
Ansible Tower Administration Guide

Release Ansible Tower 3.4.2

Red Hat, Inc.

May 05, 2021

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	Tower Component Licenses	3
2	Starting, Stopping, and Restarting Tower	4
3	Custom Inventory Scripts	5
3.1	Writing Inventory Scripts	7
4	Inventory File Importing	8
4.1	Custom Dynamic Inventory Scripts	8
4.2	SCM Inventory Source Fields	9
5	Multi-Credential Assignment	10
5.1	Background	10
5.2	Important Changes	10
5.3	Launch Time Considerations	11
5.4	Backwards Compatibility Concerns	11
5.5	Multi-Vault Credentials	12
6	Management Jobs	15
6.1	Removing Old Activity Stream Data	15
6.2	Removing Old Job History	19
7	Clustering	20
7.1	Setup Considerations	21
7.2	Install and Configure	21
7.3	Configuring Instances and Instance Groups from the API	23
7.4	Instance group policies	24
7.5	Isolated Instance Groups	25
7.6	Status and Monitoring via Browser API	28
7.7	Instance Services and Failure Behavior	28
7.8	Job Runtime Behavior	28
7.9	Deprovision Instances and Instance Groups	31
8	OpenShift Deployment and Configuration	32
8.1	Tower and OpenShift Basics	33
8.2	Configuration Options	33

8.3	Basic Configuration	33
8.4	Resource Requests and Request Planning	34
8.5	Database Configuration and Usage	35
8.6	Backup and Restore	36
8.7	Upgrading	36
8.8	Migrating	36
8.9	Build custom virtual environments	36
9	Proxy Support	38
9.1	Configure Known Proxies	39
9.2	Reverse Proxy	40
10	Tower Logfiles	41
11	Tower Logging and Aggregation	42
11.1	Loggers	42
11.2	Set Up Logging with Tower	47
12	The <i>awx-manage</i> Utility	49
12.1	Inventory Import	49
12.2	Cleanup of old data	50
12.3	Cluster management	50
12.4	Token and session management	50
13	Tower Configuration	53
13.1	Authentication	53
13.2	Jobs	54
13.3	System	55
13.4	User Interface	56
13.5	License	58
14	Bubblewrap functionality and variables	61
15	Token-Based Authentication	62
15.1	Managing OAuth 2 Applications and Tokens	62
15.2	Using OAuth 2 Token System for Personal Access Tokens (PAT)	65
15.3	Application Functions	66
15.4	Application Token Functions	70
16	Setting up Social Authentication	72
16.1	Basic Authentication Settings	73
16.2	Google OAuth2 Settings	73
16.3	GitHub OAuth2 Settings	74
16.4	Organization and Team Mapping	79
17	Setting up Enterprise Authentication	82
17.1	Azure Active Directory (AD)	82
17.2	SAML Authentication Settings	84
17.3	RADIUS Authentication Settings	92
17.4	TACACS+ Authentication Settings	92
18	Setting up LDAP Authentication	94
18.1	Referrals	100
18.2	Enabling Logging for LDAP	100
18.3	LDAP Organization and Team Mapping	100

19	Changing the Default Timeout for Authentication	103
20	User Authentication with Kerberos	104
20.1	AD and Kerberos Credentials	105
20.2	Working with Kerberos Tickets	106
21	Working with Session Limits	107
22	Backing Up and Restoring Tower	108
22.1	Backup/Restore Playbooks	108
22.2	Backup and Restoration Considerations	109
22.3	Backup and Restore for Clustered Environments	109
23	Using Custom Logos in Ansible Tower	111
24	Troubleshooting Tower	113
24.1	Error logs	113
24.2	Problems connecting to your host	113
24.3	Unable to login to Tower via HTTP	113
24.4	WebSockets port for live events not working	114
24.5	Problems running a playbook	114
24.6	Problems when running a job	114
24.7	Playbooks aren't showing up in the "Job Template" drop-down	114
24.8	Playbook stays in pending	114
24.9	Cancel a Tower job	115
24.10	Reusing an external database causes installations to fail	115
24.11	Private EC2 VPC Instances in Tower Inventory	116
24.12	Troubleshooting "Error: provided hosts list is empty"	116
25	Tower Tips and Tricks	117
25.1	Using the Tower CLI Tool	117
25.2	Launching a Job Template via the API	117
25.3	<code>tower-cli</code> Job Template Launching	119
25.4	Changing the Tower Admin Password	119
25.5	Creating a Tower Admin from the commandline	120
25.6	Setting up a jump host to use with Tower	120
25.7	View Ansible outputs for JSON commands when using Tower	121
25.8	Locate and configure the Ansible configuration file	121
25.9	View a listing of all <code>ansible_</code> variables	121
25.10	Using <code>virtualenv</code> with Ansible Tower	121
25.11	Configuring the <code>towerhost</code> hostname for notifications	123
25.12	Launching Jobs with <code>curl</code>	123
25.13	Dynamic Inventory and private IP addresses	124
25.14	Filtering instances returned by the dynamic inventory sources in Tower	124
25.15	Using an unreleased module from Ansible source with Tower	125
25.16	Using callback plugins with Tower	125
25.17	Connecting to Windows with <code>winrm</code>	126
25.18	Importing existing inventory files and <code>host/group</code> vars into Tower	126
26	Introduction to <code>tower-cli</code>	128
26.1	License	128
26.2	Capabilities	128
26.3	Installation	128
26.4	Configuration	129

27 Usability Analytics and Data Collection	133
28 Postface	134
29 Index	137
30 Copyright © 2019 Red Hat, Inc.	138
Index	139

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

The *Ansible Tower Administration Guide* documents the administration of Ansible Tower through custom scripts, management jobs, and more. Written for DevOps engineers and administrators, the *Ansible Tower Administration Guide* assumes a basic understanding of the systems requiring management with Tower's easy-to-use graphical interface. This document has been updated to include information for the latest release of Ansible Tower 3.4.2.

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 3.4.2; March 06, 2019; <https://access.redhat.com/>

TOWER LICENSING, UPDATES, AND SUPPORT

Red Hat Ansible Tower (“**Ansible Tower**”) is a software product provided as part of an annual 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>

1.1 Support

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

If you or your company has purchased a subscription for Ansible Automation, you can contact the support team at <https://access.redhat.com>. To better understand the levels of support which match your Ansible Tower Subscription, refer to *Subscription Types*. For details of what is covered under an Ansible Automation 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 is provided at various levels of support and number of machines as an annual Subscription.

- **Standard (F.K.A. “Enterprise: 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>

- **Enterprise (F.K.A. “Enterprise: 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.

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

1.4 Node Counting in Licenses

The Tower license defines the number of Managed Nodes that can be managed by Ansible Tower. A typical license will say ‘License Count: 500’, which sets the maximum number of Managed Nodes at 500.

Ansible Tower counts Managed Nodes by the number of nodes in inventory. If more Managed Nodes are in the Ansible Tower inventory than are supported by the license, you will be unable to start any Jobs in Ansible Tower. If a dynamic inventory sync causes Ansible Tower to exceed the Managed Node count specified in the license, the dynamic inventory sync will fail.

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

1.5 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.

STARTING, STOPPING, AND RESTARTING TOWER

Ansible Tower ships with an *admin utility script*, `ansible-tower-service`, that can start, stop, and restart all Tower services running on the current Tower node (including the message queue components, and the database if it is an integrated installation). External databases must be explicitly managed by the administrator. The services script resides in `/usr/bin/ansible-tower-service` and can be invoked as follows:

```
root@localhost:~$ ansible-tower-service restart
```

You can also invoke it via distribution-specific service management commands. Distribution packages often provide a similar script, sometimes as an init script, to manage services. Refer to your distribution-specific service management system for more information.


Note: When running Tower containerized in *OpenShift*, do not use the `ansible-tower-service` script. Restart the pod using OpenShift instead.

Note: Beginning with version 2.2.0, Ansible Tower has moved away from using an init script in favor of using an admin utility script. Previous versions of Ansible Tower shipped with a standard `ansible-tower` init script that could be used to `start`, `stop`, and `query` the full Tower infrastructure. It was evoked via the service command: `/etc/init.d/ansible-tower script`. For those using a 2.2.0 or later version of Ansible Tower, the new admin utility script, `ansible-tower-service`, should be used instead.

CUSTOM INVENTORY SCRIPTS

Tower includes built-in support for syncing dynamic inventory from cloud sources such as Amazon AWS, Google Compute Engine, among others. Tower also offers the ability to use a custom script to pull from your own inventory source.

Note: With the release of Ansible Tower 2.4.0, edits and additions to Inventory host variables now persist beyond an inventory sync as long as `--overwrite_vars` is **not** set. To have inventory syncs behave as they did before, it is now required that both `--overwrite` and `--overwrite_vars` are set.


To manage the custom inventory scripts available in Tower, choose **Inventory Scripts** from the Setup () menu.

[SETTINGS](#) / INVENTORY SCRIPTS



INVENTORY SCRIPTS 0 + ADD

PLEASE ADD ITEMS TO THIS LIST

To add a new custom inventory script, click the  button.



NEW CUSTOM INVENTORY +

*NAME DESCRIPTION *ORGANIZATION

*CUSTOM SCRIPT ?

Enter the name for the script, plus an optional description. Then select the **Organization** that this script belongs to.

You can then either drag and drop a script on your local system into the **Custom Script** text box, or cut and paste the contents of the inventory script there.

HOST-A-NATOR +

*NAME DESCRIPTION *ORGANIZATION

*CUSTOM SCRIPT ?

```
#!/usr/bin/env python

# Python
import json
import optparse
import os

nhosts = int(os.environ.get('NHOSTS', 100))

inv_list = {
```

INVENTORY SCRIPTS + ADD

NAME 🔍

NAME	DESCRIPTION	ORGANIZATION	ACTIONS
Host-a-nator	Host Populator	Honey Dog, Inc.	🔍 🗑️

ITEMS 1-1 OF 1

3.1 Writing Inventory Scripts

You can write inventory scripts in any dynamic language that you have installed on the Tower machine (such as shell or python). They must start with a normal script shebang line such as `#!/bin/bash` or `#!/usr/bin/python`. They run as the `awx` user. The inventory script invokes with `'--list'` to list the inventory, which returns in a JSON hash/dictionary.

Generally, they connect to the network to retrieve the inventory from other sources. When enabling multi-tenancy security (refer to [Security](#) for details), the inventory script will not be able to access most of the Tower machine. If this access to the local Tower machine is necessary, configure it in `/etc/tower/settings.py`.

For more information on dynamic inventory scripts and how to write them, refer to the [Intro to Dynamic Inventory](#) and [Developing Dynamic Inventory Sources](#) sections of the Ansible documentation, or review the [example dynamic inventory scripts](#) on GitHub.

INVENTORY FILE IMPORTING

Ansible Tower 3.2 introduces the ability to choose an inventory file from source control, rather than creating one from scratch. This function is the same as custom inventory scripts, except that the contents are obtained from source control instead of editing their contents browser. This means, the files are non-editable and as inventories are updated at the source, the inventories within the projects are also updated accordingly, including the `group_vars` and `host_vars` files or directory associated with them. SCM types can consume both inventory files and scripts, the overlap between inventory files and custom types in that both do scripts.

Note: These features are compatible with Ansible version 2.4 and later. However, previous versions of Ansible are supported, but with some limitations.

4.1 Custom Dynamic Inventory Scripts

A custom dynamic inventory script stored in version control can be imported and run. This makes it much easier to make changes to an inventory script — rather than having to copy and paste one into Tower, it is pulled directly from source control and then executed. The script must be written to handle any credentials needed for doing its work and you are responsible for installing any Python libraries needed by the script (which is the same requirement for custom dynamic inventory scripts). And this applies to both user-defined inventory source scripts and SCM sources as they are both exposed to Ansible *virtualenv* requirements related to playbooks.

You can specify environment variables when you edit the SCM inventory source itself. For some scripts, this will be sufficient, however, this is not a secure way to store secret information that gives access to cloud providers or inventory.

The better way is to create a new credential type for the inventory script you are going to use. The credential type will need to specify all the necessary types of inputs. Then, when you create a credential of this type, the secrets will be stored in an encrypted form. If you apply that credential to the inventory source, the script will have access to those inputs like environment variables or files.

4.1.1 Update on Project Change

If the inventory source contains static content, it may be desirable to automatically update its content whenever the SHA-1 hash of its source project changes. This can be done by configuring the inventory source to Update on Project Change.

When this box is checked, the inventory source will not allow update-on-launch. Update-on-launch is important because some configurations require it. For example, when you set up a project that the inventory references to update in series before a Job Template runs, so that the inventory that the Job Template runs will have the updated form of that inventory. However, there are two other alternative ways to accomplish this:

- You can make a job template that uses a project as well as an inventory that updates from that same project. In this case, you can set the project to `update_on_launch`, in which case it will trigger an inventory update, if needed.
- If you must use a different project for the playbook than for the inventory source, then you can still place the project in a workflow and then have a job template run on success of the project update.

This is guaranteed to have the inventory update “on time” (meaning that the inventory changes are complete before the job template is launched), because the project does not transition to the completed state until the inventory update is finished.

Note: A failed inventory update does not mark the project as failed. Also, not every project update will trigger a corresponding inventory update. If the project revision has not changed and the inventory has not been edited, the inventory update will not execute.

4.2 SCM Inventory Source Fields

The source fields used are:

- `source_project`: project to use
- `source_path`: relative path inside the project indicating a directory or a file. If left blank, “” is still a relative path indicating the root directory of the project
- `source_vars`: if set on a “file” type inventory source then they will be passed to the environment vars when running

An update of the project automatically triggers an inventory update where it is used. An update of the project is scheduled immediately after creation of the inventory source.

You can specify a location manually in the Tower User Interface from the Create Inventory Source page.

Refer to [Refer to the Inventories](#) for instructions on creating an inventory source.

This listing should be refreshed to latest SCM info on a project update. If no inventory sources use a project as an SCM inventory source, then the inventory listing may not be refreshed on update.

4.2.1 Supported File Syntax

Ansible Tower uses the `ansible-inventory` module from Ansible 2.4 and later that supports all valid inventory syntax that Tower requires.

In order to make it configurable on the command line, the option `--method` is available with the `awx-manage inventory_import` command. Inventory updates from files will use a backported version of the `ansible-inventory` command for Ansible versions 2.4 and earlier.

For versions of Ansible 2.4 and later, the officially distributed `ansible-inventory` command will be used to process inventory files.

MULTI-CREDENTIAL ASSIGNMENT

Starting with version 3.3, Ansible Tower provides support for assigning zero or more credentials to a job template.

5.1 Background

Prior to Ansible Tower 3.3, job templates had a certain set of requirements with respect to credentials:

- All job templates (and jobs) were required to have exactly *one* Machine/SSH or Vault credential (or one of both).
- All job templates (and jobs) could have zero or more “extra” credentials.
- Extra credentials represented “Cloud” and “Network” credentials that could be used to provide authentication to external services via environment variables (e.g., `AWS_ACCESS_KEY_ID`).

This model required a variety of disjoint interfaces for specifying credentials on a job template and it lacked the ability associate multiple Vault credentials with a playbook run, a use case supported by Ansible core from Ansible 2.4 onwards.

This model also poses a stumbling block for certain playbook execution workflows, such as having to attach a “dummy” Machine/SSH credential to the job template simply to satisfy the requirement.

5.2 Important Changes

Job templates now have a single interface for credential assignment. From the API endpoint:

```
GET /api/v2/job_templates/N/credentials/
```

You can associate and disassociate credentials using `POST` requests, similar to the behavior in the deprecated `extra_credentials` endpoint:

```
POST /api/v2/job_templates/N/credentials/ {'associate': true, 'id': X'}
POST /api/v2/job_templates/N/credentials/ {'disassociate': true, 'id': Y'}
```

Under this model, a job template is considered valid even when there are *no* credentials assigned to it. This model also provides users the ability to assign multiple Vault credentials to a job template.

5.3 Launch Time Considerations

Prior to Ansible Tower 3.3, job templates had a configurable attribute, `ask_credential_on_launch`. This value was used at launch time to determine which missing credential values were necessary for launch - this was primarily used as a way to specify a Machine/SSH credential to satisfy the minimum credential requirement.

Under the new unified credential list model, this attribute still exists, but it is no longer “requiring” a credential. Now when `ask_credential_on_launch` is `True`, it signifies that if desired, you may specify a list of credentials at launch time to override those defined on the job template. For example:

```
POST /api/v2/job_templates/N/launch/ {'credentials': [A, B, C]}
```

If `ask_credential_on_launch` is `False`, it signifies that custom credentials provided in the `POST /api/v2/job_templates/N/launch/` will be ignored.

Under this model, the only purpose for `ask_credential_on_launch` is to signal API clients to prompt the user for (optional) changes at launch time.

5.4 Backwards Compatibility Concerns

A variety of API clients rely on now-deprecated mechanisms for credential retrieval and assignment, and those are still supported in a backwards-compatible way under this new API change. Requests to update `JobTemplate.credential` and `JobTemplate.vault_credential` will still behave as they did before:

```
PATCH /api/v2/job_templates/N/ {'credential': X, 'vault_credential': Y}
```

Under this model, when a job template with multiple vault credentials is updated in this way, the new underlying list will *only* contain the single Vault credential specified in the deprecated request.

GET requests to `/api/v2/job_templates/N/` have traditionally included a variety of metadata in the response through `related_fields`:

```
{
  "related": {
    ...
    "credential": "/api/v2/credentials/1/",
    "vault_credential": "/api/v2/credentials/3/",
    "extra_credentials": "/api/v2/job_templates/5/extra_credentials/",
  }
}
```

And `summary_fields`:

```
{
  "summary_fields": {
    "credential": {
      "description": "",
      "credential_type_id": 1,
      "id": 1,
      "kind": "ssh",
      "name": "Demo Credential"
    },
    "vault_credential": {
      "description": "",
      "credential_type_id": 3,
    }
  }
}
```

(continues on next page)

(continued from previous page)

```

        "id": 3,
        "kind": "vault",
        "name": "some-vault"
    },
    "extra_credentials": [
        {
            "description": "",
            "credential_type_id": 5,
            "id": 2,
            "kind": "aws",
            "name": "some-aws"
        },
        {
            "description": "",
            "credential_type_id": 10,
            "id": 4,
            "kind": "gce",
            "name": "some-gce"
        }
    ],
}
}

```

These metadata will continue to exist and function in a backwards-compatible way.

The `/api/v2/job_templates/N/extra_credentials` endpoint has been deprecated, but will also continue to exist and function in the same manner.

The `/api/v2/job_templates/N/launch/` endpoint also provides deprecated, backwards compatible support for specifying credentials at launch time via the `credential`, `vault_credential`, and `extra_credentials` fields:

```

POST /api/v2/job_templates/N/launch/ {'credential': A, 'vault_credential': B, 'extra_
↪credentials': [C, D]}

```

5.5 Multi-Vault Credentials

As it possible to assign multiple credentials to a job, starting with Ansible Tower 3.3, you can now specify multiple Vault credentials to decrypt when your job template runs. This functionality mirrors the support for [multiple vault passwords for a playbook run in Ansible 2.4 and later](#).

Vault credentials now have an optional field, `vault_id`, which is analogous to the `--vault-id` argument to `ansible-playbook`. To run a playbook which makes use of multiple vault passwords:

1. Create a Vault credential in Tower for each vault password; specify the Vault ID as a field on the credential and input the password (which will be encrypted and stored).
2. Assign multiple vault credentials to the job template via the new credentials endpoint:

```

POST /api/v2/job_templates/N/credentials/

{
    'associate': true,
    'id': X
}

```

Alternatively, you can perform the same assignment in the Tower User Interface in the *Create Credential* page:

In the above example, the credential created specifies the secret to be used by its Vault Identifier (“first”) and password pair. When this credential is used in a Job Template, as in the example below, it will only decrypt the secret associated with the “first” Vault ID:

If you have a playbook that is setup the traditional way with all the secrets in one big file without distinction, then leave the **Vault Identifier** field blank when setting up the Vault credential:

5.5.1 Prompted Vault Credentials


Passwords for Vault credentials that are marked with “Prompt on launch”, the launch endpoint of any related Job Templates will communicate necessary Vault passwords via the `passwords_needed_to_start` key:

```
GET /api/v2/job_templates/N/launch/
{
  'passwords_needed_to_start': [
    'vault_password.X',
    'vault_password.Y',
  ]
}
```

X and Y in the above example are primary keys of the associated Vault credentials.







```
POST /api/v2/job_templates/N/launch/
{
  'credential_passwords': {
    'vault_password.X': 'first-vault-password'
    'vault_password.Y': 'second-vault-password'
  }
}
```

MANAGEMENT JOBS

Management Jobs assist in the cleaning of old data from Tower, including system tracking information, job histories, and activity streams. You can use this if you have specific retention policies or need to decrease the storage used by your Tower database. From the Settings () menu, click on **Management Jobs**.

MANAGEMENT JOBS

MANAGEMENT JOBS 2

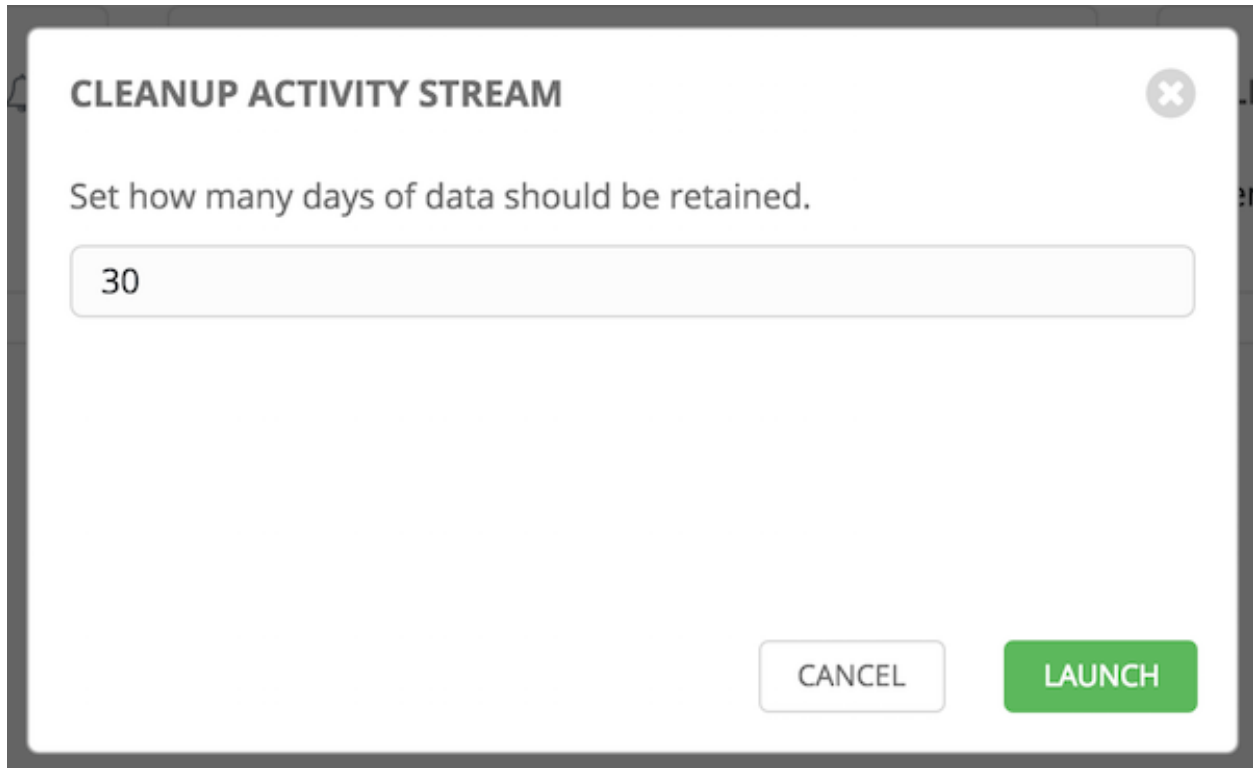
<p>Cleanup Activity Stream   </p> <p>Remove activity stream history</p>	<p>Cleanup Job Details   </p> <p>Remove job history</p>
-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Job types are available for you to schedule and launch:

- **Cleanup Activity Stream:** Remove activity stream history older than a specified number of days
- **Cleanup Job Details:** Remove job history older than a specified number of days


6.1 Removing Old Activity Stream Data

To remove older activity stream data, click on the launch () button beside **Cleanup Activity Stream**.

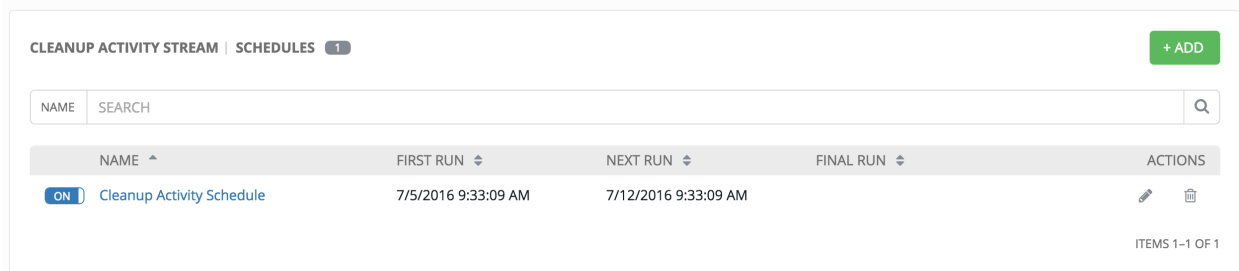


Enter the number of days of data you would like to save and click **Launch**.

6.1.1 Scheduling


To review or set a schedule for purging data marked for deletion, click on the  button.

[SETTINGS](#) / [MANAGEMENT JOBS](#) / [SCHEDULES](#)



Note that you can turn this scheduled management job on and off easily using the **ON/OFF** toggle button to the left of the Job Name.

Click on the Job Name, in this example “Cleanup Activity Schedule”, to review or edit the schedule settings. You can

also use the  button to create a new schedule for this management job.

Enter the appropriate details into the following fields and click **Save**:

- Name (required)

- Start Date (required)
- Start Time (required)
- Local Time Zone (the entered Start Time should be in this timezone)
- Repeat Frequency (the appropriate options display as the update frequency is modified.)

The **Details** tab displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

SETTINGS / MANAGEMENT JOBS / SCHEDULES / EDIT SCHEDULED JOB

CLEANUP ACTIVITY SCHEDULE
✕

*** NAME**

*** START DATE (MM/DD/YYYY)**

*** START TIME (HH24:MM:SS)**

*** LOCAL TIME ZONE**

*** REPEAT FREQUENCY**

*** DAYS OF DATA TO KEEP**

FREQUENCY DETAILS

*** EVERY**

 WEEKS

*** ON DAYS**

 SUN MON TUE WED THU FRI SAT

*** END**

SCHEDULE DESCRIPTION

every week on Tuesday


OCCURRENCES (Limited to first 10) DATE FORMAT LOCAL TIME UTC

```

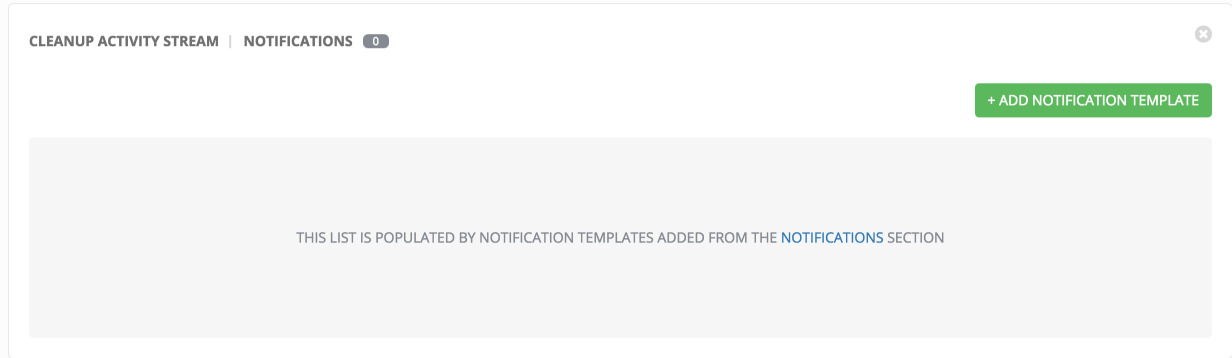
07/05/2016 09:33:09 EDT
07/12/2016 09:33:09 EDT
07/19/2016 09:33:09 EDT
07/26/2016 09:33:09 EDT
08/02/2016 09:33:09 EDT
08/09/2016 09:33:09 EDT
08/16/2016 09:33:09 EDT
08/23/2016 09:33:09 EDT
08/30/2016 09:33:09 EDT
09/06/2016 09:33:09 EDT
          
```

Note: Jobs are scheduled in UTC. Repeating jobs that runs at a specific time of day may move relative to a local timezone when Daylight Saving Time shifts occur.

6.1.2 Notifications

To set or review notifications associated with this management job, click the Notifications () icon.

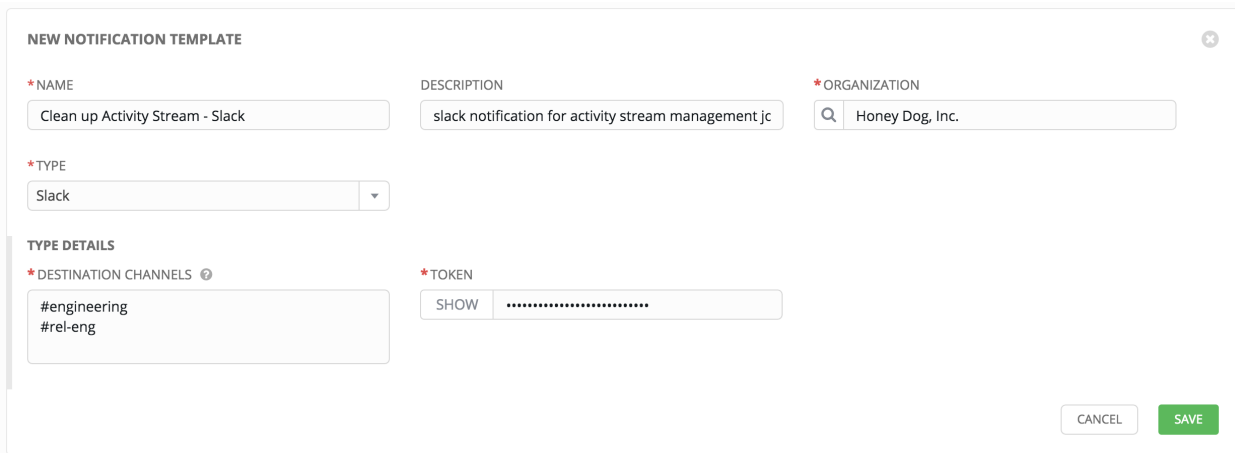
SETTINGS / MANAGEMENT JOBS / NOTIFICATIONS



+ ADD NOTIFICATION TEMPLATE


Click the **+ ADD NOTIFICATION TEMPLATE** button to create a new notification. Notification types include:

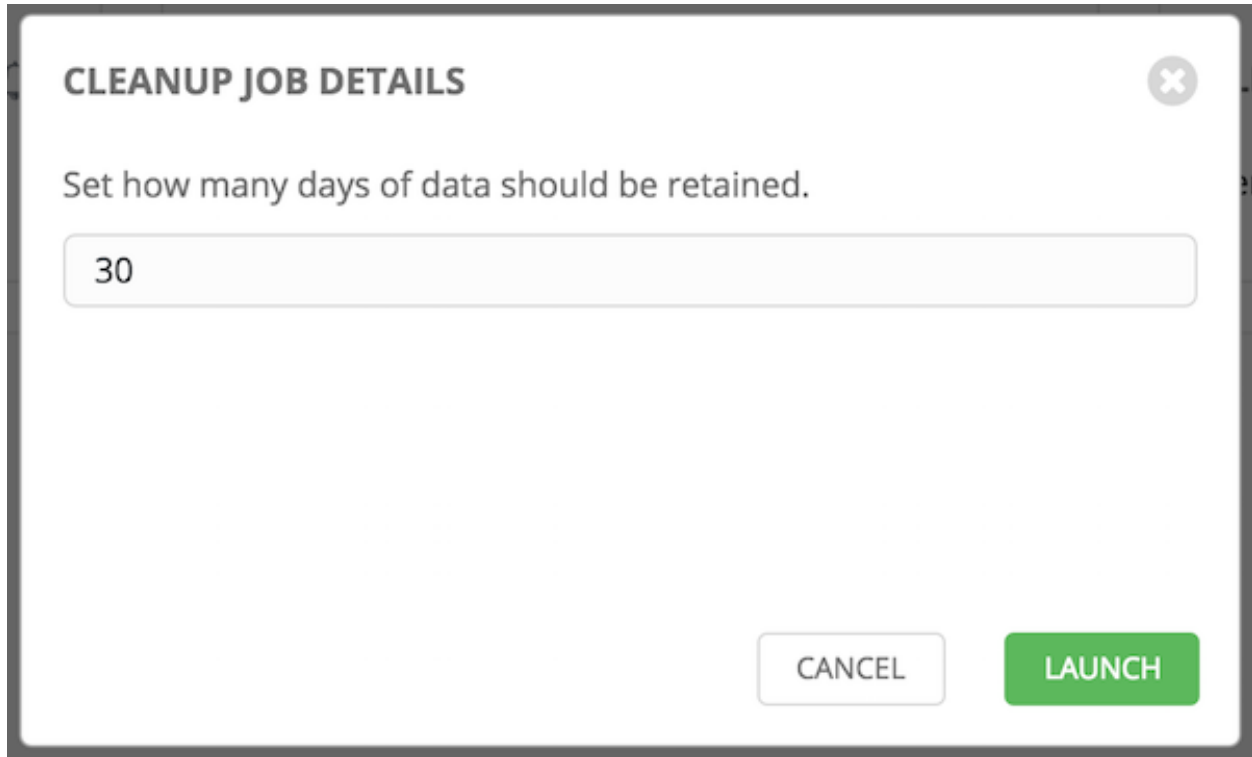
- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC
- Mattermost
- Rocket.Chat



Refer to [Notifications](#) in the *Ansible Tower User Guide* for more information.

6.2 Removing Old Job History

To remove job history older than a specified number of days, click on the launch () button beside **Cleanup Job Details**.



CLEANUP JOB DETAILS ✕

Set how many days of data should be retained.

CANCEL **LAUNCH**

Enter the number of days of data you would like to save and click **Launch**.

You can review or set a schedule for cleaning up old job history by performing the same procedure described for activity stream management jobs. See [Scheduling](#) for detail.

You can also set or review notifications associated with this management job the same way as described in [Notifications](#) for activity stream management jobs, and refer to [Notifications](#) in the *Ansible Tower User Guide* for more detail.

CLUSTERING

Ansible Tower 3.1 introduces Clustering as an alternate approach to redundancy, replacing the redundancy solution configured with the active-passive nodes that involves primary and secondary instances. For versions older than 3.1, refer to the older versions of this chapter of the *Ansible Tower Administration Guide*.

Clustering is sharing load between hosts. Each instance should be able to act as an entry point for UI and API access. This should enable Tower administrators to use load balancers in front of as many instances as they wish and maintain good data visibility.

Note: Load balancing is optional and is entirely possible to have ingress on one or all instances as needed.

Each instance should be able to join the Tower cluster and expand its ability to execute jobs. This is a simple system where jobs can and will run anywhere rather than be directed on where to run. Ansible Tower 3.2 introduced the ability for clustered instances to be grouped into different pools/queues.

Instances can be grouped into one or more Instance Groups. Instance groups can be assigned to one or more of the resources listed below.

- Organizations
- Inventories
- Job Templates

When a job associated with one of the resources executes, it will be assigned to the instance group associated with the resource. During the execution process, instance groups associated with Job Templates are checked before those associated with Inventories. Similarly, instance groups associated with Inventories are checked before those associated with Organizations. Thus, Instance Group assignments for the three resources form a hierarchy: Job Template > Inventory > Organization.

Ansible Tower 3.3 adds support for container-based clusters using OpenShift or Kubernetes, meaning new Tower instances can be installed on these platforms without any variation or diversion in functionality. However, the OpenShift Tower configuration has some differences that must be considered. For more detail about Tower OpenShift, see the *OpenShift Deployment and Configuration* section of this *Ansible Tower Administration Guide*.

Supported Operating Systems

The following operating systems are supported for establishing a clustered environment:

- RHEL-7 (can be either RHEL-7 or Centos-7 instances)
- Ubuntu 16
- Isolated instances can be installed only on RHEL-7 at this time

Note: Isolated instances are not supported in conjunction with running Ansible Tower in OpenShift.

7.1 Setup Considerations

This section covers initial setup of clusters only. For upgrading an existing cluster, refer to [Upgrade Planning](#) of the *Ansible Tower Upgrade and Migration Guide*.

Important considerations to note in the new clustering environment:

- PostgreSQL is still a standalone instance and is not clustered. Tower does not manage replica configuration or database failover (if the user configures standby replicas).
- The number of instances in a cluster should always be an odd number and it is **strongly recommended** that a minimum of three Tower instances be in a cluster and the supported maximum is 20. When it gets that high, the requirement for odd number of nodes becomes less important.
- All instances should be reachable from all other instances and they should be able to reach the database. It is also important for the hosts to have a stable address and/or hostname (depending on how the Tower host is configured).
- All instances must be geographically collocated, with reliable low-latency connections between instances.
- RabbitMQ is the cornerstone of Tower's clustering system. A lot of the configuration requirements and behavior is dictated by its needs. Therefore, customization beyond Tower's setup playbook is limited. Each Tower instance has a deployment of RabbitMQ that will cluster with the other instances' RabbitMQ instances.
- For purposes of upgrading to a clustered environment, your primary instance must be part of the `tower` group in the inventory *AND* it needs to be the first host listed in the `tower` group.
- Manual projects must be manually synced to all instances by the customer, and updated on all instances at once.
- There is no concept of primary/secondary in the new Tower system. All systems are primary.
- Setup playbook changes to configure RabbitMQ and provide the type of network the hosts are on.
- The `inventory` file for Tower deployments should be saved/persisted. If new instances are to be provisioned, the passwords and configuration options, as well as host names, must be made available to the installer.
- Do not create a group named `instance_group_tower`.
- Do not name any instance the same as a group name.

7.2 Install and Configure

Provisioning new instances involves updating the `inventory` file and re-running the setup playbook. It is important that the `inventory` file contains all passwords and information used when installing the cluster or other instances may be reconfigured. The current standalone instance configuration does not change for a 3.1 or later deployment. The `inventory` file does change in some important ways:

- Since there is no primary/secondary configuration, those inventory groups go away and are replaced with a single inventory group, `tower`.
- You may optionally define other groups and group instances in those groups. These groups should be prefixed with `instance_group_`. Instances are not required to be in the `tower` group alongside other `instance_group_` groups, but one instance **must** be present in the `tower` group. Technically, `tower` is a group like any other `instance_group_` group, but it must always be present, and if a specific group is

not associated with a specific resource, then job execution will always fall back to the `tower` group. In 3.3, the `tower` instance group always exists (it cannot be deleted nor renamed).

Note: All instances are responsible for various housekeeping tasks related to task scheduling, like determining where jobs are supposed to be launched and processing playbook events, as well as periodic cleanup.

```
[tower]
hostA
hostB
hostC

[instance_group_east]
hostB
hostC

[instance_group_west]
hostC
hostD
```

Note: If no groups are selected for a resource then the `tower` group is used, but if any other group is selected, then the `tower` group will not be used in any way.

The database group remains for specifying an external Postgres. If the database host is provisioned separately, this group should be empty:

```
[tower]
hostA
hostB
hostC

[database]
hostDB
```

It is common to provision Tower instances externally, but it is best to reference them by internal addressing. This is most significant for RabbitMQ clustering where the service is not available at all on an external interface. For this purpose, it is necessary to assign the internal address for RabbitMQ links as such:

```
[tower]
hostA rabbitmq_host=10.1.0.2
hostB rabbitmq_host=10.1.0.3
hostC rabbitmq_host=10.1.0.4
```

Note: The number of instances in a cluster should always be an odd number and it is **strongly recommended** that a minimum of three Tower instances be in a cluster and the supported maximum is 20. When it gets that high, the requirement for odd number of nodes becomes less important.

- The `redis_password` field is removed from `[all:vars]`
- Fields for RabbitMQ are as follows:
 - `rabbitmq_port=5672`: RabbitMQ is installed on each instance and is not optional, it is also not possible to externalize it. This setting configures what port it listens on.

- `rabbitmq_vhost=tower`: Controls the setting for which Tower configures a RabbitMQ virtualhost to isolate itself.
- `rabbitmq_username=tower` and `rabbitmq_password=tower`: Each instance and each instance's Tower instance are configured with these values. This is similar to Tower's other uses of usernames/passwords.
- `rabbitmq_cookie=<somevalue>`: This value is unused in a standalone deployment but is critical for clustered deployments. This acts as the secret key that allows RabbitMQ cluster members to identify each other.
- `rabbitmq_enable_manager`: Set this to `true` to expose the RabbitMQ Management Web Console on each instance.

7.2.1 RabbitMQ Default Settings

The following configuration shows the default settings for RabbitMQ:

```
rabbitmq_port=5672
rabbitmq_vhost=tower
rabbitmq_username=tower
rabbitmq_password=''
rabbitmq_cookie=cookiemonster
```

Note: `rabbitmq_cookie` is a sensitive value, it should be treated like the `secret` key in Tower.

7.2.2 Instances and Ports Used by Tower

Ports and instances used by Tower are as follows:

- 80, 443 (normal Tower ports)
- 22 (ssh)
- 5432 (database instance - if the database is installed on an external instance, needs to be opened to the tower instances)

Clustering/RabbitMQ ports:

- 4369, 25672 (ports specifically used by RabbitMQ to maintain a cluster, needs to be open between each instance)
- 15672 (if the RabbitMQ Management Interface is enabled, this port needs to be opened (optional))

7.3 Configuring Instances and Instance Groups from the API

Instance groups can be created by POSTing to `/api/v2/instance_groups` as a system administrator.

Once created, instances can be associated with an instance group with:

```
HTTP POST /api/v2/instance_groups/x/instances/ {'id': y}`
```

An instance that is added to an instance group will automatically reconfigure itself to listen on the group's work queue. See the following section, *Instance group policies*, for more details.

7.4 Instance group policies

Starting with Ansible Tower 3.3, users have the ability to configure Tower instances to automatically join Instance Groups when they come online by defining a policy. These policies are evaluated for every new instance that comes online.

Instance Group Policies are controlled by three optional fields on an Instance Group:

- `policy_instance_percentage`: This is a number between 0 - 100. It guarantees that this percentage of active Tower instances will be added to this Instance Group. As new instances come online, if the number of Instances in this group relative to the total number of instances is less than the given percentage, then new ones will be added until the percentage condition is satisfied.
- `policy_instance_minimum`: This policy attempts to keep at least this many instances in the Instance Group. If the number of available instances is lower than this minimum, then all instances will be placed in this Instance Group.
- `policy_instance_list`: This is a fixed list of instance names to always include in this Instance Group.

The Instance Groups list view from the Ansible Tower User Interface provides a summary of the capacity levels for each instance group according to instance group policies:

INSTANCE GROUPS				+				
SEARCH				Q	KEY			
Group Eleven								
INSTANCES	1	RUNNING JOBS	0	TOTAL JOBS	0	USED CAPACITY	0%	
Grouper Instance Group								
INSTANCES	1	RUNNING JOBS	0	TOTAL JOBS	0	USED CAPACITY	0%	
tower								
INSTANCES	1	RUNNING JOBS	0	TOTAL JOBS	19	USED CAPACITY	0%	

ITEMS 1 - 3

7.4.1 Notable policy considerations

- `policy_instance_percentage` and `policy_instance_minimum` both set minimum allocations. The rule that results in more instances assigned to the group will take effect. For example, if you have a `policy_instance_percentage` of 50% and a `policy_instance_minimum` of 2 and you start 6 instances, 3 of them would be assigned to the Instance Group. If you reduce the number of total instances in the cluster to 2, then both of them would be assigned to the Instance Group to satisfy `policy_instance_minimum`. This way, you can set a lower bound on the amount of available resources.
- Policies do not actively prevent instances from being associated with multiple Instance Groups, but this can effectively be achieved by making the percentages add up to 100. If you have 4 instance groups, assign each a percentage value of 25 and the instances will be distributed among them with no overlap.

7.4.2 Manually pinning instances to specific groups

If you have a special instance which needs to be exclusively assigned to a specific Instance Group but don't want it to automatically join other groups via “percentage” or “minimum” policies:

1. Add the instance to one or more Instance Groups' `policy_instance_list`
2. Update the instance's `managed_by_policy` property to be `False`.

This will prevent the Instance from being automatically added to other groups based on percentage and minimum policy; it will only belong to the groups you've manually assigned it to:

```
HTTP PATCH /api/v2/instance_groups/N/
{
  "policy_instance_list": ["special-instance"]
}

HTTP PATCH /api/v2/instances/X/
{
  "managed_by_policy": False
}
```

7.5 Isolated Instance Groups

Ansible Tower versions 3.2 and later added the ability to optionally define isolated groups inside security-restricted networking zones from which to run jobs and ad hoc commands. Instances in these groups will not have a full installation of Tower, but will have a minimal set of utilities used to run jobs. Isolated groups must be specified in the inventory file prefixed with `isolated_group_`. Below is an example of an inventory file for an isolated instance group.

```
[tower]
towerA
towerB
towerC

[instance_group_security]
towerB
towerC

[isolated_group_govcloud]
isolatedA
isolatedB

[isolated_group_govcloud:vars]
controller=security
```

In the isolated instance group model, “controller” instances interact with “isolated” instances via a series of Ansible playbooks over SSH. At installation time, by default, a randomized RSA key is generated and distributed as an authorized key to all “isolated” instances. The private half of the key is encrypted and stored within the Tower database, and is used to authenticate from “controller” instances to “isolated” instances when jobs are run.

When a job is scheduled to run on an “isolated” instance:

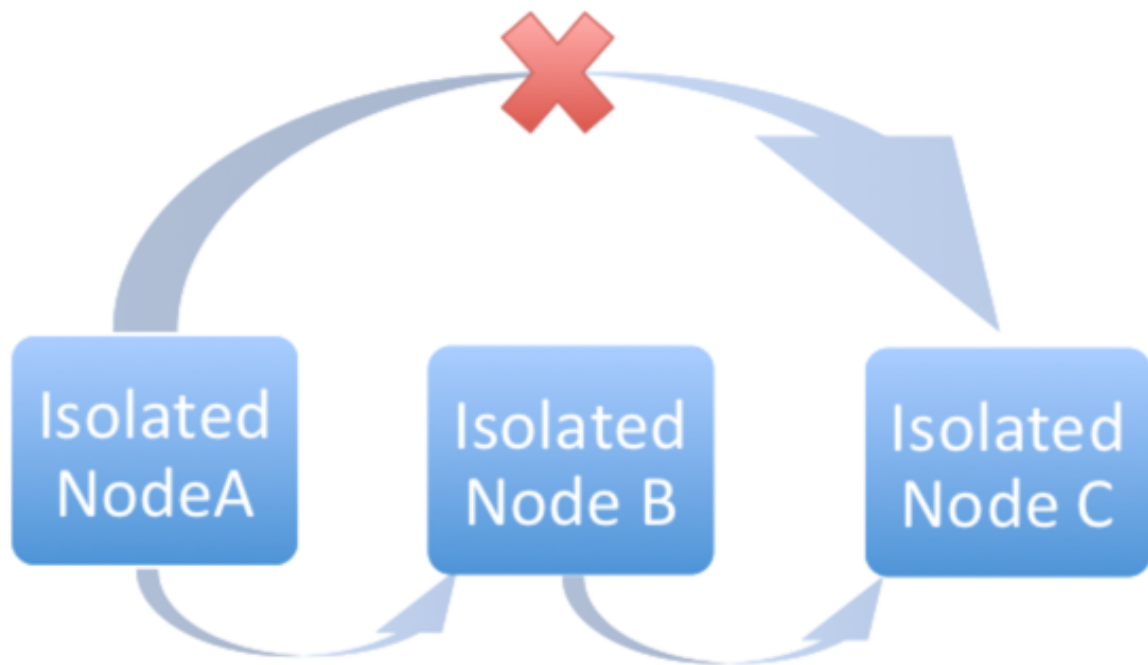
- The “controller” instance compiles metadata required to run the job and copies it to the “isolated” instance.
- Once the metadata has been synchronized to the isolated host, the “controller” instance starts a process on the “isolated” instance, which consumes the metadata and starts running `ansible/ansible-playbook`. As

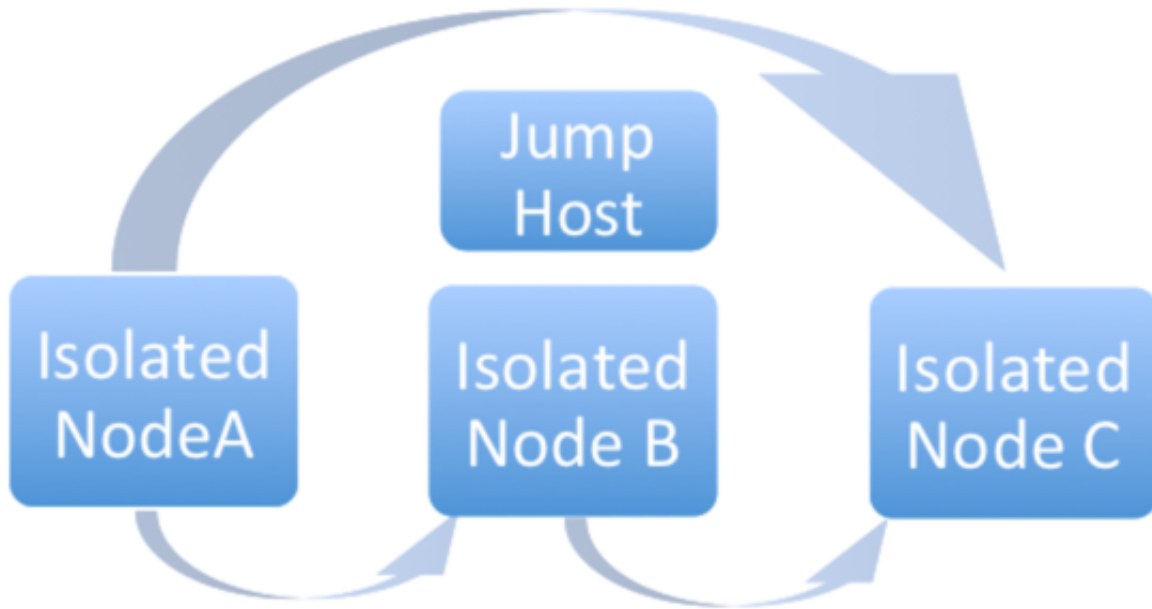
the playbook runs, job artifacts (such as stdout and job events) are written to disk on the “isolated” instance.

- While the job runs on the “isolated” instance, the “controller” instance periodically copies job artifacts (stdout and job events) from the “isolated” instance. It consumes these until the job finishes running on the “isolated” instance.

Note: Controller nodes fail if they go offline in the middle of an isolated run. If a Tower node restarts, or the dispatcher stops during playbook runs, jobs running on that node fails and won’t start again when the dispatcher comes online.

Isolated groups (nodes) may be created in a way that allow them to exist inside of a VPC with security rules that only permit the instances in its controller group to access them; only ingress SSH traffic from “controller” instances to “isolated” instances is required. When provisioning isolated nodes, your install machine needs to be able to have connectivity to the isolated nodes. In cases where an isolated node is not directly accessible but can be reached indirectly through other hosts, you can designate a “jump host” by using ProxyCommand in your SSH configuration to specify the jump host and then run the installer.





The recommended system configurations with isolated groups are as follows:

- Do not create a group named `isolated_group_tower`.
- Do not put any isolated instances inside the tower group or other ordinary instance groups.
- Define the controller variable as either a group variable or as a host variable on all the instances in the isolated group. Do not allow isolated instances in the same group to have a different value for this variable - the behavior in this case cannot be predicted.
- Do not put an isolated instance in more than one isolated group.
- Do not put an instance in both ordinary groups and isolated groups.
- Do not use fact caching with isolated instances.

7.5.1 Optional SSH Authentication

For users who wish to manage SSH authentication from “controller” nodes to “isolated” nodes via some system outside of Tower (such as externally-managed passwordless SSH keys), this behavior can be disabled by unsetting two Tower API settings values:

```
HTTP PATCH /api/v2/settings/jobs/ {'AWX_ISOLATED_PRIVATE_KEY': '', 'AWX_ISOLATED_
↪PUBLIC_KEY': ''}
```


7.6 Status and Monitoring via Browser API

Tower itself reports as much status as it can via the Browsable API at `/api/v2/ping` in order to provide validation of the health of the cluster, including:

- The instance servicing the HTTP request
- The timestamps of the last heartbeat of all other instances in the cluster
- Instance Groups and Instance membership in those groups

View more details about Instances and Instance Groups, including running jobs and membership information at `/api/v2/instances/` and `/api/v2/instance_groups/`.

7.7 Instance Services and Failure Behavior

Each Tower instance is made up of several different services working collaboratively:

- HTTP Services - This includes the Tower application itself as well as external web services.
- Callback Receiver - Receives job events from running Ansible jobs.
- Dispatcher - The worker queue that processes and runs all jobs.
- RabbitMQ - This message broker is used as a signaling mechanism for the dispatcher as well as any event data propagated to the application.
- Memcached - local caching service for the instance it lives on.

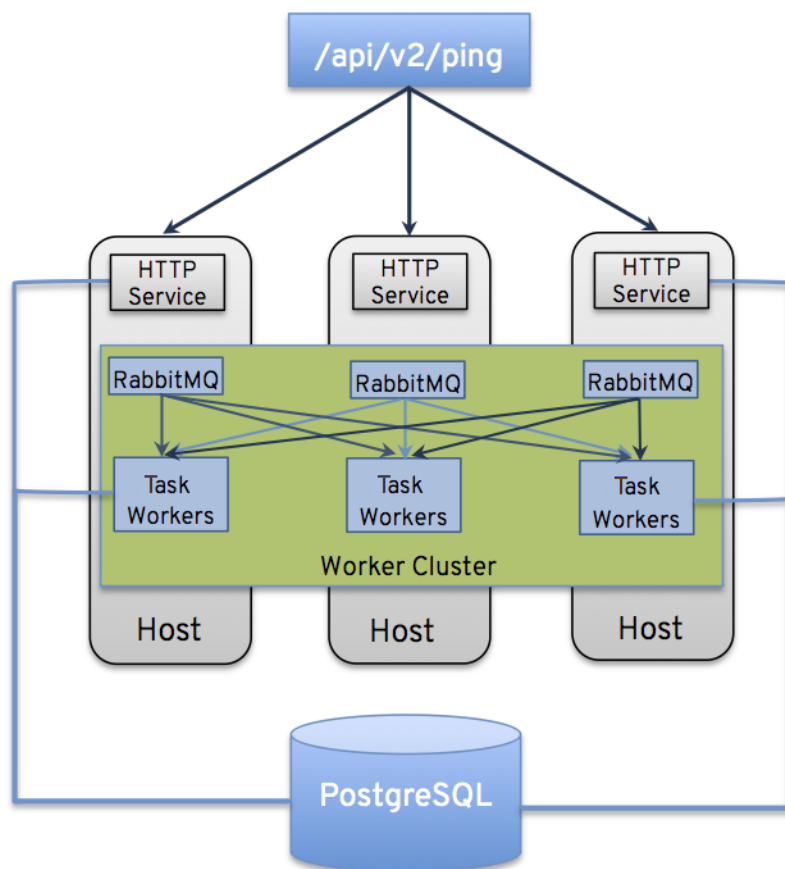
Tower is configured in such a way that if any of these services or their components fail, then all services are restarted. If these fail sufficiently often in a short span of time, then the entire instance will be placed offline in an automated fashion in order to allow remediation without causing unexpected behavior.

For backing up and restoring a clustered environment, refer to *Backup and Restore for Clustered Environments* section.

7.8 Job Runtime Behavior

The way jobs are run and reported to a ‘normal’ user of Tower does not change. On the system side, some differences are worth noting:

- When a job is submitted from the API interface it gets pushed into the dispatcher queue on RabbitMQ. A single RabbitMQ instance is the responsible master for individual queues but each Tower instance will connect to and receive jobs from that queue using a particular scheduling algorithm. Any instance in the cluster is just as likely to receive the work and execute the task. If a instance fails while executing jobs, then the work is marked as permanently failed.



- If a cluster is divided into separate instance groups, then the behavior is similar to the cluster as a whole. If two instances are assigned to a group then either one is just as likely to receive a job as any other in the same group.
- As Tower instances are brought online, it effectively expands the work capacity of the Tower system. If those instances are also placed into instance groups, then they also expand that group's capacity. If an instance is performing work and it is a member of multiple groups, then capacity will be reduced from all groups for which it is a member. De-provisioning an instance will remove capacity from the cluster wherever that instance was assigned. See *Deprovision Instances and Instance Groups* in the next section for more details.

Note: Not all instances are required to be provisioned with an equal capacity.

- Project updates behave differently than they did before. Previously, they were ordinary jobs that ran on a single instance. It is now important that they run successfully on any instance that could potentially run a job. Projects will now sync themselves to the correct version on the instance immediately prior to running the job.
- If an instance group is configured but all instances in that group are offline or unavailable, any jobs that are launched targeting only that group will be stuck in a waiting state until instances become available. Fallback or backup resources should be provisioned to handle any work that might encounter this scenario.

7.8.1 Control Where a Job Runs

By default, when a job is submitted to the tower queue, it can be picked up by any of the workers. However, you can control where a particular job runs, such as restricting the instances from which a job runs on. If any of the job template, inventory, or organization has instance groups associated with them, a job ran from that job template will not be eligible for the default behavior. That means that if all of the instances inside of the instance groups associated with these 3 resources are out of capacity, the job will remain in the pending state until capacity becomes available.

The order of preference in determining which instance group to submit the job to is as follows:

1. job template
2. inventory
3. organization (by way of project)

If instance groups are associated with the job template, and all of these are at capacity, then the job will be submitted to instance groups specified on inventory, and then organization. Jobs should execute in those groups in preferential order as resources are available.

The global `tower` group can still be associated with a resource, just like any of the custom instance groups defined in the playbook. This can be used to specify a preferred instance group on the job template or inventory, but still allow the job to be submitted to any instance if those are out of capacity.

In order to support temporarily taking an instance offline, there is a property enabled defined on each instance. When this property is disabled, no jobs will be assigned to that instance. Existing jobs will finish, but no new work will be assigned.

Job Run Examples

As an example, by associating `group_a` with a Job Template and also associating the `tower` group with its inventory, you allow the `tower` group to be used as a fallback in case `group_a` gets out of capacity.

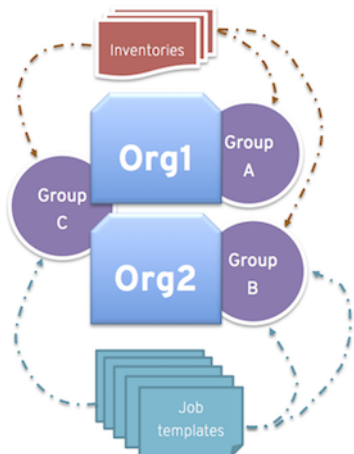
In addition, it is possible to not associate an instance group with one resource but designate another resource as the fallback. For example, not associating an instance group with a job template and have it fall back to the inventory and/or the organization's instance group.

This presents two other great use cases:

1. Associating instance groups with an inventory (omitting assigning the job template to an instance group) will allow the user to ensure that any playbook run against a specific inventory will run only on the group associated with it. This can be super useful in the situation where only those instances have a direct link to the managed nodes.
2. An administrator can assign instance groups to organizations. This effectively allows the administrator to segment out the entire infrastructure and guarantee that each organization has capacity to run jobs without interfering with any other organization's ability to run jobs.

Likewise, an administrator could assign multiple groups to each organization as desired, as in the following scenario:

- There are three instance groups: A, B, and C. There are two organizations: Org1 and Org2.
- The administrator assigns group A to Org1, group B to Org2 and then assign group C to both Org1 and Org2 as an overflow for any extra capacity that may be needed.
- The organization administrators are then free to assign inventory or job templates to whichever group they want (or just let them inherit the default order from the organization).



Arranging resources in this way offers a lot of flexibility. Also, you can create instance groups with only one instance, thus allowing you to direct work towards a very specific Host in the Tower cluster.

7.9 Deprovision Instances and Instance Groups

Re-running the setup playbook does not automatically deprovision instances since clusters do not currently distinguish between an instance that was taken offline intentionally or due to failure. Instead, shut down all services on the Tower instance and then run the deprovisioning tool from any other instance:

1. Shut down the instance or stop the service with the command, `ansible-tower-service stop`.
2. Run the deprovision command `$ awx-manage deprovision_instance --hostname=<name used in inventory file>` from another instance to remove it from the Tower cluster registry AND the RabbitMQ cluster registry.

Example: `awx-manage deprovision_instance --hostname=hostB`

Similarly, deprovisioning instance groups in Tower does not automatically deprovision or remove instance groups, even though re-provisioning will often cause these to be unused. They may still show up in API endpoints and stats monitoring. These groups can be removed with the following command:

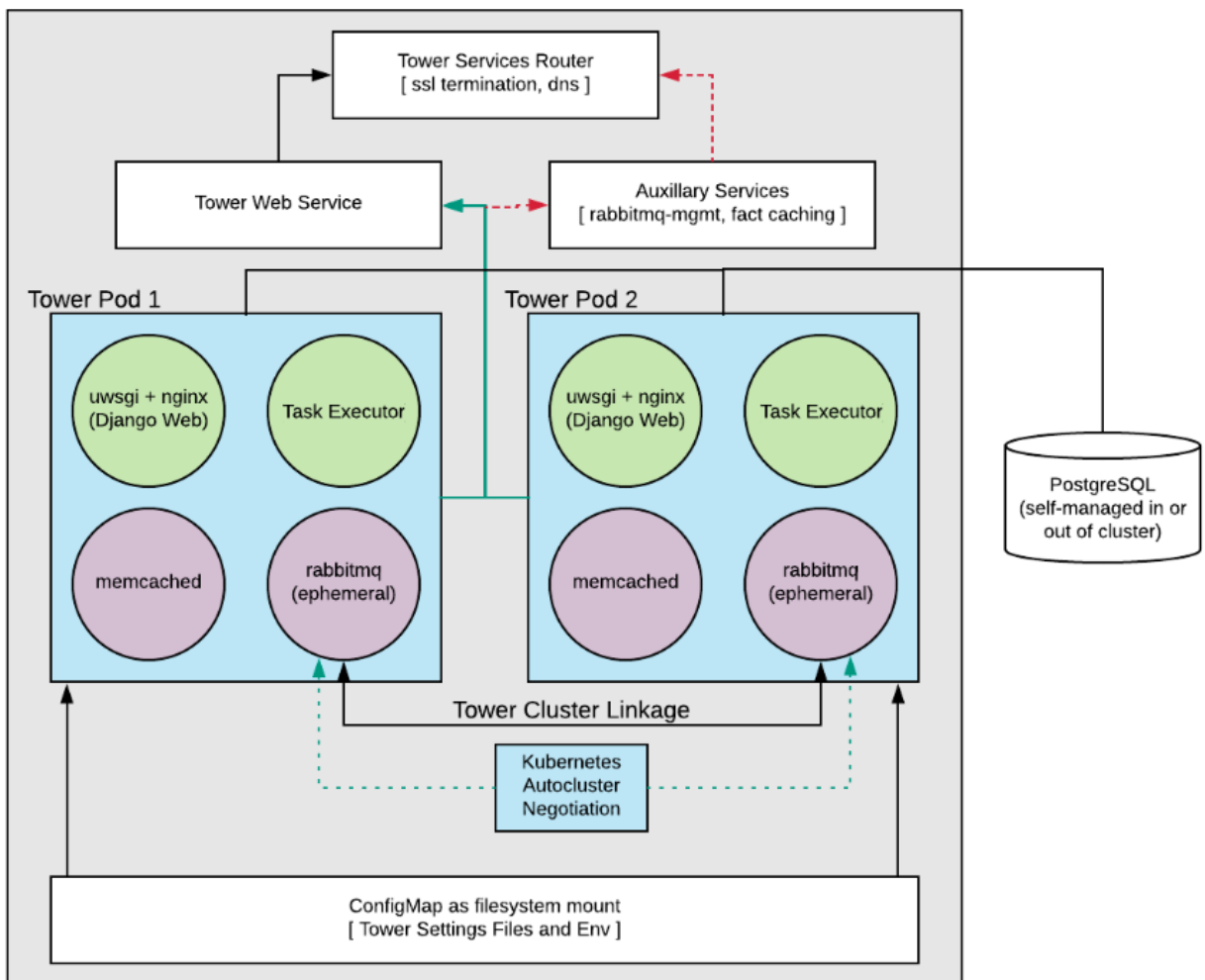
Example: `awx-manage unregister_queue --queuename=<name>`

Removing an instance's membership from an instance group in the inventory file and re-running the setup playbook does not ensure the instance won't be added back to a group. To be sure that an instance will not be added back to a group, remove via the API and also remove it in your inventory file, or you can stop defining instance groups in the inventory file altogether. You can also manage instance group topology through the Ansible Tower User Interface. For more information on managing instance groups in the UI, refer to [Instance Groups](#) in the *Ansible Tower User Guide*.

OPENSIFT DEPLOYMENT AND CONFIGURATION

Ansible Tower supports container-based clusters running on OpenShift. This section provides a high-level overview of OpenShift and Tower Pod configuration, notably the following:

- The main Differences in standard Tower vs OpenShift Tower (i.e., auto-removal of instances)
- Tower deploys as a single pod first and can scale up after migrations
- Migrations run in the task-runner pod



8.1 Tower and OpenShift Basics

The Tower OpenShift documentation assumes an understanding of how to use OpenShift at an administrative level and should include some experience maintaining container based infrastructure. The differences are:

- Standalone Tower and OpenShift Tower use different installers. For the OpenShift installer, go to http://releases.ansible.com/ansible-tower/setup_openshift.
- Tower (via RabbitMQ) links to OpenShift itself in order to facilitate scaling up and down without requiring you to manually execute the playbook (to bring up new nodes) or run management commands in the shell (to take nodes offline purposefully). The user can configure the Tower StatefulSet once the system is up to add more or remove extra Tower Pods.
- Tower pods are configured without HTTPs and the installer will configure an OpenShift Route which will handle SSL termination and distribute requests to all Tower Pods. This is somewhat of an internal OpenShift load balancer
- Database migrations run as part of the process of bringing up the task executor container within the pod (see diagram) and thus will likely happen after the playbook has completed.
- Capacity / Performance Detection (see the section on *Resource Requests and Request Planning*).
- Isolated instances are not supported in conjunction with running Ansible Tower in OpenShift.

8.2 Configuration Options

Requirements

- OpenShift 3.6+
- **Per pod default resource requirements:**
 - 6GB RAM
 - 3CPU cores
- Openshift command-line tool (oc) on the machine running the installer
- A setup and running Openshift cluster
- Admin privileges for the account running the Openshift Installer

8.3 Basic Configuration

An OpenShift install requires the following parameters to be set:

- `openshift_host`
- `openshift_project`
- `openshift_user`
- `openshift_password`
- `admin_password`
- `secret_key`
- `pg_username`

- `pg_password`
- `rabbitmq_password`
- `rabbitmq_erlang_cookie`

The Project will be created if it doesn't exist but the user given there should have either:

- The ability to create the project and populate it with Tower-needed pods

OR

- Access to create whatever pods are needed in the project, if it already exists

The password should be given on the command line as shown when executing the installer.

The `oc` command line client should be installed and available and the client version should match the server version.

The `secret-key`, `admin password`, and `postgresql username and password` should be populated in the inventory file prior to running the installer.

```
./setup_openshift.sh -e openshift_password=$OPENSIFT_PASSWORD -- -v
```

Note: Tower uses Bubblewrap (from Project Atomic) as a mechanism to give the (relatively) unprivileged `awx` user the ability to isolate Ansible processes from each other. There are certain privileges that need to be granted to the container that necessitates running the Tower web and task containers in privileged mode.

8.4 Resource Requests and Request Planning

Normally Tower examines the system that it runs on in order to determine what its own capacity is for running Jobs and performing background requests. On OpenShift this works differently since pods and containers will tend to coexist on systems. Pods can also migrate between hosts depending on current conditions (for instance, if the OpenShift cluster is being upgraded or is experiencing an outage).

It's common for Pods and Containers to Request the resources that they need. OpenShift then uses this information to decide Where things run (or even if they can run).

Tower will also use this information to configure its own capacity for how many (and the size of) individual jobs can be run.

Each Tower pod is made up of 4 containers (see *diagram*), each container is configured with a conservative default, but taken all together they can be somewhat substantial. These defaults are also configurable but it's helpful to know what effect that has on the Tower cluster.

The two most important values control the CPU and memory allocation for the task execution container. This container is the one that is actually responsible for launching jobs, as such these values directly control how many and what size jobs can run. The settings can be changed in the inventory and here are the default values:

```
task_cpu_request=1500
```

This is the amount of CPU to dedicate, the value of 1500 refers to how OpenShift itself views CPU requests (see https://docs.openshift.com/container-platform/3.9/dev_guide/compute_resources.html#dev-cpu-requests) (for value meanings see: https://docs.openshift.com/container-platform/3.9/dev_guide/compute_resources.html#dev-compute-resources)

1500 is 1500 millicores which translates to roughly 1.5 CPU Cores.

This value is used to configure the Tower capacity in the following way:

```
((task_cpu_request / 1000) * 4)
```

Which is to say that, by default, Tower in OpenShift (when configured entirely for cpu-based algorithm) can run at most **6** simultaneous forks.

The other value that can be tuned:

```
task_mem_request=2 - This is the amount of memory to dedicate (in gigabytes).
```

This value is used to configure the Tower capacity in the following way

```
((task_mem_request * 1024) / 100)
```

Which is to say that, by default, Tower can run at most **40** simultaneous forks when configured for mem-based algorithm.

For the default resource requests, see `roles/kubernetes/defaults/main.yml`.

All together the default requested resources for a single Tower pod total to:

- 3 CPU Cores
- 6 GB memory

The OpenShift instances that you want to run Tower on should at least match that. If the defaults are changed then the system will need to be updated to match the new requirements.

Note: If other Pods are running on the OpenShift instance or the systems are too small to meet these requirements then Tower may not be able to run anywhere. Refer to [Capacity Algorithm](#) for more detail.

8.5 Database Configuration and Usage

There are two methods for configuring the Tower PostgreSQL database for Tower running in Openshift:

- (Recommended) **Externally Managed Database** (not installed by the Tower setup playbook). The PostgreSQL server is installed before Tower either inside or outside of the Openshift cluster and Tower is configured to point at it
- PostgreSQL is installed in Openshift using the Tower Installer by providing a pre-created `PersistentVolumeClaim` and providing it the Tower install playbook inventory file as `pg_pvc_name`.

If you are installing Tower for demo/evaluation purposes you may set `openshift_pg_emptydir=true` and OpenShift will create a temporary volume for use by the pod.

Warning: This volume is temporary for demo/evaluation purposes only, and will be deleted when the pod is stopped.

8.6 Backup and Restore

You must backup and restore into the same version before upgrading. The process for backup and restore resembles that of traditional Tower. From the root of the installer directory, run:

```
./setup_openshift.sh -b # Backup
./setup_openshift.sh -r # Restore
```

Note: `configmap` will be recreated from values in the inventory file. The inventory file is included in backup tarball.

8.7 Upgrading

Prior to performing an upgrade, you must backup and restore into the same version before upgrading. To upgrade a Tower deployment in OpenShift, download the most recent installer from http://releases.ansible.com/ansible-tower/setup_openshift. Expect some downtime, just as traditional Tower installations.

8.8 Migrating

Tower supports migration from traditional setup to a setup in OpenShift, as outlined below:

1. First, upgrade your traditional Tower setup to the latest release of Ansible Tower (or to version 3.3 at minimum), using the normal upgrade procedure.
2. Download the OpenShift installer.
3. Edit the `inventory` file and change `pg_username`, `pg_password`, `pg_database`, and `pg_port` to point to the upgraded Tower database from your traditional Tower setup.
4. Run the OpenShift installer as normal.

8.9 Build custom virtual environments

It is possible to override the base container image to build custom virtual environments (*virtualenvs*). Overriding the base container is used for customization and custom virtualenv support or for local mirroring. If you want to use custom virtual environments with Tower deployed in OpenShift, you will need to customize the container image used by Tower.

Here is a Dockerfile that can be used as an example. This installs Ansible 2.3 into a custom virtual environment:

```
FROM registry.access.redhat.com/ansible-tower/ansible-tower:3.3.0
USER root
RUN mkdir -p /var/lib/awx/venv/ansible2.3
RUN virtualenv --system-site-packages /var/lib/awx/venv/ansible2.3
RUN cp -a /var/lib/awx/venv/ansible/lib64/python2.7/site-packages/* /var/lib/awx/venv/
↳ansible2.3/lib64/python2.7/site-packages/
RUN sh -c ". /var/lib/awx/venv/ansible2.3/bin/activate ; pip install ansible==2.3.3.0"
```

If you need to install other python dependencies (such as those for custom modules) you can add additional **RUN** commands to the docker file that activate the virtual environment and call `pip`.

Once the image is built, make sure that image is in your registry and that the OpenShift cluster and installer have access to it.

Override the following variables in `group_vars/all` in the OpenShift installer to point to the image you have pushed to your registry:

```
kubernetes_web_image: registry.example.com/my-custom-tower
kubernetes_task_image: registry.example.com/my-custom-tower
```

Note: The image must be tagged with 3.3.0 or pass version variable to installer to override.

When hosting all images in a local registry, such as offline installs, you'll need to include these other images:

```
kubernetes_rabbitmq_image: registry.example.com/ansible-tower-messaging
```

If mirroring the vanilla Red Hat images:

```
kubernetes_web_image: registry.example.com/ansible-tower
kubernetes_task_image: registry.example.com/ansible-tower
```

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 [The Setup Playbook](#) 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 ELB in front of Tower, HAProxy, Squid, and tinyproxy) via the `REMOTE_HOST_HEADERS` list variable in the System settings of the Configure Tower user interface. Prior to Ansible Tower 3.2, this setting was configured in the `remote_host_headers.py` file, which is no longer supplied with Tower installations.

The screenshot shows the 'SYSTEM' configuration page in Ansible Tower. It features several tabs: 'MISC. SYSTEM', 'ACTIVITY STREAM', and 'LOGGING'. The 'MISC. SYSTEM' tab is active. The page contains various settings, each with a 'REVERT' link. The 'REMOTE_HOST_HEADERS' setting is highlighted with a red rectangular box. It is currently set to 'REMOTE_ADDR, REMOTE_HOST'. Other visible settings include 'BASE URL OF THE TOWER HOST' (https://ec2-34-238-80-214.compute-1.amazonaws.com), 'ENABLE ADMINISTRATOR ALERTS' (ON), 'ALL USERS VISIBLE TO ORGANIZATION ADMINS' (ON), 'ORGANIZATION ADMINS CAN MANAGE USERS AND TEAMS' (ON), 'IDLE TIME FORCE LOG OUT' (1800), 'MAXIMUM NUMBER OF SIMULTANEOUS LOGGED IN SESSIONS' (-1), 'ENABLE HTTP BASIC AUTH' (ON), and 'ALLOW EXTERNAL USERS TO CREATE OAUTH2 TOKENS' (OFF). At the bottom right, there are 'CANCEL' and 'SAVE' buttons.

By default `REMOTE_HOST_HEADERS` is set to `REMOTE_ADDR, REMOTE_HOST`. To enable proxy server support, the `REMOTE_HOST_HEADERS` parameter should be defined with a simple comma-separated string, like the following: `REMOTE_HOST_HEADERS = HTTP_X_FORWARDED_FOR, REMOTE_ADDR, REMOTE_HOST`.

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/settings.py` file accordingly:

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

9.1 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_WHITELIST` setting via the settings API. Load balancers and hosts that are not on the list will result in a rejected request.

`PROXY_IP_WHITELIST` 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_WHITELIST` 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
- `PROXY_IP_WHITELIST` 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.

9.2 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
```

Note: If you find that your notifications are not working behind a proxy, refer to the knowledge base article, [How do I Use Ansible Tower Notifications from Behind a Proxy?](#)

TOWER LOGFILES

Tower logfiles have been consolidated and can be easily accessed from two centralized locations:

- `/var/log/tower/`
- `/var/log/supervisor/`

In the `/var/log/tower/` directory, you can view logfiles related to:

- `callback_receiver.log`
- `fact_receiver.log`
- `setup-XX-XX-XX-XX.log`
- `task_system.log`
- `tower.log`

In the `/var/log/supervisor/` directory, you can view logfiles related to:

- `awx-dispatcher.log`
- `awx-daphne.log`
- `awx-channels-worker.log`
- `awx-uwsgi.log`
- `supervisord.log`

The `/var/log/supervisor/` directory include `stdout` files for all services as well.

```
"Mooving around: Consolidated logfiles for easier access!"
```

```
  \
  \  ^ _ ^
    (oo) \_____
      (__) \       ) \ / \
           ||-----w |
           ||         ||
```

TOWER LOGGING AND AGGREGATION

Logging is a standalone feature introduced in Ansible Tower 3.1.0 that provides the capability to send detailed logs to several kinds of 3rd party external log aggregation services. Services connected to this data feed serve as a useful means in gaining insight into Tower usage or technical trends. The data can be used to analyze events in the infrastructure, monitor for anomalies, and correlate events from one service with events in another. The types of data that are most useful to Tower are job fact data, job events/job runs, activity stream data, and log messages. The data is sent in JSON format over a HTTP connection using minimal service-specific tweaks engineered in a custom handler or via an imported library. Tower discards any uncaptured data if the logging aggregator is down.

11.1 Loggers

Below are special loggers (except for `awx`, which constitutes generic server logs) that provide large amount of information in a predictable structured or semi-structured format, following the same structure as one would expect if obtaining the data from the API:

- `job_events`: Provides data returned from the Ansible callback module
- `activity_stream`: Displays the record of changes to the objects within the Ansible Tower application
- `system_tracking`: Provides data gathered by Ansible scan modules ran by scan job templates
- `awx`: Provides generic server logs, which include logs that would normally be written to a file. It contains the standard metadata that all logs have, except it only has the message from the log statement.

These loggers only use log-level of INFO, except for the `awx` logger, which may be any given level.

Additionally, the standard Tower logs are be deliverable through this same mechanism. It is apparent how to enable or disable each of these five sources of data without manipulating a complex dictionary in your local settings file, as well as adjust the log-level consumed from the standard Tower logs.

11.1.1 Log message schema

Common schema for all loggers:

- `cluster_host_id`: Unique identifier of the host within the Tower cluster
- `level`: Standard python log level, roughly reflecting the significance of the event All of the data loggers as a part of this feature use INFO level, but the other Tower logs will use different levels as appropriate
- `logger_name`: Name of the logger we use in the settings, for example, “`activity_stream`”
- `@timestamp`: Time of log
- `path`: File path in code where the log was generated

11.1.2 Activity stream schema

- (common): This uses all the fields common to all loggers listed above
- actor: Username of the user who took the action documented in the log
- changes: JSON summary of what fields changed, and their old/new values.
- operation: The basic category of the changed logged in the activity stream, for instance, “associate”.
- object1: Information about the primary object being operated on, consistent with what we show in the activity stream
- object2: If applicable, the second object involved in the action

11.1.3 Job event schema

This logger reflects the data being saved into job events, except when they would otherwise conflict with expected standard fields from the logger, in which case the fields are nested. Notably, the field host on the `job_event` model is given as `event_host`. There is also a sub-dictionary field, `event_data` within the payload, which contains different fields depending on the specifics of the Ansible event.

This logger also includes the common fields.

11.1.4 Scan / fact / system tracking data schema

These contain a detailed dictionary-type fields that are either services, packages, or files.

- (common): This uses all the fields common to all loggers listed above
- services: For services scans, this field is included and has keys based on the name of the service. **NOTE:** Periods are disallowed by elastic search in names, and are replaced with “_” by our log formatter
- package: Included for log messages from package scans
- files: Included for log messages from file scans
- host: Name of host scan applies to
- inventory_id: Inventory id host is inside of

11.1.5 Job status changes

This is intended to be a lower-volume source of information about changes in job states compared to job events, and also intended to capture changes to types of unified jobs other than job template based jobs.

In addition to common fields, these logs include fields present on the job model.

11.1.6 Tower logs

In addition to the common fields, this contains a `msg` field with the log message. Errors contain a separate `traceback` field. These logs can be enabled or disabled in the Configure Tower User Interface `ENABLE EXTERNAL LOGGING` setting.

11.1.7 Logging Aggregator Services

The logging aggregator service works with the following monitoring and data analysis systems:

- *Splunk*
- *Loggly*
- *Sumologic*
- *Elastic stack (formerly ELK stack)*

Splunk

Ansible Tower's Splunk logging integration uses the Splunk HTTP Collector. When configuring a SPLUNK logging aggregator, add the full URL to the HTTP Event Collector host, like in the following example:

```
https://yourtowerfqdn.com/api/v2/settings/logging
{
  "LOG_AGGREGATOR_HOST": "https://yoursplunk:8088/services/collector/event",
  "LOG_AGGREGATOR_PORT": null,
  "LOG_AGGREGATOR_TYPE": "splunk",
  "LOG_AGGREGATOR_USERNAME": "",
  "LOG_AGGREGATOR_PASSWORD": "$encrypted$",
  "LOG_AGGREGATOR_LOGGERS": [
    "awx",
    "activity_stream",
    "job_events",
    "system_tracking"
  ],
  "LOG_AGGREGATOR_INDIVIDUAL_FACTS": false,
  "LOG_AGGREGATOR_ENABLED": true,
  "LOG_AGGREGATOR_TOWER_UUID": ""
}
```

Splunk HTTP Event Collector listens on 8088 by default so it is necessary to provide the full HEC event URL (with port) in order for incoming requests to be processed successfully. These values are entered in the example below:

The screenshot shows the 'CONFIGURE Tower' interface with the 'SYSTEM' tab selected. The 'SUB CATEGORY' is set to 'Logging'. The configuration includes several fields and controls:

- ENABLE EXTERNAL LOGGING:** A toggle switch set to 'ON'.
- LOGGING AGGREGATOR:** A text input field containing 'http://%SPLUNK_IP%/services/collector/event'.
- LOGGING AGGREGATOR PORT:** An empty text input field.
- LOGGING AGGREGATOR TYPE:** A dropdown menu set to 'splunk'.
- LOGGING AGGREGATOR USERNAME:** An empty text input field.
- LOGGING AGGREGATOR PASSWORD/TOKEN:** A text input field with 'SHOW' and a masked password '.....'.
- LOGGERS SENDING DATA TO LOG AGGREGATOR FORM:** A text input field containing 'aws, activity_stream, job_events, system_tracking'.
- LOG SYSTEM TRACKING FACTS INDIVIDUALLY:** A toggle switch set to 'OFF'.
- LOGGING AGGREGATOR PROTOCOL:** A dropdown menu set to 'HTTPS'.
- TCP CONNECTION TIMEOUT:** A text input field containing '5'.
- LOGGING AGGREGATOR LEVEL THRESHOLD:** A dropdown menu set to 'INFO'.
- ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION:** A toggle switch set to 'OFF'.

At the bottom right, there are 'TEST', 'CANCEL', and 'SAVE' buttons. A 'REVERT ALL TO DEFAULT' link is located at the bottom left.

For further instructions on configuring the HTTP Event Collector, refer to the [Splunk documentation](#).

Loggly

To set up the sending of logs through Loggly's HTTP endpoint, refer to <https://www.loggly.com/docs/http-endpoint/>. Loggly uses the URL convention described at <http://logs-01.loggly.com/inputs/TOKEN/tag/http/>, which is shown inputted in the **Logging Aggregator** field in the example below:

The screenshot shows a single configuration field for the 'Logging Aggregator'. The field is labeled '* LOGGING AGGREGATOR' and has a 'REVERT' button to its right. The text inside the field is 'http://logs-01.loggly.com/inputs/5b9ad697-81f9-4249-9e76-'.

Sumologic

In Sumologic, create a search criteria containing the json files that provide the parameters used to collect the data you need.

The screenshot shows the Sumologic search interface. At the top, there's a navigation bar with 'sumologic' logo and menu items: Library, Search, Metrics, Dashboards, Manage, Help. The user 'Alan (Re)' is logged in. Below the navigation bar, there's a search bar with the query: `| json field=_raw "message" as message2`, `| json field=_raw "actor" as actor`, and `| json field=_raw "object1" as object1`. The search results are displayed as a time-series chart from 11/30/2016 2:58:21 PM to 3:13:21 PM. A blue bar indicates activity around 3:05 PM. Below the chart, there's a 'Messages' section with a 'Display Fields' list including 'Time', 'actor', 'message2', 'object1', and 'Message'. A 'Hidden Fields' list includes 'Collector', 'Size', 'Source', 'Source Category', 'Source Host', and 'Source Name'. A modal window is open for the 'actor' field, showing a 'VALUES' table with 'admin' having 2 occurrences (100.00%). Below the table are options for 'DRILLDOWN' (Top Values, Top Values Over Time, Bottom Values) and an 'Expand JSON' button. The main message view shows a JSON log entry with fields like 'object1', 'host', 'logger_name', 'path', 'message', 'operation', 'changes', 'level', '@version', 'object2', 'actor', and 'type'. The message content is: "Activity Stream update entry for project".

Elastic stack (formerly ELK stack)

If starting from scratch, standing up your own version the elastic stack, the only change you required is to add the following lines to the logstash `logstash.conf` file:


```
filter {
  json {
    source => "message"
  }
}
```

Note: Backward-incompatible changes were introduced with Elastic 5.0.0, and different configurations may be required depending on what versions you are using.

11.2 Set Up Logging with Tower


To set up logging to any of the aggregator types:



1. Click the Settings () icon from the left navigation bar.
2. Select the **System** tab.
3. Select **Logging** from the Sub Category drop-down menu list.
4. Set the configurable options from the fields provided:
 - **Enable External Logging:** Click the toggle button to **ON** if you want to send logs to an external log aggregator.
 - **Logging Aggregator:** Enter the hostname or IP address you want to send logs.
 - **Logging Aggregator Port:** Specify the port for the aggregator if it requires one.

Note: When the connection type is HTTPS, you can enter the hostname as a URL with a port number and therefore, you are not required to enter the port again. But TCP and UDP connections are determined by the hostname and port number combination, rather than URL. So in the case of TCP/UDP connection, supply the port in the specified field. If instead a URL is entered in host field (**Logging Aggregator** field), its hostname portion will be extracted as the actual hostname.

- **Logging Aggregator Type:** Click to select the aggregator service from the drop-down menu:

LOGGING AGGREGATOR TYPE  REVERT

Select types ▲



logstash

splunk

loggly

sumologic

other

- **Logging Aggregator Username:** Enter the username of the logging aggregator if it requires it.
- **Logging Aggregator Password/Token:** Enter the password of the logging aggregator if it requires it.
- **Loggers to Send Data to the Log Aggregator Form:** All four types of data are pre-populated by default. Click the tooltip  icon next to the field for additional information on each data type. Delete the data types you do not want.
- **Log System Tracking Facts Individually:** Click the tooltip  icon for additional information whether or not you want to turn it on, or leave it off by default.

- **Logging Aggregator Protocol:** Click to select a connection type (protocol) to communicate with the log aggregator. Subsequent options vary depending on the selected protocol.
- **TCP Connection Timeout:** Specify the connection timeout in seconds. This option is only applicable to HTTPS and TCP log aggregator protocols.
- **Logging Aggregator Level Threshold:** Select the level of severity you want the log handler to report.
- **Enable/Disable HTTPS Certificate Verification:** Certificate verification is enabled by default for HTTPS log protocol. Click the toggle button to **OFF** if you do not want the log handler to verify the HTTPS certificate sent by the external log aggregator before establishing a connection.

5. Review your entries for your chosen logging aggregation. Below is an example of one set up for Splunk:

The screenshot shows the 'CONFIGURE Tower' interface for logging aggregation. It includes tabs for 'AUTHENTICATION', 'JOBS', 'SYSTEM', and 'USER INTERFACE'. The 'SYSTEM' tab is active. The 'SUB CATEGORY' is set to 'Logging'. The configuration includes several fields and toggle buttons: 'ENABLE EXTERNAL LOGGING' (OFF), 'LOGGING AGGREGATOR' (172.16.185.132), 'LOGGING AGGREGATOR PORT' (80), 'LOGGING AGGREGATOR TYPE' (splunk), 'LOGGING AGGREGATOR USERNAME', 'LOGGING AGGREGATOR PASSWORD/TOKEN' (SHOW), 'LOGGING AGGREGATOR PROTOCOL' (HTTPS), 'LOGGING AGGREGATOR LEVEL THRESHOLD' (INFO), 'LOG SYSTEM TRACKING FACTS INDIVIDUALLY' (OFF), 'LOGGERS SENDING DATA TO LOG AGGREGATOR FORM' (@www, activity_stream, job_events), and '*TCP CONNECTION TIMEOUT' (5). There are 'REVERT' buttons for many fields and a 'TEST' button at the bottom right.

6. To verify if your configuration is set up correctly, click **Test**. This verifies the logging configuration by sending a test log message and checking the response code is OK.
7. When done, click **Save** to apply the settings or **Cancel** to abandon the changes.

THE AWX-MANAGE UTILITY

The `awx-manage` (formerly `tower-manage`) utility is used to access detailed internal information of Tower. Commands for `awx-manage` should run as the `awx` or `root` user.

12.1 Inventory Import

`awx-manage` is a mechanism by which a Tower administrator can import inventory directly into Tower, for those who cannot use Custom Inventory Scripts.

To use `awx-manage` properly, you must first create an inventory in Tower to use as the destination for the import.

For help with `awx-manage`, run the following command: `awx-manage inventory_import [--help]`

The `inventory_import` command synchronizes a Tower inventory object with a text-based inventory file, dynamic inventory script, or a directory of one or more of the above as supported by core Ansible.

When running this command, specify either an `--inventory-id` or `--inventory-name`, and the path to the Ansible inventory source (`--source`).

```
awx-manage inventory_import --source=/ansible/inventory/ --inventory-id=1
```

By default, inventory data already stored in Tower blends with data from the external source. To use only the external data, specify `--overwrite`. To specify that any existing hosts get variable data exclusively from the `--source`, specify `--overwrite_vars`. The default behavior adds any new variables from the external source, overwriting keys that already exist, but preserves any variables that were not sourced from the external data source.

```
awx-manage inventory_import --source=/ansible/inventory/ --inventory-id=1 --overwrite
```

Note: With the release of Ansible Tower 2.4.0, edits and additions to Inventory host variables now persist beyond an inventory sync as long as `--overwrite_vars` is **not** set. To have inventory syncs behave as they did before, it is now required that both `--overwrite` and `--overwrite_vars` are set.

12.2 Cleanup of old data

`awx-manage` has a variety of commands used to clean old data from Tower. Tower administrators can use the Tower Management Jobs interface for access or use the command line.

- `awx-manage cleanup_jobs [--help]`

This permanently deletes the job details and job output for jobs older than a specified number of days.

- `awx-manage cleanup_activitystream [--help]`

This permanently deletes any *activity stream* data older than a specific number of days.

12.3 Cluster management

Refer to the *Clustering* section for details on the `awx-manage provision_instance` and `awx-manage deprovision_instance` commands.

Note: Do not run other `awx-manage` commands unless instructed by Ansible Support.

12.4 Token and session management

Ansible Tower supports the following commands for OAuth2 token management:

- `create_oauth2_token`
- `revoke_oauth2_tokens`
- `cleartokens`
- `expire_sessions`
- `clearsessions`

12.4.1 create_oauth2_token

Use this command to create OAuth2 tokens (specify actual username for `example_user` below):

```
$ awx-manage create_oauth2_token --user example_user
```

```
New OAuth2 token for example_user: j89ia80079te6IAZ97L7E8bMgXCON2
```

Make sure you provide a valid user when creating tokens. Otherwise, you will get an error message that you tried to issue the command without specifying a user, or supplying a username that does not exist.

12.4.2 revoke_oauth2_tokens

Use this command to revoke OAuth2 tokens (both application tokens and personal access tokens (PAT)). By default, it revokes all application tokens (but not their associated refresh tokens), and revokes all personal access tokens. However, you can also specify a user for whom to revoke all tokens.

To revoke all existing OAuth2 tokens:

```
$ awx-manage revoke_oauth2_tokens
```

To revoke all OAuth2 tokens & their refresh tokens:

```
$ awx-manage revoke_oauth2_tokens --revoke_refresh
```

To revoke all OAuth2 tokens for the user with `id=example_user` (specify actual username for `example_user` below):

```
$ awx-manage revoke_oauth2_tokens --user example_user
```

To revoke all OAuth2 tokens and refresh token for the user with `id=example_user`:

```
$ awx-manage revoke_oauth2_tokens --user example_user --revoke_refresh
```

12.4.3 cleartokens

Use this command to clear tokens which have already been revoked. Refer to [Django's Oauth Toolkit documentation on cleartokens](#) for more detail.

12.4.4 expire_sessions

Use this command to terminate all sessions or all sessions for a specific user. Consider using this command when a user changes role in an organization, is removed from assorted groups in LDAP/AD, or the administrator wants to ensure the user can no longer execute jobs due to membership in these groups.

```
$ awx-manage expire_sessions
```

This command terminates all sessions by default. The users associated with those sessions will be consequently logged out. To only expire the sessions of a specific user, you can pass their username using the `--user` flag (specify actual username for `example_user` below):

```
$ awx-manage expire_sessions --user example_user
```

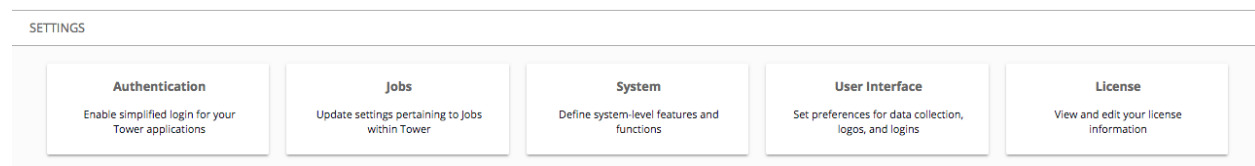

12.4.5 `clearsessions`

Use this command to delete all sessions that have expired. Refer to [Django's documentation on clearsessions](#) for more detail.

For more information on OAuth2 token management in the Tower User Interface, see the [Applications](#) section of the *Ansible Tower User Guide*.

TOWER CONFIGURATION

You can configure various Tower settings within the Settings screen in the following tabs:



Each tab contains fields with a **Reset** button, allowing you to revert any value entered back to the default value. **Reset All** allows you to revert all the values in the Edit Tower Configuration to their factory default values.

Save applies changes you make, but it does not exit the edit dialog. To return to the Configure Tower screen, click the



Settings () icon from the left navigation bar or use the breadcrumbs at the top of the current view.

13.1 Authentication

Through the Tower user interface, you can set up a simplified login through various authentication types: GitHub, Google, LDAP, RADIUS, and SAML. After you create and register your developer application with the appropriate service, you can set up authorizations for them. Since configuration files are now saved to the postgres DB in Ansible Tower 3.1 instead of flat files, setting up authorizations in the Ansible Tower User Interface is the recommended method.



1. From the left navigation bar, hover over the Settings () icon and select **Authentication** or click the **Authentication** tab from the Settings screen.
2. The Authentication window opens. Select the appropriate authentication type from the row of tabs across the top of the window.

SETTINGS / AUTHENTICATION

AUTHENTICATION

AZURE AD OAUTH2 CALLBACK URL ⓘ
 AZURE AD OAUTH2 KEY ⓘ REVERT
 AZURE AD OAUTH2 SECRET ⓘ REVERT

AZURE AD OAUTH2 ORGANIZATION MAP ⓘ REVERT

AZURE AD OAUTH2 TEAM MAP ⓘ REVERT

Different authentication types require you to enter different information. Be sure to include all the information as required.



Note: For more detail about each authentication type, refer to the [Setting Up Authentication](#) section of the Administration Guide.

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

13.2 Jobs

The Jobs tab allows you to configure the types of modules that are allowed to be used by Tower’s Ad Hoc Commands feature, set limits on the number of jobs that can be scheduled, define their output size, and other details pertaining to working with Jobs in Tower.



1. From the left navigation bar, hover over the Settings () icon and select **Jobs** or click the **Jobs** tab from the Settings screen.
2. Set the configurable options from the fields provided. Click the tooltip  icon next to the field that you need additional information or details about.


Note: The value for **Default Job Timeout** is in seconds.

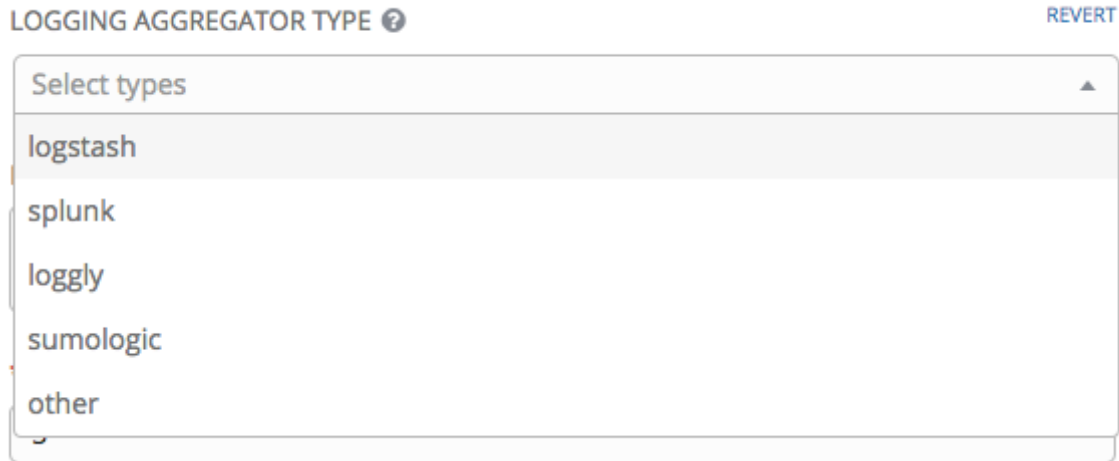
3. Click **Save** to apply the settings or **Cancel** to abandon the changes.

13.3 System


The System tab allows you to define the base URL for the Tower host, configure alerts, enable activity capturing, control visibility of users, enable certain Tower features and functionality through a license file, and configure logging aggregation options.

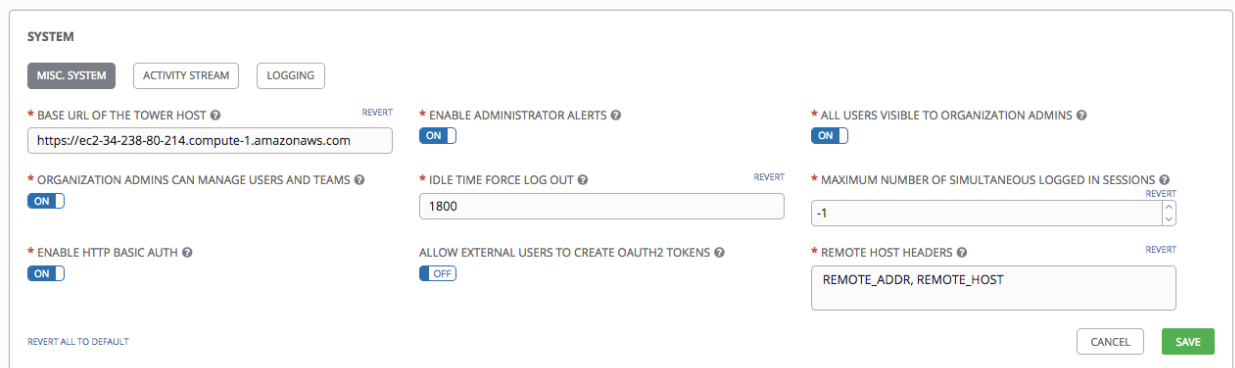


1. From the left navigation bar, hover over the Settings () icon and select **System** or click the **System** tab from the Settings screen.
2. The System window opens. Select an option from the row of tabs across the top of the window:
 - **Misc. System:** define the base URL for the Tower host, enable tower administration alerts, and allow all users to be visible to organization administrators.
 - **Activity Stream:** enable or disable activity stream.
 - **Logging:** configure logging options based on the type you choose:



For more information about each of the logging aggregation types, refer to the [Tower Logging and Aggregation](#) section of the *Ansible Tower Administration Guide*.

3. Set the configurable options from the fields provided. Click the tooltip  icon next to the field that you need additional information or details about.




Note: The **Allow External Users to Create Oauth2 Tokens** setting is disabled by default. This ensures external users cannot *create* their own tokens. If you enable then disable it, any tokens created by external users in the meantime will still exist, and are not automatically revoked.

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

13.4 User Interface

The User Interface tab allows you to set Tower analytics settings, as well as configure custom logos and login messages.



Access the User Interface settings by hovering over the Settings () icon from the left navigation bar and select **User Interface** or click the **User Interface** tab from the Settings screen.


USER INTERFACE

* ANALYTICS TRACKING STATE REVERT

Off

CUSTOM LOGO REVERT

REMOVE Choose file

Current Image: 

CUSTOM LOGIN INFO REVERT

REVERT ALL TO DEFAULT

CANCEL SAVE

13.4.1 Usability Analytics and Data Collection

In Ansible Tower version 2.4.0, a behind the scenes functionality was added to Tower to collect usability data. This software was introduced 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.

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.

1. Select the desired level of data collection from the 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.

Note: This setting was previously configured via PENDO, which is no longer supported.

13.4.2 Custom Logos and Images

You can also add a custom logo by uploading an image; and supply a custom login message from the User Interface screen in Configure Tower.

USER INTERFACE

* ANALYTICS TRACKING STATE REVERT

Off

CUSTOM LOGO REVERT

REMOVE angry-spud.png

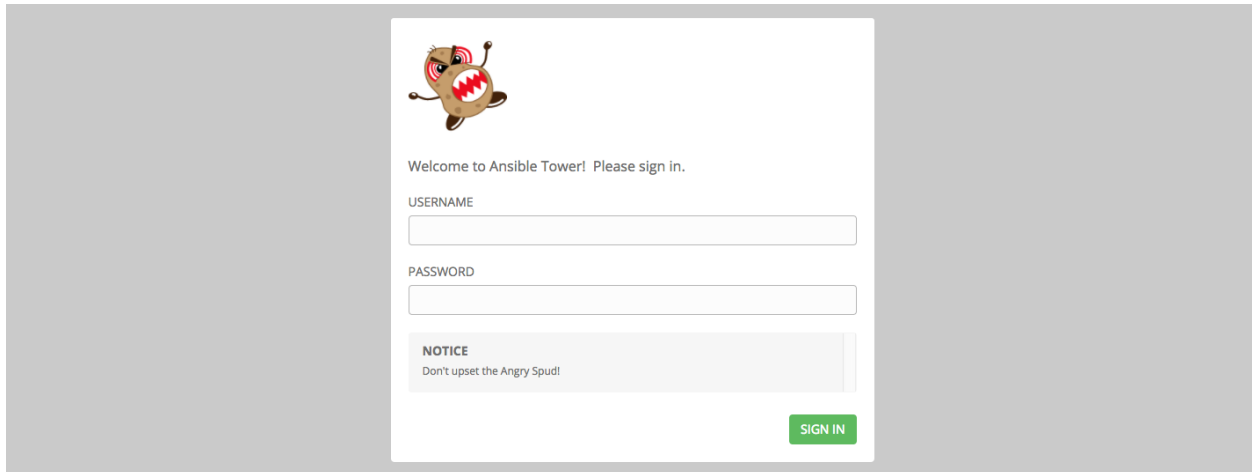
CUSTOM LOGIN INFO REVERT

Don't upset the Angry Spud!

REVERT ALL TO DEFAULT

CANCEL SAVE

Refer to the tooltips () for acceptable formats.



13.5 License

Tower requires a valid license to run. If you did not receive a license from Ansible directly or via email, or have issues with the license you received, refer to <http://www.ansible.com/license> for free and paid license options (including free trial licenses) or contact Ansible via the Red Hat Customer portal at <https://access.redhat.com/>.

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

LICENSE MANAGEMENT

Choose your license file, agree to the End User License Agreement, and click submit.

*** LICENSE FILE**

BROWSE No file selected.

*** END USER LICENSE AGREEMENT**

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

This end user license agreement ("EULA") governs the use of the Ansible Tower software and any related updates, upgrades, versions, appearance, structure and organization (the "Ansible Tower Software"), regardless of the delivery mechanism.

1. License Grant. Subject to the terms of this EULA, Red Hat, Inc. and its affiliates ("Red Hat") grant to you


I agree to the End User License Agreement

SUBMIT

To add your license:

1. Save your license (or save the license contents to a text file locally, if needed).




2. Click the Settings () icon from the left navigation bar and select the **License** tab from the Settings screen.
3. Click the **Browse** button and navigate to the location where the license file is saved to upload it. The uploaded license may be a plain text file or a JSON file, and must include properly formatted JSON code.
4. Once uploaded, check to agree to the End User License Agreement and click **Submit**.

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

For later reference, you can view this license from the **License** tab of the Settings screen, accessible through the



Settings () icon from the left navigation bar.

LICENSE

DETAILS

LICENSE	● Valid License
VERSION	3.4.0
LICENSE TYPE	Enterprise
SUBSCRIPTION	Enterprise Tower Up To 30 Nodes
LICENSE KEY	367ce98cba3e62f4c5665c567f60 18ec74fdb35530247d6e0bcbe75 3cbb8fa88
EXPIRES ON	01/01/2025
TIME REMAINING	2262 Days
HOSTS AVAILABLE	30
HOSTS USED	1
HOSTS REMAINING	29

If you are ready to upgrade, please contact us by clicking the button below

UPGRADE

LICENSE MANAGEMENT

Choose your license file, agree to the End User License Agreement, and click submit.

* LICENSE FILE

BROWSE No file selected.

* END USER LICENSE AGREEMENT

ANSIBLE TOWER BY RED HAT END USER LICENSE AGREEMENT

This end user license agreement ("EULA") governs the use of the Ansible Tower software and any related updates, upgrades, versions, appearance, structure and organization (the "Ansible Tower Software"), regardless of the delivery mechanism.

1. License Grant. Subject to the terms of this EULA, Red Hat, Inc. and its affiliates ("Red Hat") grant to you ("You") a non-transferable, non-exclusive, worldwide, non-sublicensable, limited, revocable license to use the Ansible Tower Software for the term of the associated Red Hat

I agree to the End User License Agreement


SUBMIT

BUBBLEWRAP FUNCTIONALITY AND VARIABLES

The bubblewrap functionality in Ansible Tower limits which directories on the Tower file system are available for playbooks to see and use during playbook runs. You may find that you need to customize your bubblewrap settings in some cases. To fine tune your usage of bubblewrap, there are certain variables that can be set.

To disable or enable bubblewrap support for running jobs (playbook runs only), ensure you are logged in as the Admin user:



1. Click the Settings () icon from the left navigation bar.
2. Click the “Jobs” tab.
3. Scroll down until you see “Enable Job Isolation” and change the toggle button selection to **OFF** to disable bubblewrap support or select **ON** to enable it.

By default, the Tower will use the system’s `tmp` directory (`/tmp` by default) as its staging area. This can be changed in the **Job Isolation Execution Path** field of the Configure tower screen, or by updating the following entry in the settings file:

```
AWX_PROOT_BASE_PATH = "/opt/tmp"
```

If there is other information on the system that is sensitive and should be hidden, you can specify those in the Configure Tower screen in the **Paths to Hide to Isolated Jobs** or by updating the following entry in the settings file:

```
AWX_PROOT_HIDE_PATHS = ['/list/of/', '/paths']
```

If there are any directories that should specifically be exposed, you can specify those in the Configure Tower screen in the **Paths to Expose to Isolated Jobs** or by updating the following entry in the settings file:

```
AWX_PROOT_SHOW_PATHS = ['/list/of/', '/paths']
```

Note: The primary file you may want to add to `AWX_PROOT_SHOW_PATHS` is `/var/lib/awx/.ssh`, if your playbooks need to use keys or settings defined there.

If you made changes in the settings file, be sure to restart services with the `ansible-tower-service restart` command after your changes have been saved.

TOKEN-BASED AUTHENTICATION

Starting with Ansible Tower 3.3, OAuth 2 is used for token-based authentication. You can manage OAuth tokens as well as applications, a server-side representation of API clients used to generate tokens. By including an OAuth token as part of the HTTP authentication header, you can authenticate yourself and adjust the degree of restrictive permissions in addition to the base RBAC permissions. Refer to [RFC 6749](#) for more details of OAuth 2 specification.

15.1 Managing OAuth 2 Applications and Tokens

Applications and tokens can be managed as a top-level resource at `/api/<version>/applications` and `/api/<version>/tokens`. These resources can also be accessed relative to the user at `/api/<version>/users/N/<resource>`. Applications can be created by making a **POST** to either `api/<version>/applications` or `/api/<version>/users/N/applications`.

Each OAuth 2 application represents a specific API client on the server side. For an API client to use the API via an application token, it must first have an application and issue an access token. Individual applications are accessible via their primary keys: `/api/<version>/applications/<pk>/`. Here is a typical application:

```
{
  "id": 1,
  "type": "o_auth2_application",
  "url": "/api/v2/applications/2/",
  "related": {
    "tokens": "/api/v2/applications/2/tokens/"
  },
  "summary_fields": {
    "organization": {
      "id": 1,
      "name": "Default",
      "description": ""
    },
    "user_capabilities": {
      "edit": true,
      "delete": true
    },
    "tokens": {
      "count": 0,
      "results": []
    }
  },
  "created": "2018-07-02T21:16:45.824400Z",
  "modified": "2018-07-02T21:16:45.824514Z",
}
```

(continues on next page)

(continued from previous page)

```

    "name": "My Application",
    "description": "",
    "client_id": "Ecmc6RjjhKUOWJzDYEP8TZ35P3dvsKt0AKdIjgHV",
    "client_secret":
→ "7Ft7ym8MpE54yWGUNvxxg6KqGwPFsyhYn9QQfYHlgBxai74Qp1GE4zsvJduOfSFkTfWFnPzYpxqcRsy1KacD0HH0vOAQUDJDC
→ ",
    "client_type": "confidential",
    "redirect_uris": "",
    "authorization_grant_type": "password",
    "skip_authorization": false,
    "organization": 1
}

```

As shown in the example above, name is the human-readable identifier of the application. The rest of the other fields, like client_id and redirect_uris, are mainly used for OAuth2 authorization, which is covered later in *Using OAuth 2 Token System for Personal Access Tokens (PAT)*.

The values for the client_id and client_secret fields are generated during creation and are non-editable identifiers of applications, while organization and authorization_grant_type are required upon creation and become non-editable.

15.1.1 Access Rules for Applications

Access rules for applications are as follows:

- System administrators can view and manipulate all applications in the system
- Organization administrators can view and manipulate all applications belonging to Organization members
- Other users can only view, update, and delete their own applications, but cannot create any new applications

Tokens, on the other hand, are resources used to actually authenticate incoming requests and mask the permissions of the underlying user. There are two ways of creating a token:

- POSTing to the /api/v2/tokens/ endpoint by providing application and scope fields to point to the related application and specify token scope
- POSTing to /api/v2/applications/<pk>/tokens/ by providing only scope, while the parent application will be automatically linked

Individual tokens will be accessible via their primary keys: /api/<version>/tokens/<pk>/. Here is an example of a typical token:

```

{
  "id": 4,
  "type": "o_auth2_access_token",
  "url": "/api/v2/tokens/4/",
  "related": {
    "user": "/api/v2/users/1/",
    "application": "/api/v2/applications/1/",
    "activity_stream": "/api/v2/tokens/4/activity_stream/"
  },
  "summary_fields": {
    "application": {
      "id": 1,
      "name": "Default application for root",
      "client_id": "mcU5J5uGQcEQMgAZyr5JUUnM3BqBJpgbgL9fLOVch"
    }
  }
}

```

(continues on next page)

(continued from previous page)

```
    },
    "user": {
      "id": 1,
      "username": "root",
      "first_name": "",
      "last_name": ""
    }
  },
  "created": "2018-02-23T14:39:32.618932Z",
  "modified": "2018-02-23T14:39:32.643626Z",
  "description": "App Token Test",
  "user": 1,
  "token": "*****",
  "refresh_token": "*****",
  "application": 1,
  "expires": "2018-02-24T00:39:32.618279Z",
  "scope": "read"
},
```

For an OAuth 2 token, the only fully editable fields are scope and description. The application field is non-editable on update, and all other fields are entirely non-editable, and are auto-populated during creation, as follows:

- user field corresponds to the user the token is created for, and in this case, is also the user creating the token
- expires is generated according to the Tower configuration setting OAUTH2_PROVIDER
- token and refresh_token are auto-generated to be non-clashing random strings

Both application tokens and personal access tokens are shown at the `/api/v2/tokens/` endpoint. The application field in the personal access tokens is always **null**. This is a good way to differentiate the two types of tokens.

15.1.2 Access rules for tokens

Access rules for tokens are as follows:

- Users can create a token if they are able to view the related application; and are also able to create a personal token for themselves
- System administrators are able to view and manipulate every token in the system
- Organization administrators are able to view and manipulate all tokens belonging to Organization members
- System Auditors can view all tokens and applications
- Other normal users are only able to view and manipulate their own tokens

Note: Users can only view the token or refresh the token value at the time of creation only.

15.2 Using OAuth 2 Token System for Personal Access Tokens (PAT)

The most common use of OAuth 2 is authenticating users. The `token` field of a token is used as part of HTTP authentication header, in the format of `Authorization: Bearer <token field value>`. The *Bearer* token can be obtained by issuing a `curl` command at the `/api/v2/users/<USER_ID>/personal_tokens/` endpoint, as shown in this example below:

```
curl -XPOST -k -H "Content-type: application/json" -d '{"description": "Tower CLI",
↪ "application": null, "scope": "write"}' https://<USERNAME>:<PASSWORD>@<TOWER_SERVER>/
↪ api/v2/users/<USER_ID>/personal_tokens/ | python -m json.tool
```

You could also pipe the JSON output through `jq`, if installed.

Note: The special OAuth 2 endpoints only support using the `x-www-form-urlencoded` **Content-type**, so as a result, none of the `api/o/*` endpoints accept `application/json`.

Following is an example of using the personal token to access an API endpoint using `curl`:

```
curl -k -H "Authorization: Bearer <token>" -H "Content-Type: application/json" -X_
↪ POST -d '{}' https://tower/api/v2/job_templates/5/launch/
```

According to OAuth 2 specification, users can acquire, revoke, and refresh an access token. In Ansible Tower, the equivalent, and most efficient way to refresh a token, is create a token, delete a token, and then quickly followed by creating a new one. To revoke a token, simply delete it in the Applications configuration of the user interface, or at the token's detail page in the API.

In Ansible Tower, the OAuth 2 system is built on top of the [Django OAuth Toolkit](#), which provides dedicated endpoints for authorizing, revoking, and refreshing tokens. Tower implements them and puts related endpoints under the `/api/v2/users/<USER_ID>/personal_tokens/` endpoint, which also provides detailed examples on some typical usage of those endpoints.

Note: You can also request tokens using the `/api/o/token` endpoint by specifying `null` for the application type.

15.2.1 Token scope mask over RBAC system

The scope of an OAuth 2 token is a space-separated string composed of valid scope keywords, 'read' and 'write'. These keywords are configurable and used to specify permission level of the authenticated API client. Read and write scopes provide a mask layer over the Role-Based Access Control (RBAC) permission system of Ansible Tower. Specifically, a 'write' scope gives the authenticated user the full permissions the RBAC system provides, while a 'read' scope gives the authenticated user only read permissions the RBAC system provides. Note that 'write' implies 'read' as well.

For example, if you have administrative permissions to a job template, you can view, modify, launch, and delete the job template if authenticated via session or basic authentication. In contrast, if you are authenticated using OAuth 2 token, and the related token scope is 'read', you can only view, but not manipulate or launch the job template, despite being an administrator. If the token scope is 'write' or 'read write', you can take full advantage of the job template as its administrator.

To acquire and use a token, first create an application token:

1. Make an application with `authorization_grant_type` set to `password`. HTTP POST the following to the `/api/v2/applications/` endpoint (supplying your own organization ID):

```
{
  "name": "Admin Internal Application",
  "description": "For use by secure services & clients. ",
  "client_type": "confidential",
  "redirect_uris": "",
  "authorization_grant_type": "password",
  "skip_authorization": false,
  "organization": <organization-id>
}
```

2. Make a token and POST to the `/api/v2/tokens/` endpoint:

```
{
  "description": "My Access Token",
  "application": <application-id>,
  "scope": "write"
}
```

This returns a `<token-value>` that you can use to authenticate with for future requests (this will not be shown again).

3. Use the token to access a resource. The following uses curl as an example:

```
curl -H "Authorization: Bearer <token-value>" -H "Content-Type: application/json" -X GET https://<tower>/api/v2/users/
```

The `-k` flag may be needed if you have not set up a CA yet and are using SSL.

To revoke a token, you can make a DELETE on the detail page for that token, using that token's ID. For example:

```
curl -ku <user>:<password> -X DELETE https://<tower>/api/v2/tokens/<pk>/
```

Similarly, using a token:

```
curl -H "Authorization: Bearer <token-value>" -X DELETE https://<tower>/api/v2/tokens/<pk>/ -k
```

15.3 Application Functions

This page lists OAuth 2 utility endpoints used for authorization, token refresh, and revoke. Endpoints other than `/api/o/authorize/` are not meant to be used in browsers and do not support HTTP GET. The endpoints prescribed here strictly follow RFC specifications for OAuth 2, so use that for detailed reference. The `implicit` grant type can only be used to acquire an access token if you are already logged in via session authentication, as that confirms that you are authorized to create an access token. The following is an example of the typical usage of these endpoints in Tower, in particular, when creating an application using various grant types:

- Authorization Code
- Implicit
- Password

15.3.1 Application using authorization code grant type

The application `authorization code` grant type should be used when access tokens need to be issued directly to an external application or service.

Note:

You can only use the `authorization code` type to acquire an access token when using an application. When integrating an external webapp with Ansible Tower, that webapp may need to create OAuth2 Tokens on behalf of users in that other webapp. Creating an application in Tower with the `authorization code` grant type is the preferred way to do this because:

- this allows an external application to obtain a token from Tower for a user, using their credentials.
- compartmentalized tokens issued for a particular application allows those tokens to be easily managed (revoke all tokens associated with that application without having to revoke *all* tokens in the system, for example)

To create an application named `AuthCodeApp` with the `authorization-code` grant type, perform a POST to the `/api/v2/applications/` endpoint:

```
{
  "name": "AuthCodeApp",
  "user": 1,
  "client_type": "confidential",
  "redirect_uris": "http://<tower>/api/v2",
  "authorization_grant_type": "authorization-code",
  "skip_authorization": false
}
```

The workflow that occurs when you issue a **GET** to the `authorize` endpoint from the client application with the `response_type`, `client_id`, `redirect_uris`, and `scope`:

1. Tower responds with the authorization code and status to the `redirect_uri` specified in the application.
2. The client application then makes a **POST** to the `api/o/token/` endpoint on Tower with the `code`, `client_id`, `client_secret`, `grant_type`, and `redirect_uri`.
3. Tower responds with the `access_token`, `token_type`, `refresh_token`, and `expires_in`.

Refer to [Django's Test Your Authorization Server](#) toolkit to test this flow.

The best way to set up app integrations with Ansible Tower using the Authorization Code grant type is to whitelist the origins for those cross-site requests. More generally, you need to whitelist the service or application you are integrating with Tower, for which you want to provide access tokens. To do this, have your Administrator add this whitelist to their local Tower settings:

```
CORS_ORIGIN_REGEX_WHITELIST = [
    r"http://django-oauth-toolkit.herokuapp.com*",
    r"http://www.example.com*"
]
```

Where `http://django-oauth-toolkit.herokuapp.com` and `http://www.example.com` are applications needing tokens with which to access Tower.

15.3.2 Application using implicit grant type

An implicit grant type is used when a single page web application can't keep a `client_secret` secure. This method skips the authorization code part of the flow and just returns an access token. The following supposes the application administrator's app of grant type is `implicit`:

```
{
  "id": 1,
  "type": "application",
  "related": {
    ...
    "name": "admin's app",
    "user": 1,
    "client_id": "L0uQQWW8pKX51hoqIRQGsuqmIdPi2AcXZ9EJRGmj",
    "client_secret":
↪ "9Wp4dUrUsigI8J15fQYJ3jn0MJHLkAjoy7ikBsABeWTNJbZwy7eB2Xro9ykYuuygerTPQ2gIF2DCTtN3kurkt0Me3AhanEw6p
↪",
    "client_type": "confidential",
    "redirect_uris": "http://<tower>/api/",
    "authorization_grant_type": "implicit",
    "skip_authorization": false
  }
}
```

In the API browser, first make sure you are logged in via session auth, then visit the authorization endpoint with the given parameters:

```
http://localhost:8013/api/o/authorize/?response_type=token&client_
↪ id=L0uQQWW8pKX51hoqIRQGsuqmIdPi2AcXZ9EJRGmj&scope=read
```

This is case, the `client_id` must be the same as the `client_id` field of the underlying application. On success, an authorization page displays with a prompt, asking you to grant or deny the access token. After you click **grant**, the API browser will POST to the same endpoint with the same parameters in the POST body, on success, a “302 redirect” will be returned:

```
HTTP/1.1 302 Found
Connection:keep-alive
Content-Language:en
Content-Length:0
Content-Type:text/html; charset=utf-8
Date:Tue, 05 Dec 2017 20:36:19 GMT
Location:http://localhost:8013/api/#access_token=01VJJkolFTwYawHyGkk7NTmSKdzBen&token_
↪ type=Bearer&state=&expires_in=31536000000&scope=read
Server:nginx/1.12.2
Strict-Transport-Security:max-age=15768000
Vary:Accept-Language, Cookie
```

Note: Tokens created with implicit applications do not have a refresh token.

15.3.3 Application using password grant type

The password grant type or Resource owner password-based grant type is ideal for users who have native access to the web app and should be used when the client is the Resource owner. The following supposes an application, 'Default Application' with grant type password:

```
{
  "id": 6,
  "type": "application",
  ...
  "name": "Default Application",
  "user": 1,
  "client_id": "gwSPoasWSdNkMDtBN3Hu2WYQpPwCO9SwUEsKK221",
  "client_secret":
  ↪ "fI6ZpfocHYBGfmltP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569eIaUBsaVDgt2eiwOGe0bg5m5vCSstClZmtdy359RVx2rQK5Y1IWYp1rolpt2LEpVeKXWaiybo",
  ↪,
  "client_type": "confidential",
  "redirect_uris": "",
  "authorization_grant_type": "password",
  "skip_authorization": false
}
```

Logging in is not required for password grant type, so you can simply use curl to acquire a personal access token through the `/api/o/token/` endpoint:

```
curl -X POST \
-d "grant_type=password&username=<username>&password=<password>&scope=read" \
-u
  ↪ "gwSPoasWSdNkMDtBN3Hu2WYQpPwCO9SwUEsKK221:fI6ZpfocHYBGfmltP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569eIaUBsaVDgt2eiwOGe0bg5m5vCSstClZmtdy359RVx2rQK5Y1IWYp1rolpt2LEpVeKXWaiybo" \
http://<tower>/api/o/token/ -i
```

In the above POST request, parameters, username and password are the username and password of the related Tower user of the underlying application, and the authentication information is of format `<client_id>:<client_secret>`, where `client_id` and `client_secret` are the corresponding fields of the underlying application.

Note: The special OAuth 2 endpoints only support using the `x-www-form-urlencoded` **Content-type**, so as a result, none of the `api/o/*` endpoints accept `application/json`.

Upon success, a response displays in JSON format containing the access token, refresh token and other information:

```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 05 Dec 2017 16:48:09 GMT
Content-Type: application/json
Content-Length: 163
Connection: keep-alive
Content-Language: en
Vary: Accept-Language, Cookie
Pragma: no-cache
Cache-Control: no-store
Strict-Transport-Security: max-age=15768000

{"access_token": "9epHOqHhnXUcgYK8QanOmUQPSgX92g", "token_type": "Bearer", "expires_in": 315360000000, "refresh_token": "jMRX6QvzOTf046KHee3TU5mT3nyXsz", "scope": "read"}
```

15.4 Application Token Functions

This section describes the refresh and revoke functions associated with tokens. Everything that follows (Refreshing and revoking tokens at the `/api/o/` endpoints) can currently only be done with application tokens. However, tokens created with implicit applications do not have a refresh token.

15.4.1 Refresh an existing access token

The following example shows an existing access token with a refresh token provided:

```
{
  "id": 35,
  "type": "access_token",
  ...
  "user": 1,
  "token": "omMFLk7UKpB36WN2Qma9H3gbwEBSOc",
  "refresh_token": "AL0NK9TTpv0qp54dGbC4VUZtsZ9r8z",
  "application": 6,
  "expires": "2017-12-06T03:46:17.087022Z",
  "scope": "read write"
}
```

The `/api/o/token/` endpoint is used for refreshing the access token:

```
curl -X POST \
  -d "grant_type=refresh_token&refresh_token=AL0NK9TTpv0qp54dGbC4VUZtsZ9r8z" \
  -u
↪ "gwSPoasWSdNkMDtBN3Hu2WYQpPWC09SwUEsKK221:fI6ZpfocHYBGfm1tP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569e.
↪ " \
  http://<tower>/api/o/token/ -i
```

In the above POST request, `refresh_token` is provided by `refresh_token` field of the access token above that. The authentication information is of format `<client_id>:<client_secret>`, where `client_id` and `client_secret` are the corresponding fields of the underlying related application of the access token.

Note: The special OAuth 2 endpoints only support using the `x-www-form-urlencoded` **Content-type**, so as a result, none of the `api/o/*` endpoints accept `application/json`.

Upon success, a response displays in JSON format containing the new (refreshed) access token with the same scope information as the previous one:

```
HTTP/1.1 200 OK
Server: nginx/1.12.2
Date: Tue, 05 Dec 2017 17:54:06 GMT
Content-Type: application/json
Content-Length: 169
Connection: keep-alive
Content-Language: en
Vary: Accept-Language, Cookie
Pragma: no-cache
Cache-Control: no-store
Strict-Transport-Security: max-age=15768000

{"access_token": "NDInWxGJI4iZgqpsreuujbvzCfJqgR", "token_type": "Bearer", "expires_in": 315360000000, "refresh_token": "DqOrimz8Bx3sr1HkZNKmDpqA86bnQkT", "scope": "read write"}
(continues on next page)
```

(continued from previous page)

Essentially, the refresh operation replaces the existing token by deleting the original and then immediately creating a new token with the same scope and related application as the original one. Verify that new token is present and the old one is deleted in the `/api/v2/tokens/` endpoint.

15.4.2 Revoke an access token

Similarly, you can revoke an access token by using the `/api/o/revoke-token/` endpoint.

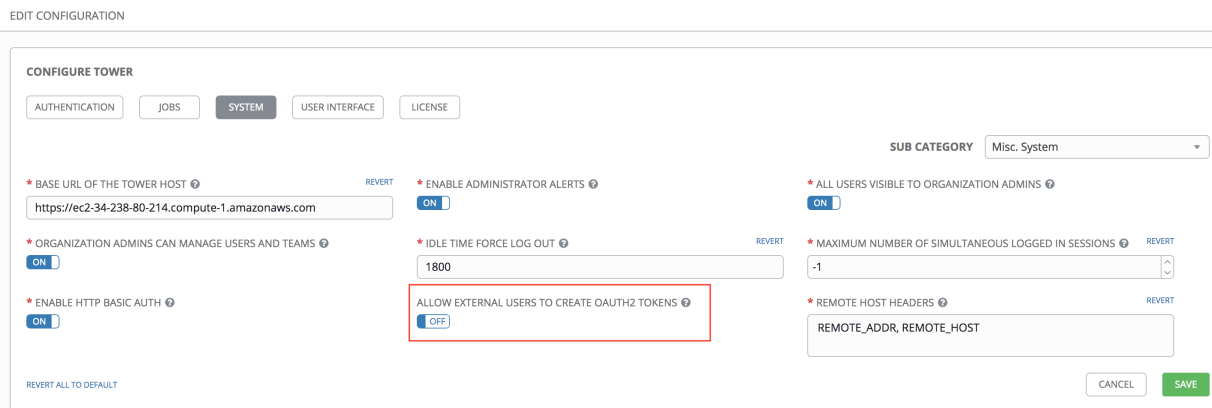
Revoking an access token by this method is the same as deleting the token resource object, but it allows you to delete a token by providing its token value, and the associated `client_id` (and `client_secret` if the application is confidential). For example:

```
curl -X POST -d "token=rQONsve372fQwuc2pn76k3IHDCYpi7" \
-u
↪ "gwSPoasWSdNkMDtBN3Hu2WYQpPWC09SwUEsKK221:fI6ZpfocHYBGfm1tP92r0yIgCyfRdDQt0Tos9L8a4fNsJjQQMwp9569e:
↪ " \
http://<tower>/api/o/revoke_token/ -i
```

Note: The special OAuth 2 endpoints only support using the `x-www-form-urlencoded` **Content-type**, so as a result, none of the `api/o/*` endpoints accept `application/json`.

Note: The **Allow External Users to Create Oauth2 Tokens** (`ALLOW_OAUTH2_FOR_EXTERNAL_USERS` in the API) setting is disabled by default. External users refer to users authenticated externally with a service like LDAP, or any of the other SSO services. This setting ensures external users cannot *create* their own tokens. If you enable then disable it, any tokens created by external users in the meantime will still exist, and are not automatically revoked.

This setting can be configured at the system-level in the Ansible Tower User Interface:



Upon success, a response of `200 OK` displays. Verify the deletion by checking whether the token is present in the `/api/v2/tokens/` endpoint.

SETTING UP SOCIAL AUTHENTICATION

Authentication methods help simplify logins for end users—offering single sign-ons using existing login information to sign into a third party website rather than creating a new login account specifically for that website.

Prior to Ansible Tower version 3.1, account authentication can only be configured in the `/etc/tower/settings.py` or the configuration files within `/etc/tower/conf.d/`. Starting with Ansible Tower version 3.1, instead of flat files, the configuration files are now saved to the Postgres database. Therefore, it is important that account authentication be configured in the Ansible Tower User Interface. For instructions, refer to the [Tower Configuration](#) section.

This section describes setting up authentication for the following systems:

- *Basic Authentication Settings*
- *Google OAuth2 Settings*
- *GitHub OAuth2 Settings*
 - *GitHub Org Settings*
 - *GitHub Team Settings*
- *Organization and Team Mapping*
 - *Organization mapping*
 - *Team mapping*

Account authentication in Ansible Tower can be configured to centrally use OAuth2, while enterprise-level account authentication can be configured for SAML, RADIUS, or even LDAP as a source for authentication information.

For websites, such as Microsoft Azure, Google or GitHub, that provide account information, account information is often implemented using the OAuth standard. OAuth is a secure authorization protocol which is commonly used in conjunction with account authentication to grant 3rd party applications a “session token” allowing them to make API calls to providers on the user’s behalf.

SAML (Security Assertion Markup Language) is an XML-based, open-standard data format for exchanging account authentication and authorization data between an identity provider and a service provider.

The RADIUS distributed client/server system allows you to secure networks against unauthorized access and can be implemented in network environments requiring high levels of security while maintaining network access for remote users.

16.1 Basic Authentication Settings


To enable or disable HTTP basic authentication as used in the API browser, edit the `sessions.py` file, setting the following line as either `True` or `False`:

```
AUTH_BASIC_ENABLED = False
```

After saving your changes, run the `ansible-tower-service restart` command to ensure your changes take effect.

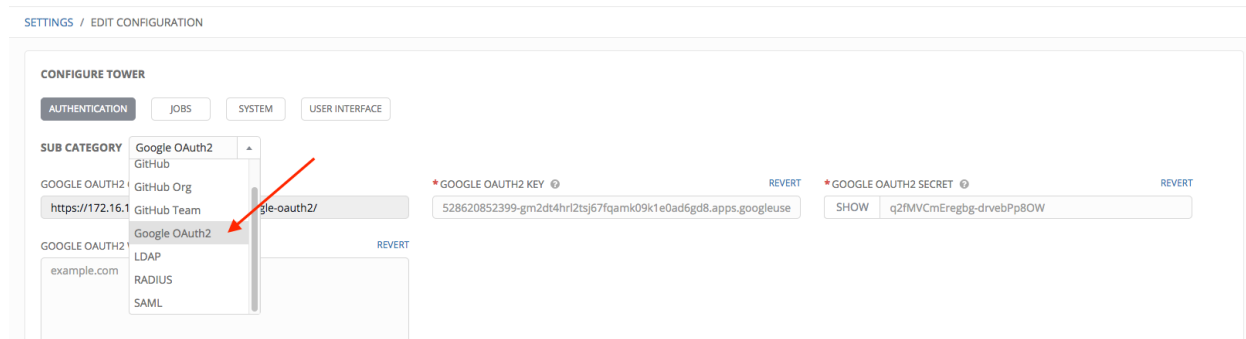
16.2 Google OAuth2 Settings

To set up social authentication for Google, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first create a project and set it up with Google. Refer to <https://support.google.com/googleapi/answer/6158849> for instructions. If you already completed the setup process, you can access those credentials by going to the Credentials section of the [Google API Manager Console](#). The OAuth2 key (Client ID) and secret (Client secret) will be used to supply the required fields in the Ansible Tower User Interface.

1. In the Ansible Tower User Interface, click **Configure Tower** from the Settings () Menu screen.

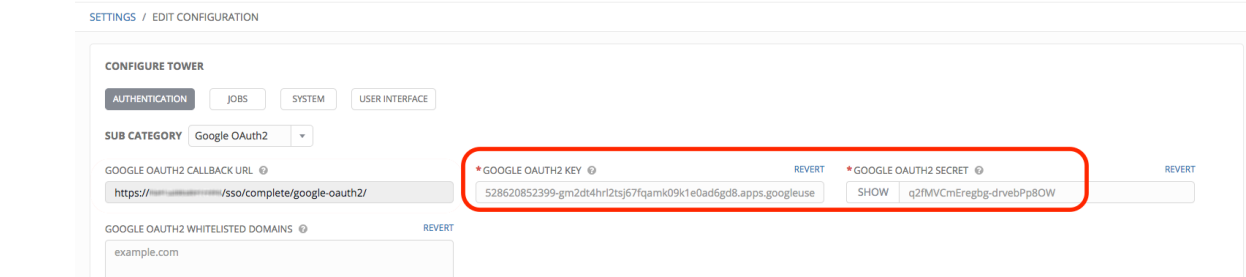
The Authentication tab displays initially by default.

2. In the Sub Category field, select **Google OAuth2** from the drop-down list.



The **Google OAuth2 Callback URL** field is already pre-populated and non-editable.


3. Using the credentials Google supplied during the web application setup process:
 - Copy and paste Google's Client ID into the **Google OAuth2 Key** field of the Configure Tower - Authentication screen. Look for the value with the same format as the one shown in the text field.
 - Copy and paste Google's Client secret into the **Google OAuth2 Secret** field of the Configure Tower - Authentication screen. Look for the value with the same format as the one shown in the text field.



4. To complete the remaining optional fields, refer to the tooltips in each of the fields for instructions and required format.
5. For details on completing the mapping fields, see *Organization and Team Mapping*.
7. Click **Save** when done.
8. To verify that the authentication was configured correctly, logout of Ansible Tower and the login screen will now display the Google logo to indicate it as a alternate method of logging into Ansible Tower.

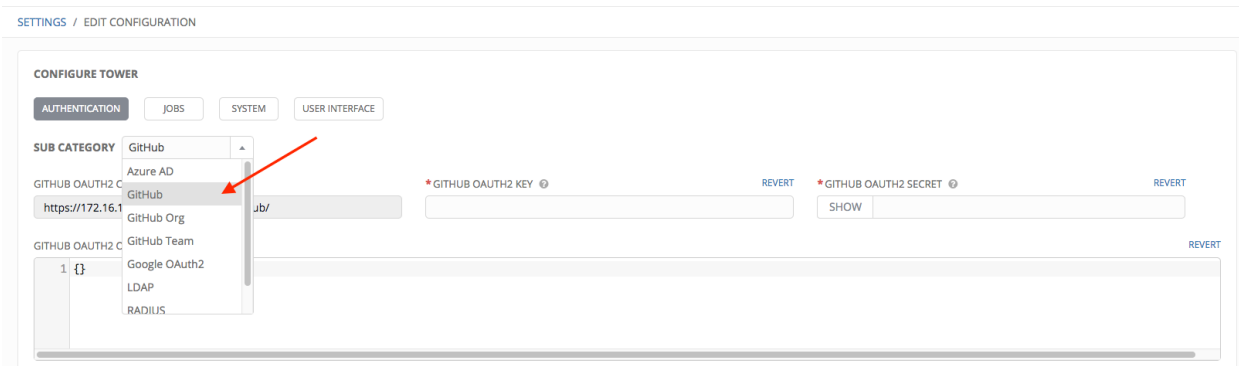
16.3 GitHub OAuth2 Settings

To set up social authentication for GitHub, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first register the new application with GitHub at <https://github.com/settings/developers>. In order to register the application, you must supply it with your homepage URL, which is the Callback URL shown in the Configure Tower user interface. The OAuth2 key (Client ID) and secret (Client Secret) will be used to supply the required fields in the Ansible Tower User Interface.

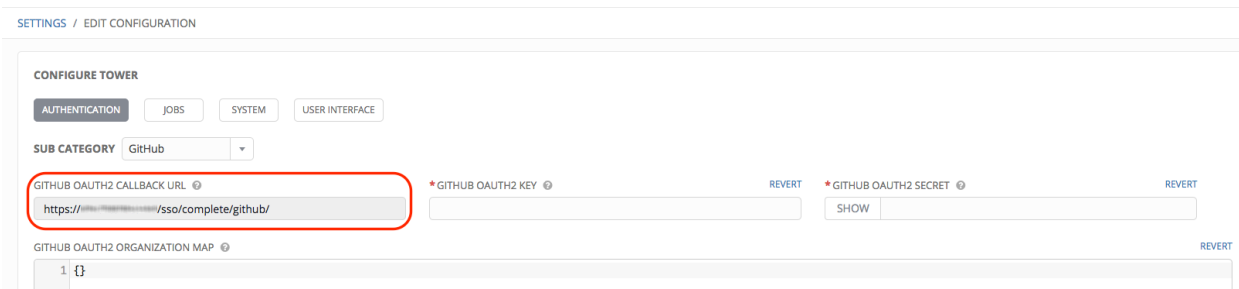
1. In the Ansible Tower User Interface, click **Configure Tower** from the Settings () Menu screen.

The Authentication tab displays initially by default.

2. In the Sub Category field, select **GitHub** from the drop-down list.



3. Use the **GitHub OAuth2 Callback URL** to supply GitHub when it prompts for your application's Homepage URL.



Once the application is registered, GitHub displays the Client ID and Client Secret.

4. Copy and paste GitHub's Client ID into the **GitHub OAuth2 Key** field of the Configure Tower - Authentication screen.
5. Copy and paste GitHub's Client Secret into the **GitHub OAuth2 Secret** field of the Configure Tower - Authentication screen.
6. For details on completing the mapping fields, see *Organization and Team Mapping*.
7. Click **Save** when done.
8. To verify that the authentication was configured correctly, logout of Ansible Tower and the login screen will now display the GitHub logo to allow logging in with those credentials.


16.3.1 Github Org Settings

When defining account authentication with either an organization or a team within an organization, you should use the specific organization and team settings. Account authentication can be limited by an organization as well as by a team within an organization.

You can also choose to allow all by specifying non-organization or non-team based settings (as shown above).

You can limit users who can login to Tower by limiting only those in an organization or on a team within an organization.

To set up social authentication for a GitHub Organization, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first register your organization-owned application at <https://github.com/organizations/<yourorg>/settings/applications>. In order to register the application, you must supply it with your Authorization callback URL, which is the Callback URL shown in the Configure Tower user interface. Each key and secret must belong to a unique application and cannot be shared or reused between different authentication backends. The OAuth2 key (Client ID) and secret (Client Secret) will be used to supply the required fields in the Ansible Tower User Interface.

1. In the Ansible Tower User Interface, click **Configure Tower** from the Settings () Menu screen.

The Authentication tab displays initially by default.

2. In the Sub Category field, select **GitHub Org** from the drop-down list.

SETTINGS / EDIT CONFIGURATION

The screenshot shows the 'CONFIGURE TOWER' interface with the 'AUTHENTICATION' tab active. The 'SUB CATEGORY' dropdown is open, and 'GitHub Org' is selected. A red arrow points to the 'GitHub Org' option in the dropdown. The 'GITHUB ORGANIZATION OAUTH2 CALLBACK URL' field is highlighted with a red box. Other fields include 'GITHUB ORGANIZATION OAUTH2 KEY', 'GITHUB ORGANIZATION OAUTH2 SECRET', and 'GITHUB ORGANIZATION NAME'.

3. Use the **GitHub Organization OAuth2 Callback URL** to supply GitHub when it prompts for the Authorization callback URL.

SETTINGS / EDIT CONFIGURATION

The screenshot shows the 'CONFIGURE TOWER' interface with the 'AUTHENTICATION' tab active. The 'SUB CATEGORY' dropdown is set to 'GitHub Org'. The 'GITHUB ORGANIZATION OAUTH2 CALLBACK URL' field is highlighted with a red box. Other fields include 'GITHUB ORGANIZATION OAUTH2 KEY', 'GITHUB ORGANIZATION OAUTH2 SECRET', and 'GITHUB ORGANIZATION NAME'.


Once the application is registered, GitHub displays the Client ID and Client Secret.

4. Copy and paste GitHub's Client ID into the **GitHub Organization OAuth2 Key** field of the Configure Tower - Authentication screen.
5. Copy and paste GitHub's Client Secret into the **GitHub Organization OAuth2 Secret** field of the Configure Tower - Authentication screen.
6. For details on completing the mapping fields, see *Organization and Team Mapping*.
7. Click **Save** when done.
8. To verify that the authentication was configured correctly, logout of Ansible Tower and the login screen will now display the GitHub Organization logo to allow logging in with those credentials.

16.3.2 Github Team Settings

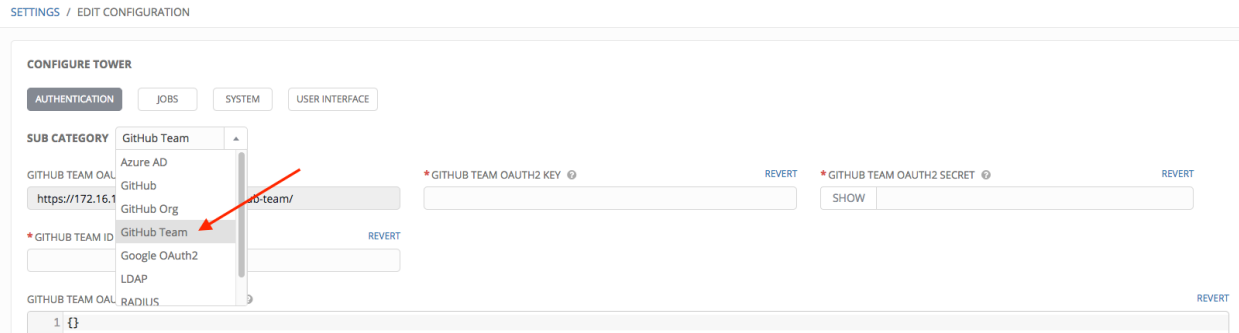
To set up social authentication for a GitHub Team, you will need to obtain an OAuth2 key and secret for a web application. To do this, you must first register your team-owned application at <https://github.com/organizations/<yourorg>/settings/applications>. In order to register the application, you must supply it with your Authorization callback URL, which is the Callback URL shown in the Configure Tower user interface. Each key and secret must belong to a unique application and cannot be shared or reused between different authentication backends. The OAuth2 key (Client ID) and secret (Client Secret) will be used to supply the required fields in the Ansible Tower User Interface.

1. Find the numeric team ID using the Github API: <http://fabian-kostadinov.github.io/2015/01/16/how-to-find-a-github-team-id/>. The Team ID will be used to supply a required field in the Ansible Tower User Interface.

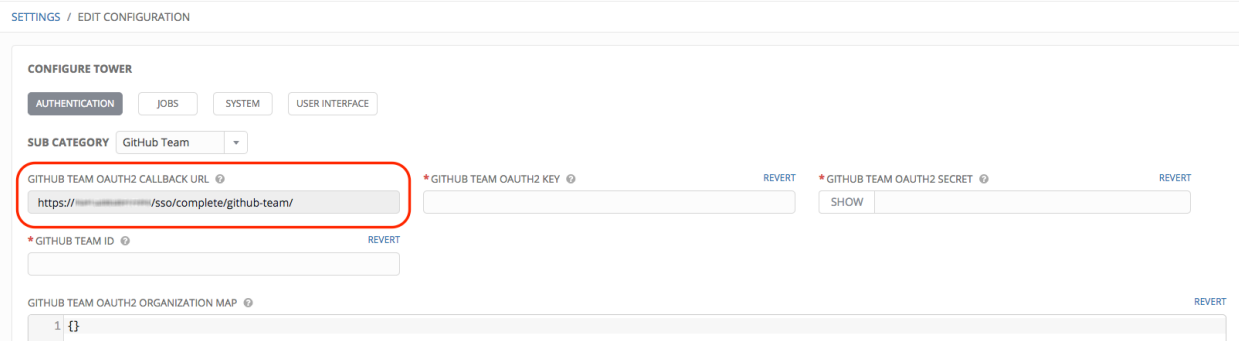
2. In the Ansible Tower User Interface, click **Configure Tower** from the Settings () Menu screen.

The Authentication tab displays initially by default.

3. In the Sub Category field, select **GitHub Team** from the drop-down list.



4. Use the **GitHub Team OAuth2 Callback URL** to supply GitHub when it prompts for the Authorization callback URL.



Once the application is registered, GitHub displays the Client ID and Client Secret.

5. Copy and paste GitHub's Client ID into the **GitHub Team OAuth2 Key** field of the Configure Tower - Authentication screen.
6. Copy and paste GitHub's Client Secret into the **GitHub Team OAuth2 Secret** field of the Configure Tower - Authentication screen.
7. For details on completing the mapping fields, see *Organization and Team Mapping*.
8. Click **Save** when done.
9. To verify that the authentication was configured correctly, logout of Ansible Tower and the login screen will now display the GitHub Team logo to allow logging in with those credentials.

16.4 Organization and Team Mapping

16.4.1 Organization mapping

You will need to control which users are placed into which Tower organizations based on their username and email address (mapping out your organization admins/users from social or enterprise-level authentication accounts).

Dictionary keys are organization names. Organizations will be created, if not already present and if the license allows for multiple organizations. Otherwise, the single default organization is used regardless of the key.

Values are dictionaries defining the options for each organization's membership. For each organization, it is possible to specify which users are automatically users of the organization and also which users can administer the organization.

admins: None, True/False, string or list/tuple of strings.

- If **None**, organization admins will not be updated.

- If **True**, all users using account authentication will automatically be added as admins of the organization.
- If **False**, no account authentication users will be automatically added as admins of the organization.
- If a string or list of strings, specifies the usernames and emails for users who will be added to the organization. Strings beginning and ending with / will be compiled into regular expressions; modifiers *i* (case-insensitive) and *m* (multi-line) may be specified after the ending /.

remove_admins: True/False. Defaults to **True**.

- When **True**, a user who does not match is removed from the organization’s administrative list.

users: None, True/False, string or list/tuple of strings. Same rules apply as for **admins**.

remove_users: True/False. Defaults to **True**. Same rules apply as for **remove_admins**.

```
{
  "Default": {
    "users": true
  },
  "Test Org": {
    "admins": ["admin@example.com"],
    "users": true
  },
  "Test Org 2": {
    "admins": ["admin@example.com", "/^tower-[^@]+?@.*$/i"],
    "users": "/^[^@].*?@example\\.com$/i"
  }
}
```

Organization mappings may be specified separately for each account authentication backend. If defined, these configurations will take precedence over the global configuration above.

```
SOCIAL_AUTH_GOOGLE_OAUTH2_ORGANIZATION_MAP = {}
SOCIAL_AUTH_GITHUB_ORGANIZATION_MAP = {}
SOCIAL_AUTH_GITHUB_ORG_ORGANIZATION_MAP = {}
SOCIAL_AUTH_GITHUB_TEAM_ORGANIZATION_MAP = {}
SOCIAL_AUTH_SAML_ORGANIZATION_MAP = {}
```

16.4.2 Team mapping

Team mapping is the mapping of team members (users) from social auth accounts. Keys are team names (will be created if not present). Values are dictionaries of options for each team’s membership, where each can contain the following parameters:

organization: string. The name of the organization to which the team belongs. The team will be created if the combination of organization and team name does not exist. The organization will first be created if it does not exist. If the license does not allow for multiple organizations, the team will always be assigned to the single default organization.

users: None, True/False, string or list/tuple of strings.

- If **None**, team members will not be updated.
- If **True/False**, all social auth users will be added/removed as team members.
- If a string or list of strings, specifies expressions used to match users. User will be added as a team member if the username or email matches. Strings beginning and ending with / will be compiled into regular expressions; modifiers *i* (case-insensitive) and *m* (multi-line) may be specified after the ending /.

remove: True/False. Defaults to **True**. When **True**, a user who does not match the rules above is removed from the team.

```
{
  "My Team": {
    "organization": "Test Org",
    "users": ["/^[^@]+?@test\\.example\\.com$/"],
    "remove": true
  },
  "Other Team": {
    "organization": "Test Org 2",
    "users": ["/^[^@]+?@test\\.example\\.com$/"],
    "remove": false
  }
}
```

Team mappings may be specified separately for each account authentication backend, based on which of these you setup. When defined, these configurations take precedence over the the global configuration above.

```
SOCIAL_AUTH_GOOGLE_OAUTH2_TEAM_MAP = {}
SOCIAL_AUTH_GITHUB_TEAM_MAP = {}
SOCIAL_AUTH_GITHUB_ORG_TEAM_MAP = {}
SOCIAL_AUTH_GITHUB_TEAM_TEAM_MAP = {}
SOCIAL_AUTH_SAML_TEAM_MAP = {}
```

Uncomment the line below (i.e. set SOCIAL_AUTH_USER_FIELDS to an empty list) to prevent new user accounts from being created. Only users who have previously logged in to Tower using social or enterprise-level authentication or have a user account with a matching email address will be able to login.

```
SOCIAL_AUTH_USER_FIELDS = []
```

SETTING UP ENTERPRISE AUTHENTICATION

This section describes setting up authentication for the following enterprise systems:

- *Azure Active Directory (AD)*
- *SAML Authentication Settings*
- *RADIUS Authentication Settings*
- *TACACS+ Authentication Settings*

Note: For LDAP authentication, see *Setting up LDAP Authentication*.

SAML, RADIUS, and TACACS+ users are categorized as ‘Enterprise’ users. The following rules apply to Enterprise users:

- Enterprise users can only be created via the first successful login attempt from remote authentication backend.
- Enterprise users cannot be created/authenticated if non-enterprise users with the same name has already been created in Tower.
- Tower passwords of enterprise users should always be empty and cannot be set by any user if there are enterprise backend-enabled.
- If enterprise backends are disabled, an enterprise user can be converted to a normal Tower user by setting the password field. However, this operation is irreversible, as the converted Tower user can no longer be treated as enterprise user.

17.1 Azure Active Directory (AD)

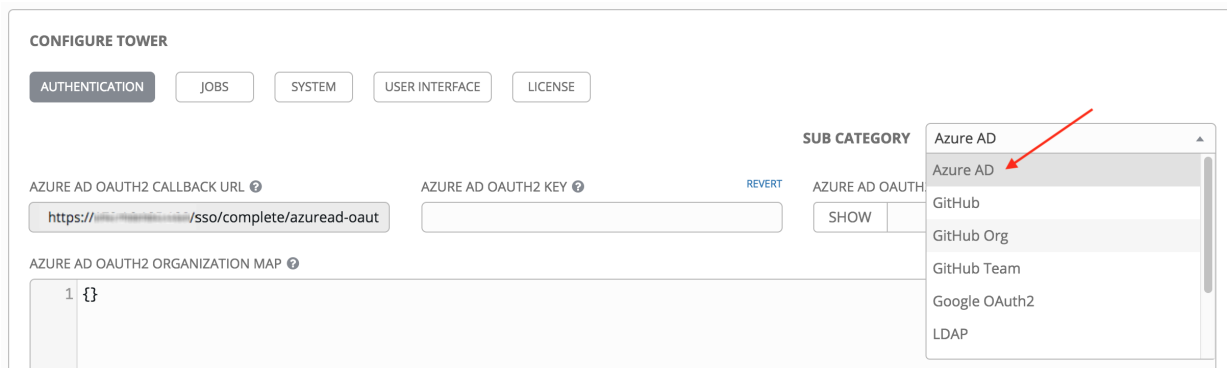
To set up enterprise authentication for Microsoft Azure Active Directory (AD), you will need to obtain an OAuth2 key and secret by registering your organization-owned application from Azure at <https://auth0.com/docs/connections/enterprise/azure-active-directory>. Each key and secret must belong to a unique application and cannot be shared or reused between different authentication backends. In order to register the application, you must supply it with your webpage URL, which is the Callback URL shown in the Configure Tower user interface.



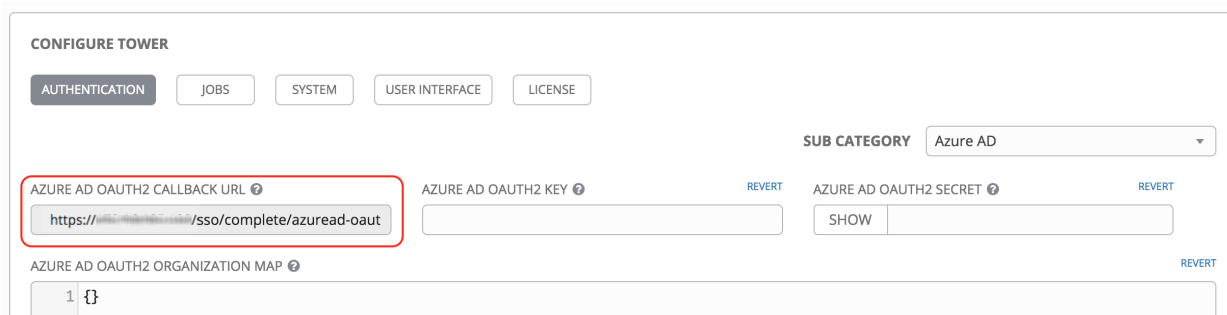
1. In the Ansible Tower User Interface, click the Settings () icon from the left navigation bar.

The **Configure Tower** window opens, displaying the Authentication tab initially by default.

2. In the Sub Category field, select **Azure AD** from the drop-down list.



3. Use the **Azure AD OAuth2 Callback URL** to supply Azure AD when it prompts for your application’s Sign-on URL.



Once the application is registered, Azure displays the Application ID and Object ID.

4. Copy and paste Azure’s Application ID to the **Azure AD OAuth2 Key** field of the Configure Tower - Authentication screen.

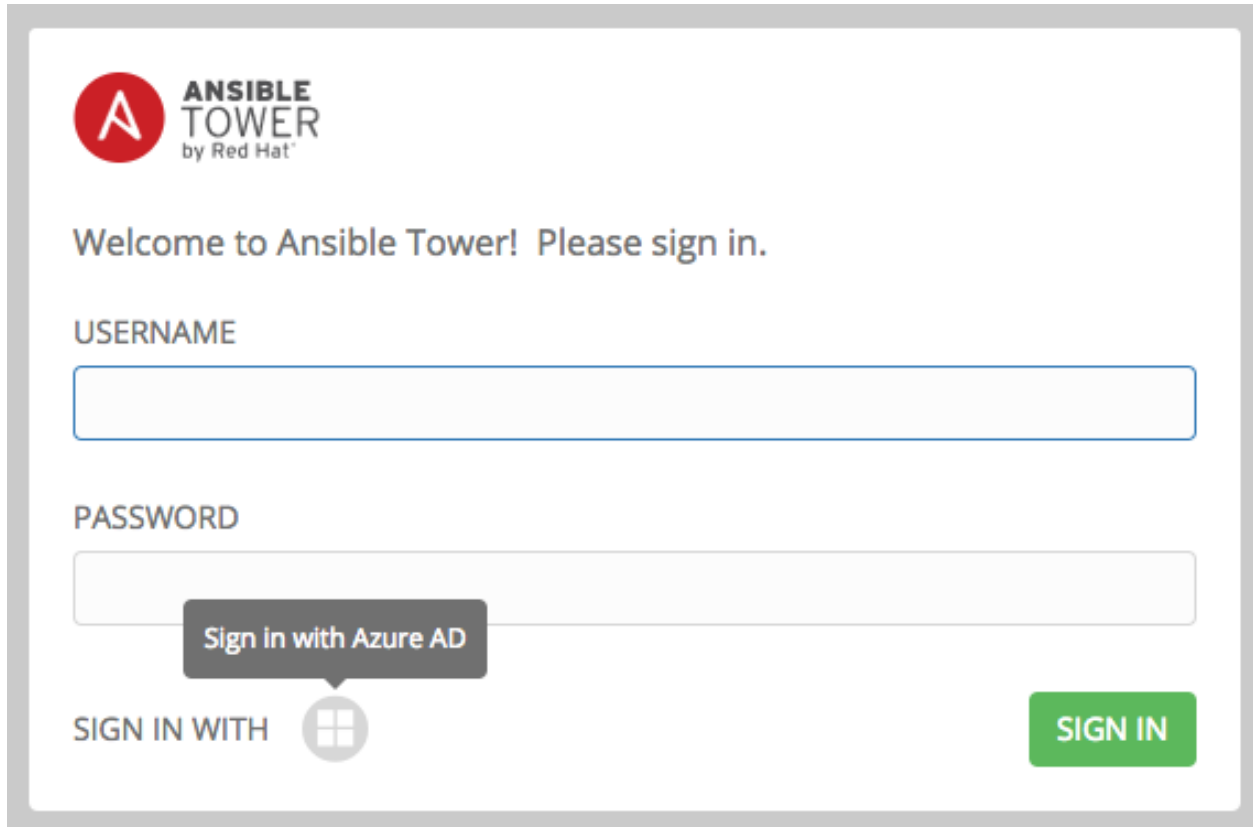
Following Azure AD’s documentation, [Connect your app to Microsoft Azure Active Directory: Create the Key](#), supply the key (shown at one time only) to the client for authentication.

5. Copy and paste the actual secret key created for your Azure AD application to the **Azure AD OAuth2 Secret** field of the Configure Tower - Authentication screen.

6. For details on completing the mapping fields, see *Organization and Team Mapping*.

7. Click **Save** when done.

8. To verify that the authentication was configured correctly, logout of Ansible Tower and the login screen will now display the Microsoft Azure logo to allow logging in with those credentials.




ANSIBLE TOWER
by Red Hat

Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

Sign in with Azure AD

SIGN IN WITH 

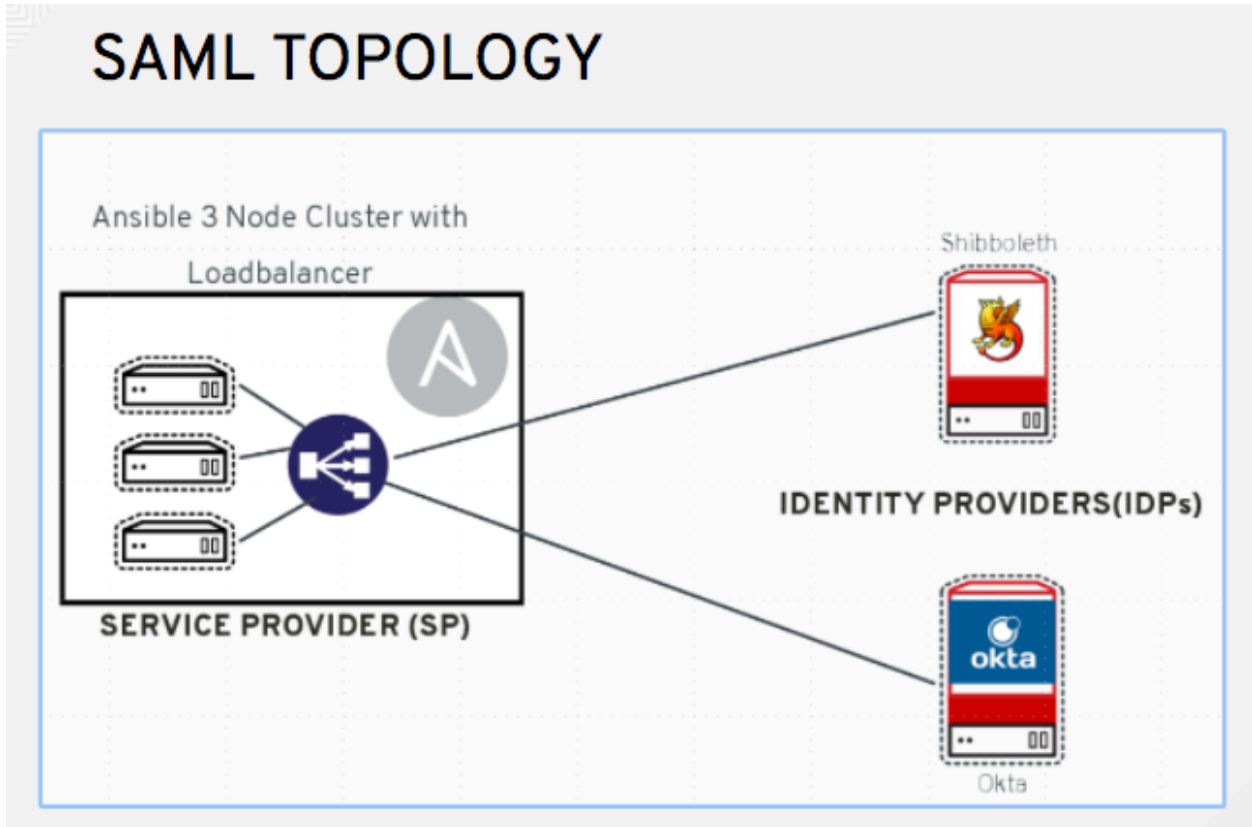
SIGN IN

For application registering basics in Azure AD, refer to: <https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-authentication-scenarios#basics-of-registering-an-application-in-azure-ad>

17.2 SAML Authentication Settings

Note: SAML authentication is a feature specific to Enterprise-level license holders.

SAML allows the exchange of authentication and authorization data between an Identity Provider (IdP - a system of servers that provide the Single Sign On service) and a Service Provider (in this case, Ansible Tower). Ansible Tower can be configured to talk with SAML in order to authenticate (create/login/logout) Tower users. User Team and Organization membership can be embedded in the SAML response to Tower.



The following instructions describe Ansible Tower as the service provider. To authenticate users through RHSSO (keycloak), refer to the [Red Hat Single Sign On Integration with Ansible Tower](#) blog.

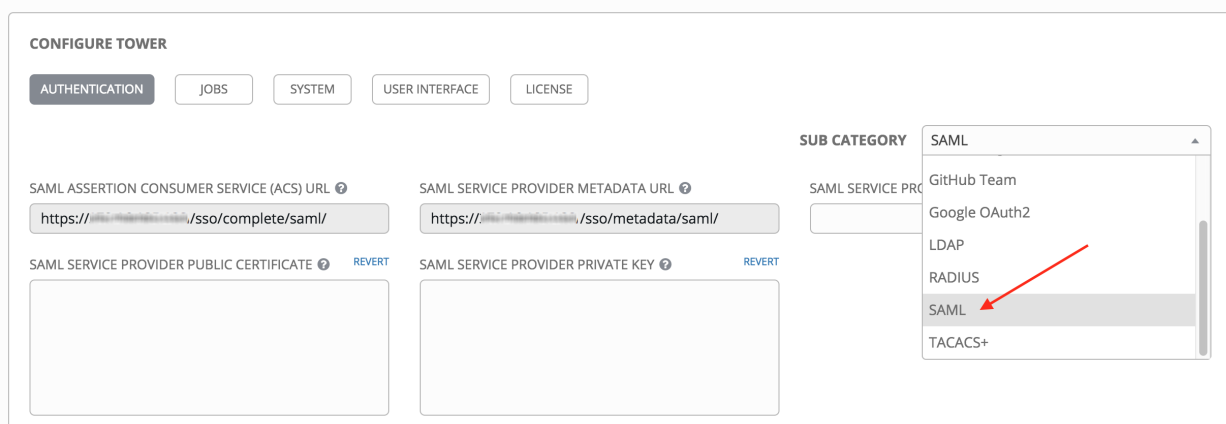
To setup SAML authentication:



1. In the Ansible Tower User Interface, click the Settings () icon from the left navigation bar.

The **Configure Tower** window opens, displaying the Authentication tab initially by default.

2. In the Sub Category field, select **SAML** from the drop-down list.



3. The **SAML Assertion Consume Service (ACS) URL** and **SAML Service Provider Metadata URL** fields are pre-populated and are non-editable. Contact the Identity Provider administrator and provide the information

contained in these fields.

4. Set the **SAML Service Provider Entity ID** to be the same as the Tower Base URL. The Tower Base URL can be found in the **System** tab of the Configure Tower screen, which you can access through the Settings



icon. Through the API, it can be viewed in the `/api/v2/settings/system`, under the `TOWER_BASE_URL` variable. The Entity ID can be set to any one of the individual Tower Cluster Nodes, but it is good practice to set it to the URL of the Service Provider. Ensure that the Base URL matches the FQDN of the load balancer (if used).

Note: The Tower Base URL is different for each node in a cluster. Commonly, a load balancer will sit in front of many tower cluster nodes to provide a single entry point, Tower Cluster FQDN. The SAML Service Provider must be able establish an outbound connection and route to the Tower Cluster Node or Tower Cluster FQDN set in the SAML Service Provider Entity ID.

In this example, the Service Provider is the Tower Cluster, and therefore, the ID is set to the Tower Cluster FQDN.

5. Create a server certificate for the Ansible cluster. Typically when an Ansible cluster is configured, the Tower nodes will be configured to handle HTTP traffic only and the load balancer will be an SSL Termination Point. In this case, an SSL certificate is required for the load balancer, and not for the individual Tower Cluster Nodes. SSL can either be enabled or disabled per individual Tower node, but should be disabled when using an SSL terminated load balancer. It is recommended to use a non-expiring self signed certificate to avoid periodically updating certificates. This way, authentication will not fail in case someone forgets to update the certificate.

Note: The **SAML Service Provider Public Certificate** field should contain the entire certificate, including the “`-----BEGIN CERTIFICATE-----`” and “`-----END CERTIFICATE-----`”.

If you are using a CA bundle with your certificate, include the entire bundle in this field.

As an example for public certs:

```
-----BEGIN CERTIFICATE-----
... cert text ...
-----END CERTIFICATE-----
```

6. Create an optional private key for Tower to use as a service provider (SP) and enter it in the **SAML Service Provider Private Key** field.

As an example for private keys:

```
-----BEGIN PRIVATE KEY--
... key text ...
-----END PRIVATE KEY-----
```

7. Optionally provide the IdP with some details about the Tower cluster during the SSO process in the **SAML Service Provider Organization Info** field.

```
{
  "en-US": {
    "url": "http://www.example.com",
    "displayname": "Example",
    "name": "example"
  }
}
```

For example:



8. Provide the IdP with the technical contact information in the **SAML Service Provider Technical Contact** field. Do not remove the contents of this field.

```
{
"givenName": "Some User",
"emailAddress": "suser@example.com"
}
```

For example:



9. Provide the IdP with the support contact information in the **SAML Service Provider Support Contact** field. Do not remove the contents of this field.

```
{
"givenName": "Some User",
"emailAddress": "suser@example.com"
}
```

For example:

```
SAML SERVICE PROVIDER SUPPORT CONTACT ⓘ
1 {
2   "givenName": "Central IT",
3   "emailAddress": "central-it@company.com"
4 }
```

10. In the **SAML Enabled Identity Providers** field, provide information on how to connect to each Identity Provider listed. Tower expects the following SAML attributes in the example below:

```
Username (urn:oid:0.9.2342.19200300.100.1.1)
Email (urn:oid:0.9.2342.19200300.100.1.3)
FirstName (urn:oid:2.5.4.42)
LastName (urn:oid:2.5.4.4)
```

If these attributes are not known, map existing SAML attributes to lastname, firstname, email and username.

Configure the required keys for each IDp:

- `attr_user_permanent_id` - the unique identifier for the user. It can be configured to match any of the attribute sent from the IDp. Usually, it is set to `name_id` if `SAML:nameid` attribute is sent to the Tower node or it can be the username attribute, or a custom unique identifier.
- `entity_id` - the Entity ID provided by the Identity Provider administrator. The admin creates a SAML profile for Tower and it generates a unique URL.
- `url` - the Single Sign On (SSO) URL Tower redirects the user to, when SSO is activated.
- `x509_cert` - the certificate provided by the IdP admin generated from the SAML profile created on the Identity Provider. Remove the `--BEGIN CERTIFICATE--` and `--END CERTIFICATE--` headers, then enter the cert as one non-breaking string.

Multiple SAML IDPs are supported. Some IDPs may provide user data using attribute names that differ from the default OIDs (<https://github.com/omab/python-social-auth/blob/master/social/backends/saml.py>). The SAML NameID is a special attribute used by some Identity Providers to tell the Service Provider (Tower cluster) what the unique user identifier is. If it is used, set the `attr_user_permanent_id` to `name_id` as shown in the example. Other attribute names may be overridden for each IDp as shown below.

```
{
  "myidp": {
    "entity_id": "https://idp.example.com",
    "url": "https://myidp.example.com/sso",
    "x509cert": ""
  },
  "onelogin": {
    "entity_id": "https://app.onelogin.com/saml/metadata/123456",
    "url": "https://example.onelogin.com/trust/saml2/http-post/sso/123456",
    "x509cert": "",
    "attr_user_permanent_id": "name_id",
    "attr_first_name": "User.FirstName",
    "attr_last_name": "User.LastName",
    "attr_username": "User.email",
    "attr_email": "User.email"
  }
}
```

```

1 {
2   "myidp": {
3     "entity_id": "https://idp.example.com",
4     "url": "https://myidp.example.com/sso",
5     "x509cert": "MIIEJjCCAwGgAwIBAgIUFuSD540PSBhdDhh3gZorvvrIaoAwDQYJKoZIhvcNAQEF\nbQAwXTElMkUkUDEFMhBGA
6   },
7   "onelogin": {
8     "entity_id": "https://app.onelogin.com/saml/metadata/123456",
9     "url": "https://example.onelogin.com/trust/saml2/http-post/sso/123456",
10    "x509cert": "MIIEJjCCAwGgAwIBAgIUFuSD540PSBhdDhh3gZorvvrIaoAwDQYJKoZIhvcNAQEF\nbQAwXTElMkUkUDEFMhBGA
11    "attr_user_permanent_id": "name_id",
12    "attr_first_name": "User.FirstName",
13    "attr_last_name": "User.LastName",
14    "attr_username": "User.email",
15    "attr_email": "User.email"
16  }
17 }
    
```

Note: The IdP provides the email, last name and firstname using the well known SAML urn. The IdP uses a custom SAML attribute to identify a user, which is an attribute that Tower is unable to read. Instead, Tower can understand the unique identifier name, which is the URN. Use the URN listed in the SAML “Name” attribute for the user attributes as shown in the example below.

```

SOCIAL_AUTH_SAML_ENABLED_IDPS": {
  "myidp": {
    "entity_id": "http://www.okta.com/exkaxvfm3m8SckPTG0h7",
    "x509cert": "MIIDpDCCAoygAwIBAgIGAVyqE/WRMA0GCSQ...[its a long string]",
    "url": "https://dev-643645.oktapreview.com/app/redhatdevtest_1/exkaxh7/sso/saml",
    "attr_user_permanent_id": "urn:oid:1.3.6.1.4.1.5555.610.2.2.1.11",
    "attr_username": "urn:oid:1.3.6.1.4.1.5555.610.2.2.1.11"
  }
}
    
```

11. Optionally provide in the **SAML Organization Map**. For further detail, see *Organization and Team Mapping*.
12. Tower can be configured to look for particular attributes that contain Team and Organization membership to associate with users when they log into Tower. The attribute names are defined in the **SAML Organization Attribute Mapping** and the **SAML Team Map** fields.

Example SAML Organization Attribute Mapping Below is an example SAML attribute that embeds user organization membership in the attribute *member-of*.

```

<saml2:AttributeStatement>
  <saml2:Attribute FriendlyName="member-of" Name="member-of"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue>Engineering</saml2:AttributeValue>
  <saml2:AttributeValue>IT</saml2:AttributeValue>
  <saml2:AttributeValue>HR</saml2:AttributeValue>
  <saml2:AttributeValue>Sales</saml2:AttributeValue>
  </saml2:Attribute>
  <saml2:Attribute FriendlyName="admin-of" Name="admin-of"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue>Engineering</saml2:AttributeValue>
  </saml2:Attribute>
</saml2:AttributeStatement>
    
```

Below is the corresponding Tower configuration.

```

{
  "saml_attr": "member-of",
  "saml_admin_attr": "admin-of",
}
    
```

(continues on next page)

(continued from previous page)

```
"remove": true,
"remove_admins": false
}
```

saml_attr: is the SAML attribute name where the organization array can be found and remove is set to **True** to remove a user from all organizations before adding the user to the list of Organizations. To keep the user in whatever Organization(s) they are in while adding the user to the Organization(s) in the SAML attribute, set remove to **False**.

saml_admin_attr: Similar to the saml_attr attribute, but instead of conveying organization membership, this attribute conveys admin organization permissions.

Example SAML Team Map Below is another example of a SAML attribute that contains a Team membership in a list.

```
<saml:AttributeStatement>
  <saml:Attribute
    xmlns:x500="urn:oasis:names:tc:SAML:2.0:profiles:attribute:X500"
    x500:Encoding="LDAP"
    NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format-uri"
    Name="urn:oid:1.3.6.1.4.1.5923.1.1.1.1"
    FriendlyName="eduPersonAffiliation">
    <saml:AttributeValue
      xsi:type="xs:string">member</saml:AttributeValue>
    <saml:AttributeValue
      xsi:type="xs:string">staff</saml:AttributeValue>
    </saml:Attribute>
  </saml:AttributeStatement>
```

```
{
  "saml_attr": "eduPersonAffiliation",
  "remove": true,
  "team_org_map": [
    {
      "team": "member",
      "organization": "Default1"
    },
    {
      "team": "staff",
      "organization": "Default2"
    }
  ]
}
```

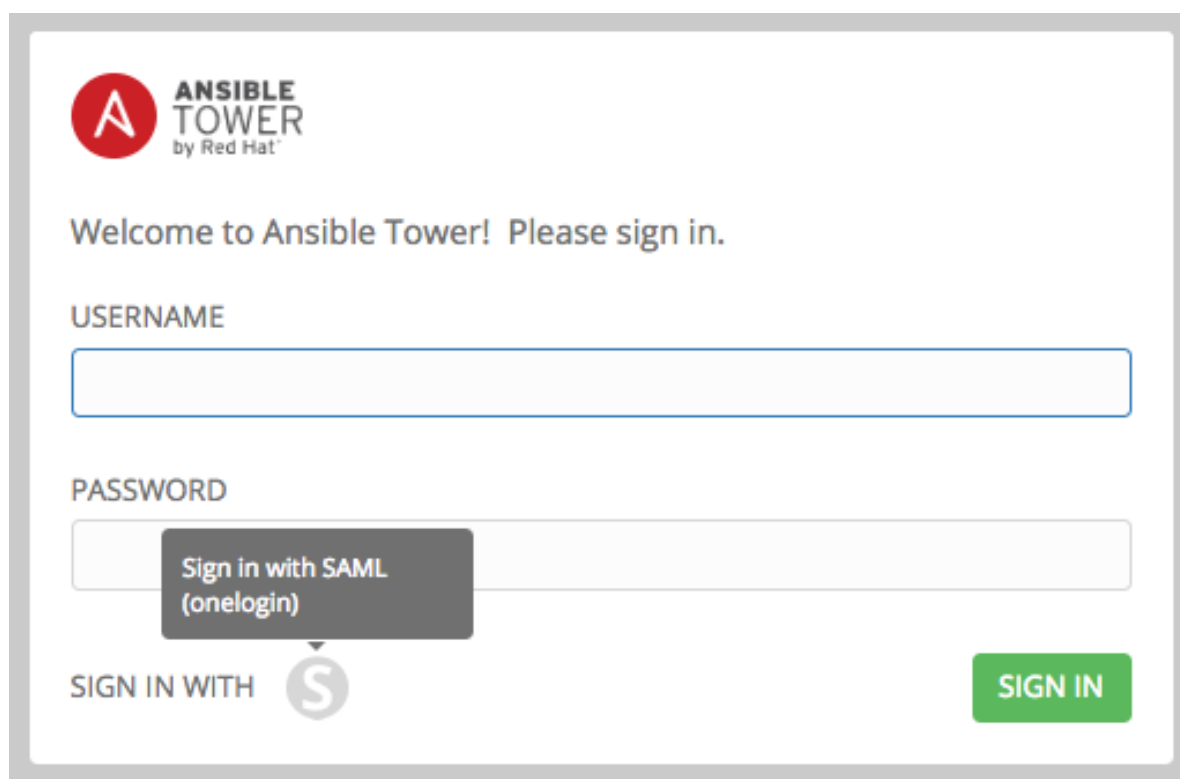
- saml_attr: The SAML attribute name where the team array can be found.
 - remove: Set remove to **True** to remove user from all Teams before adding the user to the list of Teams. To keep the user in whatever Team(s) they are in while adding the user to the Team(s) in the SAML attribute, set remove to **False**.
 - team_org_map: An array of dictionaries of the form { "team": "<AWX Team Name>", "organization": "<AWX Org Name>" } that defines mapping from Tower Team -> Tower Organization. This is needed because the same named Team can exist in multiple Organizations in Tower. The organization to which a team listed in a SAML attribute belongs to, would be ambiguous without this mapping.
13. Optionally provide team membership mapping in the **SAML Team Attribute Mapping** field. For further detail, see *Organization and Team Mapping*.
 14. Optionally provide security settings in the **SAML Security Config** field. This field is the equivalent to the

SOCIAL_AUTH_SAML_SECURITY_CONFIG field in the API. Refer to the [OneLogin's SAML Python Toolkit](#) for further detail.

Tower uses the `python-social-auth` library when users log in through SAML. This library relies on the `python-saml` library to make available the settings for the next two optional fields, **SAML Service Provider Extra Configuration Data** and **SAML IDP to EXTRA_DATA Attribute Mapping**.

15. The **SAML Service Provider Extra Configuration Data** field is equivalent to the SOCIAL_AUTH_SAML_SP_EXTRA in the API. Refer to the [python-saml library documentation](#) to learn about the valid service provider extra (SP_EXTRA) parameters.
16. The **SAML IDP to EXTRA_DATA Attribute Mapping** field is equivalent to the SOCIAL_AUTH_SAML_EXTRA_DATA in the API. See Python's [SAML Advanced Settings](#) documentation for more information.
17. Click **Save** when done.
18. To verify that the authentication was configured correctly, load the auto-generated URL found in the **SAML Service Provider Metadata URL** into a browser. It should output XML output, otherwise, it is not configured correctly.

Alternatively, logout of Ansible Tower and the login screen will now display the SAML logo to indicate it as a alternate method of logging into Ansible Tower.



17.3 RADIUS Authentication Settings

Note: RADIUS account authentication is a feature specific to Enterprise-level license holders.

Ansible Tower can be configured to centrally use RADIUS as a source for authentication information.



1. In the Ansible Tower User Interface, click the Settings () icon from the left navigation bar.

The **Configure Tower** window opens, displaying the Authentication tab initially by default.

2. In the Sub Category field, select **RADIUS** from the drop-down list.

3. Enter the Host or IP of the Radius server in the **Radius Server** field. If this field is left blank, Radius authentication is disabled.
4. Enter the port and secret information in the next two fields.

5. Click **Save** when done.

17.4 TACACS+ Authentication Settings

Note: TACACS+ account authentication is a feature specific to Enterprise-level license holders.

Terminal Access Controller Access-Control System Plus (TACACS+) is a protocol that handles remote authentication and related services for networked access control through a centralized server. In particular, TACACS+ provides authentication, authorization and accounting (AAA) services, in which you can configure Ansible Tower to use as a source for authentication.



1. In the Ansible Tower User Interface, click **Configure Tower** from the Settings () Menu screen. The Authentication tab displays initially by default.
2. In the Sub Category field, select **TACACS+** from the drop-down list.

The screenshot shows the 'CONFIGURE TOWER' interface with the 'AUTHENTICATION' tab selected. The 'SUB CATEGORY' dropdown menu is open, displaying a list of authentication methods: GitHub Team, Google OAuth2, LDAP, RADIUS, SAML, and TACACS+. A red arrow points to the 'TACACS+' option at the bottom of the list. Other fields visible include 'TACACS+ SERVER', 'TACACS+ PORT' (49), 'TACACS+ AUTH SESSION TIMEOUT' (5), and 'TACACS+ AUTHENTICATION PROTOCOL' (ascii).

3. Enter information in the following fields:

- **TACACS+ Server:** Provide the hostname or IP address of the TACACS+ server with which to authenticate. If this field is left blank, TACACS+ authentication is disabled.
- **TACACS+ Port:** TACACS+ uses port 49 by default, which is already pre-populated.
- **TACACS+ Secret:** Secret key for TACACS+ authentication server.
- **TACACS+ Auth Session Timeout:** Session timeout value in seconds. The default is 5 seconds.
- **TACACS+ Authentication Protocol:** The protocol used by TACACS+ client. Options are **ascii** or **pap**.

The screenshot shows the 'CONFIGURE TOWER' interface with the 'AUTHENTICATION' tab selected. The 'SUB CATEGORY' dropdown is now set to 'TACACS+'. The fields are filled with the following values: 'TACACS+ SERVER' is 'tacacsplus.example.com', 'TACACS+ PORT' is '49', 'TACACS+ AUTH SESSION TIMEOUT' is '5', and 'TACACS+ AUTHENTICATION PROTOCOL' is 'ascii'. The 'TACACS+ SECRET' field is masked with dots. There are 'CANCEL' and 'SAVE' buttons at the bottom right.

4. Click **Save** when done.

SETTING UP LDAP AUTHENTICATION

Note: LDAP authentication is a feature specific to Enterprise-level license holders. You must have an active enterprise license before beginning the configuration process. Additionally, if the LDAP server you want to connect to has a certificate that is self-signed or signed by a corporate internal certificate authority (CA), the CA certificate must be added to the system's trusted CAs. Otherwise, connection to the LDAP server will result in an error that the certificate issuer is not recognized.

Administrators use LDAP as a source for account authentication information for Tower users. User authentication is provided, but not the synchronization of user permissions and credentials. Organization membership (as well as the organization admin) and team memberships can be synchronized.

When so configured, a user who logs in with an LDAP username and password automatically gets a Tower account created for them and they can be automatically placed into organizations as either regular users or organization administrators.

Users created via an LDAP login cannot change their username, first name, last name, or set a local password for themselves. This is also tunable to restrict editing of other field names.

To configure LDAP integration for Tower:


1. First, create a user in LDAP that has access to read the entire LDAP structure.
2. Test if you can make successful queries to the LDAP server, use the `ldapsearch` command, which is a command line tool that can be installed on the tower system's command line as well as on other Linux and OSX systems. Use the following command to query the ldap server, where *josie* and *Josie4Cloud* are replaced by attributes that work for your setup:

```
ldapsearch -x -H ldap://win -D "CN=josie,CN=Users,DC=website,DC=com" -b "dc=website,  
↪dc=com" -w Josie4Cloud
```

Here `CN=josie,CN=users,DC=website,DC=com` is the Distinguished Name of the connecting user.

Note: The `ldapsearch` utility is not automatically pre-installed with Ansible Tower, however, you can install it from the `openldap-clients` package.



3. From the left navigation bar, click the Settings () icon.

The Authentication tab displays initially by default.

4. In the Sub Category field, select **LDAP** from the drop-down list.

Note: Starting with Ansible Tower 3.3, you can configure multiple LDAP servers by specifying the server to configure (otherwise, leave the server at **Default**):

The equivalent API endpoints will show AUTH_LDAP_* repeated: AUTH_LDAP_1_*, AUTH_LDAP_2_*, ..., AUTH_LDAP_5_* to denote server designations.

5. Enter the LDAP server address to connect to in the **LDAP Server URI** field using the same format as the one shown in the text field. Below is an example:

Note: If the LDAP server you want to connect to has a certificate that is self-signed or signed by a corporate internal certificate authority (CA), the CA certificate must be added to the system's trusted CAs. Otherwise, connection to the LDAP server will result in an error that the certificate issuer is not recognized.

6. Enter the Distinguished Name in the **LDAP Bind DN** text field to specify the user that Tower uses to connect (Bind) to the LDAP server. Below uses the example, CN=josie, CN=users, DC=website, DC=com:

LDAP BIND DN ?
REVERT

CN=josie,CN=users,DC=website,DC=com

7. Enter the password to use for the Binding user in the **LDAP Bind Password** text field. In this example, the password is 'passme':

LDAP BIND PASSWORD ?

SHOW
.....

8. If that name is stored in key `sAMAccountName`, the **LDAP User DN Template** populates with `(sAMAccountName=%(user)s)`. Active Directory stores the username to `sAMAccountName`. Similarly, for OpenLDAP, the key is `uid`—hence the line becomes `(uid=%(user)s)`.
9. Click to select a group type from the **LDAP Group Type** drop-down menu list.

LDAP Group Types include:

- PosixGroupType
- GroupOfNamesType
- GroupOfUniqueNamesType
- ActiveDirectoryGroupType
- OrganizationalRoleGroupType
- MemberDNGroupType
- NISGroupType
- NestedGroupOfNamesType
- NestedGroupOfUniqueNamesType
- NestedActiveDirectoryGroupType
- NestedOrganizationalRoleGroupType
- NestedMemberDNGroupType
- PosixUIDGroupType

The LDAP Group Types that are supported by Tower leverage the underlying `django-auth-ldap` library.

Each **LDAP Group Type** can potentially take different parameters. Tower exposes `LDAP_GROUP_TYPE_PARAMS` to account for this. `LDAP_GROUP_TYPE_PARAMS` is a dictionary, which will be converted by Tower to kwargs and passed to the LDAP Group Type class selected. There are two common parameters used by any of the LDAP Group Type; `name_attr` and `member_attr`. Where `name_attr` defaults to `cn` and `member_attr` defaults to `member`:

```
{"name_attr": "cn", "member_attr": "member"}
```

To determine what parameters a specific LDAP Group Type expects, refer to the `django_auth_ldap` documentation around the classes `init` parameters.

10. Enter the group distinguish name to allow users within that group to access Tower in the **LDAP Require Group** field, using the same format as the one shown in the text field. In this example, use: `CN=Tower Users, OU=Users, DC=website, DC=com`

11. Enter the group distinguish name to prevent users within that group to access Tower in the **LDAP Deny Group** field, using the same format as the one shown in the text field. In this example, leave the field blank.
12. The **LDAP Start TLS** is disabled by default. To enable TLS when the LDAP connection is not using SSL, click the toggle to **ON**.

13. Enter where to search for users while authenticating in the **LDAP USER SEARCH** field using the same format as the one shown in the text field. In this example, use:

```
[
"OU=Users,DC=website,DC=com",
"SCOPE_SUBTREE",
"(cn=%(user)s)"
]
```

The first line specifies where to search for users in the LDAP tree. In the above example, the users are searched recursively starting from `DC=website,DC=com`.

The second line specifies the scope where the users should be searched:

- **SCOPE_BASE**: This value is used to indicate searching only the entry at the base DN, resulting in only that entry being returned
- **SCOPE_ONELEVEL**: This value is used to indicate searching all entries one level under the base DN - but not including the base DN and not including any entries under that one level under the base DN.
- **SCOPE_SUBTREE**: This value is used to indicate searching of all entries at all levels under and including the specified base DN.

The third line specifies the key name where the user name is stored.

```
LDAP USER SEARCH ?
1 [
2 "OU=Users,DC=website,DC=com",
3 "SCOPE_SUBTREE",
4 "(cn=%(user)s)"
5 ]
```

Note: For multiple search queries, the proper syntax is:

```
[
  [
    "OU=Users,DC=northamerica,DC=acme,DC=com",
    "SCOPE_SUBTREE",
    "(sAMAccountName=%(user)s)"
  ],
  [
    "OU=Users,DC=apac,DC=corp,DC=com",
    "SCOPE_SUBTREE",
    "(sAMAccountName=%(user)s)"
  ],
  [
    "OU=Users,DC=emea,DC=corp,DC=com",
    "SCOPE_SUBTREE",
    "(sAMAccountName=%(user)s)"
  ]
]
```

14. In the **LDAP Group Search** text field, specify which groups should be searched and how to search them. In this example, use:

```
[
"dc=example,dc=com",
"SCOPE_SUBTREE",
"(objectClass=group)"
]
```

- The first line specifies the BASE DN where the groups should be searched.
- The second lines specifies the scope and is the same as that for the user directive.
- The third line specifies what the `objectclass` of a group object is in the LDAP you are using.

```
LDAP GROUP SEARCH ?
1 [
2 "dc=example,dc=com",
3 "SCOPE_SUBTREE",
4 "(objectClass=group)"
5 ]
6
```

15. Enter the user attributes in the **LDAP User Attribute Map** the text field. In this example, use:

```
{
"first_name": "givenName",
"last_name": "sn",
"email": "mail"
}
```

The above example retrieves users by last name from the key `sn`. You can use the same LDAP query for the user to figure out what keys they are stored under.

```
LDAP USER ATTRIBUTE MAP REVERT
1 {
2 "first_name": "givenName",
3 "last_name": "sn",
4 "email": "mail"
5 }
```

16. Enter the user profile flags in the **LDAP User Flags by Group** the text field. In this example, use the following syntax to set LDAP users as “Superusers” and “Auditors”:

```
{
"is_superuser": "cn=superusers,ou=groups,dc=website,dc=com",
"is_system_auditor": "cn=auditors,ou=groups,dc=website,dc=com"
}
```

The above example retrieves users who are flagged as superusers or as auditor in their profile.

```
LDAP USER FLAGS BY GROUP REVERT
1 {
2 "is_superuser": "cn=superusers,ou=groups,dc=website,dc=com",
3 "is_system_auditor": "cn=auditors,ou=groups,dc=website,dc=com"
4 }
```

17. For details on completing the mapping fields, see *LDAP Organization and Team Mapping*.

18. Click **Save** when done.

With these values entered on this form, you can now make a successful authentication with LDAP.

Note: Tower does not actively sync users, but they are created during their initial login.

18.1 Referrals

Active Directory uses “referrals” in case the queried object is not available in its database. It has been noted that this does not work properly with the django LDAP client and, most of the time, it helps to disable referrals. Disable LDAP referrals by adding the following lines to your `/etc/tower/conf.d/custom.py` file:

```
AUTH_LDAP_GLOBAL_OPTIONS = {
    ldap.OPT_REFERRALS: False,
}
```

Note: “Referrals” are disabled by default in Ansible Tower version 2.4.3 and above. If you are running an earlier version of Tower, you should consider adding this parameter to your configuration file.

For details on completing the mapping fields, see *LDAP Organization and Team Mapping*.

18.2 Enabling Logging for LDAP

To enable logging for LDAP, you must set the level to `DEBUG` in the LDAP configuration file, `/etc/tower/conf.d/custom.py`:

```
LOGGING['handlers']['tower_warnings']['level'] = 'DEBUG'
```

18.3 LDAP Organization and Team Mapping

Next, you will need to control which users are placed into which Tower organizations based on LDAP attributes (mapping out between your organization admins/users and LDAP groups).

Keys are organization names. Organizations will be created if not present. Values are dictionaries defining the options for each organization’s membership. For each organization, it is possible to specify what groups are automatically users of the organization and also what groups can administer the organization.

admins: `None`, `True/False`, string or list/tuple of strings.

- If `None`, organization admins will not be updated based on LDAP values.
- If `True`, all users in LDAP will automatically be added as admins of the organization.
- If `False`, no LDAP users will be automatically added as admins of the organization.
- If a string or list of strings, specifies the group DN(s) that will be added of the organization if they match any of the specified groups.

remove_admins: `True/False`. Defaults to `False`.

- When `True`, a user who is not a member of the given groups will be removed from the organization’s administrative list.

users: None, True/False, string or list/tuple of strings. Same rules apply as for **admins**.

remove_users: True/False. Defaults to **False**. Same rules apply as **remove_admins**.

```
{
"LDAP Organization": {
  "admins": "cn=engineering_admins,ou=groups,dc=example,dc=com",
  "remove_admins": false,
  "users": [
    "cn=engineering,ou=groups,dc=example,dc=com",
    "cn=sales,ou=groups,dc=example,dc=com",
    "cn=it,ou=groups,dc=example,dc=com"
  ],
  "remove_users": false
},
"LDAP Organization 2": {
  "admins": [
    "cn=Administrators,cn=Builtin,dc=example,dc=com"
  ],
  "remove_admins": false,
  "users": true,
  "remove_users": false
}
}
```

Mapping between team members (users) and LDAP groups. Keys are team names (will be created if not present). Values are dictionaries of options for each team's membership, where each can contain the following parameters:

organization: string. The name of the organization to which the team belongs. The team will be created if the combination of organization and team name does not exist. The organization will first be created if it does not exist.

users: None, True/False, string or list/tuple of strings.

- If **None**, team members will not be updated.
- If **True/False**, all LDAP users will be added/removed as team members.
- If a string or list of strings, specifies the group DN(s). User will be added as a team member if the user is a member of ANY of these groups.

remove: True/False. Defaults to **False**. When **True**, a user who is not a member of the given groups will be removed from the team.

```
{
"LDAP Engineering": {
  "organization": "LDAP Organization",
  "users": "cn=engineering,ou=groups,dc=example,dc=com",
  "remove": true
},
"LDAP IT": {
  "organization": "LDAP Organization",
  "users": "cn=it,ou=groups,dc=example,dc=com",
  "remove": true
},
"LDAP Sales": {
  "organization": "LDAP Organization",
  "users": "cn=sales,ou=groups,dc=example,dc=com",
  "remove": true
}
```

(continues on next page)


(continued from previous page)

```
}  
}
```

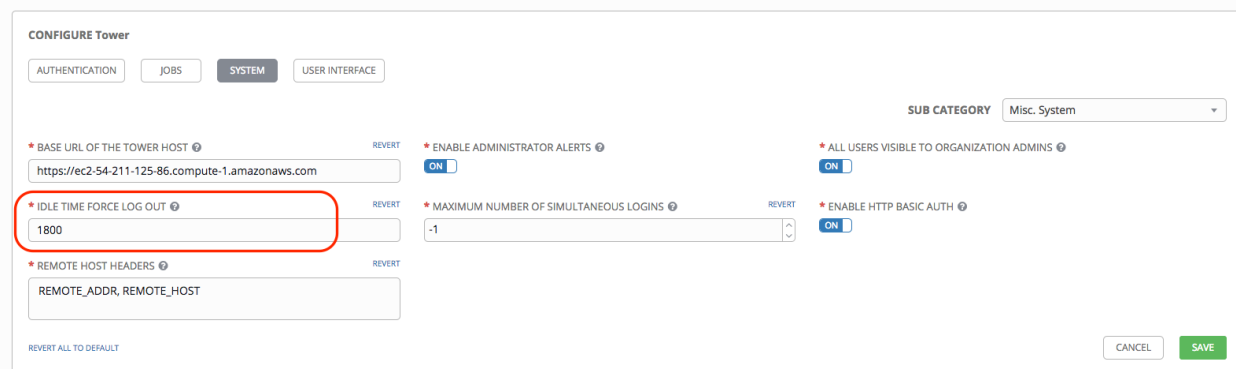
CHANGING THE DEFAULT TIMEOUT FOR AUTHENTICATION

Introduced in Ansible Tower 2.4 is a feature which adds an `Auth-Token-Timeout` to every response that includes a valid user-supplied token. This setting changed to `Session-Timeout` in Ansible Tower 3.3. The value of `Session-Timeout` is determined by the configuration (time expressed in seconds) of the `SESSION_COOKIE_AGE`.

The value of `Session-Timeout` indicates the length of time, in seconds, that the supplied token is valid, from the moment the request was initiated. Starting in Ansible Tower 3.2, you can change this setting in the Configure Tower user interface:

1. From the Settings () Menu screen, click on **Configure Tower**.
2. Select the **System** tab.
3. Select **Misc. System** option from the Sub Category drop-down menu list if not already selected.
4. Enter the timeout period in seconds in the **Idle Time Force Log Out** text field.

SETTINGS / EDIT CONFIGURATION



The screenshot shows the 'CONFIGURE Tower' interface with the 'SYSTEM' tab selected. The 'SUB CATEGORY' is set to 'Misc. System'. The 'IDLE TIME FORCE LOG OUT' field is highlighted with a red box and contains the value '1800'. Other visible settings include 'BASE URL OF THE TOWER HOST' (https://ec2-54-211-125-86.compute-1.amazonaws.com), 'ENABLE ADMINISTRATOR ALERTS' (ON), 'ALL USERS VISIBLE TO ORGANIZATION ADMINS' (ON), 'MAXIMUM NUMBER OF SIMULTANEOUS LOGINS' (-1), and 'ENABLE HTTP BASIC AUTH' (ON). There are 'REVERT' and 'REVERT ALL TO DEFAULT' buttons, and 'CANCEL' and 'SAVE' buttons at the bottom right.

5. Click **Save** to apply your changes.

Note: The `local_settings.json` file is no longer used starting with Ansible Tower 3.2.

Note: If you are accessing Tower directly and are having trouble getting your authentication to stay, in that you have to keep logging in over and over, try clearing your web browser's cache. In situations like this, it is often found that the authentication token has been cached in the browser session and must be cleared.

USER AUTHENTICATION WITH KERBEROS

User authentication via Active Directory (AD), also referred to as authentication through Kerberos, is supported through Ansible Tower.

To get started, first setup the Kerberos packages in the Tower system so that you can successfully generate a Kerberos ticket. To install the packages, use the following steps:

```
yum install krb5-workstation
yum install krb5-devel
yum install krb5-libs
```

Once installed, edit the `/etc/krb.conf` file, as follows, to provide the address of the AD, the domain, etc.:

```
[logging]
default = FILE:/var/log/krb5libs.log
kdc = FILE:/var/log/krb5kdc.log
admin_server = FILE:/var/log/kadmind.log

[libdefaults]
default_realm = WEBSITE.COM
dns_lookup_realm = false
dns_lookup_kdc = false
ticket_lifetime = 24h
renew_lifetime = 7d
forwardable = true

[realms]
WEBSITE.COM = {
  kdc = WIN-SA2TXZOTVMV.website.com
  admin_server = WIN-SA2TXZOTVMV.website.com
}

[domain_realm]
.website.com = WEBSITE.COM
website.com = WEBSITE.COM
```

After the configuration file has been updated, you should be able to successfully authenticate and get a valid token. The following steps show how to authenticate and get a token:

```
[root@ip-172-31-26-180 ~]# kinit username
Password for username@WEBSITE.COM:
[root@ip-172-31-26-180 ~]#

Check if we got a valid ticket.
```

(continues on next page)

(continued from previous page)

```
[root@ip-172-31-26-180 ~]# klist
Ticket cache: FILE:/tmp/krb5cc_0
Default principal: username@WEBSITE.COM

Valid starting      Expires            Service principal
01/25/16 11:42:56  01/25/16 21:42:53  krbtgt/WEBSITE.COM@WEBSITE.COM
    renew until 02/01/16 11:42:56
[root@ip-172-31-26-180 ~]#
```

Once you have a valid ticket, you can check to ensure that everything is working as expected from command line. To test this, make sure that your inventory looks like the following:

```
[windows]
win01.WEBSITE.COM

[windows:vars]
ansible_user = username@WEBSITE.COM
ansible_connection = winrm
ansible_port = 5986
```

You should also:

- Ensure that the hostname is the proper client hostname matching the entry in AD and is not the IP address.
- In the username declaration, ensure that the domain name (the text after @) is properly entered with regard to upper- and lower-case letters, as Kerberos is case sensitive. For Tower, you should also ensure that the inventory looks the same.

Note: If you encounter a `Server not found in Kerberos database` error message, and your inventory is configured using FQDNs (**not IP addresses**), ensure that the service principal name is not missing or mis-configured.

Now, running a playbook should run as expected. You can test this by running the playbook as the `awx` user.

Once you have verified that playbooks work properly, integration with Tower is easy. Generate the Kerberos ticket as the `awx` user and Tower should automatically pick up the generated ticket for authentication.

Note: The python `kerberos` package must be installed. Ansible is designed to check if `kerberos` package is installed and, if so, it uses kerberos authentication.

20.1 AD and Kerberos Credentials

Active Directory only:

- If you are only planning to run playbooks against Windows machines with AD usernames and passwords as machine credentials, you can use “`user@<domain>`” format for the username and an associated password.

With Kerberos:

- If Kerberos is installed, you can create a machine credential with the username and password, using the “`user@<domain>`” format for the username.

20.2 Working with Kerberos Tickets

Ansible defaults to automatically managing kerberos tickets (as of Ansible 2.3) when both the username and password are specified in the machine credential for a host that is configured for kerberos. A new ticket is created in a temporary credential cache for each host, before each task executes (to minimize the chance of ticket expiration). The temporary credential caches are deleted after each task, and will not interfere with the default credential cache.

To disable automatic ticket management (e.g., to use an existing SSO ticket or call `kinit` manually to populate the default credential cache), set `ansible_winrm_kinit_mode=manual` via the inventory.

Automatic ticket management requires a standard `kinit` binary on the control host system path. To specify a different location or binary name, set the `ansible_winrm_kinit_cmd` inventory variable to the fully-qualified path to an MIT `krb5` `kinit`-compatible binary.

WORKING WITH SESSION LIMITS

Setting a session limit allows administrators to limit the number of simultaneous sessions per user or per IP address.

In Ansible Tower, a session is created for each browser that a user uses to log in, which forces the user to log out any extra sessions after they exceed the administrator-defined maximum.

Session limits may be important, depending on your particular setup. For example, perhaps you only want a single user on your system with a single login per device (where the user could log in on his work laptop, phone, or home computer). In such a case, you would want to create a session limit equal to 1 (one). If the user logs in on his laptop, for example, then logs in using his phone, his laptop session expires (times out) and only the login on the phone persists.

While session counts can be very limited, they can also be expanded to cover as many session logins as are needed by your organization.

When a user logs in and their login results in other users being logged out, the session limit has been reached and those users who are logged out are notified as to why the logout occurred.

To make changes to your session limits, navigate to `/etc/tower/conf.d` and edit the `sessions.py` file or use the [Browsable API](#) if you are comfortable with making REST requests. The settings you should change are detailed below:

```
# Seconds before auth tokens expire.
SESSION_COOKIE_AGE = 1800

# Maximum number of per-user valid, concurrent tokens.
# -1 is unlimited
SESSIONS_PER_USER = -1

# Enable / Disable HTTP Basic Authentication used in the API browser
# Note: Session limits are not enforced when using HTTP Basic Authentication.
AUTH_BASIC_ENABLED = True
```

Note: To make the best use of session limits, disable `AUTH_BASIC_ENABLED` by changing the value to `False`, as it falls outside of the scope of session limit enforcement.

Caution: Proactive session limits will kick the user out when the session is idle. It is strongly recommended that you do not set the session limit to anything less than 1 minute, as doing so will break your Ansible Tower instance.

BACKING UP AND RESTORING TOWER

The ability to backup and restore your system(s) has been integrated into the Tower setup playbook. Refer to *Backup and Restore for Clustered Environments* for additional considerations.

Note: When restoring, be sure to restore to the same version from which it was backed up. However, you should always use the most recent minor version of a release to backup and/or restore your Tower installation version. For example, if the current version of Tower that you are on is 3.0.0, 3.0.1, or 3.0.2, use only the 3.0.2 installer.

The Tower setup playbook is invoked as `setup.sh` from the path where you unpacked the Tower installer tarball. It uses the same inventory file used by the install playbook. The setup script takes the following arguments for backing up and restoring:

- `-b` Perform a database backup rather than an installation.
- `-r` Perform a database restore rather than an installation.

As the root user, call `setup.sh` with the appropriate parameters and Tower backup or restored as configured.

```
root@localhost:~# ./setup.sh -b
```

```
root@localhost:~# ./setup.sh -r
```

A default restore path is used unless `EXTRA_VARS` are provided with a non-default path, as shown in the example below:

```
root@localhost:~# ./setup.sh -e 'restore_backup_file=/path/to/nondefault/backup.tar.gz  
↪' -r
```

Optionally, you can override the inventory file used by passing it as an argument to the setup script:

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

22.1 Backup/Restore Playbooks

In addition to the `install.yml` file included with your `setup.sh` setup playbook, there are also `backup.yml` and `restore.yml` files for your backup and restoration needs.

These playbooks serve two functions—backup and restore.

- The overall backup will backup:
 1. the database

2. the `SECRET_KEY` file
- The per-system backups include:
 1. custom user config files
 2. job stdout
 3. manual projects
 - The restore will restore the backed up files and data to a freshly installed and working second instance of Tower.

When restoring your system, Tower checks to see that the backup file exists before beginning the restoration. If the backup file is not available, your restoration will fail.

Note: Ensure your Tower host(s) are properly set up with SSH keys or user/pass variables in the hosts file, and that the user has sudo access.

22.2 Backup and Restoration Considerations

- **Disk Space:** Review your disk space requirements to ensure you have enough room to backup configuration files, keys, and other relevant files, plus the database of the Tower installation.
- **System Credentials:** Confirm you have the system credentials you need when working with a local database or a remote database. On local systems, you may need root or `sudo` access, depending on how credentials were setup. On remote systems, you may need different credentials to grant you access to the remote system you are trying to backup or restore.
- You should always use the most recent minor version of a release to backup and/or restore your Tower installation version. For example, if the current version of Tower that you are on is 3.0.0, 3.0.1, or 3.0.2, use only the 3.0.2 installer.
- When using `setup.sh` to do a restore from the default restore file path, `/var/lib/awx`, `-r` is still required in order to do the restore, but it no longer accepts an argument. If a non-default restore file path is needed, the user must provide this as an extra var (`root@localhost:~# ./setup.sh -e 'restore_backup_file=/path/to/nondefault/backup.tar.gz' -r`).
- If the backup file is placed in the same directory as the `setup.sh` installer, the restore playbook will automatically locate the restore files. In this case, you do not need to use the `restore_backup_file` extra var to specify the location of the backup file.

22.3 Backup and Restore for Clustered Environments

The procedure for backup and restore for a clustered environment is similar to a single install, except with some considerations described in this section.

- If restoring to a new cluster, make sure the old cluster is shut down before proceeding because they could conflict with each other when accessing the database.
- Per-node backups will only be restored to nodes bearing the same hostname as the backup.

When restoring to an existing cluster, the restore contains:

- Dump of the Postgres database
- UI artifacts (included in database dump)

- Tower configuration (retrieved from `/etc/tower`)
- Tower secret key
- Manual projects
- Job stdout

22.3.1 Restoring to a different cluster

When restoring a backup to a separate instance or cluster, host-specific configuration (including manual projects, job output, and custom settings under `/etc/tower`) is not restored to any hosts in the new environment.

The restore process will not alter instance groups present before the restore (neither will it introduce any new instance groups). Restored Tower resources that were associated to instance groups will likely need to be reassigned to instance groups present on the new Tower cluster.

USING CUSTOM LOGOS IN ANSIBLE TOWER

Note: Custom rebranding was added to Ansible Tower with version 2.4.0 and is available to Enterprise-level license holders.

Ansible Tower supports the use of a custom logo. To set up a custom logo, navigate to `Configure Tower` in the settings menu and upload it via the `Custom Logo` item. For the custom logo to look its best, use a `.png` file with a transparent background. GIF, PNG, and JPEG formats are supported.

Selecting `Revert` will result in the appearance of the standard Ansible Tower logo.

If needed, you can add specific information (such as a legal notice or a disclaimer) to a text box in the login modal by adding it to the `Custom Login Info` box on the same page.

For example, if you uploaded a specific logo, and added the following text:

```
"The mustard indicates progress."
```

The Tower login dialog would look like this:



Welcome to Ansible Tower! Please sign in.

* Username

* Password

Notice

The mustard indicates progress.

➔ Sign in

TROUBLESHOOTING TOWER

24.1 Error logs

Tower server errors are logged in `/var/log/tower`. Supervisors logs can be found in `/var/log/supervisor/`. Nginx web server errors are logged in the `httpd` error log. Configure other Tower logging needs in `/etc/tower/conf.d/`.

Explore client-side issues using the JavaScript console built into most browsers and report any errors to Ansible via the Red Hat Customer portal at <https://access.redhat.com/>.

24.2 Problems connecting to your host

If you are unable to run the `helloworld.yml` example playbook from the Quick Start Guide or other playbooks due to host connection errors, try the following:

- Can you `ssh` to your host? Ansible depends on SSH access to the servers you are managing.
- Are your hostnames and IPs correctly added in your inventory file? (Check for typos.)

24.3 Unable to login to Tower via HTTP

Access to Tower is intentionally restricted through a secure protocol (HTTPS). In cases where your configuration is set up to run a Tower node behind a load balancer or proxy as “HTTP only”, and you only want to access it without SSL (for troubleshooting, for example), you must adjust the following settings in the `settings.py` file located at `/etc/tower/` of your tower instance:

```
SESSION_COOKIE_SECURE = False
CSRF_COOKIE_SECURE = False
```

Changing these settings to `False` will allow Tower to manage cookies and login sessions when using the HTTP protocol. This must be done on every node of a cluster installation to properly take effect.

To apply the changes, run:

```
ansible-tower-service restart
```

24.4 WebSockets port for live events not working

Ansible Tower uses port 80/443 on the Tower server to stream live updates of playbook activity and other events to the client browser. These ports are configured for 80/443 by default, but if they are blocked by firewalls, close any firewall rules that opened up or added for the previous websocket ports, this will ensure your firewall allows traffic through this port.

24.5 Problems running a playbook

If you are unable to run the `helloworld.yml` example playbook from the Quick Start Guide or other playbooks due to playbook errors, try the following:

- Are you authenticating with the user currently running the commands? If not, check how the username has been setup or pass the `--user=username` or `-u username` commands to specify a user.
- Is your YAML file correctly indented? You may need to line up your whitespace correctly. Indentation level is significant in YAML. You can use `yamllint` to check your playbook. For more information, refer to the YAML primer at: <http://docs.ansible.com/YAMLSyntax.html>
- Items beginning with a `-` are considered list items or plays. Items with the format of `key: value` operate as hashes or dictionaries. Ensure you don't have extra or missing `-` plays.

24.6 Problems when running a job

If you are having trouble running a job from a playbook, you should review the playbook YAML file. When importing a playbook, either manually or via a source control mechanism, keep in mind that the host definition is controlled by Tower and should be set to `hosts: all`.

24.7 Playbooks aren't showing up in the "Job Template" drop-down

If your playbooks are not showing up in the Job Template drop-down list, here are a few things you can check:

- Make sure that the playbook is valid YML and can be parsed by Ansible.
- Make sure the permissions and ownership of the project path (`/var/lib/awx/projects`) is set up so that the "awx" system user can view the files. You can run this command to change the ownership:

```
chown awx -R /var/lib/awx/projects/
```

24.8 Playbook stays in pending

If you are attempting to run a playbook Job and it stays in the "Pending" state indefinitely, try the following:

- Ensure all supervisor services are running via `supervisorctl status`.
- Check to ensure that the `/var/` partition has more than 1 GB of space available. Jobs will not complete with insufficient space on the `/var/` partition.
- Run `ansible-tower-service restart` on the Tower server.

If you continue to have problems, run `sosreport` as root on the Tower server, then file a [support request](#) with the result.

24.9 Cancel a Tower job

When issuing a `cancel` request on a currently running Tower job, Tower issues a `SIGINT` to the `ansible-playbook` process. While this causes Ansible to stop dispatching new tasks and exit, in many cases, module tasks that were already dispatched to remote hosts will run to completion. This behavior is similar to pressing `Ctrl-C` during a command-line Ansible run.

With respect to software dependencies, if a running job is canceled, the job is essentially removed but the dependencies will remain.

24.10 Reusing an external database causes installations to fail

Instances have been reported where reusing the external DB during subsequent installation of nodes causes installation failures.

For example, say that you performed a clustered installation. Next, say that you needed to do this again and performed a second clustered installation reusing the same external database, only this subsequent installation failed.


When setting up an external database which has been used in a prior installation, the database used for the clustered node must be manually cleared before any additional installations can succeed.

24.10.1 Bubblewrap functionality and variables

The bubblewrap functionality in Ansible Tower limits which directories on the Tower file system are available for playbooks to see and use during playbook runs. You may find that you need to customize your bubblewrap settings in some cases. To fine tune your usage of bubblewrap, there are certain variables that can be set.

To disable or enable bubblewrap support for running jobs (playbook runs only), ensure you are logged in as the Admin user:



1. Click the Settings () icon from the left navigation bar.
2. Click the “Jobs” tab.
3. Scroll down until you see “Enable Job Isolation” and change the toggle button selection to **OFF** to disable bubblewrap support or select **ON** to enable it.

By default, the Tower will use the system’s `tmp` directory (`/tmp` by default) as its staging area. This can be changed in the **Job Isolation Execution Path** field of the Configure tower screen, or by updating the following entry in the settings file:

```
AWX_PROOT_BASE_PATH = "/opt/tmp"
```

If there is other information on the system that is sensitive and should be hidden, you can specify those in the Configure Tower screen in the **Paths to Hide to Isolated Jobs** or by updating the following entry in the settings file:

```
AWX_PROOT_HIDE_PATHS = ['/list/of/', 'paths']
```

If there are any directories that should specifically be exposed, you can specify those in the Configure Tower screen in the **Paths to Expose to Isolated Jobs** or by updating the following entry in the settings file:


```
AWX_PROOT_SHOW_PATHS = ['/list/of/', '/paths']
```

Note: The primary file you may want to add to `AWX_PROOT_SHOW_PATHS` is `/var/lib/awx/.ssh`, if your playbooks need to use keys or settings defined there.

If you made changes in the settings file, be sure to restart services with the `ansible-tower-service restart` command after your changes have been saved.

24.11 Private EC2 VPC Instances in Tower Inventory

By default, Tower only shows instances in a VPC that have an Elastic IP (EIP) associated with them. To see all of your VPC instances, perform the following steps:

1. In the Tower interface, select your inventory.
2. Click on the group that has the Source set to AWS, and click on the Source tab.
3. In the `Source Variables` box, enter:

```
vpc_destination_variable: private_ip_address
```

Next, save and then trigger an update of the group. Once this is done, you should be able to see all of your VPC instances.

Note: Tower must be running inside the VPC with access to those instances if you want to configure them.

24.12 Troubleshooting “Error: provided hosts list is empty”

If you receive the message “Skipping: No Hosts Matched” when you are trying to run a playbook through Tower, here are a few things to check:

- Make sure that your hosts declaration line in your playbook matches the name of your group/host in inventory exactly (these are case sensitive).
- If it does match and you are using Ansible Core 2.0 or later, check your group names for spaces and modify them to use underscores or no spaces to ensure that the groups can be recognized.
- Make sure that if you have specified a Limit in the Job Template that it is a valid limit value and still matches something in your inventory. The Limit field takes a pattern argument, described here: http://docs.ansible.com/intro_patterns.html

Please file a support ticket if you still run into issues after checking these options.

TOWER TIPS AND TRICKS

25.1 Using the Tower CLI Tool

Ansible Tower has a full-featured command line interface. It communicates with Tower via Tower's REST API. You can install it from any machine with access to your Tower machine, or on Tower itself.

Installation can be done using the `pip` command:

```
pip install ansible-tower-cli
```

Refer to *Introduction to tower-cli* and <https://github.com/ansible/tower-cli/blob/master/README.rst> for configuration and usage instructions.

25.2 Launching a Job Template via the API

Ansible Tower makes it simple to launch a job based on a Job Template from Tower's API or by using the `tower-cli` command line tool.

Launching a Job Template also:

- Creates a Job Record
- Gives that Job Record all of the attributes on the Job Template, combined with certain data you can give in this launch endpoint ("runtime" data)
- Runs Ansible with the combined data from the JT and runtime data

Runtime data takes precedence over the Job Template data, contingent on the `ask__on_launch` field on the job template being set to `True`. For example, a runtime credential is only accepted if the Job Template has `ask_credential_on_launch` set to `True`.

Launching from Job Templates via the API follows the following workflow:

- GET `https://your.tower.server/api/v2/job_templates/<your job template id>/launch/`. The response will contain data such as `job_template_data` and `defaults` which give information about the job template.
- Inspect returned data for runtime data that is needed to launch. Inspecting the `OPTIONS` of the launch endpoint may also help deduce what `POST` fields are allowed.

Warning: Providing certain runtime credentials could introduce the need for a password not listed in `passwords_needed_to_start`.

- passwords_needed_to_start: List of passwords needed
 - credential_needed_to_start: Boolean
 - inventory_needed_to_start: Boolean
 - variables_needed_to_start: List of fields that need to be passed inside of the `extra_vars` dictionary
- Inspect returned data for optionally allowed runtime data that the user should be asked for.
 - ask_variables_on_launch: Boolean specifying whether to prompt the user for additional variables to pass to Ansible inside of `extra_vars`
 - ask_tags_on_launch: Boolean specifying whether to prompt the user for `job_tags` on launch (allow allows use of `skip_tags` for convenience)
 - ask_job_type_on_launch: Boolean specifying whether to prompt the user for `job_type` on launch
 - ask_limit_on_launch: Boolean specifying whether to prompt the user for `limit` on launch
 - ask_inventory_on_launch: Boolean specifying whether to prompt the user for the related field `inventory` on launch
 - ask_credential_on_launch: Boolean specifying whether to prompt the user for the related field `credential` on launch
 - survey_enabled: Boolean specifying whether to prompt the user for additional `extra_vars`, following the job template's `survey_spec` Q&A format
 - POST `https://your.tower.server/api/v2/job_templates/<your job template id>/launch/` with any required data gathered during the previous step(s). The variables that can be passed in the request data for this action include the following.
 - extra_vars: A string that represents a JSON or YAML formatted dictionary (with escaped parentheses) which includes variables given by the user, including answers to survey questions
 - job_tags: A string that represents a comma-separated list of tags in the playbook to run
 - limit: A string that represents a comma-separated list of hosts or groups to operate on
 - inventory: A integer value for the foreign key of an inventory to use in this job run
 - credential: A integer value for the foreign key of a credential to use in this job run

The POST will return data about the job and information about whether the runtime data was accepted. The job id is given in the `job` field to maintain compatibility with tools written before 3.0. The response will look similar to:

```
{
  "ignored_fields": {
    "credential": 2,
    "job_tags": "setup,teardown"
  }
  "id": 4,
  ...more data about the job...
  "job": 4,
}
```

In this example, values for `credential` and `job_tags` were given while the job template `ask_credential_on_launch` and `ask_tags_on_launch` were `False`. These were rejected because the job template author did not allow using runtime values for them.

You can see details about the job in this response. To get an updated status, you will need to do a GET request to the job page, `/jobs/4`, or follow the `url` link in the response. You can also find related links to cancel, relaunch, and so fourth.

Note: When querying a job on a non-execution node, an error message, `stdout capture is missing` displays for the `result_stdout` field and on the related stdout page. In order to generate the stdout, use the `format=txt_download` query parameter for the related stdout page. This generates the stdout file and any refreshes to either the job or the related std will display the job output.

Note: You cannot assign a new inventory at the time of launch to a scan job. Scan jobs must be tied to a fixed inventory.

Note: You cannot change the Job Type at the time of launch to or from the type of “scan”. The `ask_job_type_on_launch` option only enables you to toggle “run” versus “check” at launch time.

25.3 tower-cli Job Template Launching

From the Tower command line, you can use `tower-cli` as a method of launching your Job Templates.

For help with `tower-cli` launch, use:

```
tower-cli job launch --help.
```

For launching from a job template, invoke `tower-cli` in a way similar to:

For an example of how to use the API, you can also add the `-v` flag here:

```
tower-cli job launch --job-template=4 -v
```

25.4 Changing the Tower Admin Password

During the installation process, you are prompted to enter an administrator password which is used for the `admin` superuser/first user created in Tower. If you log into the instance via SSH, it will tell you the default admin password in the prompt. If you need to change this password at any point, run the following command as root on the Tower server:

```
awx-manage changepassword admin
```

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

25.5 Creating a Tower Admin from the commandline

Once in a while you may find it helpful to create an admin (superuser) account from the commandline. To create an admin, run the following command as root on the Tower server and enter in the admin information as prompted:

```
awx-manage createsuperuser
```

25.6 Setting up a jump host to use with Tower

Credentials supplied by Tower will not flow to the jump host via ProxyCommand. They are only used for the end-node once the tunneled connection is set up.

To make this work, configure a fixed user/keyfile in the AWX user's SSH config in the ProxyCommand definition that sets up the connection through the jump host. For example:

```
Host tampa
Hostname 10.100.100.11
IdentityFile [privatekeyfile]

Host 10.100..
Proxycommand ssh -W [jumphostuser]@%h:%p tampa
```

Note: You must disable PRoot by default if you need to use a jump host. You can disable PRoot through the Configure Tower user interface by setting the **Enable Job Isolation** toggle to **OFF** from the Jobs tab:

SETTINGS / EDIT CONFIGURATION

CONFIGURE Tower

AUTHENTICATION | **JOBS** | SYSTEM | USER INTERFACE

ANSIBLE MODULES ALLOWED FOR AD HOC JOBS REVERT

- command
- shell
- yum
- apt
- apt_key
- apt_repository
- apt_rpm
- service
- group
- user
- mount
- ping
- selinux
- setup
- win_ping
- win_service
- win_updates
- win_group
- win_user

PATHS TO EXPOSE TO ISOLATED JOBS REVERT

ANSIBLE CALLBACK PLUGINS REVERT

PATHS TO HIDE FROM ISOLATED JOBS REVERT

*** ENABLE JOB ISOLATION** REVERT

OFF

DEFAULT PROJECT UPDATE TIMEOUT REVERT

0

PER-HOST ANSIBLE FACT CACHE TIMEOUT REVERT

0

EXTRA ENVIRONMENT VARIABLES REVERT

1

You can also add a jump host to your Tower instance through Inventory variables. These variables can be set at either the inventory, group, or host level. To add this, navigate to your inventory and in the `variables` field of whichever level you choose, add the following variables:

```
ansible_user: <user_name>
ansible_connection: ssh
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q <user_name>@<jump_server_
↵name>"'
```

25.7 View Ansible outputs for JSON commands when using Tower

When working with Ansible Tower, you can use the API to obtain the Ansible outputs for commands in JSON format.

To view the Ansible outputs, browse to:

```
https://<tower server name>/api/v2/jobs/<job_id>/job_events/
```

25.8 Locate and configure the Ansible configuration file

While Ansible does not require a configuration file, OS packages often include a default one in `/etc/ansible/ansible.cfg` for possible customization. In order to use a custom `ansible.cfg` file, place it at the root of your project. Ansible Tower runs `ansible-playbook` from the root of the project directory, where it will then find the custom `ansible.cfg` file. An `ansible.cfg` anywhere else in the project will be ignored.

To learn which values you can use in this file, refer to the [configuration file on github](#).

Using the defaults are acceptable for starting out, but know that you can configure the default module path or connection type here, as well as other things.

Tower overrides some `ansible.cfg` options. For example, Tower stores the SSH ControlMaster sockets, the SSH agent socket, and any other per-job run items in a per-job temporary directory, secured by multi-tenancy access control restrictions via PRoot.

25.9 View a listing of all `ansible_` variables

Ansible by default gathers “facts” about the machines under its management, accessible in Playbooks and in templates. To view all facts available about a machine, run the `setup` module as an ad hoc action:

```
ansible -m setup hostname
```

This prints out a dictionary of all facts available for that particular host. For more information, refer to: https://docs.ansible.com/ansible/playbooks_variables.html#information-discovered-from-systems-facts

25.10 Using `virtualenv` with Ansible Tower

Ansible Tower 3.0 and later uses `virtualenv`. `Virtualenv` creates isolated Python environments to avoid problems caused by conflicting dependencies and differing versions. `Virtualenv` works by simply creating a folder which contains all of the necessary executables and dependencies for a specific version of Python. Ansible Tower creates two `virtualenv`s during installation—one is used to run Tower, while the other is used to run Ansible. This allows Tower to run in a stable environment, while allowing you to add or update modules to your Ansible Python environment as necessary to run your playbooks. For more information on `virtualenv`, see the Python Guide to [Virtual Environments](#) and the *Python virtualenv* project itself.

By default, the `virtualenv` is located at `/var/lib/awx/venv/ansible` on the file system.

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

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

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

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

25.10.1 Preparing a new custom virtualenv

Tower allows a different virtualenv to be specified and used on Job Template runs. To choose a custom virtualenv, first create one under `/var/lib/awx/venv/`:

```
$ sudo virtualenv /var/lib/awx/venv/my-custom-venv
```

Your newly created virtualenv needs a few base dependencies to properly run playbooks (Tower uses memcached as a placeholder for playbook artifacts):

```
$ sudo /var/lib/awx/venv/my-custom-venv/bin/pip install python-memcached psutil
```

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

```
$ sudo /var/lib/awx/venv/my-custom-venv/bin/pip install -U "ansible == X.Y.Z"
```

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

```
$ sudo /var/lib/awx/venv/my-custom-venv/bin/pip install -U python-digitalocean
```

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

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

In a clustered Tower installation, you need to ensure that the same custom virtualenv exists on every local file system at `/var/lib/awx/venv/`. Custom virtualenvs are supported on isolated instances. If you are using a custom virtual environment, it needs to also be copied or replicated on any isolated node you would be using, not just on the Tower node.

25.10.2 Assigning custom virtualenvs

Once you have created a custom virtualenv, you can assign it at the Organization, Project, or Job Template level:

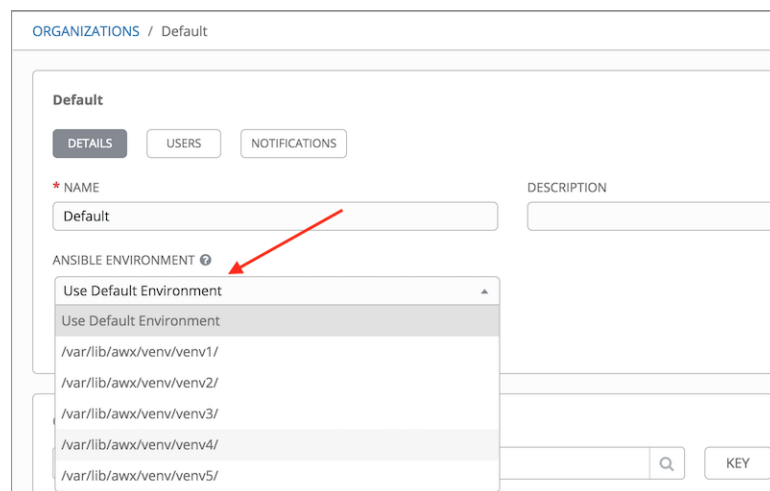
```
PATCH https://awx-host.example.org/api/v2/organizations/N/
PATCH https://awx-host.example.org/api/v2/projects/N/
PATCH https://awx-host.example.org/api/v2/job_templates/N/

Content-Type: application/json
{
    'custom_virtualenv': '/var/lib/awx/venv/my-custom-venv'
}
```

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

```
{
  "custom_virtualenvs": [
    "/var/lib/awx/venv/my-custom-venv",
    "/var/lib/awx/venv/my-other-custom-venv",
  ],
  ...
}
```

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



25.11 Configuring the `towerhost` hostname for notifications

In `/etc/tower/settings.py`, you can modify `TOWER_URL_BASE='https://tower.example.com'` to change the notification hostname, replacing `https://tower.example.com` with your preferred hostname. You must restart Tower services after saving your changes with `ansible-tower-service restart`.

Refreshing your Tower license also changes the notification hostname. New installations of Ansible Tower 3.0 should not have to set the hostname for notifications.

25.12 Launching Jobs with curl

Note: Tower now offers a full-featured command line interface called `tower-cli` which may be of interest to you if you are considering using `curl`.

This method works with Tower versions 2.1.x and newer.

Launching jobs with the Tower API is simple. Here are some easy to follow examples using the `curl` tool.

Assuming that your Job Template ID is '1', your Tower IP is 192.168.42.100, and that `admin` and `awxsecret` are valid login credentials, you can create a new job this way:


```
curl -f -k -H 'Content-Type: application/json' -XPOST \  
  --user admin:awssecret \  
  http://192.168.42.100/api/v2/job_templates/1/launch/
```

This returns a JSON object that you can parse and use to extract the ‘id’ field, which is the ID of the newly created job.

You can also pass extra variables to the Job Template call, such as is shown in the following example:

```
curl -f -k -H 'Content-Type: application/json' -XPOST \  
  -d '{"extra_vars": "{\\"foo\\": \\"bar\\"}"}' \  
  --user admin:awssecret http://192.168.42.100/api/v2/job_templates/1/launch/
```

You can view the live API documentation by logging into <http://192.168.42.100/api/> and browsing around to the various objects available.

Note: The `extra_vars` parameter needs to be a string which contains JSON, not just a JSON dictionary, as you might expect. Use caution when escaping the quotes, etc.

25.13 Dynamic Inventory and private IP addresses

By default, Tower only shows instances in a VPC that have an Elastic IP (EIP) address associated with them. To view all of your VPC instances, perform the following steps:

- In the Tower interface, select your inventory.
- Click on the group that has the Source set to AWS, and click on the Source tab.
- In the “Source Variables” box, enter: `vpc_destination_variable: private_ip_address`

Save and trigger an update of the group. You should now be able to see all of your VPC instances.

Note: Tower must be running inside the VPC with access to those instances in order to usefully configure them.

25.14 Filtering instances returned by the dynamic inventory sources in Tower

By default, the dynamic inventory sources in Tower (AWS, Rackspace, etc) return all instances available to the cloud credentials being used. They are automatically joined into groups based on various attributes. For example, AWS instances are grouped by region, by tag name and value, by security groups, etc. To target specific instances in your environment, write your playbooks so that they target the generated group names. For example:

```
---  
- hosts: tag_Name_webserver  
  tasks:  
  ...
```

You can also use the `Limit` field in the Job Template settings to limit a playbook run to a certain group, groups, hosts, or a combination thereof. The syntax is the same as the `--limit` parameter on the `ansible-playbook` command line.

You may also create your own groups by copying the auto-generated groups into your custom groups. Make sure that the `Overwrite` option is disabled on your dynamic inventory source, otherwise subsequent synchronization operations will delete and replace your custom groups.

25.15 Using an unreleased module from Ansible source with Tower

If there is a feature that is available in the latest Ansible core branch that you would like to leverage with your Tower system, making use of it in Tower is fairly simple.

First, determine which is the updated module you want to use from the available Ansible Core Modules or Ansible Extra Modules GitHub repositories.

Next, create a new directory, at the same directory level of your Ansible source playbooks, named `/library`.

Once this is created, copy the module you want to use and drop it into the `/library` directory—it will be consumed first over your system modules and can be removed once you have updated the the stable version via your normal package manager.

25.16 Using callback plugins with Tower

Ansible has a flexible method of handling actions during playbook runs, called callback plugins. You can use these plugins with Tower to do things like notify services upon playbook runs or failures, send emails after every playbook run, etc. For official documentation on the callback plugin architecture, refer to: http://docs.ansible.com/developing_plugins.html#callbacks

Note: Ansible Tower does not support the `stdout` callback plugin because Ansible only allows one, and it is already being used by Ansible Tower for streaming event data.

You may also want to review some example plugins, which should be modified for site-specific purposes, such as those available at: <https://github.com/ansible/ansible/tree/devel/lib/ansible/plugins/callback>

To use these plugins, put the callback plugin `.py` file into a directory called `/callback_plugins` alongside your playbook in your Tower Project.

To make callbacks apply to every playbook, independent of any projects, put the plugins `.py` file in one of the following directories (depending on your particular Linux distribution and method of Ansible installation):


```
* /usr/lib/python2.7/ansible/callback_plugins
* /usr/local/lib/python2.7/dist-packages/ansible/callback_plugins
* /usr/lib/python2.6/site-packages/ansible/callback_plugins
```

Note: To have most callbacks shipped with Ansible applied globally, you must add them to the `callback_whitelist` section of your `ansible.cfg`.

25.17 Connecting to Windows with winrm

By default Tower attempts to `ssh` to hosts. You must add the `winrm` connection info to the group variables to which the Windows hosts belong. To get started, edit the Windows group in which the hosts reside and place the variables in the source/edit screen for the group.

To add `winrm` connection info:

Edit the properties for the selected group by clicking on the  button to the right of the group name that contains the Windows servers. In the “variables” section, add your connection information as such: `ansible_connection: winrm`

Once done, save your edits. If Ansible was previously attempting an SSH connection and failed, you should re-run the job template.

25.18 Importing existing inventory files and host/group vars into Tower

To import an existing static inventory and the accompanying host and group vars into Tower, your inventory should be in a structure that looks similar to the following:

```
inventory/
|-- group_vars
|   `-- mygroup
|-- host_vars
|   `-- myhost
`-- hosts
```

To import these hosts and vars, run the `awx-manage` command:

```
awx-manage inventory_import --source=inventory/ \
  --inventory-name="My Tower Inventory"
```

If you only have a single flat file of inventory, a file called `ansible-hosts`, for example, import it like the following:

```
awx-manage inventory_import --source=./ansible-hosts \
  --inventory-name="My Tower Inventory"
```

In case of conflicts or to overwrite an inventory named “My Tower Inventory”, run:

```
awx-manage inventory_import --source=inventory/ \
  --inventory-name="My Tower Inventory" \
  --overwrite --overwrite-vars
```

If you receive an error, such as:

```
ValueError: need more than 1 value to unpack
```

Create a directory to hold the hosts file, as well as the `group_vars`:

```
mkdir -p inventory-directory/group_vars
```

Then, for each of the groups that have `:vars` listed, create a file called `inventory-directory/group_vars/<groupname>` and format the variables in YAML format.

Once broken out, the importer will handle the conversion correctly.

INTRODUCTION TO TOWER-CLI

tower-cli is a command line tool for Ansible Tower. It allows Tower commands to be easily run from the UNIX command line. It can also be used as a client library for other python apps, or as a reference for others developing API interactions with Tower's REST API.

Note: The `tower-cli` software is an open source project currently under development and, until a complete implementation occurs, only implements a subset of Tower's features.

26.1 License

While Tower is commercially licensed software, `tower-cli` is an open source project. Specifically, this project is licensed under the Apache 2.0 license. Pull requests, contributions, and tickets filed in GitHub are warmly welcomed.

26.2 Capabilities

`tower-cli` sends commands to the Tower API. It is capable of retrieving, creating, modifying, and deleting most objects within Tower.

A few potential uses include:

- Launching playbook runs (for instance, from Jenkins, TeamCity, Bamboo, etc)
- Checking on job statuses
- Rapidly creating objects like organizations, users, teams, and more

26.3 Installation

`tower-cli` is available as a package on PyPI.

The preferred way to install is through `pip`:

```
$ pip install ansible-tower-cli
```

The main branch of this project may also be consumed directly from source.

For more information on `tower-cli`, refer to the project page at: <https://github.com/ansible/tower-cli/>

26.4 Configuration

`tower-cli` can edit its own configuration or users can directly edit the configuration file, allowing configuration to be set in multiple ways.

26.4.1 Set configuration with `tower-cli config`

The preferred way to set configuration is with the `tower-cli config` command.

```
$ tower-cli config key value
```

By issuing the `tower-cli config` command without arguments, you can view a full list of configuration options and where they are set. The default behavior allows environment variables to override your `tower-cli.cfg` settings, but they will not override configuration values that are passed in on the command line at runtime. The available environment variables and their corresponding Tower configuration keys are as follows:

- `TOWER_COLOR`: color
- `TOWER_FORMAT`: format
- `TOWER_HOST`: host
- `TOWER_PASSWORD`: password
- `TOWER_USERNAME`: username
- `TOWER_VERIFY_SSL`: `verify_ssl`
- `TOWER_VERBOSE`: verbose
- `TOWER_DESCRIPTION_ON`: `description_on`
- `TOWER_CERTIFICATE`: certificate

You will generally need to set at least three configuration options—host, username, and password—which correspond to the location of your Ansible Tower instance and your credentials to authenticate to Tower.

```
$ tower-cli config host tower.example.com
$ tower-cli config username leeroyjenkins
$ tower-cli config password myPassw0rd
```

26.4.2 Write to the config files directly

The configuration file can also be edited directly. A configuration file is a simple file with keys and values, separated by `:` or `=`:

```
host: tower.example.com
username: admin
password: p4ssw0rd
```

26.4.3 File Locations

The order of precedence for configuration file locations is as follows, from least to greatest:

- internal defaults
- `/etc/tower/tower_cli.cfg` (written using `tower-cli config --global`)
- `~/.tower_cli.cfg` (written using `tower-cli config`)
- run-time parameters

26.4.4 Usage

`tower-cli` invocation generally follows this format:

```
$ tower-cli {resource} {action} ...
```

The **resource** is a type of object within Tower (a noun), such as `user`, `organization`, `job_template`, etc.; resource names are always singular in Tower CLI (use `tower-cli user`, never `tower-cli users`).

The **action** is the thing you want to do (a verb). Most `tower-cli` resources have the following actions—`get`, `list`, `create`, `modify`, and `delete`—and have options corresponding to fields on the object in Tower.

Examples of actions and resources include (but are not limited to):

User Actions

```
# List all users.
$ tower-cli user list

# List all non-superusers
$ tower-cli user list --is-superuser=false

# Get the user with the ID of 42.
$ tower-cli user get 42

# Get the user with the given username.
$ tower-cli user get --username=guido

# Create a new user.
$ tower-cli user create --username=guido --first-name=Guido \
    --last-name="Van Rossum" --email=guido@python.org

# Modify an existing user.
# This would modify the first name of the user with the ID of "42" to "Guido".
$ tower-cli user modify 42 --first-name=Guido

# Modify an existing user, lookup by username.
# This would use "username" as the lookup, and modify the first name.
# Which fields are used as lookups vary by resource, but are generally
# the resource's name.
$ tower-cli user modify --username=guido --first-name=Guido

# Delete a user.
$ tower-cli user delete 42
```

Job Actions

```
# Launch a job.
$ tower-cli job launch --job-template=144

# Monitor a job.
$ tower-cli job monitor 95
```

Group Actions

```
# Get a list of groups.
$ tower-cli group list

# Sync a group by the groupID.
$ tower-cli group sync $groupID
```

When in doubt, check the help for more options:

```
$ tower-cli # help
$ tower-cli user --help # resource specific help
$ tower-cli user create --help # command specific help
```

Workflow Actions

Starting with Tower 3.1.0 and Tower-CLI 3.1.0, workflow networks can be managed from Tower-CLI either by normal CRUD actions or by using a YAML file that defines the workflow network.

```
# Print out the schema for a workflow
$ tower-cli workflow schema workflow_name

# Create the network defined in file "schema.yml"
$ tower-cli workflow schema workflow_name @schema.yml
```

The following is an example of what a schema might look like.

```
- job_template: Hello world
  failure:
    - inventory_source: AWS servers (AWS servers - 42)
  success:
    - project: Ansible Examples
      always:
        - job_template: Echo variable
          success:
            - job_template: Scan localhost
```

For more details, see the tower-cli workflow doc at <https://github.com/ansible/tower-cli/blob/master/docs/WORKFLOWS.md>

26.4.5 Specify extra variables

There are a number of ways to pass extra variables to the Tower server when launching a job:

- Pass data in a file using the flag `--extra-vars="@filename.yml"`
- Include yaml data at runtime with the flag `--extra-vars="var: value"`
- A command line editor automatically pops up when the job template is marked to prompt on launch
- If the job template has extra variables, these are not over-ridden

These methods can also be combined. For instance, if you give the flag multiple times on the command line, specifying a file in addition to manually giving extra variables, these two sources are combined and sent to the Tower server.

```
# Launch a job with extra variables from filename.yml, and also a=5
$ tower-cli job launch --job-template=1 --extra-vars="a=5 b=3" \
                      --extra-vars="@filename.yml"

# Create a job template with that same set of extra variables
$ tower-cli job_template create --name=test_job_template --project=1 \
                               --inventory=1 --playbook=helloworld.yml \
                               --machine-credential=1 --extra-vars="a=5 b=3" \
                               --extra-vars="@filename.yml"
```

You may not combine multiple sources when modifying a job template. Whitespace can be used in strings like `--extra-vars="a='white space'"`, and list-valued parameters can be sent as JSON or YAML, but not key=value pairs. For instance, `--extra-vars="a: [1, 2, 3, 4, 5]"` sends the parameter "a" with that list as its value.

Note: Additional strict `extra_vars` validation was added in Ansible Tower 3.0.0. `extra_vars` passed to the job launch API are only honored if one of the following is true:

- They correspond to variables in an enabled survey
 - `ask_variables_on_launch` is set to True
-

26.4.6 SSL warnings

By default, `tower-cli` raises an error if the SSL certificate of the Tower server cannot be verified. To allow unverified SSL connections, set the config variable, `verify_ssl = false`. To allow it for a single command to override `verify_ssl` if set to true, add the `--insecure` flag:

```
# Disable insecure connection warnings permanently
$ tower-cli config verify_ssl false

# Disable insecure connection warnings for just this command
$ tower-cli job_template list --insecure
```

USABILITY ANALYTICS AND DATA COLLECTION

In Ansible Tower version 2.4.0, a behind the scenes functionality was added to Tower to collect usability data. This software was introduced 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.

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.

1. Select the desired level of data collection from the 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.

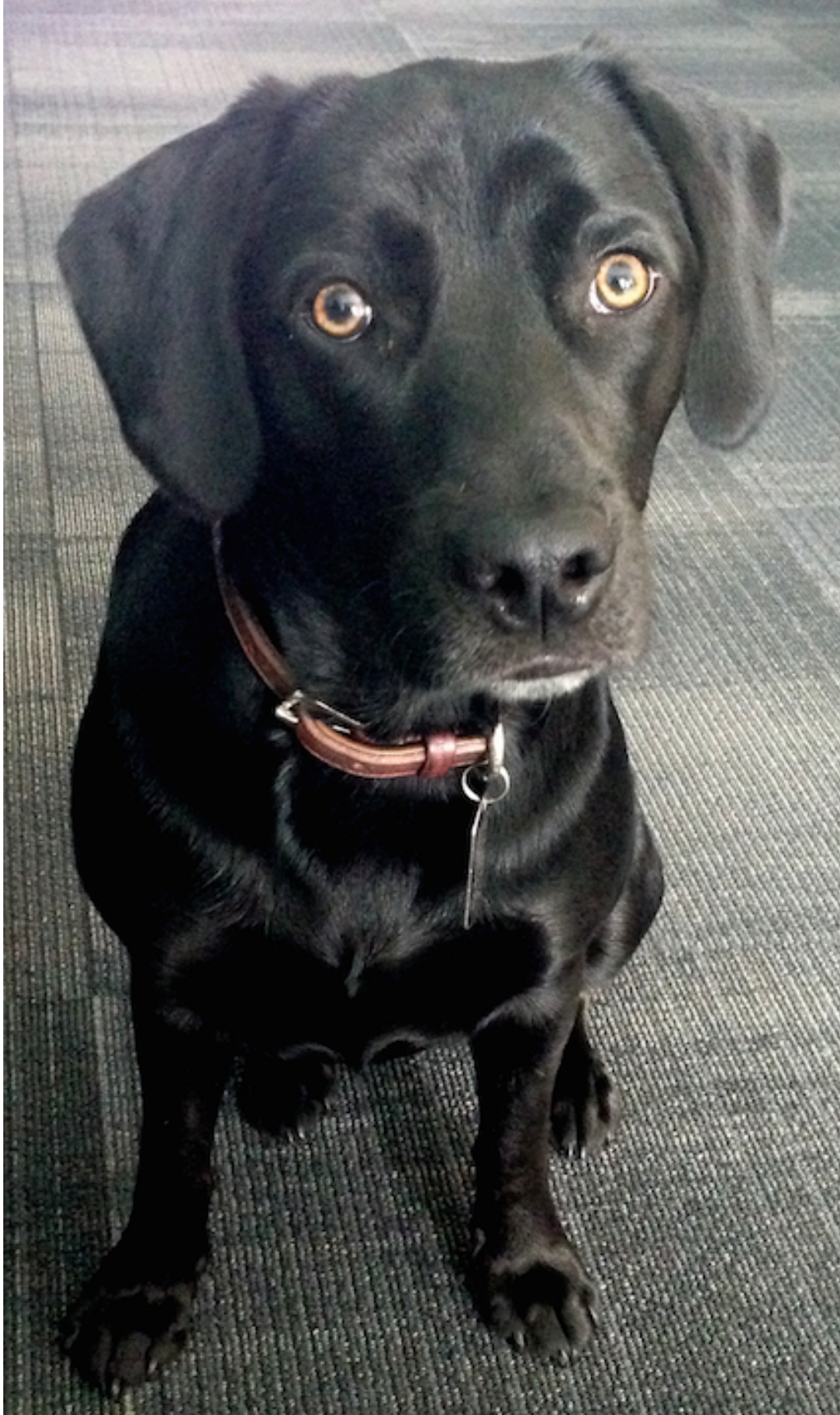
Note: This setting was previously configured via PENDO, which is no longer supported.

POSTFACE

Through community efforts, rigorous testing, dedicated engineers, enterprising sales teams, imaginative marketing, and outstanding professional services and support teams, the growing but always impressive group of individuals that make the Ansible-branded products can feel proud in saying:

Ansible, Ansible Tower, Tower CLI, and Ansible Galaxy are all, as Doge would say, “much approved.”¹

¹ <http://knowyourmeme.com/memes/doge>



Josie Tested - Doge Approved.

INDEX

- genindex

COPYRIGHT © 2019 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.

Rackspace trademarks, service marks, logos and domain names are either common-law trademarks/service marks or registered trademarks/service marks of Rackspace US, Inc., or its subsidiaries, and are protected by trademark and other laws in the United States and other countries.

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

- : API
 - launching a Job Template, 117
- A**
 - Active Directory (*AD*)
 - Kerberos, 104
 - activity stream cleanup management
 - job, 15
 - admin creation
 - commandline, 120
 - tips, 120
 - admin password
 - changing password, 119
 - admin password change
 - tips, 119
 - admin utility script, 4
 - analytics collection, 57, 133
 - Ansible configuration file, 121
 - Ansible modules, unreleased
 - tips, 125
 - Ansible output for JSON commands, 121
 - Ansible, executing in a virtual
 - environment, 121
 - ansible.cfg, 121
 - tips, 121
 - ansible_variables, viewing all
 - tips, 121
 - ansible-tower script replacement, 4
 - API
 - configure, 23
 - assignment
 - credentials, 10
 - AUTH_BASIC_ENABLED
 - session limits, 107
 - authentication, 53, 62, 72, 82
 - Azure AD, 82
 - basic auth, 73
 - configuration, 53
 - Github OAuth2, 74
 - Github Org, 76
 - Github Team, 78

- Google OAuth2, 73
 - LDAP, 94, 100
 - LDAP mapping, 100
 - LDAP team mapping, 100
 - organization mapping, 79, 100
 - RADIUS Authentication Settings, 92
 - SAML Service Provider, 84
 - TACACS+ Authentication Settings, 92
 - team mapping, 79, 100
- authentication expiring, 103
- authentication timeout
 - changing the default, 103
 - troubleshooting, 103
- authentication token, 103
- awx-manage
 - change password, 119
 - cluster management, 50
 - inventory import, 49
 - session management, 50
 - super user creation, 120
 - token management, 50
- awx-manage, data cleanup, 50
- Azure AD
 - authentication, 82
- B**
 - backups, 108
 - considerations, 109
 - playbooks, 108
 - basic auth
 - authentication, 73
 - best practices, 117
 - bubblewrap
 - functionality, 61, 115
 - troubleshooting, 61, 115
 - variables, 61, 115
- C**
 - callback plugins
 - tips, 125
 - Centos
 - clustering, 20

- change password
 - awx-manage, 119
 - changing password
 - admin password, 119
 - changing the default
 - authentication timeout, 103
 - cleaning old data, 15
 - cleanup activity stream
 - management jobs, 15
 - cleanup job history
 - management jobs, 19
 - cluster
 - configure, 23
 - deprovisioning, 31
 - clustering
 - backup, 109
 - Centos, 20
 - instance group policies, 24
 - isolated instance groups, 25
 - operating systems, 20
 - pinning, 25
 - Postgresql, 21
 - RabbitMQ, 21
 - redundancy, 20
 - restore, 109
 - RHEL, 20
 - setup considerations, 21
 - SSH authentication, 27
 - Ubuntu, 20
 - command line interface
 - tips, 117
 - Tower CLI, 117
 - commandline
 - admin creation, 120
 - components
 - licenses, 3
 - configuration
 - authentication, 53
 - custom login message, 56
 - custom logo, 56
 - data collection, 56
 - jobs, 54
 - OpenShift, 32
 - system, 55
 - UI, 56
 - configuration file configuration
 - tips, 121
 - configuration file location
 - tips, 121
 - configure
 - API, 23
 - cluster, 23
 - configure Tower, 53
 - credentials, 10
 - assignment, 10
 - multi, 10
 - curl
 - tips, 123
 - custom
 - login message, 56
 - logo, 56, 111
 - custom inventory scripts, 5
 - custom login message
 - configuration, 56
 - custom logo, 111
 - configuration, 56
- ## D
- data collection, 57, 133
 - configuration, 56
 - DEB files
 - licenses, 3
 - deployment
 - OpenShift, 32
 - dynamic inventory and instance
 - filtering
 - tips, 124
 - dynamic inventory and private IPs
 - tips, 124
- ## E
- EC2
 - VPC instances, 116
 - EC2 VPC instances
 - tips, 124
 - troubleshooting, 116
 - Elastic stack
 - logging, 42
 - ELK stack
 - logging, 42
 - enterprise authentication, 53, 82
 - error logs
 - troubleshooting, 113
 - evaluation, 2
 - external database
 - installation failure, 115
- ## F
- features, 1
 - filtering instances
 - tips, 124
 - functionality
 - bubblewrap, 61, 115
- ## G
- general help
 - troubleshooting, 113
 - Github OAuth2

- authentication, 74
- Github Org
 - authentication, 76
- Github Team
 - authentication, 78
- Google OAuth2
 - authentication, 73
- groups
 - deprovisioning, 31

H

- help, 113, 117
- host connections
 - troubleshooting, 113, 114
- host/group vars import
 - tips, 126
- hostname configuration
 - notifications, 123
- hosts list
 - troubleshooting, 116
- hosts lists (*empty*), 116

I

- import
 - license, 58
- importing host/group vars
 - importing inventory, 126
- importing inventory
 - importing host/group vars, 126
- init script replacement, 4
- installation bundle
 - licenses, 3
- installation failure
 - external database, 115
- installation wizard
 - playbook backup/restore arguments, 108
- instance filtering
 - tips, 124
- instance group policies
 - clustering, 24
- instance groups
 - isolated, 25
 - pinning, 25
 - policies, 24
 - redundancy, 20
- inventory file importing, 8
- inventory import
 - tips, 126
- inventory scripts
 - custom, 5, 8
 - writing, 7
- isolated
 - instance groups, 25

- isolated instance groups
 - clustering, 25

J

- job cancellation
 - troubleshooting, 115
- job does not run
 - troubleshooting, 114
- job history cleanup management job, 19
- Job Template drop-down list
 - playbooks are not viewable, 114
- jobs, 54
 - configuration, 54
- JSON commands, Ansible output, 121
- jump host
 - ProxyCommand, 120
 - tips, 120

K

- Kerberos
 - Active Directory (*AD*), 104
 - user authentication, 104

L

- launching a Job Template
 - : API, 117
- LDAP, 94, 100
 - authentication, 94, 100
 - referrals, 100
- LDAP mapping, 100
 - authentication, 100
- LDAP referrals
 - troubleshooting, 100
- LDAP team mapping
 - authentication, 100
- license, 1, 2
 - import, 58
 - nodes, 3
 - trial, 2
 - types, 2
 - UI, 58
- license features, 1
- licenses
 - components, 3
 - DEB files, 3
 - installation bundle, 3
 - RPM files, 3
- live events
 - port changes, 114
 - troubleshooting, 114
- log, 103
- logfiles, 41
- logging, 42
 - Elastic stack, 42

- ELK stack, 42
- loggly, 42
- logstash, 42
- schema, 42
- splunk, 42
- sumologic, 42
- loggly
 - logging, 42
- login message
 - custom, 56
- login timeout, 103
- logo
 - custom, 56, 111
- logstash
 - logging, 42
- M**
- management jobs, 15
 - cleanup activity stream, 15
 - cleanup job history, 19
- modules, using unreleased
 - tips, 125
- multi
 - credentials, 10
- N**
- no proxy
 - proxy support, 38
- notifications
 - hostname configuration, 123
- O**
- OpenShift
 - configuration, 32
 - deployment, 32
- organization mapping, 79, 100
 - authentication, 79, 100
- P**
- pending playbook
 - troubleshooting, 114
- Pendo, 57, 133
- PENDO_TRACKING_STATE, 57, 133
- pinning
 - clustering, 25
 - instance groups, 25
- playbook setup
 - backup/restore arguments, 108
- playbooks are not viewable
 - Job Template drop-down list, 114
- playbooks not appearing
 - troubleshooting, 114
- plugins, callback
 - tips, 125
- policies
 - instance groups, 24
- port changes
 - live events, 114
- postface, 134
- Postgresql
 - clustering, 21
- private IPs with dynamic inventory
 - tips, 124
- PRoot
 - troubleshooting, 114
- proxy support, 38
 - no proxy, 38
 - reverse proxy, 40
- ProxyCommand
 - jump host, 120
 - tips, 120
- R**
- RabbitMQ
 - clustering, 21
- RADIUS Authentication Settings
 - authentication, 92
- rebranding, 111
- redundancy
 - clustering, 20
 - instance groups, 20
- referrals
 - LDAP, 100
- removing old data, 15
- restart Tower, 4
- restorations, 108
 - considerations, 109
 - playbooks, 108
- restore
 - clustering, 109
- reverse proxy, 40
 - proxy support, 40
- RHEL
 - clustering, 20
- RPM files
 - licenses, 3
- S**
- SAML Service Provider
 - authentication, 84
- schema
 - logging, 42
- scripts, admin utility, 4
- session
 - timeout, 103
- session limits, 107
 - AUTH_BASIC_ENABLED, 107
 - SESSIONS_PER_USER, 107

- session.py, 107
 - SESSIONS_PER_USER
 - session limits, 107
 - social authentication, 53, 72
 - splunk
 - logging, 42
 - SSH
 - authentication, 27
 - SSH authentication
 - clustering, 27
 - start Tower, 4
 - stop Tower, 4
 - sumologic
 - logging, 42
 - super user creation
 - awx-manage, 120
 - support, 1, 2
 - system
 - configuration, 55
- ## T
- TACACS+ Authentication Settings
 - authentication, 92
 - team mapping, 79, 100
 - authentication, 79, 100
 - timeout
 - session, 103
 - timeout login, 103
 - tips, 117
 - admin creation, 120
 - admin password change, 119
 - Ansible modules, unreleased, 125
 - ansible.cfg, 121
 - ansible_variables, viewing all, 121
 - callback plugins, 125
 - command line interface, 117
 - configuration file configuration, 121
 - configuration file location, 121
 - curl, 123
 - dynamic inventory and instance filtering, 124
 - dynamic inventory and private IPs, 124
 - EC2 VPC instances, 124
 - filtering instances, 124
 - host/group vars import, 126
 - instance filtering, 124
 - inventory import, 126
 - jump host, 120
 - modules, using unreleased, 125
 - plugins, callback, 125
 - private IPs with dynamic inventory, 124
 - ProxyCommand, 120
 - Tower CLI, 117
 - unreleased modules, 125
 - Windows connection, 126
 - winrm, 126
 - token-based authentication, 62
 - tools
 - tower-cli, 128
 - Tower admin utility script, 4
 - Tower CLI
 - command line interface, 117
 - tips, 117
 - tower-cli, 128
 - capabilities, 128
 - installation, 128
 - tower-manage, 49
 - trial, 2
 - troubleshooting, 113
 - authentication timeout, 103
 - bubblewrap, 61, 115
 - EC2 VPC instances, 116
 - error logs, 113
 - general help, 113
 - host connections, 113, 114
 - hosts list, 116
 - job cancellation, 115
 - job does not run, 114
 - LDAP referrals, 100
 - live events, 114
 - pending playbook, 114
 - playbooks not appearing, 114
 - PRoot, 114
 - websockets, 114
- ## U
- Ubuntu
 - clustering, 20
 - UI
 - configuration, 56
 - license, 58
 - unreleased modules
 - tips, 125
 - updates, 2
 - usability data collection, 57, 133
 - user authentication
 - Kerberos, 104
 - user data tracking, 57, 133
- ## V
- variables
 - bubblewrap, 61, 115
 - virtual environment, 121
 - VPC instances
 - EC2, 116

W

websockets

troubleshooting, 114

Windows connection

tips, 126

winrm

tips, 126