Red Hat
CONSULTING

Red Hat

Duration:
Request demo lab: ~20 min
Lab: ~30 min
Request bonus lab: ~40 min
Bonus lab: ~15 min

# Managing the AAP Platform with Ansible (Config-as-Code)

In this lab, we'll dive into how to manage Red Hat Ansible Automation Platform (AAP) controller configurations using Configuration-as-Code (CaC) principles. CaC plays a key role in keeping your automation environment consistent, easy to track, and free from the kinds of manual errors that can slow things down. By shifting to a code-driven approach, your configurations become repeatable, scalable, and much easier to maintain over time.

We'll start with the basics - like setting up organizations - and gradually move into more advanced topics such as inventories, credentials, projects, and job templates, ultimately configuring AAP to run the Configuration-as-Code (CaC) playbooks we've created.

To wrap things up, we'll look at how to export specific configurations from AAP 2.4 in a format that's ready to be used in AAP 2.5 - something especially useful for migrations or upgrades. Throughout the lab, you'll get hands-on with practical CaC strategies that reflect the kind of work consultants do in the field, helping you build a solid foundation for managing AAP at scale.

## Deploy and prepare lab environment

| | |
|---|---|
| 1. Log in to https://catalog.demo.redhat.com | |

| | |
|---|---|
| 2. In the catalog find **Ansible Product Demos (AAP 2.5)** and order it | Activity: Practice / Enablement<br>Purpose: Practice a workshop<br>Project: PR-037001 |
| 3. Log in to AAP and confirm that only the **Default** organization exists | Access Management \| Organizations |
| 4. SSH to bastion host and confirm no collections are installed | `ansible-galaxy collection list` |
| 5. Download latest **ansible.platform** collection from Red Hat Automation Hub: https://console.redhat.com/ansible/automation-hub/repo/published/ansible/platform/<br><br>and latest **ansible.controller**: https://console.redhat.com/ansible/automation-hub/repo/published/ansible/controller/<br><br>These Red Hat certified collections are a dependency of the Red Hat validated **infra.aap_configuration** collection we are going to use later. | |
| 6. Transfer them to bastion host | `scp -P <bastion_ssh_port> ansible-platform-X.X.X.tar.gz lab-user@<bastion_host>:/home/lab-user/`<br><br>`scp -P <bastion_ssh_port> ansible-controller-X.X.X.tar.gz lab-user@<bastion_host>:/home/lab-user/` |
| 7. Install required collections | `ansible-galaxy collection install ansible-platform-X.X.X.tar.gz` |

| | |
|---|---|
| | *ansible-galaxy collection install ansible-controller-**X.X.X**.tar.gz*<br><br>*ansible-galaxy collection install infra.aap_configuration* |
| 8. Create public repository in your Github account and clone it to bastion host | *git clone https://github.com/**<your_repo>***<br><br>*cd **<your_repo>*** |

# Starting the configuration journey from managing organizations

Organizations are top-level objects in Ansible Automation Platform (AAP) used to logically group users, teams, projects, and other resources. Building a clear organizational structure is essential for managing access control and resource allocation effectively. This section walks you through creating organizations from scratch using **Configuration-as-Code (CaC)**, verifying changes, handling drift, and exporting configurations.

## Create AAP organizations with Red Hat validated tools

1. Review **organizations** role documentation - https://galaxy.ansible.com/ui/repo/published/infra/aap_configuration/content/role/controller_organizations/

   Note: **name** is the only required field to create an organization.

2. Build configuration with DevOps and QA organizations

```
mkdir vars
vi vars/organizations.yml
```

```
---
aap_organizations:
  - name: "DevOps"
    description: "Organization DevOps"
  - name: "QA"
    description: "Organization QA"
```

3. Review **dispatch** role documentation - https://galaxy.ansible.com/ui/repo/published/infra/aap_configuration/content/role/dispatch/

This role calls other roles from the collection, including **organizations** role.

4. Create a playbook to apply organization configurations created earlier

   The playbook should:
   a. Set connection variable for AAP instance
   b. Include files with configurations
   c. Call **dispatch** role that applies configurations

*vi cac.yml*

```
---
- name: Configure AAP Controller using dispatcher
  hosts: localhost
  gather_facts: false

  vars:
    aap_hostname: "<aap_hostname>"
    aap_username: "admin"
    aap_password: "<aap_password>"

  pre_tasks:
    - name: Include variables
      ansible.builtin.include_vars:
        dir: vars

  roles:
    - infra.aap_configuration.dispatch
```

5. Run the playbook to apply configuration

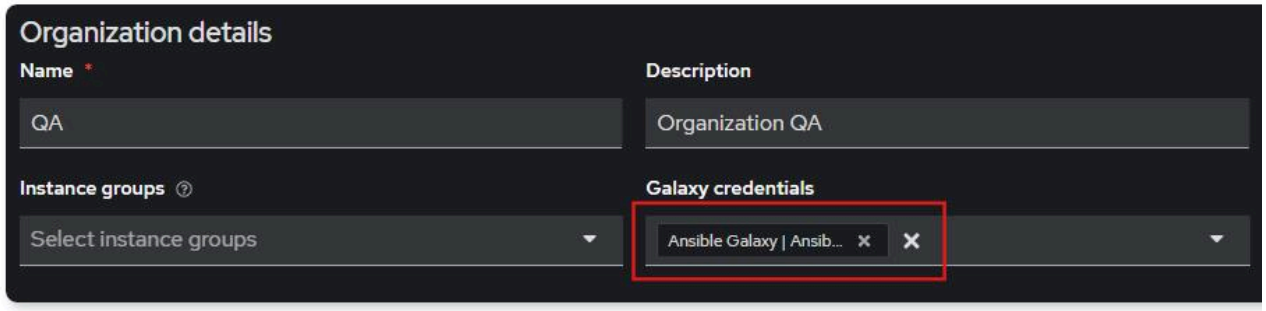
*ansible-playbook cac.yml*

```
TASK [infra.aap_configuration.gateway_organizations : Organizations | Wait for finish the configuration] ********************
FAILED - RETRYING: [localhost]: Organizations | Wait for finish the configuration (30 retries left).
changed: [localhost] => (item=Create/Update AAP Platform Organizations DevOps | Wait for finish the Organizations creation)
changed: [localhost] => (item=Create/Update AAP Platform Organizations QA | Wait for finish the Organizations creation)
```

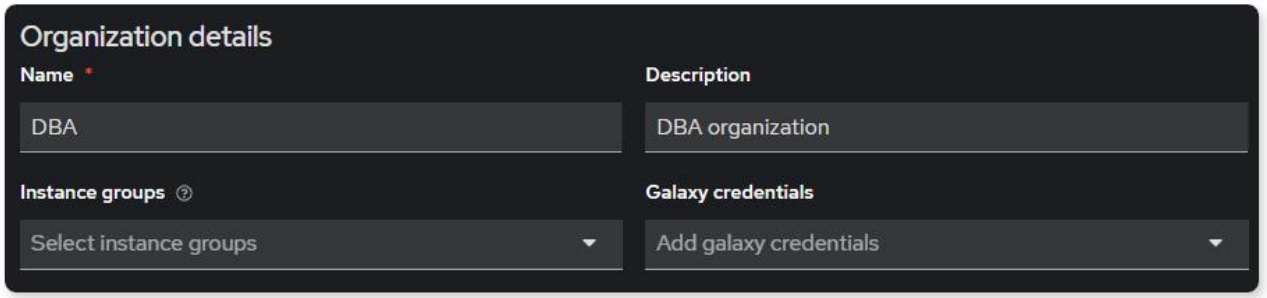
| | |
|---|---|
| 6. Log in to AAP and verify that DevOps and QA organizations have been created | |

## Introduce configuration drift

| | |
|---|---|
| 1. While logged in to AAP edit QA organization and add **Ansible Galaxy** credential | Access Management \| Organizations \| QA \| Galaxy credentials |




| | |
|---|---|
| 2. Create **DBA** organization | Access Management \| Organizations |




| | |
|---|---|
| 3. Run cac.yml playbook again<br><br>The playbook doesn't report any change | *ansible-playbook cac.yml* |

```
TASK [infra.aap_configuration.gateway_organizations : Organizations | Wait for finish the configuration] *************
FAILED - RETRYING: [localhost]: Organizations | Wait for finish the configuration (30 retries left).
ok: [localhost] => (item=Create/Update AAP Platform Organizations DevOps | Wait for finish the Organizations creation)
ok: [localhost] => (item=Create/Update AAP Platform Organizations QA | Wait for finish the Organizations creation)
```

There are two separate issues happening here.

First, the change related to the credential we added to the QA organization hasn't been picked up or fixed. This is because the configuration code we previously created doesn't include the non-required **galaxy_credentials** field, which defines the credentials assigned to an organization. To ensure our configurations are consistent and complete, we need to include all possible fields in the config-as-code - even the optional ones. This is especially important because the **infra.aap_configuration** collection doesn't enforce this.

Second, the additional organization we created hasn't been reported or deleted because such functionality doesn't currently exist in the **infra.aap_configuration** collection.

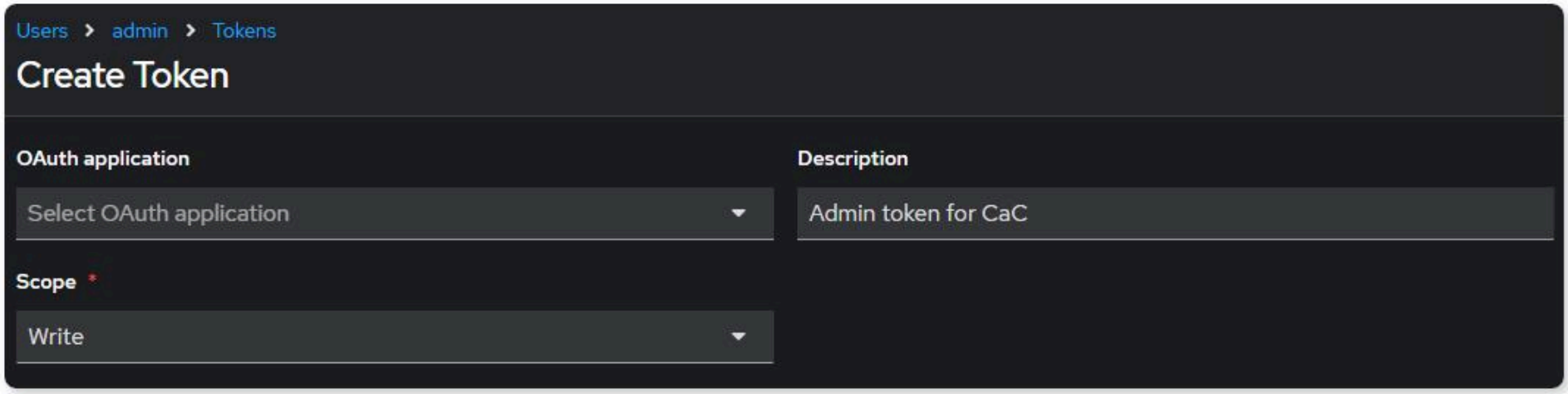
## Export configuration and manage drift

The second issue - detecting and optionally deleting organizations not defined in the code - is critical for achieving true end-to-end Configuration-as-Code (CaC) automation.

Naturally, CaC users expect that when they remove an object from the code, it should be automatically removed from the system during the next run.

To support this behavior, let's consider exploring the community-supported configify.aapconfig collection as a potential solution.

| | |
|---|---|
| 1. Install **configify.aapconfig** collection | `ansible-galaxy collection install configify.aapconfig` |
| 2. Review collection documentation - https://galaxy.ansible.com/ui/repo/published/configify/aapconfig/docs<br><br>In contrast to **infra.aap_configuration**, it enforces the inclusion of all fields to guarantee full coverage of the configuration. | |
| 3. Log in to AAP and generate an OAuth token for the admin account<br><br>This enhances security by avoiding basic authentication and prepares us to run Configuration-as-Code (CaC) automation directly within AAP - instead of from the command line - in the next steps of this lab. | admin \| User details \| Tokens \| Create Token |




4. Set AAP Credentials as Environment Variables

```
export CONTROLLER_HOST=<aap_hostname>
export CONTROLLER_OAUTH_TOKEN=<admin_token>
export GATEWAY_HOSTNAME=<aap_hostname>
export GATEWAY_API_TOKEN=<admin_token>
```

5. Create export playbook

```
vi get_objects.yml
```

```
---
- name: Run playbook to get objects
  import_playbook: configify.aapconfig.aap_audit_all.yml
```

6. Export organizations configuration

   To export only organizations we can use **export_organizations** tag.

   Note that the output includes all fields, even those that are empty. The QA organization is shown with the Ansible Galaxy credential we

```
ansible-playbook get_objects.yml --tags
export_organizations
```

added earlier, and the DBA organization created in the previous steps is also listed.

```
TASK [ORGANIZATIONS - Show all organizations (formatted)] ************************************************************************************
ok: [localhost] => {
    "controller_objects_organizations": [
        "{'name': 'DBA', 'descr': 'DBA organization', 'creds': []}",
        "{'name': 'Default', 'descr': 'The default organization for Ansible Automation Platform', 'creds': ['Ansible Galaxy', 'Automation Hub']}",
        "{'name': 'DevOps', 'descr': 'Organization DevOps', 'creds': []}",
        "{'name': 'QA', 'descr': 'Organization QA', 'creds': ['Ansible Galaxy']}"
    ]
}
```

7. Modify the exported data to **remove** the unwanted DBA organization and save it in a file

   Note: Remove double quotes.

```
vi objects_organizations
```

```
controller_objects_organizations: [
  {'name': 'Default', 'descr': 'The default organization for Ansible Automation Platform',
   'creds': ['Ansible Galaxy', 'Automation Hub']},
  {'name': 'DevOps', 'descr': 'Organization DevOps', 'creds': []},
  {'name': 'QA', 'descr': 'Organization QA', 'creds': ['Ansible Galaxy']}
]
```

8. Create playbook to apply this configuration

```
vi cac_configify.yml
```

```
---
- name: AAP configuration
  hosts: localhost
  gather_facts: false

- name: Apply controller configuration
  import_playbook: configify.aapconfig.aap_configure.yml
```

| | |
|---|---|
| 9. Run the playbook<br><br>You'll get a **"rogue organization"** warning for DBA - it exists in AAP but not in your code. | *ansible-playbook cac_configify.yml -e @objects_organizations --tags controller_config_organizations* |

```
TASK [configify.aapconfig.controller_config : ORGANIZATIONS - Notify on rogue organizations] **
changed: [localhost] => (item= | organization: DBA) => {
    "msg": "Shouldn't be there"
}
skipping: [localhost] => (item= | organization: Default)
skipping: [localhost] => (item= | organization: DevOps)
skipping: [localhost] => (item= | organization: QA)
```

| | |
|---|---|
| 10. Enforce code as source of truth<br><br>Set **delete_objects** variable to **true** | *ansible-playbook cac_configify.yml -e @objects_organizations -e delete_objects=true --tags controller_config_organizations* |
| 11. Log in to AAP and verify that DBA organization has been removed | |

# Expand configuration to Inventories

Inventories are another fundamental component of AAP, serving as the foundation for running playbooks. Let's now extend our Configuration-as-Code (CaC) approach to include inventory configuration.

1. Review documentation for the following **infra.aap_configuration** roles:

   controller_inventories - https://galaxy.ansible.com/ui/repo/published/infra/aap_configuration/content/role/controller_inventories/

   controller_host_groups - https://galaxy.ansible.com/ui/repo/published/infra/aap_configuration/content/role/controller_host_groups/

   controller_hosts - https://galaxy.ansible.com/ui/repo/published/infra/aap_configuration/content/role/controller_hosts/

2. Add configuration for inventory named **Static Inventory**

```
vi vars/inventories.yml
```

```
---
controller_inventories:
  - name: Static Inventory
    organization: Default
```

3. Add configuration for groups

```
vi vars/groups.yml
```

```
---
```

```
controller_groups:
- name: Static GroupA
  inventory: Static Inventory
  hosts:
   - HostA1
   - HostA2
   - HostA3
  children:
   - Static GroupB

- name: Static GroupB
  inventory: Static Inventory
  hosts:
   - HostB1
   - HostB2
   - HostB3
```

4. Add configuration for hosts

```
vi vars/hosts.yml
```

```
---
controller_hosts:
  - name: HostA1
    inventory: Static Inventory
    variables:
      alt_name: Host A1
  - name: HostA2
    inventory: Static Inventory
    variables:
      alt_name: Host A2
  - name: HostA3
    inventory: Static Inventory
    variables:
      alt_name: Host A3
```

```
- name: HostB1
  inventory: Static Inventory
  variables:
    alt_name: Host B1
- name: HostB2
  inventory: Static Inventory
  variables:
    alt_name: Host B2
- name: HostB3
  inventory: Static Inventory
  variables:
    alt_name: Host B3
```

5. Apply configurations

   Most configurations should now be created, however **Static GroupA** is missing and there was an error message during group creation.

```
ansible-playbook cac.yml
```

```
TASK [infra.aap_configuration.controller_host_groups : Managing Controller Groups | Wait for finish the Controller Groups management] **********************
FAILED - RETRYING: [localhost]: Managing Controller Groups | Wait for finish the Controller Groups management (30 retries left).
FAILED - RETRYING: [localhost]: Managing Controller Groups | Wait for finish the Controller Groups management (29 retries left).
FAILED - RETRYING: [localhost]: Managing Controller Groups | Wait for finish the Controller Groups management (28 retries left).
failed: [localhost] (item=Create/Update Controller Group Static GroupA | Wait for finish the Controller Group creation) => {"__group_job_async_results_item"
: {"__controller_groups_item": {"children": ["Static GroupB"], "hosts": ["HostA1", "HostA2", "HostA3"], "inventory": "Static Inventory", "name": "Static Gro
upA"}, "ansible_job_id": "j895986044625.6628", "ansible_loop_var": "__controller_groups_item", "changed": false, "failed": 0, "finished": 0, "results_file":
 "/home/lab-user/.ansible_async/j895986044625.6628", "started": 1}, "ansible_job_id": "j895986044625.6628", "ansible_loop_var": "__group_job_async_results_i
tem", "attempts": 4, "changed": false, "finished": 1, "msg": "Could not find groups with name Static GroupB", "results_file": "/home/lab-user/.ansible_async
/j895986044625.6628", "started": 1, "stderr": "", "stderr_lines": [], "stdout": "", "stdout_lines": []}
FAILED - RETRYING: [localhost]: Managing Controller Groups | Wait for finish the Controller Groups management (30 retries left).
changed: [localhost] => (item=Create/Update Controller Group Static GroupB | Wait for finish the Controller Group creation)
```

6. This happened because in the group configuration **Static GroupA** is listed before its child **Static GroupB**.

   Let's fix the configuration and re-run the playbook.

```
vi vars/groups.yml
```

```
ansible-playbook cac.yml
```

So far, we've learned that with **infra.aap_configuration**, the order of configuration items matters - and it's up to the user to manage that correctly.

Now, imagine a customer with hundreds of inventories and thousands of groups and hosts. Even if your current environment isn't that large yet, it's always wise to consider how today's work will scale by 10x or more.

The structure used to define groups does provide a clear view of host membership and child groups on a per-group basis. However, in large-scale deployments, the overall structure of each inventory - especially the full group hierarchy - may still not be immediately apparent.

Let's explore whether **configify.aapconfig** offers a better approach for handling this complexity.

1. Export inventory configuration using **configify.aapconfig** collection

   To export only inventories we can use **export_inventories** tag.

```
ansible-playbook get_objects.yml --tags export_inventories
```

```
TASK [INVENTORIES - Show static inventories (formatted)] ***********************************************************************************************
ok: [localhost] => {
    "controller_objects_inventories_static": [
        "{'name': 'Demo Inventory', 'description': '', 'org': 'Default', 'host_filter': '', 'prevent_fallback': False, 'variables': {}, 'hosts': [{'name': '
localhost', 'description': '', 'variables': {'ansible_connection': 'local', 'ansible_python_interpreter': '{{ ansible_playbook_python }}'}}], 'groups': []}"
,
        "{'name': 'Static Inventory', 'description': '', 'org': 'Default', 'host_filter': '', 'prevent_fallback': False, 'variables': {}, 'hosts': [], 'grou
ps': [{'name': 'Static GroupA', 'description': '', 'variables': {}, 'subgroups': ['Static GroupB'], 'hosts': [{'name': 'HostA1', 'description': '', 'variabl
es': {'alt_name': 'Host A1'}}, {'name': 'HostA2', 'description': '', 'variables': {'alt_name': 'Host A2'}}, {'name': 'HostA3', 'description': '', 'variables
': {'alt_name': 'Host A3'}}]}, {'name': 'Static GroupB', 'description': '', 'variables': {}, 'subgroups': [], 'hosts': [{'name': 'HostB1', 'description': ''
, 'variables': {'alt_name': 'Host B1'}}, {'name': 'HostB2', 'description': '', 'variables': {'alt_name': 'Host B2'}}, {'name': 'HostB3', 'description': '',
'variables': {'alt_name': 'Host B3'}}]}]}"
    ]
}
```

2. In the output the structure of **Static Inventory** already includes groups and hosts in hierarchical format.

   Let's add indentations to be able to see that better.

```
{'name': 'Static Inventory', 'description': '', 'org': 'Default', 'host_filter': '',
 'prevent_fallback': False, 'variables': {},
```

```
    'hosts': [],
    'groups': [{'name': 'Static GroupA', 'description': '', 'variables': {},
                'subgroups': ['Static GroupB'],
                'hosts': [{'name': 'HostA1', 'description': '', 'variables': {'alt_name': 'Host A1'}},
                          {'name': 'HostA2', 'description': '', 'variables': {'alt_name': 'Host A2'}},
                          {'name': 'HostA3', 'description': '', 'variables': {'alt_name': 'Host A3'}}]},

               {'name': 'Static GroupB', 'description': '', 'variables': {},
                'subgroups': [],
                'hosts': [{'name': 'HostB1', 'description': '', 'variables': {'alt_name': 'Host B1'}},
                          {'name': 'HostB2', 'description': '', 'variables': {'alt_name': 'Host B2'}},
                          {'name': 'HostB3', 'description': '', 'variables': {'alt_name': 'Host B3'}}]}]}
```

# Expand Configuration to Prepare for Running CaC Playbooks in AAP

To prepare Ansible Automation Platform (AAP) for running our Configuration-as-Code (CaC) playbooks, we'll expand our automation to cover **credentials**, **projects**, and **job templates**. This sets the foundation for a reproducible, code-defined automation environment.

After reviewing the documentation for the corresponding roles in the **infra.aap_configuration** collection, we can build the following variable files.

| | |
|---|---|
| 3. Add AAP and Git credentials | *vi vars/credentials.yml* |

```
---
controller_credentials:
  - name: "AAP Controller Admin"
    description: "Credential to access AAP controller"
    organization: "Default"
    credential_type: "Red Hat Ansible Automation Platform"
    inputs:
      host: "<aap_hostname>"
      oauth_token: "<admin_token>"
```

| | |
|---|---|
| 4. Add empty inventory to the configuration | *vi vars/inventories.yml* |

```
  - name: Empty Inventory
    description: This is an empty inventory that assumes localhost
    organization: Default
```

| | |
|---|---|
| 5. Add project configuration<br><br>**`update_project: true`** - syncs the project right after creation.<br>**`scm_update_on_launch: true`** - syncs the project every time it's used in a job template. | *vi vars/projects.yml* |

```
---
controller_projects:
  - name: "CaC Project"
    description: "Project with config-as-code"
    organization: "Default"
    scm_type: "git"
    scm_url: "<your_repo>"
    update_project: true
    scm_update_on_launch: true
```

6. Add job template

   This configuration brings together everything we've built so far - **credentials**, **project**, and **inventory** - into a job template that runs **cac_configify.yml** playbook created earlier.

   **Note:** The job template configuration becomes quite long when all available fields are specified. This is **necessary if you want to fully manage and track every setting** through configuration-as-code. It ensures that no default or UI-only settings drift out of sync with your desired state.

```
vi vars/job_templates.yml
```

```
---
controller_templates:
  - name: "CaC Job"
    description: "Run playbook to apply configuration as code"
    job_type: "run"
    inventory: "Empty Inventory"
    project: "CaC Project"
    playbook: "cac.yml"
    credential: "AAP Controller Admin"

    # Optional execution settings
    forks: 5
```

```
limit: ""
verbosity: 1
job_tags: ""
skip_tags: ""
timeout: 0
use_fact_cache: false

# Ask options
ask_credential_on_launch: false
ask_inventory_on_launch: false
ask_limit_on_launch: false
ask_scm_branch_on_launch: false
ask_tags_on_launch: false
ask_skip_tags_on_launch: false
ask_variables_on_launch: false
ask_verbosity_on_launch: false
ask_job_type_on_launch: false
ask_diff_mode_on_launch: false
ask_labels_on_launch: false

# SCM options
scm_branch: ""

# Execution
execution_environment: "Default execution environment"
instance_groups:
  - "default"
job_slice_count: 1
max_forks: 10

# Other settings
become_enabled: false
diff_mode: false
prevent_instance_group_fallback: false
```

```
# Variables passed to playbook
extra_vars:
  delete_objects: false

# Optional metadata
labels: []
survey_enabled: false
success_nodes: []
failure_nodes: []
error_nodes: []

# Webhooks (optional)
webhook_service: ""
webhook_credential: ""
webhook_url: ""

# Provisioning callbacks
host_config_key: ""
```

| | |
|---|---|
| 7. Push files to the repository | `git add *`<br>`git commit -m "Initial config-as-code"`<br>`git push` |
| 8. Apply configurations created | `ansible-playbook cac.yml` |
| 9. Verify configurations in AAP | Automation Execution \| Infrastructure \| Credentials<br>Automation Execution \| Infrastructure \| Inventories<br>Automation Execution \| Projects<br>Automation Execution \| Templates |

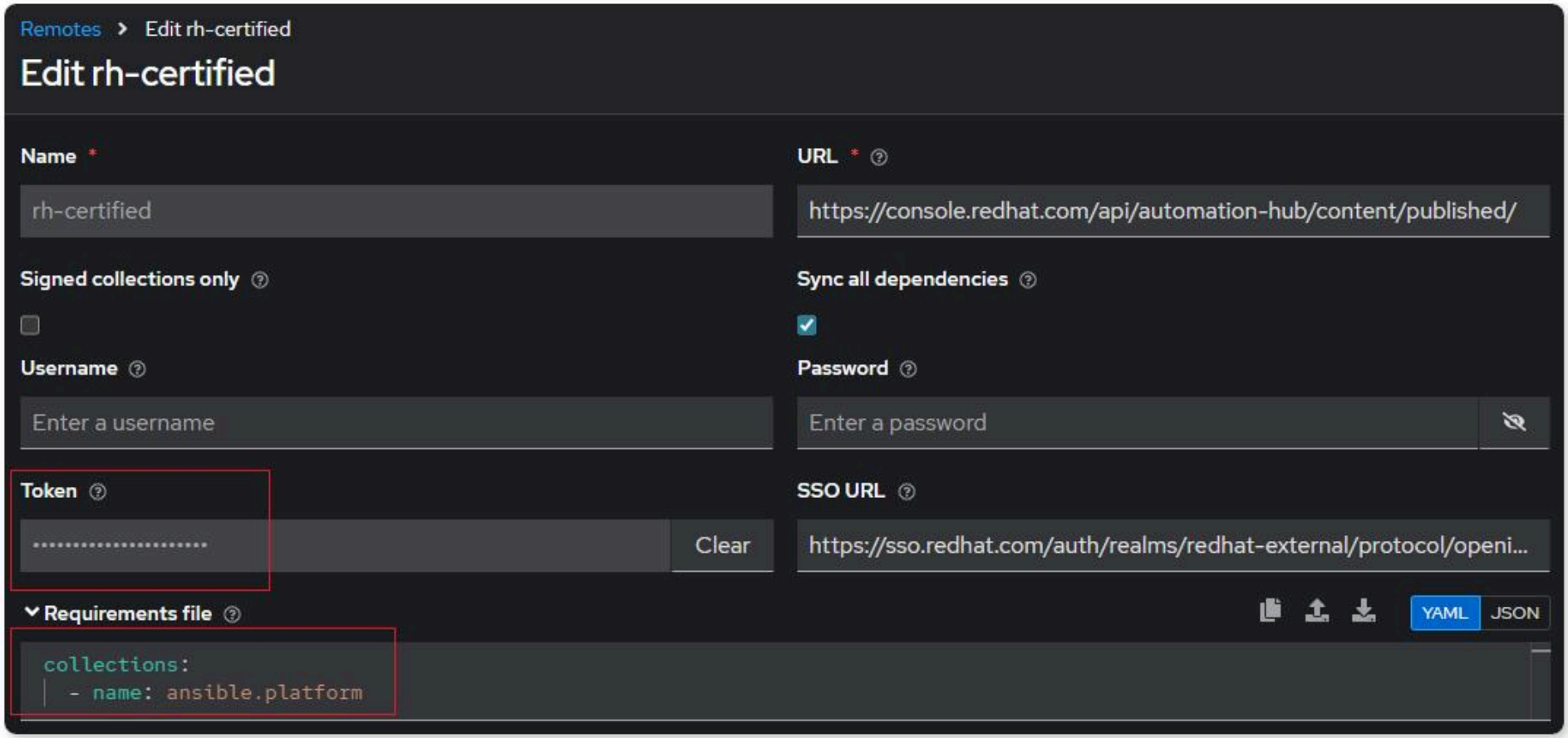
| | |
|---|---|
| 10. Launch job template<br><br>The job will fail with an error message about missing collection. | Automation Execution \| Templates \| CaC Job \| Launch |

```
ERROR! the role 'infra.aap_configuration.dispatch' was not found in
/runner/project/roles:/runner/requirements_roles:/home/runner/.ansible/roles:/usr/share/ansible/roles:/etc/ansible/
```

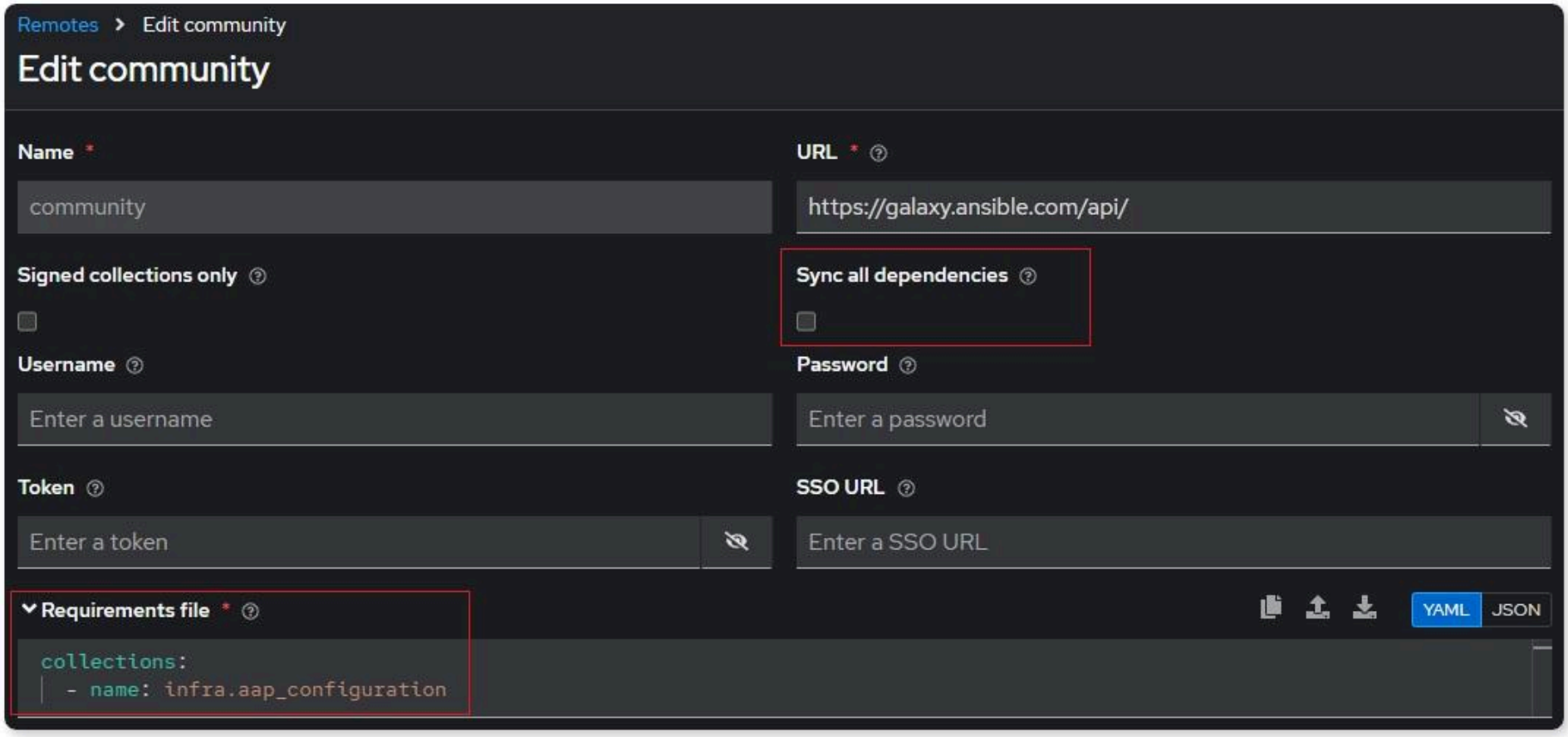
The error message is occurring because this AAP installation is missing the required collections - the same ones we previously installed on the bastion host. This can be confirmed by going to **Automation Content | Collections**.

To resolve this, we'll need to synchronize those collections into Automation Hub. While this process can be automated using Configuration-as-Code (CaC), we'll configure it manually for now to demonstrate how it's done through the GUI.

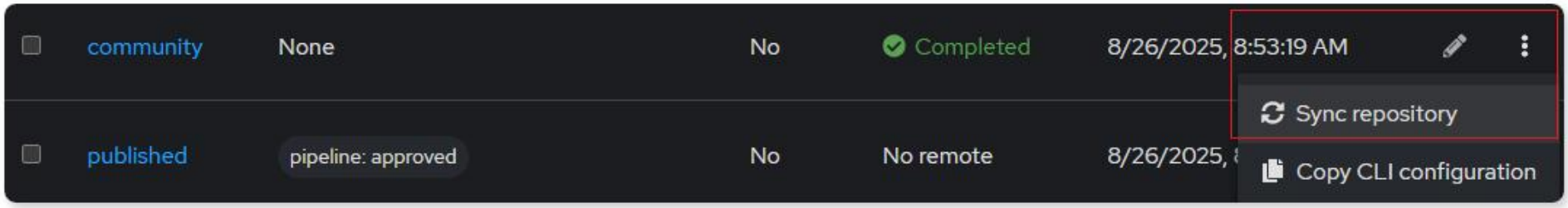
| | |
|---|---|
| 11. Log in to **Connect to Hub** and get **Offline token** - https://console.redhat.com/ansible/automation-hub/token | |
| 12. In AAP configure **rh-certified** with the token for synchronization and add **ansible.platform** to the list | Automation Content \| Remotes \| rh-certified |




| | |
|---|---|
| 13. Add **infra.aap_configuration** to community remote and uncheck **Sync all dependencies** option | Automation Content \| Remotes \| community |




| | |
|---|---|
| 14. Synchronize both repositories <u>starting from rh-certified</u><br><br>Confirm both collections have been added to the Hub - **Automation Content \| Collections** | Automation Content \| Repositories |




15. Finally, add the requirements file to the CaC repository

```
mkdir collections

vi collections/requirements.yml
```

```
---
collections:
  - name: infra.aap_configuration
```

16. Push changes into source control and run the job template again.

    There should be no errors this time.

17. Be sure to stop and delete your demo environment to avoid unnecessary charges

If you look through the job log, you'll notice that credentials are marked as changed every time the job runs. This behavior occurs because the **update_secrets** parameter is set to true by default, as documented in the **controller_credentials** role - https://galaxy.ansible.com/ui/repo/published/infra/aap_configuration/content/role/controller_credentials/. As a result, these tasks are not idempotent.

Since secrets in AAP are encrypted, Ansible has no way of determining whether the secret in the code matches the one stored in AAP. This means the user must choose between always applying the secret on each run, or only updating it when other parts of the credential change.

Additionally, in our example code, the token is stored in plain text in a public repository. Of course, in a real-world scenario, this would not be acceptable. For anything beyond this lab, you should use **Ansible Vault** or retrieve secrets securely from an external vault, such as **HashiCorp Vault**.

# Bonus Lab: Migrate configuration from AAP 2.4 to AAP 2.5

AAP 2.5 introduced significant changes across the platform. When it comes specifically to configuration management, there are three key changes to be aware of:

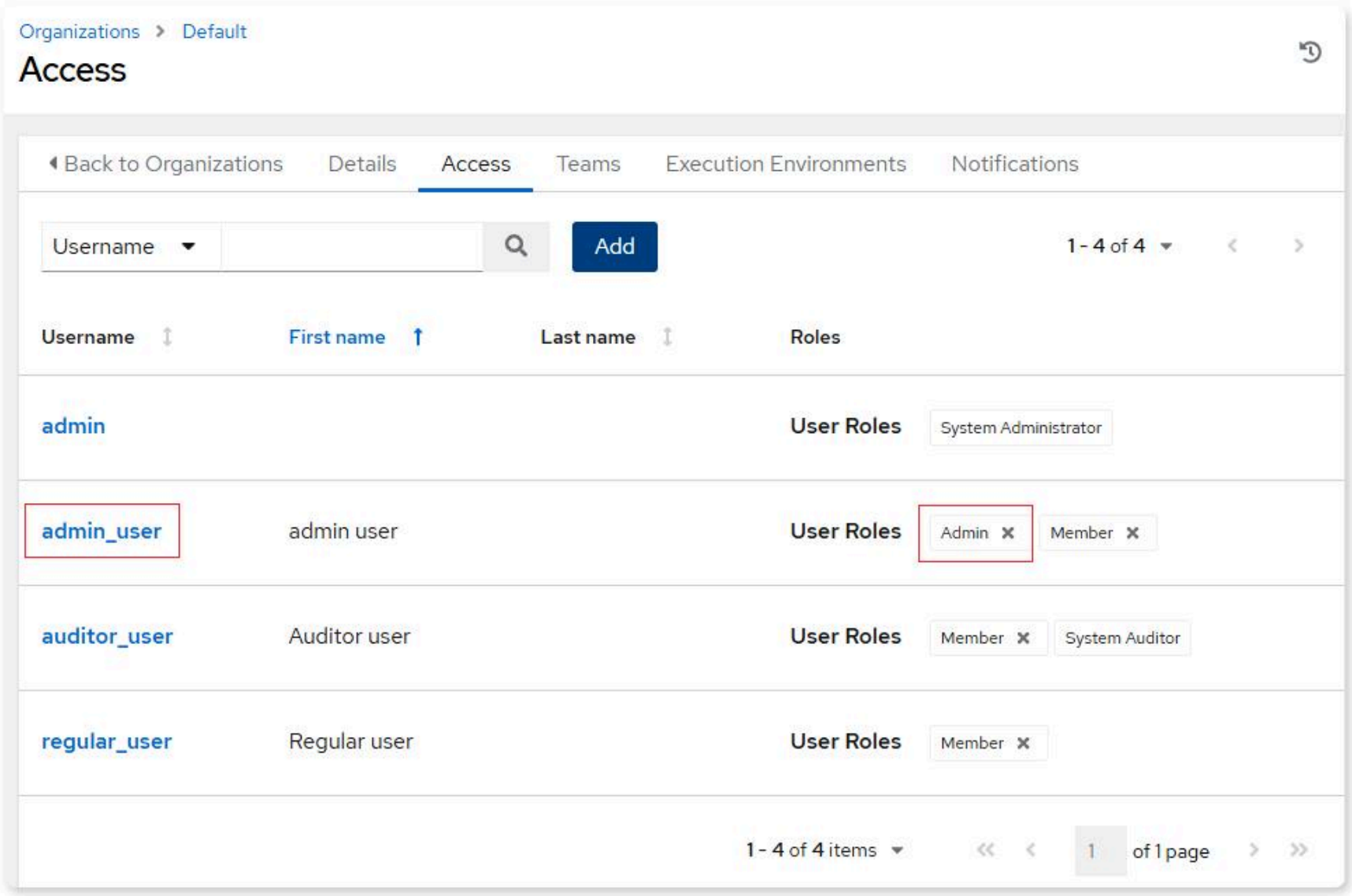
- The **auditor user** is no longer a user property - it's now handled as a standalone role.
- Roles like **Organization Member**, **Organization Admin**, and **Platform Auditor** have been moved under a new component called **gateway roles**.
- **Organization and team mappings** have been separated from authentication settings and are now managed as distinct configuration objects.

This lab shows how to prepare existing AAP 2.4 configurations for use in 2.5.

## Deploy and prepare lab environment

| | |
|---|---|
| 1. Log in to https://catalog.demo.redhat.com | |
| 2. In the catalog find **Migration AWX/Tower to AAP using CaC** and order it.<br><br>We will use AAP 2.4 that comes with this lab. | Activity: Practice / Enablement<br>Purpose: Practice a workshop<br>Project: PR-037001 |
| 3. Log in to AAP and create:<br><br>**regular_user** as **Normal User**<br>**admin_user** as **Normal User**<br>**auditor_user** as **System Auditor** | Access \| Users |

| | |
|---|---|
| 4. Make **admin_user** an administrator of **Default** organization | Access \| Organizations \| Default \| Access \| Add \| Users \| admin_user \| Admin |



| | |
|---|---|
| 5. While in AAP add LDAP configuration with organization and team mappings | Settings \| LDAP settings \| Default |

```
LDAP Server URI: ldap://example

LDAP Organization Map:
{
  "Org A": {
    "admins": [
      "CN=orgaadm,OU=Users,DC=example,DC=com"
    ],
    "remove_admins": true,
    "remove_users": true,
    "users": [
      "CN=orga,OU=Users,DC=example,DC=com"
    ]
  },
  "Org C": {
    "admins": [],
    "users": [
      "CN=orgc,OU=Users,DC=example,DC=com",
      "CN=orgc2,OU=Users,DC=example,DC=com"
    ]
  }
}

LDAP Team Map:
{
  "Team A": {
    "organization": "Org A",
    "remove": true,
    "users": [
      "CN=teama,OU=Users,DC=example,DC=com"
```

```
    ]
  },
  "Team C": {
    "organization": "Org C",
    "users": [
      "CN=teama,OU=Users,DC=example,DC=com",
      "CN=teama2,OU=Users,DC=example,DC=com"
    ]
  }
}
```

## Export configurations

During the preparation steps, we added configurations to AAP 2.4 that we expect to differ in AAP 2.5. Let's export them now - first in the 2.4 format, then in the 2.5 format - to compare the differences.

| | |
|---|---|
| 1. Copy SSH key for bastion host | `vi ~/.ssh/lab_key`<br><br>`chmod 600 ~/.ssh/lab_key`<br><br>`vi ~/.ssh/lab_key.pub`<br><br>`chmod 644 ~/.ssh/lab_key.pub` |
| 2. Transfer **ansible.platform** and **ansible.controller** collections we downloaded earlier to bastion host | `scp -i ~/.ssh/lab_key ansible-platform-`***X.X.X***`.tar.gz ec2-user@<`***bastion_host***`>:/home/ec2-user/`<br><br>`scp -i ~/.ssh/lab_key` |

| | |
|---|---|
| | *ansible-controller-**X.X.X**.tar.gz ec2-user@<**bastion_host**>:/home/ec2-user/* |
| 3. SSH to bastion host | *ssh ec2-user@<**bastion_host**> -i ~/.ssh/lab_key* |
| 4. Install required collections | *ansible-galaxy collection install ansible-platform-**X.X.X**.tar.gz*<br><br>*ansible-galaxy collection install ansible-controller-**X.X.X**.tar.gz*<br><br>*ansible-galaxy collection install configify.aapconfig* |
| 5. Log in to AAP and generate an OAuth token for the admin account | admin \| User details \| Tokens \| Create Token |
| 6. Set AAP Credentials as Environment Variables | *export CONTROLLER_HOST=**<aap_hostname>***<br>*export CONTROLLER_OAUTH_TOKEN=**<admin_token>*** |
| 7. Set API endpoint pattern to match AAP 2.4 | *export CONTROLLER_OPTIONAL_API_URLPATTERN_PREFIX='/api/'* |
| 8. Create export playbook | *vi get_objects.yml* |
| ```<br>---<br>- name: Run playbook to get objects<br>  import_playbook: configify.aapconfig.aap_audit_all.yml<br>``` | |
| 9. Export configurations for users, roles and LDAP | *ansible-playbook get_objects.yml --tags export_users,export_roles,export_settings* |

So far all the settings look the way they were created:

- **admin_user** is Organization **Admin** while the rest are Organization **Members**

```
controller_objects_roles: [
    {'user': 'admin_user', 'role': 'Organization Admin',
      'object_type': 'organization', 'object_name': 'Default'},

    {'user': 'admin_user', 'role': 'Organization Member',
      'object_type': 'organization', 'object_name': 'Default'},

    {'user': 'auditor_user', 'role': 'Organization Member',
      'object_type': 'organization', 'object_name': 'Default'},

    {'user': 'regular_user', 'role': 'Organization Member',
      'object_type': 'organization', 'object_name': 'Default'}
]
```

- **auditor_user** has **auditor field** set to true

```
controller_objects_users": [
    {'username': 'admin', 'first_name': '', 'last_name': '', 'email': 'admin@example.com',
      'superuser': True, 'auditor': False, 'pass': '$encrypted$'},

    {'username': 'admin_user', 'first_name': 'admin user', 'last_name': '', 'email': '',
      'superuser': False, 'auditor': False, 'pass': '$encrypted$'},

    {'username': 'auditor_user', 'first_name': 'Auditor user', 'last_name': '', 'email': '',
      'superuser': False, 'auditor': True, 'pass': '$encrypted$'},

    {'username': 'regular_user', 'first_name': 'Regular user', 'last_name': '', 'email': '',
      'superuser': False, 'auditor': False, 'pass': '$encrypted$'}
```

```
]
```

- **LDAP** configuration has three settings created - URI, Team and Org mappings.

```
controller_settings_ldap: {
    AUTH_LDAP_ORGANIZATION_MAP: {
        Org A: {
            admins: [
                CN=orgaadm,OU=Users,DC=example,DC=com
            ],
            remove_admins: true,
            remove_users: true,
            users: [
                CN=orga,OU=Users,DC=example,DC=com
            ]
        },
        Org C: {
            admins: [],
            users: [
                CN=orgc,OU=Users,DC=example,DC=com,
                CN=orgc2,OU=Users,DC=example,DC=com
            ]
        }
    },
    AUTH_LDAP_SERVER_URI: "ldap://example",
    AUTH_LDAP_TEAM_MAP: {
        Team A: {
            organization: "Org A",
            remove: true,
            users: [
                CN=teama,OU=Users,DC=example,DC=com
            ]
        },
        Team C: {
            organization: "Org C",
```

```
                users: [
                    CN=teama,OU=Users,DC=example,DC=com,
                    CN=teama2,OU=Users,DC=example,DC=com
                ]
            }
        }
    }
```

| | |
|---|---|
| 10. Now let's export these configurations in the format suitable for import into AAP 2.5<br><br>Set **format_for_25** variable to **true**. | *ansible-playbook get_objects.yml -e format_for_25=true --tags export_users,export_roles,export_settings* |

In the output, **controller_objects_users** remains unchanged. However, **Organization Admins** and **Members** are now included in a new variable called **gateway_objects_roles**. In AAP 2.5, these roles are managed separately from controller roles. Notably, **Platform Auditor** is now treated as a role, rather than being a user property flag:

```
gateway_objects_roles: [
  {'user': 'admin_user', 'role': 'Organization Admin',
   'object_type': 'organization', 'object_name': 'Default'},

  {'user': 'admin_user', 'role': 'Organization Member',
   'object_type': 'organization', 'object_name': 'Default'},

  {'user': 'auditor_user', 'role': 'Organization Member',
   'object_type': 'organization', 'object_name': 'Default'},

  {'user': 'regular_user', 'role': 'Organization Member',
   'object_type': 'organization', 'object_name': 'Default'},
```

```
  {'user': 'auditor_user', 'role': 'Platform Auditor',
   'object_type': 'platform', 'object_name': 'Platform'}
]
```

Additionally, LDAP configuration and mappings are split into two distinct variables, as they represent separate configurations in version 2.5:

```
controller_authentication: [
  {
    configuration: {
      CONNECTION_OPTIONS: {},
      GROUP_TYPE: MemberDNGroupType,
      GROUP_TYPE_PARAMS: {
        member_attr: member,
        name_attr: cn
      },
      SERVER_URI: [
        ldap://example
      ],
      START_TLS: false,
      USER_ATTR_MAP: {
        email: mail,
        first_name: givenName,
        last_name: sn
      }
    },
    enabled: true,
    name: Auth_LDAP,
    type: ansible_base.authentication.authenticator_plugins.ldap
  }
]

controller_authenticator_maps": [
  {'name': 'Auth_LDAP_team_Team A_map', 'authenticator': 'Auth_LDAP', 'order': 0,
   'map_type': 'team', 'role': 'Team Member',
```

```
    'organization': 'Org A', 'team': 'Team A', 'revoke': True,
    'triggers': {'groups': {'has_or': ['CN=teama,OU=Users,DC=example,DC=com']}}},

   {'name': 'Auth_LDAP_team_Team C_map', 'authenticator': 'Auth_LDAP', 'order': 0,
    'map_type': 'team', 'role': 'Team Member',
    'organization': 'Org C', 'team': 'Team C', 'revoke': False,
    'triggers': {'groups': {'has_or': ['CN=teama,OU=Users,DC=example,DC=com',
                                       'CN=teama2,OU=Users,DC=example,DC=com']}}},

   {'name': 'Auth_LDAP_org_Org A_user_map', 'authenticator': 'Auth_LDAP', 'order': 0,
    'map_type': 'organization', 'role': 'Organization Member',
    'organization': 'Org A', 'team': '', 'revoke': True,
    'triggers': {'groups': {'has_or': ['CN=orga,OU=Users,DC=example,DC=com']}}},

   {'name': 'Auth_LDAP_org_Org A_admin_map', 'authenticator': 'Auth_LDAP', 'order': 0,
    'map_type': 'organization', 'role': 'Organization Admin',
    'organization': 'Org A', 'team': '', 'revoke': True,
    'triggers': {'groups': {'has_or': ['CN=orgaadm,OU=Users,DC=example,DC=com']}}},

   {'name': 'Auth_LDAP_org_Org C_user_map', 'authenticator': 'Auth_LDAP', 'order': 0,
    'map_type': 'organization', 'role': 'Organization Member',
    'organization': 'Org C', 'team': '', 'revoke': False,
    'triggers': {'groups': {'has_or': ['CN=orgc,OU=Users,DC=example,DC=com',
                                       'CN=orgc2,OU=Users,DC=example,DC=com']}}}
]
```

| | |
|---|---|
| 11. To export all supported controller configurations skip **export_collections** and **export_repositories** tags, related to hub configurations. | *ansible-playbook get_objects.yml -e format_for_25=true --skip-tags export_collections,export_repositories* |
| 12. Be sure to stop and delete your demo environment to avoid unnecessary charges | |

## Conclusion

This lab doesn't cover the full range of possible AAP configurations, but it provides a solid starting point and a good overview of the available options for using Configuration-as-Code (CaC).

Remember, every job needs the right tool to get it done efficiently. The **infra.aap_configuration** collection is ideal for creating configurations from scratch - quickly and with minimal required fields.

However, when the goal is to **migrate existing configurations** or **manage configuration drift**, the **`configify.aapconfig`** collection may be the better choice for now, as it focuses on full configuration coverage and alignment with the current controller state.

Both collections are open source - feel free to contribute to either one, or help port features from **configify.aapconfig** into the **Red Hat-verified infra.aap_configuration**.