
Ansible Tower Installation and Reference Guide

Release Ansible Tower 3.8.1

Red Hat, Inc.

Feb 10, 2023

CONTENTS

1	Tower Licensing, Updates, and Support	2
1.1	Support	2
1.2	Trial / Evaluation	2
1.3	Subscription Types	2
1.4	Node Counting in Licenses	3
1.5	Attaching Subscriptions	3
1.6	Tower Component Licenses	4
2	Release Notes	5
2.1	Ansible Tower Version 3.8.1	5
3	General Installation Notes	6
3.1	Flags and extra vars passed with Tower	7
3.2	Additional Installation Tips	9
4	Proxy Support	11
4.1	Configure Known Proxies	12
4.2	Reverse Proxy	13
4.3	Websocket Configuration	13
5	Requirements	15
5.1	Additional Notes on Automation Platform Requirements	16
5.2	Ansible Software Requirements	17
6	Obtain the Ansible Automation Platform Installation Program	19
6.1	Using the Bundled Ansible Automation Platform Installer	19
7	Installing Ansible Automation Platform	21
7.1	Ansible Automation Platform Installation Scenarios	21
7.2	Setting up the Inventory File	23
7.3	Playbook setup	28
7.4	Changing the Password	29
8	Upgrading an Existing Tower Installation	30
8.1	Requirements	31
8.2	Back Up Your Tower Installation	31
8.3	The Setup Playbook	31
9	Usability Analytics and Data Collection	32
10	Supported Inventory Plugin Templates	33

10.1	Amazon Web Services EC2	33
10.2	Google Compute Engine	35
10.3	Microsoft Azure Resource Manager	36
10.4	VMware vCenter	37
10.5	Red Hat Satellite 6	38
10.6	OpenStack	39
10.7	Red Hat Virtualization	39
10.8	Ansible Tower	39
11	Supported Attributes for Custom Notifications	40
12	Glossary	44
13	Index	47
14	Copyright © Red Hat, Inc.	48
	Index	49

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

The *Ansible Automation Platform Installation and Reference Guide* helps you to understand the installation requirements and processes behind installing the Ansible Automation Platform. This document has been updated to include information for the latest release of Ansible Tower v3.8.1.

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 3.8.1; January 13, 2021; <https://access.redhat.com/>

TOWER LICENSING, UPDATES, AND SUPPORT

Ansible Tower (“**Ansible Tower**”) is a software product provided as part of an annual Red Hat Ansible Automation Platform subscription entered into between you and Red Hat, Inc. (“**Red Hat**”).

Ansible is an open source software project and is licensed under the GNU General Public License version 3, as detailed in the Ansible source code: <https://github.com/ansible/ansible/blob/devel/COPYING>

Starting with Ansible Tower 3.8, you **must** have valid subscriptions attached before installing the Ansible Automation Platform. See *Attaching Subscriptions* for detail.

1.1 Support

Red Hat offers support to paid Red Hat Ansible Automation Platform customers.

If you or your company has purchased a subscription for Ansible Automation Platform, you can contact the support team at <https://access.redhat.com>. To better understand the levels of support which match your Ansible Automation Platform subscription, refer to *Subscription Types*. For details of what is covered under an Ansible Automation Platform subscription, please see the Scopes of Support at: <https://access.redhat.com/support/policy/updates/ansible-tower#scope-of-coverage-4> and <https://access.redhat.com/support/policy/updates/ansible-engine>.

1.2 Trial / Evaluation

While a license is required for Ansible Tower to run, there is no fee for a trial license.

- Trial licenses for Red Hat Ansible Automation are available at: <http://ansible.com/license>
- Support is not included in a trial license or during an evaluation of the Tower Software.

1.3 Subscription Types

Red Hat Ansible Automation Platform is provided at various levels of support and number of machines as an annual Subscription.

- **Standard**
 - Manage any size environment
 - Enterprise 8x5 support and SLA
 - Maintenance and upgrades included
 - Review the SLA at: <https://access.redhat.com/support/offerings/production/sla>

- Review the Red Hat Support Severity Level Definitions at: <https://access.redhat.com/support/policy/severity>

- **Premium**

- Manage any size environment, including mission-critical environments
- Premium 24x7 support and SLA
- Maintenance and upgrades included
- Review the SLA at: <https://access.redhat.com/support/offerings/production/sla>
- Review the Red Hat Support Severity Level Definitions at: <https://access.redhat.com/support/policy/severity>

All Subscription levels include regular updates and releases of Ansible Tower, Ansible, and any other components of the Platform.

For more information, contact Ansible via the Red Hat Customer portal at <https://access.redhat.com/> or at <http://www.ansible.com/contact-us/>.

1.4 Node Counting in Licenses

The Tower license defines the number of Managed Nodes that can be managed as part of a Red Hat Ansible Automation Platform subscription. A typical license will say ‘License Count: 500’, which sets the maximum number of Managed Nodes at 500.

For more information on managed node requirements for licensing, please see <https://access.redhat.com/articles/3331481>.

1.5 Attaching Subscriptions

Starting with Tower 3.8, you **must** have valid subscriptions attached before installing the Ansible Automation Platform. Attaching an Ansible Automation Platform subscription enables Automation Hub repositories. A valid subscription needs to be attached to the Automation Hub node only. Other nodes do not need to have a valid subscription/pool attached, even if the `[automationhub]` group is blank, given this is done at the `repos_el` role level and that this role is run on both `[tower]` and `[automationhub]` hosts.

Note: Attaching subscriptions is unnecessary if your Red Hat account enabled [Simple Content Access Mode](#). But you still need to register to RHSM or Satellite before installing the Ansible Automation Platform.

To find out the `pool_id` of your Ansible Automation Platform subscription:

```
#subscription-manager list --available --all | grep "Ansible Automation Platform" -B_
↪3 -A 6
```

The command returns the following:

```
Subscription Name: Red Hat Ansible Automation Platform, Premium (5000 Managed Nodes)
Provides: Red Hat Ansible Engine
Red Hat Single Sign-On
Red Hat Ansible Automation Platform
SKU: MCT3695
```

(continues on next page)

(continued from previous page)

```
Contract: *****  
Pool ID: *****  
Provides Management: No  
Available: 4999  
Suggested: 1
```

To attach this subscription:

```
#subscription-manager attach --pool=<pool_id>
```

If this is properly done, and all nodes have Red Hat Ansible Automation Platform attached, then it will find the Automation Hub repositories correctly.

To check whether the subscription was successfully attached:

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

To remove this subscription:

```
#subscription-manager remove --pool=<pool_id>
```

1.6 Tower Component Licenses

To view the license information for the components included within Ansible Tower, refer to `/usr/share/doc/ansible-tower-<version>/README` where `<version>` refers to the version of Ansible Tower you have installed.

To view a specific license, refer to `/usr/share/doc/ansible-tower-<version>/*.txt`, where `*` is replaced by the license file name to which you are referring.

RELEASE NOTES

The following list summarizes the additions, changes, and modifications which were made to Ansible Tower v3.8.1.

2.1 Ansible Tower Version 3.8.1

- Improved analytics collection to collect the playbook status for all hosts in a playbook run
- Updated nginx on RHEL 7 to address CVE-2019-20372
- Updated autobahn to address CVE-2020-35678
- Updated the installer to ensure Automation Hub repositories are only enabled while running the installer
- Updated the installer to allow it to pin a specific version of Automation Hub that needs to be installed
- Added aggregation support for applying multiple subscriptions to a single Tower installation
- Fixed the installer to only install the DB where it belongs and not on all nodes
- Fixed the installer to only check the RHSM Automation Hub repository when not using a bundled installer
- Fixed Tower to properly handle certain uploaded subscription manifests
- Fixed Tower to properly respect the configured destination port when interacting with Red Hat Satellite 6 to obtain licensing/entitlement data
- Fixed an error in the module documentation for the tower_license module
- Fixed inventory updates from Satellite 6 and Tower to no longer fail unexpectedly
- Fixed AWS inventory hosts to now properly track across inventory updates

For older version of the release notes, as well as other reference materials, refer to the [Ansible Tower Release Notes](#).

GENERAL INSTALLATION NOTES

Ansible Tower and Automation Hub have converged to offer a unified platform experience called, Ansible Automation Platform. Starting with Ansible Tower 3.8, what used to be the Tower installer can now install Automation Platform as a whole. Refer to *Installing Ansible Automation Platform* for further detail.

- Automation Platform **requires** Ansible 2.9. Ansible 2.9.x will be the only supported version installed on Tower 3.8.
- 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. Tower and Automation Hub can run on either RHEL 7 or 8, but only Tower (not Automation Hub) is supported on an OpenShift Container Platform (OCP).
- Starting with Ansible Tower 3.8, you **must** have valid subscriptions attached before installing and running the Ansible Automation Platform. **Even if you already have valid licenses from previous versions, you must still provide your credentials or a subscriptions manifest again upon upgrading to Tower 3.8.** A valid subscription needs to be attached to the Automation Hub node only. Other nodes do not need to have a valid subscription/pool attached. See *Attaching Subscriptions* for detail.
- For customers installing Tower behind Satellite, prepare your Tower nodes by installing the Katello RPM specific to your satellite instance. See *Installing Satellite instances on Tower* for more information.
- Newly created configurations for inventory sources will contain the default plugin configuration values. If you want your newly created inventory sources in 3.8 to match the output of a 3.7 source, you must apply a specific set of configuration values for that source. To ensure backward compatibility, Tower uses “templates” for each of these sources to force the output of inventory plugins into the legacy format. Refer to *Supported Inventory Plugin Templates* in the *Ansible Automation Platform Installation and Reference Guide* for each source and their respective templates to help you migrate to the new style inventory plugin output.
- If you need to access a HTTP proxy to install software from your OS vendor, ensure that the environment variable “HTTP_PROXY” is set accordingly before running `setup.sh`.
- The Tower installer creates a self-signed SSL certificate and keyfile at `/etc/tower/tower.cert` and `/etc/tower/tower.key` for HTTPS communication. These can be replaced after install with your own custom SSL certificates if you desire, but the filenames are required to be the same. See *Using custom certificates*.
- Installing Ansible Automation Platform automatically installs the necessary versions of Node.js to run the Tower User Interface.
- If using Ansible version 1.8 or later, ensure that fact caching using Redis is not enabled in `ansible.cfg` on the Tower machine.
- Note that the Tower installation must be run from an internet connected machine that can install software from trusted 3rd-party places such as Ansible’s software repository, and your OS vendor’s software repositories. In some cases, access to the Python Package Index (PyPI) is necessary as well. If you need to be able to install in a disconnected environment and the bundled installation program is not a solution for you (refer to *Using*

the Bundled Ansible Automation Platform Installer), please contact Ansible via the Red Hat Customer portal at <https://access.redhat.com/>.

- If installing Tower on OpenShift, refer to [OpenShift Deployment and Configuration](#).

3.1 Flags and extra vars passed with Tower

Flags and/or extra variables that you can use with the Ansible Tower installer include (but are not limited to) the following:

```
Usage: setup.sh [Options] [-- Ansible Options]

Options:
  -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 will force an online install

  -b                     Perform a database backup in lieu of installing
  -r                     Perform a database restore in lieu of installing
  -k                     Generate and distribute a new SECRET_KEY

  -h                     Show this help message and exit

Ansible Options:
  Additional options to be passed to ansible-playbook can be added following the --
  ↪separator
```

Use the `-- separator` to add any Ansible arguments you wish to apply. For example: `./setup.sh -i my_awesome_inventory.yml -e matburt_is_country_gold=True -- -K`

The following table shows some extra variables that can be used during the installation of Tower.

Variable	Description	Default
upgrade_ansible	When installing Tower make sure Ansible is also up to date	False
create_preload	When installing Tower also create the Demo Org, project, credential, Job Template, etc.	True
bundle_install	When installing from a bundle where to put the bundled repos	/var/lib/ tower-bundle
nginx_disable_https	Disable HTTPS traffic through nginx, this is useful if offloading HTTPS to a load balancer	False
nginx_disable_hsts	Disable HSTS web-security policy mechanism	False
nginx_http_port	Port to configure nginx to listen to for HTTP	80
nginx_https_port	Port to configure nginx to listen to for HTTPS	443
backup_dest	Where to place the backup from setup.sh -b	{{ playbook_dir }}
backup_dir	A temp location to use when backing up	/var/ backups/ tower/
restore_backup	Specify an alternative backup file to restore from	(None)
required_ram	The minimum RAM required to install Tower (should only be changed for test installation)	3750
min_open_fds	The minimum open file descriptions (should only be changed for test installations)	4096
ignore_preflight	Ignore preflight checks, useful when installing into a template or other non-system image (overrides required_ram and min_open_fds)	False

3.1.1 Examples

The following are examples of common scenarios - be sure to supply your own values appropriate to your specific case.

- **To upgrade core:**

```
./setup.sh -e upgrade_ansible_with_tower=1
```

- **To disable https handling at nginx:**

```
./setup.sh -e nginx_disable_https=true
```

- **To specify a non-default path when restoring from a backup file:**

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

- **To override an inventory file used by passing it as an argument to the setup script:**

```
setup.sh -i <inventory file>
```

3.1.2 Using custom certificates

You may bring your own certificates as part of the default install and therefore, not rely on the self-signed one provided. The Ansible Tower installer provides three variables that allows you to configure the Tower deployment TLS-wise:

web_server_ssl_path	Path on the installer node to the custom certificate the Tower web server will serve. It will be copied as <code>/etc/tower/tower.cert</code> at install time.
web_server_ssl_privkey	Path on the installer node to the private key the certificate has been generated with. It will be copied as <code>/etc/tower/tower.key</code> at install time.
custom_ca_cert	Custom Certification Authority to add as trustworthy in the system bundle. It will be loaded into the Tower CA trusted store.

3.2 Additional Installation Tips

You may want to consider how to configure your proxies and websockets as part of the installation of Ansible Tower to align the websocket config with your nginx / load balancer configuration. See *Proxy Support* in the following section of this guide for further detail.

3.2.1 Platform-specific Installation Notes

Installing Automation Platform on Systems with FIPS Mode Enabled

Ansible Automation Platform 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

- `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 Ansible Automation Platform Installer* 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 Automation Platform Installation and Reference Guide* for details on Ubuntu.

Configuration and Installation on OpenShift

For OpenShift-based deployments, refer to [OpenShift Deployment and Configuration](#).

Installing Satellite instances on Tower

Satellite users will need to install the Katello RPM for your Satellite instance on the Tower node prior to installing Tower. This RPM automatically configures Subscription Manager to use Satellite as its content source, and the `hostname` value gets updated in `/etc/rhsm/rhsm.conf`.

Note: If you were to install the Katello RPM *after* installing Tower, Tower would not have access to the `rhsm.conf`— which it relies on for applying a subscription from Satellite. This is because the Tower installer sets an ACL rule on the `rhsm.conf` file, therefore, a subscription is unable to be applied if that file does not exist, gets later overwritten, or the user does not have the right permissions to access the file.

For detail on how to register a host with a Satellite server, refer to the [Registration](#) section of the Satellite documentation.

PROXY SUPPORT

Proxy servers act as an intermediary for requests from clients seeking resources from other servers. A client connects to the proxy server, requesting some service or available resource from a different server, and the proxy server evaluates the request as a way to simplify and control its complexity.

Note: Using SSL offloading or using a proxy that handles SSL for Tower is supported. The proxy/load balancer needs to be configured to pass the remote host information.

When offloading SSL to the load balancer or proxy, set `nginx_disable_https=true` as an extra variable passed to the setup playbook. Refer to *Playbook setup* for information on applying extra variables to the setup playbook.

Sessions in Tower associate an IP address upon creation. Tower policy requires that any use of the session match the original associated IP address.

To provide proxy server support, Tower handles proxied requests (such as ALB, NLB , HAProxy, Squid, Nginx and tinyproxy in front of Tower) via the `REMOTE_HOST_HEADERS` list variable in Tower settings (`/etc/tower/conf.d/remote_host_headers.py`). By default `REMOTE_HOST_HEADERS` is set to `['REMOTE_ADDR', 'REMOTE_HOST']`.

To enable proxy server support, setup `REMOTE_HOST_HEADERS` like the following: `REMOTE_HOST_HEADERS = ['HTTP_X_FORWARDED_FOR', 'REMOTE_ADDR', 'REMOTE_HOST']`

Note: A new installation of Ansible Tower will not contain the `remote_host_headers.py` file. However, you can still set those values in the System settings of the Configure Tower user interface.

SETTINGS / SYSTEM

SYSTEM

MISC. SYSTEM ACTIVITY STREAM LOGGING

<p>* BASE URL OF THE TOWER HOST <small>?</small> REVERT</p> <input type="text" value="https://ec2-3-89-219-44.compute-1.amazonaws.com"/>	<p>* ALL USERS VISIBLE TO ORGANIZATION ADMINS <small>?</small></p> <input checked="" type="checkbox"/>	<p>* ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS <small>?</small></p> <input checked="" type="checkbox"/>
<p>* IDLE TIME FORCE LOG OUT <small>?</small> REVERT</p> <input type="text" value="1800"/>	<p>* MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS <small>?</small> REVERT</p> <input type="text" value="-1"/>	<p>* ENABLE HTTP BASIC AUTH <small>?</small></p> <input checked="" type="checkbox"/>
<p>ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS <small>?</small></p> <input type="checkbox"/>	<p>LOGIN REDIRECT OVERRIDE URL <small>?</small> REVERT</p> <input type="text"/>	<p>ACCESS TOKEN EXPIRATION <small>?</small> REVERT</p> <input type="text" value="3153600000"/>
<p>REFRESH TOKEN EXPIRATION <small>?</small> REVERT</p> <input type="text" value="2628000"/>	<p>AUTHORIZATION CODE EXPIRATION <small>?</small> REVERT</p> <input type="text" value="600"/>	<p>* REMOTE HOST HEADERS <small>?</small> REVERT</p> <input type="text" value="REMOTE_ADDR, REMOTE_HOST"/>
<p>CUSTOM VIRTUAL ENVIRONMENT PATHS <small>?</small> REVERT</p> <input type="text"/>	<p>GATHER DATA FOR AUTOMATION ANALYTICS <small>?</small></p> <input checked="" type="checkbox"/>	<p>RED HAT CUSTOMER USERNAME <small>?</small> REVERT</p> <input type="text"/>
<p>RED HAT CUSTOMER PASSWORD <small>?</small> REVERT</p> <input type="text" value="SHOW"/>	<p>AUTOMATION ANALYTICS UPLOAD URL REVERT</p> <input type="text" value="https://cloud.redhat.com/api/ingress/v1/upload"/>	<p>AUTOMATION ANALYTICS GATHER INTERVAL <small>?</small> REVERT</p> <input type="text" value="14400"/>

REVERT ALL TO DEFAULT

Tower determines the remote host’s IP address by searching through the list of headers in REMOTE_HOST_HEADERS until the FIRST IP address is located.

Note: Header names are constructed using the following logic:

With the exception of CONTENT_LENGTH and CONTENT_TYPE, any HTTP headers in the request are converted to META keys by converting all characters to uppercase, replacing any hyphens with underscores, and adding an HTTP_ prefix to the name. For example, a header called X-Barkley would be mapped to the META key HTTP_X_Barkley.

For more information on HTTP request and response objects, refer to: <https://docs.djangoproject.com/en/1.8/ref/request-response/#django.http.HttpRequest.META>

Note: If using SSL termination at the load balancer and forwarding traffic to a different port on the tower node (443 -> 80), set the following values in the /etc/tower/conf.d/custom.py file accordingly:

```
USE_X_FORWARDED_PORT = True
USE_X_FORWARDED_HOST = True
```

4.1 Configure Known Proxies

When Tower is configured with REMOTE_HOST_HEADERS = ['HTTP_X_FORWARDED_FOR', 'REMOTE_ADDR', 'REMOTE_HOST'], it assumes that the value of X-Forwarded-For has originated from the proxy/load balancer sitting in front of Tower. In a scenario where Tower is still reachable without use of the proxy/load balancer or when the proxy does not validate the header, X-Forwarded-For can be spoofed fairly easily to fake the originating IP addresses. Using HTTP_X_FORWARDED_FOR in the REMOTE_HOST_HEADERS setting poses a vulnerability that essentially gives users access to certain resources that they should not have.

To avoid this, you can configure a list of “known proxies” that are allowed, which is the PROXY_IP_ALLOWED_LIST setting via the settings API. Load balancers and hosts that are not on the list will result in a rejected request.

`PROXY_IP_ALLOWED_LIST` only works if the proxies in the list are properly sanitizing header input and correctly setting an `X-Forwarded-For` value equal to the real source IP of the client; the crux of this setting is that Tower can rely on the IPs/hostnames in `PROXY_IP_ALLOWED_LIST` to provide non-spoofed values for the `X-Forwarded-For` field.

`HTTP_X_FORWARDED_FOR` should **never** be configured as an item in `REMOTE_HOST_HEADERS` unless all of the following are satisfied:

- You are using a proxied environment w/ ssl termination
- The proxy provides sanitization/validation of the `X-Forwarded-For` header to prevent client spoofing
- `/etc/tower/conf.d/remote_host_headers.py` defines `PROXY_IP_ALLOWED_LIST` that contains only the originating IP of trusted proxies/load balancers.

Note: If you do not need all of the traffic to be put through the proxy, then you can specify the IP scheme(s) that you want to exclude in the `no_proxy` field. The list can be IP ranges or individual IPs, separated by a comma. This example shows a specified range of IPs in JSON format:

```
"https_proxy": "example.proxy.com:8080",  
"http_proxy": "example.proxy.com:8080",  
"no_proxy": "10.0.0.0/8"
```

Also, an SCM update with a `no_proxy` configuration and a CIDR notation may not work for a particular SCM. The support for `http_proxy` and `no_proxy` depends on the the implementation in the application (gitlhlsvn). For example, `git` does not support CIDR notation for `no_proxy` because `git` is limited by its C library: https://curl.haxx.se/libcurl/c/CURLOPT_NOPROXY.html.

4.2 Reverse Proxy

If you are behind a reverse proxy, you may want to setup a header field for `HTTP_X_FORWARDED_FOR`. The `X-Forwarded-For` (XFF) HTTP header field identifies the originating IP address of a client connecting to a web server through an HTTP proxy or load balancer.

```
REMOTE_HOST_HEADERS = ['HTTP_X_FORWARDED_FOR', 'REMOTE_ADDR', 'REMOTE_HOST']
```

4.3 Websocket Configuration

A key configuration that you should consider is websocket setup in order to align the websocket config with your `nginx` / load balancer configuration.

Tower nodes connect to all other Tower nodes via websockets. This interconnect is used to distribute all websocket emitted messages to all other Tower nodes. This is required because any browser client websocket can subscribe to any job that may be running on any Tower node. In other words, websocket clients are not routed to specific Tower nodes. Any Tower node can handle any websocket request. Therefore, each Tower node must know about all websocket messages destined for all clients.

Tower will automatically handle discovery of other Tower nodes via the Instance record in the database. Tower *must* be told the port, protocol, and whether or not to verify certificates when establishing the websocket connections. This is configured using the following three settings:


```
BROADCAST_WEBSOCKET_PROTOCOL = 'http'  
BROADCAST_WEBSOCKET_PORT = 80  
BROADCAST_WEBSOCKET_VERIFY_CERT = False
```

For example, topologies that do SSL termination in front of Tower, at the load balancer level would have the following settings:

```
BROADCAST_WEBSOCKET_PROTOCOL = 'http'  
BROADCAST_WEBSOCKET_PORT = 80
```

Topologies that perform SSL at each individual Tower node via nginx would have the following settings:

```
BROADCAST_WEBSOCKET_PROTOCOL = 'https'  
BROADCAST_WEBSOCKET_PORT = 443  
BROADCAST_WEBSOCKET_VERIFY_CERT = True
```

Note: It is intended that your nodes are broadcasting websocket traffic across a private, trusted subnet (and not the open Internet). Therefore, if you turn off HTTPS for websocket broadcasting, the websocket traffic (which is comprised mostly of Ansible playbook stdout), is sent between Tower nodes unencrypted.

REQUIREMENTS

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. Also, **DO NOT** change the default `alternative` for Python 3.

For OpenShift-based deployments, refer to [OpenShift Deployment and Configuration](#).

Ansible Tower has the following requirements:

Starting with Ansible Tower 3.8, you **must** have valid subscriptions attached before installing and running the Ansible Automation Platform. **Even if you already have valid licenses from previous versions, you must still provide your credentials or a subscriptions manifest again upon upgrading to Tower 3.8.** See [Attaching Subscriptions](#) for detail.

- **Supported Operating Systems:**
 - Red Hat Enterprise Linux 8.2 or later 64-bit (x86)
 - Red Hat Enterprise Linux 7.7 or later 64-bit (x86)
 - CentOS 7.7 or later 64-bit (x86)

Note: For Automation Hub, selinux-policy package version greater or equal to `3.13.1-268.el7_9.2` is required. If your setup has `rhel-7-server-rpms` repository enabled, the `_9.2` version will be pulled automatically along with Automation Hub. Also, RHUI subscriptions **cannot** be used to install Automation Hub.

Note: The next major release of Ansible Tower will not support Red Hat Enterprise Linux 7 or CentOS (any version) as an installation platform.

- **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 Automation Platform 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 Automation Platform 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 3.7 and later. Backup and restore will *only* work on PostgreSQL versions supported by your current Ansible Tower version.
- **Ansible version 2.9 required** to run Ansible Tower versions 3.8 and later

Note: You cannot use versions of PostgreSQL and Ansible older than those stated above and be able to run Ansible Tower 3.7 and later. Both are installed by the install script if they aren't already present.

- **For Automation Hub:** 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. Tower and Automation Hub can run on either RHEL 7 or 8, but only Tower (not Automation Hub) is supported on an OpenShift Container Platform (OCP).
- **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

5.1 Additional Notes on Automation Platform Requirements

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 Ansible Automation Platform are the same as for Ansible at: http://docs.ansible.com/intro_getting_started.html

5.1.1 Notable PostgreSQL Changes

Automation Platform 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 Ansible Tower and Automation Hub are using a Software Collections version of PostgreSQL in 3.8, 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`.

5.1.2 PostgreSQL Configurations

Optionally, you can configure the PostgreSQL database as separate nodes that are not managed by the Automation Platform installer. When the Automation Platform 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.

5.2 Ansible Software Requirements

While Automation Platform 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.

Upon new installations, Tower installs the latest release package of Ansible 2.9.

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

If you choose to install Ansible on your own, the Automation Platform 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 Automation Platform to work properly. Ansible version 2.9 is required for Ansible Tower versions 3.8 and later.

OBTAIN THE ANSIBLE AUTOMATION PLATFORM INSTALLATION PROGRAM

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. Tower and Automation Hub can run on either RHEL 7 or 8, but only Tower (not Automation Hub) is supported on an OpenShift Container Platform (OCP). Before obtaining the Automation Hub installer, determine how you will be installing your environment.

Both installers have Ansible Tower and Automation Hub; and provide the exact same feature set.

- Installing *with* internet access. Download the latest version of the online installer directly from <https://releases.ansible.com/ansible-tower/setup/ansible-tower-setup-latest.tar.gz> and then extract the installation/upgrade tool:

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

Installing with internet access will retrieve the latest required repositories, packages, and dependencies. You must have a valid subscription to activate the Ansible Automation Platform programs.

After installation or upgrade, start by editing the inventory file in the `ansible-tower-setup-<tower_version>` directory, replacing `<tower_version>` with the version number, i.e., `3.8.0`.

- Installing **without** internet. Download the *bundled installer*. Use this method when you do not have direct access to online repositories, or your environment enforces a proxy.

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>

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

6.1 Using the Bundled Ansible Automation Platform Installer

The bundled Ansible Automation Platform installer is meant for customers who are unable to access the internet, or would prefer not to install separate components (and its dependencies) from online repositories. Access to Red Hat Enterprise Linux or CentOS repositories is still needed. All other dependencies are included within the tar archive.

1. Access the latest version of the bundled Ansible Automation Platform installation program directly from <https://access.redhat.com/downloads/content/480> (note, you must have a Red Hat customer account to access the downloads).

Note: The Ansible Automation Platform installer only supports Red Hat Enterprise Linux and CentOS.

2. Next, select the latest installation program to download. The single `.tar.gz` works with both RHEL 7 and RHEL 8 distributions:

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

Note: On Red Hat Enterprise Linux 7, Ansible Tower 3.8.0 requires some packages from RHSCCL repo. 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-rhsccl-7-rpms`
- Red Hat UI: `rhui-rhel-server-rhui-rhsccl-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.

3. On your chosen machine, extract the installation/upgrade tool:

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

4. Upon installation or upgrade, start by editing the inventory file in the `ansible-tower-setup-<tower_version>` directory, replacing `<tower_version>` with the version number, i.e., `3.8.0`.

INSTALLING ANSIBLE AUTOMATION PLATFORM

Ansible Automation Platform 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, you can only deploy Tower on OpenShift. Deployment of Automation Hub on OpenShift is not supported. However, you can deploy Automation Hub in a virtual environment that points to OpenShift. For more detail, refer to [OpenShift Deployment and Configuration](#).

Note: A database server will still be installed even if a node does not have a database. It is a known issue and will be addressed in a future release.

7.1 Ansible Automation Platform Installation Scenarios

Ansible Automation Platform can be installed using one of the following scenarios:

Single Machine:

- Standalone Tower with database on the same node as Tower or non-installer managed database. 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.

```
[tower]
host
```

- Standalone Tower with an external managed database. 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]
host

[database]
host2
```

- Standalone Automation Hub with a database on the same node as Automation Hub or non-installer managed database:

```
[automationhub]
host
```


- Standalone Automation Hub with an external managed database. This installs the Automation Hub server on a single machine and installs a remote PostgreSQL database via the playbook installer (managed by Automation Platform Installer).

```
[automationhub]
host

[database]
host2
```

Platform Installation:

Platform installation involves Tower and Automation Hub. The Platform installer allows you to deploy 1 and only 1 Automation Hub per inventory. Given the installer can be used as an Automation Hub standalone installer, you can run the installer any number of times with any number of different inventories if you want to deploy multiple Automation Hub nodes. The 2 options supported for the Platform installation are:

- Platform (Tower + Automation Hub) with a database on the same node as Tower or non-installer managed database:

```
[tower]
host1

[automationhub]
host2
```

- Platform (Tower + Automation Hub) with an external managed database:

```
[tower]
host1

[automationhub]
host2

[database]
host3
```

Multi-Machine Cluster

This scenario involves Platform (Clustered Tower + Automation Hub) installation with an external managed database. In this mode, multiple Tower nodes are installed and active. Any node can receive HTTP requests and all nodes can execute jobs. This installs the Platform 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]
host1
host11
host12

[automationhub]
host2

[database]
host3
```

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*.

For more information on configuring a clustered setup, refer to [Clustering](#).

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. 3). Tower and Automation Hub can not run on the same node, this is a scenario that is not supported. It means that any deployment of the Platform becomes at least a 2-node deployment topology.

Settings available for an Automation Platform install:

- `automationhub_importer_settings`: Dictionary of settings/configuration to pass to `galaxy-importer`. It will end up in `/etc/galaxy-importer/galaxy-importer.cfg`
- `automationhub_require_content_approval`: Whether or not Automation Hub enforces the approval mechanism before collections are made available
- `automationhub_disable_https`: Whether or not Automation Hub should be deployed with TLS enabled
- `automationhub_disable_hsts`: Whether or not Automation Hub should be deployed with the HTTP Strict Transport Security (HSTS) web-security policy mechanism enabled
- `automationhub_ssl_validate_certs`: Whether or not Automation Hub should validate certificate when requesting itself (default = `False`) because by default, Platform deploys with self-signed certificates
- `automationhub_ssl_cert`: Same as `web_server_ssl_cert` but for Automation Hub UI and API
- `automationhub_ssl_key`: Same as `web_server_ssl_key` but for Automation Hub UI and API
- `automationhub_backup_collections`: Automation Hub provides artifacts in `/var/lib/pulp`. By default, this is set to `true` so Tower automatically backs up the artifacts by default. If a partition (e.g., LVM, NFS, CephFS, etc.) was mounted there, an enterprise organization would ensure it is always backed up. If this is the case, you can set `automationhub_backup_collections = false` and the backup/restore process will not have to backup/restore `/var/lib/pulp`.

For OpenShift-based deployments, refer to [OpenShift Deployment and Configuration](#).

7.2 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 **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.

7.2.1 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
# 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.cert
# 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
#
```

(continues on next page)

(continued from previous page)

```

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.
# 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](#).

7.3 Playbook setup

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:

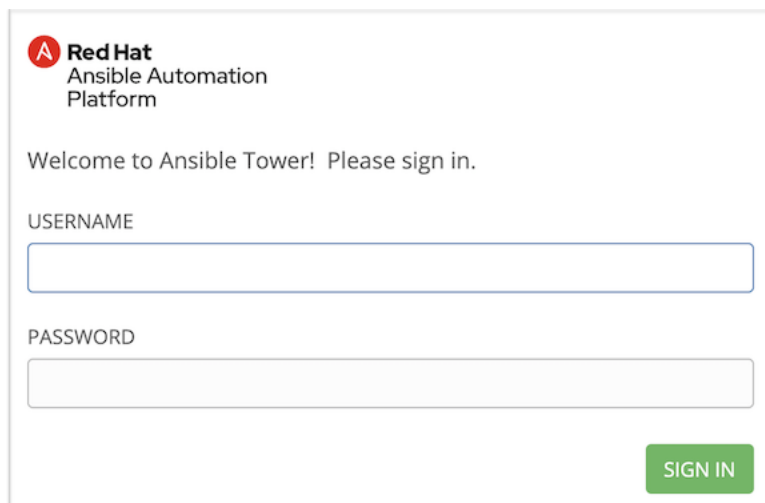
- `-h` – Show this help message and exit
- `-i INVENTORY_FILE` – Path to Ansible inventory file (default: `inventory`)
- `-e EXTRA_VARS` – Set additional Ansible variables as `key=value` or `YAML/JSON` (i.e. `-e bundle_install=false` forces an online installation)

- `-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
```

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.



Red Hat
Ansible Automation
Platform

Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

SIGN IN

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/>.

7.4 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.

UPGRADING AN EXISTING TOWER INSTALLATION

You can upgrade your existing Tower installation to the latest version easily. Tower looks for existing configuration files and recognizes when an upgrade should be performed instead of an installation.

As with installation, the upgrade process requires that the Tower server be able to access the Internet. The upgrade process takes roughly the same amount of time as a Tower installation, plus any time needed for data migration.

This upgrade procedure assumes that you have a working installation of Ansible and Tower.

Note: You can not convert an embedded-database Tower to a Clustered installation as part of an upgrade. Users who want to deploy Tower in a Clustered configuration should back up their Tower database, install a new Clustered configuration on a different VM or physical host, and then restore the database. It is possible to add additional instances later on to Tower if it is already operating on an external database. Refer to the [Clustering](#) chapter of the *Ansible Tower Administration Guide*.

If Tower is on a version of RHEL older than RHEL 8 and you want to upgrade to Ansible Tower on RHEL 8, follow the sequence outlined below:

1. *Obtain the Ansible Automation Platform Installation Program* and upgrade to Ansible Tower 3.8 on RHEL 7
2. Run Tower Backup included in the Tower setup playbook. See [Backing Up and Restoring Tower](#) in the *Ansible Tower Administration Guide* for details.
3. *Obtain the Ansible Automation Platform Installation Program* and install a fresh version of Ansible Tower 3.8 on RHEL 8
4. Run Tower Restore included in the Tower setup playbook. See [Backing Up and Restoring Tower](#) in the *Ansible Tower Administration Guide* for details.

This process ensures that the PostgreSQL database is also properly migrated to the latest version if you are upgrading an embedded-database Tower. Depending on the size of your Ansible Automation Platform installation, this may take some time. Note that if you upgrade a Tower with an external database, the client libraries will be upgraded as well, but you will need to upgrade your external PostgreSQL server manually. Be sure to check the release notes to see if this applies to you before upgrading.

If the upgrade of Tower fails or if you need assistance, please contact Ansible via the Red Hat Customer portal at <https://access.redhat.com/>.

8.1 Requirements

Before upgrading your Tower installation, refer to *Requirements* to ensure you have enough disk space and RAM as well as to review any software needs. For example, you should have the latest stable release of Ansible installed before performing an upgrade.

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 3.6.x, you must first be on version 3.4.x; i.e., there is no direct upgrade path from version 3.3.x. Refer to the [recommended upgrade path article](#) off your customer portal.

In order to run Ansible Tower 3.8 on RHEL 8, you must also have Ansible 2.9 or later installed.

Subscriptions from Satellite can be applied to a Tower instance by providing a Satellite username and password with access to subscriptions. To take advantage of this, you must register your Tower node with Satellite by installing the Katello RPM *before* upgrading to Tower 3.8.x. See *Installing Satellite instances on Tower* for more information.

8.2 Back Up Your Tower Installation

It is advised that you create a backup before upgrading the system. After the backup process has been accomplished, proceed with OS/Ansible/Tower upgrades.

Refer to [Backing Up and Restoring Tower](#) in the *Ansible Tower Administration Guide*.

8.3 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
```

USABILITY ANALYTICS AND DATA COLLECTION

Usability data collection is included with Tower to collect data to better understand how Tower users specifically interact with Tower, to help enhance future releases, and to continue streamlining your user experience.

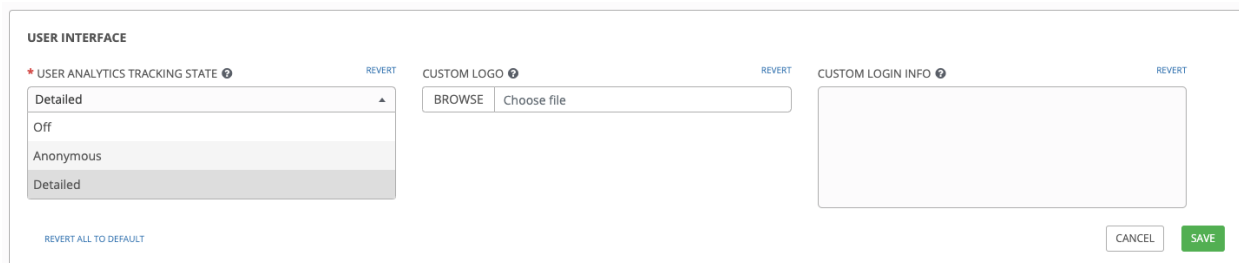
Only users installing a trial of Tower or a fresh installation of Tower are opted-in for this data collection.

If you want to change how you participate in this analytics collection, you can opt out or change your settings using



the Configure Tower user interface, accessible from the Settings () icon from the left navigation bar.

Ansible Tower collects user data automatically to help improve the Tower product. You can control the way Tower collects data by setting your participation level in the **User Interface** tab in the settings menu.



1. Select the desired level of data collection from the User Analytics Tracking State drop-down list:

- **Off:** Prevents any data collection.
- **Anonymous:** Enables data collection without your specific user data.
- **Detailed:** Enables data collection including your specific user data.

2. Click **Save** to apply the settings or **Cancel** to abandon the changes.

For more information, see the Red Hat privacy policy at <https://www.redhat.com/en/about/privacy-policy>.

SUPPORTED INVENTORY PLUGIN TEMPLATES

Configuring inventory plugins in Ansible Tower 3.8 is different from previous versions. On upgrade, existing configurations will be migrated to the new format that will produce a backwards compatible inventory output. Use the templates below to help aid in migrating your inventories to the new style inventory plugin output.

- *Amazon Web Services EC2*
- *Google Compute Engine*
- *Microsoft Azure Resource Manager*
- *VMware vCenter*
- *Red Hat Satellite 6*
- *OpenStack*
- *Red Hat Virtualization*
- *Ansible Tower*

10.1 Amazon Web Services EC2

```
compose:
  ansible_host: public_ip_address
  ec2_account_id: owner_id
  ec2_ami_launch_index: ami_launch_index | string
  ec2_architecture: architecture
  ec2_block_devices: dict(block_device_mappings | map(attribute='device_name') | list,
↪| zip(block_device_mappings | map(attribute='ebs.volume_id') | list))
  ec2_client_token: client_token
  ec2_dns_name: public_dns_name
  ec2_ebs_optimized: ebs_optimized
  ec2_eventsSet: events | default("")
  ec2_group_name: placement.group_name
  ec2_hypervisor: hypervisor
  ec2_id: instance_id
  ec2_image_id: image_id
  ec2_instance_profile: iam_instance_profile | default("")
  ec2_instance_type: instance_type
  ec2_ip_address: public_ip_address
  ec2_kernel: kernel_id | default("")
  ec2_key_name: key_name
```

(continues on next page)

(continued from previous page)

```

ec2_launch_time: launch_time | regex_replace(" ", "T") | regex_replace("(\\+)(\\d\\
↪d):(\\d)(\\d)$", ".\\g<2>\\g<3>Z")
ec2_monitored: monitoring.state in ['enabled', 'pending']
ec2_monitoring_state: monitoring.state
ec2_persistent: persistent | default(false)
ec2_placement: placement.availability_zone
ec2_platform: platform | default("")
ec2_private_dns_name: private_dns_name
ec2_private_ip_address: private_ip_address
ec2_public_dns_name: public_dns_name
ec2_ramdisk: ramdisk_id | default("")
ec2_reason: state_transition_reason
ec2_region: placement.region
ec2_requester_id: requester_id | default("")
ec2_root_device_name: root_device_name
ec2_root_device_type: root_device_type
ec2_security_group_ids: security_groups | map(attribute='group_id') | list | join(
↪',')
ec2_security_group_names: security_groups | map(attribute='group_name') | list | _
↪join(',')
ec2_sourceDestCheck: source_dest_check | default(false) | lower | string
ec2_spot_instance_request_id: spot_instance_request_id | default("")
ec2_state: state.name
ec2_state_code: state.code
ec2_state_reason: state_reason.message if state_reason is defined else ""
ec2_subnet_id: subnet_id | default("")
ec2_tag_Name: tags.Name
ec2_virtualization_type: virtualization_type
ec2_vpc_id: vpc_id | default("")
filters:
  instance-state-name:
    - running
groups:
  ec2: true
hostnames:
  - network-interface.addresses.association.public-ip
  - dns-name
  - private-dns-name
keyed_groups:
  - key: image_id | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: images
    prefix: ''
    separator: ''
  - key: placement.availability_zone
    parent_group: zones
    prefix: ''
    separator: ''
  - key: ec2_account_id | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: accounts
    prefix: ''
    separator: ''
  - key: ec2_state | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: instance_states
    prefix: instance_state
  - key: platform | default("undefined") | regex_replace("[^A-Za-z0-9\\_]", "_")
    parent_group: platforms
    prefix: platform

```

(continues on next page)

(continued from previous page)

```

- key: instance_type | regex_replace("[^A-Za-z0-9\\_]", "_")
  parent_group: types
  prefix: type
- key: key_name | regex_replace("[^A-Za-z0-9\\_]", "_")
  parent_group: keys
  prefix: key
- key: placement.region
  parent_group: regions
  prefix: ''
  separator: ''
- key: security_groups | map(attribute="group_name") | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list
  parent_group: security_groups
  prefix: security_group
- key: dict(tags.keys() | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list |
  zip(tags.values()
    | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list))
  parent_group: tags
  prefix: tag
- key: tags.keys() | map("regex_replace", "[^A-Za-z0-9\\_]", "_") | list
  parent_group: tags
  prefix: tag
- key: vpc_id | regex_replace("[^A-Za-z0-9\\_]", "_")
  parent_group: vpcs
  prefix: vpc_id
- key: placement.availability_zone
  parent_group: '{{ placement.region }}'
  prefix: ''
  separator: ''
plugin: amazon.aws.aws_ec2
use_contrib_script_compatible_sanitization: true

```

10.2 Google Compute Engine

```

auth_kind: serviceaccount
compose:
  ansible_ssh_host: networkInterfaces[0].accessConfigs[0].natIP |
  default(networkInterfaces[0].networkIP)
  gce_description: description if description else None
  gce_id: id
  gce_image: image
  gce_machine_type: machineType
  gce_metadata: metadata.get("items", []) | items2dict(key_name="key", value_name=
  "value")
  gce_name: name
  gce_network: networkInterfaces[0].network.name
  gce_private_ip: networkInterfaces[0].networkIP
  gce_public_ip: networkInterfaces[0].accessConfigs[0].natIP | default(None)
  gce_status: status
  gce_subnetwork: networkInterfaces[0].subnetwork.name
  gce_tags: tags.get("items", [])
  gce_zone: zone
hostnames:
- name

```

(continues on next page)

(continued from previous page)

```

- public_ip
- private_ip
keyed_groups:
- key: gce_subnetwork
  prefix: network
- key: gce_private_ip
  prefix: ''
  separator: ''
- key: gce_public_ip
  prefix: ''
  separator: ''
- key: machineType
  prefix: ''
  separator: ''
- key: zone
  prefix: ''
  separator: ''
- key: gce_tags
  prefix: tag
- key: status | lower
  prefix: status
- key: image
  prefix: ''
  separator: ''
plugin: google.cloud.gcp_compute
retrieve_image_info: true
use_contrib_script_compatible_sanitization: true

```

10.3 Microsoft Azure Resource Manager

```

conditional_groups:
  azure: true
default_host_filters: []
fail_on_template_errors: false
hostvar_expressions:
  computer_name: name
  private_ip: private_ipv4_addresses[0] if private_ipv4_addresses else None
  provisioning_state: provisioning_state | title
  public_ip: public_ipv4_addresses[0] if public_ipv4_addresses else None
  public_ip_id: public_ip_id if public_ip_id is defined else None
  public_ip_name: public_ip_name if public_ip_name is defined else None
  tags: tags if tags else None
  type: resource_type
keyed_groups:
- key: location
  prefix: ''
  separator: ''
- key: tags.keys() | list if tags else []
  prefix: ''
  separator: ''
- key: security_group
  prefix: ''
  separator: ''
- key: resource_group

```

(continues on next page)

(continued from previous page)

```

    prefix: ''
    separator: ''
- key: os_disk.operating_system_type
  prefix: ''
  separator: ''
- key: dict(tags.keys() | map("regex_replace", "^(.*)$", "\1_") | list | zip(tags.
↪values() | list)) if tags else []
  prefix: ''
  separator: ''
plain_host_names: true
plugin: azure.azurecollection.azure_rm
use_contrib_script_compatible_sanitization: true

```

10.4 VMware vCenter

```

compose:
  ansible_host: guest.ipAddress
  ansible_ssh_host: guest.ipAddress
  ansible_uuid: 99999999 | random | to_uuid
  availablefield: availableField
  configissue: configIssue
  configstatus: configStatus
  customvalue: customValue
  effectiverole: effectiveRole
  guestheartbeatstatus: guestHeartbeatStatus
  layoutex: layoutEx
  overallstatus: overallStatus
  parentvapp: parentVApp
  recenttask: recentTask
  resourcepool: resourcePool
  rootsnapshot: rootSnapshot
  triggeredalarmstate: triggeredAlarmState
filters:
- runtime.powerState == "poweredOn"
keyed_groups:
- key: config.guestId
  prefix: ''
  separator: ''
- key: "templates" if config.template else "guests"
  prefix: ''
  separator: ''
plugin: community.vmware.vmware_vm_inventory
properties:
- availableField
- configIssue
- configStatus
- customValue
- datastore
- effectiveRole
- guestHeartbeatStatus
- layout
- layoutEx
- name
- network

```

(continues on next page)

(continued from previous page)

```

- overallStatus
- parentVApp
- permission
- recentTask
- resourcePool
- rootSnapshot
- snapshot
- triggeredAlarmState
- value
- capability
- config
- guest
- runtime
- storage
- summary
strict: false
with_nested_properties: true

```

10.5 Red Hat Satellite 6

```

group_prefix: foreman_
keyed_groups:
- key: foreman['environment_name'] | lower | regex_replace(' ', '') | regex_replace(
  ↳ ['^A-Za-z0-9_'], '_') | regex_replace('none', '')
  prefix: foreman_environment_
  separator: ''
- key: foreman['location_name'] | lower | regex_replace(' ', '') | regex_replace(['^A-
  ↳ Za-z0-9_'], '_')
  prefix: foreman_location_
  separator: ''
- key: foreman['organization_name'] | lower | regex_replace(' ', '') | regex_replace(
  ↳ ['^A-Za-z0-9_'], '_')
  prefix: foreman_organization_
  separator: ''
- key: foreman['content_facet_attributes']['lifecycle_environment_name'] | lower |
  ↳ regex_replace(' ', '') | regex_replace(['^A-Za-z0-9_'], '_')
  prefix: foreman_lifecycle_environment_
  separator: ''
- key: foreman['content_facet_attributes']['content_view_name'] | lower | regex_
  ↳ replace(' ', '') | regex_replace(['^A-Za-z0-9_'], '_')
  prefix: foreman_content_view_
  separator: ''
legacy_hostvars: true
plugin: theforeman.foreman.foreman
validate_certs: false
want_facts: true
want_hostcollections: false
want_params: true

```

10.6 OpenStack

```
expand_hostvars: true
fail_on_errors: true
inventory_hostname: uuid
plugin: openstack.cloud.openstack
```

10.7 Red Hat Virtualization

```
compose:
  ansible_host: (devices.values() | list)[0][0] if devices else None
keyed_groups:
- key: cluster
  prefix: cluster
  separator: _
- key: status
  prefix: status
  separator: _
- key: tags
  prefix: tag
  separator: _
ovirt_hostname_preference:
- name
- fqdn
ovirt_insecure: false
plugin: ovirt.ovirt.ovirt
```

10.8 Ansible Tower

```
include_metadata: true
inventory_id: <inventory_id or url_quoted_named_url>
plugin: awx.awx.tower
validate_certs: <true or false>
```

SUPPORTED ATTRIBUTES FOR CUSTOM NOTIFICATIONS

This section describes the list of supported job attributes and the proper syntax for constructing the message text for notifications. The supported job attributes are:

- `allow_simultaneous` - (boolean) indicates if multiple jobs can run simultaneously from the JT associated with this job
- `controller_node` - (string) the instance that managed the isolated execution environment
- `created` - (datetime) timestamp when this job was created
- `custom_virtualenv` - (string) custom virtual environment used to execute job
- `description` - (string) optional description of the job
- `diff_mode` - (boolean) if enabled, textual changes made to any templated files on the host are shown in the standard output
- `elapsed` - (decimal) elapsed time in seconds that the job ran
- `execution_node` - (string) node the job executed on
- `failed` - (boolean) true if job failed
- `finished` - (datetime) date and time the job finished execution
- `force_handlers` - (boolean) when handlers are forced, they will run when notified even if a task fails on that host (note that some conditions - e.g. unreachable hosts - can still prevent handlers from running)
- `forks` - (int) number of forks requested for job
- `id` - (int) database id for this job
- `job_explanation` - (string) status field to indicate the state of the job if it wasn't able to run and capture stdout
- `job_slice_count` - (integer) if run as part of a sliced job, the total number of slices (if 1, job is not part of a sliced job)
- `job_slice_number` - (integer) if run as part of a sliced job, the ID of the inventory slice operated on (if not part of a sliced job, attribute is not used)
- `job_tags` - (string) only tasks with specified tags will execute
- `job_type` - (choice) run, check, or scan
- `launch_type` - (choice) manual, relaunch, callback, scheduled, dependency, workflow, sync, or scm
- `limit` - (string) playbook execution limited to this set of hosts, if specified
- `modified` - (datetime) timestamp when this job was last modified
- `name` - (string) name of this job

- `playbook` - (string) playbook executed
- `scm_revision` - (string) scm revision from the project used for this job, if available
- `skip_tags` - (string) playbook execution skips over this set of tag(s), if specified
- `start_at_task` - (string) playbook execution begins at the task matching this name, if specified
- `started` - (datetime) date and time the job was queued for starting
- `status` - (choice) new, pending, waiting, running, successful, failed, error, canceled
- `timeout` - (int) amount of time (in seconds) to run before the task is canceled
- `type` - (choice) data type for this job
- `url` - (string) URL for this job
- `use_fact_cache` - (boolean) if enabled for job, Tower acts as an Ansible Fact Cache Plugin, persisting facts at the end of a playbook run to the database and caching facts for use by Ansible
- `verbosity` - (choice) 0 through 5 (corresponding to Normal through WinRM Debug)
- **host_status_counts (count of hosts uniquely assigned to each status)**
 - `skipped` (integer)
 - `ok` (integer)
 - `changed` (integer)
 - `failures` (integer)
 - `dark` (integer)
 - `processed` (integer)
 - `rescued` (integer)
 - `ignored` (integer)
 - `failed` (boolean)
- **summary_fields:**
 - **inventory**
 - * `id` - (integer) database ID for inventory
 - * `name` - (string) name of the inventory
 - * `description` - (string) optional description of the inventory
 - * `has_active_failures` - (boolean) (deprecated) flag indicating whether any hosts in this inventory have failed
 - * `total_hosts` - (deprecated) (int) total number of hosts in this inventory.
 - * `hosts_with_active_failures` - (deprecated) (int) number of hosts in this inventory with active failures
 - * `total_groups` - (deprecated) (int) total number of groups in this inventory
 - * `groups_with_active_failures` - (deprecated) (int) number of hosts in this inventory with active failures
 - * `has_inventory_sources` - (deprecated) (boolean) flag indicating whether this inventory has external inventory sources

- * `total_inventory_sources` - (int) total number of external inventory sources configured within this inventory
 - * `inventory_sources_with_failures` - (int) number of external inventory sources in this inventory with failures
 - * `organization_id` - (id) organization containing this inventory
 - * `kind` - (choice) (empty string) (indicating hosts have direct link with inventory) or 'smart'
- project**
- * `id` - (int) database ID for project
 - * `name` - (string) name of the project
 - * `description` - (string) optional description of the project
 - * `status` - (choices) one of new, pending, waiting, running, successful, failed, error, canceled, never updated, ok, or missing
 - * `scm_type` (choice) - one of (empty string), git, hg, svn, insights
- job_template**
- * `id` - (int) database ID for job template
 - * `name` - (string) name of job template
 - * `description` - (string) optional description for the job template
- unified_job_template**
- * `id` - (int) database ID for unified job template
 - * `name` - (string) name of unified job template
 - * `description` - (string) optional description for the unified job template
 - * `unified_job_type` - (choice) unified job type (job, workflow_job, project_update, etc.)
- instance_group**
- * `id` - (int) database ID for instance group
 - * `name` - (string) name of instance group
- created_by**
- * `id` - (int) database ID of user
 - * `username` - (string) username
 - * `first_name` - (string) first name
 - * `last_name` - (string) last name
- labels**
- * `count` - (int) number of labels
 - * `results` - list of dictionaries representing labels (e.g. {"id": 5, "name": "database jobs"})

Information about a job can be referenced in a custom notification message using grouped curly braces `{{ }}`. Specific job attributes are accessed using dotted notation, for example `{{ job.summary_fields.inventory.name }}`. Any characters used in front or around the braces, or plain text, can be added for clarification, such as '#' for job ID and single-quotes to denote some descriptor. Custom messages can include a number of variables throughout the message:

```
{{ job_friendly_name }} {{ job.id }} ran on {{ job.execution_node }} in {{ job.  
↪elapsed }} seconds.
```

In addition to the job attributes, there are some other variables that can be added to the template:

- `approval_node_name` - (string) the approval node name
- `approval_status` - (choice) one of `approved`, `denied`, and `timed_out`
- `url` - (string) URL of the job for which the notification is emitted (this applies to start, success, fail, and approval notifications)
- `workflow_url` - (string) URL to the relevant approval node. This allows the notification recipient to go to the relevant workflow job page to see what's going on (i.e., This node can be viewed at: `{{ workflow_url }}`). In cases of approval-related notifications, both `url` and `workflow_url` are the same.
- `job_friendly_name` - (string) the friendly name of the job
- `job_metadata` - (string) job metadata as a JSON string, for example:

```
{'url': 'https://towerhost/$/jobs/playbook/13',  
'traceback': '',  
'status': 'running',  
'started': '2019-08-07T21:46:38.362630+00:00',  
'project': 'Stub project',  
'playbook': 'ping.yml',  
'name': 'Stub Job Template',  
'limit': '',  
'inventory': 'Stub Inventory',  
'id': 42,  
'hosts': {},  
'friendly_name': 'Job',  
'finished': False,  
'credential': 'Stub credential',  
'created_by': 'admin'}
```

GLOSSARY

Ad Hoc Refers to running Ansible to perform some quick command, using `/usr/bin/ansible`, rather than the orchestration language, which is `/usr/bin/ansible-playbook`. An example of an ad hoc command might be rebooting 50 machines in your infrastructure. Anything you can do ad hoc can be accomplished by writing a Playbook, and Playbooks can also glue lots of other operations together.

Callback Plugin Refers to some user-written code that can intercept results from Ansible and do something with them. Some supplied examples in the GitHub project perform custom logging, send email, or even play sound effects.

Control Groups Also known as *cgroups*, a control group is a feature in the Linux kernel that allows resources to be grouped and allocated to run certain processes. In addition to assigning resources to processes, cgroups can also report actual resource usage by all processes running inside of the cgroup.

Check Mode Refers to running Ansible with the `--check` option, which does not make any changes on the remote systems, but only outputs the changes that might occur if the command ran without this flag. This is analogous to so-called “dry run” modes in other systems, though the user should be warned that this does not take into account unexpected command failures or cascade effects (which is true of similar modes in other systems). Use this to get an idea of what might happen, but it is not a substitute for a good staging environment.

Container Groups Container Groups are a type of Instance Group that specify a configuration for provisioning a pod in a Kubernetes or OpenShift cluster where a job is run. These pods are provisioned on-demand and exist only for the duration of the playbook run.

Credentials Authentication details that may be utilized by Tower to launch jobs against machines, to synchronize with inventory sources, and to import project content from a version control system.

Credential Plugin Python code that contains definitions for an external credential type, its metadata fields, and the code needed for interacting with a secret management system.

Distributed Job A job that consists of a job template, an inventory, and slice size. When executed, a distributed job slices each inventory into a number of “slice size” chunks, which are then used to run smaller job slices.

External Credential Type A managed credential type for Tower used for authenticating with a secret management system.

Facts Facts are simply things that are discovered about remote nodes. While they can be used in playbooks and templates just like variables, facts are things that are inferred, rather than set. Facts are automatically discovered when running plays by executing the internal setup module on the remote nodes. You never have to call the setup module explicitly, it just runs, but it can be disabled to save time if it is not needed. For the convenience of users who are switching from other configuration management systems, the fact module also pulls in facts from the ‘ohai’ and ‘facter’ tools if they are installed, which are fact libraries from Chef and Puppet, respectively.

Forks Ansible and Tower talk to remote nodes in parallel and the level of parallelism can be set several ways—during the creation or editing of a Job Template, by passing `--forks`, or by editing the default in a configuration file. The default is a very conservative 5 forks, though if you have a lot of RAM, you can easily set this to a value like 50 for increased parallelism.

Group A set of hosts in Ansible that can be addressed as a set, of which many may exist within a single Inventory.

Group Vars The `group_vars/` files are files that live in a directory alongside an inventory file, with an optional filename named after each group. This is a convenient place to put variables that will be provided to a given group, especially complex data structures, so that these variables do not have to be embedded in the inventory file or playbook.

Handlers Handlers are just like regular tasks in an Ansible playbook (see Tasks), but are only run if the Task contains a “notify” directive and also indicates that it changed something. For example, if a config file is changed then the task referencing the config file templating operation may notify a service restart handler. This means services can be bounced only if they need to be restarted. Handlers can be used for things other than service restarts, but service restarts are the most common usage.

Host A system managed by Tower, which may include a physical, virtual, cloud-based server, or other device. Typically an operating system instance. Hosts are contained in Inventory. Sometimes referred to as a “node”.

Host Specifier Each Play in Ansible maps a series of tasks (which define the role, purpose, or orders of a system) to a set of systems. This “hosts:” directive in each play is often called the hosts specifier. It may select one system, many systems, one or more groups, or even some hosts that are in one group and explicitly not in another.

Instance Group A group that contains instances for use in a clustered environment. An instance group provides the ability to group instances based on policy.

Inventory A collection of hosts against which Jobs may be launched.

Inventory Script A very simple program (or a complicated one) that looks up hosts, group membership for hosts, and variable information from an external resource—whether that be a SQL database, a CMDB solution, or something like LDAP. This concept was adapted from Puppet (where it is called an “External Nodes Classifier”) and works more or less exactly the same way.

Inventory Source Information about a cloud or other script that should be merged into the current inventory group, resulting in the automatic population of Groups, Hosts, and variables about those groups and hosts.

Job One of many background tasks launched by Tower, this is usually the instantiation of a Job Template; the launch of an Ansible playbook. Other types of jobs include inventory imports, project synchronizations from source control, or administrative cleanup actions.

Job Detail The history of running a particular job, including its output and success/failure status.

Job Slice See *Distributed Job*.

Job Template The combination of an Ansible playbook and the set of parameters required to launch it.

JSON Ansible and Tower use JSON for return data from remote modules. This allows modules to be written in any language, not just Python.

Metadata Information for locating a secret in the external system once authenticated. The uses provides this information when linking an external credential to a target credential field.

Notification Template An instance of a notification type (Email, Slack, Webhook, etc.) with a name, description, and a defined configuration.

Notification A manifestation of the notification template; for example, when a job fails a notification is sent using the configuration defined by the notification template.

Notify The act of a task registering a change event and informing a handler task that another action needs to be run at the end of the play. If a handler is notified by multiple tasks, it will still be run only once. Handlers are run in the order they are listed, not in the order that they are notified.

Organization A logical collection of Users, Teams, Projects, and Inventories. The highest level in the Tower object hierarchy is the Organization.

Organization Administrator An Tower user with the rights to modify the Organization’s membership and settings, including making new users and projects within that organization. An organization admin can also grant permissions to other users within the organization.

Permissions The set of privileges assigned to Users and Teams that provide the ability to read, modify, and administer Projects, Inventories, and other Tower objects.

Plays A playbook is a list of plays. A play is minimally a mapping between a set of hosts selected by a host specifier (usually chosen by groups, but sometimes by hostname globs) and the tasks which run on those hosts to define the role that those systems will perform. There can be one or many plays in a playbook.

Playbook An Ansible playbook. Refer to <http://docs.ansible.com/> for more information.

Policy Policies dictate how instance groups behave and how jobs are executed.

Project A logical collection of Ansible playbooks, represented in Tower.

Roles Roles are units of organization in Ansible and Tower. Assigning a role to a group of hosts (or a set of groups, or host patterns, etc.) implies that they should implement a specific behavior. A role may include applying certain variable values, certain tasks, and certain handlers—or just one or more of these things. Because of the file structure associated with a role, roles become redistributable units that allow you to share behavior among playbooks—or even with other users.

Secret Management System A server or service for securely storing and controlling access to tokens, passwords, certificates, encryption keys, and other sensitive data.

Schedule The calendar of dates and times for which a job should run automatically.

Sliced Job See *Distributed Job*.

Source Credential An external credential that is linked to the field of a target credential.

Sudo Ansible does not require root logins and, since it is daemonless, does not require root level daemons (which can be a security concern in sensitive environments). Ansible can log in and perform many operations wrapped in a `sudo` command, and can work with both password-less and password-based sudo. Some operations that do not normally work with `sudo` (like `scp` file transfer) can be achieved with Ansible’s *copy*, *template*, and *fetch* modules while running in `sudo` mode.

Superuser An admin of the Tower server who has permission to edit any object in the system, whether associated to any organization. Superusers can create organizations and other superusers.

Survey Questions asked by a job template at job launch time, configurable on the job template.

Target Credential A non-external credential with an input field that is linked to an external credential.

Team A sub-division of an Organization with associated Users, Projects, Credentials, and Permissions. Teams provide a means to implement role-based access control schemes and delegate responsibilities across Organizations.

User An Tower operator with associated permissions and credentials.

Webhook Webhooks allow communication and information sharing between apps. They are used to respond to commits pushed to SCMs and launch job templates or workflow templates.

Workflow Job Template A set consisting of any combination of job templates, project syncs, and inventory syncs, linked together in order to execute them as a single unit.

YAML Ansible and Tower use YAML to define playbook configuration languages and also variable files. YAML has a minimum of syntax, is very clean, and is easy for people to skim. It is a good data format for configuration files and humans, but is also machine readable. YAML is fairly popular in the dynamic language community and the format has libraries available for serialization in many languages (Python, Perl, Ruby, etc.).

INDEX

- 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.

Symbols

- |rhell|
 - platform-specific notes, 9, 10
- 2.9
 - Ansible, 17

A

- active/passive, external database,
 - clustered
 - installation multi-machine, 21
- Ad Hoc, 44
- Amazon Web Services
 - inventories, 33
- analytics collection, 32
- Ansible
 - 2.9, 17
 - latest, 17
 - stable, 17
- Ansible Tower
 - inventories, 39
- attaching
 - subscription, 3
- attributes
 - notification, 40
- aws
 - inventories, 33
 - inventory plugins, 33
- azure
 - inventories, 36
 - inventory plugins, 36

B

- bundled |aap| installer, 19

C

- Callback Plugin, 44
- CentOS
 - platform-specific notes, 9, 10
- Check Mode, 44
- components
 - licenses, 4
- configuration

- PostgreSQL, 17
- websockets, 13
- consume
 - subscription, 3
- Container Groups, 44
- Control Groups, 44
- Credential Plugin, 44
- Credentials, 44
- current
 - release notes, 5
- custom
 - notification messages, 40
- custom SSL certificates, 6

D

- data collection, 32
- database
 - PostgreSQL, 17
- DEB files
 - licenses, 4
- Distributed Job, 44
- download Ansible Tower, 19

E

- evaluation, 2
- External Credential Type, 44
- external database
 - installation single machine, 21
- extra vars
 - flags, 7
 - installation, 7

F

- Facts, 44
- features, 1
- FIPS
 - platform-specific notes, 9
- flags
 - extra vars, 7
 - installation, 7
- Forks, 44

G

gce
 inventories, 35
 inventory plugins, 35
 glossary, 44
 Google Compute Engine
 inventories, 35
 Group, 45
 Group Vars, 45

H

Handlers, 45
 Host, 45
 Host Specifier, 45
 http
 proxy, 6

I

installation, 21
 extra vars, 7
 flags, 7
 general notes, 6
 multi-machine active/passive,
 external database, clustered,
 21
 platform-specific notes, 9
 scenarios, 21
 single machine external database, 21
 single machine integrated, 21
 installation bundle
 licenses, 4
 installation program, 19
 upgrade, 30
 installation requirements, 15
 installation script
 inventory file setup, 23
 playbook setup, 28, 31
 Instance Group, 45
 integrated
 installation single machine, 21
 inventories
 Amazon Web Services, 33
 Ansible Tower, 39
 aws, 33
 azure, 36
 gce, 35
 Google Compute Engine, 35
 Microsoft Azure Resource Manager, 36
 OpenStack, 39
 Red Hat Satellite 6, 38
 Red Hat Virtualization, 39
 rhv, 39
 satellite, 38
 tower, 39

 vmware, 37
 VMware vCenter, 37
 Inventory, 45
 inventory file setup, 23
 inventory plugins
 aws, 33
 azure, 36
 gce, 35
 OpenStack, 39
 rhv, 39
 satellite, 38
 templates, 33
 tower, 39
 vmware, 37
 Inventory Script, 45
 Inventory Source, 45

J

Job, 45
 Job Detail, 45
 Job Slice, 45
 Job Template, 45
 JSON, 45

L

latest
 Ansible, 17
 license, 1, 2
 nodes, 3
 trial, 2
 types, 2
 license features, 1
 licenses
 components, 4
 DEB files, 4
 installation bundle, 4
 RPM files, 4

M

Metadata, 45
 Microsoft Azure Resource Manager
 inventories, 36
 multi-machine
 active/passive, external database,
 clustered, installation, 21

N

no proxy
 proxy support, 11
 Notification, 45
 notification
 attributes, 40
 notification messages
 custom, 40

Notification Template, [45](#)

Notify, [45](#)

O

OCP

platform-specific notes, [10](#)

OpenShift

platform-specific notes, [10](#)

OpenStack

inventories, [39](#)

inventory plugins, [39](#)

Organization, [45](#)

Organization Administrator, [46](#)

P

password, changing, [29](#)

Pendo, [32](#)

Permissions, [46](#)

platform-specific notes

|rhel|, [9, 10](#)

CentOS, [9, 10](#)

FIPS, [9](#)

installation, [9](#)

OCP, [10](#)

OpenShift, [10](#)

Satellite, [10](#)

Playbook, [46](#)

playbook setup, [28, 31](#)

installation script, [28, 31](#)

setup.sh, [28, 31](#)

Plays, [46](#)

Policy, [46](#)

PostgreSQL

configuration, [17](#)

database, [17](#)

tuning, [17](#)

Project, [46](#)

proxy support, [11](#)

no proxy, [11](#)

reverse proxy, [13](#)

R

Red Hat Satellite 6

inventories, [38](#)

Red Hat Virtualization

inventories, [39](#)

release notes, [5](#)

current, [5](#)

requirements, [15](#)

reverse proxy, [13](#)

proxy support, [13](#)

rhv

inventories, [39](#)

inventory plugins, [39](#)

Roles, [46](#)

RPM files

licenses, [4](#)

S

Satellite

platform-specific notes, [10](#)

satellite

inventories, [38](#)

inventory plugins, [38](#)

Schedule, [46](#)

Secret Management System, [46](#)

self-signed SSL certificate, [6](#)

setup.sh

playbook setup, [28, 31](#)

single machine

external database, installation, [21](#)

integrated, installation, [21](#)

Sliced Job, [46](#)

Source Credential, [46](#)

stable

Ansible, [17](#)

subscription

attaching, [3](#)

consume, [3](#)

Sudo, [46](#)

Superuser, [46](#)

support, [1, 2](#)

Survey, [46](#)

T

Target Credential, [46](#)

Team, [46](#)

templates

inventory plugins, [33](#)

tower

inventories, [39](#)

inventory plugins, [39](#)

trial, [2](#)

tuning

PostgreSQL, [17](#)

U

updates, [2](#)

upgrade, [30](#)

installation program, [30](#)

usability data collection, [32](#)

User, [46](#)

user data tracking, [32](#)

USER_ANALYTICS_TRACKING_STATE, [32](#)

V

vmware

inventories, [37](#)

- inventory plugins, 37
- VMware vCenter
 - inventories, 37

W

- Webhook, 46
- websocket, 13
- websockets
 - configuration, 13
- Workflow Job Template, 46

Y

- YAML, 46