
Ansible Tower Upgrade and Migration

Release Ansible Tower 3.8.2

Red Hat, Inc.

Feb 10, 2023

CONTENTS

1	Release Notes for Ansible Tower Version 3.8.2	2
1.1	Ansible Tower Version 3.8.2	2
2	Upgrading to Ansible Automation Platform	3
2.1	Upgrade Planning	3
2.2	Obtaining the Installer	4
2.3	Setting up the Inventory File	4
2.4	Running the Setup Playbook	9
3	Role-Based Access Controls	10
3.1	Organization field on Job Templates	10
4	Using virtualenv with Ansible Tower	11
4.1	Preparing a new custom virtualenv	11
4.2	Assigning custom virtualenvs	13
5	Index	16
6	Copyright © Red Hat, Inc.	17
	Index	18

Thank you for your interest in Ansible Tower. Ansible Tower is a commercial offering that helps teams manage complex multi-tier deployments by adding control, knowledge, and delegation to Ansible-powered environments.

Note: You must upgrade your Ansible Tower to Ansible Tower 3.6 before you can upgrade to Ansible Tower 3.8.0.

We Need Feedback!

If you spot a typo in this documentation, or if you have thought of a way to make this manual better, we would love to hear from you! Please send an email to: docs@ansible.com

If you have a suggestion, try to be as specific as possible when describing it. If you have found an error, please include the manual's title, chapter number/section number, and some of the surrounding text so we can find it easily. We may not be able to respond to every message sent to us, but you can be sure that we will be reading them all!

Ansible Tower Version 3.8.2; March 09, 2021; <https://access.redhat.com/>

RELEASE NOTES FOR ANSIBLE TOWER VERSION 3.8.2

1.1 Ansible Tower Version 3.8.2

- Upgraded to the latest oVirt inventory plugin to resolve a number of inventory syncing issues that can occur on RHEL7
- Upgraded to the latest `theforeman.foreman` inventory plugin to resolve a few bugs and performance regressions
- Upgraded to a more recent version of Django to address CVE-2021-3281
- Upgraded to a more recent version of autobahn to address CVE-2020-35678
- Fixed a security issue that allowed a malicious playbook author to elevate to the awx user from outside the isolated environment (CVE-2021-20253)
- Fixed several issues related to how Tower rotates its log files
- Fixed the installer to no longer prevent Tower from installing on RHEL8 with certain non-en_US.UTF-8 locales
- Fixed unanticipated delays in certain playbook output
- Fixed job runs to no longer fail for playbooks that print certain types of raw binary data
- Fixed the generation of unnecessary records in the Activity Stream when Ansible Automation data is collected
- Fixed PostgreSQL backups to no longer fail when a non-default PostgreSQL username is specified
- Fixed access to encrypted Tower settings to prevent intermittent failures that caused failed job launches
- Fixed unexpected failures on certain long-running jobs running on isolated nodes
- Deprecated the global `/api/v2/job_events/` endpoint, which will be removed in a future release

UPGRADING TO ANSIBLE AUTOMATION PLATFORM

Starting with Ansible Tower 3.8, Automation Hub will act as a content provider for Ansible Tower, which requires both an Ansible Tower deployment and an Automation Hub deployment running alongside each other. The Ansible Automation Platform installer contains both of these. This section covers each component of the upgrading process:

- *Upgrade Planning*
- *Obtaining the Installer*
- *Setting up the Inventory File*
 - *Example Inventory files*
 - * *Example Standalone Automation Hub Inventory File*
 - * *Example Platform Inventory File*
 - * *Example Single Node Inventory File*
 - * *Example Multi Node Cluster Inventory File*
 - * *Example Inventory file for an external existing database*
 - * *Example Inventory file for external database which needs installation*
- *Running the Setup Playbook*

Note: All upgrades should be no more than two major versions behind what you are currently upgrading to. For example, in order to upgrade to Ansible Tower 3.7.x, you must first be on version 3.5.x; i.e., there is no direct upgrade path from version 3.4.x or earlier. Refer to the [recommended upgrade path](#) article on the Red Hat customer portal.

In order to run Ansible Tower 3.8, you must also have Ansible 2.9.

2.1 Upgrade Planning

This section covers changes that you should keep in mind as you attempt to upgrade your Ansible Tower instance.

- Even if you already have a valid license from a previous version, you must still provide your credentials or a subscriptions manifest again upon upgrading to Ansible Tower 3.8. See [Import a Subscription](#) in the *Ansible Tower User Guide*.
- If you need to upgrade Red Hat Enterprise Linux and Ansible Tower, you will need to do a backup and restore of your Tower data. Refer to [Upgrading an Existing Tower Installation](#) in the *Ansible Automation Platform*

Installation and Reference Guide for further detail.

- Clustered upgrades require special attention to instance and instance groups prior to starting the upgrade. Refer to the [Setting up the Inventory File](#) and see [Clustering](#) for details.
- Prior versions of Ansible Tower used the variable name `rabbitmq_host` during installation. If you are upgrading from a previous version of Tower, and you previously specified `rabbitmq_host` in your inventory, simply rename `rabbitmq_host` to `routable_hostname` before upgrading. See [Clustering](#) for details.

2.2 Obtaining the Installer

Refer to [Obtain the Ansible Automation Platform Installation Program](#) in the *Ansible Automation Platform Installation and Reference Guide* for detail.

2.3 Setting up the Inventory File

As you edit your inventory file, there are a few things you must keep in mind:

- The contents of the inventory file should be defined in `./inventory`, next to the `./setup.sh` installer playbook.
- For **installations and upgrades**: If you need to make use of external databases, you must ensure the database sections of your inventory file are properly setup. Edit this file and add your external database information before running the setup script.
- For **Ansible Automation Platform** or **Automation Hub**: Be sure to add an automation hub host in the `[automationhub]` group (Tower and Automation Hub cannot be installed on the same node).
- Tower will not configure replication or failover for the database that it uses, although Tower should work with any replication that you have.
- The database server should be on the same network or in the same data center as the Tower server for performance reasons.
- For **upgrading an existing cluster**: When upgrading a cluster, you may decide that you want to also reconfigure your cluster to omit existing instances or instance groups. Omitting the instance or the instance group from the inventory file will not be enough to remove them from the cluster. In addition to omitting instances or instance groups from the inventory file, you must also [deprovision instances or instance groups](#) before starting the upgrade. Otherwise, omitted instances or instance groups will continue to communicate with the cluster, which can cause issues with tower services during the upgrade.
- For **clustered installations**: If you are creating a clustered setup, you must replace `localhost` with the hostname or IP address of all instances. All nodes/instances must be able to reach any others using this hostname or address. In other words, you cannot use the `localhost ansible_connection=local` on one of the nodes *AND* all of the nodes should use the same format for the host names.

Therefore, this will *not* work:

```
[tower]
localhost ansible_connection=local
hostA
hostB.example.com
172.27.0.4
```

Instead, use these formats:

```
[tower]
hostA
hostB
hostC
```

OR

```
hostA.example.com
hostB.example.com
hostC.example.com
```

OR

```
[tower]
172.27.0.2
172.27.0.3
172.27.0.4
```

- **For all standard installations:** When performing an installation, you must supply any necessary passwords in the inventory file.

Note: Changes made to the installation process now require that you fill out all of the password fields in the inventory file. If you need to know where to find the values for these they should be:

```
admin_password='' ← Tower local admin password
pg_password='' ← Found in /etc/tower/conf.d/postgres.py
```

Warning: Do not use special characters in `pg_password` as it may cause the setup to fail.

2.3.1 Example Inventory files

- For **provisioning new nodes:** When provisioning new nodes add the nodes to the inventory file with all current nodes, make sure all passwords are included in the inventory file.
- For **upgrading a single node:** When upgrading, be sure to compare your inventory file to the current release version. It is recommended that you keep the passwords in here even when performing an upgrade.

Example Standalone Automation Hub Inventory File

```
[automationhub]
automationhub.acme.org
[all:vars]
automationhub_admin_password='<password>'
automationhub_pg_host=''
automationhub_pg_port=''
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='<password>'
automationhub_pg_sslmode='prefer'
# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
```

(continues on next page)

(continued from previous page)

```
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.crt
# automationhub_ssl_key=/path/to/automationhub.key
```

Example Platform Inventory File

```
[tower]
tower.acme.org
[automationhub]
automationhub.acme.org
[database]
database-01.acme.org
[all:vars]
admin_password='<password>'
pg_host='database-01.acme.org'
pg_port='5432'
pg_database='awx'
pg_username='awx'
pg_password='<password>'
pg_sslmode='prefer' # set to 'verify-full' for client-side enforced SSL
# Automation Hub Configuration
#
automationhub_admin_password='<password>'
automationhub_pg_host='database-01.acme.org'
automationhub_pg_port='5432'
automationhub_pg_database='automationhub'
automationhub_pg_username='automationhub'
automationhub_pg_password='<password>'
automationhub_pg_sslmode='prefer'
# The default install will deploy a TLS enabled Automation Hub.
# If for some reason this is not the behavior wanted one can
# disable TLS enabled deployment.
#
# automationhub_disable_https = False
# The default install will generate self-signed certificates for the Automation
# Hub service. If you are providing valid certificate via automationhub_ssl_cert
# and automationhub_ssl_key, one should toggle that value to True.
#
# automationhub_ssl_validate_certs = False
# Isolated Tower nodes automatically generate an RSA key for authentication;
# To disable this behavior, set this value to false
# isolated_key_generation=true
# SSL-related variables
# If set, this will install a custom CA certificate to the system trust store.
```

(continues on next page)

(continued from previous page)

```
# custom_ca_cert=/path/to/ca.crt
# Certificate and key to install in nginx for the web UI and API
# web_server_ssl_cert=/path/to/tower.crt
# web_server_ssl_key=/path/to/tower.key
# Certificate and key to install in Automation Hub node
# automationhub_ssl_cert=/path/to/automationhub.crt
# automationhub_ssl_key=/path/to/automationhub.key
# Server-side SSL settings for PostgreSQL (when we are installing it).
# postgres_use_ssl=False
# postgres_ssl_cert=/path/to/pgsql.crt
# postgres_ssl_key=/path/to/pgsql.key
```

Example Single Node Inventory File

```
[tower]
localhost ansible_connection=local

[database]

[all:vars]
admin_password='password'

pg_host=''
pg_port=''

pg_database='awx'
pg_username='awx'
pg_password='password'
```

Warning: Do not use special characters in `pg_password` as it may cause the setup to fail.

Example Multi Node Cluster Inventory File

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

[database]
dbnode.example.com

[all:vars]
ansible_become=true

admin_password='password'

pg_host='dbnode.example.com'
pg_port='5432'

pg_database='tower'
pg_username='tower'
pg_password='password'
```

Warning: Do not use special characters in `pg_password` as it may cause the setup to fail.

Example Inventory file for an external existing database

```
[tower]
node.example.com ansible_connection=local

[database]

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

pg_host='database.example.com'
pg_port='5432'

pg_database='awx'
pg_username='awx'
```

Warning: Do not use special characters in `pg_password` as it may cause the setup to fail.

Example Inventory file for external database which needs installation

```
[tower]
node.example.com ansible_connection=local

[database]
database.example.com

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

pg_host='database.example.com'
pg_port='5432'

pg_database='awx'
pg_username='awx'
```

Warning: Do not use special characters in `pg_password` as it may cause the setup to fail.

Once any necessary changes have been made, you are ready to run `./setup.sh`.

Note: Root access to the remote machines is required. With Ansible, this can be achieved in different ways:

- `ansible_user=root ansible_ssh_pass="your_password_here"` inventory host or group variables
- `ansible_user=root ansible_ssh_private_key_file="path_to_your_keyfile.pem"` inventory host or group variables

- `ANSIBLE_BECOME_METHOD='sudo' ANSIBLE_BECOME=True ./setup.sh`
- `ANSIBLE_SUDO=True ./setup.sh` (Only applies to Ansible 2.7)

The `DEFAULT_SUDO` Ansible configuration parameter was removed in Ansible 2.8, which causes the `ANSIBLE_SUDO=True ./setup.sh` method of privilege escalation to no longer work. For more information on become plugins, refer to [Understanding Privilege Escalation](#) and the [list of become plugins](#).

2.4 Running the Setup Playbook

The Tower setup playbook script uses the `inventory` file and is invoked as `./setup.sh` from the path where you unpacked the Tower installer tarball.

```
root@localhost:~$ ./setup.sh
```

The setup script takes the following arguments:

- `-h` – Show this help message and exit
- `-i INVENTORY_FILE` – Path to Ansible inventory file (default: `inventory`)
- `-e EXTRA_VARS` – Set additional Ansible variables as `key=value` or `YAML/JSON` (i.e. `-e bundle_install=false` forces an online installation)
- `-b` – Perform a database backup in lieu of installing
- `-r` – Perform a database restore in lieu of installing (a default restore path is used unless `EXTRA_VARS` are provided with a non-default path, as shown in the code example below)

```
./setup.sh -e 'restore_backup_file=/path/to/nondefault/location' -r
```

ROLE-BASED ACCESS CONTROLS

Ansible Tower 3.7 contains minor updates to the Role-Based Access Control (RBAC) system. For the latest RBAC documentation, refer to the [Role-Based Access Controls](#) section in the Tower User Guide.

3.1 Organization field on Job Templates

Job templates in Ansible Tower now include an organization field in the API. This is set on creation based on the organization of the project used by the Job Template, and cannot be changed. Because of this, a project's organization cannot be changed once it is in use by Job Templates.

This changes visibility and access to job templates. Previously, an admin of the organization that a job template's inventory belonged to would also be granted admin access to the job template. While existing permissions are preserved on an upgrade to Ansible Tower 3.7, newly created jobs will only grant view access to the job template to the inventory admin in this scenario.

USING VIRTUALENV WITH ANSIBLE TOWER

Virtualenv creates isolated Python environments to avoid problems caused by conflicting dependencies and differing versions. Virtualenv works by simply creating a folder which contains all of the necessary executables and dependencies for a specific version of Python. Ansible Tower creates two virtualenvs during installation—one is used to run Tower, while the other is used to run Ansible. This allows Tower to run in a stable environment, while allowing you to add or update modules to your Ansible Python environment as necessary to run your playbooks. For more information on virtualenv, see the Python Guide to [Virtual Environments](#) and the *Python virtualenv* project itself.

By default, the virtualenv is located at `/var/lib/awx/venv/ansible` on the file system but you can create your own custom directories and use them in inventory imports. This allows you to choose how you run your inventory imports, as inventory sources use custom virtual environments.

Tower also pre-installs a variety of third-party library/SDK support into this virtualenv for its integration points with a variety of cloud providers (such as EC2, OpenStack, Azure, etc.) Periodically, you may want to add additional SDK support into this virtualenv, which is described in further detail below.

Note: It is highly recommended that you run `umask 0022` before installing any packages to the virtual environment. Failure to properly configure permissions can result in Tower service failures. An example follows:

```
# source /var/lib/awx/venv/ansible/bin/activate
# umask 0022
# pip install --upgrade pywinrm
# deactivate
```

In addition to adding modules to the virtualenv that Tower uses to run Ansible, you can create new virtualenvs as described below.

4.1 Preparing a new custom virtualenv

You can specify a different virtualenv for running Job Templates in Tower. In order to do so, you must specify which directories those venvs reside. You could choose to keep custom venvs inside `/var/lib/awx/venv/`, but it is highly recommended that a custom directory be created. The following examples use a placeholder directory `/opt/my-envs/`, but you can replace this with a directory path of your choice anywhere this is specified.

1. Preparing a new custom virtualenv requires the virtualenv package to be pre-installed:

```
$ sudo yum install python-virtualenv
```

2. Create a directory for your custom venvs:

```
$ sudo mkdir /opt/my-envs
```

3. Make sure to give your directory the appropriate write permission, execution permission and ownership:

```
$ sudo chmod 0755 /opt/my-envs
$ sudo chown awx:awx /opt/my-envs
```

4. Optionally, you can specify in Tower which directory to look for custom venvs by adding this directory to the `CUSTOM_VENV_PATHS` setting as follows:

```
$ curl -X PATCH 'https://user:password@tower.example.org/api/v2/settings/system/' \
-d '{"CUSTOM_VENV_PATHS": ["/opt/my-envs/"]}' -H 'Content-Type:application/json'
```

If you have venvs spanned over multiple directories, add all the paths and Tower will aggregate venvs from them:

```
$ curl -X PATCH 'https://user:password@tower.example.org/api/v2/settings/system/' \
-d '{"CUSTOM_VENV_PATHS": ["/path/1/to/venv/", "/path/2/to/venv/", "/path/3/to/
↪venv/"]}' \
-H 'Content-Type:application/json'
```

5. Now that a venv directory has been set up, create a virtual environment in that location:

```
$ sudo virtualenv /opt/my-envs/custom-venv
```

Note: Multiple versions of Python are supported, but the syntax for creating virtualenvs in Python 3 has changed slightly: `$ sudo python3 -m venv /opt/my-envs/custom-venv`

6. Next, install `gcc` so that `psutil` can be compiled:

```
$ yum install gcc
```

7. Python header files are needed to compile `psutil`. The package needed to successfully compile `psutil` on RHEL 8 systems is `platform-python-devel`:

```
$ yum install platform-python-devel
```

8. Your newly created virtualenv needs a few base dependencies to properly run playbooks (eg., fact gathering):

```
$ sudo /opt/my-envs/custom-venv/bin/pip install psutil
```

From here, you can install *additional* Python dependencies that you care about, such as a per-virtualenv version of Ansible itself:

```
$ sudo /opt/my-envs/custom-venv/bin/pip install -U "ansible == X.Y.Z"
```

Or you can add an additional third-party SDK that is not included with the base Tower installation:

```
$ sudo /opt/my-envs/custom-venv/bin/pip install -U python-digitalocean
```

If you want to copy them, the libraries included in Tower's default virtualenv can be found using `pip freeze`:

```
$ sudo /var/lib/awx/venv/ansible/bin/pip freeze
```

In a clustered Tower installation, you need to ensure that the same custom virtualenv exists on **every** local file system at `/opt/my-envs/`. Custom virtualenvs are supported on isolated instances. If you are using a custom virtual

environment, it needs to also be copied or replicated on any isolated node you would be using, not just on the Tower node. For setting up custom virtual environments in containers, refer to the [OpenShift Deployment and Configuration](#) section of the *Ansible Tower Administration Guide*.

4.2 Assigning custom virtualenvs

Once you have created a custom virtualenv, you can assign it at the Organization, Project, or Job Template level to use it in job runs. You can set the custom venv on an inventory source to run inventory updates in that venv. Jobs using that inventory follow their own rules and will not use this venv. If an SCM inventory source does not have a venv selected, it can use the venv of its linked project. You can assign a custom venv on the organization, but if you do, it will not be used by inventory updates in the organization, as it is only used in job runs.

The following shows the proper way to assign a custom venv at the desired level.

```
PATCH https://awx-host.example.org/api/v2/organizations/N/
PATCH https://awx-host.example.org/api/v2/projects/N/
PATCH https://awx-host.example.org/api/v2/job_templates/N/
PATCH https://awx-host.example.org/api/v2/inventory_sources/N/

Content-Type: application/json
{
    'custom_virtualenv': '/opt/my-envs/custom-venv'
}
```

An HTTP GET request to `/api/v2/config/` provides a list of detected installed virtualenvs:

```
{
    "custom_virtualenvs": [
        "/opt/my-envs/custom-venv",
        "/opt/my-envs/my-other-custom-venv",
    ],
    ...
}
```

You can also specify the virtual environment to assign to an Organization, Project, and Job Template from their respective edit screens in the Ansible Tower User Interface. Select the virtualenv from the **Ansible Environment** drop-down menu, as shown in the example below:

ORGANIZATIONS / Default

Default

DETAILS USERS PERMISSIONS NOTIFICATIONS

* NAME DESCRIPTION

Default

ANSIBLE ENVIRONMENT ⓘ

Use Default Environment

Use Default Environment

/opt/my-envs-1/custom-venv-1

/opt/my-envs-1/custom-venv-2

/opt/my-envs-1/custom-venv-3

/opt/my-envs-2/custom-venv-4

/opt/my-envs-2/custom-venv-5

Q KEY

When you launch a job template, you will also see the virtualenv specified in the Job Details pane:

DETAILS 🚀 🗑️

STATUS	● Successful
STARTED	4/24/2019 12:07:36 PM
FINISHED	4/24/2019 12:07:43 PM
JOB TEMPLATE	Demo Job Template
JOB TYPE	Run
LAUNCHED BY	admin
INVENTORY	Inventory - CampDifference ⓘ
PROJECT	● Demo Project
REVISION	347e44f 📄
PLAYBOOK	hello_world.yml
CREDENTIAL	🔍 Demo Credential
ENVIRONMENT	<code>/var/lib/awx/venv/ansible</code>
EXECUTION NODE	localhost
INSTANCE GROUP	tower
EXTRA VARIABLES ⓘ	YAML JSON EXPAND

1	---
---	-----

- genindex

COPYRIGHT © RED HAT, INC.

Ansible, Ansible Tower, Red Hat, and Red Hat Enterprise Linux are trademarks of Red Hat, Inc., registered in the United States and other countries.

If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original version.

Third Party Rights

Ubuntu and Canonical are registered trademarks of Canonical Ltd.

The CentOS Project is copyright protected. The CentOS Marks are trademarks of Red Hat, Inc. (“Red Hat”).

Microsoft, Windows, Windows Azure, and Internet Explore are trademarks of Microsoft, Inc.

VMware is a registered trademark or trademark of VMware, Inc.

Amazon Web Services”, “AWS”, “Amazon EC2”, and “EC2”, are trademarks of Amazon Web Services, Inc. or its affiliates.

OpenStack™ and OpenStack logo are trademarks of OpenStack, LLC.

Chrome™ and Google Compute Engine™ service registered trademarks of Google Inc.

Safari® is a registered trademark of Apple, Inc.

Firefox® is a registered trademark of the Mozilla Foundation.

All other trademarks are the property of their respective owners.

INDEX

A

Ansible, executing in a virtual environment, 11

I

installation script
 inventory file setup, 4
 playbook setup, 9
inventory file setup, 4

P

permissions, 10
playbook setup, 9
 installation script, 9
 setup.sh, 9

R

RBAC, 10
roles, 10

S

setup.sh
 playbook setup, 9
singleton roles, 10
system-wide roles, 10

U

upgrade, 3
upgrade considerations, 3

V

virtual environment, 11