**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