Ansible Tower Quick Install

Release Ansible Tower 3.6.4

Red Hat, Inc.

Nov 12, 2021

CONTENTS

Thank you for your interest in Red Hat 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.

The *Ansible Tower Quick Installation Guide* covers basic installation instructions for installing Ansible Tower on Red Hat Enterprise Linux and CentOS systems. This document has been updated to include information for the latest release of Ansible Tower 3.6.4.

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 3.6.4; April 21, 2020; https://access.redhat.com/

CHAPTER

ONE

PREPARING FOR THE TOWER INSTALLATION

This guide helps you get your Ansible Tower installation up and running as quickly as possible.

At the end of the installation, using your web browser, you can access and fully utilize Tower.

1.1 Installation and Reference Guide

While this guide covers the basics, you may find that you need the more detailed information available in the Installation and Reference Guide.

You should also review the General Installation Notes before starting the installation.

1.2 Prerequisites and Requirements

For platform information, refer to *Platform-specific Installation Notes*.

Note: Tower is a full application and the installation process installs several dependencies such as PostgreSQL, Django, NGINX, and others. It is required that you install Tower on a standalone VM or cloud instance and do not co-locate any other applications on that machine (beyond possible monitoring or logging software). Although Tower and Ansible are written in Python, they are not just simple Python libraries. Therefore, Tower cannot be installed in a Python virtualenv or any similar subsystem; you must install it as described in the installation instructions in this guide. For OpenShift-based deployments, refer to OpenShift Deployment and Configuration.

Ansible Tower has the following requirements:

- **Supported Operating Systems**:
  - Red Hat Enterprise Linux 8.0 or later 64-bit (x86) (only Ansible Tower 3.5 and greater can be installed)
  - Red Hat Enterprise Linux 7.4 or later 64-bit (x86)
  - CentOS 7.4 or later 64-bit (x86)

Note: Support for all versions of Ubuntu as a Tower platform has been discontinued as of Ansible Tower version 3.6.

- **A currently supported version of Mozilla Firefox or Google Chrome**
  - Other HTML5 compliant web browsers may work but are not fully tested or supported.
- **2 CPUs minimum** for Tower installations. Refer to the capacity algorithm section of the *Ansible Tower User Guide* for determining the CPU capacity required for the number of forks in your particular configuration.

- **4 GB RAM minimum** for Tower installations
  - 4 GB RAM (minimum and recommended for Vagrant trial installations)
  - 4 GB RAM (minimum for external standalone PostgreSQL databases)
  - For specific RAM needs, refer to the capacity algorithm section of the *Ansible Tower User Guide* for determining capacity required based on the number of forks in your particular configuration
- **20 GB of dedicated hard disk space** for Tower service nodes
  - 10 GB of the 20 GB requirement must be dedicated to `/var/`, where Tower stores its files and working directories
  - The storage volume should be rated for a minimum baseline of 750 IOPS.
- **20 GB of dedicated hard disk space** for nodes containing a database (*150 GB+ recommended*)
  - The storage volume should be rated for a high baseline IOPS (1000 or more.)
  - All Tower data is stored in the database. Database storage increases with the number of hosts managed, number of jobs run, number of facts stored in the fact cache, and number of tasks in any individual job. For example, a playbook run every hour (24 times a day) across 250, hosts, with 20 tasks will store over 800000 events in the database every week.
  - If not enough space is reserved in the database, old job runs and facts will need cleaned on a regular basis. Refer to Management Jobs in the *Ansible Tower Administration Guide* for more information
- **64-bit support required** (kernel and runtime)
- **PostgreSQL** version 10 required to run Ansible Tower 3.6 and later. Backup and restore will *only* work on PostgreSQL versions supported by your current Ansible Tower version. Note: upgrading from version 9.6 to 10.X will double the size of `/var/`.
- **Ansible version 2.7 (at minimum) required** to run Ansible Tower versions 3.6 and later

---

Note: You cannot use versions of PostgreSQL and Ansible older than those stated above and be able to run Ansible Tower 3.2 and later. Both are installed by the install script if they are not already present.

- **For Amazon EC2**:
  - Instance size of m4.large or larger
  - An instance size of m4.xlarge or larger if there are more than 100 hosts

1.2.1 Additional Notes on Tower Requirements

While other operating systems may technically function, currently only the above list is supported to host an Ansible Tower installation. If you have a firm requirement to run Tower on an unsupported operating system, please contact Ansible via the Red Hat Customer portal at https://access.redhat.com/. Management of other operating systems (nodes) is documented by the Ansible project itself and allows for a wider list.

Actual RAM requirements vary based on how many hosts Tower will manage simultaneously (which is controlled by the `forks` parameter in the job template or the system `ansible.cfg` file). To avoid possible resource conflicts, Ansible recommends 1 GB of memory per 10 forks + 2GB reservation for Tower, see the capacity algorithm for further details. If `forks` is set to 400, 40 GB of memory is recommended.

For the hosts on which we install Ansible Tower, Tower checks whether or not `umask` is set to 0022. If not, the setup fails. Be sure to set `umask=0022` to avoid encountering this error.

A larger number of hosts can of course be addressed, though if the fork number is less than the total host count, more passes across the hosts are required. These RAM limitations are avoided when using rolling updates or when using the provisioning callback system built into Tower, where each system requesting configuration enters a queue and is processed as quickly as possible; or in cases where Tower is producing or deploying images such as AMIs. All of these are great approaches to managing larger environments. For further questions, please contact Ansible via the Red Hat Customer portal at https://access.redhat.com/.

The requirements for systems managed by Tower are the same as for Ansible at: http://docs.ansible.com/intro_getting_started.html

Notable PostgreSQL Changes

Ansible Tower uses PostgreSQL 10, which is an SCL package on RHEL 7 and an app stream on RHEL8. Some changes worth noting when upgrading to PostgreSQL 10 are:

- PostgreSQL user passwords will now be hashed with SCRAM-SHA-256 secure hashing algorithm before storing in the database.
- You will no longer need to provide a `pg_hashed_password` in your inventory file at the time of installation because PostgreSQL 10 can now store the user's password more securely. If users supply a password in the inventory file for the installer (`pg_password`), that password will be SCRAM-SHA-256 hashed by PostgreSQL as part of the installation process. **DO NOT** use special characters in `pg_password` as it may cause the setup to fail.
- Since Tower is using a Software Collections version of PostgreSQL in Ansible Tower 3.6, the *rh-postgresql10* scl must be enabled in order to access the database. Administrators can use the `awx-manage dbshell` command, which will automatically enable the PostgreSQL SCL.
- If you just need to determine if your Tower instance has access to the database, you can do so with the command, `awx-manage check_db`.
- Upgrading from version 9.6 to 10.X will double the size of `/var/`.

PostgreSQL Configurations

Optionally, you can configure the PostgreSQL database as separate nodes that are not managed by the Tower installer. When the Tower installer manages the database server, it configures the server with defaults that are generally recommended for most workloads. However, you can adjust these PostgreSQL settings for standalone database server node where `ansible_memtotal_mb` is the total memory size of the database server:

```
max_connections == 1024
shared_buffers == ansible_memtotal_mb*0.3
work_mem == ansible_memtotal_mb*0.03
maintenance_work_mem == ansible_memtotal_mb*0.04
```

Refer to PostgreSQL documentation for more detail on tuning your PostgreSQL server.

1.2.2 Ansible Software Requirements

While Ansible Tower depends on Ansible Playbooks and requires the installation of the latest stable version of Ansible before installing Tower, manual installations of Ansible are no longer required.

Beginning with Ansible Tower version 2.3, the Tower installation program attempts to install Ansible as part of the installation process. Previously, Tower required manual installations of the Ansible software release package before running the Tower installation program. Now, Tower attempts to install the latest stable Ansible release package.

If performing a bundled Tower installation, the installation program attempts to install Ansible (and its dependencies) from the bundle for you (refer to *Using the Bundled Tower Installation Program* for more information).

If you choose to install Ansible on your own, the Tower installation program will detect that Ansible has been installed and will not attempt to reinstall it. Note that you must install Ansible using a package manager like `yum` and that the latest stable version must be installed for Ansible Tower to work properly. At minimum, Ansible version 2.2 is required for Ansible Tower versions 3.2 and later.

For convenience, summaries of those instructions are in the following sections.

1.2.3 Platform-specific Installation Notes

Installing Tower on Systems with FIPS Mode Enabled

Tower can run on systems where FIPS mode is enabled, though there are a few limitations to keep in mind:

- Only Enterprise Linux 7+ is supported. The standard python that ships with RHEL must be used for Ansible Tower to work in FIPS mode. Using any non-standard, non-system python for Tower is therefore, unsupported.
- By default, Tower configures PostgreSQL using password-based authentication, and this process relies on the usage of `md5` when `CREATE USER` is run at install time. To run the Tower installer from a FIPS-enabled system, specify `pg_password` in your inventory file. **DO NOT** use special characters in `pg_password` as it may cause the setup to fail:

  ```
  pg_password='choose-a-password'
  ```

  For further detail, see *Setting up the Inventory File*.

  If you supply a password in the inventory file for the installer (`pg_password`), that password will be SCRAM-SHA-256 hashed by PostgreSQL as part of the installation process.

- The `ssh-keygen` command generates keys in a format (RFC4716) which uses the `md5` digest algorithm at some point in the process (as part of a transformation performed on the input passphrase). On a FIPS-enforcing system, `md5` is completely disabled, so these types of encrypted SSH keys (RFC4716 private keys protected by a passphrase) will not be usable. When FIPS mode is enabled, any encrypted SSH key you import into Ansible Tower **must** be a `PKCS8`-formatted key. Existing `AES128` keys can be converted to `PKCS8` by running the following `openssl` command:

  ```
  $ openssl pkcs8 -topk8 -v2 aes128 -in <INPUT_KEY> -out <NEW_OUTPUT_KEY>
  ```

  For more details, see: https://access.redhat.com/solutions/1519083

- Use of Ansible features that use the `paramiko` library will not be FIPS compliant. This includes setting `ansible_connection=paramiko` as a transport and using network modules that utilize the `ncclient` NETCONF library.
- The TACACS+ protocol uses `md5` to obfuscate the content of authorization packets; TACACS+ Authentication is not supported for systems where FIPS mode is enabled.

- The RADIUS protocol uses `md5` to encrypt passwords in `Access-Request` queries; RADIUS Authentication is not supported for systems where FIPS mode is enabled.

Notes for Red Hat Enterprise Linux and CentOS setups

- In order for Ansible Tower to run on RHEL 8, Ansible 2.8 or greater must be installed. Ansible 2.8 and greater are supported versions for RHEL 8.
- Starting with Ansible Tower 3.5, Tower runs with Python 3, which is automatically installed on RHEL 8 when installing Tower.
- PackageKit can frequently interfere with the installation/update mechanism. Consider disabling or removing PackageKit if installed prior to running the setup process.
- Only the "targeted" SELinux policy is supported. The targeted policy can be set to disabled, permissive, or enforcing.
- When performing a bundled install, refer to *Using the Bundled Tower Installation Program* for more information.
- When installing Ansible Tower, you only need to run `setup.sh`, any repositories needed by Tower are installed automatically.
- The latest version of Ansible is installed automatically during the setup process. No additional installation or configuration is required.

Notes for Ubuntu setups

Ansible Tower no longer supports Ubuntu. Refer to previous versions of the *Ansible Tower Installation and Reference Guide* for details on Ubuntu.

Configuration and Installation on OpenShift

For OpenShift-based deployments, refer to OpenShift Deployment and Configuration.

1.3 Tower Installation Scenarios

Tower can be installed using one of the following scenarios:

Single Machine:

- **As an integrated installation:**
  - This is a single machine install of Tower - the web frontend, REST API backend, and database are all on a single machine. This is the standard installation of Tower. It also installs PostgreSQL from your OS vendor repository, and configures the Tower service to use that as its database.
- **With an external database (2 options available):**
  - Tower with remote DB configuration: This installs the Tower server on a single machine and configures it to talk to a remote instance of PostgreSQL 10 as its database. This remote PostgreSQL can be a server you manage, or can be provided by a cloud service such as Amazon RDS.
  - Tower with a playbook install of a remote PostgreSQL system: This installs the Tower server on a single machine and installs a remote PostgreSQL database via the playbook installer (managed by Tower).

Note: 1). Tower will not configure replication or failover for the database that it uses, although Tower should work with any replication that you have. 2). The database server should be on the same network or in the same datacenter as the Tower server for performance reasons.

Settings available for a traditional Tower install:

- `pg_sslmode` controls the SSL functions of the PostgreSQL client, i.e., how the Tower server connects to the database. It defaults to `prefer`, which means if the database server offers SSL, the client will use it. You can also set it to `verify-full` to enforce SSL with full verification of certificate trust.
- `web_server_ssl_cert` and `web_server_ssl_key` allow the user to provide a certificate and key to be installed in the web server for the Tower UI and API. These must either both be provided or both be absent. If they are absent, a self-signed (untrusted) certificate will be generated at install time.
- `postgres_use_ssl` (true/false) - controls whether the PostgreSQL server will be configured to require SSL. This only has any effect with an internal/embedded database (i.e. when the Tower install script is doing the deployment of the database server). It has no effect on an external database.
- `postgres_ssl_cert` and `postgres_ssl_key` - must be supplied when postgres_use_ssl is true. These certificates should have a CN (or wildcard, subject alternate name, and so forth) that matches the hostname the Tower nodes will use to connect to the database server.
- `rabbitmq_use_ssl` (true/false) - controls whether the RabbitMQ node-to-node communications will be encrypted. If this is set to true, then a single-use, "pinned" CA and server certificates will be generated by the install script. There is no need to supply certificates for RabbitMQ.

For OpenShift-based deployments, refer to OpenShift Deployment and Configuration.

High Availability Multi-Machine Cluster:

Tower can be installed in a high availability cluster mode. In this mode, multiple Tower nodes are installed and active. Any node can receive HTTP requests and all nodes can execute jobs.

- **A Clustered Tower setup must be installed with an external database (2 options available):**
  - Tower with remote DB configuration: This installs the Tower server on a single machine and configures it to talk to a remote instance of PostgreSQL as its database. This remote PostgreSQL can be a server you manage, or can be provided by a cloud service such as Amazon RDS.
  - Tower with a playbook install of a remote PostgreSQL system: This installs the Tower server on a single machine and installs a remote PostgreSQL database via the playbook installer (managed by Tower).
- For more information on configuring a clustered setup, refer to Clustering.

Note: Running in a cluster setup requires any database that Tower uses to be external–PostgreSQL must be installed on a machine that is not one of the primary or secondary tower nodes. When in a redundant setup, the remote PostgreSQL version requirements is *PostgreSQL 10*.

CHAPTER

TWO

DOWNLOAD THE TOWER INSTALLATION PROGRAM

Note: To obtain a trial version of Ansible Tower, visit: http://www.ansible.com/tower-trial

For pricing information, visit: http://www.ansible.com/pricing

To download the latest version of Tower directly (note, you must also obtain a license before using this), visit: https://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-latest.tar.gz

For the OpenShift installer, go to http://releases.ansible.com/ansible-tower/setup_openshift

You may install standalone Tower or use the bundled installer:

- if you set up Tower on an environment with a direct Internet access, you can download the standalone Tower installer
- if you set up Tower on an environment without direct access to online repositories, or your environment enforces a proxy, you must use the bundled installer

Download and then extract the Ansible Tower installation/upgrade tool: http://releases.ansible.com/ansible-tower/setup/

```
root@localhost:~$ tar xvzf ansible-tower-setup-latest.tar.gz
root@localhost:~$ cd ansible-tower-setup-<tower_version>
```

To install or upgrade, start by editing the inventory file in the `ansible-tower-setup-<tower_version>` directory, replacing `<tower_version>` with the version number, such as `3.6.4` or `3.6.0` directory.

2.1 Using the Bundled Tower Installation Program

Beginning in Ansible Tower version 2.3.0, Tower installations can be performed using a bundled installation program. The bundled installation program is meant for customers who cannot, or would prefer not to, install Tower (and its dependencies) from online repositories. Access to Red Hat Enterprise Linux or CentOS repositories is still needed.

To download the latest version of the bundled Tower installation program directly (note, you must also obtain a license before using this), visit: https://releases.ansible.com/ansible-tower/setup-bundle/

Note: The bundled installer only supports Red Hat Enterprise Linux and CentOS.

Next, select the installation program which matches your distribution (`el7`):

```
ansible-tower-setup-bundle-latest.el7.tar.gz
```

Note: On Red Hat Enterprise Linux 7, Ansible Tower requires the Python 3 Software Collection. If you are installing Tower offline, you need either CentOS-SCL or RH-SCL repositories enabled through a local mirror:

- Red Hat Subscription Manager: `rhel-server-rhscl-7-rpms`
- Red Hat UI: `rhui-rhel-server-rhui-rhscl-7-rpms`
- CentOS: `centos-release-scl`

A list of package dependencies from Red Hat Enterprise Linux repositories can be found in the `bundle/base_packages.txt` file inside the setup bundle. Depending on what minor version of Red Hat Enterprise Linux you are running, the version and release specified in that file may be slightly different than what is available in your configured repository.

2.2 Using Vagrant/Amazon AMI Images

One easy way to try Ansible Tower is to use a Vagrant box or an Amazon EC2 instance, and launching a trial of Ansible Tower just takes a few minutes.

If you use the Vagrant box or Amazon AMI Tower images provided by Ansible, you can find the auto-generated admin password by connecting to the image and reading it from the *message of the day* (MOTD) shown at login.

2.2.1 Vagrant

For Vagrant images, use the following commands to connect:

```
$ vagrant init ansible/tower
$ vagrant up --provider virtualbox
$ vagrant ssh
```

That last command provides your admin password and the Tower log-in URL. Upon login, you will receive directions on obtaining a trial license.

An up-to-date link to Ansible's Vagrant image is available from the LAUNCH TOWER IN VAGRANT section of Ansible's main website.

2.2.2 Amazon EC2

To launch the AMI, you must have an AMI ID (which varies based on you particular AWS region). A list of regions with links to AMI IDs is available in the LAUNCH TOWER IN AMAZON EC2 section of Ansible's main website.

To connect to Amazon AMI images, use the following command:

```
ssh centos@<your amazon instance>
```

You must use the SSH key that you configured the instance to accept at launch time.

CHAPTER

THREE

INSTALLING ANSIBLE TOWER

Tower can be installed in various ways by choosing the best mode for your environment and making any necessary modifications to the inventory file. For OpenShift-based deployments, refer to OpenShift Deployment and Configuration.

3.1 Tower Installation Scenarios

Tower can be installed using one of the following scenarios:

Single Machine:

- **As an integrated installation:**
  - This is a single machine install of Tower - the web frontend, REST API backend, and database are all on a single machine. This is the standard installation of Tower. It also installs PostgreSQL from your OS vendor repository, and configures the Tower service to use that as its database.
- **With an external database (2 options available):**
  - Tower with remote DB configuration: This installs the Tower server on a single machine and configures it to talk to a remote instance of PostgreSQL 10 as its database. This remote PostgreSQL can be a server you manage, or can be provided by a cloud service such as Amazon RDS.
  - Tower with a playbook install of a remote PostgreSQL system: This installs the Tower server on a single machine and installs a remote PostgreSQL database via the playbook installer (managed by Tower).

Note: 1). Tower will not configure replication or failover for the database that it uses, although Tower should work with any replication that you have. 2). The database server should be on the same network or in the same datacenter as the Tower server for performance reasons.

Settings available for a traditional Tower install:

- `pg_sslmode` controls the SSL functions of the PostgreSQL client, i.e., how the Tower server connects to the database. It defaults to `prefer`, which means if the database server offers SSL, the client will use it. You can also set it to `verify-full` to enforce SSL with full verification of certificate trust.
- `web_server_ssl_cert` and `web_server_ssl_key` allow the user to provide a certificate and key to be installed in the web server for the Tower UI and API. These must either both be provided or both be absent. If they are absent, a self-signed (untrusted) certificate will be generated at install time.
- `postgres_use_ssl` (true/false) - controls whether the PostgreSQL server will be configured to require SSL. This only has any effect with an internal/embedded database (i.e. when the Tower install script is doing the deployment of the database server). It has no effect on an external database.

- `postgres_ssl_cert` and `postgres_ssl_key` - must be supplied when postgres_use_ssl is true. These certificates should have a CN (or wildcard, subject alternate name, and so forth) that matches the hostname the Tower nodes will use to connect to the database server.
- `rabbitmq_use_ssl` (true/false) - controls whether the RabbitMQ node-to-node communications will be encrypted. If this is set to true, then a single-use, "pinned" CA and server certificates will be generated by the install script. There is no need to supply certificates for RabbitMQ.

For OpenShift-based deployments, refer to OpenShift Deployment and Configuration.

High Availability Multi-Machine Cluster:

Tower can be installed in a high availability cluster mode. In this mode, multiple Tower nodes are installed and active. Any node can receive HTTP requests and all nodes can execute jobs.

- **A Clustered Tower setup must be installed with an external database (2 options available):**
  - Tower with remote DB configuration: This installs the Tower server on a single machine and configures it to talk to a remote instance of PostgreSQL as its database. This remote PostgreSQL can be a server you manage, or can be provided by a cloud service such as Amazon RDS.
  - Tower with a playbook install of a remote PostgreSQL system: This installs the Tower server on a single machine and installs a remote PostgreSQL database via the playbook installer (managed by Tower).
- For more information on configuring a clustered setup, refer to Clustering.

Note: Running in a cluster setup requires any database that Tower uses to be external–PostgreSQL must be installed on a machine that is not one of the primary or secondary tower nodes. When in a redundant setup, the remote PostgreSQL version requirements is *PostgreSQL 10*.

3.2 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 **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

`rabbitmq_password=''` <—- create a new password here (alpha-numeric with no special characters)

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

Example Inventory file

- 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 Single Node Inventory File

```
[tower]
localhost ansible_connection=local

[database]
```

(continues on next page)

(continued from previous page)

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

pg_host=''
pg_port=''

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

rabbitmq_port=5672
rabbitmq_username=tower
rabbitmq_password='password'
rabbitmq_cookie=rabbitmqcookie

# Needs to be true for fqdns and ip addresses
rabbitmq_use_long_name=false
# Needs to remain false if you are using localhost
```

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'

rabbitmq_port=5672
rabbitmq_username=tower
rabbitmq_password=tower
rabbitmq_cookie=rabbitmqcookie

# Needs to be true for fqdns and ip addresses
rabbitmq_use_long_name=true
```

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'
rabbitmq_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'
rabbitmq_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.

3.3 The Setup Playbook

Note: Ansible Tower 3.0 simplifies installation and removes the need to run `./configure/` as part of the installation setup. Users of older versions should follow the instructions available in the v.2.4.5 (or earlier) releases of the Tower Documentation available at: http://docs.ansible.com/

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)

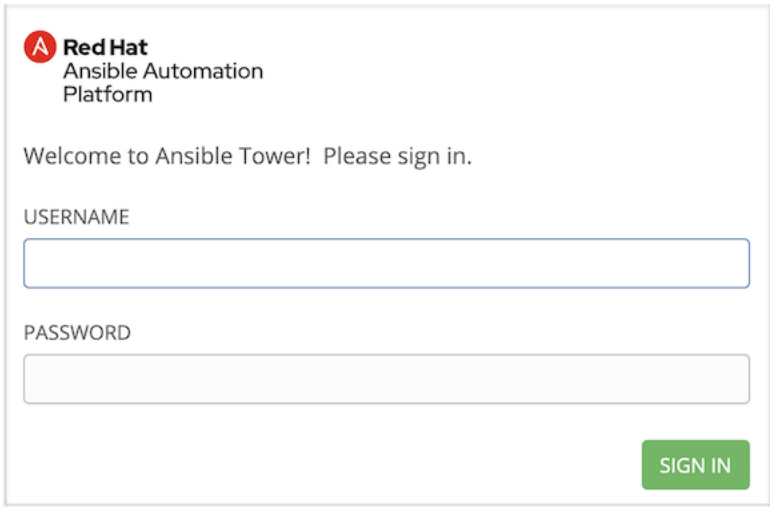
```
./setup.sh -e 'restore_backup_file=/path/to/nondefault/location' -r
```

Note: Please note that a issue was discovered in Tower 3.0.0 and 3.0.1 that prevented proper system backups and restorations.

If you need to back up or restore your Tower v3.0.0 or v3.0.1 installation, use the v3.0.2 installer to do so.

After calling `./setup.sh` with the appropriate parameters, Tower is installed on the appropriate machines as has been configured. Setup installs Tower from RPM packages using repositories hosted on **ansible.com**.

Once setup is complete, use your web browser to access the Tower server and view the Tower login screen. Your Tower server is accessible from port 80 (*https://<TOWER_SERVER_NAME>/*) but will redirect to port 443 so 443 needs to be available also.



If the installation of Tower fails and you are a customer who has purchased a valid license for Ansible Tower, please contact Ansible via the Red Hat Customer portal at https://access.redhat.com/.

3.4 Changing the Password

Once installed, if you log into the Tower instance via SSH, the default admin password is provided in the prompt. You can then change it with the following command (as root or as AWX user):

```
awx-manage changepassword admin
```

After that, the password you have entered will work as the admin password in the web UI.

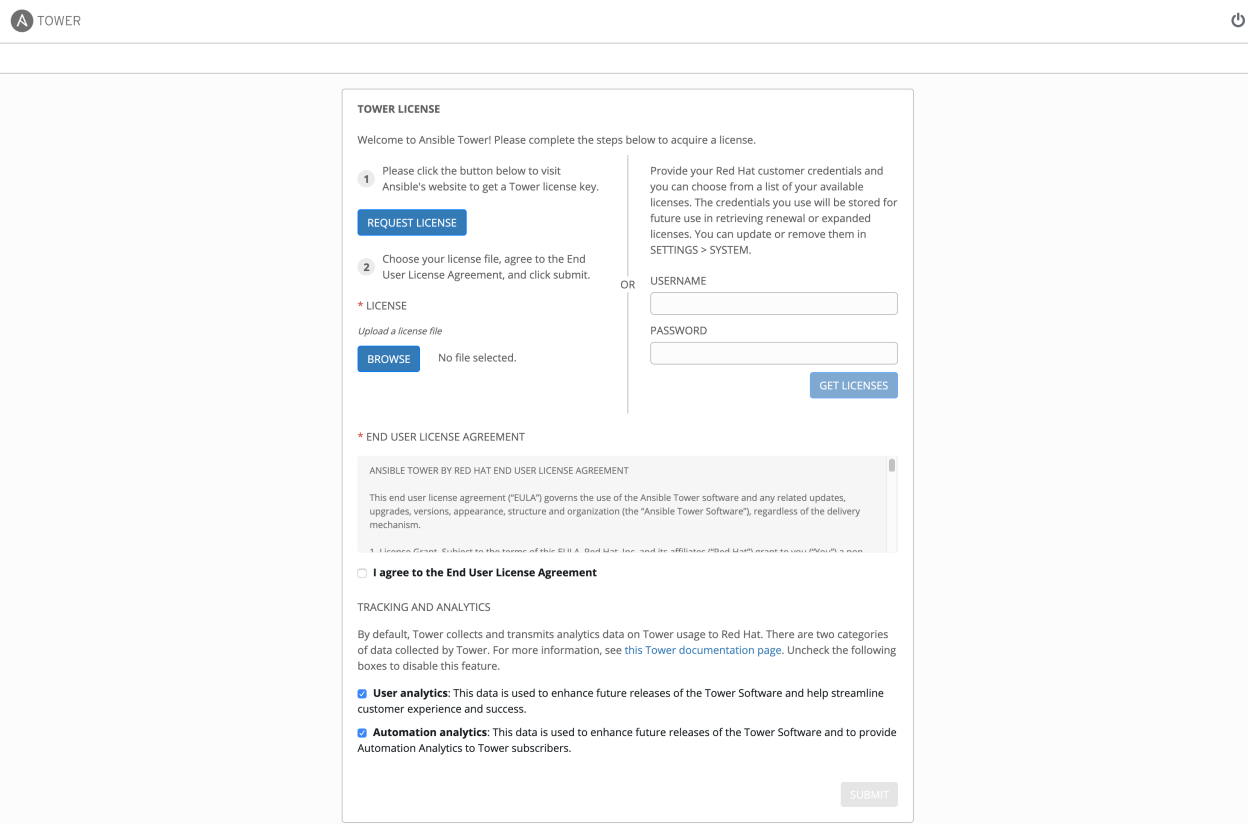
CHAPTER

FOUR

IMPORT A LICENSE

Tower requires a valid subscription to run. If you do not already have one, request one from the initial screen when you launch Tower. If you have issues with your subscription, contact Red Hat via the Red Hat Customer portal at https://access.redhat.com/.

Note: To successfully add your license, you must be logged on as the Superuser. Otherwise, the operation will fail.



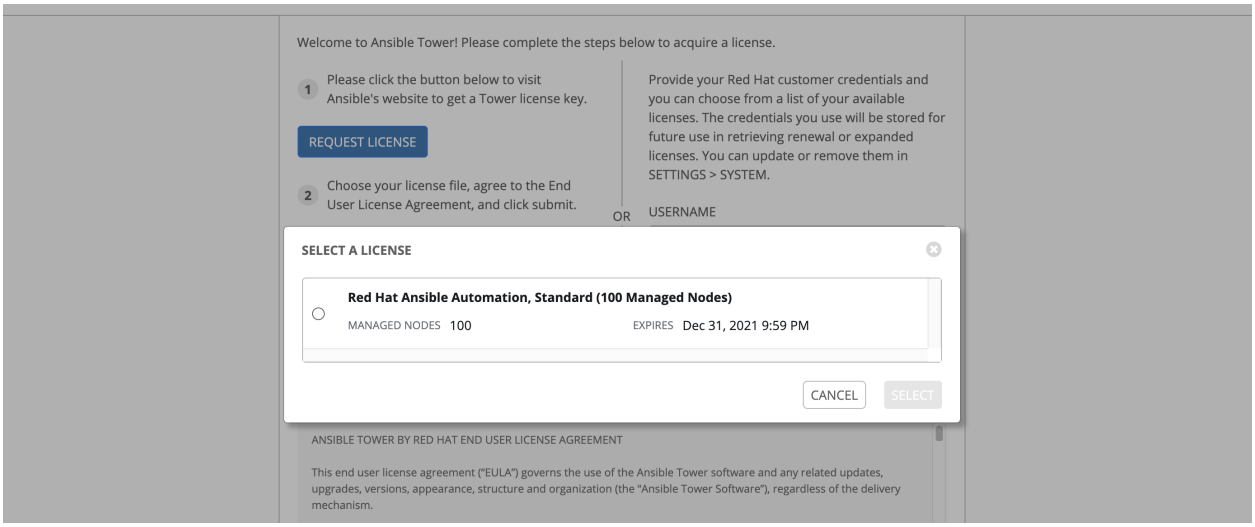
When Tower launches for the first time, the license screen automatically displays. Use your Red Hat credentials to retrieve and import your subscription, or import the license key you received from Red Hat:

1. Enter your Red Hat customer credentials on the right side of the screen. Alternatively, if you have a license file, click the **Browse** button and navigate to the location where the license file is saved to upload it. The uploaded license may be a plain text file or a JSON file, and must include properly formatted JSON code.

Note: The license import will work for Tower nodes with RHEL only. For non-RHEL (e.g. CentOS based Tower nodes), activate the license using the license file instead. When using Red Hat customer credentials, you must use your Red Hat username and password, not your Red Hat account email address and password.

2. If you entered your credential information, click **Get Licenses**.

Once your credential information (or license) is recognized, and you are on RHEL 7 and later, you will see a prompt with your Red Hat subscription(s). Choose the subscription you want to run (the example below has only one subscription). You can log in over time and retrieve new subscriptions if you have renewed.



Other non-RHEL subscribers, after uploading the license file, proceed by checking the **End User License Agreement**.

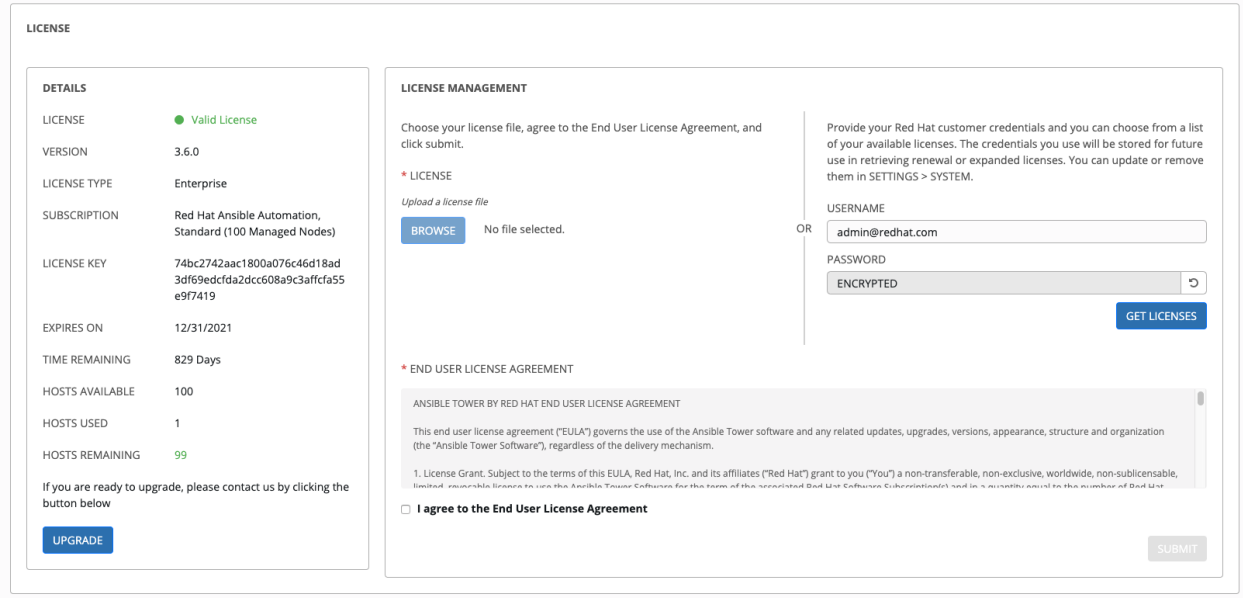
3. The bottom half of the license screen involves analytics data collection. This helps Red Hat improve the product by delivering you a much better user experience. For more information about data collection, refer to Usability Analytics and Data Collection. This option is checked by default, but you may opt out of any of the following:

- **User analytics** collects data from the Tower User Interface.
- **Automation analytics** provides a high level analysis of your automation with Ansible Tower, which is used to help you identify trends and anomalous use of Tower. For opt-in of Automation Analytics to have any effect, your instance of Ansible Tower **must** be running on Red Hat Enterprise Linux. See instructions described in the Automation Analytics section.

Note: At this time, Automation Insights is not supported when Ansible Tower is running in the OpenShift Container Platform. You may change your analytics data collection preferences at any time, as described in the Usability Analytics and Data Collection section.

4. After you have specified your tracking and analytics preferences, click **Submit**.

Once your license has been accepted, Tower briefly displays the license screen and navigates you to the Dashboard of the Ansible Tower interface (which you can access by clicking on the Ansible Tower logo at the top left of the screen as well).



For later reference, you can return to the license screen by clicking the Settings () icon from the left navigation bar and select the **License** tab from the Settings screen.

CHAPTER

FIVE

CONGRATULATIONS

Once the installation of Tower is complete, you are ready to set up and launch your first Ansible Playbook using Tower.

If you are wondering what to do next, refer to the following list of Ansible documentation sets for information on getting started, administration, and more:

- Ansible Tower Quick Setup Guide
- Ansible Tower Installation and Reference Guide
- Ansible Tower User Guide
- Ansible Tower Administration Guide
- Ansible Tower API Guide
- http://docs.ansible.com/

CHAPTER

SIX

INDEX

- genindex

CHAPTER

SEVEN

COPYRIGHT © 2020 RED HAT, INC.

INDEX

Symbols

A

B

C

D

E

I

L

M

O

P

R

S

T

V