Ansible Tower Administration Guide

Release Ansible Tower 3.2.4

Red Hat, Inc.

Jul 01, 2020

CONTENTS

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

The *Ansible Tower 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 3.2.4.

We Need Feedback!

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

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

Ansible Tower Version 3.2.4; April 27, 2018; https://access.redhat.com/

CHAPTER

ONE

TOWER LICENSING, UPDATES, AND SUPPORT

Ansible Tower by Red Hat ("**Ansible Tower**") is a proprietary software product provided via 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 for paid **Enterprise: Standard** and **Enterprise: Premium** Subscription customers seeking help with the Ansible Tower product.

If you or your company has paid for Ansible Tower, 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*.

If you are experiencing Ansible software issues, you should reach out to the "ansible-devel" mailing list or file an issue on the Github project page at https://github.com/ansible/ansible/issues/.

All of Ansible's community and OSS info can be found here: https://docs.ansible.com/ansible/community.html

1.1.1 Ansible Playbook Support

For customers with a paid Enterprise: Standard or Enterprise: Premium Ansible Tower Subscription, Red Hat offers Ansible Playbook support[1]. Playbook support consists of support for:

- Runtime execution problems for Playbooks run via Tower
- Assistance with Playbook errors and tracebacks
- Limited best practice guidance in Ansible use from the Ansible Experts

Playbook support does not consist of:

- Enhancements and fixes for Ansible modules and the Ansible engine
- Assistance with the creation of Playbooks from anew
- Long-term maintenance of a specific Ansible or Ansible Tower version

[1] Playbook support is available for customers using the current or previous minor release of Ansible. For example, if the current version of Ansible is 2.2, Red Hat provides Ansible Playbook support for versions 2.2 and 2.1. In the event an Ansible Playbook workaround is not available, and an Ansible software correction is required, a version update will be required.

Notes:

1.2 Trial / Evaluation

While a license is required for Ansible Tower to run, there is no fee for managing up to 10 hosts. Additionally, trial licenses are available for exploring Ansible Tower with a larger number of hosts.

- Trial licenses for Ansible Tower are available at: http://ansible.com/license
- To acquire a license for additional Managed Nodes, visit: http://www.ansible.com/pricing/
- Ansible Playbook Support is not included in a trial license or during an evaluation of the Tower Software.

1.3 Subscription Types

Ansible Tower is provided at various levels of support and number of machines as an annual Subscription.

- **Enterprise: Standard (F.K.A. "Enterprise")**
  - 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: Premium (F.K.A. "Premium Enterprise")**
  - 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 hosts 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.

If you have multiple hosts in inventory that have the same name, such as “webserver1”, they will be counted for licensing purposes as a single node. Note that this differs from the ‘Hosts’ count in Tower’s dashboard, which counts hosts in separate inventories separately.

1.5 License Features

The following list of features are available for all new Enterprise: Standard or Enterprise: Premium Subscriptions:

- Workflows (*added in |at| 3.1.0*)
- Clustering in Tower (*added in |at| 3.1.0*)
- Custom re-branding for login (*added in Ansible Tower 2.4.0*)
- SAML and RADIUS Authentication Support (*added in Ansible Tower 2.4.0*)
- Multi-Organization Support
- Activity Streams
- Surveys
- LDAP Support
- Active/Passive Redundancy
- System Tracking (*added in Ansible Tower 2.2.0*)

Enterprise: Standard or Enterprise: Premium license users with versions of Ansible Tower prior to 2.2 must import a new license file to enable System Tracking.

1.6 Tower Component Licenses

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

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

CHAPTER

TWO

STARTING, STOPPING, AND RESTARTING TOWER

Ansible Tower now ships with an *admin utility script*, `ansible-tower-service`, that can `start`, `stop`, and `restart` the full tower infrastructure (including the database and message queue components). 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: 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.

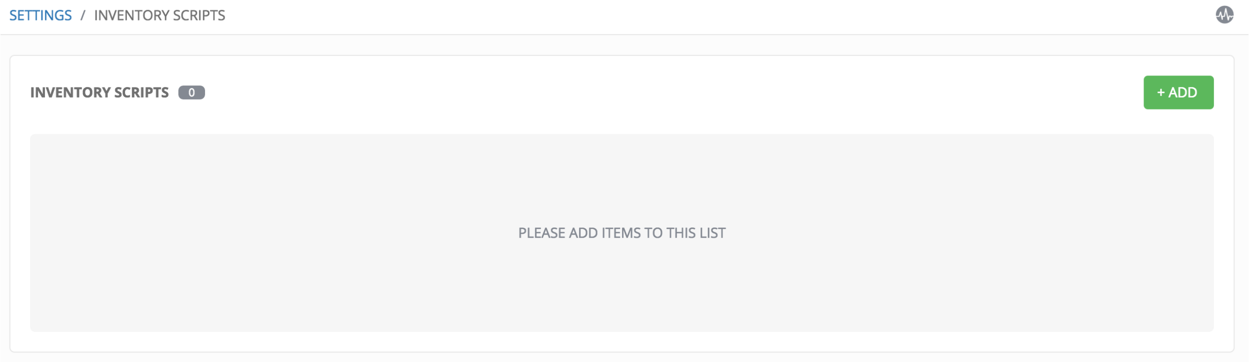
CHAPTER THREE

CUSTOM INVENTORY SCRIPTS

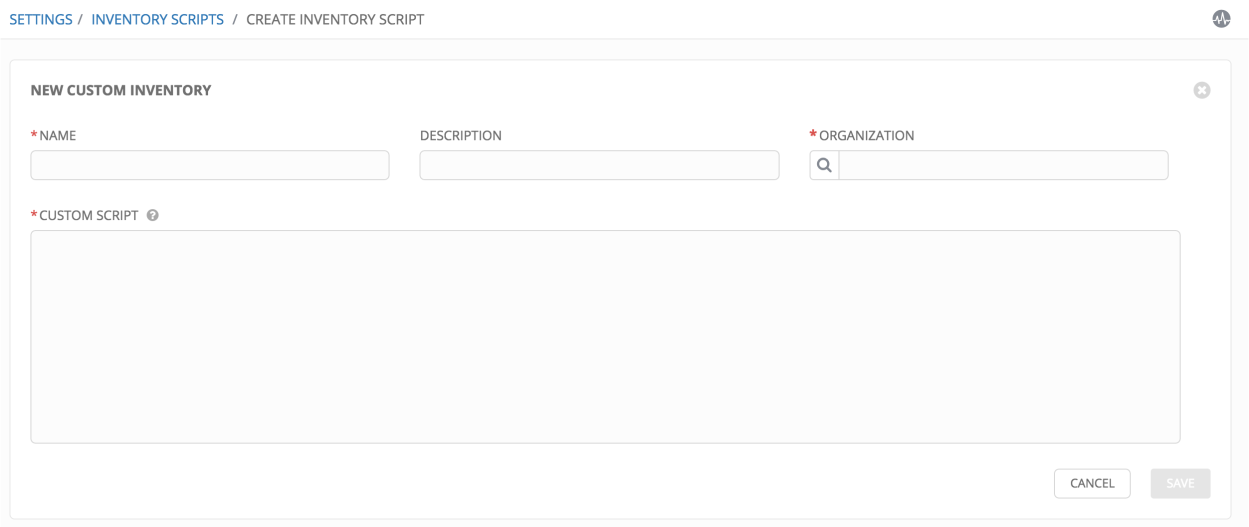
Tower includes built-in support for syncing dynamic inventory from cloud sources such as Amazon AWS, Google Compute Engine, among others. Tower also offers the ability to use a custom script to pull from your own inventory source.

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.

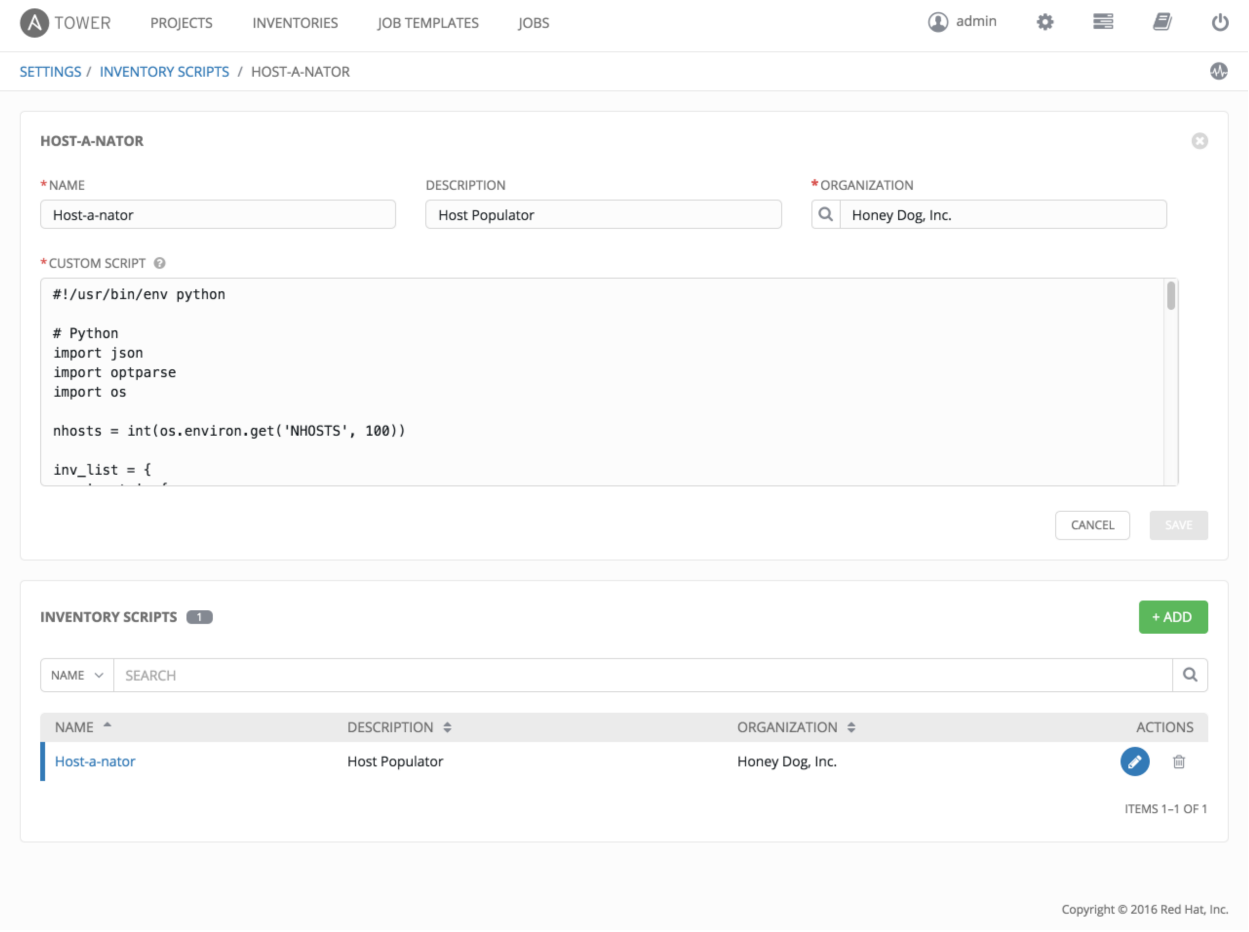


To add a new custom inventory script, click the + ADD button.



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.



*NAME: Host-a-nator

DESCRIPTION: Host Populator

*ORGANIZATION: Honey Dog, Inc.

*CUSTOM SCRIPT

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

# Python
import json
import optparse
import os

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

inv_list = {
```

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

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.

CHAPTER

FOUR

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.

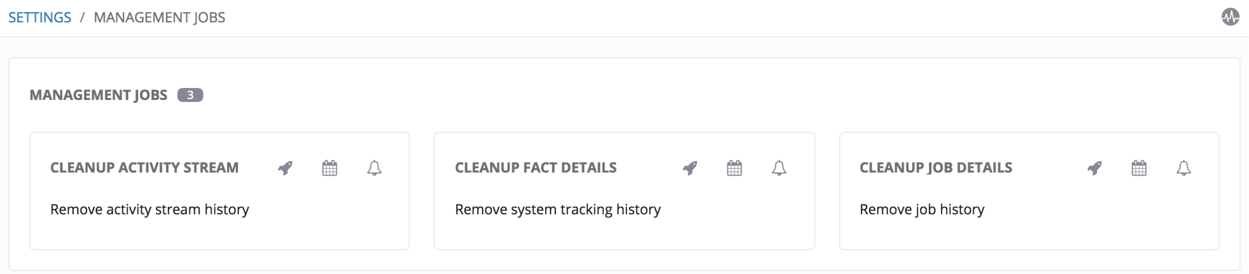
For versions of Ansible 2.4 and later, the officially distributed `ansible-inventory` command will be used to process inventory files.

CHAPTER

FIVE

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**.



SETTINGS / MANAGEMENT JOBS

MANAGEMENT JOBS 3

CLEANUP ACTIVITY STREAM

Remove activity stream history

CLEANUP FACT DETAILS

Remove system tracking history

CLEANUP JOB DETAILS

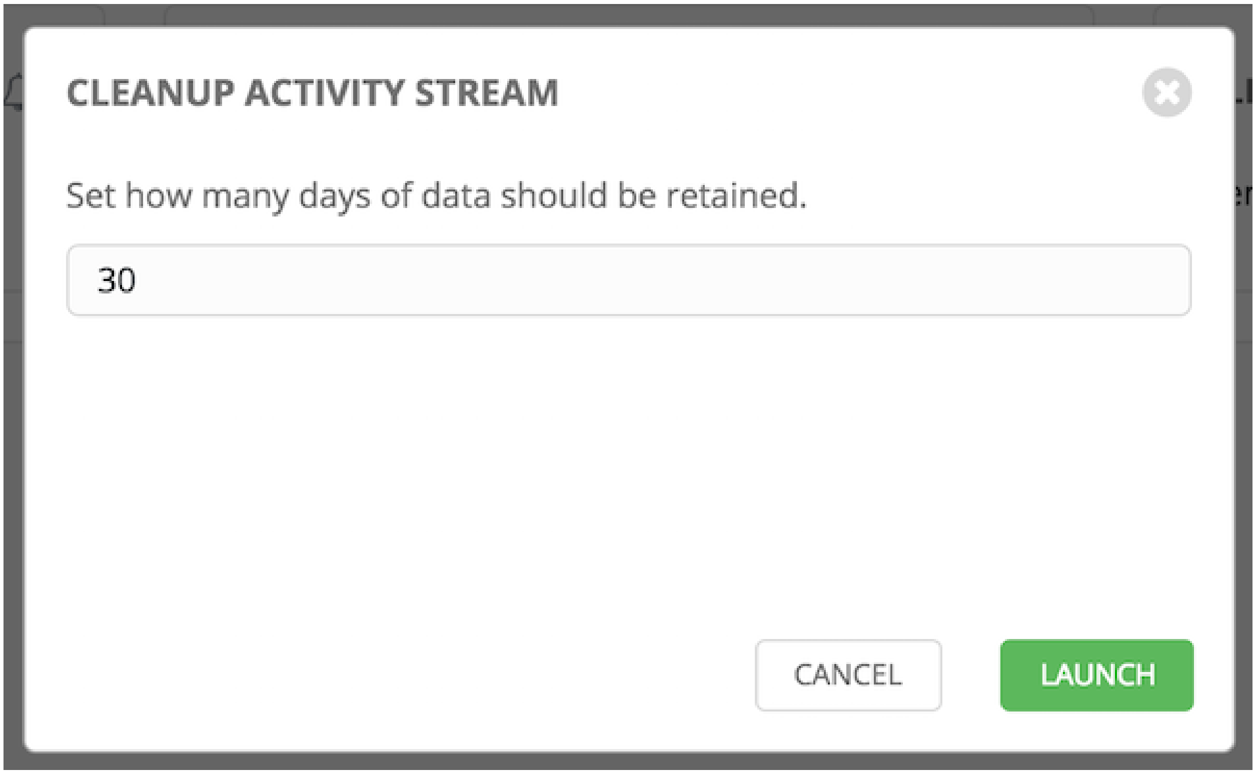
Remove job history

Several 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 Fact Details**: Remove system tracking history
- **Cleanup Job Details**: Remove job history older than a specified number of days

5.1 Removing Old Activity Stream Data

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



CLEANUP ACTIVITY STREAM

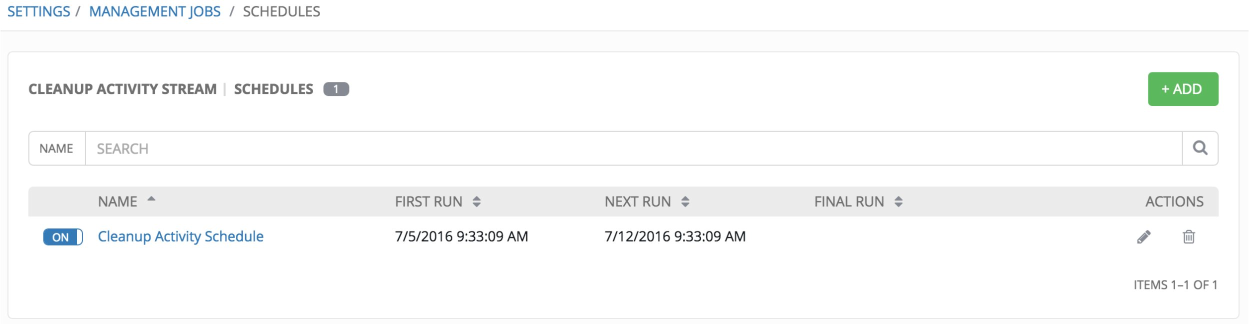
Set how many days of data should be retained.

30

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

5.1.1 Scheduling

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



SETTINGS / MANAGEMENT JOBS / SCHEDULES

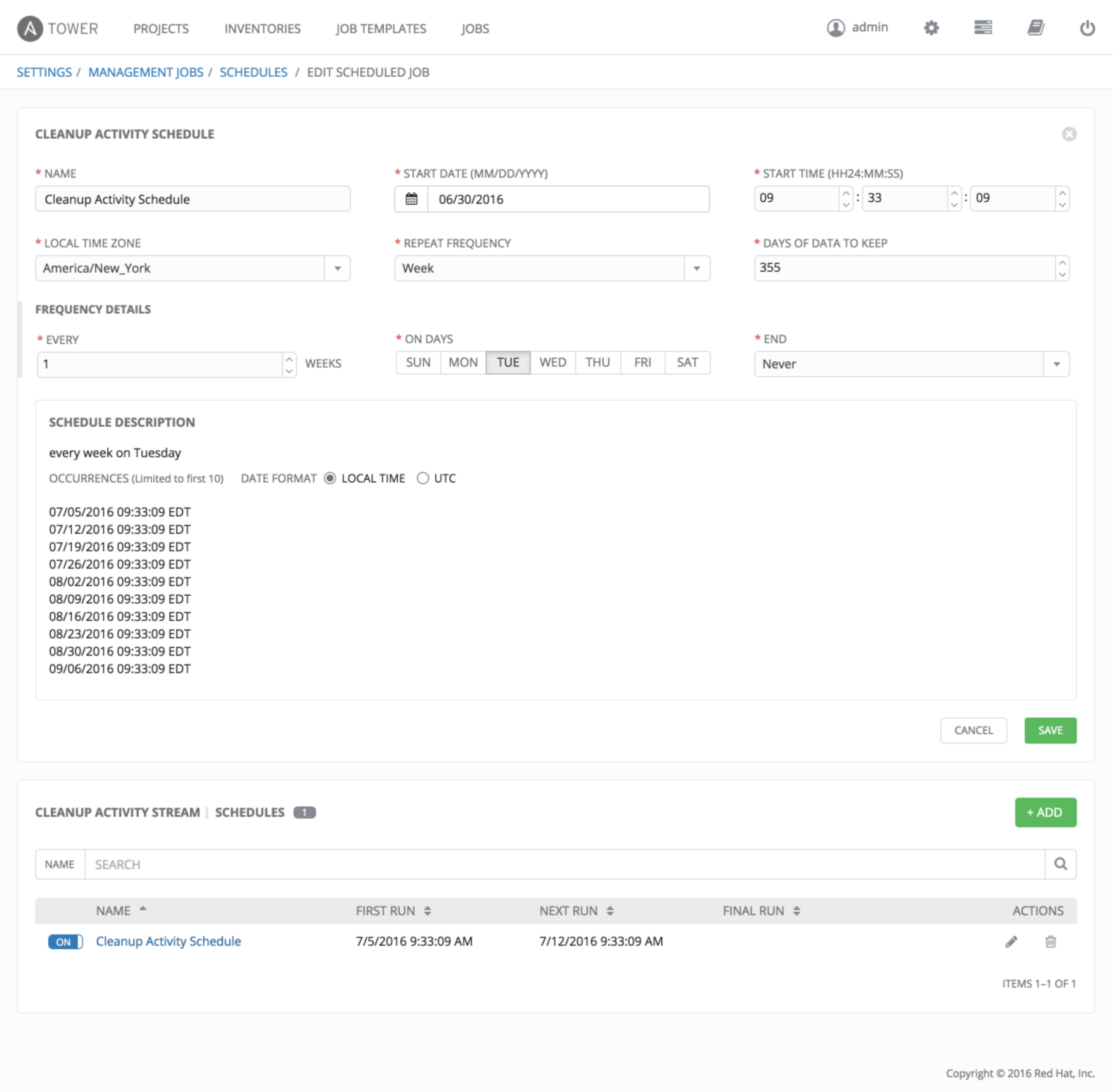
CLEANUP ACTIVITY STREAM | SCHEDULES 1

	NAME	FIRST RUN	NEXT RUN	FINAL RUN	ACTIONS
ON	Cleanup Activity Schedule	7/5/2016 9:33:09 AM	7/12/2016 9:33:09 AM		

ITEMS 1–1 OF 1

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 + ADD button to create a new schedule for this management job.


TOWER
PROJECTS
INVENTORIES
JOB TEMPLATES
JOBS
admin
SETTINGS / MANAGEMENT JOBS / SCHEDULES / EDIT SCHEDULED JOB
CANCEL
SAVE
+ ADD
NAME
SEARCH
Copyright © 2016 Red Hat, Inc.

CLEANUP ACTIVITY SCHEDULE

* NAME	* START DATE (MM/DD/YYYY)	* START TIME (HH24:MM:SS)
Cleanup Activity Schedule	06/30/2016	09 : 33 : 09
* LOCAL TIME ZONE	* REPEAT FREQUENCY	* DAYS OF DATA TO KEEP
America/New_York	Week	355

FREQUENCY DETAILS

* EVERY	* ON DAYS	* END
1 WEEKS	SUN MON [✓ TUE] WED THU FRI SAT	Never

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
```

CLEANUP ACTIVITY STREAM | SCHEDULES 1

	NAME	FIRST RUN	NEXT RUN	FINAL RUN	ACTIONS
ON	Cleanup Activity Schedule	7/5/2016 9:33:09 AM	7/12/2016 9:33:09 AM		

ITEMS 1–1 OF 1

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

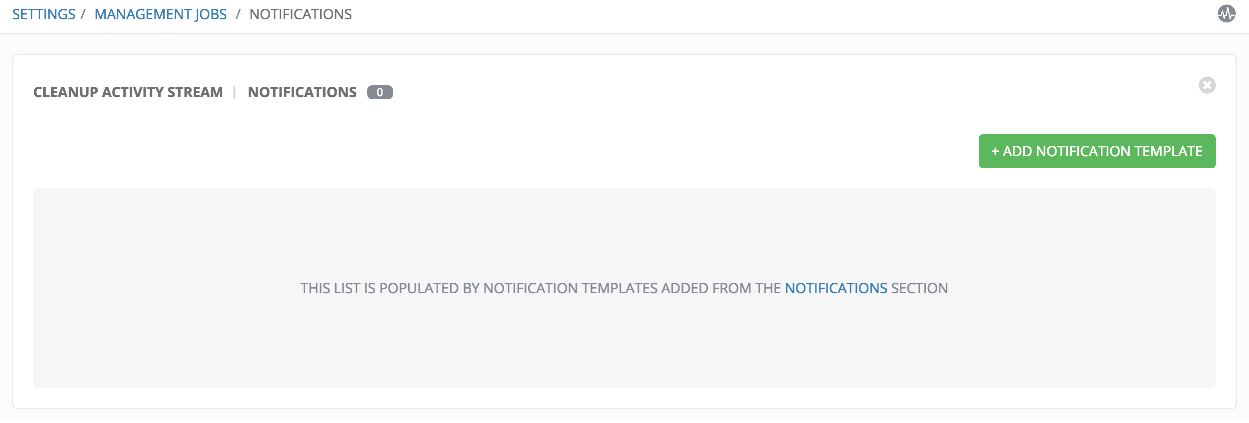
- Name (required)
- Start Date (required)
- Start Time (required)
- Local Time Zone (the entered Start Time should be in this timezone)
- Repeat Frequency (the appropriate options display as the update frequency is modified.)

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

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.

5.1.2 Notifications

To set or review notifications associated with this management job, click the Configure Notifications (🔔) button. You can also access notifications through the Settings (⚙) menu.



SETTINGS / MANAGEMENT JOBS / NOTIFICATIONS

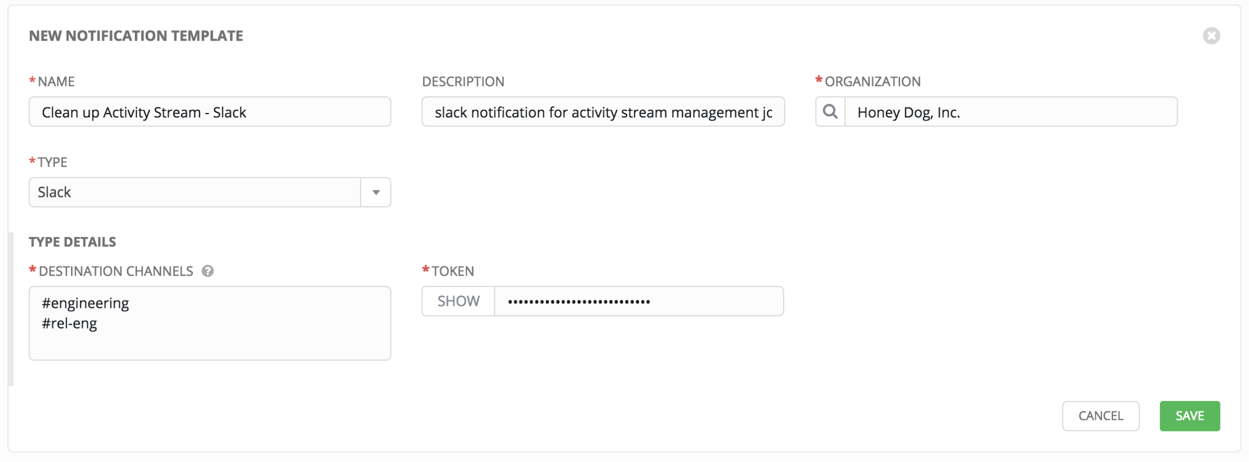
CLEANUP ACTIVITY STREAM | NOTIFICATIONS 0

+ ADD NOTIFICATION TEMPLATE

THIS LIST IS POPULATED BY NOTIFICATION TEMPLATES ADDED FROM THE NOTIFICATIONS SECTION

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

- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC



NEW NOTIFICATION TEMPLATE

*NAME: Clean up Activity Stream - Slack

DESCRIPTION: slack notification for activity stream management jc

*ORGANIZATION: Honey Dog, Inc.

*TYPE: Slack

TYPE DETAILS

*DESTINATION CHANNELS: #engineering
#rel-eng

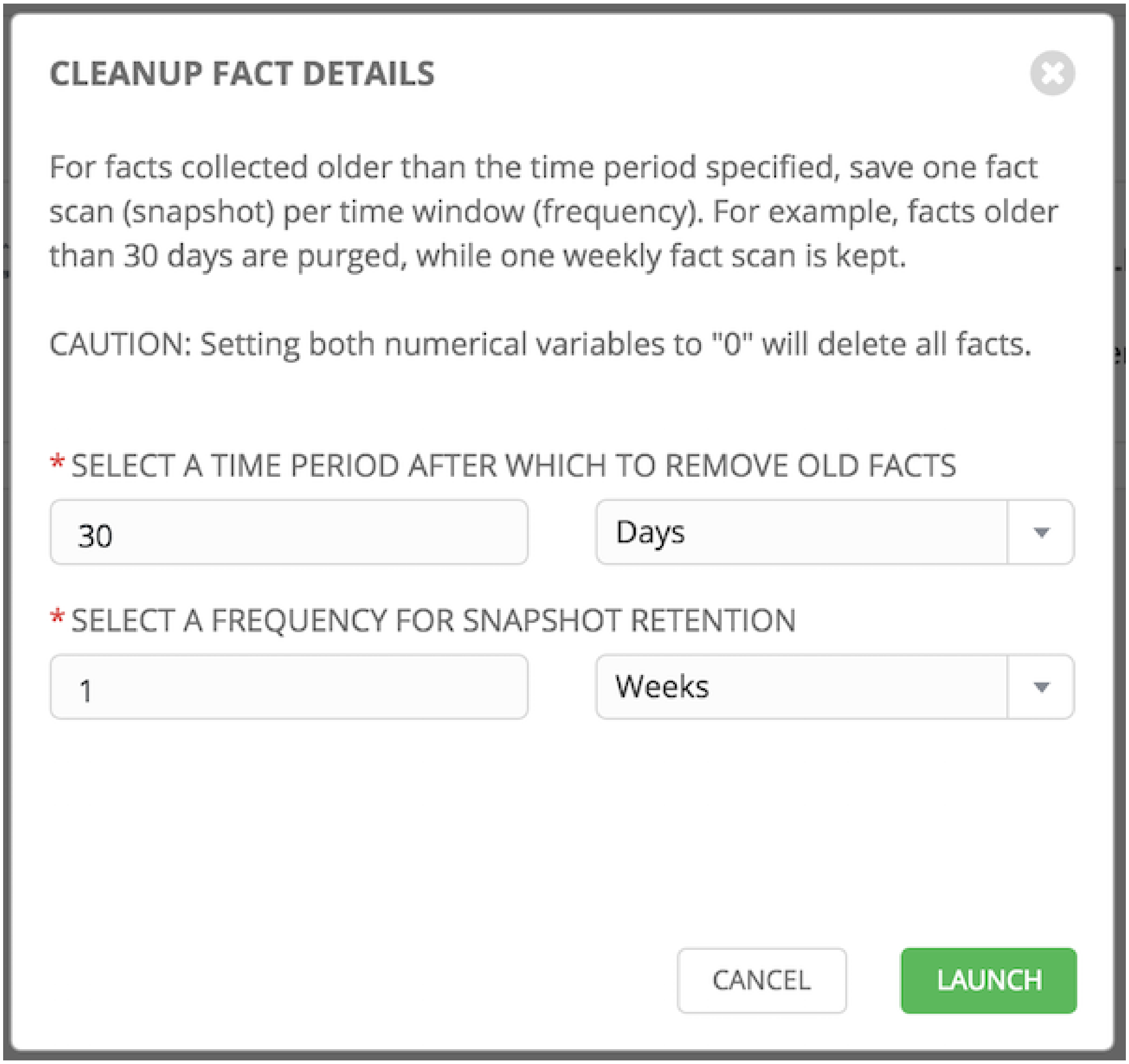
*TOKEN: SHOW ••••••••••••••••••••••••••••

CANCEL SAVE

Refer to Notifications in the *Ansible Tower User Guide* for more information.

5.2 Removing Old Fact (System Tracking) Data

To remove system tracking data, click on the 🚀 button beside **Cleanup Fact Details**.



CLEANUP FACT DETAILS

For facts collected older than the time period specified, save one fact scan (snapshot) per time window (frequency). For example, facts older than 30 days are purged, while one weekly fact scan is kept.

CAUTION: Setting both numerical variables to "0" will delete all facts.

* SELECT A TIME PERIOD AFTER WHICH TO REMOVE OLD FACTS

30 | Days

* SELECT A FREQUENCY FOR SNAPSHOT RETENTION

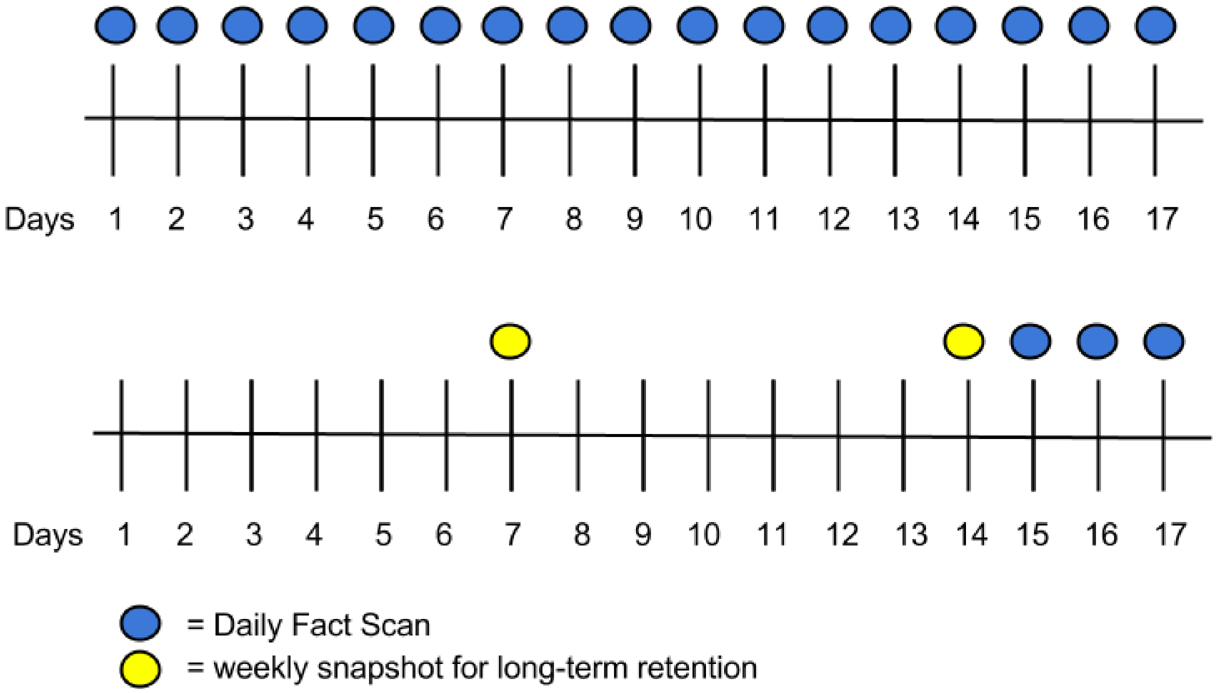
1 | Weeks

Select the **time period** after which you want to remove old data as well as the **frequency** for snapshot retention.

For facts collected older than the time period specified, you can choose to save one fact scan (or snapshot) per period of time(frequency). For example, facts older than 30 days could be purged, while one weekly fact scan is retained.

> **Warning:** Setting both numerical variables to “0” will delete all facts.

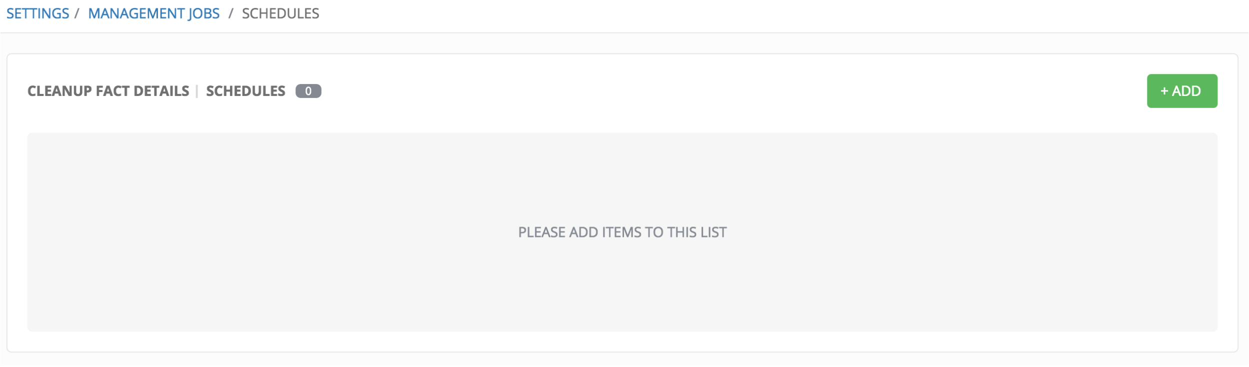
To help clarify this purge and retention schedule, consider the following timeline:


Days 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
Days 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
= Daily Fact Scan
= weekly snapshot for long-term retention

For this timeline example, consider that you have been running Tower for 17 days (since Jan 1st) and have collected 17 days of fact scans . On Jan 17, you decide to remove all fact scans older than 3 days while keeping a weekly snapshot. The most recent scan and a scan from one week earlier remains, along with the most recent data to be kept.

5.2.1 Scheduling

To review or set a schedule for cleaning up system tracking information, click on the 📅 button.



SETTINGS / MANAGEMENT JOBS / SCHEDULES

CLEANUP FACT DETAILS | SCHEDULES 0

+ ADD

PLEASE ADD ITEMS TO THIS LIST

You can use the + ADD button to create a new schedule for this management job.

Enter the appropriate details into the following fields and select **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.)

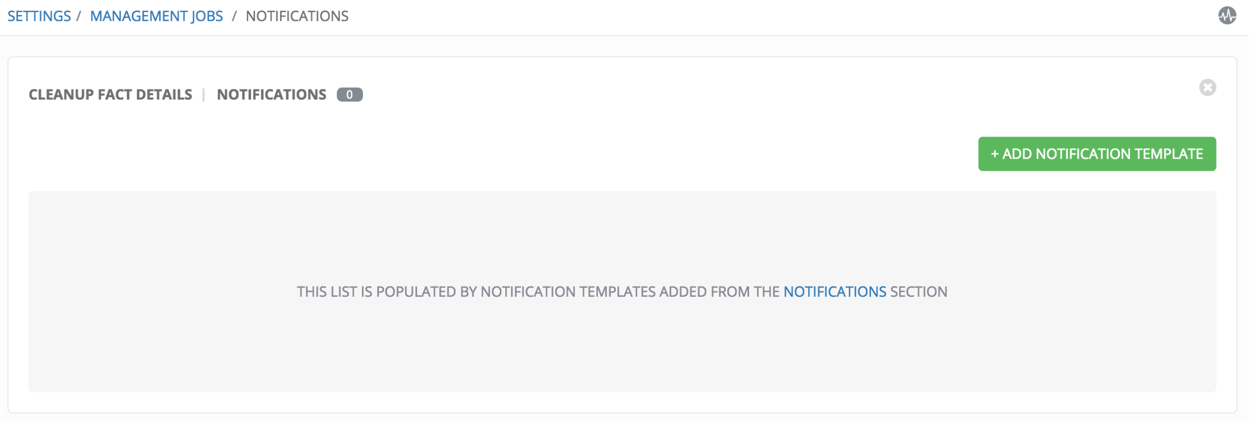
The **Details** tab displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

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.

5.2.2 Notifications

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

You can also access notifications through the Settings (⚙) menu.



SETTINGS / MANAGEMENT JOBS / NOTIFICATIONS

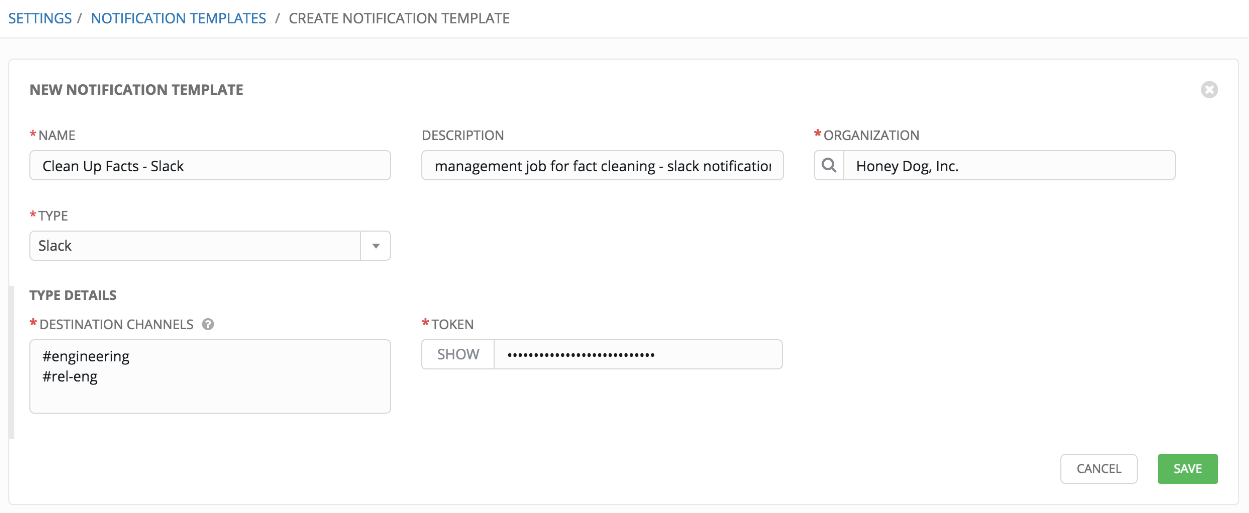
CLEANUP FACT DETAILS | NOTIFICATIONS 0

+ ADD NOTIFICATION TEMPLATE

THIS LIST IS POPULATED BY NOTIFICATION TEMPLATES ADDED FROM THE NOTIFICATIONS SECTION

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

- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC



SETTINGS / NOTIFICATION TEMPLATES / CREATE NOTIFICATION TEMPLATE

NEW NOTIFICATION TEMPLATE

*NAME	DESCRIPTION	*ORGANIZATION
Clean Up Facts - Slack	management job for fact cleaning - slack notificatioı	Honey Dog, Inc.
*TYPE		
Slack		

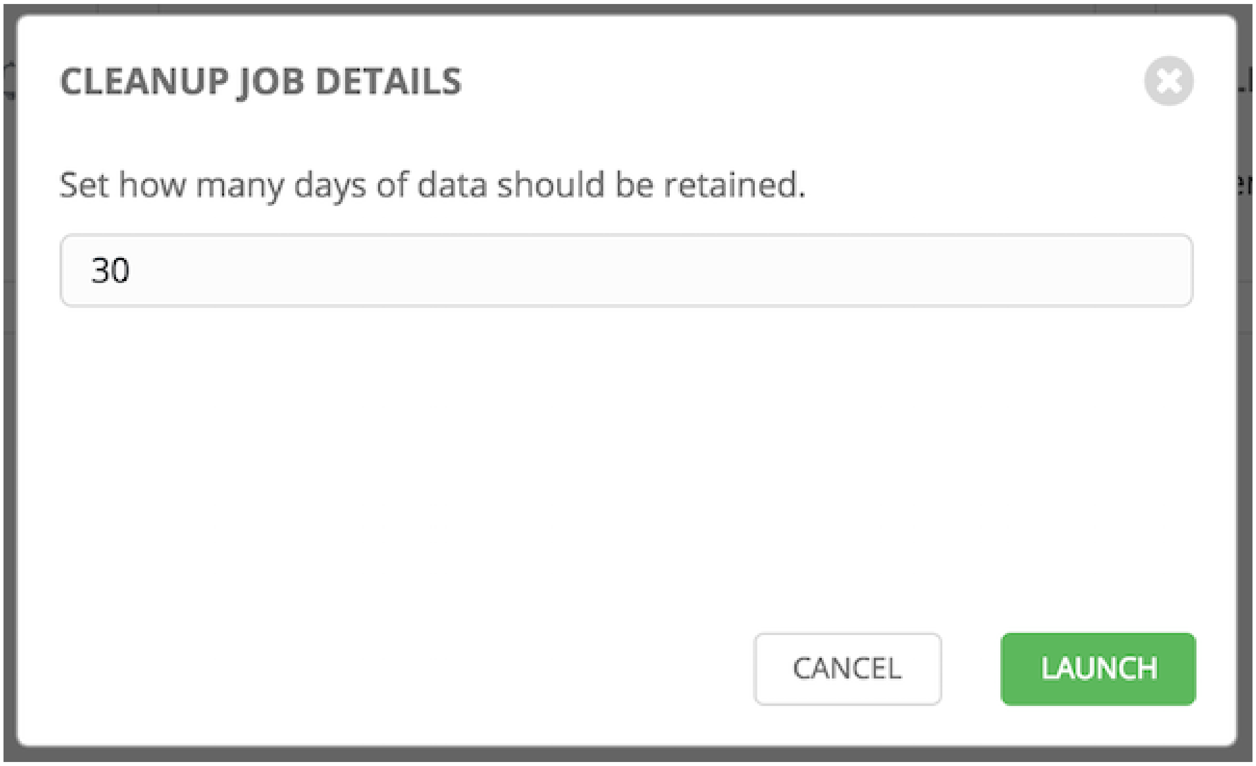
TYPE DETAILS

*DESTINATION CHANNELS	*TOKEN
#engineering #rel-eng	SHOW ••••••••••••••••••••••••••••••

Refer to Notifications in the *Ansible Tower User Guide* for more information.

5.3 Removing Old Job History

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



CLEANUP JOB DETAILS

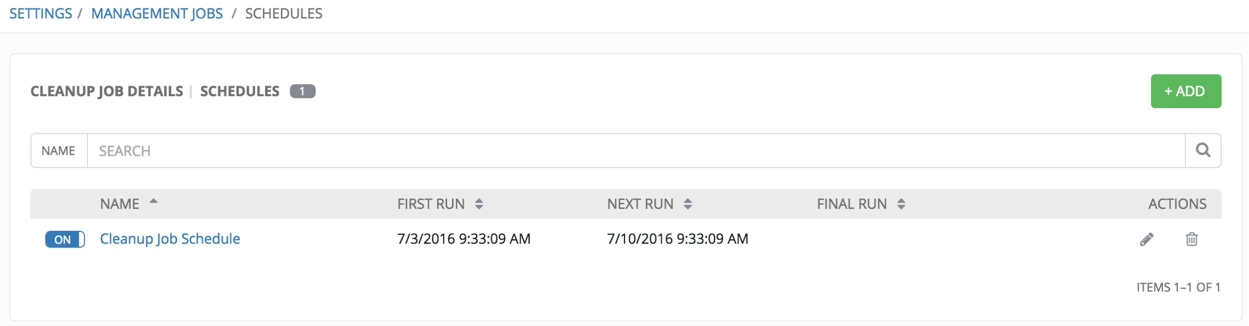
Set how many days of data should be retained.

30

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

5.3.1 Scheduling

To review or set a schedule for cleaning up job history, click on the 📅 button.



SETTINGS / MANAGEMENT JOBS / SCHEDULES

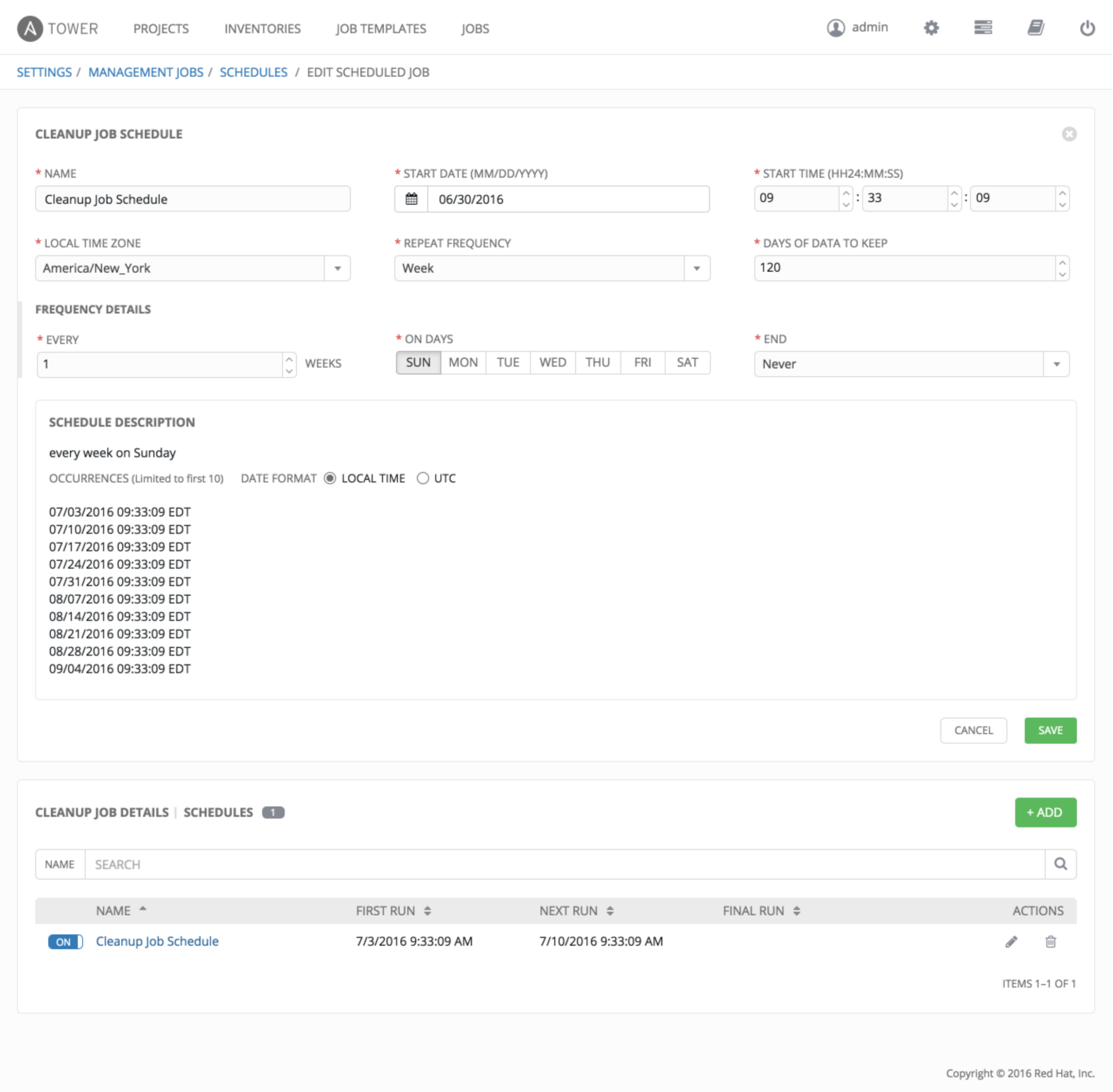
CLEANUP JOB DETAILS | SCHEDULES 1

	NAME	FIRST RUN	NEXT RUN	FINAL RUN	ACTIONS
ON	Cleanup Job Schedule	7/3/2016 9:33:09 AM	7/10/2016 9:33:09 AM		

ITEMS 1–1 OF 1

Note that you can easily 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 Job Schedule", to review or edit the schedule settings. You can also use the + ADD button to create a new schedule for this management job.


TOWER
PROJECTS
INVENTORIES
JOB TEMPLATES
JOBS
admin
SETTINGS / MANAGEMENT JOBS / SCHEDULES / EDIT SCHEDULED JOB
CANCEL
SAVE
+ ADD
Copyright © 2016 Red Hat, Inc.

CLEANUP JOB SCHEDULE

* NAME: Cleanup Job Schedule	* START DATE (MM/DD/YYYY): 06/30/2016	* START TIME (HH24:MM:SS): 09 : 33 : 09
* LOCAL TIME ZONE: America/New_York	* REPEAT FREQUENCY: Week	* DAYS OF DATA TO KEEP: 120

FREQUENCY DETAILS

* EVERY: 1 WEEKS	* ON DAYS: [✓ SUN] MON TUE WED THU FRI SAT	* END: Never

SCHEDULE DESCRIPTION

every week on Sunday

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

```
07/03/2016 09:33:09 EDT
07/10/2016 09:33:09 EDT
07/17/2016 09:33:09 EDT
07/24/2016 09:33:09 EDT
07/31/2016 09:33:09 EDT
08/07/2016 09:33:09 EDT
08/14/2016 09:33:09 EDT
08/21/2016 09:33:09 EDT
08/28/2016 09:33:09 EDT
09/04/2016 09:33:09 EDT
```

CLEANUP JOB DETAILS | SCHEDULES 1

NAME SEARCH

	NAME	FIRST RUN	NEXT RUN	FINAL RUN	ACTIONS
ON	Cleanup Job Schedule	7/3/2016 9:33:09 AM	7/10/2016 9:33:09 AM		

ITEMS 1–1 OF 1

Enter the appropriate details into the following fields and select **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.)

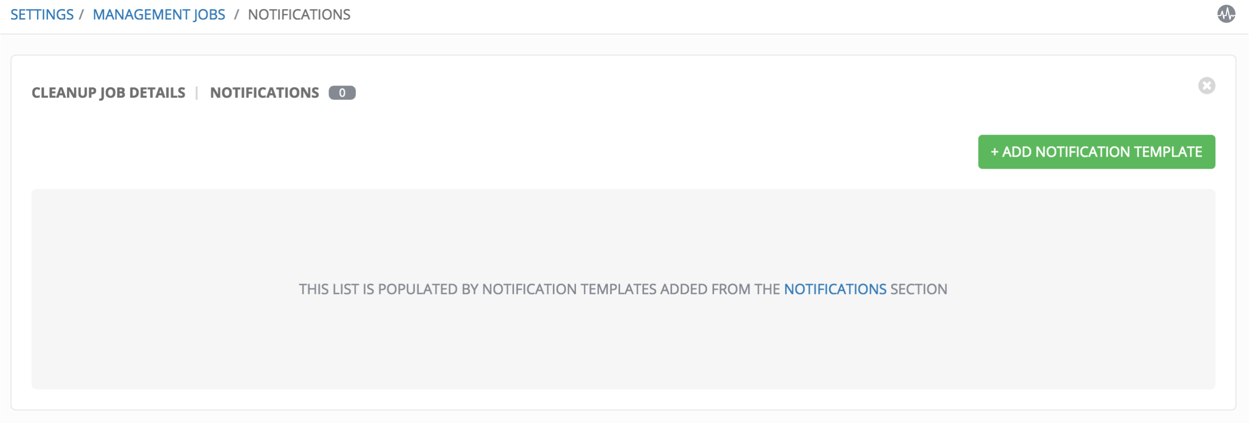
The **Details** tab displays a description of the schedule and a list of the scheduled occurrences in the selected Local Time Zone.

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.

5.3.2 Notifications

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

You can also access notifications through the Settings (⚙) menu.


SETTINGS / MANAGEMENT JOBS / NOTIFICATIONS

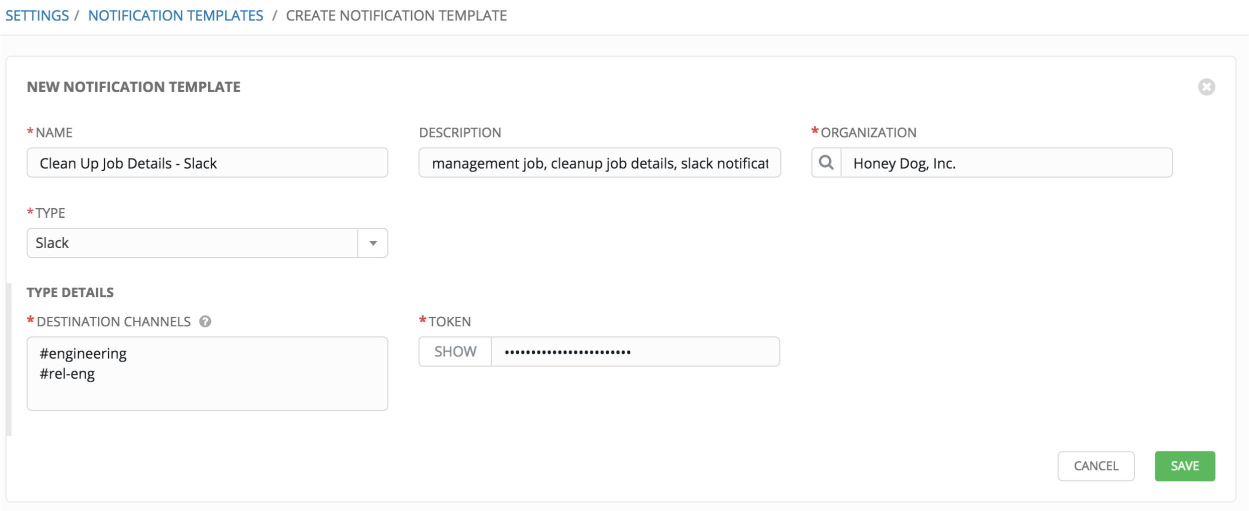
CLEANUP JOB DETAILS | NOTIFICATIONS 0

+ ADD NOTIFICATION TEMPLATE

THIS LIST IS POPULATED BY NOTIFICATION TEMPLATES ADDED FROM THE NOTIFICATIONS SECTION

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

- Email
- Slack
- Twilio
- PagerDuty
- HipChat
- Webhook
- IRC



SETTINGS / NOTIFICATION TEMPLATES / CREATE NOTIFICATION TEMPLATE

NEW NOTIFICATION TEMPLATE

*NAME: Clean Up Job Details - Slack

DESCRIPTION: management job, cleanup job details, slack notificat

*ORGANIZATION: Honey Dog, Inc.

*TYPE: Slack

TYPE DETAILS

*DESTINATION CHANNELS:
#engineering
#rel-eng

*TOKEN: ••••••••••••••••••••••••••

Refer to Notifications in the *Ansible Tower User Guide* for more information.

CHAPTER

SIX

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.

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

6.1 Setup Considerations

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.
- 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.

6.2 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.

Instances in the `tower` group are responsible for housekeeping tasks like determining where jobs are supposed to be launched and processing playbook events. Moreover, if all tower instance group members fail, then jobs might not be able to run and playbook events might not get written. Therefore, it is important to have enough cluster instances in the `tower` group to handle not only housekeeping tasks, but be able to act as backup in the event of a failure as well.

```
[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.

- 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_use_long_names` : RabbitMQ is sensitive to what each instance is named. Tower is flexible enough to allow FQDNs (host01.example.com), short names (host01), or ip addresses (192.168.5.73). Depending on what is used to identify each host in the inventory file, this value may need to be changed:
    - For FQDNs and IP addresses, this value needs to be `true`.

    - For short names, set the value to `false`.
    - If you are using localhost, do not change the default setting of `rabbitmq_use_long_name=false` to *true*.
    - If instances are provisioned to where they reference other instances internally and not on external addresses, then the value for the long name should follow the internal addressing format (see `rabbitmq_host` above).

  - `rabbitmq_enable_manager`: Set this to true to expose the RabbitMQ Management Web Console on each instance.

6.2.1 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.

6.2.2 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))

6.3 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

```

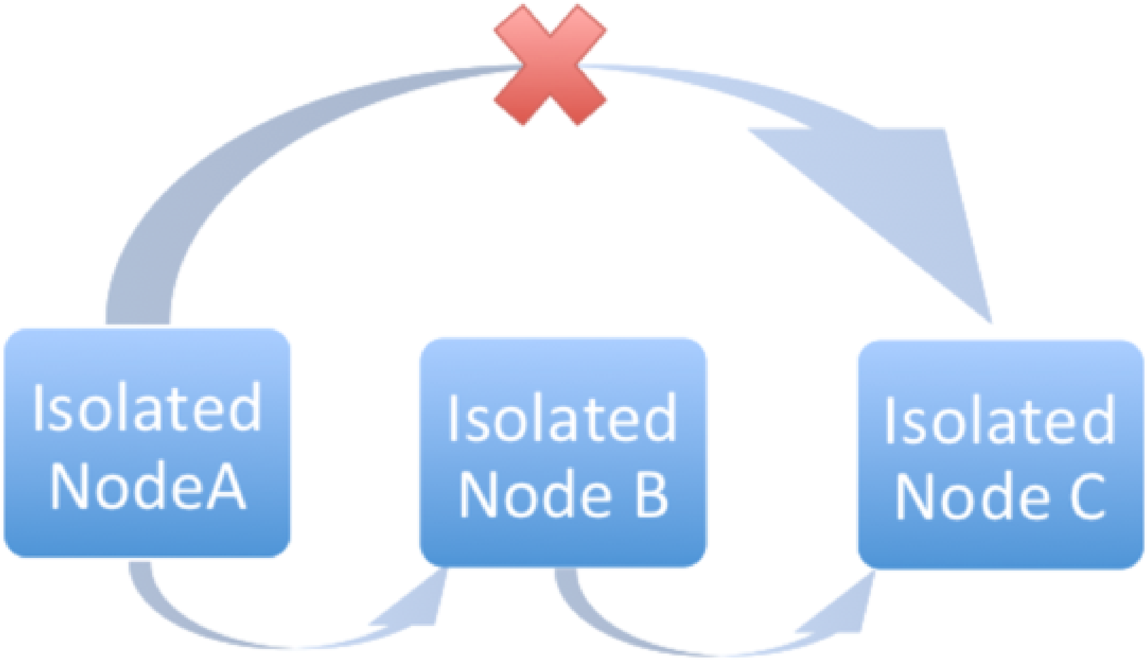
```
[instance_group_security]
towerB
towerC

[isolated_group_govcloud]
isolatedA
isolatedB

[isolated_group_govcloud:vars]
controller=security
```

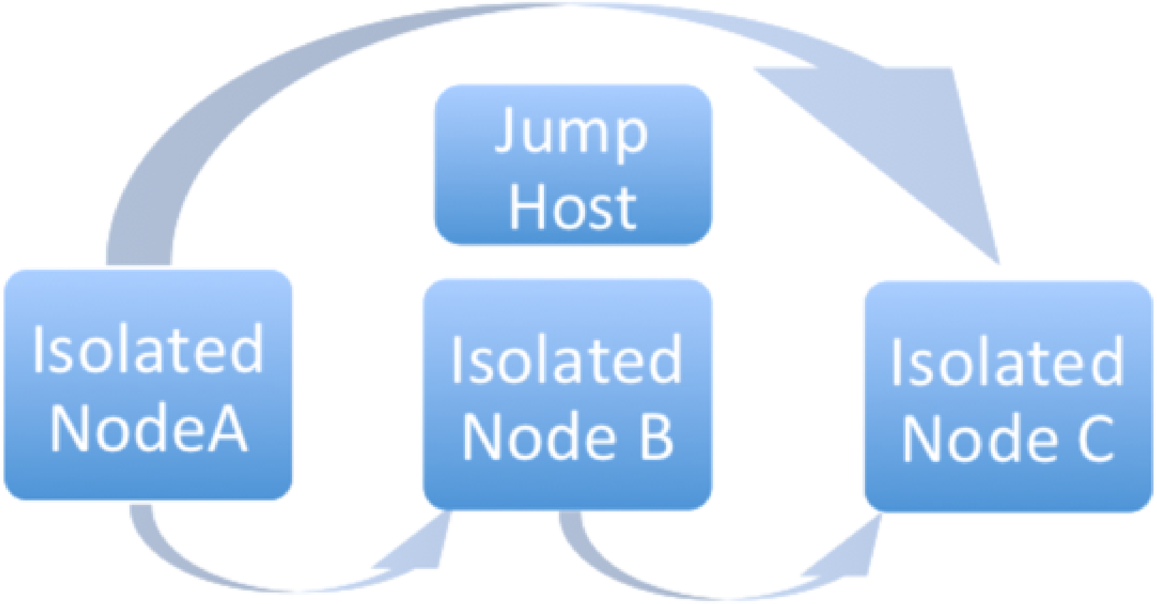
In the isolated instance group model, “controller” instances interact with “isolated” instances via a series of Ansible playbooks over SSH. At installation time, by default, a randomized RSA key is generated and distributed as an authorized key to all “isolated” instances. The private half of the key 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.

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.

6.3.1 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), you can disable this behavior by turning it off in the inventory file used to install Tower:

```
[tower]
towerA
towerB
towerC

[instance_group_security]
towerB
towerC

[isolated_group_govcloud]
isolatedA
isolatedB

[isolated_group_govcloud:vars]
controller=security

# Isolated Tower nodes automatically generate an RSA key for authentication;
# To disable this behavior, set this value to false
isolated_key_generation=false
```

6.4 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
- The status of all the instance groups
- 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/`.

6.5 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.

- Celery - The worker queue that processes and runs all jobs.
- RabbitMQ - This message broker is used as a signaling mechanism for Celery as well as any event data propagated to the application.
- Memcached - local caching service for the instance it lives on.

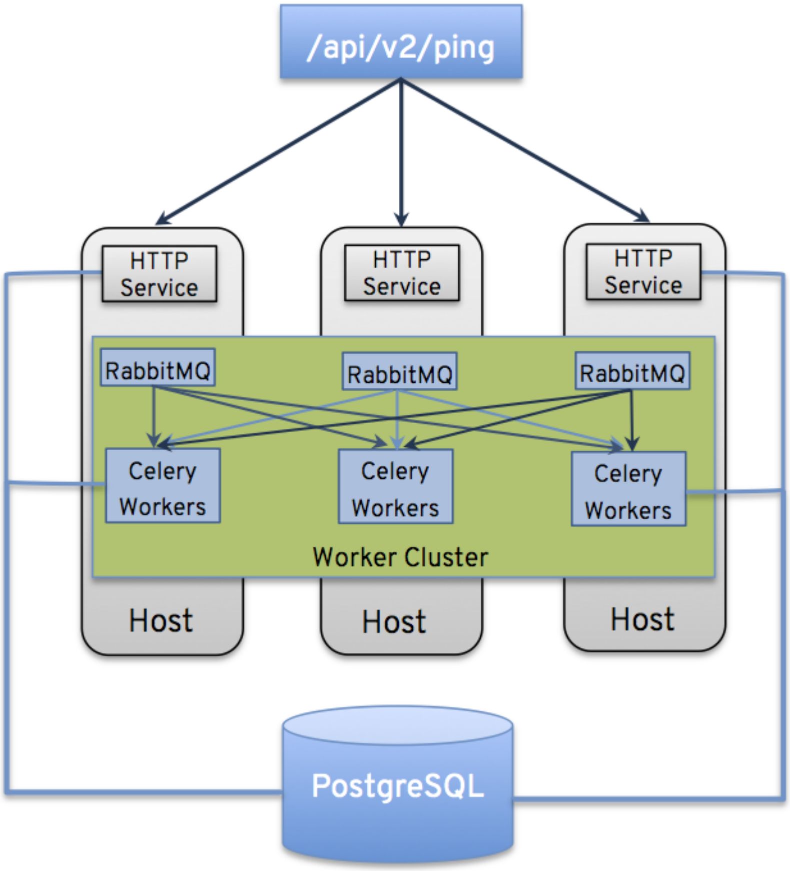
Tower is configured in such a way that if any of these services or their components fail, then all services are restarted. If these fail sufficiently often in a short span of time, then the entire instance will be placed offline in an automated fashion in order to allow remediation without causing unexpected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

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

6.6 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 Celery 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.

6.6.1 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.

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.

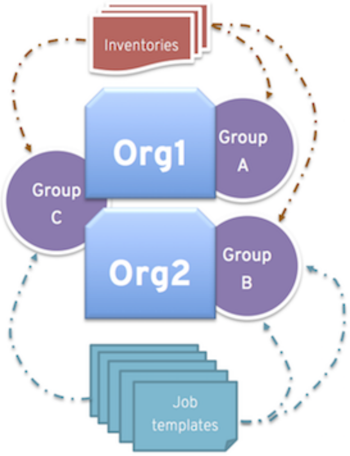
This presents two other great use cases:

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.

6.7 Deprovision Instances and Instance Groups

Deprovisioning Tower does not automatically deprovision instances since clusters do not currently distinguish between a 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>`

CHAPTER

SEVEN

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.

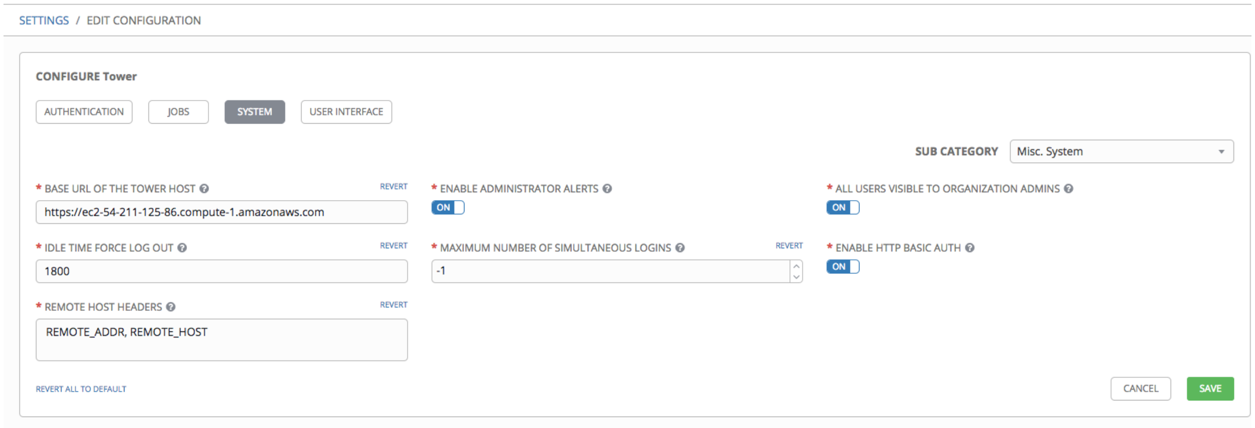
When offloading SSL to the load balancer or proxy, set `nginx_disable_https=true` as an extra variable passed to the setup playbook. Refer to The Setup Playbook for information on applying extra variables to the setup playbook.

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 Tower settings (`/etc/tower/conf.d/remote_host_headers.py`). By default `REMOTE_HOST_HEADERS` is set to `['REMOTE_ADDR', 'REMOTE_HOST']`.

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

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



SETTINGS / EDIT CONFIGURATION

CONFIGURE Tower

SUB CATEGORY: Misc. System

* 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

* IDLE TIME FORCE LOG OUT: 1800

* MAXIMUM NUMBER OF SIMULTANEOUS LOGINS: -1

* ENABLE HTTP BASIC AUTH: ON

* REMOTE HOST HEADERS: 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
```

7.1 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
- /etc/tower/conf.d/remote_host_headers.py defines 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"
```

7.2 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']
```

CHAPTER EIGHT

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
- socketio_service.log
- task_system.log
- tower.log

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

- awx-celeryd.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 |
             ||     ||
```

CHAPTER

NINE

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.

9.1 Logging Aggregator Services

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

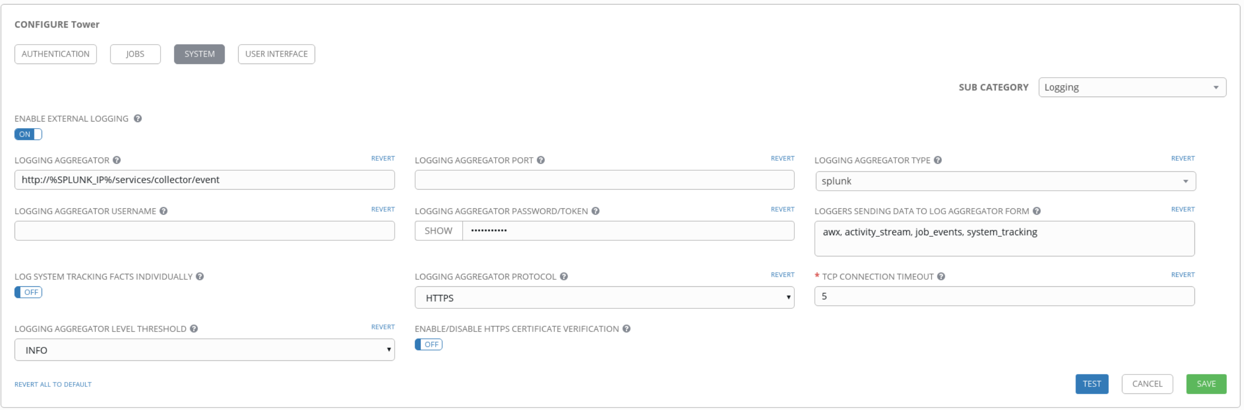
- *Splunk*
- *Loggly*
- *Sumologic*
- *Elastic stack (formerly ELK stack)*

9.1.1 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/v1/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:



SUB CATEGORY: Logging

ENABLE EXTERNAL LOGGING: ON

LOGGING AGGREGATOR: http://%SPLUNK_IP%/services/collector/event

LOGGING AGGREGATOR PORT:

LOGGING AGGREGATOR TYPE: splunk

LOGGING AGGREGATOR USERNAME:

LOGGING AGGREGATOR PASSWORD/TOKEN: SHOW •••••••••••

LOGGERS SENDING DATA TO LOG AGGREGATOR FORM: awx, activity_stream, job_events, system_tracking

LOG SYSTEM TRACKING FACTS INDIVIDUALLY: OFF

LOGGING AGGREGATOR PROTOCOL: HTTPS

* TCP CONNECTION TIMEOUT: 5

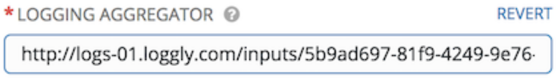
LOGGING AGGREGATOR LEVEL THRESHOLD: INFO

ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION: OFF

For further instructions on configuring the HTTP Event Collector, refer to the Splunk documentation.

9.1.2 Loggly

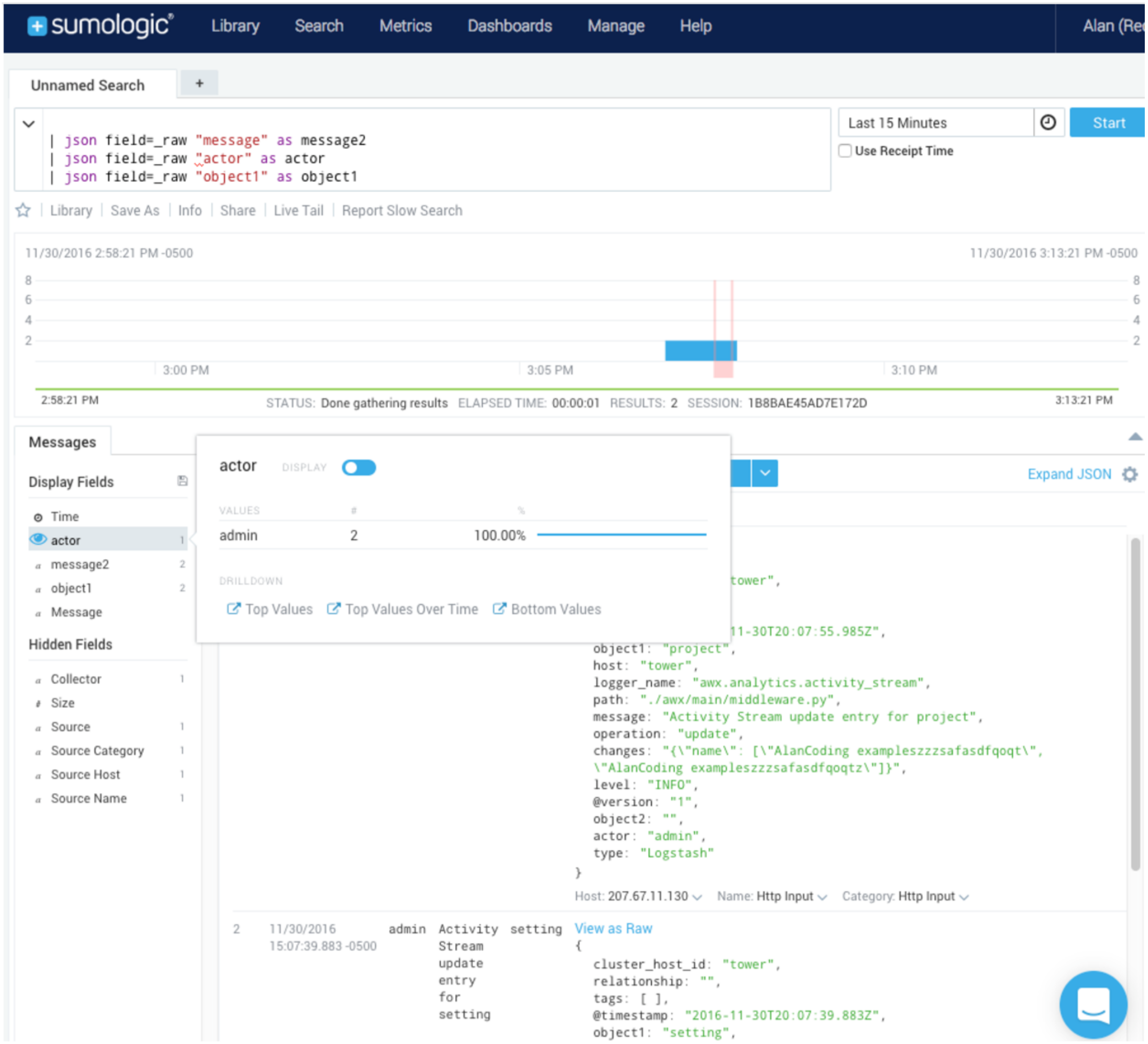
To set up the sending of logs through Loggly's HTTP endpoint, refer to https://www.loggly.com/docs/http-endpoint/. Loggly uses the URL convention described at http://logs-01.loggly.com/inputs/TOKEN/tag/http/, which is shown inputted in the **Logging Aggregator** field in the example below:



* LOGGING AGGREGATOR: http://logs-01.loggly.com/inputs/5b9ad697-81f9-4249-9e76-

9.1.3 Sumologic

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



```
| json field=_raw "message" as message2
| json field=_raw "actor" as actor
| json field=_raw "object1" as object1
```

STATUS: Done gathering results ELAPSED TIME: 00:00:01 RESULTS: 2 SESSION: 1B8BAE45AD7E172D

VALUES	#	%
admin	2	100.00%

```
                                    tower",

                                    11-30T20:07:55.985Z",
    object1: "project",
    host: "tower",
    logger_name: "awx.analytics.activity_stream",
    path: "./awx/main/middleware.py",
    message: "Activity Stream update entry for project",
    operation: "update",
    changes: "{\"name\": [\"AlanCoding exampleszzzsafasdfqoqt\",
    \"AlanCoding exampleszzzsafasdfqoqtz\"]}",
    level: "INFO",
    @version: "1",
    object2: "",
    actor: "admin",
    type: "Logstash"
}
Host: 207.67.11.130   Name: Http Input   Category: Http Input

2   11/30/2016          admin  Activity  setting  View as Raw
    15:07:39.883 -0500         Stream             {
                               update               cluster_host_id: "tower",
                               entry                relationship: "",
                               for                  tags: [ ],
                               setting              @timestamp: "2016-11-30T20:07:39.883Z",
                                                    object1: "setting",
```

9.1.4 Elastic stack (formerly ELK stack)

You can visualize information from the Tower logs in Kibana, captured via an Elastic stack consuming the logs. Ansible Tower provides compatibility with the logstash connector, and compatibility with the data model of elastic search. You can use the example settings, and either a library or provided examples to stand up containers that will demo the Elastic stack use end-to-end.

Tower uses logstash configuration to specify the source of the logs. Use this template to provide the input:

```
input {
       http {
        port => 8085
        user => logger_username
        password => "password"
       }
}
```

Add this to your configuration file in order to process the message content:

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

9.2 Set Up Logging with Tower

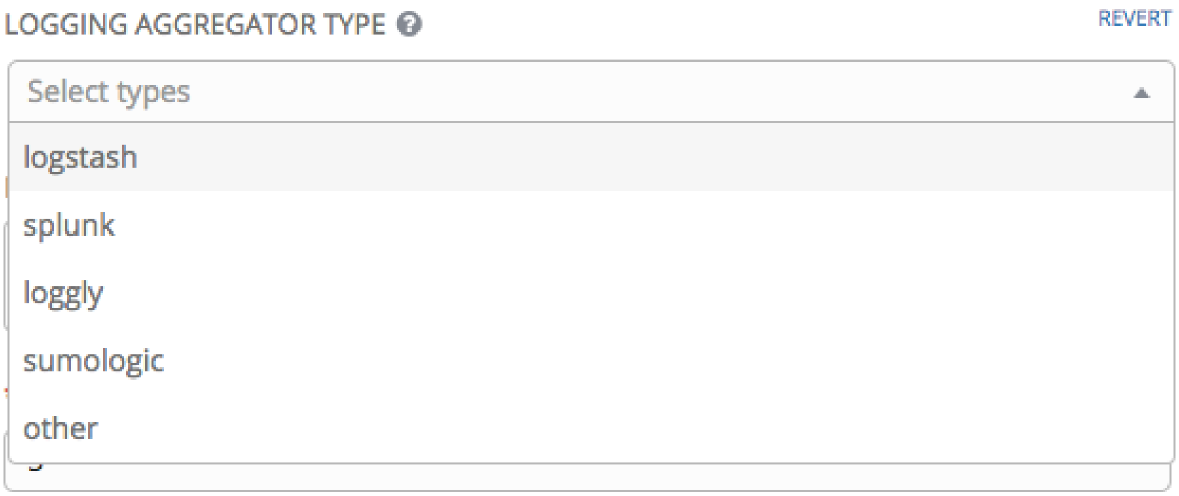
To set up logging to any of the aggregator types:

1. From the Settings (⚙) Menu screen, click on **Configure Tower**.
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.

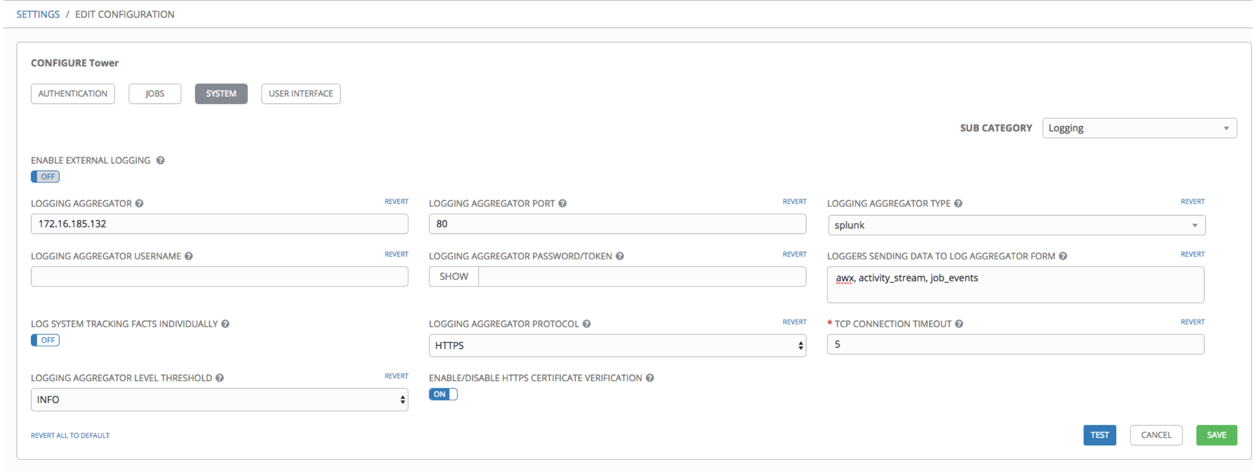
- **Logging Aggregator Type**: Click to select the aggregator service from the drop-down menu:



- **Logging Aggregator 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 Aggeregator 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:



SUB CATEGORY: Logging

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:

LOGGERS SENDING DATA TO LOG AGGREGATOR FORM: awx, activity_stream, job_events

LOG SYSTEM TRACKING FACTS INDIVIDUALLY: OFF

LOGGING AGGREGATOR PROTOCOL: HTTPS

* TCP CONNECTION TIMEOUT: 5

LOGGING AGGREGATOR LEVEL THRESHOLD: INFO

ENABLE/DISABLE HTTPS CERTIFICATE VERIFICATION: ON

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.

CHAPTER

TEN

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.

10.1 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.

10.2 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.

10.3 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.

CHAPTER

ELEVEN

TOWER CONFIGURATION

In Ansible Tower version 3.1.0, you can configure various Tower settings within the Tower user interface, in the following tabs:

- **Authentication**: Enable simplified login for your Tower applications.
- **Jobs**: Update settings pertaining to Jobs within Tower.
- **System**: Define system-level features and functions.
- **User Interface**: Set the level of data collection for use in usability analytics; and configure custom logos and login messages.

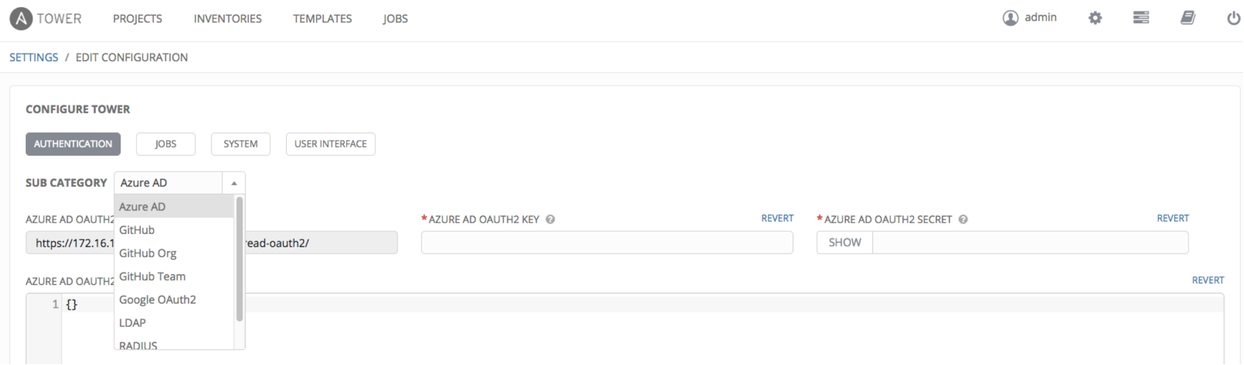
Each tab contains fields with a **Reset** button, allowing you to revert any value entered back to the default value. **Reset All** allows you to revert all the values in the Edit Tower Configuration to their factory default values.

Save applies changes you make, but it does not exit the edit dialog. Click (⚙) to return to the Settings Menu screen.

11.1 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 Settings (⚙) Menu screen, click **Configure Tower**.
2. The Authentication tab displays initially by default. Select the appropriate authentication type from the drop-down list.



SUB CATEGORY Azure AD

AZURE AD OAUTH2 https://172.16.1 ead-oauth2/

*AZURE AD OAUTH2 KEY

*AZURE AD OAUTH2 SECRET

AZURE AD OAUTH2

```
1 {}
```

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.

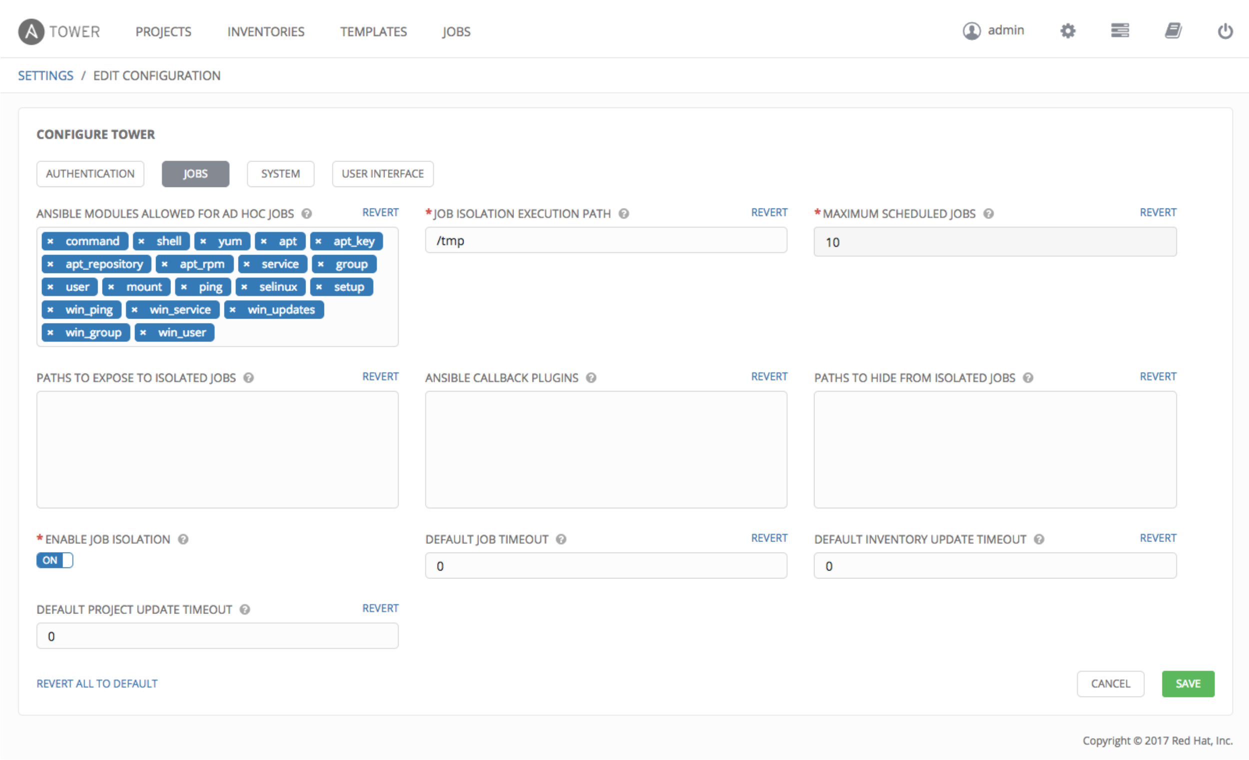
3. Click **Save** to apply the settings or **Cancel** to abandon the changes.

11.2 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 Settings (⚙) Menu screen, click on **Configure Tower**.
2. Select the **Jobs** tab.
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 value for **Default Job Timeout** is in seconds.



ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: 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

*JOB ISOLATION EXECUTION PATH: /tmp

*MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS:

ANSIBLE CALLBACK PLUGINS:

PATHS TO HIDE FROM ISOLATED JOBS:

*ENABLE JOB ISOLATION: ON

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

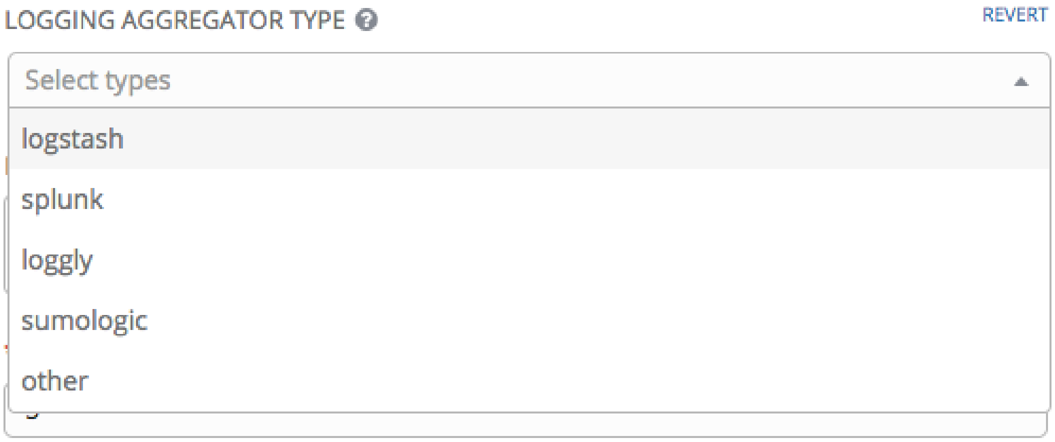
DEFAULT PROJECT UPDATE TIMEOUT: 0

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

11.3 System

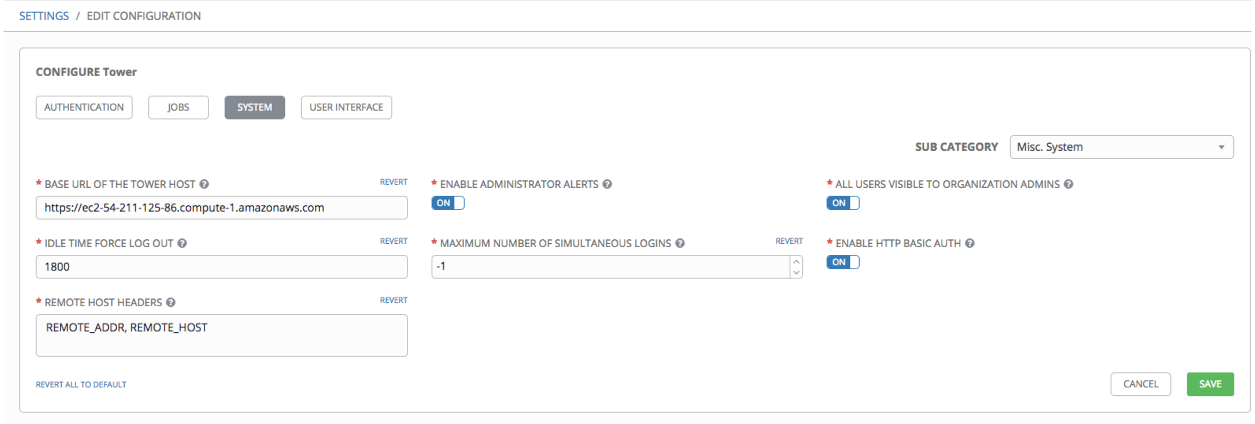
The System tab allows you to define the base URL for the Tower host, configure alerts, enable activity capturing, control visibility of users, enable certain Tower features and functionality through a license file, and configure logging aggregation options.

1. From the Settings (⚙) Menu screen, click on **Configure Tower**.
2. Select the **System** tab.
3. Select an option from the Sub Category drop-down menu list:
   - **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 Administration Guide.

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



SUB CATEGORY: Misc. System

* 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

* IDLE TIME FORCE LOG OUT: 1800

* MAXIMUM NUMBER OF SIMULTANEOUS LOGINS: -1

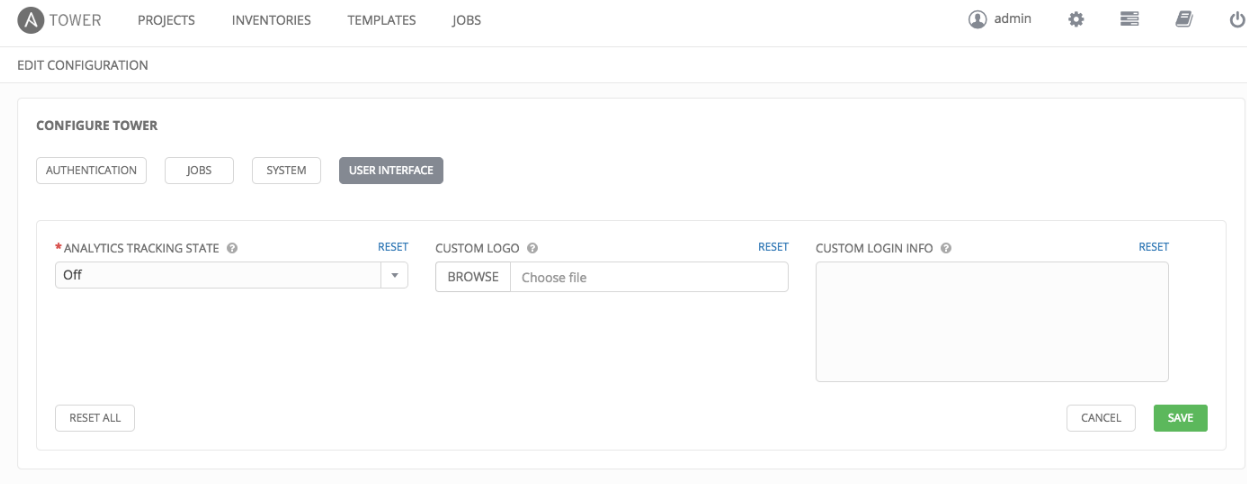
* ENABLE HTTP BASIC AUTH: ON

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

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

11.4 User Interface

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



* ANALYTICS TRACKING STATE: Off

CUSTOM LOGO

CUSTOM LOGIN INFO

11.4.1 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. From the Settings (⚙) Menu screen, click on **Configure Tower**.
2. Select the **User Interface** tab.
3. Select the desired level of data collection from the Analytics Tracking State drop-down list:

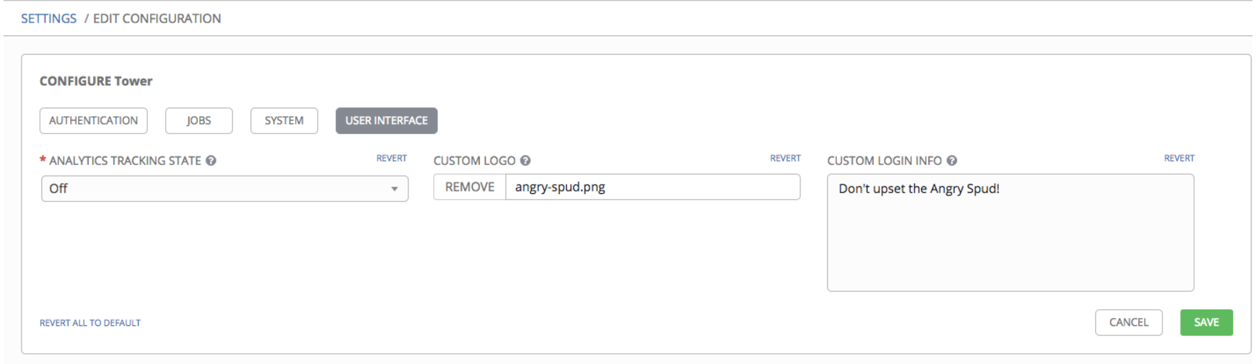
- **Off**: Prevents any data collection.
- **Anonymous**: Enables data collection without your specific user data.
- **Detailed**: Enables data collection including your specific user data.

4. 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.

11.4.2 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.


SETTINGS / EDIT CONFIGURATION

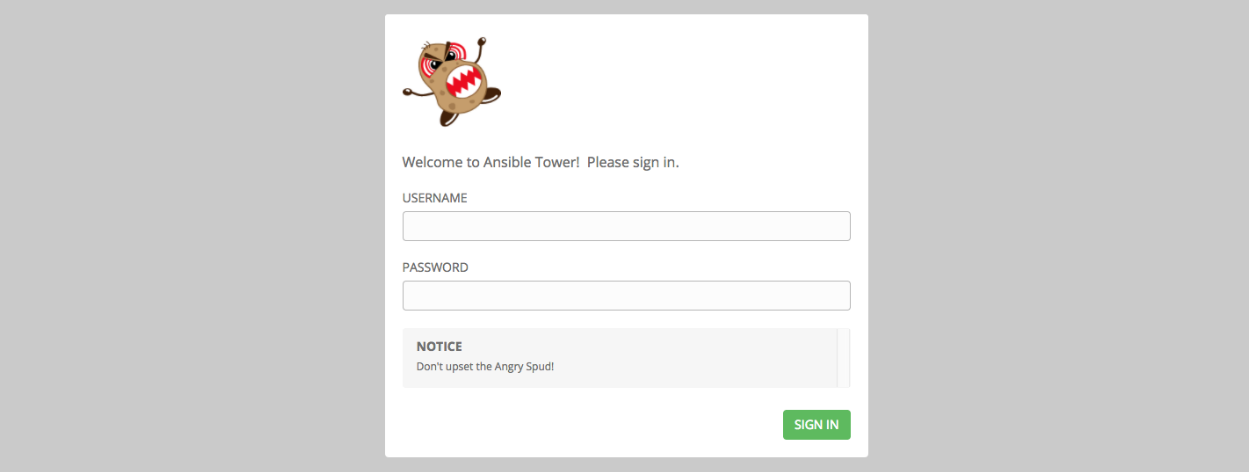
CONFIGURE Tower

* ANALYTICS TRACKING STATE: Off

CUSTOM LOGO: angry-spud.png

CUSTOM LOGIN INFO: Don't upset the Angry Spud!

Refer to the tooltips (?) for acceptable formats.


Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

NOTICE
Don't upset the Angry Spud!

CHAPTER

TWELVE

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.

Disabling bubblewrap support:

- To disable bubblewrap support for running jobs (playbook runs only), ensure you are logged in as the Admin user and click on the settings gear ⚙ in the upper right-hand corner. Click on the "Configure Tower" box, then click on the "Jobs" tab. Scroll down until you see "Enable Job Isolation" and change the radio button selection to "off".

Enabling bubblewrap support:

- To enable bubblewrap support for running jobs (playbook runs only), ensure you are logged in as the Admin user and click on the settings gear ⚙ in the upper right-hand corner. Click on the "Configure Tower" box, then click on the "Jobs" tab. Scroll down until you see "Enable Job Isolation" and change the radio button selection to "on".

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.

CHAPTER

THIRTEEN

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.

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.

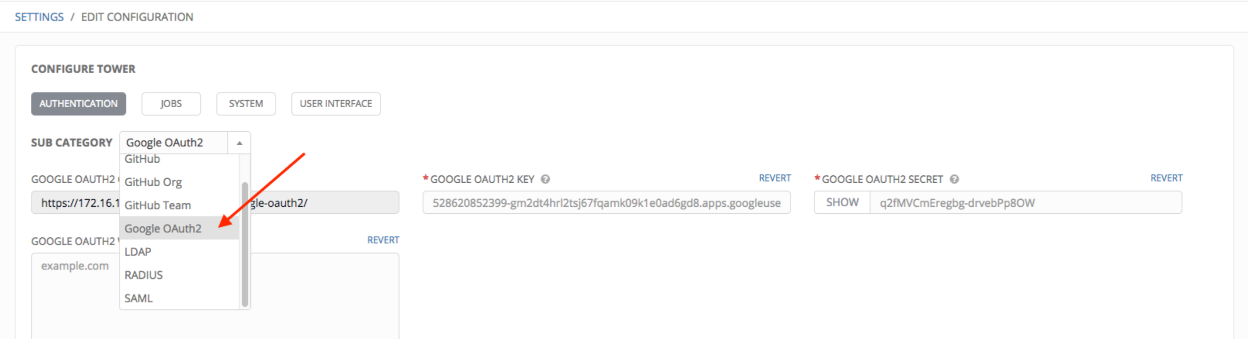
13.1 Google OAuth2 Settings

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.



SUB CATEGORY: Google OAuth2

GOOGLE OAUTH2 CALLBACK URL: https://172.16.1 … gle-oauth2/

* GOOGLE OAUTH2 KEY: 528620852399-gm2dt4hrl2tsj67fqamk09k1e0ad6gd8.apps.googleuse

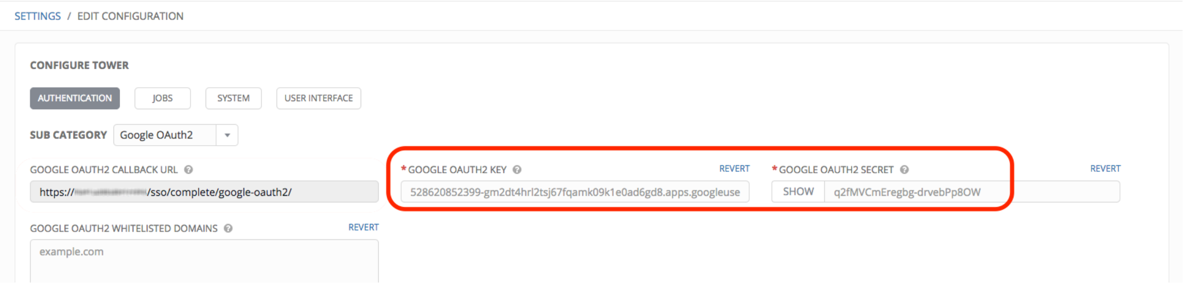
* GOOGLE OAUTH2 SECRET: q2fMVCmEregbg-drvebPp8OW

GOOGLE OAUTH2 WHITELISTED DOMAINS: example.com

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.



SUB CATEGORY: Google OAuth2

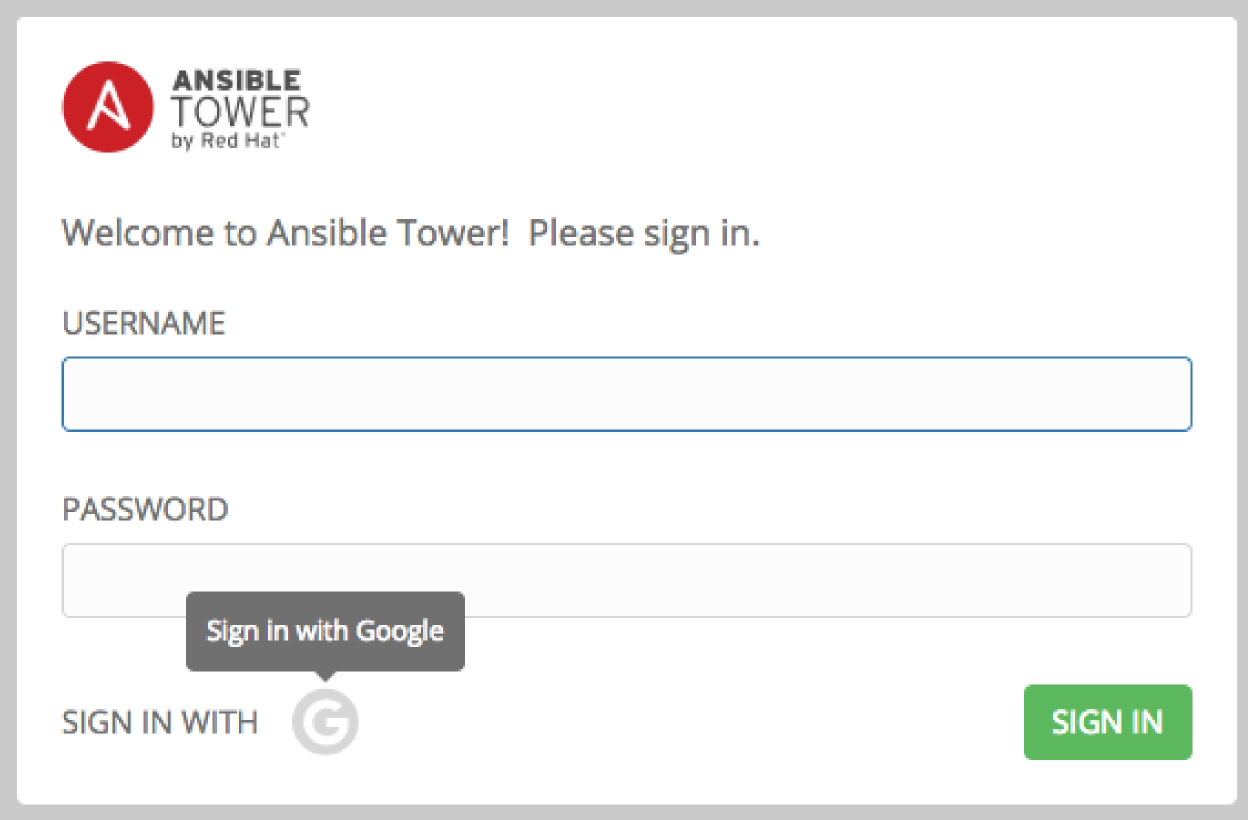
GOOGLE OAUTH2 CALLBACK URL: https://[redacted]/sso/complete/google-oauth2/

* GOOGLE OAUTH2 KEY: 528620852399-gm2dt4hrl2tsj67fqamk09k1e0ad6gd8.apps.googleuse

* GOOGLE OAUTH2 SECRET: q2fMVCmEregbg-drvebPp8OW

GOOGLE OAUTH2 WHITELISTED DOMAINS: example.com

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.



Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

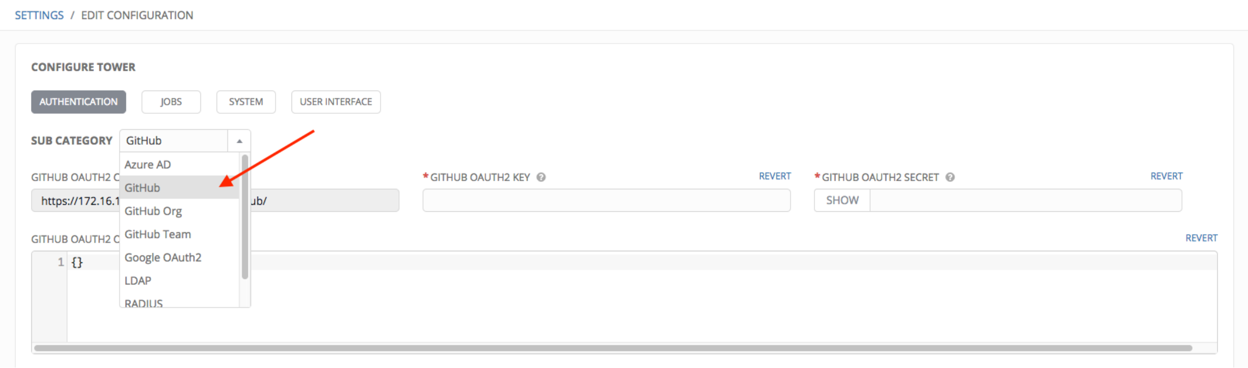
13.2 GitHub OAuth2 Settings

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.



SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

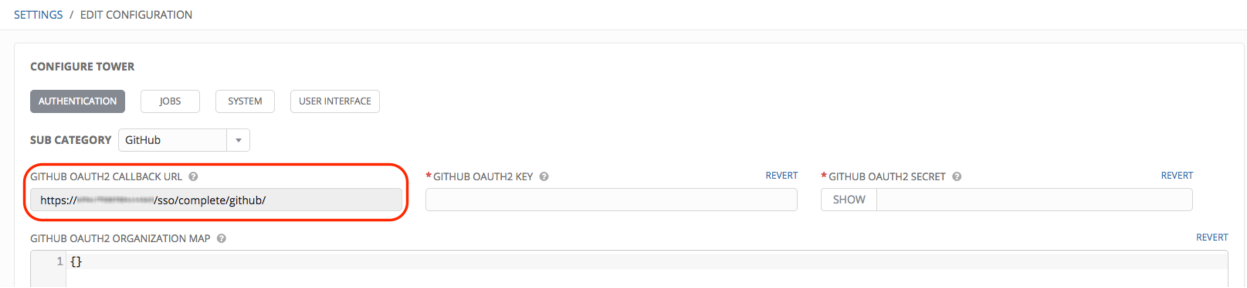
SUB CATEGORY: GitHub

GITHUB OAUTH2 C: https://172.16.1 ub/

GITHUB OAUTH2 C

```
1 {}
```

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



SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

SUB CATEGORY: GitHub

GITHUB OAUTH2 CALLBACK URL: https://[redacted]/sso/complete/github/

* GITHUB OAUTH2 KEY:

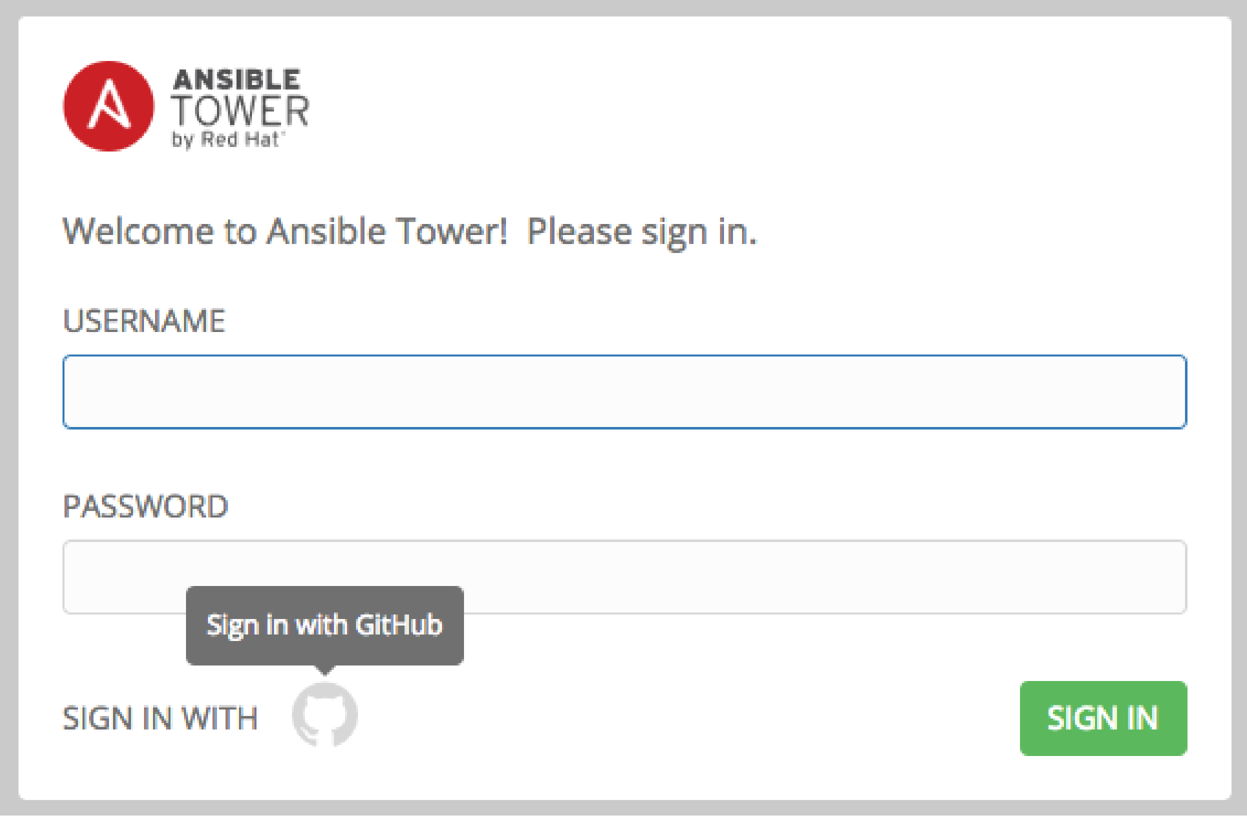
* GITHUB OAUTH2 SECRET:

GITHUB OAUTH2 ORGANIZATION MAP

```
{}
```

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.



Welcome to Ansible Tower! Please sign in.

USERNAME

PASSWORD

13.2.1 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).

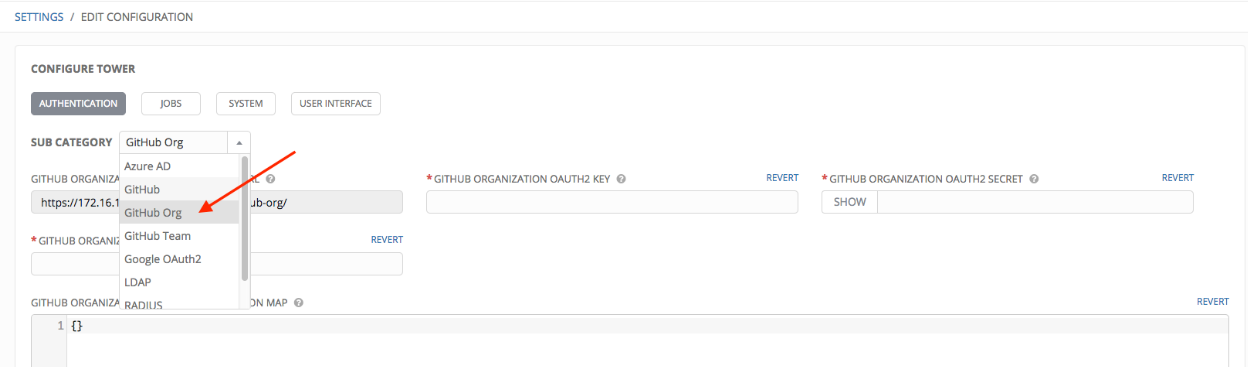
You can limit users who can login to Tower by limiting only those in an organization or on a team within an organization.

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

CONFIGURE TOWER

SUB CATEGORY: GitHub Org

GITHUB ORGANIZA... https://172.16.1...ub-org/

*GITHUB ORGANIZATION OAUTH2 KEY REVERT

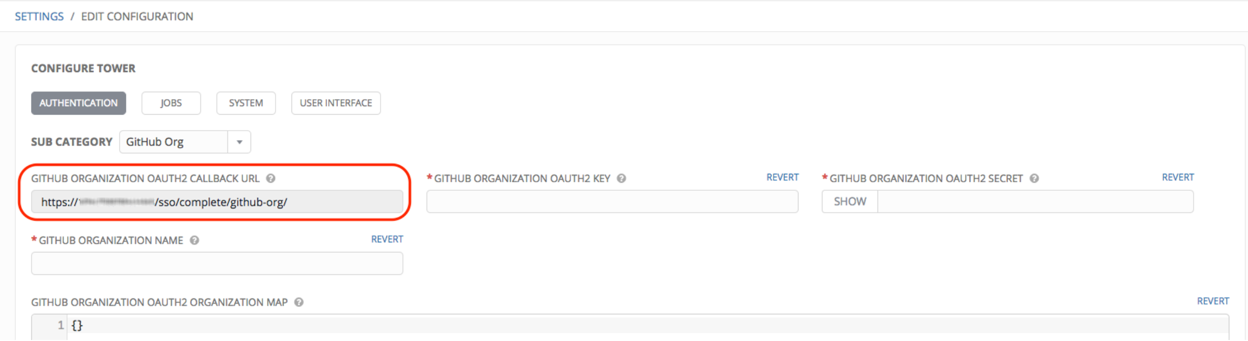
*GITHUB ORGANIZATION OAUTH2 SECRET REVERT SHOW

*GITHUB ORGANI... REVERT

GITHUB ORGANIZA...ON MAP REVERT

```
{}
```

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


SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

SUB CATEGORY: GitHub Org

GITHUB ORGANIZATION OAUTH2 CALLBACK URL: https://[redacted]/sso/complete/github-org/

*GITHUB ORGANIZATION OAUTH2 KEY REVERT

*GITHUB ORGANIZATION OAUTH2 SECRET REVERT SHOW

*GITHUB ORGANIZATION NAME REVERT

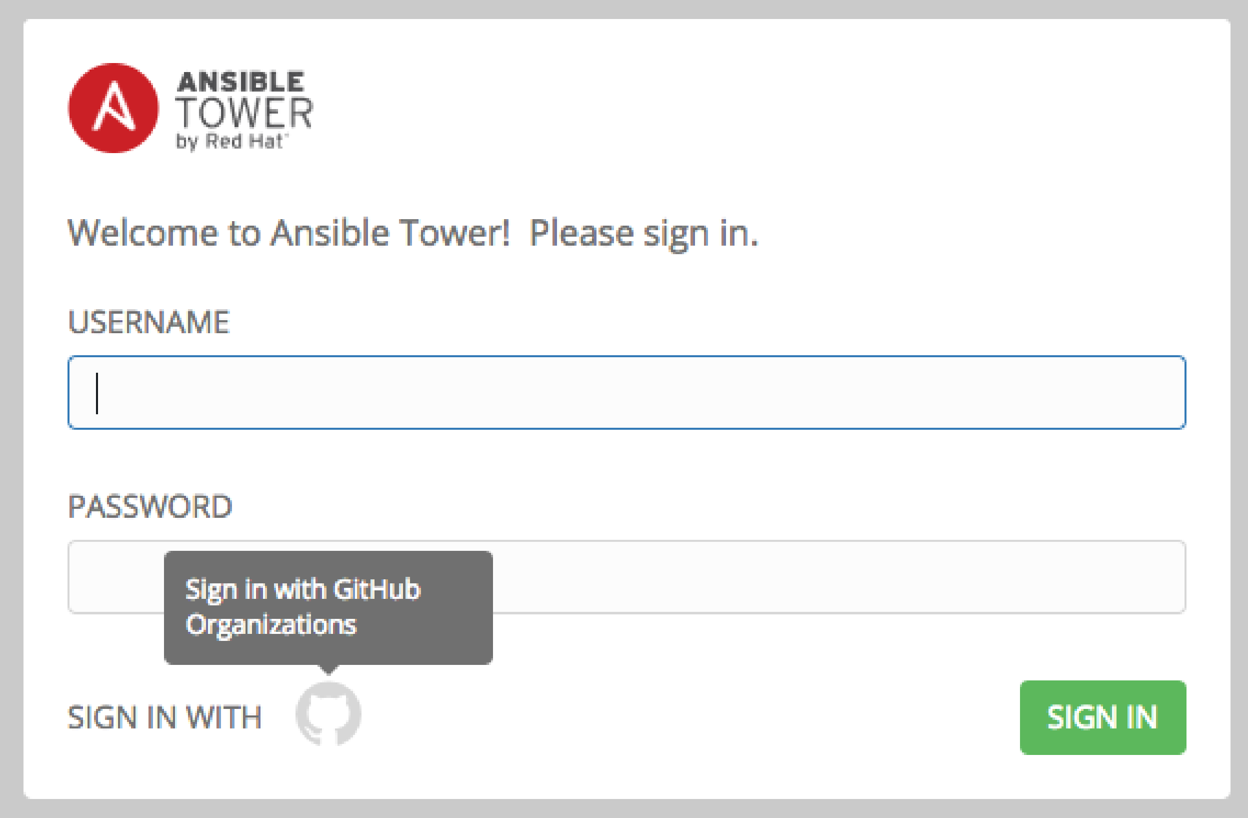
GITHUB ORGANIZATION OAUTH2 ORGANIZATION MAP REVERT

```
{}
```

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.


ANSIBLE TOWER by Red Hat
Welcome to Ansible Tower! Please sign in.
USERNAME
PASSWORD
Sign in with GitHub Organizations
SIGN IN WITH
SIGN IN

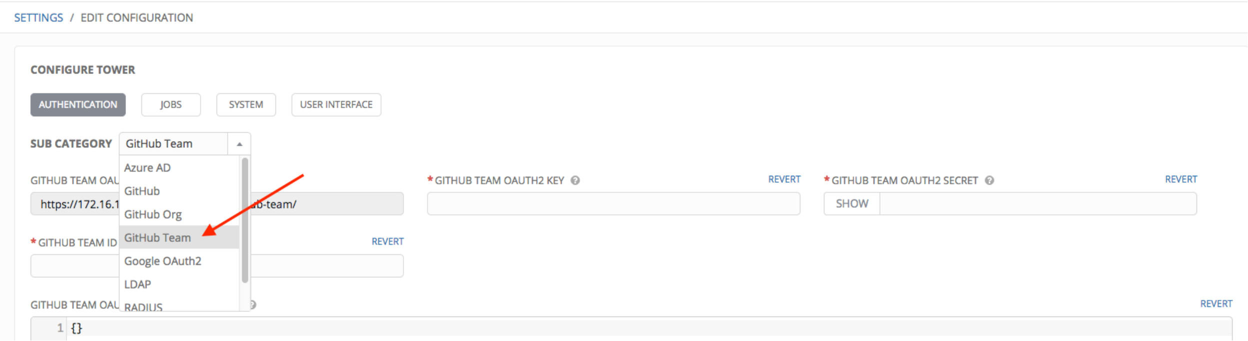
13.2.2 Github Team Settings

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.

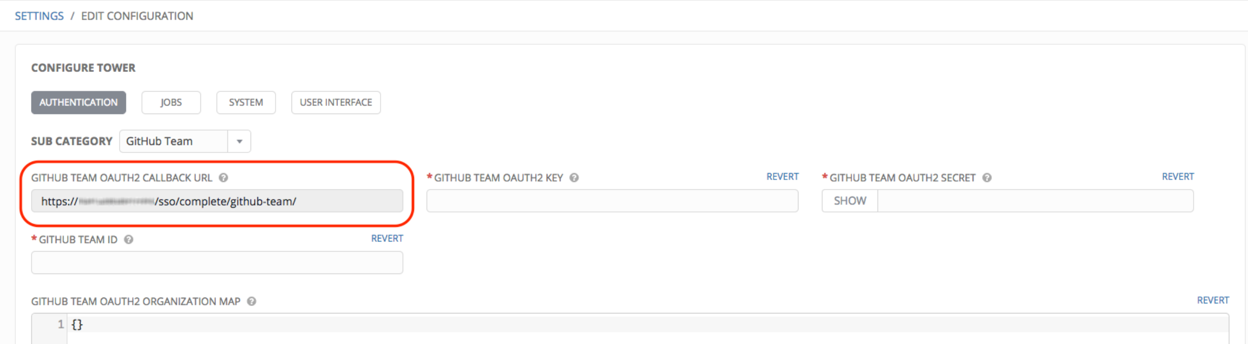
1. Find the numeric team ID using the Github API: http://fabian-kostadinov.github.io/2015/01/16/how-to-find-a-github-team-id/. The Team ID will be used to supply a required field in the Ansible Tower User Interface.
2. In the Ansible Tower User Interface, click **Configure Tower** from the Settings (⚙) 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.


SETTINGS / EDIT CONFIGURATION
CONFIGURE TOWER
AUTHENTICATION
JOBS
SYSTEM
USER INTERFACE
SUB CATEGORY
GitHub Team
GITHUB TEAM OAUTH2 CALLBACK URL
https://[redacted]/sso/complete/github-team/
* GITHUB TEAM OAUTH2 KEY
REVERT
* GITHUB TEAM OAUTH2 SECRET
REVERT
SHOW
* GITHUB TEAM ID
REVERT
GITHUB TEAM OAUTH2 ORGANIZATION MAP
REVERT
1 {}

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.

13.3 Organization and Team Mapping

Next, 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**.

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

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 = {}
```

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.

```
SOCIAL_AUTH_TEAM_MAP = {
"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 = []
```

CHAPTER

FOURTEEN

SETTING UP ENTERPRISE AUTHENTICATION

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.

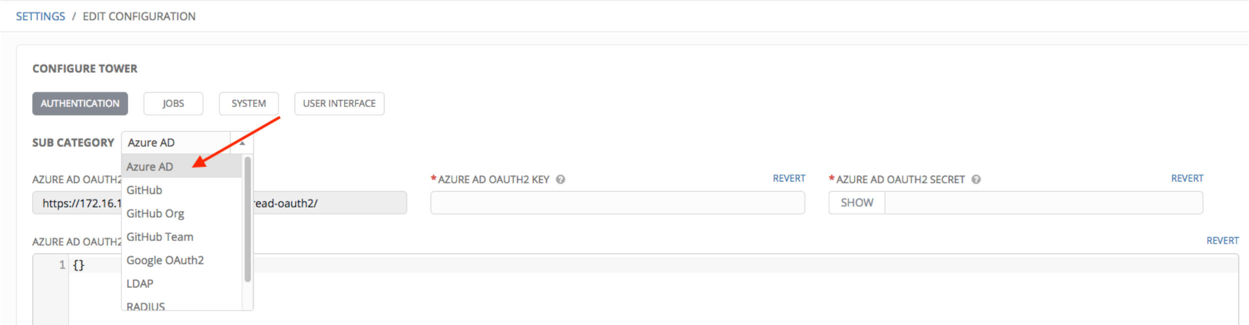
14.1 Azure Active Directory (AD)

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 **Configure Tower** from the Settings (⚙) Menu screen.

The Authentication tab displays initially by default.

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


SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

SUB CATEGORY: Azure AD

AZURE AD OAUTH2 … https://172.16.1…read-oauth2/

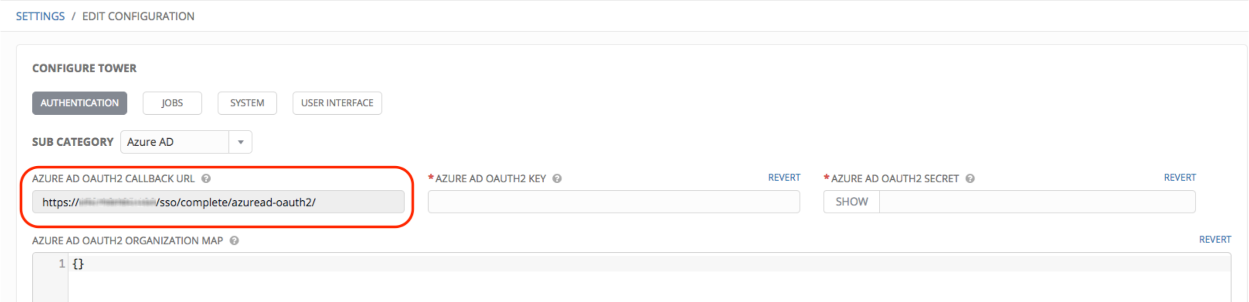
*AZURE AD OAUTH2 KEY REVERT

*AZURE AD OAUTH2 SECRET REVERT SHOW

AZURE AD OAUTH2 … REVERT

```
1 {}
```

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


SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

SUB CATEGORY: Azure AD

AZURE AD OAUTH2 CALLBACK URL: https://[redacted]/sso/complete/azuread-oauth2/

*AZURE AD OAUTH2 KEY REVERT

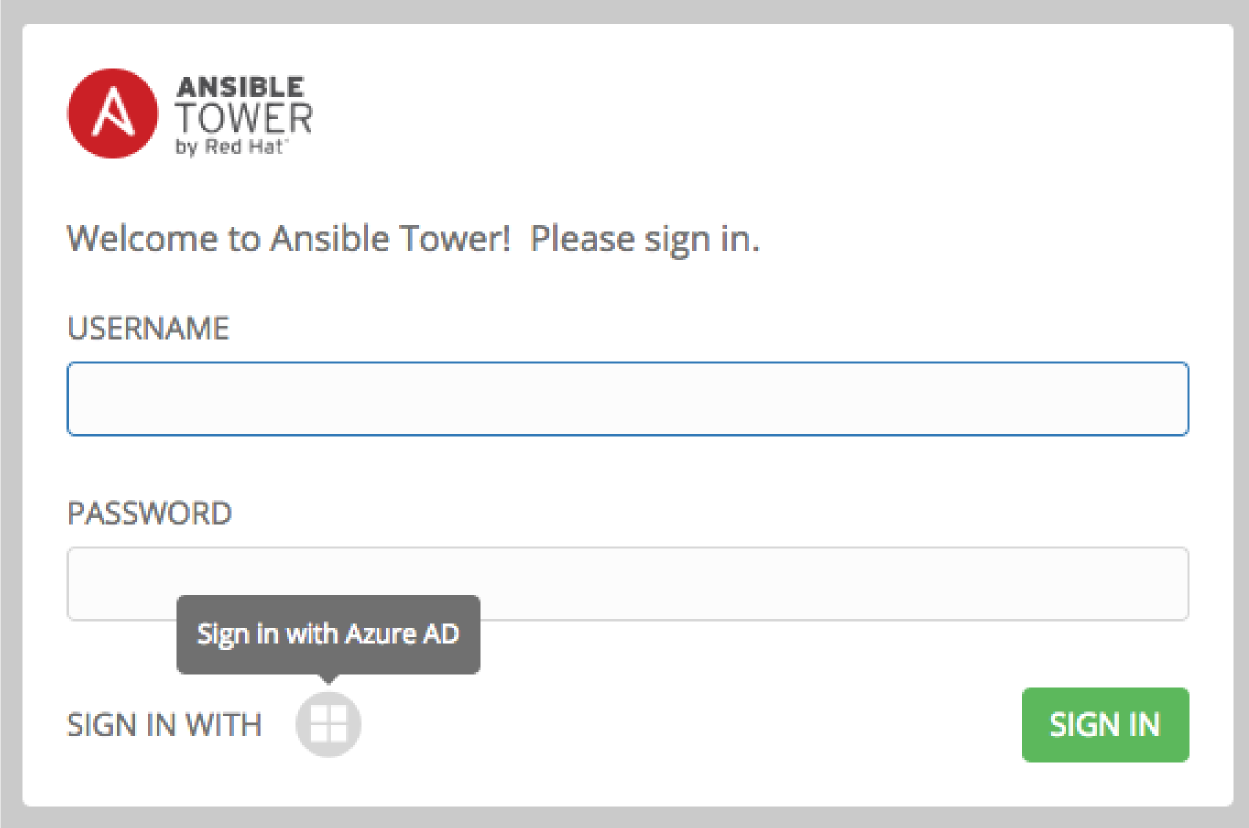
*AZURE AD OAUTH2 SECRET REVERT SHOW

AZURE AD OAUTH2 ORGANIZATION MAP REVERT

```
1 {}
```

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.
5. Copy and paste Azure's Object ID 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.

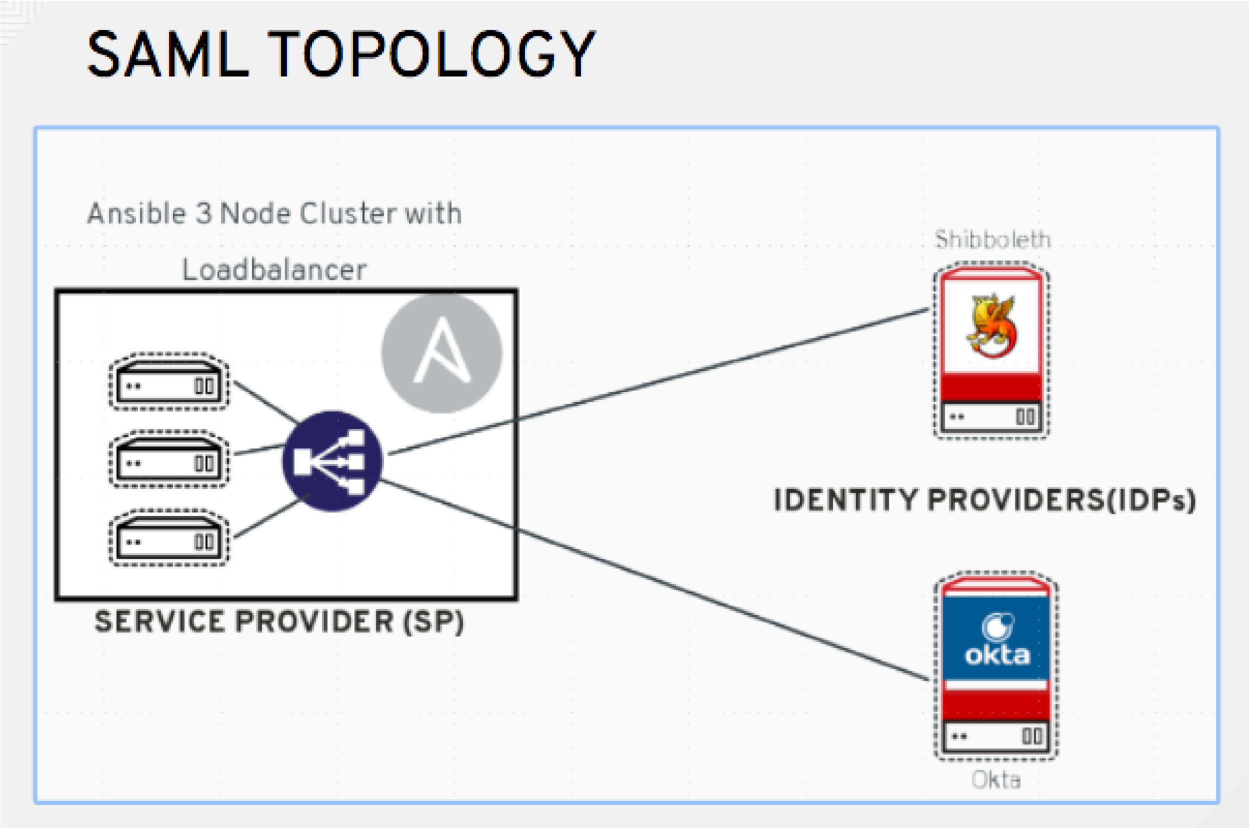


For application registering basics in Azure AD, refer to: https://docs.microsoft.com/en-us/azure/active-directory/develop/active-directory-authentication-scenarios#basics-of-registering-an-application-in-azure-ad

14.2 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).

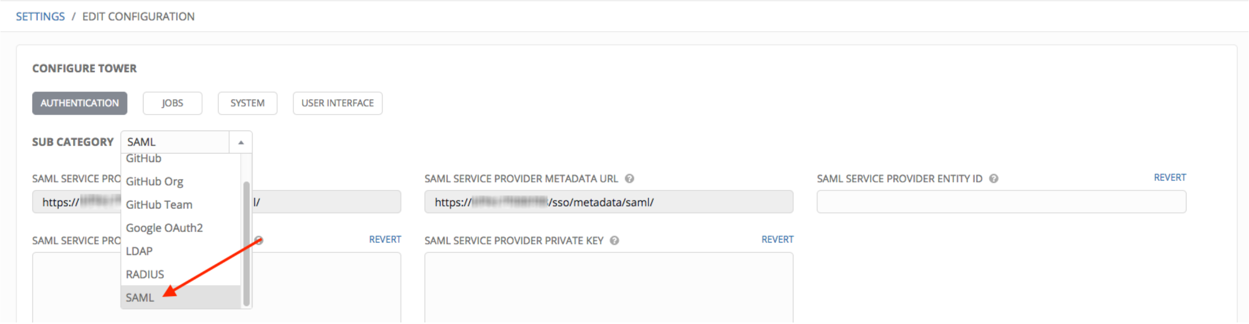


To setup SAML authentication:

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 **SAML** from the drop-down list.



SUB CATEGORY: SAML

SAML SERVICE PRO… : https://[redacted] l/

SAML SERVICE PROVIDER METADATA URL: https://[redacted]/sso/metadata/saml/

SAML SERVICE PROVIDER ENTITY ID

SAML SERVICE PRO…

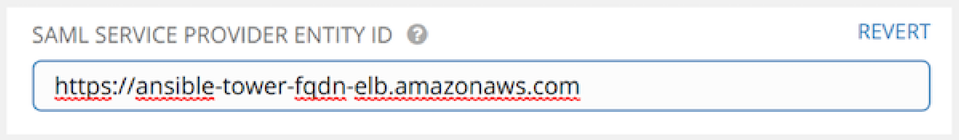
SAML SERVICE PROVIDER PRIVATE KEY

The **SAML Service Provider Callback URL** and **SAML Service Provider Metadata URL** fields are pre-populated and are non-editable.

3. 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 ⚙ menu. 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.

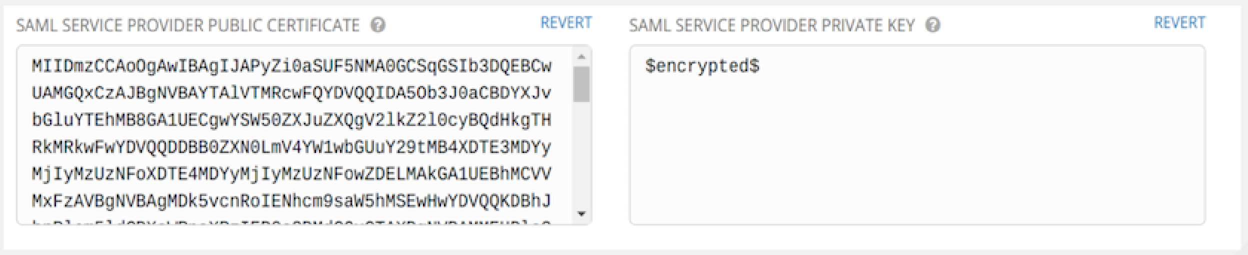
Note: The Tower Base URL is different for each node in a cluster. Commonly, a load balancer will sit in front of many tower cluster nodes to provide a single entry point, Tower Cluster FQDN. The SAML Service Provider must be able establish an outbound connection and route to the Tower Cluster Node or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



https://ansible-tower-fqdn-elb.amazonaws.com

4. 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.


SAML SERVICE PROVIDER PUBLIC CERTIFICATE
REVERT
SAML SERVICE PROVIDER PRIVATE KEY
REVERT

```
MIIDmzCCAoOgAwIBAgIJAPyZi0aSUF5NMA0GCSqGSIb3DQEBCw
UAMGQxCzAJBgNVBAYTAlVTMRcwFQYDVQQIDA5Ob3J0aCBDYXJv
bGluYTEhMB8GA1UECgwYSW50ZXJuZXQgV2lkZ2l0cyBQdHkgTH
RkMRkwFwYDVQQDDBB0ZXN0LmV4YW1wbGUuY29tMB4XDTE3MDYy
MjIyMzUzNFoXDTE4MDYyMjIyMzUzNFowZDELMAkGA1UEBhMCVV
MxFzAVBgNVBAgMDk5vcnRoIENhcm9saW5hMSEwHwYDVQQKDBhJ
```

```
$encrypted$
```

As an example for public certs:

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

5. 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----
```

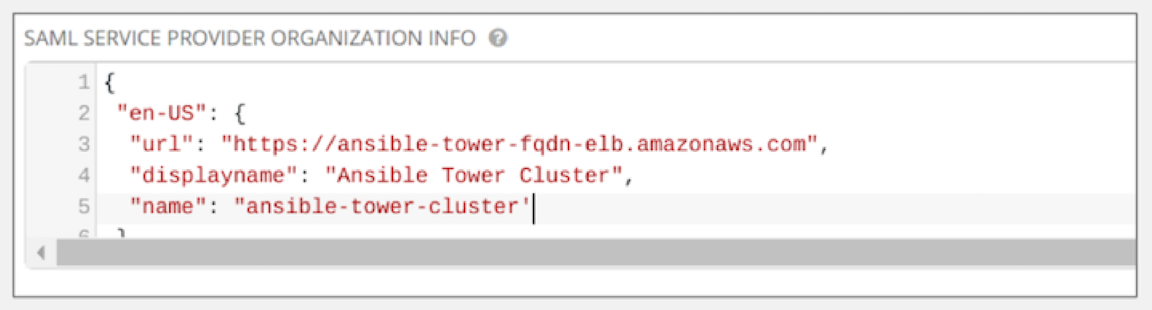
6. Contact the Identity Provider administrator and provide the following information:

- SAML Assertion Consumer Services URL. This is the Tower Callback URL auto-generated by Tower and is shown in the **SAML Service Provider Callback URL** field. Ensure that the Base URL matches the FQDN of the load balancer (if used).
- Service Provider Entity ID. This is the URL entered in the **SAML Service Provider Entity ID** field.

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:

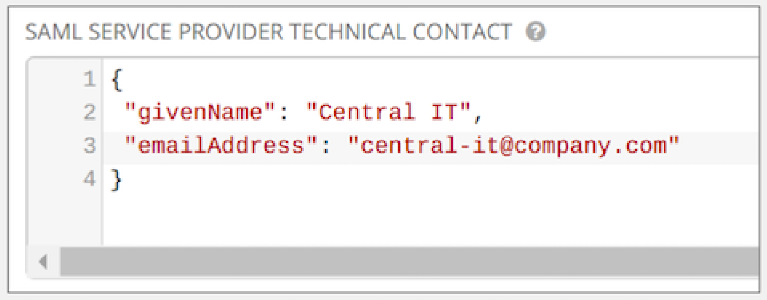


```
{
 "en-US": {
  "url": "https://ansible-tower-fqdn-elb.amazonaws.com",
  "displayname": "Ansible Tower Cluster",
  "name": "ansible-tower-cluster'
 }
```

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:

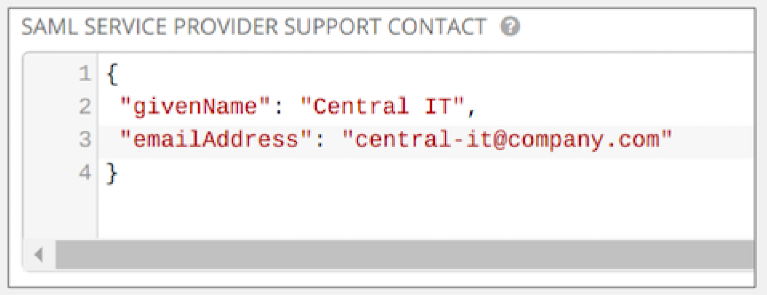


```
{
 "givenName": "Central IT",
 "emailAddress": "central-it@company.com"
}
```

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:



```
{
 "givenName": "Central IT",
 "emailAddress": "central-it@company.com"
}
```

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)
```

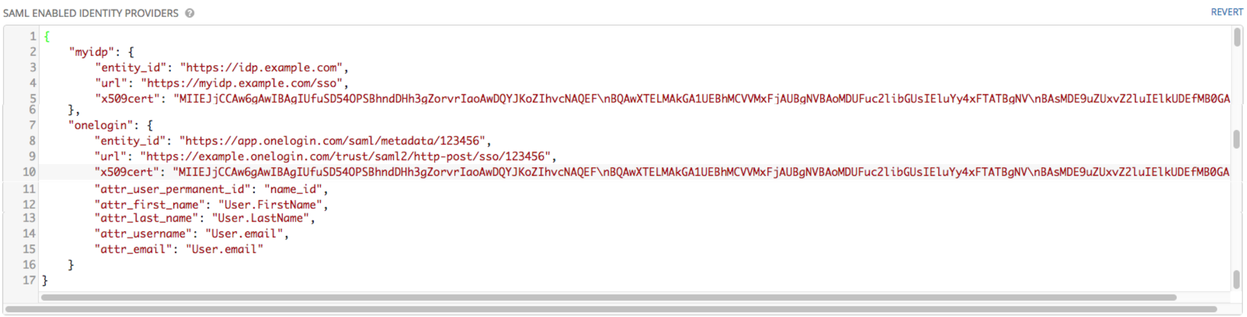
If these attributes are not known, map existing SAML attributes to lastname, firstname, email and username.

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.

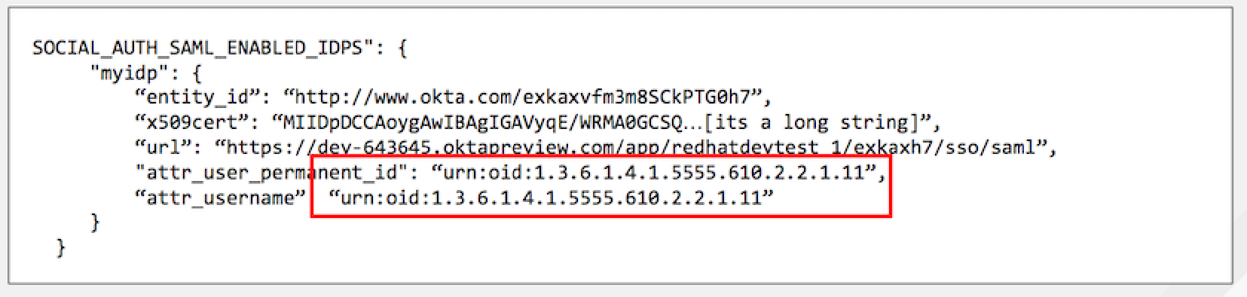
Multiple SAML IdPs are supported. Some IdPs may provide user data using attribute names that differ from the default OIDs (https://github.com/omab/python-social-auth/blob/master/social/backends/saml.py). The SAML `NameID` is a special attribute used by some Identity Providers to tell the Service Provider (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"
  }
}
```



```
{
    "myidp": {
        "entity_id": "https://idp.example.com",
        "url": "https://myidp.example.com/sso",
        "x509cert": "MIIEJjCCAw6gAwIBAgIUfuSD54OPSBhndDHh3gZorvrIaoAwDQYJKoZIhvcNAQEF\nBQAwXTELMAkGA1UEBhMCVVMxFjAUBgNVBAoMDUFuc2libGUsIEluYy4xFTATBgNV\nBAsMDE9uZUxvZ2luIElkUDEfMB0GA
    },
    "onelogin": {
        "entity_id": "https://app.onelogin.com/saml/metadata/123456",
        "url": "https://example.onelogin.com/trust/saml2/http-post/sso/123456",
        "x509cert": "MIIEJjCCAw6gAwIBAgIUfuSD54OPSBhndDHh3gZorvrIaoAwDQYJKoZIhvcNAQEF\nBQAwXTELMAkGA1UEBhMCVVMxFjAUBgNVBAoMDUFuc2libGUsIEluYy4xFTATBgNV\nBAsMDE9uZUxvZ2luIElkUDEfMB0GA
        "attr_user_permanent_id": "name_id",
        "attr_first_name": "User.FirstName",
        "attr_last_name": "User.LastName",
        "attr_username": "User.email",
        "attr_email": "User.email"
    }
}
```

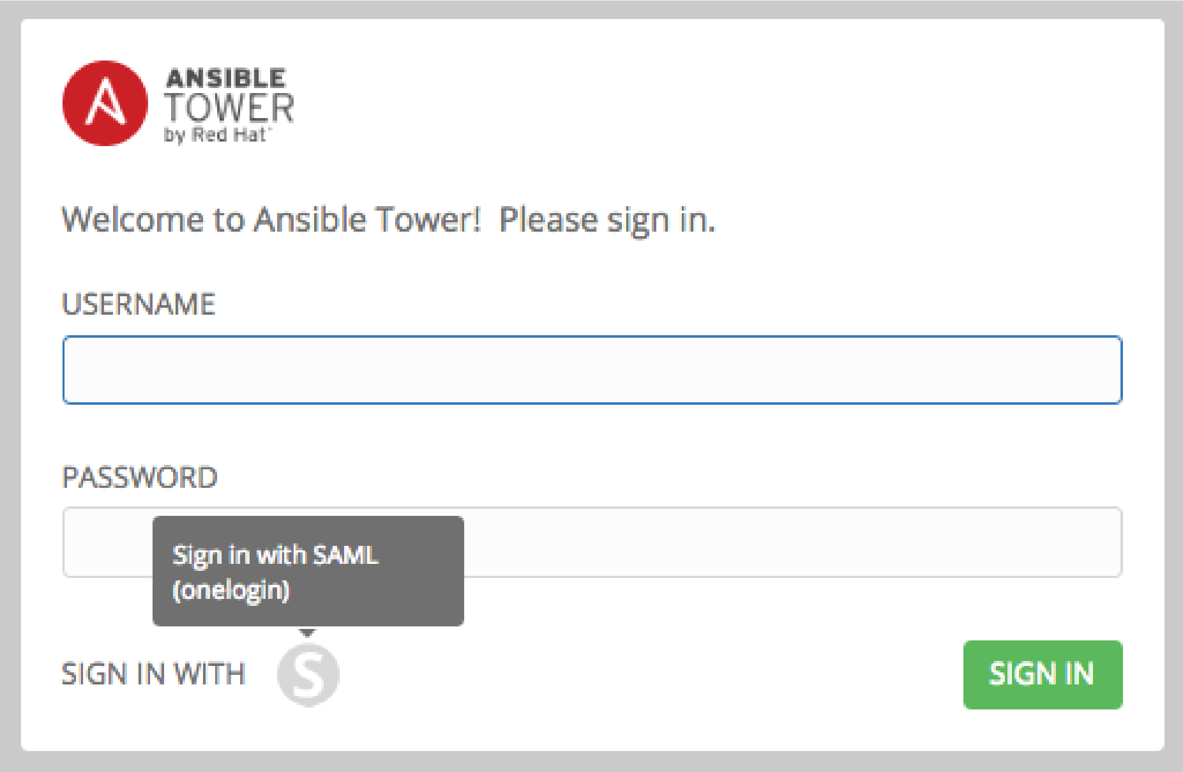
Note: The IdP provides the email, last name and firstname using the well known SAML urn. The IdP uses a custom SAML attribute to identify a user, which is an attribute that Tower is unable to read. Instead, Tower can understand the unique identifier name, which is the URN. Use the URN listed in the SAML "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. For details on completing the mapping fields, see *Organization and Team Mapping*.
12. Click **Save** when done.
13. 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.



Welcome to Ansible Tower! Please sign in.

14.3 RADIUS Authentication Settings

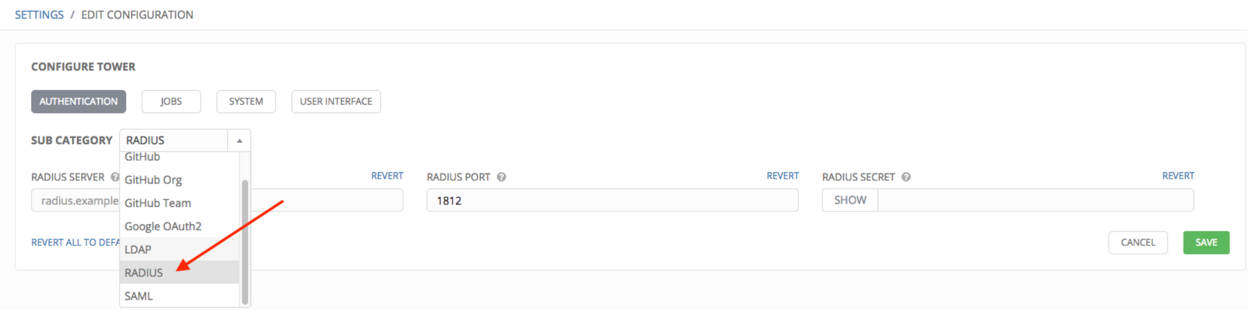
Note: RADIUS account authentication is a feature specific to Enterprise-level license holders.

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

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

The Authentication tab displays initially by default.

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



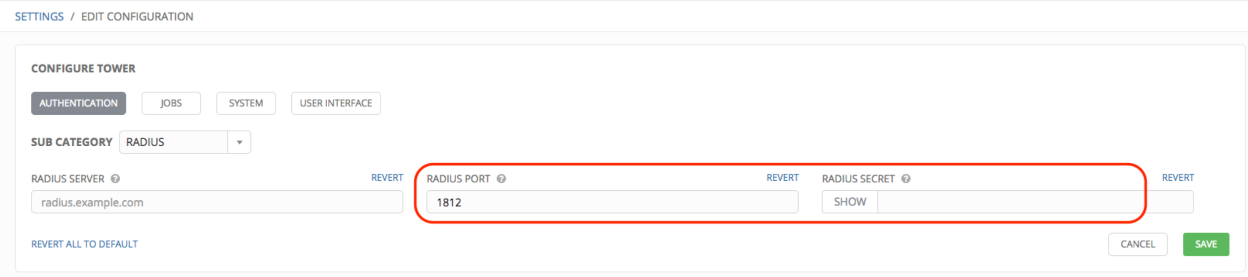
SUB CATEGORY: RADIUS

RADIUS SERVER: radius.example

RADIUS PORT: 1812

RADIUS SECRET

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.



SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

SUB CATEGORY: RADIUS

RADIUS SERVER: radius.example.com

RADIUS PORT: 1812

RADIUS SECRET: SHOW

5. Click **Save** when done.

14.4 TACACS+ Authentication Settings

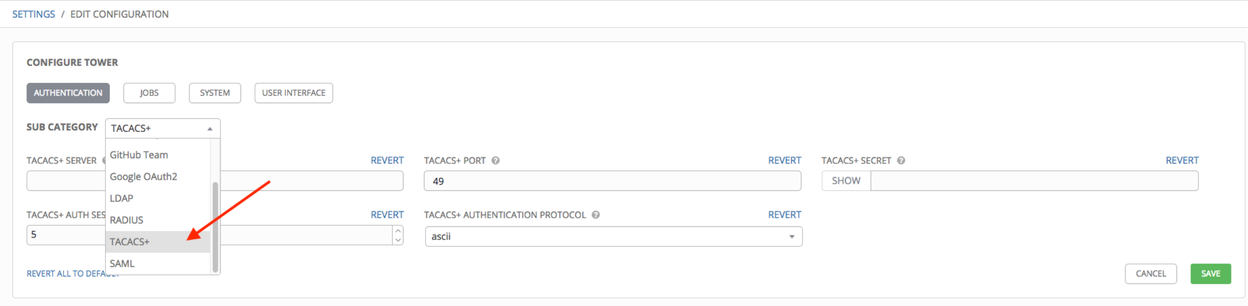
Note: TACACS+ account authentication is a feature specific to Enterprise-level license holders.

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.



SETTINGS / EDIT CONFIGURATION

CONFIGURE TOWER

SUB CATEGORY: TACACS+

TACACS+ SERVER:

TACACS+ PORT: 49

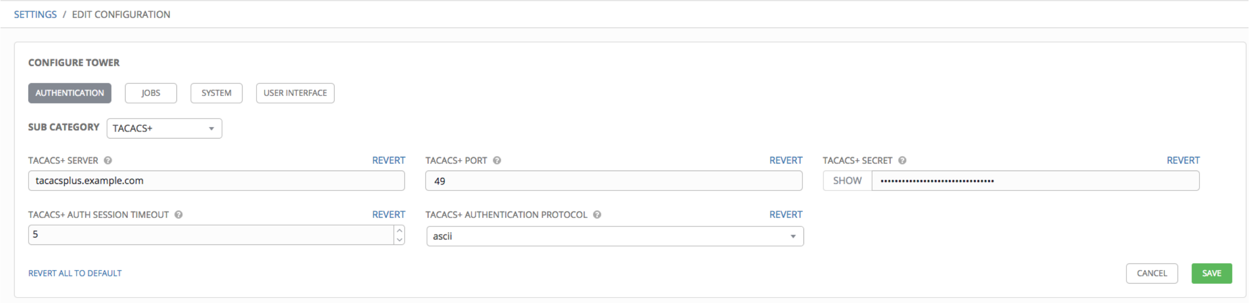
TACACS+ SECRET: SHOW

TACACS+ AUTH SES: 5

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**.



SUB CATEGORY: TACACS+

TACACS+ SERVER: tacacsplus.example.com

TACACS+ PORT: 49

TACACS+ SECRET: ••••••••••••••••••••••••••••••••

TACACS+ AUTH SESSION TIMEOUT: 5

TACACS+ AUTHENTICATION PROTOCOL: ascii

4. Click **Save** when done.

CHAPTER FIFTEEN

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
```

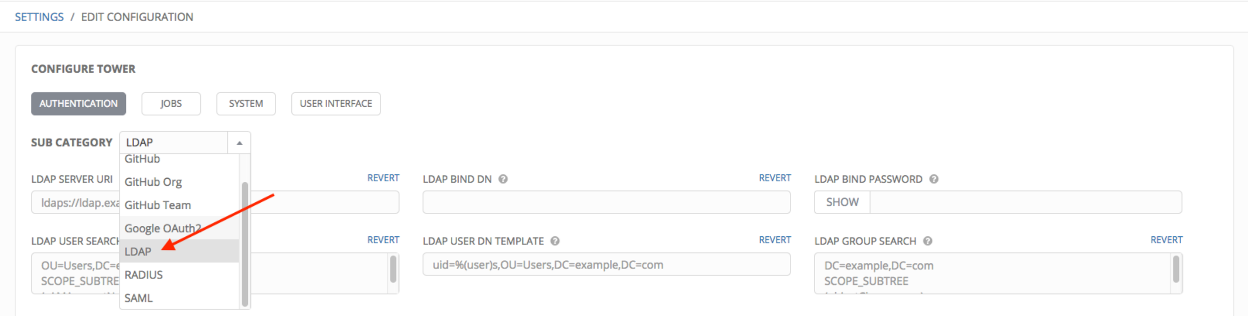
Here `CN=josie,CN=users,DC=website,DC=com` is the Distinguished Name of the connecting user.

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

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

The Authentication tab displays initially by default.

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



SUB CATEGORY: LDAP

LDAP SERVER URI: ldaps://ldap.exa

LDAP BIND DN

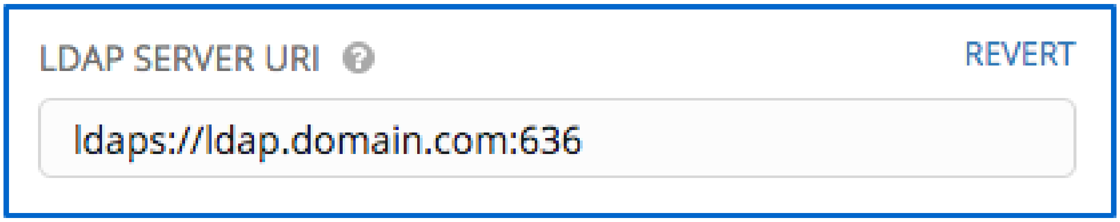
LDAP BIND PASSWORD

LDAP USER SEARCH: OU=Users,DC=e SCOPE_SUBTRE

LDAP USER DN TEMPLATE: uid=%(user)s,OU=Users,DC=example,DC=com

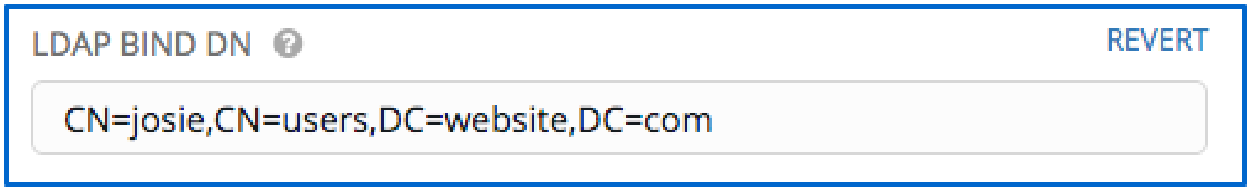
LDAP GROUP SEARCH: DC=example,DC=com SCOPE_SUBTREE

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:



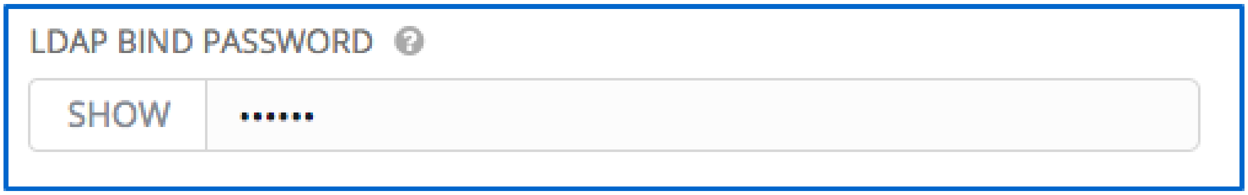
LDAP SERVER URI: ldaps://ldap.domain.com:636

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: 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: ••••••

8. 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:

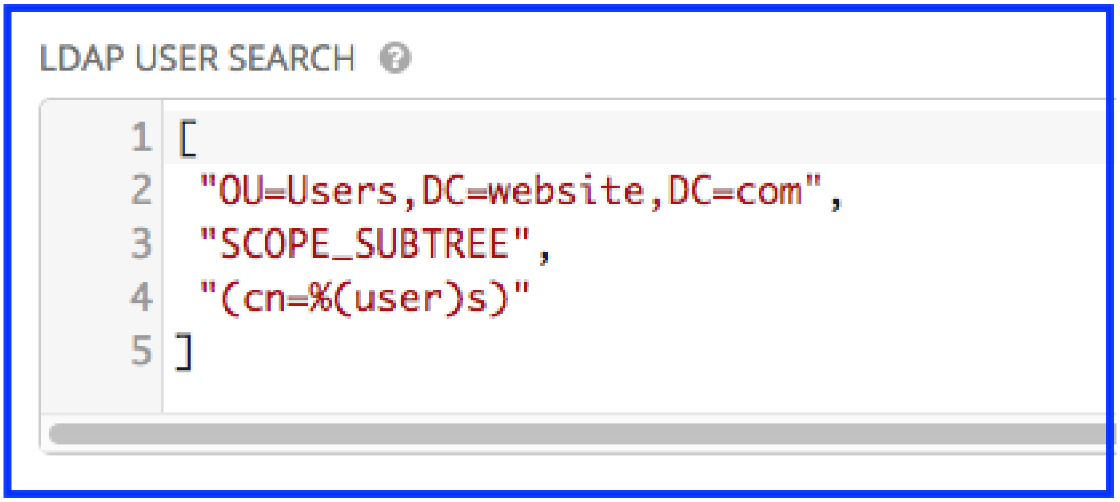
```
[
"OU=Users,DC=website,DC=com",
"SCOPE_SUBTREE",
"(cn=%(user)s)"
]
```

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:

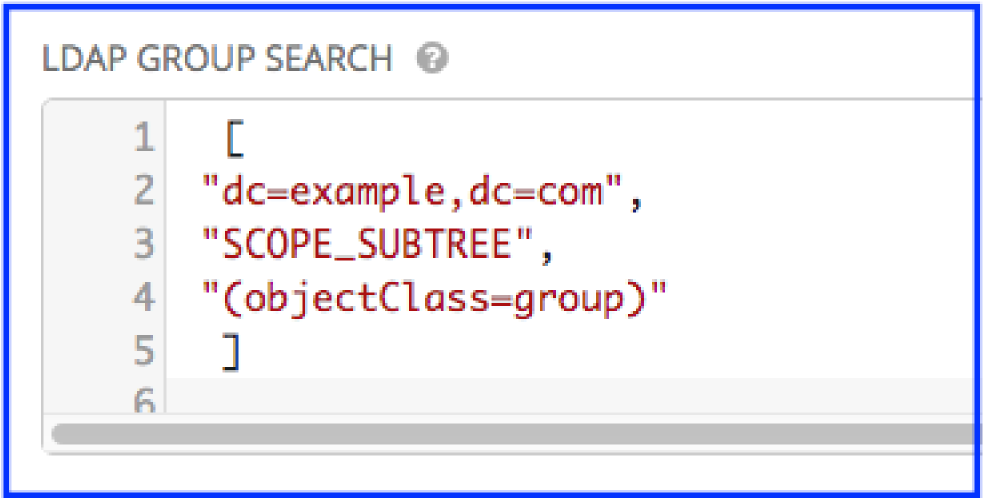
```
[
  [
  "OU=Users,DC=northamerica,DC=acme,DC=com",
  "SCOPE_SUBTREE",
  "(sAMAccountName=%(user)s)"
  ],
  [
  "OU=Users,DC=apac,DC=corp,DC=com",
  "SCOPE_SUBTREE",
  "(sAMAccountName=%(user)s)"
  ],
  [
  "OU=Users,DC=emea,DC=corp,DC=com",
  "SCOPE_SUBTREE",
  "(sAMAccountName=%(user)s)"
  ]
]
```

9. 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)`.

10. 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)"
 ]
```

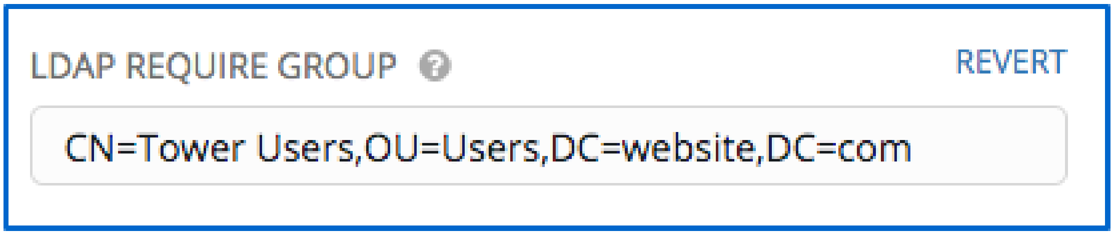
- The first line specifies the BASE DN where the groups should be searched.
- The second lines specifies the scope and is the same as that for the user directive.
- The third line specifies what the `objectclass` of a group object is in the LDAP you are using.



LDAP GROUP SEARCH

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

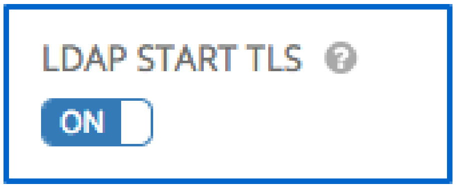
11. Click to select a group type from the **LDAP Group Type** drop-down menu list.
12. Enter the group distinguish name to allow users within that group to access Tower in the **LDAP Require Group** field, using the same format as the one shown in the text field. In this example, use: `CN=Tower Users, OU=Users,DC=website,DC=com`



LDAP REQUIRE GROUP

CN=Tower Users,OU=Users,DC=website,DC=com

13. 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.
14. The **LDAP Start TLS** is disabled by default. To enable TLS when the LDAP connection is not using SSL, click the toggle to **ON**.



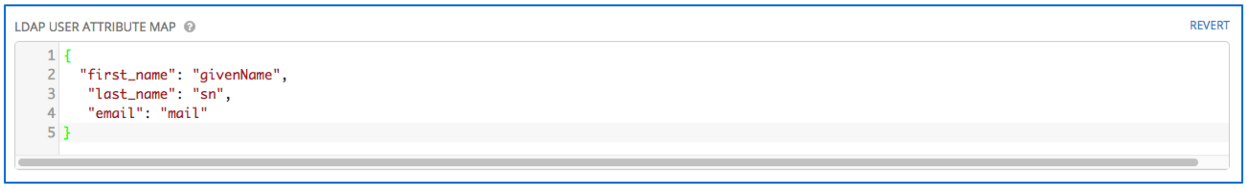
LDAP START TLS

ON

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.

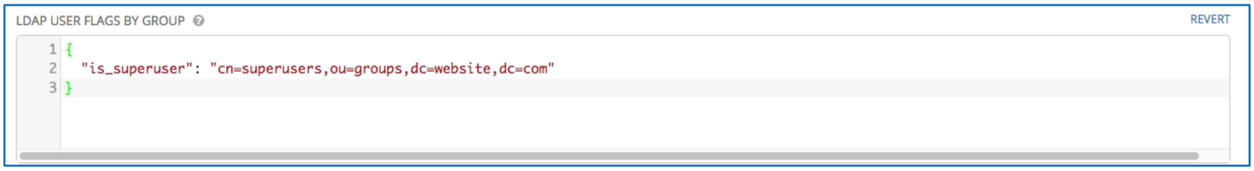


```
{
  "first_name": "givenName",
   "last_name": "sn",
   "email": "mail"
}
```

16. Enter the user profile flags in the **LDAP User Flags by Group** the text field. In this example, use:

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

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



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

17. For details on completing the mapping fields, see *LDAP 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.

15.1 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/ldap.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 Mapping*.

15.2 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/ldap.py`:

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

15.3 LDAP Mapping

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

15.3.1 Organization Mapping

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 organiation.
- 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 an 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
}
}
```

15.3.2 Team Mapping

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
}
}
```

CHAPTER

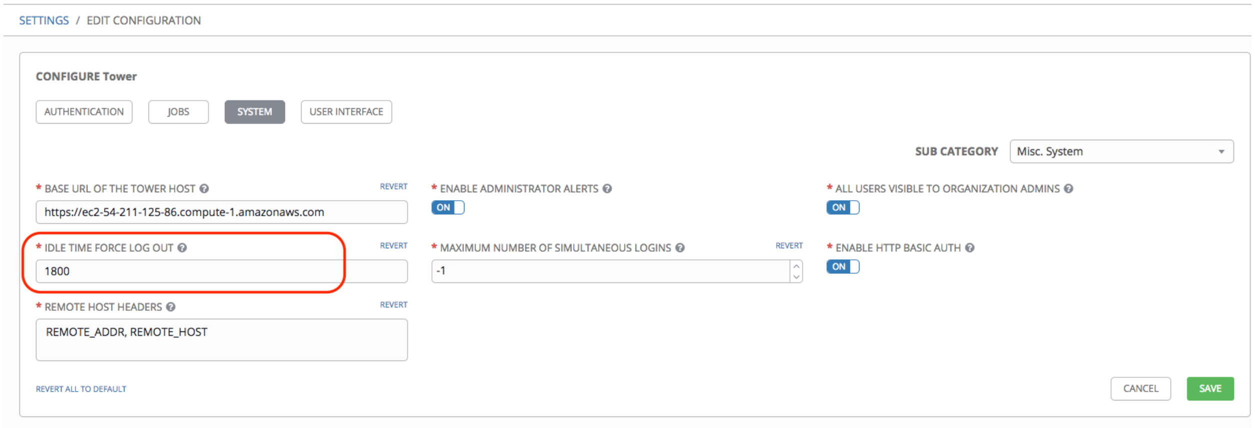
SIXTEEN

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. The value of `Auth-Token-Timeout` is determined by the configuration (time expressed in seconds) of the `AUTH_TOKEN_EXPIRATION`.

The value of `Auth-Token-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.



SUB CATEGORY: Misc. System

* BASE URL OF THE TOWER HOST: https://ec2-54-211-125-86.compute-1.amazonaws.com

* IDLE TIME FORCE LOG OUT: 1800

* REMOTE HOST HEADERS: REMOTE_ADDR, REMOTE_HOST

* ENABLE ADMINISTRATOR ALERTS: ON

* MAXIMUM NUMBER OF SIMULTANEOUS LOGINS: -1

* ALL USERS VISIBLE TO ORGANIZATION ADMINS: ON

* ENABLE HTTP BASIC AUTH: ON

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.

CHAPTER

SEVENTEEN

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.

[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.

17.1 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.

17.2 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 krbv5 kinit-compatible binary.

CHAPTER

EIGHTEEN

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.
AUTH_TOKEN_EXPIRATION = 1800

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

CHAPTER

NINETEEN

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, always use the most recent minor version of a release to backup and be sure to restore to the same version from which it was backed up.

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.py -i <inventory file>
```

19.1 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.

19.2 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.

19.3 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

19.3.1 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.

CHAPTER

TWENTY

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.

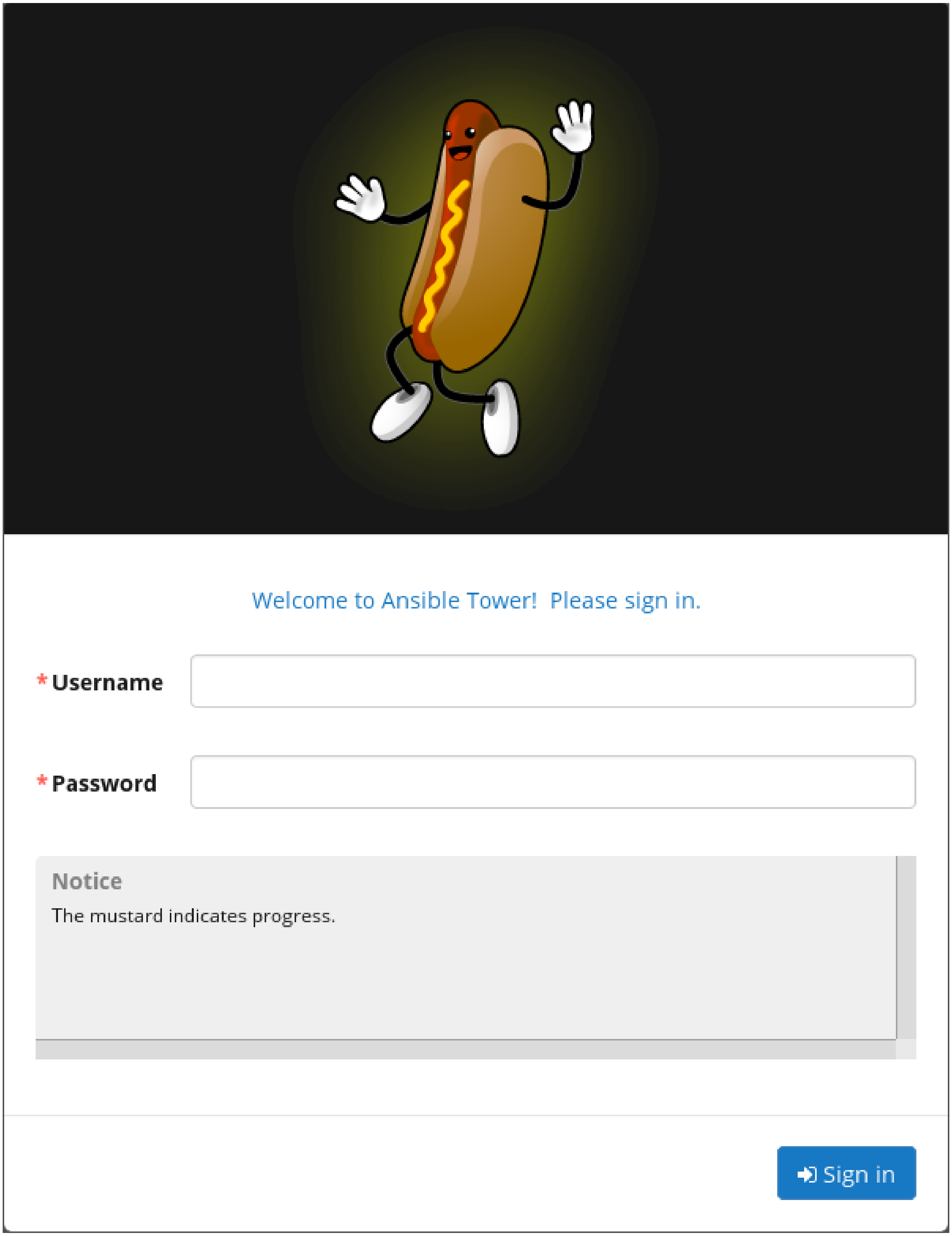
Selecting `Revert` will result in the appearance of the standard Ansible Tower logo.

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.

Notice

The mustard indicates progress.

CHAPTER

TWENTYONE

TROUBLESHOOTING TOWER

21.1 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/.

21.2 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.)

21.3 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.

21.4 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 `yamlint` 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.

21.5 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`.

21.6 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/
```

21.7 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.

21.8 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.

21.9 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.

21.9.1 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.

Disabling bubblewrap support:

- To disable bubblewrap support for running jobs (playbook runs only), ensure you are logged in as the Admin user and click on the settings gear ⚙ in the upper right-hand corner. Click on the "Configure Tower" box, then click on the "Jobs" tab. Scroll down until you see "Enable Job Isolation" and change the radio button selection to "off".

Enabling bubblewrap support:

- To enable bubblewrap support for running jobs (playbook runs only), ensure you are logged in as the Admin user and click on the settings gear ⚙ in the upper right-hand corner. Click on the "Configure Tower" box, then click on the "Jobs" tab. Scroll down until you see "Enable Job Isolation" and change the radio button selection to "on".

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.

21.10 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.

21.11 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.

CHAPTER

TWENTYTWO

TOWER TIPS AND TRICKS

22.1 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.

22.2 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 convienience)
  - `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.

22.3 `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
```

22.4 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.

22.5 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
```

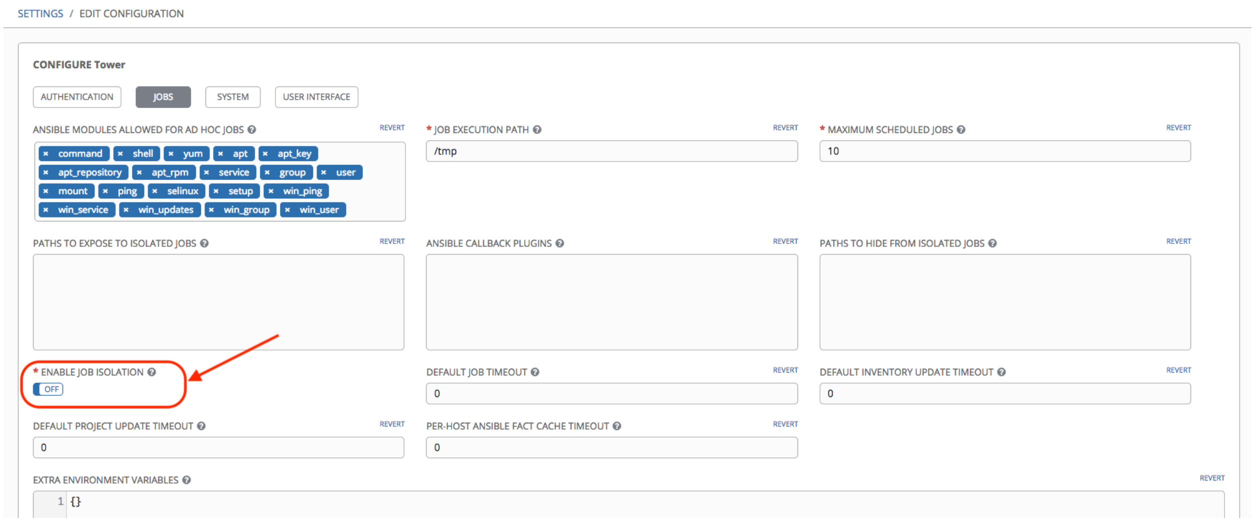
22.6 Setting up a jump host to use with Tower

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:



ANSIBLE MODULES ALLOWED FOR AD HOC JOBS: 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

* JOB EXECUTION PATH: /tmp

* MAXIMUM SCHEDULED JOBS: 10

PATHS TO EXPOSE TO ISOLATED JOBS

ANSIBLE CALLBACK PLUGINS

PATHS TO HIDE FROM ISOLATED JOBS

* ENABLE JOB ISOLATION: OFF

DEFAULT JOB TIMEOUT: 0

DEFAULT INVENTORY UPDATE TIMEOUT: 0

DEFAULT PROJECT UPDATE TIMEOUT: 0

PER-HOST ANSIBLE FACT CACHE TIMEOUT: 0

EXTRA ENVIRONMENT VARIABLES:

```
{}
```

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>"'
```

22.7 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/
```

22.8 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.

Note:

> If running a job template that uses a playbook with a custom module that has a non-default library path and fact caching enabled, you may encounter errors. To avoid this, do not set the path in `ansible.cfg library` but instead, do the following:
>
> ```
>     HTTP PATCH https://tower.example.org/api/v2/settings/jobs/
> {
>     "AWS_TASK_ENV": {"ANSIBLE_LIBRARY": "/custom/path"}
> }
> ```

If Tower sees a value set there, it won't overwrite it. Instead, it just appends the Tower-specific modules to the path so that they are used properly.

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.

22.9 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

22.10 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 virtualenvs during installation–one is used to run Tower, while the other is used to run Ansible. This allows Tower to run in a stable environment, while allowing you to add or update modules to your Ansible Python environment as necessary to run your playbooks.For more information on virtualenv, see the Python Guide to Virtual Environments.

Note: It is highly recommended that you run `umask` 0022 before installing any packages to the virtual environment, such a python package after initial install. Failure to properly configure permissions can result in Tower service failures. An example as follows:

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

22.10.1 Modifying the virtualenv

Modifying the virtualenv used by Tower is unsupported and not recommended. Instead, you can add modules to the virtualenv that Tower uses to run Ansible.

To do so, activate the Ansible virtualenv:

```
. /var/lib/awx/venv/ansible/bin/activate
```

...and then install whatever you need using `pip`:

```
pip install mypackagename
```

22.11 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.

22.12 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:awxsecret \
    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:awxsecret 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.

22.13 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.

22.14 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.

22.15 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.

22.16 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/pymodules/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`.

22.17 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.

22.18 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.

CHAPTER

TWENTYTHREE

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.

23.1 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.

23.2 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

23.3 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/

23.4 Configuration

`tower-cli` can edit its own configuration or users can directly edit the configuration file, allowing configuration to be set in multiple ways.

23.4.1 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
```

23.4.2 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
```

23.4.3 File Locations

The order of precedence for configuration file locations is as follows, from least to greatest:

- internal defaults
- `/etc/awx/tower_cli.cfg` (written using `tower-cli config --global`)

- ~/.tower_cli.cfg (written using tower-cli config)
- run-time parameters

23.4.4 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

23.4.5 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

23.4.6 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
```

CHAPTER

TWENTYFOUR

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. From the Settings (⚙) Menu screen, click on **Configure Tower**.
2. Select the **User Interface** tab.
3. Select the desired level of data collection from the Analystics 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.

4. 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.

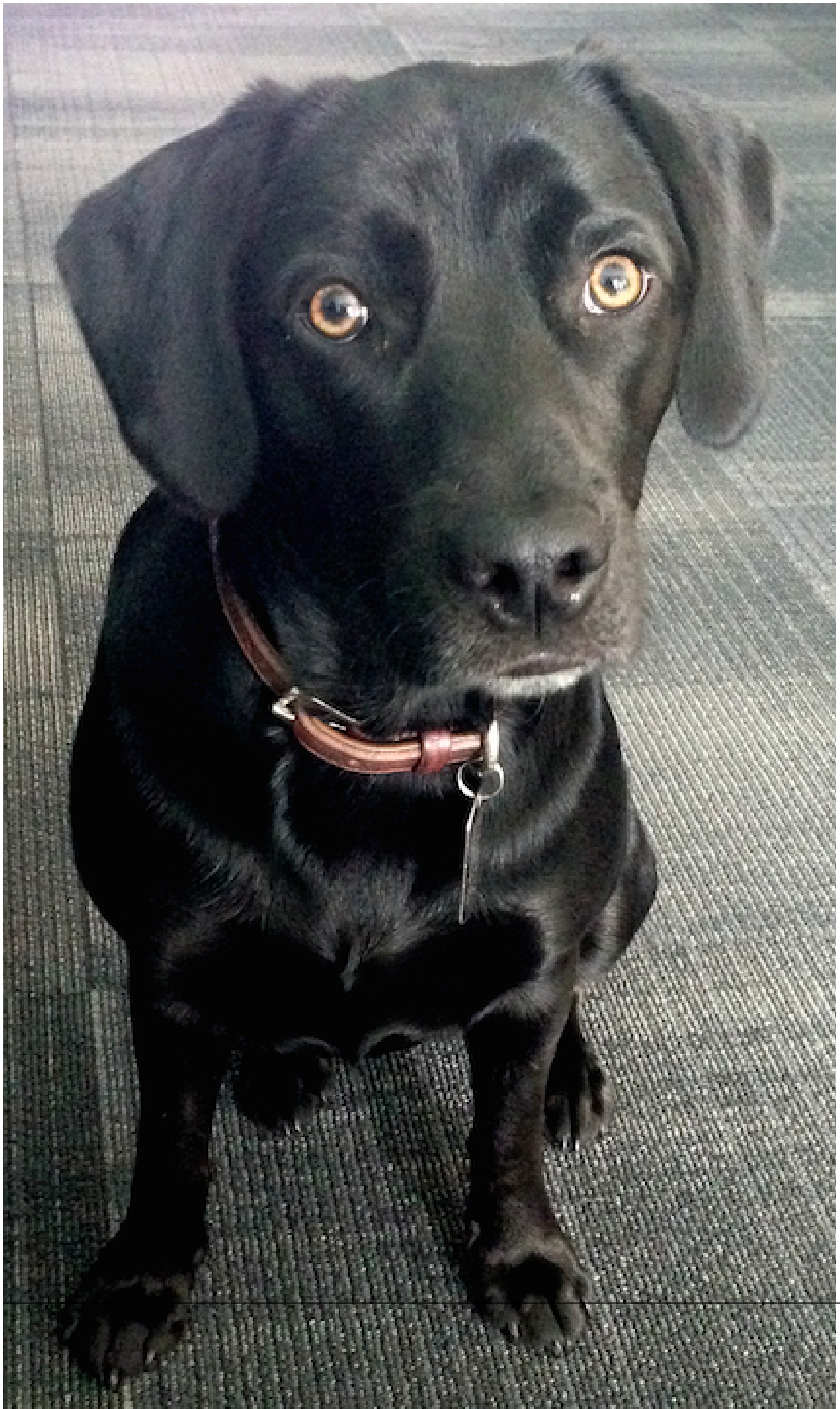
CHAPTER TWENTYFIVE

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.

CHAPTER TWENTYSIX

INDEX

- genindex

CHAPTER

TWENTYSEVEN

COPYRIGHT © 2018 RED HAT, INC.

Third Party Rights

INDEX