Lab2: Parameterize Jobs using Survey in AAP**

**🎯 Objective**

Configure a Job Template in **Ansible Automation Platform 2.5** that:

* Accepts **three dynamic parameters** via Survey:
  + dir\_state: Control if directory is created/absent
  + dir\_path: Directory path to manage
  + user\_state: Control user creation/removal
* Executes tasks accordingly on managed hosts.

**🧱 Prerequisites**

1. A working **AAP 2.5 cluster** (controller + EE).
2. SSH access from AAP to target managed nodes (e.g., m1, m2).
3. A **project repo** (local/Git) with your playbook.
4. A **machine credential** (e.g., user ubuntu or ansible).
5. Inventory with at least 1–2 hosts.

**📁 Step 1: Create Your Playbook**

Save the following as parameterized\_task.yml:

yaml

CopyEdit

- hosts: all

become: yes

connection: ssh

tasks:

- name: Create a directory (controlled via survey)

file:

path: "{{ dir\_path }}"

state: "{{ dir\_state }}"

- name: Create a user (controlled via survey)

user:

name: usertest

shell: /bin/bash

uid: 6666

comment: "my rh-aaap user"

state: "{{ user\_state }}"

✅ This playbook uses 3 variables we’ll pass at runtime via Survey.

**📂 Step 2: Upload to AAP Project**

**Option A: Git Project**

1. Push the playbook to a Git repo (GitHub/GitLab).
2. In AAP → **Resources → Projects → Add**:
   * Name: SurveyDemoProject
   * Source Control: Git
   * URL: https://github.com/<your\_repo>.git
   * Branch: main
   * Save & Sync

**Option B: Manual Project**

1. Upload playbook to AAP controller's PROJECTS\_ROOT.
2. In AAP → Projects → Add:
   * Name: SurveyDemoProject
   * Source Control: Manual
   * Playbook Directory: parameterized\_lab
   * Save

**📘 Step 3: Create a Job Template**

Go to: **Templates → Add → Job Template**

**Fill Details:**

| **Field** | **Value** |
| --- | --- |
| **Name** | Parameterized Job Survey |
| **Inventory** | Choose your inventory |
| **Project** | SurveyDemoProject |
| **Playbook** | parameterized\_task.yml |
| **Execution Env.** | Default or custom EE |
| **Credentials** | Machine credential (e.g., SSH) |

Do **not** add extra variables here — we’ll pass them via **Survey**.

Click **Save**.

**📊 Step 4: Add Survey**

1. In the Job Template view, click **Add Survey**
2. Click **Add Question** three times for the following:

**✅ Survey Question 1: Directory State**

| **Field** | **Value** |
| --- | --- |
| **Prompt** | Choose the directory state |
| **Variable Name** | dir\_state |
| **Answer Type** | Multiple Choice (single select) |
| **Choices** | directory, absent |
| **Default** | directory |
| **Required** | Yes |

**✅ Survey Question 2: Directory Path**

| **Field** | **Value** |
| --- | --- |
| **Prompt** | Enter directory path |
| **Variable Name** | dir\_path |
| **Answer Type** | Text |
| **Default** | /opt/rk |
| **Required** | Yes |

**✅ Survey Question 3: User State**

| **Field** | **Value** |
| --- | --- |
| **Prompt** | Choose the user state |
| **Variable Name** | user\_state |
| **Answer Type** | Multiple Choice (single select) |
| **Choices** | present, absent |
| **Default** | present |
| **Required** | Yes |

Click **Save Survey** and then **Save Template**.

**🚀 Step 5: Run the Job**

1. Go to **Templates**
2. Click ▶️ **Launch** on Parameterized Job Survey
3. Fill survey:
   * Directory State: absent
   * Directory Path: /opt/rk
   * User State: present
4. Click **Next → Launch**

**✅ Step 6: Verify Output**

* AAP job output will show whether:
  + Directory was created or deleted
  + User was created or removed
* You can check results via:
  + AAP Job stdout
  + SSH into the target host and verify /opt/rk and usertest

**🧪 Additional Test Scenarios**

Try re-running the job with different combinations:

| **dir\_state** | **dir\_path** | **user\_state** | **Expected Outcome** |
| --- | --- | --- | --- |
| directory | /opt/demo | present | /opt/demo is created, usertest is present |
| absent | /opt/temp | absent | /opt/temp deleted, usertest removed |
| directory | /data/logs | absent | /data/logs created, usertest removed |

**LAB3 : Using dynamic inventory instead of static :**

**🟩 Step 1: Create a New Inventory in AAP**

1. Navigate to the AAP **UI**.
2. Go to **Inventories** → **Add** → **Inventory**.
3. Fill in:
   * **Name**: raman-dynamic-inventory
   * **Organization**: Your org
   * **Description**: (Optional) Dynamic EC2 Inventory
4. Click **Save**.

**🟩 Step 2: Create AWS Cloud Credential**

1. Go to **Credentials** → **Add** → **Credential**.
2. Fill in:
   * **Name**: raman-amazon-acc-creds
   * **Credential Type**: Amazon Web Services
   * **Organization**: Your org
3. Under **Input Configuration**:
   * **Access Key**:
   * **Secret Access Key**:
4. Click **Save**.

**🟩 Step 3: Add Dynamic Source to Inventory**

1. Go to your inventory: raman-dynamic-inventory.
2. Click **Sources** → **Add Source**.
3. Fill in:
   * **Name**: raman-dynamicInv-source
   * **Source**: Amazon EC2
   * **Credential**: raman-amazon-acc-creds
4. Enable the following options:
   * ✅ **Overwrite**
   * ✅ **Overwrite Variables**
   * ✅ **Update on Launch**

**🟩 Step 4: Configure Source Variables**

-- add below variables in source variables ( not in inventory level variables )

plugin: amazon.aws.aws\_ec2

# The values for profile, access key, secret key and token can be hardcoded like:

regions:

  - us-east-2

#filters:

  ## All instances with their `Environment` tag set to `dev`

  # tag:Environment: dev

---------------------------------------------------------

plugin: amazon.aws.aws\_ec2

regions:

  - us-east-2

filters:

  tag:Env: training

    --------------------------------------------------

    -- after testing by launching inv source :

        --- we go to console and tag the servers that i want to detect by inv dynamic

        plugin: amazon.aws.aws\_ec2

regions:

  - us-east-2

filters:

  instance-state-name: running

keyed\_groups:

  - key: tags.Role

    prefix: role

compose:

  ansible\_host: public\_ip\_address

Note : make sure to tag all servers of one group as role=web and all other group servers as role=db

**🟩 Step 7: Validate Inventory Groups and Hosts**

1. Go to **Inventories** → raman-dynamic-inventory.
2. Click **Hosts** and **Groups** tabs.
3. Confirm:
   * Hosts have public IPs set as ansible\_host
   * Hosts are grouped like role\_web, role\_app based on EC2 tag

**LAB4: Implementing variables in playbook using template and survey :**

* cat third.yml

---

- name: "Deploy {{ pkgname }} webserver"

hosts: m1

become: yes

vars:

pkgname: apache2

service: apache2

desired\_state: present # 'present' or 'absent' (via survey)

listen\_port: 81 # Port number (via survey)

tasks:

- name: Update apt package index

ansible.builtin.apt:

update\_cache: yes

- name: Ensure {{ pkgname }} is in desired state {{ desired\_state }}

ansible.builtin.apt:

name: "{{ pkgname }}"

state: "{{ desired\_state }}"

- name: Update Apache listening port (only if installing)

ansible.builtin.lineinfile:

path: /etc/apache2/ports.conf

regexp: "^Listen "

line: "Listen {{ listen\_port }}"

when: desired\_state == "present"

notify:

- restart\_apache

- name: Update virtual host port (only if installing)

ansible.builtin.lineinfile:

path: /etc/apache2/sites-available/000-default.conf

regexp: '^<VirtualHost \\*:'

line: "<VirtualHost \*:{{ listen\_port }}>"

when: desired\_state == "present"

notify:

- restart\_apache

- name: Deploy Apache homepage (only if installing)

ansible.builtin.copy:

dest: /var/www/html/index.html

content: "<h1>This is a demo from Raman Khanna!</h1>"

when: desired\_state == "present"

notify:

- restart\_apache

handlers:

- name: restart\_apache

ansible.builtin.service:

name: "{{ service }}"

enabled: yes

state: restarted

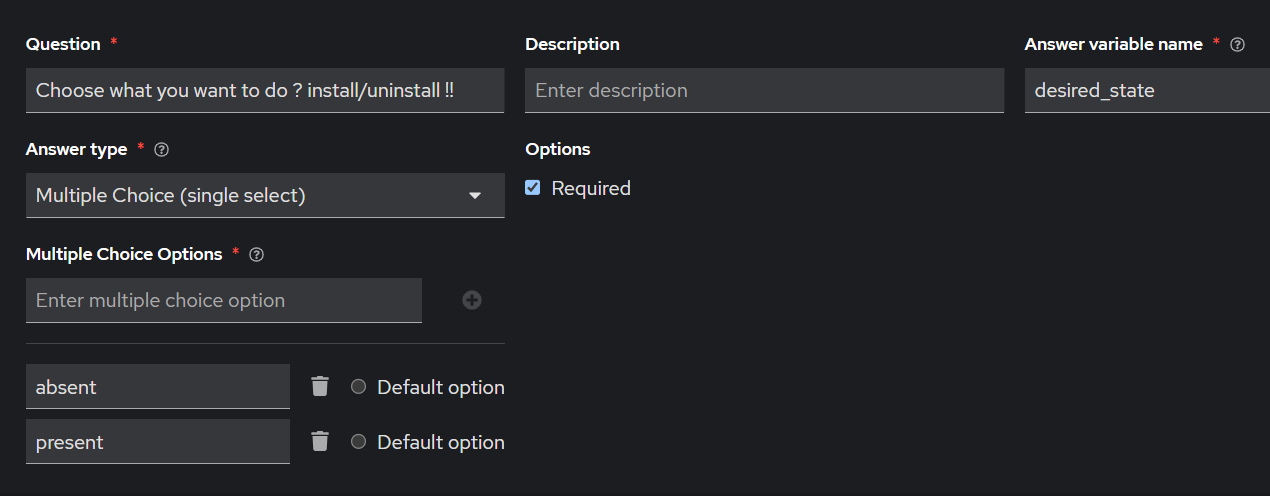
-- sync the project with the third.yml

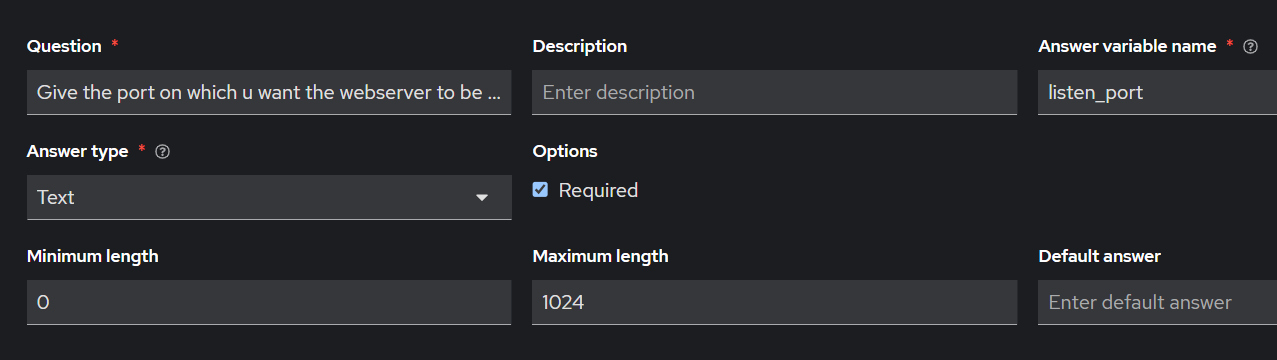
-- update ips on raman-static-inv

--- create a template :

    updating the necessry things : playbook third.yml

    --add the survey for port and state only...





    --- check it first

    -- run it

    -- browse the public ip of m1:82

**LAB5 : Grouping Hosts and Defining Variables**

**Step 1: Create a New Inventory**

1. Navigate to **Inventories** → **Add** → **Inventory**
2. Fill in:
   * **Name**: Ubuntu Inventory
   * **Organization**: <your-org>
   * Click **Save**

**Step 2: Create Groups in the Inventory**

1. Open the Ubuntu Inventory
2. Go to the **Groups** tab → **Add Group**
   * Group 1:
     + **Name**: web
   * Group 2:
     + **Name**: db

**Step 3: Add Hosts to Groups**

**Add Hosts to web group:**

1. Click on **web** group → **Hosts** tab → **Add Host**
   * **Host Name**: ubuntu-node1
   * add ansible\_host: 15.229.78.109 in variables )

**Add Hosts to db group:**

1. Click on **db** group → **Hosts** tab → **Add Host**
   * **Host Name**: ubuntu-node2
   * add ansible\_host: 192.168.1.102in variables )

**Step 4: Define Group Variables**

**For web group:**

1. Go to web group → **Variables** tab
2. Add the following in **YAML** format:

role: webserver

pkg\_name: nginx

service\_name: nginx

**For db group:**

1. Go to db group → **Variables** tab
2. Add:

role: database

pkg\_name: mysql-server

service\_name: mysql

**Step 5: Define Host-level Variable (Override Example)**

Let’s override the pkg\_name for ubuntu-node2:

1. Go to ubuntu-node2 under db group → **Variables** tab
2. Add:

pkg\_name: mariadb-server

**Step 6: Create a Project with a Git Repository**

1. Navigate to **Projects** → **Add**
2. Fill in:
   * **Name**: Grouping Lab Project
   * **SCM Type**: Git
   * **SCM URL**: https://github.com/<your-org>/aap-lab-grouping.git  
     *(You can host a repo with the below playbook)*
   * **SCM Branch**: main
   * **Execution Environment**: Default

**Step 7: Create the Playbook**

Create a file called install\_service.yml in your Git repository:

---

- name: Install and start services based on group vars

hosts: all

#hosts: web

#hosts:db

tasks:

- name: Install required package

apt:

name: "{{ pkg\_name }}"

state: present

update\_cache: yes

- name: Start and enable the service

service:

name: "{{ service\_name }}"

state: started

enabled: yes

- name: Print role info

debug:

msg: "This host is configured as a {{ role }}"

Push this playbook to GitHub.

**Step 8: Create a Credential for Ubuntu Nodes**

1. Go to **Credentials** → **Add**
2. Fill in:
   * **Name**: Ubuntu SSH Credential
   * **Credential Type**: Machine
   * **Username**: ubuntu
   * **Private Key**: Paste your SSH private key
   * **Privilege Escalation**: Checked (sudo)

**Step 9: Create a Job Template**

1. Go to **Job Templates** → **Add**
2. Fill in:
   * **Name**: Install Role Based Services
   * **Job Type**: Run
   * **Inventory**: Ubuntu Inventory
   * **Project**: Grouping Lab Project
   * **Playbook**: install\_service.yml
   * **Credentials**: Ubuntu SSH Credential
   * Click **Save**

**Step 10: Launch and Observe Output**

1. Launch the job template
2. Watch the output:
   * ubuntu-node1 installs **nginx** and sets role as webserver
   * ubuntu-node2 installs **mariadb-server** (host var overrides group var) and role as database

**🔍 Validation & Learning Points**

| **Feature** | **Demonstrated By** |
| --- | --- |
| Host grouping | Hosts assigned to web and db groups |
| Group variables | pkg\_name, role, service\_name at group level |
| Host variable override | pkg\_name: mariadb-server for ubuntu-node2 |
| Variable usage | Variables used in the playbook with {{ var }} syntax |

| **Precedence Level** | **Variable Source** | **Priority** |
| --- | --- | --- |
| 1 | defaults/ in roles | LOW |
| 2 | inventory variables |  |
| 3 | group\_vars/all |  |
| 4 | group\_vars/group\_name |  |
| 5 | host\_vars/hostname |  |
| 6 | vars: in playbook | ✅ HIGH |
| 7 | vars\_files: in playbook | ✅ HIGHER |
| 8 | include\_vars (inline vars loading) | ✅ HIGHEST (among manual includes) |
| 9 | CLI with -e (extra vars) | 🛑 HIGHEST of ALL |
|  |  |  |
|  |  |  |
|  |  |  |

**LAB 6 : VARIABLE DECLARATION USING VARS.YAML**

**- hosts: web,db**

**vars\_files:**

**- vars.yml**

**tasks:**

**- name: Update apt package index**

**apt:**

**update\_cache: yes**

**- name: Install {{ package }} package**

**package:**

**name: "{{ package }}"**

**state: "{{ desired\_state }}"**

**- name: Configure hostname and IP in webpage**

**template:**

**#copy:**

**src: index.html**

**dest: /var/www/html/index.html**

**when: desired\_state == "present"**

**notify: Restart {{ package }} service**

**handlers:**

**- name: Restart {{ package }} service**

**service:**

**name: "{{ package }}"**

**state: restarted**

* + - **vars.yml :**

**package: apache2**

**desired\_state: present**

**web\_content: |**

**<h1>This is a demo for Apache Webserver!</h1>**

**<p>Hostname: {{ ansible\_hostname }}</p>**

**<p>IP Address: {{ ansible\_default\_ipv4.address }}</p>**

**<p>OperatingSystem : {{ ansible\_os\_family }}</p>**

* + - **index.html :**

**{{ web\_content }}**

* + - **now add package: mariadb-server in groups and launch to checkif it overrides the default vars.yml**
    - **it will not override cz group/vars and host/vars have low priority than vars\_file**

| **Precedence Level** | **Variable Source** | **Priority** |
| --- | --- | --- |
| 1 | defaults/ in roles | LOW |
| 2 | inventory variables |  |
| 3 | group\_vars/all |  |
| 4 | group\_vars/group\_name |  |
| 5 | host\_vars/hostname |  |
| 6 | vars: in playbook | ✅ HIGH |
| 7 | vars\_files: in playbook | ✅ HIGHER |
| 8 | include\_vars (inline vars loading) | ✅ HIGHEST (among manual includes) |
| 9 | CLI with -e (extra vars) | 🛑 HIGHEST of ALL |
|  |  |  |
|  |  |  |

**Lab 7: Variable Precedence using group\_vars and host\_vars**

**🎯 Objective:**

Demonstrate how variables defined in group\_vars and host\_vars override each other, and how they affect playbook execution in an inventory synced from a GitHub project.

**🛠️ Lab Setup**

**1. Prepare GitHub Repo Structure**

Push the following structure to your GitHub repo (e.g., aap-lab4-demo):

\_13\_\_test\_project/

├── group\_vars/

│ ├── web

│ └── db

├── host\_vars/

│ ├── m1

│ └── m2

├── inv

├── var3.yml

├── index.html

└── README.md

**📁 File Details**

**inv (Inventory File)**

[web]

m1 ansible\_host=177.71.159.199

[web:vars]

web\_content=<h1>This is a demo</h1><p>Host: {{ ansible\_hostname }}</p>

[db]

m2 ansible\_host=52.67.104.173

**group\_vars/web**

pkg: apache2

**group\_vars/db**

pkg:

* + - mariadb-server
    - apache2

**host\_vars/m1**

location: Mumbai DC

role: WebServer

env: production

**host\_vars/m2**

location: São Paulo DC

role: DBServer

env: staging

**index.html**

<html>

<head><title>Demo Page</title></head>

<body>

<h1>This is a demo</h1>

<p>Host: {{ ansible\_hostname }}</p>

<p>Role: {{ role }}</p>

<p>Location: {{ location }}</p>

<p>Environment: {{ env }}</p>

</body>

</html>

**var3.yml (Playbook)**

- hosts: all

tasks:

- name: Update apt package index

apt:

update\_cache: yes

when: ansible\_os\_family == "Debian"

ignore\_errors: true

- name: Install {{ pkg }} package

package:

name: "{{ pkg }}"

state: present

ignore\_errors: true

- name: Configure hostname and IP in webpage

template:

src: index.html

dest: /var/www/html/index.html

notify: Restart service

ignore\_errors: true

- name: Show assigned role

debug:

msg: "Host {{ inventory\_hostname }} has role: {{ role }} and is located in {{ location }} (env: {{ env }})"

handlers:

- name: Restart service

service:

name: "{{ pkg }}"

state: restarted

**📦 AAP UI Configuration**

**1. Import GitHub Project**

* Go to **Resources → Projects**
* Create a new Project named aap-lab4-project
* Git URL: https://github.com/<your-user>/aap-lab4-demo.git
* Sync it.

**2. Create Inventory**

* Navigate to **Resources → Inventories**
* Create a new Inventory: lab4-inventory
* Add a Source:
  + Source: **Sourced from Project**
  + Project: aap-lab4-project
  + Inventory File: inv
  + Sync

**3. Create Template**

* Go to **Templates**
* Add a new Job Template:
  + Name: Install & Configure with group\_vars/host\_vars
  + Inventory: lab4-inventory
  + Project: aap-lab4-project
  + Playbook: var3.yml

**🚀 Execution & Verification**

1. **Launch the Job Template**
2. **Observe output:**
   * Correct packages are installed per host group (apache2, mariadb-server)
   * HTML file created using values from host\_vars
   * debug task outputs role, location, and env per host

**✅ Expected Output Snippets:**

* TASK [Install apache2 package] on m1
* TASK [Install mariadb-server package] on m2
* TASK [Show assigned role]:

"msg": "Host m1 has role: WebServer and is located in Mumbai DC (env: production)"

"msg": "Host m2 has role: DBServer and is located in São Paulo DC (env: staging)"

**🧪 Experiment Ideas**

* Override pkg in host\_vars/m1 and observe precedence.
* Add a variable with the same name in var3.yml using vars: and see how it overrides group/host variables.
* Create a new host and test fallback to group\_vars.

**Lab 8 : Access Control & Role-Based Access in Ansible Automation Platform (AAP)**

**🧠 Learning Objectives**

By the end of this lab, learners will be able to:

* Understand the RBAC model in AAP.
* Create and manage **Organizations**, **Users**, and **Teams**.
* Assign **Roles** to Users and Teams.
* Demonstrate **scoped access** to AAP resources (Projects, Inventories, Templates, etc).
* Practice **least-privilege** principles in automation access.

**🔧 Lab Prerequisites**

* Ansible Automation Platform 2.5+ installed (Controller access via UI).
* Admin credentials to log into AAP Controller.
* At least one Git-based Project (Optional: Sync Git inventory or playbook).
* One Inventory and one Job Template pre-created by the admin.

**🗂️ Lab Structure**

1. **Create an Organization**
2. **Create Users and assign them to Organization**
3. **Create a Team and assign users to it**
4. **Set Roles on resources (Inventory, Projects, Templates)**
5. **Verify access behavior**
6. **Test Job Template execution based on permissions**

**1️⃣ Step 1: Create an Organization**

Organizations are top-level tenants in AAP that isolate users, teams, inventories, credentials, etc.

**🖥️ Via AAP UI**

1. Go to **Access → Organizations**
2. Click **Add (+)** in the top right.
3. Fill in the following:
   * **Name:** DemoOrg
   * **Description:** Demo Organization for RBAC lab
4. Click **Save**

**2️⃣ Step 2: Create Users**

**🔧 Users:**

We’ll create 3 users with different access levels.

| **Username** | **Role** | **Description** |
| --- | --- | --- |
| alice | Admin | Organization Admin |
| bob | Operator | Limited to run job templates |
| carol | Read-only | Viewer for inventories & templates |

**🖥️ Create Users (UI Steps)**

1. Go to **Access → Users**
2. Click **Add (+)**

For each user:

* **Username:** e.g., alice
* **Password:** e.g., Redhat@123
* **Email:** alice@example.com
* **User type:** Normal
* Click **Save**

➡️ After user creation:

1. Edit each user → Assign to **DemoOrg** using **Organizations** tab.
2. For alice, assign **Admin** role under Organization → **Users → Roles tab**
3. Leave bob and carol with no Org role for now.

**3️⃣ Step 3: Create a Team**

**🤝 Why Teams?**

Teams allow group-based RBAC. Easier to manage than assigning roles individually.

1. Go to **Access → Teams**
2. Click **Add (+)**
   * **Name:** OpsTeam
   * **Organization:** DemoOrg
   * **Description:** Operations Team for executing jobs
   * Click **Save**
3. Open the newly created team → **Access → Users** → **Add Members**
   * Add bob and carol

**4️⃣ Step 4: Assign Resource Roles**

Now we’ll define **fine-grained permissions** using roles.

**🔸 Create or Use Existing Resources**

Ensure these exist (or create them if needed):

* **Project:** demo-project (Git-based)
* **Inventory:** demo-inventory
* **Credential:** demo-ssh
* **Job Template:** demo-job

All should be created by admin or alice (who has Org Admin rights)

**🔐 Assign Roles**

| **Resource** | **Role Assigned** | **To** | **Purpose** |
| --- | --- | --- | --- |
| Project | Use | OpsTeam | To allow using project in job template |
| Inventory | Use | OpsTeam | Required to launch job |
| Credential | Use | OpsTeam | Credential required to execute jobs |
| Job Template | Execute | OpsTeam | Allows job launch without editing |
| Job Template | Read | carol | View-only access |

**🖥️ UI Steps to Assign Roles**

1. Open each resource (Project, Inventory, etc.)
2. Go to **Access** tab
3. Click **Add (+)** under **Roles**
4. Select the role (Use, Execute, etc.)
5. Assign to **Team (OpsTeam)** or **User (carol)**

**5️⃣ Step 5: Test User Access**

Now test behavior by logging in as each user.

**🔍 bob (OpsTeam member)**

* ✅ Should be able to see and launch the **demo-job**
* ❌ Cannot edit inventory/project/template
* ❌ Cannot create new templates

**🔍 carol (Viewer)**

* ✅ Should be able to **view** the job template, project, and inventory.
* ❌ Cannot launch or edit anything.

**🔍 alice (Org Admin)**

* ✅ Can do **everything** inside the DemoOrg

**6️⃣ Step 6: Verify Least Privilege**

Try below actions with limited users:

* Try to **edit job template** as bob → ❌ Permission denied
* Try to **launch job** as bob → ✅ Success
* Try to **view job results** as carol → ✅ Allowed
* Try to **modify inventory/project** as carol → ❌ Not allowed

**🧪 Optional: Use CLI for RBAC Assignments (Bonus)**

AAP also supports RBAC via awx CLI or API:

bash

CopyEdit

# Login

awx login --conf.host https://<AAP\_URL> --username admin --password '<password>'

# Assign Execute role on job template to a team

awx role grant --role execute --target-type job\_template --target-name demo-job --team OpsTeam

**📝 Summary**

| **Concept** | **Description** |
| --- | --- |
| **Organization** | Isolates users/resources |
| **Users** | Individuals with login |
| **Teams** | Group of users for role assignment |
| **Roles** | Control access (Use, Execute, Admin, etc.) |
| **RBAC Model** | Assign roles per resource per user/team |

**✅ Deliverables for Lab**

* One Organization DemoOrg
* Three users with varied access
* One Team OpsTeam
* Job Template demo-job with restricted Execute permissions
* Verification of RBAC through UI login and job launch tests

**LAB 9 :Configure GitHub Webhook**

**📘 What is a Webhook?**

A **Webhook** lets GitHub/GitLab **trigger a job** automatically when:

* Code is pushed
* A PR is merged
* A specific tag is created

**🧠 Use Case**

Whenever you git push to your Ansible project repo, AAP **automatically re-runs** a deployment job.

**🛠️ Steps to Enable Webhook**

**A. Enable Webhook in Job Template**

1. Go to your Job Template → **Edit**
2. Scroll down to **Webhooks** section:
   * ✅ Enable Webhook
   * Webhook Service: GitHub (or GitLab)
3. Save

AAP generates:

* A **Webhook URL**
* A **Secret Token**

\* Save the template , come back again ; than u wl see the secret token

**B. Add Webhook in GitHub**

1. Go to your GitHub repo → **Settings → Webhooks → Add webhook**
2. Paste:
   * **Payload URL**: Use the Webhook URL from AAP
   * **Content type**: application/json
   * **Secret**: Use the secret token from AAP
3. Disable the certificate validation step
4. Choose:
   * ✅ Just the push event

Click **Save**

**🧪 Test**

1. Make a code change in your repo
2. Commit & push:

bash

CopyEdit

git commit -am "Trigger webhook"

git push origin main

1. Go to AAP → **Jobs** and see the job automatically triggered 🎉

**🔐 Lab 10: Using Ansible Vault with AAP 2.5**

**🎯 Objective:**

Securely manage sensitive data (like passwords) using Ansible Vault, and use those variables in an Ansible Automation Platform (AAP) 2.5 Job Template.

**Step 1: 🔑 Encrypt Secrets using Ansible Vault**

Create a secrets file using CLI:

ansible-vault encrypt\_string 'ramankhanna' --name 'encrypted\_web\_password' >> vault\_secrets.yml

ansible-vault encrypt\_string 'khannaraman' --name 'encrypted\_db\_password' >> vault\_secrets.yml

cat vault\_secrets.yml

**Step 2: 🔄 Upload to AAP as Project Files**

Your directory structure should look like this:

.

├── inv # Inventory file (INI or YAML format)

├── var4.yml # Playbook

├── vault\_secrets.yml # Encrypted vault file

├── group\_vars/

│ ├── web

│ └── db

* + - Var4.yml :

---

- name: Example Playbook

hosts: web,db

vars\_files:

- vault\_secrets.yml

tasks:

- name: Show encrypted web password

debug:

var: encrypted\_web\_password

- name: Show encrypted db password

debug:

var: encrypted\_db\_password

--- it will fail first ; u have to add credential as vault crdential inside ur AAP credentials in vault type : “raman”

OR

* + - Var4.yml :

---

- name: Secure Automation with Ansible Vault on Ubuntu

hosts: all

become: true

gather\_facts: true

vars\_files:

- vault\_secrets.yml

tasks:

- name: Debug current host and group

debug:

msg: "Running on host {{ inventory\_hostname }} in group {{ group\_names }}"

- name: Show Encrypted Web Password (web group only)

debug:

var: encrypted\_web\_password

when: "'web' in group\_names"

- name: Show Encrypted DB Password (db group only)

debug:

var: encrypted\_db\_password

when: "'db' in group\_names"

- name: Ensure required packages are installed

apt:

name: "{{ pkg }}"

state: present

update\_cache: yes

when: pkg is defined

- name: Create a user with vault password

user:

name: appuser

password: "{{ (encrypted\_web\_password if 'web' in group\_names else encrypted\_db\_password) }}"

shell: /bin/bash

state: present

- name: Create application configuration file with credentials

copy:

dest: "/etc/app\_secure.conf"

content: |

[credentials]

user = appuser

password = {{ encrypted\_web\_password if 'web' in group\_names else encrypted\_db\_password }}

owner: root

group: root

mode: '0600'

- name: Ensure the application service is running (apache2 or mysql)

service:

name: "{{ 'apache2' if 'web' in group\_names else 'mysql' }}"

state: started

enabled: true

📂 Sample group\_vars/web:

pkg: apache2 #optional

ansible\_password: "{{ encrypted\_web\_password }}"

📂 Sample group\_vars/db:

pkg: mariadb-server #optional

ansible\_password: "{{ encrypted\_db\_password }}"

📝 Sample inv (INI format):

[web]

m1 ansible\_host=54.207.153.32

#ansible\_user=admin

[db]

m2 ansible\_host=52.67.59.218

#ansible\_user=dbadmin

[all:vars]

ansible\_user=ubuntu

✅ Confirm that vault\_secrets.yml is in the **root of your project repo**, so it is synced with the AAP project.

**Step 3: 📦 Add Project in AAP**

1. Go to **Projects** → ➕ **Add**
2. Use your SCM (Git, etc.) or upload files manually to /var/lib/awx/projects/\_X\_\_your\_project\_name.
3. Ensure it's syncing correctly (check logs under Project).

**Step 4: 📂 Create a git based Inventory in AAP**

1. Go to **Inventories** → ➕ **Add Inventory**
2. Use the **"Custom Inventory File"** option.
3. Use inv as your inventory source file (you can upload via the Project path).

OR

1. If using dynamic inventory plugin (like AWS EC2), configure your aws\_ec2.yaml with keyed\_groups + tag filtering.

**Step 5: 🔐 Configure Vault Credential in AAP**

1. Go to **Credentials** → ➕ **Add**
2. Select **Vault** as the credential type.
3. Input the **Vault password** manually ( in our case we had setup password as “raman”)

**Step 6: 🧪 Create Job Template**

1. Go to **Templates** → ➕ **Add Job Template**
2. Set:
   * **Inventory**: The one created above.
   * **Project**: The one with the playbook and vault\_secrets.yml
   * **Playbook**: var4.yml
   * **Credential**: Select a Machine credential and the Vault credential.
3. Save and **Launch**.

Use the **community.general.password\_hash** filter (provided by the community.general collection) instead of the built-in crypt-based one.

✅ Watch the job output:

TASK [Show encrypted web password] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [m1] => {

"encrypted\_web\_password": "webpass123"

}

TASK [Show encrypted db password] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ok: [m2] => {

"encrypted\_db\_password": "dbpass456"

}

**✅ Troubleshooting Tips**

| **Error** | **Fix** |
| --- | --- |
| Inline Jinja variables are not allowed | Never use {{ }} in inventory files in AAP. Instead use them in group\_vars. |
| Vault format unhexlify error | Vault file might be malformed. Regenerate using encrypt\_string ensuring line breaks between secrets. |
| Variable not defined | Ensure vault\_secrets.yml is in project root and referenced via vars\_files: |
| YAML parse error | Always quote {{ }} when used inside YAML (ansible\_password: "{{ encrypted\_password }}") |

**🔧 Optimizing Performance & Troubleshooting in Ansible Automation Platform (AAP)**

**🚀 1. Best Practices for Scaling Ansible Automation**

**🔹 a. Efficient Inventory and Targeting**

* **Use Smart Inventories**: Dynamically select hosts using filters and host facts (e.g., OS, environment, tags).
* **Limit Scope with --limit**: Avoid running against the full inventory. Target smaller groups or individual hosts.
* **Fact Caching**: Enable Redis-based fact caching to reduce runtime setup overhead.

[defaults]

fact\_caching = redis

fact\_caching\_timeout = 7200

**🔹 b. Job Execution Optimization**

* **Adjust forks Value**:
  + Default is 5. For larger inventories, increase to 10–50 depending on system resources.
  + Set via controller → job template → Execution → Forks or in ansible.cfg.
* **Use Job Slicing**:
  + Splits a job across multiple execution nodes (in a controller cluster) or multiple slices on a single node.
  + Useful for parallel execution across large inventories.
  + Example: 10 slices for 1000 hosts = 100 hosts per slice.

**❌ Without Job Slicing**

[Single Job Template Execution]

▶ Targets: 1000 hosts

▶ Forks = 20 → Runs on 20 hosts at a time

▶ 1000 ÷ 20 = 50 rounds

▶ Total time = Long

**✅ With Job Slicing (10 Slices)**

[10 Jobs Running in Parallel]

▶ Each targets 100 hosts

▶ Each slice uses Forks = 10

▶ Each slice does 10 rounds

▶ All slices run at same time → much faster!

**✅ Key Takeaways:**

* **Forks** = How many tasks run at once **inside** a job.
* **Job Slicing** = How many jobs run **in parallel** for the same playbook.
* **More slices + right forks = huge time savings**, especially with large inventories.
* **Enable Asynchronous Tasks**:
  + Offload long-running tasks to the background:

- name: Install large package

yum:

name: some-big-pkg

state: present

async: 300

poll: 10

**🔹 c. Execution Environment Design**

* **Slim EE Images**:
  + Minimize size for faster startup.
  + Avoid installing unnecessary collections or dependencies.
* **Multiple EEs per Use Case**:
  + Avoid a "one-size-fits-all" EE. Customize EEs for network, storage, cloud, and OS-specific roles.

**🔹 d. Controller Scaling Best Practices**

* **Controller Clustering**:
  + Scale horizontally using multiple control nodes.
  + Shared PostgreSQL database + NFS or S3-compatible file storage required.
* **Job Isolation**:
  + Use isolated execution nodes to distribute workload (e.g., for data center separation or DMZ zones).
* **Heartbeat & Capacity Rules**:
  + Set **Instance Group** rules to restrict certain jobs to specific nodes.

**🛠️ 2. Identifying and Resolving Common Bottlenecks**

**🔍 a. Debugging Playbook Performance**

| **Symptom** | **Possible Cause** | **Resolution** |
| --- | --- | --- |
| Sluggish execution | Inefficient loops, facts gathering on all hosts | Use gather\_facts: false, optimize loops with with\_items |
| High latency between tasks | Sequential operations | Use async, parallelism, or batch\_size |
| Redundant task execution | Tasks not properly idempotent | Use changed\_when, check\_mode validation |
| Excessive disk I/O | Large templates, logs, or output | Redirect stdout to files, limit verbosity |

**✅ System Logs in AAP 2.5 (Automation Controller)**

**🔹 Default Log Directory:**

Even in AAP, the **container names and log directories** retain the old tower naming convention:

/var/log/tower/

**🔍 Key Log Files:**

| **Log File** | **Purpose** |
| --- | --- |
| /var/log/tower/tower.log | Main controller operations – jobs, API calls, user actions |
| /var/log/tower/task\_system.log | Task management system – job status, dispatcher |
| /var/log/tower/callback\_receiver.log | Websocket/callback event handler – job output reception |
| /var/log/tower/notifications.log | Notifications-related events (email, webhook, etc.) |
| /var/log/tower/dispatcher.log | Responsible for job distribution to execution nodes |
| /var/log/tower/awx.log | Older Tower-related logs (still relevant in some AAP 2.x builds) |

✅ **Note**: If you're running AAP in a **containerized setup**, these logs may be mounted via volumes or accessed via podman logs (or docker logs) if you’re using container images directly.

072 podman logs automation-controller-task | less

1073 podman logs automation-controller-web | less

1074 clear

Note : add -f to see real time logs

**✅ Containers of Interest (AAP 2.5)**

| **Container Name** | **Purpose** |
| --- | --- |
| automation-controller-task | Main place to check for **job execution, job failure, and task logs** |
| automation-controller-web | API calls, UI actions, user events, and REST API-related errors |
| automation-controller-rsyslog | Internal rsyslog handler for logs forwarding/debugging |
| automation-hub-\* | Logs for content syncs, publishing errors in Automation Hub |
| automation-eda-\* | Event-Driven Ansible activity logs |

**📌 Logging Configuration via the UI**

As of AAP 2.4+, some logging features can be configured via the **Automation Controller UI**:

* Navigate to:  
  ⚙️ **Settings** → **System** → **Logging Settings**

From here, you can configure:

* External log aggregation (Syslog, Splunk, etc.)
* Enable/disable **Job Event Stdout Capture**
* Set logging levels for stdout and event handlers

**🔹 1. Start With Job Execution Logs**

Most **playbook job failures, module errors**, or **template issues** will appear in:

podman logs automation-controller-task

Look for:

* "Traceback" → Python exceptions from Ansible core or modules
* "rc": 1 → Return code from task execution
* "msg": ... → Output from failed Ansible modules
* "module\_stdout", "module\_stderr" → For shell/command tasks
* "unreachable", "FAILED", "no hosts matched" → Common Ansible-level errors

You can filter with:

podman logs automation-controller-task | grep -i 'FAILED\|Traceback\|rc\|error'

**🔹 2. Dive into Web/API Issues**

If you're seeing issues in:

* Credential validation
* Inventory syncs
* SCM/project updates
* User API calls

Then use:

podman logs automation-controller-web

Look for:

* django.request → API call logs
* status\_code=403|401 → Permission/auth errors
* status\_code=500 → Internal failures
* Exception: or Traceback (most recent call last):

**🔹 3. Debugging Job Failures by Job ID (Best Practice)**

1. From AAP GUI → Go to **Jobs** → Click on a failed job → Note the **Job ID**
2. Then run:

podman logs automation-controller-task | grep "<JOB\_ID>"

But: The raw job ID **might not always appear directly** in podman logs. Instead, look for the **UUID** or the **job type**, e.g.:

podman logs automation-controller-task | grep 'running job'

Or use journal-style filtering:

podman logs automation-controller-task | less

# Then search for a host or playbook name using / in less

**🔬 Lab 11: Advanced Ansible Playbook Features on AAP 2.5**

**🧪 Lab Objectives**

* Use **handlers** to perform actions triggered by tasks.
* Implement **loops** for repetitive task execution.
* Utilize **conditionals** to control task execution.
* Apply **Jinja2 templates** for dynamic configuration generation.
* Run everything through **Ansible Automation Platform 2.5** interface (Projects, Templates, EEs).

**🧱 Prerequisites**

* AAP 2.5 installed and accessible.
* One controller node (AAP), **two Ubuntu managed nodes** (added as inventory).
* Working **Execution Environment (EE)** configured and assigned.
* A **Project** configured to pull playbooks from a Git repository or manual upload.

**📜 Playbook: advanced\_features.yml**

---

- name: Demonstrate Advanced Playbook Features

hosts: webservers

become: true

vars\_files:

- vars.yml

tasks:

- name: Install NGINX

apt:

name: nginx

state: present

update\_cache: yes

notify: restart nginx

- name: Create multiple directories using loop

file:

path: "/var/www/{{ item }}"

state: directory

owner: www-data

mode: '0755'

loop:

- site1

- site2

- site3

- name: Deploy index.html in each site folder

copy:

content: |

<html>

<head><title>Welcome to {{ item }}</title></head>

<body>

<h1>Welcome to {{ item }}!</h1>

</body>

</html>

dest: "/var/www/{{ item }}/index.html"

owner: www-data

group: www-data

mode: '0644'

loop:

- site1

- site2

- site3

- name: Conditionally deploy nginx config only on web01

template:

src: nginx.conf.j2

dest: /etc/nginx/nginx.conf

notify: restart nginx

when: inventory\_hostname == "web01"

- name: Check if port 80 is in use

shell: "lsof -i :80 | grep LISTEN | awk '{print $1}' | head -n 1"

register: port\_check

changed\_when: false

failed\_when: false

- name: Debug info on what service is using port 80

debug:

msg: "Port 80 is being used by {{ port\_check.stdout }}"

when: port\_check.stdout != ""

- name: Stop Apache if it's using port 80

service:

name: apache2

state: stopped

when: "'apache2' in port\_check.stdout"

- name: Disable Apache to prevent it from starting on boot

service:

name: apache2

enabled: false

when: "'apache2' in port\_check.stdout"

- name: Ensure NGINX is running

service:

name: nginx

state: started

enabled: true

handlers:

- name: restart nginx

service:

name: nginx

state: restarted

Inv file :

[webservers]

web01 ansible\_host=15.228.73.170

web02 ansible\_host=18.228.166.149

[webservers:vars]

ansible\_user=ubuntu

ansible\_python\_interpreter=/usr/bin/python3

Note : no need to add ansible\_ssh\_private\_key\_file=~/.ssh/id\_rsa instead create machine credentials.

**📄 Jinja2 Template: nginx.conf.j2**

user www-data;

worker\_processes auto;

pid /run/nginx.pid;

events {

worker\_connections 768;

}

http {

sendfile on;

tcp\_nopush on;

tcp\_nodelay on;

keepalive\_timeout 65;

types\_hash\_max\_size 2048;

include /etc/nginx/mime.types;

default\_type application/octet-stream;

access\_log /var/log/nginx/access.log;

error\_log /var/log/nginx/error.log;

gzip on;

server {

listen 80 default\_server;

listen [::]:80 default\_server;

root /var/www/{{ web\_root | default('html') }};

index index.html;

server\_name \_;

location / {

try\_files $uri $uri/ =404;

}

}

}

**🧾 Variables: vars.yml**

web\_root: site3

**📌 Inventory Setup in AAP**

* Create a gitbased project inventory from the inv file in github :
* It will create webservers group with web01 and web02 hosts..

**⚙️ Project Setup in AAP**

1. **Create Project**
   * Name: Advanced Playbook Project
   * SCM Type: Git / Manual
   * Source: (Git URL or Upload ZIP manually)
   * Set advanced\_features.yml as main playbook.
2. **Create Job Template**
   * Name: Advanced Playbook Demo
   * Inventory: ubuntu-lab-inventory
   * Project: Advanced Playbook Project
   * Execution Environment: (Select your custom Ubuntu EE or default EE)
   * Playbook: advanced\_features.yml

**🚀 Execution Steps**

1. Go to **Templates** → Launch Advanced Playbook Demo.
2. Observe:
   * NGINX is installed.
   * Directories /var/www/site1, site2, site3 are created.
   * Template is only deployed to web01.
   * If template changes, nginx will restart via handler.
3. Verify on web01:
4. curl http://publicIPofweb01

Should show NGINX default page served from /var/www/site3.

1. On web02 , it will remain normal nginx page ..

**🔍 Lab Validations**

| **Feature** | **Check** |
| --- | --- |
| Handlers | NGINX restarts after config changes |
| Loops | Directories site1, site2, site3 created |
| Conditionals | Template applied only on web01 |
| Jinja2 Templating | Configured NGINX root using {{ web\_root }} |

**✅ Lab Outcome**

This lab reinforces AAP users' understanding of:

* Advanced control over playbook execution using handlers and conditionals.
* Templating and data-driven configurations using Jinja2.
* Efficient use of loops to reduce redundancy.

**🧪 Lab 12: Error Handling**

**error\_handling\_demo.yml**

This includes:

✅ Triggers a **real error** by trying to install a **non-existent package**  
✅ Uses **rescue** block to recover from failure  
✅ Logs **error message to a file** on each host  
✅ Uses a **handler** triggered only during rescue

**🧾 Playbook Code**

---

- name: Demonstrate Advanced Error Handling in AAP 2.5

hosts: web

become: yes

gather\_facts: true

vars:

bad\_package: "fakepackage123"

log\_file: "/var/log/aap\_error\_handling.log"

rescue\_triggered: false # default

tasks:

- name: Task 1 - Ensure Python is installed

apt:

name: python3

state: present

- name: Block of tasks with real error (bad package)

block:

- name: Task 2 - Attempt to install a fake package (intentional error)

apt:

name: "{{ bad\_package }}"

state: present

notify: Log bad package install failure

#- name: We can add more tasks under this block

#- name:

#- name:

rescue:

- name: Set fact that rescue was triggered

set\_fact:

rescue\_triggered: true

# This user defined fact will be used later to confirm that the rescue was completed .

- name: Rescue - Log the error to file

copy:

content: |

[{{ ansible\_date\_time.iso8601 }}] ERROR: Failed to install {{ bad\_package }} on {{ inventory\_hostname }}. Rescue block executed.

dest: "{{ log\_file }}"

force: no

mode: '0644'

owner: root

group: root

remote\_src: no

- name: Rescue - Install an alternative valid package

apt:

name: cowsay

state: present

# Installs a real package (cowsay) instead of failing completely – a nice way to recover and continue.

- name: Notify rescue handler

meta: flush\_handlers

# Forces Ansible to run any queued handlers immediately, instead of at the end of the play.

always:

- name: Always - Final message after error handling block

debug:

msg: "Block + Rescue + Always complete on {{ inventory\_hostname }}"

# tasks under "always" runs no matter what, even if the above block succeeded or failed .Often used for cleanup or final status logging.

# block ,rescue , always blocks ends now ; contnuing with the playbook....

- name: Task 3 - Print message based on rescue status

debug:

msg: "Rescue was triggered on {{ inventory\_hostname }}"

when: rescue\_triggered | bool

# Only runs if the rescue\_triggered variable is true.

- name: Task 4 - Final verification

debug:

msg: "Playbook completed successfully on {{ inventory\_hostname }}"

handlers:

- name: Log bad package install failure

lineinfile:

path: "{{ log\_file }}"

line: "[{{ ansible\_date\_time.iso8601 }}] HANDLER: Package {{ bad\_package }} install failed on {{ inventory\_hostname }}."

create: yes

mode: '0644'

**🧠 How It Works**

| **Feature** | **What it does** |
| --- | --- |
| bad\_package | Simulates a real package error (fakepackage123) |
| block + rescue | Catches apt failure and logs to a host-local log file |
| copy task | Writes a log message to /var/log/aap\_error\_handling.log |
| set\_fact | Indicates rescue was triggered |
| notify and handlers | Writes a line to the log file only during failure |
| meta: flush\_handlers | Forces immediate handler execution inside rescue |

**🧪 Verification Steps**

**After Running:**

Check on both managed nodes:

cat /var/log/aap\_error\_handling.log

Expected content:

[2025-04-16T14:33:45Z] ERROR: Failed to install fakepackage123 on ubuntu-node1. Rescue block executed.

[2025-04-16T14:33:45Z] HANDLER: Package fakepackage123 install failed on ubuntu-node1.

**Lab 13 : Custom Execution Environments (EEs) – Build & Use in AAP**

**🎯 Lab Objectives**

By the end of this lab, you’ll be able to:

* Understand what Execution Environments (EEs) are
* Use default Red Hat-provided EEs
* Build custom EEs with:
  + Additional Python modules
  + Roles & collections
  + OS packages
* Upload custom EE image to a private registry
* Configure AAP to use the custom EE in Job Templates

**🧠 What are Execution Environments?**

In AAP 2.x, Execution Environments replace Python venvs. They are **container images** that encapsulate everything needed to run your Ansible automation:

* Ansible core + ansible-runner
* Python modules (e.g., boto3, docker)
* Collections & Roles
* System utilities (jq, curl, etc.)

🧱 They are **OCI-compatible container images**, usually built with [Ansible Builder](https://ansible.readthedocs.io/projects/builder/en/latest/).

**🔧 Step 1: Install Ansible Builder (Control Node)**

pip install ansible-builder

✅ Ensure you also have **podman** or **docker** installed.

**📁 Step 2: Define EE Build Files**

Create a new directory:

mkdir ee-lamp && cd ee-lamp

[ec2-user@ip-172-31-33-203 ee-lamp]$ ansible-builder --version

**📄 execution-environment.yml**

---

version: 3

images:

base\_image:

name: quay.io/rockylinux/rockylinux:9

dependencies:

python: requirements.txt

system: bindep.txt

galaxy: requirements.yml

# Optional: Install specific ansible-core version via pip

ansible\_core:

package\_pip: ansible-core==2.15.4

ansible\_runner:

package\_pip: ansible-runner==2.3.3 # Or the latest supported version

# Optional additional build steps (e.g., copying ansible.cfg)

#additional\_build\_steps:

#prepend\_final:

#- COPY \_build/files/ansible.cfg /etc/ansible/ansible.cfg

**📄 requirements.yml (for Collections)**

---

collections:

- name: community.general

version: 7.2.0

**📄 requirements.txt (Python modules)**

requests

pyyaml

lxml

boto3

**📄 bindep.txt (OS packages)**

# System-level packages for roles that use tools

dnf

git

tree

jq

gcc

python3-devel

libffi-devel

openssl-devel

[ec2-user@ip-172-31-33-203 ee-lamp]$ cat files/ansible.cfg

[defaults]\nhost\_key\_checking = False

* **Login to registry.redhat.io** using podman:

podman login registry.redhat.io

You'll be prompted for:

* **Username**: your **Red Hat Customer Portal** username (or service account username)
* **Password**: your **Red Hat password**, or preferably an **Offline Token** from a service account

**🏗️ Step 3: Build the EE**

podman image prune -a

682 ansible-builder build --tag ee-rocky:1.0 --no-cache -v 3

684 podman images

This creates an EE container image locally called ee-lamp:1.0.

The final image includes everything to run your playbooks with AWS, general Linux automation, etc.

**📤 Step 4: Push EE to AAP-accessible Container Registry**

**Example using Podman:**

Podman login docker.io

podman tag ee-rocky:1.0 docker.io/ramann123/myimage:rockyv1

podman images

podman push docker.io/ramann123/myimage:rockyv1

✅ Ensure your AAP controller node can access this registry. Use **Red Hat Quay (** [**https://quay.io/**](https://quay.io/) **)**, **Harbor**, **Docker Hub**, or a **private registry**.

**⚙️ Step 5: Register EE in AAP**

1. Go to **Execution Environments → Add**
2. Fill the form:

| **Field** | **Value** |
| --- | --- |
| Name | Raman-custom-EE |
| Image | registry-1.docker.io/ramann123/myimage:rockyv1 |
|  |  |
| Pull Credential | (Add Docker registry credential if private) |
|  |  |

**🔗 Step 6: Use Custom EE in Job Template**

1. Edit your Job Template
2. Under **Execution Environment**, choose: Custom EE
3. Launch the job and verify that tasks requiring:
   * boto3
   * OS packages (e.g., jq)

...all work as expected.

**🧪 For verification : verify\_ee.yml**

Create a debug task in your playbook:

---

- name: Verify Custom Execution Environment

hosts: localhost

gather\_facts: false

tasks:

- name: Verify boto3 is installed

ansible.builtin.command: python3 -c "import boto3; print(boto3.\_\_version\_\_)"

register: boto3\_output

changed\_when: false

failed\_when: boto3\_output.rc != 0

- debug:

msg: "boto3 version: {{ boto3\_output.stdout }}"

- name: Check jq installation

ansible.builtin.command: jq --version

register: jq\_output

changed\_when: false

failed\_when: jq\_output.rc != 0

- debug:

msg: "jq version: {{ jq\_output.stdout }}"

**🧠 Best Practices**

* Base your EEs on **ee-minimal** and layer only what you need
* Use **tagged versions** instead of latest
* Store EE definitions in Git and automate builds using CI/CD
* Align your EEs with **execution policies** (e.g., no Internet on controller? Pre-build all dependencies)

**✅ Lab Completion Checklist**

| **Task** | **Status** |
| --- | --- |
| Installed ansible-builder | ✅ |
| EE files defined correctly | ✅ |
| EE built with Ansible collections + pip + bindep | ✅ |
| EE pushed to container registry | ✅ |
| EE registered in AAP | ✅ |
| Job template tested with EE | ✅ |

**🔬 Lab14: Structuring Reusable Roles in AAP 2.5**

**⚙️ Lab Objective**

In this lab, you'll:

1. Create a reusable **Ansible role** (webserver) with advanced structure
2. Push the project to a **Git repository**
3. Create a **project** in AAP using the Git repo
4. Use the **default Execution Environment** to run a playbook
5. Deploy **NGINX** on two managed Ubuntu nodes

**🛠️ Prerequisites**

|  |  |
| --- | --- |
| **Component** | **Required** |
| AAP 2.5 installed | ✅ |
| 2 Managed nodes (Ubuntu) | ✅ (SSH and sudo access configured) |
| Internet access on AAP controller | ✅ |
| Git server (GitHub/GitLab or local Git repo) | ✅ |
| Default EE in AAP | ✅ (use Minimal EE or Supported EE) |

**📂 Directory Structure (No Collection)**

aap-webops-project/

├── roles/

│ └── webserver/

│ ├── defaults/

│ │ └── main.yml

│ ├── handlers/

│ │ └── main.yml

│ ├── meta/

│ │ └── main.yml

│ ├── tasks/

│ │ ├── install.yml

│ │ ├── config.yml

│ │ ├── service.yml

│ │ └── main.yml

│ ├── templates/

│ │ └── index.html.j2

│ └── vars/

│ └── main.yml

└── deploy-nginx.yml

**🧱 Create Role**

mkdir -p ~/aap-labs/aap-webops-project

cd ~/aap-labs/aap-webops-project

ansible-galaxy init roles/webserver

**✍️ Role Content**

**roles/webserver/defaults/main.yml**

webserver\_port: 80

webserver\_welcome\_msg: "Welcome to AAP Structured Roles Lab!"

**roles/webserver/vars/main.yml**

nginx\_package: nginx

nginx\_service: nginx

nginx\_docroot: /var/www/html

**roles/webserver/handlers/main.yml**

- name: restart nginx

service:

name: "{{ nginx\_service }}"

state: restarted

**roles/webserver/tasks/install.yml**

- name: Install NGINX package

apt:

name: "{{ nginx\_package }}"

state: present

update\_cache: yes

**roles/webserver/tasks/config.yml**

- name: Create index.html from template

template:

src: index.html.j2

dest: "{{ nginx\_docroot }}/index.html"

notify: restart nginx

**roles/webserver/tasks/service.yml**

- name: Ensure NGINX is running and enabled

service:

name: "{{ nginx\_service }}"

state: started

enabled: yes

**roles/webserver/tasks/main.yml**

- include\_tasks: install.yml

- include\_tasks: config.yml

- include\_tasks: service.yml

**roles/webserver/templates/index.html.j2**

<html>

<head><title>Welcome</title></head>

<body>

<h1>{{ webserver\_welcome\_msg }}</h1>

<p>Deployed by Ansible role: <strong>webserver</strong></p>

</body>

</html>

**roles/webserver/meta/main.yml**

dependencies: []

**📜 Playbook**

**deploy-nginx.yml**

- name: Deploy production-ready NGINX

hosts: all

become: yes

vars:

webserver\_welcome\_msg: "Hello from the Enterprise Role!"

roles:

- webserver

cat inv :

[web]

m1 ansible\_host=54.233.198.135

[db]

m2 ansible\_host=54.233.228.167

**🔁 Git Setup**

git init

git remote add origin https://github.com/<your-org>/aap-webops-project.git

git add .

git commit -m "Standalone webserver role for NGINX deployment"

git push -u origin main

**🧰 AAP Configuration**

**✅ Credential**

* Type: Machine
* Username: ubuntu or appropriate SSH user
* SSH Key: your private key

**✅ Inventory**

* Add both managed Ubuntu nodes
* Verify connection with Ping job

**✅ Project**

* Name: WebOps Project
* Source Control: Git
* URL: https://github.com/<your-org>/aap-webops-project.git
* Playbook Directory: / (root)

**✅ Job Template**

* Name: Deploy NGINX
* Inventory: your inventory
* Project: WebOps Project
* Playbook: deploy-nginx.yml
* Credential: your SSH credential
* Execution Environment: use **default (Minimal EE)**

**🔍 Test and Validate**

curl http://<managed\_node\_ip>

systemctl status nginx

You should see:

* NGINX running
* Custom HTML rendered with your welcome message

**🧪 Bonus Lab Ideas**

* Modify port using webserver\_port
* Add OS-based logic for package manager
* Integrate firewall configuration task
* Add Molecule tests for role

**Discussion on use cases :**

**✅ Use Case Overview: Automated Patching and Security Updates**

**Objective:** Automate the process of applying OS-level security patches to a group of Linux servers (e.g., RHEL, CentOS, Ubuntu) with rollback verification, error reporting, and scheduled execution using Ansible.

**📘 Assumptions & Prerequisites**

1. Ansible controller is set up (Ansible Core or AAP).
2. Target nodes are:
   * Linux systems (RHEL-based and/or Debian-based).
   * Registered with the controller and accessible via SSH.
   * Have sudo privileges.
3. Inventory file is already in place (/etc/ansible/hosts or dynamic inventory).
4. Role-based structure is preferred (reusable).

**🧩 Directory Structure (for roles-based approach)**

bash

CopyEdit

ansible-patch-management/

├── inventories/

│ └── production/

│ └── hosts.ini

├── patch.yml

├── roles/

│ └── patch\_linux/

│ ├── tasks/

│ │ ├── main.yml

│ │ ├── precheck.yml

│ │ ├── patch\_rhel.yml

│ │ └── patch\_debian.yml

│ └── handlers/

│ └── main.yml

└── group\_vars/

└── all.yml

**📄 patch.yml – Main Playbook**

yaml

CopyEdit

---

- name: Automated Patching and Security Updates

hosts: all

become: true

gather\_facts: yes

roles:

- patch\_linux

**📄 group\_vars/all.yml**

yaml

CopyEdit

patch\_reboot: true

patch\_security\_only: true

patch\_log\_path: "/var/log/ansible-patch.log"

**📄 roles/patch\_linux/tasks/main.yml**

yaml

CopyEdit

---

- name: Include pre-patch checks

include\_tasks: precheck.yml

- name: Apply security updates for RHEL

include\_tasks: patch\_rhel.yml

when: ansible\_os\_family == "RedHat"

- name: Apply security updates for Debian

include\_tasks: patch\_debian.yml

when: ansible\_os\_family == "Debian"

**📄 roles/patch\_linux/tasks/precheck.yml**

yaml

CopyEdit

---

- name: Check disk space before patching

shell: df -h /

register: disk\_check

changed\_when: false

- name: Display disk usage

debug:

msg: "{{ disk\_check.stdout\_lines }}"

- name: Backup current package list

shell: rpm -qa > /tmp/rpm-backup-{{ inventory\_hostname }}.txt

when: ansible\_os\_family == "RedHat"

- name: Backup dpkg package list

shell: dpkg --get-selections > /tmp/dpkg-backup-{{ inventory\_hostname }}.txt

when: ansible\_os\_family == "Debian"

**📄 roles/patch\_linux/tasks/patch\_rhel.yml**

yaml

CopyEdit

---

- name: Ensure yum-utils is installed

package:

name: yum-utils

state: present

- name: Update only security patches (RHEL)

yum:

name: "\*"

security: "{{ patch\_security\_only }}"

state: latest

update\_cache: yes

- name: Reboot if required (RHEL)

reboot:

reboot\_timeout: 300

when: patch\_reboot

**📄 roles/patch\_linux/tasks/patch\_debian.yml**

yaml

CopyEdit

---

- name: Update apt cache

apt:

update\_cache: yes

- name: Install unattended-upgrades if missing

apt:

name: unattended-upgrades

state: present

- name: Apply security updates (Debian)

shell: unattended-upgrade -d

args:

warn: false

- name: Reboot if required (Debian)

reboot:

reboot\_timeout: 300

when: patch\_reboot

**📄 roles/patch\_linux/handlers/main.yml**

yaml

CopyEdit

---

- name: Restart sshd

service:

name: sshd

state: restarted

**🧪 Sample Inventory – inventories/production/hosts.ini**

ini

CopyEdit

[webservers]

web1.example.com

web2.example.com

[dbservers]

db1.example.com

db2.example.com

**🛡️ Additions for Production Use**

* **Scheduled Runs** via AAP job templates or cron job in AWX/Controller.
* **Reporting** using Ansible callback plugins (e.g., logstash, email, Splunk).
* **Patch Verification** using ansible.builtin.package\_facts.
* **Error Handling**: Register & check output of each patch task.
* **Rollback Plan**: Maintain backups, snapshots (e.g., via LVM, cloud snapshots).

**🧩 Optional Enhancements**

* Integrate with Red Hat Satellite or SUSE Manager.
* Use Red Hat Insights / OVAL definitions to target CVEs.
* Add tags for --tags "patch", --tags "precheck".
* Slack/Teams webhook notification post-patch.

**🔧 Use Case: Deploy and manage a simple Dockerized application using Ansible**

**Directory Structure**

bash

CopyEdit

project/

├── playbook.yml

├── files/

│ └── app.py

├── templates/

│ └── Dockerfile.j2

│ └── docker-compose.yml.j2

**📜 playbook.yml**

yaml

CopyEdit

---

- name: Docker Container Lifecycle Management

hosts: docker\_nodes

become: true

vars:

app\_name: "flaskapp"

image\_name: "flaskapp\_image"

container\_name: "flaskapp\_container"

app\_port: 5000

compose\_project\_name: "flaskproject"

tasks:

- name: Install required packages

apt:

name:

- apt-transport-https

- ca-certificates

- curl

- software-properties-common

update\_cache: true

state: present

- name: Add Docker GPG key

apt\_key:

url: https://download.docker.com/linux/ubuntu/gpg

state: present

- name: Add Docker repository

apt\_repository:

repo: deb https://download.docker.com/linux/ubuntu focal stable

state: present

- name: Install Docker Engine and Compose

apt:

name:

- docker-ce

- docker-ce-cli

- containerd.io

- docker-compose-plugin

state: present

update\_cache: true

- name: Ensure Docker is running

service:

name: docker

state: started

enabled: true

- name: Create app directory

file:

path: /opt/{{ app\_name }}

state: directory

mode: '0755'

- name: Copy application file

copy:

src: files/app.py

dest: /opt/{{ app\_name }}/app.py

- name: Copy Dockerfile template

template:

src: templates/Dockerfile.j2

dest: /opt/{{ app\_name }}/Dockerfile

- name: Build Docker image

community.docker.docker\_image:

path: /opt/{{ app\_name }}

name: "{{ image\_name }}"

tag: latest

source: build

- name: Copy Docker Compose file

template:

src: templates/docker-compose.yml.j2

dest: /opt/{{ app\_name }}/docker-compose.yml

- name: Deploy container using Docker Compose

community.docker.docker\_compose:

project\_src: /opt/{{ app\_name }}

restarted: true

remove\_orphans: true

**🐳 templates/Dockerfile.j2**

Dockerfile

CopyEdit

FROM python:3.9-slim

WORKDIR /app

COPY app.py .

RUN pip install flask

EXPOSE {{ app\_port }}

CMD ["python", "app.py"]

**🔧 files/app.py**

python

CopyEdit

from flask import Flask

app = Flask(\_\_name\_\_)

@app.route("/")

def hello():

return "Hello from Flask in Docker!"

if \_\_name\_\_ == "\_\_main\_\_":

app.run(host='0.0.0.0', port=5000)

**🔧 templates/docker-compose.yml.j2**

yaml

CopyEdit

version: "3"

services:

web:

image: {{ image\_name }}:latest

container\_name: {{ container\_name }}

ports:

- "{{ app\_port }}:{{ app\_port }}"

restart: always

**🧪 Test the Playbook**

**Inventory (inventory.ini):**

ini

CopyEdit

[docker\_nodes]

192.168.56.10 ansible\_user=ubuntu ansible\_ssh\_private\_key\_file=~/.ssh/id\_rsa

**Run it:**

bash

CopyEdit

ansible-playbook -i inventory.ini playbook.yml

**🔧 Real-World Use Case: Multi-Cloud Infrastructure Management with Ansible**

**✅ Overview**

Modern enterprises increasingly adopt a **multi-cloud strategy**—leveraging the strengths of AWS, Azure, and GCP simultaneously. This approach helps avoid vendor lock-in, improves resilience, and optimizes workloads. However, managing infrastructure consistently across these platforms can be a challenge.

Ansible excels here by providing an **agentless**, **declarative**, and **extensible** way to manage infrastructure **uniformly** using a single language: **YAML**.

**📌 Use Case Objective**

**"Provision and manage infrastructure across AWS, Azure, and GCP using a single playbook to streamline multi-cloud operations."**

This includes:

* Provisioning compute, networking, and storage resources
* Ensuring consistent tagging and security policies
* Handling lifecycle management: create, update, destroy
* Cross-cloud orchestration of application deployments

**🏗️ Architecture Diagram (Conceptual)**

lua

CopyEdit

+-------------------------------+

| Ansible Automation Platform |

| (Controller, EE) |

+---------------+---------------+

|

+-----------------------------+-----------------------------+

| | |

+----------------+ +----------------+ +----------------+

| AWS Cloud | | Azure Cloud | | GCP Cloud |

| - EC2 | | - VMs | | - Compute Eng. |

| - VPC/Subnets | | - VNET/Subnets | | - VPC/Subnets |

| - IAM, Tags | | - NSG, Tags | | - IAM, Labels |

+----------------+ +----------------+ +----------------+

**🛠️ Key Ansible Components Used**

| **Cloud** | **Modules/Collections** |
| --- | --- |
| AWS | amazon.aws |
| Azure | azure.azcollection |
| GCP | google.cloud |

All major cloud platforms have certified Ansible collections that provide extensive support for infrastructure resources.

**🎯 Example Use Case: Provision Equivalent VMs on AWS, Azure, and GCP**

**💡 Goal:**

* Spin up a **web server** instance in **all 3 clouds**
* Ensure they use the same **tags/labels**, **instance type**, **region**, and **security settings**
* Use a **single Ansible playbook** or **dynamic include\_tasks** to orchestrate

**📁 Folder Structure (Recommended)**

python

CopyEdit

multi-cloud/

│

├── inventories/

│ ├── aws/

│ ├── azure/

│ └── gcp/

│

├── playbooks/

│ ├── provision.yml

│ ├── aws-provision.yml

│ ├── azure-provision.yml

│ └── gcp-provision.yml

│

├── vars/

│ ├── aws.yml

│ ├── azure.yml

│ └── gcp.yml

│

└── ansible.cfg

**📘 Sample provision.yml Playbook (Main Orchestrator)**

yaml

CopyEdit

- name: Multi-Cloud Provisioning Playbook

hosts: localhost

gather\_facts: no

vars:

cloud\_target: "{{ target | default('all') }}" # target = aws/azure/gcp/all

tasks:

- name: Include AWS provisioning

include\_tasks: aws-provision.yml

when: cloud\_target in ['aws', 'all']

- name: Include Azure provisioning

include\_tasks: azure-provision.yml

when: cloud\_target in ['azure', 'all']

- name: Include GCP provisioning

include\_tasks: gcp-provision.yml

when: cloud\_target in ['gcp', 'all']

**📘 Sample aws-provision.yml**

yaml

CopyEdit

- name: Launch EC2 instance on AWS

amazon.aws.ec2\_instance:

name: "web-server-aws"

key\_name: "{{ aws\_keypair }}"

instance\_type: "{{ instance\_type }}"

image\_id: "{{ aws\_ami }}"

vpc\_subnet\_id: "{{ aws\_subnet\_id }}"

security\_group: "{{ aws\_sg }}"

tags:

Project: "multi-cloud"

Environment: "dev"

wait: yes

(Similar logic applies to azure-provision.yml and gcp-provision.yml using respective modules.)

**🔒 Secrets and Authentication**

Use **Ansible Vault**, **Credential Store in AAP**, or **environment variables** for:

* AWS Access Key & Secret
* Azure Client ID, Secret, Tenant, Subscription
* GCP Service Account JSON key

In AAP, store them as **Machine or Cloud credentials**.

**⚙️ Dynamic Inventory (Optional for Post-Provisioning)**

Each cloud provider supports **dynamic inventories**, useful when managing the provisioned VMs afterward for configuration or deployments.

* amazon.aws.aws\_ec2
* azure.azcollection.azure\_rm
* google.cloud.gcp\_compute\_inventory

**📈 Benefits of This Approach**

| **Benefit** | **Description** |
| --- | --- |
| **Consistency** | Same structure and practices across cloud providers |
| **Automation** | End-to-end infra provisioning and app deployment |
| **Scalability** | Easily scale to more regions or cloud accounts |
| **Auditability** | All actions logged via Ansible Tower/AAP |
| **Cost Control** | Tear down environments cleanly with Ansible tasks |

**🧠 Real-World Scenario**

**Scenario: Multi-Cloud Disaster Recovery Testing**

An enterprise deploys its primary app in AWS but wants to simulate failover to Azure and GCP for DR. Ansible playbooks create identical VMs, load balancers, and databases in all 3 clouds. DNS routing and failover scripts are also handled by Ansible workflows in AAP.

**⚠️ Challenges & Best Practices**

| **Challenge** | **Mitigation** |
| --- | --- |
| Different APIs & Resource Names | Abstract with variables and templates |
| Credential Management | Use AAP Credential store or Ansible Vault |
| Network Config Differences | Modular playbooks per provider |
| Cost & Quotas | Monitor usage; apply tags for billing |
| Idempotency | Ensure playbooks are written with idempotent modules |

**🔁 Extensions**

* Use **Ansible Controller (AAP)** to orchestrate workflows like:
  + Provision > Configure > Deploy App > Validate
  + Multi-tenancy and RBAC per cloud team
* Add **Terraform** under community.general.terraform for hybrid provisioning
* Integrate with **CI/CD pipelines** for GitOps-style deployments

**🧩 Use Case: Automating Database Backup and Restore for MySQL and PostgreSQL**

**🎯 Objective:**

Automate the process of:

* Taking periodic database backups (daily/hourly as per SLA)
* Storing the backups securely (locally or on cloud storage)
* Providing restore capability with version control
* Notification on success/failure (email, Slack, etc.)
* Ensuring backups are encrypted and compressed

**✅ Components Involved**

| **Component** | **Purpose** |
| --- | --- |
| **Ansible** | Orchestration and automation of backup tasks |
| **MySQL/PostgreSQL** | Target databases |
| **cron** | Job scheduling |
| **Shell Scripts** | Command execution and backup logic |
| **GPG/ZIP** | Encryption and compression |
| **S3 / Blob Storage** | Off-site storage (optional) |
| **Monitoring Tool** | For status reporting (e.g., Prometheus, Alertmanager) |

**🏗️ Architecture Diagram (Simplified)**

sql

CopyEdit

+----------------------+ +--------------------+

| Ansible Controller |-----> | Target DB Servers |

+----------------------+ +--------------------+

|

v

+---------------------+

| Local/Cloud Backup |

| Repository |

+---------------------+

|

v

+--------------------+

| Restore Workflow |

+--------------------+

**📜 Playbook Structure (Ansible)**

yaml

CopyEdit

# db\_backup\_restore.yml

- name: Database Backup and Restore Automation

hosts: db\_servers

become: yes

vars:

db\_type: "mysql" # or "postgres"

backup\_dir: "/opt/db\_backups"

s3\_bucket: "s3://company-database-backups"

encrypt\_with\_gpg: true

tasks:

- import\_tasks: tasks/mysql\_backup.yml

when: db\_type == "mysql"

- import\_tasks: tasks/postgres\_backup.yml

when: db\_type == "postgres"

**🔄 MySQL Backup Task (mysql\_backup.yml)**

yaml

CopyEdit

- name: Ensure backup directory exists

file:

path: "{{ backup\_dir }}"

state: directory

mode: '0755'

- name: Perform MySQL dump

shell: |

mysqldump -u root -p'{{ mysql\_root\_password }}' --all-databases \

| gzip > {{ backup\_dir }}/mysql\_backup\_{{ ansible\_date\_time.date }}.sql.gz

args:

creates: "{{ backup\_dir }}/mysql\_backup\_{{ ansible\_date\_time.date }}.sql.gz"

- name: Encrypt backup (optional)

shell: |

gpg --batch --yes -c --passphrase '{{ gpg\_pass }}' \

{{ backup\_dir }}/mysql\_backup\_{{ ansible\_date\_time.date }}.sql.gz

when: encrypt\_with\_gpg

- name: Upload to S3

aws\_s3:

bucket: "{{ s3\_bucket }}"

object: "mysql\_backup\_{{ ansible\_date\_time.date }}.sql.gz.gpg"

src: "{{ backup\_dir }}/mysql\_backup\_{{ ansible\_date\_time.date }}.sql.gz.gpg"

mode: put

when: encrypt\_with\_gpg

**🔁 PostgreSQL Backup Task (postgres\_backup.yml)**

yaml

CopyEdit

- name: Dump PostgreSQL database

become\_user: postgres

shell: |

pg\_dumpall | gzip > {{ backup\_dir }}/pg\_backup\_{{ ansible\_date\_time.date }}.sql.gz

- name: Encrypt and Upload (same as MySQL)

...

**🔄 Restore Task (on-demand)**

yaml

CopyEdit

- name: Decrypt backup file

shell: |

gpg --batch --yes --passphrase '{{ gpg\_pass }}' \

-o /tmp/db\_restore.sql.gz -d {{ backup\_file }}

- name: Decompress

shell: gunzip /tmp/db\_restore.sql.gz

- name: Restore MySQL DB

shell: mysql -u root -p'{{ mysql\_root\_password }}' < /tmp/db\_restore.sql

when: db\_type == "mysql"

**⏰ Scheduling with Cron (via Ansible)**

yaml

CopyEdit

- name: Schedule daily backup job at 2 AM

cron:

name: "Daily DB Backup"

minute: "0"

hour: "2"

job: "ansible-playbook /opt/ansible/db\_backup\_restore.yml"

**📣 Notifications (Email/Slack)**

Add a task to notify via email or webhook after job execution:

yaml

CopyEdit

- name: Notify via Slack

uri:

url: "https://hooks.slack.com/services/XXXX/XXXX/XXXX"

method: POST

body: '{"text":"Database backup completed successfully!"}'

headers:

Content-Type: "application/json"

**🔐 Security Best Practices**

* Rotate backup credentials (DB, GPG, Cloud)
* Encrypt backups with GPG or Vault integration
* Set TTL on backups or use lifecycle rules in cloud buckets
* Use IAM roles and minimal permissions for S3 uploads

**🔍 Real-World Scenarios**

| **Scenario** | **Solution** |
| --- | --- |
| Database corrupted due to accidental delete | Restore from latest backup using restore task |
| Compliance needs (audit logs) | Store daily encrypted backups with metadata |
| Cloud migration | Backup DB → upload to cloud → restore in target cloud region |
| Periodic DR drills | Restore to test environments with versioned backups |

**🧪 Testing Strategy**

1. Validate backup integrity: Periodically restore to a test DB and verify data
2. Run Ansible playbooks in check mode (--check) before production run
3. Verify permissions of backup directories and files
4. Monitor cron logs and Ansible logs (/var/log/ansible.log)