Red Hat Ansible Automation Platform 2.4

Red Hat Ansible Automation Platform upgrade and migration guide

Upgrade and migrate legacy deployments of Ansible Automation Platform

Last Updated: 2024-08-07

Red Hat Ansible Automation Platform 2.4 Red Hat Ansible Automation Platform upgrade and migration guide

Upgrade and migrate legacy deployments of Ansible Automation Platform

Legal Notice

Abstract

This guide shows you how to upgrade to the latest version of Ansible Automation Platform and migrate legacy virtual environments to automation execution environments.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

If you have a suggestion to improve this documentation, or find an error, you can contact technical support at https://access.redhat.com to open a request.

CHAPTER 1. RED HAT ANSIBLE AUTOMATION PLATFORM 2.4 UPGRADES

Upgrade to Red Hat Ansible Automation Platform 2.4 by setting up your inventory and running the installation script. Ansible then upgrades your deployment to 2.4. If you plan to upgrade from Ansible Automation Platform 2.0 or earlier, you must migrate Ansible content for compatibility with 2.4.

1.1. ANSIBLE AUTOMATION PLATFORM UPGRADES

Upgrading to version 2.4 from Ansible Automation Platform 2.1 or later involves downloading the installation package and then performing the following steps:

- Set up your inventory to match your installation environment.
- Run the 2.4 installation program over your current Ansible Automation Platform installation.

Additional resources

- Upgrading to Red Hat Ansible Automation Platform 2.4

1.2. ANSIBLE AUTOMATION PLATFORM LEGACY UPGRADES

Upgrading to version 2.4 from Ansible Automation Platform 2.0 or earlier requires you to migrate Ansible content for compatibility.

The following steps provide an overview of the legacy upgrade process:

- Duplicate your custom virtual environments into automation execution environments by using the **awx-manage** command.
- Migrate data from isolated legacy nodes to execution nodes by performing a side-by-side upgrade so nodes are compatible with the latest automation mesh features.
- Import or generate a new automation hub API token.
- Reconfigure your Ansible content to include Fully Qualified Collection Names (FQCN) for compatibility with **ansible-core** 2.15.

Additional resources

- Migrate virtual environments to automation execution environments
- Migrate isolated nodes to execution nodes
- Migrate Ansible content

CHAPTER 2. UPGRADING TO RED HAT ANSIBLE AUTOMATION PLATFORM 2.4

To upgrade your Red Hat Ansible Automation Platform, start by reviewing planning information to ensure a successful upgrade. You can then download the desired version of the Ansible Automation Platform installer, configure the inventory file in the installation bundle to reflect your environment, and then run the installer.

2.1. ANSIBLE AUTOMATION PLATFORM UPGRADE PLANNING

Before you begin the upgrade process, review the following considerations to plan and prepare your Ansible Automation Platform deployment:

Automation controller

- Even if you have a valid license from a previous version, you must provide your credentials or a subscriptions manifest upon upgrading to the latest version of automation controller.
- If you need to upgrade Red Hat Enterprise Linux and automation controller, you must first backup and restore your automation controller data.
- Clustered upgrades require special attention to instance and instance groups before upgrading.

Additional resources

- Importing a subscription
- Backup and restore
- Clustering

Automation hub

- When upgrading to Ansible Automation Platform 2.4, you can either add an existing automation hub API token or generate a new one and invalidate any existing tokens.
- Existing container images are removed when upgrading Ansible Automation Platform. This is because, when upgrading Ansible Automation Platform with **setup.sh** script, podman **system reset -f** is executed. This removes all container images on your Ansible Automation Platform nodes then pushes the new execution environment image that is bundled with installer. See Running the Red Hat Ansible Automation Platform installer setup script .

Additional resources

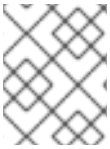
- Setting up the inventory file

Event-Driven Ansible controller

- If you are currently running Event-Driven Ansible controller and plan to deploy it when you upgrade to Ansible Automation Platform 2.4, it is recommended that you disable all Event-Driven Ansible activations before upgrading to ensure that only new activations run after the upgrade process has completed. This prevents possibilities of orphaned containers running activations from the previous version.

2.2. CHOOSING AND OBTAINING A RED HAT ANSIBLE AUTOMATION PLATFORM INSTALLER

Choose the Red Hat Ansible Automation Platform installer you need based on your Red Hat Enterprise Linux environment internet connectivity. Review the following scenarios and decide on which Red Hat Ansible Automation Platform installer meets your needs.



NOTE

A valid Red Hat customer account is required to access Red Hat Ansible Automation Platform installer downloads on the Red Hat Customer Portal.

Installing with internet access

Choose the Red Hat Ansible Automation Platform installer if your Red Hat Enterprise Linux environment is connected to the internet. Installing with internet access retrieves the latest required repositories, packages, and dependencies. Choose one of the following ways to set up your Ansible Automation Platform installer.

Tarball install

1. Navigate to the Red Hat Ansible Automation Platform download page.
2. Click **Download Now** for the **Ansible Automation Platform <latest-version> Setup**.
3. Extract the files:

```
$ tar xvzf ansible-automation-platform-setup-<latest-version>.tar.gz
```

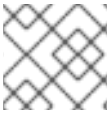
RPM install

1. Install Ansible Automation Platform Installer Package
   v.2.4 for RHEL 8 for x86_64

```
$ sudo dnf install --enablerepo=ansible-automation-platform-2.4-for-rhel-8-x86_64-rpms
ansible-automation-platform-installer
```

   v.2.4 for RHEL 9 for x86-64

```
$ sudo dnf install --enablerepo=ansible-automation-platform-2.4-for-rhel-9-x86_64-rpms
ansible-automation-platform-installer
```



NOTE

dnf install enables the repo as the repo is disabled by default.

When you use the RPM installer, the files are placed under the **/opt/ansible-automation-platform/installer** directory.

Installing without internet access

Use the Red Hat Ansible Automation Platform **Bundle** installer if you are unable to access the internet, or would prefer not to install separate components and dependencies from online repositories. Access

to Red Hat Enterprise Linux repositories is still needed. All other dependencies are included in the tar archive.

1. Navigate to the Red Hat Ansible Automation Platform download page.
2. Click **Download Now** for the **Ansible Automation Platform <latest-version> Setup Bundle**
3. Extract the files:

```
$ tar xvzf ansible-automation-platform-setup-bundle-<latest-version>.tar.gz
```

2.3. SETTING UP THE INVENTORY FILE

Before upgrading your Red Hat Ansible Automation Platform installation, edit the **inventory** file so that it matches your desired configuration. You can keep the same parameters from your existing Ansible Automation Platform deployment or you can modify the parameters to match any changes to your environment.

Procedure

1. Navigate to the installation program directory.

   Bundled installer

   ```
   $ cd ansible-automation-platform-setup-bundle-2.4-1-x86_64
   ```

   Online installer

   ```
   $ cd ansible-automation-platform-setup-2.4-1
   ```

2. Open the **inventory** file for editing.
3. Modify the **inventory** file to provision new nodes, deprovision nodes or groups, and import or generate automation hub API tokens.
   You can use the same **inventory** file from an existing Ansible Automation Platform 2.1 installation if there are no changes to the environment.

NOTE

Provide a reachable IP address or fully qualified domain name (FQDN) for the **[automationhub]** and **[automationcontroller]** hosts to ensure that users can synchronize and install content from Ansible automation hub from a different node. Do not use **localhost**. If **localhost** is used, the upgrade will be stopped as part of preflight checks.

Provisioning new nodes in a cluster

- Add new nodes alongside existing nodes in the **inventory** file as follows:

```
[controller]
clusternode1.example.com
clusternode2.example.com
clusternode3.example.com
```

```
[all:vars]
admin_password='password'

pg_host=''
pg_port=''

pg_database='<database_name>'
pg_username='<your_username>'
pg_password='<your_password>'
```

Deprovisioning nodes or groups in a cluster

- Append **node_state-deprovision** to the node or group within the **inventory** file.

Importing and generating API tokens

When upgrading from Red Hat Ansible Automation Platform 2.0 or earlier to Red Hat Ansible Automation Platform 2.1 or later, you can use your existing automation hub API token or generate a new token. In the inventory file, edit one of the following fields before running the Red Hat Ansible Automation Platform installer setup script **setup.sh**:

- Import an existing API token with the **automationhub_api_token** flag as follows:

```
automationhub_api_token=<api_token>
```

- Generate a new API token, and invalidate any existing tokens, with the **generate_automationhub_token** flag as follows:

```
generate_automationhub_token=True
```

Additional resources

- Red Hat Ansible Automation Platform Installation Guide
- Deprovisioning individual nodes or instance groups

2.4. RUNNING THE RED HAT ANSIBLE AUTOMATION PLATFORM INSTALLER SETUP SCRIPT

You can run the setup script once you have finished updating the **inventory** file.

Procedure

1. Run the **setup.sh** script

```
$ ./setup.sh
```

The installation will begin.

CHAPTER 3. MIGRATING TO AUTOMATION EXECUTION ENVIRONMENTS

3.1. WHY UPGRADE TO AUTOMATION EXECUTION ENVIRONMENTS?

Red Hat Ansible Automation Platform 2.4 introduces automation execution environments. Automation execution environments are container images that allow for easier administration of Ansible by including everything needed to run Ansible automation within a single container. Automation execution environments include:

- RHEL UBI 8
- Ansible-core 2.14 or later
- Python 3.9 or later.
- Any Ansible Content Collections
- Collection python or binary dependencies

By including these elements, Ansible provides platform administrators a standardized way to define, build, and distribute the environments the automation runs in.

Due to the new automation execution environment, it is no longer necessary for administrators to create custom plugins and automation content. Administrators can now spin up smaller automation execution environments in less time to create their content.

All custom dependencies are now defined in the development phase instead of the administration and deployment phase. Decoupling from the control plane enables faster development cycles, scalability, reliability, and portability across environments. Automation execution environments enables the Ansible Automation Platform to move to a distributed architecture allowing administrators to scale automation across their organization.

3.2. ABOUT MIGRATING LEGACY VENVS TO AUTOMATION EXECUTION ENVIRONMENTS

When upgrading from older versions of automation controller to version 4.0 or later, the controller can detect previous versions of virtual environments associated with Organizations, Inventory and Job Templates and informs you to migrate to the new automation execution environments model. A new installation of automation controller creates two virtualenvs during the installation; one runs the controller and the other runs Ansible. Like legacy virtual environments, automation execution environments allow the controller to run in a stable environment, while allowing you to add or update modules to your automation execution environments as necessary to run your playbooks.

You can duplicate your setup in an automation execution environment from a previous custom virtual environment by migrating them to the new automation execution environment. Use the **awx-manage** commands in this section to:

- list of all the current custom virtual environments and their paths (**list_custom_venvs**)
- view the resources that rely a particular custom virtual environment (**custom_venv_associations**)

- export a particular custom virtual environment to a format that can be used to migrate to an automation execution environment (**export_custom_venv**)

The below workflow describes how to migrate from legacy venvs to automation execution environments using the **awx-manage** command.

3.3. MIGRATING VIRTUAL ENVS TO AUTOMATION EXECUTION ENVIRONMENTS

Use the following sections to assist with additional steps in the migration process once you have upgraded to Red Hat Ansible Automation Platform 2.0 and automation controller 4.0.

3.3.1. Listing custom virtual environments

You can list the virtual environments on your automation controller instance using the **awx-manage** command.

Procedure

1. SSH into your automation controller instance and run:

```
$ awx-manage list_custom_venvs
```

A list of discovered virtual environments will appear.

```
# Discovered virtual environments:
/var/lib/awx/venv/testing
/var/lib/venv/new_env

To export the contents of a virtual environment, re-run while supplying the path as an argument:
awx-manage export_custom_venv /path/to/venv
```

3.3.2. Viewing objects associated with a custom virtual environment

View the organizations, jobs, and inventory sources associated with a custom virtual environment using the **awx-manage** command.

Procedure

1. SSH into your automation controller instance and run:

```
$ awx-manage custom_venv_associations /path/to/venv
```

A list of associated objects will appear.

```
inventory_sources:
- id: 15
  name: celery
job_templates:
- id: 9
  name: Demo Job Template @ 2:40:47 PM
- id: 13
```

```
  name: elephant
organizations
- id: 3
  name: alternating_bongo_meow
- id: 1
  name: Default
projects: []
```

3.3.3. Selecting the custom virtual environment to export

Select the custom virtual environment you want to export by using **awx-manage export_custom_venv** command.

Procedure

1. SSH into your automation controller instance and run:

```
$ awx-manage export_custom_venv /path/to/venv
```

The output from this command will show a **pip freeze** of what is in the specified virtual environment. This information can be copied into a **requirements.txt** file for Ansible Builder to use for creating a new automation execution environments image.

```
numpy==1.20.2
pandas==1.2.4
python-dateutil==2.8.1
pytz==2021.1
six==1.16.0

To list all available custom virtual environments run:
awx-manage list_custom_venvs
```

NOTE

Pass the **-q** flag when running **awx-manage list_custom_venvs** to reduce output.

CHAPTER 4. MIGRATING ISOLATED NODES TO EXECUTION NODES

Upgrading from version 1.x to the latest version of the Red Hat Ansible Automation Platform requires platform administrators to migrate data from isolated legacy nodes to execution nodes. This migration is necessary to deploy the automation mesh.

This guide explains how to perform a side-by-side migration. This ensures that the data on your original automation environment remains untouched during the migration process.

The migration process involves the following steps:

1. Verify upgrade configurations.
2. Backup original instance.
3. Deploy new instance for a side-by-side upgrade.
4. Recreate instance groups in the new instance using ansible controller.
5. Restore original backup to new instance.
6. Set up execution nodes and upgrade instance to Red Hat Ansible Automation Platform 2.4.
7. Configure upgraded controller instance.

4.1. PREREQUISITES FOR UPGRADING ANSIBLE AUTOMATION PLATFORM

Before you begin to upgrade Ansible Automation Platform, ensure your environment meets the following node and configuration requirements.

4.1.1. Node requirements

The following specifications are required for the nodes involved in the Ansible Automation Platform upgrade process:

- 16 GB of RAM for controller nodes, database node, execution nodes and hop nodes.
- 4 CPUs for controller nodes, database nodes, execution nodes, and hop nodes.
- 150 GB+ disk space for database node.
- 40 GB+ disk space for non-database nodes.
- DHCP reservations use infinite leases to deploy the cluster with static IP addresses.
- DNS records for all nodes.
- Red Hat Enterprise Linux 8 or later 64-bit (x86) installed for all nodes.
- Chrony configured for all nodes.
- Python 3.9 or later for all content dependencies.

4.1.2. Automation controller configuration requirements

The following automation controller configurations are required before you proceed with the Ansible Automation Platform upgrade process:

Configuring NTP server using Chrony

Each Ansible Automation Platform node in the cluster must have access to an NTP server. Use the **chronyd** to synchronize the system clock with NTP servers. This ensures that cluster nodes using SSL certificates that require validation do not fail if the date and time between nodes are not in sync.

This is required for all nodes used in the upgraded Ansible Automation Platform cluster:

1. Install **chrony**:

```
# dnf install chrony --assumeyes
```

2. Open /**etc**/**chrony.conf** using a text editor.

3. Locate the public server pool section and modify it to include the appropriate NTP server addresses. Only one server is required, but three are recommended. Add the 'iburst' option to speed up the time it takes to properly sync with the servers:

```
# Use public servers from the pool.ntp.org project.
# Please consider joining the pool (http://www.pool.ntp.org/join.html).
server <ntp-server-address> iburst
```

4. Save changes within the /**etc**/**chrony.conf** file.

5. Start the host and enable the **chronyd** daemon:

```
# systemctl --now enable chronyd.service
```

6. Verify the **chronyd** daemon status:

```
# systemctl status chronyd.service
```

Attaching Red Hat subscription on all nodes

Red Hat Ansible Automation Platform requires you to have valid subscriptions attached to all nodes. You can verify that your current node has a Red Hat subscription by running the following command:

```
# subscription-manager list --consumed
```

If there is no Red Hat subscription attached to the node, see Attaching your Ansible Automation Platform subscription for more information.

Creating non-root user with sudo privileges

Before you upgrade Ansible Automation Platform, it is recommended to create a non-root user with sudo privileges for the deployment process. This user is used for:

- SSH connectivity.

- Passwordless authentication during installation.
- Privilege escalation (sudo) permissions.

The following example uses **ansible** to name this user. On all nodes used in the upgraded Ansible Automation Platform cluster, create a non-root user named **ansible** and generate an SSH key:

1. Create a non-root user:

```
# useradd ansible
```

2. Set a password for your user:

```
# passwd ansible 1
Changing password for ansible.
Old Password:
New Password:
Retype New Password:
```

1 Replace **ansible** with the non-root user from step 1, if using a different name

3. Generate an **ssh** key as the user:

```
$ ssh-keygen -t rsa
```

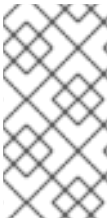
4. Disable password requirements when using **sudo**:

```
# echo "ansible ALL=(ALL) NOPASSWD:ALL" | sudo tee -a /etc/sudoers.d/ansible
```

Copying SSH keys to all nodes

With the **ansible** user created, copy the **ssh** key to all the nodes used in the upgraded Ansible Automation Platform cluster. This ensures that when the Ansible Automation Platform installation runs, it can **ssh** to all the nodes without a password:

```
$ ssh-copy-id ansible@node-1.example.com
```



NOTE

If running within a cloud provider, you might need to instead create an **~/.ssh/authorized_keys** file containing the public key for the **ansible** user on all your nodes and set the permissions to the **authorized_keys** file to only the owner (**ansible**) having read and write access (permissions 600).

Configuring firewall settings

Configure the firewall settings on all the nodes used in the upgraded Ansible Automation Platform cluster to allow access to the appropriate services and ports for a successful Ansible Automation Platform upgrade. For Red Hat Enterprise Linux 8 or later, enable the **firewalld** daemon to enable the access needed for all nodes:

1. Install the **firewalld** package:

-

```
# dnf install firewalld --assumeyes
```

2. Start the **firewalld** service:

```
# systemctl start firewalld
```

3. Enable the **firewalld** service:

```
# systemctl enable --now firewalld
```

4.1.3. Ansible Automation Platform configuration requirements

The following Ansible Automation Platform configurations are required before you proceed with the Ansible Automation Platform upgrade process:

Configuring firewall settings for execution and hop nodes

After upgrading your Red Hat Ansible Automation Platform instance, add the automation mesh port on the mesh nodes (execution and hop nodes) to enable automation mesh functionality. The default port used for the mesh networks on all nodes is **27199/tcp**. You can configure the mesh network to use a different port by specifying **recptor_listener_port** as the variable for each node within your inventory file.

Within your hop and execution node set the **firewalld** port to be used for installation.

1. Ensure that **firewalld** is running:

```
$ sudo systemctl status firewalld
```

2. Add the **firewalld** port to your controller database node (e.g. port 27199):

```
$ sudo firewall-cmd --permanent --zone=public --add-port=27199/tcp
```

3. Reload **firewalld**:

```
$ sudo firewall-cmd --reload
```

4. Confirm that the port is open:

```
$ sudo firewall-cmd --list-ports
```

4.2. BACK UP YOUR ANSIBLE AUTOMATION PLATFORM INSTANCE

Back up an existing Ansible Automation Platform instance by running the **.setup.sh** script with the **backup_dir** flag, which saves the content and configuration of your current environment:

1. Navigate to your **ansible-tower-setup-latest** directory.
2. Run the **./setup.sh** script following the example below:

```
$ ./setup.sh -e ‘backup_dir=/ansible/mybackup’ -e ‘use_compression=True’ @credentials.yml
-b 1 2
```

```
-
```

1. **backup_dir** specifies a directory to save your backup to.
2. **@credentials.yml** passes the password variables and their values encrypted via **ansible-vault**.

With a successful backup, a backup file is created at **/ansible/mybackup/tower-backup-latest.tar.gz** .

This backup will be necessary later to migrate content from your old instance to the new one.

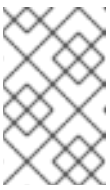
4.3. DEPLOY A NEW INSTANCE FOR A SIDE-BY-SIDE UPGRADE

To proceed with the side-by-side upgrade process, deploy a second instance of Ansible Tower 3.8.x with the same instance group configurations. This new instance will receive the content and configuration from your original instance, and will later be upgraded to Red Hat Ansible Automation Platform 2.4.

4.3.1. Deploy a new instance of Ansible Tower

To deploy a new Ansible Tower instance, do the following:

1. Download the Tower installer version that matches your original Tower instance by navigating to the Ansible Tower installer page .
2. Navigate to the installer, then open the **inventory** file using a text editor to configure the **inventory** file for a Tower installation:
   a. In addition to any Tower configurations, remove any fields containing **isolated_group** or **instance_group**.



NOTE

For more information about installing Tower using the Ansible Automation Platform installer, see the Ansible Automation Platform Installation Guide for your specific installation scenario.

3. Run the **setup.sh** script to begin the installation.

Once the new instance is installed, configure the Tower settings to match the instance groups from your original Tower instance.

4.3.2. Recreate instance groups in the new instance

To recreate your instance groups in the new instance, do the following:

NOTE

Make note of all instance groups from your original Tower instance. You will need to recreate these groups in your new instance.

1. Log in to your new instance of Tower.
2. In the navigation pane, select **Administration → Instance Groups**.

3. Click **Create instance group**.
4. Enter a **Name** that matches an instance group from your original instance, then click **Save**
5. Repeat until all instance groups from your original instance have been recreated.

4.4. RESTORE BACKUP TO NEW INSTANCE

Running the **./setup.sh** script with the **restore_backup_file** flag migrates content from the backup file of your original 1.x instance to the new instance. This effectively migrates all job histories, templates, and other Ansible Automation Platform related content.

Procedure

1. Run the following command:

```
$ ./setup.sh -r -e 'restore_backup_file=/ansible/mybackup/tower-backup-latest.tar.gz' -e
'use_compression=True' -e @credentials.yml -r -- --ask-vault-pass (1) (2) (3)
```

(1) **restore_backup_file** specifies the location of the Ansible Automation Platform backup database

(2) **use_compression** is set to **True** due to compression being used during the backup process

(3) **-r** sets the restore database option to **True**

2. Log in to your new RHEL 8 Tower 3.8 instance to verify whether the content from your original instance has been restored:
   a. Navigate to **Administration → Instance Groups**. The recreated instance groups should now contain the **Total Jobs** from your original instance.
   b. Using the side navigation panel, check that your content has been imported from your original instance, including Jobs, Templates, Inventories, Credentials, and Users.

You now have a new instance of Ansible Tower with all the Ansible content from your original instance.

You will upgrade this new instance to Ansible Automation Platform 2.4 so that you keep all your previous data without overwriting your original instance.

4.5. UPGRADING TO ANSIBLE AUTOMATION PLATFORM 2.4

To upgrade your instance of Ansible Tower to Ansible Automation Platform 2.4, copy the **inventory** file from your original Tower instance to your new Tower instance and run the installer. The Red Hat Ansible Automation Platform installer detects a pre-2.4 and offers an upgraded inventory file to continue with the upgrade process:

1. Download the latest installer for Red Hat Ansible Automation Platform from the Red Hat Ansible Automation Platform download page.
2. Extract the files:

```
$ tar xvzf ansible-automation-platform-setup-<latest_version>.tar.gz
```

3. Navigate into your Ansible Automation Platform installation directory:

```
$ cd ansible-automation-platform-setup-<latest_version>/
```

4. Copy the **inventory** file from your original instance into the directory of the latest installer:

```
$ cp ansible-tower-setup-3.8.x.x/inventory ansible-automation-platform-
setup-<latest_version>
```

5. Run the **setup.sh** script:

```
$ ./setup.sh
```

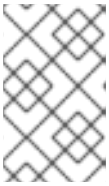
The setup script pauses and indicates that a "pre-2.x" inventory file was detected, but offers a new file called **inventory.new.ini** allowing you to continue to upgrade your original instance.

6. Open **inventory.new.ini** with a text editor.

NOTE

By running the setup script, the Installer modified a few fields from your original inventory file, such as renaming [tower] to [automationcontroller].

7. Update the newly generated **inventory.new.ini** file to configure your automation mesh by assigning relevant variables, nodes, and relevant node-to-node peer connections:



NOTE

The design of your automation mesh topology depends on the automation needs of your environment. It is beyond the scope of this document to provide designs for all possible scenarios. The following is one example automation mesh design.

Example inventory file with a standard control plane consisting of three nodes utilizing hop nodes:

```
[automationcontroller]
control-plane-1.example.com
control-plane-2.example.com
control-plane-3.example.com

[automationcontroller:vars]
node_type=control (1)
peers=execution_nodes (2)


[execution_nodes]
execution-node-1.example.com peers=execution-node-2.example.com
execution-node-2.example.com peers=execution-node-3.example.com
execution-node-3.example.com peers=execution-node-4.example.com
execution-node-4.example.com peers=execution-node-5.example.com node_type=hop
execution-node-5.example.com peers=execution-node-6.example.com node_type=hop (3)
execution-node-6.example.com peers=execution-node-7.example.com
execution-node-7.example.com
```

```
[execution_nodes:vars]
node_type=execution
```

1. Specifies a control node that runs project and inventory updates and system jobs, but not regular jobs. Execution capabilities are disabled on these nodes.
2. Specifies peer relationships for node-to-node connections in the **[execution_nodes]** group.
3. Specifies hop nodes that route traffic to other execution nodes. Hop nodes cannot execute automation.

8. Import or generate a automation hub API token.
   - Import an existing API token with the **automationhub_api_token** flag:

     ```
     automationhub_api_token=<api_token>
     ```

   - Generate a new API token, and invalidate any existing tokens, by setting the **generate_automationhub_token** flag to **True**:

     ```
     generate_automationhub_token=True
     ```

9. Once you have finished configuring your **inventory.new.ini** for automation mesh, run the setup script using **inventory.new.ini**:

   ```
   $ ./setup.sh -i inventory.new.ini -e @credentials.yml -- --ask-vault-pass
   ```

10. Once the installation completes, verify that your Ansible Automation Platform has been installed successfully by logging in to the Ansible Automation Platform dashboard UI across all automation controller nodes.

Additional resources

- For general information about using the Ansible Automation Platform installer, see the Red Hat Ansible Automation Platform installation guide.

4.6. CONFIGURING YOUR UPGRADED ANSIBLE AUTOMATION PLATFORM

4.6.1. Configuring automation controller instance groups

After upgrading your Red Hat Ansible Automation Platform instance, associate your original instances to its corresponding instance groups by configuring settings in the automation controller UI:

1. Log in to the new Controller instance.
2. Content from old instance, such as credentials, jobs, inventories should now be visible on your Controller instance.
3. Navigate to **Administration → Instance Groups**.

4. Associate execution nodes by clicking on an instance group, then click the **Instances** tab.
5. Click **Associate**. Select the node(s) to associate to this instance group, then click **Save**.
6. You can also modify the default instance to disassociate your new execution nodes.

CHAPTER 5. ANSIBLE CONTENT MIGRATION

If you are migrating from an **ansible-core** version to **ansible-core** 2.13, consider reviewing Ansible core Porting Guides to familiarize yourself with changes and updates between each version. When reviewing the Ansible core porting guides, ensure that you select the latest version of **ansible-core** or **devel**, which is located at the top left column of the guide.

For a list of fully supported and certified Ansible Content Collections, see Ansible Automation hub on console.redhat.com.

5.1. INSTALLING ANSIBLE COLLECTIONS

As part of the migration from earlier Ansible versions to more recent versions, you need to find and download the collections that include the modules you have been using. Once you find that list of collections, you can use one of the following options to include your collections locally:

1. Download and install the Collection into your runtime or execution environments using **ansible-builder**.

2. Update the 'requirements.yml' file in your Automation Controller project install roles and collections. This way every time you sync the project in Automation Controller the roles and collections will be downloaded.

NOTE

In many cases the upstream and downstream Collections could be the same, but always download your certified collections from Automation Hub.

5.2. MIGRATING YOUR ANSIBLE PLAYBOOKS AND ROLES TO CORE 2.13

When you are migrating from non collection-based content to collection-based content, you should use the Fully Qualified Collection Names (FQCN) in playbooks and roles to avoid unexpected behavior.

Example playbook with FQCN:

```
- name: get some info
  amazon.aws.ec2_vpc_net_info:
    region: "{{ec2_region}}"
  register: all_the_info
  delegate_to: localhost
  run_once: true
```

If you are using ansible-core modules and are not calling a module from a different Collection, you should use the FQCN **ansible.builtin.copy**.

Example module with FQCN:

```
- name: copy file with owner and permissions
  ansible.builtin.copy:
  src: /srv/myfiles/foo.conf
  dest: /etc/foo.conf
```

```
owner: foo
group: foo
mode: '0644'
```

5.3. CONVERTING PLAYBOOK EXAMPLES

Examples

This example is of a shared directory called **/mydata** in which we want to be able to read and write files to during a job run. Remember this has to already exist on the execution node we will be using for the automation run.

You will target the *aape1.local* execution node to run this job, because the underlying hosts already has this in place.

```
[awx@aape1 ~]$ ls -la /mydata/
total 4
drwxr-xr-x.  2 awx  awx   41 Apr 28 09:27 .
dr-xr-xr-x. 19 root root 258 Apr 11 15:16 ..
-rw-r--r--.  1 awx  awx   33 Apr 11 12:34 file_read
-rw-r--r--.  1 awx  awx    0 Apr 28 09:27 file_write
```

You will use a simple playbook to launch the automation with sleep defined to allow you access, and to understand the process, as well as demonstrate reading and writing to files.

```
# vim:ft=ansible:
```

```
- hosts: all
  gather_facts: false
  ignore_errors: yes
  vars:
    period: 120
    myfile: /mydata/file
  tasks:
    - name: Collect only selected facts
      ansible.builtin.setup:
        filter:
          - 'ansible_distribution'
          - 'ansible_machine_id'
          - 'ansible_memtotal_mb'
          - 'ansible_memfree_mb'
    - name: "I'm feeling real sleepy..."
      ansible.builtin.wait_for:
        timeout: "{{ period }}"
      delegate_to: localhost
    - ansible.builtin.debug:
        msg: "Isolated paths mounted into execution node: {{ AWX_ISOLATIONS_PATHS }}"
    - name: "Read pre-existing file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_read'
    - name: "Write to a new file..."
      ansible.builtin.copy:
        dest: "{{ myfile }}_write"
        content: |
```

```
        This is the file I've just written to.

    - name: "Read written out file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_write') }}"
```

From the Ansible Automation Platform 2 navigation panel, select **Settings**. Then select **Job settings** from the **Jobs** option.

Paths to expose isolated jobs:

```
[
"/mydata:/mydata:rw"
]
```

The volume mount is mapped with the same name in the container and has read-write capability. This will get used when you launch the job template.

The *prompt on launch* should be set for *extra_vars* so you can adjust the sleep duration for each run, The default is 30 seconds.

Once launched, and the *wait_for* module is invoked for the sleep, you can go onto the execution node and look at what is running.

To verify the run has completed successfully, run this command to get an output of the job:

```
$ podman exec -it 'podman ps -q' /bin/bash
bash-4.4#
```

You are now inside the running execution environment container.

Look at the permissions, you will see that **awx** has become 'root', but this is not really root as in the superuser, as you are using rootless Podman, which maps users into a kernel namespace similar to a sandbox. Learn more about How does rootless Podman work? for shadow-utils.

```
bash-4.4# ls -la /mydata/
Total 4
drwxr-xr-x. 2 root root 41 Apr 28 09:27 .
dr-xr-xr-x. 1 root root 77 Apr 28 09:40 ..
-rw-r—--r–. 1 root root 33 Apr 11 12:34 file_read
-rw-r—--r–. 1 root root  0 Apr 28 09:27 file_write
```

According to the results, this job failed. In order to understand why, the remaining output needs to be examined.

```
TASK [Read pre-existing file…]****************************** 10:50:12
ok: [localhost] =>  {
    “Msg”: “This is the file I am reading in.”

TASK {Write to a new file...}******************************** 10:50:12
An exception occurred during task execution. To see the full traceback, use -vvv. The error was:
PermissionError: [Errno 13] Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'
/mydata/file_write'
Fatal: [localhost]: FAILED! => {"changed": false, :checksum":
"9f576o85d584287a3516ee8b3385cc6f69bf9ce", "msg": "Unable to make
```

```
b'/root/.ansible/tmp/anisible-tim-1651139412.9808054-40-91081834383738/source' into
/mydata/file_write, failed final rename from b'/mydata/.ansible_tmpazyqyqdrfile_write': [Errno 13]
Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'/mydata/file_write}
...ignoring

TASK [Read written out file...] ****************************** 10:50:13
Fatal: [localhost]: FAILED: => {"msg": "An unhandled exception occurred while running the lookup
plugin 'file'. Error was a <class 'ansible.errors.AnsibleError;>, original message: could not locate file in
lookup: /mydate/file_write. Vould not locate file in lookup: /mydate/file_write"}
...ignoring
```

The job failed, even though **:rw** is set, so it should have write capability. The process was able to read the existing file, but not write out. This is due to SELinux protection that requires proper labels to be placed on the volume content mounted into the container. If the label is missing, SELinux may prevent the process from running inside the container. Labels set by the OS are not changed by Podman. See the Podman documentation for more information.

This could be a common misinterpretation. We have set the default to **:z**, which tells Podman to relabel file objects on shared volumes.

So we can either add **:z** or leave it off.

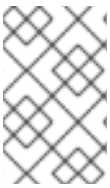
Paths to expose isolated jobs:

```
[
  "/mydata:/mydata"
]
```

The playbook will now work as expected:

```
PLAY [all] ******************************************************** 11:05:52
TASK [I'm feeling real sleepy. . .] ************************** 11:05:52
ok: [localhost]
TASK [Read pre-existing file...] ******************************** 11:05:57
ok: [localhost] => {
 "Msg": "This is the file I'm reading in."
}
TASK [Write to a new file…] ********************************** 11:05:57
ok: [localhost]
TASK [Read written out file…] ******************************** 11:05:58
ok: [localhost] => {
     “Msg”: “This is the file I’ve just written to.”
```

Back on the underlying execution node host, we have the newly written out contents.



NOTE

If you are using container groups to launch automation jobs inside Red Hat OpenShift, you can also tell Ansible Automation Platform 2 to expose the same paths to that environment, but you must toggle the default to **On** under settings.

Once enabled, this will inject this as *volumeMounts* and *volumes* inside the pod spec that will be used for execution. It will look like this:

```
apiVersion: v1
kind: Pod
Spec:
   containers:
   - image: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8
  args:
    - ansible runner
    - worker
    - –private-data-dir=/runner
  volumeMounts:
mountPath: /mnt2
name: volume-0
readOnly: true
mountPath: /mnt3
name: volume-1
readOnly: true
mountPath: /mnt4
name: volume-2
readOnly: true
volumes:
hostPath:
  path: /mnt2
  type: “”
name: volume-0
hostPath:
  path: /mnt3
  type: “”
name: volume-1
hostPath:
  path: /mnt4
  type: “”
name: volume-2
```

Storage inside the running container is using the overlay file system. Any modifications inside the running container are destroyed after the job completes, much like a tmpfs being unmounted.

CHAPTER 6. CONVERTING PLAYBOOKS FOR AAP2

With Ansible Automation Platform 2 and its containerised execution environments, the usage of *localhost* has been altered. In previous versions of Ansible Automation Platform, a job would run against *localhost*, which translated into running on the underlying Automation Controller host. This could be used to store data and persistent artifacts.

With Ansible Automation Platform 2, localhost means you are running inside a container, which is ephemeral in nature. *Localhost* is no longer tied to a particular host, and with portable execution environments, this means it can run anywhere with the right environment and software prerequisites already embedded into the execution environment container.

6.1. PERSISTING DATA FROM AUTO RUNS

Consider the local automation controller filesystem as counterproductive since that ties the data to that host. If you have a multi-node cluster, then you can contact a different host each time, causing issues if you are creating workflows that depend on each other and created directories. For example, if a directory was only created in one node while another node runs the playbook, the results would be inconsistent.

The solution is to use some form of shared storage solution, such as Amazon S3, Gist, or a role to rsync data to your data endpoint.

The option exists of injecting data or a configuration into a container at runtime. This can be achieved by using the automation controller's isolated jobs path option.

This provides a way to mount directories and files into an execution environment at runtime. This is achieved through the automation mesh, using ansible-runner to inject them into a Podman container to start the automation. What follows are some of the use cases for using isolated job paths:

- Providing SSL certificates at runtime, rather than baking them into an execution environment.
- Passing runtime configuration data, such as SSH config settings, but could be anything you want to use during automation.
- Reading and writing to files used before, during and after automation runs.

There are caveats to utilization:

- The volume mount has to pre-exist on all nodes capable of automation execution (so hybrid control plane nodes and all execution nodes).
- Where SELinux is enabled (Ansible Automation Platform default) beware of file permissions.
    - This is important since rootless Podman is run on non-OCP based installs.

The caveats need to be carefully observed. It is highly recommended to read up on rootless Podman and the Podman volume mount runtime options, the [:OPTIONS] part of the isolated job paths, as this is what is used inside Ansible Automation Platform 2.

Additional resources

- Understanding rootless Podman.
- Podman volume mount runtime options.

6.1.1. Converting playbook examples

Examples

This example is of a shared directory called **/mydata** in which we want to be able to read and write files to during a job run. Remember this has to already exist on the execution node we will be using for the automation run.

You will target the *aape1.local* execution node to run this job, because the underlying hosts already has this in place.

```
[awx@aape1 ~]$ ls -la /mydata/
total 4
drwxr-xr-x.  2 awx  awx   41 Apr 28 09:27 .
dr-xr-xr-x. 19 root root 258 Apr 11 15:16 ..
-rw-r--r--.  1 awx  awx   33 Apr 11 12:34 file_read
-rw-r--r--.  1 awx  awx    0 Apr 28 09:27 file_write
```

You will use a simple playbook to launch the automation with sleep defined to allow you access, and to understand the process, as well as demonstrate reading and writing to files.

```
# vim:ft=ansible:
```

```
- hosts: all
  gather_facts: false
  ignore_errors: yes
  vars:
    period: 120
    myfile: /mydata/file
  tasks:
    - name: Collect only selected facts
      ansible.builtin.setup:
        filter:
          - 'ansible_distribution'
          - 'ansible_machine_id'
          - 'ansible_memtotal_mb'
          - 'ansible_memfree_mb'
    - name: "I'm feeling real sleepy..."
      ansible.builtin.wait_for:
        timeout: "{{ period }}"
      delegate_to: localhost
    - ansible.builtin.debug:
        msg: "Isolated paths mounted into execution node: {{ AWX_ISOLATIONS_PATHS }}"
    - name: "Read pre-existing file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_read'
    - name: "Write to a new file..."
      ansible.builtin.copy:
        dest: "{{ myfile }}_write"
        content: |
          This is the file I've just written to.

    - name: "Read written out file..."
      ansible.builtin.debug:
        msg: "{{ lookup('file', '{{ myfile }}_write') }}"
```

- From the Ansible Automation Platform 2 navigation panel, select **Settings**. Then select **Job settings** from the **Jobs** option.

Paths to expose isolated jobs:

```
[
"/mydata:/mydata:rw"
]
```

The volume mount is mapped with the same name in the container and has read-write capability. This will get used when you launch the job template.

The *prompt on launch* should be set for *extra_vars* so you can adjust the sleep duration for each run, The default is 30 seconds.

Once launched, and the *wait_for* module is invoked for the sleep, you can go onto the execution node and look at what is running.

To verify the run has completed successfully, run this command to get an output of the job:

```
$ podman exec -it 'podman ps -q' /bin/bash
bash-4.4#
```

You are now inside the running execution environment container.

Look at the permissions, you will see that **awx** has become 'root', but this is not really root as in the superuser, as you are using rootless Podman, which maps users into a kernel namespace similar to a sandbox. Learn more about How does rootless Podman work? for shadow-utils.

```
bash-4.4# ls -la /mydata/
Total 4
drwxr-xr-x. 2 root root 41 Apr 28 09:27 .
dr-xr-xr-x. 1 root root 77 Apr 28 09:40 ..
-rw-r—--r–. 1 root root 33 Apr 11 12:34 file_read
-rw-r—--r–. 1 root root  0 Apr 28 09:27 file_write
```

According to the results, this job failed. In order to understand why, the remaining output needs to be examined.

```
TASK [Read pre-existing file…]****************************** 10:50:12
ok: [localhost] =>  {
    “Msg”: “This is the file I am reading in.”

TASK {Write to a new file...}****************************** 10:50:12
An exception occurred during task execution. To see the full traceback, use -vvv. The error was:
PermissionError: [Errno 13] Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'
/mydata/file_write'
Fatal: [localhost]: FAILED! => {"changed": false, :checksum":
"9f576o85d584287a3516ee8b3385cc6f69bf9ce", "msg": "Unable to make
b'/root/.ansible/tmp/anisible-tim-1651139412.9808054-40-91081834383738/source' into
/mydata/file_write, failed final rename from b'/mydata/.ansible_tmpazyqyqdrfile_write': [Errno 13]
Permission denied: b'/mydata/.ansible_tmpazyqyqdrfile_write' -> b'/mydata/file_write}
...ignoring
```

```
TASK [Read written out file...] ****************************** 10:50:13
Fatal: [localhost]: FAILED: => {"msg": "An unhandled exception occurred while running the lookup
plugin 'file'. Error was a <class 'ansible.errors.AnsibleError;>, original message: could not locate file in
lookup: /mydate/file_write. Vould not locate file in lookup: /mydate/file_write"}
...ignoring
```

The job failed, even though **:rw** is set, so it should have write capability. The process was able to read the existing file, but not write out. This is due to SELinux protection that requires proper labels to be placed on the volume content mounted into the container. If the label is missing, SELinux may prevent the process from running inside the container. Labels set by the OS are not changed by Podman. See the Podman documentation for more information.

This could be a common misinterpretation. We have set the default to **:z**, which tells Podman to relabel file objects on shared volumes.

So we can either add **:z** or leave it off.

Paths to expose isolated jobs:

```
[
  "/mydata:/mydata"
]
```

The playbook will now work as expected:

```
PLAY [all] ********************************************************* 11:05:52
TASK [I'm feeling real sleepy. . .] ************************** 11:05:52
ok: [localhost]
TASK [Read pre-existing file...] ****************************** 11:05:57
ok: [localhost] => {
 "Msg": "This is the file I'm reading in."
}
TASK [Write to a new file…] ********************************** 11:05:57
ok: [localhost]
TASK [Read written out file…] ******************************** 11:05:58
ok: [localhost] => {
     “Msg”: “This is the file I’ve just written to.”
```

Back on the underlying execution node host, we have the newly written out contents.

NOTE

If you are using container groups to launch automation jobs inside Red Hat OpenShift, you can also tell Ansible Automation Platform 2 to expose the same paths to that environment, but you must toggle the default to **On** under settings.

Once enabled, this will inject this as *volumeMounts* and *volumes* inside the pod spec that will be used for execution. It will look like this:

```
apiVersion: v1
kind: Pod
Spec:
  containers:
  - image: registry.redhat.io/ansible-automation-platform-24/ee-minimal-rhel8
```

```
  args:
    - ansible runner
    - worker
    - –private-data-dir=/runner
  volumeMounts:
mountPath: /mnt2
name: volume-0
readOnly: true
mountPath: /mnt3
name: volume-1
readOnly: true
mountPath: /mnt4
name: volume-2
readOnly: true
volumes:
hostPath:
  path: /mnt2
  type: “”
name: volume-0
hostPath:
  path: /mnt3
  type: “”
name: volume-1
hostPath:
  path: /mnt4
  type: “”
name: volume-2
```

Storage inside the running container is using the overlay file system. Any modifications inside the running container are destroyed after the job completes, much like a tmpfs being unmounted.